
PlatformIO Documentation

Release 4.1.0b5

PlatformIO

Oct 22, 2019

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	220
1.8	Advanced Scripting	223
1.9	Library Manager	234
1.10	Development Platforms	253
1.11	Frameworks	409
1.12	Boards	527
1.13	Custom Platform & Board	1733
1.14	PIO Account	1738
1.15	PIO Check	1739
1.16	PIO Remote	1747
1.17	PIO Unified Debugger	1753
1.18	PIO Unit Testing	1872
1.19	Cloud & Desktop IDE	1878
1.20	Continuous Integration	1981
1.21	Articles about us	1998
1.22	Frequently Asked Questions	2002
1.23	Release Notes	2010
1.24	Migrating from 3.x to 4.0	2040
	Bibliography	2049
	Index	2051

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

Social: [Twitter](#) | [Facebook](#) | [Hackaday](#) | [Bintray](#) | [Community](#)

CHAPTER 1

Contents

1.1 What is PlatformIO?

Contents

- *Press about PlatformIO*
- *Awards*
- *Problematic*
- *Overview*
- *User SHOULD have a choice*
- *How does it work?*

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”, - [Embedds].

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](#)

The screenshot shows the PlatformIO IDE interface with several windows open:

- Code Editor:** Shows the `WiFi.cpp` file with code related to WiFi provisioning.
- Memory View:** Displays memory at address `0x42000800+1`, showing variable values like `provSsid` and `strM2MAPConfig`.
- Assembly View:** Shows the assembly code for the `WiFiClass::startProvision` function.
- Terminal:** Displays the output of the `platformio device monitor` command, showing the WiFi shield starting in provisioning mode.
- Sidebar:** Contains sections for **VARIABLES**, **WATCH**, **CALL STACK**, **BREAKPOINTS** (with a checked entry for `WiFi.cpp .piolib...`), **PERIPHERALS** (including RTC, SERCOM0, I2CM, I2CS, SPI), **REGISTERS** (showing `pc = 0x00000726` and `xPSR = 0x21000000`), **MEMORY**, and **DISASSEMBLY**.

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

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

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

If you need **PlatformIO Core** commands outside PlatformIO IDE, please [Install Shell Commands](#).

1.3.1 Demo

Contents

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

“Blink Project”

Used in demo

1. Source code of [Wiring Blink Example](#)
2. *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

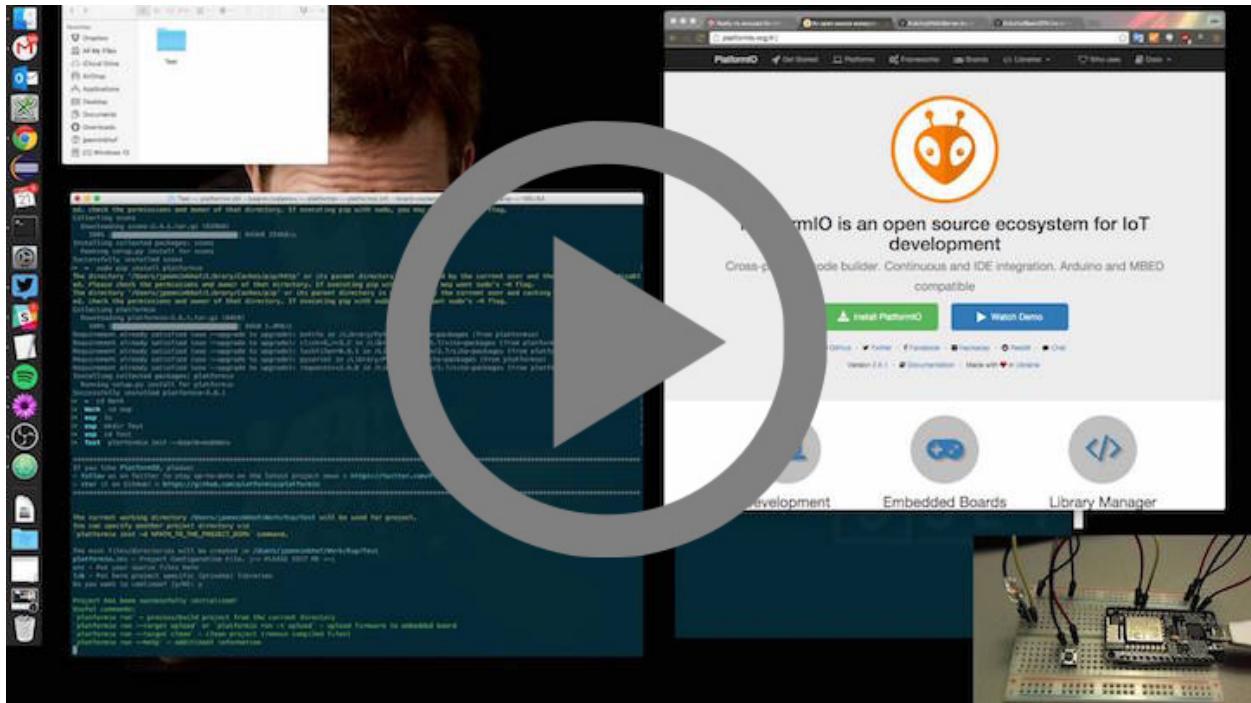
4. *platformio platform show teensy* command
5. *platformio platform update* command.

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`):

```
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`):

```
# 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:

```
# 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:

```
brew install platformio
```

Full Guide

1. Check a python version:

```
python --version
```

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

2. Install a `platformio` and related packages:

```
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:

```
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 -c
→"import os; print(os.path.join(os.getenv('PLATFORMIO_CORE_DIR', os.path.expanduser('~')),'.platformio/penv'))"
#####
##### Examples
# Windows
# C:\Users\UserName\.platformio\penv

# Linux
# ~/.platformio/penv
# /home/username/.platformio/penv

# macOS
# ~/.platformio/penv
# /Users/username/.platformio/penv
```

Prerequisites

1. Please remove existing PlatformIO Core **environment folder** if exists. See above command how to get path to environment folder.
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` or `pip2 install virtualenv` command.

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 *Settings*.

Install the latest PlatformIO from the develop branch:

```

# uninstall existing version
pip uninstall platformio

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

```

If you want to be up-to-date with the latest develop version of PlatformIO, then you need to re-install PlatformIO each time 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:

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

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

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

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

If you don't know the difference between the two, check out [this page](#).

Method 2

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):

```
ln -s ~/.platformio/pvenv/bin/platformio /usr/local/bin/platformio
ln -s ~/.platformio/pvenv/bin/pio /usr/local/bin/pio
ln -s ~/.platformio/pvenv/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

```
# 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](#):

Platform	Board	Framework
Atmel AVR	Arduino Uno	Arduino
Espressif 8266	NodeMCU 1.0 (ESP-12E Module)	Arduino
Teensy	Teensy 3.1 / 3.2	Arduino

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  at90usb1286 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 `nodemcu2`.

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*:

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

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

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

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

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

```
/**  
 * Blink  
 *  
 * Turns on an LED on for one second,  
 * then off for one second, repeatedly.  
 */  
#include "Arduino.h"  
  
#ifndef LED_BUILTIN  
define LED_BUILTIN 13  
#endif  
  
void setup()  
{  
    // initialize LED digital pin as an output.  
    pinMode(LED_BUILTIN, OUTPUT);  
}  
  
void loop()  
{  
    // turn the LED on (HIGH is the voltage level)  
    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

--no-ansi

Do not print ANSI control characters.

See also `PLATFORMIO_NO_ANSI` and `PLATFORMIO_FORCE_ANSI` environment variables.

--version

Show the version of PlatformIO

--help, -h

Show help for the available options and commands

```
$ 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

Contents

- *platformio account password*
 - *Usage*
 - *Description*

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

Contents

- *platformio account show*
 - *Usage*
 - *Description*
 - * *Options*

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

Contents

- *platformio account token*
 - *Usage*
 - *Description*
 - * *Options*

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

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

--json-output

Return the output in `JSON` format

platformio boards

Contents

- *platformio boards*
 - *Usage*
 - *Description*
 - * *Options*
 - *Examples*

Usage

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

Description

List pre-configured Embedded Boards

Options

--installed

List boards only from the installed platforms

--json-output

Return the output in [JSON](#) format

Examples

1. Show all available pre-configured embedded boards

```
$ platformio boards

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

```

2. Filter Arduino-based boards

```
$ platformio boards arduino

Platform: atmelavr
-----
ID          MCU      Frequency  Flash   RAM    Name
-----
btatmega168  atmega168  16MHz     14K    1K    Arduino BT ATmega168
btatmega328  atmega328p 16MHz     28K    2K    Arduino BT ATmega328
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
-----
ID          MCU      Frequency  Flash   RAM    Name
-----
frdm_k20d50m      mk20dx128vlh5 48MHz     128K   16K    Freescale Kinetis FRDM-
↳ K20D50M
frdm_k22f         mk22fn512vlh12 120MHz    512K   128K   Freescale Kinetis FRDM-
↳ K22F
...

```

(continues on next page)

(continued from previous page)

Platform: nordicnrf51						
ID	MCU	Frequency	Flash	RAM	Name	
wallBotBLE	nrf51822	16MHz	128K	16K	JKSoft Wallbot BLE	
nrf51_dk	nrf51822	32MHz	256K	32K	Nordic nRF51-DK	
...						
Platform: nxplpc						
ID	MCU	Frequency	Flash	RAM	Name	
blueboard_lpc11u24	lpc11u24	48MHz	32K	8K	BlueBoard-LPC11U24	
dipcortexm0	lpc11u24	50MHz	32K	8K	DipCortex M0	
lpc11u35 ↳ Board	lpc11u35	48MHz	64K	10K	EA LPC11U35 QuickStart	
...						
Platform: ststm32						
ID	MCU	Frequency	Flash	RAM	Name	
disco_f401vc	stm32f401vct6	84MHz	256K	64K	32F401CDISCOVERY	
nucleo_f030r8	stm32f030r8t6	48MHz	64K	8K	ST Nucleo F030R8	
...						

4. Filter boards which are based on ATmega168 MCU

\$ platformio boards atmega168						
Platform: atmelavr						
ID	MCU	Frequency	Flash	RAM	Name	
btagme168	atmega168	16MHz	14K	1K	Arduino BT ATmega168	
diecimilaatmega168 ↳ Diecimila ATmega168	atmega168	16MHz	14K	1K	Arduino Duemilanove or	
miniatmega168	atmega168	16MHz	14K	1K	Arduino Mini ATmega168	
atmegangatmega168 ↳ ATmega168	atmega168	16MHz	14K	1K	Arduino NG or older	
nanoatmega168	atmega168	16MHz	14K	1K	Arduino Nano ATmega168	
pro8MHzatmega168 ↳ ATmega168 (3.3V, 8 MHz)	atmega168	8MHz	14K	1K	Arduino Pro or Pro Mini	
pro16MHzatmega168 ↳ ATmega168 (5V, 16 MHz)	atmega168	16MHz	14K	1K	Arduino Pro or Pro Mini	
lilypadatmega168	atmega168	8MHz	14K	1K	LilyPad Arduino ATmega168	
168pa16m ↳ (Atmega168PA@16M, 5V)	atmega168p	16MHz	15K	1K	Microduino Core	
168pa8m ↳ (Atmega168PA@8M, 3.3V)	atmega168p	8MHz	15K	1K	Microduino Core	

5. Show boards by TI MSP430

\$ platformio boards timsp430
Platform: timsp430

(continues on next page)

(continued from previous page)

ID	MCU	Frequency	Flash	RAM	Name
lpmmsp430fr5739	msp430fr5739	16MHz	15K	1K	FraunchPad w/ msp430fr5739
lpmmsp430f5529 → (16MHz)	msp430f5529	16MHz	128K	1K	LaunchPad w/ msp430f5529
lpmmsp430f5529_25 → (25MHz)	msp430f5529	25MHz	128K	1K	LaunchPad w/ msp430f5529
lpmmsp430fr5969	msp430fr5969	8MHz	64K	1K	LaunchPad w/ msp430fr5969
lpmmsp430g2231 → (1MHz)	msp430g2231	1MHz	2K	128B	LaunchPad w/ msp430g2231
lpmmsp430g2452 → (16MHz)	msp430g2452	16MHz	8K	256B	LaunchPad w/ msp430g2452
lpmmsp430g2553 → (16MHz)	msp430g2553	16MHz	16K	512B	LaunchPad w/ msp430g2553

platformio ci

Contents

- *platformio ci*
 - *Usage*
 - *Description*
 - *Options*
 - *Examples*

Usage

```
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-

tional option `--project-option="targets=upload"`. What is more, you can specify custom upload port using `--project-option="upload_port=<port>"` option. See [`platformio ci --project-option`](#) for details.

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

