# Contents

1. **Contents** ........................................... 3
   1.1 What is PlatformIO? .................................. 3
   1.2 PlatformIO IDE ..................................... 5
   1.3 PlatformIO Core (CLI) ............................... 6
   1.4 PlatformIO Home .................................... 121
   1.5 Tutorials and Examples .............................. 126
   1.6 “platformio.ini” (Project Configuration File) .... 182
   1.7 Environment variables ............................... 217
   1.8 Advanced Scripting .................................. 221
   1.9 Library Manager .................................... 232
   1.10 Development Platforms ............................. 251
   1.11 Frameworks ....................................... 397
   1.12 Boards ............................................ 508
   1.13 Custom Platform & Board ........................... 1648
   1.14 PIO Account ....................................... 1653
   1.15 PIO Remote ....................................... 1654
   1.16 PIO Unified Debugger .............................. 1660
   1.17 PIO Unit Testing ................................... 1764
   1.18 Cloud & Desktop IDE ............................... 1771
   1.19 Continuous Integration ............................. 1875
   1.20 Articles about us ................................... 1892
   1.21 Frequently Asked Questions ........................ 1896
   1.22 Release Notes ..................................... 1904
   1.23 Migrating from 3.x to 4.0 ......................... 1933

**Bibliography** ........................................... 1941

**Index** .................................................. 1943
Cross-platform IDE and unified debugger. Remote unit testing and firmware updates.

Social: Twitter | Facebook | Hackaday | Bintray | Community
1.1 What is PlatformIO?

1.1.1 Press about PlatformIO

“Different microcontrollers normally have different developing tools. For instance Arduino rely on Arduino IDE. Few more advanced users set up different graphical interfaces like Eclipse for better project management. Sometimes it may be hard to keep up with different microcontrollers and tools. You probably thought that single unified development tool could be great. Well this is what PlatformIO open source ecosystem is for.

This is cross platform code builder and library manager with platforms like Arduino or MBED support. They took care of toolchains, debuggers, frameworks that work on most popular platforms like Windows, Mac and Linux. It supports more than 200 development boards along with more than 15 development platforms and 10 frameworks. So most of popular boards are covered. They’ve done hard work in organizing and managing hundreds of libraries that can be included in to your project. Also lots of examples allow you to start developing quickly. PlatformIO initially was developed with Command line philosophy. It’s been successfully used with other IDE’s like Eclipse or Visual Studio. Recently they’ve released a version with built in IDE based on Atom text editor”, - [Embeds].
1.1.2 Awards

PlatformIO was nominated for the year’s best Software and Tools in the 2015/16 IoT Awards.

1.1.3 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.

1.1.4 Overview

PlatformIO is independent of the platform, in which it is running. In fact, the only requirement is Python, which exists pretty much everywhere. What this means is that PlatformIO projects can be easily moved from one computer to another, as well as that PlatformIO allows for the easy sharing of projects between team members, regardless of operating system they prefer to work with. Beyond that, PlatformIO can be run not only on commonly used desktops/laptops but also on the servers without X Window System. While PlatformIO itself is a console application, it can be used in combination with one’s favorite Cloud & Desktop IDE or text editor such as PlatformIO IDE for Atom, CLion, Eclipse, Emacs, NetBeans, Qt Creator, Sublime Text, VIM, Visual Studio, PlatformIO IDE for VSCode, etc.

Alright, so PlatformIO can run on different operating systems. But more importantly, from a development perspective at least, is a list of supported boards and MCUs. To keep things short: PlatformIO supports approximately 200 Embedded Boards and all major Development Platforms.

1.1.5 User SHOULD have a choice

- Decide which operating system they want to run development process on. You can even use one OS at home and another at work.
- Choose which editor to use for writing the code. It can be a pretty simple editor or powerful favorite Cloud & Desktop IDE.
- Focus on the code development, significantly simplifying support for the Development Platforms and MCUs.

1.1.6 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:
  - 550+ embedded boards
  - 30+ development platforms
  - 15+ frameworks
- PIO Unified Debugger
- PIO Remote
- PIO 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)

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

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

1.2.1 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 IDE for VSCode / 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 Manager**
- **Library Dependency Finder (LDF)**
- **Serial Port Monitor**
- Integration components (**Cloud & Desktop IDE** and **Continuous Integration**).

**PlatformIO Core** is written in Python 2 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

<table>
<thead>
<tr>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>“Blink Project”</strong></td>
</tr>
<tr>
<td>– Used in demo</td>
</tr>
<tr>
<td><strong>Platform Manager</strong></td>
</tr>
<tr>
<td>– Used in demo</td>
</tr>
<tr>
<td><strong>Library Manager</strong></td>
</tr>
<tr>
<td>– Used in demo</td>
</tr>
<tr>
<td><strong>Over-the-Air update for ESP8266</strong></td>
</tr>
<tr>
<td>– Used in demo</td>
</tr>
</tbody>
</table>

**“Blink Project”**

**Used in demo**

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

**Platform Manager**

**Used in demo**

1. `Platform Manager`
2. `platformio platform list` command
3. `platformio platform search avr` command
4. `platformio platform show teensy` command
5. `platformio platform update` command.

**Library Manager**

**Used in demo**

1. `Library Manager`
2. `platformio lib search 1-wire` command
3. `platformio lib install 54` command
4. `platformio lib search -f mbed` command
5. `platformio lib search -k rf` command
6. `platformio lib search radiohead` command
7. `platformio lib install 124 --version “1.40”` command
8. `platformio lib show 124` command
9. `platformio lib update` command.

**Over-the-Air update for ESP8266**
Used in demo

1. `platformio run` command
2. `platformio 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
  - Python Package Manager
  - Installer Script
    * Super-Quick (Mac / Linux)
    * Local Download (Mac / Linux / Windows)
  - macOS Homebrew
  - Full Guide
  - Virtual Environment
    * Prerequisites
    * Creating
- Development Version
- Install Shell Commands
  - Unix and Unix-like
    * Method 1
    * Method 2
    * Method 3
  - Windows
    * Uninstall PIO Core and dependent packages
- Troubleshooting

### System requirements

**Operating System** Windows, macOS, Linux, FreeBSD, Linux ARMv6+
Python Interpreter  Python 2.7 or Python 3.5+. See detailed instruction how to Install Python Interpreter for Windows.

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:

- Python Package Manager
- Installer Script
  - Super-Quick (Mac / Linux)
  - Local Download (Mac / Linux / Windows)
- macOS Homebrew
- Full Guide
- Virtual Environment
  - Prerequisites
  - Creating

Python Package Manager

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

```
pip install -U platformio
```

If pip command is not available run easy_install pip or use Installer Script which will install pip and platformio automatically.

Note that you may run into permissions issues running these commands. You have a few options here:

- Run with sudo to install PlatformIO and dependencies globally
- Specify the pip install --user option to install local to your user
- Run the command in a virtualenv local to a specific project working set.

Installer Script
Super-Quick (Mac / Linux)

To install or upgrade PlatformIO paste that at a Terminal prompt (MAY require administrator access sudo):

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

Local Download (Mac / Linux / Windows)

To install or upgrade PlatformIO, download (save as...) get-platformio.py script. Then run the following (MAY require administrator access sudo):

```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\get-platformio.py\script

# run it
C:\Python27\python.exe get-platformio.py
```

macOS Homebrew

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

```bash
brew install platformio
```

Full Guide

1. Check a python version:

```bash
python --version
```

If Python is not installed (command not found), please Install Python Interpreter.

2. Install a platformio and related packages:

```bash
pip install -U platformio
```

If your computer does not recognize pip command, try to install it first using these instructions.

For upgrading platformio to the latest version:

```bash
pip install -U platformio
```
Virtual Environment

PlatformIO Core may be installed into isolated Python environment. This method is very good if you don’t want to install PlatformIO Core Python’s dependencies (packages) into your global system scope. *PlatformIO IDE* uses this method to install PlatformIO Core.

Default and recommended environment folder is “*core_dir*/penv”. You can print environment folder path using the next command in your system terminal:

```python
python -c "import os; print(os.path.join(os.getenv('PLATFORMIO_CORE_DIR', os.path.join(os.path.expanduser('~'), '.platformio')), 'penv'))"
```

### Examples

- **Windows**
  
  C:\Users\UserName\.platformio\penv

- **Linux**
  
  ~/.platformio/penv

- **macOS**
  
  ~/.platformio/penv

Prerequisites

1. Please remove existing PlatformIO Core environment folder if exists. See above how to get a path.

2. Please check that you have a valid Python interpreter running a next command in system terminal. Python 2.7.9+ or Python 3.5+ is recommended.

   ```
   python --version
   # or, for Unix (Linux, Mac), you can use `python2` or `python3` aliases
   python2 --version
   python3 --version
   ```

   **Warning:** Windows Users: If you already tried to install PlatformIO IDE and did not get success, please open system’s Control Panel > Installed Programs, and check if PlatformIO IDE tried to install an own isolated Python 2.7 version. Please uninstall it. Also is good to uninstall all Python interpreters from a system and install manually the latest Python using Install Python Interpreter guide.

3. Make sure `virtualenv --help` command exists in a system, otherwise, please install it manually using:

   ```
   pip install virtualenv
   ```

   If `pip` (Python Package Manager) does not exists, you have to install it manually. See https://pip.pypa.io/en/stable/installing/

Creating

1. Create a folder which contains all the necessary executables to use the packages that PIO Core would need using `virtualenv` command:
virtualenv /path/to/.platformio/penv

# If you want to use a custom Python interpreter
virtualenv --python=/path/to/custom/python /path/to/.platformio/penv

# EXAMPLES
# Windows
virtualenv C:\Users\UserName\.platformio\penv
virtualenv --python=C:\Python27\python.exe C:\Users\UserName\.platformio\penv

# Unix (Linux, Mac)
virtualenv ~/.platformio/penv
virtualenv -p python3 ~/.platformio/penv

2. Activate virtual environment

# Windows
C:\Users\UserName\.platformio\penv

# Unix (Linux, Mac)
source /path/to/.platformio/penv/bin/activate
# or
 . /path/to/.platformio/penv/bin/activate

3. Install PIO Core into virtual environment

pip install -U platformio

If you plan to use PIO Core commands outside virtual environment, please Install Shell Commands.

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”
- **PlatformIO IDE for VSCode**: Set platformio-ide.useDevelopmentPIOCore to true in Proxy Server Support.

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 if you see the new commits in PlatformIO GitHub repository (branch: develop).

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](#)
- `piodebugdb` - alias of `platformio 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**
  - **Method 3**
- **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:

```bash
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:

```bash
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

Go to the PlatformIO menu → Settings → PlatformIO IDE, scroll down to the Custom PATH for 'platformio' command and enter the following: `~/.platformio/penv/bin`. After you’ve done that, you’ll need to go to the PlatformIO menu → Settings → PlatformIO IDE Terminal, scroll down to the Toggles section and uncheck the Login Shell checkbox. Finally, restart your editor/IDE and check out the result.
Method 3

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):

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

Uninstall PIO Core and dependent packages

- Uninstall PIO Core tool

  ```bash
  # uninstall standalone PIO Core installed via `pip'
  pip uninstall platformio
  # uninstall Homebrew's PIO 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.

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.

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 `platformio init` command for configuring your projects. It allows one to initialize new empty project or update existing with the new data.

What is more, `platformio 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>Espresif 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**

`platformio init` command requires to specify board identifier ID. It can be found using Boards catalog, Boards Explorer or `platformio boards` command. For example, using `platformio boards` let’s try to find Teensy boards:

```
> platformio boards teensy
Platform: teensy
---------------------------------------------------------------------------
ID     MCU         Frequency  Flash  RAM   Name
---------------------------------------------------------------------------
teensy20 atmega32u4  16MHz     31K  2.5K Teensy 2.0
teensy30 mk20dx128  48MHz     128K 16K Teensy 3.0
teensy31 mk20dx256  72MHz     256K 64K Teensy 3.1 / 3.2
teensylc mk126z64  48MHz     62K  8K Teensy LC
teensy20pp at90usb128 16MHz     127K 8K Teensy++ 2.0
```

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):
> platformio 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 `platformio 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:
`platformio run` - process/build project from the current directory
`platformio run --target upload` or `platformio run -t upload` - upload firmware to embedded board
`platformio 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 platformio init again.

The result of just generated platformio.ini:

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

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

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

1.3. PlatformIO Core (CLI)
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`:

```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)
  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);
}
```

The final Project structure:

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

**Process Project**

*PlatformIO Core (CLI)* provides special `platformio 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:

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

Please follow to `platformio run --target` 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

• *CLI Guide*
  – *Usage*
  – *Options*
  – *Commands*

#### Usage

```
pio [OPTIONS] COMMAND
platformio [OPTIONS] COMMAND
```

# "pio" is the alias of "platformio" command

#### Options

**--force**, `-f`

Force to accept any confirmation prompts and disable progress bars.

**--version**

Show the version of PlatformIO

**--help**, `-h`

Show help for the available options and commands

```
$ platformio --help
$ platformio COMMAND --help
```
Commands

platformio account

Helper command for PIO Account.

# Create PIO Account
pio account register

# Login with credentials (will be sent to your e-mail)
pio account login

# Change temporary password (from e-mail) to permanent
pio account password

To print all available commands and options use:

pio account --help
platformio account --help
platformio account COMMAND --help

platformio account forgot

Contents

• platformio account forgot
  – Usage
  – Description
  * Options

Usage

platformio account forgot [OPTIONS]
pio account forgot [OPTIONS]

Description

Allows you to reset password for PIO Account using E-Mail that was specified for registration.

Options

--username, -u
User name (E-Mail). You can omit this option and enter E-Mail in Forgot Wizard later.
platformio account login

Contents

• platformio account login
  – Usage
  – Description
  • Options

Usage

platformio account login [OPTIONS]
pio account login [OPTIONS]

Description

Log in to PIO 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 PIO Remote operations.

Options

--username, -u
User name (E-Mail). You can omit this option and enter E-Mail in Login Wizard later.

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

platformio account logout

Contents

• platformio account logout
  – Usage
  – Description

Usage

platformio account logout
pio account logout
Description

Log out of PIO Account.

platformio account password

Usage

platformio account password
pio account password

Description

Change password for PIO Account.

platformio account register

Contents

• platformio account register
  – Usage
  – Description
  • Options

Usage

platformio account register [OPTIONS]
pio account register [OPTIONS]

Description

Create a new PIO Account. A registration is FREE.
Options

`--username, -u`
User name (E-Mail). You can omit this option and enter E-Mail in Register Wizard later.

`platformio account show`

Usage

```
platformio account show
pio account show
```

Description

Show detailed information about PIO Account:

- Active groups and expiration
- Group permissions

Options

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

`platformio account token`
Usage

```
platformio account token
pio account token
```

Description

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

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

Options

```
--regenerate
--json-output
```

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

Return the output in JSON format

**platformio boards**

Usage

```
platformio boards [OPTIONS] [FILTER]
pio boards [OPTIONS] [FILTER]
```

Description

List pre-configured Embedded Boards

Options

```
--installed
```

Chapter 1. Contents
List boards only from the installed platforms

```
--json-output
```

Return the output in JSON format

**Examples**

1. Show all available pre-configured embedded boards

```
$ platformio boards

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>
<tr>
<td>btatmega328</td>
<td>atmega328p</td>
<td>16MHz</td>
<td>28K</td>
<td>2K</td>
<td>Arduino BT ATmega328</td>
</tr>
</tbody>
</table>
| 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

```
$ platformio boards arduino

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>
<tr>
<td>btatmega328</td>
<td>atmega328p</td>
<td>16MHz</td>
<td>28K</td>
<td>2K</td>
<td>Arduino BT ATmega328</td>
</tr>
</tbody>
</table>
| 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      |
| ...             |          |          |       |     |                       |
```

3. Filter mbed-enabled boards

```
$ platformio 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>
</table>
| frdm_k20d50m | mk20dx128v1h5 | 48MHz | 128K  | 16K | Freescale Kinetis FRDM-  
|          | —K20D50M |          |       |     |                       |
| frdm_k22f | mk22fn512vlh12 | 120MHz | 512K  | 128K | Freescale Kinetis FRDM-  
|          | —K22F    |          |       |     |                       |
| ...             |          |          |       |     |                       |
```

(continues on next page)
### 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 Board</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>32F401C Discovery</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
$ platformio 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>
<tr>
<td>diecimilaatmega168</td>
<td>atmega168</td>
<td>16MHz</td>
<td>14K</td>
<td>1K</td>
<td>Arduino Duemilanove or Diecimila ATmega168</td>
</tr>
<tr>
<td>miniatmega168</td>
<td>atmega168</td>
<td>16MHz</td>
<td>14K</td>
<td>1K</td>
<td>Arduino Mini ATmega168</td>
</tr>
<tr>
<td>atmegangatmega168</td>
<td>atmega168</td>
<td>16MHz</td>
<td>14K</td>
<td>1K</td>
<td>Arduino NG or older ATmega168</td>
</tr>
<tr>
<td>nanoatmega168</td>
<td>atmega168</td>
<td>16MHz</td>
<td>14K</td>
<td>1K</td>
<td>Arduino Nano ATmega168</td>
</tr>
<tr>
<td>pro8MHZatmega168</td>
<td>atmega168</td>
<td>8MHz</td>
<td>14K</td>
<td>1K</td>
<td>Arduino Pro or Pro Mini ATmega168 (3.3V, 8 MHz)</td>
</tr>
<tr>
<td>pro16MHZatmega168</td>
<td>atmega168</td>
<td>16MHz</td>
<td>14K</td>
<td>1K</td>
<td>Arduino Pro or Pro Mini ATmega168 (5V, 16 MHz)</td>
</tr>
<tr>
<td>lilypadatmega168</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
$ platformio boards timsp430
```

### Platform: timsp430

(continues on next page)
### Contents

- `platformio ci`
  - Usage
  - Description
  - Options
  - Examples

### Usage

```bash
platformio ci [OPTIONS] [SRC]
pio ci [OPTIONS] [SRC]
```

### Description

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

`platformio ci` command accepts multiple `SRC` arguments, `platformio ci --lib` and `platformio 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:** `platformio ci` command is useful for library developers. It allows one to build different examples without creating own project per them. Also, is possible to upload firmware to the target device. In this case, you need to pass addi-
Options

-l, --lib
Source code which will be copied to <BUILD_DIR>/lib directly.
If platformio ci --lib is a path to file (not to directory), then PlatformIO will create temporary directory within <BUILD_DIR>/lib and copy the rest files into it.

--exclude
Exclude directories and/or files from platformio ci --build-dir. The path must be relative to PlatformIO project within platformio ci --build-dir.
For example, exclude from project src directory:
  • examples folder
  • *.h files from foo folder

platformio ci --exclude=src/examples --exclude=src/foo/*.h [SRC]

-b, --board
Build project with automatically pre-generated environments based on board settings.
For more details please look into platformio 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 platformio ci --keep-build-dir.

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

-c, --project-conf
Build project using pre-configured “platformio.ini” (Project Configuration File).

-O, --project-option
Pass additional options from “platformio.ini” (Project Configuration File) to platformio init command. For example, automatically install dependent libraries platformio ci --project-option="lib_deps=ArduinoJSON" or ignore specific library platformio 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, platformio ci --project-option="build_unflags = -std=gnu++11" --project-option="build_flags = -std=c++14"
-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 others examples please follow to Continuous Integration page.

platformio debug
Helper command for PIO Unified Debugger.

## Contents
- platformio debug
  - Usage
  - Description
  - Options
  - Examples

## Usage
```
platformio debug [OPTIONS]
pio debug [OPTIONS]
```
# A binary shortcut for "platformio debug --interface=gdb" command
```
piodebuggdb [GDB OPTIONS]
```

## Description
Prepare PlatformIO project for debugging or launch debug server.

## Options
- `-e`, `--environment`
  Debug specified environments.
  You can also specify which environments should be used for debugging by default using `default_envs` option from “platformio.ini” (Project Configuration File).
- `-d`, `--project-dir`
  Specify the path to a project directory. By default, `--project-dir` is equal to a current working directory (CWD).
- `-c`, `--project-conf`
New in version 4.0.

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

--interface

PIO 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

```sh
> platformio debug

[Sun Apr 30 01:34:01 2017] Processing mzeropro (platform: atmelavr; debug_extra_cmds:

verbose)

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

```

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

```sh
> platformio debug --interface=gdb -x .pioinit

PIO Plus (https://pioplus.com) v0.8.2

... Loading section .text, size 0x2c98 lma 0x4000
Loading section .ramfunc, size 0x60 lma 0x6c98

(continues on next page)```
Loading section .data, size 0x100 lma 0x6cf8
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: 0x0000028f4 msp: 0x20002c00
target halted due to debug-request, current mode: Thread
xPSR: 0x81000000 pc: 0x0000028f4 msp: 0x20002c00
Breakpoint 2 at 0x413a: file src/main.cpp, line 26.

platformio device

Contents

• platformio device
  – platformio device list
    • Usage
    • Description
    • Options
    • Examples
  – platformio device monitor
    • Usage
    • Description
    • Options
    • Examples

platformio device list

Usage

platformio device list [OPTIONS]
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.

1.3. PlatformIO Core (CLI)
--logical
List available logical devices.

--mdns
List multicast DNS services.

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

Examples

1. Unix OS

```bash
$ platformio 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

```bash
$ platformio 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

```bash
$ platformio 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
```

(continues on next page)
Properties: ...

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

PlatformIO._sftp-ssh._tcp.local.
-----------------------------
Type: _sftp-ssh._tcp.local.
IP: ...
Port: 22

Logical Devices
=============
/
Name:
/Volumes/PIO
Name: PIO
/Volumes/PLUS
Name: PLUS

platformio device monitor

Usage

platformio 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.

-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

**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 `platformio 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`

```bash
$ platformio device monitor --help
Usage: platformio 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
```
2. Communicate with serial device and print help inside terminal

```bash
$ platformio 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 ---
```

### `platformio home`

Helper command for **PlatformIO Home**.

### Contents
- `platformio home`
  - **Usage**
  - **Description**
  - **Options**
  - **Examples**
Usage

platformio home
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 PIO Home for inbound connections using host 0.0.0.

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

Examples

> platformio home

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

platformio init

Contents

- platformio init
  - Usage
  - Description
  - Options
  - Examples
Usage

platformio init [OPTIONS]
pio init [OPTIONS]

Description

Initialize new PlatformIO based project or update existing with new data.
This command will create:

- "platformio.ini" (Project Configuration File)
- include_dir, put project header files here
- src_dir, put project source files here (*.h, *.c, *.cpp, *.S, *.ino, etc.)
- lib_dir, put project specific (private) libraries here. See also Library Dependency Finder (LDF)
- test_dir, put project tests here. More details PIO Unit Testing
- Miscellaneous files for VCS and Continuous Integration support.

Options

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

-b, --board
If you specify board ID (you can pass multiple --board options), then 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 Development Platforms.

--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 platformio init --help command. Also, please take a look at Cloud & Desktop IDE page.

-O, --project-option

Initialize project with additional options from “platformio.ini” (Project Configuration File). For example, platformio init --project-option="lib_deps=ArduinoJSON". Multiple options are allowed.

--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].

-s, --silent
Suppress progress reporting
Examples

1. Initialize new project in a current working directory

```bash
> platformio init

The current working directory *** will be used for the new project.
You can specify another project directory via
`platformio 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:
  `platformio run` - process/build project from the current directory
  `platformio run --target upload` or `platformio run -t upload` - upload firmware to
  `platformio run --target clean` - clean project (remove compiled files)
```

2. Initialize new project in a specified directory

```bash
> platformio 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
> platformio init --board uno

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

4. Initialize project for Teensy 3.1 board with custom mbed

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

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

Library Manager

Usage

```
platformio lib [OPTIONS] COMMAND

# To print all available commands and options use
```

(continues on next page)
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

platformio lib builtin

Usage

platformio lib builtin [OPTIONS]
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

```bash
> platformio 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
Homepage: http://arduino.cc/en/Reference/Audio
Keywords: signal, input, output
Compatible frameworks: arduino
Compatible platforms: atmelsam
Authors: Arduino
```
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: 
Authors: Hristo Gochkov

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 Grokhotkov 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

(continues on next page)
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

platformio lib install

Contents

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

Usage

platformio lib [STORAGE_OPTIONS] install [OPTIONS] [LIBRARY...]
pio lib [STORAGE_OPTIONS] install [OPTIONS] [LIBRARY...]
# install all project dependencies declared via "lib_deps"
platformio lib install [OPTIONS]

# install project dependent library
platformio lib install [OPTIONS] [LIBRARY...]

# install dependencies for the specific project build environment
platformio lib -e myenv install [OPTIONS] [LIBRARY...]
platformio lib -d /path/to/platformio/project -e myenv install [OPTIONS] [LIBRARY...]

# install to global storage
platformio lib --global install [OPTIONS] [LIBRARY...]
platformio lib -g install [OPTIONS] [LIBRARY...]

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

# [LIBRARY...] forms
platformio lib [STORAGE_OPTIONS] install (with no args, project dependencies)
platformio lib [STORAGE_OPTIONS] install <id>
platformio lib [STORAGE_OPTIONS] install id=<id>
platformio lib [STORAGE_OPTIONS] install <id>@<version>
platformio lib [STORAGE_OPTIONS] install id=<name>
platformio lib [STORAGE_OPTIONS] install <name>
platformio lib [STORAGE_OPTIONS] install <name>@<version>
platformio lib [STORAGE_OPTIONS] install <name>@<version range>
platformio lib [STORAGE_OPTIONS] install <zip or tarball url>
platformio lib [STORAGE_OPTIONS] install file://<zip or tarball file>
platformio lib [STORAGE_OPTIONS] install file://<folder>
platformio lib [STORAGE_OPTIONS] install <repository>
platformio lib [STORAGE_OPTIONS] install <name>=<repository> (name it should have locally)
platformio 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

- **Atom:** “Menu: PlatformIO > Rebuild C/C++ Project Index (Autocomplete, Linter)"
- **VSCode:** “Menu: View > Command Palette... > PlatformIO: Rebuild C/C++ Project Index”

---

**Description**

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

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

---

1.3. PlatformIO Core (CLI) 45
The version supports Semantic Versioning (\texttt{<major>.<minor>.<patch>}) and can take any of the following forms:

- \texttt{1.2.3} - an exact version number. Use only this exact version
- \texttt{^1.2.3} - any compatible version (exact version for 1.x.x versions)
- \texttt{~1.2.3} - any version with the same major and minor versions, and an equal or greater patch version
- \texttt{>1.2.3} - any version greater than 1.2.3. \texttt{>=}, \texttt{<}, and \texttt{<=} are also possible
- \texttt{>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

PlatformIO supports installing from local directory or archive. Need to use \texttt{file://} prefix before local path. Also, directory or archive should contain \texttt{.library.json} manifest (see \texttt{library.json}).

- \texttt{file:///local/path/to/the/platform/dir}
- \texttt{file:///local/path/to/the/platform.zip}
- \texttt{file:///local/path/to/the/platform.tar.gz}

\section*{Storage Options}

See base options for \textit{Library Manager}.

\section*{Options}

\texttt{--save}

Save installed libraries into the \texttt{platformio.ini} (Project Configuration File) dependency list (\texttt{lib_deps}).

You can save libraries for the specific project environment using \texttt{-e}, \texttt{--environment} option from \texttt{platformio lib} command. For example, \texttt{platformio lib -e myenv install [LIBRARY...]}. 

\texttt{-s, --silent}

Suppress progress reporting

\texttt{--interactive}

Allow one to make a choice for all prompts

\texttt{-f, --force}

Reinstall/redownload library if it exists

\section*{Version control}

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

\textbf{Note:} PlatformIO requires a working VCS command on your path: \texttt{git}, \texttt{hg} or \texttt{svn}. 

46 Chapter 1. Contents
Git

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

- `user/library` (short version for GitHub repository)
- `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

```bash
> platformio lib -g install 4
Library Storage: /storage/dir/...
LibraryManager: Installing id=4
Downloading [########################################] 100%
Unpacking [########################################] 100%
IRremote @ 2.2.1 has been successfully installed!

# repeat command with name
> platformio 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

```bash
> platformio 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

```bash
> platformio 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 @ 8fd2ebfec7 has been successfully installed!
```

4. Install ARM mbed library to the global storage

```bash
> platformio lib -g install https://developer.mbed.org/users/simon/code/TextLCD/
Library Storage: /storage/dir/...
```
5. Install from archive using URL

```bash
> platformio 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!

platformio lib list

Contents

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

Usage

```bash
platformio lib [STORAGE_OPTIONS] list [OPTIONS]
pio lib [STORAGE_OPTIONS] list [OPTIONS]
```

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

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

## Description

List installed libraries

## Storage Options

See base options for Library Manager.

## Options

--json-output

Return the output in JSON format

## Examples

```
> platformio 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
```
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, ...
---MKR100, 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, ...
---usb, 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

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

LiquidCrystal
=============  
#ID: 136
LiquidCrystal Library is faster and extensible, compatible with the original LiquidCrystal library

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

TextLCD
=======
hg+https://developer.mbed.org/users/simon/code/TextLCD/

Version: 308d188a2d3a
Keywords: uncategorized

Time
====
#ID: 44
Time keeping library

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
========
(continues on next page)
#ID: 76
Arduino library to facilitate time zone conversions and automatic daylight saving time adjustments

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

U8g2
=====
#ID: 942
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, ST7920, LD7032, KS0108.
Interfaces: I2C, SPI, Parallel.

Version: 2.11.4
Homepage: https://github.com/olikraus/u8g2
Keywords: display
Compatible frameworks: arduino
Compatible platforms: atmelavr, atmelsam, expressif8266, intel_arc32, microchippic32, nordicnrf51, teensy, timsp430
Authors: oliver

USB-Host-Shield

platformio lib register

Contents

- platformio lib register
  - Usage
  - Description
  - Examples
Usage

```plaintext
platformio lib register [MANIFEST_URL]
pio lib register [MANIFEST_URL]
```

Description

Register new library in PlatformIO Library Registry.

PlatformIO Library Registry supports the next library manifests:

- PlatformIO `library.json`
- Arduino `library.properties`
- ARM mbed yotta `module.json`.

Examples

```plaintext
platformio lib register https://raw.githubusercontent.com/bblanchon/ArduinoJson/ -->master/library.json
platformio lib register https://raw.githubusercontent.com/adafruit/DHT-sensor-library/ -->master/library.properties
platformio lib register https://raw.githubusercontent.com/ARMmbed/ble/master/module. -->json
```

platformio lib search

Contents

- `platformio lib search`
  - Usage
  - Description
  - Options
  - Examples

Usage

```plaintext
platformio lib search [OPTIONS] [QUERY]
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:
### 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, 
```
platformio lib search --header "OneWire.h"
```

---

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

---

```
--page
Manually paginate through search results. This option is useful in pair with --json-output.
```

---

1.3. PlatformIO Core (CLI) 55
Examples

1. List all libraries

```
> platformio 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

Show next libraries? [y/N]:
...
2. Search for 1-Wire libraries

```shell
> platformio 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

```shell
> platformio 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.

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

(continues on next page)
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.

> platformio 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
=====
#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, 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

```
> platformio 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.

```
> platformio 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

(continues on next page)
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

```bash
> platformio 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
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

(continues on next page)
8. Search for the all sensor libraries excluding temperature.

```bash
> platformio 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]: ...

**platformio lib show**

**Contents**

- **platformio lib show**
  - **Usage**

---

1.3. PlatformIO Core (CLI)
Usage

```bash
platformio lib show [LIBRARY]
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

```bash
> platformio 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
Homepage: http://pubsubclient.knolleary.net
Repository: https://github.com/knolleary/pubsubclient.git

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

Keywords
--------
ethernet
```
(continues on next page)
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

```
platformio lib stats
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/       GroveEncoder
                     12 hours ago             https://platformio.org/lib/show/1381/       RF24G
Sim800L Library Revised 12 hours ago         https://platformio.org/lib/show/1380/     Sim800L%20Library%20Revised
                     12 hours ago             https://platformio.org/lib/show/750/         ArduinoSTL
hd44780               13 hours ago             https://platformio.org/lib/show/738/       hd44780

RECENTLY ADDED
**************
Name                   Date                      Url
-----------------------------------------------
(continues on next page)
```
GroveEncoder 12 hours ago https://platformio.org/lib/show/1382/
---GroveEncoder
RF24G 12 hours ago https://platformio.org/lib/show/1381/
---RF24G
Sim800L Library Revised 12 hours ago https://platformio.org/lib/show/1380/
---Sim800L%20Library%20Revised
DS3231 a day ago https://platformio.org/lib/show/1379/
---DS3231
ArduboyPlaytune 4 days ago https://platformio.org/lib/show/1378/
---ArduboyPlaytune

RECENT KEYWORDS
***************
Name Url
--------------------------------------------------------------------------------------
---
cobs https://platformio.org/lib/search?query=keyword%3Acobs
packet https://platformio.org/lib/search?query=keyword%3Apacket
framing https://platformio.org/lib/search?query=keyword%3Aframing
3g https://platformio.org/lib/search?query=keyword%3A3g
tdd https://platformio.org/lib/search?query=keyword%3Atdd

POPULAR KEYWORDS
***************
Name Url
--------------------------------------------------------------------------------------
---
display https://platformio.org/lib/search?query=keyword%3Adisplay
lcd https://platformio.org/lib/search?query=keyword%3Alcd
sensors https://platformio.org/lib/search?query=keyword%3Asensors
graphics https://platformio.org/lib/search?query=keyword%3Agraphics
communication https://platformio.org/lib/search?query=keyword
---%3Acommunication
oled https://platformio.org/lib/search?query=keyword%3Aoled
tft https://platformio.org/lib/search?query=keyword%3Atft
control https://platformio.org/lib/search?query=keyword%3Acontrol
device https://platformio.org/lib/search?query=keyword%3Adevice
glcd https://platformio.org/lib/search?query=keyword%3Aglcd
displaycore https://platformio.org/lib/search?query=keyword%3Adisplaycore
font https://platformio.org/lib/search?query=keyword%3Afont
other https://platformio.org/lib/search?query=keyword%3Aother
i2c https://platformio.org/lib/search?query=keyword%3Ai2c
input https://platformio.org/lib/search?query=keyword%3Ainput
signal https://platformio.org/lib/search?query=keyword%3Asignal
sensor https://platformio.org/lib/search?query=keyword%3Asensor
output https://platformio.org/lib/search?query=keyword%3Aoutput
spi https://platformio.org/lib/search?query=keyword%3Aspi
data https://platformio.org/lib/search?query=keyword%3Adata
timing https://platformio.org/lib/search?query=keyword%3Atiming
serial https://platformio.org/lib/search?query=keyword%3Aserial
temperature https://platformio.org/lib/search?query=keyword%3Atemperature
http https://platformio.org/lib/search?query=keyword%3Ahttp
wifi https://platformio.org/lib/search?query=keyword%3Awifi
rf https://platformio.org/lib/search?query=keyword%3Arf
i2cdevlib https://platformio.org/lib/search?query=keyword%3Ai2cdevlib
<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>
<tr>
<td>Adafruit Unified Sensor</td>
<td><a href="https://platformio.org/lib/show/31/Adafruit%20Unified">https://platformio.org/lib/show/31/Adafruit%20Unified</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>
<tr>
<td>RTClib</td>
<td><a href="https://platformio.org/lib/show/83/RTClib">https://platformio.org/lib/show/83/RTClib</a></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Url</th>
</tr>
</thead>
<tbody>
<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>Adafruit Unified Sensor</td>
<td><a href="https://platformio.org/lib/show/31/Adafruit%20Unified">https://platformio.org/lib/show/31/Adafruit%20Unified</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>Adafruit GFX Library</td>
<td><a href="https://platformio.org/lib/show/13/Adafruit%20GFX%20Library">https://platformio.org/lib/show/13/Adafruit%20GFX%20Library</a></td>
</tr>
<tr>
<td>12Cdevlib-Core</td>
<td><a href="https://platformio.org/lib/show/11/12Cdevlib-Core">https://platformio.org/lib/show/11/12Cdevlib-Core</a></td>
</tr>
</tbody>
</table>
platformio lib uninstall

Contents

- platformio lib uninstall
  - Usage
  - Description
  - Storage Options
  - Examples

Usage

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

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

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

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

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

Examples

> platformio lib -g uninstall AsyncMqttClient

Library Storage: /storage/dir/...
Uninstalling AsyncMqttClient @ 0.2.0: [OK]

platformio lib update

Contents

- platformio lib update
  - Usage
  - Description
  - Storage Options
  - Options
Usage

platformio lib [STORAGE_OPTIONS] update [OPTIONS]
pio lib [STORAGE_OPTIONS] update [OPTIONS]

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

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

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

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

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

```
> platformio 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)
```

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

2. Update specified libraries in global storage

```
> platformio 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]
```

Platform Manager

To print all available commands and options use:
platformio platform --help
platformio platform COMMAND --help

platformio platform frameworks

Contents

• platformio platform frameworks
  – Usage
  – Description
    • Options
  – Examples

Usage

platformio platform frameworks QUERY [OPTIONS]
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.

> platformio 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.

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

artik-sdk ~ ARTIK SDK

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

platformio platform install

Contents

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

Usage

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

# [PLATFORM...] forms
platformio platform install <name>
platformio platform install <name>@<version>
platformio platform install <name>@<version range>
platformio platform install <zip or tarball url>
platformio platform install file://<zip or tarball file>
platformio platform install file://<folder>
platformio platform install <repository>
platformio platform install <name=repository> (name it should have locally)
platformio 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

-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 - 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

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:

- `platformio/platform-NAME` (short version for GitHub repository)
- `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+hg://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+hg://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`
- `svn+ssh://svn.server.org/my-platform`

You can also give specific revisions to an SVN URL, like so:

- `svn+svn://svn.server.org/my-platform#13`
Examples

1. Install *Atmel AVR* with default packages

   ```shell
   > platformio 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

   ```shell
   > platformio 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!
   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.

3. Install the latest development *Atmel AVR* from Git repository

   ```shell
   > platformio 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/... platforms/installing-U3ucN0-package'...
   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.

platformio platform list

Contents

- platformio platform list
  - Usage
  - Description
    * Options
  - Examples

Usage

platformio platform list [OPTIONS]
pio platform list [OPTIONS]

Description

List installed Development Platforms

Options

--json-output

Return the output in JSON format
Examples

> platformio 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
...

platformio platform search

Contents

- platformio platform search
  - Usage
  - Description
  * Options
  - Examples
Usage

platformio platform search QUERY [OPTIONS]
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

   > platformio 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 industry’s most code-efficient architecture for C and assembly programming.

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

   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-arduinomasam, framework-simba, tool-openocd, framework-mbed, tool-avrdude, tool-bossac

   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.

   Home: https://platformio.org/platforms/espressif32
   Packages: toolchain-xtensa32, framework-simba, framework-arduinoespressif32,
   framework-pumbaa, framework-espidf, tool-esptoolpy

   espressif8266 ~ Espressif 8266

(continues on next page)
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: toolchain-xtensa, framework-simba, tool-esptool, tool-mkspiffs, tool-espotapy, framework-arduinoespressif8266, sdk-esp8266

Freescale Kinetis ~ 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.

2. Search for TI development platforms

> platformio 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-energiantiva

> platformio 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
Packages: toolchain-gccarmnoneeabi, framework-arduinosam, framework-simba, tool-openocd, framework-mbed, ldscripts, tool-bossac

(continues on next page)
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.

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-rflloader, toolchain-gccarmnoneeabi, framework-
arduinoindircnrf51

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

stm32 ~ 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

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 device (desktop or
PC or Macintosh with a USB port).
platformio platform show

Usage

platformio platform show PLATFORM
pio platform show PLATFORM

Description

Show details about Development Platforms

Examples

> platformio 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

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)

platformio platform uninstall

Contents

- platformio platform uninstall
  - Usage
  - Description
  - Examples
Usage

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

Description

Uninstall specified *Development Platforms*

Examples

```
> platformio 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!
```

platformio platform update

Contents

- *platformio platform update*
  - *Usage*
  - *Description*
  - *Options*
  - *Examples*

Usage

```
platformio platform update [OPTIONS] [PLATFORM...]  
pio platform update [OPTIONS] [PLATFORM...]  
# update specific platform version using Semantic Versioning  
platformio 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

```
> platformio platform update
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 espressif @ 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]
...
```

platformio remote

Helper command for PIO Remote.

To print all available commands and options use:
PIO Remote Agent

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

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

---

**Contents**

- **PIO Remote Agent**
  - platformio remote agent list
    - Usage
    - Description
    - Example
  - platformio remote agent start
    - Usage
    - Description
    - Options
  - platformio remote agent reload
    - Usage
    - Description
    - Example

---

**platformio remote agent list**

**Usage**

```
platformio remote agent list
pio remote agent list
```

**Description**

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

```bash
> platformio remote agent list
PIO Plus (https://pioplus.com)
innomac.local
-----------
ID: 98853d930......788d77375e7
Started: 2016-10-26 16:32:56
```

platformio remote agent start

Usage

```bash
platformio remote agent start [OPTIONS]
pio remote agent start [OPTIONS]
```

Description

Start *PIO 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 `platformio remote device` and `platformio remote run` commands. Good names are home, office, lab or etc.

- `-s`, `--share`
  Share your agent/devices with other PlatformIO developers who have *PIO Account*: friends, co-workers, team, etc. The valid value for `--share` option is E-Mail address that was used for `platformio account register` command.

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

platformio remote agent reload

Usage
platformio remote agent reload
pio remote agent reload

# reload specified PIO Remote Agents
platformio remote --agent NAME reload

Description

Allows gracefully reload one or more PIO Remote Agent’s.

Example

> platformio remote agent list
PIO Plus (https://pioplus.com)

innomac.local

ID: 98853d93.....77375e7
Reloaded: 2016-11-1 23:33:32

platformio remote device

Remote Device: monitor remote device or list existing.

Contents

- platformio remote device
  - platformio remote device list
    - Usage
    - Description
    - Options
    - Example
  - platformio remote device monitor
    - Usage
    - Description
    - Options
    - Examples

platformio remote device list
Usage

```
platformio remote device list [OPTIONS]
pio remote device list [OPTIONS]
```

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

Description

List Serial Ports on remote machines where *PIO 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 `platformio remote agent start --name command` with “home” and “office” options:

- `platformio remote agent start --name home`
- `platformio remote agent start --name office`

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

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

Options

`--json-output`

Return the output in JSON format

Example

```
> platformio remote device list
PIO Plus (https://pioplus.com)
Agent innomac.local
====================
/dev/cu.Bluetooth-Incoming-Port
-------------------------------
Hardware ID: n/a
Description: n/a
/dev/cu.obd2ecu-SPPDev
----------------------
Hardware ID: n/a
Description: n/a
/dev/cu.usbmodemFA1431
----------------------
Hardware ID: USB VID:PID=2A03:0043 SER=75435353038351015271 LOCATION=250-1.4.3
Description: Arduino Uno
/dev/cu.usbserial-A6004003
--------------------------
Hardware ID: USB VID:PID=0403:6001 SER=A6004003 LOCATION=253-1.3.1
Description: FT232R USB UART - FT232R USB UART
(continues on next page)
platformio remote device monitor

Remote Serial Port Monitor

Usage

platformio remote device monitor [OPTIONS]
pio remote device monitor [OPTIONS]

# Connect to a specified agent
platformio remote --agent NAME device monitor [OPTIONS]
platformio remote -a NAME device monitor [OPTIONS]

Description

Connect to Serial Port of remote device and receive or send data in real time. *PIO 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/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 platformio 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

Examples

1. Show available options for monitor

```
> platformio remote device monitor --help

Usage: platformio remote device monitor [OPTIONS]

Options:
  -p, --port TEXT   Port, a number or a device name
```
2. Communicate with serial device and print help inside terminal

```bash
> platformio 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/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 ---
```
platformio remote run

Remote Firmware Updates

Contents

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

Usage

platformio remote run [OPTIONS]
pio remote run [OPTIONS]

# process environments using specified PIO Remote Agent
platformio remote --agent NAME run [OPTIONS]

Description

Process remotely environments which are defined in “platformio.ini” (Project Configuration File) file. By default, PIO Remote 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 platformio remote run --force-remote option. In this case, PIO Remote 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.

Built-in targets:

• clean delete compiled object files, libraries and firmware/program binaries
• upload firmware “auto-uploading” for embedded platforms
• program firmware “auto-uploading” for embedded platforms using external programmer (available only for Atmel AVR)
• buildfs Uploading files to file system SPIFFS
• uploadfs Uploading files to file system SPIFFS
• envdump dump current build environment
• size print the size of the sections in a firmware/program

--upload-port
Custom upload port of embedded board. To print all available ports use platformio 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, PIO Remote 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 platformio remote run --force-remote option. In this case, PIO Remote will automatically synchronize your project with remote machine, install required toolchains, frameworks, SDKs, etc., and process project.

Example

> platformio remote run --environment uno --target upload

PIO Plus (https://pioplus.com)
Building project locally
˓→arduino)
--------------------------------------------------------------------------------
Verbose mode can be enabled via '--v, --verbose' option
Collected 25 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/HardwareSerial.o
Compiling .pio/build/uno/FrameworkArduino/HardwareSerial0.o
Compiling .pio/build/uno/FrameworkArduino/HardwareSerial1.o
Compiling .pio/build/uno/FrameworkArduino/HardwareSerial2.o
Compiling .pio/build/uno/FrameworkArduino/HardwareSerial3.o
Compiling .pio/build/uno/FrameworkArduino/IPAddress.o

(continues on next page)
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 |
==============================================================================
| SUMMARY] ==============================================================
Environment nodemcuv2 [SKIP]
Environment uno_pic32 [SKIP]
Environment teensy31 [SKIP]
Environment uno [SUCCESS]
==============================================================================
| SUMMARY] ==============================================================
Uploading firmware remotely
------------------------------------------------------------------
platformio remote test

Helper command for remote *PIO Unit Testing*.

### Contents

- platformio remote test
  - Usage
  - Description
  - Options
  - Examples

### Usage

```
platformio remote test [OPTIONS]
pio remote test [OPTIONS]

# run tests on specified PIO Remote Agent
platformio remote --agent NAME test [OPTIONS]
```

### Description

Run remotely tests from PlatformIO based project. More details about PlatformIO *PIO 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 `platformio run --environment`

- `-i, --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, `platformio remote test --ignore "mytest*" -i "test[13]"` **--upload-port**

A port that is intended for firmware uploading. To list available ports please use `platformio 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 `platformio 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, *PIO Remote* 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 `platformio remote test --force-remote` option. In this case, *PIO Remote* 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 *PIO Unit Testing* page.

**platformio remote update**
Usage

platformio remote update [OPTIONS]
pio remote update [OPTIONS]

# start update process on the specified agents/machines
platformio 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

```
> platformio remote update

PIO Plus (https://pioplus.com)

Platform Manager
===============
Platform timsp430
--------
Upgrading toolchain-timsp430 @ 1.40603.0: [Up-to-date]
Upgrading framework-energiamsp430 @ 1.17.0: [Up-to-date]
Upgrading framework-arduinomsp430 @ 1.10601.0: [Up-to-date]
Upgrading tool-scons @ 2.4.1: [Up-to-date]

Platform freescalekinetis
--------
Upgrading freescalekinetis @ 0.0.0: [Up-to-date]
Upgrading framework-mbed @ 1.121.1: [Up-to-date]
Upgrading toolchain-gccarmnoneeabi @ 1.40804.0: [Up-to-date]
Upgrading 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 toolchain-gccarmnoneeabi @ 1.40804.0: [Up-to-date]
Updating framework-spl @ 1.121.1: [Up-to-date]
Updating toolchain-gccarmnoneeabi @ 1.40804.0: [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 toolchain-icestorm @ 1.7.0: [Up-to-date]

Platform atmelavr
--------
Updating atmelavr @ 0.0.0: [Up-to-date]
Updating toolchain-atmelavr @ 1.40801.0: [Up-to-date]
Updating toolchain-atmelavr @ 1.40801.0: [Up-to-date]

Platform espressif8266
--------
Updating espressif8266 @ 0.0.0: [Up-to-date]
Updating toolchain-xtensa @ 1.40802.0: [Up-to-date]
Updating toolchain-xtensa @ 1.40802.0: [Up-to-date]
Updating toolchain-xtensa @ 1.40802.0: [Up-to-date]
Updating toolchain-xtensa @ 1.40802.0: [Up-to-date]
Updating toolchain-xtensa @ 1.40802.0: [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 toolchain-gcclinux64 @ 1.40801.0: [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 toolchain-gccmingw32 @ 1.40800.0: [Up-to-date]

Platform teensy
--------
Updating teensy @ 0.0.0: [Up-to-date]
Updating framework-arduinoteensy @ 1.128.0: [Up-to-date]
Updating framework-arduinoteensy @ 1.128.0: [Up-to-date]
Updating toolchain-atmelavr @ 1.121.1: [Up-to-date]
Updating toolchain-atmelavr @ 1.121.1: [Up-to-date]
Updating toolchain-atmelavr @ 1.121.1: [Up-to-date]
Updating toolchain-atmelavr @ 1.121.1: [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-gccлинux32 @ 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
(continues on next page)
--------
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-gccarmlinuxgenuxabi @ 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 @ 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 @ d8422efe47: [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

platformio run
Contents

- platformio run
  - Usage
  - Description
  - Options
  - Examples

Usage

platformio run [OPTIONS]
pio run [OPTIONS]

Description

Process environments which are defined in “platformio.ini” (Project Configuration File) file

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.

Note: You can configure default targets per project environment using targets option in “platformio.ini” (Project Configuration File).

Built-in targets:

- Processing
  - clean delete compiled object files, libraries and firmware/program binaries
  - upload firmware “auto-uploading” for embedded platforms
  - debug build using Debug Configuration
  - program firmware “auto-uploading” for embedded platforms using external programmer (available only for Atmel AVR)
  - fuses set fuse bits (available only for Atmel AVR)
  - buildfs Uploading files to file system SPIFFS
  - uploadfs Uploading files to file system SPIFFS
  - size print the size of the sections in a firmware/program
- `checkprogsize` check maximum allowed firmware size for uploading
- `erase` erase device flash (not available on the all Development Platforms)

• Device
  - `monitor` automatically start `platformio device monitor` after success build operation. You can configure monitor using `Monitor options`.

• Service
  - `envdump` dump current build environment
  - `idedata` export build environment for IDE (defines, build flags, CPPPATH, etc.)

`--upload-port`

Custom upload port of embedded board. To print all available ports use `platformio 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).

`-c, --project-conf`

New in version 4.0.

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

`-j, --jobs`

New in version 4.0.

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

```
> platformio run

--------------------------------------------------------------------------------------
---
Verbose mode can be enabled via `--v, --verbose` option
Collected 36 compatible libraries
```

(continues on next page)
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
Building .pio/build/uno/firmware.hex
Calculating size .pio/build/uno/firmware.elf
AVR Memory Usage
----------------
Device: atmega328p
Program: 1034 bytes (3.2% Full)
(.text + .data + .bootloader)
Data: 9 bytes (0.4% Full)
(.data + .bss + .noinit)

[SUCCESS] Took 2.47 seconds

[Wed Sep 7 15:49:01 2016] Processing nodemcu (platform: espressif8266, board:
odemcu, framework: arduino)
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/Esp.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

[Wed Sep 7 15:49:07 2016] Processing teensy31 (platform: teensy, board: teensy31,
framework: arduino)
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/DMACchannel.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
================================================================================

--------------------------------------------------------------------------------------
Verbose mode can be enabled via `--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
================================================================================

2. Process specific environment

> platformio run -e nodemcu -e teensy31

--------------------------------------------------------------------------------------
Verbose mode can be enabled via `--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/Esp.o
Compiling .pio/build/nodemcu/FrameworkArduino/FS.o
Compiling .pio/build/nodemcu/FrameworkArduino/HardwareSerial.o

(continues on next page)
... 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 ======================================

[Wed Sep  7 15:49:07 2016] Processing teensy31 (platform: teensy, board: teensy31,
                          framework: arduino)

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
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 ======================================

3. Process specific target (clean project)

> platformio 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

(continues on next page)
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/WInterrupts.o
Removed .pio/build/uno/FrameworkArduino/wiring.o
Removed .pio/build/uno/FrameworkArduino/wiring_analog.o
Removed .pio/build/uno/FrameworkArduino/wiring_digital.o
Removed .pio/build/uno/FrameworkArduino/wiring_pulse.o
Removed .pio/build/uno/FrameworkArduino/wiring_shift.o
Removed .pio/build/uno/FrameworkArduino/WMath.o
Removed .pio/build/uno/FrameworkArduino/WString.o
Removed .pio/build/uno/src/main.o
Done cleaning
=================================[SUCCESS] Took 0.49 seconds=================================

--------------------------------------------------------------------------------------
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/AudioStream.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
4. Mix environments and targets

```bash
> platformio run -e uno -t upload

----------------------------------------------------------------------------------------------------------------------
Verbose mode can be enabled via `--v, --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.
```

(continues on next page)
platformio settings

Manage PlatformIO settings

Contents

- platformio settings
  - platformio settings get
    - Usage
    - Description
    - Settings
      - auto_update_libraries
      - auto_update_platforms
      - check_libraries_interval
      - check_platformio_interval
      - check_platforms_interval
      - enable_cache
      - strict_ssl
      - enable_telemetry
      - force_verbose
      - projects_dir
    - Examples
  - platformio settings set
    - Usage
    - Description
    - Examples
  - platformio settings reset
    - Usage
    - Description
    - Examples

platformio settings get
Usage

```
platformio settings get [NAME]
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

strict_ssl

Default No
Values Yes/No
Strict SSL for PlatformIO Services

enable_telemetry

Default Yes
Values Yes/No
Share minimal diagnostics and usage information to help us make PlatformIO better:
  • PlatformIO errors/exceptions
  • The name of used platforms, boards, frameworks (for example, “espessif”, “arduino”, “uno”, etc.)
  • The name of commands (for example, “run”, “lib list”, etc.)
  • The type of IDE (for example, “atom”, “eclipse”, etc.)
This gives us a sense of what parts of the PlatformIO is most important.
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.
Thanks a lot that keep this setting enabled.

force_verbose

Default No
Values Yes/No
Force verbose output when processing environments. This setting overrides
  • platformio run --verbose
  • platformio ci --verbose
  • platformio test --verbose
projects_dir

**Default** ~/Documents/PlatformIO/Projects

**Values** Path to folder

Default location for PlatformIO projects (PIO Home)

**Examples**

1. List all settings and theirs current values

```bash
> platformio 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>
<tr>
<td>auto_update_platforms</td>
<td>No</td>
<td>Automatically update platforms (Yes/No)</td>
</tr>
<tr>
<td>check_libraries_interval</td>
<td>7</td>
<td>Check for the library updates (days)</td>
</tr>
<tr>
<td>check_platformio_interval</td>
<td>3</td>
<td>Check for the new PlatformIO (days)</td>
</tr>
<tr>
<td>check_platforms_interval</td>
<td>7</td>
<td>Check for the platform updates (days)</td>
</tr>
<tr>
<td>enable_cache</td>
<td>Yes</td>
<td>Enable caching for API requests and Library Manager</td>
</tr>
<tr>
<td>strict_ssl</td>
<td>No</td>
<td>Strict SSL for PlatformIO Services (Yes/No)</td>
</tr>
<tr>
<td>enable_telemetry</td>
<td>Yes</td>
<td>Telemetry service? (Yes/No)</td>
</tr>
<tr>
<td>force_verbose</td>
<td>No</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 (PIO Home)</td>
</tr>
</tbody>
</table>

2. Show specified setting

```bash
$ platformio 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>

**platformio settings set**

**Usage**

```bash
platformio settings set NAME VALUE
```
Description

Set new value for the setting

Examples

Change to check for the new PlatformIO each day

```
$ platformio 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 [3]</td>
<td>Check for the new PlatformIO</td>
</tr>
</tbody>
</table>

platformio settings reset

Usage

```
platformio settings reset
```

Description

Reset settings to default

Examples

```
$ platformio settings reset
The settings have been reset!
```

<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>
<tr>
<td>auto_update_platforms</td>
<td>No</td>
<td>Automatically update platforms (Yes/No)</td>
</tr>
<tr>
<td>check_libraries_interval</td>
<td>7</td>
<td>Check for the library updates</td>
</tr>
<tr>
<td>check_platformio_interval</td>
<td>3</td>
<td>Check for the new PlatformIO</td>
</tr>
<tr>
<td>check_platforms_interval</td>
<td>7</td>
<td>Check for the platform updates</td>
</tr>
<tr>
<td>enable_cache</td>
<td>Yes</td>
<td>Enable caching for API requests and Library Manager</td>
</tr>
<tr>
<td>strict_ssl</td>
<td>No</td>
<td>Enable SSL for PlatformIO Services</td>
</tr>
<tr>
<td>enable_telemetry</td>
<td>Yes</td>
<td>Telemetry service?#enable-telemetry&gt;</td>
</tr>
</tbody>
</table>

(continues on next page)
**platformio test**

Helper command for local *PIO Unit Testing*.

**Contents**

- *platformio test*
  - *Usage*
  - *Description*
  - *Options*
  - *Examples*

**Usage**

```
platformio test [OPTIONS]
pio test [OPTIONS]
```

**Description**

Run locally tests from PlatformIO based project. More details about PlatformIO *PIO 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 *platformio 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)*.

<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, `platformio test --filter "mytest*" -i "test[13]"`

-i, --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).

<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, `platformio test --ignore "mytest*" -i "test[13]"

--upload-port

A port that is intended for firmware uploading. To list available ports please use `platformio 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 `platformio 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

New in version 4.0.

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

--without-building

Skip building stage.

--without-uploading

Skip uploading 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 PIO 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 *PIO Unit Testing* page.

**platformio update**

**Contents**

- **platformio update**
  - Usage
  - Description
  - Options
  - Examples

**Usage**

```
platformio update [OPTIONS]
pio update [OPTIONS]
```

**Description**

Check or update installed PIO Core packages, Development Platforms and global Libraries. This command is combination of 2 sub-commands:

- **platformio platform update**
- **platformio lib update**

**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

```plaintext
> platformio update

Platform Manager
================
Platform timsp430
--------
Updating timsp430 @ 0.0.0: [Up-to-date]
Updating toolchain-timsp430 @ 1.40603.0: [Up-to-date]
Updating framework-energiamsp430 @ 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]
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-arduinoespressif8266 @ 1.20300.0: [Up-to-date]
Updating sdk-esp8266 @ 1.10502.0: [Up-to-date]

Platform linux_x86_64
--------
(continues on next page)
```
Updating linux_x86_64 @ 0.0.0: [Up-to-date]
Updating toolchain-gccclinux64 @ 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]
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-gccinux32 @ 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-gccarmnfeeabi @ 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-gccarmlinuxgneabi @ 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 @ 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 @ d8422ef47: [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
platformio upgrade

Contents

• platformio upgrade
  – Usage
  – Description
  – Options
  – Examples

Usage

platformio upgrade
pio upgrade

Description

Check or upgrade PlatformIO to the latest version

Options

--dev

Use development branch.

Examples
> platformio upgrade
You are up-to-date!
PlatformIO x.x.x is currently the newest version available.

# If you have problem with permissions try:
> sudo platformio upgrade

## 1.4 PlatformIO Home

**PIO Home** allows you to interact with PlatformIO ecosystem using modern and cross-platform GUI:
- PlatformIO Projects
- **PIO Account**
- **Library Manager**
- **Development Platforms**
- **Frameworks**
- **Boards**
- **Device Manager**: serial, logical, and multicast DNS services
- Library and development platform updates.

### 1.4.1 Installation

You do not need to install **PIO Home** separately, it’s already built-in in **PlatformIO IDE** and **PlatformIO Core (CLI)**.

### 1.4.2 Quick Start

**PlatformIO IDE**

Please open **PIO Home** using (HOME) button on PIO Toolbar:
- **Atom**: PlatformIO Toolbar
- **VSCode**: PlatformIO Toolbar

**PlatformIO Core**

Please launch **PIO Home** Web-server using `platformio home` command and open in your browser `http://127.0.0.1:8008`.

You can change host and port. Please check `platformio home` command for details.
1.4.3 Demo
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

Trending

1.4. PlatformIO Home
1.5 Tutorials and Examples

1.5.1 Tutorials
Unit Testing of a “Blink” Project

The goal of this tutorial is to demonstrate how is simple to use PIO 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 *Quick Start* section and create the “Blink Project”.
2. Create a *test* directory in the project (on the same level as *src*) and place *test_main.cpp* file in it (the source code is located below).
3. Run tests using *platformio test* command.

## Project structure

```
project_dir
  lib
    README
  platformio.ini
  src
    ...
  test
    test_main.cpp
```

## Source files

- **platformio.ini**

```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]
```

(continues on next page)
platform = atmelavr
framework = arduino
board = uno

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

• test/test_main.cpp

```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);
    }
    i++;
}
```

(continues on next page)
i++; 
} else if (i == max_blinks) {
UNITY_END(); // stop unit testing
}

Test results

> platformio test -e uno --verbose

PIO Plus (https://pioplus.com) v1.4.6
Verbose mode can be enabled via `-v, --verbose` option
Collected 1 items

=============== [test/*] Building... (1/3) ===============
Processing uno (platform: atmelavr; board: uno; framework: arduino)
=============== [test/*] Building... (2/3) ===============
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
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\HardwareSerial.cpp.o
Compiling .pio\build\uno\FrameworkArduino\HardwareSerial0.cpp.o
Compiling .pio\build\uno\FrameworkArduino\HardwareSerial1.cpp.o
Compiling .pio\build\uno\FrameworkArduino\HardwareSerial2.cpp.o
Compiling .pio\build\uno\FrameworkArduino\HardwareSerial3.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.c.o
Compiling .pio\build\uno\FrameworkArduino\hooks.c.o
Compiling .pio\build\uno\FrameworkArduino\main.c.o
Compiling .pio\build\uno\FrameworkArduino\new.c.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

1.5. Tutorials and Examples 129
Compiling .pio\build\uno\FrameworkArduino\wiring_shift.c.o
Compiling .pio\build\uno\UnityTestLib\unity.o
Archiving .pio\build\uno\libFrameworkArduino.a
Archiving .pio\build\uno\libUnityTestLib.a
Indexing .pio\build\uno\libFrameworkArduino.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)

============================================================================================================
<table>
<thead>
<tr>
<th>SUMMARY</th>
</tr>
</thead>
</table>
Environment uno        [SUCCESS]
Environment teensy31   [SKIP]

============================================================================================================
<table>
<thead>
<tr>
<th>[test/*] Uploading... (2/3)</th>
</tr>
</thead>
</table>
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

(continues on next page)
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 PlatformIO IDE for VSCode to develop, run and debug a simple project with Arduino framework for ESP32-DevKitC board.

- **Level:** Beginner
- **Platforms:** Windows, Mac OS X, Linux

**Requirements:**

- Downloaded and installed PlatformIO IDE for VSCode
- ESP32-DevKitC development board
- Olimex ARM-USB-OCD or Olimex ARM-USB-TINY adapter for debugging
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 ESP32-DevKitC 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 named `main.cpp` in the `src_dir` folder and replace its content with next one:

```cpp
#include <Arduino.h>

void setup()
{
    Serial.begin(9600);
}

void loop()
{
    Serial.println("Hello world!");
    delay(1000);
}
```
After this step, we created a basic project ready for compiling and uploading.

**Compiling and Uploading the Firmware**

Now we can build the project. To compile firmware we can use the 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 the 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, we need to check if the firmware is uploaded correctly. To do this we need to open the serial monitor and check that the message from the board is received. To open serial monitor we can use the next options: Monitor option from the Project Tasks menu, Serial Monitor button in PlatformIO Toolbar, Command Palette View: Command Palette > PlatformIO: Monitor, 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 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>

PIO Unified Debugger offers the easiest way to debug the board. Firstly, we need to specify `debug_tool` in “platformio.ini” (Project Configuration File). In this tutorial 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 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 debug session, and we 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 breakpoint at void loopTask(void *pvParameters) (13 line on the screenshot below, this line can change between releases)
3. Now, please press CONTINUE/RUN button on debugging toolbar (right arrow icon)
4. Debugging session should stop at the first line of 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, 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

```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 the 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 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. phone). For example, next code declares a BLE characteristic whose value can be printed to the serial port:

```c
#include <Arduino.h>
#include <BLEDevice.h>
#include <BLEUtils.h>
#include <BLEServer.h>

#define SERVICE_UUID "4fafe5-1fb5-459e-8fcd-c5c9c31914b"
#define CHARACTERISTIC_UUID "beb5483e-36e1-4688-b7f5-ee07361b26a8"

class MyCallbacks: public BLECharacteristicCallbacks {

    virtual 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 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 the successful connection to the board, we should see one “Unknown Service” with one “Unknown Characteristic” field:
To set the value we need to send the 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 ESP32-DevKitC board that we can use as boilerplate for the next 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
- Nucleo-F401RE development board

**Contents**

- Setting Up the 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**

*PIO Unified Debugger* 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 PIO Unit Testing feature that can help us test code directly on the target board. PIO Unit Testing engine by default supports only three frameworks: Arduino, ESP-IDF, mbed, and 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):

```ini
[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:

```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;
  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;
  UartHandle.Init.Parity = UART_PARITY_NONE;
  UartHandle.Init.Mode = UART_MODE_TX_RX;
  UartHandle.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  UartHandle.Init.OverrunEnabled = UART_OVERRUN_DISABLE;
  UartHandle.Init.OverrunError = UART_ERRORS_DISABLE;
  UartHandle.Init.CRCCtrlMode = UART_CRCCtrlMode_DISABLE;
  UartHandle.Init.CRCPolynomial = UART_CRCPOLY_1;
  UartHandle.Init.ReceptionLock = UART_RECEIPTION_LOCK_OFF;
  UART_Init(&UartHandle);
}
```

(continues on next page)
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

```c
#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;

    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();
}
```

(continues on next page)
HAL_GPIO_Init(LED_GPIO_PORT, &GPIO_InitStruct);
}

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 Tool-
bar:
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](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 PlatformIO IDE for 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 PlatformIO IDE for VSCode

1.5. Tutorials and Examples
• 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)
delay(100);

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**

**PIO Unified Debugger** 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);
}
```

(continues on next page)
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:

![Image of PlatformIO Toolbar with selected "Test" task]

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):

```ini
[env:nrf52_dk]
platform = nordicnrf52
board = nrf52_dk
framework = arduino
debug_tool = jlink
; SoftDevice version
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.

```c
#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.

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 PlatformIO IDE for 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 Video Tutorials

- 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)

The Project configuration file is named platformio.ini. 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 values option can be specified in 2 ways:

1. Split values with “,” (comma + space)
2. Use multi-line format, where each new line should start with 2 spaces (minimum)

There are 2 system reserved sections:

- PlatformIO Core (CLI) settings: Section [platformio]
- Environment settings: Section [env]

The other sections can be used by users, for example, for Dynamic variables. The sections and their allowable values are described below.

1.6.1 Section [platformio]

- **Generic options**
  - description
  - default_envs
  - extra_configs
- **Directory options**
  - core_dir
  - globallib_dir
  - platforms_dir
  - packages_dir
  - cache_dir
  - build_cache_dir
A `platformio` section is used for overriding 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 `$PROJECT_HASH` template variable. You can use it in a directory path. It will be replaced by a SHA1[0:10] hash of a full project path. This is very useful to declare a global storage for project workspaces. For example, `/tmp/pio-workspaces/$PROJECT_HASH` (Unix) or `${sysenv.TEMP}/pio-workspaces/$PROJECT_HASH` (Windows). You can set a global workspace directory using system environment variable `PLATFORMIO_WORKSPACE_DIR`.

See below available directory ***_dir options.

### Generic options

#### description

Type: String | Multiple: No

Describe a project with a short information. PlatformIO uses it for `PlatformIO Home` in the multiple places.

#### default_envs

Type: String | Multiple: Yes

`platformio run` command processes all environments [env:***] by default if `platformio run --environment` option is not specified. `default_envs` allows one to define environments which should be processed by default.

Also, `PIO Unified Debugger` 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

New in version 4.0.

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

```ini
[env]
platform = espressif32
framework = espidf

[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

```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:

```ini
[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)

[env:lolin32]
```

( continues on next page )
platform = espressif32
framework = espidf
board = lolin32
build_flags = -Og

Directory options

core_dir

New in version 4.0.
Type: DirPath | Multiple: No
Is used to store 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 number of installed development platforms.
A default value is 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

New in version 4.0.
Type: DirPath | Multiple: No | Default: “core_dir/lib”
Global library storage for PlatformIO projects and Library Manager where Library Dependency Finder (LDF) looks for dependencies.
This option can also be configured by the global environment variable PLATFORMIO_GLOBALLIB_DIR.

platforms_dir

New in version 4.0.
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

New in version 4.0.
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

New in version 4.0.
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 platformio update command.
This option can also be configured by the global environment variable PLATFORMIO_CACHE_DIR.

build_cache_dir

New in version 4.0.
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
```

1.6. “platformio.ini” (Project Configuration File)
workspace_dir

New in version 4.0.
Type: DirPath | Multiple: No | Default: “Project/.pio”
A path to a project workspace directory where PlatformIO keeps by default compiled objects, static libraries, firmwares, and external library dependencies. It is used by the next options:
- build_dir
- libdeps_dir.
A 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.

libdeps_dir

Type: DirPath | Multiple: No | Default: “workspace_dir/libdeps”
Internal storage where Library Manager 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”
A path to project’s default header files. PlatformIO uses it for platformio run command. A default value is include that means that folder is located in the root of project. This path will be added to CPPPATH of 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”

A path to project’s source directory. PlatformIO uses it for `platformio run` command. A default value is `src` that means that folder is located in the root of 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 source directory should have the same name as a main source file. See example project with own source directory.

**lib_dir**

Type: DirPath | Multiple: No | Default: “Project/lib”

You can put here your own/private libraries. 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).

A default value is `lib` that means that folder is located in the root of project.

This option can also be configured by the global environment variable `PLATFORMIO_LIB_DIR`.

For example, see how can be organized Foo and Bar libraries:

```
|--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:

```
#include <Foo.h>
#include <Bar.h>

// rest H/C/CPP code
```

PlatformIO will find your libraries automatically, configure preprocessor’s include paths and build them.

**data_dir**

Type: DirPath | Multiple: No | Default: “Project/data”

Data directory to store contents and Uploading files to file system SPIFFS. A default value is `data` that means that folder is located in the root of project.
This option can also be configured by the global environment variable `PLATFORMIO_DATA_DIR`.

**test_dir**

Type: DirPath | Multiple: No | Default: “Project/test”

Directory where *PIO Unit Testing* engine will look for the tests. A default value is test that means that folder is located in the root of project.

This option can also be configured by the global environment variable `PLATFORMIO_TEST_DIR`.

**boards_dir**

Type: DirPath | Multiple: No | Default: “Project/boards”

Custom board settings per project. You can change this path with your own. A default value is boards that means that folder is located in the root of project.

By default, PlatformIO looks for boards in this order:

1. Project boards_dir
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**

New in version 4.0.

Type: DirPath | Multiple: No | Default: “Project/shared”

*PIO Remote* 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 `platformio remote run --force-remote` or `platformio remote test --force-remote` commands:

- `lib_dir`
- `include_dir`
- `src_dir`
- `boards_dir`
- `data_dir`
- `test_dir`

A default value is shared that means that folder is located in the root of project.

This option can also be configured by the global environment variable `PLATFORMIO_SHARED_DIR`. 
1.6.2 Section [env]

- Global scope [env]
- Local scope [env:NAME]
- Options

Allows declaring configuration options for building, programming, debugging, unit testing, device monitoring, library dependencies, etc.

**Global scope [env]**

New in version 4.0.

Allows declaring global options which will be shared between all [env:NAME] sections in “platformio.ini” (Project Configuration File). It is very useful if the configuration file has a lot of local scopes [env:NAME] and they have common options.

For example:

```ini
[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 2 build environments release and debug. This is the same if you duplicate all options:

```ini
[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
```
Local scope [env:NAME]

A section with `env:` prefix is used to define a build environment with local options (available only for this environment). PlatformIO uses `[env:NAME]` environments for `platformio run`, `platformio test`, `platformio debug`, and other commands.

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]`. Multiple `[env:NAME]` environments with different `NAME` are allowed.

If you have more than one build environment and you need to process only a few of them, please check `-e`, `--environment` option for commands mentioned above.

Options

General options

- `platform`
- `platform_packages`
- `framework`

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

Other forms are the same as for the `platformio platform install` command.

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

[env:custom_git_branch]
platform = https://github.com/platformio/platform-espressif8266.git#feature/stage

[env:specific_git_commit]
platform = https://github.com/platformio/platform-espressif8266.git#921855a9c530082efdb5d48b44c1f4be0e2dfa2

platform_packages

New in version 4.0.

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:

[env:override_default_toolchain]
platform = atmelavr
platform_packages =
  toolchain-gccarmnoneeabi@>1.50000.0

[env:override_framework]
platform = espressif8266
platform_packages =
  framework-arduinoespressif8266 @ https://github.com/esp8266/Arduino.git

[env:external_package]
platform = ststm32
platform_packages =
  tool-openocd
  tool-stlink-source @ https://github.com/texane/stlink.git

framework

Type: String | Multiple: Yes

Frameworks name.

Board options
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 platformio 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

An 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.

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 maximum program size
board_upload.maximum_size = 1310720

Build options

- build_type
- build_flags
  - Built-in Variables
  - Dynamic build flags
- src_build_flags
- build_unflags
- src_filter
- targets

build_type

New in version 4.0.
Type: String | Multiple: No | Default: release
See extended documentation for Build Configurations.

build_flags

Type: String | Multiple: Yes
These flags/options control preprocessing, compilation, assembly and linking processes:
<table>
<thead>
<tr>
<th>Format</th>
<th>Scope</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-D name</td>
<td>CPPDE-FINES</td>
<td>Predefine name as a macro, with definition 1.</td>
</tr>
<tr>
<td>-D name=definition</td>
<td>CPPDE-FINES</td>
<td>The contents of definition are tokenized and processed as if they appeared during translation phase three in a #define directive.</td>
</tr>
<tr>
<td>-U name</td>
<td>CPPDE-FINES</td>
<td>Cancel any previous definition of name, either built-in or provided with a -D option.</td>
</tr>
<tr>
<td>-Wp,option</td>
<td>CPPFLAGS</td>
<td>Bypass the compiler driver and pass option directly through to the preprocessor.</td>
</tr>
<tr>
<td>-Wall</td>
<td>CCFLAGS</td>
<td>Make all warnings into hard errors. Source code which triggers warnings will be rejected.</td>
</tr>
<tr>
<td>-Werror</td>
<td>CCFLAGS</td>
<td>Suppress all warnings, including those which GNU CPP issues by default.</td>
</tr>
<tr>
<td>-w</td>
<td>CCFLAGS</td>
<td>Process file as if #include &quot;file&quot; appeared as the first line of the primary source file.</td>
</tr>
<tr>
<td>-Idir file</td>
<td>CPPPATH</td>
<td>Add the directory dir to the list of directories to be searched for header files.</td>
</tr>
<tr>
<td>-Wa,option</td>
<td>ASFLAGS,CCFLAGS</td>
<td>Pass option as an option to the assembler. If option contains commas, it is split into multiple options at the commas.</td>
</tr>
<tr>
<td>-Wl,option</td>
<td>LINKFLAGS</td>
<td>Pass option as an option to the linker. If option contains commas, it is split into multiple options at the commas.</td>
</tr>
<tr>
<td>-llibrary</td>
<td>LIBS</td>
<td>Search the library named library when linking.</td>
</tr>
<tr>
<td>-Ldir dir</td>
<td>LIBPATH</td>
<td>Add directory dir to the list of directories to be searched for -l.</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:

```
[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\"

[env:specific_inclibs]
```

(continues on next page)
Built-in Variables

You can inject into build flags built-in variables, 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, name of framework
- $PROJECT_DIR, project directory
- $PROJECTCORE_DIR, PlatformIO Core directory, see core_dir
- $PROJECTBUILD_DIR, project build directory per all environments
- $BUILD_DIR, build directory per current environment
- Need more PlatformIO variables?

Please use target envdump for `platformio run --target` command to see ALL variables from a build environment.

Dynamic build flags

PlatformIO allows one to run external command/script which outputs build flags into STDOUT. PlatformIO will automatically parse this output and append flags to a build environment.

You can use any shell or programming language.

This external command will be called on each `platformio 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 PlatformIO Build System environment, please refer to Advanced Scripting.

Example:

```python
[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 “PIO_SRC_REV” macro with the latest Git revision

You will need to create a separate file named `git_rev_macro.py` and place it near `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()
print ('-DPIO_SRC_REV=%s' % revision)
```

**src_build_flags**

Type: String | Multiple: Yes

An option `src_build_flags` has the same behavior like `build_flags` but will be applied only for the project source code from `src_dir` directory.

This option can also be set by global environment variable `PLATFORMIO_SRC_BUILD_FLAGS`.

**build_unflags**

Type: String | Multiple: Yes

Remove base/initial flags which were set by 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/excluded from `src_dir` for a build process. Filter supports 2 templates:

- `+<PATH>` include template
• `<PATH>` exclude template

`PATH` MUST BE related from `src_dir`. All patterns will be applied in their order. GLOB Patterns are allowed.

By default, `src_filter` is predefined to `+<*> -<.git/> -<svn/> -<example/> -<examples/> -<test/> -<tests/>`, that means “includes ALL files, then exclude .git and svn repository folders, example... folder.

This option can also be set by global environment variable `PLATFORMIO_SRC_FILTER`.

**targets**

Type: String | Multiple: Yes

A list of targets which will be processed by `platformio run` command by default. You can enter more than one target, please split them with comma+space “,”.

The list with available targets is located in `platformio run --target`.

**Examples**

1. Build a project using Release Configuration, upload 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 `platformio run --target` command. For example:

```
# clean project
platformio run -t clean

# dump current build environment
platformio run --target envdump
```

When no targets are defined, PlatformIO will build only sources by default.

**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 `platformio 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>[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:uno]
platform = atmelavr
framework = arduino
; any port that starts with /dev/ttyUSB
upload_port = /dev/ttyUSB*

; COM1 or COM3
upload_port = COM[13]
```

**upload_protocol**

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.

**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.

```
[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**

New in version 4.0.

Type: String | Multiple: No

Override default *Development Platforms* upload command with a custom. You can pass a full upload command with arguments and options or mix with *upload_flags*.

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](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 `platformio run --target envdump` command.

**Examples**

1. Override default upload command but handle pre-uploading actions (looking for serial port, extra image preparation, etc.). Normally, the pre-configured upload options will be stored in `$UPLOADFLAGS` build variable. A classic default upload command for *Development Platforms* may look as `some-flash-bin-tool $UPLOADFLAGS $SOURCE`, where `$SOURCE` will be replaced by a real program/firmware binary.

   `$PROJECTPACKAGES_DIR` build variable points to `packages_dir`.

```
[env:program_via_AVR_ISP]
platform = atmelavr
framework = arduino
board = uno
upload_flags =
    -C
```

(continues on next page)
$PROJECTPACKAGES_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_flags =
-C $PROJECTPACKAGES_DIR/tool-avrdude/avrdude.conf
-p atmega328p
-P usb
-c
-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_command = $PROJECTPACKAGES_DIR/tool-stlink/st-flash write $SOURCE 0x8000000

Monitor options

- monitor_port
- monitor_speed
- monitor_rts
- monitor_dtr
- monitor_flags

Custom options for platformio device monitor command.
monitor_port

Type: String | Multiple: No

Port, a number or a device name. See `platformio device monitor --port`. To print all available serial ports please use `platformio 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:

```ini
[env:custom_monitor_port]
...
; Unix
monitor_port = /dev/ttyUSB1

; Windows, COM1 or COM3
monitor_port = COM[13]
```

monitor_speed

Type: Integer | Multiple: No | Default: 9600

A monitor speed (baud rate). See `platformio device monitor --baud`.

Example:

```ini
[env:custom_monitor_speedrate]
...
monitor_speed = 115200
```

monitor_rts

Type: Integer (0 or 1) | Multiple: No

A monitor initial RTS line state. See `platformio device monitor --rts`.

monitor_dtr

Type: Integer (0 or 1) | Multiple: No

A monitor initial DTR line state. See `platformio device monitor --dtr`. 
**monitor_flags**

New in version 4.0.

Type: String | Multiple: Yes

Pass extra flags and options to `platformio 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 `platformio device monitor` command.

Example:

```plaintext
[env:extra_monitor_flags]
platform = ...
board = ...
monitor_flags=  
   --parity N
   --encoding hexlify
```

**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_archive`

**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**
; one line definition (comma + space)
[env:myenv]
lib_deps = LIBRARY_1, LIBRARY_2, LIBRARY_N

; multi-line definition
[env:myenv2]
lib_deps =
  LIBRARY_1
  LIBRARY_2
  LIBRARY_N

The each line with LIBRARY_1... LIBRARY_N will be passed automatically to `platformio lib install` command. Please follow to `platformio lib install` for detailed documentation about possible values.

Example:

[env:myenv]
lib_deps =
  13
  PubSubClient
  ArduinoJson@~5.6,!=5.4
  https://github.com/gioblu/PJON.git#v2.0
  me-no-dev/ESPAsyncTCP
  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

```plaintext
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
|-- <ArduinoJson> v5.13.1
```

(continues on next page)
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:**
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:

```ini
[env:myenv]
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 PIO 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:

```ini
[env:myenv]
lib_archive = no
```

Test options

See also:
Please make sure to read *PIO Unit Testing* guide first.

- `test_filter`
- `test_ignore`
- `test_port`
- `test_speed`
test_filter

Type: String (Pattern) | Multiple: Yes

Process only the PIO Unit Testing tests where the name matches specified patterns.
Also, you can filter some tests using `platformio 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

```
[env:myenv]
test_filter = footest, bartest_*, test[13]
```

test_ignore

Type: String (Pattern) | Multiple: Yes

Ignore PIO Unit Testing tests where the name matches specified patterns.
Also, you can ignore some tests using `platformio 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

```
[env:myenv]
test_ignore = footest
              bartest_*
              test[13]
```

test_port

Type: String (Pattern) | Multiple: No

This option specifies communication interface (Serial/UART) between PlatformIO PIO 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 platformio 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

PIO 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.

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);
}

```

(continues on next page)
### 2. STM32Cube HAL and Nucleo-F401RE: debugging and unit testing

#### test_build_project_src

Type: Bool (yes or no) | Multiple: No | Default: no

Force PIO 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 = ...
[env:myenv]
test_build_project_src = yes
```

#### Debugging options

**See also:**

Please make sure to read PIO Unified Debugger guide first.

- `debug_tool`
- `debug_init_break`
- `debug_init_cmds`
- `debug_extra_cmds`
- `debug_load_cmds`
- `debug_load_mode`
- `debug_server`
- `debug_port`
- `debug_svd_path`

#### 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_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**

```ini
[env:debug]
platform = ...
board = ...

; 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:
debug_init_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:

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.

default_extra_cmds

New in version 4.0.

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:

```
[env:debug]
platform = ...
board = ...
download_commands =
  monitor flash erase_sector 0 0 11
  load
download_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 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.

download_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:

```
[env:debug]
platform = ...
board = ...
download_server = /path/to/debugging/server
  arg1
  arg2
  ...
  argN
download_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 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**

**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 PIO Remote machine, please put them to shared_dir.

### 1.6.3 Build Configurations

New in version 4.0.0.

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.

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 platformio run --target command.

**Note:** PIO Unified Debugger automatically switches to debug configuration when you do project debugging from PlatformIO IDE or use platformio debug command.

To avoid project rebuilding, please create a separate build environment and add build_type = debug. See example below where mydebug build environment will be used automatically by PIO Unified Debugger:

```ini
[env]
platform = ...
board = ...
framework = ...
... other common configuration

[env:myrelease]
(continues on next page)```
Please note that you can set a default build environment per a project using `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 all build environments, please take a look at “Global scope” in Section `[env]`.

Example:

```ini
[env]
; Unix
lib_extra_dirs = ${sysenv.HOME}/Documents/Arduino/libraries
; Windows
lib_extra_dirs = ${sysenv.HOMEDRIVE}${sysenv.HOMEPATH}\Documents\Arduino\libraries

; 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 = ArduinoJson@>5.6.0

[env:uno]
platform = atmelavr
framework = arduino
board = uno
build_flags = ${extra.build_flags}
lib_deps =
  ${extra.lib_deps_builtin}
  ${extra.lib_deps_external}
```

(continues on next page)
[env:nodemcuv2]
platform = espressif8266
framework = arduino
board = nodemcuv2
build_flags = ${extra.build_flags} -DSSID_NAME=HELLO -DSSID_PASWORD=WORLD
lib_deps =
  ${extra.lib_deps_builtin}
  ${extra.lib_deps_external}
  PubSubClient@2.6
  OneWire

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

Example

[platformio]
default_envs = nodemcuv2

; 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:nodemcuv2]
platform = espressif8266
framework = arduino
board = nodemcuv2

; Build options
build_flags =
  ${common_env_data.build_flags}
  -DSSID_NAME=HELLO
  -DSSID_PASWORD=WORLD
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

```bash
set VARIABLE_NAME=VALUE
```

# Windows GUI -> https://www.youtube.com/watch?v=bEroNNzqlF4

# Unix (bash, zsh)

```bash
export VARIABLE_NAME=VALUE
```

# Unix (fish)

```bash
set -x VARIABLE_NAME VALUE
```

## 1.7.1 General

PlatformIO uses General environment variables for the common operations/commands.

### CI

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 in to PIO Account. It is very useful for Continuous Integration systems and PIO Remote operations where you are not able manually authorize.

You can get own Personal Authentication Token using `platformio account token` command.

### PLATFORMIO_FORCE_COLOR

Force to output color ANSI-codes even if the output is a pipe (not a tty). The possible values are true and false. Default is `PLATFORMIO_FORCE_COLOR=false`.

### 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.

### 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**
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 `platformio remote agent start --working-dir`.

**PLATFORMIO_LIB_EXTRA_DIRS**
Allows one to set “platformio.ini” (Project Configuration File) option `lib_extra_dirs`.

1.7. Environment variables
1.7.3 Building

PLATFORMIO_BUILD_FLAGS
Allows one to set “platformio.ini” (Project Configuration File) option build_flags.

Examples:

# Unix:
export PLATFORMIO_BUILD_FLAGS=-DFOO
export PLATFORMIO_BUILD_FLAGS=-DFOO -DBAR=1 -DFLOAT_VALUE=1.23457e+07
export PLATFORMIO_BUILD_FLAGS="-DWIFI_PASS="My password"" -DWIFI_SSID="My ssid name"

# Windows:
SET PLATFORMIO_BUILD_FLAGS=-DFOO
SET PLATFORMIO_BUILD_FLAGS=-DFOO -DBAR=1 -DFLOAT_VALUE=1.23457e+07
SET PLATFORMIO_BUILD_FLAGS="-DWIFI_PASS="My password"" "-DWIFI_SSID="My ssid name"

PLATFORMIO_SRC_BUILD_FLAGS
Allows one to set “platformio.ini” (Project Configuration File) option src_build_flags.

PLATFORMIO_SRC_FILTER
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 platformio 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 STRICT_SSL**

Allows one to override setting `strict_ssl`.

**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 Python language knowledge.

We highly recommend to take a look at Dynamic build flags option where you can use any programming language. Also, this option is useful if you need to apply changes to the project before building/uploading process:

- 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

More details **Dynamic build flags**.
PlatformIO Build System allows one to extend build process with the custom extra_scripts using Python interpreter and SCons construction tool. Build and upload flags, targets, toolchains data, and other information are stored in SCons Construction Environments.

**Warning:** You cannot run/debug these scripts directly with Python interpreter. They will be loaded automatically when you process project environment using platformio run command.

### 1.8.1 Launch types

There are 2 launch type of extra scripts:

1. **PRE** - executes before a main script of Development Platforms
2. **POST** - executes after a 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, “platformio.ini” (Project Configuration File)

```ini
[env:my_env_1]
platform = ...
; without prefix, POST script
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 global environment variable PLATFORMIO_EXTRA_SCRIPTS.

### 1.8.2 Construction Environments

There are 2 built-in construction environments which PlatformIO Build System uses to process a project:

- **env, Import("env")** - global construction environment which is used for Development Platforms and Frameworks build scripts, upload tools, Library Dependency Finder (LDF), and other internal operations
• `projenv`. `Import("projenv")` - isolated construction environment which is used for processing of a project source code located in `src_dir`. Please note that `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.

```python
my_pre_extra_script.py:

```python
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")
])
```

```python
my_post_extra_script.py:

```python
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")
])
```

(continues on next page)
See examples below how to import construction environments and modify existing data or add new.

**1.8.3 Before/Pre and After/Post actions**

PlatformIO Build System has rich API that allows one to attach different pre-/post actions (hooks) using `env.AddPreAction(target, callback)` or `env.AddPreAction(target, [callback1, callback2, ...])` function. A first argument `target` can be a name of target that is passed using `platformio run --target` command, a name of built-in targets (buildprog, size, upload, program, buildfs, uploadfs, uploadfsota) or path to file which PlatformIO processes (ELF, HEX, BIN, OBJ, etc.).

**Examples**

`extra_script.py` file is located on the same level as `platformio.ini`.

`platformio.ini`:

```
[env:pre_and_post_hooks]
extra_scripts = post:extra_script.py
```

`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")
```

(continues on next page)
# do some actions

```python
print("Current build targets", map(str, BUILD_TARGETS))
```

```python
env.AddPreAction("upload", before_upload)
env.AddPostAction("upload", after_upload)
```

```python
# # Custom actions when building program/firmware
#

```python
env.AddPreAction("buildprog", callback...)
env.AddPostAction("buildprog", callback...)
```

```python
# # Custom actions for specific files/objects
#

```python
env.AddPreAction("$BUILD_DIR/${PROGNAME}.elf", [callback1, callback2,...])
env.AddPostAction("$BUILD_DIR/${PROGNAME}.hex", callback...)
```

```python
# custom action before building SPIFFS image. For example, compress HTML, etc.
env.AddPreAction("$BUILD_DIR/spiffs.bin", callback...)
```

```python
# custom action for project's main.cpp
env.AddPostAction("$BUILD_DIR/src/main.cpp.o", callback...)
```

```python
# Custom HEX from ELF
env.AddPostAction(
    "$BUILD_DIR/${PROGNAME}.elf",
                             "$BUILD_DIR/${PROGNAME}.elf", "$BUILD_DIR/${PROGNAME}.hex"]), "Building $BUILD_DIR/${PROGNAME}.hex")
)
```

### 1.8.4 Custom target

There is a list with built-in targets which could be processed using `platformio run --target` option. You can create unlimited number of the own targets and declare custom handlers for them.

We will use SCons’s `Alias(alias, [targets, [action]]) , env.Alias(alias, [targets, [action]])` function to declare a custom target/alias.

**Command shortcut**

Create a custom node target (alias) which will print a NodeJS version

```bash
platformio ini:
```

```
[env:myenv]
platform = ...
...
extra_scripts = extra_script.py
```

1.8. Advanced Scripting
extra_script.py:

```python
import("env")
env.AlwaysBuild(env.Alias("node", None, ["node --version"])))
```

Now, run `pio run -t node`.

**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.AlwaysBuild(env.Alias("ota",
    "$BUILD_DIR/${PROGNAME}.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 `platformio 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
try:
    import configparser
except ImportError:
    import ConfigParser as configparser

import("env")
config = configparser.ConfigParser()
config.read("platformio.ini")
host = config.get("env_custom_target", "custom_ping_host")

def mytarget_callback(*args, **kwargs):
    print("Hello PlatformIO!")
```

(continues on next page)
env.Execute("ping " + host)

env.AlwaysBuild(env.Alias("ping", None, mytarget_callback))

1.8.5 Examples

The beast examples are PlatformIO development platforms. Please check builder folder for the main and framework scripts.

Custom options in platformio.ini

platformio.ini:

```
[env:my_env]
platform = ...
extra_scripts = extra_script.py

custom_option1 = value1
custom_option2 = value2
```

extra_script.py:

```
try:
    import configparser
except ImportError:
    import ConfigParser as configparser

config = configparser.ConfigParser()
config.read("platformio.ini")

value1 = config.get("my_env", "custom_option1")
value2 = config.get("my_env", "custom_option2")
```

Split C/C++ build flags

platformio.ini:

```
[env:my_env]
platform = ...
extra_scripts = extra_script.py
```

extra_script.py (place it near platformio.ini):

```
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"'])
```

(continues on next page)
# General options that are passed to the C++ compiler

```python
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
```

**extra_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 `platformio run -t upload`.

Please note that some development platforms can have more than 1 upload command. For example, *Atmel AVR* has `UPLOADHEXCMD` (firmware) and `UPLOADEPCMD` (EEPROM data).

See examples below:

**Template**

**platformio.ini:**

```ini
[env:my_custom_upload_tool]
platform = ...
; place it into the root of project or use full path
extra_scripts = extra_script.py
upload_protocol = custom
; each flag in a new line
upload_flags =
    -arg1
    -arg2
    -argN
```
extra_script.py (place it near platformio.ini):

```python
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 $UPLOADFLAGS $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:

```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
upload_flags =
    -f
    scripts/interface/stlink.cfg
    -f
    scripts/target/stm32f4x.cfg
```

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",
```

(continues on next page)
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:
```

```
[env:env_custom_prog_name]
platform = espressif8266
board = nodemcuv2
framework = arduino
build_flags = -D VERSION=13
extra_scripts = pre:extra_script.py
```

```
extra_script.py:
```

```
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 

```
#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 ~/.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*:
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:**

```ini
[env:uno]
platform = atmelavr
board = uno
framework = arduino
extra_scripts = pre:apply_patches.py
```

```
apply_patches.py:
from os.path import join, isfile

FRAMEWORK_DIR = env.PioPlatform().get_package_dir("framework-arduinoavr")
patchflag_path = join(FRAMEWORK_DIR, ".patching-done")

# skip patch process if we did it before
if isfile(join(FRAMEWORK_DIR, ".patching-done")):
    env.Exit(0)

original_file = join(FRAMEWORK_DIR, "variants", "standard", "pins_arduino.h")
patched_file = join("patches", "1-framework-arduinoavr-add-pin-a8.patch")

assert isfile(original_file) and isfile(patched_file)
env.Execute("patch %s %s" % (original_file, patched_file))

# env.Execute("touch " + patchflag_path)

def _touch(path):
    with open(path, "w") as fp:
        fp.write(""

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 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:

1.8. Advanced Scripting
platformio.ini:

```
[env:uno]
platform = atmelavr
board = uno
framework = arduino
extra_scripts = pre:custom_hwids.py
```

custom_hwids.py:

```
Import("env")

board_config = env.BoardConfig()
board_config.update("build.hwids", [
    ["0x2341", "0x0243"],
    ["0x2A03", "0x0043"]
])
```

1.9 Library Manager

**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:
You can manage different library storages using `platformio lib --global` or `platformio lib --storage-dir` options. If you change current working directory in terminal to project folder, then `platformio 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). You can use library ID, Name or even repository URL. For example,

```
[env:myenv]
platform = ...
framework = ...
board = ...
lib_deps =
  13
  PubSubClient
  ArduinoJson@~5.6,!=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 `platformio 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

- **Atom:** “Menu: PlatformIO > Rebuild C/C++ Project Index (Autocomplete, Linter)”
- **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:

- 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.
Library Dependency Finder is a core part of PlatformIO Build System that operates with the C/C++ source files and looks for \#include ... directives.

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 significantly reduce 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. \texttt{lib_extra_dirs} - extra storages per build environment
2. \texttt{lib_dir} - own/private library storage per project
3. \texttt{libdeps_dir} - project dependency storage used by Library Manager
4. \texttt{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 ...) from the libraries. It also parses C, CC, CPP files from libraries which have the same name as included header file. **Does not evaluates 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 evaluates 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 the library source file, and not the 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 “Bar” library (contains #include <bar.h>)
- #include <foo.h> is located in one of the project source files

Here are 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 “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 “Bar” library because it doesn’t know about 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 “Foo” library, then it parses all sources from “Foo” library and finds `Foo/extra.cpp` that depends on `#include <bar.h>`. Secondly, it will parse all sources from “Bar” library and 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,

```ini
[env:myenv]
lib_ldf_mode = chain+
build_flags = -D MY_PROJECT_VERSION=13
```

```c
mylib.h

#ifndef PROJECT_VERSION
// include common file for the project
#include "my_common.h"
#endif

#if 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 development library. It allows developers to keep project in own structure and define:

- location of source code
- examples list
- compatible frameworks and platforms
- library dependencies
- advanced build settings

PlatformIO Library Crawler uses library.json manifest to extract source code from developer’s location and keeps a cleaned library in own Library Registry.

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.

### Fields

- `name`
- `description`
- `keywords`
- `authors`
- `repository`
- `version`
- `license`
- `downloadUrl`
- `homepage`
- `export`  
  - `include`
  - `exclude`
- `frameworks`
- `platforms`
- `dependencies`
- `examples`
- `build`  
  - `flags`
  - `unflags`
  - `includeDir`
  - `srcDir`
  - `srcFilter`
extraScript
- libArchive
- libLDFMode
- libCompatMode

Examples

name

**Required** | Type: *String* | Max. Length: 50

A name of the library.

- Must be unique.
- Should be slug style for simplicity, consistency and compatibility. Example: *Arduino-SPI*
- Title Case, Aa-z, can contain digits and dashes (but not start/end with them).
- Consecutive dashes are not allowed.

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* | 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 ,

authors

**Required** if *repository* field is not defined | Type: *Object* or *Array*

An author contact information

- name Full name (**Required**)
- email
- url An author’s contact page
- maintainer Specify “maintainer” status

Examples:
"authors":
{
   "name": "John Smith",
   "email": "me@john-smith.com",
   "url": "http://www.john-smith/contact"
}
...

"authors":
[
   {
      "name": "John Smith",
      "email": "me@john-smith.com",
      "url": "http://www.john-smith/contact"
   },
   {
      "name": "Andrew Smith",
      "email": "me@andrew-smith.com",
      "url": "http://www.andrew-smith/contact",
      "maintainer": true
   }
]

Note: You can omit authors field and define repository field. Only GitHub-based repository is supported now. In this case PlatformIO Library Registry Crawler will use information from GitHub API Users.

repository

Required if downloadUrl field is not defined | 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:

"repository":
{
   "type": "git",
   "url": "https://github.com/foo/bar.git"
}

version

Required if repository field is not defined | Type: String | Max. Length: 20
A version of the current library source code. Can contain a-z, digits, dots or dash. Semantic Versioning IS RECOMMENDED.
Case 1 version and repository fields are defined. The repository is hosted on GitHub or Bitbucket.

*PlatformIO Library Registry Crawler* will lookup for release tag named as value of version or with v prefix (you do not need to pass this v prefix to the version field).

Case 2 version and repository fields are defined and repository does not contain tag/release with value of version.

*PlatformIO Library Registry Crawler* will use the latest source code from repository and link it with specified version. If repository branch is not specified, then default branch will be used. Also, if you push new commits to repository and do not update version field, the library will not be updated until you change the version.

Case 3 version field is not defined and repository field is defined.

*PlatformIO Library Registry Crawler* will use the VCS revision from the latest commit as “current version”. For example, 13 (SVN) or first 10 chars of SHA digest e4564b7da4 (Git). If repository branch is not specified, then default branch will be used.

We recommend to use version field and specify the real release version and make appropriate tag in the repository. In other case, users will receive updates for library with each new commit to repository.