```
> platformio debug

[Sun Apr 30 01:34:01 2017] Processing mzeropro (platform: atmelsam; debug_extra_cmds:_
↳b main.cpp:26; board: mzeropro; framework: arduino)
-----
↳-----  
Verbose mode can be enabled via `'-v, --verbose` option
Collected 26 compatible libraries
Looking for dependencies...
Project does not have dependencies
Compiling .pio/build/mzeropro/src/main.o
Compiling .pio/build/mzeropro/FrameworkArduinoVariant/variant.o
Compiling .pio/build/mzeropro/FrameworkArduino/IPAddress.o
Compiling .pio/build/mzeropro/FrameworkArduino/Print.o
Archiving .pio/build/mzeropro/libFrameworkArduinoVariant.a
Indexing .pio/build/mzeropro/libFrameworkArduinoVariant.a
...
Compiling .pio/build/mzeropro/FrameworkArduino/wiring_analog.o
Compiling .pio/build/mzeropro/FrameworkArduino/wiring_digital.o
Compiling .pio/build/mzeropro/FrameworkArduino/wiring_private.o
Compiling .pio/build/mzeropro/FrameworkArduino/wiring_shift.o
Archiving .pio/build/mzeropro/libFrameworkArduino.a
Indexing .pio/build/mzeropro/libFrameworkArduino.a
Linking .pio/build/mzeropro/firmware.elf
Calculating size .pio/build/mzeropro/firmware.elf
Building .pio/build/mzeropro/firmware.bin
text      data      bss      dec      hex filename
11512       256     1788    13556    34f4 .pio/build/mzeropro/firmware.elf
===== [SUCCESS] Took 7.82 seconds =====
```

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

```
> 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)

(continued from previous 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: 0x000028f4 msp: 0x20002c00
target halted due to debug-request, current mode: Thread
xPSR: 0x81000000 pc: 0x000028f4 msp: 0x20002c00
Breakpoint 2 at 0x413a: file src/main.cpp, line 26.
```

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.

--logical

List available logical devices.

--mdns

List multicast DNS services.

--json-output

Return the output in [JSON](#) format.

Examples

1. Unix OS

```
$ 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

```
$ 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

```
$ 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)

(continued from previous 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*

```
$ 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
```

(continues on next page)

(continued from previous page)

--quiet -h, --help	control miniterm (menu), default=20 (DEC) Diagnostics: suppress non-error messages, default=Off Show this message and exit.
-----------------------	---

2. Communicate with serial device and print help inside terminal

```
$ 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 0.0.0.0.

--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

```
> 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
↪ embedded board
`platformio run --target clean` - clean project (remove compiled files)
```

2. Initialize new project in a specified directory

```
> 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

```
> 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*

```
> 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)

(continued from previous page)

```
platformio lib --help  
platformio lib COMMAND --help
```

Options

-d, --storage-dir

Manage custom library storage. It can be used later for the `lib_extra_dirs` option from “*platformio.ini*” (*Project Configuration File*). Multiple options are allowed.

-g, --global

Manage global PlatformIO’s library storage (“`core_dir/lib`”) where *Library Dependency Finder (LDF)* will look for dependencies by default.

-e, --environment

Manage libraries for the specific project build environments declared in “*platformio.ini*” (*Project Configuration File*). Works for `--storage-dir` which is valid PlatformIO project.

Demo

Commands

platformio lib builtin

Contents

- *platformio lib builtin*
 - *Usage*
 - *Description*
 - *Options*
 - *Examples*

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

```
> 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
```

(continues on next page)

(continued from previous page)

```
...
framework-arduinoespressif32
*****
SPI
===
Enables the communication with devices that use the Serial Peripheral Interface (SPI) ↳
↳ Bus. For all Arduino boards, BUT Arduino DUE.

Version: 1.0
Homepage: http://arduino.cc/en/Reference/SPI
Keywords: signal, input, output
Compatible frameworks: arduino
Compatible platforms:
Authors: Hristo Gochkov

...
framework-arduinoespressif8266
*****
ArduinoOTA
=====
Enables Over The Air upgrades, via wifi and espota.py UDP request/TCP download.

Version: 1.0
Keywords: communication
Compatible frameworks: arduino
Compatible platforms: espressif8266
Authors: Ivan 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

...
framework-arduinointel
*****
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: *
```

(continues on next page)

(continued from previous page)

```

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)

(continued from previous 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...]
```

(continues on next page)

(continued from previous page)

```

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

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

# install dependencies for the specific project build environment
# (run it from a project root where is located "platformio.ini")
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>@<version range>
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.

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`

PlatformIO supports installing from local directory or archive. Need to use `file://` prefix before local path. Also, directory or archive should contain `.library.json` manifest (see [library.json](#)).

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

Storage Options

See base options for [Library Manager](#).

Options

--save

Save installed libraries into the “[platformio.ini](#)” (*Project Configuration File*) dependency list (`lib_deps`).

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

-s, --silent

Suppress progress reporting

--interactive

Allow one to make a choice for all prompts

-f, --force

Reinstall/redownload library if it exists

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:

- `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

```
> 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

```
> 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

```
> 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

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

Library Storage: /storage/dir/...
```

(continues on next page)

(continued from previous page)

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

Copyright (C) 2005-2016 Matt Mackall and others
This is free software; see the source for copying conditions. There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
requesting all changes
adding changesets
adding manifests
adding file changes
added 9 changesets with 18 changes to 6 files
updating to branch default
2 files updated, 0 files merged, 0 files removed, 0 files unresolved
TextLCD @ 308d188a2d3a has been successfully installed!
```

5. Install from archive using URL

```
> 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

```
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]
```

(continues on next page)

(continued from previous page)

```
# 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
```

(continues on next page)

(continued from previous page)

```

Authors: Benoit Blanchon

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

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

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

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

Blynk
=====
# ID: 415
Build a smartphone app for your project in minutes. Blynk allows creating IoT, ↵
↳ solutions easily. It supports WiFi, BLE, Bluetooth, Ethernet, GSM, USB, Serial. ↵
↳ Works with many boards like ESP8266, ESP32, Arduino UNO, Nano, Due, Mega, Zero, ↵
↳ 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

```

(continues on next page)

(continued from previous page)

```
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)

(continued from previous page)

```
#ID: 76
Arduino library to facilitate time zone conversions and automatic daylight saving,_
↪(summer) 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, NT7534, 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, espressif8266, intel_arc32, microchippic32,_
↪nordicnrf51, teensy, timsp430
Authors: oliver

USB-Host-Shield-20
=====
#ID: 59
Revision 2.0 of MAX3421E-based USB Host Shield Library

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

platformio lib register

Contents

- *platformio lib register*
 - *Usage*
 - *Description*
 - *Examples*

Usage

```
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

```
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

```
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:

Operator	Description
+	A leading or trailing plus sign indicates that this word must be present in library fields (see above) that is returned.
-	A leading or trailing minus sign indicates that this word must not be present in any of the libraries that are returned.
(no operator)	By default (when neither + nor - is specified), the word is optional, but the libraries that contain it are rated higher.
> <	These two operators are used to change a word's contribution to the relevance value that is assigned to a library. The > operator increases the contribution and the < operator decreases it.
()	Parentheses group words into subexpressions. Parenthesized groups can be nested.
~	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.
*	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.
"	A phrase that is enclosed within double quote ("") characters matches only libraries that contain the phrase literally, as it was typed.

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.

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

```
> 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

```
> 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)

(continued from previous 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,...
```

(continues on next page)

(continued from previous page)

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)

(continued from previous 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`

```
> 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)

(continued from previous page)

Authors: Adrian McEwen

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

8. Search for the all sensor libraries excluding temperature.

```
> 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*

- *Description*
- *Options*
- *Examples*

Usage

```
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

```
> 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
Manifest: https://raw.githubusercontent.com/ivankravets/pubsubclient/patch-2/library.json
Homepage: http://pubsubclient.knolleary.net
Repository: https://github.com/knolleary/pubsubclient.git

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

Keywords
-----
ether
```

(continues on next page)

(continued from previous 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

```

platformio lib stats

Contents

- *platformio lib stats*
 - *Usage*
 - *Description*
 - *Options*
 - *Examples*

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
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
ArduinoSTL     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)
```

(continued from previous page)

GroveEncoder ↳ GroveEncoder	12 hours ago	https://platformio.org/lib/show/1382/
RF24G ↳ RF24G	12 hours ago	https://platformio.org/lib/show/1381/
Sim800L Library Revised ↳ Sim800L%20Library%20Revised	12 hours ago	https://platformio.org/lib/show/1380/
DS3231 ↳ DS3231	a day ago	https://platformio.org/lib/show/1379/
ArduboyPlaytune ↳ ArduboyPlaytune	4 days ago	https://platformio.org/lib/show/1378/
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	

(continues on next page)

(continued from previous page)

processing	https://platformio.org/lib/search?query=keyword%3Aprocessing
storage	https://platformio.org/lib/search?query=keyword%3Astorage
radio	https://platformio.org/lib/search?query=keyword%3Aradio
web	https://platformio.org/lib/search?query=keyword%3Aweb
accelerometer	https://platformio.org/lib/search?query=keyword%3Aaccelerometer
↳ %3Aaccelerometer	
wireless	https://platformio.org/lib/search?query=keyword%3Awireless
protocol	https://platformio.org/lib/search?query=keyword%3Aprotocol
server	https://platformio.org/lib/search?query=keyword%3Aserver
wi-fi	https://platformio.org/lib/search?query=keyword%3Awi-fi
ethernet	https://platformio.org/lib/search?query=keyword%3Aethernet
mbed	https://platformio.org/lib/search?query=keyword%3Ambed
openag	https://platformio.org/lib/search?query=keyword%3Aopenag
led	https://platformio.org/lib/search?query=keyword%3Aled
esp8266	https://platformio.org/lib/search?query=keyword%3Aesp8266
humidity	https://platformio.org/lib/search?query=keyword%3Ahumidity
time	https://platformio.org/lib/search?query=keyword%3Atime
iot	https://platformio.org/lib/search?query=keyword%3Aiot
json	https://platformio.org/lib/search?query=keyword%3Ajson
timer	https://platformio.org/lib/search?query=keyword%3Atimer
client	https://platformio.org/lib/search?query=keyword%3Aclient
driver	https://platformio.org/lib/search?query=keyword%3Adriver
button	https://platformio.org/lib/search?query=keyword%3Abutton
mbed-official	https://platformio.org/lib/search?query=keyword%3Ambed-official
↳ official	
FEATURED: TODAY	

Name	Url
-----	-----
↳ -----	-----
↳ -----	-----
PubSubClient	https://platformio.org/lib/show/89/PubSubClient
Adafruit Unified Sensor	https://platformio.org/lib/show/31/Adafruit%20Unified%20Sensor
ESPAsyncUDP	https://platformio.org/lib/show/359/ESPAsyncUDP
NtpClientLib	https://platformio.org/lib/show/727/NtpClientLib
Embedis	https://platformio.org/lib/show/408/Embedis
Blynk	https://platformio.org/lib/show/415/Blynk
SimpleTimer	https://platformio.org/lib/show/419/SimpleTimer
Adafruit DHT Unified	https://platformio.org/lib/show/18/Adafruit%20DHT%20Unified
RTClib	https://platformio.org/lib/show/83/RTClib
FEATURED: WEEK	

Name	Url
-----	-----
↳ -----	-----
↳ -----	-----
DHT sensor library	https://platformio.org/lib/show/19/DHT%20sensor%20library
Adafruit Unified Sensor	https://platformio.org/lib/show/31/Adafruit%20Unified%20Sensor
ESPAsyncWebServer	https://platformio.org/lib/show/306/ESPAsyncWebServer
Adafruit GFX Library	https://platformio.org/lib/show/13/Adafruit%20GFX%20Library
I2Cdevlib-Core	https://platformio.org/lib/show/11/I2Cdevlib-Core

(continues on next page)

(continued from previous page)

TimeAlarms	https://platformio.org/lib/show/68/TimeAlarms
PubSubClient	https://platformio.org/lib/show/89/PubSubClient
Timer	https://platformio.org/lib/show/75/Timer
esp8266_mdns	https://platformio.org/lib/show/1091/esp8266_mdns
FEATURED: MONTH	

Name	Url
-----	-----
-----	-----
-----	-----
ArduinoJson	https://platformio.org/lib/show/64/ArduinoJson
DHT sensor library	https://platformio.org/lib/show/19/DHT%20sensor%20library
Adafruit Unified Sensor	https://platformio.org/lib/show/31/Adafruit%20Unified%20Sensor
PubSubClient	https://platformio.org/lib/show/89/PubSubClient
OneWire	https://platformio.org/lib/show/1/OneWire
ESPAsyncTCP	https://platformio.org/lib/show/305/ESPAsyncTCP
Time	https://platformio.org/lib/show/44/Time
DallasTemperature	https://platformio.org/lib/show/54/DallasTemperature
ESPAsyncWebServer	https://platformio.org/lib/show/306/ESPAsyncWebServer
WiFiManager	https://platformio.org/lib/show/567/WifiManager

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...]
```

(continues on next page)

(continued from previous page)

```
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](#)

– *Examples*

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)

(continued from previous 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

```
=====
Espressif IoT Development Framework. Official development framework for ESP32.
```

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.
```

(continues on next page)

(continued from previous page)

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

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

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

```
wiringpi ~ WiringPi
=====
WiringPi is a GPIO access library written in C for the BCM2835 used in the Raspberry_
↪Pi._  
→It's designed to be familiar to people who have used the Arduino "wiring" system.
```

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

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>
```

(continues on next page)

(continued from previous page)

```
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

```
> 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

```
> 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!
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

```
> 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'...
remote: Counting objects: 176, done.
remote: Compressing objects: 100% (55/55), done.
remote: Total 176 (delta 114), reused 164 (delta 109), pack-reused 0
Receiving objects: 100% (176/176), 38.86 KiB | 0 bytes/s, done.
Resolving deltas: 100% (114/114), done.
Checking connectivity... done.
Submodule 'examples/arduino-external-libs/lib/OneWire' (https://github.com/
→PaulStoffregen/OneWire.git) registered for path 'examples/arduino-external-libs/lib/
→OneWire'
```

(continues on next page)

(continued from previous page)

```

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 industrys 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-arduinomas, framework-mbed, framework-simba, toolchain-
gccarmnoneabi, 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 industrys most code-efficient architecture,
for C and assembly programming.