Note:

*PlatformIO Library Registry Crawler* updates library only if:

- the version is changed
- library.json is modified

Example:

```
"repository": {
  "type": "git",
  "url": "https://github.com/foo/bar.git"
},
"version": "1.0.0"
```

**license**

Optional | Type: String

A license of the library. You can check the full list of SPDX license IDs. Ideally you should pick one that is OSI approved.

```
"license": "Apache-2.0"
```

**downloadUrl**

Required if repository field is not defined | Type: String

It is the HTTP URL to the archived source code of library. It should end with the type of archive (.zip or .tar.gz).

Note: downloadUrl has higher priority than repository.
Example with detached release/tag on GitHub:

```
"version": "1.0.0",
"downloadUrl": "https://github.com/foo/bar/archive/v1.0.0.tar.gz",
"include": "bar-1.0.0"
```

See more library.json Examples.

**homepage**

*Optional* | Type: String | Max. Length: 255

Home page of library (if is different from repository url).

**export**

*Optional* | Type: Object

Explain PlatformIO Library Crawler which content from the repository/archive should be exported as “source code” of the library. This option is useful if need to exclude extra data (test code, docs, images, PDFs, etc). It allows one to reduce size of the final archive.

Possible options:

```
  • include
  • exclude
```

**include**

*Optional* | Type: String or Array | Glob Pattern

If include field is a type of String, then PlatformIO Library Registry Crawler will recognize it like a “relative path inside repository/archive to library source code”. See example below where the only source code from the relative directory LibrarySourceCodeHere will be included.

```
"include": "some/child/dir/LibrarySourceCodeHere"
```

If include field is a type of Array, then PlatformIO Library Registry Crawler firstly will apply exclude filter and then include only directories/files which match with include patterns.

Example:

```
"export": {
  "include": [
    "dir/*.[ch]pp",
    "dir/examples/*",
    "*/*/.*.h"
  ]
}
```

Pattern Meaning
See more `library.json Examples`.

**exclude**

*Optional* | **Type**: String or 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 types are defined in the *Frameworks* section.
If the library is compatible with the all frameworks, then you can use * symbol:

```
"frameworks": "*"
```

**platforms**

*Optional* | **Type**: String or Array
---|---
A list with compatible platforms. The available platform types are defined in *Development Platforms* section.
If the library is compatible with the all platforms, then you can use * symbol:

```
"platforms": "*"
```

**dependencies**

*Optional* | **Type**: Array or Object
---|---
A list of dependent libraries. They will be installed automatically with `platformio lib install` command.
Allowed requirements for dependent library:
- name | **Type**: String
- version | **Type**: String
- authors | **Type**: String or Array
- frameworks | **Type**: String or Array
- platforms | **Type**: String or Array
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 `platformio lib install` command.

Example:

```json
"dependencies": [
  {
    "name": "Library-Foo",
    "authors": [
      "Jhon Smith",
      "Andrew Smith"
    ]
  },
  {
    "name": "Library-Bar",
    "version": "~1.2.3"
  },
  {
    "name": "lib-from-repo",
    "version": "https://github.com/user/package.git#1.2.3"
  }
]
```

A short definition of dependencies is allowed:

```json
"dependencies": {
  "mylib": "1.2.3",
  "lib-from-repo": "githubuser/package"
}
```

See more library.json Examples.

**examples**

*Optional* | Type: String or Array | Glob Pattern
--- | --- | ---

A list of example patterns. This field is predefined with default value:

```json
"examples": [
  "[Ee]xamples/*.c",
  "[Ee]xamples/*.cpp",
  "[Ee]xamples/*.ino",
  "[Ee]xamples/*.pde",
  "[Ee]xamples/*/.*.c",
  "[Ee]xamples/*/.*.cpp",
  "[Ee]xamples/*/.*.ino",
  "[Ee]xamples/*/.*.pde",
  "[Ee]xamples/*/.*/*.c",
  "[Ee]xamples/*/.*/*.cpp",
  "[Ee]xamples/*/.*/*.ino",
  "[Ee]xamples/*/.*/*.pde"
]
```

(continues on next page)
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

New in version 4.0.

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

New in version 4.0.
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 from a root of 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
      bar
        hal.c
        hal.h
      foo
        hal.c
        hal.h
    library.json
    SomeLib.c
    SomeLib.h

platformio.ini

src
  test.c
```

**platformio.ini**

```
[env:foo]
platform = native
build_flags = -DHAL=foo

[env:bar]
platform = native
build_flags = -DHAL=bar
```

**library.json**
extra_script.py

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>", "+%<*>" % 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 PIO 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: Integer
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>"]
}
```

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

*PlatformIO Library Manager* doesn’t have any requirements to a library source code structure. The only one requirement is library’s manifest file - `library.json`, `library.properties` or `module.json`. It can be located inside your library or in the another location where *PlatformIO Library Registry Crawler* will have `HTTP` access.
Updates to existing libraries are done every 24 hours. In case a more urgent update is required, you can post a request on PlatformIO community.

Contents

- Source Code Location
  - At GitHub
  - Under VCS (SVN/GIT)
  - Self-hosted
- Register
- Examples

Source Code Location

There are several ways how to share your library with the whole world (see examples).

You can hold a lot of libraries (split into separated folders) inside one of the repository/archive. In this case, you need to specify include option of export field to relative path to your library’s source code.

At GitHub

Recommended

If a library source code is located at GitHub, then you need to specify only these fields in the library.json:

- name
- version (is not required, but highly recommended for new Library Manager)
- keywords
- description
- repository

PlatformIO Library Registry Crawler will populate the rest fields, like authors with an actual information from GitHub.

Example, DallasTemperature:

```json
{
    "name": "DallasTemperature",
    "keywords": "onewire, l-wire, bus, sensor, temperature",
    "description": "Arduino Library for Dallas Temperature ICs (DS18B20, DS18S20, DS1822, DS1820)",
    "repository": {
        "type": "git",
        "url": "https://github.com/milesburton/Arduino-Temperature-Control-Library.git"
    },
    "authors": [
    ]
}
```

(continues on next page)
"name": "Miles Burton",
"email": "miles@mnetcs.com",
"url": "http://www.milesburton.com",
"maintainer": true
],
{
"name": "Tim Newsome",
"email": "nuisance@casualhacker.net"
},
{
"name": "Guil Barros",
"email": "gfbarros@bappos.com"
},
{
"name": "Rob Tillaart",
"email": "rob.tillaart@gmail.com"
]
},
"dependencies": {
"name": "OneWire",
"authors": "Paul Stoffregen",
"frameworks": "arduino"
},
"version": "3.7.7",
"frameworks": "arduino",
"platforms": "*"
}

**Under VCS (SVN/GIT)**

*PlatformIO Library Registry Crawler* can operate with a library source code that is under VCS control. The list of required fields in the *library.json* will look like:

- **name**
- **keywords**
- **description**
- **authors**
- **repository**

Example:

```json
{
"name": "XBee",
"keywords": "xbee, protocol, radio",
"description": "Arduino library for communicating with XBees in API mode",
"authors": {
"name": "Andrew Rapp",
"email": "andrew.rapp@gmail.com",
"url": "https://code.google.com/u/andrew.rapp@gmail.com/
}
,"repository":
```
 SELF-HOSTED

You can manually archive (Zip, Tar.Gz) your library source code and host it in the Internet. Then you should specify the additional fields, like version and downloadUrl. The final list of required fields in the library.json will look like:

- name
- keywords
- description
- authors
- version
- downloadUrl

```json
{
   "name": "OneWire",
   "keywords": "onewire, 1-wire, bus, sensor, temperature, ibutton",
   "description": "Control devices (from Dallas Semiconductor) that use the One Wire protocol (DS18S20, DS18B20, DS2482 and etc)",
   "authors": {
      "name": "Paul Stoffregen",
      "url": "http://www.pjrc.com/teensy/td_libs_OneWire.html"
   },
   "version": "2.2",
   "downloadUrl": "http://www.pjrc.com/teensy/arduino/libraries/OneWire.zip",
   "export": {
      "include": "OneWire"
   },
   "frameworks": "arduino",
   "platforms": "atmelavr"
}
```

REGISTER

The registration requirements:

- A library must adhere to the library manifest specification - library.json, library.properties or module.json.
- There must be public HTTP access to the library manifest file.

Now, you can register your library and allow others to install it.
Examples

Command:

```bash
$ platformio lib register http://my.example.com/library.json
$ platformio lib register http://my.example.com/library.properties
$ platformio lib register http://my.example.com/module.json
```

- GitHub + detached release
- Dependencies by author and framework
- Multiple libraries in the one repository

1.10 Development Platforms

PlatformIO ecosystem has decentralized architecture. Build scripts, toolchains, the pre-built tools for the popular OS (Mac OS X, Linux (+ARM) and Windows) are organized into the multiple development platforms.

Each development platform contains:

- **PlatformIO Build System** based build scripts for the supported frameworks and SDKs
- Pre-configured presets for embedded boards
- Pre-compiled toolchains and relative tools for multiple architectures.

A platform name or its specific version could be specified using `platform` option in “`platformio.ini`” (Project Configuration File).

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:

- Compass
- Framework
- IMU
- INS
- Leveler
- VG_AHRS

Debugging

PIO Unified Debugger - “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 PIO Unified Debugger 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 300ZA</td>
<td>STM32F405RG</td>
<td>120MHz</td>
<td>1MB</td>
<td>128KB</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

```python
; Latest stable version
[env:latest_stable]
platform = aceinna_imu
board = ...

; Custom stable version
[env:custom_stable]
platform = aceinna_imu@x.y.z
board = ...
```

Upstream

```python
[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>SEGGER J-Link Software and Documentation Pack</td>
</tr>
<tr>
<td>tool-openocd</td>
<td>OpenOCD</td>
</tr>
<tr>
<td>toolchain-gccarmnoneabi</td>
<td>gcc-arm-embedded</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 `platformio boards` command or [PlatformIO Boards Explorer](#).
- For more detailed `board` information please scroll tables below by horizontal.

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 OpenIMU 300ZA</td>
<td>External</td>
<td>STM32F405RG</td>
<td>120MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
</tbody>
</table>
Atmel AVR

Configuration  \texttt{platform} = \texttt{atmelavr}

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.

For more detailed information please visit vendor site.

Contents

- Configuration
- Examples
- Stable and upstream versions
- Packages
- Frameworks
- Boards

Configuration

Upload using Programmer

Upload EEPROM data

Fuses

- Custom Fuses

Upload using Programmer

To upload firmware using programmer you need to use \texttt{program} target instead of \texttt{upload} for \texttt{platformio run --target} command. For example, \texttt{platformio run -t program}.

Warning: Upload options like \texttt{upload_port} don't work as expected with \texttt{platformio run -t program}. You need to use \texttt{upload_flags} if you want to specify custom port or speed (see examples below).

Note: List of avrdude supported programmers are accessible with \texttt{avrdude -c ?}

Configuration for the programmers:

- AVR ISP
• AVRISP mkII

```yaml
[env:myenv]
platform = atmelavr
framework = arduino
upload_protocol = stk500v2
; each flag in a new line
upload_flags =
   -Pusb
```

• USBtinyISP

```yaml
[env:myenv]
platform = atmelavr
framework = arduino
upload_protocol = usbtiny
```

• ArduinoISP

```yaml
[env:myenv]
platform = atmelavr
framework = arduino
upload_protocol = arduinoisp
```

• USBasp

```yaml
[env:myenv]
platform = atmelavr
framework = arduino
upload_protocol = usbasp
; each flag in a new line
upload_flags =
   -Pusb
```

• Parallel Programmer

```yaml
[env:myenv]
platform = atmelavr
framework = arduino
upload_protocol = dapa
; each flag in a new line
upload_flags =
   -F
```

• Arduino as ISP
To upload EEPROM data (from EEMEM directive) you need to use `uploadeep` target instead of `upload` for `platformio run --target` command. For example, `platformio run -t uploadeep`.

### Fuses

PlatformIO has built-in target named `fuses` for setting fuse bits. The default fuse bits are predefined in board manifest file in `fuses` section. For example, Arduino Uno Fuses.

To set fuse bits you need to use target `fuses` for `platformio run --target` command.

### Custom Fuses

You can specify custom fuse bits. Please create custom `extra_scripts` and override default “fuses” command:

```ini
[env:custom_fuses]
platform = atmelavr
extra_scripts = extra_script.py
```

```py
extra_script.py:
```

```bash
platformio run --target fuses
```
Examples

Examples are listed from Atmel AVR development platform repository:

- arduino-blink
- arduino-external-libs
- arduino-internal-libs
- arduino-own-src_dir
- digitstump-mouse
- engduino-magnetometer
- native-blink
- simba-blink

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

```ini
; Latest stable version
[env:latest_stable]
platform = atmelavr
board = ...

; Custom stable version
[env:custom_stable]
platform = atmelavr@x.y.z
board = ...
```

Upstream

```ini
[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-arduinoavr</td>
<td>Arduino Wiring-based Framework (AVR Core)</td>
</tr>
<tr>
<td>framework-simba</td>
<td>Simba Framework</td>
</tr>
<tr>
<td>tool-avrdude</td>
<td>AVRDUDE</td>
</tr>
<tr>
<td>tool-micronucleus</td>
<td>Micronucleus</td>
</tr>
<tr>
<td>toolchain-atmelavr</td>
<td>avr-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>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached 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. It aims to make embedded programming easy and portable.</td>
</tr>
</tbody>
</table>

Boards

Note:
- You can list pre-configured boards by `platformio boards` command or [PlatformIO Boards Explorer](#).
- For more detailed board information please scroll tables below by horizontal.
### 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>No</td>
<td>ATMEGA32U4</td>
<td>8MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Adafruit Circuit Playground Classic</td>
<td>No</td>
<td>ATMEGA32U4</td>
<td>8MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Adafruit Feather 328P</td>
<td>No</td>
<td>ATMEGA328P</td>
<td>8MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Adafruit Feather 32u4</td>
<td>No</td>
<td>ATMEGA32U4</td>
<td>8MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Adafruit Flora</td>
<td>No</td>
<td>ATMEGA32U4</td>
<td>8MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Adafruit Gemma</td>
<td>No</td>
<td>ATTINY85</td>
<td>8MHz</td>
<td>8KB</td>
<td>512B</td>
</tr>
<tr>
<td>Adafruit ItsyBitsy 3V/8MHz</td>
<td>No</td>
<td>ATMEGA32U4</td>
<td>8MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Adafruit ItsyBitsy 5V/16MHz</td>
<td>No</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Adafruit Metro</td>
<td>No</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>No</td>
<td>ATMEGA328P</td>
<td>12MHz</td>
<td>28KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Adafruit Pro Trinket 3V/12MHz (USB)</td>
<td>No</td>
<td>ATMEGA328P</td>
<td>12MHz</td>
<td>28KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Adafruit Pro Trinket 5V/16MHz (FTDI)</td>
<td>No</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>28KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Adafruit Pro Trinket 5V/16MHz (USB)</td>
<td>No</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>28KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Adafruit Trinket 3V/8MHz</td>
<td>No</td>
<td>ATTINY85</td>
<td>8MHz</td>
<td>8KB</td>
<td>512B</td>
</tr>
<tr>
<td>Adafruit Trinket 5V/16MHz</td>
<td>No</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>No</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Alorium Sno</td>
<td>No</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Alorium XLR8</td>
<td>No</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>No</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>No</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Arduboy DevKit</td>
<td>No</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>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino BT ATmega168</td>
<td>No</td>
<td>ATMEGA168</td>
<td>16MHz</td>
<td>14KB</td>
<td>1KB</td>
</tr>
<tr>
<td>Arduino BT ATmega328</td>
<td>No</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>28KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Arduino Duemilanove or Diecimila ATmega168</td>
<td>No</td>
<td>ATMEGA168</td>
<td>16MHz</td>
<td>14KB</td>
<td>1KB</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>Arduino Duemilanove or Diecimila ATmega328</td>
<td>No</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>30KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Arduino Explora</td>
<td>No</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Arduino Ethernet</td>
<td>No</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Arduino Fio</td>
<td>No</td>
<td>ATMEGA328P</td>
<td>8MHz</td>
<td>30KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Arduino Industrial 101</td>
<td>No</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Arduino Leonardo</td>
<td>No</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Arduino LilyPad ATmega168</td>
<td>No</td>
<td>ATMEGA168</td>
<td>8MHz</td>
<td>14KB</td>
<td>1KB</td>
</tr>
<tr>
<td>Arduino LilyPad ATmega328</td>
<td>No</td>
<td>ATMEGA328P</td>
<td>8MHz</td>
<td>30KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Arduino LilyPad USB</td>
<td>No</td>
<td>ATMEGA32U4</td>
<td>8MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Arduino Mega ADK</td>
<td>No</td>
<td>ATMEGA2560</td>
<td>16MHz</td>
<td>248KB</td>
<td>8KB</td>
</tr>
<tr>
<td>Arduino Mega or Mega 2560 ATmega1280</td>
<td>No</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>No</td>
<td>ATMEGA2560</td>
<td>16MHz</td>
<td>248KB</td>
<td>8KB</td>
</tr>
<tr>
<td>Arduino Micro</td>
<td>No</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Arduino Mini ATmega168</td>
<td>No</td>
<td>ATMEGA168</td>
<td>16MHz</td>
<td>14KB</td>
<td>1KB</td>
</tr>
<tr>
<td>Arduino Mini ATmega328</td>
<td>No</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>28KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Arduino NG or older ATmega168</td>
<td>No</td>
<td>ATMEGA168</td>
<td>16MHz</td>
<td>14KB</td>
<td>1KB</td>
</tr>
<tr>
<td>Arduino NG or older ATmega8</td>
<td>No</td>
<td>ATMEGA8</td>
<td>16MHz</td>
<td>7KB</td>
<td>1KB</td>
</tr>
<tr>
<td>Arduino Nano ATmega168</td>
<td>No</td>
<td>ATMEGA168</td>
<td>16MHz</td>
<td>14KB</td>
<td>1KB</td>
</tr>
<tr>
<td>Arduino Nano ATmega328</td>
<td>No</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>30KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Arduino Nano ATmega328 (New Bootloader)</td>
<td>No</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>No</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>No</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>No</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>No</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>30KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Arduino Robot Control</td>
<td>No</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Arduino Robot Motor</td>
<td>No</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Arduino Uno</td>
<td>No</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Arduino Yun</td>
<td>No</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Arduino Yun Mini</td>
<td>No</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 ATtiny13</td>
<td>No</td>
<td>ATTINY13</td>
<td>1MHz</td>
<td>1KB</td>
<td>64B</td>
</tr>
<tr>
<td>Generic ATtiny13A</td>
<td>No</td>
<td>ATTINY13A</td>
<td>1MHz</td>
<td>1KB</td>
<td>64B</td>
</tr>
<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>No</td>
<td>ATTINY2313</td>
<td>8MHz</td>
<td>2KB</td>
<td>128B</td>
</tr>
<tr>
<td>Generic ATtiny24</td>
<td>No</td>
<td>ATTINY24</td>
<td>8MHz</td>
<td>2KB</td>
<td>128B</td>
</tr>
<tr>
<td>Generic ATtiny25</td>
<td>No</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>No</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>No</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>No</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>No</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>No</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>No</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>No</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>LoRa32u4III (868-915MHz)</td>
<td>No</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>No</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>No</td>
<td>ATMEGA2560</td>
<td>16MHz</td>
<td>248KB</td>
<td>8KB</td>
</tr>
<tr>
<td>Controllino Maxi Automation</td>
<td>No</td>
<td>ATMEGA2560</td>
<td>16MHz</td>
<td>248KB</td>
<td>8KB</td>
</tr>
<tr>
<td>Controllino Mega</td>
<td>No</td>
<td>ATMEGA2560</td>
<td>16MHz</td>
<td>248KB</td>
<td>8KB</td>
</tr>
<tr>
<td>Controllino Mini</td>
<td>No</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>No</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>No</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>No</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>No</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>No</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>No</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>No</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31KB</td>
<td>2KB</td>
</tr>
<tr>
<td>LowPowerLab Moteino</td>
<td>No</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>LowPowerLab Moteino (8MHz)</td>
<td>No</td>
<td>ATMEGA328P</td>
<td>8MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>LowPowerLab MoteinoMEGA</td>
<td>No</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>No</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>127KB</td>
<td>4KB</td>
</tr>
<tr>
<td>AT90CAN32</td>
<td>No</td>
<td>AT90CAN32</td>
<td>16MHz</td>
<td>31KB</td>
<td>2KB</td>
</tr>
<tr>
<td>AT90CAN64</td>
<td>No</td>
<td>AT90CAN64</td>
<td>16MHz</td>
<td>63KB</td>
<td>4KB</td>
</tr>
<tr>
<td>ATmega128/A</td>
<td>No</td>
<td>ATMEGA128</td>
<td>16MHz</td>
<td>127KB</td>
<td>4KB</td>
</tr>
<tr>
<td>ATmega1280</td>
<td>No</td>
<td>ATMEGA1280</td>
<td>16MHz</td>
<td>127KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ATmega1281</td>
<td>No</td>
<td>ATMEGA1281</td>
<td>16MHz</td>
<td>127KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ATmega1284</td>
<td>No</td>
<td>ATMEGA1284</td>
<td>16MHz</td>
<td>127KB</td>
<td>16KB</td>
</tr>
<tr>
<td>ATmega1284P</td>
<td>No</td>
<td>ATMEGA1284P</td>
<td>16MHz</td>
<td>127KB</td>
<td>16KB</td>
</tr>
<tr>
<td>ATmega16</td>
<td>No</td>
<td>ATMEGA16</td>
<td>16MHz</td>
<td>15.50KB</td>
<td>1KB</td>
</tr>
<tr>
<td>ATmega164A</td>
<td>No</td>
<td>ATMEGA164A</td>
<td>16MHz</td>
<td>15.50KB</td>
<td>1KB</td>
</tr>
<tr>
<td>ATmega164P/PA</td>
<td>No</td>
<td>ATMEGA164P</td>
<td>16MHz</td>
<td>15.50KB</td>
<td>1KB</td>
</tr>
<tr>
<td>ATmega168/A</td>
<td>No</td>
<td>ATMEGA168</td>
<td>16MHz</td>
<td>15.50KB</td>
<td>1KB</td>
</tr>
<tr>
<td>ATmega168P/PA</td>
<td>No</td>
<td>ATMEGA168P</td>
<td>16MHz</td>
<td>15.50KB</td>
<td>1KB</td>
</tr>
<tr>
<td>ATmega168PB</td>
<td>No</td>
<td>ATMEGA168PB</td>
<td>16MHz</td>
<td>15.50KB</td>
<td>1KB</td>
</tr>
<tr>
<td>ATmega2560</td>
<td>No</td>
<td>ATMEGA2560</td>
<td>16MHz</td>
<td>255KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ATmega2561</td>
<td>No</td>
<td>ATMEGA2561</td>
<td>16MHz</td>
<td>255KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ATmega32</td>
<td>No</td>
<td>ATMEGA32</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
</tbody>
</table>

Continued on next page
Table 2 – 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>ATmega324A</td>
<td>No</td>
<td>ATMEGA324A</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>ATmega324P</td>
<td>No</td>
<td>ATMEGA324P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>ATmega324PA</td>
<td>No</td>
<td>ATMEGA324PA</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>ATmega324PB</td>
<td>No</td>
<td>ATMEGA324PB</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>ATmega328</td>
<td>No</td>
<td>ATMEGA328</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>ATmega328P/PA</td>
<td>No</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>ATmega328PB</td>
<td>No</td>
<td>ATMEGA328PB</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>ATmega48A</td>
<td>No</td>
<td>ATMEGA48</td>
<td>16MHz</td>
<td>4KB</td>
<td>512B</td>
</tr>
<tr>
<td>ATmega48PB</td>
<td>No</td>
<td>ATMEGA48PB</td>
<td>16MHz</td>
<td>4KB</td>
<td>512B</td>
</tr>
<tr>
<td>ATmega64A</td>
<td>No</td>
<td>ATMEGA64</td>
<td>16MHz</td>
<td>63KB</td>
<td>4KB</td>
</tr>
<tr>
<td>ATmega640</td>
<td>No</td>
<td>ATMEGA640</td>
<td>16MHz</td>
<td>63KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ATmega644A</td>
<td>No</td>
<td>ATMEGA644A</td>
<td>16MHz</td>
<td>63KB</td>
<td>4KB</td>
</tr>
<tr>
<td>ATmega644P/PA</td>
<td>No</td>
<td>ATMEGA644P</td>
<td>16MHz</td>
<td>63KB</td>
<td>4KB</td>
</tr>
<tr>
<td>Atmel AT90PWM216</td>
<td>No</td>
<td>AT90PWM216</td>
<td>16MHz</td>
<td>16KB</td>
<td>1KB</td>
</tr>
<tr>
<td>Atmel AT90PWM316</td>
<td>No</td>
<td>AT90PWM316</td>
<td>16MHz</td>
<td>16KB</td>
<td>1KB</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 (Atmega168PA@16M,5V)</td>
<td>No</td>
<td>ATMEGA168P</td>
<td>16MHz</td>
<td>15.50KB</td>
<td>1KB</td>
</tr>
<tr>
<td>Microduino Core (Atmega168PA@8M,3.3V)</td>
<td>No</td>
<td>ATMEGA168P</td>
<td>8MHz</td>
<td>15.50KB</td>
<td>1KB</td>
</tr>
<tr>
<td>Microduino Core (Atmega328P@16M,5V)</td>
<td>No</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Microduino Core (Atmega328P@8M,3.3V)</td>
<td>No</td>
<td>ATMEGA328P</td>
<td>8MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Microduino Core+ (ATmega1284P@16M,5V)</td>
<td>No</td>
<td>ATMEGA1284P</td>
<td>16MHz</td>
<td>127KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Microduino Core+ (ATmega1284P@8M,3.3V)</td>
<td>No</td>
<td>ATMEGA1284P</td>
<td>8MHz</td>
<td>127KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Microduino Core+ (Atmega644PA@16M,5V)</td>
<td>No</td>
<td>ATMEGA644P</td>
<td>16MHz</td>
<td>63KB</td>
<td>4KB</td>
</tr>
<tr>
<td>Microduino Core+ (Atmega644PA@8M,3.3V)</td>
<td>No</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>No</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>No</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>No</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2.50KB</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>No</td>
<td>ATMEGA328P</td>
<td>8MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>LightBlue Bean+</td>
<td>No</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>No</td>
<td>ATMEGA32U4</td>
<td>8MHz</td>
<td>28KB</td>
<td>2.50KB</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 Blend</td>
<td>No</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>No</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>No</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>No</td>
<td>ATMEGA2560</td>
<td>16MHz</td>
<td>252KB</td>
<td>8KB</td>
</tr>
</tbody>
</table>

1.10. Development Platforms
### 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>No</td>
<td>ATMEGA1284P</td>
<td>8MHz</td>
<td>127KB</td>
<td>16KB</td>
</tr>
<tr>
<td>SODAQ Mbili</td>
<td>No</td>
<td>ATMEGA1284P</td>
<td>8MHz</td>
<td>127KB</td>
<td>16KB</td>
</tr>
<tr>
<td>SODAQ Maja</td>
<td>No</td>
<td>ATMEGA328P</td>
<td>8MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>SODAQ Ndogo</td>
<td>No</td>
<td>ATMEGA1284P</td>
<td>8MHz</td>
<td>127KB</td>
<td>16KB</td>
</tr>
<tr>
<td>SODAQ Tatu</td>
<td>No</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>No</td>
<td>ATMEGA1284P</td>
<td>16MHz</td>
<td>127KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Sanguino ATmega1284p (8MHz)</td>
<td>No</td>
<td>ATMEGA1284P</td>
<td>8MHz</td>
<td>127KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Sanguino ATmega644 or ATmega644A (16 MHz)</td>
<td>No</td>
<td>ATMEGA644</td>
<td>16MHz</td>
<td>63KB</td>
<td>4KB</td>
</tr>
<tr>
<td>Sanguino ATmega644 or ATmega644A (8 MHz)</td>
<td>No</td>
<td>ATMEGA644</td>
<td>8MHz</td>
<td>63KB</td>
<td>4KB</td>
</tr>
<tr>
<td>Sanguino ATmega644P or ATmega644PA (16 MHz)</td>
<td>No</td>
<td>ATMEGA644P</td>
<td>16MHz</td>
<td>63KB</td>
<td>4KB</td>
</tr>
<tr>
<td>Sanguino ATmega644P or ATmega644PA (8 MHz)</td>
<td>No</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>No</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>No</td>
<td>ATMEGA128RFA1</td>
<td>16MHz</td>
<td>16KB</td>
<td>124KB</td>
</tr>
<tr>
<td>SparkFun Digital Sandbox</td>
<td>No</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>No</td>
<td>ATMEGA32U4</td>
<td>8MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>SparkFun Makey Makey</td>
<td>No</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>No</td>
<td>ATMEGA2560</td>
<td>8MHz</td>
<td>252KB</td>
<td>8KB</td>
</tr>
<tr>
<td>SparkFun Mega Pro 5V/16MHz</td>
<td>No</td>
<td>ATMEGA2560</td>
<td>16MHz</td>
<td>248KB</td>
<td>8KB</td>
</tr>
<tr>
<td>SparkFun Mega Pro Mini 3.3V</td>
<td>No</td>
<td>ATMEGA2560</td>
<td>8MHz</td>
<td>252KB</td>
<td>8KB</td>
</tr>
<tr>
<td>SparkFun MicroView</td>
<td>No</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>No</td>
<td>ATMEGA32U4</td>
<td>8MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>SparkFun Pro Micro 5V/16MHz</td>
<td>No</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>SparkFun Qduino Mini</td>
<td>No</td>
<td>ATMEGA32U4</td>
<td>8MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>SparkFun RedBoard</td>
<td>No</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>SparkFun Serial 7-Segment Display</td>
<td>No</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>No</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>No</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>ftDuino</td>
<td>No</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>No</td>
<td>ATMEGA328P</td>
<td>8MHz</td>
<td>30KB</td>
<td>2KB</td>
</tr>
<tr>
<td>TinyCircuits TinyLily Mini Processor</td>
<td>No</td>
<td>ATMEGA328P</td>
<td>8MHz</td>
<td>30KB</td>
<td>2KB</td>
</tr>
</tbody>
</table>

### 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>No</td>
<td>ATMEGA1284P</td>
<td>16MHz</td>
<td>120.00KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Wicked Device WildFire V3</td>
<td>No</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>No</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>No</td>
<td>ATMEGA88</td>
<td>8MHz</td>
<td>8KB</td>
<td>1KB</td>
</tr>
<tr>
<td>nicai-systems NIBO 2 robot</td>
<td>No</td>
<td>ATMEGA128</td>
<td>16MHz</td>
<td>128KB</td>
<td>4KB</td>
</tr>
<tr>
<td>nicai-systems NIBO burger robot</td>
<td>No</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>No</td>
<td>ATMEGA1284P</td>
<td>20MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>nicai-systems NIBObee robot</td>
<td>No</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>No</td>
<td>ATMEGA1284P</td>
<td>20MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
</tbody>
</table>

ublQio

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ublQio Ardhat</td>
<td>No</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
</tbody>
</table>

Atmel SAM

Configuration  \( platform = \text{atmelsam} \)

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:

- arduino-blink
- arduino-external-libs
- arduino-internal-libs
- arduino-web-thing-led
- mbed-blink
- mbed-dsp
• mbed-events
• mbed-serial
• simba-blink

Debugging

PIO Unified Debugger - “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 SAMD21-XPRO</td>
<td>SAMD21J18A</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 PIO Unified Debugger 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 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>
</tbody>
</table>

Continued on next page
Table 3 – 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>Adafruit Feather M0 Express</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</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 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 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 PyGamer M4 AirLift Lite</td>
<td>SAMD51J20A</td>
<td>120MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>Adafruit PyGamer Advance M4</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 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 pIRkey</td>
<td>SAMD21E18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Adafruit pyBadge M4 Express</td>
<td>SAMD51J19A</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 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>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>Macchina M2</td>
<td>AT91SAM3X8E</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
<tr>
<td>Minitrionics 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 LoRaWAN</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>Tuino 096</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>
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-arduinosam</td>
<td>Arduino Wiring-based Framework (SAM Core)</td>
</tr>
<tr>
<td>framework-mbed</td>
<td>mbed Framework</td>
</tr>
<tr>
<td>framework-simba</td>
<td>Simba Framework</td>
</tr>
<tr>
<td>tool-avrdude</td>
<td>AVRDUDE</td>
</tr>
<tr>
<td>tool-bossac</td>
<td>BOSSA CLI</td>
</tr>
<tr>
<td>tool-jlink</td>
<td>SEGGER J-Link Software and Documentation Pack</td>
</tr>
<tr>
<td>tool-openocd</td>
<td>OpenOCD</td>
</tr>
<tr>
<td>toolchain-gccarmnoneabi</td>
<td>gcc-arm-embedded</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>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.</td>
</tr>
<tr>
<td>Simba</td>
<td>Simba is an RTOS and build framework. It aims to make embedded programming easy and portable.</td>
</tr>
</tbody>
</table>

Boards

Note:
- You can list pre-configured boards by `platformio boards` command or PlatformIO Boards Explorer.
- For more detailed board information please scroll tables below by horizontal.

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 Circuit Playground Express</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Adafruit Cricket M0</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 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 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 Metro M0 Expresss</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</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 Metro M4 AirLift Lite</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>SAMD51J20A</td>
<td>120MHz</td>
<td>1MB</td>
<td>256KB</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 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 pHKey</td>
<td>External</td>
<td>SAMD21E18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</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 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 MKRZERO</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 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>
## PlatformIO Documentation, Release 4.0.1b3

### Macchina

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Macchina M2</td>
<td>External</td>
<td>AT91SAM3X8E</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</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>Mintronics 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 LoRaWAN</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 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>
</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
  • Examples
  • Debugging
  • Stable and upstream versions
  • Packages
  • Frameworks
  • Boards

Tutorials

• Get started with Arduino and ESP32-DevKitC: debugging and unit testing
• 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
• Enable C++ exceptions
• Partition Tables
• Embedding Binary Data
• ULP coprocessor programming
• 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)

```ini
[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:

- `40000000L` (default)
- `80000000L`

```ini
[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`

```ini
[env:myenv]
board_build.flash_mode = qio
```

## 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*:
Framework **ESP-IDF**:  

```ini
[env:myenv]
platform = espressif32
framework = espidf
board = ...
build_flags =
  -DCONFIG_SPIRAM_CACHE_WORKAROUND
```

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 = ...
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)
Erase Flash

Please `platformio run --target` the next command to erase the entire flash chip (all data replaced with 0xFF bytes):

```
> platformio run --target erase
# or short version
> pio run -t erase
```

Enable C++ exceptions

Please add `-D PIO_FRAMEWORK_ESP_IDF_ENABLE_EXCEPTIONS` to `build_flags` of “platformio.ini” (Project Configuration File) to enable C++ exceptions for ESP-IDF.

See project example.

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 ESP-IDF (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:

```
[env:custom_table]
board_build.partitions = partitions_custom.csv

[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.

You can set a macro (define) COMPONENT_EMBED_TXTFILES using build_flags in “platformio.ini” (Project Con-
figuration File), giving the names of the files to embed in this way:

```
[env:myenv]
platform = espressif32
board = ...
bUILD_FLAGS =
    -DCOMPONENT_EMBED_TXTFILES=src/private.pem.key:src/certificate.pem.crt:src/aws-root-ca.pem
```

Multiple files are allowed and should be split by colon - :

The file’s 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, as given in COMPONENT_EMBED_TXTFILES. 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.

See full example with embedding Amazon AWS certificates:

- https://github.com/platformio/platform-espressif32/tree/develop/examples/espidf-aws-iot

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
framework ESP-IDF.

First of all, to use ULP in your project you need to make sure that it is enabled in your sdkconfig.h configuration
file. The next 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.

Secondly, 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
    │    └── README
    └── lib
```

(continues on next page)
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.

### Uploading files to file system SPIFFS

1. Create new project using PlatformIO IDE or initialize project using PlatformIO Core (CLI) and platformio 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 platformio run --target 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 platformio run --upload-port in command line
platformio 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,

- platformio run -t upload --upload-port 192.168.0.255
- platformio 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/tool-espotapy/espota.py -h
```

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.
- `p ESP_PORT, --port=ESP_PORT`
  ESP32 ota Port. Default **8266**

Authentication:

- `a AUTH, --auth=AUTH`
  Set authentication password.

Image:

- `f FILE, --file=FILE`

(continues on next page)
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

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 PIO Home > Platforms > Updates.

1. Please install PlatformIO IDE
2. Initialize a new project, open “platformio.ini” (Project Configuration File) and set platform to https://github.com/platformio/platform-espressif32.git#feature/stage. For example,

   [env:esp32dev]
   platform = https://github.com/platformio/platform-espressif32.git#feature/stage
   board = esp32dev
   framework = arduino

3. Try to build 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 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:

- arduino-blink
- arduino-wifiscan
- espidf-arduino-blink
- espidf-arduino-wifiscan
- espidf-aws-iot
- espidf-ble-adv
- espidf-blink
- espidf-coap-server
- espidf-exceptions
- espidf-hello-world
- espidf-http-request
- espidf-peripherals-uart
- espidf-storage-sdcard
- espidf-ulp-adc
- espidf-ulp-pulse
- pumbaa-blink
- simba-blink

**Debugging**

*PIO Unified Debugger* - “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 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 PIO Unified Debugger 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>ALKS ESP32</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>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>
</tbody>
</table>

Continued on next page
Table 4 – 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>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>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>ESPin32</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>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-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>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 T-Beam</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>1.25MB</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 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]
```

(continues on next page)
platform = espressif32@x.y.z
board = ...

Upstream

[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-arduinoespressif32</td>
<td>Arduino Wiring-based Framework (ESP32 Core)</td>
</tr>
<tr>
<td>framework-espidf</td>
<td>Espressif IoT Development Framework</td>
</tr>
<tr>
<td>framework-pumbaa</td>
<td>Pumbaa Framework</td>
</tr>
<tr>
<td>framework-simba</td>
<td>Simba Framework</td>
</tr>
<tr>
<td>tool-esptoolpy</td>
<td>ESP8266 and ESP32 serial bootloader utility</td>
</tr>
<tr>
<td>tool-mkspiffs</td>
<td>Tool to build and unpack SPIFFS images</td>
</tr>
<tr>
<td>tool-openocd-esp32</td>
<td>OpenOCD for Espressif 32</td>
</tr>
<tr>
<td>toolchain-esp32ulp</td>
<td>Binutils fork with support for the ESP32 ULP co-processor</td>
</tr>
<tr>
<td>toolchain-xtensa32</td>
<td>xtensa32-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>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.</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. It aims to make embedded programming easy and portable.</td>
</tr>
</tbody>
</table>
Note:

- You can list pre-configured boards by `platformio boards` command or PlatformIO Boards Explorer
- For more detailed board information please scroll tables below by horizontal.

### 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>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</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>
<tr>
<td>Name</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>
</tr>
<tr>
<td><strong>DOIT</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DOIT ESP32 DEVKIT V1</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td><strong>DSTIKE</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D-duino-32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td><strong>Dongsen Technology</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dongsen Tech Pocket 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td><strong>DycodeX</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ESPectro32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td><strong>ESP32vn</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ESP32vn IoT Uno</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td><strong>Electronic SweetPeas</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Electronic SweetPeas ESP320</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td><strong>Espressif</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ESP32 Pico Kit</td>
<td>No</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 WiFi LoRa 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>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>

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>

### 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-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>
</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><em>Onehorse ESP32 Dev Module</em></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><em>Pycom LoPy</em></td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td><em>Pycom LoPy4</em></td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>1.25MB</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><em>ALKS ESP32</em></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><em>Silicognition wESP32</em></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><em>SparkFun LoRa Gateway 1-Channel</em></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><em>SparkFun ESP32 Thing</em></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><em>TTGO LoRa32-OLED VI</em></td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td><em>TTGO T-Beam</em></td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>1.25MB</td>
</tr>
<tr>
<td>Name</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>
</tr>
<tr>
<td><strong>ThaiEasyElec</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ESPino32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td><strong>Turta</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Turta IoT Node</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td><strong>Unknown</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ESP32 FM DevKit</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td><strong>WEMOS</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<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 WiFi and Bluetooth Battery</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td><strong>Widora</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Widora AIR</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>16MB</td>
<td>320KB</td>
</tr>
<tr>
<td><strong>XinaBox</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>XinaBox CW02</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td><strong>oddWires</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<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**  \( \text{platform} = \text{espressif8266} \)

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
- SSL Support
- Serial Debug
- Debug Level
- VTables
- Exceptions
- Uploading files to file system SPIFFS
- Over-the-Air (OTA) update

1.10. Development Platforms
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 `build_flags` from "platformio.ini" (Project Configuration File).

```
[env:myenv]
build_flags = -Wl,-Teagle.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 passing a custom macro using project `build_flags`.
For example, switch to lwIP v1.4

```
[env:myenv]
... 
build_flags = -D PIO_FRAMEWORK_ARDUINO_LWIP_HIGHER_BANDWIDTH
```

**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):

```
[env:myenv]
... 
build_flags = -D BEARSSL_SSL_BASIC
```

**Serial Debug**

Please use the next `build_flags` to enable Serial debug:

```
[env:myenv]
... 
build_flags = -DDEBUG_ESP_PORT=Serial
; or for Serial1
build_flags = -DDEBUG_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 = -DDEBUG_ESP_PORT=Serial -DDEBUG_ESP_SSL` ...

Actual information is available in Arduino for ESP8266 Board Manifest. Please scroll to `generic.menu.lvl` section.

```
[env:myenv]
platform = ...
board = ...
framework = arduino

;;;;; Possible options ;;;;;

; SSL
build_flags = -DDEBUG_ESP_SSL

; TLS_MEM
build_flags = -DDEBUG_ESP_TLS_MEM
```

(continues on next page)
HTTP_CLIENT
build_flags = -DDEBUG_ESP_HTTP_CLIENT

HTTP_SERVER
build_flags = -DDEBUG_ESP_HTTP_SERVER

SSL+TLS_MEM
build_flags =
   -DDEBUG_ESP_SSL
   -DDEBUG_ESP_TLS_MEM

SSL+HTTP_CLIENT
build_flags =
   -DDEBUG_ESP_SSL
   -DDEBUG_ESP_HTTP_CLIENT

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 =

(continues on next page)
<table>
<thead>
<tr>
<th>Build Flags</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSL+TLS_MEM+HTTP_CLIENT+HTTP_SERVER</td>
</tr>
<tr>
<td>build_flags =</td>
</tr>
<tr>
<td>-DDEBUG_ESP_SSL</td>
</tr>
<tr>
<td>-DDEBUG_ESP_TLS_MEM</td>
</tr>
<tr>
<td>-DDEBUG_ESP_HTTP_CLIENT</td>
</tr>
<tr>
<td>-DDEBUG_ESP_HTTP_SERVER</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Build Flags</th>
</tr>
</thead>
<tbody>
<tr>
<td>CORE</td>
</tr>
<tr>
<td>build_flags =</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Build Flags</th>
</tr>
</thead>
<tbody>
<tr>
<td>WIFI</td>
</tr>
<tr>
<td>build_flags =</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Build Flags</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP_UPDATE</td>
</tr>
<tr>
<td>build_flags =</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Build Flags</th>
</tr>
</thead>
<tbody>
<tr>
<td>UPDATER</td>
</tr>
<tr>
<td>build_flags =</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Build Flags</th>
</tr>
</thead>
<tbody>
<tr>
<td>OTA</td>
</tr>
<tr>
<td>build_flags =</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Build Flags</th>
</tr>
</thead>
<tbody>
<tr>
<td>OOM</td>
</tr>
<tr>
<td>build_flags =</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Build Flags</th>
</tr>
</thead>
<tbody>
<tr>
<td>CORE+HTTP_UPDATE+UPDATER+OTA+OOM</td>
</tr>
<tr>
<td>build_flags =</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Build Flags</th>
</tr>
</thead>
<tbody>
<tr>
<td>SSL+TLS_MEM+HTTP_CLIENT+HTTP_SERVER+CORE+WIFI+HTTP_UPDATE+UPDATER+OTA+OOM</td>
</tr>
<tr>
<td>build_flags =</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Build Flags</th>
</tr>
</thead>
<tbody>
<tr>
<td>NoAssert-NDEBUG</td>
</tr>
<tr>
<td>build_flags =</td>
</tr>
</tbody>
</table>
VTables

Please use one of the next `build_flags`:

```bash
[env:myenv]
...
; Flash (default)
built_flags = -DVTABLES_IN_FLASH
; Heap
built_flags = -DVTABLES_IN_DRAM
; IRAM
built_flags = -DVTABLES_IN_IRAM
```

Exceptions

Exceptions are disabled by default. To enable exceptions, use the following `build_flags` and `build_unflags`:

```bash
[env:myenv]
...
; Remove default exceptions disabled flag
build_unflags = -fno-exceptions
; Enable exceptions
built_flags = -fexceptions
```

Uploading files to file system SPIFFS

**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 `platformio 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 `platformio 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`). 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 SPIFFS offsets (start, end, page, block). You can override it using `Flash Size`.

Active discussion is located in issue #382.

**Over-the-Air (OTA) update**

Firstly, please read What is OTA? How to use it?
There are 2 options:

- Directly specify `platformio run --upload-port` in command line

```
platformio 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,

- `platformio run -t upload --upload-port 192.168.0.255`
- `platformio 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
upload_port = IP_ADDRESS_HERE or mDNS_NAME.local
upload_flags = --port=8266
```

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

```
~/.platformio/packages/tool-espotapy/espota.py --help
```

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:

(continues on next page)
-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 ESP8266 to accept invitation

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 PIO Home > Platforms > Updates.

1. Please install PlatformIO IDE

2. Initialize a new project, open “platformio.ini” (Project Configuration File) and set platform to https://github.com/platformio/platform-espressif8266.git#feature/stage. For example,
3. Try to build 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 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-asncudp
- arduino-blink
- arduino-webserver
- arduino-wifiscan
- esp8266-nonos-sdk-blink
- esp8266-rtos-sdk-blink
- simba-blink

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

```python
[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-arduinoespressif8266</td>
<td>Arduino Wiring-based Framework (ESP8266 Core)</td>
</tr>
<tr>
<td>framework-esp8266-nanos-sdk</td>
<td>ESP8266 Non-OS SDK</td>
</tr>
<tr>
<td>framework-esp8266-RTOS-sdk</td>
<td>ESP8266 SDK based on FreeRTOS</td>
</tr>
<tr>
<td>framework-simba</td>
<td>Simba Framework</td>
</tr>
<tr>
<td>tool-esptool</td>
<td>esptool-ck</td>
</tr>
<tr>
<td>tool-esptoolpy</td>
<td>ESP8266 and ESP32 serial bootloader utility</td>
</tr>
<tr>
<td>tool-mksptfs</td>
<td>Tool to build and unpack SPIFFS images</td>
</tr>
<tr>
<td>toolchain-xtensa</td>
<td>xtensa-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>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached 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. It aims to make embedded programming easy and portable.</td>
</tr>
</tbody>
</table>

Boards

Note:
- You can list pre-configured boards by `platformio boards` command or PlatformIO Boards Explorer
• For more detailed board information please scroll tables below by horizontal.

### 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 I/O 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>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</td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
<tr>
<td>Generic ESP8266 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>WifInfo</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>

### 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>
### 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 \texttt{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:

- mbed-blink
- mbed-dsp
- mbed-events
- mbed-rtos
- mbed-rtos-ethernet
- mbed-rtos-tls-client
- mbed-serial

Debugging

PIO Unified Debugger - “1-click” solution for debugging with a zero configuration.

Tools & Debug Probes

- 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 Actions & Debug Probes using debug_tool option in “platformio.ini” conﬁguration.
**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>Ethernet IoT Starter Kit</td>
<td>MK64FN1M0VLL12</td>
<td>120MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>Freescale Kinetics FRDM-K20D50M</td>
<td>MK20DX128VLH5</td>
<td>48MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Freescale Kinetics FRDM-K22F</td>
<td>MK22FN512VLH12</td>
<td>120MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>Freescale Kinetics FRDM-K64F</td>
<td>MK64FN1M0VLL12</td>
<td>120MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>Freescale Kinetics FRDM-K66F</td>
<td>MK66FN2M0VMD18</td>
<td>180MHz</td>
<td>2MB</td>
<td>256KB</td>
</tr>
<tr>
<td>Freescale Kinetics FRDM-K82F</td>
<td>MK82FN256VLL15</td>
<td>150MHz</td>
<td>256KB</td>
<td>256KB</td>
</tr>
<tr>
<td>Freescale Kinetics FRDM-KL05Z</td>
<td>MKL05Z32VFM4</td>
<td>48MHz</td>
<td>32KB</td>
<td>4KB</td>
</tr>
<tr>
<td>Freescale Kinetics FRDM-KL25Z</td>
<td>MKL25Z128VLK4</td>
<td>48MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Freescale Kinetics FRDM-KL27Z</td>
<td>MKL27Z64VHL4</td>
<td>48MHz</td>
<td>64KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Freescale Kinetics FRDM-KLA31Z</td>
<td>MKL43Z256VHL4</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Freescale Kinetics FRDM-KLA62Z</td>
<td>MKL46Z256VLL4</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Freescale Kinetics 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 **PIO Unified Debugger** 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 Kinetics FRDM-KL82Z</td>
<td>MKL82Z128VLK7</td>
<td>96MHz</td>
<td>128KB</td>
<td>96KB</td>
</tr>
<tr>
<td>Freescale Kinetics 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>
</tbody>
</table>

### Stable and upstream versions

You can switch between **stable releases** of Freescale Kinetics 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 = freescalekinetis
board = ...
```

(continues on next page)
; Custom stable version
[env: custom_stable]
platform = freescalekinetis@x.y.z
board = ...

Upstream

[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>mbed Framework</td>
</tr>
<tr>
<td>tool-jlink</td>
<td>SEGGER J-Link Software and Documentation Pack</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>gcc-arm-embedded</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>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.</td>
</tr>
</tbody>
</table>

Boards

Note:

- You can list pre-configured boards by `platformio boards` command or PlatformIO Boards Explorer
- For more detailed board information please scroll tables below by horizontal.

**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>MK22FN512VLH12</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>MKL05Z32VFM4</td>
<td>48MHz</td>
<td>32KB</td>
<td>4KB</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-KL25Z</td>
<td>On-board</td>
<td>MKL25Z128VLK4</td>
<td>48MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-KL27Z</td>
<td>On-board</td>
<td>MKL27Z64VLMH4</td>
<td>48MHz</td>
<td>64KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-KL32Z</td>
<td>On-board</td>
<td>MKL32Z256VLL4</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-KL43Z</td>
<td>On-board</td>
<td>MKL43Z256VLL4</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>

**Infineon XMC**

*Configuration* platform = 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 vendor site.
Examples

Examples are listed from Infineon XMC development platform repository:

- arduino-blink
- arduino-wire
- device-control
- ifx9201
- radar
- rtc
- spi
- ultrasonic

Debugging

PIO Unified Debugger - “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>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**

```ini
; Latest stable version
[env:latest_stable]
platform = infineonxmc
board = ...

; Custom stable version
[env:custom_stable]
platform = infineonxmc@x.y.z
board = ...
```

**Upstream**

```ini
[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 (Infineon XMC Core)</td>
</tr>
<tr>
<td>tool-jlink</td>
<td>SEGGER J-Link Software and Documentation Pack</td>
</tr>
<tr>
<td>toolchain-gccarmnoneabi</td>
<td>gcc-arm-embedded</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 `platformio boards` command or PlatformIO Boards Explorer
- For more detailed board information please scroll tables below by horizontal.

### 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-blink
- arduino-curie-imu
- 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

```
; Latest stable version
[env:latest_stable]
platform = intel_arc32
board = ...

; Custom stable version
[env:custom_stable]
platform = intel_arc32@x.y.z
board = ...
```

### Upstream

```
[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 ARC Core)</td>
</tr>
<tr>
<td>tool-arduino101load</td>
<td>Genuino101 uploader</td>
</tr>
<tr>
<td>toolchain-intelarc32</td>
<td>GCC for Intel ARC</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 `platformio boards` command or PlatformIO Boards Explorer.
- For more detailed board information please scroll tables below by horizontal.

## 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.

## Contents

- Examples
  - Stable and upstream versions
- Packages
- Boards

## Examples

Examples are listed from Intel MCS-51 (8051) development platform repository:

- native-blink
- stc-blink
- stc-header

## 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

```markdown
; Latest stable version
[env:latest_stable]
platform = intel_mcs51
board = ...

; Custom stable version
[env:custom_stable]
platform = intel_mcs51@x.y.z
board = ...
```

Upstream

```markdown
[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</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 `platformio boards` command or [PlatformIO Boards Explorer](#).
- For more detailed board information please scroll tables below by horizontal.
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 platform = kendryte210

Kendryte K210 is an AI capable RISCV64 dual core SoC, this platform is ported by Sipeed.

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-freertos-sdk_hello
- kendryte-standalone-sdk_hello
Debugging

PIO Unified Debugger - “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 PIO Unified Debugger 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>
</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

; Latest stable version
[env:latest_stable]
platform = kendryte210
board = ...

; Custom stable version
[env:custom_stable]
platform = kendryte210@x.y.z
board = ...
Upstream

```
[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 SDK with FreeRTOS support</td>
</tr>
<tr>
<td>framework-kendryte-standalone-sdk</td>
<td>Kendryte standalone SDK without OS support</td>
</tr>
<tr>
<td>framework-maixduino</td>
<td>Arduino Wiring-based Framework (K210 Core)</td>
</tr>
<tr>
<td>tool-kflash-kendryte210</td>
<td>kflash, A Python-based Kendryte K210 UART ISP Utility</td>
</tr>
<tr>
<td>tool-openocd-kendryte</td>
<td>OpenOCD for RISC-V Kendryte</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 Standalone SDK</td>
<td>Kendryte Standalone SDK without OS support</td>
</tr>
<tr>
<td>Kendryte FreeRTOS SDK</td>
<td>Kendryte SDK with FreeRTOS support</td>
</tr>
</tbody>
</table>

Boards

Note:
- You can list pre-configured boards by platformio boards command or PlatformIO Boards Explorer
- For more detailed board information please scroll tables below by horizontal.
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**  \( \text{platform} = \text{lattice\_ice40} \)

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:

- counter
- leds

### 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.

```ini
; Latest stable version
[env:latest_stable]
platform = lattice\_ice40
board = ...
```

(continues on next page)
}; 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 platformio boards command or PlatformIO Boards Explorer
- For more detailed board information please scroll tables below by horizontal.

FPGAwars

<table>
<thead>
<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** \texttt{platform = 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-blink
- mbed-dsp
- mbed-events
- mbed-rtos
- mbed-serial

## Debugging

*PIO Unified Debugger* - “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 PIO Unified Debugger 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>mbed Framework</td>
</tr>
<tr>
<td>tool-jlink</td>
<td>SEGGER J-Link Software and Documentation Pack</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>gcc-arm-embedded</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>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.</td>
</tr>
</tbody>
</table>

Boards

**Note:**
- You can list pre-configured boards by `platformio boards` command or PlatformIO Boards Explorer
- For more detailed board information please scroll tables below by horizontal.
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

```ini
[env:latest_stable]
platform = microchippic32
board = ...

; Custom stable version
[env:custom_stable]
platform = microchippic32@x.y.z
board = ...
```

### Upstream

```ini
[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 (PIC32 Core)</td>
</tr>
<tr>
<td>tool-pic32prog</td>
<td>pic32prog</td>
</tr>
<tr>
<td>toolchain-microchippic32</td>
<td>GCC 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 `platformio boards` command or PlatformIO Boards Explorer
- For more detailed board information please scroll tables below by horizontal.

4DSystems

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>4DSystems PICadillo 35T</td>
<td>No</td>
<td>32MX795F512L</td>
<td>80MHz</td>
<td>508KB</td>
<td>128KB</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 Cmd</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>
</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>
</tbody>
</table>

1.10. Development Platforms
### 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>

### 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>

element14

<table>
<thead>
<tr>
<th>Name</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>No</td>
<td>32MX250F128B</td>
<td>40MHz</td>
<td>120KB</td>
<td>32KB</td>
</tr>
</tbody>
</table>

Nordic nRF51

**Configuration** `platform = 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](#).

---

**Contents**

- Examples
- Debugging
- Stable and upstream versions
- Packages
- Frameworks
- Boards

**Examples**

Examples are listed from [Nordic nRF51 development platform repository](#):

- arduino-ble-led
- arduino-blink
- arduino-internal-libs
- mbed-ble-thermometer
- mbed-blink
- mbed-events
- mbed-microbit-hello-world
- mbed-serial
Debugging

PIO Unified Debugger - “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</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>32KB</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 VBLUNO51</td>
<td>NRF51822</td>
<td>16MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>y5 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 PIO Unified Debugger 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

```plaintext
; Latest stable version
[env:lastest_stable]
platform = nordicnrf51
board = ...

; Custom stable version
[env:custom_stable]
platform = nordicnrf51@x.y.z
board = ...
```

Upstream

```plaintext
[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 (Nordic NRF5 Core)</td>
</tr>
<tr>
<td>framework-mbed</td>
<td>mbed Framework</td>
</tr>
<tr>
<td>tool-jlink</td>
<td>SEGGER J-Link Software and Documentation Pack</td>
</tr>
<tr>
<td>tool-nrfjprog</td>
<td>nRF5x command line tool</td>
</tr>
<tr>
<td>tool-openocd</td>
<td>OpenOCD</td>
</tr>
<tr>
<td>tool-sreccat</td>
<td>Merging tool</td>
</tr>
<tr>
<td>toolchain-gccarmnoneabi</td>
<td>gcc-arm-embedded</td>
</tr>
</tbody>
</table>

1.10. Development Platforms 331
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>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.</td>
</tr>
</tbody>
</table>

Boards

Note:

- You can list pre-configured boards by `platformio boards` command or PlatformIO Boards Explorer
- For more detailed board information please scroll tables below by horizontal.

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><strong>BBC micro:bit</strong></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><strong>BluzDK</strong></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><strong>Calliope mini</strong></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>OSHChip</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>RedBearLab BLE Nano 1.5</td>
<td>On-board</td>
<td>NRF51822</td>
<td>16MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>RedBearLab nRF51822</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>Seeed Arch BLE</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>On-board</td>
<td>NRF51822</td>
<td>16MHz</td>
<td>256KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Seeed Tiny BLE</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>Switch Science mbed HRM1017</td>
<td>On-board</td>
<td>NRF51822</td>
<td>16MHz</td>
<td>256KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Switch Science mbed TY51822r3</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>VNG VBLUNO51</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>Waveshare BLE400</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>ng-beacon</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.
Tutorials

• Arduino and Nordic nRF52-DK: debugging and unit testing

Examples

Examples are listed from Nordic nRF52 development platform repository:

• arduino-ble-led
• arduino-blink
• arduino-bluefruit-bleuart
• arduino-nina-b1-generic-example
• mbed-ble-thermometer
• mbed-blink
• mbed-dsp
• mbed-events
• mbed-nfc
• mbed-rtos
• mbed-serial

Debugging

PIO Unified Debugger - “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>Adafruit Bluefruit nRF52832 Feather</td>
<td>nRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>Adafruit Feather nRF52840 Express</td>
<td>nRF52840</td>
<td>64MHz</td>
<td>796KB</td>
<td>243KB</td>
</tr>
<tr>
<td>Delta DFBM-NQ620</td>
<td>nRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</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>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 PIO Unified Debugger 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>Bluey nRF52832 IoT</td>
<td>nRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>SDT52832B</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

```plaintext
; Latest stable version
[env:latest_stable]
platform = nordicnrf52
```

(continues on next page)
board = ...

; Custom stable version

[env:custom_stable]
platform = nordicnrf52@x.y.z
board = ...

Upstream

[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-arduinoadafruitnrf52</td>
<td>Arduino Wiring-based Framework (Nordic nRF52 BLE SoC)</td>
</tr>
<tr>
<td>framework-arduionordicnrf5</td>
<td>Arduino Wiring-based Framework (Nordic NRF5 Core)</td>
</tr>
<tr>
<td>framework-mbed</td>
<td>mbed Framework</td>
</tr>
<tr>
<td>tool-jlink</td>
<td>SEGGER J-Link Software and Documentation Pack</td>
</tr>
<tr>
<td>tool-nrfjprog</td>
<td>nRF5x command line tool</td>
</tr>
<tr>
<td>tool-openocd</td>
<td>OpenOCD</td>
</tr>
<tr>
<td>tool-sreccat</td>
<td>Merging tool</td>
</tr>
<tr>
<td>toolchain-gccarmnoneabi</td>
<td>gcc-arm-embedded</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>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.</td>
</tr>
</tbody>
</table>
Boards

Note:

- You can list pre-configured boards by `platformio boards` command or PlatformIO Boards Explorer
- For more detailed board information please scroll tables below by horizontal.

### 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>On-board</td>
<td>nRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>Adafruit Feather nRF52840 Express</td>
<td>On-board</td>
<td>nRF52840</td>
<td>64MHz</td>
<td>796KB</td>
<td>243KB</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>

### 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 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>

### 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>
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>

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>

NXP LPC

**Configuration** \( \text{platform} = \text{nxplpc} \)

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
- Boards

**Examples**

Examples are listed from NXP LPC development platform repository:

- mbed-blinc
- mbed-custom-target
- mbed-dsp
- mbed-events
• mbed-http-client
• mbed-rtos
• mbed-rtos-ethernet
• mbed-serial

Debugging

PIO Unified Debugger - “1-click” solution for debugging with a zero configuration.

Tools & Debug Probes

• 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>ARM mbed LPC11U24 (+CAN)</td>
<td>LPC11U24</td>
<td>48MHz</td>
<td>32KB</td>
<td>8KB</td>
</tr>
<tr>
<td>Bambino-210E</td>
<td>LPC4330</td>
<td>204MHz</td>
<td>8MB</td>
<td>264KB</td>
</tr>
<tr>
<td>CoCo-ri-Co!</td>
<td>LPC812</td>
<td>30MHz</td>
<td>16KB</td>
<td>4KB</td>
</tr>
<tr>
<td>Embedded Artists LPC4088 Display Module</td>
<td>LPC4088</td>
<td>120MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
<tr>
<td>Embedded Artists LPC4088 QuickStart Board</td>
<td>LPC4088</td>
<td>120MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
<tr>
<td>LPCXpresso11U68</td>
<td>LPC11U68</td>
<td>50MHz</td>
<td>256KB</td>
<td>36KB</td>
</tr>
<tr>
<td>LPCXpresso824-MAX</td>
<td>LPC824</td>
<td>30MHz</td>
<td>32KB</td>
<td>8KB</td>
</tr>
<tr>
<td>NXP LPC800-MAX</td>
<td>LPC812</td>
<td>30MHz</td>
<td>16KB</td>
<td>4KB</td>
</tr>
<tr>
<td>NXP LPCXpresso54114</td>
<td>LPC54114J256BD64</td>
<td>100MHz</td>
<td>256KB</td>
<td>192KB</td>
</tr>
<tr>
<td>NXP LPCXpresso54608</td>
<td>LPC54608ET512</td>
<td>180MHz</td>
<td>512KB</td>
<td>200KB</td>
</tr>
<tr>
<td>NXP mbed LPC11U24</td>
<td>LPC11U24</td>
<td>48MHz</td>
<td>32KB</td>
<td>8KB</td>
</tr>
<tr>
<td>NXP mbed LPC1768</td>
<td>LPC1768</td>
<td>96MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>Seeed Arch Pro</td>
<td>LPC1768</td>
<td>96MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>Switch Science mbed LPC1114FN28</td>
<td>LPC1114FN28</td>
<td>48MHz</td>
<td>32KB</td>
<td>4KB</td>
</tr>
<tr>
<td>Switch Science mbed LPC824</td>
<td>LPC824</td>
<td>30MHz</td>
<td>32KB</td>
<td>8KB</td>
</tr>
<tr>
<td>u-blox C027</td>
<td>LPC1768</td>
<td>96MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
</tbody>
</table>
External Debug Tools

Boards listed below are compatible with PIO Unified Debugger 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 = ...

; Custom stable version
[env:custom_stable]
platform = nxplpc@x.y.z
board = ...
```

**Upstream**

```ini
[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>mbed Framework</td>
</tr>
<tr>
<td>tool-jlink</td>
<td>SEGGER J-Link Software and Documentation Pack</td>
</tr>
<tr>
<td>tool-openocd</td>
<td>OpenOCD</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>gcc-arm-embedded</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>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.</td>
</tr>
</tbody>
</table>

Boards

Note:
- You can list pre-configured boards by platformio boards command or PlatformIO Boards Explorer
- For more detailed board information please scroll tables below by horizontal.

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>50MHz</td>
<td>32KB</td>
<td>10KB</td>
</tr>
</tbody>
</table>

## Micromint

(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>LPC5411J256BD64</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 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>

### 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 `platformio 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

```ini
[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 `platformio 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 = reqloop ioloop start wait
```

- Run “Upload” task in PlatformIO IDE or use PlatformIO Core (CLI) and `platformio 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 = start
```

- Run “Upload” task in *PlatformIO IDE* or use *PlatformIO Core (CLI)* and `platformio run --target upload` command with `upload` target.

**Uploading files to HyperFlash**

1. Create new project using *PlatformIO IDE* or initialize project using *PlatformIO Core (CLI)* and `platformio 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 `platformio run --target uploadfs` command with `uploadfs` target.

Examples:

- PULP OS File System

**Examples**

Examples are listed from RISC-V GAP development platform repository:

- gapuino-mbed-autotiler-cifar10
- gapuino-mbed-driver-raw-serial
- gapuino-mbed-driver-hyper-flash
- gapuino-mbed-driver-hyper-rtc-alarm
- gapuino-mbed-events-queue
- gapuino-mbed-features-cluster-dma
- gapuino-mbed-features-filesystem
- gapuino-mbed-fft2d
- gapuino-mbed-matadd
- gapuino-mbed-os-irq
- gapuino-mbed-os-memory-pool
- gapuino-pulp-os-autotiler-bilinear-resize
• gapuino-pulp-os-autotiler-cifar10
• gapuino-pulp-os-filesystem
• gapuino-pulp-os-hello-world
• gapuino-pulp-os-i2c-eeprom
• gapuino-pulp-os-kernel-dma

Debugging

PIO Unified Debugger - “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>GAPuino 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/pioplus/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>The GAP8 SDK allows you to compile and execute applications on the 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>RISC-V GCC toolchain for 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>mbed</td>
<td>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.</td>
</tr>
<tr>
<td>PULP</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 `platformio boards` command or PlatformIO Boards Explorer.
• For more detailed board information please scroll tables below by horizontal.

### 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>

### Samsung ARTIK

**Configuration** `platform = samsung_artik`

The Samsung ARTIK Smart IoT platform brings hardware modules and cloud services together, with built-in security and an ecosystem of tools and partners to speed up your time-to-market. For more detailed information please visit vendor site.

### Contents

- **Configuration**
- **Examples**
- **Debugging**
- **Stable and upstream versions**
- **Packages**
- **Frameworks**
- **Boards**

### Configuration

- **Windows**
- **macOS**
- **Linux**

If you are using the Samsung ARTIK platform on macOS or Linux, you need to perform the configuration steps detailed below to enable support for deployment and debugging.

**Windows**

Usually Windows Update Service will automatically install FTDI driver. You can choose an off-line installation and install FTDI driver manually.
After installation, you will have two FTDI devices. Please use zadig tool to change one FTDI device to a “libusb” compatible device.

**macOS**

First check that you have a FTDI compatible driver on your mac:

```
kextstat | grep FTDI
```

You should have `com.apple.driver.AppleUSBFTDI`.

1. Install ARTIK FTDI Driver
2. Reboot your system.

**Linux**

Create a new file named `/etc/udev/rules.d/51-artik053.rules` and add the following line:

```
SUBSYSTEM=="usb", ATTR{idVendor}=="0403", ATTR{idProduct}=="6010", MODE="0660", GROUP="plugdev", SYMLINK+="artik053-%n"
```

**Examples**

Examples are listed from Samsung ARTIK development platform repository:

- `artik_sdk`
- `blink_led_wifi`
- `hello`

**Debugging**

*PIO Unified Debugger* - “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>Samsung ARTIK053</td>
<td>S5JT200</td>
<td>320MHz</td>
<td>8MB</td>
<td>1.25MB</td>
</tr>
</tbody>
</table>

Stable and upstream versions

You can switch between stable releases of Samsung ARTIK 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 = samsung_artik
board = ...

; Custom stable version
[env:custom_stable]
platform = samsung_artik@x.y.z
board = ...
```

**Upstream**

```ini
[env:upstream_develop]
platform = https://github.com/platformio/platform-samsung_artik.git
board = ...
```

Packages

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>framework-tizenrt</td>
<td>TizenRT RTOS with library</td>
</tr>
<tr>
<td>tool-artik-openocd</td>
<td>OpenOCD for ARTIK</td>
</tr>
<tr>
<td>toolchain-gccarmnoneabi</td>
<td>gcc-arm-embedded</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

1.10. Development Platforms
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tizen RT</td>
<td>Tizen RT is a lightweight RTOS-based platform to support low-end IoT devices</td>
</tr>
</tbody>
</table>

Boards

Note:
- You can list pre-configured boards by `platformio boards` command or PlatformIO Boards Explorer
- For more detailed board information please scroll tables below by horizontal.

### Samsung

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Samsung ARTIK053</td>
<td>On-board</td>
<td>S5JT200</td>
<td>320MHz</td>
<td>8MB</td>
<td>1.25MB</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_hello`
- `freedom-e-sdk_multicore-hello`
- `freedom-e-sdk_sifive-welcome`
• freedom-e-sdk_spi
• freedom-e-sdk_test-coreip
• freedom-e-sdk_timer-interrupt
• freedom-e-sdk_user-mode
• freedom-e-sdk_user-syscall
• native-blink_asm

Debugging

PIO Unified Debugger - “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>
</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

```plaintext
; Latest stable version
[env:latest_stable]
platform = sifive
board = ...

; Custom stable version
[env:custom_stable]
platform = sifive@x.y.z
board = ...
```

Upstream

```plaintext
[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>tool-jlink</td>
<td>SEGGER J-Link Software and Documentation Pack</td>
</tr>
<tr>
<td>tool-openocd-riscv</td>
<td>OpenOCD for RISC-V</td>
</tr>
<tr>
<td>tool-qemu-riscv</td>
<td>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>Freedom E SDK</td>
<td>Open Source Software for Developing on the SiFive Freedom E Platform</td>
</tr>
</tbody>
</table>

Boards

Note:
• You can list pre-configured boards by `platformio boards` command or PlatformIO Boards Explorer.
• For more detailed board information please scroll tables below by horizontal.

<table>
<thead>
<tr>
<th>SiFive</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Name</strong></td>
</tr>
<tr>
<td>HiFive Unleashed</td>
</tr>
<tr>
<td>HiFive1</td>
</tr>
<tr>
<td>HiFive1 Rev B</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Xilinx</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Name</strong></td>
</tr>
<tr>
<td>Arty FPGA Dev Kit</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](#).

## Contents

- **Examples**
- **Debugging**
- **Stable and upstream versions**
- **Packages**
- **Frameworks**
- **Boards**

## Examples

Examples are listed from Silicon Labs EFM32 development platform repository:

- mbed-blink
- mbed-dsp
- mbed-events

1.10. Development Platforms 357
• mbed-serial

Debugging

PIO Unified Debugger - “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>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>Thunderboard Sense 2 Sensor-to-Cloud Advanced IoT</td>
<td>EFR32MG12P432F1024</td>
<td>40MHz</td>
<td>1MB</td>
<td>256KB</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
; 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>mbed Framework</td>
</tr>
<tr>
<td>tool-link</td>
<td>SEGGER J-Link Software and Documentation Pack</td>
</tr>
<tr>
<td>toolchain-gccarmnoneabi</td>
<td>gcc-arm-embedded</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>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.</td>
</tr>
</tbody>
</table>

Boards

Note:

- You can list pre-configured boards by platformio boards command or PlatformIO Boards Explorer

1.10. Development Platforms
• For more detailed board information please scroll tables below by horizontal.

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>EFM32GG-STK3700 Giant Gecko</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>On-board</td>
<td>EFM32LG990F256</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>EFM32WG-STK3800 Wonder Gecko</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>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>On-board</td>
<td>EFM32HG322F64</td>
<td>25MHz</td>
<td>64KB</td>
<td>8KB</td>
</tr>
<tr>
<td>SLSTK3401A Pearl Gecko PG1</td>
<td>On-board</td>
<td>EFR32MG12P432F1024</td>
<td>40MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>Thunderboard Sense 2 Sensor-to-Cloud Ad-</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>vanced IoT</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

ST STM32

Configuration `platform = 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.
Tutorials

- STM32Cube HAL and Nucleo-F401RE: debugging and unit testing

Configuration

Switching between Arduino cores

There are two different Arduino cores for STM32 microcontrollers: STM32Duino and Arduino STM32 (maple). Both of them have been developed independently, therefore, have different functionality and set of internal libraries. By default, official STM32Duino core is used. Some of the boards support both cores. To change the core you can use a board_build.core option that needs be added to build_flags:

An example of “platformio.ini” (Project Configuration File) with maple core

```
[env:hy_tinystm103tb]
platform = stm32
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):

Table 5: 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 6: 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 7: 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 supersedes 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 8: 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:

```python
[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>
<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:

```python
[env:disco_f407vg]
platform = ststm32
framework = arduino
board = disco_f407vg
board_build.core = maple
build_flags = -D ENABLE_USB_MASS_STORAGE
```

Examples

Examples are listed from ST STM32 development platform repository:

- arduino-blink
- arduino-external-libs
- arduino-internal-libs
- arduino-mxchip-azureiot
- arduino-mxchip-filesystem
- arduino-mxchip-sensors
- arduino-mxchip-wifiscan
- cmsis-blink
- libopencm3-1bitsy
- libopencm3-blink
- mbed-blink
- mbed-custom-target
• mbed-dsp
• mbed-events
• mbed-filesystem
• mbed-rtos
• mbed-rtos-ethernet
• mbed-rtos-ethernet-tls
• mbed-rtos-mesh-minimal
• mbed-rtos-semaphore
• mbed-serial
• spl-blink
• stm32cube-hal-blink
• stm32cube-ll-blink

Debugging

PIO Unified Debugger - “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>L476DMW1K</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>STM32F412ZG76</td>
<td>100MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>Nucleo G071RB</td>
<td>STM32G071RBT6</td>
<td>24MHz</td>
<td>2MB</td>
<td>128KB</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 32F3348DISCOVERY</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 32F429IDISCOVERY</td>
<td>STM32F429ZIT6</td>
<td>180MHz</td>
<td>2MB</td>
<td>256KB</td>
</tr>
<tr>
<td>ST 32F469IDISCOVERY</td>
<td>STM32F469NIH6</td>
<td>180MHz</td>
<td>1MB</td>
<td>384KB</td>
</tr>
<tr>
<td>ST 32F46GDISCOVERY</td>
<td>STM32F46NGH6</td>
<td>216MHz</td>
<td>1MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ST 32F769IDISCOVERY</td>
<td>STM32F769NIH6</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 DISCO-L072CZ-LRWAN1</td>
<td>STM32L072CZ</td>
<td>32MHz</td>
<td>192KB</td>
<td>20KB</td>
</tr>
<tr>
<td>ST DISCO-L475VG-IOT01A</td>
<td>STM32L475VGT6</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</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 F070RB</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>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>16KB</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>STM32F412ZG6</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 F746ZG</td>
<td>STM32F746ZG6</td>
<td>216MHz</td>
<td>1MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ST Nucleo F756ZG</td>
<td>STM32F756ZG</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>1MB</td>
</tr>
</tbody>
</table>

Continued on next page
### Table 10 – 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>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 L051R8</td>
<td>STM32L053R8T6</td>
<td>32MHz</td>
<td>64KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ST Nucleo L073RZ</td>
<td>STM32L073RZ</td>
<td>32MHz</td>
<td>192KB</td>
<td>20KB</td>
</tr>
<tr>
<td>ST Nucleo L152RE</td>
<td>STM32L152RE</td>
<td>32MHz</td>
<td>512KB</td>
<td>80KB</td>
</tr>
<tr>
<td>ST Nucleo L432KC</td>
<td>STM32L432KCUC6</td>
<td>80MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
<tr>
<td>ST Nucleo L433RC-P</td>
<td>STM32L433RC</td>
<td>80MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
<tr>
<td>ST Nucleo L452RE</td>
<td>STM32L452RE</td>
<td>80MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
<tr>
<td>ST Nucleo L476RG</td>
<td>STM32L476RG</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo L486RG</td>
<td>STM32L486RG</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo L496ZG</td>
<td>STM32L496ZG</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo L496ZGP</td>
<td>STM32L496ZGT6P</td>
<td>80MHz</td>
<td>1MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ST Nucleo LAR5ZI</td>
<td>STM32L4R5ZIT6</td>
<td>120MHz</td>
<td>2MB</td>
<td>640KB</td>
</tr>
<tr>
<td>ST STM32F0308DISCOVERY</td>
<td>STM32F030R8T6</td>
<td>48MHz</td>
<td>64KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ST STM32F0DISCOVERY</td>
<td>STM32F051R8T6</td>
<td>48MHz</td>
<td>64KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ST STM32F3DISCOVERY</td>
<td>STM32F303VCT6</td>
<td>72MHz</td>
<td>256KB</td>
<td>48KB</td>
</tr>
<tr>
<td>ST STM32F4DISCOVERY</td>
<td>STM32F407VGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>192KB</td>
</tr>
<tr>
<td>ST STM32L073Z-EVAL</td>
<td>STM32L073VZ6T6</td>
<td>32MHz</td>
<td>192KB</td>
<td>20KB</td>
</tr>
<tr>
<td>ST STM32LDISCOVERY</td>
<td>STM32L152RB6T6</td>
<td>32MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>ST STM32VLDISCOVERY</td>
<td>STM32F100RB6T6</td>
<td>24MHz</td>
<td>128KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ST Sensor Node</td>
<td>STM32L476JG</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST STM32F7508-DK</td>
<td>STM32F750N8H6</td>
<td>216MHz</td>
<td>64KB</td>
<td>340KB</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>STM32F439VI</td>
<td>180MHz</td>
<td>2MB</td>
<td>256KB</td>
</tr>
<tr>
<td>sakura.io Evaluation Board</td>
<td>STM32F411RET6</td>
<td>100MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>u-blox C030-R410M IoT</td>
<td>STM32F437VG</td>
<td>180MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
</tbody>
</table>

### External Debug Tools

Boards listed below are compatible with [PIO Unified Debugger](https://docs.platformio.org) 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>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>Black STM32F407VE</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Black STM32F407VG</td>
<td>STM32F407VGT6</td>
<td>168MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Black STM32F407ZE</td>
<td>STM32F407ZGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>192KB</td>
</tr>
<tr>
<td>Black STM32F407ZE</td>
<td>STM32F407ZET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>192KB</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>Blue STM32F407VE Mini</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>192KB</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>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>Demo F030F4</td>
<td>STM32F030F4P6</td>
<td>48MHz</td>
<td>16KB</td>
<td>4KB</td>
</tr>
<tr>
<td>Espotel LoRa Module</td>
<td>STM32F411RET6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>F4-07VG</td>
<td>STM32F407VGT6</td>
<td>168MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>FK407M1</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>M200 V2</td>
<td>STM32F070CBT6</td>
<td>48MHz</td>
<td>120KB</td>
<td>14.81KB</td>
</tr>
<tr>
<td>MTS Dragonfly</td>
<td>STM32F411RET6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>Malyan M200 V1</td>
<td>STM32F103CBT6</td>
<td>72MHz</td>
<td>120KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Maple</td>
<td>STM32F103RBT6</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>RAK811 LoRa Tracker</td>
<td>STM32L151RBT6</td>
<td>32MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>RAK811 LoRa Tracker</td>
<td>STM32L151RBT6</td>
<td>32MHz</td>
<td>128KB</td>
<td>16KB</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>STM32F103RB (20k RAM, 64 Flash)</td>
<td>STM32F103R8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>20KB</td>
</tr>
<tr>
<td>STM32F103RB (20k RAM, 128k Flash)</td>
<td>STM32F103RB6</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>STM32F103RE (64k RAM, 512k Flash)</td>
<td>STM32F103RET6</td>
<td>72MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F103T8 (20k RAM, 64k Flash)</td>
<td>STM32F103T8T6</td>
<td>72MHz</td>
<td>20KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F103TB (20k RAM, 128k Flash)</td>
<td>STM32F103TBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>STM32F103VB (20k RAM, 128k Flash)</td>
<td>STM32F103VBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>STM32F103VC (48k RAM, 256k Flash)</td>
<td>STM32F103VCT6</td>
<td>72MHz</td>
<td>256KB</td>
<td>48KB</td>
</tr>
<tr>
<td>STM32F103VD (64k RAM, 384k Flash)</td>
<td>STM32F103VDT6</td>
<td>72MHz</td>
<td>384KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F103VE (64k RAM, 512k Flash)</td>
<td>STM32F103VET6</td>
<td>72MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F103VC (48k RAM, 256k Flash)</td>
<td>STM32F103ZCT6</td>
<td>72MHz</td>
<td>256KB</td>
<td>48KB</td>
</tr>
<tr>
<td>STM32F103VD (64k RAM, 384k Flash)</td>
<td>STM32F103ZDT6</td>
<td>72MHz</td>
<td>384KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F103ZE (64k RAM, 512k Flash)</td>
<td>STM32F103ZET6</td>
<td>72MHz</td>
<td>512KB</td>
<td>64KB</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>STM32F407VE (192k RAM, 512k Flash)</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>502.23KB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F407VG (192k RAM, 1024k Flash)</td>
<td>STM32F407VGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>192KB</td>
</tr>
<tr>
<td>STM32F4Signature F405</td>
<td>STM32F405RGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>192KB</td>
</tr>
<tr>
<td>Sparky V1 F303</td>
<td>STM32F303CCT6</td>
<td>72MHz</td>
<td>256KB</td>
<td>40KB</td>
</tr>
<tr>
<td>Tiny STM103T</td>
<td>STM32F103TBU6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>VAE v1.0</td>
<td>STM32F446RET6</td>
<td>180MHz</td>
<td>512KB</td>
<td>128KB</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>
<tr>
<td>u-blox 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 (ST STM32 MXChip Core)</td>
</tr>
<tr>
<td>framework-arduinoststm32</td>
<td>Arduino Wiring-based Framework (STM32 Core)</td>
</tr>
<tr>
<td>framework-arduinoststm32-maple</td>
<td>Arduino Wiring-based Framework (ST STM32 Maple Core)</td>
</tr>
<tr>
<td>framework-cmsis</td>
<td>Vendor-independent hardware abstraction layer for the Cortex-M processor series</td>
</tr>
<tr>
<td>framework-libopencm3</td>
<td>libOpenCM3 Framework</td>
</tr>
<tr>
<td>framework-mbed</td>
<td>mbed Framework</td>
</tr>
<tr>
<td>framework-spl</td>
<td>Standard Peripheral Library for STM32 MCUs</td>
</tr>
<tr>
<td>framework-stm32cube</td>
<td>STM32Cube embedded software libraries</td>
</tr>
<tr>
<td>tool-dfutil</td>
<td>Host side implementation of the DFU 1.0 and DFU 1.1 specifications</td>
</tr>
<tr>
<td>tool-jlink</td>
<td>SEGGER J-Link Software and Documentation Pack</td>
</tr>
<tr>
<td>tool-openocd</td>
<td>OpenOCD</td>
</tr>
<tr>
<td>tool-stm32duino</td>
<td>STM32Duino Tools</td>
</tr>
<tr>
<td>toolchain-gccarmnoneabi</td>
<td>gcc-arm-embedded</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](#).

1.10. Development Platforms
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>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>libOpenCM3</td>
<td>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.</td>
</tr>
<tr>
<td>mbed</td>
<td>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.</td>
</tr>
<tr>
<td>SPL</td>
<td>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.</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>

Boards

Note:
- You can list pre-configured boards by `platformio boards` command or PlatformIO Boards Explorer
- For more detailed board information please scroll tables below by horizontal.

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 B96B-F446VE</td>
<td>On-board</td>
<td>STM32F446VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</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>External</td>
<td>STM32F417VGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>192KB</td>
</tr>
<tr>
<td>Armstrap Eagle 2048</td>
<td>External</td>
<td>STM32F427VIT6</td>
<td>168MHz</td>
<td>1.99MB</td>
<td>256KB</td>
</tr>
<tr>
<td>Armstrap Eagle 512</td>
<td>External</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>192KB</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>STM32F411RET6</td>
<td>100MHz</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><strong>BlackPill F103C8</strong></td>
<td>External</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>20KB</td>
</tr>
<tr>
<td><strong>BlackPill F103C8 (128k)</strong></td>
<td>External</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td><strong>BluePill F103C6</strong></td>
<td>External</td>
<td>STM32F103C6T6</td>
<td>72MHz</td>
<td>32KB</td>
<td>10KB</td>
</tr>
<tr>
<td><strong>BluePill F103C8</strong></td>
<td>External</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>20KB</td>
</tr>
<tr>
<td><strong>BluePill F103C8 (128k)</strong></td>
<td>External</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td><strong>Demo F030F4</strong></td>
<td>External</td>
<td>STM32F030F4P6</td>
<td>48MHz</td>
<td>16KB</td>
<td>4KB</td>
</tr>
<tr>
<td><strong>FK407M1</strong></td>
<td>External</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td><strong>STM32F103C8 (20k RAM. 64k Flash)</strong></td>
<td>External</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>20KB</td>
</tr>
<tr>
<td><strong>STM32F103CB (20k RAM. 128k Flash)</strong></td>
<td>External</td>
<td>STM32F103CBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td><strong>STM32F103R8 (20k RAM. 64 Flash)</strong></td>
<td>External</td>
<td>STM32F103R8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>20KB</td>
</tr>
<tr>
<td><strong>STM32F103RB (20k RAM. 128k Flash)</strong></td>
<td>External</td>
<td>STM32F103RBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td><strong>STM32F103RC (48k RAM. 256k Flash)</strong></td>
<td>External</td>
<td>STM32F103RCT6</td>
<td>72MHz</td>
<td>256KB</td>
<td>48KB</td>
</tr>
<tr>
<td><strong>STM32F103RE (64k RAM. 512k Flash)</strong></td>
<td>External</td>
<td>STM32F103RET6</td>
<td>72MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td><strong>STM32F103T8 (20k RAM. 64 Flash)</strong></td>
<td>External</td>
<td>STM32F103T8T6</td>
<td>72MHz</td>
<td>20KB</td>
<td>64KB</td>
</tr>
<tr>
<td><strong>STM32F103TB (20k RAM. 128k Flash)</strong></td>
<td>External</td>
<td>STM32F103TBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td><strong>STM32F103VB (20k RAM. 128k Flash)</strong></td>
<td>External</td>
<td>STM32F103VBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td><strong>STM32F103VC (48k RAM. 256k Flash)</strong></td>
<td>External</td>
<td>STM32F103VCT6</td>
<td>72MHz</td>
<td>256KB</td>
<td>20KB</td>
</tr>
<tr>
<td><strong>STM32F103VD (64k RAM. 384k Flash)</strong></td>
<td>External</td>
<td>STM32F103VDT6</td>
<td>72MHz</td>
<td>384KB</td>
<td>48KB</td>
</tr>
<tr>
<td><strong>STM32F103VE (64k RAM. 512k Flash)</strong></td>
<td>External</td>
<td>STM32F103VET6</td>
<td>72MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td><strong>STM32F103ZC (48k RAM. 256k Flash)</strong></td>
<td>External</td>
<td>STM32F103ZCT6</td>
<td>72MHz</td>
<td>256KB</td>
<td>48KB</td>
</tr>
<tr>
<td><strong>STM32F103ZD (64k RAM. 384k Flash)</strong></td>
<td>External</td>
<td>STM32F103ZDT6</td>
<td>72MHz</td>
<td>384KB</td>
<td>64KB</td>
</tr>
<tr>
<td><strong>STM32F103ZE (64k RAM. 512k Flash)</strong></td>
<td>External</td>
<td>STM32F103ZET6</td>
<td>72MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td><strong>STM32F303CB (32k RAM. 128k Flash)</strong></td>
<td>External</td>
<td>STM32F303CBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td><strong>STM32F407VE (192k RAM. 512k Flash)</strong></td>
<td>External</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>502.23KB</td>
<td>128KB</td>
</tr>
<tr>
<td><strong>STM32F407VG (192k RAM. 1024k Flash)</strong></td>
<td>External</td>
<td>STM32F407VGT6</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>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>

### 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>

1.10. Development Platforms
## 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>

## 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>32KB</td>
</tr>
<tr>
<td>RAK811 LoRa Tracker</td>
<td>External</td>
<td>STM32L151RBT6</td>
<td>32MHz</td>
<td>128KB</td>
<td>16KB</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>STM32F446RET6</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>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>STM32F407VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Black STM32F407VG</td>
<td>External</td>
<td>STM32F407VGT6</td>
<td>168MHz</td>
<td>512KB</td>
<td>192KB</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>Black STM32F407ZE</td>
<td>External</td>
<td>STM32F407ZGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>192KB</td>
</tr>
<tr>
<td>Black STM32F407ZE</td>
<td>External</td>
<td>STM32F407ZET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Blue STM32F407VE Mini</td>
<td>External</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Nucleo G071RB</td>
<td>On-board</td>
<td>STM32G071RBT6</td>
<td>24MHz</td>
<td>2MB</td>
<td>128KB</td>
</tr>
<tr>
<td>P-Nucleo WB55RG</td>
<td>On-board</td>
<td>STM32WB55RG</td>
<td>64MHz</td>
<td>512KB</td>
<td>192.00KB</td>
</tr>
<tr>
<td>ST 32F3348DISCOVERY</td>
<td>On-board</td>
<td>STM32F334C8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>12KB</td>
</tr>
<tr>
<td>ST 32F401CDISCOVERY</td>
<td>On-board</td>
<td>STM32F401VCT6</td>
<td>84MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
<tr>
<td>ST 32F411EDISCOVERY</td>
<td>On-board</td>
<td>STM32F411VET6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST 32F413HDISCOVERY</td>
<td>On-board</td>
<td>STM32F413ZHT6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST 32F429IDISCOVERY</td>
<td>On-board</td>
<td>STM32F429ZIT6</td>
<td>180MHz</td>
<td>2MB</td>
<td>256KB</td>
</tr>
<tr>
<td>ST 32F469IDISCOVERY</td>
<td>On-board</td>
<td>STM32F469NIH6</td>
<td>180MHz</td>
<td>1MB</td>
<td>384KB</td>
</tr>
<tr>
<td>ST 32F769IDISCOVERY</td>
<td>On-board</td>
<td>STM32F769NIH6</td>
<td>216MHz</td>
<td>1MB</td>
<td>512KB</td>
</tr>
<tr>
<td>ST 32L0538DISCOVERY</td>
<td>On-board</td>
<td>STM32L053C8T6</td>
<td>32MHz</td>
<td>64KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ST 32L100DISCOVERY</td>
<td>On-board</td>
<td>STM32L100RCT6</td>
<td>32MHz</td>
<td>256KB</td>
<td>16KB</td>
</tr>
<tr>
<td>ST 32L476GDISCOVERY</td>
<td>On-board</td>
<td>STM32L476VGT6</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST 32L496GDISCOVERY</td>
<td>On-board</td>
<td>STM32L496AGI6</td>
<td>80MHz</td>
<td>1MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ST DISCO-L072CZ-LRWAN1</td>
<td>On-board</td>
<td>STM32L072CZ</td>
<td>32MHz</td>
<td>192KB</td>
<td>20KB</td>
</tr>
<tr>
<td>ST DISCO-L475VG-IOT01A</td>
<td>On-board</td>
<td>STM32L475VGT6</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Discovery F072RB</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>On-board</td>
<td>STM32F030R8T6</td>
<td>48MHz</td>
<td>64KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ST Nucleo F031K6</td>
<td>On-board</td>
<td>STM32F031K6T6</td>
<td>48MHz</td>
<td>32KB</td>
<td>4KB</td>
</tr>
<tr>
<td>ST Nucleo F042K6</td>
<td>On-board</td>
<td>STM32F042K6T6</td>
<td>48MHz</td>
<td>32KB</td>
<td>6KB</td>
</tr>
<tr>
<td>ST Nucleo F070RB</td>
<td>On-board</td>
<td>STM32F070RBT6</td>
<td>48MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>ST Nucleo F072RB</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>On-board</td>
<td>STM32F091RCT6</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>ST Nucleo F103RB</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>On-board</td>
<td>STM32F207ZGT6</td>
<td>120MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo F302R8</td>
<td>On-board</td>
<td>STM32F302R8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>16KB</td>
</tr>
<tr>
<td>ST Nucleo F303K8</td>
<td>On-board</td>
<td>STM32F303K8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>16KB</td>
</tr>
<tr>
<td>ST Nucleo F303RE</td>
<td>On-board</td>
<td>STM32F303RET6</td>
<td>72MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>ST Nucleo F303ZE</td>
<td>On-board</td>
<td>STM32F303ZET6</td>
<td>72MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>ST Nucleo F334R8</td>
<td>On-board</td>
<td>STM32F334R8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>16KB</td>
</tr>
<tr>
<td>ST Nucleo F401RE</td>
<td>On-board</td>
<td>STM32F401RET6</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
<tr>
<td>ST Nucleo F410RB</td>
<td>On-board</td>
<td>STM32F410RBT6</td>
<td>100MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>ST Nucleo F411RE</td>
<td>On-board</td>
<td>STM32F411RET6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo F412ZG</td>
<td>On-board</td>
<td>STM32F412ZGT6</td>
<td>100MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>ST Nucleo F413ZH</td>
<td>On-board</td>
<td>STM32F413ZHT6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo F429ZI</td>
<td>On-board</td>
<td>STM32F429ZIT6</td>
<td>180MHz</td>
<td>2MB</td>
<td>256KB</td>
</tr>
<tr>
<td>ST Nucleo F439ZI</td>
<td>On-board</td>
<td>STM32F439ZIT6</td>
<td>180MHz</td>
<td>2MB</td>
<td>256KB</td>
</tr>
<tr>
<td>ST Nucleo F446RE</td>
<td>On-board</td>
<td>STM32F446RET6</td>
<td>180MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<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>STM32F746ZGT6</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 F767ZI</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>On-board</td>
<td>STM32H743ZIT6</td>
<td>400MHz</td>
<td>2MB</td>
<td>1MB</td>
</tr>
<tr>
<td>ST Nucleo L011K4</td>
<td>On-board</td>
<td>STM32L011K4T6</td>
<td>32MHz</td>
<td>16KB</td>
<td>2KB</td>
</tr>
<tr>
<td>ST Nucleo L031K6</td>
<td>On-board</td>
<td>STM32L031K6T6</td>
<td>32MHz</td>
<td>32KB</td>
<td>8KB</td>
</tr>
<tr>
<td>Name</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>
</tr>
<tr>
<td>ST Nucleo L053R8</td>
<td>On-board</td>
<td>STM32L053R8T6</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>STM32L476RGT6</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo L486RG</td>
<td>On-board</td>
<td>STM32L486RGT6</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo L496ZG</td>
<td>On-board</td>
<td>STM32L496ZGT6</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 STM32F0308DISCOVERY</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>STM32F407VG7T6</td>
<td>168MHz</td>
<td>1MB</td>
<td>192KB</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 STM32LDISSCOVERY</td>
<td>On-board</td>
<td>STM32L152BRT6</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>STM32F7508-DK</td>
<td>On-board</td>
<td>STM32F750N8H6</td>
<td>216MHz</td>
<td>64KB</td>
<td>340KB</td>
</tr>
<tr>
<td>Seeed Studio</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>Debug</td>
<td>MCU</td>
<td>Frequency</td>
<td>Flash</td>
<td>RAM</td>
</tr>
<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>STM32F439VI</td>
<td>180MHz</td>
<td>2MB</td>
<td>256KB</td>
</tr>
<tr>
<td>Semtech</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>Debug</td>
<td>MCU</td>
<td>Frequency</td>
<td>Flash</td>
<td>RAM</td>
</tr>
<tr>
<td>NAMote72</td>
<td>External</td>
<td>STM32L152RC</td>
<td>32MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>TauLabs</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>Debug</td>
<td>MCU</td>
<td>Frequency</td>
<td>Flash</td>
<td>RAM</td>
</tr>
<tr>
<td>Sparky V1 F303</td>
<td>External</td>
<td>STM32F303CCT6</td>
<td>72MHz</td>
<td>256KB</td>
<td>40KB</td>
</tr>
<tr>
<td>VAE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Name</td>
<td>Debug</td>
<td>MCU</td>
<td>Frequency</td>
<td>Flash</td>
<td>RAM</td>
</tr>
<tr>
<td>VAKE v1.0</td>
<td>External</td>
<td>STM32F446RET6</td>
<td>180MHz</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>External</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:
- arduino-fade-all-pins
- arduino-internal-lib
• arduino-ping-hc04
• spl-blink
• spl-flash
• spl-uart

Debugging

PIO Unified Debugger - “1-click” solution for debugging with a zero configuration.

Tools & Debug Probes

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

```plaintext
; Latest stable version
[env:latest_stable]
platform = ststm8
board = ...

; Custom stable version
[env:custom_stable]
```
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-arduinostm8</td>
<td>Arduino Wiring-based Framework (STM8 Core)</td>
</tr>
<tr>
<td>framework-ststm8spl</td>
<td>STM8S/A Standard peripheral library</td>
</tr>
<tr>
<td>tool-openocd</td>
<td>OpenOCD</td>
</tr>
<tr>
<td>tool-stm8binutils</td>
<td>STM8 GNU binutils</td>
</tr>
<tr>
<td>tool-stm8tools</td>
<td>STM8 upload tools</td>
</tr>
<tr>
<td>toolchain-sdcc</td>
<td>Small Device C Compiler</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>SPL</td>
<td>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.</td>
</tr>
</tbody>
</table>

Boards

Note:
- You can list pre-configured boards by platformio boards command or PlatformIO Boards Explorer
- For more detailed board information please scroll tables below by horizontal.

1.10. Development Platforms
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** \( \text{platform} = \text{teensy} \)

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 `platformio 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_KEYBOARDONLY
• -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_MTPDISK`
• `-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:

```plaintext
[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:

• `arduino-blink`
• `arduino-hid-usb-mouse`
• `arduino-internal-libs`
• `MBED-BLINK`
• `MBED-DSP`
• `MBED-EVENTS`
• `MBED-RTOS`
• `MBED-SERIAL`

**Debugging**

*PIO Unified Debugger* - “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 PIO Unified Debugger 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 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

; Latest stable version
[env:latest_stable]
platform = teensy
board = ...

; Custom stable version
[env:custom_stable]
platform = teensy@x.y.z
board = ...

Upstream

[env:upstream_develop]
platform = https://github.com/platformio/platform-teensy.git
board = ...

1.10. Development Platforms
Packages

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>framework-arduinoteensy</td>
<td>Arduino Wiring-based Framework</td>
</tr>
<tr>
<td>framework-mbed</td>
<td>mbed Framework</td>
</tr>
<tr>
<td>tool-jlink</td>
<td>SEGGER J-Link Software and Documentation Pack</td>
</tr>
<tr>
<td>tool-teensty</td>
<td>Teensy Loader</td>
</tr>
<tr>
<td>toolchain-atmelavr</td>
<td>avr-gcc</td>
</tr>
<tr>
<td>toolchain-gccarmnoneabi</td>
<td>gcc-arm-embedded</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>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.</td>
</tr>
</tbody>
</table>

Boards

Note:

- You can list pre-configured boards by `platformio boards` command or PlatformIO Boards Explorer
- For more detailed board information please scroll tables below by horizontal.
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 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](https://www.ti.com/).