Home: https://platformio.org/platforms/atmelavr
Packages: toolchain-atmelavr, framework-arduinoavr, framework-simba, tool-avrdude,
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-gccarmnoneabi, framework-arduinomas, framework-simba, tool-
openocd, framework-mbed, tool-avrdude, tool-bossac

espresif32 ~ 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/espresif32
Packages: toolchain-xtensa32, framework-simba, framework-arduinoespresif32,
framework-pumabaa, framework-espifdf, tool-esptoolpy

espresif8266 ~ Espressif 8266
```

(continues on next page)

(continued from previous 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

freescalekinetis ~ 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 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.

Home: https://platformio.org/platforms/titiva
Packages: ldscripts, framework-libopencm3, toolchain-gccarmnoneabi, tool-lm4flash,
↳ framework-energiativa
```

```
> 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-gccarmnoneabi, framework-arduinomas, framework-simba, tool-
↳ openocd, framework-mbed, ldscripts, tool-bossac

freescalekinetis ~ Freescale Kinetis
=====
```

(continues on next page)

(continued from previous 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/freescalekinetics>

Packages: framework-mbed, toolchain-gccarmnoneabi

nordicnrf51 ~ Nordic nRF51

=====

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

Home: <https://platformio.org/platforms/nordicnrf51>

Packages: framework-mbed, tool-rfdloader, toolchain-gccarmnoneabi, framework-arduinonordicnrf51

nxplpc ~ NXP LPC

=====

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

Home: <https://platformio.org/platforms/nxplpc>

Packages: framework-mbed, toolchain-gccarmnoneabi

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-gccarmnoneabi

ststm32 ~ ST STM32

=====

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

Home: <https://platformio.org/platforms/ststm32>

Packages: framework-libopencm3, toolchain-gccarmnoneabi, 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 (continues on next page) PC **or** Macintosh **with** a USB port.

(continued from previous page)

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

platformio platform show

Contents

- *platformio platform show*
 - *Usage*
 - *Description*
 - *Examples*

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 industrys 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
```

(continues on next page)

(continued from previous page)

```
Description: avr-gcc
Url: http://www.atmel.com/products/microcontrollers/avr/32-bitavruc3.aspx?tab=tools
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-arduinopespressif8266 @ 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-gccarmnoneabi @ 1.40804.0: [Up-to-date]
```

platformio remote

Helper command for *PIO Remote*.

To print all available commands and options use:

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

# run command on the specified PIO Remote Agents
platformio remote --agent NAME_1 --agent NAME_N COMMAND
```

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

```
> 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

```
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: 2016-11-11 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)

(continued from previous page)

```
/dev/cu.SLAB_USBtoUART
-----
Hardware ID: USB VID:PID=10C4:EA60 SER=0001 LOCATION=253-1.3.2
Description: CP2102 USB to UART Bridge Controller - CP2102 USB to UART Bridge
    ↳ Controller
/dev/cu.usbmodem589561
-----
Hardware ID: USB VID:PID=16C0:0483 SER=589560 LOCATION=250-1.4.1
Description: USB Serial
```

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
```

(continues on next page)

(continued from previous page)

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

2. Communicate with serial device and print help inside terminal

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

(continued from previous 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
=====
===== [SUCCESS] Took 10.01 seconds =====
===== [SUMMARY] =====
Environment nodemcuv2    [SKIP]
Environment uno_pic32    [SKIP]
Environment teensy31     [SKIP]
Environment uno          [SUCCESS]
=====
===== [SUCCESS] Took 10.01 seconds =====
Uploading firmware remotely
[Wed Oct 26 19:35:20 2016] Processing uno (platform: atmelavr, board: uno, framework: arduino)
-----
-----
Verbose mode can be enabled via `"-v, --verbose` option
Looking for upload port...
Auto-detected: /dev/cu.usbmodemFA1431
Uploading .pio/build/uno/firmware.hex
avrdude: AVR device initialized and ready to accept instructions
Reading | #####| #####| #####| #####| #####| #####| #####| #####| #####| 100% 0.00s
avrdude: Device signature = 0x1e950f
avrdude: reading input file ".pio/build/uno/firmware.hex"
avrdude: writing flash (2622 bytes):
Writing | #####| #####| #####| #####| #####| #####| #####| #####| #####| 100% 0.43s
avrdude: 2622 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 2622 bytes
avrdude: reading on-chip flash data:
Reading | #####| #####| #####| #####| #####| #####| #####| #####| #####| 100% 0.34s
avrdude: verifying ...
avrdude: 2622 bytes of flash verified
avrdude done. Thank you.
=====
===== [SUCCESS] Took 3.04 seconds =====

```

(continues on next page)

(continued from previous page)

```
===== [SUMMARY] =====
Environment nodemcuv2    [SKIP]
Environment uno_pic32     [SKIP]
Environment teensy31      [SKIP]
Environment uno           [SUCCESS]
===== [SUCCESS] Took 3.04 seconds =====
```

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*).

Pattern	Meaning
*	matches everything
?	matches any single character
[seq]	matches any character in seq
[!seq]	matches any character not in seq

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

Contents

- [`platformio remote update`](#)

- *Usage*
- *Description*
- *Options*
- *Examples*

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
-----
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 framework-arduinomsp430 @ 1.10601.0: [Up-to-date]
Updating tool-scons @ 2.4.1: [Up-to-date]

Platform freescalekinetis
-----
Updating freescalekinetis @ 0.0.0: [Up-to-date]
Updating framework-mbed @ 1.121.1: [Up-to-date]
Updating toolchain-gccarmnoneabi @ 1.40804.0: [Up-to-date]
Updating tool-scons @ 2.4.1: [Up-to-date]
```

(continues on next page)

(continued from previous page)

```

Platform ststm32
-----
Updating ststm32 @ 0.0.0: [Up-to-date]
Updating framework-libopencm3 @ 1.1.0: [Up-to-date]
Updating toolchain-gccarmnoneabi @ 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
-----
Updating linux_x86_64 @ 0.0.0: [Up-to-date]
Updating toolchain-gcclinux64 @ 1.40801.0: [Up-to-date]
Updating tool-scons @ 2.4.1: [Up-to-date]

Platform windows_x86
-----
Updating windows_x86 @ 0.0.0: [Up-to-date]
Updating toolchain-gccmingw32 @ 1.40800.0: [Up-to-date]
Updating tool-scons @ 2.4.1: [Up-to-date]

Platform teensy
-----
Updating teensy @ 0.0.0: [Up-to-date]
Updating framework-arduinoteensy @ 1.128.0: [Up-to-date]
Updating tool-teensy @ 1.1.0: [Up-to-date]
Updating framework-mbed @ 1.121.1: [Up-to-date]
Updating tool-scons @ 2.4.1: [Up-to-date]
Updating toolchain-atmelavr @ 1.40801.0: [Up-to-date]
Updating toolchain-gccarmnoneabi @ 1.40804.0: [Up-to-date]

```

(continues on next page)

(continued from previous page)

```
Platform nordicnrf51
-----
Updating nordicnrf51 @ 0.0.0: [Up-to-date]
Updating toolchain-gccarmnoneabi @ 1.40804.0: [Up-to-date]
Updating framework-arduino nordicnrf51 @ 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-gccarmnoneabi @ 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-gccarmnoneabi @ 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-gccarmnoneabi @ 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-arduino microchippic32 @ 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-gcclinux32 @ 1.40801.0: [Up-to-date]
Updating tool-scons @ 2.4.1: [Up-to-date]

Platform intel_arc32
-----
Updating intel_arc32 @ 0.0.0: [Up-to-date]
Updating framework-arduino intel @ 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)

(continued from previous page)

```
-----
Updating nxplpc @ 0.0.0:      [Up-to-date]
Updating framework-mbed @ 1.121.1:  [Up-to-date]
Updating toolchain-gccarmnoneabi @ 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-gccarmlinuxgnueabi @ 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

[Wed Sep  7 15:48:58 2016] Processing uno (platform: atmelavr, board: uno, framework:arduino)
-----
-----  

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

(continues on next page)

(continued from previous 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:_
↳nodemcu, framework: arduino)
-----
↳-----
Verbose mode can be enabled via `‐v, --verbose` option
Collected 34 compatible libraries
Looking for dependencies...
Project does not have dependencies
Compiling .pio/build/nodemcu/src/main.o
Archiving .pio/build/nodemcu/libFrameworkArduinoVariant.a
Indexing .pio/build/nodemcu/libFrameworkArduinoVariant.a
Compiling .pio/build/nodemcu/FrameworkArduino/Esp.o
Compiling .pio/build/nodemcu/FrameworkArduino/FS.o
Compiling .pio/build/nodemcu/FrameworkArduino/HardwareSerial.o
...
Archiving .pio/build/nodemcu/libFrameworkArduino.a
Indexing .pio/build/nodemcu/libFrameworkArduino.a
Linking .pio/build/nodemcu/firmware.elf
Calculating size .pio/build/nodemcu/firmware.elf
text      data      bss      dec      hex   filename
221240       888     29400   251528   3d688 .pio/build/nodemcu/firmware.elf
Building .pio/build/nodemcu/firmware.bin
=====
[SUCCESS] Took 6.43 seconds =====

[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...

```

(continues on next page)

(continued from previous page)

```

Project does not have dependencies
Compiling .pio/build/teensy31/src/main.o
Compiling .pio/build/teensy31/FrameworkArduino/AudioStream.o
Compiling .pio/build/teensy31/FrameworkArduino/DMAChannel.o
...
Compiling .pio/build/teensy31/FrameworkArduino/yield.o
Archiving .pio/build/teensy31/libFrameworkArduino.a
Indexing .pio/build/teensy31/libFrameworkArduino.a
Linking .pio/build/teensy31/firmware.elf
Calculating size .pio/build/teensy31/firmware.elf
text      data      bss      dec      hex filename
11288       168     2288    13744    35b0 .pio/build/teensy31/firmware.elf
Building .pio/build/teensy31/firmware.hex
===== [SUCCESS] Took 5.36 seconds =====

[Wed Sep 7 15:49:12 2016] Processing lpmesp430g2553 (platform: timsp430, build_flags:_
→ -D LED_BUILTIN=RED_LED, board: lpmesp430g2553, framework: arduino)
-----
→ -----
Verbose mode can be enabled via `'-v, --verbose` option
Collected 29 compatible libraries
Looking for dependencies...
Project does not have dependencies
Compiling .pio/build/lpmesp430g2553/src/main.o
Compiling .pio/build/lpmesp430g2553/FrameworkAnergia/HardwareSerial.o
Compiling .pio/build/lpmesp430g2553/FrameworkAnergia/IPAddress.o
...
Compiling .pio/build/lpmesp430g2553/FrameworkAnergia/wiring_digital.o
Compiling .pio/build/lpmesp430g2553/FrameworkAnergia/wiring_pulse.o
Compiling .pio/build/lpmesp430g2553/FrameworkAnergia/wiring_shift.o
Archiving .pio/build/lpmesp430g2553/libFrameworkAnergia.a
Indexing .pio/build/lpmesp430g2553/libFrameworkAnergia.a
Linking .pio/build/lpmesp430g2553/firmware.elf
Calculating size .pio/build/lpmesp430g2553/firmware.elf
text      data      bss      dec      hex filename
820        0       20     840     348 .pio/build/lpmesp430g2553/firmware.elf
Building .pio/build/lpmesp430g2553/firmware.hex
===== [SUCCESS] Took 2.34 seconds =====

```

2. Process specific environment

```

> platformio run -e nodemcu -e teensy31

[Wed Sep 7 15:49:01 2016] Processing nodemcu (platform: espressif8266, board:_ 
→ nodemcu, framework: arduino)
-----
→ -----
Verbose mode can be enabled via `'-v, --verbose` option
Collected 34 compatible libraries
Looking for dependencies...
Project does not have dependencies
Compiling .pio/build/nodemcu/src/main.o
Archiving .pio/build/nodemcu/libFrameworkArduinoVariant.a
Indexing .pio/build/nodemcu/libFrameworkArduinoVariant.a
Compiling .pio/build/nodemcu/FrameworkArduino/Esp.o
Compiling .pio/build/nodemcu/FrameworkArduino/FS.o
Compiling .pio/build/nodemcu/FrameworkArduino/HardwareSerial.o

```

(continues on next page)