**Contents**

- Examples
- Debugging
- Stable and upstream versions
- Packages
- Frameworks
- Boards

**Examples**

Examples are listed from TI MSP430 development platform repository:

- arduino-blink
- arduino-internal-libs
- native-blink

**Debugging**

**PIO Unified Debugger** - “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 FraunchPad MSP-EXP430FR5739LP</td>
<td>MSP430FR5739</td>
<td>16MHz</td>
<td>16KB</td>
<td>512B</td>
</tr>
<tr>
<td>TI LaunchPad MSP-EXP430F5529LP</td>
<td>MSP430F5529</td>
<td>16MHz</td>
<td>128KB</td>
<td>8KB</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>64KB</td>
<td>2KB</td>
</tr>
<tr>
<td>TI LaunchPad MSP-EXP430FR6989LP</td>
<td>MSP430FR6989</td>
<td>8MHz</td>
<td>127KB</td>
<td>2KB</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 = ...
```

**Upstream**

```ini
[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 (MSP430 Core)</td>
</tr>
<tr>
<td>tool-mspdebug</td>
<td>MSPDebug</td>
</tr>
<tr>
<td>toolchain-timsp430</td>
<td>msp-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>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached 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 `platformio boards` command or [PlatformIO Boards Explorer](#).
- For more detailed board information please scroll tables below by horizontal.

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 LaunchPad MSP-EXP430FR5739LP</td>
<td>On-board</td>
<td>MSP430FR5739</td>
<td>16MHz</td>
<td>16KB</td>
<td>512B</td>
</tr>
<tr>
<td>TI LaunchPad MSP-EXP430FR5529LP</td>
<td>On-board</td>
<td>MSP430FR5529</td>
<td>16MHz</td>
<td>128KB</td>
<td>8KB</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>64KB</td>
<td>2KB</td>
</tr>
<tr>
<td>TI LaunchPad MSP-EXP430FR6989LP</td>
<td>On-board</td>
<td>MSP430FR6989</td>
<td>8MHz</td>
<td>127KB</td>
<td>2KB</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 `platform = tiva`

1.10. Development Platforms
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:

- arduino-blink
- arduino-internal-libs
- libopencm3-blink
- native-blink

**Debugging**

*PIO Unified Debugger* - “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 (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>LPTM4C1294NCPDT</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**

```ini
; Latest stable version
[env:latest_stable]
platform = titiva
board = ...

; Custom stable version
[env:custom_stable]
platform = titiva@x.y.z
board = ...
```

**Upstream**

```ini
[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 (LM4F Core)</td>
</tr>
<tr>
<td>framework-libopencm3</td>
<td>libOpenCM3 Framework</td>
</tr>
<tr>
<td>tool-openocd</td>
<td>OpenOCD</td>
</tr>
<tr>
<td>toolchain-gccarmnoneabi</td>
<td>gcc-arm-embedded</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](#).

1.10. Development Platforms
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/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.</td>
</tr>
</tbody>
</table>

Boards

Note:
- You can list pre-configured boards by `platformio boards` command or PlatformIO Boards Explorer
- For more detailed board information please scroll tables below by horizontal.

**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 LaunchPad (Stellaris) w/ lm4f120</td>
<td>On-board</td>
<td>LPLM4F120H5QR</td>
<td>80MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>(80MHz)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TI LaunchPad (Tiva C) w/ tm4c123</td>
<td>On-board</td>
<td>LPTM4C1230C3PM</td>
<td>80MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>(80MHz)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TI LaunchPad (Tiva C) w/ tm4c129</td>
<td>On-board</td>
<td>LPTM4C1294NCPDT</td>
<td>120MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>(120MHz)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></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
Examples

Examples are listed from WIZNet W7500 development platform repository:

- mbed-blink
- mbed-dsp
- mbed-events
- mbed-rtos
- mbed-serial

Debugging

PIO Unified Debugger - “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>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>mbed Framework</td>
</tr>
<tr>
<td>tool-jlink</td>
<td>SEGGER J-Link Software and Documentation Pack</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>gcc-arm-embedded</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>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.</td>
</tr>
</tbody>
</table>

Boards

Note:
- You can list pre-configured boards by `platformio boards` command or PlatformIO Boards Explorer
- For more detailed board information please scroll tables below by horizontal.

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

```
; Latest stable version
[env:latest_stable]
platform = native
board = ...

; Custom stable version
[env:custom_stable]
platform = native@x.y.z
board = ...
```

### Upstream

```
[env:upstream_develop]
platform = https://github.com/platformio/platform-native.git
board = ...
```

### Linux ARM

**Configuration** `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.

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-blink
- wiringpi-serial
Stable and upstream versions

You can switch between stable releases of Linux ARM 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_arm
board = ...

; Custom stable version
[env:custom_stable]
platform = linux_arm@x.y.z
board = ...
```

**Upstream**

```ini
[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-artik-sdk</td>
<td>ARTIK SDK is a C/C++ SDK targeting Samsung ARTIK platforms</td>
</tr>
<tr>
<td>framework-wiringpi</td>
<td>GPIO Interface library for the Raspberry Pi</td>
</tr>
<tr>
<td>toolchain-gccarmlinuxgnuabi</td>
<td>GCC for Linux ARM GNU EABI</td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AR-TIK SDK</td>
<td>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.</td>
</tr>
<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**

- You can list pre-configured boards by `platformio boards` command or PlatformIO Boards Explorer
- For more detailed board information please scroll tables below by horizontal.
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>

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 Kitra 520</td>
<td>No</td>
<td>EXYNOS3250</td>
<td>1000MHz</td>
<td>4GB</td>
<td>512MB</td>
</tr>
</tbody>
</table>

Samsung

<table>
<thead>
<tr>
<th>Name</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Samsung ARTIK 1020</td>
<td>No</td>
<td>EXYNOS5422</td>
<td>1500MHz</td>
<td>16GB</td>
<td>2GB</td>
</tr>
<tr>
<td>Samsung ARTIK 520</td>
<td>No</td>
<td>EXYNOS3250</td>
<td>1000MHz</td>
<td>4GB</td>
<td>512MB</td>
</tr>
<tr>
<td>Samsung ARTIK 530</td>
<td>No</td>
<td>S5P4418</td>
<td>1200MHz</td>
<td>4GB</td>
<td>512MB</td>
</tr>
<tr>
<td>Samsung ARTIK 710</td>
<td>No</td>
<td>S5P6818</td>
<td>1400MHz</td>
<td>4GB</td>
<td>1GB</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.

Contents

- Examples
- Stable and upstream versions
- Packages

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

```
# Latest stable version
[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-gcclinux32</td>
<td>GCC for Linux i686</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-gcclinux64</td>
<td>GCC for Linux x86_64</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.
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**

```ini
; Latest stable version
[env:latest_stable]
platform = windows_x86
board = ...

; Custom stable version
[env:custom_stable]
platform = windows_x86@x.y.z
board = ...
```

**Upstream**

```ini
[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</td>
</tr>
</tbody>
</table>

**1.11 Frameworks**

**1.11.1 Arduino**

Configuration `framework = 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.
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

Please read official documentation how to configure MCUdude’s Cores:

- Configure “MiniCore”
- Configure “MightyCore”
- Configure “MegaCore”

Debugging

PIO Unified Debugger - “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>3D printer controller</td>
<td>ST STM32</td>
<td>STM32F765VIT6</td>
<td>216MHz</td>
<td>2MB</td>
<td>512K</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>96K</td>
</tr>
<tr>
<td>Adafruit Bluefruit nRF52832 Feather</td>
<td>Nordic nRF52</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64K</td>
</tr>
<tr>
<td>Adafruit Feather nRF52840 Express</td>
<td>Nordic nRF52</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>796KB</td>
<td>64K</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 Zero (Programming/Debug Port)</td>
<td>Atmel SAM</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32K</td>
</tr>
<tr>
<td>Atmel ATSAMW25-XPRO</td>
<td>Atmel SAM</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32K</td>
</tr>
<tr>
<td>BBC micro:bit</td>
<td>Nordic nRF51</td>
<td>NRF51822</td>
<td>16MHz</td>
<td>256KB</td>
<td>16K</td>
</tr>
<tr>
<td>Calliope mini</td>
<td>Nordic nRF51</td>
<td>NRF51822</td>
<td>16MHz</td>
<td>256KB</td>
<td>16K</td>
</tr>
<tr>
<td>Delta DFBM-NQ620</td>
<td>Nordic nRF52</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64K</td>
</tr>
<tr>
<td>Essafisf ESP-WROVER-KIT</td>
<td>ESP32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320K</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>256K</td>
</tr>
<tr>
<td>Nordic Beacon Kit (PCA20006)</td>
<td>Nordic nRF51</td>
<td>NRF51822</td>
<td>32MHz</td>
<td>256KB</td>
<td>32K</td>
</tr>
<tr>
<td>Nordic nRF51 Dongle (PCA10031)</td>
<td>Nordic nRF51</td>
<td>NRF51822</td>
<td>32MHz</td>
<td>256KB</td>
<td>32K</td>
</tr>
<tr>
<td>Nordic nRF51X22 Development Kit (PCA1000X)</td>
<td>Nordic nRF51</td>
<td>NRF51822</td>
<td>32MHz</td>
<td>256KB</td>
<td>32K</td>
</tr>
<tr>
<td>Nordic nRF52-DK</td>
<td>Nordic nRF52</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64K</td>
</tr>
<tr>
<td>Nordic nRF52840-DK</td>
<td>Nordic nRF52</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>1MB</td>
<td>256K</td>
</tr>
<tr>
<td>Nordic nRF52840-DK (Adafruit BSP)</td>
<td>Nordic nRF52</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>796KB</td>
<td>243K</td>
</tr>
<tr>
<td>Nucleo G071RB</td>
<td>ST STM32</td>
<td>STM32G071RBT6</td>
<td>24MHz</td>
<td>2MB</td>
<td>128K</td>
</tr>
<tr>
<td>P-Nucleo WB55RG</td>
<td>ST STM32</td>
<td>STM32WB55RG</td>
<td>64MHz</td>
<td>512KB</td>
<td>192K</td>
</tr>
<tr>
<td>RedBearLab BLE Nano 1.5</td>
<td>Nordic nRF51</td>
<td>NRF51822</td>
<td>16MHz</td>
<td>256KB</td>
<td>32K</td>
</tr>
<tr>
<td>RedBearLab BLE Nano 2</td>
<td>Nordic nRF52</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64K</td>
</tr>
<tr>
<td>RedBearLab Blend 2</td>
<td>Nordic nRF52</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64K</td>
</tr>
<tr>
<td>RedBearLab nRF51822</td>
<td>Nordic nRF51</td>
<td>NRF51822</td>
<td>16MHz</td>
<td>256KB</td>
<td>16K</td>
</tr>
<tr>
<td>ST 32F746GDISCOVERY</td>
<td>ST STM32</td>
<td>STM32F746NGH6</td>
<td>216MHz</td>
<td>1MB</td>
<td>320K</td>
</tr>
<tr>
<td>ST DISCO-L072CZ-LRWAN1</td>
<td>ST STM32</td>
<td>STM32L072CZ</td>
<td>32MHz</td>
<td>192KB</td>
<td>20K</td>
</tr>
<tr>
<td>ST DISCO-L475VG-IOT01A</td>
<td>ST STM32</td>
<td>STM32L475VGT6</td>
<td>80MHz</td>
<td>1MB</td>
<td>128K</td>
</tr>
<tr>
<td>ST Discovery F072RB</td>
<td>ST STM32</td>
<td>STM32F072BT6</td>
<td>48MHz</td>
<td>128KB</td>
<td>16K</td>
</tr>
<tr>
<td>ST Nucleo F030R8</td>
<td>ST STM32</td>
<td>STM32F030R8T6</td>
<td>48MHz</td>
<td>64KB</td>
<td>8K</td>
</tr>
<tr>
<td>ST Nucleo F091RC</td>
<td>ST STM32</td>
<td>STM32F091RCT6</td>
<td>48MHz</td>
<td>256KB</td>
<td>32K</td>
</tr>
<tr>
<td>ST Nucleo F103RB</td>
<td>ST STM32</td>
<td>STM32F103RBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20K</td>
</tr>
<tr>
<td>ST Nucleo F207ZG</td>
<td>ST STM32</td>
<td>STM32F207ZGT6</td>
<td>120MHz</td>
<td>1MB</td>
<td>128K</td>
</tr>
<tr>
<td>ST Nucleo F302R8</td>
<td>ST STM32</td>
<td>STM32F302R8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>16K</td>
</tr>
<tr>
<td>ST Nucleo F303K8</td>
<td>ST STM32</td>
<td>STM32F303K8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>16K</td>
</tr>
<tr>
<td>ST Nucleo F303RE</td>
<td>ST STM32</td>
<td>STM32F303RET6</td>
<td>72MHz</td>
<td>512KB</td>
<td>64K</td>
</tr>
<tr>
<td>ST Nucleo F401RE</td>
<td>ST STM32</td>
<td>STM32F401RE6T6</td>
<td>84MHz</td>
<td>512KB</td>
<td>96K</td>
</tr>
<tr>
<td>ST Nucleo F411RE</td>
<td>ST STM32</td>
<td>STM32F411RE6T6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128K</td>
</tr>
<tr>
<td>ST Nucleo F429ZI</td>
<td>ST STM32</td>
<td>STM32F429ZIT6</td>
<td>180MHz</td>
<td>2MB</td>
<td>256K</td>
</tr>
<tr>
<td>ST Nucleo F446RE</td>
<td>ST STM32</td>
<td>STM32F446RE6T6</td>
<td>180MHz</td>
<td>512KB</td>
<td>128K</td>
</tr>
<tr>
<td>ST Nucleo F767ZI</td>
<td>ST STM32</td>
<td>STM32F767ZIT6</td>
<td>216MHz</td>
<td>2MB</td>
<td>512K</td>
</tr>
<tr>
<td>ST Nucleo H743ZI</td>
<td>ST STM32</td>
<td>STM32H743ZIT6</td>
<td>400MHz</td>
<td>2MB</td>
<td>1MB</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>STM32L073RZ</td>
<td>32MHz</td>
<td>192KB</td>
<td>20K</td>
</tr>
<tr>
<td>ST Nucleo L152RE</td>
<td>ST STM32</td>
<td>STM32L152RET6</td>
<td>32MHz</td>
<td>512KB</td>
<td>80K</td>
</tr>
</tbody>
</table>

Continued on next page
### Table 13 – 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 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 L452RE</td>
<td>ST STM32</td>
<td>STM32L452RET6</td>
<td>80MHz</td>
<td>256KB</td>
<td>40KB</td>
</tr>
<tr>
<td>ST Nucleo L476RG</td>
<td>ST STM32</td>
<td>STM32L476RGT6</td>
<td>80MHz</td>
<td>1MB</td>
<td>128K</td>
</tr>
<tr>
<td>ST Nucleo L496ZG</td>
<td>ST STM32</td>
<td>STM32L496ZGT6</td>
<td>80MHz</td>
<td>1MB</td>
<td>128K</td>
</tr>
<tr>
<td>ST Nucleo L496ZG-P</td>
<td>ST STM32</td>
<td>STM32L496ZGT6P</td>
<td>80MHz</td>
<td>320KB</td>
<td></td>
</tr>
<tr>
<td>ST Nucleo L4S2ZI</td>
<td>ST STM32</td>
<td>STM32L452KBU6</td>
<td>80MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
<tr>
<td>ST STM32F308DISCOVERY</td>
<td>ST STM32</td>
<td>STM32F308R8T6</td>
<td>48MHz</td>
<td>64KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ST STM32F4DISCOVERY</td>
<td>ST STM32</td>
<td>STM32F407VGT6</td>
<td>16MHz</td>
<td>1MB</td>
<td>192K</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 STM8S-DISCOVERY</td>
<td>ST STM8</td>
<td>STM8S105C6T6</td>
<td>16MHz</td>
<td>32KB</td>
<td>2KB</td>
</tr>
<tr>
<td>Seeed Tiny BLE</td>
<td>Nordic nRF51</td>
<td>NRF51822</td>
<td>16MHz</td>
<td>256KB</td>
<td>16K</td>
</tr>
<tr>
<td>TI FraunPad MSP-EXP430FR5739LP</td>
<td>TI MSP430</td>
<td>MSP430F5739</td>
<td>16MHz</td>
<td>16K</td>
<td>512K</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>32K</td>
</tr>
<tr>
<td>TI LaunchPad (Tiva) C w/ m4c123 (80MHz)</td>
<td>TI TIVA</td>
<td>LPTM4C1230C3PM</td>
<td>80MHz</td>
<td>256KB</td>
<td>32K</td>
</tr>
<tr>
<td>TI LaunchPad (Tiva) C w/ tiva c129 (120MHz)</td>
<td>TI TIVA</td>
<td>LPTM4C1294NCPDT</td>
<td>120MHz</td>
<td>1MB</td>
<td>256K</td>
</tr>
<tr>
<td>TI LaunchPad MSP-EXP430F5529LP</td>
<td>TI MSP430</td>
<td>MSP430F5529</td>
<td>16MHz</td>
<td>128KB</td>
<td>8KB</td>
</tr>
<tr>
<td>TI LaunchPad MSP-EXP430F4133LP</td>
<td>TI MSP430</td>
<td>MSP430F4133</td>
<td>8MHz</td>
<td>15K</td>
<td>2KB</td>
</tr>
<tr>
<td>TI LaunchPad MSP-EXP430F3969LP</td>
<td>TI MSP430</td>
<td>MSP430F3969</td>
<td>8MHz</td>
<td>64K</td>
<td>2KB</td>
</tr>
<tr>
<td>TI LaunchPad MSP-EXP430F6989LP</td>
<td>TI MSP430</td>
<td>MSP430F6989</td>
<td>8MHz</td>
<td>127KB</td>
<td>2KB</td>
</tr>
<tr>
<td>TI LaunchPad MSP-EXP430G2553LP</td>
<td>TI MSP430</td>
<td>MSP430G2553</td>
<td>16MHz</td>
<td>16K</td>
<td>512K</td>
</tr>
<tr>
<td>XMC1100 Boot Kit</td>
<td>Infineon XMC</td>
<td>XMC1100</td>
<td>32MHz</td>
<td>64KB</td>
<td>16K</td>
</tr>
<tr>
<td>XMC1100 H-Bridge 2Go</td>
<td>Infineon XMC</td>
<td>XMC1100</td>
<td>32MHz</td>
<td>64KB</td>
<td>16K</td>
</tr>
<tr>
<td>XMC1100 XMC2Go</td>
<td>Infineon XMC</td>
<td>XMC1100</td>
<td>32MHz</td>
<td>64KB</td>
<td>16K</td>
</tr>
<tr>
<td>XMC1300 Boot Kit</td>
<td>Infineon XMC</td>
<td>XMC1300</td>
<td>32MHz</td>
<td>64KB</td>
<td>16K</td>
</tr>
<tr>
<td>XMC1300 Sense2GoL</td>
<td>Infineon XMC</td>
<td>XMC1300</td>
<td>32MHz</td>
<td>32KB</td>
<td>16K</td>
</tr>
<tr>
<td>XMC1400 Boot Kit</td>
<td>Infineon XMC</td>
<td>XMC1400</td>
<td>48MHz</td>
<td>1.95MB</td>
<td>16K</td>
</tr>
<tr>
<td>XMC4200 Distance2Go</td>
<td>Infineon XMC</td>
<td>XMC4200</td>
<td>80MHz</td>
<td>256KB</td>
<td>40K</td>
</tr>
<tr>
<td>XMC4700 Relax Kit</td>
<td>Infineon XMC</td>
<td>XMC4700</td>
<td>144MHz</td>
<td>2.00MB</td>
<td>1.95</td>
</tr>
<tr>
<td>u-blox EVK-NINA-B1</td>
<td>Nordic nRF52</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64K</td>
</tr>
</tbody>
</table>

#### External Debug Tools

Boards listed below are compatible with PIO Unified Debugger 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>192K</td>
</tr>
<tr>
<td>3D Printer control board</td>
<td>ST STM32</td>
<td>STM32F446RET6</td>
<td>180MHz</td>
<td>512KB</td>
<td>128K</td>
</tr>
<tr>
<td>ALKS ESP32</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320K</td>
</tr>
<tr>
<td>Adafruit Circuit Playground Express</td>
<td>Atmel SAM</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32K</td>
</tr>
<tr>
<td>Adafruit Crickit M0</td>
<td>Atmel SAM</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32K</td>
</tr>
<tr>
<td>Adafruit ESP32 Feather</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320K</td>
</tr>
<tr>
<td>Adafruit Feather M0</td>
<td>Atmel SAM</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32K</td>
</tr>
<tr>
<td>Adafruit Feather M0 Express</td>
<td>Atmel SAM</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32K</td>
</tr>
<tr>
<td>Adafruit Feather M4 Express</td>
<td>Atmel SAM</td>
<td>SAMD51J19A</td>
<td>120MHz</td>
<td>512KB</td>
<td>192K</td>
</tr>
<tr>
<td>Adafruit Gemma M0</td>
<td>Atmel SAM</td>
<td>SAMD21E18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32K</td>
</tr>
<tr>
<td>Adafruit Grand Central M4</td>
<td>Atmel SAM</td>
<td>SAMD51P20A</td>
<td>120MHz</td>
<td>1MB</td>
<td>256K</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>Adafruit Hallowing M0</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>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Adafruit ItsyBitsy M4</td>
<td>Atmel SAM</td>
<td>SAMD51G19A</td>
<td>120MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Adafruit Metro M0 Express</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 AirLiftLite</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 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 piRkey</td>
<td>Atmel SAM</td>
<td>SAMD21E18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</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>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 WAN 1300</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 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>192KB</td>
</tr>
<tr>
<td>Black STM32F407VG</td>
<td>ST STM32</td>
<td>STM32F407VGT6</td>
<td>168MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Black STM32F407ZE</td>
<td>ST STM32</td>
<td>STM32F407ZGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>192KB</td>
</tr>
<tr>
<td>Black STM32F407ZE</td>
<td>ST STM32</td>
<td>STM32F407ZET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>192KB</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>Blue STM32F407VE Mini</td>
<td>ST STM32</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>192KB</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>Bluey nRF52832 IoT</td>
<td>Nordic nRF52</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>BluedK</td>
<td>Nordic nRF51</td>
<td>NRF51822</td>
<td>32MHz</td>
<td>256KB</td>
<td>32KB</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>DIOF 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>ESP32yn 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>Expressif ESP32 Dev Module</td>
<td>Espressif 32</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>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>F407VG</td>
<td>ST STM32</td>
<td>STM32F407VGT6</td>
<td>168MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>FK407M1</td>
<td>ST STM32</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>FireBeetle-ESP32</td>
<td>Espressif32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Frog Board ESP32</td>
<td>Espressif32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Heltec WiFi LoRa 32</td>
<td>Espressif32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>8MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Heltec WiFi LoRa 32 (V2)</td>
<td>Espressif32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>8MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Heltec Wireless Stick</td>
<td>Espressif32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Hornbill ESP32 Dev</td>
<td>Espressif32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Hornbill ESP32 Minima</td>
<td>Espressif32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>M200 V2</td>
<td>ST STM32</td>
<td>STM32F070CBT6</td>
<td>48MHz</td>
<td>120KB</td>
<td>32KB</td>
</tr>
<tr>
<td>MH ET LIVE ESP32DevKIT</td>
<td>Espressif32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>MH ET LIVE ESP32MiniKit</td>
<td>Espressif32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</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>Macchina M2</td>
<td>Atmel SAM</td>
<td>AT91SAM3X8E</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</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>STM32F103CBT6</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>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>STM32F050GW6</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>Espressif32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Olimex ESP32-EVB</td>
<td>Espressif32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Olimex ESP32-GATEWAY</td>
<td>Espressif32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</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>Pycom LoPy</td>
<td>Espressif32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Pycom LoPy4</td>
<td>Espressif32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>1.25MB</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>RAK811 LoRa Tracker</td>
<td>ST STM32</td>
<td>STM32L151RBT6</td>
<td>32MHz</td>
<td>128KB</td>
<td>16KB</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 ExpLoRe</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>STM32F103C8 (20 RAM, 64k Flash)</td>
<td>ST STM32</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>20KB</td>
</tr>
<tr>
<td>STM32F103CB (20 RAM, 128k Flash)</td>
<td>ST STM32</td>
<td>STM32F103CBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>STM32F103R8 (20 RAM, 64 Flash)</td>
<td>ST STM32</td>
<td>STM32F103R8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>20KB</td>
</tr>
<tr>
<td>STM32F103RR (20 RAM, 128 Flash)</td>
<td>ST STM32</td>
<td>STM32F103RBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>STM32F103RC (48 RAM, 256k Flash)</td>
<td>ST STM32</td>
<td>STM32F103RCT6</td>
<td>72MHz</td>
<td>256KB</td>
<td>48KB</td>
</tr>
<tr>
<td>STM32F103RE (64 RAM, 512 Flash)</td>
<td>ST STM32</td>
<td>STM32F103RET6</td>
<td>72MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F103T8 (20 RAM, 64k Flash)</td>
<td>ST STM32</td>
<td>STM32F103T8T6</td>
<td>72MHz</td>
<td>20KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F103TB (20 RAM, 128 Flash)</td>
<td>ST STM32</td>
<td>STM32F103TBT6</td>
<td>72MHz</td>
<td>20KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F103VB (20 RAM, 128 Flash)</td>
<td>ST STM32</td>
<td>STM32F103VB6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>STM32F103VC (48 RAM, 256k Flash)</td>
<td>ST STM32</td>
<td>STM32F103VCT6</td>
<td>72MHz</td>
<td>256KB</td>
<td>48KB</td>
</tr>
<tr>
<td>STM32F103VD (64 RAM, 384k Flash)</td>
<td>ST STM32</td>
<td>STM32F103VDT6</td>
<td>72MHz</td>
<td>384KB</td>
<td>64KB</td>
</tr>
</tbody>
</table>

Continued on next page
Table 14 – 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>STM32F103VE (64k RAM, 512k Flash)</td>
<td>ST STM32</td>
<td>STM32F103VE6</td>
<td>72MHz</td>
<td>512KB</td>
<td>64KB</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>STM32F1032D (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>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>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 LoRaWAN</td>
<td>Atmel SAM</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Silecognition wESP32</td>
<td>Espressif32</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 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>SparkFun LoRa ESP32 Thing</td>
<td>Espressif32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>SparkFun LoRa Gateway I-Channel</td>
<td>Espressif32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</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>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>Espressif32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>TTGO T-Beam</td>
<td>Espressif32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>1.25MB</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>
<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 LC</td>
<td>Teensy</td>
<td>MKL26Z64</td>
<td>48MHz</td>
<td>62KB</td>
<td>8KB</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>VAKé v1.0</td>
<td>ST STM32</td>
<td>STM32F446RET6</td>
<td>180MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>WEMOS LOLIN D32</td>
<td>Espressif32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>WEMOS LOLIN D32 PRO</td>
<td>Espressif32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>WEMOS LOLIN L32</td>
<td>Espressif32</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>WeMos WiFi and Bluetooth Battery</td>
<td>Espressif32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>XinaBox CW02</td>
<td>Espressif32</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 nRF52</td>
<td>NRF51822</td>
<td>16MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>oddWires IoT-Bus Io</td>
<td>Espressif32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>oddWires IoT-Bus Proteus</td>
<td>Espressif32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>

**Examples**

- Arduino for Atmel AVR
- Arduino for Atmel SAM
- Arduino for Espressif 32

1.11. Frameworks
• Arduino for Espressif 8266
• 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>Atmel AVR</strong></td>
<td>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.</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>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 RISC-V64 dual core SoC, this platform is ported by Sipeed.</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>
<tr>
<td><strong>TI TIVA</strong></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 `platformio boards` command or PlatformIO Boards Explorer.
- For more detailed board information please scroll tables below by horizontal.

### 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>

### 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>
</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>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</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 Micro</td>
<td>Atmel AVR</td>
<td>No</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>On-board</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>Adafruit Circuit Playground Classic</td>
<td>Atmel AVR</td>
<td>No</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 Circuit 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 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>No</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>No</td>
<td>ATMega32u4</td>
<td>8MHz</td>
<td>28KB</td>
<td>2.50KB</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 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 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 Feather nRF52840 Express</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52840</td>
<td>64MHz</td>
<td>796KB</td>
<td>243KB</td>
</tr>
<tr>
<td>Adafruit Flora</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATMega32U4</td>
<td>8MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>Adafruit Gemma</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATTINY85</td>
<td>8MHz</td>
<td>8KB</td>
<td>512B</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 ItsyBitsy 3V/8MHz</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATMega32U4</td>
<td>8MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
</tbody>
</table>

Continued on next page
Table 15 – 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 ItsyBitsy 5V/16MHz</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>2KB</td>
<td>2.50KB</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 Metro</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATMEGA32P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</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 Pro Trinket 3V/12MHz (FTDI)</td>
<td>Atmel AVR</td>
<td>No</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>No</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>No</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>No</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 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>No</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>No</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>
<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 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>

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 Hinx</td>
<td>Atmel AVR</td>
<td>No</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>No</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>No</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>No</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
</tbody>
</table>

1.11. Frameworks
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>No</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>No</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>No</td>
<td>ATMEGA168</td>
<td>16MHz</td>
<td>14KB</td>
<td></td>
</tr>
<tr>
<td>Arduino BT ATmega328</td>
<td>Atmel AVR</td>
<td>No</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>AT91SAM33X8E</td>
<td>84MHz</td>
<td>512KB</td>
<td></td>
</tr>
<tr>
<td>Arduino Due (USB Native Port)</td>
<td>Atmel SAM</td>
<td>External</td>
<td>AT91SAM33X8E</td>
<td>84MHz</td>
<td>512KB</td>
<td></td>
</tr>
<tr>
<td>Arduino Dueilanove or Diecimila ATmega168</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATMEGA168</td>
<td>16MHz</td>
<td>14KB</td>
<td></td>
</tr>
<tr>
<td>Arduino Dueilanove or Diecimila ATmega328</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>30KB</td>
<td></td>
</tr>
<tr>
<td>Arduino Explora</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td></td>
</tr>
<tr>
<td>Arduino Ethernet</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td></td>
</tr>
<tr>
<td>Arduino Fio</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATMEGA328P</td>
<td>8MHz</td>
<td>30KB</td>
<td></td>
</tr>
<tr>
<td>Arduino Industrial 101</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td></td>
</tr>
<tr>
<td>Arduino Leonardo</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td></td>
</tr>
<tr>
<td>Arduino Leonardo ETH</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td></td>
</tr>
<tr>
<td>Arduino LilyPad ATmega168</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATMEGA168</td>
<td>8MHz</td>
<td>14KB</td>
<td></td>
</tr>
<tr>
<td>Arduino LilyPad ATmega328</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATMEGA328P</td>
<td>8MHz</td>
<td>30KB</td>
<td></td>
</tr>
<tr>
<td>Arduino LilyPad USB</td>
<td>Atmel AVR</td>
<td>No</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 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>No</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>No</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>No</td>
<td>ATMEGA2560</td>
<td>16MHz</td>
<td>248KB</td>
<td></td>
</tr>
<tr>
<td>Arduino Micro</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td></td>
</tr>
<tr>
<td>Arduino Mini ATmega168</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATMEGA168</td>
<td>16MHz</td>
<td>14KB</td>
<td></td>
</tr>
<tr>
<td>Arduino Mini ATmega328</td>
<td>Atmel AVR</td>
<td>No</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>No</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>No</td>
<td>ATMEGA8</td>
<td>16MHz</td>
<td>7KB</td>
<td></td>
</tr>
</tbody>
</table>
Table 16 – 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 Nano ATmega168</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATMEGA168</td>
<td>16MHz</td>
<td>14KB</td>
<td></td>
</tr>
<tr>
<td>Arduino Nano ATmega328</td>
<td>Atmel AVR</td>
<td>No</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>No</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>30KB</td>
<td></td>
</tr>
<tr>
<td>Arduino Pro or Pro Mini ATmega168 (3.3V, 8 MHz)</td>
<td>Atmel AVR</td>
<td>No</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>No</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>No</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>No</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>30KB</td>
<td></td>
</tr>
<tr>
<td>Arduino Robot Control</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td></td>
</tr>
<tr>
<td>Arduino Robot Motor</td>
<td>Atmel AVR</td>
<td>No</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>No</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td></td>
</tr>
<tr>
<td>Arduino Yun</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td></td>
</tr>
<tr>
<td>Arduino Yun Mini</td>
<td>Atmel AVR</td>
<td>No</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>
### 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 ATtiny13</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATTINY13</td>
<td>1MHz</td>
<td>1KB</td>
<td>64B</td>
</tr>
<tr>
<td>Generic ATtiny13A</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATTINY13A</td>
<td>1MHz</td>
<td>1KB</td>
<td>64B</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>No</td>
<td>ATTINY2313</td>
<td>8MHz</td>
<td>2KB</td>
<td>128B</td>
</tr>
<tr>
<td>Generic ATtiny24</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATTINY24</td>
<td>8MHz</td>
<td>2KB</td>
<td>128B</td>
</tr>
<tr>
<td>Generic ATtiny25</td>
<td>Atmel AVR</td>
<td>No</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>No</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>No</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>No</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>No</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>No</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>No</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>
</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>No</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>No</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>No</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>

### 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>No</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>No</td>
<td>ATMEGA2560</td>
<td>16MHz</td>
<td>248KB</td>
<td>8KB</td>
</tr>
<tr>
<td>Controllino Mega</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATMEGA2560</td>
<td>16MHz</td>
<td>248KB</td>
<td>8KB</td>
</tr>
<tr>
<td>Controllino Mini</td>
<td>Atmel AVR</td>
<td>No</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>
</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</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>Atmel</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>Atmel</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</td>
<td>No</td>
<td>ATTINY85</td>
<td>16MHz</td>
<td>5.87KB</td>
<td>512B</td>
</tr>
<tr>
<td>Digistump DigiX</td>
<td>Atmel</td>
<td>Extern</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>192KB</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>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>

1.11. Frameworks
### 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>
## 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>No</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>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Espressif ESP-WROOM-KIT</td>
<td>Espressif 32</td>
<td>No</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>No</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>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</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>

## 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>No</td>
<td>ATMEGA2560</td>
<td>16MHz</td>
<td>252KB</td>
<td>8KB</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>HAM</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>
</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>Black 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>Black Pill 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>Blue Pill F103C6</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103C6T6</td>
<td>72MHz</td>
<td>32KB</td>
<td>10KB</td>
</tr>
<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>Blue Pill 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>192KB</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>STM32F103C8 (20k RAM. 128k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>STM32F103RB (20k RAM. 128k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103RB6</td>
<td>72MHz</td>
<td>64KB</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>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>STM32F103T8 (20k RAM. 64k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103T8T6</td>
<td>72MHz</td>
<td>20KB</td>
<td>64KB</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>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>STM32F103VE (64k RAM. 384k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103VET6</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>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>
<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>STM32F4Stamp F405</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F405RG76</td>
<td>168MHz</td>
<td>1MB</td>
<td>192KB</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 STM32</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 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>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>
## 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>

## 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>

## 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>No</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>No</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>No</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31KB</td>
<td>2KB</td>
</tr>
<tr>
<td>LowPowerLab Moteino</td>
<td>Atmel AVR</td>
<td>No</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>No</td>
<td>ATMEGA328P</td>
<td>8MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>LowPowerLab Moteino MEGA</td>
<td>Atmel AVR</td>
<td>No</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>

## 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>STM32F412ZGT6</td>
<td>1600MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
</tbody>
</table>

## Macchina

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Macchina M2</td>
<td>Atmel SAM</td>
<td>External</td>
<td>AT91SAM3X8E</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</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>

### 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>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>No</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>AT90CAN128</td>
<td>Atmel AVR</td>
<td>No</td>
<td>AT90CAN128</td>
<td>16MHz</td>
<td>127KB</td>
<td>4KB</td>
</tr>
<tr>
<td>AT90CAN32</td>
<td>Atmel AVR</td>
<td>No</td>
<td>AT90CAN32</td>
<td>16MHz</td>
<td>31KB</td>
<td>2KB</td>
</tr>
<tr>
<td>AT90CAN64</td>
<td>Atmel AVR</td>
<td>No</td>
<td>AT90CAN64</td>
<td>16MHz</td>
<td>63KB</td>
<td>4KB</td>
</tr>
<tr>
<td>ATmega128/A</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATMEGA128</td>
<td>16MHz</td>
<td>127KB</td>
<td>4KB</td>
</tr>
<tr>
<td>ATmega1280</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATMEGA1280</td>
<td>16MHz</td>
<td>127KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ATmega1281</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATMEGA1281</td>
<td>16MHz</td>
<td>127KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ATmega1284</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATMEGA1284</td>
<td>16MHz</td>
<td>127KB</td>
<td>16KB</td>
</tr>
<tr>
<td>ATmega1284P</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATMEGA1284P</td>
<td>16MHz</td>
<td>127KB</td>
<td>16KB</td>
</tr>
<tr>
<td>ATmega16</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATMEGA16</td>
<td>16MHz</td>
<td>15.50KB</td>
<td>1KB</td>
</tr>
<tr>
<td>ATmega164A</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATMEGA164A</td>
<td>16MHz</td>
<td>15.50KB</td>
<td>1KB</td>
</tr>
<tr>
<td>ATmega164P/PA</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATMEGA164P</td>
<td>16MHz</td>
<td>15.50KB</td>
<td>1KB</td>
</tr>
<tr>
<td>ATmega168/A</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATMEGA168</td>
<td>16MHz</td>
<td>15.50KB</td>
<td>1KB</td>
</tr>
<tr>
<td>ATmega168P/PA</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATMEGA168P</td>
<td>16MHz</td>
<td>15.50KB</td>
<td>1KB</td>
</tr>
<tr>
<td>ATmega168PB</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATMEGA168PB</td>
<td>16MHz</td>
<td>15.50KB</td>
<td>1KB</td>
</tr>
<tr>
<td>ATmega2560</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATMEGA2560</td>
<td>16MHz</td>
<td>255KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ATmega2561</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATMEGA2561</td>
<td>16MHz</td>
<td>255KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ATmega32</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATMEGA32</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>ATmega324/A</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATMEGA324A</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>ATmega324P</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATMEGA324P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>ATmega324PA</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATMEGA324PA</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>ATmega324PB</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATMEGA324PB</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>ATmega328</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATMEGA328</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>ATmega328P/PA</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>ATmega328PB</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATMEGA328PB</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
<tr>
<td>ATmega48/A</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATMEGA48</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>
</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>ATmega64/A</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATMEGA64</td>
<td>16MHz</td>
<td>63KB</td>
<td>4KB</td>
</tr>
<tr>
<td>ATmega640</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATMEGA640</td>
<td>16MHz</td>
<td>63KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ATmega644/A</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATMEGA644A</td>
<td>16MHz</td>
<td>63KB</td>
<td>4KB</td>
</tr>
<tr>
<td>ATmega644P/PA</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATMEGA644P</td>
<td>16MHz</td>
<td>63KB</td>
<td>4KB</td>
</tr>
<tr>
<td>ATmega8/A</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATMEGA8</td>
<td>16MHz</td>
<td>7.50KB</td>
<td>1KB</td>
</tr>
<tr>
<td>ATmega8535</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATMEGA8535</td>
<td>16MHz</td>
<td>7.50KB</td>
<td>512B</td>
</tr>
<tr>
<td>ATmega88/A</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATMEGA88</td>
<td>16MHz</td>
<td>7.50KB</td>
<td>1KB</td>
</tr>
<tr>
<td>ATmega88P/PA</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATMEGA88P</td>
<td>16MHz</td>
<td>7.50KB</td>
<td>1KB</td>
</tr>
<tr>
<td>ATmega88PB</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATMEGA88PB</td>
<td>16MHz</td>
<td>7.50KB</td>
<td>1KB</td>
</tr>
<tr>
<td>ATmega8P/PA</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATMEGA48P</td>
<td>16MHz</td>
<td>4KB</td>
<td>512B</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>No</td>
<td>ATMEGA168P</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>No</td>
<td>ATMEGA168P</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>No</td>
<td>ATMEGA328P</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>No</td>
<td>ATMEGA328P</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>STM32</td>
<td>External</td>
<td>STM32F103CB</td>
<td>162MHz</td>
<td>105.47KB</td>
<td>16.60KB</td>
</tr>
<tr>
<td>Microduino Core USB (Atmega32U4@16M,5V)</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATMEGA32U4</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>No</td>
<td>ATMEGA1284P</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>No</td>
<td>ATMEGA1284P</td>
<td>8MHz</td>
<td>127KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Microduino Core+ (Atmega644PA@16M,5V)</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATMEGA644P</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>No</td>
<td>ATMEGA644P</td>
<td>8MHz</td>
<td>63KB</td>
<td>4KB</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>
## PlatformIO Documentation, Release 4.0.1b3

### 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>

### 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>

### 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-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>
</tbody>
</table>

### Frameworks

1.11. Frameworks

423
## 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><strong>OROCA EduBot</strong></td>
<td><strong>Espressif 32</strong></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><strong>OSHChip</strong></td>
<td><strong>Nordic nRF51</strong></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><strong>Olimex MOD-WIFI-ESP8266(-DEV)</strong></td>
<td><strong>Espressif 8266</strong></td>
<td>No</td>
<td>ESP8266</td>
<td>80MHz</td>
<td>2MB</td>
<td>80KB</td>
</tr>
<tr>
<td><strong>Olimex PIC32-PINGUINO</strong></td>
<td><strong>Microchip PIC32</strong></td>
<td>No</td>
<td>32MX440F256H</td>
<td>80MHz</td>
<td>252KB</td>
<td>32KB</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><strong>Onehorse ESP32 Dev Module</strong></td>
<td><strong>Espressif 32</strong></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><strong>OpenBCI 32bit</strong></td>
<td><strong>Microchip PIC32</strong></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><strong>OpenEnergyMonitor emonPi</strong></td>
<td><strong>Atmel AVR</strong></td>
<td>No</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><strong>PONTECH UAV100</strong></td>
<td><strong>Microchip PIC32</strong></td>
<td>No</td>
<td>32MX440F512H</td>
<td>80MHz</td>
<td>508KB</td>
<td>32KB</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>No</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>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>No</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>

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>No</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>No</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 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>No</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>32KB</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>16KB</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>

## 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</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</td>
<td>On-board</td>
<td>NRF5232</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>RedBearLab Blend</td>
<td>Atmel</td>
<td>No</td>
<td>ATMEGA32U4</td>
<td>16MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>RedBearLab Blend 2</td>
<td>Nordic</td>
<td>On-board</td>
<td>NRF5232</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>RedBearLab Blend Micro 3.3V/16MHz (overclock)</td>
<td>Atmel</td>
<td>No</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>Atmel</td>
<td>No</td>
<td>ATMEGA32U4</td>
<td>8MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>RedBearLab nRF51822</td>
<td>Nordic</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</td>
<td>No</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>

### 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>No</td>
<td>ATMEGA1284P</td>
<td>8MHz</td>
<td>127KB</td>
<td>16KB</td>
</tr>
<tr>
<td>SODAQ Mbili</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATMEGA1284P</td>
<td>8MHz</td>
<td>127KB</td>
<td>16KB</td>
</tr>
<tr>
<td>SODAQ Moja</td>
<td>Atmel AVR</td>
<td>No</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>No</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>No</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>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>96KB</td>
</tr>
<tr>
<td>Black STM32F407VE</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407VE16</td>
<td>168MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Black STM32F407VG</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407VG16</td>
<td>168MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Black STM32F407ZE</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407ZG16</td>
<td>168MHz</td>
<td>1MB</td>
<td>192KB</td>
</tr>
<tr>
<td>Black STM32F407ZE</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407ZE16</td>
<td>168MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Blue STM32F407VE Mini</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407VE16</td>
<td>168MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Nucleo G071RB</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32G071RBT6</td>
<td>24MHz</td>
<td>2MB</td>
<td>128KB</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>192.00KB</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>320KB</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>20KB</td>
</tr>
<tr>
<td>ST DISCO-L475VG-IOT01A</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L475VG16</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

Continued on next page
Table 18 – 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 Discovery F072RB</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F072RB-T6</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>STM32F030R8-T6</td>
<td>48MHz</td>
<td>64KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ST Nucleo F091RC</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F091K-R7T6</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>STM32F103RBT6</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>STM32F207ZG-T6</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>STM32F302R8-T6</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>STM32F303K8-T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>16KB</td>
</tr>
<tr>
<td>ST Nucleo F303RE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F303RE-T6</td>
<td>72MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>ST Nucleo F401RE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F401RE-T6</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
<tr>
<td>ST Nucleo F411RE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F411RE-T6</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>STM32F429ZIT6</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>STM32F446RE-T6</td>
<td>180MHz</td>
<td>512KB</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>2MB</td>
<td>512KB</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>1MB</td>
</tr>
<tr>
<td>ST Nucleo L031K6</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L031K6-T6</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>STM32L053R8-T6</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>STM32L073RZ-T6</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>STM32L152RE-T6</td>
<td>32MHz</td>
<td>512KB</td>
<td>80KB</td>
</tr>
<tr>
<td>ST Nucleo L412KB</td>
<td>ST STM32</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>ST STM32</td>
<td>On-board</td>
<td>STM32L432KC-U6</td>
<td>80MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
<tr>
<td>ST Nucleo L452RE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L452RE-T6</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>STM32L476RG-T6</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>STM32L496ZG-T6</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo L496ZG-P</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L496ZG-T6-P</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>STM32F0308R8-T6</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>192KB</td>
</tr>
<tr>
<td>ST STM32VL DISCOVERY</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>
</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>No</td>
<td>ATMEGA1284P</td>
<td>16MHz</td>
<td>127KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Sanguino ATmega1284p (8MHz)</td>
<td>Atmel</td>
<td>No</td>
<td>AtMEGA1284P</td>
<td>8MHz</td>
<td>127KB</td>
<td>16KB</td>
</tr>
<tr>
<td>Sanguino ATmega644 or ATmega644A (16 MHz)</td>
<td>Atmel</td>
<td>No</td>
<td>ATMEGA644</td>
<td>16MHz</td>
<td>63KB</td>
<td>4KB</td>
</tr>
<tr>
<td>Sanguino ATmega644 or ATmega644A (8 MHz)</td>
<td>Atmel</td>
<td>No</td>
<td>ATMEGA644</td>
<td>8MHz</td>
<td>63KB</td>
<td>4KB</td>
</tr>
<tr>
<td>Sanguino ATmega644P or ATmega644PA (16 MHz)</td>
<td>Atmel</td>
<td>No</td>
<td>ATMEGA644P</td>
<td>16MHz</td>
<td>63KB</td>
<td>4KB</td>
</tr>
<tr>
<td>Sanguino ATmega644P or ATmega644PA (8 MHz)</td>
<td>Atmel</td>
<td>No</td>
<td>ATMEGA644P</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 LoRaWAN</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>No</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 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>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>
</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 ATmega128RFA1 Dev Board</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATMEGA128RFA1</td>
<td>16MHz</td>
<td>16KB</td>
<td>124KB</td>
</tr>
<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 Digital Sandbox</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATMEGA328P</td>
<td>8MHz</td>
<td>31.50KB</td>
<td>2KB</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>
<tr>
<td>SparkFun Fio V3 3.3V/8MHz</td>
<td>Atmel AVR</td>
<td>No</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 32</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>No</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>No</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>No</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>No</td>
<td>ATMEGA2560</td>
<td>8MHz</td>
<td>252KB</td>
<td>8KB</td>
</tr>
<tr>
<td>SparkFun MicroView</td>
<td>Atmel AVR</td>
<td>No</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>No</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>No</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>No</td>
<td>ATMEGA32U4</td>
<td>8MHz</td>
<td>28KB</td>
<td>2.50KB</td>
</tr>
<tr>
<td>SparkFun RedBoard</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</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 Serial 7-Segment Display</td>
<td>Atmel AVR</td>
<td>No</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>No</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 LaunchPad MSP-EXP430FR5739LP</td>
<td>TI MSP430</td>
<td>On-board</td>
<td>MSP430FR5739</td>
<td>16MHz</td>
<td>16KB</td>
<td>512B</td>
</tr>
<tr>
<td>TI LaunchPad (Stellaris) w/ tm4f120 (80MHz)</td>
<td>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>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>TIVA</td>
<td>On-board</td>
<td>LPTM4C1294NCPD</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>16MHz</td>
<td>128KB</td>
<td>8KB</td>
</tr>
<tr>
<td>TI LaunchPad 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 EXP430FR5969LP</td>
<td>TI MSP430</td>
<td>On-board</td>
<td>MSP430FR5969</td>
<td>8MHz</td>
<td>64KB</td>
<td>2KB</td>
</tr>
<tr>
<td>TI LaunchPad EXP430FR6989LP</td>
<td>TI MSP430</td>
<td>On-board</td>
<td>MSP430FR6989</td>
<td>8MHz</td>
<td>127KB</td>
<td>2KB</td>
</tr>
<tr>
<td>TI LaunchPad 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 T-Beam</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>1.25MB</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>No</td>
<td>MK20DX256</td>
<td>72MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
<tr>
<td>Teensy 3.5</td>
<td>Teensy</td>
<td>No</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>No</td>
<td>MK66FX1M0</td>
<td>180MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>Teensy LC</td>
<td>Teensy</td>
<td>No</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>No</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>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>No</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>No</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>No</td>
<td>ATMEGA328P</td>
<td>8MHz</td>
<td>30KB</td>
<td>2KB</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>

## 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 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 8266</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>
</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>No</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>No</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>WiFiduino</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>No</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>

### 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>

### 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>

### 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>No</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>No</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>No</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>No</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>No</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>No</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>NRF5232</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>No</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
</tbody>
</table>

1.11.2 ARTIK SDK

Configuration framework = 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.

For more detailed information please visit vendor site.

Examples

- ARTIK SDK 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 platformio boards command or PlatformIO Boards Explorer
- For more detailed board information please scroll tables below by horizontal.
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 Kitra 520</td>
<td>Linux ARM</td>
<td>No</td>
<td>EXYNOS3250</td>
<td>1000MHz</td>
<td>4GB</td>
<td>512MB</td>
</tr>
</tbody>
</table>

Samsung

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Samsung ARTIK 1020</td>
<td>Linux ARM</td>
<td>No</td>
<td>EXYNOS5422</td>
<td>1500MHz</td>
<td>16GB</td>
<td>2GB</td>
</tr>
<tr>
<td>Samsung ARTIK 520</td>
<td>Linux ARM</td>
<td>No</td>
<td>EXYNOS3250</td>
<td>1000MHz</td>
<td>4GB</td>
<td>512MB</td>
</tr>
<tr>
<td>Samsung ARTIK 530</td>
<td>Linux ARM</td>
<td>No</td>
<td>S5P4418</td>
<td>1200MHz</td>
<td>4GB</td>
<td>512MB</td>
</tr>
<tr>
<td>Samsung ARTIK 710</td>
<td>Linux ARM</td>
<td>No</td>
<td>S5P6818</td>
<td>1400MHz</td>
<td>4GB</td>
<td>1GB</td>
</tr>
</tbody>
</table>

1.11.3 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

PIO Unified Debugger - “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>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>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>192KB</td>
</tr>
<tr>
<td>ST STM32LDISCOVERY</td>
<td>ST STM32</td>
<td>STM32L152RBT6</td>
<td>32MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
</tbody>
</table>

### External Debug Tools

Boards listed below are compatible with PIO Unified Debugger 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>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>Black STM32F407VE</td>
<td>ST STM32</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Black STM32F407VG</td>
<td>ST STM32</td>
<td>STM32F407VGT6</td>
<td>168MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Black STM32F407ZE</td>
<td>ST STM32</td>
<td>STM32F407ZGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>192KB</td>
</tr>
<tr>
<td>Black STM32F407ZE</td>
<td>ST STM32</td>
<td>STM32F407ZET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>192KB</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>Blue STM32F407VE Mini</td>
<td>ST STM32</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>192KB</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>BlackPill F103C8 (128k)</td>
<td>ST STM32</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>F407VG</td>
<td>ST STM32</td>
<td>STM32F407VGT6</td>
<td>168MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>FK407M1</td>
<td>ST STM32</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>192KB</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>STM32F103RR (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, 128 Flash)</td>
<td>ST STM32</td>
<td>STM32F103RBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
</tbody>
</table>
Table 19 – 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>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>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>STM32F103T8 (20k RAM, 64k Flash)</td>
<td>ST STM32</td>
<td>STM32F103T8T6</td>
<td>72MHz</td>
<td>20KB</td>
<td>64KB</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>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>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>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>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>STM32F407VE (192k RAM, 1024k 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>
</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 platformio boards command or PlatformIO Boards Explorer
- 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>

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>

1.11. Frameworks
## 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>External</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>External</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>External</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>192KB</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>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>BlackPill F103C8</td>
<td>ST STM32</td>
<td></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></td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>BluePill F103C6</td>
<td>ST STM32</td>
<td></td>
<td>STM32F103C6T6</td>
<td>72MHz</td>
<td>32KB</td>
<td>10KB</td>
</tr>
<tr>
<td>BluePill F103C8</td>
<td>ST STM32</td>
<td></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></td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>FK407M1</td>
<td>ST STM32</td>
<td></td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>192KB</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>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>STM32F103RBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>STM32F103RC (48k RAM. 256 Flash)</td>
<td>ST STM32</td>
<td></td>
<td>STM32F103RCT6</td>
<td>72MHz</td>
<td>256KB</td>
<td>48KB</td>
</tr>
<tr>
<td>STM32F103RE (64k RAM. 512 Flash)</td>
<td>ST STM32</td>
<td></td>
<td>STM32F103RE6T</td>
<td>72MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F103T8 (20k RAM. 64 Flash)</td>
<td>ST STM32</td>
<td></td>
<td>STM32F103T8T6</td>
<td>72MHz</td>
<td>20KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F103TB (20k RAM. 128 Flash)</td>
<td>ST STM32</td>
<td></td>
<td>STM32F103TB16</td>
<td>72MHz</td>
<td>128KB</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. 256 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. 384 Flash)</td>
<td>ST STM32</td>
<td></td>
<td>STM32F103VD16</td>
<td>72MHz</td>
<td>384KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F103VE (64k RAM. 512 Flash)</td>
<td>ST STM32</td>
<td></td>
<td>STM32F103VE16</td>
<td>72MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F103ZC (48k RAM. 256 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. 384 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. 512 Flash)</td>
<td>ST STM32</td>
<td></td>
<td>STM32F103ZET6</td>
<td>72MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F303CB (32k RAM. 128k Flash)</td>
<td>ST STM32</td>
<td></td>
<td>STM32F303CBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>STM32F407VE (192k RAM. 512 Flash)</td>
<td>ST STM32</td>
<td></td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>502.23KB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F407VG (192k RAM. 1024 Flash)</td>
<td>ST STM32</td>
<td></td>
<td>STM32F407VG16</td>
<td>168MHz</td>
<td>1MB</td>
<td>192KB</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>Black STM32F407VE</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Black STM32F407VG</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407VGT6</td>
<td>168MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Black STM32F407ZE</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407ZGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>192KB</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>192KB</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>
<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>192KB</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>
</tbody>
</table>

### 1.11.4 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 `platformio boards` command or PlatformIO Boards Explorer
- 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>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>

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

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<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>

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

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

## 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>

### 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 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 `platformio boards` command or PlatformIO Boards Explorer
- For more detailed board information please scroll tables below by horizontal.

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>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><code>ESP-WROOM-02</code></td>
<td><code>Espressif 8266</code></td>
<td>No</td>
<td><code>ESP8266</code></td>
<td>80MHz</td>
<td>2MB</td>
<td>80KB</td>
</tr>
<tr>
<td><code>Espressif ESP8266 ESP-12E</code></td>
<td><code>Espressif 8266</code></td>
<td>No</td>
<td><code>ESP8266</code></td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
<tr>
<td><code>Espressif Generic ESP8266 ESP-01 1M</code></td>
<td><code>Espressif 8266</code></td>
<td>No</td>
<td><code>ESP8266</code></td>
<td>80MHz</td>
<td>1MB</td>
<td>80KB</td>
</tr>
<tr>
<td><code>Espressif Generic ESP8266 ESP-01 512k</code></td>
<td><code>Espressif 8266</code></td>
<td>No</td>
<td><code>ESP8266</code></td>
<td>80MHz</td>
<td>512KB</td>
<td>80KB</td>
</tr>
<tr>
<td><code>Espressif Generic ESP8266 ESP-07</code></td>
<td><code>Espressif 8266</code></td>
<td>No</td>
<td><code>ESP8266</code></td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
<tr>
<td><code>Generic ESP8285 Module</code></td>
<td><code>Espressif 8266</code></td>
<td>No</td>
<td><code>ESP8266</code></td>
<td>80MHz</td>
<td>1MB</td>
<td>80KB</td>
</tr>
<tr>
<td><code>Phoenix 1.0</code></td>
<td><code>Espressif 8266</code></td>
<td>No</td>
<td><code>ESP8266</code></td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
<tr>
<td><code>Phoenix 2.0</code></td>
<td><code>Espressif 8266</code></td>
<td>No</td>
<td><code>ESP8266</code></td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
<tr>
<td><code>WifInfo</code></td>
<td><code>Espressif 8266</code></td>
<td>No</td>
<td><code>ESP8266</code></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><code>Heltec Wifi kit 8</code></td>
<td><code>Espressif 8266</code></td>
<td>No</td>
<td><code>ESP8266</code></td>
<td>80MHz</td>
<td>4MB</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><code>Invent One</code></td>
<td><code>Espressif 8266</code></td>
<td>No</td>
<td><code>ESP8266</code></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><code>NodeMCU 0.9 (ESP-12 Module)</code></td>
<td><code>Espressif 8266</code></td>
<td>No</td>
<td><code>ESP8266</code></td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
</tr>
<tr>
<td><code>NodeMCU 1.0 (ESP-12E Module)</code></td>
<td><code>Espressif 8266</code></td>
<td>No</td>
<td><code>ESP8266</code></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><code>Olimex MOD-WIFI-ESP8266(-DEV)</code></td>
<td><code>Espressif 8266</code></td>
<td>No</td>
<td><code>ESP8266</code></td>
<td>80MHz</td>
<td>2MB</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>
1.11.6 ESP-IDF

Configuration `framework = espidf`

For more detailed information please visit vendor site.

Contents

- Debugging
- Examples
- Platforms
- Boards

Debugging

PIO Unified Debugger - “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>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 PIO Unified Debugger 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>Adafruit ESP32 Feather</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</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>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>ESP32yn 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 Minima</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>MH ET LIVE ESP32 DevKIT</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-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>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 T-Beam</td>
<td>Espressif 32</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>1.25MB</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 LOLLIN32</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

- ESP-IDF for Espressif 32

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>
</tbody>
</table>
## Boards

**Note:**

- You can list pre-configured boards by `platformio boards` command or PlatformIO Boards Explorer
- For more detailed board information please scroll tables below by horizontal.

### 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>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</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>No</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>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><strong>Frog Board ESP32</strong></td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</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><strong>ODROID-GO</strong></td>
<td>Espressif 32</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>16MB</td>
<td>320KB</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><strong>Heltec WiFi Kit 32</strong></td>
<td>Espressif 32</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td><strong>Heltec WiFi LoRa 32</strong></td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td><strong>Heltec WiFi LoRa 32 (V2)</strong></td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>8MB</td>
<td>320KB</td>
</tr>
<tr>
<td><strong>Heltec Wireless Stick</strong></td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>8MB</td>
<td>320KB</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><strong>Hornbill ESP32 Dev</strong></td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td><strong>Hornbill ESP32 Minima</strong></td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</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><strong>IntoRobot Fig</strong></td>
<td>Espressif 32</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</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><strong>M5Stack Core ESP32</strong></td>
<td>Espressif 32</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td><strong>M5Stack FIRE</strong></td>
<td>Espressif 32</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>16MB</td>
<td>6.25MB</td>
</tr>
<tr>
<td><strong>M5Stack GREY ESP32</strong></td>
<td>Espressif 32</td>
<td>No</td>
<td>ESP32</td>
<td>240MHz</td>
<td>16MB</td>
<td>520KB</td>
</tr>
<tr>
<td><strong>M5Stick-C</strong></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>

## 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>

## 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-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>
</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 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>

### 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 T-Beam</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>1.25MB</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>
### 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>

### 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 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>

### 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>
1.11.7 Freedom E SDK

Configuration framework = freedom-e-sdk

Open Source Software for Developing on the SiFive Freedom E Platform

For more detailed information please visit vendor site.

Contents
• Debugging
• Examples
• Platforms
• Boards

Debugging

PIO Unified Debugger - “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>
</tbody>
</table>
Examples

- Freedom E SDK for SiFive

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>SiFive</td>
<td>SiFive</td>
<td>On-board</td>
<td>FU540</td>
<td>1500MHz</td>
<td>32MB</td>
<td>8GB</td>
</tr>
<tr>
<td>HiFive Unleashed</td>
<td>SiFive</td>
<td>On-board</td>
<td>FE310</td>
<td>320MHz</td>
<td>16MB</td>
<td>16KB</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>

Boards

Note:

- You can list pre-configured boards by *platformio boards* command or PlatformIO Boards Explorer
- For more detailed board information please scroll tables below by horizontal.

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>

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 Kendryte FreeRTOS SDK

Configuration `framework = kendryte-freertos-sdk`

Kendryte SDK with FreeRTOS support

For more detailed information please visit vendor site.

Contents

- Debugging
- Examples
- Platforms
- Boards
Debugging

PIO Unified Debugger - “1-click” solution for debugging with a zero configuration.

Tools & Debug Probes

- External Debug Tools

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 PIO Unified Debugger 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>
</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 RISC-V64 dual core SoC, this platform is ported by Sipeed.</td>
</tr>
</tbody>
</table>

Boards

Note:

- You can list pre-configured boards by platformio boards command or PlatformIO Boards Explorer
- For more detailed board information please scroll tables below by horizontal.
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 MAIXDUNO</td>
<td>Kendryte K210</td>
<td>External</td>
<td>K210</td>
<td>400MHz</td>
<td>16MB</td>
<td>6MB</td>
</tr>
</tbody>
</table>

1.11.9 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

**PIO Unified Debugger** - “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 *PIO Unified Debugger* 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>
</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, this platform is ported by Sipeed.</td>
</tr>
</tbody>
</table>

Boards

---

**Note:**

- You can list pre-configured boards by *platformio boards* command or PlatformIO Boards Explorer
- For more detailed board information please scroll tables below by horizontal.

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>
</tbody>
</table>

### 1.11.10 libOpenCM3

**Configuration** *framework = 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, Tī Tiva and Stellaris, NXP LPC 11xx, 13xx, 15xx, 17xx parts, Atmel SAM3, Energy Micro EFM32 and others.

For more detailed information please visit [vendor site](#).
Debugging

PIO Unified Debugger - “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>ST Nucleo F103RB</td>
<td>ST STM32</td>
<td>STM32F103RBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</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>192KB</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>TI LaunchPad (Stellaris) w/ lm4f120</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</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</td>
<td>TI TIVA</td>
<td>LPTM4C1294NCPDT</td>
<td>120MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
</tbody>
</table>

**External Debug Tools**

Boards listed below are compatible with **PIO Unified Debugger** 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>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>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>Maple</td>
<td>ST STM32</td>
<td>STM32F103RBT6</td>
<td>72MHz</td>
<td>108KB</td>
<td>17KB</td>
</tr>
<tr>
<td>Maple (RE76)</td>
<td>ST STM32</td>
<td>STM32F103RE76</td>
<td>72MHz</td>
<td>256KB</td>
<td>48KB</td>
</tr>
<tr>
<td>Maple Mini Bootloader 2.0</td>
<td>ST STM32</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>120KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Maple Mini Original</td>
<td>ST STM32</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>108KB</td>
<td>17KB</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>STM32F103CB (20k RAM, 128k Flash)</td>
<td>ST STM32</td>
<td>STM32F103CBT6</td>
<td>72MHz</td>
<td>64KB</td>
<td>20KB</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>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>STM32F103TB (20k RAM, 64k Flash)</td>
<td>ST STM32</td>
<td>STM32F103T8T6</td>
<td>72MHz</td>
<td>20KB</td>
<td>64KB</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>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>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>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>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>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>
</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 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>TI TIVA</td>
<td>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.</td>
</tr>
</tbody>
</table>

Boards

Note:

- You can list pre-configured boards by platformio boards command or PlatformIO Boards Explorer
- 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>
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>
<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>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>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>STM32F103T8 (20k RAM. 64k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103T8T6</td>
<td>72MHz</td>
<td>20KB</td>
<td>64KB</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>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>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>

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>
### ST

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>HAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ST Nucleo F103RB</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F103RBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</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>192KB</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>
</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>HAM</th>
</tr>
</thead>
<tbody>
<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>LPTM4C1294NCPI</td>
<td>120MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
</tbody>
</table>

### 1.11.11 mbed

**Configuration** framework = 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.

For more detailed information please visit vendor site.
The configuration system

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-tls-client
- ST STM32: mbed-rtos-mesh-minimal

Mbed lib and Mbed OS 5

PlatformIO allows compiling projects with or without Mbed OS. 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>PIO_FRAMEWORK_MBED_RTOS_PRESENT</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 PIO_FRAMEWORK_MBED_RTOS_PRESENT
```

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>MBED_BUILD_PROFILE_RELEASE</td>
<td>Release profile (smallest firmware, minimal error info)</td>
</tr>
<tr>
<td>MBED_BUILD_PROFILE_DEBUG</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.

**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": ["FAMILY_STM32"],
    "supported_form_factors": ["ARDUINO", "MORPHO"],
    "core": "Cortex-M4F",
    "extra_labels_add": ["STM32F4", "STM32F401xE", "STM32F401RE"],
    "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"],
    "macros_add": ["USB_STM_HAL", "USBHOST_OTHER"],
    "device_has_add": [
      "SERIAL_ASYNCH",
      "FLASH",
      "MPU"
    ],
    "release_versions": ["2", "5"],
    "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 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$PROJECTSRC_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 Board*

After these steps, your project structure should look like this:

```
project_dir
  ├── include
  │    └── my_custom_board.json
  └── src
```

(continues on next page)
More information about adding custom targets can be found on the official page [Adding and configuring targets.](#)

See full examples with a custom board:

- [https://github.com/platformio/platform-nxplpc/tree/develop/examples/mbed-custom-target](https://github.com/platformio/platform-nxplpc/tree/develop/examples/mbed-custom-target)

### Debugging

**PIO Unified Debugger** - “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>
</tr>
</thead>
<tbody>
<tr>
<td>96Boards B96B-F446VE</td>
<td>ST STM32</td>
<td>STM32F446VET6</td>
<td>168MHz</td>
</tr>
<tr>
<td>ARM mbed LPC11U24 (+CAN)</td>
<td>NXP LPC</td>
<td>LPC11U24</td>
<td>48MHz</td>
</tr>
<tr>
<td>Atmel ATSAMR21-XPRO</td>
<td>Atmel SAM</td>
<td>SAMR21G18A</td>
<td>48MHz</td>
</tr>
<tr>
<td>Atmel ATSAMW25-XPRO</td>
<td>Atmel SAM</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
</tr>
<tr>
<td>Atmel SAMD21-XPRO</td>
<td>Atmel SAM</td>
<td>SAMD21J18A</td>
<td>48MHz</td>
</tr>
<tr>
<td>Atmel SAML21-XPRO-B</td>
<td>Atmel SAM</td>
<td>SAML21J18B</td>
<td>48MHz</td>
</tr>
<tr>
<td>BBC micro:bit</td>
<td>Nordic nRF51</td>
<td>NRF51822</td>
<td>16MHz</td>
</tr>
<tr>
<td>Bambino-210E</td>
<td>NXP LPC</td>
<td>LPC4330</td>
<td>204MHz</td>
</tr>
<tr>
<td>CoCo-ri-Co!</td>
<td>NXP LPC</td>
<td>LPC812</td>
<td>30MHz</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>Delta DFBM-NQ620</td>
<td>Nordic nRF52</td>
<td>NRF52832</td>
<td>64MHz</td>
</tr>
<tr>
<td>Delta DFCM-NNN40</td>
<td>Nordic nRF51</td>
<td>NRF51822</td>
<td>32MHz</td>
</tr>
<tr>
<td>Delta DFCM-NNN50</td>
<td>Nordic nRF51</td>
<td>NRF51822</td>
<td>32MHz</td>
</tr>
<tr>
<td>EFM32GG-STK3700 Giant Gecko</td>
<td>Silicon Labs EFM32</td>
<td>EFM32GG990F1024</td>
<td>48MHz</td>
</tr>
<tr>
<td>EFM32LG-STK3600 Leopard Gecko</td>
<td>Silicon Labs EFM32</td>
<td>EFM32LG990F256</td>
<td>48MHz</td>
</tr>
<tr>
<td>EFM32WG-STK3800 Wonder Gecko</td>
<td>Silicon Labs EFM32</td>
<td>EFM32WG990F256</td>
<td>48MHz</td>
</tr>
<tr>
<td>EFM32ZG-STK3200 Zero Gecko</td>
<td>Silicon Labs EFM32</td>
<td>EFM32ZG222F32</td>
<td>24MHz</td>
</tr>
<tr>
<td>Embedded Artists LPC4088 Display Module</td>
<td>NXP LPC</td>
<td>LPC4088</td>
<td>120MHz</td>
</tr>
<tr>
<td>Embedded Artists LPC4088 QuickStart Board</td>
<td>NXP LPC</td>
<td>LPC4088</td>
<td>120MHz</td>
</tr>
<tr>
<td>Ethernet IoT Starter Kit</td>
<td>Freescale Kinetis</td>
<td>MK64FN1M0VLL12</td>
<td>120MHz</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-K20D50M</td>
<td>Freescale Kinetis</td>
<td>MK20DX128VLT5</td>
<td>48MHz</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-K22F</td>
<td>Freescale Kinetis</td>
<td>MK22FN512VLT12</td>
<td>120MHz</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-K64F</td>
<td>Freescale Kinetis</td>
<td>MK64FN1M0VLL12</td>
<td>120MHz</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-K66F</td>
<td>Freescale Kinetis</td>
<td>MK66FN2M0VMD18</td>
<td>180MHz</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-K82F</td>
<td>Freescale Kinetis</td>
<td>MK82FN256VLL15</td>
<td>150MHz</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-KL05Z</td>
<td>Freescale Kinetis</td>
<td>MKL05Z32VF3M4</td>
<td>48MHz</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-KL25Z</td>
<td>Freescale Kinetis</td>
<td>MKL25Z128VLT4</td>
<td>48MHz</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-KL27Z</td>
<td>Freescale Kinetis</td>
<td>MKL27Z64VLT4</td>
<td>48MHz</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-KL45Z</td>
<td>Freescale Kinetis</td>
<td>MKL43Z256VLT4</td>
<td>48MHz</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-KL46Z</td>
<td>Freescale Kinetis</td>
<td>MKL46Z256VLT4</td>
<td>48MHz</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-KW41Z</td>
<td>Freescale Kinetis</td>
<td>MKW41Z512VHT4</td>
<td>48MHz</td>
</tr>
<tr>
<td>GAPuino GAP8</td>
<td>RISC-V GAP</td>
<td>GAP8</td>
<td>250MHz</td>
</tr>
<tr>
<td>JKSoft Wallbot BLE</td>
<td>Nordic nRF51</td>
<td>NRF51822</td>
<td>16MHz</td>
</tr>
<tr>
<td>L476DMW1K</td>
<td>ST STM32</td>
<td>STM32L476VGT6</td>
<td>80MHz</td>
</tr>
<tr>
<td>LPCXpresso11U68</td>
<td>NXP LPC</td>
<td>LPC11U68</td>
<td>50MHz</td>
</tr>
<tr>
<td>LPCXpresso824-MAX</td>
<td>NXP LPC</td>
<td>LPC824</td>
<td>30MHz</td>
</tr>
<tr>
<td>Maxim ARM mbed Enabled Development Platform for MAX32600</td>
<td>Maxim 32</td>
<td>MAX32600</td>
<td>24MHz</td>
</tr>
<tr>
<td>Mbed Connect Cloud</td>
<td>ST STM32</td>
<td>STM32F439ZIY6</td>
<td>168MHz</td>
</tr>
<tr>
<td>NXP LPC800-MAX</td>
<td>NXP LPC</td>
<td>LPC812</td>
<td>30MHz</td>
</tr>
<tr>
<td>NXP LPCXpresso54114</td>
<td>NXP LPC</td>
<td>LPC511U24</td>
<td>48MHz</td>
</tr>
<tr>
<td>NXP LPCXpresso5408</td>
<td>NXP LPC</td>
<td>LPC511U24</td>
<td>48MHz</td>
</tr>
<tr>
<td>NXP mbed LPC11U24</td>
<td>NXP LPC</td>
<td>LPC1768</td>
<td>96MHz</td>
</tr>
<tr>
<td>Nordic nRF51 Dongle (PCA10031)</td>
<td>Nordic nRF51</td>
<td>NRF51822</td>
<td>32MHz</td>
</tr>
<tr>
<td>Nordic nRF51822-nKIT</td>
<td>Nordic nRF51</td>
<td>NRF51822</td>
<td>16MHz</td>
</tr>
<tr>
<td>Nordic nRF51X22 Development Kit(PCA1000X)</td>
<td>Nordic nRF51</td>
<td>NRF51822</td>
<td>32MHz</td>
</tr>
<tr>
<td>Nordic nRF52-2DK</td>
<td>Nordic nRF52</td>
<td>NRF52832</td>
<td>64MHz</td>
</tr>
<tr>
<td>Nordic nRF52840-DK</td>
<td>Nordic nRF52</td>
<td>NRF52840</td>
<td>64MHz</td>
</tr>
<tr>
<td>RedBearLab BLE Nano 1.5</td>
<td>Nordic nRF51</td>
<td>NRF51822</td>
<td>16MHz</td>
</tr>
<tr>
<td>RedBearLab BLE Nano 2</td>
<td>Nordic nRF52</td>
<td>NRF52832</td>
<td>64MHz</td>
</tr>
<tr>
<td>RedBearLab nRF51822</td>
<td>Nordic nRF51</td>
<td>NRF51822</td>
<td>16MHz</td>
</tr>
<tr>
<td>RushUp Cloud-JAM</td>
<td>ST STM32</td>
<td>STM32F401RET6</td>
<td>84MHz</td>
</tr>
<tr>
<td>RushUp Cloud-JAM L4</td>
<td>ST STM32</td>
<td>STM32L476VGT6</td>
<td>80MHz</td>
</tr>
<tr>
<td>SLSTK3400A USB-enabled Happy Gecko</td>
<td>Silicon Labs EFM32</td>
<td>EFM32HG322F36</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>ST 32F3348DISCOVERY</td>
<td>ST STM32</td>
<td>STM32F3348CT6</td>
<td>72MHz</td>
</tr>
<tr>
<td>ST 32F401CDISCOVERY</td>
<td>ST STM32</td>
<td>STM32F401VCT6</td>
<td>84MHz</td>
</tr>
<tr>
<td>ST 32F413HDISCOVER</td>
<td>ST STM32</td>
<td>STM32F413ZHT6</td>
<td>100MHz</td>
</tr>
<tr>
<td>ST 32F429IDISCOVER</td>
<td>ST STM32</td>
<td>STM32F429ZIT6</td>
<td>180MHz</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 32F4691DISCOVERY</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 32F7691DISCOVERY</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>STM32L476VGTT6</td>
<td>80MHz</td>
</tr>
<tr>
<td>ST 32L496GDISCOVERY</td>
<td>ST STM32</td>
<td>STM32L496AGI6</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 DISCO-L475VG-IOT01A</td>
<td>ST STM32</td>
<td>STM32L475VGTT6</td>
<td>80MHz</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>STM32F303RETT6</td>
<td>72MHz</td>
</tr>
<tr>
<td>ST Nucleo F303ZE</td>
<td>ST STM32</td>
<td>STM32F303ZET6</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>STM32F401RETT6</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>STM32F411RETT6</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>STM32F446RETT6</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 L031K6</td>
<td>ST STM32</td>
<td>STM32L031K6T6</td>
<td>32MHz</td>
</tr>
<tr>
<td>ST Nucleo L053R8</td>
<td>ST STM32</td>
<td>STM32L053R8T6</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>STM32L152RETT6</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>ST Nucleo L476RG</td>
<td>ST STM32</td>
<td>STM32L476RGTT6</td>
<td>80MHz</td>
</tr>
<tr>
<td>ST Nucleo L486RG</td>
<td>ST STM32</td>
<td>STM32L486RGT6</td>
<td>80MHz</td>
</tr>
<tr>
<td>ST Nucleo L496ZG</td>
<td>ST STM32</td>
<td>STM32L496ZGT6</td>
<td>80MHz</td>
</tr>
<tr>
<td>ST Nucleo L496ZG-P</td>
<td>ST STM32</td>
<td>STM32L496ZGT6P</td>
<td>80MHz</td>
</tr>
<tr>
<td>ST Nucleo L4R5ZI</td>
<td>ST STM32</td>
<td>STM32L4R5ZIT6</td>
<td>120MHz</td>
</tr>
<tr>
<td>ST STM32F0308DISCOVERY</td>
<td>ST STM32</td>
<td>STM32F030R8T6</td>
<td>48MHz</td>
</tr>
<tr>
<td>ST STM32F0DISCOVERY</td>
<td>ST STM32</td>
<td>STM32F051R8T6</td>
<td>48MHz</td>
</tr>
<tr>
<td>ST STM32F3DISCOVERY</td>
<td>ST STM32</td>
<td>STM32F303VCT6</td>
<td>72MHz</td>
</tr>
<tr>
<td>ST STM32F4DISCOVERY</td>
<td>ST STM32</td>
<td>STM32F407VGT6</td>
<td>168MHz</td>
</tr>
</tbody>
</table>
## External Debug Tools

Boards listed below are compatible with PIO Unified Debugger 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>ST32F103C8T6</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>Expotet LoRa Module</td>
<td>ST STM32</td>
<td>ST32F411RET6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>Freescale Kinetics FRDM-KL82Z</td>
<td>Freescale Kinetics</td>
<td>MKL82Z128VLK7</td>
<td>96MHz</td>
<td>128KB</td>
<td>96KB</td>
</tr>
<tr>
<td>Freescale Kinetics FRDM-KW24D512</td>
<td>Freescale Kinetics</td>
<td>MKW24D512</td>
<td>50MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>Hexiwear</td>
<td>Freescale Kinetics</td>
<td>MK64FN1M0VDC12</td>
<td>120MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<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>ST32F411RET6</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>ST32F411RET6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>MultiTech mDot F411</td>
<td>ST STM32</td>
<td>ST32F411RET6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>MultiTech xDot</td>
<td>ST STM32</td>
<td>ST32L151CCU6</td>
<td>32MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>NAMote72</td>
<td>ST STM32</td>
<td>ST32L152RC</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>
</tbody>
</table>

Continued on next page
Table 22 – 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>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>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>u-blox 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 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</td>
<td>SAM</td>
</tr>
<tr>
<td>Freescale</td>
<td>Kinetis</td>
</tr>
<tr>
<td>Maxim 32</td>
<td></td>
</tr>
<tr>
<td>Nordic nRF51</td>
<td></td>
</tr>
<tr>
<td>Nordic nRF52</td>
<td></td>
</tr>
<tr>
<td>NXP LPC</td>
<td></td>
</tr>
<tr>
<td>RISC-V GAP</td>
<td></td>
</tr>
<tr>
<td>Silicon Labs EFM32</td>
<td></td>
</tr>
<tr>
<td>ST STM32</td>
<td></td>
</tr>
<tr>
<td>Teensy</td>
<td></td>
</tr>
<tr>
<td>WIZ-Net W7500</td>
<td></td>
</tr>
</tbody>
</table>

### Note:

- You can list pre-configured boards by `platformio boards` command or `PlatformIO Boards Explorer`
- For more detailed `board` information please scroll tables below by horizontal.
### 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>SAMD21G18A</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>MK20DX128VLLH5</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>MK22FN512VLLH12</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>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>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>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>MKL25Z128VLLK4</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>MKL27Z64VLLH4</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>MKL43Z256VLLH4</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>MKL46Z256VLL4</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>MKL82Z128VLLK7</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>

### 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>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>

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>

MikroElektronikana

<table>
<thead>
<tr>
<th>Name</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 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 (+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>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 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>
### 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>

### 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 mBuino</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>ST 32F3348DISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F334C8T6</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>STM32F401VCT6</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>STM32F413ZHT6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</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>256KB</td>
</tr>
<tr>
<td>ST 32F4691DISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F469NIH6</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>STM32F746NGH6</td>
<td>216MHz</td>
<td>1MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ST 32F7691DISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F769NIH6</td>
<td>216MHz</td>
<td>1MB</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>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>ST 32L0538DISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L053C8T6</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>STM32L476VGT6</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>STM32L496AGI6</td>
<td>80MHz</td>
<td>1MB</td>
<td>320KB</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>20KB</td>
</tr>
<tr>
<td>ST DISCO-L475VGT-1OT01A</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L475VGT6</td>
<td>80MHz</td>
<td>1MB</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>
<td>16KB</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>8KB</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>
<td>4KB</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>
<td>6KB</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>
<td>16KB</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>
<td>16KB</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>
<td>32KB</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>
<td>20KB</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>128KB</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>
<td>16KB</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>16KB</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>
<td>64KB</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>
<td>64KB</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>16KB</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>
<td>96KB</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>
<td>32KB</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>
<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>1MB</td>
<td>256KB</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>
<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>2MB</td>
<td>256KB</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>256KB</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>
<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>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo F476ZG</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F476ZGT6</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>STM32F756ZG</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>STM32F767ZIT6</td>
<td>216MHz</td>
<td>2MB</td>
<td>512KB</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>8KB</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>8KB</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>
<td>20KB</td>
</tr>
<tr>
<td>ST Nucleo L152RE</td>
<td>ST STM32</td>
<td>On-board</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>On-board</td>
<td>STM32L432KC6</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>STM32L433RC</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>STM32L476RT6</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>STM32L486RT6</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>STM32L496ZGT6</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</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>128KB</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>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>STM32F303VC16</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>192KB</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>
</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 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>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>SDT32832B</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>64MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>Thunderboard Sense 2 Sensor-to-Cloud Advanced IoT</td>
<td>Silicon Labs EFM32</td>
<td>On-board</td>
<td>EFR32MG12P432F10</td>
<td>60MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
</tbody>
</table>
# PlatformIO Documentation, Release 4.0.1b3

## 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</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</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</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>External</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>

1.11.12 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

PIO Unified Debugger - “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>GAPuno 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 `platformio boards` command or PlatformIO Boards Explorer
- For more detailed board information please scroll tables below by horizontal.

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>

1.11.13 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.

Examples

- Pumbaa for Espressif 32

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>
</tbody>
</table>

Boards

Note:

- You can list pre-configured boards by `platformio boards` command or PlatformIO Boards Explorer
- For more detailed board information please scroll tables below by horizontal.
1.11.14 Simba

Configuration framework = simba

Simba is an RTOS and build framework. It aims to make embedded programming easy and portable.

For more detailed information please visit vendor site.

Contents

• Debugging
• Examples
• Platforms
• Boards

Debugging

PIO Unified Debugger - “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 PIO Unified Debugger but DEPEND ON external debug probe. They ARE NOT READY for debugging. Please click on board name for the further details.
PlatformIO Documentation, Release 4.0.1b3

### Platforms

<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 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>
</tbody>
</table>

### Examples

- Simba for Atmel AVR
- Simba for Atmel SAM
- Simba for Espressif 32
- Simba for Espressif 8266

### Platforms

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atmel AVR</td>
<td>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.</td>
</tr>
<tr>
<td>Atmel SAM</td>
<td>Atmel</td>
</tr>
<tr>
<td>Espres sif 32</td>
<td>Espres sif 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>Espres sif 8266</td>
<td>Espres sif 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 `platformio boards` command or PlatformIO Boards Explorer
- For more detailed board information please scroll tables below by horizontal.

### 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>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 Mega or Mega 2560 ATmega2560</td>
<td>Atmel AVR</td>
<td>No</td>
<td>AT-MEGA2560</td>
<td>16MHz</td>
<td>248KB</td>
<td>8KB</td>
</tr>
<tr>
<td>(Mega 2560)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Arduino Nano ATmega328</td>
<td>Atmel AVR</td>
<td>No</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>No</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>
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>Seeeduino</td>
<td>Atmel AVR</td>
<td>No</td>
<td>ATMEGA328P</td>
<td>16MHz</td>
<td>31.50KB</td>
<td>2KB</td>
</tr>
</tbody>
</table>

1.11.15 SPL

Configuration `framework = 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.

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

PIO Unified Debugger - “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>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>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>192KB</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 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 PIO Unified Debugger 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>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>
</tbody>
</table>

Examples

- SPL for ST STM32
- SPL 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 `platformio boards` command or PlatformIO Boards Explorer.
- 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>

#### 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>External</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>External</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>External</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>192KB</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>192KB</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>

1.11.16 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.

## Contents

- Tutorials
- Debugging
- Examples
- Platforms
- Boards

### Tutorials

- **STM32Cube HAL and Nucleo-F401RE: debugging and unit testing**

### Debugging

**PIO Unified Debugger** - “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>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>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>96Boards B96B-F446VE</td>
<td>ST STM32</td>
<td>STM32F446VET6</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>Mbed Connect Cloud</td>
<td>ST STM32</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>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 32L76GDISCOVERY</td>
<td>ST STM32</td>
<td>STM32L76VGT6</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST 32L96GDISCOVERY</td>
<td>ST STM32</td>
<td>STM32L496AGI6</td>
<td>80MHz</td>
<td>1MB</td>
<td>320KB</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 DISCO-L475VG-IOT01A</td>
<td>ST STM32</td>
<td>STM32L475VGT6</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</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>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 F303K8</td>
<td>ST STM32</td>
<td>STM32F303K8T6</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 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>STM32F401RET6</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>STM32F411RET6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
</tbody>
</table>
### Table 24 – 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><strong>ST Nucleo F412ZG</strong></td>
<td>ST STM32</td>
<td>STM32F412ZGT6</td>
<td>100MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td><strong>ST Nucleo F413ZH</strong></td>
<td>ST STM32</td>
<td>STM32F413ZHT6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td><strong>ST Nucleo F429ZI</strong></td>
<td>ST STM32</td>
<td>STM32F429ZIT6</td>
<td>180MHz</td>
<td>2MB</td>
<td>256KB</td>
</tr>
<tr>
<td><strong>ST Nucleo F439ZI</strong></td>
<td>ST STM32</td>
<td>STM32F439ZIT6</td>
<td>180MHz</td>
<td>2MB</td>
<td>256KB</td>
</tr>
<tr>
<td><strong>ST Nucleo F440RE</strong></td>
<td>ST STM32</td>
<td>STM32F440RE</td>
<td>180MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td><strong>ST Nucleo F440ZE</strong></td>
<td>ST STM32</td>
<td>STM32F440ZET6</td>
<td>180MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td><strong>ST Nucleo F722ZE</strong></td>
<td>ST STM32</td>
<td>STM32F722ZET6</td>
<td>216MHz</td>
<td>512KB</td>
<td>256KB</td>
</tr>
<tr>
<td><strong>ST Nucleo F746ZG</strong></td>
<td>ST STM32</td>
<td>STM32F746ZGT6</td>
<td>216MHz</td>
<td>1MB</td>
<td>320KB</td>
</tr>
<tr>
<td><strong>ST Nucleo F756ZG</strong></td>
<td>ST STM32</td>
<td>STM32F756ZGT6</td>
<td>216MHz</td>
<td>1MB</td>
<td>320KB</td>
</tr>
<tr>
<td><strong>ST Nucleo F767ZI</strong></td>
<td>ST STM32</td>
<td>STM32F767ZIT6</td>
<td>400MHz</td>
<td>2MB</td>
<td>1MB</td>
</tr>
<tr>
<td><strong>ST Nucleo L011K4</strong></td>
<td>ST STM32</td>
<td>STM32L011K4T6</td>
<td>32MHz</td>
<td>16KB</td>
<td>2KB</td>
</tr>
<tr>
<td><strong>ST Nucleo L031K6</strong></td>
<td>ST STM32</td>
<td>STM32L031K6T6</td>
<td>32MHz</td>
<td>32KB</td>
<td>8KB</td>
</tr>
<tr>
<td><strong>ST Nucleo L053R8</strong></td>
<td>ST STM32</td>
<td>STM32L053R8T6</td>
<td>32MHz</td>
<td>64KB</td>
<td>8KB</td>
</tr>
<tr>
<td><strong>ST Nucleo L073RZ</strong></td>
<td>ST STM32</td>
<td>STM32L073RZ</td>
<td>32MHz</td>
<td>192KB</td>
<td>20KB</td>
</tr>
<tr>
<td><strong>ST Nucleo L052RE</strong></td>
<td>ST STM32</td>
<td>STM32L052RE</td>
<td>32MHz</td>
<td>512KB</td>
<td>80KB</td>
</tr>
<tr>
<td><strong>ST Nucleo L412KB</strong></td>
<td>ST STM32</td>
<td>STM32L412KBU6</td>
<td>80MHz</td>
<td>128KB</td>
<td>40KB</td>
</tr>
<tr>
<td><strong>ST Nucleo L432KC</strong></td>
<td>ST STM32</td>
<td>STM32L432KC6</td>
<td>80MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
<tr>
<td><strong>ST Nucleo L433RC-P</strong></td>
<td>ST STM32</td>
<td>STM32L433RC</td>
<td>80MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
<tr>
<td><strong>ST Nucleo L452RE</strong></td>
<td>ST STM32</td>
<td>STM32L452RE</td>
<td>80MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
<tr>
<td><strong>ST Nucleo L476RG</strong></td>
<td>ST STM32</td>
<td>STM32L476RGT6</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td><strong>ST Nucleo L486RG</strong></td>
<td>ST STM32</td>
<td>STM32L486RGT6</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td><strong>ST Nucleo L496ZG</strong></td>
<td>ST STM32</td>
<td>STM32L496ZGT6</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td><strong>ST Nucleo L496ZG-P</strong></td>
<td>ST STM32</td>
<td>STM32L496ZGT6</td>
<td>80MHz</td>
<td>1MB</td>
<td>320KB</td>
</tr>
<tr>
<td><strong>ST Nucleo L452RE</strong></td>
<td>ST STM32</td>
<td>STM32L452RE</td>
<td>80MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
<tr>
<td><strong>ST Nucleo L476RG</strong></td>
<td>ST STM32</td>
<td>STM32L476RGT6</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td><strong>ST Nucleo L486RG</strong></td>
<td>ST STM32</td>
<td>STM32L486RGT6</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td><strong>ST Nucleo L496ZG</strong></td>
<td>ST STM32</td>
<td>STM32L496ZGT6</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td><strong>ST Nucleo L496ZG-P</strong></td>
<td>ST STM32</td>
<td>STM32L496ZGT6</td>
<td>80MHz</td>
<td>1MB</td>
<td>320KB</td>
</tr>
<tr>
<td><strong>ST Nucleo L452RE</strong></td>
<td>ST STM32</td>
<td>STM32L452RE</td>
<td>80MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
<tr>
<td><strong>ST Nucleo L476RG</strong></td>
<td>ST STM32</td>
<td>STM32L476RGT6</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td><strong>ST Nucleo L486RG</strong></td>
<td>ST STM32</td>
<td>STM32L486RGT6</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td><strong>ST Nucleo L496ZG</strong></td>
<td>ST STM32</td>
<td>STM32L496ZGT6</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td><strong>ST Nucleo L496ZG-P</strong></td>
<td>ST STM32</td>
<td>STM32L496ZGT6</td>
<td>80MHz</td>
<td>1MB</td>
<td>320KB</td>
</tr>
<tr>
<td><strong>ST Sensor Node</strong></td>
<td>ST STM32</td>
<td>STM32L476JG</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td><strong>STM32F7508-DK</strong></td>
<td>ST STM32</td>
<td>STM32F750N8H6</td>
<td>216MHz</td>
<td>64KB</td>
<td>340KB</td>
</tr>
<tr>
<td><strong>Seeed Arch Max</strong></td>
<td>ST STM32</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td><strong>Seeed Wio 3G</strong></td>
<td>ST STM32</td>
<td>STM32F439VI</td>
<td>180MHz</td>
<td>2MB</td>
<td>256KB</td>
</tr>
<tr>
<td><strong>sakura.io Evaluation Board</strong></td>
<td>ST STM32</td>
<td>STM32F411RET6</td>
<td>100MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td><strong>u-blox C030-R410M IoT</strong></td>
<td>ST STM32</td>
<td>STM32F411RET6</td>
<td>100MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td><strong>1Bitsy</strong></td>
<td>ST STM32</td>
<td>STM32F415RGT</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td><strong>3D Printer Controller</strong></td>
<td>ST STM32</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td><strong>3D Printer control board</strong></td>
<td>ST STM32</td>
<td>STM32F446RET6</td>
<td>180MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
</tbody>
</table>

### External Debug Tools

Boards listed below are compatible with PIO Unified Debugger 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><strong>1Bitsy</strong></td>
<td>ST STM32</td>
<td>STM32F415RGT</td>
<td>168MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td><strong>3D Printer Controller</strong></td>
<td>ST STM32</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td><strong>3D Printer control board</strong></td>
<td>ST STM32</td>
<td>STM32F446RET6</td>
<td>180MHz</td>
<td>512KB</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>Armstrap Eagle 1024</td>
<td>ST ST M32</td>
<td>STM32F417VGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>192KB</td>
</tr>
<tr>
<td>Armstrap Eagle 2048</td>
<td>ST ST M32</td>
<td>STM32F427VIT6</td>
<td>168MHz</td>
<td>1.99MB</td>
<td>256KB</td>
</tr>
<tr>
<td>Armstrap Eagle 512</td>
<td>ST ST M32</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Black STM32F407VE</td>
<td>ST ST M32</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Black STM32F407VG</td>
<td>ST ST M32</td>
<td>STM32F407VGT6</td>
<td>168MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Black STM32F407ZE</td>
<td>ST ST M32</td>
<td>STM32F407ZGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>192KB</td>
</tr>
<tr>
<td>Black STM32F407ZE</td>
<td>ST ST M32</td>
<td>STM32F407ZET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>BlackPill F103C8</td>
<td>ST ST M32</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>20KB</td>
</tr>
<tr>
<td>BlackPill F103C8 (128k)</td>
<td>ST ST M32</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Blue STM32F407VE Mini</td>
<td>ST ST M32</td>
<td>STM32F103C6T6</td>
<td>72MHz</td>
<td>32KB</td>
<td>10KB</td>
</tr>
<tr>
<td>BluePill F103C6</td>
<td>ST ST M32</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>20KB</td>
</tr>
<tr>
<td>BluePill F103C8</td>
<td>ST ST M32</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Demo F030F4</td>
<td>ST ST M32</td>
<td>STM32F030F4P6</td>
<td>48MHz</td>
<td>16KB</td>
<td>4KB</td>
</tr>
<tr>
<td>Espotel LoRa Module</td>
<td>ST ST M32</td>
<td>STM32F411RET6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>F407VG</td>
<td>ST ST M32</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>FK407M1</td>
<td>ST ST M32</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>M200 V2</td>
<td>ST ST M32</td>
<td>STM32F70C8T6</td>
<td>48MHz</td>
<td>120KB</td>
<td>14.81KB</td>
</tr>
<tr>
<td>MTS Dragonfly</td>
<td>ST ST M32</td>
<td>STM32F411RET6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>Malyan M200 V1</td>
<td>ST ST M32</td>
<td>STM32F103CBT6</td>
<td>72MHz</td>
<td>120KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Maple</td>
<td>ST ST M32</td>
<td>STM32F103CBT6</td>
<td>72MHz</td>
<td>108KB</td>
<td>17KB</td>
</tr>
<tr>
<td>Maple (RET6)</td>
<td>ST ST M32</td>
<td>STM32F103RET6</td>
<td>72MHz</td>
<td>256KB</td>
<td>48KB</td>
</tr>
<tr>
<td>Maple Mini Bootloader 2.0</td>
<td>ST ST M32</td>
<td>STM32F103CBT6</td>
<td>72MHz</td>
<td>120KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Maple Mini Original</td>
<td>ST ST M32</td>
<td>STM32F103CBT6</td>
<td>72MHz</td>
<td>108KB</td>
<td>17KB</td>
</tr>
<tr>
<td>Microduino Core STM32 to Flash</td>
<td>ST ST M32</td>
<td>STM32F103CBT6</td>
<td>72MHz</td>
<td>105.47KB</td>
<td>16.60KB</td>
</tr>
<tr>
<td>MultiTech mDot</td>
<td>ST ST M32</td>
<td>STM32F411RET6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>MultiTech mDot F411</td>
<td>ST ST M32</td>
<td>STM32F411RET6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>MultiTech xDot</td>
<td>ST ST M32</td>
<td>STM32L151CCU6</td>
<td>32MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>N2+</td>
<td>ST ST M32</td>
<td>STM32F405RGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>192KB</td>
</tr>
<tr>
<td>NAMote72</td>
<td>ST ST M32</td>
<td>STM32L152RC</td>
<td>32MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>RAK811 LoRa Tracker</td>
<td>ST ST M32</td>
<td>STM32L151RBT6</td>
<td>32MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>RAK811 LoRa Tracker</td>
<td>ST ST M32</td>
<td>STM32L151RBT6</td>
<td>32MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>STM32F103C8 (20k RAM. 64k Flash)</td>
<td>ST ST M32</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>20KB</td>
</tr>
<tr>
<td>STM32F103C8 (20k RAM. 128k Flash)</td>
<td>ST ST M32</td>
<td>STM32F103CBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>STM32F103RB (20k RAM. 128k Flash)</td>
<td>ST ST M32</td>
<td>STM32F103RBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>STM32F103RC (48k RAM. 256k Flash)</td>
<td>ST ST M32</td>
<td>STM32F103RCT6</td>
<td>72MHz</td>
<td>256KB</td>
<td>48KB</td>
</tr>
<tr>
<td>STM32F103RE (64k RAM. 512k Flash)</td>
<td>ST ST M32</td>
<td>STM32F103RET6</td>
<td>72MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F103TF8 (20k RAM. 64k Flash)</td>
<td>ST ST M32</td>
<td>STM32F103TBT6</td>
<td>72MHz</td>
<td>64KB</td>
<td>20KB</td>
</tr>
<tr>
<td>STM32F103T8 (20k RAM. 128k Flash)</td>
<td>ST ST M32</td>
<td>STM32F103TBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>STM32F103VB (20k RAM. 128k Flash)</td>
<td>ST ST M32</td>
<td>STM32F103VBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>STM32F103VC (48k RAM. 256k Flash)</td>
<td>ST ST M32</td>
<td>STM32F103VCT6</td>
<td>72MHz</td>
<td>256KB</td>
<td>48KB</td>
</tr>
<tr>
<td>STM32F103VD (64k RAM. 384k Flash)</td>
<td>ST ST M32</td>
<td>STM32F103VDT6</td>
<td>72MHz</td>
<td>384KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F103VE (64k RAM. 512k Flash)</td>
<td>ST ST M32</td>
<td>STM32F103VET6</td>
<td>72MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F103ZC (48k RAM. 256k Flash)</td>
<td>ST ST M32</td>
<td>STM32F103ZCT6</td>
<td>72MHz</td>
<td>256KB</td>
<td>48KB</td>
</tr>
<tr>
<td>STM32F103ZD (64k RAM. 384k Flash)</td>
<td>ST ST M32</td>
<td>STM32F103ZDT6</td>
<td>72MHz</td>
<td>384KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F103ZE (64k RAM. 512k Flash)</td>
<td>ST ST M32</td>
<td>STM32F103ZET6</td>
<td>72MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F303CB (32k RAM. 128k Flash)</td>
<td>ST ST M32</td>
<td>STM32F303CBT6</td>
<td>72MHz</td>
<td>128KB</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>STM32F407VE (192k RAM, 512k Flash)</td>
<td>ST STM32</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>STM32F407VG</td>
<td>168MHz</td>
<td>1MB</td>
<td>192KB</td>
</tr>
<tr>
<td>STM32F4Stamp F405</td>
<td>ST STM32</td>
<td>STM32F405</td>
<td>168MHz</td>
<td>1MB</td>
<td>192KB</td>
</tr>
<tr>
<td>Sparky V1 F303</td>
<td>ST STM32</td>
<td>STM32F303CCT</td>
<td>72MHz</td>
<td>256KB</td>
<td>40KB</td>
</tr>
<tr>
<td>Tiny STM103T</td>
<td>ST STM32</td>
<td>STM32F103</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>VArE v1.0</td>
<td>ST STM32</td>
<td>STM32F446ET</td>
<td>180MHz</td>
<td>512KB</td>
<td>128KB</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>STM32F439ZIY</td>
<td>168MHz</td>
<td>2MB</td>
<td>256KB</td>
</tr>
<tr>
<td>u-blox ODIN-W2</td>
<td>ST STM32</td>
<td>STM32F439ZIY</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>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 `platformio boards` command or PlatformIO Boards Explorer
- 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 B96B-F446VE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F446VE</td>
<td>168MHz</td>
<td>512KB</td>
<td>128KB</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>External</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>External</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>External</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>192KB</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>
## 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>Black Pill F103C8</td>
<td>ST</td>
<td>External</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Black Pill F103C8 (128k)</td>
<td>ST</td>
<td>External</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Blue Pill F103C6</td>
<td>ST</td>
<td>External</td>
<td>STM32F103C6T6</td>
<td>72MHz</td>
<td>32KB</td>
<td>10KB</td>
</tr>
<tr>
<td>Blue Pill F103C8</td>
<td>ST</td>
<td>External</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Blue Pill F103C8 (128k)</td>
<td>ST</td>
<td>External</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>Demo F030F4</td>
<td>ST</td>
<td>External</td>
<td>STM32F030F4P6</td>
<td>48MHz</td>
<td>16KB</td>
<td>4KB</td>
</tr>
<tr>
<td>FK407M1</td>
<td>ST</td>
<td>External</td>
<td>STM32F407VE10</td>
<td>168MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>STM32F103C8 (20k RAM, 64k Flash)</td>
<td>ST</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</td>
<td>External</td>
<td>STM32F103CBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>STM32F103R8 (20k RAM, 64 Flash)</td>
<td>ST</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</td>
<td>External</td>
<td>STM32F103RBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>STM32F103RC (48k RAM, 256 Flash)</td>
<td>ST</td>
<td>External</td>
<td>STM32F103RCT6</td>
<td>72MHz</td>
<td>256KB</td>
<td>48KB</td>
</tr>
<tr>
<td>STM32F103RE (64k RAM, 512 Flash)</td>
<td>ST</td>
<td>External</td>
<td>STM32F103RET6</td>
<td>72MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F103T8 (20k RAM, 64 Flash)</td>
<td>ST</td>
<td>External</td>
<td>STM32F103T8T6</td>
<td>72MHz</td>
<td>20KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F103TB (20k RAM, 128 Flash)</td>
<td>ST</td>
<td>External</td>
<td>STM32F103TBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>STM32F103VB (20k RAM, 128 Flash)</td>
<td>ST</td>
<td>External</td>
<td>STM32F103VBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>20KB</td>
</tr>
<tr>
<td>STM32F103VC (48k RAM, 256 Flash)</td>
<td>ST</td>
<td>External</td>
<td>STM32F103VCT6</td>
<td>72MHz</td>
<td>256KB</td>
<td>48KB</td>
</tr>
<tr>
<td>STM32F103VD (64k RAM, 384 Flash)</td>
<td>ST</td>
<td>External</td>
<td>STM32F103VDT6</td>
<td>72MHz</td>
<td>384KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F103VE (64k RAM, 512 Flash)</td>
<td>ST</td>
<td>External</td>
<td>STM32F103VET6</td>
<td>72MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F103ZC (48k RAM, 256 Flash)</td>
<td>ST</td>
<td>External</td>
<td>STM32F103ZCT6</td>
<td>72MHz</td>
<td>256KB</td>
<td>48KB</td>
</tr>
<tr>
<td>STM32F103ZD (64k RAM, 384 Flash)</td>
<td>ST</td>
<td>External</td>
<td>STM32F103ZDT6</td>
<td>72MHz</td>
<td>384KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F103ZE (64k RAM, 512 Flash)</td>
<td>ST</td>
<td>External</td>
<td>STM32F103ZET6</td>
<td>72MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>STM32F303CB (32k RAM, 128 Flash)</td>
<td>ST</td>
<td>External</td>
<td>STM32F303CBT6</td>
<td>72MHz</td>
<td>128KB</td>
<td>32KB</td>
</tr>
<tr>
<td>STM32F407VE (192k RAM, 512 Flash)</td>
<td>ST</td>
<td>External</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>502.3KB</td>
<td>128KB</td>
</tr>
<tr>
<td>STM32F407VGG (192k RAM, 1024 Flash)</td>
<td>ST</td>
<td>External</td>
<td>STM32F407VGTT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>192KB</td>
</tr>
<tr>
<td>STM32F407M1</td>
<td>ST</td>
<td>External</td>
<td>STM32F407M1T6</td>
<td>168MHz</td>
<td>1MB</td>
<td>192KB</td>
</tr>
</tbody>
</table>

---

**PlatformIO Documentation, Release 4.0.1b3**
## 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>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>

### 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>32KB</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>16KB</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>32F723EDISCOVERY</td>
<td>ST STM32</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>ST STM32</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 STM32</td>
<td>External</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Black STM32F407VG</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407VGT6</td>
<td>168MHz</td>
<td>512KB</td>
<td>192KB</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>Black STM32F407ZE</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407ZGT6</td>
<td>168MHz</td>
<td>1MB</td>
<td>192KB</td>
</tr>
<tr>
<td>Black STM32F407ZE</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407ZET6</td>
<td>168MHz</td>
<td>512KB</td>
<td>192KB</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>192KB</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>12KB</td>
</tr>
<tr>
<td>ST 32F401CDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F401VC8T6</td>
<td>84MHz</td>
<td>256KB</td>
<td>64KB</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>128KB</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>128KB</td>
</tr>
<tr>
<td>ST 32F429DISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F429ZIT6</td>
<td>180MHz</td>
<td>2MB</td>
<td>256KB</td>
</tr>
<tr>
<td>ST 32F469DISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F469NHT6</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>STM32F746NGHT6</td>
<td>216MHz</td>
<td>1MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ST 32F769DISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F769NHT6</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>STM32L053C8T6</td>
<td>32MHz</td>
<td>8KB</td>
<td></td>
</tr>
<tr>
<td>ST 32L100DISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L100RCT6</td>
<td>32MHz</td>
<td>16KB</td>
<td>256KB</td>
</tr>
<tr>
<td>ST 32L476GDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L476VG8T6</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>STM32L496AG8T6</td>
<td>80MHz</td>
<td>1MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ST DISCO-L072CZ-LRWAN1</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L072C8T6</td>
<td>32MHz</td>
<td>20KB</td>
<td></td>
</tr>
<tr>
<td>ST DISCO-L475VG-IOT01A</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L475V8T6</td>
<td>80MHz</td>
<td>1MB</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>
<td>16KB</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>8KB</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>
<td>4KB</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>
<td>6KB</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>
<td>16KB</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>
<td>16KB</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>
<td>32KB</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>
<td>20KB</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>128KB</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>
<td>16KB</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>16KB</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>
<td>64KB</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>
<td>64KB</td>
</tr>
<tr>
<td>ST Nucleo F303R8</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F303R8T6</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>STM32F401RET6</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
<tr>
<td>ST Nucleo F410R8</td>
<td>ST STM32</td>
<td>On-board</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>On-board</td>
<td>STM32F411RET6</td>
<td>100MHz</td>
<td>128KB</td>
<td>32KB</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>256KB</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>
<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>2MB</td>
<td>256KB</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>256KB</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>
<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>512KB</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>512KB</td>
<td>256KB</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>320KB</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>320KB</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>512KB</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>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>
<td>2KB</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>8KB</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>8KB</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>
<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>ST Nucleo L152RE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L152RE</td>
<td>32MHz</td>
<td>512KB</td>
<td>80KB</td>
</tr>
<tr>
<td>ST Nucleo L412KB</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L412KB</td>
<td>80MHz</td>
<td>128KB</td>
<td>40KB</td>
</tr>
<tr>
<td>ST Nucleo L432KC</td>
<td>ST STM32</td>
<td>On-board</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>On-board</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>On-board</td>
<td>STM32L452RE</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>STM32L476RG</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>STM32L486RG</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>STM32L496ZG</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo L496ZG-P</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L496ZG</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>STM32L4R5ZI</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>STM32F0308R</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>STM32F051R</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>STM32F303V</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>STM32F407V</td>
<td>168MHz</td>
<td>1MB</td>
<td>192KB</td>
</tr>
<tr>
<td>ST STM32L073Z-EVAL</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L073ZV</td>
<td>32MHz</td>
<td>192KB</td>
<td>20KB</td>
</tr>
<tr>
<td>ST STM32LDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L152R</td>
<td>32MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>ST STM32VLDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F100K</td>
<td>24MHz</td>
<td>128KB</td>
<td>8KB</td>
</tr>
<tr>
<td>STM32F7508-DK</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F750N</td>
<td>216MHz</td>
<td>64KB</td>
<td>340KB</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>STM32F407VET</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>STM32F303CCT</td>
<td>72MHz</td>
<td>256KB</td>
<td>40KB</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>STM32F446RET</td>
<td>180MHz</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>
<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-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>External</td>
<td>STM32F439ZIY6</td>
<td>168MHz</td>
<td>2MB</td>
<td>256KB</td>
</tr>
</tbody>
</table>

1.11.17 Tizen RT

Configuration framework = tizenrt

Tizen RT is a lightweight RTOS-based platform to support low-end IoT devices
For more detailed information please visit vendor site.

Contents

- Debugging
- Examples
- Platforms
- Boards

Debugging

PIO Unified Debugger - “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>Samsung ARTIK053</td>
<td>Samsung ARTIK</td>
<td>S5JT200</td>
<td>320MHz</td>
<td>8MB</td>
<td>1.25MB</td>
</tr>
</tbody>
</table>

Examples

- Tizen RT for Samsung ARTIK

Platforms

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Samsung ARTIK</td>
<td>The Samsung ARTIK Smart IoT platform brings hardware modules and cloud services together, with built-in security and an ecosystem of tools and partners to speed up your time-to-market.</td>
</tr>
</tbody>
</table>

Boards

**Note:**

- You can list pre-configured boards by `platformio boards` command or PlatformIO Boards Explorer
- For more detailed board information please scroll tables below by horizontal.
1.11.18 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.

**Contents**

- **Examples**
- **Platforms**
- **Boards**

**Examples**

- WiringPi for Linux ARM

**Platforms**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Linux ARM</strong></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 `platformio boards` command or PlatformIO Boards Explorer
- For more detailed board information please scroll tables below by horizontal.
Raspberry Pi

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

1.12 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 `platformio boards` command or PlatformIO Boards Explorer
- For more detailed board information please scroll tables below by horizontal.

1.12.1 Aceinna IMU

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.

<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

PIO Unified 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).

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.
Compatible Tools | On-board | Default
--- | --- | ---
Black Magic Probe |  | Yes
J-LINK |  | 
ST-LINK |  | 

1.12.2 Atmel AVR

AT90CAN128

Contents

• AT90CAN128
  – Hardware
  – Configuration
  – Debugging
  – Frameworks

Hardware

Platform 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.

<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>127KB</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):

```
[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.

```
[env:AT90CAN128]
platform = atmelavr
board = AT90CAN128
; change microcontroller
```

(continues on next page)
board_build.mcu = at90can128

; change MCU frequency
board_build.f_cpu = 16000000L

Debugging

PIO Unified Debugger 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- 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 industy'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>31KB</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):
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.

```json
[env:AT90CAN32]
platform = atmelavr
board = AT90CAN32

; change microcontroller
board_build.mcu = at90can32

; change MCU frequency
board_build.f_cpu = 16000000L
```

### Debugging

PIO Unified Debugger 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- 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.
### 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

PIO Unified Debugger 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
Hardware

Platform *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.

<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>127KB</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*):

```ini
[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.

```ini
[env:ATmega128]
platform = atmelavr
board = ATmega128

; change microcontroller
board_build.mcu = atmega128

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

*PIO Unified Debugger* currently does not support ATmega128/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>
ATmega1280

Contents

• ATmega1280
  – Hardware
  – Configuration
  – Debugging
  – Frameworks

Hardware

Platform **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.

<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>127KB</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
board_build.mcu = atmega1280

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

**PIO Unified Debugger** currently does not support ATmega1280 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>

### ATmega1281

#### Contents

- ATmega1281
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

#### Hardware

Platform **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.

<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>127KB</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
```

(continues on next page)
board_build.mcu = atmega1281

; change MCU frequency
board_build.f_cpu = 16000000L

Debugging

PIO Unified Debugger currently does not support ATmega1281 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>

ATmega1284

Contents

- ATmega1284
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform 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.

<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>127KB</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):
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.

```plaintext
[env:ATmega1284]
platform = atmelavr
board = ATmega1284

; change microcontroller
board_build.mcu = atmega1284

; change MCU frequency
board_build.f_cpu = 16000000L
```

### Debugging

PIO Unified Debugger currently does not support ATmega1284 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>

### ATmega1284P

#### Contents

- ATmega1284P
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

#### Hardware

Platform **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.
### 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

*PIO Unified Debugger* currently does not support ATmega1284P 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>

### ATmega16

**Contents**

- **ATmega16**
  - Hardware
  - Configuration
Hardware

Platform 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.

<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>15.50KB</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):

```
[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.

```
[env:ATmega16]
platform = atmelavr
board = ATmega16

; change microcontroller
board_build.mcu = atmega16

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

PIO Unified Debugger currently does not support ATmega16 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
  - Debugging
  - Frameworks

### Hardware

Platform **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.

<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>15.50KB</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):

```ini
[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.

```ini
[env:ATmega164A]
platform = atmelavr
board = ATmega164A

; change microcontroller
board_build.mcu = atmega164a

; change MCU frequency
board_build.f_cpu = 16000000L
```

### Debugging

PIO Unified Debugger currently does not support ATmega164A board.
PlatformsIO Documentation, Release 4.0.1b3

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached 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- 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.

<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>15.50KB</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
```

(continues on next page)
board_build.mcu = atmega164p

; change MCU frequency
board_build.f_cpu = 16000000L

Debugging

PIO Unified Debugger currently does not support ATmega164P/PA 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>

ATmega168/A

Contents

- ATmega168/A
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform 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.

<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>15.50KB</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):
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.

; change microcontroller
board_build.mcu = atmega168

; change MCU frequency
board_build.f_cpu = 16000000L

Debugging

PIO Unified Debugger currently does not support ATmega168/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>

ATmega168P/PA

Hardware

Platform 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 industries most code-efficient architecture for C and assembly programming.
### Microcontroller

<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>15.50KB</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):

```ini
[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.

```ini
[env:ATmega168P]
platform = atmelavr
board = ATmega168P

; change microcontroller
board_build.mcu = atmega168p

; change MCU frequency
board_build.f_cpu = 16000000L
```

### Debugging

_PIO Unified Debugger_ currently does not support ATmega168P/PA 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>

### ATmega168PB

#### Contents

- ATmega168PB
  - Hardware
  - Configuration
Hardware

Platform *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.

<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>15.50KB</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)*:

```
[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.

```
[env:ATmega168PB]
platform = atmelavr
board = ATmega168PB

; change microcontroller
board_build.mcu = atmega168pb

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

*PIO Unified Debugger* currently does not support ATmega168PB 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>
ATmega2560

Contents

- ATmega2560
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform **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.

<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>255KB</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*:

```ini
[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.

```ini
[env:ATmega2560]
platform = atmelavr
board = ATmega2560

; change microcontroller
board_build.mcu = atmega2560

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

**PIO Unified Debugger** currently does not support ATmega2560 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>

ATmega2561

Contents

- ATmega2561
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform *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.

<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>255KB</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

PIO Unified Debugger 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- 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.

<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>31.50KB</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.

```
[env:ATmega32]
platform = atmelavr
board = ATmega32

; change microcontroller
board_build.mcu = atmega32

; change MCU frequency
board_build.f_cpu = 16000000L
```

### Debugging

**PIO Unified Debugger** 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- 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.
### Configuration

Please use ATmega324A ID for `board` option in “platformio.ini” *(Project Configuration File)*:

```python
[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.:

```python
[env:ATmega324A]
platform = atmelavr
board = ATmega324A

; change microcontroller
board_build.mcu = atmega324a

; change MCU frequency
board_build.f_cpu = 16000000L
```

### Debugging

*PIO Unified Debugger* currently does not support ATmega324A 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>

### ATmega324P

#### Contents

- *ATmega324P*
  - Hardware
  - Configuration
Hardware

Platform **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.

<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>31.50KB</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
board_build.mcu = atmega324p

; change MCU frequency
board_build.f_cpu = 16000000L
```

**Debugging**

**PIO Unified Debugger** currently does not support ATmega324P 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>
ATmega324PA

Contents

- ATmega324PA
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform **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.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA324PA</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>Microchip</td>
</tr>
</tbody>
</table>

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.f_cpu`, etc.

```
[env:ATmega324PA]
platform = atmelavr
board = ATmega324PA

; change microcontroller
board_build.mcu = atmega324pa

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

**PIO Unified Debugger** currently does not support ATmega324PA 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>

ATmega324PB

Contents

- ATmega324PB
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform 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.

<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>31.50KB</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

PIO Unified Debugger 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- 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.

<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>31.50KB</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.

```py
[env:ATmega328]
platform = atmelavr
board = ATmega328

; change microcontroller
board_build.mcu = atmega328

; change MCU frequency
board_build.f_cpu = 16000000L
```

**Debugging**

*PIO Unified Debugger* currently does not support ATmega328 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>

**ATmega328P/PA**

**Contents**

- ATmega328P/PA
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

**Hardware**

Platform *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.
<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>Microchip</td>
</tr>
</tbody>
</table>

### Configuration

Please use ATmega328P ID for `board` option in `platformio.ini` *(Project Configuration File)*:

```ini
[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.:

```ini
[env:ATmega328P]
platform = atmelavr
board = ATmega328P

; change microcontroller
board_build.mcu = atmega328p

; change MCU frequency
board_build.f_cpu = 16000000L
```

### Debugging

*PIO Unified Debugger* currently does not support ATmega328P/PA 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>

### ATmega328PB

**Contents**

- `ATmega328PB`
  - `Hardware`
  - `Configuration`
Hardware

Platform **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.

<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>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.

```
[env:ATmega328PB]
platform = atmelavr
board = ATmega328PB

; change microcontroller
board_build.mcu = atmega328pb

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

**PIO Unified Debugger** 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- 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.

<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>

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

PIO Unified Debugger currently does not support ATmega48/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>

ATmega48PB

Contents

- **ATmega48PB**
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform *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 industries most code-efficient architecture for C and assembly programming.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATmega48PB</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 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
```

(continues on next page)
board_build.mcu = atmega48pb

; change MCU frequency
board_build.f_cpu = 16000000L

Debugging

PIO Unified Debugger 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- 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.

<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>63KB</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):
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.

```plaintext
[env:ATmega64]
platform = atmelavr
board = ATmega64

; change microcontroller
board_build.mcu = atmega64

; change MCU frequency
board_build.f_cpu = 16000000L
```

### Debugging

*PIO Unified Debugger* 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

#### Contents

- ATmega640
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

### Hardware

Platform *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.
### Microcontroller

<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>63KB</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):

```ini
[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.

```ini
[env:ATmega640]
platform = atmelavr
board = ATmega640

; change microcontroller
board_build.mcu = atmega640

; change MCU frequency
board_build.f_cpu = 16000000L
```

### Debugging

PIO Unified Debugger 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
Hardware

Platform *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.

<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>63KB</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
board_build.mcu = atmega644a

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

*PIO Unified Debugger* 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- 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.

<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>Microchip</td>
</tr>
</tbody>
</table>

Configuration

Please use ATmega644P ID for board option in “platformio.ini” (Project Configuration File):

```
[env:ATmega644P]
platform = atmelavr
board = ATmega644P
```

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.

```
[env:ATmega644P]
platform = atmelavr
board = ATmega644P

; change microcontroller
board_build.mcu = atmega644p

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

PIO Unified Debugger currently does not support ATmega644P/PA 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>

ATmega8/A

Contents

- ATmega8/A
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform 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.

<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>7.50KB</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
```

(continues on next page)
; change MCU frequency
board_build.f_cpu = 16000000L

Debugging

PIO Unified Debugger currently does not support ATmega8/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>

ATmega8535

Contents

- ATmega8535
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform 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.

<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>7.50KB</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):
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}
platform = atmelavr
board = ATmega8535

; change microcontroller
board_build.mcu = ATmega8535

; change MCU frequency
board_build.f_cpu = 16000000L
```

### Debugging

*PIO Unified Debugger* 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- 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.
**PlatformIO Documentation, Release 4.0.1b3**

**Microcontroller**

<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>7.50KB</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)*:

```plaintext
[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.

```plaintext
[env:ATmega88]
platform = atmelavr
board = ATmega88

; change microcontroller
board_build.mcu = atmega88

; change MCU frequency
board_build.f_cpu = 16000000L
```

**Debugging**

*PIO Unified Debugger* currently does not support ATmega88/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>

**ATmega88P/PA**

<table>
<thead>
<tr>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>ATmega88P/PA</em></td>
</tr>
<tr>
<td>- <strong>Hardware</strong></td>
</tr>
<tr>
<td>- <strong>Configuration</strong></td>
</tr>
</tbody>
</table>
Hardware

Platform **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.

<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>7.50KB</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
board_build.mcu = atmega88p

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

*PIO Unified Debugger* currently does not support ATmega88P/PA 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>
**ATmega88PB**

## Contents

- **ATmega88PB**
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

### Hardware

Platform **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.

<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>7.50KB</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

*PIO Unified Debugger* 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>

ATmega8P/PA

Contents

- ATmega8P/PA
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform 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.

<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 ATmega8P/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

(continues on next page)
board.Build.mcu = atmega48p
; change MCU frequency
board.Build.f_cpu = 16000000L

Debugging

PIO Unified Debugger currently does not support ATmega8P/PA 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>

Adafruit Bluefruit Micro

Contents

- Adafruit Bluefruit Micro
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform 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.

<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):
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.

```plaintext
[env:bluefruitmicro]
platform = atmelavr
board = bluefruitmicro

; change microcontroller
board_build.mcu = atmega32u4

; change MCU frequency
board_build.f_cpu = 8000000L
```

### Debugging

**PIO Unified Debugger** currently does not support Adafruit Bluefruit Micro 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>

### Adafruit Circuit Playground Classic

#### Contents

- Adafruit Circuit Playground Classic
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

### Hardware

Platform **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.
### Microcontroller

<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)*:

```ini
[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.

```ini
[env:circuitplay_classic]
platform = atmelavr
board = circuitplay_classic

; change microcontroller
board_build.mcu = atmega32u4

; change MCU frequency
board_build.f_cpu = 8000000L
```

### Debugging

*PIO Unified Debugger* currently does not support Adafruit Circuit Playground Classic 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>

### Adafruit Feather 328P

#### Contents

- *Adafruit Feather 328P*
  - Hardware
  - Configuration
Hardware

Platform **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.

<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

**PIO Unified Debugger** currently does not support Adafruit Feather 328P 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>
Adafruit Feather 32u4

Contents

- Adafruit Feather 32u4
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform **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.

<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):

```ini
[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.

```ini
[env:feather32u4]
platform = atmelavr
board = feather32u4

; change microcontroller
board_build.mcu = atmega32u4

; change MCU frequency
board_build.f_cpu = 8000000L
```

Debugging

**PIO Unified Debugger** currently does not support Adafruit Feather 32u4 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>

Adafruit Flora

Contents

- Adafruit Flora
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform **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.

<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
```

(continues on next page)
board_build.mcu = atmega32u4
; change MCU frequency
board_build.f_cpu = 8000000L

Debugging

PIO Unified Debugger currently does not support Adafruit Flora 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>

Adafruit Gemma

Contents

- Adafruit Gemma
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform 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.

<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):
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.

```yaml
[env:gemma]
platform = atmelavr
board = gemma

; change microcontroller
board_build.mcu = attiny85

; change MCU frequency
board_build.f_cpu = 8000000L
```

### Debugging

**PIO Unified Debugger** currently does not support Adafruit Gemma 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>

## Adafruit ItsyBitsy 3V/8MHz

### Hardware

Platform **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.
**Microcontroller** | ATMEGA32U4  
---|---  
**Frequency** | 8MHz  
**Flash** | 28KB  
**RAM** | 2.50KB  
**Vendor** | Adafruit  

**Configuration**

Please use itsybitsy32u4_3V ID for `board` option in “platformio.ini” (Project Configuration File):

```
[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.

```
[env:itsybitsy32u4_3V]
platform = atmelavr
board = itsybitsy32u4_3V

; change microcontroller
board_build.mcu = atmega32u4

; change MCU frequency
board_build.f_cpu = 8000000L
```

**Debugging**

*PIO Unified Debugger* currently does not support Adafruit ItsyBitsy 3V/8MHz 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>

**Adafruit ItsyBitsy 5V/16MHz**

**Contents**

- Adafruit ItsyBitsy 5V/16MHz
  - Hardware
  - Configuration
Hardware

Platform *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.

<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>Adafruit</td>
</tr>
</tbody>
</table>

Configuration

Please use itsybitsy32u4_5V ID for board option in “platformio.ini” (Project Configuration File):

```
[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.

```
[env:itsybitsy32u4_5V]
platform = atmelavr
board = itsybitsy32u4_5V

; change microcontroller
board_build.mcu = atmega32u4

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

*PIO Unified Debugger* currently does not support Adafruit ItsyBitsy 5V/16MHz 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>
Adafruit Metro

Contents

- Adafruit Metro
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform 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.

<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>Adafruit</td>
</tr>
</tbody>
</table>

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

PIO Unified Debugger currently does not support Adafruit Metro board.

1.12. Boards 563
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached 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- 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.

<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):

```
[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.

```
[env:protrinket3ftdi]
platform = atmelavr
board = protrinket3ftdi

; change microcontroller
```

(continues on next page)
```python
board_build.mcu = atmega328p
;
# change MCU frequency
board_build.f_cpu = 12000000L
```

### Debugging

*PIO Unified Debugger* currently does not support Adafruit Pro Trinket 3V/12MHz (FTDI) board.

### Frameworks

#### Adafruit Pro Trinket 3V/12MHz (USB)

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.</td>
</tr>
</tbody>
</table>

#### Hardware

Platform *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.

<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):”
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.

```ini
[env:protrinket3]
platform = atmelavr
board = protrinket3

; change microcontroller
board_build.mcu = atmega328p

; change MCU frequency
board_build.f_cpu = 12000000L
```

## Debugging

PIO Unified Debugger currently does not support Adafruit Pro Trinket 3V/12MHz (USB) 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>

## Adafruit Pro Trinket 5V/16MHz (FTDI)

### Contents

- Adafruit Pro Trinket 5V/16MHz (FTDI)
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

### Hardware

Platform 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.
<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
board_build.mcu = atmega328p

; change MCU frequency
board_build.f_cpu = 16000000L
```

### Debugging

*PIO Unified Debugger* currently does not support Adafruit Pro Trinket 5V/16MHz (FTDI) 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>

### Adafruit Pro Trinket 5V/16MHz (USB)

**Contents**

- *Adafruit Pro Trinket 5V/16MHz (USB)*
  - Hardware
  - Configuration
Hardware

Platform **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.

<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):

```
[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.

```
[env:protrinket5]
platform = atmelavr
board = protrinket5

; change microcontroller
board_build.mcu = atmega328p

; change MCU frequency
board_build.f_cpu = 16000000L
```

**Debugging**

*PIO Unified Debugger* currently does not support Adafruit Pro Trinket 5V/16MHz (USB) 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>
Adafruit Trinket 3V/8MHz

Contents

• Adafruit Trinket 3V/8MHz
  – Hardware
  – Configuration
  – Debugging
  – Frameworks

Hardware

Platform 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.

<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):

```
[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.

```
[env:trinket3]
platform = atmelavr
board = trinket3

; change microcontroller
board_build.mcu = attiny85

; change MCU frequency
board_build.f_cpu = 8000000L
```

Debugging

PIO Unified Debugger currently does not support Adafruit Trinket 3V/8MHz 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>

Adafruit Trinket 5V/16MHz

Contents

- Adafruit Trinket 5V/16MHz
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform 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.

<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>8KB</td>
</tr>
<tr>
<td>RAM</td>
<td>512B</td>
</tr>
<tr>
<td>Vendor</td>
<td>Adafruit</td>
</tr>
</tbody>
</table>

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
```

(continues on next page)
board_build.mcu = attiny85
; change MCU frequency
board_build.f_cpu = 16000000L

Debugging

PIO Unified Debugger currently does not support Adafruit Trinket 5V/16MHz 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>

Alorium Hinj

Contents

- Alorium Hinj
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform 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.

<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):
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.

```plaintext
[env:alorium_hinj]
platform = atmelavr
board = alorium_hinj

; change microcontroller
board_build.mcu = atmega328p

; change MCU frequency
board_build.f_cpu = 16000000L
```

### Debugging

PIO Unified Debugger currently does not support Alorium Hinj 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>

### Alorium Sno

#### Contents

- Alorium Sno
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

#### Hardware

Platform 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.
### 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>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):

```
[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.

```
[env:alorium_sno]
platform = atmelavr
board = alorium_sno

; change microcontroller
board_build.mcu = atmega328p

; change MCU frequency
board_build.f_cpu = 16000000L
```

### Debugging

*PIO Unified Debugger* currently does not support Alorium Sno 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>

### Alorium XLR8

#### Contents

- *Alorium XLR8*
  - Hardware
  - Configuration
Hardware

Platform *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.

<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*):

```ini
[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 `alarium_xlr8.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:alorium_xlr8]
platform = atmelavr
board = alorium_xlr8

; change microcontroller
board_build.mcu = atmega328p

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

*PIO Unified Debugger* currently does not support Alorium XLR8 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>
Altair

Contents

- Altair
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform *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.

<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

*PIO Unified Debugger* 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
  - Debugging
  - Frameworks

Hardware

Platform **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.

<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):

```
[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.

```
[env:miniwireless]
platform = atmelavr
board = miniwireless

; change microcontroller
```

(continues on next page)
board_build.mcu = atmega328p

; change MCU frequency
board_build.f_cpu = 16000000L

Debugging

PIO Unified Debugger currently does not support Anarduino MiniWireless 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>

Arduboy

Contents

- Arduboy
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform 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.

<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):
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.

```json
[env:arduboy]
platform = atmelavr
board = arduboy

; change microcontroller
board_build.mcu = atmega32u4

; change MCU frequency
board_build.f_cpu = 16000000L
```

**Debugging**

PIO Unified Debugger currently does not support Arduboy 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>

**Arduboy DevKit**

**Contents**

- Arduboy DevKit
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

**Hardware**

Platform **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.
<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**

*PIO Unified Debugger* currently does not support Arduboy DevKit 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>
</tbody>
</table>

**Arduino BT ATmega168**

**Contents**

- Arduino BT ATmega168
  - Hardware
  - Configuration
Hardware

Platform *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.

<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
board_build.mcu = atmega168

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

*PIO Unified Debugger* currently does not support Arduino BT ATmega168 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 BT ATmega328

## Contents

- Arduino BT ATmega328
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

### Hardware

Platform *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.

<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 `btatmega328` ID for `board` option in *`platformio.ini` (Project Configuration File)*:

```ini
[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.

```ini
[env:btatmega328]
platform = atmelavr
board = btatmega328

; change microcontroller
board_build.mcu = atmega328p

; change MCU frequency
board_build.f_cpu = 16000000L
```

### Debugging

*PIO Unified Debugger* currently does not support Arduino BT ATmega328 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 Duemilanove or Diecimila ATmega168

Contents

- Arduino Duemilanove or Diecimila ATmega168
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform 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.

Platform 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.

<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):

```
[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.

```
[env:diecimilaatmega168]
platform = atmelavr
board = diecimilaatmega168
; change microcontroller
```

(continues on next page)
board_build.mcu = atmega168

; change MCU frequency
board_build.f_cpu = 16000000L

Debugging

PIO Unified Debugger currently does not support Arduino Duemilanove or Diecimila ATmega168 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 Duemilanove or Diecimila ATmega328

Contents

- Arduino Duemilanove or Diecimila ATmega328
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform 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.

<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):
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.

```plaintext
[env:diecimilaatmega328]
platform = atmelavr
board = diecimilaatmega328

; change microcontroller
board_build.mcu = atmega328p

; change MCU frequency
board_build.f_cpu = 16000000L
```

### Debugging

*PIO Unified Debugger* currently does not support Arduino Duemilanove or Diecimila ATmega328 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 Esplora

#### Contents

- Arduino Esplora
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

### Hardware

Platform *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.
### Configuration

Please use `esplora` ID for `board` option in "platformio.ini" *(Project Configuration File):*

```ini
[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.

```ini
[env:esplora]
platform = atmelavr
board = esplora

; change microcontroller
board_build.mcu = atmega32u4

; change MCU frequency
board_build.f_cpu = 16000000L
```

### Debugging

*PIO Unified Debugger* currently does not support Arduino Esplora 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 Ethernet

- Arduino Ethernet
  - Hardware
  - Configuration
Hardware

Platform *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 industries 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 *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

*PIO Unified Debugger* currently does not support Arduino Ethernet 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 Fio

Hardware

Platform 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.

<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):

```ini
[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.

```ini
[env:fio]
platform = atmelavr
board = fio

; change microcontroller
board_build.mcu = atmega328p

; change MCU frequency
board_build.f_cpu = 8000000L
```

Debugging

PIO Unified Debugger currently does not support Arduino Fio 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 Industrial 101

Contents

- Arduino Industrial 101
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform **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.

<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 `chiwawa` ID for `board` option in "platformio.ini" (Project Configuration File):

```
[env:chiwawa]
platform = atmelavr
board = chiwawa
```

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.`

```
[env:chiwawa]
platform = atmelavr
board = chiwawa
; change microcontroller
```

(continues on next page)
```plaintext
board_build.mcu = atmega32u4

; change MCU frequency
board_build.f_cpu = 16000000L
```

### Debugging

*PIO Unified Debugger* currently does not support Arduino Industrial 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>

### Arduino Leonardo

### Contents

- Arduino Leonardo
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

### Hardware

Platform *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.

<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)*:
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.

```yaml
[env:leonardo]
platform = atmelavr
board = leonardo

; change microcontroller
board_build.mcu = atmega32u4

; change MCU frequency
board_build.f_cpu = 16000000L
```

### Debugging

*PIO Unified Debugger* currently does not support Arduino Leonardo 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 Leonardo ETH

#### Contents

- Arduino Leonardo ETH
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

#### Hardware

Platform *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.
### Configuration

Please use `leonardoeth` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[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.

```
[env:leonardoeth]
platform = atmelavr
board = leonardoeth

; change microcontroller
board_build.mcu = atmega32u4

; change MCU frequency
board_build.f_cpu = 16000000L
```

### Debugging

PIO Unified Debugger currently does not support Arduino Leonardo ETH 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 LilyPad ATmega168

#### Contents

- Arduino LilyPad ATmega168
  - Hardware
  - Configuration
Hardware

Platform *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 industries 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)*:

```ini
[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.

```ini
[env:lilypadatmega168]
platform = atmelavr
board = lilypadatmega168

; change microcontroller
board_build.mcu = atmega168

; change MCU frequency
board_build.f_cpu = 8000000L
```

Debugging

*PIO Unified Debugger* currently does not support Arduino LilyPad ATmega168 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 LilyPad ATmega328**

### Contents

- Arduino LilyPad ATmega328
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

### Hardware

Platform *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.

<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 `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

*PIO Unified Debugger* currently does not support Arduino LilyPad ATmega328 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 LilyPad USB

Contents

- Arduino LilyPad USB
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform *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.

<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

*PIO Unified Debugger* currently does not support Arduino LilyPad USB 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 Mega ADK

#### Contents

- Arduino Mega ADK
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

#### Hardware

Platform *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.

<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 `megaADK` ID for *board* option in “platformio.ini” *(Project Configuration File)*:
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.

```json
[env:megaADK]
platform = atmelavr
board = megaADK

; change microcontroller
board_build.mcu = atmega2560

; change MCU frequency
board_build.f_cpu = 16000000L
```

**Debugging**

*PIO Unified Debugger* currently does not support Arduino Mega ADK 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 Mega or Mega 2560 ATmega1280**

**Contents**

- Arduino Mega or Mega 2560 ATmega1280
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

**Hardware**

Platform *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.
Microcontroller | ATMEGA1280
--- | ---
Frequency | 16MHz
Flash | 124KB
RAM | 8KB
Vendor | Arduino

**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
board_build.mcu = atmega1280

; change MCU frequency
board_build.f_cpu = 16000000L
```

**Debugging**

*PIO Unified Debugger* currently does not support Arduino Mega or Mega 2560 ATmega1280 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 Mega or Mega 2560 ATmega2560 (Mega 2560)**
Hardware

Platform **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.

<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

**PIO Unified Debugger** currently does not support Arduino Mega or Mega 2560 ATmega2560 (Mega 2560) 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>Simba</td>
<td>Simba is an RTOS and build framework. It 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- 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.

<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

; change microcontroller
board_build.mcu = atmega32u4

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

*PIO Unified Debugger* currently does not support Arduino Micro 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 Mini ATmega168

Contents

- Arduino Mini ATmega168
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform **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.

<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):

```ini
[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.

```ini
[env:miniatmega168]
platform = atmelavr
board = miniatmega168
```

; change microcontroller

(continues on next page)
board_build.mcu = atmega168

; change MCU frequency
board_build.f_cpu = 16000000L

### Debugging

*PIO Unified Debugger* currently does not support Arduino Mini ATmega168 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 Mini ATmega328

#### Contents

- *Arduino Mini ATmega328*
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

#### Hardware

Platform *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.

<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)*:
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.

```ini
[env:miniatmega328]
platform = atmelavr
board = miniatmega328

; change microcontroller
board_build.mcu = atmega328p

; change MCU frequency
board_build.f_cpu = 16000000L
```

### Debugging

*PIO Unified Debugger* currently does not support Arduino Mini ATmega328 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 NG or older ATmega168

#### Contents

- Arduino NG or older ATmega168
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

#### Hardware

Platform *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.
<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):

```
[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.

```
[env:atmegangatmega168]
platform = atmelavr
board = atmegangatmega168

; change microcontroller
board_build.mcu = atmega168

; change MCU frequency
board_build.f_cpu = 16000000L
```

### Debugging

PIO Unified Debugger currently does not support Arduino NG or older ATmega168 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 NG or older ATmega8

#### Contents

- Arduino NG or older ATmega8
  - Hardware
  - Configuration
Hardware

Platform *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.

<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)*:

```
[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.

```
[env:atmegangatmega8]
platform = atmelavr
board = atmegangatmega8

; change microcontroller
board_build.mcu = atmega8

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

*PIO Unified Debugger* currently does not support Arduino NG or older ATmega8 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 ATmega168

Contents

- Arduino Nano ATmega168
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform 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.

<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 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

PIO Unified Debugger currently does not support Arduino Nano ATmega168 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 ATmega328

Contents

- Arduino Nano ATmega328
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform 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.

<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):

```ini
[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.

```ini
[env:nanoatmega328]
platform = atmelavr
board = nanoatmega328

; change microcontroller
```

(continues on next page)
board_build.mcu = atmega328p
; change MCU frequency
board_build.f_cpu = 16000000L

Debugging

PIO Unified Debugger currently does not support Arduino Nano ATmega328 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>Simba</td>
<td>Simba is an RTOS and build framework. It aims to make embedded programming easy and portable.</td>
</tr>
</tbody>
</table>

Arduino Nano ATmega328 (New Bootloader)

Hardware

Platform 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.

<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):
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.

```json
[env:nanoatmega328new]
platform = atmelavr
board = nanoatmega328new

; change microcontroller
board_build.mcu = atmega328p

; change MCU frequency
board_build.f_cpu = 16000000L
```

### Debugging

*PIO Unified Debugger* currently does not support Arduino Nano ATmega328 (New Bootloader) 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 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- and 32-bit MCUs deliver a unique combination of performance, power efficiency and design flexibility. Optimized to speed 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 `pro8MHzatmega168` ID for `board` option in "platformio.ini" (Project Configuration File):

```
[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.

```
[env:pro8MHzatmega168]
platform = atmelavr
board = pro8MHzatmega168

; change microcontroller
board_build.mcu = atmega168

; change MCU frequency
board_build.f_cpu = 8000000L
```

### Debugging

PIO Unified Debugger currently does not support Arduino Pro or Pro Mini ATmega168 (3.3V, 8 MHz) 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>
</tbody>
</table>

### Arduino Pro or Pro Mini ATmega168 (5V, 16 MHz)

#### Contents

- **Arduino Pro or Pro Mini ATmega168 (5V, 16 MHz)**
  - Hardware
  - Configuration
Hardware

Platform 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.

<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

PIO Unified Debugger currently does not support Arduino Pro or Pro Mini ATmega168 (5V, 16 MHz) 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 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- 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.

<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
board_build.mcu = atmega328p

; change MCU frequency
board_build.f_cpu = 8000000L
```

Debugging

**PIO Unified Debugger** currently does not support Arduino Pro or Pro Mini ATmega328 (3.3V, 8 MHz) 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 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- 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.

<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):

```
[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.

```
[env:pro16MHzatmega328]
platform = atmelavr
board = pro16MHzatmega328
; change microcontroller
```

(continues on next page)
board_build.mcu = atmega328

; change MCU frequency
board_build.f_cpu = 16000000L

### Debugging

PIO Unified Debugger currently does not support Arduino Pro or Pro Mini ATmega328 (5V, 16 MHz) 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 Robot Control

#### Contents

- Arduino Robot Control
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

#### Hardware

Platform 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.

<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):
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.

```python
[env:robotControl]
platform = atmelavr
board = robotControl

; change microcontroller
board_build.mcu = atmega32u4

; change MCU frequency
board_build.f_cpu = 16000000L
```

### Debugging

**PIO Unified Debugger** currently does not support Arduino Robot Control 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 Robot Motor

#### Contents

- **Arduino Robot Motor**
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

#### Hardware

Platform *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.
### Configuration

Please use `robotMotor` ID for `board` option in "platformio.ini" (Project Configuration File):

```plaintext
[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.

```plaintext
[env:robotMotor]
platform = atmelavr
board = robotMotor

; change microcontroller
board_build.mcu = atmega32u4

; change MCU frequency
board_build.f_cpu = 16000000L
```

### Debugging

PIO Unified Debugger currently does not support Arduino Robot Motor 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

- Arduino Uno
  - Hardware
  - Configuration
Hardware

Platform *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.

<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

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

*PIO Unified Debugger* currently does not support Arduino Uno 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>Simba</td>
<td>Simba is an RTOS and build framework. It 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- 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.

<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

*PIO Unified Debugger* currently does not support Arduino Yun 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 Yun Mini

Contents

- Arduino Yun Mini
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform **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.

<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)
PlatformIO Documentation, Release 4.0.1b3

(continued from previous page)

```plaintext
platformio.ini

board_build.mcu = atmega32u4

; change MCU frequency
board_build.f_cpu = 16000000L
```

### Debugging

**PIO Unified Debugger** currently does not support Arduino Yun 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>

#### Atmel AT90PWM216

### Hardware

Platform **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.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>AT90PWM216</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 `at90pwm216` ID for `board` option in “[platformio.ini](https://project_configuration_file)” (Project Configuration File):
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.

```plaintext
[env:at90pwm216]
platform = atmelavr
board = at90pwm216

; change microcontroller
board_build.mcu = at90pwm216

; change MCU frequency
board_build.f_cpu = 16000000L
```

**Debugging**

PIO Unified Debugger currently does not support Atmel AT90PWM216 board.

**Atmel AT90PWM316**

### Contents

- **Atmel AT90PWM316**
  - Hardware
  - Configuration
  - Debugging

**Hardware**

Platform **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.

<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):
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.

```python
[env:at90pwm316]
platform = atmelavr
board = at90pwm316

; change microcontroller
board_build.mcu = at90pwm316

; change MCU frequency
board_build.f_cpu = 16000000L
```

**Debugging**

*PIO Unified Debugger* 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- 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.

<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>BQ</td>
</tr>
</tbody>
</table>
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

PIO Unified Debugger currently does not support BQ ZUM BT-328 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>

BitWizard Raspduino

Contents

- BitWizard Raspduino
  - Hardware
  - Configuration
  - Debugging
  - Frameworks
Hardware

Platform *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.

<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 `raspduino` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:raspduino]
platform = atmelavr
board = raspduino
```

You can override default BitWizard Raspduino settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `raspduino.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:raspduino]
platform = atmelavr
board = raspduino

; change microcontroller
board_build.mcu = atmega328p

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

*PIO Unified Debugger* currently does not support BitWizard Raspduino 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>

Controllino Maxi
Contents

- Controllino Maxi
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform **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.

<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

**PIO Unified Debugger** currently does not support Controllino Maxi 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>

Controllino Maxi Automation

Contents

- Controllino Maxi Automation
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform 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.

<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_maxiAutomation ID for board option in "platformio.ini" (Project Configuration File):

```ini
[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.

```ini
[env:controllino_maxi_automation]
platform = atmelavr
board = controllino_maxi_automation
```

(continues on next page)
### Debugging

*PIO Unified Debugger* currently does not support Controllino Maxi Automation 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>

### Controllino Mega

#### Contents

- *Controllino Mega*
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

### Hardware

Platform *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.

<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)*:
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.

```yaml
[env:controllino_mega]
platform = atmelavr
board = controllino_mega

; change microcontroller
board_build.mcu = atmega2560

; change MCU frequency
board_build.f_cpu = 16000000L
```

### Debugging

`PIO Unified Debugger` currently does not support Controllino Mega 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>

### Controllino Mini

#### Contents

- Controllino Mini
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

### Hardware

Platform `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.
### 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>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
board_build.mcu = atmega328p

; change MCU frequency
board_build.f_cpu = 16000000L
```

### Debugging

`PIO Unified Debugger` currently does not support Controllino 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>

### Digispark Pro

#### Contents

- Digispark Pro
  - Hardware
  - Configuration
Hardware

Platform *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.

<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

*PIO Unified Debugger* 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
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform 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.

<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):

```
[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.

```
[env:digispark-pro64]
platform = atmelavr
board = digispark-pro64

; change microcontroller
board_build.mcu = attiny167

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

PIO Unified Debugger 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- 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 industries 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):

```
[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.

```
[env:digispark-pro32]
platform = atmelavr
board = digispark-pro32
```

; change microcontroller

(continues on next page)
board_build.mcu = attiny167
; change MCU frequency
board_build.f_cpu = 16000000L

Debugging

*PIO Unified Debugger* 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- 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.

<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*):
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.

```lua
[env:digispark-tiny]
platform = atmelavr
board = digispark-tiny

; change microcontroller
board_build.mcu = attiny85

; change MCU frequency
board_build.f_cpu = 16500000L
```

**Debugging**

*PIO Unified Debugger* currently does not support Digispark USB 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>

**Dwenguino**

**Contents**

- **Dwenguino**
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

**Hardware**

Platform *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.
### Configuration

Please use `dwenguino` ID for `board` option in “`platformio.ini` (Project Configuration File):

```ini
[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.

```ini
[env:dwenguino]
platform = atmelavr
board = dwenguino

; change microcontroller
board_build.mcu = at90usb646

; change MCU frequency
board_build.f_cpu = 16000000L
```

### Debugging

`PIO Unified Debugger` 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- 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 industries 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)*:

```
[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.

```
[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

*PIO Unified Debugger* 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- 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.

<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):

```
[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.

```
[env:engduinov3]
platform = atmelavr
board = engduinov3

; change microcontroller
board_build.mcu = atmega32u4

; change MCU frequency
board_build.f_cpu = 8000000L
```

Debugging

PIO Unified Debugger currently does not support Engduino 3 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>

EnviroDIY Mayfly

Contents

- EnviroDIY Mayfly
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform 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.

<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):

```ini
[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.

```ini
[env:mayfly]
platform = atmelavr
board = mayfly
; change microcontroller
```

(continues on next page)
board_build.mcu = atmega1284p
; change MCU frequency
board_build.f_cpu = 8000000L

Debugging

*PIO Unified Debugger* currently does not support EnviroDIY Mayfly 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>

FYSETC F6 V1.3

**Contents**

- FYSETC F6 V1.3
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform *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.

<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)*:
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.

```yaml
[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

*PIO Unified Debugger* currently does not support FYSETC F6 V1.3 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 ATtiny13

#### Contents

- *Generic ATtiny13*
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

#### Hardware

Platform *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.
PlatformIO Documentation, Release 4.0.1b3

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATTINY13</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>1MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>1KB</td>
</tr>
<tr>
<td>RAM</td>
<td>64B</td>
</tr>
<tr>
<td>Vendor</td>
<td>Atmel</td>
</tr>
</tbody>
</table>

**Configuration**

Please use `attiny13` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:attiny13]
platform = atmelavr
board = attiny13
```

You can override default Generic 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.

```
[env:attiny13]
platform = atmelavr
board = attiny13

; change microcontroller
board_build.mcu = attiny13

; change MCU frequency
board_build.f_cpu = 1200000L
```

**Debugging**

PIO Unified Debugger currently does not support Generic ATtiny13 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 ATtiny13A**

**Contents**

- Generic ATtiny13A
  - Hardware
  - Configuration
Hardware

Platform **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.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATTINY13A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>1MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>1KB</td>
</tr>
<tr>
<td>RAM</td>
<td>64B</td>
</tr>
<tr>
<td>Vendor</td>
<td>Atmel</td>
</tr>
</tbody>
</table>

Configuration

Please use `attiny13a` ID for `board` option in "platformio.ini" (Project Configuration File):

```ini
[env:attiny13a]
platform = atmelavr
board = attiny13a
```

You can override default Generic 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.

```ini
[env:attiny13a]
platform = atmelavr
board = attiny13a

; change microcontroller
board_build.mcu = attiny13a

; change MCU frequency
board_build.f_cpu = 1200000L
```

Debugging

*PIO Unified Debugger* currently does not support Generic ATtiny13A 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 ATtiny1634

Contents

- Generic ATtiny1634
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform **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.

<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

**PIO Unified Debugger** 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

Hardware

Platform 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.

<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):

```
[env:attiny167]
platform = atmelavr
board = attiny167
```

You can override default Generic ATtiny17 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.

```
[env:attiny167]
platform = atmelavr
board = attiny167

; change microcontroller
```

(continues on next page)
board_build.mcu = attiny167
;
change MCU frequency
board_build.f_cpu = 8000000L

Debugging

PIO Unified Debugger 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

Contents

- Generic ATtiny2313
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform 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.

<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):
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.

```yaml
[env:attiny2313]
platform = atmelavr
board = attiny2313

; change microcontroller
board_build.mcu = attiny2313

; change MCU frequency
board_build.f_cpu = 8000000L
```

### Debugging

PIO Unified Debugger currently does not support Generic ATtiny2313 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 ATtiny24

#### Contents

- Generic ATtiny24
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

#### Hardware

Platform *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.
### Microcontroller

<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

**PIO Unified Debugger** currently does not support Generic ATtiny24 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 ATtiny25

#### Contents

- *Generic ATtiny25*
  - Hardware
  - Configuration
Hardware

Platform *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.

<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):

```ini
[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.

```ini
[env:attiny25]
platform = atmelavr
board = attiny25

; change microcontroller
board_build.mcu = attiny25

; change MCU frequency
board_build.f_cpu = 8000000L
```

Debugging

*PIO Unified Debugger* currently does not support Generic ATtiny25 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 ATtiny261

Contents

- Generic ATtiny261
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform **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.

<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

**PIO Unified Debugger** 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- 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.

<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
```

(continues on next page)
board_build.mcu = \texttt{attiny4313} \\
; change MCU frequency 
board_build.f_cpu = \texttt{8000000L}

\section*{Debugging}

\textit{PIO Unified Debugger} currently does not support Generic ATtiny4313 board.

\section*{Frameworks}

\begin{tabular}{|c|c|}
\hline
\textbf{Name} & \textbf{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}

\section*{Generic ATtiny43U}

\subsection*{Contents}

\begin{itemize}
\item \textit{Generic ATtiny43U}
  \begin{itemize}
  \item \textit{Hardware}
  \item \textit{Configuration}
  \item \textit{Debugging}
  \item \textit{Frameworks}
  \end{itemize}
\end{itemize}

\subsection*{Hardware}

Platform \textit{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.

\begin{tabular}{|c|c|}
\hline
\textbf{Microcontroller} & \texttt{ATTINY43U} \\
\hline
\textbf{Frequency} & 8MHz \\
\hline
\textbf{Flash} & 4KB \\
\hline
\textbf{RAM} & 256B \\
\hline
\textbf{Vendor} & Atmel \\
\hline
\end{tabular}

\subsection*{Configuration}

Please use \texttt{attiny43} ID for \textit{board} option in \texttt{“platformio.ini”} (\textit{Project Configuration File}):
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.

```yaml
platform = atmelavr
board = attiny43

; change microcontroller
board_build.mcu = attiny43u

; change MCU frequency
board_build.f_cpu = 8000000L
```

### Debugging

*PIO Unified Debugger* 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- 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.
### Configuration

Please use `attiny44` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:attiny44]
platform = atmelavr
board = attiny44
```

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.

```
[env:attiny44]
platform = atmelavr
board = attiny44

; change microcontroller
board_build.mcu = attiny44

; change MCU frequency
board_build.f_cpu = 8000000L
```

### Debugging

*PIO Unified Debugger* currently does not support Generic ATtiny44 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 ATtiny441

**Contents**

- *Generic ATtiny441*
  - Hardware
  - Configuration
Hardware

Platform *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.

<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):

```
[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.

```
[env:attiny441]
platform = atmelavr
board = attiny441

; change microcontroller
board_build.mcu = attiny441

; change MCU frequency
board_build.f_cpu = 8000000L
```

Debugging

*PIO Unified Debugger* 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>
PlatformIO Documentation, Release 4.0.1b3

Generic ATtiny45

Contents

• Generic ATtiny45
  – Hardware
  – Configuration
  – Debugging
  – Frameworks

Hardware

Platform 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.

<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

PIO Unified Debugger currently does not support Generic ATtiny45 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 ATtiny461

Contents

- Generic ATtiny461
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform 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.

<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
```

(continues on next page)
board_build.mcu = attiny461
;

Debugging

PIO Unified Debugger 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- 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.

<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):
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.

```yaml
[env:attiny48]
platform = atmelavr
board = attiny48

; change microcontroller
board_build.mcu = attiny48

; change MCU frequency
board_build.f_cpu = 8000000L
```

## Debugging

*PIO Unified Debugger* 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

### Contents

- `Generic ATtiny828`
  - `Hardware`
  - `Configuration`
  - `Debugging`
  - `Frameworks`

### Hardware

Platform *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.
Microcontroller | ATTINY828  
---|---  
Frequency | 8MHz  
Flash | 8KB  
RAM | 512B  
Vendor | Atmel  

### Configuration

Please use `attiny828` ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[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.

```ini
[env:attiny828]
platform = atmelavr
board = attiny828

; change microcontroller
board_build.mcu = attiny828

; change MCU frequency
board_build.f_cpu = 8000000L
```

### Debugging

`PIO Unified Debugger` 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
Hardware

Platform **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.

<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):

```ini
[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.

```ini
[env:attiny84]
platform = atmelavr
board = attiny84

; change microcontroller
board_build.mcu = attiny84

; change MCU frequency
board_build.f_cpu = 8000000L
```

Debugging

**PIO Unified Debugger** currently does not support Generic ATtiny84 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 ATtiny841

## Hardware

Platform **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.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATTINY841</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 `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**

*PIO Unified Debugger* 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
  - Debugging
  - Frameworks

Hardware

Platform 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.

<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
```

(continues on next page)
board_build.mcu = attiny85

; change MCU frequency
board_build.f_cpu = 8000000L

Debugging

PIO Unified Debugger currently does not support Generic ATtiny85 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 ATtiny861

Hardware

Platform 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.

<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):
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.

```yaml
[env:attiny861]
platform = atmelavr
board = attiny861

; change microcontroller
board_build.mcu = attiny861

; change MCU frequency
board_build.f_cpu = 8000000L
```

### Debugging

*PIO Unified Debugger* 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- 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.
Microcontroller | ATTINY87  
--- | ---  
Frequency | 8MHz  
Flash | 8KB  
RAM | 512B  
Vendor | Atmel

### Configuration

Please use `attiny87` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[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.

```
[env:attiny87]
platform = atmelavr
board = attiny87

; change microcontroller
board_build.mcu = attiny87

; change MCU frequency
board_build.f_cpu = 8000000L
```

### Debugging

PIO Unified Debugger 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
Hardware

Platform **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.

<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**

**PIO Unified Debugger** 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

Contents

- LightBlue Bean
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform 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.

<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):

```
[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.

```
[env:lightblue-bean]
platform = atmelavr
board = lightblue-bean

; change microcontroller
board_build.mcu = atmega328p

; change MCU frequency
board_build.f_cpu = 8000000L
```

Debugging

PIO Unified Debugger currently does not support LightBlue Bean 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+

Contents

- LightBlue Bean+
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform *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.

<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
```

(continues on next page)
board_build.mcu = atmega328p
; change MCU frequency
board_build.f_cpu = 16000000L

Debugging

PIO Unified Debugger currently does not support LightBlue Bean+ 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>

LightUp

Contents

- LightUp
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform 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.

<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):
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**

*PIO Unified Debugger* currently does not support LightUp 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>

**Linino One**

**Contents**

- *Linino One*
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

**Hardware**

Platform *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.
### 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

PIO Unified Debugger currently does not support Linino 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>
</tbody>
</table>

**LinkIt Smart 7688 Duo**

**Contents**

- LinkIt Smart 7688 Duo
  - Hardware
  - Configuration
  - Debugging
Hardware

Platform **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.

<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

**PIO Unified Debugger** currently does not support LinkIt Smart 7688 Duo 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>
LoRa32u4II (868-915MHz)

Contents

- LoRa32u4II (868-915MHz)
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform **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.

<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):

```
[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.

```
[env:lora32u4II]
platform = atmelavr
board = lora32u4II

; change microcontroller
board_build.mcu = atmega32u4

; change MCU frequency
board_build.f_cpu = 8000000L
```

Debugging

**PIO Unified Debugger** currently does not support LoRa32u4II (868-915MHz) 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>

LowPowerLab MightyHat

Contents

- LowPowerLab MightyHat
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform 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.

<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):

```ini
[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.

```ini
[env:mightyhat]
platform = atmelavr
board = mightyhat

; change microcontroller
```

(continues on next page)
board_build.mcu = atmega328

; change MCU frequency
board_build.f_cpu = 16000000L

Debugging

PIO Unified Debugger currently does not support LowPowerLab MightyHat 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>

LowPowerLab Moteino

Contents

- LowPowerLab Moteino
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform 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.

<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):
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.

```python
[env:moteino]
platform = atmelavr
board = moteino

; change microcontroller
board_build.mcu = atmega328p

; change MCU frequency
board_build.f_cpu = 16000000L
```

### Debugging

PIO Unified Debugger currently does not support LowPowerLab Moteino 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>

### LowPowerLab Moteino (8Mhz)

#### Hardware

Platform **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.
### Configuration

Please use moteino8mhz ID for board option in “platformio.ini” (Project Configuration File):

```plaintext
[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.

```plaintext
[env:moteino8mhz]
platform = atmelavr
board = moteino8mhz

; change microcontroller
board_build.mcu = atmega328p

; change MCU frequency
board_build.f_cpu = 8000000L
```

### Debugging

PIO Unified Debugger currently does not support LowPowerLab Moteino (8Mhz) 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>

### LowPowerLab MoteinoMEGA

#### Contents

- LowPowerLab MoteinoMEGA
  - Hardware
  - Configuration
Hardware

Platform 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.

<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):

```ini
[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.

```ini
[env:moteinomega]
platform = atmelavr
board = moteinomega

; change microcontroller
board_build.mcu = atmega1284p

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

PIO Unified Debugger currently does not support LowPowerLab MoteinoMEGA 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>
Microduino Core (Atmega168PA@16M,5V)

Contents

- Microduino Core (Atmega168PA@16M,5V)
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform **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.

<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>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 `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

**PIO Unified Debugger** currently does not support Microduino Core (Atmega168PA@16M,5V) 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>

Microduino Core (Atmega168PA@8M,3.3V)

Contents

- Microduino Core (Atmega168PA@8M,3.3V)
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform **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.

<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):

```
[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`.

```
[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

PIO Unified Debugger currently does not support Microduino Core (Atmega168PA@8M,3.3V) 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>

Microduino Core (Atmega328P@16M,5V)

Contents

- Microduino Core (Atmega328P@16M,5V)
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform 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.

<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>Microduino</td>
</tr>
</tbody>
</table>

Configuration

Please use 328p16m ID for board option in “platformio.ini” (Project Configuration File):
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`.

```yaml
platform = atmelavr
board = 328p16m

; change microcontroller
board_build.mcu = atmega328p

; change MCU frequency
board_build.f_cpu = 16000000L
```

### Debugging

PIO Unified Debugger currently does not support Microduino Core (Atmega328P@16M,5V) 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>

### Microduino Core (Atmega328P@8M,3.3V)

#### Contents

- Microduino Core (Atmega328P@8M,3.3V)
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

#### Hardware

Platform 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.
## 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
board_build.mcu = atmega328p

; change MCU frequency
board_build.f_cpu = 8000000L
```

## Debugging

PIO Unified Debugger currently does not support Microduino Core (Atmega328P@8M,3.3V) 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>

### Microduino Core USB (ATmega32U4@16M,5V)

- Microduino Core USB (ATmega32U4@16M,5V)
  - Hardware
  - Configuration
Hardware

Platform *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.

<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):

```ini
[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.

```ini
[env:32u416m]
platform = atmelavr
board = 32u416m

; change microcontroller
board_build.mcu = atmega32u4

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

*PIO Unified Debugger* currently does not support Microduino Core USB (ATmega32U4@16M,5V) 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>
Microduino Core+ (ATmega1284P@16M,5V)

Contents

- Microduino Core+ (ATmega1284P@16M,5V)
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform 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.

<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
board_build.mcu = atmega1284p

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

PIO Unified Debugger currently does not support Microduino Core+ (ATmega1284P@16M,5V) 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>

Microduino Core+ (ATmega1284P@8M,3.3V)

Contents

- Microduino Core+ (ATmega1284P@8M,3.3V)
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform 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.

<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
```

(continues on next page)
board_build.mcu = atmega1284p

; change MCU frequency
board_build.f_cpu = 8000000L

Debugging

PIO Unified Debugger currently does not support Microduino Core+ (ATmega1284P@8M,3.3V) 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>

Microduino Core+ (Atmega644PA@16M,5V)

Contents

- Microduino Core+ (Atmega644PA@16M,5V)
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform 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.

<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 644p16m ID for board option in “platformio.ini” (Project Configuration File):
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.

```systemshell
[env:644pa16m]
platform = atmelavr
board = 644pa16m

; change microcontroller
board_build.mcu = atmega644p

; change MCU frequency
board_build.f_cpu = 16000000L
```

### Debugging

PIO Unified Debugger currently does not support Microduino Core+ (Atmega644PA@16M,5V) 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>

**Microduino Core+ (Atmega644PA@8M,3.3V)**

### Contents

- **Microduino Core+ (Atmega644PA@8M,3.3V)**
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

### Hardware

Platform **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.
**PlatformIO Documentation, Release 4.0.1b3**

<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>Microduino</td>
</tr>
</tbody>
</table>

**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**

*PIO Unified Debugger* currently does not support Microduino Core+ (`Atmega644PA@8M,3.3V`) 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>

**OpenEnergyMonitor emonPi**

**Contents**

- *OpenEnergyMonitor emonPi*
  - Hardware
  - Configuration
Hardware

Platform **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.

<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):

```ini
[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.

```ini
[env:emonpi]
platform = atmelavr
board = emonpi

; change microcontroller
board_build.mcu = atmega328p

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

**PIO Unified Debugger** currently does not support OpenEnergyMonitor emonPi 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>
PanStamp AVR

Contents

- PanStamp AVR
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform 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.

<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):

```ini
[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.

```ini
[env:panStampAVR]
platform = atmelavr
board = panStampAVR
```

; change microcontroller
board_build.mcu = atmega328p

; change MCU frequency
board_build.f_cpu = 8000000L

Debugging

PIO Unified Debugger currently does not support PanStamp AVR 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>

Pinoccio Scout

Contents

- Pinoccio Scout
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform 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.

<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
```

(continues on next page)
board_build.mcu = atmega256rfr2

; change MCU frequency
board_build.f_cpu = 16000000L

Debugging

PIO Unified Debugger 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

Hardware

Platform 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.

<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):
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.

```yaml
[env:a-star32U4]
platform = atmelavr
board = a-star32U4

; change microcontroller
board_build.mcu = atmega32u4

; change MCU frequency
board_build.f_cpu = 16000000L
```

## Debugging

*PIO Unified Debugger* currently does not support Pololu A-Star 32U4 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>

## Quirkbot

### Contents

- *Quirkbot*
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

### Hardware

Platform *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.
PlatformIO Documentation, Release 4.0.1b3

<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**

PIO Unified Debugger currently does not support Quirkbot 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>

**RedBearLab Blend**

**Contents**

- RedBearLab Blend
  - Hardware
  - Configuration
  - Debugging
Hardware

Platform *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.

<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):

```ini
[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.

```ini
[env:blend]
platform = atmelavr
board = blend

; change microcontroller
board_build.mcu = atmega32u4

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

*PIO Unified Debugger* currently does not support RedBearLab Blend 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>
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- 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.

<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):

```ini
[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.

```ini
[env:blendmicro16]
platform = atmelavr
board = blendmicro16

; change microcontroller
board_build.mcu = atmega32u4

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

**PIO Unified Debugger** currently does not support RedBearLab Blend Micro 3.3V/16MHz (overclock) 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>

RedBearLab Blend Micro 3.3V/8MHz

Contents

- RedBearLab Blend Micro 3.3V/8MHz
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform *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.

<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):

```ini
[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.

```ini
[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

PIO Unified Debugger currently does not support RedBearLab Blend Micro 3.3V/8MHz 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>

RepRap RAMBo

Contents

- RepRap RAMBo
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform 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.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
<th>Vendor</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATMEGA2560</td>
<td>16MHz</td>
<td>252KB</td>
<td>8KB</td>
<td>RepRap</td>
</tr>
</tbody>
</table>

Configuration

Please use reprap_rambo ID for board option in “platformio.ini” (Project Configuration File):
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.

```yaml
[env:reprap_rambo]
platform = atmelavr
board = reprap_rambo
```

```yaml
; change microcontroller
board_build.mcu = atmega2560

; change MCU frequency
board_build.f_cpu = 16000000L
```

## Debugging

*PIO Unified Debugger* currently does not support RepRap RAMBo 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>

## SODAQ GaLoRa

### Contents

- SODAQ GaLoRa
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

### Hardware

Platform *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.
### 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
board_build.mcu = atmega1284p

; change MCU frequency
board_build.f_cpu = 8000000L
```

### Debugging

*PIO Unified Debugger* currently does not support SODAQ GaLoRa 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>

### SODAQ Mbili

#### Contents

- *SODAQ Mbili*
  - Hardware
  - Configuration
Hardware

Platform **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.

<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_mbili` ID for `board` option in “platformio.ini” (*Project Configuration File*):

```ini
[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.

```ini
[env:sodaq_mbili]
platform = atmelavr
board = sodaq_mbili

; change microcontroller
board_build.mcu = atmega1284p

; change MCU frequency
board_build.f_cpu = 8000000L
```

**Debugging**

*PIO Unified Debugger* currently does not support SODAQ Mbili 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>
SODAQ Moja

Contents

- SODAQ Moja
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform 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.

<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
board_build.mcu = atmega328p

; change MCU frequency
board_build.f_cpu = 8000000L
```

Debugging

PIO Unified Debugger currently does not support SODAQ Moja 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>

SODAQ Ndogo

Contents

- SODAQ Ndogo
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform 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 indusry'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):

```ini
[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.

```ini
[env:sodaq_ndogo]
platform = atmelavr
board = sodaq_ndogo

; change microcontroller
```

(continues on next page)
board_build.mcu = atmega1284p

; change MCU frequency
board_build.f_cpu = 8000000L

Debugging

PIO Unified Debugger currently does not support SODAQ Ndogo 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>

SODAQ Tatu

Contents

- SODAQ Tatu
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform 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.

<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):
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.

```yaml
[env:sodaq_tatu]
platform = atmelavr
board = sodaq_tatu

; change microcontroller
board_build.mcu = atmega1284p

; change MCU frequency
board_build.f_cpu = 8000000L
```

### Debugging

*PIO Unified Debugger* currently does not support SODAQ Tatu board.

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
</table>
| 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 ATmega1284p (16MHz)

#### Contents

- Sanguino ATmega1284p (16MHz)
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

#### Hardware

Platform *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.
### Configuration

Please use `sanguino_atmega1284p` ID for `board` option in “`platformio.ini` (Project Configuration File):

```ini
[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.

```ini
[env:sanguino_atmega1284p]
platform = atmelavr
board = sanguino_atmega1284p

; change microcontroller
board_build.mcu = atmega1284p

; change MCU frequency
board_build.f_cpu = 16000000L
```

### Debugging

PIO Unified Debugger currently does not support Sanguino ATmega1284p (16MHz) 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>

### Sanguino ATmega1284p (8MHz)

#### Contents

- Sanguino ATmega1284p (8MHz)
  - Hardware
  - Configuration
Hardware

Platform **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.

<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):

```ini
[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.

```ini
[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

*PIO Unified Debugger* currently does not support Sanguino ATmega1284p (8MHz) 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>
Sanguino ATmega644 or ATmega644A (16 MHz)

## Contents

- **Sanguino ATmega644 or ATmega644A (16 MHz)**
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

### Hardware

Platform **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.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>ATMEGA644</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_atmega644` ID for `board` option in `platformio.ini` *(Project Configuration File)*:

```ini
[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.

```ini
[env:sanguino_atmega644]
platform = atmelavr
board = sanguino_atmega644

; change microcontroller
board_build.mcu = atmega644

; change MCU frequency
board_build.f_cpu = 16000000L
```

### Debugging

*PIO Unified Debugger* currently does not support Sanguino ATmega644 or ATmega644A (16 MHz) 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>

Sanguino ATmega644 or ATmega644A (8 MHz)

Contents

- Sanguino ATmega644 or ATmega644A (8 MHz)
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform *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.

<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
[env: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
[env:sanguino_atmega644_8m]
platform = atmelavr
board = sanguino_atmega644_8m

; change microcontroller
```

(continues on next page)
board_build.mcu = atmega644

; change MCU frequency
board_build.f_cpu = 8000000L

**Debugging**

*PIO Unified Debugger* currently does not support Sanguino ATmega644 or ATmega644A (8 MHz) 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>

**Sanguino ATmega644P or ATmega644PA (16 MHz)**

**Contents**

- Sanguino ATmega644P or ATmega644PA (16 MHz)
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

**Hardware**

Platform *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.

<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*):
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.

```yaml
platform = atmelavr
board = sanguino_atmega644p

; change microcontroller
board_build.mcu = atmega644p

; change MCU frequency
board_build.f_cpu = 16000000L
```

### Debugging

`PIO Unified Debugger` currently does not support Sanguino ATmega644P or ATmega644PA (16 MHz) 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>

### Sanguino ATmega644P or ATmega644PA (8 MHz)

#### Contents

- Sanguino ATmega644P or ATmega644PA (8 MHz)
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

### Hardware

Platform `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.
### 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

*PIO Unified Debugger* currently does not support Sanguino ATmega644P or ATmega644PA (8 MHz) 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>

### Seeeduino

**Contents**

- Seeeduino
  - Hardware
  - Configuration
Hardware

Platform *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.

<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>SeeedStudio</td>
</tr>
</tbody>
</table>

Configuration

Please use *seeeduino* ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[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.

```
[env:seeeduino]
platform = atmelavr
board = seeeduino

; change microcontroller
board_build.mcu = atmega328p

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

*PIO Unified Debugger* currently does not support Seeeduino 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>Simba</td>
<td>Simba is an RTOS and build framework. It 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- 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.

<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_satmega128rfa1 ID for board option in “platformio.ini” (Project Configuration File):

```ini
[env:sparkfun_satmega128rfa1]
platform = atmelavr
board = sparkfun_satmega128rfa1
```

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_satmega128rfa1.json. For example, board_build.mcu, board_build.f_cpu, etc.

```ini
[env:sparkfun_satmega128rfa1]
platform = atmelavr
board = sparkfun_satmega128rfa1

; change microcontroller
board_build.mcu = atmega128rfa1

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

PIO Unified Debugger currently does not support SparkFun ATmega128RFA1 Dev 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>

SparkFun Digital Sandbox

Contents

- SparkFun Digital Sandbox
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform **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.

<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
```

You can override default SparkFun Digital Sandbox settings per build environment using **board_*** option, where *** is a JSON object path from board manifest **sparkfun_digitalsandbox.json**. For example, **board_build.mcu**, **board_build.f_cpu**, etc.

```ini
[env:sparkfun_digitalsandbox]
platform = atmelavr
board = sparkfun_digitalsandbox

; change microcontroller
```

(continues on next page)
board_build.mcu = atmega328p

; change MCU frequency
board_build.f_cpu = 8000000L

Debugging

PIO Unified Debugger currently does not support SparkFun Digital Sandbox 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>

SparkFun Fio V3 3.3V/8MHz

Contents

- SparkFun Fio V3 3.3V/8MHz
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform 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.

<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):
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
board_build.mcu = atmega32u4

; change MCU frequency
board_build.f_cpu = 8000000L
```

**Debugging**

PIO Unified Debugger currently does not support SparkFun Fio V3 3.3V/8MHz 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>

**SparkFun Makey Makey**

**Contents**

- SparkFun Makey Makey
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

**Hardware**

Platform **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.
Microcontroller | ATMEGA32U4
---|---
Frequency | 16MHz
Flash | 28KB
RAM | 2.50KB
Vendor | SparkFun

### 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

PIO Unified Debugger currently does not support SparkFun Makey Makey 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>

### SparkFun Mega Pro 3.3V/8MHz

#### Contents

- SparkFun Mega Pro 3.3V/8MHz
  - Hardware
  - Configuration
Hardware

Platform **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.

<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)*:

```ini
[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.

```ini
[env:sparkfun_megapro8MHz]
platform = atmelavr
board = sparkfun_megapro8MHz

; change microcontroller
board_build.mcu = atmega2560

; change MCU frequency
board_build.f_cpu = 8000000L
```

Debugging

**PIO Unified Debugger** currently does not support SparkFun Mega Pro 3.3V/8MHz 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>
</tbody>
</table>
SparkFun Mega Pro 5V/16MHz

**Hardware**

Platform *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.

<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)*:

```
[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.

```
[env:sparkfun_megapro16MHz]
platform = atmelavr
board = sparkfun_megapro16MHz

; change microcontroller
board_build.mcu = atmega2560

; change MCU frequency
board_build.f_cpu = 16000000L
```

**Debugging**

**PIO Unified Debugger** currently does not support SparkFun Mega Pro 5V/16MHz 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>

SparkFun Mega Pro Mini 3.3V

Contents

- SparkFun Mega Pro Mini 3.3V
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform **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.

<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

PIO Unified Debugger currently does not support SparkFun Mega Pro Mini 3.3V 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>

SparkFun MicroView

Contents

- SparkFun MicroView
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform 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.

<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):
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.

```python
[env:uview]
platform = atmelavr
board = uview

; change microcontroller
board_build.mcu = atmega328p

; change MCU frequency
board_build.f_cpu = 16000000L
```

**Debugging**

*PIO Unified Debugger* currently does not support SparkFun MicroView 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>

**SparkFun Pro Micro 3.3V/8MHz**

**Contents**

- SparkFun Pro Micro 3.3V/8MHz
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

**Hardware**

Platform *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.
### Configuration

Please use sparkfun_promicro8 ID for `board` option in "platformio.ini" (Project Configuration File):

```ini
[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.

```ini
[env:sparkfun_promicro8]
platform = atmelavr
board = sparkfun_promicro8

; change microcontroller
board_build.mcu = atmega32u4

; change MCU frequency
board_build.f_cpu = 8000000L
```

### Debugging

PIO Unified Debugger currently does not support SparkFun Pro Micro 3.3V/8MHz 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>
Hardware

Platform **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.

<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_promicro16` ID for `board` option in **“platformio.ini” (Project Configuration File)**:

```plaintext
[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.

```plaintext
[env:sparkfun_promicro16]
platform = atmelavr
board = sparkfun_promicro16

; change microcontroller
board_build.mcu = atmega32u4

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

PIO Unified Debugger currently does not support SparkFun Pro Micro 5V/16MHz 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>
SparkFun Qduino Mini

Contents

- SparkFun Qduino Mini
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform 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.

<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):

```
[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.

```
[env:sparkfun_qduinomini]
platform = atmelavr
board = sparkfun_qduinomini

; change microcontroller
board_build.mcu = atmega32u4

; change MCU frequency
board_build.f_cpu = 8000000L
```

Debugging

PIO Unified Debugger currently does not support SparkFun Qduino 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>

SparkFun RedBoard

Contents

- SparkFun RedBoard
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform *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.

<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)*:

```
[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.

```
[env:sparkfun_redboard]
platform = atmelavr
board = sparkfun_redboard

; change microcontroller
```

(continues on next page)
board_build.mcu = atmega328p

; change MCU frequency
board_build.f_cpu = 16000000L

Debugging

PIO Unified Debugger currently does not support SparkFun RedBoard 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>

SparkFun Serial 7-Segment Display

Hardware

Platform 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.

<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):
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.

```python
[env:sparkfun_serial7seg]
platform = atmelavr
board = sparkfun_serial7seg

; change microcontroller
board_build.mcu = atmega328p

; change MCU frequency
board_build.f_cpu = 8000000L
```

### Debugging

*PIO Unified Debugger* currently does not support SparkFun Serial 7-Segment Display 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>

### SpellFoundry Sleepy Pi 2

#### Contents

- SpellFoundry Sleepy Pi 2
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

#### Hardware

Platform *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.
### PlatformIO Documentation, Release 4.0.1b3

<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

*PIO Unified Debugger* currently does not support SpellFoundry Sleepy Pi 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>

### Talk2 Whisper Node

**Contents**

- *Talk2 Whisper Node*
  - Hardware
  - Configuration
Hardware

Platform **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.

<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
board_build.mcu = atmega328p

; change MCU frequency
board_build.f_cpu = 16000000L
```

**Debugging**

**PIO Unified Debugger** currently does not support Talk2 Whisper 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>
</tbody>
</table>
The Things Uno

Contents

- The Things Uno
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform 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 industries 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_things_uno ID for board option in "platformio.ini" (Project Configuration File):

```
[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.

```
[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

PIO Unified Debugger currently does not support The Things Uno 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>

TinyCircuits TinyDuino Processor Board

Contents

- TinyCircuits TinyDuino Processor Board
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform *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.

<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

*PIO Unified Debugger* currently does not support TinyCircuits TinyDuino Processor 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>

### TinyCircuits TinyLily Mini Processor

#### Contents

- TinyCircuits TinyLily Mini Processor
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

#### Hardware

Platform *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.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
<th>Vendor</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATMEGA328P</td>
<td>8MHz</td>
<td>30KB</td>
<td>2KB</td>
<td>TinyCircuits</td>
</tr>
</tbody>
</table>

#### Configuration

Please use `tinylily` ID for *board* option in “platformio.ini” (*Project Configuration File***):
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.

```python
[env:tinylily]
platform = atmelavr
board = tinylily

; change microcontroller
board_build.mcu = atmega328p

; change MCU frequency
board_build.f_cpu = 8000000L
```

**Debugging**

PIO Unified Debugger currently does not support TinyCircuits TinyLily Mini Processor 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>

**USBasp stick**

**Contents**

- **USBasp stick**
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

**Hardware**

Platform **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.
### 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
board_build.mcu = atmega8

; change MCU frequency
board_build.f_cpu = 12000000L
```

### Debugging

**PIO Unified Debugger** currently does not support USBasp 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>

### Wicked Device WildFire V2

**Contents**

- Wicked Device WildFire V2
  - Hardware
  - Configuration
Hardware

Platform *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.

<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>120.00KB</td>
</tr>
<tr>
<td>RAM</td>
<td>16KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Wicked Device</td>
</tr>
</tbody>
</table>

Configuration

Please use *wildfirev2* ID for `board` option in “*platformio.ini*” (*Project Configuration File*):

```
[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.

```
[env:wildfirev2]
platform = atmelavr
board = wildfirev2

; change microcontroller
board_build.mcu = atmegal284p

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

*PIO Unified Debugger* currently does not support Wicked Device WildFire V2 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>
Wicked Device WildFire V3

Contents

- Wicked Device WildFire V3
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform *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.

<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
board_build.mcu = atmega1284p

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

*PIO Unified Debugger* currently does not support Wicked Device WildFire V3 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>

ftDuino

Contents

- ftDuino
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform *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.

<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>Till Harbaum</td>
</tr>
</tbody>
</table>

Configuration

Please use `ftduino` ID for *board* option in “platformio.ini” (*Project Configuration File*):

```ini
[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.`

```ini
[env:ftduino]
platform = atmelavr
board = ftduino

; change microcontroller
board_build.mcu = atmega32u4
```

(continues on next page)
Debugging

*PIO Unified Debugger* currently does not support ftDuino 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>

nicai-systems BOB3 coding bot

Contents

- nicai-systems BOB3 coding bot
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform *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.

<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*):
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

*PIO Unified Debugger* currently does not support nicai-systems BOB3 coding bot 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>

### nicai-systems NIBO 2 robot

#### Contents

- nicai-systems NIBO 2 robot
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

#### Hardware

Platform *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.
### PlatformIO Documentation, Release 4.0.1b3

<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):

```ini
[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.

```ini
[env:nibo2]
platform = atmelavr
board = nibo2

; change microcontroller
board_build.mcu = atmega128

; change MCU frequency
board_build.f_cpu = 16000000L
```

### Debugging

*PIO Unified Debugger* currently does not support nicai-systems NIBO 2 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>

### nicai-systems NIBO burger robot

#### Contents

- nicai-systems NIBO burger robot
  - Hardware
  - Configuration
Hardware

Platform *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.

<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*):

```ini
[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.

```ini
[env:niboburger]
platform = atmelavr
board = niboburger

; change microcontroller
board_build.mcu = atmegal6

; change MCU frequency
board_build.f_cpu = 15000000L
```

Debugging

*PIO Unified Debugger* currently does not support nicai-systems NIBO burger 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>
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- 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.

<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

**PIO Unified Debugger** currently does not support nicai-systems NIBO burger robot with Tuning Kit 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>

nicai-systems NIBObee robot

Contents

- nicai-systems NIBObee robot
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform **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.

<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
```

(continues on next page)
board_build.mcu = atmega16

; change MCU frequency
board_build.f_cpu = 15000000L

Debugging

PIO Unified Debugger currently does not support nicai-systems NIBObee 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>

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- 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.

<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)*:
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.

Debugging

PIO Unified Debugger currently does not support nicai-systems NIBObee robot with Tuning Kit 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>

ubIQio Ardhat

<table>
<thead>
<tr>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>* ubIQio Ardhat</td>
</tr>
<tr>
<td>  Hardware</td>
</tr>
<tr>
<td>  Configuration</td>
</tr>
<tr>
<td>  Debugging</td>
</tr>
<tr>
<td>  Frameworks</td>
</tr>
</tbody>
</table>

Hardware

Platform 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.
Microcontroller | ATMEGA328P  
---|---
Frequency | 16MHz  
Flash | 31.50KB  
RAM | 2KB  
Vendor | ubIQio

**Configuration**

Please use `ardhat` ID for `board` option in “platformio.ini” *(Project Configuration File)*:

```ini
[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.

```ini
[env:ardhat]
platform = atmelavr
board = ardhat

; change microcontroller
board_build.mcu = atmega328p

; change MCU frequency
board_build.f_cpu = 16000000L
```

**Debugging**

PIO Unified Debugger currently does not support ubIQio Ardhat 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 SAM**

*Adafruit Circuit Playground Express*

**Contents**

- *Adafruit Circuit Playground 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>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):

```
[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.

```
[env:adafruit_circuitplayground_m0]
platform = atmelsam
board = adafruit_circuitplayground_m0

; change microcontroller
board_build.mcu = samd21g18a

; change MCU frequency
board_build.f_cpu = 48000000L
```

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:

```ini
[env:adafruit_circuitplayground_m0]
platform = atmelsam
board = adafruit_circuitplayground_m0
upload_protocol = sam-ba
```

**Debugging**

*PIO Unified 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).

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
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):

```
[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.

```
[env:adafruit_crickit_m0]
platform = atmelsam
board = adafruit_crickit_m0

; 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

PIO Unified 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).

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*):

```ini
[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.

```ini
[env:adafruit_feather_m0]
platform = atmelsam
board = adafruit_feather_m0

; change microcontroller
board_build.mcu = samd21g18a

; change MCU frequency
board_build.f_cpu = 48000000L
```

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

PIO Unified 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).

Adafruit Feather M0 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>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 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):

```ini
[env:adafruit_feather_m0_express]
platform = atmelsem
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.

```ini
[env:adafruit_feather_m0_express]
platform = atmelsem
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:

```ini
[env:adafruit_feather_m0_express]
platform = atmelsem
board = adafruit_feather_m0_express

upload_protocol = sam-ba
```

**Debugging**

PIO Unified 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*).

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>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 M4 Express**

**Contents**

- *Adafruit Feather 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_feather_m4` ID for `board` option in “`platformio.ini`” (Project Configuration File):

```
[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.

```
[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

PIO Unified 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).

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.
Compatible Tools | On-board | Default
--- | --- | ---
Atmel-ICE | 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>
</tbody>
</table>

Adafruit Gemma M0

Contents

- Adafruit Gemma 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.

| Microcontroller | SAMD21E18A |
| Frequency | 48MHz |
| Flash | 256KB |
| RAM | 32KB |
| Vendor | Adafruit |

Configuration

Please use `adafruit_gemma_m0` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:adafruit_gemma_m0]
platform = atmel-sam
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.
[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:

[env:adafruit_gemma_m0]
platform = atmelsam
board = adafruit_gemma_m0

upload_protocol = sam-ba

Debugging

PIO Unified 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).

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
```

(continues on next page)
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:

```ini
[env:adafruit_grandcentral_m4]
platform = atmelstm
board = adafruit_grandcentral_m4
upload_protocol = sam-ba
```

Debugging

**PIO Unified 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).

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.

<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_hallowing ID for board option in “platformio.ini” (Project Configuration File):

```
[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.

```
[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

**PIO Unified 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).

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.

#### 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>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.

#### Adafruit ItsyBitsy M0

**Contents**

- **Adafruit ItsyBitsy M0**
  - **Hardware**
PlatformIO Documentation, Release 4.0.1b3

- 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 = atmel
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.

```ini
[env:adafruit_itsybitsy_m0]
platform = atmel
board = adafruit_itsybitsy_m0

; change microcontroller
board_build.mcu = samd21g18a

; change MCU frequency
board_build.f_cpu = 48000000L
```

**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:

```yaml
[env:adafruit_itsybitsy_m0]
platform = atmelsam
board = adafruit_itsybitsy_m0
upload_protocol = sam-ba
```

## Debugging

**PIO Unified 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)

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

**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 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*):

```ini
[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.f_cpu`, etc.

```ini
[env:adafruit_itsybitsy_m4]
platform = atmelsam
board = adafruit_itsybitsy_m4

; 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

**PIO Unified 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).

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></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.
### 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:adafruit_metro_m0]
platform = atmelsam
board = adafruit_metro_m0

upload_protocol = sam-ba
```

### Debugging

**PIO Unified Debugger** - “1-click” solution for debugging with a zero configuration.
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
  - 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` ID for `board` option in “`platformio.ini`” (Project Configuration File):

```
[env:adafruit_metro_m4]
platform = atmelsam
board = adafruit_metro_m4
```

You can override default Adafruit Metro M4 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `adafruit_metro_m4.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:adafruit_metro_m4]
platform = atmelsam
board = adafruit_metro_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 = atmelsam
board = adafruit_metro_m4

upload_protocol = sam-ba
```

Debugging

PIO Unified 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).

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.
### 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 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*):

```ini
[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_airliflite.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:adafruit_metro_m4_airliflite]
platform = atmelsam
board = adafruit_metro_m4_airliflite

; 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:

```
[env:adafruit_metro_m4_airliflite]
platform = atmelsam
board = adafruit_metro_m4_airliflite

upload_protocol = sam-ba
```

### Debugging

**PIO Unified 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).

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><strong>Atmel-ICE</strong></td>
<td></td>
<td><strong>Yes</strong></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>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached 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
```

(continues on next page)
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 = atmel sam
board = adafruit_pygamer_advance_m4
upload_protocol = sam-ba
```

Debugging

PIO Unified 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).

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:
PlatformIO Documentation, Release 4.0.1b3

- atmel-ice
- jlink
- sam-ba

Default protocol is sam-ba

You can change upload protocol using `upload_protocol` option:

```ini
[env:adafruit_pygamer_m4]
platform = atmel-sam
board = adafruit_pygamer_m4
upload_protocol = sam-ba
```

### Debugging

**PIO Unified 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).

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.

<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

#### Contents

- **Adafruit PyPortal M4**
  - 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>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):

```ini
[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.

```ini
[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

PIO Unified 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).

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 Trellis M4

Contents

- Adafruit Trellis 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>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):

```ini
[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.

```ini
[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:

```ini
[env:adafruit_trellis_m4]
platform = atmelsam
board = adafruit_trellis_m4

upload_protocol = sam-ba
```
Debugging

PIO Unified 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).

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.
### 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

*PIO Unified 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).

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>
</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.

<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_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

PIO Unified 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).
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 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.

```
[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:

```
[env:adafruit_pybadge_m4]
platform = atmelsam
board = adafruit_pybadge_m4

upload_protocol = sam-ba
```

### Debugging

*PIO Unified 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).

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):

```ini
[env:due]
platform = atmelsam
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.

```ini
[env:due]
platform = atmelsam
board = due
```

(continues on next page)
Uploading

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:

```snippet
[env:due]
platform = atmel-sam
board = due
upload_protocol = sam-ba
```

Debugging

**PIO Unified 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 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. It aims to make embedded programming easy and portable.</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 = atmelsam
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.

```
[env:dueUSB]
platform = atmelsam
board = dueUSB
```

(continues on next page)
; change microcontroller
board_build.mcu = at91sam3x8e

; change MCU frequency
board_build.f_cpu = 84000000L

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:

```
[env:dueUSB]
platform = atmelsam
board = dueUSB
upload_protocol = sam-ba
```

Debugging

PIO Unified 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 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>
<tr>
<td>Simba</td>
<td>Simba is an RTOS and build framework. It 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 = atmelsam
board = mzeroUSB
upload_protocol = stk500v2
```

Debugging

PIO Unified 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 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 `mzeroproUSB` ID for `board` option in “platformio.ini” *(Project Configuration File)*:

```
[env:mzeroproUSB]
platform = atmelsam
board = mzeroproUSB
```

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 `mzeroproUSB.json`. For example, `board_build.mcu,board_build.f_cpu`, etc.

```
[env:mzeroproUSB]
platform = atmelsam
board = mzeroproUSB
```

(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

**PIO Unified 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 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 = atmel-sam
board = mzeropro
upload_protocol = cmsis-dap
```

Debugging

PIO Unified 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 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 mkrfox1200 ID for board option in “platformio.ini” (Project Configuration File):

```ini
[env:mkrfox1200]
platform = atmelsam
board = mkrfox1200
```

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.

```ini
[env:mkrfox1200]
platform = atmelsam
board = mkrfox1200
```
PlatformIO Documentation, Release 4.0.1b3

(continued from previous page)

```plaintext
; change microcontroller
board_build.mcu = samd21g18a

; change MCU frequency
board_build.f_cpu = 48000000L
```

### 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 = atmel-sam
board = mkrfox1200
upload_protocol = sam-ba
```

### Debugging

**PIO Unified 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 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><strong>Atmel-ICE</strong></td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Black Magic Probe</strong></td>
<td></td>
<td></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>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached 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):

```plaintext
[env:mkrgsm1400]
platform = atmelsam
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.

```plaintext
[env:mkrgsm1400]
platform = atmelsam
board = mkrgsm1400
```

(continues on next page)
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

PIO Unified 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 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>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>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached 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):

```ini
[env:mkrnb1500]
platform = atmelsam
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.

```ini
[env:mkrnb1500]
platform = atmelsam
board = mkrnb1500
```
; 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:

```ini
[env:mkrnb1500]
platform = atmelsam
board = mkrnb1500
upload_protocol = sam-ba
```

## Debugging

**PIO Unified 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 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>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 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 = atmelsam
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 = atmelsam
board = mkrwan1300
```

(continues on next page)
}; change microcontroller
board_build.mcu = samd21g18a

}; change MCU frequency
board_build.f_cpu = 48000000L

### 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

*PIO Unified 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 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>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):

```
[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.

```
[env:mkrwifi1010]
platform = atmelsam
board = mkrwifi1010
```
### 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:

```ini
[env:mkrwifi1010]
platform = atmelsam
board = mkrwifi1010
upload_protocol = sam-ba
```

### Debugging

**PIO Unified 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 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>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 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)
### 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

**PIO Unified 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 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><strong>Atmel-ICE</strong></td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td><strong>Black Magic Probe</strong></td>
<td></td>
<td></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>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached 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):

```ini
[env:mkrzero]
platform = atmelSsam
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.

```ini
[env:mkrzero]
platform = atmelSsam
board = mkrzero
```

(continues on next page)
Uploading

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:

```ini
[env:mkrzero]
platform = atmelsam
board = mkrzero
upload_protocol = sam-ba
```

Debugging

PIO Unified 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 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>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>

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 Tian

Contents

- Arduino Tian
  - 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 *tian* ID for *board* option in “*platformio.ini*” *(Project Configuration File)*:

```
[env:tian]
platform = atmel_sam
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_sam
board = tian

; change microcontroller
```

(continues on next page)
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 = atmelsam
board = tian
upload_protocol = stk500v2
```

Debugging

PIO Unified 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 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><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 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:

```
[env:zero]
platform = atmelsam
board = zero
upload_protocol = cmsis-dap
```

### Debugging

**PIO Unified 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 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>
</tbody>
</table>

### Arduino Zero (USB Native Port)

**Contents**

- Arduino Zero (USB Native Port)
PlatformIO Documentation, Release 4.0.1b3

- 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 zeroUSB ID for board option in “platformio.ini” (Project Configuration File):

```
[env:zeroUSB]
platform = atmelsam
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 = atmelsam
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

**PIO Unified 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 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
  - Debugging
  - Frameworks
Hardware

Platform Atmel SAM: 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.

<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:

```ini
[env:samr21_xpro]
platform = atmelsam
board = samr21_xpro

upload_protocol = cmsis-dap
```
Debugging

**PIO Unified 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)*.

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>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.</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.
### PlatformIO Documentation, Release 4.0.1b3

<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

**PIO Unified 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).

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>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.</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.
**Microcontroller**  |  SAMD21J18A  
--- | ---  
**Frequency**  |  48MHz  
**Flash**  |  256KB  
**RAM**  |  32KB  
**Vendor**  |  Atmel  

### Configuration

Please use `samd21_xpro` ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[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

**PIO Unified 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).

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>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>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.</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.
### PlatformIO Documentation, Release 4.0.1b3

<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)*:

```ini
[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.

```ini
[env:saml21_xpro_b]
platform = atmelsam
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:

```ini
[env:saml21_xpro_b]
platform = atmelsam
board = saml21_xpro_b

upload_protocol = cmsis-dap
```

### Debugging

*PIO Unified 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).

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>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.</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.
### Configuration

Please use `digix` ID for `board` option in "platformio.ini" (Project Configuration File):

```
[env:digix]
platform = atmega328
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.

```
[env:digix]
platform = atmega328
board = digix

; change microcontroller
board_build.mcu = at91sam3x8e

; change MCU frequency
board_build.f_cpu = 84000000L
```

### 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 = atmega328
board = digix

upload_protocol = sam-ba
```

### Debugging

**PIO Unified 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).

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 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>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

; change MCU frequency
board_build.f_cpu = 48000000L
```

Debugging

`PIO Unified Debugger` 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):

```ini
[env:mkrvidor4000]
platform = atmelsam
board = mkrvidor4000
```

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

*PIO Unified 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).

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>

### Macchina M2

**Contents**

- **Macchina M2**
  - 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 macchina2 ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:macchina2]
platform = atmelsam
board = macchina2
```

You can override default Macchina M2 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `macchina2.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```ini
[env:macchina2]
platform = atmelsam
board = macchina2

; change microcontroller
board_build.mcu = at91sam3x8e

; change MCU frequency
board_build.f_cpu = 84000000L
```

### Uploading

Macchina M2 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:macchina2]
platform = atmelsam
board = macchina2

upload_protocol = sam-ba
```

### Debugging

**PIO Unified 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).

Macchina M2 does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one 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>

**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.

```
[env:minitronics20]
platform = atmelsam
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:

```
[env:minitronics20]
platform = atmelsam
board = minitronics20

upload_protocol = sam-ba
```

Debugging

`PIO Unified 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).
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>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>

### 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**):

```ini
[env:moteino_zero]
platform = atmelsam
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.

```ini
[env:moteino_zero]
platform = atmelsam
board = moteino_zero

; change microcontroller
board_build.mcu = samd21g18a

; change MCU frequency
board_build.f_cpu = 48000000L
```

### 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 = atmelsam
board = moteino_zero

upload_protocol = sam-ba
```

### Debugging

*PIO Unified 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).

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.
### 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>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)*:

```ini
[env:nano_33_iot]
platform = atmel|sam
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.

```ini
[env:nano_33_iot]
platform = atmelSAM
board = nano_33_iot

; change microcontroller
board_build.mcu = samd21g18a

; change MCU frequency
board_build.f_cpu = 48000000L
```

### 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

**PIO Unified 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).

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>
</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):

```
[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 = atmelsam
board = sodaq_autonomo
```

(continues on next page)
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 = atmelsam
board = sodaq_autonomo
upload_protocol = sam-ba
```

Debugging

**PIO Unified 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).

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>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 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):

```
[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.