(continued from previous 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
Calculating size .pio/build/teensy31/firmware.elf
text      data     bss     dec     hex filename
11288      168   2288   13744   35b0 .pio/build/teensy31/firmware.elf
Building .pio/build/teensy31/firmware.hex
===== [SUCCESS] Took 5.36 seconds =====
```

3. Process specific target (clean project)

```
> platformio run -t clean
[Wed Sep 7 15:53:26 2016] Processing uno (platform: atmelavr, board: uno, framework: arduino)
-----
→-----  

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/HardwareSerial0.o
Removed .pio/build/uno/FrameworkArduino/HardwareSerial1.o
Removed .pio/build/uno/FrameworkArduino/HardwareSerial2.o
Removed .pio/build/uno/FrameworkArduino/HardwareSerial3.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)

(continued from previous 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 =====

[Wed Sep 7 15:53:27 2016] Processing nodemcu (platform: espressif8266, board: nodemcu, framework: arduino)
-----
→
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 =====

[Wed Sep 7 15:53:27 2016] Processing teensy31 (platform: teensy, board: teensy31, framework: arduino)
-----
→
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 =====

[Wed Sep 7 15:53:28 2016] Processing lpmesp430g2553 (platform: timsp430, build_flags: -D LED_BUILTIN=RED_LED, board: lpmesp430g2553, framework: energia)
-----
→
Removed .pio/build/lpmesp430g2553/firmware.elf
Removed .pio/build/lpmesp430g2553/firmware.hex
Removed .pio/build/lpmesp430g2553/libFrameworkAnergia.a
Removed .pio/build/lpmesp430g2553/FrameworkAnergia/atof.o
...
Removed .pio/build/lpmesp430g2553/FrameworkAnergia/avr/dtostrf.o
Removed .pio/build/lpmesp430g2553/src/main.o

```

(continues on next page)

(continued from previous page)

```
Done cleaning
===== [SUCCESS] Took 0.49 seconds =====
```

4. Mix environments and targets

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

[Wed Sep 7 15:55:11 2016] Processing uno (platform: atmelavr, board: uno, framework:arduino)
-----
→-----  

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

avrduude: AVR device initialized and ready to accept instructions

Reading | ##### | 100% 0.01s

avrduude: Device signature = 0x1e950f
avrduude: reading input file ".pio/build/uno/firmware.hex"
avrduude: writing flash (1034 bytes):

Writing | ##### | 100% 0.18s

avrduude: 1034 bytes of flash written
avrduude: verifying flash memory against .pio/build/uno/firmware.hex:
avrduude: load data flash data from input file .pio/build/uno/firmware.hex:
avrduude: input file .pio/build/uno/firmware.hex contains 1034 bytes
avrduude: reading on-chip flash data:

Reading | ##### | 100% 0.15s

avrduude: verifying ...
avrduude: 1034 bytes of flash verified

avrduude: safemode: Fuses OK (H:00, E:00, L:00)

avrduude done. Thank you.
```

(continues on next page)

(continued from previous page)

```
===== [SUCCESS] Took 4.14 seconds =====
```

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, “espressif”, “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

Name	Value [Default]	Description
<hr/>		
auto_update_libraries	No	Automatically update libraries (Yes/ No)
auto_update_platforms	No	Automatically update platforms (Yes/ No)
check_libraries_interval	7	Check for the library updates interval (days)
check_platformio_interval	3	Check for the new PlatformIO interval (days)
check_platforms_interval	7	Check for the platform updates interval (days)
enable_cache	Yes	Enable caching for API requests and Library Manager
strict_ssl	No	Strict SSL for PlatformIO Services
enable_telemetry	Yes	Telemetry service?#enable-telemetry> (Yes/No)
force_verbose	No	Force verbose output when processing environments
projects_dir	~/Documents/PlatformIO/Projects	Default location for PlatformIO projects (PIO Home)

2. Show specified setting

\$ platformio settings get auto_update_platforms	Name	Value [Default]	Description
<hr/>			
auto_update_platforms	Yes		Automatically update platforms (Yes/ No)

platformio settings set

Usage

```
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!
Name          Value [Default]  Description
-----
check_platformio_interval      1 [3]           Check for the new PlatformIO_
                                ↵interval (days)
```

platformio settings reset

Usage

```
platformio settings reset
```

Description

Reset settings to default

Examples

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

Name          Value [Default]  Description
-----
auto_update_libraries      No           Automatically update libraries (Yes/
                                ↵No)
auto_update_platforms     No           Automatically update platforms (Yes/
                                ↵No)
check_libraries_interval   7            Check for the library updates_
                                ↵interval (days)
check_platformio_interval  3             Check for the new PlatformIO_
                                ↵interval (days)
check_platforms_interval   7             Check for the platform updates_
                                ↵interval (days)
enable_cache               Yes          Enable caching for API requests and_
                                ↵Library Manager
strict_ssl                 No           Enable SSL for PlatformIO Services
enable_telemetry            Yes          Telemetry service?#enable-telemetry>
                                ↵ (Yes/No)
```

(continues on next page)

(continued from previous page)

force_verbose	No	Force verbose output when <code>_</code>
↳ processing environments		
projects_dir		~/Documents/PlatformIO/Projects Default location <code>for _</code>
↳ PlatformIO projects (PIO Home)		

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*).

Pattern	Meaning
*	matches everything
?	matches any single character
[seq]	matches any character in seq
[!seq]	matches any character not in seq

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*).

Pattern	Meaning
*	matches everything
?	matches any single character
[seq]	matches any character in seq
[!seq]	matches any character not in seq

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

```
> 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)

(continued from previous page)

```

Updating linux_x86_64 @ 0.0.0: [Up-to-date]
Updating toolchain-gcclinux64 @ 1.40801.0: [Up-to-date]
Updating tool-scons @ 2.4.1: [Up-to-date]

Platform windows_x86
-----
Updating windows_x86 @ 0.0.0: [Up-to-date]
Updating toolchain-gccmingw32 @ 1.40800.0: [Up-to-date]
Updating tool-scons @ 2.4.1: [Up-to-date]

Platform teensy
-----
Updating teensy @ 0.0.0: [Up-to-date]
Updating framework-arduinoteensy @ 1.128.0: [Up-to-date]
Updating tool-teensy @ 1.1.0: [Up-to-date]
Updating framework-mbed @ 1.121.1: [Up-to-date]
Updating tool-scons @ 2.4.1: [Up-to-date]
Updating toolchain-atmelavr @ 1.40801.0: [Up-to-date]
Updating toolchain-gccarmnoneeabi @ 1.40804.0: [Up-to-date]

Platform nordicnrf51
-----
Updating nordicnrf51 @ 0.0.0: [Up-to-date]
Updating toolchain-gccarmnoneeabi @ 1.40804.0: [Up-to-date]
Updating framework-arduinonordicnrf51 @ 1.20302.0: [Up-to-date]
Updating framework-mbed @ 1.121.1: [Up-to-date]
Updating tool-scons @ 2.4.1: [Up-to-date]

Platform titiva
-----
Updating titiva @ 0.0.0: [Up-to-date]
Updating framework-libopencm3 @ 1.1.0: [Up-to-date]
Updating toolchain-gccarmnoneeabi @ 1.40804.0: [Up-to-date]
Updating framework-energiativa @ 1.17.0: [Up-to-date]
Updating tool-scons @ 2.4.1: [Up-to-date]

Platform atmelsam
-----
Updating atmelsam @ 0.0.0: [Up-to-date]
Updating toolchain-gccarmnoneeabi @ 1.40804.0: [Up-to-date]
Updating tool-openocd @ 1.900.0: [Up-to-date]
Updating framework-mbed @ 1.121.1: [Up-to-date]
Updating tool-scons @ 2.4.1: [Up-to-date]
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]

```

(continues on next page)

(continued from previous page)

```

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-gcclinux32 @ 1.40801.0: [Up-to-date]
Updating tool-scons @ 2.4.1: [Up-to-date]

Platform intel_arc32
-----
Updating intel_arc32 @ 0.0.0: [Up-to-date]
Updating framework-arduinointel @ 1.10006.0: [Up-to-date]
Updating tool-arduino101load @ 1.124.0: [Up-to-date]
Updating toolchain-intelarc32 @ 1.40805.0: [Up-to-date]
Updating tool-scons @ 2.4.1: [Up-to-date]

Platform nxplpc
-----
Updating nxplpc @ 0.0.0: [Up-to-date]
Updating framework-mbed @ 1.121.1: [Up-to-date]
Updating toolchain-gccarmnoneeabi @ 1.40804.0: [Up-to-date]
Updating tool-scons @ 2.4.1: [Up-to-date]

Platform linux_arm
-----
Updating linux_arm @ 0.0.0: [Up-to-date]
Updating toolchain-gccarmlinuxgnueabi @ 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

```

(continues on next page)

(continued from previous page)

```
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 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

Welcome & Project Manager

 < > ⌂ Have an account? Log in

Welcome to PlatformIO



Quick Access

- + New Project
- Import Arduino Project
- Open Project
- Project Examples

Home 0.6.0 · Core 3.5.0

Web · Open Source · Get Started · Docs · News · Community · Report an Issue · Donate · Contact

Recent Projects

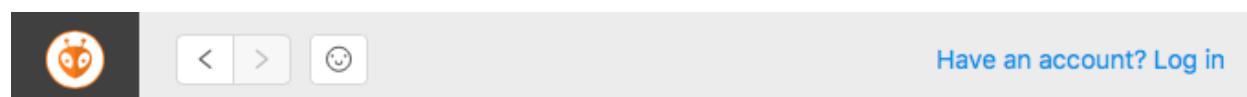
Search project...

Name	Boards	Modified	Action
debug/vscode	Arduino M0 Pro (Programming/Debug Port)	14 hours ago	Hide Open
unit-testing/calculator	Arduino Uno , NodeMCU 1.0 (ESP-12E Module)	18 hours ago	Hide Open
examples/wiring-blink	Arduino Uno , NodeMCU 0.9 (ESP-12 Module) , Teensy 3.1 / 3.2 , TI LaunchPad MSP- EXP430G2553LP	23 hours ago	Hide Open
debug/mbed	ST Nucleo L053RB	12 days ago	Hide Open

1.4. PlatformIO Home If you enjoy using PlatformIO, please star our projects on GitHub! 123

★ PlatformIO Core ★

Library Manager



Home



Account



Libraries



Boards



Platforms



Devices

[Registry](#)

[Installed](#)

[Built-in](#)

[Updates](#) 3

[Have an account? Log in](#)

Search libraries

[tft display](#)

[header:RH_ASK.h](#)

[keyword:mqtt](#)

[framework:mbed](#)

[more...](#)

Recently

[All Libraries](#)

[Register](#)

[Advanced](#)

Updated

[U8g2](#) 3 hours ago

[esp8266ndn](#) 3 hours ago

[Adafruit_BME280_Library](#) 12 hours ago

[IHCSoapClient](#) 17 hours ago

[RapidJSON](#) 20 hours ago

Added

[esp8266ndn](#) 3 hours ago

[Cryptosuite](#) 1 day ago

[CryptoC](#) 1 day ago

[Crypto](#) 1 day ago

[NTPtimeESP](#) 1 day ago

Keywords

[sha1](#)

[hmac](#)

[upnp](#)

[smartthings](#)

[ssdp](#)

Popular Tags

[display](#)

[communication](#)

[sensors](#)

[control](#)

[device](#)

[lcd](#)

[graphics](#)

[tft](#)

[oled](#)

[displaycore](#)

[glcd](#)

[other](#)

[input](#)

[signal](#)

[output](#)

[data](#)

[font](#)

[sensor](#)

[i2c](#)

[spi](#)

[timing](#)

[esp8266](#)

[processing](#)

[storage](#)

[serial](#)

[wifi](#)

[temperature](#)

[http](#)

[arduino](#)

[iot](#)

[rf](#)

[led](#)

[radio](#)

[web](#)

[i2cdevlib](#)

[ethernet](#)

[uncategorized](#)

[time](#)

[timer](#)

[wireless](#)

[mqtt](#)

[mbed](#)

[server](#)

[protocol](#)

[accelerometer](#)

[wi-fi](#)

[button](#)

[rtc](#)

[humidity](#)

[rest](#)

Trending

1.4. PlatformIO Home

125

Today

Week

Month

[SerialESP8266wifi](#)

[PubSubClient](#)

[ArduinoJson](#)

Board Explorer

The screenshot shows the PlatformIO Board Explorer interface. On the left is a dark sidebar with icons for Home, Account (with a red notification badge), Libraries (with a red notification badge), Boards (selected), Platforms, and Devices. The main area has a header with navigation buttons and a 'Have an account? Log in' link. Below is a section titled 'Board Explorer' with a count of 469 boards. It contains a message about supporting over 400 boards and a search bar. A filter bar at the top right includes 'Clear filters', 'IoT-enabled', and 'Debugger'. The main table lists boards with columns for Name, Platform, Frameworks, MCU, FRQ, ROM, RAM, and Extra. Each board entry includes a '+' icon, the manufacturer name, the board name, the platform, frameworks, MCU type, frequency, memory sizes, and two small blue icons.

Name	Platform	Frameworks	MCU	FRQ	ROM	RAM	Extra	
1Bitisy	ST STM32	CMSIS, libOpen CM3, SPL, STM 32Cube	STM32F415RG	168 Mhz	1 MB	128 KB		
4DSystems PICadillo 35 T	Microchip PIC32	Arduino	32MX7L	95F512	80 Mhz	508 KB	128 KB	
96Boards B 96B-F446VE	ST STM32	mbed, STM32Cube	STM32F446VET6	168 Mhz	512 KB	128 KB		
Adafruit Bluefruit Micro	Atmel AVR	Arduino	ATMEGA32U4	8 Mhz	28 KB	2.5 KB		
Adafruit Circuit Playground Express	Atmel SAM	Arduino	SAMD21G18A	48 Mhz	256 KB	32 KB		
Adafruit ESP32 Feather	Espressif 32	Arduino, ESP-IDF	ESP32	240 Mhz	1 MB	288 KB		

1.5 Tutorials and Examples

1.5.1 Tutorials

Unit Testing of a “Blink” Project

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

Project structure