```
[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 = atmelsam
board = sodaq_explorer
upload_protocol = sam-ba
```

### Debugging

**PIO Unified 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).

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):

```
[env:sodaq_one]
platform = atmelsam
board = sodaq_one
```

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.

```
[env:sodaq_one]
platform = atmelsam
board = sodaq_one
```

(continues on next page)
; change microcontroller
board_build.mcu = \texttt{samd21g18a}

; change MCU frequency
board_build.f_cpu = \texttt{48000000L}

\section*{Uploading}

SODAQ ONE supports the next uploading protocols:

- \texttt{atmel-ice}
- \texttt{blackmagic}
- \texttt{jlink}
- \texttt{sam-ba}

Default protocol is \texttt{sam-ba}

You can change upload protocol using \texttt{upload\_protocol} option:

\begin{verbatim}
[env:sodaq\_one]
platform = \texttt{atmelsam}
board = \texttt{sodaq\_one}
upload\_protocol = \texttt{sam-ba}
\end{verbatim}

\section*{Debugging}

\textit{PIO Unified Debugger} - “1-click” solution for debugging with a zero configuration.

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|}
\hline
Compatible Tools & On-board & Default \tabularnewline
\hline
\texttt{Atmel-ICE} & & Yes \tabularnewline
\texttt{Black Magic Probe} & & \tabularnewline
\texttt{J-LINK} & & \tabularnewline
\hline
\end{tabular}
\caption{Compatible debug tools for SODAQ ONE.}
\end{table}

\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 \texttt{Tools & Debug Probes} using \texttt{debug\_tool} option in “\texttt{platformio.ini}” (Project Configuration File).

SODAQ ONE does not have on-board debug probe and \textbf{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>

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):

```
[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.

```
[env:sodaq_sara]
platform = atmelsam
board = sodaq_sara
```

(continues on next page)
; change microcontroller
board_build.mcu = samd21j18a

; change MCU frequency
board_build.f_cpu = 48000000L

Uploading

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:

[env:sodaq_sara]
platform = atmelsam
board = sodaq_sara
upload_protocol = sam-ba

Debugging

PIO Unified 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).

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 | 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):

```ini
[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.

```ini
[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:

```ini
[env:sodaq_sff]
platform = atmelSAM
board = sodaq_sff
upload_protocol = sam-ba
```

### Debugging

**PIO Unified 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).

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):

```
[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.

```
[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:
• 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**

*PIO Unified 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)*.

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):

```ini
[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.

```ini
[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:

```ini
[env:sainSmartDueUSB]
platform = atmel-sam
board = sainSmartDueUSB
upload_protocol = sam-ba
```

### Debugging

**PIO Unified 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).

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

- **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 LoRaWAN

- **Seeeduino LoRaWAN**
  - **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 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

PIO Unified 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).

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached 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

; change microcontroller
board_build.mcu = samd21g18a

; change MCU frequency
board_build.f_cpu = 48000000L
```

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:

```ini
[env:sparkfun_samd21_dev_usb]
platform = atmelsam
board = sparkfun_samd21_dev_usb

upload_protocol = sam-ba
```
Debugging

PIO Unified 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).

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.
Microcontroller | SAMD21G18A
---|---
Frequency | 48MHz
Flash | 256KB
RAM | 32KB
Vendor | SparkFun

### 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

; change microcontroller
board_build.mcu = samd21g18a

; change MCU frequency
board_build.f_cpu = 48000000L
```

### 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

PIO Unified 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).

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>

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):

```ini
[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.

```ini
[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:

```ini
[env:tuinozero96]
platform = atmelsam
board = tuinozero96

upload_protocol = stk500v2
```

Debugging

PIO Unified 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).
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>

### 1.12.4 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.

<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)*:
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:

- espota
- esptool

Default protocol is esptool

You can change upload protocol using `upload_protocol` option:

```
[env:esp32cam]
platform = espressif32
board = esp32cam

upload_protocol = esptool
```

**Debugging**

*PIO Unified Debugger* currently does not support AI Thinker ESP32-CAM 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>

**ALKS 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>RoboticsBrno</td>
</tr>
</tbody>
</table>

Configuration

Please use `alksesp32` ID for `board` option in “platformio.ini” (*Project Configuration File)*:

```
[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.

```
[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 = alkses32
upload_protocol = esptool
```

### Debugging

**PIO Unified 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).

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>

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):

```
[env:featheresp32]
platform = esp32
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 = esp32
board = featheresp32
```

(continues on next page)
Uploading

Adafruit ESP32 Feather 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:featheresp32]
platform = espressif32
board = featheresp32
upload_protocol = esptool
```

Debugging

*PIO Unified 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)*.

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.
Compatible Tools | On-board | Default
--- | --- | ---
ESP-Prog | | Yes
oddWires IOT-Bus JTAG | | |
J-LINK | | |
Mini-Module FT2232H | | |
Olimex ARM-USB-OCD | | |
Olimex ARM-USB-OCD-H | | |
Olimex ARM-USB-TINY-H | | |
Olimex ARM-USB-TINY | | |
TIAO USB Multi-Protocol Adapter (TUMPA) | | |

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.</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 ESPe32 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 ESPe32 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

`PIO Unified Debugger` currently does not support April Brother ESPe32 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>
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):

```
[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.

```
[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**

*PIO Unified Debugger* 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>
</tbody>
</table>

**D-duino-32**

**Contents**

- D-duino-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>DSTIKE</td>
</tr>
</tbody>
</table>

**Configuration**

Please use d-duino-32 ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[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.

```ini
[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:
Debugging

PIO Unified 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).

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>
</tbody>
</table>

DOIT ESP32 DEVKIT V1

Contents

- `DOIT ESP32 DEVKIT V1`
  - 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>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 = espressif32
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.

```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
```

**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:

```
[env:esp32doit-devkit-v1]
platform = espressif32
board = esp32doit-devkit-v1
upload_protocol = esptool
```

**Debugging**

*PIO Unified 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).

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></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>

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):

```
[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.
[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:

[env:pocket_32]
platform = espressif32
board = pocket_32

upload_protocol = esptool

Debugging

PIO Unified 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).
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>
</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*):

```plaintext
[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.

```plaintext
[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-octd
- olimex-arm-usb-octd-h
- olimex-arm-usb-tiny-h
- olimex-jtag-tiny
- tumpa

Default protocol is *esptool*

You can change upload protocol using *upload_protocol* option:

```plaintext
[env:fm-devkit]
platform = espressif32
board = fm-devkit

upload_protocol = esptool
```
Debugging

PIO Unified 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).

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><code>ESP-Prog</code></td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td><code>oddWires IOT-Bus JTAG</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>J-LINK</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>Mini-Module FT2232H</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>Olimex ARM-USB-OCD</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>Olimex ARM-USB-OCD-H</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>Olimex ARM-USB-TINY-H</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>Olimex ARM-USB-TINY</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>TIAO USB Multi-Protocol Adapter (TUMPA)</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>

ESP32 Pico Kit

Contents

- ESP32 Pico 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 `pico32` ID for `board` option in "platformio.ini" (Project Configuration File):

```ini
[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.

```ini
[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:

- espota
- esptool

Default protocol is esptool

You can change upload protocol using `upload_protocol` option:

```ini
[env:pico32]
platform = espressif32
board = pico32

upload_protocol = esptool
```
Debugging

*PIO Unified Debugger* currently does not support ESP32 Pico Kit 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>

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):

```ini
[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.

```ini
[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:

```ini
[env:esp32vn-iot-uno]
platform = espressif32
board = esp32vn-iot-uno

upload_protocol = esptool
```

## Debugging

**PIO Unified 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).

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>
</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.
Microcontroller | ESP32  
---|---
Frequency | 240MHz  
Flash | 4MB  
RAM | 320KB  
Vendor | Dycodex

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

PIO Unified 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)**.

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>
</tbody>
</table>

ESPino32

Contents

- **ESPino32**
  - 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>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:

```
[env:espino32]
platform = espressif32
board = espino32
upload_protocol = esptool
```

## Debugging

**PIO Unified 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).

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>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>
Electronic SweetPeas ESP32

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)**:

```ini
[env:esp320]
platform = espressif32
board = esp320
```

You can override default Electronic SweetPeas ESP32 protocol 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.

```ini
[env:esp320]
platform = espressif32
board = esp320

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

Uploading

Electronic SweetPeas ESP32 supports the next uploading protocols:
• espota
• esptool

Default protocol is esptool

You can change upload protocol using *upload_protocol* option:

```
[env:esp320]
platform = espressif32
board = esp32

upload_protocol = esptool
```

**Debugging**

*PIO Unified Debugger* 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>
</tbody>
</table>

**Espressif ESP-WROVER-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.
## 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-octd
- olimex-arm-usb-octd-h
- olimex-arm-usb-tiny-h
- olimex-jtag-tiny
- tumpa

Default protocol is `esptool`

You can change upload protocol using `upload_protocol` option:
[env:esp-wrover-kit]
platform = espressif32
board = esp-wrover-kit
upload_protocol = esptool

**Debugging**

*PIO Unified 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).

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>
</tbody>
</table>

**Espressif ESP32 Dev Module**

**Contents**

- *Espressif 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>
</tbody>
</table>

Vendor: Espressif

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**

**PIO Unified 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).

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>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>

FireBeetle-ESP32

Contents

- FireBeetle-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>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.ff_cpu, etc.`
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

PIO Unified 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).
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>
</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 \texttt{frogboard} ID for \texttt{board} option in “\texttt{platformio.ini}” (Project Configuration File):

```
[env:frogboard]
platform = espressif32
board = frogboard
```

You can override default Frog Board ESP32 settings per build environment using \texttt{board\_***} option, where *** is a JSON object path from board manifest \texttt{frogboard.json}. For example, \texttt{board\_build.mcu}, \texttt{board\_build.f\_cpu}, etc.

```
[env:frogboard]
platform = espressif32
board = frogboard

; change microcontroller
board\_build.mcu = esp32

; change MCU frequency
board\_build.f\_cpu = 240000000L
```

Uploading

Frog Board ESP32 supports the next uploading protocols:

- \texttt{esp-prog}
- \texttt{espota}
- \texttt{esptool}
- \texttt{iot-bus-jtag}
- \texttt{jlink}
- \texttt{minimodule}
- \texttt{olimex-arm-usb-ocd}
- \texttt{olimex-arm-usb-ocd-h}
- \texttt{olimex-arm-usb-tiny-h}
- \texttt{olimex-jtag-tiny}
- \texttt{tumpa}

Default protocol is \texttt{esptool}

You can change upload protocol using \texttt{upload\_protocol} option:

```
[env:frogboard]
platform = espressif32
board = frogboard

upload\_protocol = esptool
```
Debugging

PIO Unified 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).

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>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>

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)**:

```ini
[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.

```ini
[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:

```ini
[env:heltec_wifi_kit_32]
platform = espressif32
board = heltec_wifi_kit_32

upload_protocol = esptool
```
Debugging

PIO Unified Debugger currently does not support Heltec WiFi Kit 32 board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.</td>
<td></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):

```
[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.

```
[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:

```
[env:heltec_wifi_lora_32]
platform = espressif32
board = heltec_wifi_lora_32

upload_protocol = esptool
```

**Debugging**

**PIO Unified 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).

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>
</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.
## PlatformIO Documentation, Release 4.0.1b3

### 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>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-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:heltec_wifi_lora_32_V2]
platform = espressif32
board = heltec_wifi_lora_32_V2
upload_protocol = esptool

### Debugging

*PIO Unified 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).

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>
</tbody>
</table>

### Heltec Wireless Stick

#### Contents

- *Heltec Wireless Stick*
  - Hardware
Platform \textit{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 \textit{board} option in “\textit{platformio.ini}” (Project Configuration File):

```ini
[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.

```ini
[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-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:heltec_wireless_stick]
platform = espressif32
board = heltec_wireless_stick
upload_protocol = esptool
```

### Debugging

**PIO Unified 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).

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>
</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

PIO Unified 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).
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></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>

### 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

**PIO Unified 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).

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>
</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*):

```ini
[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.

```ini
[env:intorobot]
platform = espressif32
board = intorobot

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

Uploading

IntoRobot Fig supports the next uploading protocols:

* espota
* esptool

Default protocol is *esptool*

You can change upload protocol using *upload_protocol* option:

```ini
[env:intorobot]
platform = espressif32
board = intorobot

upload_protocol = esptool
```
Debugging

PIO Unified Debugger 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>
</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):

```bash
[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.

```plaintext
[env:m5stack-core-esp32]
platform = espressif32
board = m5stack-core-esp32

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000
```

**Uploading**

M5Stack Core ESP32 supports the next uploading protocols:

- `espota`
- `esptool`

Default protocol is `esptool`

You can change upload protocol using `upload_protocol` option:

```plaintext
[env:m5stack-core-esp32]
platform = espressif32
board = m5stack-core-esp32

upload_protocol = esptool
```

**Debugging**

*PIO Unified Debugger* 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>
</tbody>
</table>

**M5Stack FIRE**

- M5Stack FIRE
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>6.25MB</td>
</tr>
<tr>
<td>Vendor</td>
<td>M5Stack</td>
</tr>
</tbody>
</table>

Configuration

Please use `m5stack-fire` ID for `board` option in “`platformio.ini`” (Project Configuration File):

```
[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.

```
[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:


Debugging

PIO Unified Debugger 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>
</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):

```
[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

PIO Unified Debugger 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>
</tbody>
</table>
M5Stick-C

Contents

- M5Stick-C
  - 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 `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:
Default protocol is `esptool`

You can change upload protocol using `upload_protocol` option:

```
[env:m5stick-c]
platform = espressif32
board = m5stick-c

upload_protocol = esptool
```

### Debugging

`PIO Unified Debugger` 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>
</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.
PlatformIO Documentation, Release 4.0.1b3

<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)*:

```
[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.

```
[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:
[env:mhetesp32devkit]
platform = espressif32
board = mhetesp32devkit
upload_protocol = esptool

**Debugging**

*PIO Unified 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).

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>
</tbody>
</table>

**MH ET LIVE ESP32MiniKit**

**Contents**

- **MH ET LIVE ESP32MiniKit**
  - 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>MH-ET Live</td>
</tr>
</tbody>
</table>

Configuration

Please use `mhetesp32minikit` ID for *board* option in “platformio.ini” *(Project Configuration File)*:

```
[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.

```
[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-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:mhetesp32minikit]
platform = espressif32
board = mhetesp32minikit
upload_protocol = esptool
```

## Debugging

**PIO Unified 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).

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>
</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*):

```
[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.
[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:

[env:nano32]
platform = espressif32
board = nano32

upload_protocol = esptool

**Debugging**

*PIO Unified Debugger* 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>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. It aims to make embedded programming easy and portable.</td>
</tr>
</tbody>
</table>

**Microduino Core 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>Microduino</td>
</tr>
</tbody>
</table>

Configuration

Please use *microduino-core-esp32* ID for *board* option in “platformio.ini” (*Project Configuration File*):

```
[env:microduino-core-esp32]
platform = espressif32
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.

```
[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:
Debugging

PIO Unified Debugger 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>
</tbody>
</table>

Node32s

Contents

- 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-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:node32s]
platform = espressif32
board = node32s

upload_protocol = esptool
```
Debugging

PIO Unified 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)**.

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>
</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
Default protocol is `esptool`

You can change upload protocol using `upload_protocol` option:

```ini
[env:nodemcu-32s]
platform = espressif32
board = nodemcu-32s
upload_protocol = esptool
```

### Debugging

**PIO Unified 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).

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>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>

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)*:

```ini
[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.

```ini
[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

`PIO Unified Debugger` 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>
</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):

```
[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.

```
[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

PIO Unified Debugger 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>
</tbody>
</table>
OLIMEX ESP32-EVB

Contents

- OLIMEX ESP32-EVB
  - 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-evb` ID for `board` option in “platformio.ini” (Project Configuration File):

```python
[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.

```python
[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-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-evb]
platform = espressif32
board = esp32-evb
upload_protocol = esptool
```

### Debugging

**PIO Unified 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).

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>
</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.
[env:esp32-gateway]
platform = espressif32
board = esp32-gateway

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L

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:

[env:esp32-gateway]
platform = espressif32
board = esp32-gateway

upload_protocol = esptool

Debugging

PIO Unified 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).
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>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>

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):

```
[env:esp32-pro]
platform = espressif32
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.

```
[env:esp32-pro]
platform = espressif32
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:

```
[env:esp32-pro]
platform = espressif32
board = esp32-pro

upload_protocol = esptool
```

Debugging

PIO Unified Debugger 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>
</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):

```
[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.

```
[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:
PlatformIO Documentation, Release 4.0.1b3

- 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

PIO Unified Debugger 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>
</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.
### Configuration

Please use oroca_edubot ID for `board` option in "platformio.ini" (Project Configuration File):

```
[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.`

```
[env:oroca_edubot]
platform = espressif32
board = oroca_edubot

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

### Uploading

OROCA EduBot supports the next uploading protocols:

- espota
- esptool

Default protocol is `esptool`

You can change upload protocol using `upload_protocol` option:

```
[env:oroca_edubot]
platform = espressif32
board = oroca_edubot

upload_protocol = esptool
```

### Debugging

PIO Unified Debugger 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>
</tbody>
</table>

Onehorse ESP32 Dev Module

Contents

- Onehorse ESP32 Dev Module
  - 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>Onehorse</td>
</tr>
</tbody>
</table>

Configuration

Please use onehorse32dev ID for board option in “platformio.ini” (Project Configuration File):

```ini
[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:

[env:onehorse32dev]
platform = espressif32
board = onehorse32dev

upload_protocol = esptool

Debugging

PIO Unified Debugger 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>
</tbody>
</table>

Pycom LoPy

Contents

- Pycom LoPy
  - Hardware
  - Configuration
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):

```ini
[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.

```ini
[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:

```plaintext
[env:lopy]
platform = espressif32
board = lopy

upload_protocol = esptool
```

### Debugging

**PIO Unified 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).

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><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>

1.12. Boards
**Pycom LoPy4**

### 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):

```ini
[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.

```ini
[env:lopy4]
platform = espressif32
board = lopy4

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

### 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:

```
[env:lopy4]
platform = espressif32
board = lopy4
upload_protocol = esptool
```

### Debugging

**PIO Unified 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).

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-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>

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:

```ini
[env:wesp32]
platform = espressif32
board = wesp32
upload_protocol = esptool
```

Debugging

*PIO Unified 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*).
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>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached 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 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):

```ini
[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.```

```ini
[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:

```ini
[env:esp32thing]
platform = espressif32
board = esp32thing

upload_protocol = esptool
```
Debugging

PIO Unified 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).

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><code>ESP-Prog</code></td>
<td></td>
<td>Yes</td>
</tr>
<tr>
<td><code>oddWires IOT-Bus JTAG</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>J-LINK</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>Mini-Module FT2232H</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>Olimex ARM-USB-OCD</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>Olimex ARM-USB-OCD-H</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>Olimex ARM-USB-TINY-H</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>Olimex ARM-USB-TINY</code></td>
<td></td>
<td></td>
</tr>
<tr>
<td><code>TIAO USB Multi-Protocol Adapter (TUMPA)</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>

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

1.12. Boards
Default protocol is `esptool`

You can change upload protocol using `upload_protocol` option:

```
[env:sparkfun_lora_gateway_1-channel]
platform = espressif32
board = sparkfun_lora_gateway_1-channel
upload_protocol = esptool
```

### Debugging

**PIO Unified 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).

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>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</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY-H</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>

### TTGO LoRa32-OLED V1
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

**PIO Unified 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).

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-OCDB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-OCDB-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>

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>
<tr>
<td>Flash</td>
<td>4MB</td>
</tr>
<tr>
<td>RAM</td>
<td>1.25MB</td>
</tr>
<tr>
<td>Vendor</td>
<td>TTGO</td>
</tr>
</tbody>
</table>

Configuration

Please use `ttgo-t-beam` ID for `board` option in "platformio.ini" (Project Configuration File):

```plaintext
[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.
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:

```
[env:ttgo-t-beam]
platform = espressif32
board = ttgo-t-beam
upload_protocol = esptool
```

Debugging

PIO Unified 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).
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>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.</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>EPS32</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):

```ini
[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.

```ini
[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:

```ini
[env:turta_iot_node]
platform = espressif32
board = turta_iot_node

upload_protocol = esptool
```

Debugging

PIO Unified Debugger 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>
</tbody>
</table>
WEMOS LOLIN D32

Contents

- WEMOS LOLIN D32
  - 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 ID for board option in “platformio.ini” (Project Configuration File):

```
[env:lolin_d32]
platform = espressif32
board = lolin_d32
```

You can override default WEMOS LOLIN D32 settings per build environment using board_*** option, where *** is a JSON object path from board manifest lolin_d32.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:lolin_d32]
platform = espressif32
board = lolin_d32

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

Uploading

WEMOS LOLIN D32 supports the next uploading protocols:
Default protocol is `esptool`

You can change upload protocol using `upload_protocol` option:

```ini
[env:lolin_d32]
platform = espressif32
board = lolin_d32
upload_protocol = esptool
```

### Debugging

**PIO Unified 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).

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>
</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>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>WEMOS</td>
</tr>
</tbody>
</table>

Configuration

Please use lolin_d32_pro ID for board option in “platformio.ini” (Project Configuration File):

```
[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.
[env:lolin_d32_pro]
platform = espressif32
board = lolin_d32_pro

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L

Uploading

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:

[env:lolin_d32_pro]
platform = espressif32
board = lolin_d32_pro

upload_protocol = esptool

Debugging

PIO Unified 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).
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>
</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

PIO Unified 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).

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>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>

WeMos WiFi and Bluetooth Battery

Contents

- WeMos WiFi and Bluetooth Battery
  - 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 *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
Default protocol is `esptool`

You can change upload protocol using `upload_protocol` option:

```
[env:wemosbat]
platform = espressif32
board = wemosbat
upload_protocol = esptool
```

### Debugging

**PIO Unified 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).

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</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Olimex ARM-USB-TINY-H</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>

### Widora AIR
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):

```python
[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.

```python
[env:widora-air]
platform = espressif32
board = widora-air

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

Uploading

Widora AIR supports the next uploading protocols:

- `espota`
- `esptool`
Default protocol is `esptool`

You can change upload protocol using `upload_protocol` option:

```yaml
[env:widora-air]
platform = espressif32
board = widora-air
upload_protocol = esptool
```

**Debugging**

*PIO Unified Debugger* 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>
</tbody>
</table>

**XinaBox CW02**

**Contents**

- XinaBox CW02
  - 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>XinaBox</td>
</tr>
</tbody>
</table>
### Configuration

Please use `xinabox_cw02` ID for `board` option in "platformio.ini" (Project Configuration File):

```ini
[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.

```ini
[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:

```ini
[env:xinabox_cw02]
platform = espressif32
board = xinabox_cw02

upload_protocol = esptool
```
Debugging

PIO Unified 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).

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>
</tbody>
</table>

oddWires IoT-Bus Io

**Contents**

- *oddWires IoT-Bus Io*
  - 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 `iotbusio` ID for `board` option in “`platformio.ini`” *(Project Configuration File)*:

```ini
[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.

```ini
[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:

```yaml
[env:iotbusio]
platform = espressif32
board = iotbusio
upload_protocol = esptool
```

**Debugging**

*PIO Unified 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).

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></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>

**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.

```
[env:iotbusproteus]
platform = espressif32
board = iotbusproteus

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

Uploading

oddWires IoT-Bus Proteus 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:iotbusproteus]
platform = espressif32
board = iotbusproteus
upload_protocol = esptool
```

### Debugging

**PIO Unified 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).

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>
</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

PIO Unified Debugger 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>
</tbody>
</table>

1.12.5 Espressif 8266

4D Systems gen4 IoD Range

Contents

• 4D Systems gen4 IoD Range
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:
Debugging

PIO Unified Debugger 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):

```ini
[env:huzzah]
platform = espressif8266
board = huzzah
```

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.

```ini
[env:huzzah]
platform = espressif8266
board = huzzah

; change microcontroller
board_build.mcu = esp8266

; change MCU frequency
board_build.f_cpu = 80000000L
```

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

PIO Unified Debugger currently does not support Adafruit HUZZAH ESP8266 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. It 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):

```ini
[env:oak]
platform = espressif8266
board = oak
```
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
[env:oak]
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
[env:oak]
platform = espressif8266
board = oak

upload_protocol = esptool
```

### Debugging

PIO Unified Debugger 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-WROOM-02

#### Contents

- `ESP-WROOM-02`
  - `Hardware`
  - `Configuration`
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>Espressif</td>
</tr>
</tbody>
</table>

Configuration

Please use `esp_wroom_02` ID for `board` option in *“platformio.ini”* (*Project Configuration File*):

```ini
[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.

```ini
[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

PIO Unified Debugger 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. It 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.
### 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:

- espota
- esptool

Default protocol is `esptool`

You can change upload protocol using `upload_protocol` option:

```
[env:espduino]
platform = espressif8266
board = espduino

upload_protocol = esptool
```

### Debugging

*PIO Unified Debugger* 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

Contents

- ESpectro Core
  - 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>Dycodex</td>
</tr>
</tbody>
</table>

Configuration

Please use **espectro ID** for **board** option in “platformio.ini” (Project Configuration File):

```ini
[env:espectro]
platform = espressif8266
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 = espressif8266
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:

```
[env:espectro]
platform = espressif8266
board = espectro

upload_protocol = esptool
```

### Debugging

PIO Unified Debugger 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 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>

### ESPino
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:

```
[env:espino]
platform = espressif8266
board = espino
upload_protocol = esptool
```

**Debugging**

*PIO Unified Debugger* currently does not support 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>

**ESPpresso Lite 1.0**

**Contents**

- ESPpresso Lite 1.0
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

**Hardware**

Platform *Epressif 8266*: Epressif 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 4.0.1b3

<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):

```ini
[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:

```ini
[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:

```ini
[env:espresso_lite_v1]
platform = espressif8266
board = espresso_lite_v1

upload_protocol = esptool
```

**Debugging**

PIO Unified Debugger currently does not support ESPresso Lite 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>

ESPresso Lite 2.0

Contents

- **ESPresso Lite 2.0**
  - 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_v2` ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:espresso_lite_v2]
platform = espressif8266
board = espresso_lite_v2
```

1.12. Boards
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 = espessif8266
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 = espessif8266
board = espresso_lite_v2

upload_protocol = esptool
```

### Debugging

`PIO Unified Debugger` currently does not support ESPRESSo Lite 2.0 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>

### 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):

```
[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.

```
[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:

```yaml
[env:esp12e]
platform = espressif8266
board = esp12e
upload_protocol = esptool
```

### Debugging

**PIO Unified Debugger** currently does not support Espressif ESP8266 ESP-12E 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. It 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.
### Configuration

Please use `esp01_1m` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```ini
[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.

```ini
[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:

```ini
[env:esp01_1m]
platform = espressif8266
board = esp01_1m

upload_protocol = esptool
```

### Debugging

*PIO Unified Debugger* 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>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached 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-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.

```yaml
[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:

```yaml
[env:esp01]
platform = espressif8266
board = esp01

upload_protocol = esptool
```

### Debugging

*PIO Unified Debugger* 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. It aims to make embedded programming easy and portable.</td>
</tr>
</tbody>
</table>
Espressif Generic ESP8266 ESP-07

Contents

• Espressif Generic ESP8266 ESP-07
  – 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.

<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 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 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 supports the next uploading protocols:
• espota
• esptool

Default protocol is esptool

You can change upload protocol using upload_protocol option:

```plaintext
[env:esp07]
platform = espressif8266
board = esp07
upload_protocol = esptool
```

Debugging

PIO Unified Debugger currently does not support Espressif Generic ESP8266 ESP-07 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>

Generic ESP8285 Module

Contents

- Generic ESP8285 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.
## Configuration

Please use `esp8285` ID for `board` option in "`platformio.ini` (Project Configuration File):

```ini
[env:esp8285]
platform = espresif8266
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.:

```ini
[env:esp8285]
platform = espresif8266
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:

```ini
[env:esp8285]
platform = espresif8266
board = esp8285

upload_protocol = esptool
```

## Debugging

`PIO Unified Debugger` currently does not support Generic ESP8285 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>

Heltec Wifi kit 8

Contents

- Heltec Wifi kit 8
  - 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>Frequency</th>
<th>Flash</th>
<th>RAM</th>
<th>Vendor</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP8266</td>
<td>80MHz</td>
<td>4MB</td>
<td>80KB</td>
<td>Heltec</td>
</tr>
</tbody>
</table>

Configuration

Please use heltec_wifi_kit_8 ID for board option in “platformio.ini” (Project Configuration File):

```
[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.

```plaintext
[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:

```plaintext
[env:heltec_wifi_kit_8]
platform = espressif8266
board = heltec_wifi_kit_8

upload_protocol = esptool
```

**Debugging**

PIO Unified Debugger 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**
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.`

```
[env:inventone]
platform = espressif8266
board = inventone

; change microcontroller
board_build.mcu = esp8266

; change MCU frequency
board_build.f_cpu = 80000000L
```

Uploading

Invent One supports the next uploading protocols:

- espota
- esptool
Default protocol is `esptool`

You can change upload protocol using `upload_protocol` option:

```yaml
[env:inventone]
platform = espressif8266
board = inventone
upload_protocol = esptool
```

**Debugging**

*PIO Unified Debugger* 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. It 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.
## Configuration

Please use `nodemcu` ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[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.

```ini
[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:

```ini
[env:nodemcu]
platform = espressif8266
board = nodemcu

upload_protocol = esptool
```

## Debugging

`PIO Unified Debugger` 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. It 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.

<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.

```yaml
[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:

```yaml
[env:nodemcuv2]
platform = espressif8266
board = nodemcuv2

upload_protocol = esptool
```

### Debugging

PIO Unified Debugger 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. It 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):

```ini
[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.

```ini
[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:

```
[env:modwifi]
platform = espessif8266
board = modwifi
upload_protocol = esptool
```

### Debugging

PIO Unified Debugger 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

#### Contents

- **Phoenix 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.
PlatformIO Documentation, Release 4.0.1b3

<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):

```plaintext
[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.

```plaintext
[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:

```plaintext
[env:phoenix_v1]
platform = espressif8266
board = phoenix_v1

upload_protocol = esptool
```

### Debugging

PIO Unified Debugger 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 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_v2 ID for board option in “platformio.ini” (Project Configuration File):

```
[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.

```
[env:phoenix_v2]
platform = espressif8266
board = phoenix_v2

; change microcontroller
board_build.mcu = esp8266

; change MCU frequency
board_build.f_cpu = 80000000L
```

**Uploading**

Phoenix 2.0 supports the next uploading protocols:

- `espota`
- `esptool`

Default protocol is `esptool`

You can change upload protocol using `upload_protocol` option:

```
[env:phoenix_v2]
platform = espressif8266
board = phoenix_v2

upload_protocol = esptool
```

**Debugging**

*PIO Unified Debugger* 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>

**SparkFun Blynk Board**
PlatformIO Documentation, Release 4.0.1b3

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:

```plaintext
[env:sparkfunBlynk]
platform = espressif8266
board = sparkfunBlynk
upload_protocol = esptool
```

## Debugging

*PIO Unified Debugger* currently does not support SparkFun Blynk Board 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>

## SparkFun ESP8266 Thing

### Contents

- SparkFun ESP8266 Thing
  - 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 `thing ID` for `board` option in "platformio.ini" (Project Configuration File):

```ini
[env:thing]
platform = espressif8266
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.

```ini
[env:thing]
platform = espressif8266
board = thing

; change microcontroller
board_build.mcu = esp8266

; change MCU frequency
board_build.f_cpu = 80000000L
```

Uploading

SparkFun ESP8266 Thing supports the next uploading protocols:

- `espota`
- `esptool`

Default protocol is `esptool`

You can change upload protocol using `upload_protocol` option:

```ini
[env:thing]
platform = espressif8266
board = thing

upload_protocol = esptool
```

Debugging

`PIO Unified Debugger` 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

Contents

- SparkFun ESP8266 Thing 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>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):

```
[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.

```yaml
[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 `esptool`

You can change upload protocol using `upload_protocol` option:

```yaml
[env:thingdev]
platform = espressif8266
board = thingdev

upload_protocol = esptool
```

### Debugging

**PIO Unified Debugger** 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
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>SweetPea</td>
</tr>
</tbody>
</table>

Configuration

Please use `esp210` ID for `board` option in "platformio.ini" *(Project Configuration File)*:

```
[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.

```
[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:

```
[env:esp210]
platform = espressif8266
board = esp210
upload_protocol = esptool
```

### Debugging

*PIO Unified Debugger* currently does not support SweetPea ESP-210 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>

### 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.
PlatformIO Documentation, Release 4.0.1b3

<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.

```
[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:

```
[env:espinotee]
platform = espressif8266
board = espinotee

upload_protocol = esptool
```

**Debugging**

`PIO Unified Debugger` 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

Contents

- WEMOS D1 R1
  - 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>WEMOS</td>
</tr>
</tbody>
</table>

Configuration

Please use d1 ID for board option in “platformio.ini” (Project Configuration File):

```
[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.
Uploading

WEMOS D1 R1 supports the next uploading protocols:

- espota
- esptool

Default protocol is esptool

You can change upload protocol using upload_protocol option:

```
[env:d1]
platform = espressif8266
board = d1

upload_protocol = esptool
```

Debugging

PIO Unified Debugger currently does not support WEMOS D1 R1 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>

WeMos D1 R2 and mini
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_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:

```
[env:d1_mini]
platform = espressif8266
board = d1_mini

upload_protocol = esptool
```

**Debugging**

*PIO Unified Debugger* 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.
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

`PIO Unified Debugger` 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>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached 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
  - 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>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
```

(continues on next page)
; 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:

```yaml
[env:d1_mini_pro]
platform = espressif8266
board = d1_mini_pro
upload_protocol = esptool
```

### Debugging

*PIO Unified Debugger* 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

- WiFi Slot
  - 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>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*):

```
[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.

```
[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:
[env:wifi_slot]
platform = espressif8266
board = wifi_slot
upload_protocol = esptool

Debugging

PIO Unified Debugger 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 wifiduino ID for `board` option in "platformio.ini" (Project Configuration File):

```
[env:wifiduino]
platform = espressif8266
board = wifiduino
```

You can override default WiFiduino settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest wifiduino.json. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:wifiduino]
platform = espressif8266
board = wifiduino

; change microcontroller
board_build.mcu = esp8266

; change MCU frequency
board_build.f_cpu = 80000000L
```

Uploading

WiFiduino supports the next uploading protocols:

- espota
- esptool

Default protocol is esptool

You can change upload protocol using `upload_protocol` option:

```
[env:wifiduino]
platform = espressif8266
board = wifiduino

upload_protocol = esptool
```

Debugging

PIO Unified Debugger 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

Contents

- 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):

```python
[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.
Uploading

WifInfo supports the next uploading protocols:

- espota
- esptool

Default protocol is esptool

You can change upload protocol using `upload_protocol` option:

```yaml
[env:wifinfo]
platform = espressif8266
board = wifinfo

upload_protocol = esptool
```

Debugging

PIO Unified Debugger 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
• **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.

![Microcontroller specifications](table)

<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)*:

```
[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.

```
[env:wio_link]
platform = espressif8266
board = wio_link

; change microcontroller
board_build.mcu = esp8266

; change MCU frequency
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:
Debugging

PIO Unified Debugger 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.

```ini
[env:wio_node]
platform = espressif8266
board = wio_node

; change microcontroller
board_build.mcu = esp8266

; change MCU frequency
board_build.f_cpu = 80000000L
```

**Uploading**

Wio Node supports the next uploading protocols:

- `espota`
- `esptool`

Default protocol is `esptool`

You can change upload protocol using `upload_protocol` option:

```ini
[env:wio_node]
platform = espressif8266
board = wio_node

upload_protocol = esptool
```

**Debugging**

PIO Unified Debugger 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):

```plaintext
[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.

```yaml
[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:

```yaml
[env:xinabox_cw01]
platform = espressif8266
board = xinabox_cw01

upload_protocol = esptool
```

**Debugging**

*PIO Unified Debugger* currently does not support XinaBox CW01 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>

**1.12.6 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.

<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 IBMEthernetKit ID for board option in "platformio.ini" (Project Configuration File):

```
[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.

```
[env:IBMEthernetKit]
platform = freescalekinetis
board = IBMEthernetKit

; change microcontroller
board_build.mcu = mk64fn1m0vll12

; change MCU frequency
board_build.f_cpu = 120000000L
```
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

PIO Unified 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).

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>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.</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

**PIO Unified 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).

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>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.</td>
</tr>
</tbody>
</table>

### Freescale Kinetis FRDM-K22F

**Contents**

• **Freescale Kinetis FRDM-K22F**
  - Hardware
  - Configuration
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):

```
[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**

*PIO Unified 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).

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><em>mbed</em></td>
<td>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.</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):

```
[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.

```
[env:frdm_k64f]
platform = freescalekinetis
board = frdm_k64f

; change microcontroller
board_build.mcu = mk64fn1m0vll12

; change MCU frequency
board_build.f_cpu = 120000000L
```

Uploading

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

PIO Unified 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**).

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>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.</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.

```ini
[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:

```ini
[env:frdm_k66f]
platform = freescalekinetis
board = frdm_k66f

upload_protocol = mbed
```

### Debugging

*PIO Unified 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).

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>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.</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):

```ini
[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.

```ini
[env:frdm_k82f]
platform = freescalekinetis
board = frdm_k82f

; change microcontroller
board_build.mcu = mk82fn256vll15

; change MCU frequency
board_build.f_cpu = 150000000L
```

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:

```ini
[env:frdm_k82f]
platform = freescalekinetis
board = frdm_k82f

upload_protocol = mbed
```

Debugging

PIO Unified 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).

Freescale Kinetis FRDM-K82F 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>mbed</td>
<td>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.</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 a 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.

```ini
[env:frdm_kl05z]
platform = freescalekinetis
board = frdm_kl05z

; change microcontroller
board_build.mcu = mkl05z32vfmd4

; change MCU frequency
board_build.f_cpu = 48000000L
```

**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:

```ini
[env:frdm_kl05z]
platform = freescalekinetis
board = frdm_kl05z

upload_protocol = mbed
```

**Debugging**

PIO Unified 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).

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>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.</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):

```
[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

; change microcontroller
board_build.mcu = mk125z128v1k4

; change MCU frequency
board_build.f_cpu = 48000000L

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:

[env:frdm_kl25z]
platform = freescalekinetis
board = frdm_kl25z
upload_protocol = mbed

Debugging

PIO Unified 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).

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>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.</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.
```ini
[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:

```ini
[env:frdm_kl27z]
platform = freescalekinetis
board = frdm_kl27z

upload_protocol = mbed
```

### Debugging

**PIO Unified 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).

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>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.</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.
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:

```
[env:frdm_kl43z]
platform = freescalekinetis
board = frdm_kl43z
upload_protocol = mbed
```

Debugging

PIO Unified 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).

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>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.</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.
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

PIO Unified 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).

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>mbed</td>
<td>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.</td>
</tr>
</tbody>
</table>

Freescale Kinetis FRDM-KL82Z

Contents

- Freescale Kinetis FRDM-KL82Z
  - 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>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):

```
[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.
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

**PIO Unified 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).

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>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.</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.
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

PIO Unified 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`).

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>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.</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.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>MKW41Z512VHT4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>48MHz</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_kw41z ID for board option in “platformio.ini” (Project Configuration File):

```
[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.
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:

```
[env:frdm_kw41z]
platform = freescalekinetis
board = frdm_kw41z
upload_protocol = mbed
```

Debugging

PIO Unified 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).

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>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.</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>

Configuration

Please use hexiwear ID for board option in “platformio.ini” (Project Configuration File):

```
[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.
[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:

[env:hexiwear]
platform = freescalekinetis
board = hexiwear

upload_protocol = mbed

### Debugging

**PIO Unified 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).

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>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.</td>
</tr>
</tbody>
</table>

1.12.7 Infineon XMC

XMC1100 Boot Kit

Contents

- XMC1100 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>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

PIO Unified 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).

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):

```
[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

*PIO Unified 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).

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
```

(continues on next page)
Debugging

PIO Unified 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).

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>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 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.
### 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

; change MCU frequency
board_build.f_cpu = 32000000L
```

### Debugging

**PIO Unified 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).

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

Contents

- XMC1300 Sense2GoL
  - 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>32KB</td>
</tr>
<tr>
<td>RAM</td>
<td>16KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Infineon</td>
</tr>
</tbody>
</table>

Configuration

Please use xmc1300_sense2go1 ID for board option in "platformio.ini" (Project Configuration File):

```
[env:xmc1300_sense2go1]
platform = infineonxmc
board = xmc1300_sense2go1
```

You can override default XMC1300 Sense2GoL settings per build environment using board_*** option, where *** is a JSON object path from board manifest xmc1300_sense2go1.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:xmc1300_sense2go1]
platform = infineonxmc
board = xmc1300_sense2go1

; change microcontroller
board_build.mcu = XMC1300
```
; change MCU frequency
board_build.f_cpu = 32000000L

Debugging

*PIO Unified 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).

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.
PlatformIO Documentation, Release 4.0.1b3

<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

; change MCU frequency
board_build.f_cpu = 48000000L
```

**Debugging**

*PIO Unified 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).

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>

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>

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

<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>

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
```

(continues on next page)
// change MCU frequency
board_build.f_cpu = 80000000L

## Debugging

**PIO Unified 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).

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>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached 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.
PlatformIO Documentation, Release 4.0.1b3

<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)*:

```
[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.

```
[env:xmc4700_relax_kit]
platform = infineonxmc
board = xmc4700_relax_kit

; change microcontroller
board_build.mcu = XMC4700

; change MCU frequency
board_build.f_cpu = 144000000L
```

### Debugging

**PIO Unified 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)*.

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>

1.12.8 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):

```python
[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.

```python
[env:genuino101]
platform = intel_arc32
board = genuino101
```

(continues on next page)
; change microcontroller
board_build.mcu = ARCy2EM

; change MCU frequency
board_build.f_cpu = 32000000L

**Debugging**

*PIO Unified Debugger* 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>

### 1.12.9 Intel MCS-51 (8051)

**Generic N79E8432**

**Contents**

- *Generic N79E8432*
  - 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>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):

```python
[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.

```python
[env:n79e8432]
platform = intel_mcs51
board = n79e8432

; change microcontroller
board_build.mcu = n79e8432

; change MCU frequency
board_build.f_cpu = 22118400L
```

Debugging

PIO Unified Debugger currently does not support Generic N79E8432 board.

Generic N79E844

**Contents**

- Generic N79E844
  - 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>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

PIO Unified Debugger 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 \texttt{n79e845} ID for \texttt{board} option in “\texttt{platformio.ini} (Project Configuration File):

```
[env:n79e845]
platform = intel_mcs51
board = n79e845
```

You can override default Generic N79E845 settings per build environment using \texttt{board\_\*\*\*} option, where \*\*\* is a JSON object path from board manifest \texttt{n79e845.json}. For example, \texttt{board\_build.mcu}, \texttt{board\_build.f\_cpu}, etc.

```
[env:n79e845]
platform = intel_mcs51
board = n79e845

; change microcontroller
board\_build.mcu = n79e845

; change MCU frequency
board\_build.f\_cpu = 22118400L
```

Debugging

\textit{PIO Unified Debugger} currently does not support Generic N79E845 board.

Generic N79E854

### Contents

- Generic N79E854
  - Hardware
  - Configuration
  - Debugging

Hardware

Platform \textit{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):

```ini
[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.

```ini
[env:n79e854]
platform = intel_mcs51
board = n79e854

; change microcontroller
board_build.mcu = n79e854

; change MCU frequency
board_build.f_cpu = 22118400L
```

Debugging

`PIO Unified Debugger` currently does not support Generic N79E854 board.

Generic N79E855

Contents

- Generic N79E855
  - 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>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

PIO Unified Debugger 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

*PIO Unified Debugger* currently does not support Generic STC15F204EA board.

Generic STC15F2K60S2

**Contents**

- Generic STC15F2K60S2
  - 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>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):

```python
[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.

```python
[env:stc15f2k60s2]
platform = intel_mcs51
board = stc15f2k60s2

; change microcontroller
board_build.mcu = stc15f2k60s2

; change MCU frequency
board_build.f_cpu = 6000000L
```

Debugging

PIO Unified Debugger currently does not support Generic STC15F2K60S2 board.

Generic STC15W204S

Contents

- Generic STC15W204S
  - 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>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>
**PlatformIO Documentation, Release 4.0.1b3**

### 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

*PIO Unified Debugger* currently does not support Generic STC15W204S board.

### Generic STC15W404AS

#### Contents

- *Generic STC15W404AS*
  - 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>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

PIO Unified Debugger currently does not support Generic STC15W404AS board.

Generic STC15W408AS

Contents

- Generic STC15W408AS
  - 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>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):

```
[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.

```
[env:stc15w408as]
platform = intel_mcs51
board = stc15w408as

; change microcontroller
board_build.mcu = stc15w408as

; change MCU frequency
board_build.f_cpu = 11059200L
```

Debugging

PIO Unified Debugger currently does not support Generic STC15W408AS board.

Generic STC89C52RC

Contents

- Generic STC89C52RC
  - 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>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):

```ini
[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.

```ini
[env:stc89c52rc]
platform = intel_mcs51
board = stc89c52rc

; change microcontroller
board_build.mcu = stc89c52rc

; change MCU frequency
board_build.f_cpu = 11059200L
```

Debugging

PIO Unified Debugger currently does not support Generic STC89C52RC board.

1.12.10 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, this platform is ported by Sipeed.
PlatformIO Documentation, Release 4.0.1b3

Microcontroller  | K210
---|---
Frequency      | 400MHz
Flash          | 16MB
RAM            | 6MB
Vendor         | Sipeed

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-oecd`
- `olimex-arm-usb-oecd-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:
PlatformIO Documentation, Release 4.0.1b3

[env:sipeed-maix-bit]
platform = kendryte210
board = sipeed-maix-bit
upload_protocol = kflash

Debugging

PIO Unified 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).

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></td>
<td>Yes</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 Standalone SDK</td>
<td>Kendryte Standalone SDK without OS support</td>
</tr>
<tr>
<td>Kendryte FreeRTOS SDK</td>
<td>Kendryte SDK with FreeRTOS support</td>
</tr>
</tbody>
</table>

Sipeed MAIX BiT with Mic
Hardware

Platform **Kendryte K210**: Kendryte K210 is an AI capable RISCV64 dual core SoC, this platform is ported by Sipeed.

<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.

```
[env:sipeed-maix-bit-mic]
platform = kendryte210
board = sipeed-maix-bit-mic

; change microcontroller
board_build.mcu = K210

; change MCU frequency
board_build.f_cpu = 400000000L
```

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

**PIO Unified 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).

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.

<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 Standalone SDK</td>
<td>Kendryte Standalone SDK without OS support</td>
</tr>
<tr>
<td>Kendryte FreeRTOS SDK</td>
<td>Kendryte SDK with FreeRTOS 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, this platform is ported by Sipeed.

![Microcontroller specifications](attachment:image)

**Microcontroller** | **K210**  | **Frequency** | **400MHz**  | **Flash** | **16MB**  | **RAM** | **6MB**  | **Vendor** | **Sipeed**  |

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.`
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:
### 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>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 Standalone SDK</td>
<td>Kendryte Standalone SDK without OS support</td>
</tr>
<tr>
<td>Kendryte FreeRTOS SDK</td>
<td>Kendryte SDK with FreeRTOS 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, this platform is ported by Sipeed.

<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):

```
[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.

```
[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

PIO Unified 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).

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>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 Standalone SDK</td>
<td>Kendryte Standalone SDK without OS support</td>
</tr>
<tr>
<td>Kendryte FreeRTOS SDK</td>
<td>Kendryte SDK with FreeRTOS 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 RISC-V64 dual core SoC, this platform is ported by Sipeed.

<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):

```ini
[env:sipeed-maixduino]
platform = kendrayte210
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.

```ini
[env:sipeed-maixduino]
platform = kendrayte210
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:

- `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-maixduino]
platform = kendryte210
board = sipeed-maixduino
upload_protocol = kflash
```

### Debugging

**PIO Unified 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).

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></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 Standalone SDK</td>
<td>Kendryte Standalone SDK without OS support</td>
</tr>
<tr>
<td>Kendryte FreeRTOS SDK</td>
<td>Kendryte SDK with FreeRTOS support</td>
</tr>
</tbody>
</table>

### 1.12.11 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>FPGAwars</td>
</tr>
</tbody>
</table>

Configuration

Please use icezum ID for board option in “platformio.ini” (Project Configuration File):

```
[env:icezum]
platform = lattice_ice40
board = icezum
```

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.

```
[env:icezum]
platform = lattice_ice40
board = icezum

; change microcontroller
board_build.mcu = ICE40-HX1K-TQ144

; change MCU frequency
board_build.f_cpu = 12000000L
```

Debugging

PIO Unified Debugger 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):

```
[env:icestick]
platform = lattice_ice40
board = icestick
```

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

PIO Unified Debugger currently does not support Lattice iCEstick FPGA Evaluation Kit board.
1.12.12 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):

```
[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.

```
[env:raspberrypi_1b]
platform = linux_arm
board = raspberrypi_1b

; change microcontroller
board_build.mcu = bcm2835

; change MCU frequency
board_build.f_cpu = 700000000L
```
Debugging

PIO Unified Debugger 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.
```python
[env:raspberrypi_2b]
platform = linux_arm
board = raspberrypi_2b

; change microcontroller
board_build.mcu = bcm2836

; change MCU frequency
board_build.f_cpu = 900000000L
```

**Debugging**

PIO Unified Debugger 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>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 3 Model B**

**Contents**

- Raspberry Pi 3 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>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

PIO Unified Debugger 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**

*PIO Unified Debugger* 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>

**RushUp Kitra 520**

**Contents**

- RushUp Kitra 520
  - 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>EXYNOS3250</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>1000MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4GB</td>
</tr>
<tr>
<td>RAM</td>
<td>512MB</td>
</tr>
<tr>
<td>Vendor</td>
<td>RushUp</td>
</tr>
</tbody>
</table>

Configuration

Please use kitra_520 ID for board option in “platformio.ini” (Project Configuration File):

```ini
[env:kitra_520]
platform = linux_arm
board = kitra_520
```

You can override default RushUp Kitra 520 settings per build environment using board_*** option, where *** is a JSON object path from board manifest kitra_520.json. For example, board_build.mcu, board_build.f_cpu, etc.

```ini
[env:kitra_520]
platform = linux_arm
board = kitra_520

; change microcontroller
board_build.mcu = exynos3250

; change MCU frequency
board_build.f_cpu = 1000000000L
```

Debugging

PIO Unified Debugger currently does not support RushUp Kitra 520 board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARTIK SDK</td>
<td>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.</td>
</tr>
</tbody>
</table>
PlatformIO Documentation, Release 4.0.1b3

Samsung ARTIK 1020

Contents

- Samsung ARTIK 1020
  - 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>EXYNOS5422</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>1500MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>16GB</td>
</tr>
<tr>
<td>RAM</td>
<td>2GB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Samsung</td>
</tr>
</tbody>
</table>

Configuration

Please use artik_1020 ID for board option in “platformio.ini” (Project Configuration File):

```
[env:artik_1020]
platform = linux_arm
board = artik_1020
```

You can override default Samsung ARTIK 1020 settings per build environment using board_*** option, where *** is a JSON object path from board manifest artik_1020.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:artik_1020]
platform = linux_arm
board = artik_1020

; change microcontroller
board_build.mcu = exynos5422

; change MCU frequency
board_build.f_cpu = 1500000000L
```

Debugging

PIO Unified Debugger currently does not support Samsung ARTIK 1020 board.
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AR-TIK SDK</td>
<td>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.</td>
</tr>
</tbody>
</table>

Samsung ARTIK 520

Contents

- Samsung ARTIK 520
  - 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>EXYNOS3250</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>1000MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4GB</td>
</tr>
<tr>
<td>RAM</td>
<td>512MB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Samsung</td>
</tr>
</tbody>
</table>

Configuration

Please use `artik_520` ID for `board` option in “`platformio.ini` (Project Configuration File):

```
[env:artik_520]
platform = linux_arm
board = artik_520
```

You can override default Samsung ARTIK 520 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `artik_520.json`. For example, `board_build.mcu, board_build.f_cpu`, etc.

```
[env:artik_520]
platform = linux_arm
board = artik_520

; change microcontroller
```

(continues on next page)
board_build.mcu = exynos3250

; change MCU frequency
board_build.f_cpu = 1000000000L

Debugging

PIO Unified Debugger currently does not support Samsung ARTIK 520 board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARTIK</td>
<td>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.</td>
</tr>
</tbody>
</table>

Samsung ARTIK 530

Contents

- Samsung ARTIK 530
  - 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>S5P4418</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>1200MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>4GB</td>
</tr>
<tr>
<td>RAM</td>
<td>512MB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Samsung</td>
</tr>
</tbody>
</table>

Configuration

Please use artik_530 ID for board option in “platformio.ini” (Project Configuration File):
You can override default Samsung ARTIK 530 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `artik_530.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```yaml
platform = linux_arm
board = artik_530

; change microcontroller
board_build.mcu = s5p4418

; change MCU frequency
board_build.f_cpu = 1200000000L
```

**Debugging**

*PIO Unified Debugger* currently does not support Samsung ARTIK 530 board.

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AR-TIK</td>
<td>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.</td>
</tr>
</tbody>
</table>

**Samsung ARTIK 710**

**Contents**

- Samsung ARTIK 710
  - 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.
### Configuration

Please use `artik_710` ID for `board` option in "platformio.ini" *(Project Configuration File)*:

```
[env:artik_710]
platform = linux_arm
board = artik_710
```

You can override default Samsung ARTIK 710 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `artik_710.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:artik_710]
platform = linux_arm
board = artik_710

; change microcontroller
board_build.mcu = s5p6818

; change MCU frequency
board_build.f_cpu = 1400000000L
```

### Debugging

*PIO Unified Debugger* currently does not support Samsung ARTIK 710 board.

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARTIK</td>
<td>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.</td>
</tr>
</tbody>
</table>

### 1.12.13 Maxim 32

MAX32620FTHR

### Contents

- MAX32620FTHR
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 max32620fthr ID for board option in "platformio.ini" (Project Configuration File):

```
[env:max32620fthr]
platform = maxim32
board = max32620fthr
```

You can override default MAX32620FTHR settings per build environment using board_*** option, where *** is a JSON object path from board manifest max32620fthr.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:max32620fthr]
platform = maxim32
board = max32620fthr

; change microcontroller
board_build.mcu = max32620fthr

; change MCU frequency
board_build.f_cpu = 96000000L
```

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

**PIO Unified 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).

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>mbed</td>
<td>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.</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

PIO Unified Debugger currently does not support MAX32625MBED board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mbed</td>
<td>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.</td>
</tr>
</tbody>
</table>
**MAX32625NEXPAQ**

## 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

*PIO Unified Debugger* currently does not support MAX32625NEXPAQ board.
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mbed</td>
<td>The mbed framework 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.</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>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 `max32625pico` ID for `board` option in `platformio.ini` (Project Configuration File):

```
[env:max32625pico]
platform = maxim32
board = max32625pico
```

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.

```
[env:max32625pico]
platform = maxim32
```

(continues on next page)
board = max32625pico

; change microcontroller
board_build.mcu = max32625

; change MCU frequency
board_build.f_cpu = 96000000L

Debugging

PIO Unified Debugger currently does not support MAX32625PICO board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mbed</td>
<td>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.</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.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>MAX32600</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 `max32600mbed` ID for `board` option in “`platformio.ini`” (Project Configuration File):

```
[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.

```
[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:

```
[env:max32600mbed]
platform = maxim32
board = max32600mbed

upload_protocol = mbed
```

Debugging

PIO Unified 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).

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.
## Compatible Tools

<table>
<thead>
<tr>
<th>CMSIS-DAP</th>
<th>On-board</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>

## Frameworks

### 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.

## 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):

```ini
[env:max32620hsp]
platform = maxim32
board = max32620hsp
```
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

**PIO Unified 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).

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>The mbed framework 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.</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):

```
[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.

```
[env:max32630fthr]
platform = maxim32
```

(continues on next page)
board = max32630fthr
; change microcontroller
board_build.mcu = max32630
; change MCU frequency
board_build.f_cpu = 96000000L

Debugging

PIO Unified Debugger currently does not support Maxim MAX32630FTHR Application Platform board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mbed</td>
<td>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.</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):

```
[env:maxwsnenv]
platform = maxim32
board = maxwsnenv
```

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.

```
[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:

```
[env:maxwsnenv]
platform = maxim32
board = maxwsnenv

upload_protocol = mbed
```

Debugging

`PIO Unified 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).

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.
Compatible Tools | On-board | Default
--- | --- | ---
CMSIS-DAP | Yes | 

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mbed</td>
<td>The mbed framework 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.</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):

```
[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.
```{env:sdt32620b}
platform = maxim32
board = sdt32620b

; change microcontroller
board_build.mcu = max32620iwg

; change MCU frequency
board_build.f_cpu = 96000000L
```

### Debugging

*PIO Unified Debugger* currently does not support SDT32620B board.

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mbed</td>
<td>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.</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):

```ini
[env:sdt32625b]
platform = maxim32
board = sdt32625b
```

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.

```ini
[env:sdt32625b]
platform = maxim32
board = sdt32625b

; change microcontroller
board_build.mcu = max32625itk

; change MCU frequency
board_build.f_cpu = 96000000L
```

Debugging

PIO Unified Debugger currently does not support SDT32625B board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mbed</td>
<td>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.</td>
</tr>
</tbody>
</table>

1.12.14 Microchip PIC32

4DSystems PICadillo 35T

Contents

- 4DSystems 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!

<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>4DSystems</td>
</tr>
</tbody>
</table>

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 4DSystems 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

*PIO Unified Debugger* currently does not support 4DSystems 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>

**Digilent Cerebot 32MX4**
PlatformIO Documentation, Release 4.0.1b3

Contents

- Digilent Cerebot 32MX4
  - 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 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

PIO Unified Debugger 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>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 cerebot32mx7 ID for board option in “platformio.ini” *(Project Configuration File)*:

```ini
[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.

```ini
[env:cerebot32mx7]
platform = microchippic32
board = cerebot32mx7

; change microcontroller
```

(continues on next page)
board_build.mcu = 32MX795F512L

; change MCU frequency
board_build.f_cpu = 80000000L

Debugging

PIO Unified Debugger currently does not support Digilent Cerebot 32MX7 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 OpenScope

Contents

- Digilent OpenScope
  - 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>32MZ2048EFG124</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 openscope ID for board option in “platformio.ini” (Project Configuration File):
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.

```plaintext
[env:openscope]
platform = microchippic32
board = openscope

; change microcontroller
board_build.mcu = 32MZ2048EFG124

; change MCU frequency
board_build.f_cpu = 200000000L
```

### Debugging

PIO Unified Debugger 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!
### Configuration

Please use `chipkit_cmod` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[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.

```
[env:chipkit_cmod]
platform = microchippic32
board = chipkit_cmod

; change microcontroller
board_build.mcu = 32MX150F128D

; change MCU frequency
board_build.f_cpu = 40000000L
```

### Debugging

**PIO Unified Debugger** 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):

```ini
[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.

```ini
[env:chipkit_dp32]
platform = microchippic32
board = chipkit_dp32

; change microcontroller
board_build.mcu = 32MX250F128B

; change MCU frequency
board_build.f_cpu = 40000000L
```

Debugging

PIO Unified Debugger 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

PIO Unified Debugger 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):

```
[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.

```
[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

PIO Unified Debugger 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.

```python
[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**

PIO Unified Debugger 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**

**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):

```
[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.

```
[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**

_*PIO Unified Debugger* 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)*:

```ini
[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.

```ini
[env:uno_pic32]
platform = microchippic32
board = uno_pic32

; change microcontroller
board_build.mcu = 32MX320F128H

; change MCU frequency
board_build.f_cpu = 80000000L
```

Debugging

**PIO Unified Debugger** 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):

```ini
[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.

```ini
[env:chipkit_wf32]
platform = microchippic32
board = chipkit_wf32

; change microcontroller
board_build.mcu = 32MX695F512L

; change MCU frequency
board_build.f_cpu = 80000000L
```

Debugging

PIO Unified Debugger 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_wifi ID for *board* option in "platformio.ini" *(Project Configuration File):*

```
[env:chipkit_wifi]
platform = microchippic32
board = chipkit_wifi
```

You can override default Digilent chipKIT WiFi settings per build environment using *board_**** option, where *** is a JSON object path from board manifest chipkit_wifi.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:chipkit_wifi]
platform = microchippic32
board = chipkit_wifi
; change microcontroller
```

(continues on next page)
board_build.mcu = 32MZ2048ECG100

; change MCU frequency
board_build.f_cpu = 200000000L

### Debugging

PIO Unified Debugger 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.

```python
[env:chipkit_uc32]
platform = microchippic32
board = chipkit_uc32

; change microcontroller
board_build.mcu = 32MX340F512H

; change MCU frequency
board_build.f_cpu = 80000000L
```

**Debugging**

*PIO Unified Debugger* 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):

```ini
[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.

```ini
[env:chipkit_pi]
platform = microchippic32
board = chipkit_pi

; change microcontroller
board_build.mcu = 32MX250F128B

; change MCU frequency
board_build.f_cpu = 40000000L
```

### Debugging

PIO Unified Debugger 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

- 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>

**Microcontroller**

**Frequency**

**Flash**

**RAM**

**Vendor**

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

PIO **Unified Debugger** 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):

```
[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.

```
[env:fubarino_sd]
platform = microchippic32
board = fubarino_sd

; change microcontroller
board_build.mcu = 32MX795F512H

; change MCU frequency
board_build.f_cpu = 80000000L
```

Debugging

PIO Unified Debugger 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>

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):

```ini
[env:clicker2]
platform = microchippic32
board = clicker2
```

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.

```ini
[env:clicker2]
platform = microchippic32
board = clicker2

; change microcontroller
```

(continues on next page)
board_build.mcu = 32MX460F512L

; change MCU frequency
board_build.f_cpu = 80000000L

### Debugging

PIO Unified Debugger 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!

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>32MZ2048EFH100</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>252MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>1.98MB</td>
</tr>
<tr>
<td>RAM</td>
<td>512KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>MikroElektronika</td>
</tr>
</tbody>
</table>

#### Configuration

Please use flipnclickmz ID for board option in “platformio.ini” (Project Configuration File):
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.

```yaml
[env:flipnclickmz]
platform = microchippic32
board = flipnclickmz

; change microcontroller
board_build.mcu = 32MZ2048EFH100

; change MCU frequency
board_build.f_cpu = 252000000L
```

### Debugging

*PIO Unified Debugger* 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>

### 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!
### Configuration

Please use pinguino32 ID for `board` option in “platformio.ini” *(Project Configuration File)*:

```ini
[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.

```ini
[env:pinguino32]
platform = microchippic32
board = pinguino32

; change microcontroller
board_build.mcu = 32MX440F256H

; change MCU frequency
board_build.f_cpu = 80000000L
```

### Debugging

*PIO Unified Debugger* 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
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>OpenBCI</td>
</tr>
</tbody>
</table>

Configuration

Please use openbc_id for board option in “platformio.ini” (Project Configuration File):

```
[env:openbc]
platform = microchippic32
board = openbc
```

You can override default OpenBCI 32bit settings per build environment using board_*** option, where *** is a JSON object path from board manifest openbc.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:openbc]
platform = microchippic32
board = openbc

; change microcontroller
board_build.mcu = 32MX250F128B

; change MCU frequency
board_build.f_cpu = 40000000L
```

Debugging

PIO Unified Debugger 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):

```
[env:usbono_pic32]
platform = microchippic32
board = usbono_pic32
```

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.

```
[env:usbono_pic32]
platform = microchippic32
board = usbono_pic32

; change microcontroller
board_build.mcu = 32MX440F512H

; change MCU frequency
board_build.f_cpu = 80000000L
```

Debugging

PIO Unified Debugger 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>

Pontech NoFire

Contents

- Pontech NoFire
  - 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>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):

```
[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.

```
[env:nofire]
platform = microchippic32
board = nofire

; change microcontroller
```

(continues on next page)
board_build.mcu = 32M2Z48F100

; change MCU frequency
board_build.f_cpu = 200000000L

Debugging

*PIO Unified Debugger* 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):
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.

```bash
[env:quick240_usb]
platform = microchippic32
board = quick240_usb

; change microcontroller
board_build.mcu = 32MX795F512L

; change MCU frequency
board_build.f_cpu = 80000000L
```

### Debugging

*PIO Unified Debugger* 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>

### SeeedStudio CUI32stem

#### Contents

- SeeedStudio CUI32stem
  - 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  |  32MX795F512H
---|---
Frequency  |  80MHz
Flash  |  508KB
RAM  |  128KB
Vendor  |  SeeedStudio

## Configuration

Please use `cui32stem` ID for `board` option in "platformio.ini" (Project Configuration File):

```ini
[env:cui32stem]
platform = microchippic32
board = cui32stem
```

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.

```ini
[env:cui32stem]
platform = microchippic32
board = cui32stem

; change microcontroller
board_build.mcu = 32MX795F512H

; change MCU frequency
board_build.f_cpu = 80000000L
```

## Debugging

*PIO Unified Debugger* 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
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>UBW32</td>
</tr>
</tbody>
</table>

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

**PIO Unified Debugger** 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
  - 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>UBW32</td>
</tr>
</tbody>
</table>

Configuration

Please use ubw32_mx795 ID for board option in “platformio.ini” (Project Configuration File):

```
[env:ubw32_mx795]
platform = microchippic32
board = ubw32_mx795
```

You can override default UBW32 MX795 settings per build environment using board_*** option, where *** is a JSON object path from board manifest ubw32_mx795.json. For example, board_build.mcu, board_build.f_cpu, etc.

```
[env:ubw32_mx795]
platform = microchippic32
board = ubw32_mx795

; change microcontroller
board_build.mcu = 32MX795F512L

; change MCU frequency
board_build.f_cpu = 80000000L
```

Debugging

PIO Unified Debugger 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

Contents

- chipKIT Lenny
  - 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>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):

```
[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.

```
[env:lenny]
platform = microchippic32
board = lenny
; change microcontroller
```

(continues on next page)
board_build.mcu = 32MX270F256D
; change MCU frequency
board_build.f_cpu = 40000000L

### Debugging

PIO Unified Debugger 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>

### 1.12.15 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

PIO Unified 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).

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

<table>
<thead>
<tr>
<th>Tool</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>
<tr>
<td>mbed</td>
<td>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.</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 BluzDK 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

BluzDK 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

*PIO Unified 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).

BluzDK does 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>

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 = nrficnrf51
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

*PIO Unified 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).

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>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>NRF51822</td>
<td>32MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>256KB</td>
</tr>
<tr>
<td>RAM</td>
<td>32KB</td>
</tr>
</tbody>
</table>

Vendor Delta

Configuration

Please use dfcm_nnn40 ID for board option in “platformio.ini” (Project Configuration File):

```ini
[env:dfcm_nnn40]
platform = nordinrf51
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 = nordinrf51
board = dfcm_nnn40
```

(continues on next page)
; change microcontroller
board_build.mcu = nrf51822

; change MCU frequency
board_build.f_cpu = 32000000L

### 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:

```ini
[env:dfcm_nnn40]
platform = nordicnrf51
board = dfcm_nnn40
upload_protocol = cmsis-dap
```

### Debugging

**PIO Unified 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).

Delta DFCM-NNN40 has on-board debug probe and **IS READY** for debugging. You don’t need to 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>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.</td>
</tr>
</tbody>
</table>
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 = nrf51822
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 = nrf51822
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

PIO Unified 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).

Delta DFCM-NNN50 has on-board debug probe and IS READY for debugging. You don’t need to 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>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.</td>
</tr>
</tbody>
</table>

JKSoft Wallbot BLE

Contents

- JKSoft Wallbot BLE

1.12. Boards
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):

```ini
[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.

```ini
[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:

```ini
[env:wallbot_ble]
platform = nordicnrf51
board = wallbot_ble
upload_protocol = cmsis-dap
```

### Debugging

**PIO Unified 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).

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

**Name**  | **Description**
--- | ---
**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.

### 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):

```ini
[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.

```ini
[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:

```ini
[env:nrf51_beacon]
platform = nordicnrf51
board = nrf51_beacon
```

(continues on next page)
upload_protocol = jlink

Debugging

**PIO Unified 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)**.

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>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td>Yes</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>

**Nordic nRF51 Dongle (PCA10031)**

**Contents**

- **Nordic nRF51 Dongle (PCA10031)**
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks
PlatformIO Documentation, Release 4.0.1b3

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)*:

```markdown
[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.

```markdown
[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:

```markdown
[env:nrf51_dongle]
platform = nordicnrf51
board = nrf51_dongle

upload_protocol = jlink
```
Debugging

PIO Unified 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).

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>Yes</td>
</tr>
<tr>
<td>J-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>mbed</td>
<td>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.</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**

PIO Unified 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).

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>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.</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):

```
[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.

```
[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:

```
[env:nrf51_dk]
platform = nordicnrf51
board = nrf51_dk

upload_protocol = jlink
```

Debugging

PIO Unified 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).

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>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td>Yes</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>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.</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

PIO Unified 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).

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):

```ini
[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**

*PIO Unified 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).”

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

<table>
<thead>
<tr>
<th></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>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.</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>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>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

PIO Unified 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).

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>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.</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

**PIO Unified 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).

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>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.</td>
</tr>
</tbody>
</table>

Seeed Arch Link

Contents

- 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

PIO Unified 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).

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>mbed</td>
<td>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.</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

**PIO Unified 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).

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>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.</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>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>sino:bit</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

*PIO Unified 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)*.

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.
## 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>
</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)*:

```ini
[env:hrm1017]
platform = nordicnrf51
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

**PIO Unified 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).

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>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mbed</td>
<td>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.</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.
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

**PIO Unified 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).

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>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.</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)*:

```
[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.
[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

*PIO Unified 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).

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>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.</td>
</tr>
</tbody>
</table>

Waveshare BLE400

Contents

- Waveshare BLE400
  - 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>Waveshare</td>
</tr>
</tbody>
</table>

Configuration

Please use `waveshare_ble400` ID for `board` option in `"platformio.ini"` (Project Configuration File):

```ini
[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.
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

PIO Unified 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)

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>
</tbody>
</table>

ng-beacon

Contents

- ng-beacon
  - 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>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
```

(continues on next page)
; 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**

*PIO Unified 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)*.

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>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>
</tbody>
</table>

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):

```ini
[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.

```ini
[env:nrf51822_y5_mbug]
platform = nordicnrf51
board = nrf51822_y5_mbug
```

(continues on next page)
### 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:

```ini
[env:nrf51822_y5_mbug]
platform = nordicnrf51
board = nrf51822_y5_mbug
upload_protocol = cmsis-dap
```

### Debugging

**PIO Unified 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)*.

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>The mbed framework 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.</td>
</tr>
</tbody>
</table>

### 1.12.16 Nordic nRF52

**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.

<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):

```
[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.
Uploading

Adafruit Bluefruit nRF52832 Feather supports the next uploading protocols:

- jlink
- nrfjprog
- nrfutil

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

PIO Unified 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).

Adafruit Bluefruit nRF52832 Feather has on-board debug probe and IS READY for debugging. You don’t need to 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>

**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):

```
[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.

```
[env:adafruit_feather_nrf52840]
platform = nordicnrf52
board = adafruit_feather_nrf52840
```
PlatformIO Documentation, Release 4.0.1b3

(continued from previous page)

```plaintext
; change microcontroller
board_build.mcu = nrf52840

; change MCU frequency
board_build.f_cpu = 64000000L
```

Uploading

Adafruit Feather nRF52840 Express supports the next uploading protocols:

- jlink
- nrfjprog
- nrfutil

Default protocol is nrfutil

You can change upload protocol using `upload_protocol` option:

```
[env:adafruit_feather_nrf52840]
platform = nordicnrf52
board = adafruit_feather_nrf52840
upload_protocol = nrfutil
```

Debugging

PIO Unified 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).

Adafruit Feather nRF52840 Express has on-board debug probe and **IS READY** for debugging. You don’t need to 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>
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.

<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 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
- jlink
- nrfjprog
- stlink

Default protocol is jlink

You can change upload protocol using `upload_protocol` option:

```python
[env:bluey]
platform = nordinf52
board = bluey
upload_protocol = jlink
```

Debugging

PIO Unified 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).

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>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
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>

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:

```yaml
[env:delta_dfbm_nq620]
platform = nordicnrf52
board = delta_dfbm_nq620
upload_protocol = cmsis-dap
```

**Debugging**

*PIO Unified 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)*.

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><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>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>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.</td>
</tr>
</tbody>
</table>
PlatformIO Documentation, Release 4.0.1b3

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>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>Nordic</td>
</tr>
</tbody>
</table>

Configuration

Please use nrf52_dk ID for board option in "platformio.ini" (Project Configuration File):

```
[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.

```
[env:nrf52_dk]
platform = nordicnrf52
board = nrf52_dk

; change microcontroller
board_build.mcu = nrf52832

; change MCU frequency
board_build.f_cpu = 64000000L
```

1.12. Boards
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:

```
[env:nrf52_dk]
platform = nordicnrf52
board = nrf52_dk

upload_protocol = jlink
```

Debugging

PIO Unified 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).

Nordic nRF52-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>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td>Yes</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>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.</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)*:

```ini
[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.

```ini
[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:

```ini
[env:nrf52840.dk]
platform = nordicnrf52
board = nrf52840.dk

upload_protocol = jlink
```

### Debugging

*PIO Unified 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).

Nordic nRF52840-DK 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 | Yes |  
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>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.</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):

```ini
[env:nrf52840_dk_adafruit]
platform = nrf52
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.

```ini
[env:nrf52840_dk_adafruit]
platform = nrf52
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:

- `jlink`
- `nrfjprog`

Default protocol is `jlink`

You can change upload protocol using `upload_protocol` option:

```ini
[env:nrf52840_dk_adafruit]
platform = nrf52
board = nrf52840_dk_adafruit

upload_protocol = jlink
```

Debugging

**PIO Unified 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).

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.
Compatible Tools | On-board | Default
--- | --- | ---
**J-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>
</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):

```ini
[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.
Uploader

RedBearLab BLE Nano 2 supports the following upload protocols:

- blackmagic
- cmsis-dap
- jlink
- nrfjprog
- stlink

Default protocol is `cmsis-dap`

You can change the upload protocol using the `upload_protocol` option:

```ini
[env:redbear_blenano2]
platform = nordicnrf52
board = redbear_blenano2

upload_protocol = cmsis-dap
```

Debugger

PIO Unified 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 further instructions and configuration information.

You can switch between debugging tools and debug probes using the `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 an 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>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.</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 = nordicnrf52
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.

```
[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:

```
[env:redbear_blend2]
platform = nordicnrf52
board = redbear_blend2

upload_protocol = cmsis-dap
```

### Debugging

**PIO Unified 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)*.

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

<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>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.</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 = nordinrf52
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**

**PIO Unified 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*).

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>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.</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):
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`
- `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

**PIO Unified 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).”

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 | 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>

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.

```ini
[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
- jlink
- nrfjprog
- stlink

Default protocol is jlink

You can change upload protocol using `upload_protocol` option:

```ini
[env:hackaBLE]
platform = nordicnrf52
board = hackaBLE

upload_protocol = jlink
```

### Debugging

**PIO Unified 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*).

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>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>

**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>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>u-blox</td>
</tr>
</tbody>
</table>

**Configuration**

Please use `ublox_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 `ublox_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)
; change microcontroller
board_build.mcu = nrf52832

; change MCU frequency
board_build.f_cpu = 64000000L

Uploading

u-blox EVK-NINA-B1 supports the next uploading protocols:

- blackmagic
- jlink
- mbed
- nrfjprog
- stlink

Default protocol is jlink

You can change upload protocol using upload_protocol option:

```
[env:ublox_evk_nina_b1]
platform = nordicnrf52
board = ublox_evk_nina_b1
upload_protocol = jlink
```

Debugging

PIO Unified 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).

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>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>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.</td>
</tr>
</tbody>
</table>

1.12.17 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.

```python
[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:

```python
[env:lpc11u24_301]
platform = nxplpc
board = lpc11u24_301

upload_protocol = mbed
```

### Debugging

**PIO Unified 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).

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>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.</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)*:

```ini
[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 = nxplpc
board = lpc4330
upload_protocol = mbed
```

Debugging

*PIO Unified 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).

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>The mbed framework 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.</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.
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:

```ini
[env:lpc11u35_501]
platform = nxplpc
board = lpc11u35_501
upload_protocol = mbed
```

Debugging

`PIO Unified 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).

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>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.</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):

```ini
[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:

```ini
[env:elektor_cocorico]
platform = nxplpc
board = elektor_cocorico

upload_protocol = mbed
```

Debugging

PIO Unified 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).

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></td>
</tr>
<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>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.</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.
DipCortex M3 supports the next uploading protocols:

- jlink
- mbed

Default protocol is mbed

You can change upload protocol using upload_protocol option:

```plaintext
[env:lpc1347]
platform = nxplpc
board = lpc1347

; change microcontroller
board_build.mcu = lpc1347

; change MCU frequency
board_build.f_cpu = 72000000L
```

### Debugging

**PIO Unified 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).

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mbed</td>
<td>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.</td>
</tr>
</tbody>
</table>

### 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.
### 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:

```ini
[env:lpc11u35]
platform = nxplpc
board = lpc11u35

upload_protocol = mbed
```

### Debugging

*PIO Unified 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).

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>mbed</td>
<td>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.</td>
</tr>
</tbody>
</table>

Embedded Artists LPC4088 Display Module

Contents

- Embedded Artists LPC4088 Display Module
  - 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_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.
### 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:

```ini
[env:lpc4088_dm]
platform = nxplpc
board = lpc4088_dm

; change microcontroller
board_build.mcu = lpc4088

; change MCU frequency
board_build.f_cpu = 120000000L
```

### Debugging

**PIO Unified 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).

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>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.</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.`
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:

```ini
[env:lpc4088]
platform = nxplpc
board = lpc4088
upload_protocol = mbed
```

Debugging

PIO Unified 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).

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>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.</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.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>LPC11U68</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>50MHz</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 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.
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

PIO Unified 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).

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>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.</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):

```
[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.
[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:

[env:lpc824]
platform = nxplpc
board = lpc824

upload_protocol = mbed

Debugging

PIO Unified 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).

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>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.</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>Frequency</th>
<th>Flash</th>
<th>RAM</th>
<th>Vendor</th>
</tr>
</thead>
<tbody>
<tr>
<td>LPC11U34</td>
<td>48MHz</td>
<td>48KB</td>
<td>10KB</td>
<td>AppNearMe</td>
</tr>
</tbody>
</table>

Configuration

Please use **micronfcboard** ID for **board** option in “platformio.ini” (Project Configuration File):

```ini
[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.
### Debugging

*PIO Unified Debugger* currently does not support MicroNFCBoard board.

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mbed</td>
<td>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.</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.
## Configuration

Please use `blueboard_lpc11u24` ID for `board` option in "platformio.ini" (Project Configuration File):

```ini
[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.

```ini
[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:

```ini
[env:blueboard_lpc11u24]
platform = nxplpc
board = blueboard_lpc11u24

upload_protocol = mbed
```

## Debugging

**PIO Unified 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).

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>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>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.</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.
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

PIO Unified 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).

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

**Name** | **Description**
--- | ---
*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.

**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):

```ini
[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.

```ini
[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:

```ini
[env:lpc11u34_421]
platform = nxplpc
board = lpc11u34_421

upload_protocol = mbed
```

**Debugging**

*PIO Unified 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).

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.

<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>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.</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.
## Configuration

Please use `lpc11u37_501` ID for `board` option in “`platformio.ini`” (Project Configuration File):

```
[env:lpc11u37_501]
platform = nxplpc
board = lpc11u37_501
```

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

**PIO Unified 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).

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>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.</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.
PlatformIO Documentation, Release 4.0.1b3

<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):

```
[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**

PIO Unified 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).

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>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.</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.
**PlatformIO Documentation, Release 4.0.1b3**

### Microcontroller

<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)*:

```ini
[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.

```ini
[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:

```ini
[env:lpc1549]
platform = nxplpc
board = lpc1549

upload_protocol = mbed
```

### Debugging

**PIO Unified 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).

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>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.</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):

```
[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.

```
[env:lpc54114]
platform = nxplpc
board = lpc54114

; change microcontroller
board_build.mcu = lpc54114j256bd64

; change MCU frequency
board_build.f_cpu = 100000000L
```

## 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

**PIO Unified 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).

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>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.</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.
## 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.

```ini
[env:lpc546xx]
platform = nxplpc
board = lpc546xx

; change microcontroller
board_build.mcu = lpc54608et512

; change MCU frequency
board_build.f_cpu = 180000000L
```

## 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

*PIO Unified 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).

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>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>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.</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

upload_protocol = mbed
```

Debugging

PIO Unified 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).
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>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.</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)*:
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

upload_protocol = mbed
```

**Debugging**

*PIO Unified 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).

NXP mbed LPC1768 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>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mbed</td>
<td>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.</td>
</tr>
</tbody>
</table>

### Outrageous Circuits mBuino

#### 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.

```yaml
[env:mbuino]
platform = nxplpc
board = mbuino

; change microcontroller
board_build.mcu = lpc11u24

; change MCU frequency
board_build.f_cpu = 48000000L
```

**Debugging**

*PIO Unified Debugger* currently does not support Outrageous Circuits mBuino board.

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mbed</td>
<td>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.</td>
</tr>
</tbody>
</table>

**Seeed Arch GPRS V2**

**Contents**

- Seeed Arch GPRS V2
  - 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.
### Microcontroller

<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 = 48000000L
```

### Debugging

PIO Unified Debugger 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>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.</td>
</tr>
</tbody>
</table>

### Seeed Arch Pro

**Contents**

- Seeed Arch Pro
  - Hardware
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.

```
[env:seeedArchPro]
platform = nxplpc
board = seeedArchPro

; change microcontroller
board_build.mcu = lpc1768

; change MCU frequency
board_build.f_cpu = 96000000L
```

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:
Debugging

PIO Unified 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).

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

**Name** | **Description**
--- | ---
**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.

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):

```
[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.

```
[env:xadow_m0]
platform = nxplpc
board = xadow_m0

; change microcontroller
board_build.mcu = lpc11u35

; change MCU frequency
board_build.f_cpu = 48000000L
```

Debugging

PIO Unified Debugger 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>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.</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)*:

```
[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.

```
[env:xbed_lpc1768]
platform = nxplpc
board = xbed_lpc1768

; change microcontroller
board_build.mcu = lpc1768

; change MCU frequency
board_build.f_cpu = 96000000L
```

### Debugging

**PIO Unified Debugger** 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>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.</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):

```ini
[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

PIO Unified 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).

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.

<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>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.</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.
**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:

```ini
[env:lpc1114fn28]
platform = nxplpc
board = lpc1114fn28

upload_protocol = mbed
```

**Debugging**

**PIO Unified 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)

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><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>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.</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 `ssci824` ID for `board` option in "platformio.ini" (Project Configuration File):

```
   [env:ssci824]
   platform = nxplpc
   board = ssci824
```

You can override default Switch Science mbed LPC824 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `ssci824.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:

```
[env:ssci824]
platform = nxplpc
board = ssci824

upload_protocol = mbed
```

Debugging

PIO Unified 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).

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>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>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.</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):

```
[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.

```
[env:oc_mbuino]
platform = nxplpc
```

(continues on next page)
board = oc_mbuino

; change microcontroller
board_build.mcu = lpc11u24

; change MCU frequency
board_build.f_cpu = 50000000L

Debugging

PIO Unified Debugger currently does not support mBuino board.

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mbed</td>
<td>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.</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

PIO Unified 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).

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>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>
</tbody>
</table>

## Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mbed</td>
<td>The mbed framework is 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.</td>
</tr>
</tbody>
</table>

## 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):
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.

```python
[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:

```python
[env:lpc11u35_y5_mbug]
platform = nxplpc
board = lpc11u35_y5_mbug

upload_protocol = mbed
```

### Debugging

**PIO Unified 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).

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>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.</td>
</tr>
</tbody>
</table>

1.12.18 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

PIO Unified 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).

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>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.</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>

1.12.19 Samsung ARTIK

Samsung ARTIK053

Contents

- Samsung ARTIK053
  - Hardware
Hardware

Platform **Samsung ARTIK**: The Samsung ARTIK Smart IoT platform brings hardware modules and cloud services together, with built-in security and an ecosystem of tools and partners to speed up your time-to-market.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>S5JT200</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>320MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>8MB</td>
</tr>
<tr>
<td>RAM</td>
<td>1.25MB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Samsung</td>
</tr>
</tbody>
</table>

Configuration

Please use `artik_053` ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[env:artik_053]
platform = samsung_artik
board = artik_053

; change microcontroller
board_build.mcu = s5jt200

; change MCU frequency
board_build.f_cpu = 320000000L
```

Debugging

*PIO Unified 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).

Samsung ARTIK053 has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.
### 1.12.20 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)*:

```ini
[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.`

<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>
[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**

*PIO Unified 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).

Arty FPGA Dev Kit 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>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></td>
<td>Yes</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>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):

```ini
[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.

```ini
[env:hifive-unleashed]
platform = sifive
board = hifive-unleashed

; change microcontroller
board_build.mcu = fu540

; change MCU frequency
board_build.f_cpu = 1500000000L
```

### Debugging

*PIO Unified 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).

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>
</tbody>
</table>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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):

```ini
[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.

```ini
[env:hifive1]
platform = sifive
board = hifive

; change microcontroller
board_build.mcu = fe310

; change MCU frequency
board_build.f_cpu = 320000000L
```

Debugging

**PIO Unified 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).

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>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Freedom E SDK</td>
<td>Open Source Software for Developing on the SiFive Freedom E Platform</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>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-revb ID for board option in “platformio.ini” (Project Configuration File):

```
[env:hifive1-revb]
platform = sifive
board = hifive-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:hifive1-revb]
platform = sifive
board = hifive-revb

; change microcontroller
board_build.mcu = fe310
```

(continues on next page)
Debugging

PIO Unified 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).

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>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Freedom E SDK</td>
<td>Open Source Software for Developing on the SiFive Freedom E Platform</td>
</tr>
</tbody>
</table>

1.12.21 Silicon Labs EFM32

EFM32GG-STK3700 Giant Gecko

Contents

- EFM32GG-STK3700 Giant 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

```cem
; change MCU frequency
board_build.f_cpu = 320000000L
```
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):

```
[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.

```
[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:

- blackmagic
- jlink
- mbed

Default protocol is `mbed`

You can change upload protocol using `upload_protocol` option:

```
[env:efm32gg_stk3700]
platform = siliconlabsefm32
board = efm32gg_stk3700

upload_protocol = mbed
```

**Debugging**

*PIO Unified 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).

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mbed</td>
<td>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.</td>
</tr>
</tbody>
</table>

EFM32LG-STK3600 Leopard 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>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:

```
[env:efm32lg_stk3600]
platform = siliconlabsefm32
board = efm32lg_stk3600

upload_protocol = mbed
```

**Debugging**

*PIO Unified 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).

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>Yes</td>
<td></td>
</tr>
<tr>
<td>J-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>mbed</td>
<td>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.</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**

*PIO Unified 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).

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>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>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.</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.
**PlatformIO Documentation, Release 4.0.1b3**

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>EFM32ZG222F32</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>24MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>32KB</td>
</tr>
<tr>
<td>RAM</td>
<td>4KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Silicon Labs</td>
</tr>
</tbody>
</table>

**Configuration**

Please use `efm32zg_stk3200` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[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.

```
[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:

```
[env:efm32zg_stk3200]
platform = siliconlabsefm32
board = efm32zg_stk3200

upload_protocol = mbed
```

**Debugging**

*PIO Unified 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).

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>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>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.</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.
**Microcontroller** | EFM32HG322F64  
---|---  
**Frequency** | 25MHz  
**Flash** | 64KB  
**RAM** | 8KB  
**Vendor** | Silicon Labs

## 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

*PIO Unified 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).

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.

<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>Yes</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mbed</td>
<td>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.</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.
Configuration

Please use efm32pg_stk3401 ID for `board` option in "platformio.ini" (Project Configuration File):

```ini
[env:efm32pg_stk3401]
platform = siliconlabsefm32
board = efm32pg_stk3401
```

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.

```ini
[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:

```ini
[env:efm32pg_stk3401]
platform = siliconlabsefm32
board = efm32pg_stk3401

upload_protocol = mbed
```

Debugging

`PIO Unified 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).

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>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.</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.
PlatformIO Documentation, Release 4.0.1b3

<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.

```
[env:tb_sense_12]
platform = siliconlabsefm32
board = tb_sense_12

; change microcontroller
board_build.mcu = EFR32MG12P432F1024

; change MCU frequency
board_build.f_cpu = 40000000L
```

## 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

**PIO Unified Debugger** - “1-click” solution for debugging with a zero configuration.
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>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.</td>
</tr>
</tbody>
</table>

### 1.12.22 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.
PlatformIO Documentation, Release 4.0.1b3

<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
- 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

PIO Unified 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).

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></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>libOpenCM3</td>
<td>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.</td>
</tr>
<tr>
<td>SPL</td>
<td>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.</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>

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):

```ini
[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.

```ini
[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
- jlink
- stlink

Default protocol is stlink

You can change upload protocol using *upload_protocol* option:

```ini
[env:disco_f723ie]
platform = ststm32
board = disco_f723ie

upload_protocol = stlink
```
Debugging

PIO Unified 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).

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>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>STM32Cube</td>
<td>STM32Cube embedded software libraries, including: The HAL hardware abstraction</td>
</tr>
<tr>
<td></td>
<td>layer, enabling portability between different STM32 devices via standardized</td>
</tr>
<tr>
<td></td>
<td>API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented</td>
</tr>
<tr>
<td></td>
<td>set of APIs designed for both performance and runtime efficiency.</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.
### 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
- jlink
- serial
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```ini
[env:armed_v1]
platform = ststm32
board = armed_v1

upload_protocol = stlink
```

### Debugging

*PIO Unified Debugger* - “1-click” solution for debugging with a zero configuration.

<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>
Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for 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>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Frameworks

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached 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>
</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):

```ini
[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.

```ini
[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
- jlink
- serial
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```ini
[env:rumba32_f446ve]
platform = ststm32
board = rumba32_f446ve

upload_protocol = stlink
```
Debugging

**PIO Unified 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).

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></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>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>
</tbody>
</table>

3D printer controller

- **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):

```
[env:remram_v1]
platform = ststm32
board = remram_v1
```

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.

```
[env:remram_v1]
platform = ststm32
board = remram_v1

; change microcontroller
board_build.mcu = ststm32f765vit6

; change MCU frequency
board_build.f_cpu = 216000000L
```

**Uploading**

3D printer controller supports the next uploading protocols:

- blackmagic
- jlink
- serial
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```
[env:remram_v1]
platform = ststm32
board = remram_v1

upload_protocol = stlink
```
Debugging

PIO Unified 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).

3D printer controller has on-board debug probe and IS READY for debugging. You don’t need to 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>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached 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>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>
</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.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F401VGT6</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 st3dp001_eval ID for board option in “platformio.ini” (Project Configuration File):

```
[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.

```
[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
- jlink
- stlink

Default protocol is stlink

You can change upload protocol using upload Protocol option:

```
[env:st3dp001_eval]
platform = ststm32
board = st3dp001_eval

upload_protocol = stlink
```

Debugging

PIO Unified 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).

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>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>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>

96Boards B96B-F446VE

Contents

- 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.
**Configuration**

Please use `b96b_f446ve` ID for `board` option in "platformio.ini" (*Project Configuration File)*:

```
[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.

```
[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
- jlink
- mbed
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```
[env:b96b_f446ve]
platform = ststm32
board = b96b_f446ve

upload_protocol = stlink
```

**Debugging**

*PIO Unified 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).

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>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>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.</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>

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.
### Configuration

Please use `armstrap_eagle1024` ID for `board` option in `platformio.ini` (Project Configuration File):

```ini
[env:armstrap_eagle1024]
platform = ststm32
board = armstrap_eagle1024
```

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
- 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

`PIO Unified 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).

Armstrap Eagle 1024 does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one 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>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>SPL</td>
<td>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.</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>

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.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F427VIT6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>168MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>1.99MB</td>
</tr>
<tr>
<td>RAM</td>
<td>256KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Armstrap</td>
</tr>
</tbody>
</table>

**Configuration**

Please use armstrap_eagle2048 ID for *board* option in "platformio.ini" (*Project Configuration File)*:

```ini
[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.

```ini
[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
- jlink
- stlink

Default protocol is blackmagic

You can change upload protocol using *upload_protocol* option:

```ini
[env:armstrap_eagle2048]
platform = ststm32
board = armstrap_eagle2048

upload_protocol = blackmagic
```

**Debugging**

*PIO Unified 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).

Armstrap Eagle 2048 does not have on-board debug probe and IS NOT READY for debugging. You will need to use/buy one 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>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>SPL</td>
<td>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.</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>

Armstrap Eagle 512

Contents

- Armstrap Eagle 512
  - 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>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
- 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**

*PIO Unified 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).

Armstrap Eagle 512 does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one 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>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>SPL</td>
<td>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.</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>
</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>192KB</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):

```ini
[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.`

```ini
[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
- jlink
- serial
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```ini
[env:black_f407ve]
platform = ststm32
board = black_f407ve

upload_protocol = stlink
```
Debugging

**PIO Unified 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).

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></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>
</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>192KB</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`
- `jlink`
- `serial`
- `stlink`

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```
[env:black_f407vg]
platform = ststm32
board = black_f407vg
```

(continues on next page)
upload_protocol = stlink

Debugging

**PIO Unified 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).

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></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>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>
</tbody>
</table>

Black STM32F407ZE

Contents

- **Black STM32F407ZE**
  - 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>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>192KB</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):

```ini
[env:black_f407zg]
platform = ststm32
board = black_f407zg
```

You can override default Black STM32F407ZE 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.`

```ini
[env:black_f407zg]
platform = ststm32
board = black_f407zg

; change microcontroller
board_build.mcu = stm32f407zgt6

; change MCU frequency
board_build.f_cpu = 168000000L
```

**Uploading**

Black STM32F407ZE supports the next uploading protocols:

- blackmagic
- jlink
- serial
- stlink

Default protocol is stlink

You can change upload protocol using `upload_protocol` option:
### Debugging

**PIO Unified 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).

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></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>
</tbody>
</table>

### Black STM32F407ZE

**Contents**

- Black STM32F407ZE
  - 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>STM32F407ZET6</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>ST</td>
</tr>
</tbody>
</table>

Configuration

Please use `black_f407ze` ID for `board` option in "`platformio.ini` (Project Configuration File):

```
[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.

```
[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
- 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

**PIO Unified 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)*.

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>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>

### BlackPill F103C8

**Contents**

- 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):

```
[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.`

```
[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
- jlink
- serial
• stlink

Default protocol is stlink

You can change upload protocol using `upload_protocol` option:

```plaintext
[env:blackpill_f103c8]
platform = ststm32
board = blackpill_f103c8
upload_protocol = stlink
```

### Debugging

**PIO Unified 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).

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

Default protocol is stlink

You can change upload protocol using `upload_protocol` option:

```yaml
[env:blackpill_f103c8_128]
platform = ststm32
board = blackpill_f103c8_128
upload_protocol = stlink
```

Debugging

*PIO Unified 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).

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>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>libOpenCM3</td>
<td>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.</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>

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*):
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.

```ini
[env:robotdyn_blackpill_f303cc]
platform = stm32
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
- jlink
- stlink

Default protocol is stlink

You can change upload protocol using `upload_protocol` option:

```ini
[env:robotdyn_blackpill_f303cc]
platform = stm32
board = robotdyn_blackpill_f303cc

upload_protocol = stlink
```

### Debugging

**PIO Unified 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).

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.
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>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>

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>192KB</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
- 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

**PIO Unified 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)*.

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.
<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>
</tbody>
</table>

### Blue Pill F103C6

- Blue Pill F103C6
  - 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>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):

```
[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.

```
[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`
- `jlink`
- `serial`
- `stlink`

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```
[env:bluepill_f103c6]
platform = ststm32
board = bluepill_f103c6

upload_protocol = stlink
```

Debugging

*PIO Unified 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).”
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>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>
</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
- dfu
- jlink
- mbed
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```ini
[env:bluepill_f103c8]
platform = ststm32
board = bluepill_f103c8

upload_protocol = stlink
```

Debugging

PIO Unified 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).

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>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>libOpenCM3</td>
<td>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.</td>
</tr>
<tr>
<td>mbed</td>
<td>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.</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>

BluePill F103C8 (128k)

Contents

- BluePill F103C8 (128k)
  - 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>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
- dfu
- jlink
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:
[env:bluepill_f103c8_128k]
platform = stm32
board = bluepill_f103c8_128k
upload_protocol = stlink

Debugging

PIO Unified 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).

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></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>libOpenCM3</td>
<td>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.</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>

Demo F030F4
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):

```plaintext
[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.

```plaintext
[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
- jlink
- serial
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```ini
[env:demo_f030f4]
platform = ststm32
board = demo_f030f4
upload_protocol = stlink
```

### Debugging

**PIO Unified 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).

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>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>

### Esportel LoRa Module

**Contents**

- Esportel 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):

```ini
[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.

```ini
[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
- 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

**PIO Unified 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).

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>J-LINK</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ST-LINK</td>
<td></td>
<td>Yes</td>
</tr>
</tbody>
</table>

### Frameworks

**Name** | **Description**
---|---
**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.

**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.

### F407VG

**Contents**

- **F407VG**
  - 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>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>192KB</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):

```ini
[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.

```ini
[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
- jlink
- serial
- stlink
Default protocol is `stlink`.

You can change upload protocol using `upload_protocol` option:

```
[env:diymore_f407vgt]
platform = stm32
board = diymore_f407vgt
upload_protocol = stlink
```

### Debugging

**PIO Unified 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).

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>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>

### FK407M1

**Contents**

- **FK407M1**
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>Generic</td>
</tr>
</tbody>
</table>

Configuration

Please use `fk407m1` ID for *board* option in “platformio.ini” (*Project Configuration File*):

```
[env:fk407m1]
platform = ststm32
board = fk407m1
```

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.

```
[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`
- `jlink`
- `serial`
• stlink

Default protocol is stlink

You can change upload protocol using upload_protocol option:

```
[env:fk407m1]
platform = stm32
board = fk407m1
upload_protocol = stlink
```

## Debugging

**PIO Unified 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).

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.

<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>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>
</tbody>
</table>

L476DMW1K
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
upload_protocol = cmsis-dap
```

**Debugging**

PIO Unified 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).

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>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>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.</td>
</tr>
<tr>
<td>STM32</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>
</tbody>
</table>
M200 V2

Contents

- M200 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>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
- jlink
- serial
- stlink

Default protocol is stlink

You can change upload protocol using `upload_protocol` option:

```ini
[env:malyanm200_f070cb]
platform = stm32
board = malyanm200_f070cb
upload_protocol = stlink
```

Debugging

PIO Unified 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).

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></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>
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):

```ini
[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
- 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

**PIO Unified 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).

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.

<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>mbed</td>
<td>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.</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>
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):

```ini
[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.

```ini
[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
- jlink
- serial
- stlink

Default protocol is stlink

You can change upload protocol using `upload_protocol` option:

```plaintext
[env:malyanm200_f103cb]
platform = stm32
board = malyanm200_f103cb
upload_protocol = stlink
```

Debugging

PIO Unified 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).

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></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>
Maple

Contents

- Maple
  - 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>108KB</td>
</tr>
<tr>
<td>RAM</td>
<td>17KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>LeafLabs</td>
</tr>
</tbody>
</table>

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 = stm32f103rkt6

; change MCU frequency
board_build.f_cpu = 72000000L
```
Uploading

Maple supports the next uploading protocols:

- blackmagic
- 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

**PIO Unified 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).

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></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>libOpenCM3</td>
<td>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.</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>
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):

```ini
[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.

```ini
[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
- dfu
- jlink
- stlink

Default protocol is dfu

You can change upload protocol using `upload_protocol` option:

```ini
[env:maple_ret6]
platform = stm32
board = maple_ret6
upload_protocol = dfu
```

Debugging

PIO Unified 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).

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>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>libOpenCM3</td>
<td>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.</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>
Maple Mini Bootloader 2.0

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
- dfu
- jlink
- stlink

Default protocol is dfu

You can change upload protocol using upload_protocol option:

```plaintext
[env:maple_mini_b20]
platform = stm32
board = maple_mini_b20
upload_protocol = dfu
```

Debugging

**PIO Unified 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).

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>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>libOpenCM3</td>
<td>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.</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>
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>STM32F103CBT6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>72MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>108KB</td>
</tr>
<tr>
<td>RAM</td>
<td>17KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>LeafLabs</td>
</tr>
</tbody>
</table>

Configuration

Please use `maple_mini_origin` ID for `board` option in “platformio.ini” (*Project Configuration File)*:

```
[env:maple_mini_origin]
platform = ststm32
board = maple_mini_origin
```

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.

```
[env:maple_mini_origin]
platform = ststm32
board = maple_mini_origin

; change microcontroller
board_build.mcu = stm32f103cbt6

; change MCU frequency
board_build.f_cpu = 72000000L
```
Uploading

Maple Mini Original 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:maple_mini_origin]
platform = ststm32
board = maple_mini_origin
upload_protocol = dfu
```

Debugging

**PIO Unified 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).

Maple Mini Original 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>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>libOpenCM3</td>
<td>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.</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>

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.

<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 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 = stm32f439ziy6

; 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

**PIO Unified 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).

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>The mbed framework 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.</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>

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):

```
[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
- 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
```

### Debugging

**PIO Unified 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).

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>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>

Microsoft Azure IoT Development Kit (MXChip AZ3166)

Contents

- Microsoft Azure IoT Development Kit (MXChip AZ3166)
  - 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>MXChip</td>
</tr>
</tbody>
</table>

Configuration

Please use `mxchip_az3166` ID for `board` option in "platformio.ini" (Project Configuration File):

```
[env:mxchip_az3166]
platform = ststm32
board = mxchip_az3166
```

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.`
Uploading

Microsoft Azure IoT Development Kit (MXChip AZ3166) supports the next uploading protocols:

- blackmagic
- jlink
- stlink

Default protocol is stlink

You can change upload protocol using `upload_protocol` option:

```
[env:mxchip_az3166]
platform = ststm32
board = mxchip_az3166
upload_protocol = stlink
```

Debugging

PIO Unified 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).

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>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>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>

MultiTech mDot

Contents

- MultiTech mDot
  - 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_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.
Uploading

MultiTech mDot supports the next uploading protocols:

- blackmagic
- 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

PIO Unified 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).

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>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>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.</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>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>

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
- 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

**PIO Unified 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).

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>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>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.</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>
</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_1151cc ID for board option in “platformio.ini” (Project Configuration File):

```ini
[env:xdot_1151cc]
platform = ststm32
board = xdot_1151cc
```
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.

```ini
[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`
- `jlink`
- `stlink`

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```ini
[env:xdot_l151cc]
platform = ststm32
board = xdot_l151cc

upload_protocol = stlink
```

## Debugging

**PIO Unified 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).

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>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>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.</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>

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):

```ini
[env:netduino2plus]
platform = ststm32
board = netduino2plus
```
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.

```
[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:

```
[env:netduino2plus]
platform = ststm32
board = netduino2plus

upload_protocol = dfu
```

### Debugging

PIO Unified 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).

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>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>

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):

```
[env:mote_l152rc]
platform = ststm32
board = mote_l152rc
```

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_1152rc]
platform = ststm32
board = mote_1152rc

; change microcontroller
board_build.mcu = stm32l152rc

; change MCU frequency
board_build.f_cpu = 32000000L

**Uploading**

NAMote72 supports the next uploading protocols:

- blackmagic
- jlink
- mbed
- stlink

Default protocol is mbed

You can change upload protocol using `upload_protocol` option:

```ini
[env:mote_1152rc]
platform = ststm32
board = mote_1152rc

upload_protocol = mbed
```

**Debugging**

**PIO Unified 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).

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.

<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>mbed</td>
<td>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.</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>

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.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32G071RBT6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>24MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>2MB</td>
</tr>
<tr>
<td>RAM</td>
<td>128KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>ST</td>
</tr>
</tbody>
</table>

Configuration

Please use nucleo_g071rb ID for board option in “platformio.ini” (Project Configuration File):

```ini
[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.

```ini
[env:nucleo_g071rb]
platform = ststm32
board = nucleo_g071rb

# change microcontroller
board_build.mcu = stm32g071rbt6

# change MCU frequency
board_build.f_cpu = 24000000L
```

### Uploading

Nucleo G071RB supports the next uploading protocols:

- blackmagic
- jlink
- mbed
- stlink

Default protocol is stlink

You can change upload protocol using upload_protocol option:

```ini
[env:nucleo_g071rb]
platform = ststm32
board = nucleo_g071rb
upload_protocol = stlink
```

### Debugging

**PIO Unified 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).

Nucleo G071RB has on-board debug probe and **IS READY** for debugging. You don’t need to 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>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.</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):

```ini
[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.
P-Nucleo WB55RG supports the next uploading protocols:

- blackmagic
- 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**

*PIO Unified 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*).

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>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>

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>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):

```
[env:rak811_tracker_32]
platform = stm32
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.
Uploading

RAK811 LoRa Tracker supports the next uploading protocols:

- blackmagic
- jlink
- serial
- stlink

Default protocol is stlink

You can change upload protocol using `upload_protocol` option:

```ini
[env:rak811_tracker_32]
platform = stm32
board = rak811_tracker_32
upload_protocol = stlink
```

Debugging

PIO Unified 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).

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>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>

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):

```ini
[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
- jlink
- serial
- stlink

Default protocol is stlink

You can change upload protocol using upload_protocol option:

[env:rak811_tracker]
platform = ststm32
board = rak811_tracker

upload_protocol = stlink

Debugging

PIO Unified 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).

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>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>

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):

```
[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.

1.12. Boards
Uploading

RushUp Cloud-JAM supports the next uploading protocols:

- blackmagic
- jlink
- mbed
- stlink

Default protocol is stlink

You can change upload protocol using `upload_protocol` option:

```ini
[env:cloud_jam]
platform = stm32
board = cloud_jam

upload_protocol = stlink
```

Debugging

PIO Unified 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).

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>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>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.</td>
</tr>
<tr>
<td>SPL</td>
<td>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.</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>

RushUp Cloud-JAM L4

Contents

- RushUp Cloud-JAM L4
  - 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>RushUp</td>
</tr>
</tbody>
</table>
**Configuration**

Please use `cloud_jam_l4` ID for `board` option in "platformio.ini" (Project Configuration File):

```
[env:cloud_jam_l4]
platform = stm32
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.

```
[env:cloud_jam_l4]
platform = stm32
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
- jlink
- mbed
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```
[env:cloud_jam_l4]
platform = stm32
board = cloud_jam_l4

upload_protocol = stlink
```

**Debugging**

PIO Unified 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).
RushUp Cloud-JAM L4 has on-board debug probe and **IS READY** for debugging. You don’t need to 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><em>mbed</em></td>
<td>The mbed framework is 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.</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 32F3348DISCOVERY**

### Contents

- *ST 32F3348DISCOVERY*
  - 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>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
- 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

`PIO Unified 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).
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>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>mbed</strong></td>
<td>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.</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>
</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><strong>ST</strong></td>
</tr>
</tbody>
</table>
Configuration

Please use `disco_f401vc` ID for `board` option in "platformio.ini" (Project Configuration File):

```ini
[env:disco_f401vc]
platform = stm32
board = disco_f401vc
```

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 = stm32
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
- jlink
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```ini
[env:disco_f401vc]
platform = stm32
board = disco_f401vc

upload_protocol = stlink
```

Debugging

PIO Unified 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).

ST 32F401CDISCOVERY 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 |  |  
ST-LINK | Yes | Yes

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mbed</td>
<td>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.</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 32F411EDISCOVERY

Contents

- 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.

<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):

```
[env:disco_f411ve]
platform = stm32
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.

```
[env:disco_f411ve]
platform = stm32
board = disco_f411ve

; change microcontroller
board_build.mcu = stm32f411vet6

; change MCU frequency
board_build.f_cpu = 10000000L
```

Uploading

ST 32F411EDISCOVERY supports the next uploading protocols:

- blackmagic
- jlink
- stlink

Default protocol is stlink

You can change upload protocol using upload_protocol option:

```
[env:disco_f411ve]
platform = stm32
board = disco_f411ve

upload_protocol = stlink
```

Debugging

PIO Unified 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).

ST 32F411EDISCOVERY 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>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>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 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):

```lua
[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.

```yaml
[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
- jlink
- mbed
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```yaml
[env:disco_f413zh]
platform = ststm32
board = disco_f413zh

upload_protocol = stlink
```

### Debugging

**PIO Unified 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).

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>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>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.</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 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
- 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**

*PIO Unified 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).

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>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>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.</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 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.

```plaintext
[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
- jlink
- mbed
- stlink

Default protocol is stlink

You can change upload protocol using `upload_protocol` option:

```plaintext
[env:disco_f469ni]
platform = ststm32
board = disco_f469ni

upload_protocol = stlink
```

### Debugging

**PIO Unified 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*).

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>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>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.</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 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):

```ini
[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.

```ini
[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
- jlink
- mbed
- stlink

Default protocol is `stlink`.

You can change upload protocol using `upload_protocol` option:

```ini
[env:disco_f746ng]
platform = ststm32
board = disco_f746ng

upload_protocol = stlink
```

### Debugging

**PIO Unified 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).

ST 32F746GDISCOVERY has on-board debug probe and **IS READY** for debugging. You don’t need to 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>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached 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>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.</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 32F769IDISCOVERY

Contents

- ST 32F769IDISCOVERY
  - Chrome
  - 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>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):
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 = stm32
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
- jlink
- mbed
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```ini
[env:disco_f769ni]
platform = stm32
board = disco_f769ni

upload_protocol = stlink
```

**Debugging**

*PIO Unified 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)*.

ST 32F769IDISCOVERY 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></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>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.</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 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
- 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

PIO Unified 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).
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>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>The mbed framework is 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.</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 32L100DISCOVERY

#### Contents

- **ST 32L100DISCOVERY**
  - 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>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):

```
[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.

```
[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:

```
[env:disco_l100rc]
platform = ststm32
board = disco_l100rc

upload_protocol = stlink
```

Debugging

**PIO Unified 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).

ST 32L100DISCOVERY 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></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>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 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):

```ini
[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
- 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

**PIO Unified 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).

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>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>The mbed framework 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.</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 32L496GDISCOVERY

Contents

- ST 32L496GDISCOVERY
  - 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>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):

```ini
[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.

```ini
[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
- jlink
- mbed
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```ini
[env:disco_l496ag]
platform = ststm32
board = disco_l496ag

upload_protocol = stlink
```

### Debugging

**PIO Unified 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).

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>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>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.</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>
</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)*:

```bash
[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.

```
[env:disco_1072cz_lrwan1]
platform = ststm32
board = disco_1072cz_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
- jlink
- mbed
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```
[env:disco_1072cz_lrwan1]
platform = ststm32
board = disco_1072cz_lrwan1

upload_protocol = stlink
```

### Debugging

**PIO Unified 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).

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>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>mbed</td>
<td>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.</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 DISCO-L475VG-IOT01A

Contents

- **ST DISCO-L475VG-IOT01A**
  - 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>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):
You can override default ST DISCO-L475VG-IOT01A 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.

```
[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 DISCO-L475VG-IOT01A supports the next uploading protocols:

- blackmagic
- 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**

*PIO Unified 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)*.

ST DISCO-L475VG-IOT01A 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>
<tr>
<td>mbed</td>
<td>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.</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 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 = stm32
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 = stm32
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
- jlink
- stlink

Default protocol is stlink

You can change upload protocol using upload_protocol option:

```
[env:disco_f072rb]
platform = stm32
board = disco_f072rb

upload_protocol = stlink
```

Debugging

PIO Unified 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).

ST Discovery F072RB 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></td>
<td></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>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached 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>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.</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 F030R8

#### 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>

#### Contents

- **ST Nucleo F030R8**
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks
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
- 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

PIO Unified 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).
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>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>mbed</td>
<td>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.</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 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.
Microcontroller | STM32F031K6T6  
---|---  
Frequency | 48MHz  
Flash | 32KB  
RAM | 4KB  
Vendor | ST

**Configuration**

Please use **nucleo_f031k6** ID for `board` option in "platformio.ini" *(Project Configuration File)*:

```
[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.

```
[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
- jlink
- mbed
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```
[env:nucleo_f031k6]
platform = ststm32
board = nucleo_f031k6

upload_protocol = stlink
```

**Debugging**

`PIO Unified 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).

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>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>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.</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 F042K6

Contents

- **ST Nucleo F042K6**
  - 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 4.0.1b3

<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):

```ini
[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.

```ini
[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
- jlink
- mbed
- stlink

Default protocol is stlink

You can change upload protocol using `upload_protocol` option:

```ini
[env:nucleo_f042k6]
platform = ststm32
board = nucleo_f042k6

upload_protocol = stlink
```

**Debugging**

*PIO Unified 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).

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>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>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.</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 F070RB**

**Contents**

- **ST Nucleo F070RB**
  - 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 4.0.1b3

<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
- jlink
- mbed
- stlink

Default protocol is stlink

You can change upload protocol using upload_protocol option:

```ini
[env:nucleo_f070rb]
platform = ststm32
board = nucleo_f070rb

upload_protocol = stlink
```

**Debugging**

*PIO Unified 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).

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>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>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.</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 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.
## Configuration

Please use `nucleo_f072rb` ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[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.

```ini
[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
- jlink
- mbed
- stlink

Default protocol is stlink

You can change upload protocol using `upload_protocol` option:

```ini
[env:nucleo_f072rb]
platform = ststm32
board = nucleo_f072rb

upload_protocol = stlink
```

## Debugging

PIO Unified 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).

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>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>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.</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 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.
**Microcontroller** | STM32F091RCT6  
---|---  
**Frequency** | 48MHz  
**Flash** | 256KB  
**RAM** | 32KB  
**Vendor** | ST

### 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
- 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

*PIO Unified 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).

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>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>mbed</td>
<td>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.</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 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):

```
[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.

```
[env:nucleo_f103rb]
platform = ststm32
board = nucleo_f103rb

; change microcontroller
board_build.mcu = stm32f103rbt6

; change MCU frequency
board_build.f_cpu = 72000000L
```

Uploading

ST Nucleo F103RB supports the next uploading protocols:

- blackmagic
- jlink
- mbed
- stlink

Default protocol is stlink

You can change upload protocol using upload_protocol option:

```
[env:nucleo_f103rb]
platform = ststm32
board = nucleo_f103rb

upload_protocol = stlink
```
Debugging

PIO Unified 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).

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>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>libOpenCM3</td>
<td>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.</td>
</tr>
<tr>
<td>mbed</td>
<td>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.</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 F207ZG

Contents

- ST Nucleo F207ZG
  - 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>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)*:

```ini
[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.

```ini
[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
- jlink
- mbed
- stlink

Default protocol is stlink

You can change upload protocol using `upload_protocol` option:
Debugging

PIO Unified 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).

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>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>mbed</td>
<td>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.</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 F302R8

Contents

- ST Nucleo F302R8
  - 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>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
- 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

**PIO Unified 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).

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>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>mbed</td>
<td>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.</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 F303K8

**Contents**

- *ST Nucleo F303K8*
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>16KB</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
- jlink
- mbed
• stlink

Default protocol is stlink

You can change upload protocol using upload_protocol option:

```python
[env:nucleo_f303k8]
platform = ststm32
board = nucleo_f303k8
upload_protocol = stlink
```

**Debugging**

**PIO Unified 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).

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>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>mbed</td>
<td>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.</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 F303RE**
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 = stm32
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 = stm32
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
• jlink
• mbed
• stlink

Default protocol is stlink

You can change upload protocol using `upload_protocol` option:

```ini
[env:nucleo_f303re]
platform = ststm32
board = nucleo_f303re

upload_protocol = stlink
```

## Debugging

**PIO Unified 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).

ST Nucleo F303RE 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></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>mbed</td>
<td>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.</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 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):

```
[env: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.

```
[env: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
- jlink
- mbed
- stlink

Default protocol is stlink

You can change upload protocol using `upload_protocol` option:

```
[env:nucleo_f303ze]
platform = stm32
board = nucleo_f303ze
upload_protocol = stlink
```

Debugging

PIO Unified 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).

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>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>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.</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 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 = ststm32
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 = ststm32
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
- jlink
- mbed
- stlink

Default protocol is stlink

You can change upload protocol using `upload_protocol` option:

```plaintext
[env:nucleo_f334r8]
platform = ststm32
board = nucleo_f334r8
upload_protocol = stlink
```

Debugging

**PIO Unified 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).

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>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>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.</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 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):

```
[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.

```
[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
- jlink
- mbed
- stlink

Default protocol is stlink

You can change upload protocol using `upload_protocol` option:

```
[env:nucleo_f401re]
platform = ststm32
board = nucleo_f401re
upload_protocol = stlink
```

Debugging

*PIO Unified 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).

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>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>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.</td>
</tr>
<tr>
<td>SPL</td>
<td>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.</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>
</tbody>
</table>

ST Nucleo F410RB

Contents

- ST Nucleo F410RB
  - 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>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
- 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

PIO Unified 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).
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>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>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.</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 F411RE**

**Contents**

- **ST Nucleo F411RE**
  - 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>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
- jlink
- mbed
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```
[env:nucleo_f411re]
platform = ststm32
board = nucleo_f411re

upload_protocol = stlink
```

Debugging

PIO Unified 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).
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>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>mbed</td>
<td>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.</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 F412ZG**

**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_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
- 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

PIO Unified Debugger - “1-click” solution for debugging with a zero configuration.
You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for 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>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>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.</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>

### ST Nucleo F413ZH

<table>
<thead>
<tr>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>- <strong>ST Nucleo F413ZH</strong></td>
</tr>
<tr>
<td>- Hardware</td>
</tr>
<tr>
<td>- Configuration</td>
</tr>
<tr>
<td>- Uploading</td>
</tr>
<tr>
<td>- Debugging</td>
</tr>
<tr>
<td>- Frameworks</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.
**Microcontroller** | STM32F413ZHT6  
---|---  
**Frequency** | 100MHz  
**Flash** | 512KB  
**RAM** | 128KB  
**Vendor** | ST  

### Configuration

Please use `nucleo_f413zh` ID for `board` option in "platformio.ini" (Project Configuration File):

```
[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`.

```
[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
- jlink
- mbed
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```
[env:nucleo_f413zh]
platform = ststm32
board = nucleo_f413zh

upload_protocol = stlink
```

### Debugging

**PIO Unified 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).

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>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>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.</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>

ST Nucleo F429ZI

Contents

- ST Nucleo F429ZI
  - 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 4.0.1b3

<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
- jlink
- mbed
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```
[env:nucleo_f429zi]
platform = ststm32
board = nucleo_f429zi

upload_protocol = stlink
```

### Debugging

PIO Unified 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*).

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>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>mbed</td>
<td>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.</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 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
- 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

**PIO Unified 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).

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>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>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.</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>

ST Nucleo F446RE

Contents

- **ST Nucleo F446RE**
  - 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>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
- 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
```

(continues on next page)
upload_protocol = stlink

**Debugging**

PIO Unified 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).

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>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>mbed</td>
<td>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.</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 F446ZE**

**Contents**

- **ST Nucleo F446ZE**
  - **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>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)**:

```
[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.`

```
[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
- jlink
- mbed
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:
[env:nucleo_f446ze]
platform = stm32
board = nucleo_f446ze
upload_protocol = stlink

Debugging

PIO Unified 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).

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>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>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.</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 F722ZE

Contents

- ST Nucleo F722ZE
  - 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>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)*:

```ini
[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.*

```ini
[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
- jlink
- mbed
- stlink

Default protocol is stlink

You can change upload protocol using *upload_protocol* option:
[env:nucleo_f722ze]
platform = stm32
board = nucleo_f722ze
upload_protocol = stlink

Debugging

PIO Unified 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).

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>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>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>

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):**

```
[env:nucleo_f746zg]
platform = ststm32
board = nucleo_f746zg
```

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.

```
[env:nucleo_f746zg]
platform = ststm32
board = nucleo_f746zg

; change microcontroller
board_build.mcu = stm32f746zgt6

; change MCU frequency
board_build.f_cpu = 216000000L
```

Uploading

ST Nucleo F746ZG supports the next uploading protocols:

- blackmagic
- jlink
- mbed
- stlink

Default protocol is stlink

You can change upload protocol using `upload_protocol` option:

```
[env:nucleo_f746zg]
platform = ststm32
board = nucleo_f746zg
```

(continues on next page)
upload_protocol = stlink

Debugging

PIO Unified 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).

ST Nucleo F746ZG has on-board debug probe and **IS READY** for debugging. You don’t need to 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>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.</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 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
- jlink
- mbed
- stlink

Default protocol is stlink

You can change upload protocol using upload_protocol option:

```
[env:nucleo_f756zg]
platform = ststm32
board = nucleo_f756zg
```

(continues on next page)
upload_protocol = stlink

Debugging

PIO Unified 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).

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>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>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.</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 F767ZI

Contents

- ST Nucleo F767ZI
  - 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>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
- 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
```

(continues on next page)
Debugging

PIO Unified 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).

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>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>mbed</td>
<td>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.</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>
</tbody>
</table>

ST Nucleo H743ZI

**Contents**

- **ST Nucleo H743ZI**
  - 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>STM32H743ZIT6</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>1MB</td>
</tr>
<tr>
<td>Vendor</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
- jlink
- mbed
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:
[env:nucleo_h743zi]
platform = ststm32
board = nucleo_h743zi
upload_protocol = stlink

Debugging

PIO Unified 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).

ST Nucleo H743ZI 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           |            |            |
| 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>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 L011K4

Contents

- ST Nucleo L011K4
  - 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>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
- jlink
- stlink

Default protocol is stlink

You can change upload protocol using `upload_protocol` option:

```ini
[env:nucleo_l011k4]
platform = ststm32
board = nucleo_l011k4

upload_protocol = stlink
```
Debugging

PIO Unified 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).

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>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>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 L031K6

Contents

- **ST Nucleo L031K6**
  - 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_l031k6` ID for `board` option in "platformio.ini" (Project Configuration File):

```ini
[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.

```ini
[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
- jlink
- mbed
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```ini
[env:nucleo_l031k6]
platform = ststm32
board = nucleo_l031k6

upload_protocol = stlink
```

### Debugging

*PIO Unified 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).

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>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>mbed</td>
<td>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.</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 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 = ststm32
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 = ststm32
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
- jlink
- mbed
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```ini
[env:nucleo_l053r8]
platform = ststm32
board = nucleo_l053r8

upload_protocol = stlink
```
Debugging

PIO Unified 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).

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>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>mbed</td>
<td>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.</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 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>

Configuration

Please use nucleo_l073rz ID for board option in “platformio.ini” (Project Configuration File):

```ini
[env:nucleo_l073rz]
platform = ststm32
board = nucleo_l073rz
```

You can override default ST Nucleo L073RZ settings per build environment using board_*** option, where *** is a JSON object path from board manifest nucleo_l073rz.json. For example, board_build.mcu, board_build.f_cpu, etc.

```ini
[env:nucleo_l073rz]
platform = ststm32
board = nucleo_l073rz

; change microcontroller
board_build.mcu = stm32l073rz

; change MCU frequency
board_build.f_cpu = 32000000L
```

Uploading

ST Nucleo L073RZ supports the next uploading protocols:

- blackmagic
- jlink
- mbed
- stlink

Default protocol is stlink

You can change upload protocol using upload_protocol option:

```ini
[env:nucleo_l073rz]
platform = ststm32
board = nucleo_l073rz
```

(continues on next page)
upload_protocol = stlink

**Debugging**

*PIO Unified 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).

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>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>mbed</td>
<td>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.</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>X-NUCLEO-F8P451M board, an 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 L152RE**

**Contents**

- *ST Nucleo L152RE*
  - 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>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):

```
[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.

```
[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
- jlink
- mbed
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:
Debugging

PIO Unified 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).

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></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>mbed</td>
<td>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.</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>
</tbody>
</table>

ST Nucleo L412KB

Contents

- ST Nucleo L412KB
  - 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>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**
- **jlink**
- **stlink**
Default protocol is stlink

You can change upload protocol using `upload_protocol` option:

```yaml
[env:nucleo_l412kb]
platform = ststm32
board = nucleo_l412kb
upload_protocol = stlink
```

### Debugging

**PIO Unified 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).

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>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>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
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)**:

```
[env:nucleo_l432kc]
platform = ststm32
board = nucleo_l432kc
```

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.

```
[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
- jlink
- mbed
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:
[env:nucleo_l432kc]
platform = stm32
board = nucleo_l432kc
upload_protocol = stlink

Debugging

PIO Unified 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.

ST Nucleo L432KC has on-board debug probe and IS READY for debugging. You don’t need to 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>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached 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>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.</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 L433RC-P

Contents

- ST Nucleo L433RC-P
  - Hardware
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>Frequency</th>
<th>Flash</th>
<th>RAM</th>
<th>Vendor</th>
</tr>
</thead>
<tbody>
<tr>
<td>STM32L433RC</td>
<td>80MHz</td>
<td>256KB</td>
<td>64KB</td>
<td>ST</td>
</tr>
</tbody>
</table>

Configuration

Please use `nucleo_l433rc_p` ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[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.

```ini
[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
- jlink
- mbed
- stlink
Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```ini
[env:nucleo_1433rc_p]
platform = ststm32
board = nucleo_1433rc_p
upload_protocol = stlink
```

## Debugging

**PIO Unified 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).

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>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>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.</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 L452RE

### Contents

- **ST Nucleo L452RE**
  - 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>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)*:

```ini
[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.

```ini
[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
- jlink
- mbed
- stlink
Default protocol is `stlink`.

You can change upload protocol using `upload_protocol` option:

```ini
[env:nucleo_l452re]
platform = ststm32
board = nucleo_l452re
upload_protocol = stlink
```

**Debugging**

*PIO Unified 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).

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>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>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>

**ST Nucleo L476RG**

**Contents**

- **ST Nucleo L476RG**
  - 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>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)*:

```ini
[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.

```ini
[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
- jlink
- mbed
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:
Debugging

PIO Unified 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).

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>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>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.</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 L486RG

Contents

- ST Nucleo L486RG
  - 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>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):

```ini
[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.`

```ini
[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
- 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

**PIO Unified 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).

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>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>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.</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>

### ST Nucleo L496ZG

<table>
<thead>
<tr>
<th>Contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>• <strong>ST Nucleo L496ZG</strong></td>
</tr>
<tr>
<td>• <strong>Hardware</strong></td>
</tr>
<tr>
<td>• <strong>Configuration</strong></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>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)*:

```
[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.

```
[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
- 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

**PIO Unified 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).

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>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>mbed</td>
<td>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.</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 L496ZG-P

**Contents**

- *ST Nucleo L496ZG-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>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
- 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

**PIO Unified 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).

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>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>mbed</td>
<td>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.</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 L4R5ZI
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:
PlatformIO Documentation, Release 4.0.1b3

- blackmagic
- jlink
- mbed
- stlink

Default protocol is stlink

You can change upload protocol using upload_protocol option:

```
[env:nucleo_l4r5zi]
platform = ststm32
board = nucleo_l4r5zi
upload_protocol = stlink
```

### Debugging

**PIO Unified 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).

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>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>mbed</td>
<td>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.</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 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
- jlink
- stlink

Default protocol is stlink

You can change upload protocol using `upload_protocol` option:

```plaintext
[env:disco_f030r8]
platform = ststm32
board = disco_f030r8
upload_protocol = stlink
```

Debugging

PIO Unified 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).

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>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>mbed</td>
<td>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.</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 STM32F0DISCOVERY

Contents

- ST STM32F0DISCOVERY
  - 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>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
- jlink
- stlink

Default protocol is stlink

You can change upload protocol using upload_protocol option:

```ini
[env:disco_f051r8]
platform = ststm32
board = disco_f051r8
upload_protocol = stlink
```

Debugging

PIO Unified 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).

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>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>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.</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>
ST STM32F3DISCOVERY

**Contents**

- **ST STM32F3DISCOVERY**
  - 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>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
- jlink
- mbed
- stlink

Default protocol is stlink

You can change upload protocol using upload_protocol option:

```ini
[env:disco_f303vc]
platform = stm32
board = disco_f303vc
upload_protocol = stlink
```

Debugging

PIO Unified 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).

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>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>libOpenCM3</td>
<td>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.</td>
</tr>
<tr>
<td>mbed</td>
<td>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.</td>
</tr>
<tr>
<td>SPL</td>
<td>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.</td>
</tr>
<tr>
<td>STM32Cube</td>
<td>The ST 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 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.

<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>ST</td>
</tr>
</tbody>
</table>
**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
- 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**

PIO Unified 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).
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>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>libOpenCM3</td>
<td>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.</td>
</tr>
<tr>
<td>mbed</td>
<td>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.</td>
</tr>
<tr>
<td>SPL</td>
<td>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.</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 STM32L073Z-EVAL

**Contents**

- **ST STM32L073Z-EVAL**
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

1.12. Boards 1535
PlatformIO Documentation, Release 4.0.1b3

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)*:

```ini
[env:eval_l073z]
platform = ststm32
board = eval_l073z
```

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
- 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

*PIO Unified 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).

ST STM32L073Z-EVAL has on-board debug probe and **IS READY** for debugging. You don’t need to 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>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 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):

```
[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.

```
[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
- jlink
- stlink

Default protocol is stlink

You can change upload protocol using `upload_protocol` option:

```
[env:disco_l152rb]
platform = ststm32
board = disco_l152rb

upload_protocol = stlink
```

**Debugging**

PIO Unified 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).

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>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>libOpenCM3</td>
<td>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.</td>
</tr>
<tr>
<td>SPL</td>
<td>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.</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 STM32VLDISCOVERY

Contents

- **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
- jlink
- stlink

Default protocol is stlink

You can change upload protocol using `upload_protocol` option:

```
[env:disco_f100rb]
platform = ststm32
board = disco_f100rb

upload_protocol = stlink
```
Debugging

PIO Unified 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).

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>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>mbed</td>
<td>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.</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 Sensor Node

Contents

- ST Sensor Node
  - 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>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
- 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
```

(continues on next page)
upload_protocol = mbed

**Debugging**

**PIO Unified 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).

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>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>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.</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>

**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
- dfu
- jlink
- serial
- stlink

Default protocol is stlink

You can change upload protocol using `upload_protocol` option:
[env:genericSTM32F103C8]
platform = ststm32
board = genericSTM32F103C8
upload_protocol = stlink

Debugging

PIO Unified 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).

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>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>libOpenCM3</td>
<td>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.</td>
</tr>
<tr>
<td>mbed</td>
<td>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.</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>
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
- dfu
- jlink
- serial
- stlink

Default protocol is stlink

You can change upload protocol using `upload_protocol` option:

```
[env:genericSTM32F103CB]
platform = stm32
board = genericSTM32F103CB
upload_protocol = stlink
```

Debugging

PIO Unified 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).

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></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>libOpenCM3</td>
<td>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.</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>

STM32F103R8 (20k RAM. 64 Flash)

Contents

- STM32F103R8 (20k RAM. 64 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>STM32F103R8T6</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):
You can override default STM32F103R8 (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 = stm32f103r8t6

; change MCU frequency
board_build.f_cpu = 72000000L
```

### Uploading

STM32F103R8 (20k RAM. 64 Flash) 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:genericSTM32F103R8]
platform = ststm32
board = genericSTM32F103R8

upload_protocol = stlink
```

### Debugging

**PIO Unified 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*).

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.
## Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached 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>libOpenCM3</td>
<td>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.</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>

### STM32F103RB (20k RAM. 128k Flash)

#### Contents

- STM32F103RB (20k 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.
PlatformIO Documentation, Release 4.0.1b3

<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):

```ini
[env:genericSTM32F103RB]
platform = ststm32
board = genericSTM32F103RB
```

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`
- `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**

*PIO Unified 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).

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></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>libOpenCM3</td>
<td>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.</td>
</tr>
<tr>
<td>mbed</td>
<td>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.</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>

STM32F103RC (48k RAM. 256k Flash)
-- 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>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
- dfu
- jlink
- serial
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:
[env:genericSTM32F103RC]
platform = stm32
board = genericSTM32F103RC
upload_protocol = stlink

Debugging

PIO Unified 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).

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>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>libOpenCM3</td>
<td>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.</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>

STM32F103RE (64k RAM. 512k 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>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):

```ini
[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,

```ini
[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
- dfu
• jlink
• serial
• stlink

Default protocol is stlink

You can change upload protocol using upload_protocol option:

```
[env:genericSTM32F103RE]
platform = ststm32
board = genericSTM32F103RE
upload_protocol = stlink
```

**Debugging**

**PIO Unified 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).

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>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>libOpenCM3</td>
<td>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.</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>
STM32F103T8 (20k RAM, 64k Flash)

## Contents

- **STM32F103T8 (20k RAM, 64k 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>STM32F103T8T6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequency</td>
<td>72MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>20KB</td>
</tr>
<tr>
<td>RAM</td>
<td>64KB</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):

```
[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.

```
[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
- 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

PIO Unified 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).

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>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>libOpenCM3</td>
<td>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.</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>

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.

```
[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
- dfu
- jlink
- serial
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```
[env:genericSTM32F103TB]
platform = ststm32
board = genericSTM32F103TB

upload_protocol = stlink
```

### Debugging

**PIO Unified 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).

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.
### 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>libOpenCM3</td>
<td>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.</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>

### 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.
### Configuration

Please use `genericSTM32F103VB` ID for `board` option in “`platformio.ini` (Project Configuration File):

```ini
[env:genericSTM32F103VB]
platform = ststm32
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.

```ini
[env:genericSTM32F103VB]
platform = ststm32
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
- dfu
- jlink
- serial
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```ini
[env:genericSTM32F103VB]
platform = ststm32
board = genericSTM32F103VB

upload_protocol = stlink
```

### Debugging

*PIO Unified 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).

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>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>libOpenCM3</td>
<td>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.</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>

STM32F103VC (48k RAM. 256k Flash)

Contents

- STM32F103VC (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>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:

- blackmagic
- dfu
- jlink
- serial
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:
Debugging

PIO Unified 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).

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>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>libOpenCM3</td>
<td>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.</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>

STM32F103VD (64k RAM. 384k Flash)
• **STM32F103VD (64k 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>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*):

```
[env:genericSTM32F103VD]
platform = ststm32
board = genericSTM32F103VD
```

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.**

```
[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
- dfu
- jlink
- serial
- stlink

Default protocol is stlink

You can change upload protocol using `upload_protocol` option:

```
[env:genericSTM32F103VD]
platform = ststm32
board = genericSTM32F103VD
upload_protocol = stlink
```

## Debugging

**PIO Unified 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).

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></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>libOpenCM3</td>
<td>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.</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>
STM32F103VE (64k RAM. 512k Flash)

Contents

- STM32F103VE (64k 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>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
• 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

PIO Unified 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).

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>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>libOpenCM3</td>
<td>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.</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>

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
- 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

**PIO Unified 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).

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.
### 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>libOpenCM3</td>
<td>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.</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>

### STM32F103ZD (64k RAM. 384k Flash)

#### Contents

- STM32F103ZD (64k 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.
**Configuration**

Please use `genericSTM32F103ZD` ID for `board` option in "platformio.ini" (Project Configuration File):

```
[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.

```
[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
- dfu
- jlink
- serial
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```
[env:genericSTM32F103ZD]
platform = ststm32
board = genericSTM32F103ZD

upload_protocol = stlink
```

**Debugging**

*PIO Unified 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).

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>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>libOpenCM3</td>
<td>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.</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>

STM32F103ZE (64k RAM. 512k Flash)

- STM32F103ZE (64k 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>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
- dfu
- jlink
- serial
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:
Debugging

PIO Unified 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).

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></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>libOpenCM3</td>
<td>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.</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>

STM32F303CB (32k RAM. 128k 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>STM32F303CBT6</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>32KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>Generic</td>
</tr>
</tbody>
</table>

Configuration

Please use `genericSTM32F303CB` ID for `board` option in "platformio.ini" (Project Configuration File):

```
[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.

```
[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
- 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

**PIO Unified 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).

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>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>STM3C</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>

### STM32F407VE (192k RAM. 512k Flash)

Contents

- STM32F407VE (192k RAM. 512k Flash)
  - 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>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):

```ini
[env:genericSTM32F407VET6]
platform = ststm32
board = genericSTM32F407VET6
```

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:

```
[env:genericSTM32F407VET6]
platform = ststm32
board = genericSTM32F407VET6
upload_protocol = stlink
```

### Debugging

**PIO Unified 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).

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>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached 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>

### STM32F407VG (192k RAM. 1024k Flash)

- **STM32F407VG (192k RAM. 1024k 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>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):

```
[env:genericSTM32F407VGT6]
platform = ststm32
board = genericSTM32F407VGT6
```

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.

```
[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:

```
[env:genericSTM32F407VGT6]
platform = ststm32
board = genericSTM32F407VGT6
upload_protocol = stlink
```

## Debugging

**PIO Unified 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).

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>

### Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached 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>

### STM32F4Stamp F405

**Contents**

- STM32F4Stamp F405
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):

```ini
[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.

```ini
[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:

```ini
[env:stm32f4stamp]
platform = ststm32
board = stm32f4stamp
upload_protocol = dfu
```

### Debugging

**PIO Unified 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).

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>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>

### STM32F7508-DK

**Contents**

- **STM32F7508-DK**
  - 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>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
- jlink
- stlink

Default protocol is stlink

You can change upload protocol using `upload_protocol` option:
Debugging

PIO Unified 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).

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>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>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>

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):

```
[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.

```
[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
- jlink
- mbed
- stlink

Default protocol is mbed

You can change upload protocol using `upload_protocol` option:

```
[env:seeedArchMax]
platform = ststm32
board = seeedArchMax
```

(continues on next page)
`upload_protocol = mbed`

**Debugging**

*PIO Unified 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).

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>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><em>mbed</em></td>
<td>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.</td>
</tr>
<tr>
<td><em>STM32Cube</em></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>

**Seeed Wio 3G**

- **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.

<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 = stm32
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 = stm32
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 = stm32
board = wio_3g

upload_protocol = stlink
```
Debugging

PIO Unified 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).

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>mbed</td>
<td>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.</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>

Sparky V1 F303

Contents

- Sparky V1 F303
  - 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>TauLabs</td>
</tr>
</tbody>
</table>

Configuration

Please use `sparky_v1` ID for `board` option in “platformio.ini” (Project Configuration File):

```
[env:sparky_v1]
platform = stm32
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 = stm32
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
- jlink
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```
[env:sparky_v1]
platform = stm32
board = sparky_v1

upload_protocol = stlink
```
Debugging

PIO Unified 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)*.

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>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>

Tiny STM103T

**Contents**

- Tiny STM103T
  - 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>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
- dfu
- jlink
- serial
- stlink

Default protocol is *dfu*

You can change upload protocol using *upload_protocol* option:

```
[env:hy_tinystm103tb]
platform = ststm32
board = hy_tinystm103tb

upload_protocol = dfu
```
Debugging

PIO Unified 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).

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>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>

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
- 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**

*PIO Unified 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).

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></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>ST STM32</td>
<td>ST 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>
</tbody>
</table>

sakura.io Evaluation Board

- 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.
### Configuration

Please use `sakuraio_evb_01` ID for `board` option in “`platformio.ini`” *(Project Configuration File):*

```ini
[env:sakuraio_evb_01]
platform = ststm32
board = sakuraio_evb_01
```

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.

```ini
[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:

```ini
[env:sakuraio_evb_01]
platform = ststm32
board = sakuraio_evb_01

upload_protocol = mbed
```

### Debugging

**PIO Unified 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).

sakura.io Evaluation 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>mbed</td>
<td>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.</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>

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

**PIO Unified 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).

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>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.</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>

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

**PIO Unified 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).

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>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.</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>

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.
PlatformIO Documentation, Release 4.0.1b3

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>STM32F437VG</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>u-blox</td>
</tr>
</tbody>
</table>

## 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

*PIO Unified 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*).

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>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.</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.
<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 `ublox_evk_odin_w2` ID for `board` option in “platformio.ini” (Project Configuration File):

```ini
[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.

```ini
[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
- jlink
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```ini
[env:ublox_evk_odin_w2]
platform = ststm32
board = ublox_evk_odin_w2

upload_protocol = stlink
```

## Debugging

PIO Unified 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).

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>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>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.</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>

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.
### PlatformIO Documentation, Release 4.0.1b3

<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):

```ini
[env:mtb_ublox_odin_w2]
platform = stm32
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.

```ini
[env:mtb_ublox_odin_w2]
platform = stm32
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
- jlink
- stlink

Default protocol is stlink

You can change upload protocol using `upload_protocol` option:

```ini
[env:mtb_ublox_odin_w2]
platform = stm32
board = mtb_ublox_odin_w2

upload_protocol = stlink
```

#### Debugging

**PIO Unified 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).

u-blox 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>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>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.</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>

### 1.12.23 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.
### Configurations

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

PIO Unified 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).

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>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.</td>
</tr>
<tr>
<td>SPL</td>
<td>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.</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 `stm8sblue` ID for `board` option in "platformio.ini" (Project Configuration File):

```
[env:stm8sblue]
platform = ststm8
board = stm8sblue
```

You can override default ST STM8S103F3 Breakout Board settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `stm8sblue.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:stm8sblue]
platform = ststm8
board = stm8sblue

; change microcontroller
board_build.mcu = stm8s103f3p6

; change MCU frequency
board_build.f_cpu = 16000000L
```

Uploading

ST STM8S103F3 Breakout Board supports the next uploading protocols:

- `serial`
- `stlinkv2`

Default protocol is `serial`

You can change upload protocol using `upload_protocol` option:

```
[env:stm8sblue]
platform = ststm8
board = stm8sblue

upload_protocol = serial
```

Debugging

PIO Unified Debugger 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>SPL</td>
<td>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.</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):

```ini
[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.

```ini
[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:

```yaml
[env:stm8sblack]
platform = ststm8
board = stm8sblack
upload_protocol = serial
```

**Debugging**

**PIO Unified Debugger** 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>SPL</td>
<td>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.</td>
</tr>
</tbody>
</table>

**sduino MB (STM8S208MBT6B)**

**Contents**

- `sduino MB (STM8S208MBT6B)`
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

**Hardware**

Platform **ST STMS8**: 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.
## Configuration

Please use `mb208` ID for `board` option in "platformio.ini" (Project Configuration File):

```ini
[env:mb208]
platform = ststm8
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 = ststm8
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 = ststm8
board = mb208

upload_protocol = serial
```

## Debugging

`PIO Unified Debugger` 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>SPL</td>
<td>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.</td>
</tr>
</tbody>
</table>

sduino UNO (STM8S105K6)

Contents

- sduino UNO (STM8S105K6)
  - Hardware
  - Configuration
  - Uploading
  - Debugging
  - Frameworks

Hardware

Platform **ST STMS**: 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):**

```
[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 following uploading protocols:

- `serial`
- `stlinkv2`

Default protocol is `serial`

You can change the upload protocol using the `upload_protocol` option:

```
[env:s8uno]
platform = stm8
board = s8uno

upload_protocol = serial
```

### Debugging

**PIO Unified Debugger** 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>SPL</td>
<td>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.</td>
</tr>
</tbody>
</table>

### 1.12.24 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):

```
[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.

```
[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:
**Debugger**

`PIO Unified Debugger` 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>Frequency</th>
<th>Flash</th>
<th>RAM</th>
<th>Vendor</th>
</tr>
</thead>
<tbody>
<tr>
<td>MK20DX128</td>
<td>48MHz</td>
<td>128KB</td>
<td>16KB</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.

```
[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:

```
[env:teensy30]
platform = teensy
board = teensy30

upload_protocol = teensy-gui
```

### Debugging

*PIO Unified Debugger* 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
• Teensy 3.1 / 3.2
  – 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>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):

```
[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.

```
[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
• teensy-gui

Default protocol is teensy-gui

You can change upload protocol using upload_protocol option:

```
[env:teensy31]
platform = teensy
board = teensy31
upload_protocol = teensy-gui
```

### Debugging

**PIO Unified 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).

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>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.</td>
</tr>
</tbody>
</table>

### Teensy 3.5

#### Contents

- Teensy 3.5
  - Hardware
  - Configuration
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:
[env:teensy35]
platform = teensy
board = teensy35
upload_protocol = teensy-gui

**Debugging**

*PIO Unified 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).

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>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 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)*:

```ini
[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.

```ini
[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:

```ini
[env:teensy36]
platform = teensy
board = teensy36

upload_protocol = teensy-gui
```
Debugging

PIO Unified 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).

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 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):

```
[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.

```
[env:teensylc]
platform = teensy
board = teensylc

; change microcontroller
board_build.mcu = mkl26z64

; change MCU frequency
board_build.f_cpu = 48000000L
```

## 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:

```
[env:teensylc]
platform = teensy
board = teensylc

upload_protocol = teensy-gui
```

## Debugging

*PIO Unified 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)*.

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><strong>Yes</strong></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.

**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):

```ini
[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.

```ini
[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:

```ini
[env:teensy2pp]
platform = teensy
board = teensy2pp

upload_protocol = teensy-gui
```

Debugging

*PIO Unified Debugger* 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>
1.12.25 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.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>MSP430FR5739</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>512B</td>
</tr>
<tr>
<td>Vendor</td>
<td>TI</td>
</tr>
</tbody>
</table>

Configuration

Please use **lpmsp430fr5739** ID for **board** option in “*
platformio.ini*” (Project Configuration File):

```
[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.

```
[env:lpmsp430fr5739]
platform = timsp430
board = lpmsp430fr5739

; change microcontroller
board_build.mcu = msp430fr5739

; change MCU frequency
board_build.f_cpu = 16000000L
```
Debugging

**PIO Unified 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).

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>16MHz</td>
</tr>
<tr>
<td>Flash</td>
<td>128KB</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
board_build.mcu = msp430f5529

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

PIO Unified 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).

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>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.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>MSP430FR4133</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>2KB</td>
</tr>
<tr>
<td>Vendor</td>
<td>TI</td>
</tr>
</tbody>
</table>

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

PIO Unified 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).

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>64KB</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):

```
[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.

```
[env:lpmsp430fr5969]
platform = timsp430
board = lpmsp430fr5969

; change microcontroller
board_build.mcu = msp430fr5969

; change MCU frequency
board_build.f_cpu = 8000000L
```

Debugging

**PIO Unified 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).

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-EXP430FR6989LP
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>127KB</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):

```ini
[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.

```ini
[env:lpmsp430fr6989]
platform = timsp430
board = lpmsp430fr6989

; change microcontroller
board_build.mcu = msp430fr6989

; change MCU frequency
board_build.f_cpu = 8000000L
```

Debugging

**PIO Unified 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).

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-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.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>MSP430G2553</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>512B</td>
</tr>
<tr>
<td>Vendor</td>
<td>TI</td>
</tr>
</tbody>
</table>
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

**PIO Unified 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).

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>

1.12.26 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 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>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):

```ini
[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.

```ini
[env:lplm4f120h5qr]
platform = titiva
board = lplm4f120h5qr

; change microcontroller
board_build.mcu = lplm4f120h5qr

; change MCU frequency
board_build.f_cpu = 80000000L
```

Debugging

**PIO Unified 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).

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-JCDI</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/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.</td>
</tr>
</tbody>
</table>

TI LaunchPad (Tiva C) w/ tm4c123 (80MHz)

Contents

- TI LaunchPad (Tiva C) w/ tm4c123 (80MHz)
  - Hardware
  - Configuration
  - Debugging
  - Frameworks

Hardware

Platform **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>
PlatformIO Documentation, Release 4.0.1b3

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

PIO Unified 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).

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>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/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.</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

**PIO Unified 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**).

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-JCDI</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/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.</td>
</tr>
</tbody>
</table>

### 1.12.27 WIZNet W7500

**WIZwiki-W7500**

**Contents**

- **WIZwiki-W7500**
  - 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_w7500 ID for `board` option in "platformio.ini" (Project Configuration File):

```
[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.:

```
[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

*PIO Unified 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).

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>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.</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):

```
[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.

```
[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:

```
[env:wizwiki_w7500eco]
platform = wiznet7500
board = wizwiki_w7500eco

upload_protocol = mbed
```

Debugging

PIO Unified 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).

WIZwiki-W7500ECO 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>CMSIS-DAP</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>J-LINK</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

## Frameworks

### mbed

**Description:**
The mbed framework is 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.

### 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 platforms especially requiring Internet of things.

<table>
<thead>
<tr>
<th>Microcontroller</th>
<th>WIZNET7500P</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_w7500p` ID for `board` option in "platformio.ini" (Project Configuration File):

```
[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.

```
[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:

```
[env:wizwiki_w7500p]
platform = wiznet7500
board = wizwiki_w7500p

upload_protocol = mbed
```

### Debugging

**PIO Unified 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).

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>mbed</td>
<td>The mbed framework 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.</td>
</tr>
</tbody>
</table>

1.13 Custom Platform & Board

1.13.1 Custom Development Platform

*PlatformIO* was developed like a tool that may build the same source code for the different development platforms via single command `platformio run` without any dependent software or requirements.

For this purpose *PlatformIO* uses own pre-configured platforms data: build scripts, toolchains, the settings for the most popular embedded boards and etc. These data are pre-built and packaged to the different packages. It allows *PlatformIO* to have multiple development platforms which can use the same packages(toolchains, frameworks), but have different/own build scripts, uploader and etc.

**Note:** If you want to change some build flags for the existing Development Platforms, you don’t need to create (or duplicate) own development platforms! Please use `build_flags` option.

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 Platform
  - Packages
  - Manifest File `platform.json`
  - Build Script `main.py`
  - Installation
  - Examples

Packages

PlatformIO has own registry with pre-built packages for the most popular systems and you can use them in your manifest. These packages are stored in super-fast and reliably CDN storage provided by JFrog Bintray:
• https://bintray.com/platformio/dl-packages

**Manifest File platform.json**

See example with a manifest for packages:

• http://dl.platformio.org/packages/manifest.json

```
{
    "name": "myplatform",
    "title": "My Platform",
    "description": "My custom development platform",
    "url": "http://example.com",
    "homepage": "https://platformio.org/platforms/myplatform",
    "license": "Apache-2.0",
    "engines": {
        "platformio": "~3.0.0"
    },
    "repository": {
        "type": "git",
        "url": "https://github.com/platformio/platform-myplatform.git"
    },
    "version": "0.0.0",
    "packageRepositories": [
        "https://dl.bintray.com/platformio/dl-packages/manifest.json",
        "http://dl.platformio.org/packages/manifest.json",
        {
            "my_custom_package": [
                {
                    "url": "http://dl.example.com/my_custom_package-darwin_x86_64-1.2.3.tar.gz",
                    "sha1": "bb7ddac56a314b5cb1926cc1790ae4de3a03e65c",
                    "version": "1.2.3",
                    "system": [
                        "darwin_x86_64",
                        "darwin_i386"
                    ]
                },
                {
                    "url": "http://dl.example.com/my_custom_package-linux_aarch64-1.2.3.tar.gz",
                    "sha1": "127ddac56a314b5cb1926cc1790ae4de3a03e65c",
                    "version": "1.2.3",
                    "system": "linux_aarch64"
                }
            ],
            "framework-%FRAMEWORK_NAME_1%": [
                {
                    "url": "http://dl.example.com/packages/framework-%FRAMEWORK_NAME_1%-1.10607.0.tar.gz",
                    "sha1": "adce2cd30a830d71cb6572575bf08461b7b73c07",
                    "version": "1.10607.0",
                    "system": "*
                ]
            ],
            "frameworks": {
                "%FRAMEWORK_NAME_1%": {
```

(continues on next page)
Build Script main.py

Platform’s build script is based on a next-generation build tool named SCons. PlatformIO has own built-in firmware builder env.BuildProgram with the deep libraries search. Please look into a base template of main.py.

```python
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",
)```

---

Build Script main.py

Platform’s build script is based on a next-generation build tool named SCons. PlatformIO has own built-in firmware builder env.BuildProgram with the deep libraries search. Please look into a base template of main.py.

```python
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="...",
    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

1. Create `platforms` directory in `core_dir` if it doesn’t exist.
2. Create `myplatform` directory in `platforms`.
3. Copy `platform.json` and `builder/main.py` files to `myplatform` directory.
4. Search available platforms via `platformio platform search` command. You should see `myplatform` platform.
5. Install `myplatform` platform via `platformio platform install` command.

Now, you can use `myplatform` for the `platform` option in “`platformio.ini`” (Project Configuration File).

Examples

Please take a look at the source code of PlatformIO Development Platforms.

1.13.2 Custom Embedded Board

`PlatformIO` has pre-built settings for the most popular embedded boards. This list is available:

- Embedded Boards Explorer (Web)
- `platformio boards` (CLI command)

Nevertheless, PlatformIO allows one to create own board or override existing board’s settings. All data is declared using JSON-style via associative array (name/value pairs).

Contents

- Custom Embedded Board
  - JSON Structure
  - Installation
  - Examples

JSON Structure

The key fields:

- `build` data will be used by Development Platforms and Frameworks builders
- `frameworks` is the list with supported Frameworks
- `platform` name of Development Platforms
- `upload` upload settings which depend on the platform

```json
{
    "build": {
        "extra_flags": "-DHELLO_PLATFORMIO",
        "f_cpu": "16000000L",
        "hwids": [
            "0x1234",
            "0x1235"
        ]
    }
}
```

(continues on next page)
"0x0013",
],
[
"0x4567",
"0x0013"
]
},
"mcu": "%MCU_TYPE_HERE%",
"frameworks": ["%LIST_WITH_SUPPORTED_FRAMEWORKS%"],
"name": "My Test Board",
"upload": {
"maximum_ram_size": 2048,
"maximum_size": 32256
},
"url": "http://example.com",
"vendor": "MyCompany"
}

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 platformio 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 PIO Account

PIO Account is required for using:

- PIO Remote
- Integration with Cloud IDE

A registration is FREE. No Credit Card Required.

1.14.1 PlatformIO IDE

PlatformIO IDE has built-in UI in PIO Home to manage PIO Account. You can create a new account, reset your password or fetch an authentication token.
1.15 PIO Remote

Your devices are always with you!

PIO Remote allows you to work remotely with devices from Anywhere In The World. No matter where are you now! Run a small and cross-platform PIO 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 PIO Remote 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 PIO Account to work with PIO Remote. A registration is FREE.
1.15.1 Features

- Remote Device Manager
- Remote Serial Port Monitor
- Remote Firmware Updates
- PIO Remote Share
- Continuous Deployment
- Continuous Delivery
- Remote Unit Testing

1.15.2 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
- *PIO 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 *PIO Remote* 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. PIO Remote allows you to connect to a remote agent and list connected devices with their serial ports. See platformio remote device monitor command for details.
1.15.3 Technology
PIO Remote is an own PIO Plus 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 PIO Remote Agent and Client (platformio remote, Cloud IDE, Continuous Integration, SDKs, etc.). When you start PIO 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.

PIO Remote 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 platformio device list and platformio remote device list.

1.15.4 Installation

PIO Remote is built into PlatformIO IDE. Please open PlatformIO IDE Terminal and run pio remote --help command for usage (see platformio remote).

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).

1.15.5 Quick Start

1. Start PIO Remote Agent using platformio remote agent start command on a remote machine where devices are connected physically or are accessible via network. PIO Remote 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 platformio remote agent start --share option.

2. Using host machine (platformio remote, Cloud IDE Terminal in a browser, SDKs, etc.), please authorize via platformio account login command with the same credentials that you used on the previous step. Now, you can use platformio remote commands to work with remote machine and its devices.

   You don’t need to have networking or other access to remote machine where PIO Remote Agent is started.

   If you use PIO Remote in pair with Continuous Integration or want automatically authorize, please set PLATFORMIO_AUTH_TOKEN system environment variable instead of using platformio 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 PIO Remote 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 PIO Remote Agent is started but supports natively Over-the-Air (OTA) updates. For example, Espresif 8266 and Over-the-Air (OTA) update. In this case, the final command for remote OTA update will look as platformio remote run -t upload --upload-port 192.168.0.255 or platformio remote run -t upload --upload-port myesp8266.local.
1.15.6 CLI Guide

1.16 PIO Unified Debugger

It Simply Works. Easier than ever before!

Note: Demo, discussions, request a support for new hardware.

PIO Plus 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 Plus 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 PlatformIO IDE for VSCode
- Integration with Eclipse and Sublime Text

You should have PIO Account to work with PIO Unified Debugger. A registration is FREE.

Hint: In our experience, PlatformIO IDE for VSCode has the best system performance, modern interface for PIO Unified Debugger, and users have found it easier to get started. Key debugging features of PlatformIO IDE for 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
1.16. PIO Unified Debugger
1.16.1 Tutorials

- 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
- Arduino and Nordic nRF52-DK: debugging and unit testing
- STM32Cube HAL and Nucleo-F401RE: debugging and unit testing

1.16.2 Configuration

PIO Unified Debugger can be configured from “platformio.ini” (Project Configuration File):

1.16.3 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.

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.
Configuration

You can configure debugging tool using `debug_tool` option in “platformio.ini” (Project Configuration File):

```
[env:myenv]
platform = ...
board = ...
debug_tool = atmel-ice
```

If you would like to use this tool for firmware uploading, please change upload protocol:

```
[env:myenv]
platform = ...
board = ...
debug_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</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>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.</td>
</tr>
<tr>
<td>Simba</td>
<td>Simba is an RTOS and build framework. It 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>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 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 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 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>SAMD51J19A</td>
<td>120MHz</td>
<td>512KB</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>SAMD51J20A</td>
<td>120MHz</td>
<td>1MB</td>
<td>256KB</td>
</tr>
<tr>
<td>Adafruit PyGamer Advance 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 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 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 pHKey</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 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>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>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>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>On-board</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 Wi-Fi 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 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>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>Macchina M2</td>
<td>Atmel SAM</td>
<td>External</td>
<td>AT91SAM3X8E</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</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>
</tbody>
</table>

Continued on next page
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>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>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 LoRaWAN</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>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>

**Black Magic Probe**

![Black Magic Probe](image)

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
- Boards
**Configuration**

You can configure debugging tool using `debug_tool` option in "platformio.ini" (Project Configuration File):

```ini
[env:myenv]
platform = ...
board = ...
debug_tool = blackmagic
dump_port = <CONFIGURE GDB PORT>

; Debug Port Examples

; Linux
dump_port = /dev/ttyACM0

; Windows for COM1-COM9
dump_port = COM3
; Windows for COM10-XXX
dump_port = \..\..\COM13

; macOS
dump_port = /dev/cu.usbmodemF2D0C4C6
```

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 `platformio 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 = ...
debug_tool = blackmagic
dump_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 1</td>
<td>VCC 1</td>
</tr>
<tr>
<td>GND 3</td>
<td>GND 3</td>
</tr>
<tr>
<td>GND 5</td>
<td>GND 5</td>
</tr>
<tr>
<td>N/U 7</td>
<td>N/U 7</td>
</tr>
<tr>
<td>GND 9</td>
<td>GND 10</td>
</tr>
<tr>
<td>SWDIO 2</td>
<td>VCC 2 (optional)</td>
</tr>
<tr>
<td>SWCLK 4</td>
<td>N/U 4</td>
</tr>
<tr>
<td>SWO 6</td>
<td>N/U 6</td>
</tr>
<tr>
<td>SWDO 8</td>
<td>SWDO 8</td>
</tr>
<tr>
<td>SWCLK 9</td>
<td>SWCLK 9</td>
</tr>
<tr>
<td>N/U 11</td>
<td>N/U 12</td>
</tr>
<tr>
<td>SWO 13</td>
<td>SWO 14</td>
</tr>
<tr>
<td>RESET 15</td>
<td>SWDIO 16</td>
</tr>
<tr>
<td>N/C 17</td>
<td>N/C 18</td>
</tr>
<tr>
<td>N/C 19</td>
<td>N/C 20</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Black Magic Probe 10-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>SWDIO</td>
<td>Positive Supply Voltage — Power supply for JTAG interface drivers</td>
</tr>
<tr>
<td>3</td>
<td>GND</td>
<td>SWDIO</td>
<td>Digital ground</td>
</tr>
<tr>
<td>2</td>
<td>SWDIO</td>
<td>SWCLK</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>
<tr>
<td>Name</td>
<td>Description</td>
<td></td>
<td></td>
</tr>
<tr>
<td>------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td></td>
<td></td>
</tr>
<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>
<td></td>
<td></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>
<td></td>
<td></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>
<td></td>
<td></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>
<td></td>
<td></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>
<td></td>
<td></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>
<td></td>
<td></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>
<td></td>
<td></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>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached 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>libOpenCM3</td>
<td>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.</td>
</tr>
<tr>
<td>mbed</td>
<td>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.</td>
</tr>
<tr>
<td>Simba</td>
<td>Simba is an RTOS and build framework. It aims to make embedded programming easy and portable.</td>
</tr>
<tr>
<td>SPL</td>
<td>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.</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>

Boards

Note: For more detailed board information please scroll tables below by horizontal.
## Table 28 – continued from previous page

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adafruit Metro M0 Expresss</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
</tr>
<tr>
<td>Adafruit Trinket M0</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21E18A</td>
<td>48MHz</td>
</tr>
<tr>
<td>Adafruit pHKey</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21E18A</td>
<td>48MHz</td>
</tr>
<tr>
<td>Arduino Due (Programming Port)</td>
<td>Atmel SAM</td>
<td>External</td>
<td>AT91SAM3X8E</td>
<td>84MHz</td>
</tr>
<tr>
<td>Arduino Due (USB Native Port)</td>
<td>Atmel SAM</td>
<td>External</td>
<td>AT91SAM3X8E</td>
<td>84MHz</td>
</tr>
<tr>
<td>Arduino M0</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
</tr>
<tr>
<td>Arduino M0 Pro (Native USB Port)</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</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>
</tr>
<tr>
<td>Arduino MKR FOX 1200</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
</tr>
<tr>
<td>Arduino MKR GSM 1400</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
</tr>
<tr>
<td>Arduino MKR NB 1500</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
</tr>
<tr>
<td>Arduino MKR WAN 1300</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
</tr>
<tr>
<td>Arduino MKR WiFi 1010</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
</tr>
<tr>
<td>Arduino MKR1000</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
</tr>
<tr>
<td>Arduino MKRZERO</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
</tr>
<tr>
<td>Arduino Tian</td>
<td>Atmel SAM</td>
<td>On-board</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
</tr>
<tr>
<td>Arduino Zero (Programming/Debug Port)</td>
<td>Atmel SAM</td>
<td>On-board</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
</tr>
<tr>
<td>Arduino Zero (USB Native Port)</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
</tr>
<tr>
<td>Armstrap Eagle 1024</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F417VGT6</td>
<td>168MHz</td>
</tr>
<tr>
<td>Armstrap Eagle 2048</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F427VIT6</td>
<td>168MHz</td>
</tr>
<tr>
<td>Armstrap Eagle 512</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
</tr>
<tr>
<td>Atmel ATSAMR21-XPRO</td>
<td>Atmel SAM</td>
<td>On-board</td>
<td>SAMR21G18A</td>
<td>48MHz</td>
</tr>
<tr>
<td>Atmel ATSAMW25-XPRO</td>
<td>Atmel SAM</td>
<td>On-board</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
</tr>
<tr>
<td>Atmel SAMD21-XPRO</td>
<td>Atmel SAM</td>
<td>On-board</td>
<td>SAMD21J18A</td>
<td>48MHz</td>
</tr>
<tr>
<td>Atmel SAML21-XPRO-B</td>
<td>Atmel SAM</td>
<td>On-board</td>
<td>SAML21J18B</td>
<td>48MHz</td>
</tr>
<tr>
<td>Bambino-210E</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC4330</td>
<td>204MHz</td>
</tr>
<tr>
<td>Black STM32F407VE</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
</tr>
<tr>
<td>Black STM32F407VG</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407VG</td>
<td>168MHz</td>
</tr>
<tr>
<td>Black STM32F407ZE</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407ZG</td>
<td>168MHz</td>
</tr>
<tr>
<td>Black STM32F407ZE (128k)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407ZT6</td>
<td>168MHz</td>
</tr>
<tr>
<td>BlackPill F103C8</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
</tr>
<tr>
<td>BlackPill F103C8 (128k)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
</tr>
<tr>
<td>BluePill F303CC</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F303CC</td>
<td>72MHz</td>
</tr>
<tr>
<td>Blue STM32F407VE Mini</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
</tr>
<tr>
<td>BluePill F103C6</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103C6T6</td>
<td>72MHz</td>
</tr>
<tr>
<td>BluePill F103C8</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
</tr>
<tr>
<td>BluePill F103C8 (128k)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
</tr>
<tr>
<td>Bluey nRF52832 IoT</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52832</td>
<td>64MHz</td>
</tr>
<tr>
<td>Bluey nRF51</td>
<td>Nordic nRF51</td>
<td>External</td>
<td>NRF51822</td>
<td>32MHz</td>
</tr>
<tr>
<td>CQ Publishing TG-LPC11U35-501</td>
<td>NXP LPC</td>
<td>External</td>
<td>LPC11U35</td>
<td>48MHz</td>
</tr>
<tr>
<td>CoCo-ri-Co!</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC812</td>
<td>30MHz</td>
</tr>
<tr>
<td>Delta DFBM-NQ620</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52832</td>
<td>64MHz</td>
</tr>
<tr>
<td>Demo F030F4</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F030F4P6</td>
<td>48MHz</td>
</tr>
<tr>
<td>DigiStump DigiX</td>
<td>Atmel SAM</td>
<td>External</td>
<td>AT91SAM3X8E</td>
<td>84MHz</td>
</tr>
<tr>
<td>EA LPC11U35 QuickStart Board</td>
<td>NXP LPC</td>
<td>External</td>
<td>LPC11U35</td>
<td>48MHz</td>
</tr>
<tr>
<td>EFM32GG-STK3700 Giant Gecko</td>
<td>Silicon Labs EFM32</td>
<td>On-board</td>
<td>EFM32GG9990F1024</td>
<td>48MHz</td>
</tr>
<tr>
<td>EFM32LG-STK3600 Leopard Gecko</td>
<td>Silicon Labs EFM32</td>
<td>On-board</td>
<td>EFM32LG9990F256</td>
<td>48MHz</td>
</tr>
<tr>
<td>EFM32WG-STK3800 Wonder Gecko</td>
<td>Silicon Labs EFM32</td>
<td>On-board</td>
<td>EFM32WG9990F256</td>
<td>48MHz</td>
</tr>
<tr>
<td>EFM32ZG-STK3200 Zero Gecko</td>
<td>Silicon Labs EFM32</td>
<td>On-board</td>
<td>EFM32ZG222F32</td>
<td>24MHz</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>Espotel LoRa Module</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411RET6</td>
<td>100M</td>
</tr>
<tr>
<td>F407VG</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407VG76</td>
<td>168M</td>
</tr>
<tr>
<td>FK407M1</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407VET6</td>
<td>168M</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-KL25Z</td>
<td>Freescale Kinetis</td>
<td>On-board</td>
<td>MKL25Z128VLK4</td>
<td>48M</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-KL27Z</td>
<td>Freescale Kinetis</td>
<td>On-board</td>
<td>MKL27Z64VLH4</td>
<td>48M</td>
</tr>
<tr>
<td>L476DMW1K</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L476VGT6</td>
<td>80M</td>
</tr>
<tr>
<td>LPCXpresso11U68</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC11U68</td>
<td>50M</td>
</tr>
<tr>
<td>LPCXpresso824-MAX</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC824</td>
<td>30M</td>
</tr>
<tr>
<td>M200 V2</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F070CBT6</td>
<td>48M</td>
</tr>
<tr>
<td>MKR Vidor 4000</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48M</td>
</tr>
<tr>
<td>MTS Dragonfly</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411RET6</td>
<td>100M</td>
</tr>
<tr>
<td>Macchina M2</td>
<td>Atmel SAM</td>
<td>External</td>
<td>AT91SAM3X8E</td>
<td>84M</td>
</tr>
<tr>
<td>Malyan M200 V1</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103CBT6</td>
<td>72M</td>
</tr>
<tr>
<td>Maple</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103RBT6</td>
<td>72M</td>
</tr>
<tr>
<td>Maple (RET6)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103RBT6</td>
<td>72M</td>
</tr>
<tr>
<td>Maple Mini Bootloader 2.0</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103CBT6</td>
<td>72M</td>
</tr>
<tr>
<td>Maple Mini Original</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103CBT6</td>
<td>72M</td>
</tr>
<tr>
<td>Mbed Connect Cloud</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F439ZIT6</td>
<td>168M</td>
</tr>
<tr>
<td>Microduino Core STM32 to Flash</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103CBT6</td>
<td>72M</td>
</tr>
<tr>
<td>Microsoft Azure IoT Development Kit (MXChip AZ3166)</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F412ZGT6</td>
<td>100M</td>
</tr>
<tr>
<td>Minitronics v2.0</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21J18A</td>
<td>48M</td>
</tr>
<tr>
<td>Moteino M0</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48M</td>
</tr>
<tr>
<td>MultiTech mDot</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411RET6</td>
<td>100M</td>
</tr>
<tr>
<td>MultiTech mDot F411</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411RET6</td>
<td>100M</td>
</tr>
<tr>
<td>MultiTech xDot</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L151CC86</td>
<td>32M</td>
</tr>
<tr>
<td>NAMote72</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L152RC</td>
<td>32M</td>
</tr>
<tr>
<td>NANO 33 IoTF</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48M</td>
</tr>
<tr>
<td>NGX Technologies BlueBoard-LPC11U24</td>
<td>NXP LPC</td>
<td>External</td>
<td>LPC11U24</td>
<td>48M</td>
</tr>
<tr>
<td>NXP LPC11C24</td>
<td>NXP LPC</td>
<td>External</td>
<td>LPC11C24</td>
<td>48M</td>
</tr>
<tr>
<td>NXP LPC11U34</td>
<td>NXP LPC</td>
<td>External</td>
<td>LPC11U34</td>
<td>48M</td>
</tr>
<tr>
<td>NXP LPC11U37</td>
<td>NXP LPC</td>
<td>External</td>
<td>LPC11U37</td>
<td>48M</td>
</tr>
<tr>
<td>NXP LPC800-MAX</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC812</td>
<td>30M</td>
</tr>
<tr>
<td>NXP LPCxpresso1549</td>
<td>NXP LPC</td>
<td>External</td>
<td>LPC1549</td>
<td>72M</td>
</tr>
<tr>
<td>NXP mbed LPC11U24</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC11U24</td>
<td>48M</td>
</tr>
<tr>
<td>NXP mbed LPC1768</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC1768</td>
<td>96M</td>
</tr>
<tr>
<td>Nordic Beacon Kit (PCA20006)</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>32M</td>
</tr>
<tr>
<td>Nordic nRF51X22 Development Kit (PCA1000X)</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>32M</td>
</tr>
<tr>
<td>Nordic nRF52-DK</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52832</td>
<td>64M</td>
</tr>
<tr>
<td>Nordic nRF52840-DK</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52840</td>
<td>64M</td>
</tr>
<tr>
<td>Nucleo G071RB</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32G071RBT6</td>
<td>24M</td>
</tr>
<tr>
<td>OSHChip</td>
<td>Nordic nRF51</td>
<td>External</td>
<td>NRF51822</td>
<td>32M</td>
</tr>
<tr>
<td>P-Nucleo WB55RG</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32WB55RG</td>
<td>64M</td>
</tr>
<tr>
<td>RAK811 LoRa Tracker</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L151RBT6</td>
<td>32M</td>
</tr>
<tr>
<td>RAK811 LoRa Tracker</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L151RBT6</td>
<td>32M</td>
</tr>
<tr>
<td>RedBearLab BLE Nano 1.5</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>16M</td>
</tr>
<tr>
<td>RedBearLab BLE Nano 2</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52832</td>
<td>64M</td>
</tr>
<tr>
<td>RedBearLab BLE Blend 2</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52832</td>
<td>64M</td>
</tr>
<tr>
<td>RedBearLab nRF51822</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>16M</td>
</tr>
<tr>
<td>RushUp Cloud-JAM</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F401RET6</td>
<td>84M</td>
</tr>
</tbody>
</table>

1.16. PIO Unified Debugger
<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>RushUp Cloud-JAM L4</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L476RGT6</td>
<td>80MHz</td>
</tr>
<tr>
<td>SDT52832B</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52832</td>
<td>64MHz</td>
</tr>
<tr>
<td>SLTK3400A USB-enabled Happy Gecko</td>
<td>Silicon Labs EFM32</td>
<td>On-board</td>
<td>EFM32HG322F64</td>
<td>25MHz</td>
</tr>
<tr>
<td>SLTK3401A Pearl Gecko PG1</td>
<td>Silicon Labs EFM32</td>
<td>On-board</td>
<td>EFM32PG1B200F256GM48</td>
<td>40MHz</td>
</tr>
<tr>
<td>SODAQ Autonomo</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21J18A</td>
<td>48MHz</td>
</tr>
<tr>
<td>SODAQ ExpLoRer</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
</tr>
<tr>
<td>SODAQ ONE</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21J18A</td>
<td>48MHz</td>
</tr>
<tr>
<td>SODAQ SFF</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
</tr>
<tr>
<td>ST 32F3348DISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F334C8T6</td>
<td>72MHz</td>
</tr>
<tr>
<td>ST 32F401CDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F401VCST6</td>
<td>84MHz</td>
</tr>
<tr>
<td>ST 32F411EDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F411VET6</td>
<td>100MHz</td>
</tr>
<tr>
<td>ST 32F413HDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F412ZHT6</td>
<td>100MHz</td>
</tr>
<tr>
<td>ST 32F4291DISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F429ZIT6</td>
<td>180MHz</td>
</tr>
<tr>
<td>ST 32F4691DISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F469NHT6</td>
<td>180MHz</td>
</tr>
<tr>
<td>ST 32F746GDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F746NGH6</td>
<td>216MHz</td>
</tr>
<tr>
<td>ST 32F769IDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F769NHT6</td>
<td>216MHz</td>
</tr>
<tr>
<td>ST 32L0538DISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L053C8T6</td>
<td>32MHz</td>
</tr>
<tr>
<td>ST 32L100DISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L100RCT6</td>
<td>32MHz</td>
</tr>
<tr>
<td>ST 32L1476GDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L476VGT6</td>
<td>80MHz</td>
</tr>
<tr>
<td>ST 32L496GDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L496AGI6</td>
<td>80MHz</td>
</tr>
<tr>
<td>ST DISCO-L072CZ-LRWAN1</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L072CZ</td>
<td>32MHz</td>
</tr>
<tr>
<td>ST DISCO-L475VG-IOT01A</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L475VGT6</td>
<td>80MHz</td>
</tr>
<tr>
<td>ST Discovery F072RB</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F072RBT6</td>
<td>48MHz</td>
</tr>
<tr>
<td>ST Nucleo F030R8</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F030R8T6</td>
<td>48MHz</td>
</tr>
<tr>
<td>ST Nucleo F031K6</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F031K6T6</td>
<td>48MHz</td>
</tr>
<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 F103RB</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>STM32F302RBT6</td>
<td>72MHz</td>
</tr>
<tr>
<td>ST Nucleo F303K8</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F303KBT6</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>STM32F334RBT6</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>STM32F411RBT6</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>STM32F756ZGT6</td>
<td>216MHz</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>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>STM32L073RZ</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>STM32L412KBU6</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-P</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>STM32L452RE76</td>
<td>80MHz</td>
</tr>
<tr>
<td>ST Nucleo L476RG</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L476RGT6</td>
<td>80MHz</td>
</tr>
<tr>
<td>ST Nucleo L496ZG</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L496ZGT6</td>
<td>80MHz</td>
</tr>
<tr>
<td>ST Nucleo L496ZG-P</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L496ZGT6P</td>
<td>80MHz</td>
</tr>
<tr>
<td>ST Nucleo L453ZI</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L453ZIT6</td>
<td>120MHz</td>
</tr>
<tr>
<td>STM32F0308DISCOVERY</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F030R8T6</td>
<td>48MHz</td>
</tr>
<tr>
<td>STM32F0DISCOVERY</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F051R8T6</td>
<td>48MHz</td>
</tr>
<tr>
<td>STM32F103VCT6</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F030VCT6</td>
<td>72MHz</td>
</tr>
<tr>
<td>STM32F4DISCOVERY</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407VGT6</td>
<td>168MHz</td>
</tr>
<tr>
<td>ST STM32L073Z-EVAL</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L073ZVT6</td>
<td>32MHz</td>
</tr>
<tr>
<td>ST STM32L6DISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L152RB76</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>STM32L476G</td>
<td>80MHz</td>
</tr>
<tr>
<td>ST32F030CH (20k RAM. 64k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
</tr>
<tr>
<td>ST32F030CB (20k RAM. 128k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103CBT6</td>
<td>72MHz</td>
</tr>
<tr>
<td>ST32F030RR (20k RAM. 64 Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103R8T6</td>
<td>72MHz</td>
</tr>
<tr>
<td>ST32F030RB (20k RAM. 128 Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103RBT6</td>
<td>72MHz</td>
</tr>
<tr>
<td>ST32F030RC (48k RAM. 256 Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103RCT6</td>
<td>72MHz</td>
</tr>
<tr>
<td>ST32F030RE (64k RAM. 512 Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103RET6</td>
<td>72MHz</td>
</tr>
<tr>
<td>ST32F030TS (20k RAM. 64 Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103T8T6</td>
<td>72MHz</td>
</tr>
<tr>
<td>ST32F030TB (20k RAM. 128 Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103TBT6</td>
<td>72MHz</td>
</tr>
<tr>
<td>ST32F030VC (48k RAM. 256 Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103VCT6</td>
<td>72MHz</td>
</tr>
<tr>
<td>ST32F030VD (64k RAM. 384 Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103VD76</td>
<td>72MHz</td>
</tr>
<tr>
<td>ST32F030VE (64k RAM. 512 Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103VET6</td>
<td>72MHz</td>
</tr>
<tr>
<td>ST32F030ZC (48k RAM. 256 Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103ZCT6</td>
<td>72MHz</td>
</tr>
<tr>
<td>ST32F030ZD (64k RAM. 384 Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103ZDT6</td>
<td>72MHz</td>
</tr>
<tr>
<td>ST32F030ZE (64k RAM. 512 Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103ZET6</td>
<td>72MHz</td>
</tr>
<tr>
<td>ST32F030CB (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>STM32F7508H6</td>
<td>216MHz</td>
</tr>
<tr>
<td>SainSmart Due (Programming Port)</td>
<td>Atmel SAM</td>
<td>External</td>
<td>AT91SAM3X8E</td>
<td>84MHz</td>
</tr>
<tr>
<td>SainSmart Due (USB Native Port)</td>
<td>Atmel SAM</td>
<td>External</td>
<td>AT91SAM3X8E</td>
<td>84MHz</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 Tiny BLE</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>nRF51822</td>
<td>16MHz</td>
</tr>
<tr>
<td>Seeeduino LoRaWAN</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
</tr>
</tbody>
</table>

1.16. PIO Unified Debugger
Table 28 – continued from previous page

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sino:Bit</td>
<td>Nordic nRF51</td>
<td>External</td>
<td>NRF51822</td>
<td>32MHz</td>
</tr>
<tr>
<td>Solder Splash Labs DipCortex M0</td>
<td>NXP LPC</td>
<td>External</td>
<td>LPC11U24</td>
<td>50MHz</td>
</tr>
<tr>
<td>SparkFun SAMD21 Dev Breakout</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
</tr>
<tr>
<td>SparkFun SAMD21 Mini Breakout</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
</tr>
<tr>
<td>Sparky V1 F303</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F303CCT6</td>
<td>72MHz</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>Taida Century nRF52 mini board</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52832</td>
<td>64MHz</td>
</tr>
<tr>
<td>Thunderboard Sense 2 Sensor-to-Cloud Advanced IoT</td>
<td>Silicon Labs EFM32</td>
<td>On-board</td>
<td>EFR32MG12P33F1024</td>
<td>40MHz</td>
</tr>
<tr>
<td>Tiny STM103T</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103TBU6</td>
<td>72MHz</td>
</tr>
<tr>
<td>Tuino 096</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
</tr>
<tr>
<td>VAKE v1.0</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F446RET6</td>
<td>180MHz</td>
</tr>
<tr>
<td>Waveshare BLE400</td>
<td>Nordic nRF51</td>
<td>External</td>
<td>NRF51822</td>
<td>32MHz</td>
</tr>
<tr>
<td>hackaBLE</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52832</td>
<td>64MHz</td>
</tr>
<tr>
<td>ng-beacon</td>
<td>Nordic nRF51</td>
<td>External</td>
<td>NRF51822</td>
<td>16MHz</td>
</tr>
<tr>
<td>sakura.io Evaluation Board</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F411RET6</td>
<td>100MHz</td>
</tr>
<tr>
<td>u-blox C027</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC1768</td>
<td>96MHz</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>
</tr>
<tr>
<td>u-blox C030-R410M IoT</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F437VG</td>
<td>180MHz</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>
</tr>
<tr>
<td>u-blox EVK-NINA-B1</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52832</td>
<td>64MHz</td>
</tr>
<tr>
<td>u-blox EVK-ODIN-W2</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F439ZIY6</td>
<td>168MHz</td>
</tr>
<tr>
<td>u-blox ODIN-W2</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F439ZIY6</td>
<td>168MHz</td>
</tr>
<tr>
<td>y5 LPC11U33 mbug</td>
<td>NXP LPC</td>
<td>External</td>
<td>LPC11U35</td>
<td>48MHz</td>
</tr>
<tr>
<td>y5 nRF51822 mbug</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>16MHz</td>
</tr>
</tbody>
</table>

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.

Contents

- Configuration
- Drivers
- Platforms
**Configuration**

You can configure debugging tool using `debug_tool` option in "platformio.ini" (Project Configuration File):

```ini
[env:myenv]
platform = ...
board = ...
double_tool = cmsis-dap
```

If you would like to use this tool for firmware uploading, please change upload protocol:

```ini
[env:myenv]
platform = ...
board = ...
double_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><strong>Atmel SAM</strong></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><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>Maxim 32</strong></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><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 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>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>WIZ-Net W7500</strong></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>libOpenCM3</td>
<td>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.</td>
</tr>
<tr>
<td>mbed</td>
<td>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.</td>
</tr>
<tr>
<td>Simba</td>
<td>Simba is an RTOS and build framework. It aims to make embedded programming easy and portable.</td>
</tr>
<tr>
<td>SPL</td>
<td>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.</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>

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>
</tr>
</thead>
<tbody>
<tr>
<td>ARM mbed LPC11U24 (+CAN)</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC11U24</td>
<td>48MHz</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>
</tr>
<tr>
<td>Arduino Zero (Programming/Debug Port)</td>
<td>Atmel SAM</td>
<td>On-board</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
</tr>
<tr>
<td>Atmel ATSAMR21-XPRO</td>
<td>Atmel SAM</td>
<td>On-board</td>
<td>SAMR21G18A</td>
<td>48MHz</td>
</tr>
<tr>
<td>Atmel ATSAMW25-XPRO</td>
<td>Atmel SAM</td>
<td>On-board</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
</tr>
<tr>
<td>Atmel SAMD21-XPRO</td>
<td>Atmel SAM</td>
<td>On-board</td>
<td>SAMD21J18A</td>
<td>48MHz</td>
</tr>
<tr>
<td>Atmel SAML21-XPRO-B</td>
<td>Atmel SAM</td>
<td>On-board</td>
<td>SAML21J18B</td>
<td>48MHz</td>
</tr>
<tr>
<td>BBC micro:bit</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>16MHz</td>
</tr>
<tr>
<td>Bambino-210E</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC4330</td>
<td>204MHz</td>
</tr>
<tr>
<td>Calliope mini</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>16MHz</td>
</tr>
<tr>
<td>CoCo-ri-Co!</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>LPC812</td>
<td>38MHz</td>
</tr>
<tr>
<td>Delta DFBM-NQ620</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52382</td>
<td>64MHz</td>
</tr>
<tr>
<td>Delta DFCM-NNN40</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>32MHz</td>
</tr>
<tr>
<td>Delta DFCM-NNN50</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>32MHz</td>
</tr>
<tr>
<td>Embedded Artists LPC4088 Display Module</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC4088</td>
<td>120MHz</td>
</tr>
<tr>
<td>Embedded Artists LPC4088 QuickStart Board</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC4088</td>
<td>120MHz</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>Ethernet IoT Starter Kit</td>
<td>Freescale Kinetis</td>
<td>On-board</td>
<td>MK64FN1M0VLL12</td>
<td>120MHz</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>MK66FN2M0VMD18</td>
<td>180MHz</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-K82F</td>
<td>Freescale Kinetis</td>
<td>On-board</td>
<td>MK82F256VLL15</td>
<td>150MHz</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-KL05Z</td>
<td>Freescale Kinetis</td>
<td>On-board</td>
<td>MKL05Z32VFM4</td>
<td>48MHz</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-KL20D50M</td>
<td>Freescale Kinetis</td>
<td>On-board</td>
<td>MK20DX128VLK4</td>
<td>48MHz</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-KL22F</td>
<td>Freescale Kinetis</td>
<td>On-board</td>
<td>MKL22Z64VLL4</td>
<td>48MHz</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-KL42F</td>
<td>Freescale Kinetis</td>
<td>On-board</td>
<td>MKL42Z256VLL4</td>
<td>48MHz</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-KL46F</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>MKW41Z512VHT4</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>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>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>Moteino M0</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</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>Nordic nRF51 Dongle (PCA10031)</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>32MHz</td>
</tr>
<tr>
<td>Nordic nRF51822-nKIT</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 nRF52-32DK</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>RedBearLab BLE Nano 1.5</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>16MHz</td>
</tr>
<tr>
<td>RedBearLab BLE Nano 2</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52832</td>
<td>64MHz</td>
</tr>
<tr>
<td>RedBearLab BLE Nano 3</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52832</td>
<td>64MHz</td>
</tr>
<tr>
<td>RedBearLab BLE Nano 4</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52832</td>
<td>64MHz</td>
</tr>
<tr>
<td>SDT52832B</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52832</td>
<td>64MHz</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 Pro</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>16MHz</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>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>Nordic nRF51</td>
<td>On-board</td>
<td>LPC1114FN28</td>
<td>48MHz</td>
</tr>
<tr>
<td>Switch Science mbed LPC824</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>LPC824</td>
<td>30MHz</td>
</tr>
<tr>
<td>Switch Science mbed TY51822r3</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>32MHz</td>
</tr>
<tr>
<td>VNG VBLUNO51</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>16MHz</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>sakura.io Evaluation Board</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F411RET6</td>
<td>100MHz</td>
</tr>
</tbody>
</table>
Table 29 – continued from previous page

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>u-blox C027</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC1768</td>
<td>96MHz</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>
</tr>
<tr>
<td>u-blox C030-U201 IoT Starter Kit</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F437VG</td>
<td>180MHz</td>
</tr>
<tr>
<td>y5 nRF51822 mbug</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>16MHz</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
- Platforms
- Frameworks
- Boards

Configuration

You can configure debugging tool using `debug_tool` option in "platformio.ini" (Project Configuration File):

```
[env:myenv]
platform = ...
board = ...
download_subdir = ...
download_tool = ...
download_port = ...
download_protocol = ...
download_speed = ...
```

If you would like to use this tool for firmware uploading, please change upload protocol:
[env:myenv]
platform = ...
board = ...
debug_tool = esp-prog
upload_protocol = esp-prog

More options:

- *Debugging options*
- *Upload options*

**Drivers**

**Windows**  See https://community.platformio.org/t/esp32-pio-unified-debugger/4541/20

**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>ESP-Prog JTAG 10-Pin Connector</th>
<th>Board JTAG Pin</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>VDD</td>
<td>Positive Supply Voltage — Power supply for JTAG interface drivers</td>
</tr>
<tr>
<td>3</td>
<td>GND</td>
<td>Digital ground</td>
</tr>
<tr>
<td>2</td>
<td>ESP_TMS</td>
<td>Test Mode State</td>
</tr>
<tr>
<td>4</td>
<td>ESP_TCK</td>
<td>JTAG Return Test Clock</td>
</tr>
<tr>
<td>6</td>
<td>ESP_TDO</td>
<td>Test Data Out</td>
</tr>
<tr>
<td>8</td>
<td>ESP_TDI</td>
<td>Test Data In</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>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached 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. It 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>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>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>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>Dongen 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>ESP32v2n 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>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>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>OLMEX 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>OLMEX 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>Silicognition wES32</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 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>WEMOS LOLIN D32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
</tbody>
</table>
### 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>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 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):

```ini
[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** See [https://community.platformio.org/t/esp32-pio-unified-debugger/4541/20](https://community.platformio.org/t/esp32-pio-unified-debugger/4541/20)

**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>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>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>Samsung AR-TIK</td>
<td>The Samsung ARTIK Smart IoT platform brings hardware modules and cloud services together, with built-in security and an ecosystem of tools and partners to speed up your time-to-market.</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>Freedom E SDK</td>
<td>Open Source Software for Developing on the SiFive Freedom E Platform</td>
</tr>
<tr>
<td>mbed</td>
<td>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.</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>Simba</td>
<td>Simba is an RTOS and build framework. It aims to make embedded programming easy and portable.</td>
</tr>
<tr>
<td>Tizen RT</td>
<td>Tizen RT is a lightweight RTOS-based platform to support low-end IoT devices</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>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>FUS40</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>Samsung ARTIK053</td>
<td>Samsung ARTIK</td>
<td>On-board</td>
<td>S5JT200</td>
<td>320MHz</td>
<td>8MB</td>
<td>1.25MB</td>
</tr>
</tbody>
</table>
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.

### 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 = ...
ddebug_tool = iot-bus-jtag
```

If you would like to use this tool for firmware uploading, please change upload protocol:

```ini
[env:myenv]
platform = ...
board = ...
ddebug_tool = iot-bus-jtag
upload_protocol = iot-bus-jtag
```

More options:

- **Debugging options**
- **Upload options**
Drivers

**Windows**  See https://community.platformio.org/t/esp32-pio-unified-debugger/4541/20

**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 Board Pin</th>
<th>JTAG 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>12</td>
<td>TDI</td>
<td>Test Data In pin</td>
</tr>
<tr>
<td>14</td>
<td>TMS</td>
<td>Test Mode State pin</td>
</tr>
<tr>
<td>13</td>
<td>TCK</td>
<td>JTAG Return Test Clock</td>
</tr>
<tr>
<td>15</td>
<td>TDO</td>
<td>Test Data Out pin</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>
</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, this platform is ported by Sipeed.</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached 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 Standalone SDK</td>
<td>Kendryte Standalone SDK without OS support</td>
</tr>
<tr>
<td>Kendryte FreeRTOS SDK</td>
<td>Kendryte SDK with FreeRTOS support</td>
</tr>
<tr>
<td>Simba</td>
<td>Simba is an RTOS and build framework. It 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>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>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>D-duinou 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>ESP32n 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>Expressif 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>MH ET LIVE 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>MH ET LIVE ESP32 MiniKit</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-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>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>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 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>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 LOLIN-32</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
- 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:

```
[env:myenv]
platform = ...
board = ...
```

(continues on next page)
`debug_tool = jlink`

; SWD interface
`upload_protocol = jlink`

; JTAG interface
`upload_protocol = jlink-jtag`

More options:

- **Debugging options**
- **Upload options**

**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 VCC</td>
<td></td>
<td>Positive Supply Voltage — Power supply for JTAG interface drivers</td>
</tr>
<tr>
<td>4 GND</td>
<td></td>
<td>Digital ground</td>
</tr>
<tr>
<td>5 TDI</td>
<td></td>
<td>Test Data In pin</td>
</tr>
<tr>
<td>7 TMS</td>
<td></td>
<td>Test Mode State pin</td>
</tr>
<tr>
<td>9 TCK</td>
<td></td>
<td>JTAG Return Test Clock</td>
</tr>
<tr>
<td>13 TDO</td>
<td></td>
<td>Test Data Out pin</td>
</tr>
<tr>
<td>15 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>

## 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 VCC</td>
<td></td>
<td>Positive Supply Voltage — Power supply for JTAG interface drivers</td>
</tr>
<tr>
<td>4 GND</td>
<td></td>
<td>Digital ground</td>
</tr>
<tr>
<td>7 SWDIO</td>
<td></td>
<td>Data I/O</td>
</tr>
<tr>
<td>9 SWCLK</td>
<td></td>
<td>Clock</td>
</tr>
<tr>
<td>15 RESET</td>
<td></td>
<td>Connect this pin to the (active low) reset input of the target CPU</td>
</tr>
<tr>
<td>Name</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>---------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td></td>
</tr>
<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>
<td></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>
<td></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>
<td></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>
<td></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>
<td></td>
</tr>
<tr>
<td>Kendryte K210</td>
<td>Kendryte K210 is an AI capable RISCV64 dual core SoC, this platform is ported by Sipeed.</td>
<td></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>
<td></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>
<td></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>
<td></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>
<td></td>
</tr>
<tr>
<td>SilFive</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>
<td></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>
<td></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>
<td></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>
<td></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>
<td></td>
</tr>
</tbody>
</table>
Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached 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>Freedom E SDK</td>
<td>Open Source Software for Developing on the SiFive Freedom E Platform</td>
</tr>
<tr>
<td>Kendryte Standalone SDK</td>
<td>Kendryte Standalone SDK without OS support</td>
</tr>
<tr>
<td>Kendryte FreeRTOS SDK</td>
<td>Kendryte SDK with FreeRTOS support</td>
</tr>
<tr>
<td>libOpenCM3</td>
<td>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.</td>
</tr>
<tr>
<td>mbed</td>
<td>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.</td>
</tr>
<tr>
<td>Simba</td>
<td>Simba is an RTOS and build framework. It aims to make embedded programming easy and portable.</td>
</tr>
<tr>
<td>SPL</td>
<td>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.</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>
</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>168MHz</td>
</tr>
<tr>
<td>32F723EDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F723IEK6</td>
<td>216MHz</td>
</tr>
</tbody>
</table>

Note: For more detailed board information please scroll tables below by horizontal.