```
project_dir
├── lib
│   └── README
├── platformio.ini
└── src
    └── ...
└── test
    └── test_main.cpp
```

Source files

- `platformio.ini`

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

[env:uno]

(continues on next page)

(continued from previous page)

```
platform = atmelavr
framework = arduino
board = uno

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

- test/test_main.cpp

```
#include <Arduino.h>
#include <unity.h>

// void setUp(void) {
// // set stuff up here
// }

// void tearDown(void) {
// // clean stuff up here
// }

void test_led_builtin_pin_number(void) {
    TEST_ASSERT_EQUAL(13, LED_BUILTIN);
}

void test_led_state_high(void) {
    digitalWrite(LED_BUILTIN, HIGH);
    TEST_ASSERT_EQUAL(HIGH, digitalRead(LED_BUILTIN));
}

void test_led_state_low(void) {
    digitalWrite(LED_BUILTIN, LOW);
    TEST_ASSERT_EQUAL(LOW, digitalRead(LED_BUILTIN));
}

void setup() {
    // NOTE!!! Wait for >2 secs
    // if board doesn't support software reset via Serial.DTR/RTS
    delay(2000);

    UNITY_BEGIN();      // IMPORTANT LINE!
    RUN_TEST(test_led_builtin_pin_number);

    pinMode(LED_BUILTIN, OUTPUT);
}

uint8_t i = 0;
uint8_t max_blinks = 5;

void loop() {
    if (i < max_blinks)
    {
        RUN_TEST(test_led_state_high);
        delay(500);
        RUN_TEST(test_led_state_low);
        delay(500);
    }
}
```

(continues on next page)

(continued from previous 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 `--verbose` option
Collected 1 items

===== [test/*] Building... (1/3) =====
Processing uno (platform: atmelavr; board: uno; framework: arduino)
-----
Verbose mode can be enabled via `--verbose` option
PLATFORM: Atmel AVR > Arduino Uno
SYSTEM: ATMEGA328P 16MHz 2KB RAM (31.50KB Flash)
Library Dependency Finder -> http://bit.ly/configure-pio-ldf
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\CMSIS_Driver_Include\Hardware\Serial\CMSIS_HardwareSerial.h
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.cpp.o
Compiling .pio\build\uno\FrameworkArduino\hooks.c.o
Compiling .pio\build\uno\FrameworkArduino\main.cpp.o
Compiling .pio\build\uno\FrameworkArduino\new.cpp.o
Compiling .pio\build\uno\FrameworkArduino\wiring.c.o
Compiling .pio\build\uno\FrameworkArduino\wiring_analog.c.o
Compiling .pio\build\uno\FrameworkArduino\wiring_digital.c.o
Compiling .pio\build\uno\FrameworkArduino\wiring_pulse.S.o
Compiling .pio\build\uno\FrameworkArduino\wiring_pulse.c.o
```

(continues on next page)

(continued from previous page)

```

Compiling .pio\build\uno\FrameworkArduino\wiring_shift.c.o
Compiling .pio\build\uno\UnityTestLib\unity.o
Archiving .pio\build\uno\libFrameworkArduino.a
Indexing .pio\build\uno\libFrameworkArduino.a
Archiving .pio\build\uno\libUnityTestLib.a
Indexing .pio\build\uno\libUnityTestLib.a
Linking .pio\build\uno\firmware.elf
Checking size .pio\build\uno\firmware.elf
Building .pio\build\uno\firmware.hex
Memory Usage -> http://bit.ly/pio-memory-usage
DATA: [==] 20.0% (used 410 bytes from 2048 bytes)
PROGRAM: [=] 12.6% (used 4060 bytes from 32256 bytes)

=====
[SUMMARY]
=====
Environment uno [SUCCESS]
Environment teensy31 [SKIP]
===== [SUCCESS] Took 2.54 seconds
=====

[test/*] Uploading... (2/3)
=====
Processing uno (platform: atmelavr; board: uno; framework: arduino)
-----
Verbose mode can be enabled via `'-v, --verbose` option
PLATFORM: Atmel AVR > Arduino Uno
SYSTEM: ATMEGA328P 16MHz 2KB RAM (31.50KB Flash)
Library Dependency Finder -> http://bit.ly/configure-pio-ldf
LDF MODES: FINDER(chain) COMPATIBILITY(soft)
Collected 24 compatible libraries
Scanning dependencies...
No dependencies
Checking size .pio\build\uno\firmware.elf
Memory Usage -> http://bit.ly/pio-memory-usage
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

avrduude: AVR device initialized and ready to accept instructions

Reading | ##### | 100% 0.00s

avrduude: Device signature = 0x1e950f (probably m328p)
avrduude: reading input file ".pio\build\uno\firmware.hex"
avrduude: writing flash (4060 bytes):

Writing | ##### | 100% 0.76s

avrduude: 4060 bytes of flash written
avrduude: verifying flash memory against .pio\build\uno\firmware.hex:
avrduude: load data flash data from input file .pio\build\uno\firmware.hex:
avrduude: input file .pio\build\uno\firmware.hex contains 4060 bytes

```

(continues on next page)

(continued from previous page)

```

avrduude: reading on-chip flash data:

Reading | ##### | 100% 0.48s

avrduude: verifying ...
avrduude: 4060 bytes of flash verified

avrduude: safemode: Fuses OK (E:00, H:00, L:00)

avrduude done. Thank you.

=====
[SUMMARY]
Environment uno [SUCCESS]
Environment teensy31 [SKIP]
===== [SUCCESS] Took 4.45 seconds =====

=====
[test/*] Testing... (3/3) ↵
=====
If you don't see any output for the first 10 secs, please reset board (press reset ↵
button)

test\test_main.cpp:30:test_led_builtin_pin_number [PASSED]
test\test_main.cpp:41:test_led_state_high [PASSED]
test\test_main.cpp:43:test_led_state_low [PASSED]
=====
11 Tests 0 Failures 0 Ignored

=====
[TEST SUMMARY]
test/*/env:uno [PASSED]
test/*/env:teensy31 [IGNORED]
===== [PASSED] Took 12.99 seconds =====

```

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 the [Arduino](#) framework for the ESP32-DevKitC board.

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

Requirements:

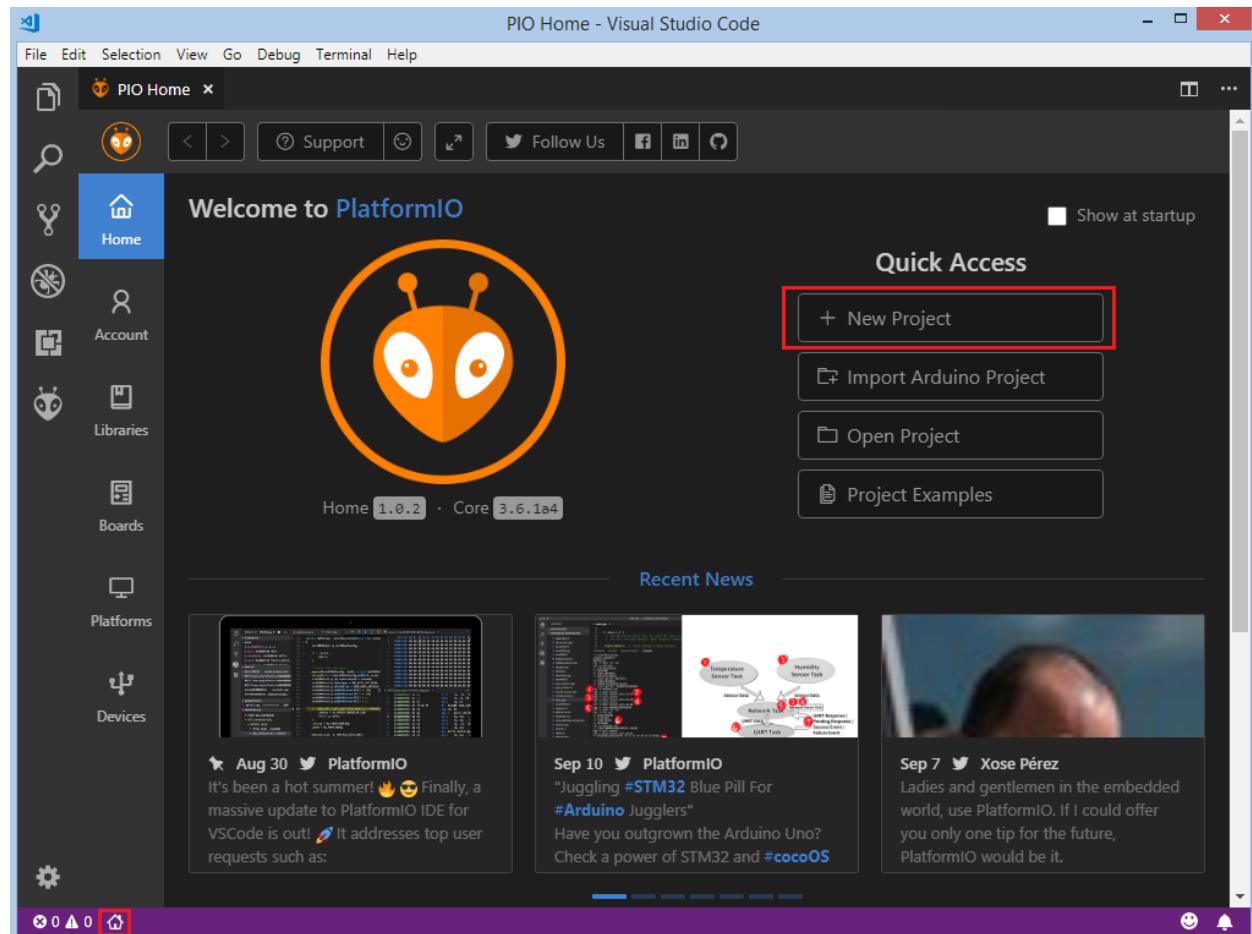
- Downloaded and installed *PlatformIO IDE for VSCode*
- [ESP32-DevKitC](#) development board
- [Olimex ARM-USB-OCD](#) or [Olimex ARM-USB-TINY](#) adapter for debugging

Contents

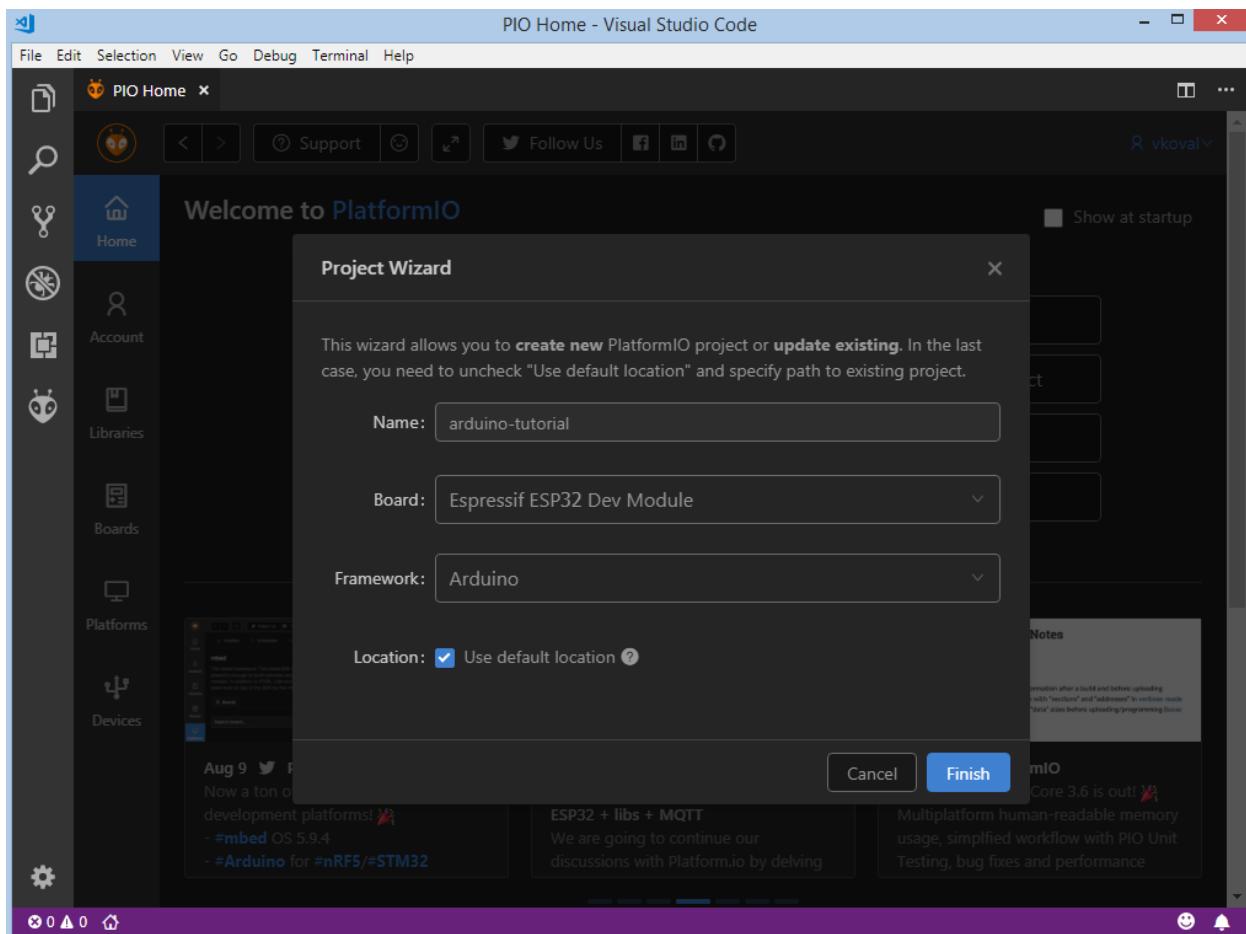
- [Setting Up the Project](#)
- [Adding Code to the Generated Project](#)
- [Compiling and Uploading the Firmware](#)
- [Debugging the Firmware](#)
 - [Setting Up the Hardware](#)
- [Writing Unit Tests](#)
- [Adding Bluetooth LE features](#)
- [Conclusion](#)

Setting Up the Project

First, we need to create a new project using the PlatformIO Home Page (to open this page, just press the Home icon on the toolbar):



Next, we need to select `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 time (PlatformIO will download and install all required packages). After that, we have a fully configured project that is ready for developing code with the *Arduino* framework.

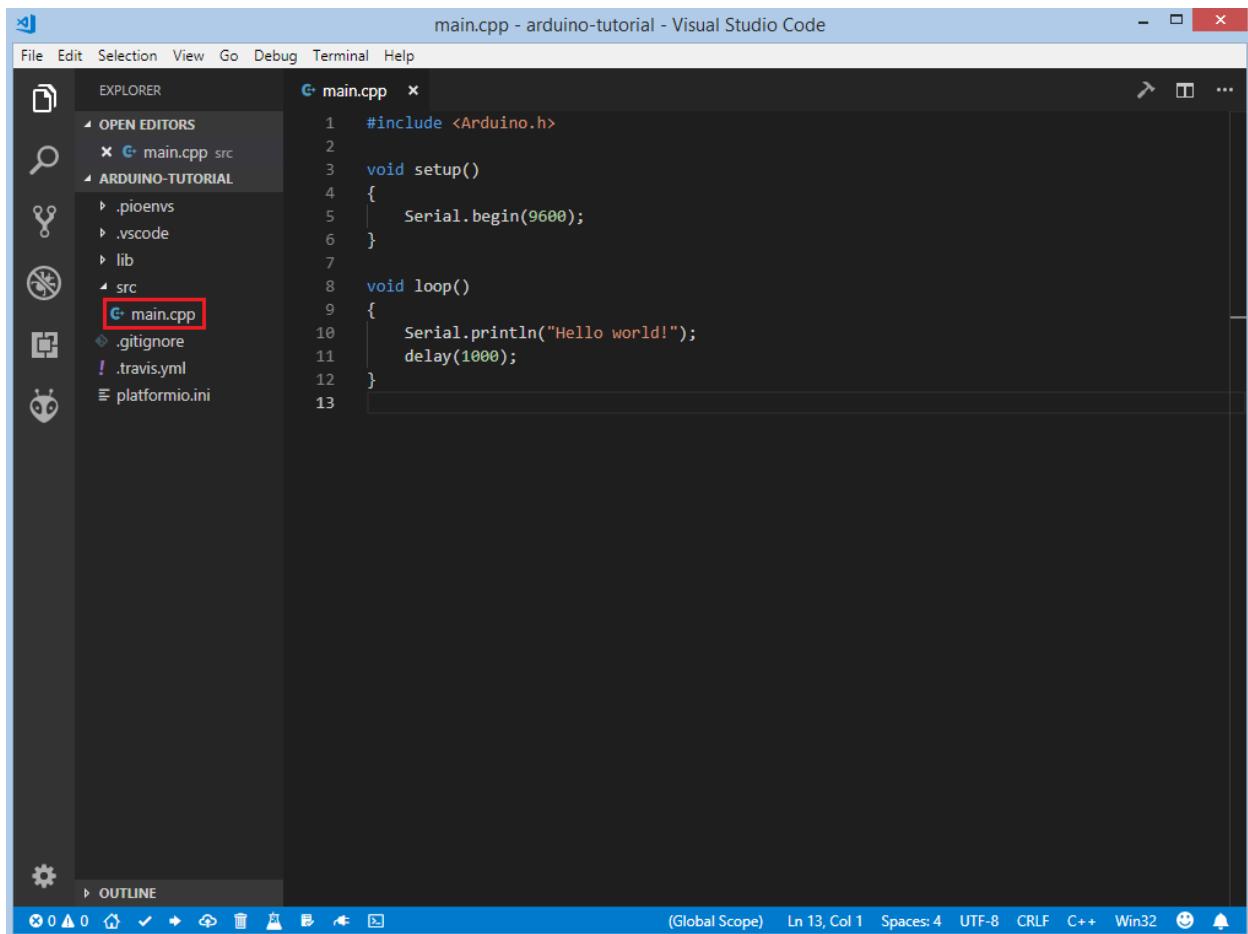
Adding Code to the Generated Project

Let's add some actual code to the project. Firstly, we open a default main file named `main.cpp` in the `src_dir` folder and replace its content with following:

```
#include <Arduino.h>

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

void loop()
{
    Serial.println("Hello world!");
    delay(1000);
}
```



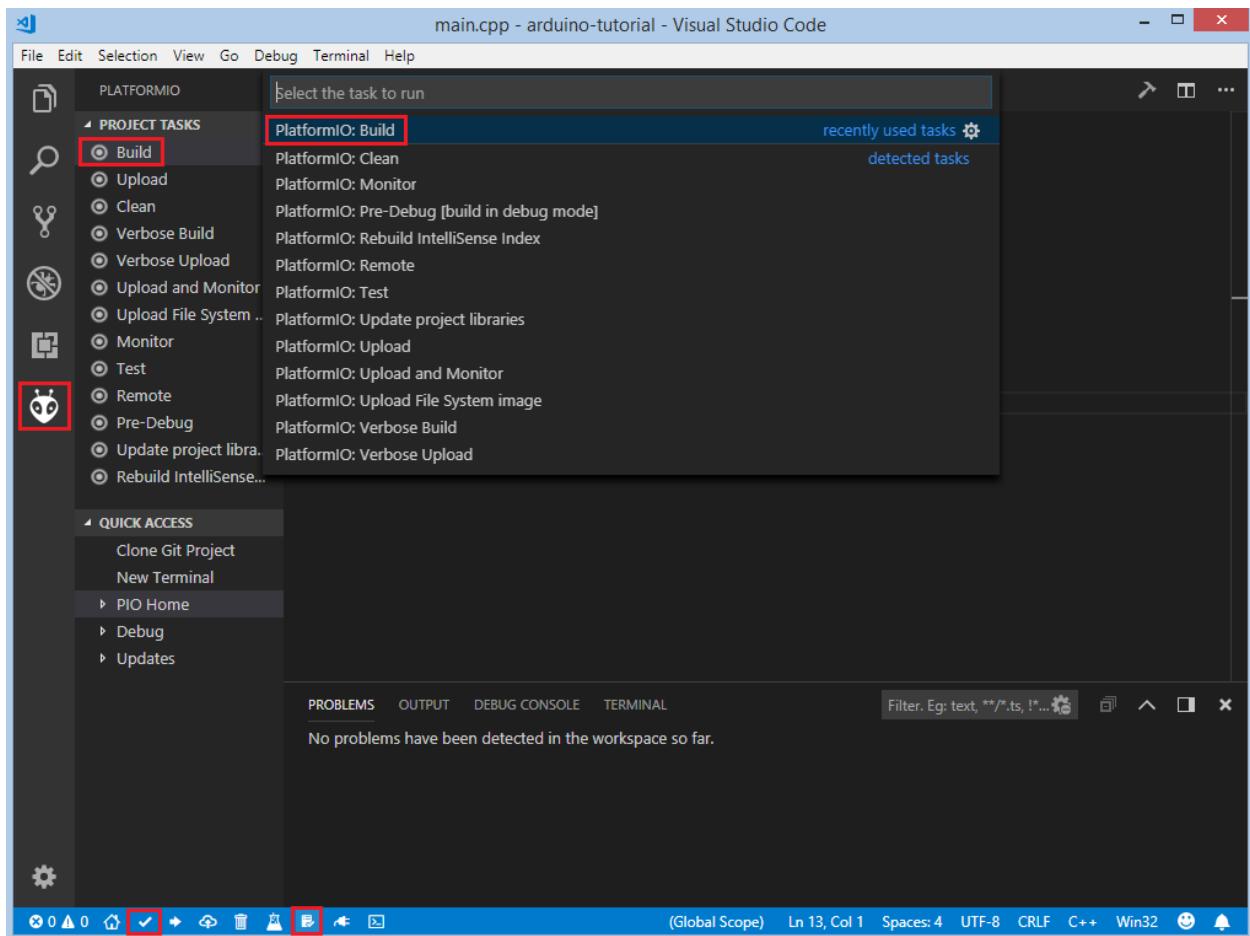
We have now created a basic project ready for compiling and uploading.

Compiling and Uploading the Firmware

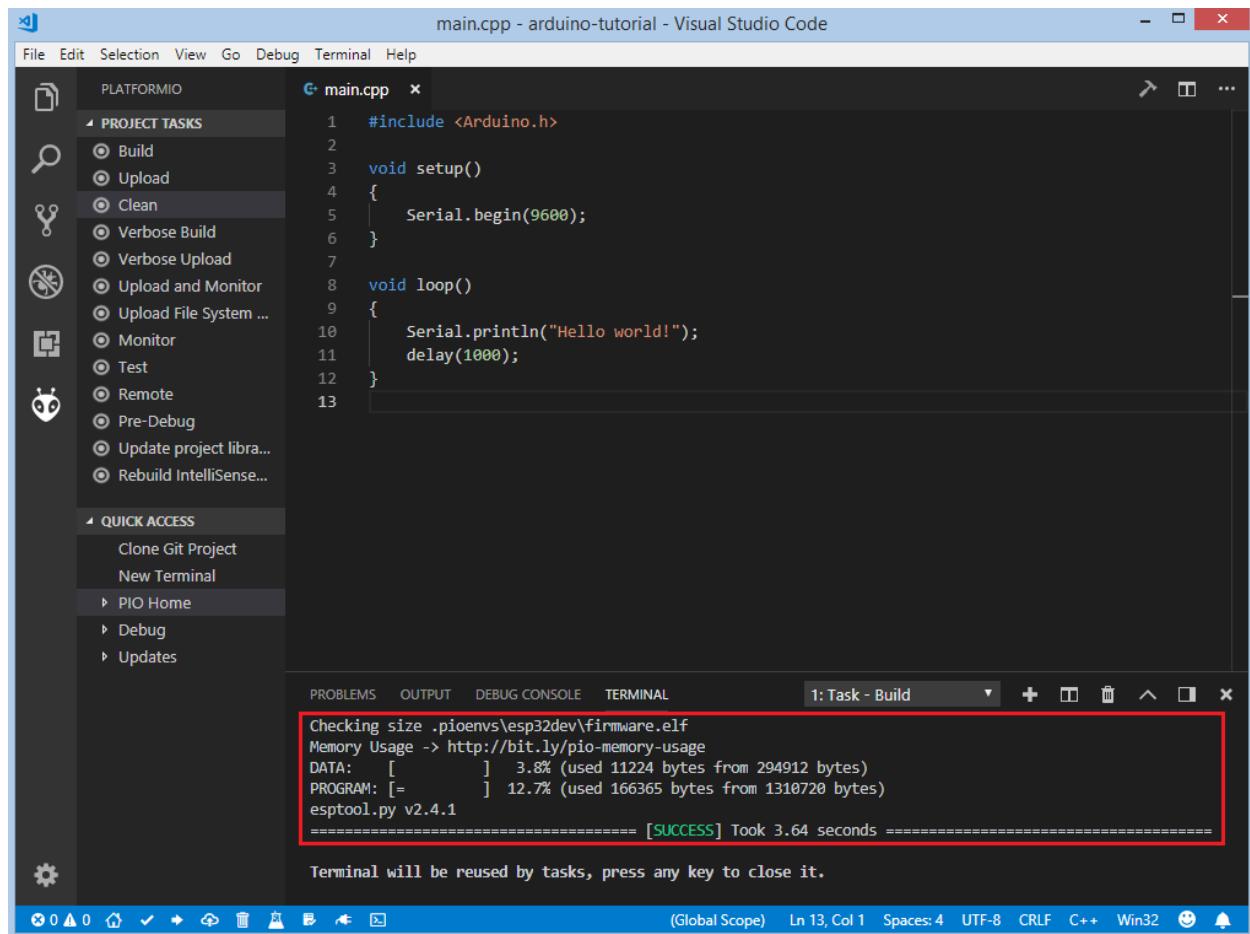
Now we can build the project. There are several ways to compile firmware:

- Build option in the Project Tasks menu,
- Build button in *PlatformIO Toolbar*,
- Task Menu: Tasks: Run Task... > PlatformIO: Build, or in the *PlatformIO Toolbar*,
- Command Palette: View: Command Palette > PlatformIO: Build, or
- via hotkeys cmd+alt+b / ctrl+alt+b

Marked in red:

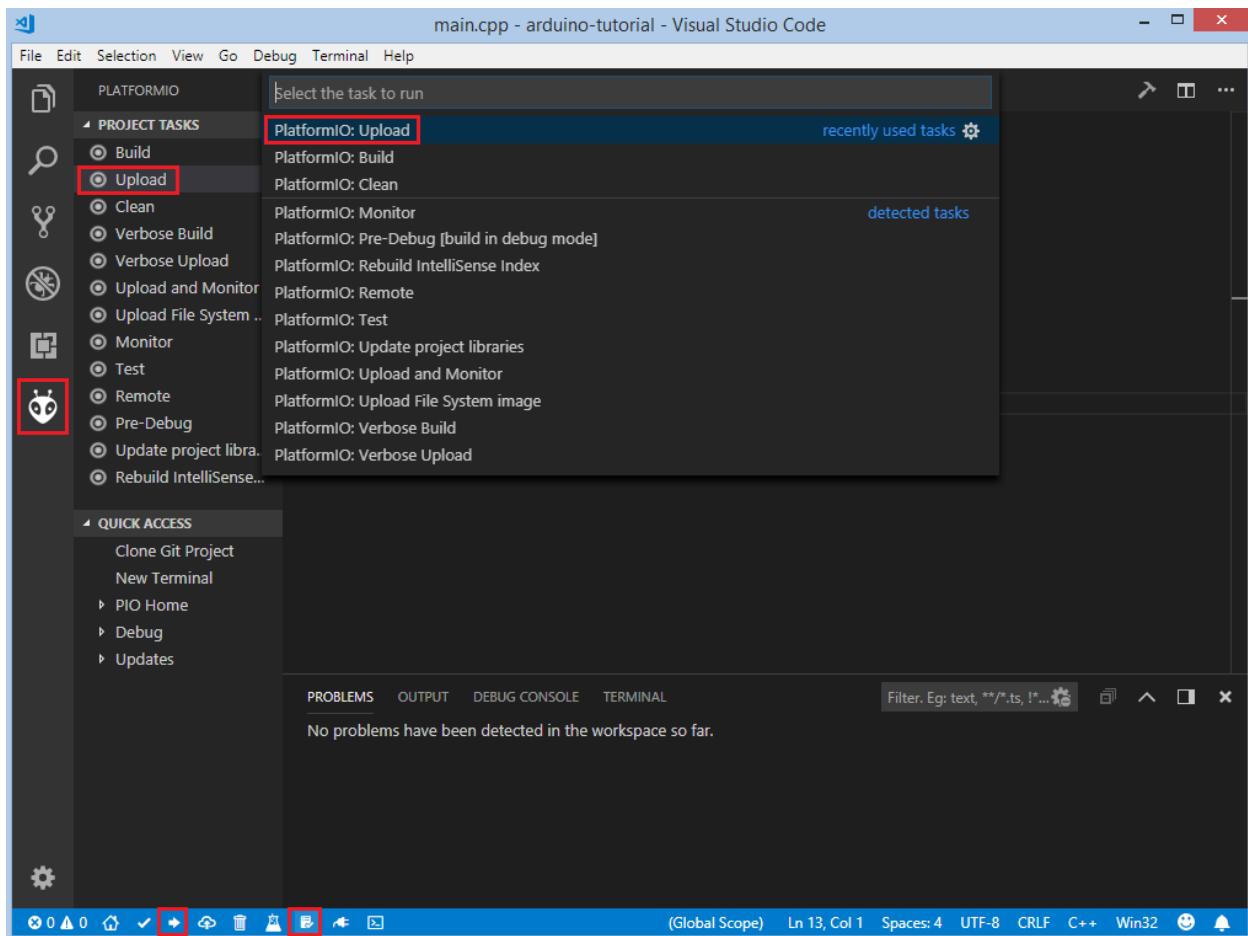


If everything went well, we should see a Success message in the terminal window:



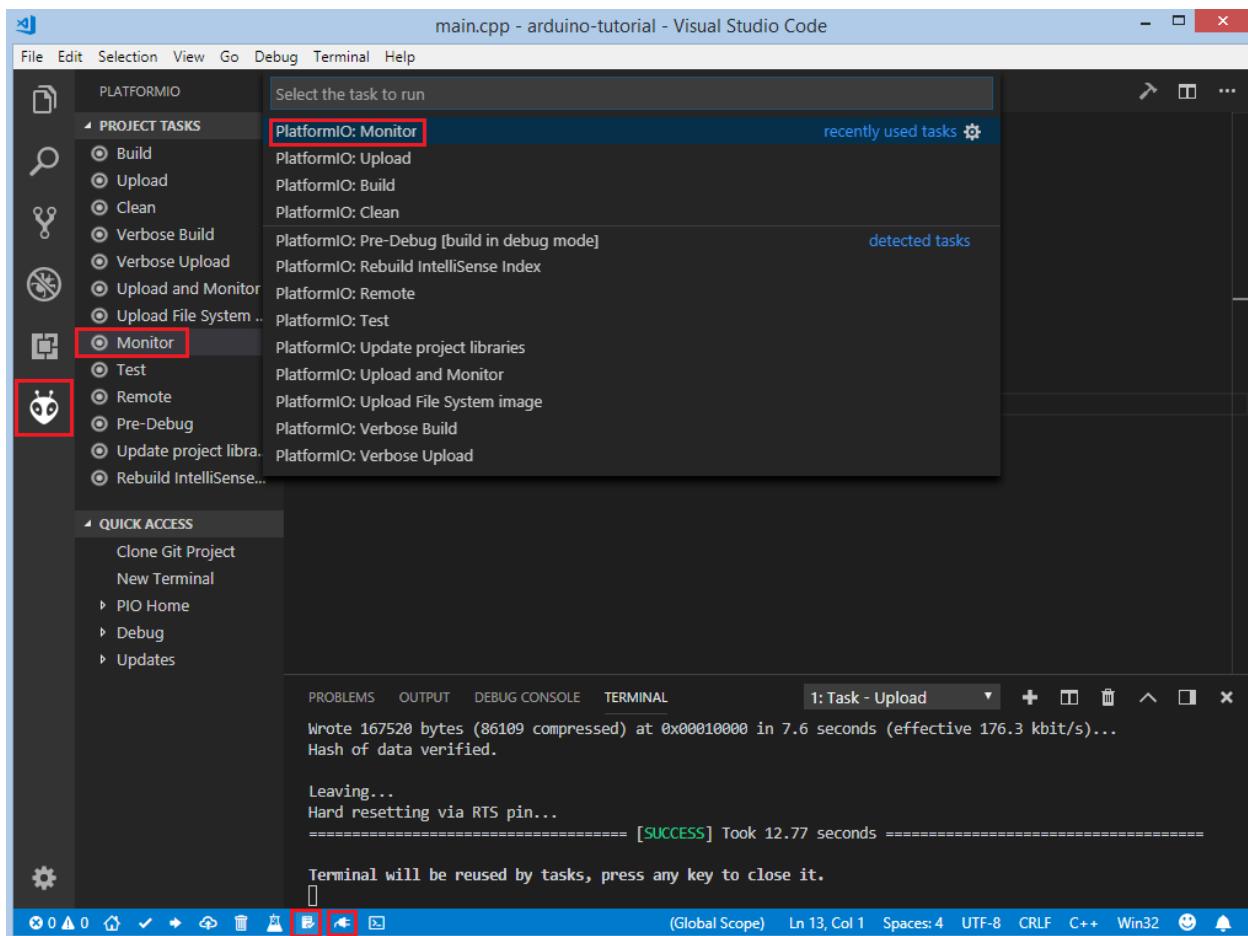
There are also several ways to upload the firmware to the board:

- Upload option in the `Project Tasks` menu,
- Upload button in `PlatformIO Toolbar`,
- Command Palette: `View > Command Palette > PlatformIO: Upload`,