1.16. PIO Unified Debugger
<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>3D Printer Controller</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
</tr>
<tr>
<td>3D Printer control board</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F446RET6</td>
<td>180MHz</td>
</tr>
<tr>
<td>3D printer controller</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F765VIT6</td>
<td>216MHz</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>
</tr>
<tr>
<td>96Boards B96B-F446VE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F446VET6</td>
<td>168MHz</td>
</tr>
<tr>
<td>ALKS ESP-32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
</tr>
<tr>
<td>ARM mbed LPC11U24 (+CAN)</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC11U24</td>
<td>48MHz</td>
</tr>
<tr>
<td>Aceinna OpenIMU 300ZA</td>
<td>Aceinna IMU</td>
<td>External</td>
<td>STM32F405RG</td>
<td>120MHz</td>
</tr>
<tr>
<td>Adafruit Bluefruit nRF52832 Feather</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52832</td>
<td>64MHz</td>
</tr>
<tr>
<td>Adafruit Circuit Playground Express</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
</tr>
<tr>
<td>Adafruit Cricket M0</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
</tr>
<tr>
<td>Adafruit ESP32 Feather</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
</tr>
<tr>
<td>Adafruit Feather M0</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
</tr>
<tr>
<td>Adafruit Feather M0 Express</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
</tr>
<tr>
<td>Adafruit Feather M4 Express</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD51J19A</td>
<td>120MHz</td>
</tr>
<tr>
<td>Adafruit Feather nRF52840 Express</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52840</td>
<td>64MHz</td>
</tr>
<tr>
<td>Adafruit Gemma M0</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21E18A</td>
<td>48MHz</td>
</tr>
<tr>
<td>Adafruit Grand Central M4</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD51P20A</td>
<td>120MHz</td>
</tr>
<tr>
<td>Adafruit Hallowing M0</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
</tr>
<tr>
<td>Adafruit ItsyBitsy M0</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
</tr>
<tr>
<td>Adafruit ItsyBitsy M4</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD51G19A</td>
<td>120MHz</td>
</tr>
<tr>
<td>Adafruit Metro M0 Express</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
</tr>
<tr>
<td>Adafruit Metro M4</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD51J19A</td>
<td>120MHz</td>
</tr>
<tr>
<td>Adafruit Metro M4 AirLift Lite</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD51J19A</td>
<td>120MHz</td>
</tr>
<tr>
<td>Adafruit PyGamer Advance M4</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD51J20A</td>
<td>120MHz</td>
</tr>
<tr>
<td>Adafruit PyGamer M4 Express</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD51J19A</td>
<td>120MHz</td>
</tr>
<tr>
<td>Adafruit PyPortal M4</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD51J20A</td>
<td>120MHz</td>
</tr>
<tr>
<td>Adafruit Trellis M4</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD51J19A</td>
<td>120MHz</td>
</tr>
<tr>
<td>Adafruit Trinket M0</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21E18A</td>
<td>48MHz</td>
</tr>
<tr>
<td>Adafruit pIRkey</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21E18A</td>
<td>48MHz</td>
</tr>
<tr>
<td>Adafruit pyBadge M4 Express</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD51J19A</td>
<td>120MHz</td>
</tr>
<tr>
<td>Arduino Due (Programming Port)</td>
<td>Atmel SAM</td>
<td>External</td>
<td>AT91SAM3X8E</td>
<td>84MHz</td>
</tr>
<tr>
<td>Arduino Due (USB Native Port)</td>
<td>Atmel SAM</td>
<td>External</td>
<td>AT91SAM3X8E</td>
<td>84MHz</td>
</tr>
<tr>
<td>Arduino M0</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
</tr>
<tr>
<td>Arduino M0 Pro (Native USB Port)</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</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>
</tr>
<tr>
<td>Arduino MKR FOX 1200</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
</tr>
<tr>
<td>Arduino MKR GSM 1400</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
</tr>
<tr>
<td>Arduino MKR NB 1500</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
</tr>
<tr>
<td>Arduino MKR WAN 1300</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
</tr>
<tr>
<td>Arduino MKR WiFi 1010</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
</tr>
<tr>
<td>Arduino MKR1000</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
</tr>
<tr>
<td>Arduino MKRZERO</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
</tr>
<tr>
<td>Arduino Tian</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
</tr>
<tr>
<td>Arduino Zero (Programming/Debug Port)</td>
<td>Atmel SAM</td>
<td>On-board</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
</tr>
<tr>
<td>Arduino Zero (USB Native Port)</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
</tr>
<tr>
<td>Armstrap Eagle 1024</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F417VGT6</td>
<td>168MHz</td>
</tr>
<tr>
<td>Armstrap Eagle 2048</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F427VIT6</td>
<td>168MHz</td>
</tr>
<tr>
<td>Armstrap Eagle 512</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407VET6</td>
<td>168MHz</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>Arty FPGA Dev Kit</td>
<td>SiFive</td>
<td>On-board</td>
<td>FE310</td>
<td>450MHz</td>
</tr>
<tr>
<td>Atmel ATSAM21-XPRO</td>
<td>Atmel SAM</td>
<td>On-board</td>
<td>SAMR21G18A</td>
<td>48MHz</td>
</tr>
<tr>
<td>Atmel ATSAMW25-XPRO</td>
<td>Atmel SAM</td>
<td>On-board</td>
<td>SAML21G18A</td>
<td>48MHz</td>
</tr>
<tr>
<td>Atmel SAMD21-XPRO</td>
<td>Atmel SAM</td>
<td>On-board</td>
<td>SAMD21J18A</td>
<td>48MHz</td>
</tr>
<tr>
<td>Atmel SAML21-XPRO-B</td>
<td>Atmel SAM</td>
<td>On-board</td>
<td>SAML21J18B</td>
<td>48MHz</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>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>168MHz</td>
</tr>
<tr>
<td>Black STM32F407VG</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407VG6</td>
<td>168MHz</td>
</tr>
<tr>
<td>Black STM32F407ZE</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407ZGT6</td>
<td>168MHz</td>
</tr>
<tr>
<td>Black STM32F407ZE</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407ZET6</td>
<td>168MHz</td>
</tr>
<tr>
<td>BlackPill F103C8</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
</tr>
<tr>
<td>BlackPill F103C8 (128k)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
</tr>
<tr>
<td>BlackPill F303CC</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F303CCT6</td>
<td>72MHz</td>
</tr>
<tr>
<td>Blue STM32F407VE Mini</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
</tr>
<tr>
<td>BluePill F103C6</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103C6T6</td>
<td>72MHz</td>
</tr>
<tr>
<td>BluePill F103C8</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
</tr>
<tr>
<td>BluePill F103C8 (128k)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
</tr>
<tr>
<td>Bluey nRF52832 IoT</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52832</td>
<td>64MHz</td>
</tr>
<tr>
<td>BluzDK</td>
<td>Nordic nRF51</td>
<td>External</td>
<td>NRF51822</td>
<td>32MHz</td>
</tr>
<tr>
<td>CQ Publishing TG-LPC11U35-501</td>
<td>NXP LPC</td>
<td>External</td>
<td>LPC11U35</td>
<td>48MHz</td>
</tr>
<tr>
<td>Callioppe mini</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>16M</td>
</tr>
<tr>
<td>CoCo-ri-Co!</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC812</td>
<td>30M</td>
</tr>
<tr>
<td>D-daino-32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240M</td>
</tr>
<tr>
<td>DOIT ESP32 DEVKIT V1</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240M</td>
</tr>
<tr>
<td>Delta DFBM-NQ620</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52832</td>
<td>64MHz</td>
</tr>
<tr>
<td>Demo F030F4</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F030F4P6</td>
<td>48MHz</td>
</tr>
<tr>
<td>Dipstick DigiX</td>
<td>Atmel SAM</td>
<td>External</td>
<td>AT91SAM3X8E</td>
<td>84MHz</td>
</tr>
<tr>
<td>DipCortex M3</td>
<td>NXP LPC</td>
<td>External</td>
<td>LPC1347</td>
<td>72MHz</td>
</tr>
<tr>
<td>Dongsen Tech Pocket 32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240M</td>
</tr>
<tr>
<td>EA LPC11U35 QuickStart Board</td>
<td>NXP LPC</td>
<td>External</td>
<td>LPC11U35</td>
<td>48MHz</td>
</tr>
<tr>
<td>ESP32 FM DevKit</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240M</td>
</tr>
<tr>
<td>ESP32vn IoT Uno</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240M</td>
</tr>
<tr>
<td>ESPectro32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240M</td>
</tr>
<tr>
<td>ESPino32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240M</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>100MHz</td>
</tr>
<tr>
<td>Espressif ESP-WROVER-KIT</td>
<td>Espressif 32</td>
<td>On-board</td>
<td>ESP32</td>
<td>240M</td>
</tr>
<tr>
<td>Espressif ESP32 Dev Module</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240M</td>
</tr>
<tr>
<td>Ethernet IoT Starter Kit</td>
<td>Freescale Kinetis</td>
<td>On-board</td>
<td>MK64FN1M0VLL12</td>
<td>120M</td>
</tr>
<tr>
<td>F407VG</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407VG6</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>FireBeetle-ESP32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240M</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-K20D50M</td>
<td>Freescale Kinetis</td>
<td>On-board</td>
<td>MK20DX128VLH5</td>
<td>48M</td>
</tr>
</tbody>
</table>
Table 32 – continued from previous page

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Freescale Kinetis FRDM-K22F</td>
<td>Freescale Kinetis</td>
<td>On-board</td>
<td>MK22FNN512VLH12</td>
<td>120M</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-K64F</td>
<td>Freescale Kinetis</td>
<td>On-board</td>
<td>MK64FNN1M0VLL12</td>
<td>120M</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-K66F</td>
<td>Freescale Kinetis</td>
<td>On-board</td>
<td>MK66FNN2M0VMD18</td>
<td>180M</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-K82F</td>
<td>Freescale Kinetis</td>
<td>On-board</td>
<td>MK82FNN256VLL15</td>
<td>150M</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-KL05Z</td>
<td>Freescale Kinetis</td>
<td>On-board</td>
<td>MKL05ZS32VFM4</td>
<td>48M</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-KL25Z</td>
<td>Freescale Kinetis</td>
<td>On-board</td>
<td>MKL25Z128VLLK4</td>
<td>48M</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-KL27Z</td>
<td>Freescale Kinetis</td>
<td>On-board</td>
<td>MKL27Z64VLLH4</td>
<td>48M</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-KL43Z</td>
<td>Freescale Kinetis</td>
<td>On-board</td>
<td>MKL43Z256VLLH4</td>
<td>48M</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-KL46Z</td>
<td>Freescale Kinetis</td>
<td>On-board</td>
<td>MKL46Z256VLL4</td>
<td>48M</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-KL82Z</td>
<td>Freescale Kinetis</td>
<td>External</td>
<td>MKL8Z128VLLK7</td>
<td>96M</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-KW24D512</td>
<td>Freescale Kinetis</td>
<td>External</td>
<td>MKW24D512</td>
<td>50M</td>
</tr>
<tr>
<td>Freescale Kinetis FRDM-KW41Z</td>
<td>Freescale Kinetis</td>
<td>On-board</td>
<td>MKW41ZS125VHT4</td>
<td>48M</td>
</tr>
<tr>
<td>Frog Board ESP32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240M</td>
</tr>
<tr>
<td>Heltec WiFi LoRa 32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240M</td>
</tr>
<tr>
<td>Heltec WiFi LoRa 32 (V2)</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240M</td>
</tr>
<tr>
<td>Heltec Wireless Stick</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240M</td>
</tr>
<tr>
<td>Hexiwear</td>
<td>Freescale Kinetis</td>
<td>External</td>
<td>MK64FNN1M0VDC12</td>
<td>120M</td>
</tr>
<tr>
<td>HiFive I Rev B</td>
<td>SiFive</td>
<td>On-board</td>
<td>FE310</td>
<td>320M</td>
</tr>
<tr>
<td>Hornbill ESP32 Dev</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240M</td>
</tr>
<tr>
<td>Hornbill ESP32 Minima</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240M</td>
</tr>
<tr>
<td>L476DMW1K</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L476VGT6</td>
<td>80M</td>
</tr>
<tr>
<td>LPCXpresso11U68</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC11U68</td>
<td>50M</td>
</tr>
<tr>
<td>LPCXpresso824-MAX</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC824</td>
<td>30M</td>
</tr>
<tr>
<td>M200 V2</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F070CBT6</td>
<td>48M</td>
</tr>
<tr>
<td>MAX32620FTHR</td>
<td>Maxim 32</td>
<td>External</td>
<td>MAX32620FTHR</td>
<td>96M</td>
</tr>
<tr>
<td>MH ET LIVE ESP32DevKit</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240M</td>
</tr>
<tr>
<td>MH ET LIVE ESP32MiniKit</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240M</td>
</tr>
<tr>
<td>MKR Vidor 4000</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48M</td>
</tr>
<tr>
<td>MTS Dragonfly</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411RET6</td>
<td>100M</td>
</tr>
<tr>
<td>Macchina M2</td>
<td>Atmel SAM</td>
<td>External</td>
<td>AT91SAM3X8E</td>
<td>84M</td>
</tr>
<tr>
<td>Malyan M200 V1</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103CBT6</td>
<td>72M</td>
</tr>
<tr>
<td>Maple</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103CBT6</td>
<td>72M</td>
</tr>
<tr>
<td>Maple (RET6)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103CBT6</td>
<td>72M</td>
</tr>
<tr>
<td>Maple Mini Bootloader 2.0</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103CBT6</td>
<td>72M</td>
</tr>
<tr>
<td>Maple Mini Original</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103CBT6</td>
<td>72M</td>
</tr>
<tr>
<td>Maxim Health Sensor Platform</td>
<td>Maxim 32</td>
<td>External</td>
<td>MAX32620</td>
<td>96M</td>
</tr>
<tr>
<td>Mbed Connect Cloud</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F439ZIY6</td>
<td>168M</td>
</tr>
<tr>
<td>Microduino Core STM32 to Flash</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103CBT6</td>
<td>72M</td>
</tr>
<tr>
<td>Microsoft Azure IoT Development Kit (MXChip AZ3166)</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F412ZGT6</td>
<td>100M</td>
</tr>
<tr>
<td>Mintronitons v2.0</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21J18A</td>
<td>48M</td>
</tr>
<tr>
<td>Moteinno M0</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48M</td>
</tr>
<tr>
<td>MultiTech mDot</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411RET6</td>
<td>100M</td>
</tr>
<tr>
<td>MultiTech mDot F411</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411RET6</td>
<td>100M</td>
</tr>
<tr>
<td>MultiTech xDot</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L151CCU6</td>
<td>32M</td>
</tr>
<tr>
<td>N2+</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F405SRG6</td>
<td>168M</td>
</tr>
<tr>
<td>NAMote72</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L152RC</td>
<td>32M</td>
</tr>
<tr>
<td>NANO 33 IoT</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48M</td>
</tr>
<tr>
<td>NGX Technologies BlueBoard-LPC11U24</td>
<td>NXP LPC</td>
<td>External</td>
<td>LPC11U24</td>
<td>48M</td>
</tr>
<tr>
<td>NXP LPC11C24</td>
<td>NXP LPC</td>
<td>External</td>
<td>LPC11C24</td>
<td>48M</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>NXP LPC11U34</td>
<td>NXP LPC</td>
<td>External</td>
<td>LPC11U34</td>
<td>48MHz</td>
</tr>
<tr>
<td>NXP LPC11U37</td>
<td>NXP LPC</td>
<td>External</td>
<td>LPC11U37</td>
<td>48MHz</td>
</tr>
<tr>
<td>NXP LPC800-MAX</td>
<td>NXP LPC</td>
<td>External</td>
<td>LPC812</td>
<td>30MHz</td>
</tr>
<tr>
<td>NXP LPCXpresso1549</td>
<td>NXP LPC</td>
<td>External</td>
<td>LPC1549</td>
<td>72MHz</td>
</tr>
<tr>
<td>NXP LPCXpresso54114</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC5411J256BD64</td>
<td>100MHz</td>
</tr>
<tr>
<td>NXP LPCXpresso54608</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC54608ET512</td>
<td>180MHz</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>Node32s</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
</tr>
<tr>
<td>NodeMCU-32S</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</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>Nordic nRF51 Dongle (PCA10031)</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>32MHz</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>Nucleo G071RB</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32G071RBTT</td>
<td>24MHz</td>
</tr>
<tr>
<td>OLIMEX ESP32-EVB</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
</tr>
<tr>
<td>OLIMEX ESP32-GATEWAY</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
</tr>
<tr>
<td>OSHChip</td>
<td>Nordic nRF51</td>
<td>External</td>
<td>NRF51822</td>
<td>32MHz</td>
</tr>
<tr>
<td>P-Nucleo WB55RG</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32WB55RG</td>
<td>64MHz</td>
</tr>
<tr>
<td>Pycom LoPy</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
</tr>
<tr>
<td>Pycom LoPy4</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
</tr>
<tr>
<td>RAK811 LoRa Tracker</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L151RBTT</td>
<td>32MHz</td>
</tr>
<tr>
<td>RedBearLab BLE Nano 1.5</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>16MHz</td>
</tr>
<tr>
<td>RedBearLab BLE Nano 2</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52832</td>
<td>64MHz</td>
</tr>
<tr>
<td>RedBearLab BLE Nano 3</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52832</td>
<td>64MHz</td>
</tr>
<tr>
<td>RedBearLab BLE Nano 4</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52832</td>
<td>64MHz</td>
</tr>
<tr>
<td>RushUp Cloud-JAM</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L476GRT6</td>
<td>80MHz</td>
</tr>
<tr>
<td>SODAQ Autonomo</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21J18A</td>
<td>48MHz</td>
</tr>
<tr>
<td>SODAQ ExpLoRe</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21J18A</td>
<td>48MHz</td>
</tr>
<tr>
<td>SODAQ ONE</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
</tr>
<tr>
<td>SODAQ SARA</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21J18A</td>
<td>48MHz</td>
</tr>
<tr>
<td>SODAQ SFF</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
</tr>
<tr>
<td>ST 32F3348DISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F334C8T6</td>
<td>72MHz</td>
</tr>
<tr>
<td>ST 32F401CDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F401VC6</td>
<td>84MHz</td>
</tr>
<tr>
<td>ST 32F411EDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F411VB6</td>
<td>100MHz</td>
</tr>
<tr>
<td>ST 32F413HDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F413ZHT6</td>
<td>100MHz</td>
</tr>
<tr>
<td>ST 32F429IDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F429ZHT6</td>
<td>180MHz</td>
</tr>
<tr>
<td>ST 32F469IDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F469NH6</td>
<td>180MHz</td>
</tr>
<tr>
<td>ST 32F746GDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F746NGH6</td>
<td>216MHz</td>
</tr>
<tr>
<td>ST 32F769IDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F769NH6</td>
<td>216MHz</td>
</tr>
<tr>
<td>ST 32L0538DISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L053C8T6</td>
<td>32MHz</td>
</tr>
<tr>
<td>ST 32L100DISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L100RCT6</td>
<td>32MHz</td>
</tr>
</tbody>
</table>

1.16. PIO Unified Debugger

1697
<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>ST 32L476GDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L476VGT6</td>
<td>80MHz</td>
</tr>
<tr>
<td>ST 32L496GDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L496AGI6</td>
<td>80MHz</td>
</tr>
<tr>
<td>ST DISCO-L072CZ-LRWAN1</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L072CZ</td>
<td>32MHz</td>
</tr>
<tr>
<td>ST DISCO-L475VG-IOT01A</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L475VGIT6</td>
<td>80MHz</td>
</tr>
<tr>
<td>Discovery F072RB</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F072RBT6</td>
<td>48MHz</td>
</tr>
<tr>
<td>ST Nucleo F030R8</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F030R8T6</td>
<td>48MHz</td>
</tr>
<tr>
<td>ST Nucleo F031K6</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F031K6T6</td>
<td>48MHz</td>
</tr>
<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 F103RB</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F103RBT6</td>
<td>72MHz</td>
</tr>
<tr>
<td>ST DISCO-L072CZ-LRWAN1</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L072CZ</td>
<td>32MHz</td>
</tr>
<tr>
<td>ST DISCO-L475VG-IOT01A</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L475VGIT6</td>
<td>80MHz</td>
</tr>
<tr>
<td>Discovery F072RB</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F072RBT6</td>
<td>48MHz</td>
</tr>
<tr>
<td>ST Nucleo F030R8</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F030R8T6</td>
<td>48MHz</td>
</tr>
<tr>
<td>ST Nucleo F031K6</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F031K6T6</td>
<td>48MHz</td>
</tr>
<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 F103RB</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F103RBT6</td>
<td>72MHz</td>
</tr>
<tr>
<td>ST DISCO-L072CZ-LRWAN1</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L072CZ</td>
<td>32MHz</td>
</tr>
<tr>
<td>ST DISCO-L475VG-IOT01A</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L475VGIT6</td>
<td>80MHz</td>
</tr>
<tr>
<td>Discovery F072RB</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F072RBT6</td>
<td>48MHz</td>
</tr>
<tr>
<td>ST Nucleo F030R8</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F030R8T6</td>
<td>48MHz</td>
</tr>
<tr>
<td>ST Nucleo F031K6</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F031K6T6</td>
<td>48MHz</td>
</tr>
<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 F103RB</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F103RBT6</td>
<td>72MHz</td>
</tr>
<tr>
<td>ST DISCO-L072CZ-LRWAN1</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L072CZ</td>
<td>32MHz</td>
</tr>
<tr>
<td>ST DISCO-L475VG-IOT01A</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L475VGIT6</td>
<td>80MHz</td>
</tr>
<tr>
<td>Discovery F072RB</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F072RBT6</td>
<td>48MHz</td>
</tr>
<tr>
<td>ST Nucleo F030R8</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F030R8T6</td>
<td>48MHz</td>
</tr>
<tr>
<td>ST Nucleo F031K6</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F031K6T6</td>
<td>48MHz</td>
</tr>
<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 F103RB</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F103RBT6</td>
<td>72MHz</td>
</tr>
<tr>
<td>ST DISCO-L072CZ-LRWAN1</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L072CZ</td>
<td>32MHz</td>
</tr>
<tr>
<td>ST DISCO-L475VG-IOT01A</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L475VGIT6</td>
<td>80MHz</td>
</tr>
<tr>
<td>Discovery F072RB</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F072RBT6</td>
<td>48MHz</td>
</tr>
<tr>
<td>ST Nucleo F030R8</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F030R8T6</td>
<td>48MHz</td>
</tr>
<tr>
<td>ST Nucleo F031K6</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F031K6T6</td>
<td>48MHz</td>
</tr>
<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 F103RB</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F103RBT6</td>
<td>72MHz</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>ST STM32F4DISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>ST STM32F407VG T6</td>
<td>16MHz</td>
</tr>
<tr>
<td>ST STM32L073Z-EVAL</td>
<td>ST STM32</td>
<td>On-board</td>
<td>ST STM32L073VZ T6</td>
<td>32MHz</td>
</tr>
<tr>
<td>ST STM32LDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>ST STM32L152RT T6</td>
<td>32MHz</td>
</tr>
<tr>
<td>ST STM32VLDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>ST STM32F100RT T6</td>
<td>24MHz</td>
</tr>
<tr>
<td>ST Sensor Node</td>
<td>ST STM32</td>
<td>On-board</td>
<td>ST STM32L476JG</td>
<td>80MHz</td>
</tr>
<tr>
<td>STM32F103C8 (20k RAM, 64k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>ST STM32F103C8T6</td>
<td>72MHz</td>
</tr>
<tr>
<td>STM32F103CB (20k RAM, 128k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>ST STM32F103CBT6</td>
<td>72MHz</td>
</tr>
<tr>
<td>STM32F103R8 (20k RAM, 64 Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>ST STM32F103R8T6</td>
<td>72MHz</td>
</tr>
<tr>
<td>STM32F103RRB (20k RAM, 128k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>ST STM32F103RBT6</td>
<td>72MHz</td>
</tr>
<tr>
<td>STM32F103RC (48k RAM, 256k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>ST STM32F103RCT6</td>
<td>72MHz</td>
</tr>
<tr>
<td>STM32F103RE (64k RAM, 512k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>ST STM32F103RE T6</td>
<td>72MHz</td>
</tr>
<tr>
<td>STM32F103RT8 (20k RAM, 64 Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>ST STM32F103T8T6</td>
<td>72MHz</td>
</tr>
<tr>
<td>STM32F103TB (20k RAM, 128k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>ST STM32F103TBT6</td>
<td>72MHz</td>
</tr>
<tr>
<td>STM32F103VB (20k RAM, 128k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>ST STM32F103VBT6</td>
<td>72MHz</td>
</tr>
<tr>
<td>STM32F103VC (48k RAM, 256k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>ST STM32F103VC T6</td>
<td>72MHz</td>
</tr>
<tr>
<td>STM32F103VD (64k RAM, 384k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>ST STM32F103VDT6</td>
<td>72MHz</td>
</tr>
<tr>
<td>STM32F103VE (64k RAM, 512k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>ST STM32F103VET6</td>
<td>72MHz</td>
</tr>
<tr>
<td>STM32F103ZC (48k RAM, 256k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>ST STM32F103ZCT6</td>
<td>72MHz</td>
</tr>
<tr>
<td>STM32F105ZD (64k RAM, 384k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>ST STM32F103ZDT6</td>
<td>72MHz</td>
</tr>
<tr>
<td>STM32F103ZEE (64k RAM, 512k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>ST STM32F103ZET6</td>
<td>72MHz</td>
</tr>
<tr>
<td>STM32F303CB (32k RAM, 128k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>ST STM32F303CBT6</td>
<td>72MHz</td>
</tr>
<tr>
<td>STM32F407VE (192k RAM, 512k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>ST STM32F407VE T6</td>
<td>168MHz</td>
</tr>
<tr>
<td>STM32F407VG (192k RAM, 1024k Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>ST STM32F407VGT6</td>
<td>168MHz</td>
</tr>
<tr>
<td>STM32F4Stamp F405</td>
<td>ST STM32</td>
<td>External</td>
<td>ST STM32F405GRT6</td>
<td>168MHz</td>
</tr>
<tr>
<td>STM32F730R-DK</td>
<td>ST STM32</td>
<td>On-board</td>
<td>ST STM32F750N8H6</td>
<td>216MHz</td>
</tr>
<tr>
<td>SainSmart Due (Programming Port)</td>
<td>Atmel SAM</td>
<td>External</td>
<td>AT91SAM3X8E</td>
<td>84MHz</td>
</tr>
<tr>
<td>SainSmart Due (USB Native Port)</td>
<td>Atmel SAM</td>
<td>External</td>
<td>AT91SAM3X8E</td>
<td>84MHz</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>ST STM32F407VET6</td>
<td>168MHz</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>Seeed Wio 3G</td>
<td>ST STM32</td>
<td>On-board</td>
<td>ST STM32F439V1</td>
<td>180MHz</td>
</tr>
<tr>
<td>Seeeduino LoRaWAN</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD2TG18A</td>
<td>48MHz</td>
</tr>
<tr>
<td>Silicognition wESP32</td>
<td>Espressif32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
</tr>
<tr>
<td>Sino:Bit</td>
<td>Nordic nRF51</td>
<td>External</td>
<td>NRF51822</td>
<td>32MHz</td>
</tr>
<tr>
<td>Speed MAIX BiT</td>
<td>Kendryte K210</td>
<td>External</td>
<td>K210</td>
<td>400MHz</td>
</tr>
<tr>
<td>Speed MAIX BiT with Mic</td>
<td>Kendryte K210</td>
<td>External</td>
<td>K210</td>
<td>400MHz</td>
</tr>
<tr>
<td>Speed MAIX GO</td>
<td>Kendryte K210</td>
<td>External</td>
<td>K210</td>
<td>400MHz</td>
</tr>
<tr>
<td>Speed MAIX ONE DOCK</td>
<td>Kendryte K210</td>
<td>External</td>
<td>K210</td>
<td>400MHz</td>
</tr>
<tr>
<td>Speed MAIXDUINO</td>
<td>Kendryte K210</td>
<td>External</td>
<td>K210</td>
<td>400MHz</td>
</tr>
<tr>
<td>Solder Splash Labs DipCortex M0</td>
<td>NXP LPC</td>
<td>External</td>
<td>LPC11U24</td>
<td>50MHz</td>
</tr>
<tr>
<td>SparkFun Esp32 Thing</td>
<td>Espressif32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
</tr>
<tr>
<td>SparkFun LoRa Gateway 1-Channel</td>
<td>Espressif32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
</tr>
<tr>
<td>SparkFun SAMD21 Dev Breakout</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
</tr>
<tr>
<td>SparkFun SAMD21 Mini Breakout</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48MHz</td>
</tr>
<tr>
<td>Sparky V1 F303</td>
<td>ST STM32</td>
<td>External</td>
<td>ST STM32F303CCT6</td>
<td>72MHz</td>
</tr>
<tr>
<td>Switch Science mbad LPC1114FN28</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC1114FN28</td>
<td>48MHz</td>
</tr>
<tr>
<td>Switch Science mbad LPC824</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC824</td>
<td>30MHz</td>
</tr>
<tr>
<td>TTGO LoRa32-OLED V1</td>
<td>Espressif32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</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>TTGO T-Beam</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240M</td>
</tr>
<tr>
<td>Taida Century nRF52 mini board</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52832</td>
<td>64M</td>
</tr>
<tr>
<td>Teensy 3.1 / 3.2</td>
<td>Teensy</td>
<td>External</td>
<td>MK20DX256</td>
<td>72M</td>
</tr>
<tr>
<td>Teensy 3.5</td>
<td>Teensy</td>
<td>External</td>
<td>MK64FX512</td>
<td>120M</td>
</tr>
<tr>
<td>Teensy 3.6</td>
<td>Teensy</td>
<td>External</td>
<td>MK66FX1M0</td>
<td>180M</td>
</tr>
<tr>
<td>Teensy LC</td>
<td>Teensy</td>
<td>External</td>
<td>MKL26Z64</td>
<td>48M</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>40M</td>
</tr>
<tr>
<td>Tiny STM103T</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103TB6</td>
<td>72M</td>
</tr>
<tr>
<td>Tuino 096</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</td>
<td>48M</td>
</tr>
<tr>
<td>VAK E v1.0</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F446RT6</td>
<td>180M</td>
</tr>
<tr>
<td>WEMOS LOLIN D32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240M</td>
</tr>
<tr>
<td>WEMOS LOLIN D32 PRO</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240M</td>
</tr>
<tr>
<td>WEMOS LOLIN32</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240M</td>
</tr>
<tr>
<td>WIZwiki-W7500</td>
<td>WIZNet W7500</td>
<td>On-board</td>
<td>WIZNET7500</td>
<td>48M</td>
</tr>
<tr>
<td>WIZwiki-W7500ECO</td>
<td>WIZNet W7500</td>
<td>On-board</td>
<td>WIZNET7500ECO</td>
<td>48M</td>
</tr>
<tr>
<td>WIZwiki-W7500P</td>
<td>WIZNet W7500</td>
<td>On-board</td>
<td>WIZNET7500P</td>
<td>48M</td>
</tr>
<tr>
<td>Waveshare BLE400</td>
<td>Nordic nRF51</td>
<td>External</td>
<td>NRF51822</td>
<td>32M</td>
</tr>
<tr>
<td>WeMos WiFi and Bluetooth Battery</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240M</td>
</tr>
<tr>
<td>XMC1100 Boot Kit</td>
<td>Infineon XMC</td>
<td>On-board</td>
<td>XMC1100</td>
<td>32M</td>
</tr>
<tr>
<td>XMC1100 H-Bridge 2Go</td>
<td>Infineon XMC</td>
<td>On-board</td>
<td>XMC1100</td>
<td>32M</td>
</tr>
<tr>
<td>XMC1100 XMC2Go</td>
<td>Infineon XMC</td>
<td>On-board</td>
<td>XMC1100</td>
<td>32M</td>
</tr>
<tr>
<td>XMC1300 Boot Kit</td>
<td>Infineon XMC</td>
<td>On-board</td>
<td>XMC1300</td>
<td>32M</td>
</tr>
<tr>
<td>XMC1300 Sense2GoL</td>
<td>Infineon XMC</td>
<td>On-board</td>
<td>XMC1300</td>
<td>32M</td>
</tr>
<tr>
<td>XMC1400 Boot Kit</td>
<td>Infineon XMC</td>
<td>On-board</td>
<td>XMC1400</td>
<td>48M</td>
</tr>
<tr>
<td>XMC4200 Distance2Go</td>
<td>Infineon XMC</td>
<td>On-board</td>
<td>XMC4200</td>
<td>80M</td>
</tr>
<tr>
<td>XMC4700 Relax Kit</td>
<td>Infineon XMC</td>
<td>On-board</td>
<td>XMC4700</td>
<td>144M</td>
</tr>
<tr>
<td>XinaBox CW02</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240M</td>
</tr>
<tr>
<td>hackaBLE</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52832</td>
<td>64M</td>
</tr>
<tr>
<td>ng-beacon</td>
<td>Nordic nRF51</td>
<td>External</td>
<td>NRF51822</td>
<td>16M</td>
</tr>
<tr>
<td>oddWires IoT-Bus lo</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240M</td>
</tr>
<tr>
<td>oddWires IoT-Bus Proteus</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240M</td>
</tr>
<tr>
<td>sakura.io Evaluation Board</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F411RT6</td>
<td>100M</td>
</tr>
<tr>
<td>u-blox C027</td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC1768</td>
<td>96M</td>
</tr>
<tr>
<td>u-blox C030-N211 IoT Starter Kit</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F437VG</td>
<td>180M</td>
</tr>
<tr>
<td>u-blox C030-R410M IoT</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F437VG</td>
<td>180M</td>
</tr>
<tr>
<td>u-blox C030-U201 IoT Starter Kit</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F437VG</td>
<td>180M</td>
</tr>
<tr>
<td>u-blox EVK-NINA-B1</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52832</td>
<td>64M</td>
</tr>
<tr>
<td>u-blox EVK-ODIN-W2</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F439ZIY6</td>
<td>168M</td>
</tr>
<tr>
<td>u-blox ODIN-W2</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F439ZIY6</td>
<td>168M</td>
</tr>
<tr>
<td>y5 LPC11U35 mbug</td>
<td>NXP LPC</td>
<td>External</td>
<td>LPC11U35</td>
<td>48M</td>
</tr>
<tr>
<td>y5 nRF51822 mbug</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>16M</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):

```ini
[env:myenv]
platform = ...
board = ...
debug_tool = minimodule
```

If you would like to use this tool for firmware uploading, please change upload protocol:

```ini
[env:myenv]
platform = ...
board = ...
debug_tool = minimodule
upload_protocol = minimodule
```

More options:
• Debugging options
• Upload options

Drivers

Windows  See [https://community.platformio.org/t/esp32-pio-unified-debugger/4541/20](https://community.platformio.org/t/esp32-pio-unified-debugger/4541/20)

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](file). 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 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>
<tr>
<td>RESET#</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>

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](file)

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, this platform is ported by Sipeed.</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>Freedom E SDK</td>
<td>Open Source Software for Developing on the SiFive Freedom E Platform</td>
</tr>
<tr>
<td>Kendryte Standalone SDK</td>
<td>Kendryte Standalone SDK without OS support</td>
</tr>
<tr>
<td>Kendryte FreeRTOS SDK</td>
<td>Kendryte SDK with FreeRTOS support</td>
</tr>
<tr>
<td>Simba</td>
<td>Simba is an RTOS and build framework. It 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>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>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>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>Duit 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>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>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>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-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>

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>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>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>Siped 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>Siped 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>Siped 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>Siped MAIXDUINO</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 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>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 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.

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 = ...
download_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>TI</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.
### Platform Selection

<table>
<thead>
<tr>
<th>Name</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 EXP430FR5739LP</td>
<td>MSP-TI MSP430</td>
<td>On-board</td>
<td>MSP430FR5739</td>
<td>16MHz</td>
<td>16KB</td>
<td>512B</td>
</tr>
<tr>
<td>TI LaunchPad EXP430F5529LP</td>
<td>MSP-TI MSP430</td>
<td>On-board</td>
<td>MSP430F5529</td>
<td>16MHz</td>
<td>128KB</td>
<td>8KB</td>
</tr>
<tr>
<td>TI LaunchPad EXP430FR4133LP</td>
<td>MSP-TI MSP430</td>
<td>On-board</td>
<td>MSP430FR4133</td>
<td>8MHz</td>
<td>15KB</td>
<td>2KB</td>
</tr>
<tr>
<td>TI LaunchPad EXP430FR5969LP</td>
<td>MSP-TI MSP430</td>
<td>On-board</td>
<td>MSP430FR5969</td>
<td>8MHz</td>
<td>64KB</td>
<td>2KB</td>
</tr>
<tr>
<td>TI LaunchPad EXP430FR6989LP</td>
<td>MSP-TI MSP430</td>
<td>On-board</td>
<td>MSP430FR6989</td>
<td>8MHz</td>
<td>127KB</td>
<td>2KB</td>
</tr>
<tr>
<td>TI LaunchPad EXP430G2553LP</td>
<td>MSP-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):
If you would like to use this tool for firmware uploading, please change upload protocol:

```
[env:myenv]
platform = ...
board = ...
debug_tool = olimex-arm-usb-ocd-h
upload_protocol = olimex-arm-usb-ocd-h
```

More options:

- Debugging options
- Upload options

**Drivers**

**Windows** See [https://community.platformio.org/t/esp32-pio-unified-debugger/4541/20](https://community.platformio.org/t/esp32-pio-unified-debugger/4541/20)

**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](https://community.platformio.org/t/esp32-pio-unified-debugger/4541/20). 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-OCD-H JTAG 20-Pin Connector

<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>3 RESET</td>
<td>Connect this pin to the (active low) reset input of the target CPU (EN for ESP32)</td>
</tr>
</tbody>
</table>

### Platforms

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Espresif</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, this platform is ported by Sipeed.</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>Freedom E SDK</td>
<td>Open Source Software for Developing on the SiFive Freedom E Platform</td>
</tr>
<tr>
<td>Kendryte Standalone SDK</td>
<td>Kendryte Standalone SDK without OS support</td>
</tr>
<tr>
<td>Kendryte FreeRTOS SDK</td>
<td>Kendryte SDK with FreeRTOS support</td>
</tr>
<tr>
<td>Simba</td>
<td>Simba is an RTOS and build framework. It 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>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>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>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>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>ESP M32</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>
<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>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-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></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>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 T-Beam</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>1.25MB</td>
<td></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 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>
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:

```ini
[env:myenv]
platform = ...
board = ...
debug_tool = olimex-arm-usb-ocd
upload_protocol = olimex-arm-usb-ocd
```

More options:
- Debugging options
- Upload options
Drivers

**Windows** See [https://community.platformio.org/t/esp32-pio-unified-debugger/4541/20](https://community.platformio.org/t/esp32-pio-unified-debugger/4541/20)

**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 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>3</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>
## Platforms

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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>Kendryte K210</strong></td>
<td>Kendryte K210 is an AI capable RISC-V64 dual core SoC, this platform is ported by Sipeed.</td>
</tr>
<tr>
<td><strong>SiFive</strong></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><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>Freedom E SDK</strong></td>
<td>Open Source Software for Developing on the SiFive Freedom E Platform</td>
</tr>
<tr>
<td><strong>Kendryte Standalone SDK</strong></td>
<td>Kendryte Standalone SDK without OS support</td>
</tr>
<tr>
<td><strong>Kendryte FreeRTOS SDK</strong></td>
<td>Kendryte SDK with FreeRTOS support</td>
</tr>
<tr>
<td><strong>Simba</strong></td>
<td>Simba is an RTOS and build framework. It 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><strong>ALKS ESP32</strong></td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td><strong>Adafruit ESP32 Feather</strong></td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td><strong>Arty FPGA Dev Kit</strong></td>
<td>SiFive</td>
<td>On-board</td>
<td>FE310</td>
<td>450MHz</td>
<td>16MB</td>
<td>256MB</td>
</tr>
<tr>
<td><strong>D-duino-32</strong></td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td><strong>DOIT ESP32 DEVKIT V1</strong></td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td><strong>Dongsen Tech Pocket 32</strong></td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td><strong>ESP32 FM DevKit</strong></td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td><strong>ESP32vn IoT Uno</strong></td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td><strong>ESPectro32</strong></td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td><strong>ESPino32</strong></td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td><strong>Espressif ESP-WROVER-KIT</strong></td>
<td>Espressif 32</td>
<td>On-board</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td><strong>Espressif ESP32 Dev Module</strong></td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td><strong>FireBeetle-ESP32</strong></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 35 – 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>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>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-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>340MHz</td>
<td>4MB</td>
<td>1.25MB</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>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 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>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 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 1o</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](#).
## Configuration

You can configure debugging tool using `debug_tool` option in "platformio.ini" (Project Configuration File):

```
[env:myenv]
platform = ...
board = ...
debg_tool = olimex-arm-usb-tiny-h
```

If you would like to use this tool for firmware uploading, please change upload protocol:

```
[env:myenv]
platform = ...
board = ...
debg_tool = olimex-arm-usb-tiny-h
upload_protocol = olimex-arm-usb-tiny-h
```

More options:

- Debugging options
- Upload options

## Drivers

- **Windows** See https://community.platformio.org/t/esp32-pio-unified-debugger/4541/20
- **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

Olimex ARM-USB-TINY-H JTAG 20-Pin Connector

<table>
<thead>
<tr>
<th>Board</th>
<th>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>3</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>

### 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, this platform is ported by Sipeed.</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>Freedom E SDK</td>
<td>Open Source Software for Developing on the SiFive Freedom E Platform</td>
</tr>
<tr>
<td>Kendryte Standalone SDK</td>
<td>Kendryte Standalone SDK without OS support</td>
</tr>
<tr>
<td>Kendryte FreeRTOS SDK</td>
<td>Kendryte SDK with FreeRTOS support</td>
</tr>
<tr>
<td>Simba</td>
<td>Simba is an RTOS and build framework. It 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>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>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>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>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>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>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-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>

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>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>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>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 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>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 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

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):

```
[env:myenv]
platform = ...
board = ...
debug_tool = olimex-jtag-tiny
```

If you would like to use this tool for firmware uploading, please change upload protocol:

```
[env:myenv]
platform = ...
board = ...
debug_tool = olimex-jtag-tiny
upload_protocol = olimex-jtag-tiny
```

More options:

- Debugging options
- Upload options

Drivers

**Windows**  See [https://community.platformio.org/t/esp32-pio-unified-debugger/4541/20](https://community.platformio.org/t/esp32-pio-unified-debugger/4541/20)

**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-TINY 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>3</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>

## 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, this platform is ported by Sipeed.</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>Freedom E SDK</td>
<td>Open Source Software for Developing on the SiFive Freedom E Platform</td>
</tr>
<tr>
<td>Kendryte Standalone SDK</td>
<td>Kendryte Standalone SDK without OS support</td>
</tr>
<tr>
<td>Kendryte FreeRTOS SDK</td>
<td>Kendryte SDK with FreeRTOS support</td>
</tr>
<tr>
<td>Simba</td>
<td>Simba is an RTOS and build framework. It 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>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>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>Internal</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</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>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>Internal</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 LoRa32 (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>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-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>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>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 T-Beam</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>1.25MB</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 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>
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 = ...
download_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><strong>Freedom E SDK</strong></td>
<td>Open Source Software for Developing on the SiFive Freedom E Platform</td>
</tr>
</tbody>
</table>

Boards

**Note**: For more detailed board information please scroll tables below by horizontal.
**PlatformIO Documentation, Release 4.0.1b3**

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

**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):

```ini
[env:myenv]
platform = ...
board = ...
download_tool = sipeed-rv-debugger
```

If you would like to use this tool for firmware uploading, please change upload protocol:

```ini
[env:myenv]
platform = ...
board = ...
download_tool = sipeed-rv-debugger
upload_protocol = sipeed-rv-debugger
```
More options:

- *Debugging options*
- *Upload options*

**Drivers**

*Windows*  See https://community.platformio.org/t/esp32-pio-unified-debugger/4541/20

*Mac*  macOS contains default FTDLUSBSerialDriver 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 Connector</th>
<th>RV Debugger Connector</th>
<th>Board Pin</th>
<th>JTAG</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>GND</td>
<td>Digital ground</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>TDI</td>
<td>Test Data In pin</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>TMS</td>
<td>Test Mode State pin</td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>TCK</td>
<td>JTAG Return Test Clock</td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>TDO</td>
<td>Test Data Out pin</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>RST</td>
<td>Connect this pin to the (active low) reset input of the target CPU</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**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, this platform is ported by Sipeed.</td>
</tr>
</tbody>
</table>

**Frameworks**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached 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 Standalone SDK</td>
<td>Kendryte Standalone SDK without OS support</td>
</tr>
<tr>
<td>Kendryte FreeRTOS SDK</td>
<td>Kendryte SDK with FreeRTOS 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>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>
</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):
[env:myenv]
platform = ...
board = ...
debug_tool = stlink

If you would like to use this tool for firmware uploading, please change upload protocol:

[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>SWCLK</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</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>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>libOpenCM3</td>
<td>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.</td>
</tr>
<tr>
<td>mbed</td>
<td>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.</td>
</tr>
<tr>
<td>Simba</td>
<td>Simba is an RTOS and build framework. It aims to make embedded programming easy and portable.</td>
</tr>
<tr>
<td>SPL</td>
<td>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.</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>

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>1MB</td>
</tr>
<tr>
<td>32F723EDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F723VEK6</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>2MB</td>
</tr>
<tr>
<td>3DP001V1 Evaluation board for 3D printer</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F401VG16</td>
<td>84MHz</td>
<td>512K</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>Aceinna OpenIMU 300ZA</td>
<td>Aceinna IMU</td>
<td>External</td>
<td>STM32F405RG</td>
<td>120MHz</td>
<td>1MB</td>
</tr>
<tr>
<td>Arduino Due (Programming Port)</td>
<td>Atmel SAM</td>
<td>External</td>
<td>AT91SAM3X8E</td>
<td>84MHz</td>
<td>512K</td>
</tr>
<tr>
<td>Arduino Due (USB Native Port)</td>
<td>Atmel SAM</td>
<td>External</td>
<td>AT91SAM3X8E</td>
<td>84MHz</td>
<td>512K</td>
</tr>
<tr>
<td>Armstrap Eagle 1024</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F417VG16</td>
<td>168MHz</td>
<td>1MB</td>
</tr>
<tr>
<td>Armstrap Eagle 2048</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F427VIT6</td>
<td>168MHz</td>
<td>1.99M</td>
</tr>
<tr>
<td>Armstrap Eagle 512</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>512K</td>
</tr>
<tr>
<td>Black STM32F407VE</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407VET6</td>
<td>168MHz</td>
<td>512K</td>
</tr>
<tr>
<td>Black STM32F407VG</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407VGT6</td>
<td>168MHz</td>
<td>512K</td>
</tr>
<tr>
<td>Black STM32F407ZE</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407ZGT6</td>
<td>168MHz</td>
<td>1MB</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>Black STM32F407ZE</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407ZE6</td>
<td>168MHz</td>
<td>512K</td>
</tr>
<tr>
<td>BlackPill F103C8</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>64KB</td>
</tr>
<tr>
<td>BlackPill F103C8 (128k)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103C8T6</td>
<td>128K</td>
<td></td>
</tr>
<tr>
<td>BlackPill F303CC</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F303CCT6</td>
<td>72MHz</td>
<td>256K</td>
</tr>
<tr>
<td>Blue STM32F407VE Mini</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407VE6</td>
<td>168MHz</td>
<td>512K</td>
</tr>
<tr>
<td>BluePill F103C6</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103C6T6</td>
<td>72MHz</td>
<td>32KB</td>
</tr>
<tr>
<td>BluePill F103C8</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>64KB</td>
</tr>
<tr>
<td>BluePill F103C8 (128k)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103C8T6</td>
<td>128K</td>
<td></td>
</tr>
<tr>
<td>Bluey nRF52832 IoT</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512K</td>
</tr>
<tr>
<td>BluC3D</td>
<td>Nordic nRF51</td>
<td>External</td>
<td>NRF51822</td>
<td>32MHz</td>
<td>256K</td>
</tr>
<tr>
<td>Delta DFBM-NQ620</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512K</td>
</tr>
<tr>
<td>Demo F030F4</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F030F4P6</td>
<td>48MHz</td>
<td>16KB</td>
</tr>
<tr>
<td>Digistump DigiX</td>
<td>Atmel SAM</td>
<td>External</td>
<td>AT91SAM3X8E</td>
<td>84MHz</td>
<td>512K</td>
</tr>
<tr>
<td>Espotel LoRa Module</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411RET6</td>
<td>100MHz</td>
<td>512K</td>
</tr>
<tr>
<td>F-407VG</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407VT6</td>
<td>168MHz</td>
<td>512K</td>
</tr>
<tr>
<td>FK407M1</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F407VT6</td>
<td>168MHz</td>
<td>512K</td>
</tr>
<tr>
<td>L476DMW1K</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L476VT6</td>
<td>80MHz</td>
<td>1MB</td>
</tr>
<tr>
<td>M200 V2</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F070CBT6</td>
<td>48MHz</td>
<td>120K</td>
</tr>
<tr>
<td>MTS Dragonfly</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411RET6</td>
<td>100MHz</td>
<td>512K</td>
</tr>
<tr>
<td>Macchina M2</td>
<td>Atmel SAM</td>
<td>External</td>
<td>AT91SAM3X8E</td>
<td>84MHz</td>
<td>512K</td>
</tr>
<tr>
<td>Malyan M200 V1</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103CBT6</td>
<td>72MHz</td>
<td>120K</td>
</tr>
<tr>
<td>Maple</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103RBT6</td>
<td>72MHz</td>
<td>108K</td>
</tr>
<tr>
<td>Maple (RET6)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103RET6</td>
<td>72MHz</td>
<td>256K</td>
</tr>
<tr>
<td>Maple Mini Bootloader 2.0</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103CBT6</td>
<td>72MHz</td>
<td>120K</td>
</tr>
<tr>
<td>Maple Mini Original</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103CBT6</td>
<td>72MHz</td>
<td>108K</td>
</tr>
<tr>
<td>Mbed Connect Cloud</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F439ZY6</td>
<td>168MHz</td>
<td>2MB</td>
</tr>
<tr>
<td>Microduino Core STM32 to Flash</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103CBT6</td>
<td>72MHz</td>
<td>105.4K</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>
<td>1MB</td>
</tr>
<tr>
<td>MultiTech mDot</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411RET6</td>
<td>100MHz</td>
<td>512K</td>
</tr>
<tr>
<td>MultiTech mDot F411</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F411RET6</td>
<td>100MHz</td>
<td>512K</td>
</tr>
<tr>
<td>MultiTech xDot</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L151CCU6</td>
<td>32MHz</td>
<td>256K</td>
</tr>
<tr>
<td>N2+</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F405RG6</td>
<td>168MHz</td>
<td>1MB</td>
</tr>
<tr>
<td>NAMote72</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L152RC</td>
<td>32MHz</td>
<td>256K</td>
</tr>
<tr>
<td>Nordic Beacon Kit (PCA20006)</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>32MHz</td>
<td>256K</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>256K</td>
</tr>
<tr>
<td>Nordic nRF52- DK</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512K</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>
</tr>
<tr>
<td>Nucleo G071RB</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32G071RBT6</td>
<td>24MHz</td>
<td>2MB</td>
</tr>
<tr>
<td>OSHChip</td>
<td>Nordic nRF51</td>
<td>External</td>
<td>NRF51822</td>
<td>32MHz</td>
<td>256K</td>
</tr>
<tr>
<td>P-Nucleo WB55RG</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32WB55RG</td>
<td>64MHz</td>
<td>512K</td>
</tr>
<tr>
<td>RAK811 LoRa Tracker</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L151RBT6</td>
<td>32MHz</td>
<td>128K</td>
</tr>
<tr>
<td>RAK811 LoRa Tracker</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32L151RBT6</td>
<td>32MHz</td>
<td>128K</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>256K</td>
</tr>
<tr>
<td>RedBearLab BLE Nano 2</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512K</td>
</tr>
<tr>
<td>RedBearLab BLE Blend 2</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512K</td>
</tr>
<tr>
<td>RedBearLab nRF51822</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>16MHz</td>
<td>256K</td>
</tr>
<tr>
<td>RushUp Cloud-JAM</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F401RET6</td>
<td>84MHz</td>
<td>512K</td>
</tr>
<tr>
<td>RushUp Cloud-JAM L4</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L476RG6</td>
<td>80MHz</td>
<td>1MB</td>
</tr>
<tr>
<td>SDT52832B</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512K</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>
</tr>
</thead>
<tbody>
<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>256K</td>
</tr>
<tr>
<td>ST 32F411EDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F411VET6</td>
<td>100MHz</td>
<td>512K</td>
</tr>
<tr>
<td>ST 32F413HDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F413ZHT6</td>
<td>100MHz</td>
<td>512K</td>
</tr>
<tr>
<td>ST 32F4291DISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F429ZIT6</td>
<td>180MHz</td>
<td>2MB</td>
</tr>
<tr>
<td>ST 32F4691DISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F469NIH6</td>
<td>180MHz</td>
<td>1MB</td>
</tr>
<tr>
<td>ST 32F746GDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F746NGH6</td>
<td>216MHz</td>
<td>1MB</td>
</tr>
<tr>
<td>ST 32F7691DISCOVERY</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>256K</td>
</tr>
<tr>
<td>ST DISCO-L072CZ-LRWAN1</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L072CZ</td>
<td>32MHz</td>
<td>192K</td>
</tr>
<tr>
<td>ST DISCO-L475VG-IOT01A</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L475VGT6</td>
<td>80MHz</td>
<td>1MB</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>
</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>STM32F070RBT6</td>
<td>48MHz</td>
<td>128K</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>
</tr>
<tr>
<td>ST Nucleo F091RC</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F091RCT6</td>
<td>48MHz</td>
<td>256K</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>
</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>512K</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>
</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>STM32F401RE</td>
<td>84MHz</td>
<td>512K</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>
</tr>
<tr>
<td>ST Nucleo F411RE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F411RBT6</td>
<td>100MHz</td>
<td>512K</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>512K</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>STM32F446RE</td>
<td>180MHz</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>
</tr>
<tr>
<td>ST Nucleo F722ZE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F722ZET6</td>
<td>216MHz</td>
<td>512K</td>
</tr>
<tr>
<td>ST Nucleo F746ZG</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F746ZET6</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</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32H743ZIT6</td>
<td>400MHz</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>
</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>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>
</tr>
<tr>
<td>ST Nucleo L412KB</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L412KBU6</td>
<td>80MHz</td>
<td>128K</td>
</tr>
<tr>
<td>ST Nucleo L432KC</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L432KCUC6</td>
<td>80MHz</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>ST Nucleo L433RC-P</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L433RC</td>
<td>80MHz</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>
</tr>
<tr>
<td>ST Nucleo L476RG</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L476RGT6</td>
<td>80MHz</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>
</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 L452RE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L452RET6</td>
<td>80MHz</td>
<td>1MB</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>
</tr>
<tr>
<td>ST Nucleo L486RG</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L486RGT6</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 L452RE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L452RET6</td>
<td>80MHz</td>
<td>1MB</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>
</tr>
<tr>
<td>ST Nucleo L486RG</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L486RGT6</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 L4R5ZI</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L4R5ZIT6</td>
<td>120MHz</td>
<td>2MB</td>
</tr>
<tr>
<td>ST STM32F0308DISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F030R8T6</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 STM32F103C8</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F103C8T6</td>
<td>72MHz</td>
<td>64KB</td>
</tr>
<tr>
<td>ST STM32F103CB</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F103CBT6</td>
<td>72MHz</td>
<td>128K</td>
</tr>
<tr>
<td>ST STM32F103R8</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F103R8T6</td>
<td>72MHz</td>
<td>64KB</td>
</tr>
<tr>
<td>ST STM32F103R8</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F103R8T6</td>
<td>72MHz</td>
<td>64KB</td>
</tr>
<tr>
<td>ST STM32F103RC</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F103RCT6</td>
<td>72MHz</td>
<td>256K</td>
</tr>
<tr>
<td>ST STM32F103RE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F103RET6</td>
<td>72MHz</td>
<td>512K</td>
</tr>
<tr>
<td>ST STM32F103T8</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F103T8T6</td>
<td>72MHz</td>
<td>20KB</td>
</tr>
<tr>
<td>ST STM32F103TB</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F103BTC6</td>
<td>72MHz</td>
<td>128K</td>
</tr>
<tr>
<td>ST STM32F103VB</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F103VBT6</td>
<td>72MHz</td>
<td>128K</td>
</tr>
<tr>
<td>ST STM32F103VC</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F103VCT6</td>
<td>72MHz</td>
<td>256K</td>
</tr>
<tr>
<td>ST STM32F103VD</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F103VDT6</td>
<td>72MHz</td>
<td>384K</td>
</tr>
<tr>
<td>ST STM32F103VE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F103VET6</td>
<td>72MHz</td>
<td>512K</td>
</tr>
<tr>
<td>ST STM32F103ZC</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F103ZCT6</td>
<td>72MHz</td>
<td>256K</td>
</tr>
<tr>
<td>ST STM32F103ZD</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F103ZDT6</td>
<td>72MHz</td>
<td>384K</td>
</tr>
<tr>
<td>ST STM32F103ZE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F103ZET6</td>
<td>72MHz</td>
<td>512K</td>
</tr>
<tr>
<td>ST STM32F303CB</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F303CBT6</td>
<td>72MHz</td>
<td>128K</td>
</tr>
<tr>
<td>ST STM32F307VE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F307VET6</td>
<td>168MHz</td>
<td>502.2K</td>
</tr>
<tr>
<td>ST STM32F307VG</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F307VGT6</td>
<td>168MHz</td>
<td>1MB</td>
</tr>
<tr>
<td>ST STM32F4Stamp F405</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F405RGT6</td>
<td>168MHz</td>
<td>1MB</td>
</tr>
<tr>
<td>ST STM32F7508-DK</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F7508N8H6</td>
<td>216MHz</td>
<td>64KB</td>
</tr>
<tr>
<td>SainSmart Due (Programming Port)</td>
<td>Atmel SAM</td>
<td>External</td>
<td>AT91SAM3X8E</td>
<td>84MHz</td>
<td>512K</td>
</tr>
<tr>
<td>SainSmart Due (USB Native Port)</td>
<td>Atmel SAM</td>
<td>External</td>
<td>AT91SAM3X8E</td>
<td>84MHz</td>
<td>512K</td>
</tr>
<tr>
<td>Seeed Arch BLE</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>16MHz</td>
<td>128K</td>
</tr>
<tr>
<td>Seeed Arch Link</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>16MHz</td>
<td>256K</td>
</tr>
<tr>
<td>Seeed Arch Max</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>16MHz</td>
<td>512K</td>
</tr>
<tr>
<td>Seeed Tiny BLE</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>16MHz</td>
<td>256K</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>
</tr>
<tr>
<td>Sino:Bit</td>
<td>Nordic nRF51</td>
<td>External</td>
<td>NRF51822</td>
<td>32MHz</td>
<td>256K</td>
</tr>
<tr>
<td>Sparky VI F303</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F303CCT6</td>
<td>72MHz</td>
<td>256K</td>
</tr>
<tr>
<td>Taida Century nRF52 mini board</td>
<td>Nordic nRF52</td>
<td>External</td>
<td>NRF51822</td>
<td>64MHz</td>
<td>512K</td>
</tr>
<tr>
<td>Tiny STM103T</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103TBU6</td>
<td>72MHz</td>
<td>128K</td>
</tr>
<tr>
<td>VAKE v1.0</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F446RET6</td>
<td>180MHz</td>
<td>512K</td>
</tr>
<tr>
<td>Waveshare BLE400</td>
<td>Nordic nRF51</td>
<td>External</td>
<td>NRF51822</td>
<td>32MHz</td>
<td>256K</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](#).

#### 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 = ...
done
```

If you would like to use this tool for firmware uploading, please change upload protocol:

```ini
[env:myenv]
platform = ...
board = ...
done
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><strong>TI</strong></td>
<td>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.</td>
</tr>
</tbody>
</table>

Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached 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/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.</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><strong>TI LaunchPad (Stellaris)</strong> 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><strong>TI LaunchPad (Tiva C)</strong> 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><strong>TI LaunchPad (Tiva C)</strong> w/ tm4c129 (120MHz)</td>
<td>TI</td>
<td>On-board</td>
<td>LPTM4C1294NCPL</td>
<td>120MHz</td>
<td>1MB</td>
<td>256KB</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 = ...
debug_tool = tumpa
```

If you would like to use this tool for firmware uploading, please change upload protocol:

```
[env:myenv]
platform = ...
board = ...
debug_tool = tumpa
upload_protocol = tumpa
```

More options:
• Debugging options
• Upload options

Drivers

Windows  See https://community.platformio.org/t/esp32-pio-unified-debugger/4541/20

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>TUMPA JTAG 20-Pin Connector</th>
<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>
<td></td>
</tr>
<tr>
<td>4 GND</td>
<td>Digital ground</td>
<td></td>
</tr>
<tr>
<td>5 TDI</td>
<td>Test Data In pin</td>
<td></td>
</tr>
<tr>
<td>7 TMS</td>
<td>Test Mode State pin</td>
<td></td>
</tr>
<tr>
<td>9 TCK</td>
<td>JTAG Return Test Clock</td>
<td></td>
</tr>
<tr>
<td>13 TDO</td>
<td>Test Data Out pin</td>
<td></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>
<td></td>
</tr>
</tbody>
</table>
Serial Wire Mode Interface (SWD)

<table>
<thead>
<tr>
<th>TUMPA 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></td>
<td>Positive Supply Voltage — Power supply for JTAG interface drivers</td>
</tr>
<tr>
<td>4</td>
<td>GND</td>
<td></td>
<td>Digital ground</td>
</tr>
<tr>
<td>7</td>
<td>SWDIO</td>
<td></td>
<td>Data I/O</td>
</tr>
<tr>
<td>9</td>
<td>SWCLK</td>
<td></td>
<td>Clock</td>
</tr>
<tr>
<td>15</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>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, this platform is ported by Sipeed.</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>Freedom E SDK</td>
<td>Open Source Software for Developing on the SiFive Freedom E Platform</td>
</tr>
<tr>
<td>Kendryte Standalone SDK</td>
<td>Kendryte Standalone SDK without OS support</td>
</tr>
<tr>
<td>Kendryte FreeRTOS SDK</td>
<td>Kendryte SDK with FreeRTOS support</td>
</tr>
<tr>
<td>Simba</td>
<td>Simba is an RTOS and build framework. It 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.
## PlatformIO Documentation, Release 4.0.1b3

<table>
<thead>
<tr>
<th>Name</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>340MHz</td>
<td>4MB</td>
<td>320KB</td>
</tr>
<tr>
<td>Adafruit ESP32 Feather</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>340MHz</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>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>ESP32v1 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>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>4MB</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 Miniina</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 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>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>NodeMCU32-32S</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-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></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>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 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>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 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>

### Custom

**Configuration**  
`debug_tool = custom`

PIO Unified Debugger can be configured from “platformio.ini” (Project Configuration File):
Examples

- Black Magic Probe
- J-Link and ST Nucleo
- J-Link as debugger and uploader
- ST-Util and ST-Link
- OpenOCD and ST-Link
- pyOCD and CMSIS-DAP

Black Magic Probe

Black Magic Probe with a custom `debug_port` (list ports with `platformio device list`)

```bash
[env:debug]
platform = ...
board = ...
framework = ...
debug_tool = custom
; set here a valid port...
debug_port = /dev/cu.usbmodem7BB07991
debug_init_cmds =
  target extended-remote $DEBUG_PORT
  monitor swdp_scan
  attach 1
  set mem inaccessible-by-default off
$INIT_BREAK
$LOAD_CMDS
```

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.

```bash
[env:debug_jlink]
platform = ststm32
framework = mbed
board = nucleo_f446re
debug_tool = custom
debug_server =
```

(continues on next page)
J-Link as debugger and uploader

Segger J-Link probe as debugger and uploader for a custom Teensy-based 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

```
[env:jlink_debug_and_upload]
platform = teensy
framework = arduino
board = teensy31
extra_scripts = extra_script.py
upload_protocol = custom
debug_tool = custom
debug_server = /full/path/to/JLinkGDBServerCL
    -singlerun
    -if
    SWD
    -select
    USB
    -port
    2331
    -device
    STM32F446RE
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")

# Optional block, only for Teensy
env.AddPostAction(""
    "$BUILD_DIR/firmware.hex",
    env.VerboseAction(" ".join([ 
        "sed", "-i.bak",
        "s/:10040000FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF0000000000000000000000000000000000:", 
        "$BUILD_DIR/firmware.hex"
    ]")
)```

(continues on next page)
Fixing "$BUILD_DIR/firmware.hex secure flash flags"

```python
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
```

```ruby
env.Replace(
    _jlink_cmd_script=_jlink_cmd_script,
    UPLOADCMD='$UPLOADER $UPLOADFLAGS -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:

```ini
[env:debug]
platform = stm32
framework = mbed
board = ...
download_tool = custom
download_port = 4242
download_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:

```ini
[env:debug]
platform = stm32
framework = mbed
board = ...
download_tool = custom
download_server = $PLATFORMIO_CORE_DIR/packages/tool-openocd/bin/openocd -f $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
```
## 1.16.4 CLI Guide

### 1.16.5 Platforms

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<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>Atmel SAM</td>
<td>Atmel SAM 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 Kinetics</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>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, this platform is ported by Sipeed.</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>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>Samsung ARTIK</td>
<td>The Samsung ARTIK Smart IoT platform brings hardware modules and cloud services together, with built-in security and an ecosystem of tools and partners to speed up your time-to-market.</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>
<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>
<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>
1.16.6 Frameworks

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino</td>
<td>Arduino Wiring-based Framework allows writing cross-platform software to control devices attached 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>Freedom E SDK</td>
<td>Open Source Software for Developing on the SiFive Freedom E Platform</td>
</tr>
<tr>
<td>Kendryte</td>
<td>Kendryte Standalone SDK without OS support</td>
</tr>
<tr>
<td>Kendryte FreeRTOS SDK</td>
<td>Kendryte SDK with FreeRTOS support</td>
</tr>
<tr>
<td>libOpenCM3</td>
<td>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.</td>
</tr>
<tr>
<td>mbed</td>
<td>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.</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>Simba</td>
<td>Simba is an RTOS and build framework. It aims to make embedded programming easy and portable.</td>
</tr>
<tr>
<td>SPL</td>
<td>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.</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>Tizen RT</td>
<td>Tizen RT is a lightweight RTOS-based platform to support low-end IoT devices</td>
</tr>
</tbody>
</table>

1.16.7 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 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>

## 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 OpenIMU 300ZA</td>
<td>Aceinna IMU</td>
<td>External</td>
<td>STM32F405RG</td>
<td>120MHz</td>
<td>1MB</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 Bluefruit nRF52832 Feather</td>
<td>Nordic nRF52</td>
<td>On-board</td>
<td>NRF52832</td>
<td>64MHz</td>
<td>512KB</td>
<td>64KB</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 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 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 M4 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 nRF52840 Express</td>
<td>Nordic nRF52</td>
<td>On-board</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>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 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 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 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 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 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 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>
### PlatformIO Documentation, Release 4.0.1b3

**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>

**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>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>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>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>On-board</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 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>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>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>

**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>

1.16. PIO Unified Debugger
### 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>External</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>External</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>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 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 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>

### 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>
### 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>

### DSIKE

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

### 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 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>192KB</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><strong>ESP32vn IoT Uno</strong></td>
<td>Espressif 32</td>
<td>External</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>

### 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><strong>Embedded Artists LPC4088 Display Module</strong></td>
<td>NXP LPC</td>
<td>On-board</td>
<td>LPC4088</td>
<td>120MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
<tr>
<td><strong>Embedded Artists LPC4088 QuickStart Board</strong></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>

### 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>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>
## 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</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>MK20DX128VHL5</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>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>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>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>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>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>MKL25Z128VLM4</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>MKL27Z64VLM4</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>MKL43Z256VLM4</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>MKL46Z256VLL4</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>MKL82Z128VLM7</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>

1.16. PIO Unified Debugger
## 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>192KB</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>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>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>STM32F103T8 (20k RAM. 64 Flash)</td>
<td>ST STM32</td>
<td>External</td>
<td>STM32F103T8T6</td>
<td>72MHz</td>
<td>20KB</td>
<td>64KB</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>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>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>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>
<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>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>4MB</td>
<td>192KB</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>STM32F103TBU6</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>

## 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>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>
### 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>

### 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>

### 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>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-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>

### Macchina

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Macchina M2</td>
<td>Atmel SAM</td>
<td>External</td>
<td>AT91SAM3X8E</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</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>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>

1752 Chapter 1. Contents
### 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>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>

### 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>

### 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>

### MikroElektronik

<table>
<thead>
<tr>
<th>Name</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</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>ST32F411RET6</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>ST32F411RET6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>MultiTech mDot</td>
<td>ST STM32</td>
<td>External</td>
<td>ST32F411RET6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>MultiTech xDot</td>
<td>ST STM32</td>
<td>External</td>
<td>ST32L151CCU6</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>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 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 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>

### 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-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>

### 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>

### 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>32KB</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>16KB</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>

### 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 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>

### 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>

### 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>STM32L476RGRT6</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</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 ONE</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21J18A</td>
<td>48MHz</td>
<td>256KB</td>
<td>32KB</td>
</tr>
<tr>
<td>SODAQ SARA</td>
<td>Atmel SAM</td>
<td>External</td>
<td>SAMD21G18A</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>
</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>32F723EDISCOVERY</td>
<td>ST ST32</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>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>STM32F407VE</td>
<td>168MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Black STM32F407VG</td>
<td>ST ST32</td>
<td>External</td>
<td>STM32F407VG</td>
<td>168MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Black STM32F407ZE</td>
<td>ST ST32</td>
<td>External</td>
<td>STM32F407ZE</td>
<td>168MHz</td>
<td>1MB</td>
<td>192KB</td>
</tr>
<tr>
<td>Blue STM32F407VE Mini</td>
<td>ST ST32</td>
<td>External</td>
<td>STM32F407VE</td>
<td>168MHz</td>
<td>512KB</td>
<td>192KB</td>
</tr>
<tr>
<td>Nucleo G071RB</td>
<td>ST ST32</td>
<td>On-board</td>
<td>STM32G071RBT6</td>
<td>24MHz</td>
<td>2MB</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>192KB</td>
</tr>
<tr>
<td>ST 32F3348DISCOVERY</td>
<td>ST ST32</td>
<td>On-board</td>
<td>STM32F334C8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>12KB</td>
</tr>
<tr>
<td>ST 32F401CDISCOVERY</td>
<td>ST ST32</td>
<td>On-board</td>
<td>STM32F401VCT6</td>
<td>84MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>ST 32F411EDISCOVERY</td>
<td>ST ST32</td>
<td>On-board</td>
<td>STM32F411VE</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST 32F413HDISCOVERY</td>
<td>ST ST32</td>
<td>On-board</td>
<td>STM32F413ZHT6</td>
<td>100MHz</td>
<td>512KB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST 32F429IDISCOVERY</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 32F469IDISCOVERY</td>
<td>ST ST32</td>
<td>On-board</td>
<td>STM32F469NIH6</td>
<td>180MHz</td>
<td>1MB</td>
<td>384KB</td>
</tr>
<tr>
<td>ST 32F474GDISCOVERY</td>
<td>ST ST32</td>
<td>On-board</td>
<td>STM32F474NGH6</td>
<td>216MHz</td>
<td>1MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ST 32F479IDISCOVERY</td>
<td>ST ST32</td>
<td>On-board</td>
<td>STM32F479NIH6</td>
<td>216MHz</td>
<td>1MB</td>
<td>512KB</td>
</tr>
<tr>
<td>ST 32L0538DISCOVERY</td>
<td>ST ST32</td>
<td>On-board</td>
<td>STM32L053C8T6</td>
<td>32MHz</td>
<td>64KB</td>
<td>8KB</td>
</tr>
<tr>
<td>ST 32L100DISCOVERY</td>
<td>ST ST32</td>
<td>On-board</td>
<td>STM32L100RCT6</td>
<td>32MHz</td>
<td>256KB</td>
<td>16KB</td>
</tr>
<tr>
<td>ST 32L476GDISCOVERY</td>
<td>ST ST32</td>
<td>On-board</td>
<td>STM32L476VGT6</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST 32L496GDISCOVERY</td>
<td>ST ST32</td>
<td>On-board</td>
<td>STM32L496AGI6</td>
<td>80MHz</td>
<td>1MB</td>
<td>320KB</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 DISCO-L475VG-IOT01A</td>
<td>ST ST32</td>
<td>On-board</td>
<td>STM32L475VGT6</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</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>STM32F030R8T6</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>STM32F031K6T6</td>
<td>48MHz</td>
<td>32KB</td>
<td>4KB</td>
</tr>
<tr>
<td>ST Nucleo F042K6</td>
<td>ST ST32</td>
<td>On-board</td>
<td>STM32F042K6T6</td>
<td>48MHz</td>
<td>32KB</td>
<td>6KB</td>
</tr>
<tr>
<td>ST Nucleo F070RB</td>
<td>ST ST32</td>
<td>On-board</td>
<td>STM32F070RBT6</td>
<td>48MHz</td>
<td>128KB</td>
<td>16KB</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>STM32F207ZGT6</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>STM32F302R8T6</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>STM32F303K8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>16KB</td>
</tr>
<tr>
<td>ST Nucleo F303RE</td>
<td>ST ST32</td>
<td>On-board</td>
<td>STM32F303RET6</td>
<td>72MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>ST Nucleo F303ZE</td>
<td>ST ST32</td>
<td>On-board</td>
<td>STM32F303ZET6</td>
<td>72MHz</td>
<td>512KB</td>
<td>64KB</td>
</tr>
<tr>
<td>ST Nucleo F334R8</td>
<td>ST ST32</td>
<td>On-board</td>
<td>STM32F334R8T6</td>
<td>72MHz</td>
<td>64KB</td>
<td>16KB</td>
</tr>
</tbody>
</table>

Continued on next page
Table 40 – 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 F401RE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F401RE6</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>STM32F410RB6</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>STM32F411RE6</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>STM32F412ZG6</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>STM32F413ZH6</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>STM32F429ZI6</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>STM32F439ZI6</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>STM32F446RE6</td>
<td>180MHz</td>
<td>512KB</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>512KB</td>
<td>256KB</td>
</tr>
<tr>
<td>ST Nucleo F746ZG</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F746ZG6</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>STM32F756ZG6</td>
<td>1MB</td>
<td>1MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ST Nucleo F767ZI</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F767ZI6</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>STM32L011K4T6</td>
<td>400MHz</td>
<td>2MB</td>
<td>1MB</td>
</tr>
<tr>
<td>ST Nucleo L011K6</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L011K6T6</td>
<td>32MHz</td>
<td>16KB</td>
<td>2KB</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>8KB</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>8KB</td>
</tr>
<tr>
<td>ST Nucleo L073RZ</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L073RZ6</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>STM32L152RE6</td>
<td>32MHz</td>
<td>512KB</td>
<td>80KB</td>
</tr>
<tr>
<td>ST Nucleo L412KB</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L412KB6</td>
<td>80MHz</td>
<td>128KB</td>
<td>40KB</td>
</tr>
<tr>
<td>ST Nucleo L432KC</td>
<td>ST STM32</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>ST STM32</td>
<td>On-board</td>
<td>STM32L433RC6</td>
<td>80MHz</td>
<td>256KB</td>
<td>64KB</td>
</tr>
<tr>
<td>ST Nucleo L452RE</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L452RE6</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>STM32L476RG6</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>STM32L486RG6</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>STM32L496ZG6</td>
<td>80MHz</td>
<td>1MB</td>
<td>128KB</td>
</tr>
<tr>
<td>ST Nucleo L496ZG-P</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L496ZG6P</td>
<td>80MHz</td>
<td>1MB</td>
<td>320KB</td>
</tr>
<tr>
<td>ST Nucleo L4RSZI</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L4R5ZT6</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>STM32F303VC6</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>STM32F407VG6</td>
<td>168MHz</td>
<td>1MB</td>
<td>192KB</td>
</tr>
<tr>
<td>ST STM32L073Z-EVAL</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L073VZ6</td>
<td>32MHz</td>
<td>192KB</td>
<td>20KB</td>
</tr>
<tr>
<td>ST STM32L1DISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32L152KB6</td>
<td>32MHz</td>
<td>128KB</td>
<td>16KB</td>
</tr>
<tr>
<td>ST STM32VLDISCOVERY</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F100KB6</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>STM8S105G6T6</td>
<td>16MHz</td>
<td>32KB</td>
<td>2KB</td>
</tr>
<tr>
<td>STM32F7308-DK</td>
<td>ST STM32</td>
<td>On-board</td>
<td>STM32F7308H6</td>
<td>216MHz</td>
<td>64KB</td>
<td>340KB</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</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</td>
<td>External</td>
<td>AT91SAM3X8E</td>
<td>84MHz</td>
<td>512KB</td>
<td>96KB</td>
</tr>
</tbody>
</table>

1758 Chapter 1. Contents
### Samsung

<table>
<thead>
<tr>
<th>Name</th>
<th>Platform</th>
<th>Debug</th>
<th>MCU</th>
<th>Frequency</th>
<th>Flash</th>
<th>RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Samsung ARTIK</td>
<td>Samsung ARTIK</td>
<td>On-board</td>
<td>S5JT200</td>
<td>320MHz</td>
<td>8MB</td>
<td>1.25MB</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 LoRaWAN</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>STM32F407VG6T6</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>
</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>HiFive</td>
<td>SiFive</td>
<td>On-board</td>
<td>FE310</td>
<td>320MHz</td>
<td>16MB</td>
<td>16KB</td>
</tr>
<tr>
<td>HiFive 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</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>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>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</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</td>
<td>On-board</td>
<td>EFM32HG322F64</td>
<td>25MHz</td>
<td>64KB</td>
<td>8KB</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>Thundeboard Sense 2 Sensor-to-Cloud Advanced IoT</td>
<td>Silicon Labs</td>
<td>On-board</td>
<td>EFR32MG12P432FID</td>
<td>240MHz</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>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>
</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>

## 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>
<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>
</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>

## 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>16KB</td>
<td>512B</td>
</tr>
<tr>
<td>TI LaunchPad (Stellaris) w/ tm4f120 (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>LPTM4C1294NCDD</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>16MHz</td>
<td>128KB</td>
<td>8KB</td>
</tr>
<tr>
<td>TI LaunchPad 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 EXP430FR5969LP</td>
<td>TI MSP430</td>
<td>On-board</td>
<td>MSP430FR5969</td>
<td>8MHz</td>
<td>64KB</td>
<td>2KB</td>
</tr>
<tr>
<td>TI LaunchPad EXP430FR6989LP</td>
<td>TI MSP430</td>
<td>On-board</td>
<td>MSP430FR6989</td>
<td>8MHz</td>
<td>127KB</td>
<td>2KB</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 T-Beam</td>
<td>Espressif 32</td>
<td>External</td>
<td>ESP32</td>
<td>240MHz</td>
<td>4MB</td>
<td>1.25MB</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 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 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>

### 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>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>

### 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 VBLUNOS1</td>
<td>Nordic nRF51</td>
<td>On-board</td>
<td>NRF51822</td>
<td>16MHz</td>
<td>128KB</td>
<td>32KB</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 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>
</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>

## 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>

## 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>

## 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>
## 1.17 PIO Unit Testing

PIO Unit Testing allows segregating each part of the firmware/program and test that the individual parts are working correctly. Using **PIO Unit Testing Engine** 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 in both cases. In the last case, PIO Plus builds firmware on the host machine, uploads into a target device, starts tests and collects test results into test reports. The final information will be shown on the host side with informative output and statistic.

Using PIO Remote you can start unit tests on the Remote Device from anywhere in the world or integrate with Continuous Integration systems.

1.17.1 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 platformio test command.
1.17.2 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.

### 1.17.3 Configuration

PIO Unit Testing Engine can be configured from “platformio.ini” (Project Configuration File).

### 1.17.4 Test Types

#### Desktop

PIO 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

PIO Unit Testing Engine builds a special firmware for a target device (board) and program it. Then, it connects to this device using configured Serial test_port and communicate via test_transport. Finally, it runs test on an embedded side, collects results, analyzes them and provides a summary on a host machine side (desktop).

**Note:** Please note that the PIO Unit Testing Engine uses the first available Serial/UART implementation (depending on a framework) as a communication interface between the PIO 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 base interface.

### 1.17.5 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 platformio test command.
Remote

Allows you to run tests 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 PIO Remote. In this case, you need to use the special command `platformio 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.

### 1.17.6 Workflow

1. Create PlatformIO project using the `platformio init` command. For Desktop Unit Testing (on a host machine), you need to use `Native`.

```
; 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
;
; Embedded platforms
;
[env:uno]
platform = atmelavr
framework = arduino
board = uno

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

; Desktop platforms (Win, Mac, Linux, Raspberry Pi, etc)
; See https://platformio.org/platforms/native
;
[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 >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 **PIO Unit Testing Engine** will treat the source code of `test` folder as SINGLE test.

5. Run test using the `platformio test` command.

**Shared Code**

**PIO 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 to split a program source code into multiple components and place them into `lib_dir` (project’s private libraries and components). **Library Dependency Finder (LDF)** will find these libraries automatically and include to build process. You will need to include any library/component header file in your test or program source code via `#include <MyComponent.h>`. See Local & Embedded: Calculator example. We have “calculator” component in `lib_dir` folder and include it in tests and 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):

   ```ini
   [env:myenv]
   platform = ...
   test_build_project_src = true
   ```

   This is very useful if you unit test independent library 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.

**1.17.7 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)
  - TEST_ASSERT_EQUAL(expected, actual)
  - TEST_ASSERT_INT_WITHIN(delta, expected, actual)

- 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)

### 1.17.8 CLI Guide

### 1.18 Cloud & Desktop IDE

#### 1.18.1 PlatformIO IDE

“PlatformIO IDE” is an official extension/plugin which provides native integration with IDEs/Text Editors and contains built-in PlatformIO Core (CLI) and PlatformIO Home.

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

#### 1.18.2 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
  - Integration
  - Quick Start
- PlatformIO Build System
- Remote Device Manager
- Remote Firmware Uploading
- Remote Serial Port Monitor
- Multi-Project workspace
Note:

1. Please make sure to read "PIO Remote" guide first.
2. You need "PIO Account" if you don’t have it. Registration is FREE.
3. You should have a running "PIO Remote Agent" on a remote machine where hardware devices are connected physically or accessible for the remote operations. See "PIO Remote 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
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 PIO Account using platformio 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:

```bash
platformio init --board <ID>
# initialize project for Arduino Uno
platformio init --board uno
```

To get board ID please use platformio 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:

```cpp
#include <Arduino.h>

int i = 0;

void setup() {
    Serial.begin(9600);
    Serial.println("Hello Cloud9!");
}
```

(continues on next page)
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:

   - `platformio 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
   - `platformio remote device list` - list available remote devices
PlatformIO Documentation, Release 4.0.1b3

- platformio remote device monitor - Remote Serial Port Monitor

If you are interested in better integration with Cloud9 and GUI, please read guide below where we will explain how to create custom Build System for PlatformIO and own Runners.

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:

```
{
    "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 PIO Remote. You can list remote devices that are connected to host machine where PIO 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 PIO Remote. You can deploy firmware (program) to any devices which are visible for PIO 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 4.0.1b3

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
text data bss dec hex filename
2324 48 168 2540 9ec .pioenvs/uno/firmware.elf
================================================================================== [SUCCESS] Took 0.46 seconds

Uploading firmware remotely

Verbose mode can be enabled via `-v, --verbose` option
Looking for upload port...
Auto-detected: /dev/cu.usbmodemF1A1411
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 (2372 bytes):
Writing  | 100% 0.39s
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% 0.31s
avrdude: verifying ...
avrdude: 2372 bytes of flash verified

avrdude done. Thank you.
================================================================================== [SUCCESS] Took 3.19 seconds
Remote Serial Port Monitor

Remote Serial Port Monitor works in pair with PIO Remote. You can read or send data to any device that is connected to host machine where PIO Remote Agent is started. To list active agents please use this command platformio 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”

![Image of Run Configurations](image-url)

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 `platformio 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 *PIO Remote* guide first.
2. You need *PIO Account* if you don’t have it. Registration is FREE.
3. You should have a running *PIO Remote Agent* on a remote machine where hardware devices are connected physically or accessible for the remote operations. See *PIO Remote 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

```bash
sudo python -c "$(curl -fsSL https://raw.githubusercontent.com/platformio/platformio/develop/scripts/get-platformio.py)"
```
5. Log in to PIO Account using platformio 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:

```bash
platformio init --board <ID>
```

# initialize project for Arduino Uno
platformio init --board uno

To get board ID please use platformio 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:

```cpp
#include <Arduino.h>

int i = 0;

void setup() {
```
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), then you can process project using Cloud IDE SSH Terminal and the next commands:
- `platformio 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
- `platformio remote device list` - list available remote devices
- `platformio 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.
```
```
```
Remote Device Manager

Remote Device Manager works in pair with PIP Remote. You can list remote devices that are connected to host machine where PIP Remote Agent is started or are visible for it.

1. Open Cloud IDE SSH Terminal
2. Paste this command
   
   ```
   pio remote device list
   ```

Remote Firmware Uploading

Remote Firmware Uploading works in pair with PIP Remote. You can deploy firmware to any devices which are visible for PIP Remote Agent.

1. Open Cloud IDE SSH Terminal
2. Paste this command
   
   ```
   pio remote run -t upload
   ```
Remote Serial Port Monitor

Remote Serial Port Monitor works in pair with PIO Remote. You can read or send data to any device that is connected to host machine where PIO Remote Agent is started. To list active agents please use this command platformio remote agent list.

1. Open Cloud IDE SSH Terminal
2. Paste this command

```bash
pio remote device 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 `platformio run` or `platformio 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 `platformio 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 *PIO Remote* guide first.
2. You need *PIO Account* if you don’t have it. Registration is FREE.
3. You should have a running *PIO Remote Agent* on a remote machine where hardware devices are connected physically or accessible for the remote operations. See *PIO Remote Quick Start* for details.
Demo

1. Sign in to Codenvy (based on Eclipse Che). A registration is FREE and gives you unlimited private projects.
2. Open Workspaces tab

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 **PIO Account** using `platformio 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:
   - `platformio 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
- `platformio remote device list` - list available remote devices
- `platformio 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 `platformio run` or `platformio 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 `platformio remote run --project-dir` option.

**1.18.3 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:
  - 550+ embedded boards
  - 30+ development platforms
  - 15+ frameworks
- **PIO Unified Debugger**
- **PIO Remote**
- **PIO 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)

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.

Contents

• Installation
  – I. Atom
  – II. Clang for Intelligent Code Completion
• Quick Start
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 official GitHub’s 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 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. TEMPORARY, WE SUPPORT ONLY CLANG 3.9.
If you see 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 Clang completion engine that should work after it 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 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.

Launch

After installation, you launch PlatformIO IDE by opening Atom. Once Atom is opened, PlatformIO IDE auto installer will continue to install dependent packages and *PlatformIO Core (CLI)*. Please be patient and let the installation
complete. In the final result PlatformIO IDE will ask you to reload Atom window to apply installed components. Please click on **Reload Now**. After it PlatformIO IDE is ready for using. Happy coding!

**Setting Up the Project**

1. Click on “PlatformIO Home” button on the **PlatformIO Toolbar**

2. Click on “New Project”, select a board and create new PlatformIO Project
3. Open `main.cpp` file from `src` folder and replace its contents with the next:

```c
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.
 */
```

(continues on next page)
```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 project (build, clean, upload firmware, run other targets) using:

- **PlatformIO Toolbar**
- **Menu item PlatformIO**
- **Building / Uploading / Targets** and hotkeys
5. Run **Build** and you should see green “success” result in the building 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 run built-in terminal interface choose Menu: PlatformIO > Terminal or press the corresponding icon in the PlatformIO toolbar:
It provides you fast access to all set of powerful *PlatformIO Core (CLI)* CLI commands:
8. To run built-in “Serial Monitor” choose **Menu: PlatformIO > Serial Monitor** or press the corresponding icon in the PlatformIO toolbar:
It has several settings to adjust your connection:
And 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 hint (delay mouse on it).

- **PlatformIO Home**
- PlatformIO: Build
- PlatformIO: Upload
- PlatformIO: Clean
- **PIO Unified Debugger**
• Run other target (Build environments, PIO Unit Testing)
• Toggle build panel
• ||
• Find in Project...
• PIO 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.

More options Menu: PlatformIO > Settings > Build.

Intelligent Code Completion

PlatformIO IDE uses clang for the Intelligent Code Completion. To install it or check if it is already installed, please follow to step II. Clang for Intelligent Code Completion from Installation guide.

Warning: The libraries which are added/installed after initializing process will not be reflected in 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 PIO 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 PIO 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

```
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

```
rm /usr/local/bin/atom
rm /usr/local/bin/apm
rm -rf ~/.atom
rm -rf ~/.config/Atom-Shell
rm -rf /usr/local/share/atom/
rm -rf /Library/Caches/com.github.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

The CLion is a cross-platform C/C++ IDE for Linux, OS X, and Windows integrated with the CMake build system. The initial version will support the GCC and Clang compilers and GDB debugger. Clion includes such features as a smart editor, code quality assurance, automated refactorings, project manager, integrated version control systems.

Refer to the CLion Documentation page for more detailed information.
Contents

• Integration

• Known issues
  – Arduino .ino files are not supported

• Articles / Manuals

Integration

Integration process consists of these steps:

1. Install File Watchers plugin via “Clion: Preferences > Plugins”. We need it to automatically update project configuration when changes are made in “platformio.ini” (Project Configuration File)

2. Open system Terminal and install PlatformIO Core (CLI)

3. Create new folder for your project and change directory (cd) to it
4. Generate a project using PIO Core Project Generator (platformio init --ide)

5. Import project in IDE.

Choose board ID using platformio boards or Embedded Boards Explorer command and generate project via platformio init --ide command:

```bash
platformio init --ide clion --board <ID>

# For example, generate project for Arduino UNO
platformio init --ide clion --board uno
```

Then:

1. Place source files (*.c, *.cpp, *.h, *.hpp) to src directory and repeat platformio init --ide command above (to refresh source files list)

2. Import this project via Menu: File > Import Project and specify root directory where is located "platformio.ini" (Project Configuration File)

3. Open source file from src directory

4. Build project (DO NOT use “Run” button, see marks on the screenshot above): Menu: Run > Build.

**Warning:**

1. **PlatformIO Core (CLI) DOES NOT** depend on Cmake, it has own cross-platform Build System. All data related to build flags and source code filtering should be specified using Build options in "platformio.ini" (Project Configuration File).

2. See know issue: Arduino .ino files are not supported and how to resolve it.

There are 6 predefined targets for building (NOT FOR RUNNING, see marks on the screenshot above):

- PLATFORMIO_BUILD - Build project without auto-uploading
- PLATFORMIO_BUILD_VERBOSE - Build project without auto-uploading in verbose mode
- PLATFORMIO_UPLOAD - Build and upload (if no errors)
- PLATFORMIO_CLEAN - Clean compiled objects
- PLATFORMIO_MONITOR - Device monitor platformio device monitor
- PLATFORMIO_TEST - PIO Unit Testing
- PLATFORMIO_PROGRAM - Build and upload using external programmer (if no errors), see Upload using Programmer
- PLATFORMIO_UPLOADFS - Upload files to file system SPIFFS, see Uploading files to file system SPIFFS
- PLATFORMIO_UPDATE - Update installed platforms and libraries via platformio update
- PLATFORMIO_REBUILD_PROJECT_INDEX - Rebuild C/C++ Index for the Project. Allows one to fix code completion and code linting issues.

**Warning:** The libraries which are added, installed or used in a project after generating process will not be reflected in IDE. Please run PLATFORMIO_REBUILD_PROJECT_INDEX target to resolve this issue.
Known issues

**Arduino .ino files are not supported**

CLion uses “CMake” tool for code completion and code linting. As result, it 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.

See how to *Convert Arduino file to C++ manually*.

**Articles / Manuals**

- Dec 01, 2015 - JetBrains CLion Blog - C++ Annotated: Fall 2015. Arduino Support in CLion using PlatformIO
- 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)

See more Articles about us.

**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 PIO Core Project Generator (platformio init --ide)
4. Import project in IDE.

Choose board ID using platformio boards or Embedded Boards Explorer command and generate project via platformio init --ide command:

```
platformio init --ide codeblocks --board <ID>
```
# For example, generate project for Arduino UNO
platformio 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 reinitialize project using `platformio 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
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 PIO Core Project Generator (platformio init --ide)
4. Import project in IDE.

Choose board ID using platformio boards or Embedded Boards Explorer command and generate project via platformio init --ide command:

```
platformio init --ide eclipse --board <ID>
# For example, generate project for Arduino UNO
platformio 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 - PIO Unit Testing
   - PlatformIO: Upload - Build and upload (if no errors)
   - PlatformIO: Upload using Programmer see Upload using Programmer
   - PlatformIO: Upload SPIFFS image see Uploading files to file system SPIFFS
   - PlatformIO: Update platforms and libraries - Update installed platforms and libraries via platformio 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: 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 PIO Unified Debugger 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
    - PlatformIO-Mode
    - Project Generator

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 PIO Core Project Generator (platformio init --ide)
4. Import project in IDE.

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.

Code completion can optionally be provided by installing irony-mode.

Project Generator

Choose board ID using platformio boards or Embedded Boards Explorer command and generate project via platformio init --ide command:

```
platformio init --ide emacs --board <ID>
```

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 Uploading files to file system SPIFFS). (C-c i s)
- `platformio_update` - Update installed platforms and libraries. (C-c i d)

**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 `platformio init` (repeat it).

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).
Contents

- NetBeans
  - Integration
  - Articles / Manuals
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 PIO Core Project Generator (platformio init --ide)
4. Import project in IDE.

Choose board ID using platformio boards or Embedded Boards Explorer command and generate project via platformio init --ide command:

```
platformio init --ide netbeans --board <ID>

# For example, generate project for Arduino UNO
platformio 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 platformio 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

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 PIO Core Project Generator(*platformio init --ide*)
4. Import project in IDE.
Project Generator

Choose board ID using platformio boards or Embedded Boards Explorer command and generate project via platformio init --ide command:

```bash
platformio init --ide qtcreator --board <ID>
# For example, generate project for Arduino UNO
platformio 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):

```bash
# 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 `platformio 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:

```plaintext
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 platformio project with `platformio 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 extensible 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 PIO Core Project Generator (platformio init --ide)
4. Import project in IDE.

Choose board ID using platformio boards or Embedded Boards Explorer command and generate project via platformio init --ide command:

```
platformio init --ide sublimetext --board <ID>
```

# For example, generate project for Arduino UNO
```
platformio init --ide sublimetext --board uno
```

Then:

1. Import project via Menu: Project > Open Project... and select platformio.sublime-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 - PIO Unit Testing
- PlatformIO - Upload - Build and upload (if no errors)
- PlatformIO - Upload using Programmer see Upload using Programmer
• PlatformIO - Upload SPIFFS image see *Uploading files to file system SPIFFS*
• PlatformIO - Update platforms and libraries - Update installed platforms and libraries via *platformio 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...

void setup()
{
  pinMode(WLED, OUTPUT); // set pin as output
}
```

(continues on next page)
```c
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
[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 `PIO Unified Debugger` 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 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 PIO Core Project Generator (`platformio init --ide`)
4. Import project in IDE.
“Neomake-PlatformIO” Plugin

Please visit neomake-platformio for the further installation steps and documentation.

Project Generator

Choose board ID using platformio boards or Embedded Boards Explorer command and generate project via platformio init --ide command:

```
platformio init --ide vim --board <ID>
```

Recommended bundles:

- Syntax highlight - Arduino-syntax-file
- Code Completion - YouCompleteMe (see configuration example by Anthony Ford PlatformIO/YouCompleteMe Integration)
- Syntax checking - Syntastic

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:    platformio -f -c vim run
upload: platformio -f -c vim run --target upload
clean:  platformio -f -c vim run --target clean
program:platformio -f -c vim run --target program
uploadfs: platformio -f -c vim run --target uploadfs
update:  platformio -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 Uploading files to file system SPIFFS
- Update platforms and libraries - Update installed platforms and libraries via platformio 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
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 PIO Core Project Generator (platformio init --ide)
4. Import project in IDE.

Project Generator

Choose board ID using platformio boards or Embedded Boards Explorer command and generate project via platformio init --ide command:

```
platformio init --ide visualstudio --board <ID>
# For example, generate project for Arduino UNO
platformio 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 platformio 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:

```
$(HOMEDRIVE)$(HOMEPATH)\.platformio\packages\toolchain-atmelavr\avr\include
$(HOMEDRIVE)$(HOMEPATH)\.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"
#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
}
```

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

PlatformIO IDE for VSCode

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

- Cross-platform build system without external dependencies to the OS software:
  - 550+ embedded boards
  - 30+ development platforms
  - 15+ frameworks
- PIO Unified Debugger
• **PIO Remote**
• **PIO 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**)

**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
  - Custom Build Task
- Key Bindings
- Project Tasks
  - Task Explorer
  - Task Runner
PlatformIO Documentation, Release 4.0.1b3

– **Custom Tasks**
  • Multi-project Workspaces
  • Serial Port Monitor
  • Debugging
    – Variable Format
    – Watchpoints
  • Install Shell Commands
  • Proxy Server Support
  • Settings
    – `platformio-ide.activateOnlyOnPlatformIOProject`
    – `platformio-ide.autoCloseSerialMonitor`
    – `platformio-ide.autoRebuildAutocompleteIndex`
    – `platformio-ide.buildTask`
    – `platformio-ide.customPATH`
    – `platformio-ide.forceUploadAndMonitor`
    – `platformio-ide.reopenSerialMonitorDelay`
    – `platformio-ide.updateTerminalPathConfiguration`
    – `platformio-ide.useBuiltinPIOCore`
    – `platformio-ide.useDevelopmentPIOCore`
  • Known issues
    – Package Manager 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 *PlatformIO IDE for 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 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:

```c
/**
 * 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)

Warning: The code below works only in pair with Arduino-based boards. Please follow to PlatformIO Project Examples repository for other pre-configured projects.
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 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
Custom Build Task

You can override default “PlatformIO: Build” task for “Build” command which is used by “Build” button in PlatformIO Toolbar and Key Bindings. See platformio-ide.buildTask setting in Proxy Server Support for more details.

Built-in PlatformIO tasks are available in “Menu > Terminal > Run Task…” list.

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.

Project Tasks

Task Explorer

PlatformIO provides access to “Project Task Explorer” where you can control build process of declared environments in “platformio.ini” (Project Configuration File). Project Task Explorer is located in VSCode Activity Bar under branded PlatformIO icon. You can also access it via “VSCode Menu > Open View…” > PlatformIO”.

1.18. Cloud & Desktop IDE
Task Runner

PlatformIO IDE provides base tasks Menu > Terminal > Run Task... (Build, Upload, Clean, Monitor, etc) and custom tasks per "platformio.ini" (Project Configuration File) environment ([env:***]). A default behavior is to use Terminal Panel for presentation. Also, we use dedicated panel per unique task.

PlatformIO IDE provides own Problems Matcher named $platformio. You can use it later if decide to change base task settings.

You can override existing task with own presentation options. For example, let configure PlatformIO Task Runner to use NEW Terminal panel per each “Build” command:

1. Please click on “gear” icon near “Build” task in Menu > Tasks
2. Replace template in tasks.json with this code

```json
{
    "version": "2.0.0",
    "tasks": [
```
Custom Tasks

Custom tasks can be added to tasks.json file located in .vscode folder in the root of project. Please read official documentation Tasks in VSCode.

This simple example demonstrates a custom build process in verbose mode. There are a lot of other commands, please read more about PlatformIO Core (CLI) and its commands (CLI Guide).

If platformio executable file is not in your system environment “PATH”, you can provide a full path to binary folder using options field for task. For example, platformio binary is located in home folder “~/.platformio/penv/bin”:

(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 one time. Read more in 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:

```
[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 PIO Unified Debugger. You should have PIO Account to work with it.

VSCode has a separate activity view named “Debug” (bug icon on the left toolbar). PIO Unified Debugger extends it with the next 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 2 pre-configured debugging configurations:

**PIO Debug Default configuration.** PlatformIO runs **Pre-Debug** task and builds project using **Debug Configuration**. Also, it checks for project changes.

**PIO Debug (skip Pre-Debug)** PlatformIO skips **Pre-Debug** stage and DOES NOT build or check project changes. If you do changes in project source files, they will not be reflected in a debug session until you switch back to “PIO Debug” configuration or manually run “Pre-Debug” task.

This configuration is very useful for quick debug session. It is super fast and skips different checks. You manually control project changes.

---

**Note:** Please note that **PIO Unified Debugger** will use the first declared build environment in "platformio.ini" (Project Configuration File) if **default_envs** option is not specified.
Variable Format

Currently, VSCode does not provide UI/API to change variable format. See related VSCode Issue #28025.

Temporary solution is to set the default numeric base in which the debugger displays numeric output using Debug Console (you will see it during active debugging session). For example, show variables in hexadecimal format (copy below and paste into “Debug Console”):

```
set output-radix 16
```

Possible values, listed in decimal base, are: 8, 10, 16.

Watchpoints

Please read GDB: Setting Watchpoints before.
Currently, VSCode does not provide an API to change value format of watch points. You can manually cast output setting it as a watch of a pointer:

- $pc, default decimal integer format
- *0x10012000, an address, default decimal integer format
- (void*)$pc, $pc register, hexadecimal format
- *(void**)0x10012000, an address, hexadecimal format

### Install Shell Commands

Please navigate to PIO Core Install Shell Commands.

### Proxy Server Support

There 2 options how to configure a proxy server:

1. Declare HTTP_PROXY and HTTPS_PROXY system environment variables (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”.

### Settings

How to configure VSCode settings?

**platformio-ide.activateOnlyOnPlatformIOProject**

Activate extension only when PlatformIO-based project (with "platformio.ini" (Project Configuration File)) is opened in workspace, default value is false.

**platformio-ide.autoCloseSerialMonitor**

Automatically close platformio device monitor before uploading/testing, default value is true.

**platformio-ide.autoRebuildAutocompleteIndex**

Automatically rebuild C/C++ Project Index when "platformio.ini" (Project Configuration File) is changed or when new libraries are installed, default value is true.

**platformio-ide.buildTask**

A build task (label) which is used by “Build” button in the PlatformIO Toolbar and Key Bindings. Default is set to PlatformIO: Build.

You can create custom Custom Tasks and assign to platformio-ide.buildTask.
**platformio-ide.customPATH**

Custom PATH for `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 custom version of `PlatformIO Core (CLI)`, default value is null.

**platformio-ide.forceUploadAndMonitor**

Force “Upload and Monitor” task for Upload (`platformio-ide.upload`) command, default value is false.

**platformio-ide.reopenSerialMonitorDelay**

Configure time in milliseconds after which reopen Serial Port Monitor, default value is 0, which means reopen instantly.

**platformio-ide.updateTerminalPathConfiguration**

Update Terminal configuration with patched PATH environment, default value is true.

**platformio-ide.useBuiltinPIOCore**

Use built-in `PlatformIO Core (CLI)`, default value is true.

**platformio-ide.useDevelopmentPIOCore**

Use development version of `PlatformIO Core (CLI)`, default value is false.

**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*.

Now, back to VSCode.

**Changelog**

Please visit releases page.
1.19 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.

`platformio 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.

1.19.1 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 `platformio 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.

## Contents

- AppVeyor
  - Integration
  - Examples

### Integration

Put `appveyor.yml` to the root directory of the GitHub repository.

```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: platformio ci --board=<ID_1> --board=<ID_2> --board=<ID_N>
```

For more details as for PlatformIO build process please look into `platformio ci` command.
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: platformio ci --lib="." --lib="C:\\spi4teensy" --board=uno --board=teensy31 --board=due
```

1.19.2 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 `platformio 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
    - Project as a library
    - Library dependencies
      - Install dependent library using Library Manager
      - Manually download dependent library and include in build process via `--lib` option
    - Custom Build Flags
    - Advanced configuration
  - Examples
Integration

Please make sure to read CircleCI Getting Started guide first.

```yaml
dependencies:
  pre:
    # Install the latest stable PlatformIO
    - sudo pip install -U platformio
  test:
    override:
      - platformio ci path/to/test/file.c --board=<ID_1> --board=<ID_2> --board=<ID_N>
      - platformio ci examples/file.ino --board=<ID_1> --board=<ID_2> --board=<ID_N>
      - platformio ci path/to/test/directory --board=<ID_1> --board=<ID_2> --board=<ID_N>
```

For more details as for PlatformIO build process please look into `platformio ci`.

Project as a library

When project is written as a library (where own examples or testing code use it), please use `--lib="."` option for `platformio ci` command

```bash
script:
  - platformio ci --lib="." --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 Manager

```yaml
dependencies:
  pre:
    # Install the latest stable PlatformIO
    - sudo pip install -U platformio
    # OneWire Library with ID=1 https://platformio.org/lib/show/1/OneWire
    - platformio lib -g install 1
  test:
    override:
      - platformio 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

```yaml
dependencies:
  pre:
    # Install the latest stable PlatformIO
    - sudo pip install -U platformio
    # OneWire Library with ID=1 https://platformio.org/lib/show/1/OneWire
    - platformio lib -g install 1
  test:
    override:
      - platformio ci path/to/test/file.c --board=<ID_1> --board=<ID_2> --board=<ID_N>
```

(continues on next page)
# 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:
    - platformio 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

```
machine:
  environment:
    PLATFORMIO_BUILD_FLAGS: -D 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 "platformio.ini" (Project Configuration File).

Instead of `--board` option, please use `platformio ci --project-conf`

```
test:
  override:
    - platformio ci path/to/test/file.c --project-conf=/path/to/platoformio.ini
```

**Examples**

1. Custom build flags

```
dependencies:
  cache_directories:
    - "~/.platformio"

pre:
  - sudo pip install -U platformio

  # pre-install PlatformIO development platforms, they will be cached
  - platformio platform install atmelavr atmelsam teensy

  # Libraries from PlatformIO Library Registry:
  #
  # https://platformio.org/lib/show/416/TinyGPS
```
# https://platformio.org/lib/show/417/SPI4Teensy3
- platformio lib -g install 416 417

test:
  override:
  - platformio ci examples/acm/acm_terminal --board=uno --board=teensy31 --
    
  - platformio ci examples/adk/adk_barcode --board=uno --board=teensy31 --
  
  - platformio ci examples/adk/ArduinoBlinkLED --board=uno --board=teensy31 --
  
  - platformio ci examples/adk/demokit_20 --board=uno --board=teensy31 --
  
  test:
  override:
  
  - platformio ci examples/acm/acm_terminal --board=uno --board=teensy31 --
  
  - platformio ci examples/adk/adk_barcode --board=uno --board=teensy31 --
  
  - platformio ci examples/adk/ArduinoBlinkLED --board=uno --board=teensy31 --
  
  - platformio ci examples/adk/demokit_20 --board=uno --board=teensy31 --
  
  dependencies:
  
  dependencies:
  
  dependencies:
  
  dependencies:
  
  # 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:
  
  - platformio ci examples/backSoon/backSoon.ino --lib="." --lib="/tmp/jeelib-
    master" --lib="/tmp/Gamebuino-master/libraries/tinyFAT" --board=uno --
    board=megaatmega2560
  
  - platformio ci examples/etherNode/etherNode.ino --lib="." --lib="/tmp/jeelib-
    master" --lib="/tmp/Gamebuino-master/libraries/tinyFAT" --board=uno --
    board=megaatmega2560
  
  - platformio ci examples/getDHCPandDNS/getDHCPandDNS.ino --lib="." --lib="/tmp/jeelib-
    master" --lib="/tmp/Gamebuino-master/libraries/tinyFAT" --board=uno --
    board=megaatmega2560
  
  - platformio ci examples/getStaticIP/getStaticIP.ino --lib="." --lib="/tmp/jeelib-
    master" --lib="/tmp/Gamebuino-master/libraries/tinyFAT" --board=uno --
    board=megaatmega2560
  
  dependencies:
  
  dependencies:
  
  dependencies:
  
  dependencies:
  
  # ...
• Configuration file: hhttps://github.com/ivankravets/ethercard/blob/master/circle.yaml
• Build History: https://circleci.com/gh/ivankravets/ethercard/tree/master

1.19.3 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 `platformio 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
  – Examples

Integration

Please fill all fields for your project in the Drone control panel:

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
platformio ci --board=<ID_1> --board=<ID_2> --board=<ID_N>
```
For more details as for PlatformIO build process please look into `platformio ci` command.

**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:**
   
   ```
   ```
pip install -U platformio
unzip /tmp/spi4teensy3.zip -d /tmp
platformio ci --lib="." --lib="/tmp/spi4teensy3-master" --board=uno --board=teensy31 --board=due

1.19.4 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 `platformio 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.

### Contents

- **GitLab**
  - Integration
  - Examples

### Integration

Please put `.gitlab-ci.yml` to the root directory of the repository.

```yaml
image: python:2.7

stages:
- test

before_script:
- "pip install -U platformio"

job:
  stage: test
  script: "platformio ci --board=<ID_1> --board=<ID_2> --board=<ID_N>"
  variables: {
    PLATFORMIO_CI_SRC: "path/to/test/file.c"
  }

For more details as for PlatformIO build process please look into `platformio ci` command.

### Examples

1. Integration for ArduinoJson library project. The `.gitlab-ci.yml` configuration file:

```yaml
image: python:2.7

stages:
- test

.job_template: &pio_run
  script:
  - "platformio ci --lib='.' --board=uno --board=teensy31 --board=nodemcuv2, \n    --$PLATFORMIO_CI_EXTRA_ARGS"

before_script:
- "pip install -U platformio"

JsonGeneratorExample:
  <<: *pio_run
```

(continues on next page)
variables:
    PLATFORMIO_CI_EXTRA_ARGS: "--board=due"
    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

1.19.5 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.
Integration

See step-by-step guide in ThingForward’s blog post Setting up a Jenkins CI engine for embedded projects.

1.19.6 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 `platformio 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.

```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:
  - platformio ci --board=<ID_1> --board=<ID_2> --board=<ID_N>
```

For more details as for PlatformIO build process please look into `platformio ci` command.

Examples

1. Integration for USB_Host_Shield_2.0 project. The `shippable.yml` or `.travis.yml` configuration file:
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 platformio 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
  - Project as a library
  - Library dependencies
    * Install dependent library using Library Manager
**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 PIO Unit Testing or PIO Remote you will need to define `PLATFORMIO_AUTH_TOKEN` environment variable in project settings. See Defining Variables in Repository Settings guide.

PlatformIO is written in Python and is recommended to be run within Travis CI Python isolated environment:

```yaml
language: python
python:
  - "2.7"

# Cache PlatformIO packages using Travis CI container-based infrastructure
sudo: false
cache:
  directories:
    - "~/.platformio"

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
  - platformio update

script:
  - platformio 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/

For more details as for PlatformIO build process please look into platformio ci.

### Project as a library

When project is written as a library (where own examples or testing code use it), please use `--lib="."` option for platformio ci command

```bash
script:
  - platformio ci --lib="." --board=<ID_1> --board=<ID_2> --board=<ID_N>
```
Library dependencies

There are 2 options to test source code with dependent libraries:

Install dependent library using Library Manager

```
install:
- pip install -U platformio

# Libraries from PlatformIO Library Registry:
# https://platformio.org/lib/show/1/OneWire
- platformio lib -g install 1
```

Manually download dependent library and include in build process via `--lib` option

```
install:
- pip install -U platformio
- wget https://github.com/PaulStoffregen/OneWire/archive/master.zip -O /tmp/onewire_source.zip
- unzip /tmp/onewire_source.zip -d /tmp/

script:
- platformio 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 `platformio ci --project-conf`
Unit Testing

See PlatformIO Remote Unit Testing Example.

Examples

1. Custom build flags

```
language: python
python:
  - "2.7"

# Cache PlatformIO packages using Travis CI container-based infrastructure
sudo: false
cache:
  directories:
    - "~/.platformio"
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
  - platformio update

# Libraries from PlatformIO Library Registry:
# https://platformio.org/lib/show/416/TinyGPS
# https://platformio.org/lib/show/417/SPI4Teensy3
  - platformio lib -g install 416 417
script:
  - platformio ci --board=uno --board=teensy31 --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

```
language: python
python:
  - "2.7"

# Cache PlatformIO packages using Travis CI container-based infrastructure
sudo: false
```

(continues on next page)
cache:
  directories:
    - "~/.platformio"

env:
- PLATFORMIO_CI_SRC=examples/backSoon/backSoon.ino
- PLATFORMIO_CI_SRC=examples/etherNode/etherNode.ino
  
install:
- pip install -U platformio
- platformio 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:
- platformio ci --lib="." --lib="/tmp/jeelib-master" --lib="/tmp/Gamebuino-master/
  →libraries/tinyFAT" --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:
  - "2.7"

# Cache PlatformIO packages using Travis CI container-based infrastructure
sudo: false
cache:
  directories:
    - "~/.platformio"

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
- platformio update
- rm -rf ./linux

# Libraries from PlatformIO Library Registry:
# 
# https://platformio.org/lib/show/416/TinyGPS

(continues on next page)
- platformio lib -g install 416 421 422

**script:**
- platformio ci --lib="." --board=uno --board=teensy20 $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

```python
language: python
python:
  - "2.7"

# Cache PlatformIO packages using Travis CI container-based infrastructure
sudo: false
cache:
  directories:
    - "~/../platformio"

env:
  - PLATFORMIO_CI_SRC=examples/Boards_Bluetooth/Adafruit_Bluefruit_LE
  - PLATFORMIO_CI_SRC=examples/Boards_Bluetooth/Arduino_101_BLE PLATFORMIO_CI_EXTRA_ARGS="--board=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=nodemcuv2 --project-option='lib_ignore=WiFi101'"
    
install:
- pip install -U platformio
- platformio 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


**script:**
- make travis-build

1.19. Continuous Integration
• Configuration file: https://github.com/blynkkk/blynk-library/blob/master/travis.yml
• Build History: https://travis-ci.org/blynkkk/blynk-library

1.20 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:

1.20.1 2019

• Jul 04, 2019 - Jean-Claude Wippler - The PlatformIO command line
• Mar 08, 2019 - Nathan Glover - Amazon Alexa controlled IoT Traffic Lights

1.20.2 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
• 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

1.20.3 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-contôleurs (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

1.20.4 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
• Sep 19, 2016 - Doc Walker - How to automatically test build Arduino libraries
• Sep 18, 2016 - Kadda Sahnine - LoRaWAN network practice with Objenious, PlatformIO and Node-RED
• Sep 13, 2016 - Xose Pérez MQTT LED Matrix Display
• Sep 12, 2016 - Pedro Minatel - OTA – Como programar o ESP8266 pelo WiFi no platformIO (OTA programming for ESP8266 via Wi-Fi using PlatformIO, Portuguese)
• Sep 2, 2016 - Xose Pérez ESP8266: Optimizing files for SPIFFS with Gulp
• Aug 28, 2016 - Tom Parker Using the BBC micro:bit with PlatformIO
• Aug 24, 2016 - Primal Cortex Cloud based continuous integration and delivery for IOT using PlatformIO
• Aug 18, 2016 - Primal Cortex - Installing PlatformIO on Arch Linux
• Aug 14, 2016 - Rodrigo Castro - PlatformIO o comó usar Arduino con ATOM (Spanish)
• Jul 27, 2016 - Francesco Azzola - Arduino Alternative IDE: PlatformIO IoT integrated platform
• Jul 26, 2016 - Embedded Systems Laboratory - PlatformIO IDE Arduino ESP8266 (Get started with PlatformIO IDE for Arduino board and ESP8266, Thai)
• Jul 15, 2016 - Jaime - ESP8266 Mobile Rick Roll Captive Portal
• Jul 5, 2016 - Ivan Kravets, Ph.D. - Explore the new development instruments for Arduino with PlatformIO ecosystem
• Jul 5, 2016 - Belinda - Monte Bianco Arduino Developer Summit
• Jul 1, 2016 - Tam Hanna - Microcontroller-Gipfel in den Alpen: Arduino Developer Summit, Tag eins (Microcontroller peaks in the Alps: Arduino Developer Summit, Day One, German)
• Jun 14, 2016 - Glyn Hudson - OpenEnergyMonitor Part 2/3: Firmware Continuous Test & Build
• Jun 13, 2016 - Daniel Eichhorn - New Weather Station Demo on Github
• Jun 12, 2016 - Glyn Hudson - OpenEnergyMonitor Part 1/3: PlatformIO open-source embedded development ecosystem
• Jun 12, 2016 - Uli Wolf - Nutzung von PlatformIO im Atom Editor zur Entwicklung von Arduino Code (Use PlatformIO and Atom Editor to develop Arduino code, German)
• Jun 3, 2016 - Daniel Eichhorn - ESP8266: Continuous Delivery Pipeline – Push To Production
• May 30, 2016 - Ron Moerman - IoT Development with PlatformIO
• May 29, 2016 - Chris Synan - Reverse Engineer RF Remote Controller for IoT!
• May 26, 2016 - Charlie Key - 7 Best Developer Tools To Build Your NEXT Internet of Things Application
• May 22, 2016 - Pedro Minatel - Estação meteorológica com ESP8266 (Weather station with ESP8266, Portuguese)
• May 16, 2016 - Pedro Minatel - Controle remoto WiFi com ESP8266 (WiFi remote control using ESP8266, Portuguese)
• May 11, 2016 - Jo Vandeginste - Using PlatformIO to compile for Jeelabs’ Jeenode Micro
• May 08, 2016 - Radoslaw Bob - Touch controlled buzzer (Nodemcu ESP8266)
• May 06, 2016 - Jean Roux - The IoT building blocks I use for my home-automation projects
• May 05, 2016 - Ivan Kravets, Ph.D. / Eclipse Virtual IoT Meetup - PlatformIO: a cross-platform IoT solution to build them all!
• 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 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 - Nieuwe wereld met 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 Hol dys - 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

1.20.5 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 - Michał 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 - nodc5 - PlatformIOMbedSTM32 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

1.20. Articles about us
• 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 línea 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

### 1.20.6 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

### 1.21 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*
1.21.1 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

Bash completion

Bash completion support will complete subcommands and parameters. To enable Bash completion for platformio subcommands you need to put into your .bashrc:

```bash
eval "\$_PLATFORMIO_COMPLETE=source platformio"
eval "\$_PIO_COMPLETE=source pio"
```

ZSH completion

To enable zsh completion please run these commands:

```bash
autoload bashcompinit && bashcompinit
eval "\$_PLATFORMIO_COMPLETE=source platformio"
eval "\$_PIO_COMPLETE=source pio"
```

Note: For permanent command completion you need to place commands above to ~/.bashrc or ~/.zshrc file.

1.21.2 Install Python Interpreter

PlatformIO Core (CLI) is written in Python that is installed by default on the all popular OS except Windows.

Windows Users, please Download the latest Python and install it. DON’T FORGET to select Add python.exe to Path feature on the “Customize” stage, otherwise, python command will not be available.
1.21.3 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:

```c
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:

```c
#include <Arduino.h>

void someFunction(int num);

void setup () {
    someFunction(13);
}

void loop() {
    delay(1000);
}

void someFunction(int num) {
}
```

Finish.

### 1.21.4 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,

```c
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 `platformio run --verbose` command.

Recommended for reading:

- [text, data and bss: Code and Data Size Explained](https://en.wikipedia.org/wiki/Data_segment)
1.21.5 Troubleshooting

Installation

Multiple PIO 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 PIO Core or use built-in PIO Core in PlatformIO IDE:

- **PlatformIO IDE for Atom** - Menu PlatformIO: Settings > PlatformIO IDE > Use built-in PlatformIO Core
- **PlatformIO IDE for VSCode** - Proxy Server Support > 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 PIO Core:

```bash
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 PIO 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

Linux users have to install udev rules for PlatformIO supported boards/devices. The latest version of rules may be found at https://raw.githubusercontent.com/platformio/platformio-core/master/scripts/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 `platformio 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

```bash
# Recommended

# OR, manually download and copy this file to destination folder
```
Restart “udev” management tool:

```
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

```
sudo usermod -a -G dialout $USER
sudo usermod -a -G plugdev $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 2.7 interpreter, see [Install Python Interpreter](#) 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**
- **Solution 2: Antivirus**
- **Solution 3: Run from Terminal**
- **Solution 4: Manual**

**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 PIO Home dependencies
platformio home

# Force PlatformIO to install toolchains
platformio run --target upload
```

If “platformio” 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
- 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.

1.22 Release Notes

1.22.1 PlatformIO 4.0

4.0.1 (2019-??-??)

- Print debug tool name for the active debugging session
- Do not shutdown PIO Home Server for “upgrade” operations (issue #2784)
- Improved computing of project check sum (structure, configuration) and avoid unnecessary rebuilding
- Renamed “enable_ssl” setting to `strict_ssl`
- Fixed an issue with incorrect escaping of Windows slashes when using PIO Unified Debugger and “piped” openOCD
- Fixed an issue when “debug”, “home”, “run”, and “test” commands were not shown in “platformio –help” CLI
4.0.0 (2019-07-10)

Migration Guide from 3.0 to 4.0.

- **PlatformIO Plus Goes Open Source**
  - Built-in PIO Unified Debugger
  - Built-in PIO Unit Testing

- **Project Configuration**
  - New project configuration parser with a strict options typing (API)
  - Unified workspace storage (workspace_dir -> .pio) for PlatformIO Build System, Library Manager, and other internal services (issue #1778)
  - Share common (global) options between project environments using [env] section (issue #1643)
  - Include external configuration files with extra_configs option (issue #1590)
  - Custom project ***_dir options declared in platformio section have higher priority than Environment variables
  - Added support for Unix shell-style wildcards for monitor_port option (issue #2541)
  - Added new monitor_flags option which allows passing extra flags and options to platformio device monitor command (issue #2165)
  - Added support for PLATFORMIO_DEFAULT_ENVS system environment variable (issue #1967)
  - Added support for shared_dir where you can place an extra files (extra scripts, LD scripts, etc.) which should be transferred to a PIO Remote machine

- **Library Management**
  - Switched to workspace .pio/libdeps folder for project dependencies instead of .piolibdeps
  - Save libraries passed to platformio lib install command into the project dependency list (lib_deps) with a new --save flag (issue #1028)
  - Install all project dependencies declared via lib_deps option using a simple platformio lib install command (issue #2147)
  - Use isolated library dependency storage per project build environment (issue #1696)
  - Look firstly in built-in library storages for a missing dependency instead of PlatformIO Registry (issue #1654)
  - Override default source and include directories for a library via library.json manifest using includeDir and srcDir fields
  - Fixed an issue when library keeps reinstalling for non-latin path (issue #1252)
  - Fixed an issue when lib_compat_mode = strict does not ignore libraries incompatible with a project framework

- **Build System**
  - Switched to workspace .pio/build folder for build artifacts instead of .pioenvs
  - Switch between Build Configurations (release and debug) with a new project configuration option build_type
  - Custom platform_packages per a build environment with an option to override default (issue #1367)
  - Print platform package details, such as version, VSC source and commit (issue #2155)
– Control a number of parallel build jobs with a new `-j, --jobs` option
– Override default “platformio.ini” (Project Configuration File) with a custom using `-c, --project-conf` option for `platformio run, platformio debug, or platformio test` commands (issue #1913)
– Override default development platform upload command with a custom `upload_command` (issue #2599)
– Configure a shared folder for the derived files (objects, firmwares, ELFs) from a build system using `build_cache_dir` option (issue #2674)
– Fixed an issue when `-U` in `build_flags` does not remove macro previously defined via `-D` flag (issue #2508)

• **Infrastructure**
  – Python 3 support (issue #895)
  – Significantly speedup back-end for PIO Home. It works super fast now!
  – Added support for the latest Python “Click” package (CLI) (issue #349)
  – Added options to override default locations used by PlatformIO Core (`core_dir, globallib_dir, platforms_dir, packages_dir, cache_dir`) (issue #1615)
  – Removed line-buffering from `platformio run` command which was leading to omitting progress bar from upload tools (issue #856)
  – Fixed numerous issues related to “UnicodeDecodeError” and international locales, or when project path contains non-ASCII chars (issue #143, issue #1342, issue #1959, issue #2100)

• **Integration**
  – Support custom CMake configuration for CLion IDE using `CMakeListsUser.txt` file
  – Fixed an issue with hardcoded C standard version when generating project for CLion IDE (issue #2527)
  – Fixed an issue with Project Generator when an include path search order is inconsistent to what passed to the compiler (issue #2509)
  – Fixed an issue when generating invalid “Eclipse CDT Cross GCC Built-in Compiler Settings” if a custom `PLATFORMIO_CORE_DIR` is used (issue #806)

• **Miscellaneous**
  – Deprecated `--only-check` PlatformIO Core CLI option for “update” sub-commands, please use `--dry-run` instead
  – Fixed “systemd-udevd” warnings in `99-platformio-udev.rules` (issue #2442)
  – Fixed an issue when package cache (Library Manager) expires too fast (issue #2559)

### 1.22.2 PlatformIO 3.0

#### 3.6.7 (2019-04-23)

- **PIO Unified Debugger**: improved debugging in `debug_load_mode = modified` and fixed an issue with useless project rebuilding
- **Project Generator**: fixed a VSCode C/C++’s “Cannot find” warning when CPPPATH folder does not exist
- **Fixed an “IndexError: list index out of range” for Arduino sketch preprocessor (issue #2268)**
• Fixed an issue when invalid “env_default” in “platformio.ini” (Project Configuration File) results into unhandled errors (issue #2265)

3.6.6 (2019-03-29)

• Project Generator: fixed a warning “Property !!! WARNING !!! is not allowed” for VSCode (issue #2243)
• Fixed an issue when PlatformIO Build System does not pick up “mbed_lib.json” files from libraries (issue #2164)
• Fixed an error with conflicting declaration of a prototype (Arduino sketch preprocessor)
• Fixed “FileExistsError” when platformio ci command is used in pair with --keep-build-dir option
• Fixed an issue with incorrect order of project “include” and “src” paths in CPPPATH (issue #1914)

3.6.5 (2019-03-07)

• Project Generator: added new targets for CLion IDE “BUILD_VERBOSE” and “MONITOR” (serial port monitor) (issue #359)
• Fixed an issue with slow updating of PlatformIO Core packages on Windows
• Fixed an issue when platformio ci recompiles project if --keep-build-dir option is passed (issue #2109)
• Fixed an issue when $PROJECT_HASH template was not expanded for the other directory ***_dir options in “platformio.ini” (Project Configuration File) (issue #2170)

3.6.4 (2019-01-23)

• Improved Project Generator for IDEs:
  – Use full path to PlatformIO CLI when generating a project (issue #1674)
  – CLion: Improved project portability using “$(CMAKE_CURRENT_LIST_DIR)” instead of full path
  – Eclipse: Provide language standard to a project C/C++ indexer (issue #1010)
• Fixed an issue with incorrect detecting of compatibility (LDF) between generic library and Arduino or ARM mbed frameworks
• Fixed “Runtime Error: Dictionary size changed during iteration” (issue #2003)
• Fixed an error “Could not extract item...” when extracting TAR archive with symbolic items on Windows platform (issue #2015)

3.6.3 (2018-12-12)

• Ignore *.asm and *.ASM files when building Arduino-based library (compatibility with Arduino builder)
• Fixed spurious project’s “Problems” for PlatformIO IDE for VSCode when ARM mbed framework is used
• Fixed an issue with a broken headers list when generating “.clang_complete” for Emacs (issue #1960)
3.6.2 (2018-11-29)

- Improved IntelliSense for PlatformIO IDE for VSCode via passing extra compiler information for C/C++ Code Parser (resolves issues with spurious project’s “Problems”)
- Fixed an issue with VSCode IntelliSense warning about the missed headers located in include folder
- Fixed incorrect wording when initializing/updating project
- Fixed an issue with incorrect order for library dependencies CPPPATH (issue #1914)
- Fixed an issue when Library Dependency Finder (LDF) does not handle project src_filter (issue #1905)
- Fixed an issue when Library Dependency Finder (LDF) finds spurious dependencies in chain+ and deep+ modes (issue #1930)

3.6.1 (2018-10-29)

- Generate an include and test directories with a README file when initializing a new project
- Support in-line comments for multi-line value (lib_deps, build_flags, etc) in “platformio.ini” (Project Configuration File)
- Added $PROJECT_HASH template variable for build_dir. One of the use cases is setting a global storage for project artifacts using PLATFORMIO_BUILD_DIR system environment variable. For example, /tmp/pio-build/$PROJECT_HASH (Unix) or $[sysenv.TEMP]/pio-build/$PROJECT_HASH (Windows)
- Improved a loading speed of PIO Home “Recent News”
- Improved PIO Unified Debugger for “mbed” framework and fixed issue with missed local variables
- Introduced “Release” and “Debug” Build Configurations
- Build project in “Debug Mode” including debugging information with a new debug target using platformio run command or targets option in platformio.ini. The last option allows avoiding project rebuilding between “Run/Debug” modes. (issue #1833)
- Process build_unflags for the cloned environment when building a static library
- Report on outdated 99-platformio-udev.rules (issue #1823)
- Show a valid error when the Internet is off-line while initializing a new project (issue #1784)
- Do not re-create “.gitignore” and “.travis.yml” files if they were removed from a project
- Fixed an issue when dynamic build flags were not handled correctly (issue #1799)
- Fixed an issue when pio run -t monitor always uses the first monitor_port even with multiple environments (issue #1841)
- Fixed an issue with broken includes when generating .clang_complete and space is used in a path (issue #1873)
- Fixed an issue with incorrect handling of a custom package name when using platformio lib install or platformio platform install commands

3.6.0 (2018-08-06)

- Program Memory Usage
  - Print human-readable memory usage information after a build and before uploading
– Print detailed memory usage information with “sections” and “addresses” in verbose mode
– Check maximum allowed “program” and “data” sizes before uploading/programming (issue #1412)

• PIO Unit Testing:
  – Documented Project Shared Code
  – Force building of project source code using test_build_project_src option
  – Fixed missed UNIT_TEST macro for unit test components/libraries

• Check package structure after unpacking and raise error when antivirus tool blocks PlatformIO package manager (issue #1462)
• Lock interprocess requests to PlatformIO Package Manager for install/uninstall operations (issue #1594)
• Fixed an issue with PIO Remote when upload process depends on the source code of a project framework
• Fixed an issue when srcFilter field in library.json breaks a library build (issue #1735)

3.5.4 (2018-07-03)

• Improved removing of default build flags using build_unflags option (issue #1712)
• Export LIBS, LIBPATH, and LINKFLAGS data from project dependent libraries to the global build environment
• Don’t export CPPPATH data of project dependent libraries to framework’s build environment (issue #1665)
• Handle “architectures” data from “library.properties” manifest in lib_compat_mode = strict
• Added workaround for Python SemVer package’s issue #61 with caret range and pre-releases
• Replaced conflicted “env” pattern by “sysenv” for “platformio.ini” Dynamic Variables” (issue #1705)
• Removed “date&time” when processing project with platformio run command (issue #1343)
• Fixed issue with invalid LD script if path contains space
• Fixed preprocessor for Arduino sketch when function returns certain type (issue #1683)
• Fixed issue when platformio lib uninstall removes initial source code (issue #1023)

3.5.3 (2018-06-01)

• PlatformIO Home - interact with PlatformIO ecosystem using modern and cross-platform GUI:
  – “Recent News” block on “Welcome” page
  – Direct import of development platform’s example
• Simplify configuration for PIO Unit Testing: separate main program from a test build process, drop requirement for #ifdef UNIT_TEST guard
• Override any option from board manifest in “platformio.ini” (Project Configuration File) (issue #1612)
• Configure a custom path to SVD file using debug_svd_path option
• Custom project description which will be used by PlatformIO Home
• Updated Unity tool to 2.4.3
• Improved support for Black Magic Probe in “uploader” mode
• Renamed “monitor_baud” option to “monitor_speed”
• Fixed issue when a custom lib_dir was not handled correctly (issue #1473)
• Fixed issue with useless project rebuilding for case insensitive file systems (Windows)
• Fixed issue with build_unflags option when a macro contains value (e.g., -DNAME=VALUE)
• Fixed issue which did not allow to override runtime build environment using extra POST script
• Fixed “RuntimeError: maximum recursion depth exceeded” for library manager (issue #1528)

3.5.2 (2018-03-13)

• PlatformIO Home - interact with PlatformIO ecosystem using modern and cross-platform GUI:
  – Multiple themes (Dark & Light)
  – Ability to specify a name for new project
• Control PIO Unified Debugger and its firmware loading mode using debug_load_mode option
• Added aliases (off, light, strict) for LDF Compatibility Mode
• Search for a library using PIO Library Registry ID id:X (e.g. pio lib search id:13)
• Show device system information (MCU, Frequency, RAM, Flash, Debugging tools) in a build log
• Show all available upload protocols before firmware uploading in a build log
• Handle “os.mbed.com” URL as a Mercurial (hg) repository
• Improved support for old mbed libraries without manifest
• Fixed project generator for Qt Creator IDE (issue #1303, issue #1323)
• Mark project source and library directories for CLion IDE (issue #1359, issue #1345, issue #897)
• Fixed issue with duplicated “include” records when generating data for IDE (issue #1301)

3.5.1 (2018-01-18)

• New test_speed option to control a communication baudrate/speed between PIO Unit Testing engine and a target device (issue #1273)
• Show full library version in “Library Dependency Graph” including VCS information (issue #1274)
• Configure a custom firmware/program name in build directory (example)
• Renamed envs_dir option to build_dir in “platformio.ini” (Project Configuration File)
• Refactored code without “arrow” dependency (resolve issue with “ImportError: No module named backports.functools_lru_cache”)
• Improved support of PIO Unified Debugger for Eclipse Oxygen
• Improved a work in off-line mode
• Fixed project generator for CLion and Qt Creator IDE (issue #1299)
• Fixed PIO Unified Debugger for mbed framework
• Fixed library updates when a version is declared in VCS format (not SemVer)
3.5.0 (2017-12-28)

- **PlatformIO Home** - interact with PlatformIO ecosystem using modern and cross-platform GUI:
  - Library Manager:
    * 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.
  - PlatformIO Projects
  - PIO Account
  - Development platforms, frameworks and board explorer
  - Device Manager: serial, logical, and multicast DNS services

- **Integration with Jenkins CI**

- **New include folder for project’s header files (issue #1107)**

- **Depend on development platform using VCS URL (Git, Mercurial and Subversion) instead of a name in “platformio.ini” (Project Configuration File). Drop support for *_stage dev/platform names (use VCS URL instead).**

- **Reinstall/redownload package with a new --force option for platformio lib install and platformio platform install commands (issue #778)**

- **Handle missed dependencies and provide a solution based on PlatformIO Library Registry (issue #781)**

- **New setting projects_dir that allows to override a default PIO Home Projects location (issue #1161)**

- **Library Dependency Finder (LDF):**
  - Search for dependencies used in PIO Unit Testing (issue #953)
  - Parse library source file in pair with a header when they have the same name (issue #1175)
  - Handle library dependencies defined as VCS or SemVer in “platformio.ini” (Project Configuration File) (issue #1155)
  - Added option to configure library Compatible Mode using library.json

- **New options for platformio device list command:**
  - --serial list available serial ports (default)
  - --logical list logical devices
  - --mdns discover multicast DNS services (issue #463)

- **Fixed platforms, packages, and libraries updating behind proxy (issue #1061)**

- **Fixed missing toolchain include paths for project generator (issue #1154)**

- **Fixed “Super-Quick (Mac / Linux)” installation in “get-platformio.py” script (issue #1017)**

- **Fixed “get-platformio.py” script which hangs on Windows 10 (issue #1118)**

- **Other bug fixes and performance improvements**
### 3.4.1 (2017-08-02)

- Pre/Post extra scripting for advanced control of PIO Build System (issue #891)
- New `lib_archive` option to control library archiving and linking behavior (issue #993)
- Add “inc” folder automatically to CPPPATH when “src” is available (works for project and library) (issue #1003)
- Use a root of library when filtering source code using `library.json` and `srcFilter` field
- Added `monitor_*` options to white-list for “platformio.ini” (Project Configuration File) (issue #982)
- Do not ask for board ID when initialize project for desktop platform
- Handle broken PIO Core state and create new one
- Fixed an issue with a custom transport for PIO Unit Testing when multiple tests are present
- Fixed an issue when can not upload firmware to SAM-BA based board (Due)

### 3.4.0 (2017-06-26)

- **PIO Unified Debugger**
  - “1-click” solution, zero configuration
  - Support for 100+ embedded boards
  - Multiple architectures and development platforms
  - Windows, MacOS, Linux (+ARMv6-8)
  - Built-in into PlatformIO IDE for Atom and PlatformIO IDE for VScode
  - Integration with Eclipse and Sublime Text
- Filter PIO Unit Testing tests using a new `test_filter` option in “platformio.ini” (Project Configuration File) or `platformio test –filter` command (issue #934)
- Custom `test_transport` for PIO Unit Testing Engine
- Configure Serial Port Monitor in “platformio.ini” (Project Configuration File) (issue #787)
- New `monitor` target which allows to launch Serial Monitor automatically after successful “build” or “upload” operations (issue #788)
- Project generator for VIM
- Multi-line support for the different options in “platformio.ini” (Project Configuration File), such as: `build_flags`, `build_unflags`, etc. (issue #889)
- Handle dynamic `SRC_FILTER` environment variable from `library.json` extra script
- Notify about multiple installations of PIO Core (issue #961)
- Improved auto-detecting of mbed-enabled media disks
- Automatically update Git-submodules for development platforms and libraries that were installed from repository
- Add support for `.cc` extension (issue #939)
- Handle `env_default` in “platformio.ini” (Project Configuration File) when re-initializing a project (issue #950)
- Use root directory for PIO Home when path contains non-ascii characters (issue #951, issue #952)
• Don’t warn about known `boards_dir` option (pull #949)
• Escape non-valid file name characters when installing a new package (library) (issue #985)
• Fixed infinite dependency installing when repository consists of multiple libraries (issue #935)
• Fixed linter error “unity.h does not exist” for Unit Testing (issue #947)
• Fixed issue when Library Dependency Finder (LDF) does not handle custom `src_dir` (issue #942)
• Fixed cloning a package (library) from a private Git repository with custom user name and SSH port (issue #925)

3.3.1 (2017-05-27)

• Hotfix for recently updated Python Requests package (2.16.0)

3.3.0 (2017-03-27)

• PlatformIO Library Registry statistics with new `pio lib stats` command
  – Recently updated and added libraries
  – Recent and popular keywords
  – Featured libraries (today, week, month)
• List built-in libraries based on development platforms with a new `pio lib builtin` command
• Show detailed info about a library using `pio lib show` command (issue #430)
• List supported frameworks, SDKs with a new `pio platform frameworks` command
• Visual Studio Code extension for PlatformIO (issue #619)
• Added new options `--no-reset`, `--monitor-rts` and `--monitor-dtr` to `pio test` command (allows to avoid automatic board’s auto-reset when gathering test results)
• Added support for templated methods in `*.ino` to `*.cpp` converter (pull #858)
• Package version as “Repository URL” in manifest of development version ("version": "https://github.com/user/repo.git")
• Produce less noisy output when `-s/--silent` options are used for `platformio init` and `platformio run` commands (issue #850)
• Use C++11 by default for CLion IDE based projects (pull #873)
• Escape project path when Glob matching is used
• Do not overwrite project configuration variables when system environment variables are set
• Handle dependencies when installing non-registry package/library (VCS, archive, local folder) (issue #913)
• Fixed package installing with VCS branch for Python 2.7.3 (issue #885)

3.2.1 (2016-12-07)

• Changed default LDF Mode from `chain+` to `chain`
3.2.0 (2016-12-07)

• PIO Remote™. Your devices are always with you!
  – Over-The-Air (OTA) Device Manager
  – OTA Serial Port Monitor
  – OTA Firmware Updates
  – Continuous Deployment
  – Continuous Delivery

• Integration with Cloud IDEs
  – Cloud9
  – Codeanywhere
  – Eclipse Che

• PIO Account and PLATFORMIO_AUTH_TOKEN environment variable for CI systems (issue #808, issue #467)

• Inject system environment variables to configuration settings in “platformio.ini” (Project Configuration File) (issue #792)

• Custom boards per project with boards_dir option in “platformio.ini” (Project Configuration File) (issue #515)

• Unix shell-style wildcards for upload_port (issue #839)

• Refactored Library Dependency Finder (LDF) C/C++ Preprocessor for conditional syntax (#ifdef, #if, #else, #elif, #define, etc.) (issue #837)

• Added new LDF Modes: chain+ and deep+ and set chain+ as default

• Added global lib_extra_dirs option to [platformio] section for “platformio.ini” (Project Configuration File) (issue #842)

• Enabled caching by default for API requests and Library Manager (see enable_cache setting)

• Native integration with VIM/Neovim using neomake-platformio plugin

• Changed a default exit combination for Device Monitor from Ctrl+] to Ctrl+C

• Improved detecting of ARM mbed media disk for uploading

• Improved Project Generator for CLion IDE when source folder contains nested items

• Improved handling of library dependencies specified in library.json manifest (issue #814)

• Improved Library Dependency Finder (LDF) for circular dependencies

• Show vendor version of a package for platformio platform show command (issue #838)

• Fixed unable to include SSH user in lib_deps repository url (issue #830)

• Fixed merging of “.gitignore” files when re-initialize project (issue #848)

• Fixed issue with PATH auto-configuring for upload tools

• Fixed 99-platformio-udev.rules checker for Linux OS
3.1.0 (2016-09-19)

• New! Dynamic variables/templates for “platformio.ini” (Project Configuration File) (issue #705)
• Summary about processed environments (issue #777)
• Implemented LocalCache system for API and improved a work in off-line mode
• Improved Project Generator when custom --project-option is passed to platformio init command
• Deprecated lib_force option, please use lib_deps instead
• Return valid exit code from platformio test command
• Fixed Project Generator for CLion IDE using Windows OS (issue #785)
• Fixed SSL Server-Name-Indication for Python < 2.7.9 (issue #774)

3.0.1 (2016-09-08)

• Disabled temporary SSL for PlatformIO services (issue #772)

3.0.0 (2016-09-07)

• PlatformIO Plus
  – Local and Embedded Unit Testing (issue #408, issue #519)
• Decentralized Development Platforms
  – Development platform manifest “platform.json” and open source development platforms
  – Semantic Versioning for platform commands, development platforms and dependent packages
  – Custom package repositories
  – External embedded board configuration files, isolated build scripts (issue #479)
  – Embedded Board compatibility with more than one development platform (issue #456)
• Library Manager 3.0
  – Project dependencies per build environment using lib_deps option (issue #413)
  – Semantic Versioning for library commands and dependencies (issue #410)
  – Multiple library storages: Project’s Local, PlatformIO’s Global or Custom (issue #475)
  – Install library by name (issue #414)
  – Depend on a library using VCS URL (GitHub, Git, ARM mbed code registry, Hg, SVN) (issue #498)
  – Strict search for library dependencies (issue #588)
  – Allowed library.json to specify sources other than PlatformIO’s Repository (issue #461)
  – Search libraries by headers/includes with platformio lib search --header option
• New Intelligent Library Build System
  – Library Dependency Finder that interprets C/C++ Preprocessor conditional macros with deep search behavior
  – Check library compatibility with project environment before building (issue #415)
  – Control Library Dependency Finder for compatibility using lib_compat_mode option
– Custom library storages/directories with `lib_extra_dirs` option (issue #537)
– Handle extra build flags, source filters and build script from `library.json` (issue #289)
– Allowed to disable library archiving (*.ar) (issue #719)
– Show detailed build information about dependent libraries (issue #617)
– Support for the 3rd party manifests (Arduino IDE “library.properties” and ARM mbed “module.json”)

• Removed `enable_prompts` setting. Now, all PlatformIO CLI is non-blocking!

• Switched to SSL PlatformIO API

• Renamed `platformio serialports` command to `platformio device`

• Build System: Attach custom Before/Pre and After/Post actions for targets (issue #542)

• Allowed passing custom project configuration options to `platformio ci` and `platformio init` commands using `-O`, `--project-option`.

• Print human-readable information when processing environments without `-v`, `--verbose` option (issue #721)

• Improved INO to CPP converter (issue #659, issue #765)

• Added `license` field to `library.json` (issue #522)

• Warn about unknown options in project configuration file `platformio.ini` (issue #740)

• Fixed wrong line number for INO file when `#warning` directive is used (issue #742)

• Stopped supporting Python 2.6

1.22.3 PlatformIO 2.0

2.11.2 (2016-08-02)

• Improved support for Microchip PIC32 development platform and ChipKIT boards (issue #438)

• Added support for Pinoccio Scout board (issue #52)

• Added support for Teensy USB Features (HID, SERIAL_HID, DISK, DISK_SDFLASH, MIDI, etc.) (issue #722)

• Switched to built-in GCC LwIP library for Espressif development platform

• Added support for local `--echo` for Serial Port Monitor (issue #733)

• Updated `udev` rules for the new STM32F407DISCOVERY boards (issue #731)

• Implemented firmware merging with base firmware for Nordic nRF51 development platform (issue #500, issue #533)

• Fixed Project Generator for ESP8266 and ARM mbed based projects (resolves incorrect linter errors)

• Fixed broken LD Script for Element14 chipKIT Pi board (issue #725)

• Fixed firmware uploading to Atmel SAMD21-XPRO board using ARM mbed framework (issue #732)
2.11.1 (2016-07-12)

- Added support for Arduino M0, M0 Pro and Tian boards (issue #472)
- Added support for Microchip chipKIT Lenny board
- Updated Microchip PIC32 Arduino framework to v1.2.1
- Documented uploading of EEPROM data (from EEMEM directive)
- Added Rebuild C/C++ Project Index target to CLion and Eclipse IDEs
- Improved project generator for CLion IDE
- Added udev rules for OpenOCD CMSIS-DAP adapters (issue #718)
- Auto-remove project cache when PlatformIO is upgraded
- Keep user changes for .gitignore file when re-generate/update project data
- Ignore [platformio] section from custom project configuration file when platformio ci –project-conf command is used
- Fixed missed --boot flag for the firmware uploader for ATSAM3X8E Cortex-M3 MCU based boards (Arduino Due, etc) (issue #710)
- Fixed missing trailing \ for the source files list when generate project for Qt Creator IDE (issue #711)
- Split source files to HEADERS and SOURCES when generate project for Qt Creator IDE (issue #713)

2.11.0 (2016-06-28)

- New ESP8266-based boards: Generic ESP8285 Module, Phoenix 1.0 & 2.0, WifInfo
- Added support for Arduino M0 Pro board (issue #472)
- Added support for Arduino MKR1000 board (issue #620)
- Added support for Adafruit Feather M0, SparkFun SAMD21 and SparkFun SAMD21 Mini Breakout boards (issue #520)
- Updated Arduino ESP8266 core for Espressif platform to 2.3.0
- Better removing unnecessary flags using build_unflags option (issue #698)
- Fixed issue with platformio init --ide command for Python 2.6

2.10.3 (2016-06-15)

- Fixed issue with platformio init --ide command

2.10.2 (2016-06-15)

- Added support for ST Nucleo L031K6 board to ARM mbed framework
- Process build_unflags option for ARM mbed framework
- Updated Intel ARC32 Arduino framework to v1.0.6 (issue #695)
- Improved a check of program size before uploading to the board
- Fixed issue with ARM mbed framework -u _printf_float and -u _scanf_float when parsing $LINKFLAGS
• Fixed issue with ARM mbed framework and extra includes for the custom boards, such as Seeeduino Arch Pro

2.10.1 (2016-06-13)

• Re-submit a package to PyPI

2.10.0 (2016-06-13)

• Added support for emonPi, the OpenEnergyMonitor system (issue #687)
• Added support for SPL framework for STM32F0 boards (issue #683)
• Added support for Arduboy DevKit, the game system the size of a credit card
• Updated ARM mbed framework package to v121
• Check program size before uploading to the board (issue #689)
• Improved firmware uploading to Arduino Leonardo based boards (issue #691)
• Fixed issue with -L relative/path when parsing build_flags (issue #688)

2.9.4 (2016-06-04)

• Show udev warning only for the Linux OS while uploading firmware

2.9.3 (2016-06-03)

• Added support for Arduboy, the game system the size of a credit card
• Updated 99-platformio-udev.rules for Linux OS
• Refactored firmware uploading to the embedded boards with SAM-BA bootloader

2.9.2 (2016-06-02)

• Simplified Continuous Integration with AppVeyor (issue #671)
• Automatically add source directory to CPPPATH of Build System
• Added support for Silicon Labs SLSTK3401A (Pearl Gecko) and MultiTech mDot F411 ARM mbed based boards
• Added support for MightyCore ATmega8535 board (issue #585)
• Added stlink as the default uploader for STM32 Discovery boards (issue #665)
• Use HTTP mirror for Package Manager in a case with SSL errors (issue #645)
• Improved firmware uploading to Arduino Leonardo/Due based boards
• Fixed bug with env_default when pio run -e is used
• Fixed issue with src_filter option for Windows OS (issue #652)
• Fixed configuration data for TI LaunchPads based on msp430fr4133 and msp430fr6989 MCUs (issue #676)
• Fixed issue with ARM mbed framework and multiple definition errors on FRDM-KL46Z board (issue #641)
• Fixed issue with ARM mbed framework when abstract class breaks compile for LPC1768 (issue #666)
2.9.1 (2016-04-30)

- Handle prototype pointers while converting *.ino to .cpp (issue #639)

2.9.0 (2016-04-28)

- Project generator for CodeBlocks IDE (issue #600)
- New Lattice iCE40 FPGA development platform with support for Lattice iCEstick FPGA Evaluation Kit and BQ IceZUM Alhambra FPGA (issue #480)
- New Intel ARC 32-bit development platform with support for Arduino/Genuino 101 board (issue #535)
- New Microchip PIC32 development platform with support for 20+ different PIC32 based boards (issue #438)
- New RTOS and build Framework named Simba (issue #412)
- New boards for ARM mbed framework: ST Nucleo F410RB, ST Nucleo L073RZ and BBC micro:bit
- Added support for Generic ATTiny boards: ATTiny13, ATTiny24, ATTiny25, ATTiny45 and ATTiny85 (issue #636)
- Added support for MightyCore boards: ATmega1284, ATmega644, ATmega324, ATmega164, ATmega32, ATmega16 and ATmega8535 (issue #585)
- Added support for TI MSP430 boards: TILaunchPad w/ msp430fr4133 and TI LaunchPad w/ msp430fr6989
- Updated Arduino core for Espressif platform to 2.2.0 (issue #627)
- Updated native SDK for ESP8266 to 1.5 (issue #366)
- PlatformIO Library Registry in JSON format! Implemented --json-output and --page options for platformio lib search command (issue #604)
- Allowed to specify default environments env_default which should be processed by default with platformio run command (issue #576)
- Allowed to unflag(remove) base/initial flags using build_unflags option (issue #559)
- Allowed multiple VID/PID pairs when detecting serial ports (issue #632)
- Automatically add -DUSB_MANUFACTURER with vendor’s name (issue #631)
- Automatically reboot Teensy board after upload when Teensy Loader GUI is used (issue #609)
- Refactored source code converter from *.ino to *.cpp (issue #610)
- Forced -std=gnu++11 for Atmel SAM development platform (issue #601)
- Don’t check OS type for ARM mbed-enabled boards and ST STM32 development platform before uploading to disk (issue #596)
- Fixed broken compilation for Atmel SAMD based boards except Arduino Due (issue #598)
- Fixed firmware uploading using serial port with spaces in the path
- Fixed cache system when project’s root directory is used as src_dir (issue #635)
2.8.6 (2016-03-22)

- Launched PlatformIO Community Forums (issue #530)
- Added support for ARM mbed-enabled board Seed Arch Max (STM32F407VET6) (issue #572)
- Improved DNS lookup for PlatformIO API
- Updated Arduino Wiring-based framework to the latest version for Atmel AVR/SAM development platforms
- Updated “Teensy Loader CLI” and fixed uploading of large .hex files (issue #568)
- Updated the support for Sanguino Boards (issue #586)
- Better handling of used boards when re-initialize/update project
- Improved support for non-Unicode user profiles for Windows OS
- Disabled progress bar for download operations when prompts are disabled
- Fixed multiple definition errors for ST STM32 development platform and ARM mbed framework (issue #571)
- Fixed invalid board parameters (reset method and baudrate) for a few ESP8266 based boards
- Fixed “KeyError: ‘content-length’” in PlatformIO Download Manager (issue #591)

2.8.5 (2016-03-07)

- Project generator for NetBeans IDE (issue #541)
- Created package for Homebrew Mac OS X Package Manager: brew install platformio (issue #395)
- Updated Arduino core for Espressif platform to 2.1.0 (issue #544)
- Added support for the ESP8266 ESP-07 board to Espressif (issue #527)
- Improved handling of String-based CPPDEFINES passed to extra build_flags (issue #526)
- Generate appropriate project for CLion IDE and CVS (issue #523)
- Use src_dir directory from Project Configuration File platformio.ini when initializing project otherwise create base src directory (issue #536)
- Fixed issue with incorrect handling of user’s build flags where the base flags were passed after user’s flags to GCC compiler (issue #528)
- Fixed issue with Project Generator when optional build flags were passed using system environment variables: PLATFORMIO_BUILD_FLAGS or PLATFORMIO_BUILD_SRC_FLAGS
- Fixed invalid detecting of compiler type (issue #550)
- Fixed issue with updating package which was deleted manually by user (issue #555)
- Fixed incorrect parsing of GCC -include flag (issue #552)

2.8.4 (2016-02-17)

- Added support for the new ESP8266-based boards (ESPDuino, ESP-WROOM-02, ESPpresso Lite 1.0 & 2.0, SparkFun ESP8266 Thing Dev, ThaiEasyElec ESPino) to Espressif development platform
- Added board_f_flash option to Project Configuration File platformio.ini which allows to specify custom flash chip frequency for Espressif development platform (issue #501)
• Added `board_flash_mode` option to Project Configuration File `platformio.ini` which allows to specify custom flash chip mode for Espressif development platform
• Handle new environment variables `PLATFORMIO_UPLOAD_PORT` and `PLATFORMIO_UPLOAD_FLAGS` (issue #518)
  • Fixed issue with `CPPDEFINES` which contain space and break PlatformIO IDE Linter (IDE issue #34)
  • Fixed unable to link C++ standard library to Espressif platform build (issue #503)
  • Fixed issue with pointer (char* myfunc()) while converting from *.ino to *.cpp (issue #506)

2.8.3 (2016-02-02)

• Better integration of PlatformIO Builder with PlatformIO IDE Linter
• Fixed issue with removing temporary file while converting *.ino to *.cpp
• Fixed missing dependency (mbed framework) for Atmel SAM development platform (issue #487)

2.8.2 (2016-01-29)

• Corrected RAM size for NXP LPC1768 based boards (issue #484)
• Exclude only `test` and `tests` folders from build process
• Reverted `-Wl,-whole-archive` hook for ST STM32 and mbed

2.8.1 (2016-01-29)

• Fixed a bug with Project Initialization in PlatformIO IDE

2.8.0 (2016-01-29)

• PlatformIO IDE for Atom (issue #470)
• Added `pio` command line alias for `platformio command` (issue #447)
• Added SPL-Framework support for Nucleo F401RE board (issue #453)
• Added `upload_resetmethod` option to Project Configuration File `platformio.ini` which allows to specify custom upload reset method for Espressif development platform (issue #444)
• Allowed to force output of color ANSI-codes or to disable progress bar even if the output is a pipe (not a tty) using Environment variables (issue #465)
• Set 1Mb SPIFFS for Espressif boards by default (issue #458)
• Exclude `test` * folder by default from build process
• Generate project for IDEs with information about installed libraries
• Fixed builder for mbed framework and ST STM32 platform
2.7.1 (2016-01-06)

- Initial support for Arduino Zero board (issue #356)
- Added support for completions to Atom text editor using .clang_complete
- Generate default targets for supported IDE (CLion, Eclipse IDE, Emacs, Sublime Text, VIM): Build, Clean, Upload, Upload SPIFFS image, Upload using Programmer, Update installed platforms and libraries (issue #427)
- Updated Teensy Arduino Framework to 1.27 (issue #434)
- Fixed uploading of EEPROM data using uploadeep target for Atmel AVR development platform
- Fixed project generator for CLion IDE (issue #422)
- Fixed package shasum validation on Mac OS X 10.11.2 (issue #429)
- Fixed CMakeLists.txt add_executable has only one source file (issue #421)

2.7.0 (2015-12-30)

Happy New Year!

- Moved SCons to PlatformIO packages. PlatformIO does not require SCons to be installed in your system. Significantly simplified installation process of PlatformIO. pip install platformio rocks!
- Implemented uploading files to file system of ESP8266 SPIFFS (including OTA) (issue #382)
- Added support for the new Adafruit boards Bluefruit Micro and Feather (issue #403)
- Added support for RFDuino (issue #319)
- Project generator for Emacs text editor (pull #404)
- Updated Arduino framework for Atmel AVR development platform to 1.6.7
- Documented firmware uploading for Atmel AVR development platform using Programmers: AVR ISP, AVRISP mkII, USBtinyISP, USBasp, Parallel Programmer and Arduino as ISP
- Fixed issue with current Python interpreter for Python-based tools (issue #417)

2.6.3 (2015-12-21)

- Restored support for Espressif ESP8266 ESP-01 1MB board (ready for OTA)
- Fixed invalid ROM size for ESP8266-based boards (issue #396)

2.6.2 (2015-12-21)

- Removed SCons from requirements list. PlatformIO will try to install it automatically, otherwise users need to install it manually
- Fixed ChunkedEncodingError when SF connection is broken (issue #356)

2.6.1 (2015-12-18)

- Added support for the new ESP8266-based boards (SparkFun ESP8266 Thing, NodeMCU 0.9 & 1.0, Olimex MOD-WIFI-ESP8266-DEV), Adafruit HUZZAH ESP8266, ESPino, SweetPea ESP-210, WeMos D1, WeMos D1 mini) to Espressif development platform
• Created public platformio-pkg-lscripts repository for LD scripts. Moved common configuration for ESP8266 MCU to esp8266.flash.common.ld (issue #379)
• Improved documentation for Espressif development platform: OTA update, custom Flash Size, Upload Speed and CPU frequency
• Fixed reset method for Espressif NodeMCU (ESP-12E Module) (issue #380)
• Fixed issue with code builder when build path contains spaces (issue #387)
• Fixed project generator for Eclipse IDE and “duplicate path entries found in project path” (issue #383)

2.6.0 (2015-12-15)

• Install only required packages depending on build environment (issue #308)
• Added support for Raspberry Pi WiringPi framework (issue #372)
• Implemented Over The Air (OTA) upgrades for Espressif development platform. (issue #365)
• Updated CMSIS framework and added CMSIS support for Nucleo F401RE board (issue #373)
• Added support for Espressif ESP8266 ESP-01-1MB board (ready for OTA)
• Handle upload_flags option in platformio.ini (issue #368)
• Improved PlatformIO installation on the Mac OS X El Capitan

2.5.0 (2015-12-08)

• Improved code builder for parallel builds (up to 4 times faster than before)
• Generate .travis.yml CI and .gitignore files for embedded projects by default (issue #354)
• Removed prompt with “auto-uploading” from platformio init command and added --enable-auto-uploading option (issue #352)
• Fixed incorrect behaviour of platformio serialports monitor in pair with PySerial 3.0

2.4.1 (2015-12-01)

• Restored PLATFORMIO macros with the current version

2.4.0 (2015-12-01)

• Added support for the new boards: Atmel ATSAMR21-XPRO, Atmel SAML21-XPRO-B, Atmel SAMD21-XPRO, ST 32F469IDISCOVERY, ST 32L476GDISCOVERY, ST Nucleo F031K6, ST Nucleo F042K6, ST Nucleo F303K8 and ST Nucleo L476RG
• Updated Arduino core for Espressif platform to 2.0.0 (issue #345)
• Added to FAQ explanation of Can not compile a library that compiles without issue with Arduino IDE (issue #331)
• Fixed ESP-12E flash size (pull #333)
• Fixed configuration for LowPowerLab MoteinoMEGA board (issue #335)
• Fixed “LockFailed: failed to create appstate.json.lock” error for Windows
• Fixed relative include path for preprocessor using build_flags (issue #271)
2.3.5 (2015-11-18)

- Added support of libOpenCM3 framework for Nucleo F103RB board (issue #309)
- Added support for Espressif ESP8266 ESP-12E board (NodeMCU) (issue #310)
- Added support for pySerial 3.0 (issue #307)
- Updated Arduino AVR/SAM frameworks to 1.6.6 (issue #321)
- Upload firmware using external programmer via platformio run –target program target (issue #311)
- Fixed handling of upload port when board option is not specified in platformio.ini (issue #313)
- Fixed firmware uploading for nordicrf51 development platform (issue #316)
- Fixed installation on Mac OS X El Capitan (issue #312)
- Fixed project generator for CLion IDE under Windows OS with invalid path to executable (issue #326)
- Fixed empty list with serial ports on Mac OS X (isge #294)
- Fixed compilation error TWI_Disable not declared for Arduino Due board (issue #329)

2.3.4 (2015-10-13)

- Full support of CLion IDE including code auto-completion (issue #132)
- PlatformIO command completion in Terminal for bash and zsh
- Added support for ubIQio Ardhat board (pull #302)
- Install SCons automatically and avoid error: option --single-version-externally-managed not recognized (issue #279)
- Use Teensy CLI Loader for upload of .hex files on Mac OS X (issue #306)
- Fixed missing framework-mbed package for teensy platform (issue #305)

2.3.3 (2015-10-02)

- Added support for LightBlue Bean board (pull #292)
- Added support for ST Nucleo F446RE board (pull #293)
- Fixed broken lock file for “appstate” storage (issue #288)
- Fixed ESP8266 compile errors about RAM size when adding 1 library (issue #296)

2.3.2 (2015-09-10)

- Allowed to use ST-Link uploader for mbed-based projects
- Explained how to use lib directory from the PlatformIO based project in readme.txt which will be automatically generated using platformio init command (issue #273)
- Found solution for “pip/scons error: option --single-version-externally-managed not recognized” when install PlatformIO using pip package manager (issue #279)
- Fixed firmware uploading to Arduino Leonardo board using Mac OS (issue #287)
- Fixed SConsNotInstalled error for Linux Debian-based distributives
2.3.1 (2015-09-06)

- Fixed critical issue when platformio init –ide command hangs PlatformIO (issue #283)

2.3.0 (2015-09-05)

- Added native, linux_arm, linux_i686, linux_x86_64, windows_x86 development platforms (issue #263)
- Added PlatformIO Demo page to documentation
- Simplified installation process of PlatformIO (issue #274)
- Significantly improved Project Generator which allows to integrate with the most popular IDE
- Added short –h help option for PlatformIO and sub-commands
- Updated mbed framework
- Updated tool-teensty package for Teensy platform (issue #268)
- Added FAQ answer when Program “platformio” not found in PATH (issue #272)
- Generate “readme.txt” for project “lib” directory (issue #273)
- Use toolchain’s includes pattern include* for Project Generator (issue #277)
- Added support for Adafruit Gemma board to atmelavr platform (pull #256)
- Fixed includes list for Windows OS when generating project for Eclipse IDE (issue #270)
- Fixed AttributeError: 'module' object has no attribute 'packages' (issue #252)

2.2.2 (2015-07-30)

- Integration with Atom IDE
- Support for off-line/unpublished/private libraries (issue #260)
- Disable project auto-clean while building/uploading firmware using platformio run –disable-auto-clean option (issue #255)
- Show internal errors from “Miniterm” using platformio serialports monitor command (issue #257)
- Fixed platformio serialports monitor –help information with HEX char for hotkeys (issue #253)
- Handle “OSError: [Errno 13] Permission denied” for PlatformIO installer script (issue #254)

2.2.1 (2015-07-17)

- Project generator for CLion IDE (issue #132)
- Updated tool-bossac package to 1.5 version for atmelsam platform (issue #251)
- Updated sdk-esp8266 package for espressif platform
- Fixed incorrect arguments handling for platformio serialports monitor command (issue #248)
2.2.0 (2015-07-01)

- Allowed to exclude/include source files from build process using src_filter (issue #240)
- Launch own extra script before firmware building/uploading processes (issue #239)
- Specify own path to the linker script (ld) using build_flags option (issue #233)
- Specify library compatibility with the all platforms/frameworks using * symbol in library.json
- Added support for new embedded boards: ST 32L0538DISCOVERY and Delta DFCM-NNN40 to Framework mbed
- Updated packages for Framework Arduino (AVR, SAM, Espressif and Teensy cores, Framework mbed, Espressif ESP8266 SDK (issue #246)
  - Fixed stk500v2_command(): command failed (issue #238)
- Fixed IDE project generator when board is specified (issue #242)
- Fixed relative path for includes when generating project for IDE (issue #243)
- Fixed ESP8266 native SDK exception (issue #245)

2.1.2 (2015-06-21)

- Fixed broken link to SCons installer

2.1.1 (2015-06-09)

- Automatically detect upload port using VID:PID board settings (issue #231)
- Improved detection of build changes
- Avoided LibInstallDependencyError when more than 1 library is found (issue #229)

2.1.0 (2015-06-03)

- Added Silicon Labs EFM32 siliconlabsefm32 development platform (issue #226)
- Integrate PlatformIO with Circle CI and Shippable CI
- Described in documentation how to create/register own board for PlatformIO
- Disabled “nano.specs” for ARM-based platforms (issue #219)
- Fixed “ConnectionError” when PlatformIO SF Storage is off-line
- Fixed resolving of C/C++ std libs by Eclipse IDE (issue #220)
- Fixed firmware uploading using USB programmer (USBasp) for atmelavr platform (issue #221)

2.0.2 (2015-05-27)

- Fixed libraries order for “Library Dependency Finder” under Linux OS
2.0.1 (2015-05-27)

- Handle new environment variable PLATFORMIO_BUILD_FLAGS
- Pass to API requests information about Continuous Integration system. This information will be used by PlatformIO-API.
- Use include directories from toolchain when initialising project for IDE (issue #210)
- Added support for new WildFire boards from Wicked Device to atmelavr platform
- Updated Arduino Framework to 1.6.4 version (issue #212)
- Handle Atmel AVR Symbols when initialising project for IDE (issue #216)
- Fixed bug with converting *.ino to *.cpp
- Fixed failing with platformio init --ide eclipse without boards (issue #217)

2.0.0 (2015-05-22)

Made in Paradise

- PlatformIO as Continuous Integration (CI) tool for embedded projects (issue #108)
- Initialise PlatformIO project for the specified IDE (issue #151)
- PlatformIO CLI 2.0: “platform” related commands have been moved to platformio platforms subcommand (issue #158)
- Created PlatformIO gitter.im room (issue #174)
- Global --force option which will force to accept any confirmation prompts (issue #152)
- Run project with platformio run --project-dir option without changing the current working directory (issue #192)
- Control verbosity of platformio run command via -v/--verbose option
- Add library dependencies for build environment using lib_install option in platformio.ini (issue #134)
- Specify libraries which are compatible with build environment using lib_use option in platformio.ini (issue #148)
- Add more boards to PlatformIO project with platformio init --board command (issue #167)
- Choose which library to update (issue #168)
- Specify platformio init --env-prefix when initialise/update project (issue #182)
- Added new Armstrap boards (issue #204)
- Updated SDK for espressif development platform to v1.1 (issue #179)
- Disabled automatic updates by default for platforms, packages and libraries (issue #171)
- Fixed bug with creating copies of source files (issue #177)

1.22.4 PlatformIO 1.0

1.5.0 (2015-05-15)

- Added support of Framework mbed for Teensy 3.1 (issue #183)
- Added GDB as alternative uploader to ststm32 platform (issue #175)
• Added examples with preconfigured IDE projects (issue #154)
• Fixed firmware uploading under Linux OS for Arduino Leonardo board (issue #178)
• Fixed invalid “mbed” firmware for Nucleo F411RE (issue #185)
• Fixed parsing of includes for PlatformIO Library Dependency Finder (issue #189)
• Fixed handling symbolic links within source code directory (issue #190)
• Fixed cancelling any previous definition of name, either built in or provided with a -D option (issue #191)

1.4.0 (2015-04-11)

• Added espressif development platform with ESP01 board
• Integrated PlatformIO with AppVeyor Windows based Continuous Integration system (issue #149)
• Added support for Teensy LC board to teensy platform
• Added support for new Arduino based boards by SparkFun, BQ, LightUp, LowPowerLab, Quirkbot, RedBear-Lab, TinyCircuits to atmelavr platform
• Upgraded Arduino Framework to 1.6.3 version (issue #156)
• Upgraded Energia Framework to 0101E0015 version (issue #146)
• Upgraded Arduino Framework with Teensy Core to 1.22 version (issue #162, issue #170)
• Fixed exceptions with PlatformIO auto-updates when Internet connection isn’t active

1.3.0 (2015-03-27)

• Moved PlatformIO source code and repositories from Ivan Kravets account to PlatformIO Organisation (issue #138)
• Added support for new Arduino based boards by SparkFun, RepRap, Sanguino to atmelavr platform (issue #127, issue #131)
• Added integration instructions for Visual Studio and Sublime Text IDEs
• Improved handling of multi-file *.ino/pde sketches (issue #130)
• Fixed wrong insertion of function prototypes converting *.ino/pde (issue #137, issue #140)

1.2.0 (2015-03-20)

• Added full support of mbed framework including libraries: RTOS, Ethernet, DSP, FAT, USB.
• Added freescalekinetis development platform with Freescale Kinetis Freedom boards
• Added nordicnrf51 development platform with supported boards from JKSoft, Nordic, RedBearLab, Switch Science
• Added nxplpc development platform with supported boards from CQ Publishing, Embedded Artists, NGX Technologies, NXP, Outrageous Circuits, SeeedStudio, Solder Splash Labs, Switch Science, u-blox
• Added support for ST Nucleo boards to ststm32 development platform
• Created new Frameworks page in documentation and added to PlatformIO Web Site (issue #115)
• Introduced online Embedded Boards Explorer
• Automatically append define -DPLATFORMIO=%version% to builder (issue #105)
• Renamed stm32 development platform to ststm32
• Renamed opencm3 framework to libopencm3
• Fixed uploading for atmelSAM development platform
• Fixed re-arranging the *.ino/pde files when converting to *.cpp (issue #100)

1.1.0 (2015-03-05)
• Implemented PLATFORMIO_* environment variables (issue #102)
• Added support for SainSmart boards to atmelSAM development platform
• Added Project Configuration option named envs_dir
• Disabled “prompts” automatically for Continuous Integration systems (issue #103)
• Fixed firmware uploading for atmelavr boards which work within usbtiny protocol
• Fixed uploading for Digispark board (issue #106)

1.0.1 (2015-02-27)
PlatformIO 1.0 - recommended for production
• Changed development status from beta to Production/Stable
• Added support for ARM-based credit-card sized computers: Raspberry Pi, BeagleBone and CubieBoard
• Added atmelSAM development platform with supported boards: Arduino Due and Digistump DigiX (issue #71)
• Added ststm32 development platform with supported boards: Discovery kit for STM32L151/152, STM32F303xx, STM32F407/417 lines and libOpenCM3 Framework (issue #73)
• Added teensy development platform with supported boards: Teensy 2.x & 3.x (issue #72)
• Added new Arduino boards to atmelavr platform: Arduino NG, Arduino BT, Arduino Esplora, Arduino Ethernet, Arduino Robot Control, Arduino Robot Motor and Arduino Yun
• Added support for Adafruit boards to atmelavr platform: Adafruit Flora and Adafruit Trinkets (issue #65)
• Added support for Digispark boards to atmelavr platform: Digispark USB Development Board and Digispark Pro (issue #47)
• Covered code with tests (issue #2)
• Refactored Library Dependency Finder (issues #48, #50, #55)
• Added src_dir option to [platformio] section of platformio.ini which allows to redefine location to project’s source directory (issue #83)
• Added --json-output option to platformio boards and platformio search commands which allows to return the output in JSON format (issue #42)
• Allowed to ignore some libs from Library Dependency Finder via lib_ignore option
• Improved platformio run command: asynchronous output for build process, timing and detailed information about environment configuration (issue #74)
• Output compiled size and static memory usage with platformio run command (issue #59)
• Updated framework-arduino AVR & SAM to 1.6 stable version
• Fixed an issue with the libraries that are git repositories (issue #49)
• Fixed handling of assembly files (issue #58)
• Fixed compiling error if space is in user’s folder (issue #56)
• Fixed AttributeError: ‘module’ object has no attribute ‘disable_warnings’ when a version of requests package is less than 2.4.0
• Fixed bug with invalid process’s “return code” when PlatformIO has internal error (issue #81)
• Several bug fixes, increased stability and performance improvements

1.22.5 PlatformIO 0.0

0.10.2 (2015-01-06)
• Fixed an issue with --json-output (issue #42)
• Fixed an exception during platformio upgrade under Windows OS (issue #45)

0.10.1 (2015-01-02)
• Added --json-output option to platformio list, platformio serialports list and platformio lib list commands which allows to return the output in JSON format (issue #42)
• Fixed missing auto-uploading by default after platformio init command

0.10.0 (2015-01-01)
Happy New Year!
• Implemented platformio boards command (issue #11)
• Added support of Engduino boards for atmelavr platform (issue #38)
• Added --board option to platformio init command which allows to initialise project with the specified embedded boards (issue #21)
• Added example with uploading firmware via USB programmer (USBasp) for atmelavr MCUs (issue #35)
• Automatic detection of port on platformio serialports monitor (issue #37)
• Allowed auto-installation of platforms when prompts are disabled (issue #43)
• Fixed urllib3’s SSL warning under Python <= 2.7.2 (issue #39)
• Fixed bug with Arduino USB boards (issue #40)

0.9.2 (2014-12-10)
• Replaced “dark blue” by “cyan” colour for the texts (issue #33)
• Added new setting enable_prompts and allowed to disable all PlatformIO prompts (useful for cloud compilers) (issue #34)
• Fixed compilation bug on Windows with installed MSVC (issue #18)
0.9.1 (2014-12-05)

- Ask user to install platform (when it hasn’t been installed yet) within platformio run and platformio show commands
- Improved main documentation
- Fixed “OSError: [Errno 2] No such file or directory” within platformio run command when PlatformIO isn’t installed properly
- Fixed example for Eclipse IDE with Tiva board (issue #32)
- Upgraded Eclipse Project Examples to latest Luna and PlatformIO releases

0.9.0 (2014-12-01)

- Implemented platformio settings command
- Improved platformio init command. Added new option --project-dir where you can specify another path to directory where new project will be initialized (issue #31)
- Added Migration Manager which simplifies process with upgrading to a major release
- Added Telemetry Service which should help us make PlatformIO better
- Implemented PlatformIO AppState Manager which allow to have multiple .platformio states.
- Refactored Package Manager
- Download Manager: fixed SHA1 verification within Cygwin Environment (issue #26)
- Fixed bug with code builder and built-in Arduino libraries (issue #28)

0.8.0 (2014-10-19)

- Avoided trademark issues in library.json with the new fields: frameworks, platforms and dependencies (issue #17)
- Switched logic from “Library Name” to “Library Registry ID” for all platformio lib commands (install, uninstall, update and etc.)
- Renamed author field to authors and allowed to setup multiple authors per library in library.json
- Added option to specify “maintainer” status in authors field
- New filters/options for platformio lib search command: --framework and --platform

0.7.1 (2014-10-06)

- Fixed bug with order for includes in conversation from INO/PDE to CPP
- Automatic detection of port on upload (issue #15)
- Fixed lib update crashing when nolibs are installed (issue #19)

0.7.0 (2014-09-24)

- Implemented new [platformio] section for Configuration File with home_dir option (issue #14)
- Implemented Library Manager (issue #6)
0.6.0 (2014-08-09)

- Implemented platformio serialports monitor (issue #10)
- Fixed an issue ImportError: No module named platformio.util (issue #9)
- Fixed bug with auto-conversation from Arduino *.ino to *.cpp

0.5.0 (2014-08-04)

- Improved nested lookups for libraries
- Disabled default warning flag “-Wall”
- Added auto-conversation from *.ino to valid *.cpp for Arduino/Energia frameworks (issue #7)
- Added Arduino example with external library (Adafruit CC3000)
- Implemented platformio upgrade command and “auto-check” for the latest version (issue #8)
- Fixed an issue with “auto-reset” for Raspduino board
- Fixed a bug with nested libs building

0.4.0 (2014-07-31)

- Implemented platformio serialports command
- Allowed to put special build flags only for src files via src_build_flags environment option
- Allowed to override some of settings via system environment variables such as: PLATFORMIO_SRC_BUILD_FLAGS and PLATFORMIO_ENVS_DIR
- Added --upload-port option for platformio run command
- Implemented (especially for SmartAnthill) platformio run -t uploadlazy target (no dependencies to framework libs, ELF and etc.)
- Allowed to skip default packages via platformio install --skip-default-package option
- Added tools for Raspberry Pi platform
- Added support for Microuino and Raspduino boards in atmelavr platform

0.3.1 (2014-06-21)

- Fixed auto-installer for Windows OS (bug with %PATH% custom installation)

0.3.0 (2014-06-21)

- Allowed to pass multiple “SomePlatform” to install/uninstall commands
- Added “IDE Integration” section to README with Eclipse project examples
- Created auto installer script for PlatformIO (issue #3)
- Added “Super-Quick” way to Installation section (README)
- Implemented “build_flags” option for environments (issue #4)
**0.2.0 (2014-06-15)**

- Resolved issue #1 “Build referred libraries”
- Renamed project’s “libs” directory to “lib”
- Added arduino-internal-library example
- Changed to beta status

**0.1.0 (2014-06-13)**

- Birth! First alpha release

### 1.23 Migrating from 3.x to 4.0

Guidance on how to upgrade from *PlatformIO Core (CLI)* v3.x to v4.x with emphasis on major changes, what is new, and what has been removed.

Please read *PlatformIO 4.0 Release Notes* before.

**Contents**

- Compatibility
- Infrastructure
- What is new
  - “platformio.ini” (Project Configuration File)
    - * Global scope [env]
    - * External Configuration Files
    - * New Options
  - Library Management
  - Build System
    - * Package Dependencies
    - * Custom Platform Packages
    - * Custom Upload Command
    - * Shared Cache Directory
- What is changed or removed
  - Command Line Interface
  - “platformio.ini” (Project Configuration File)

### 1.23.1 Compatibility

*PlatformIO Core 4.0 is fully backward compatible with v3.x.* The only major change is a new location for project build artifacts and library dependencies. The previous `.pioenvs (build_dir)` and `.piolibdeps (libdeps_dir)`
folders were moved to a new \textit{workspace_dir}.

\textbf{Note:} If you manually added library dependencies to old .pioilibdeps folder, please declare them in \textit{lib_deps}. We do not recommend modifying any files or folders in \textit{workspace_dir}. This is an internal location for PlatformIO Core artifacts and temporary files. PlatformIO Core 4.0 may delete/cleanup this folder in a service purpose any time.

\subsection*{1.23.2 Infrastructure}

Finally, Python 3 interpreter is officially supported! The minimum requirements are Python 2.7 or Python 3.5+.

We also added full support for operating system locales other than UTF-8. So, your project path can contain non-ASCII/non-Latin chars now.

If you are Development Platforms maintainer or you need to show a progress bar (upload progress, connecting status...), PlatformIO Core 4.0 has re-factored target runner where line-buffering was totally removed. Just print any progress information in real time and PlatformIO Core will display it instantly on user the side. For example, a writing progress from \textit{Atmel AVR} “avrdude” programmer:

```
...
Looking for upload port...
Auto-detected: /dev/cu.usbmodemFA141
Uploading 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 "build/uno/firmware.hex"
avrdude: writing flash (930 bytes):

Writing | #####################################################################
```

\subsection*{1.23.3 What is new}

In this section, we are going to highlight the most important changes and features introduced in \textit{PlatformIO Core (CLI)} 4.0. Please visit \textit{PlatformIO 4.0 Release Notes} for more detailed information.

\textbf{“platformio.ini” (Project Configuration File)}

A project configuration parser was rewritten from scratch. It has strict options typing (API) and helps to avoid issues when option values are invalid (for example, invalid \textit{Dependency Finder Mode} or non-existing \textit{debug_svd_path}).

\textbf{Global scope [env]}

One of the most requested features was a “global” or “common” project environment (\textit{Section [env]}) where developers can share common configuration between all declared build environments [env:NAME].

The previous solution in PlatformIO Core 3.0 was using \textit{Dynamic variables}. As practice has shown, this approach was not good and more advanced “platformio.ini” (Project Configuration File) looked so complicated and hard for managing (for example, open source projects MarlinFirmware, Espurna).
PlatformIO Core 4.0 introduces a new global scope named `[env]` which allows declaring global options that will be shared between all `[env:NAME]` sections in “`platformio.ini`” (Project Configuration File). 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 2 build environments release and debug. This is the same if you duplicate all options (PlatformIO Core 3.0 compatible):

```
[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
```

External Configuration Files

To simplify the project configuration process, PlatformIO Core 4.0 adds support for external “`platformio.ini`” (Project Configuration File). Yes! You can finally extend one configuration file with another or with a list of them. The cool feature is a support for Unix shell-style wildcards. So, you can dynamically generate “`platformio.ini`” (Project Configuration File) files or load bunch of them from a folder. See `extra_configs` option for details and a simple example below:

Base “`platformio.ini`”

```
[platformio]
extra_configs =
   extra_envs.ini
   extra_debug.ini

[common]
debug_flags = -D RELEASE
lib_flags = -lc -lm
```

(continues on next page)
```
[env:esp-wrover-kit]
platform = espressif32
framework = espidf
board = esp-wrover-kit
build_flags = ${common.debug_flags}
```

```
“extra_envs.ini”
```

```
[env:esp32dev]
platform = espressif32
framework = espidf
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}
```

```
[env:lolin32]
platform = espressif32
framework = espidf
board = lolin32
build_flags = -Og
```
New Options

We have added new options and changed some existing ones. Here are the new or updated options.

<table>
<thead>
<tr>
<th>Section</th>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>platformio</td>
<td>extra_configs</td>
<td>Extend base configuration with external &quot;platformio.ini&quot; (Project Configuration File)</td>
</tr>
<tr>
<td>platformio</td>
<td>core_dir</td>
<td>Directory where PlatformIO stores development platform packages (toolchains, frameworks, SDKs, upload and debug tools), global libraries for Library Dependency Finder (LDF), and other PlatformIO Core service data</td>
</tr>
<tr>
<td>platformio</td>
<td>global-lib_dir</td>
<td>Global library storage for PlatformIO projects and Library Manager where Library Dependency Finder (LDF) looks for dependencies</td>
</tr>
<tr>
<td>platformio</td>
<td>platforms_dir</td>
<td>Global storage where PlatformIO Package Manager installs Development Platforms</td>
</tr>
<tr>
<td>platformio</td>
<td>packages_dir</td>
<td>Global storage where PlatformIO Package Manager installs Development Platforms dependencies (toolchains, Frameworks, SDKs, upload and debug tools)</td>
</tr>
<tr>
<td>platformio</td>
<td>cache_dir</td>
<td>PlatformIO Core (CLI) uses this folder to store caching information (requests to PlatformIO Registry, downloaded packages and other service information)</td>
</tr>
<tr>
<td>platformio</td>
<td>workspace_dir</td>
<td>A path to a project workspace directory where PlatformIO keeps by default compiled objects, static libraries, firmwares, and external library dependencies</td>
</tr>
<tr>
<td>platformio</td>
<td>shared_dir</td>
<td>PIO Remote uses this folder to synchronize extra files between remote machine</td>
</tr>
<tr>
<td>env</td>
<td>build_type</td>
<td>See extended documentation for Build Configurations</td>
</tr>
<tr>
<td>env</td>
<td>monitor_flags</td>
<td>Pass extra flags and options to platformio device monitor command</td>
</tr>
<tr>
<td>env</td>
<td>upload_command</td>
<td>Override default Development Platforms upload command with a custom one.</td>
</tr>
</tbody>
</table>

Library Management

Library management brings a few new changes which resolve historical issues presented in PlatformIO 3.0:

1. .piolibdeps folder was moved to libdeps_dir of project workspace.

   If you manually added library dependencies to old .piolibdeps folder, please declare them in lib_deps. We do not recommend modifying any files or folders in workspace_dir. This is an internal location for PlatformIO Core artifacts and temporary files. PlatformIO Core 4.0 may delete/cleanup this folder in a service purpose any time.

2. Library Dependency Finder (LDF) now uses isolated library dependency storage per project build environment.

   It resolves conflicts when the libraries from different build environments declared via lib_deps option were installed into the same .piolibdeps folder.

See Library Management section in PlatformIO 4.0 release notes for more details.

Build System

PlatformIO Core 4.0 uses a new build_dir instead of .pioenvs for compiled objects, archived libraries, firmware binaries and, other artifacts. A new build_type option allows you to control a build process between “Release” and “Debug” modes (see Build Configurations).

See Build System section in PlatformIO 4.0 release notes for more details.
Package Dependencies

PlatformIO has decentralized architecture and allows platform maintainers to create Custom Development Platform for PlatformIO ecosystem. Each development platform depends on a list of packages (toolchains, SDKs, debugging servers, etc). PlatformIO Package Manager installs these packages automatically and PlatformIO Build System uses them later.

Starting from PlatformIO Core 4.0, developers can see which versions of a development platform or its dependent packages will be used. This is a great addition to track changes (Frameworks, SDKs) between Development Platforms updates. See an example with “staging” (Git) version of Espressif 8266 development platform:

```
Processing nodemcuv2 (platform: https://github.com/platformio/platform-espressif8266.˓→git#feature/stage; board: nodemcuv2; framework: arduino)
Verbose mode can be enabled via `-v, --verbose` option
CONFIGURATION: https://docs.platformio.org/page/boards/espressif8266/nodemcuv2.html
PLATFORM: Espressif 8266 (Stage) 2.3.0-alpha.1 #990141d > NodeMCU 1.0 (ESP-12E Module)
HARDWARE: ESP8266 80MHz, 80KB RAM, 4MB Flash
PACKAGES: toolchain-xtensa 2.40802.190218 (4.8.2), tool-esptool 1.413.0 (4.13), tool-˓→esptoolpy 1.20600.0 (2.6.0), framework-arduinoespressif8266 78a1a66
LDF Modes: Finder ~ chain+, Compatibility ~ soft
Found 35 compatible libraries
Scanning dependencies...
```

Custom Platform Packages

Sometimes you need to override default Development Platforms packages or add an extra. PlatformIO Core 4.0 introduces a new configuration option platform_packages per a build environment. Also, using this option you can install external packages and use them later as programmers or upload tools. See a few examples:

```
[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
```
Custom Upload Command

PlatformIO’s Development Platforms have pre-configured settings to program boards or devices. They depend on a type of bootloader or programming interface. PlatformIO Core 4.0 allows you to override default upload command using `upload_command` option in “platformio.ini” (Project Configuration File):

```
[env:custom_upload_cmd]
platform = ...
framework = ...
board = ...
upload_command = /my/flasher arg1 arg2 --flag1 $SOURCE
```

See real examples for `upload_command`.

Shared Cache Directory

PlatformIO Core 4.0 allows you to configure a shared folder for the derived files (objects, firmwares, ELFs) from a build system using `build_cache_dir`. You can use it in multi-environments project configuration to avoid multiple compilations of the same source code files.

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 = /tmp/platformio-shared-cache

[env:bluepill_f103c6]
platform = ststm32
framework = stm32cube
board = bluepill_f103c6

[env:nucleo_f411re]
platform = ststm32
framework = stm32cube
board = nucleo_f411re
```

You can also use the same `build_cache_dir` between different projects if they use the same Development Platforms and Frameworks.

1.23.4 What is changed or removed

Command Line Interface

The following commands have been changed in v4.0.
<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>platformio run</td>
<td>Added <em>platformio run --jobs</em> option</td>
</tr>
<tr>
<td>platformio update</td>
<td>Replaced <code>-c, --only-check</code> with <em>platformio update --dry-run</em></td>
</tr>
<tr>
<td>platformio lib update</td>
<td>Replaced <code>-c, --only-check</code> with <em>platformio lib update --dry-run</em></td>
</tr>
<tr>
<td>platformio platform update</td>
<td>Replaced <code>-c, --only-check</code> with <em>platformio platform update --dry-run</em></td>
</tr>
<tr>
<td>platformio remote update</td>
<td>Replaced <code>-c, --only-check</code> with <em>platformio remote update --dry-run</em></td>
</tr>
</tbody>
</table>

“*platformio.ini*” (Project Configuration File)  

The following options have been changed in v4.0.

<table>
<thead>
<tr>
<th>Section</th>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>platformio</td>
<td>env_default</td>
<td>Renamed to <em>default_envs</em></td>
</tr>
<tr>
<td>platformio</td>
<td>home_dir</td>
<td>Renamed to <em>core_dir</em></td>
</tr>
<tr>
<td>env</td>
<td>debug_load_cmd</td>
<td>Renamed to <em>debug_load_cmds</em> and allowed to pass more than one load command</td>
</tr>
</tbody>
</table>
[Embeds] Embeds.com: Develop easier with PlatformIO ecosystem
Symbols

--quiet
  platformio-device-monitor command line option, 35
platformio-remote-device-monitor command line option, 91

-build-dir
  platformio-ci command line option, 28

core-packages
  platformio-update command line option, 116

-dev
  platformio-upgrade command line option, 120

-disable-auto-clean
  platformio-remote-run command line option, 94
platformio-run command line option, 103

dry-run
  platformio-lib-update command line option, 70
platformio-platform-update command line option, 85
platformio-remote-update command line option, 98
platformio-update command line option, 116

dtr
  platformio-device-monitor command line option, 34
platformio-remote-device-monitor command line option, 91

echo
  platformio-device-monitor command line option, 34
platformio-remote-device-monitor command line option, 91

encoding
  platformio-device-monitor command line option, 34
platformio-remote-device-monitor command line option, 91

-env-prefix
  platformio-init command line option, 38

eol
  platformio-device-monitor command line option, 35
platformio-remote-device-monitor command line option, 91

-exclude
  platformio-ci command line option, 28

-exit-char
  platformio-device-monitor command line option, 35
platformio-remote-device-monitor command line option, 91

-force, -f
  platformio command line option, 19

-help, -h
  platformio command line option, 19

-host
  platformio-home command line option, 37

-id
  platformio-lib-search command line option, 55

-ide
  platformio-init command line option, 38

-installed
  platformio-boards command line option, 24

-interactive
  platformio-lib-install command line option, 46
-interface
  platformio-debug command line option, 30
-json-output
  platformio-account-show command line option, 23
  platformio-account-token command line option, 24
  platformio-boards command line option, 25
  platformio-device-list command line option, 32
  platformio-lib-built-in command line option, 41
  platformio-lib-list command line option, 50
  platformio-lib-search command line option, 55
  platformio-lib-show command line option, 62, 64
  platformio-lib-update command line option, 70
  platformio-platform-frameworks command line option, 71
  platformio-platform-list command line option, 77
  platformio-platform-search command line option, 79
  platformio-platform-update command line option, 85
  platformio-remote-device-list command line option, 89
-keep-build-dir
  platformio-ci command line option, 28
-logical
  platformio-device-list command line option, 31
-mdns
  platformio-device-list command line option, 32
-menu-char
  platformio-device-monitor command line option, 35
  platformio-remote-device-monitor command line option, 91
-monitor-dtr
  platformio-test command line option, 115
-monitor-rts
  platformio-test command line option, 115
-no-open
  platformio-home command line option, 37
-no-reset
  platformio-test command line option, 115
-page
  platformio-lib-search command line option, 55
-parity
  platformio-device-monitor command line option, 34
  platformio-remote-device-monitor command line option, 90
-password, -p
  platformio-account-login command line option, 21
-port
  platformio-home command line option, 37
-raw
  platformio-device-monitor command line option, 35
  platformio-remote-device-monitor command line option, 91
-regenerate
  platformio-account-token command line option, 24
-rts
  platformio-device-monitor command line option, 34
  platformio-remote-device-monitor command line option, 91
-rtscts
  platformio-device-monitor command line option, 34
  platformio-remote-device-monitor command line option, 90
-save
  platformio-lib-install command line option, 46
-serial
  platformio-device-list command line option, 31
-skip-default
  platformio-platform-install command line option, 74
-storage
  platformio-lib-built-in command line option, 41
-test-port
  platformio-remote-test command line option, 97
  platformio-test command line option, 115
-upload-port
platformio-remote-run command line option, 94
platformio-remote-test command line option, 97
platformio-run command line option, 103
platformio-test command line option, 115
-name, -u
platformio-account-forgot command line option, 20
platformio-account-login command line option, 21
platformio-account-register command line option, 23
-version
platformio command line option, 19
-with-package
platformio-platform-install command line option, 74
-without-building
platformio-remote-test command line option, 97
platformio-test command line option, 115
-without-package
platformio-platform-install command line option, 74
-without-uploading
platformio-remote-test command line option, 97
platformio-test command line option, 115
-xonxoff
platformio-device-monitor command line option, 34
platformio-remote-device-monitor command line option, 90
-O, -project-option
platformio-ci command line option, 28
platformio-init command line option, 38
-c, -only-check
platformio-lib-update command line option, 70
platformio-platform-update command line option, 85
platformio-remote-update command line option, 98
platformio-update command line option, 116
-c, -project-conf
platformio-ci command line option, 28
platformio-debug command line option, 29
platformio-run command line option, 103
platformio-test command line option, 115
-d, -project-dir
platformio-debug command line option, 29
platformio-device-monitor command line option, 35
platformio-init command line option, 38
platformio-remote-run command line option, 94
platformio-remote-test command line option, 97
platformio-test command line option, 103
platformio-run command line option, 115
platformio-test command line option, 115
-d, -storage-dir
platformio-ci command line option, 28
platformio-remote-agent-start command line option, 90
platformio-lib command line option, 40
-d, -working-dir
platformio-ci command line option, 28
platformio-init command line option, 38
-e, -environment
platformio-debug command line option, 29
platformio-device-monitor command line option, 35
platformio-lib command line option, 40
platformio-remote-run command line option, 93
platformio-remote-test command line option, 96
platformio-run command line option, 102
platformio-test command line
  option, 114
-f, -filter
  platformio-device-monitor command
  line option, 34
  platformio-remote-device-monitor
  command line option, 91
  platformio-test command line
  option, 114
-f, -force
  platformio-lib-install command
  line option, 46
  platformio-platform-install
  command line option, 74
-f, -framework
  platformio-lib-search command line
  option, 55
-g, -global
  platformio-lib command line option, 40
-i, -header
  platformio-lib-search command line
  option, 55
-i, -ignore
  platformio-remote-test command
  line option, 96
  platformio-test command line
  option, 115
-j, -jobs
  platformio-run command line option, 103
-k, -keyword
  platformio-lib-search command line
  option, 55
-l, -lib
  platformio-ci command line option, 28
-n, -name
  platformio-lib-search command line
  option, 55
  platformio-remote-agent-start
  command line option, 87
-p, -only-packages
  platformio-platform-update command
  line option, 85
-p, -platform
  platformio-lib-search command line
  option, 55
-p, -port
  platformio-device-monitor command
  line option, 34
  platformio-remote-device-monitor
  command line option, 90
-r, -force-remote
platformio-remote-run command line
  option, 94
platformio-remote-test command
  line option, 97
-s, -share
  platformio-remote-agent-start
  command line option, 87
-s, -silent
  platformio-init command line
  option, 38
  platformio-lib-install command
  line option, 46
  platformio-run command line option, 103
-t, -target
  platformio-remote-run command line
  option, 93
  platformio-run command line option, 102
-v, -verbose
  platformio-ci command line option, 28
  platformio-debug command line
  option, 30
  platformio-remote-run command line
  option, 94
  platformio-remote-test command
  line option, 97
  platformio-run command line option, 103
  platformio-test command line
  option, 115

E
 environment variable
 CI, 218
 PLATFORMIO_AUTH_TOKEN, 21, 24, 218, 1659,
  1887
 PLATFORMIO_BOARDS_DIR, 190, 219
 PLATFORMIO_BUILD_CACHE_DIR, 187, 219
 PLATFORMIO_BUILD_DIR, 188, 219
 PLATFORMIO_BUILD_FLAGS, 196, 220, 1878,
  1888
 PLATFORMIO_CACHE_DIR, 187, 219
 PLATFORMIO_CORE_DIR, 186, 218, 1904
 PLATFORMIO_DATA_DIR, 190, 219
 PLATFORMIO_DEFAULT_ENVS, 193, 219
 PLATFORMIO_DISABLE_PROGRESSBAR, 218
 PLATFORMIO_EXTRA_SCRIPTS, 220, 222
 PLATFORMIO_FORCE_COLOR, 218
 PLATFORMIO_GLOBALLIB_DIR, 186, 218
 PLATFORMIO_INCLUDE_DIR, 188, 219
 PLATFORMIO_LIB_DIR, 189, 219
 PLATFORMIO_LIB_EXTRA_DIRS, 206, 219
platformio command line option
-force, -f, 19
-help, -h, 19
-version, 19
platformio-account-forgot command line option
-username, -u, 20
platformio-account-login command line option
-password, -p, 21
-username, -u, 21
platformio-account-register command line option
-username, -u, 23
platformio-account-show command line option
-username, -u, 24
platformio-boards command line option
-installed, 24
platformio-ci command line option
-build-dir, 28
-exclude, 28
-platformio-ci command line option
-build-dir, 28
-exclude, 28
platformio-debug command line option
-interface, 30
-platformio-device-list command line option
--quiet, 35
dtr, 34
echo, 34
encoding, 34
eol, 35
exit-char, 35
menu-char, 35
parity, 34
raw, 35
rts, 34
rtscts, 34
xonxoff, 34
-b, -baud, 34
d, -project-dir, 35
e, -environment, 35
-f, -filter, 34
-p, -port, 34
platformio-device-monitor command line option
--quiet, 35
dtr, 34
echo, 34
encoding, 34
eol, 35
exit-char, 35
menu-char, 35
parity, 34
raw, 35
rts, 34
rtscts, 34
xonxoff, 34
-b, -baud, 34
d, -project-dir, 35
e, -environment, 35
-f, -filter, 34
-p, -port, 34
platformio-home command line option
-host, 37
-no-open, 37
-port, 37
platformio-init command line option
-ide, 38
-env-prefix, 38
-o, -project-option, 38
-b, -board, 38
d, -project-dir, 38
-s, -silent, 38
platformio-lib command line option
-d, -storage-dir, 40
-e, -environment, 40
-g, -global, 40
platformio-lib-builtin command line option
-json-output, 41
-storage, 41
platformio-lib-install command line option
-interactive, 46
-save, 46
-f, -force, 46
-s, -silent, 46
platformio-lib-list command line option
-json-output, 50
platformio-lib-search command line option
-id, 55
-json-output, 55
-page, 55
-a, -author, 55
-f, -framework, 55
-i, -header, 55
-k, -keyword, 55
-n, -name, 55
-p, -platform, 55
platformio-lib-show command line option
-json-output, 62, 64
platformio-lib-update command line option
-dry-run, 70
-json-output, 70
-c, -only-check, 70
platformio-platform-frameworks command line option
-json-output, 71
platformio-platform-install command line option
-skip-default, 74
-with-package, 74
-without-package, 74
-f, -force, 74
platformio-platform-list command line option
-json-output, 77
platformio-platform-search command line option
-json-output, 79
platformio-platform-update command line option
-dry-run, 85
-json-output, 85
-c, -only-check, 85
-p, -only-packages, 85
platformio-remote-agent-start command line option
-d, -working-dir, 87
-n, -name, 87
-s, -share, 87
platformio-remote-device-list command line option
-json-output, 89
platformio-remote-device-monitor command line option
--quiet, 91
dtr, 91
-echo, 91
-encoding, 91
eol, 91
-exit-char, 91
-menu-char, 91
-parity, 90
-raw, 91
-rts, 91
-rtscts, 90
-xonxoff, 90
-b, -baud, 90
-f, -filter, 91
-p, -port, 90
platformio-remote-run command line option
-disable-auto-clean, 94
-upload-port, 94
d, -project-dir, 94
-e, -environment, 93
-r, -force-remote, 94
t, -target, 93
-v, -verbose, 94
platformio-remote-test command line option
-test-port, 97
-upload-port, 97
-without-building, 97
-without-uploading, 97
d, -project-dir, 97
e, -environment, 96
-i, -ignore, 96
-r, -force-remote, 97
-v, -verbose, 97
platformio-remote-update command line option
-dry-run, 98
-c, -only-check, 98
platformio-run command line option
-disable-auto-clean, 103
-upload-port, 103
c, -project-conf, 103
-d, -project-dir, 103
-e, -environment, 102
-j, -jobs, 103
-s, -silent, 103
-t, -target, 102
-v, -verbose, 103
platformio-test command line option
  -monitor-dtr, 115
  -monitor-rts, 115
  -no-reset, 115
  -test-port, 115
  -upload-port, 115
  -without-building, 115
  -without-uploading, 115
  -c, -project-conf, 115
  -d, -project-dir, 115
  -e, -environment, 114
  -f, -filter, 114
  -i, -ignore, 115
  -v, -verbose, 115
platformio-update command line option
  -core-packages, 116
  -dry-run, 116
  -c, -only-check, 116
platformio-upgrade command line option
  -dev, 120
PLATFORMIO_AUTH_TOKEN, 21, 24, 1659, 1887
PLATFORMIO_BOARDS_DIR, 190
PLATFORMIO_BUILD_CACHE_DIR, 187
PLATFORMIO_BUILD_DIR, 188
PLATFORMIO_BUILD_FLAGS, 196, 1878, 1888
PLATFORMIO_CACHE_DIR, 187
PLATFORMIO_CORE_DIR, 186, 1904
PLATFORMIO_DATA_DIR, 190
PLATFORMIO_DEFAULT_ENVS, 183
PLATFORMIO_DISABLE_PROGRESSBAR, 218
PLATFORMIO_EXTRA_SCRIPTS, 222
PLATFORMIO_GLOBALLIB_DIR, 186
PLATFORMIO_INCLUDE_DIR, 188
PLATFORMIO_LIB_DIR, 189
PLATFORMIO_LIB_EXTRA_DIRS, 206
PLATFORMIO_LIBDEPS_DIR, 188
PLATFORMIO_PACKAGES_DIR, 187
PLATFORMIO_PLATFORMS_DIR, 186
PLATFORMIO_SETTING_FORCE_VERBOSE, 29, 30, 94, 97, 103, 116
PLATFORMIO_SHARED_DIR, 190
PLATFORMIO_SRC_BUILD_FLAGS, 198
PLATFORMIO_SRC_DIR, 189
PLATFORMIO_SRC_FILTER, 199
PLATFORMIO_TEST_DIR, 190
PLATFORMIO_UPLOAD_FLAGS, 200
PLATFORMIO_UPLOAD_PORT, 200
PLATFORMIO_WORKSPACE_DIR, 183, 188