- using the Task Menu: `Tasks: Run Task... > PlatformIO: Upload`, or
- via hotkeys: `cmd+alt+u / ctrl+alt+u`:

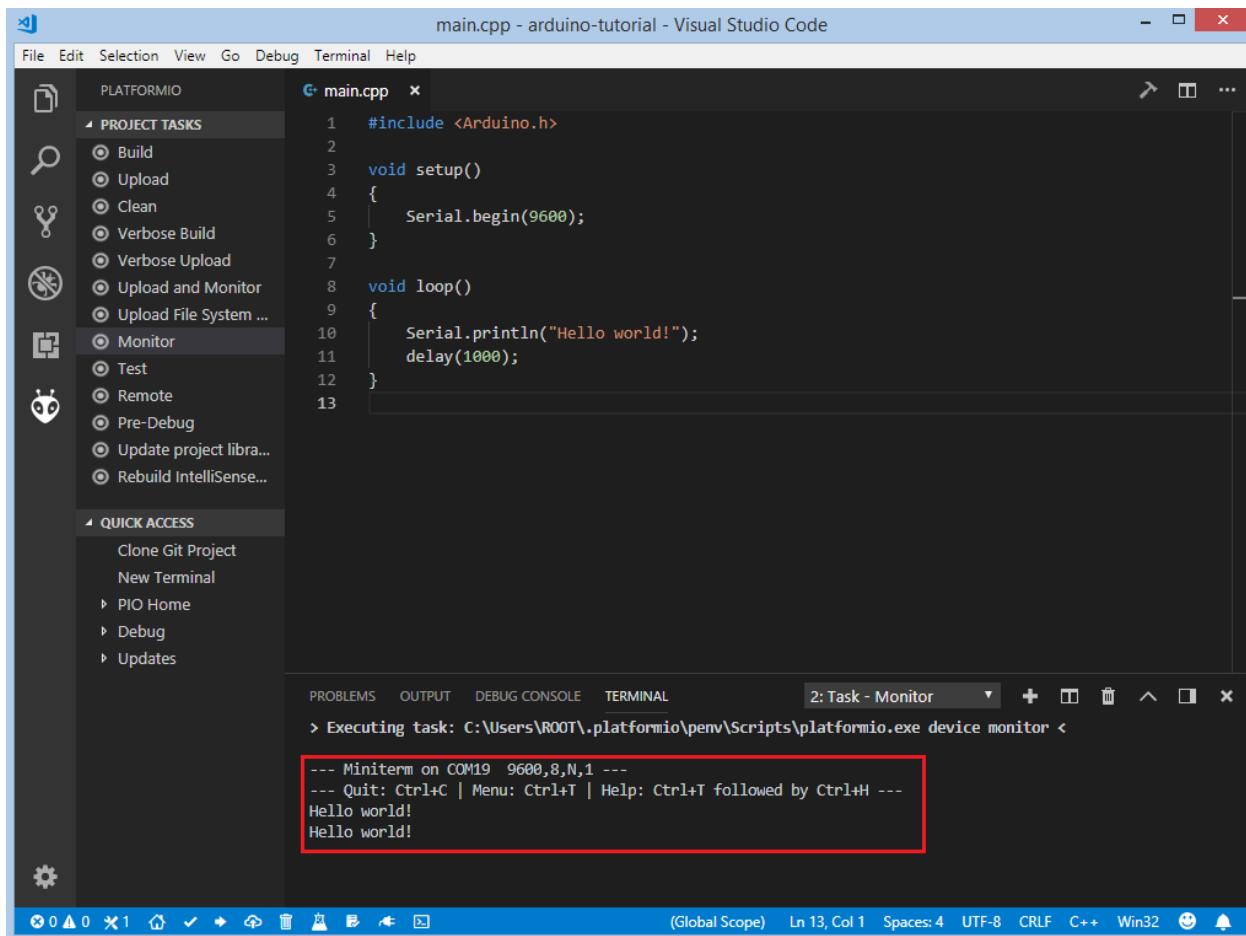


After uploading, we need to check if the firmware is uploaded correctly. To do this, open the serial monitor and check that the message from the board is received. To open the serial monitor, we can use the following options:

- Monitor option in the Project Tasks menu,
- Serial Monitor button in the *PlatformIO Toolbar*,
- Command Palette: View: Command Palette > PlatformIO: Monitor, or
- Task Menu: Tasks: Run Task... > PlatformIO: Monitor:



If the firmware works as expected, the message from the board can be observed in the terminal window:



Debugging the Firmware

Setting Up the Hardware

In order to use a JTAG probe with an ESP32, we need to connect the following pins:

ESP32 pin	JTAG probe pin
3.3V	Pin 1 (VTref)
GPIO 9 (EN)	Pin 3 (nTRST)
GND	Pin 4 (GND)
GPIO 12 (TDI)	Pin 5 (TDI)
GPIO 14 (TMS)	Pin 7 (TMS)
GPIO 13 (TCK)	Pin 9 (TCK)
GPIO 15 (TDO)	Pin 13 (TDO)

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, an *Olimex ARM-USB-OCD-H* debug probe is used:

```

[env:esp32dev]
platform = espressif32
board = esp32dev

```

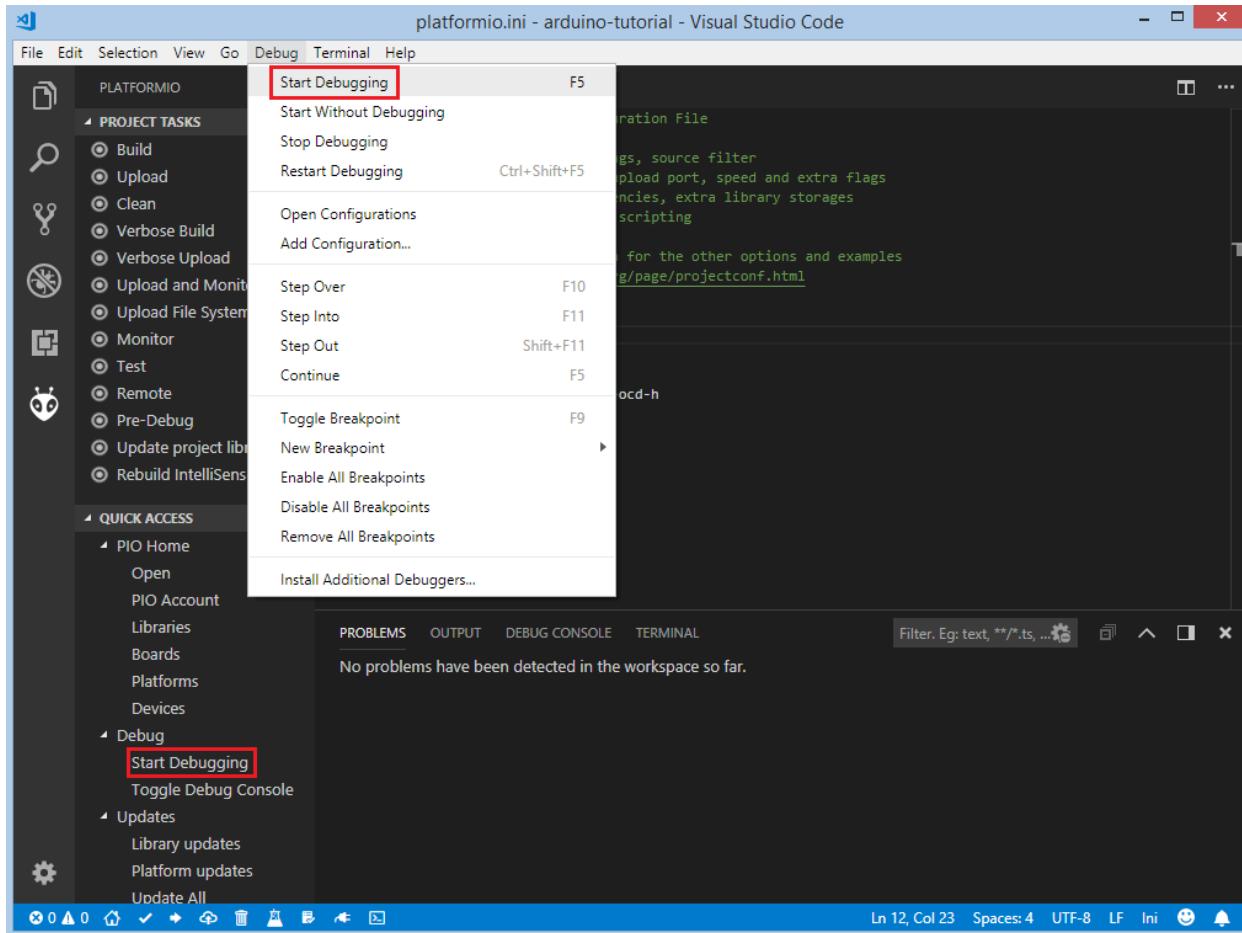
(continues on next page)

(continued from previous page)

```
framework = arduino
debug_tool = olimex-arm-usb-ocd-h
```

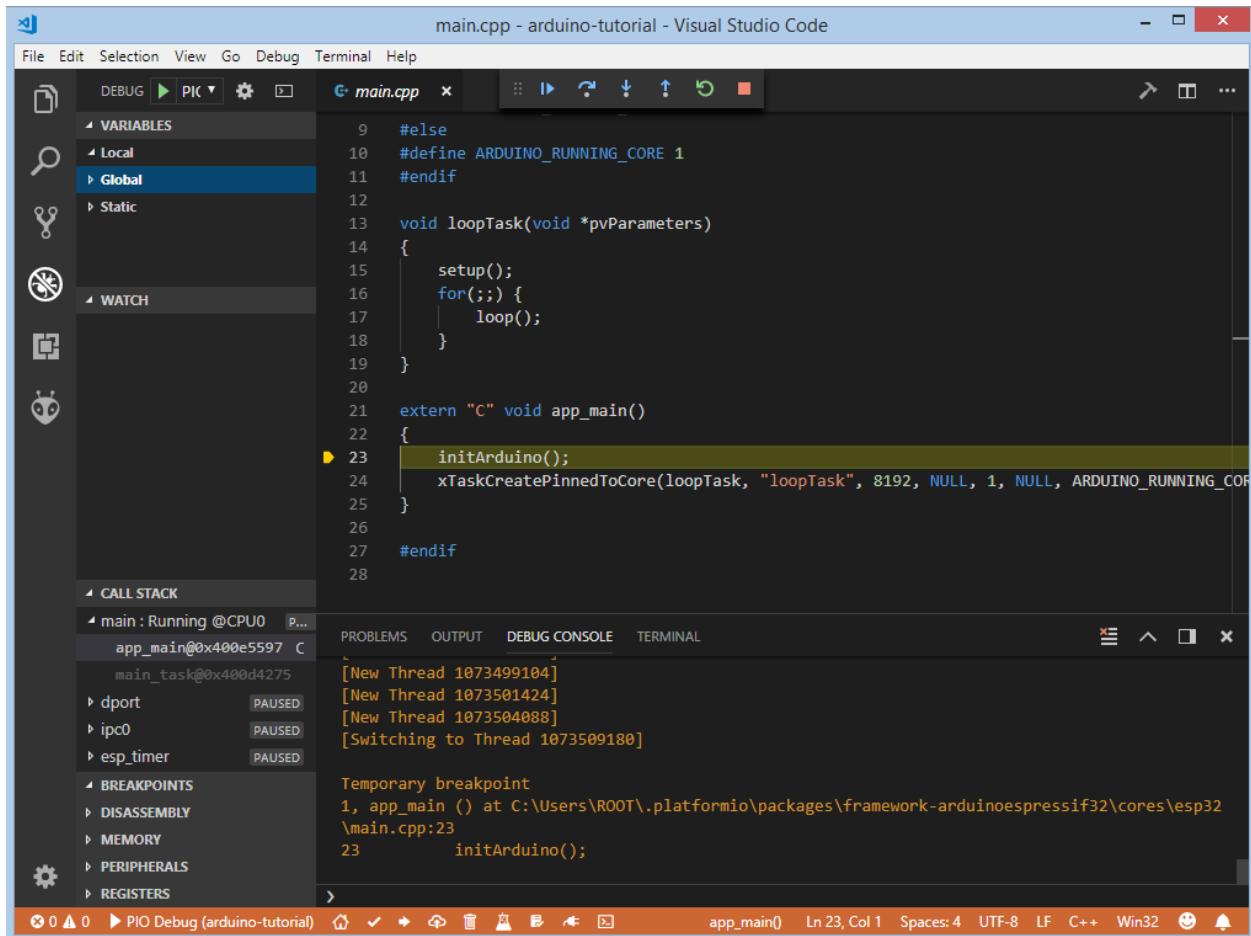
To start the debug session we can use the following methods:

- Debug: Start debugging in the top menu,
- Start Debugging option in the Quick Access menu, or
- hotkey button F5:

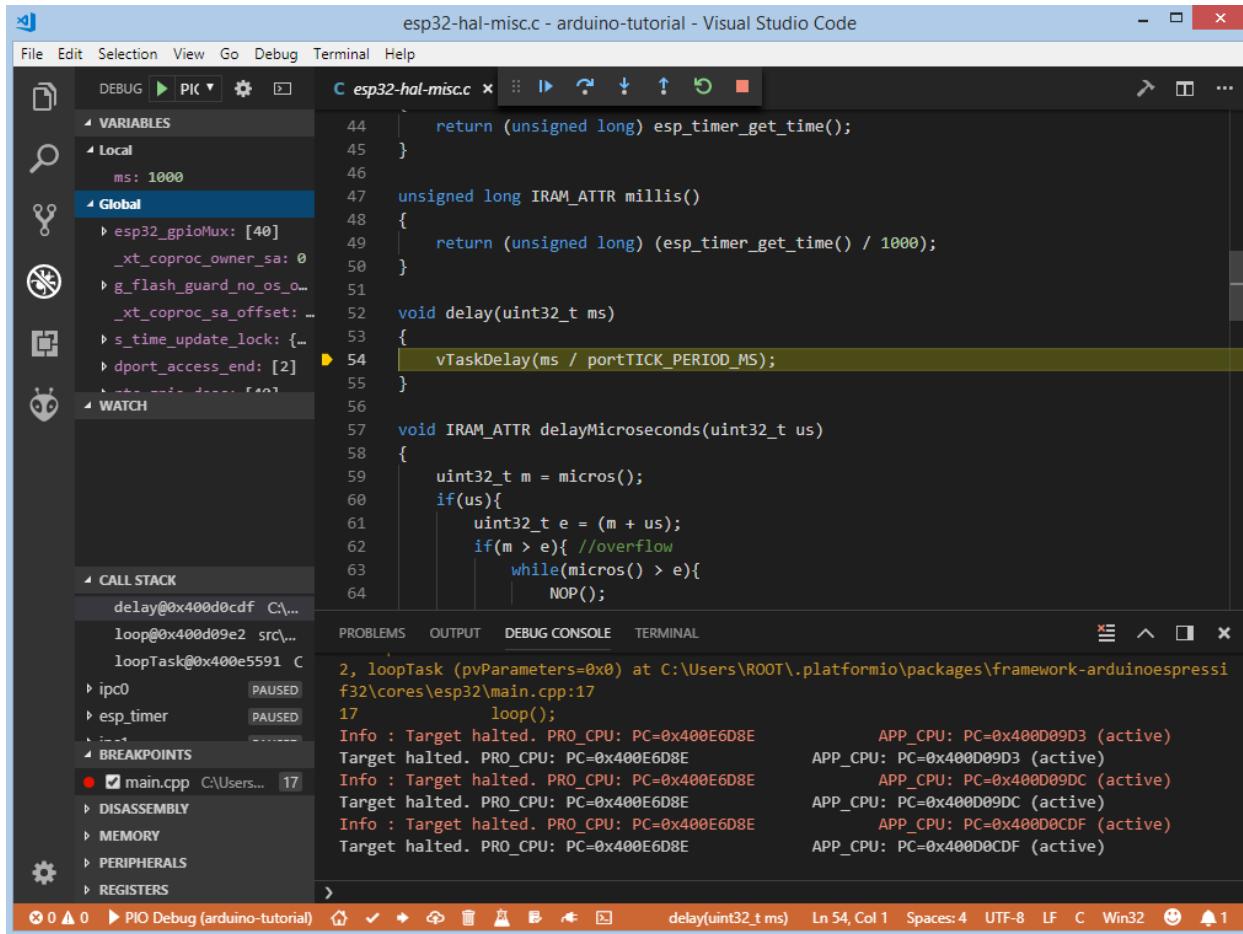


We need to wait some time while PlatformIO initializes the debug session, and are ready to debug when the first line after the main function is highlighted.

1. Please wait when debugging session is stopped at the first line of `app_main()` function
2. **WARNING!** Please set a breakpoint at `void loopTask(void *pvParameters)` (line 13 in the screenshot below - this line can change between releases)
3. Now, please press CONTINUE/RUN button on debugging toolbar (right arrow icon)
4. The debugging session should stop at the first line of the `void loopTask(void *pvParameters)` function
5. Now, navigate to your Arduino setup/loop code and do classic debugging.



We can walk through the code using control buttons, set breakpoints, and add variables to the Watch window:



Writing Unit Tests

Test cases can be added to a single file that may include multiple tests. First of all, in this file, we need to add four default functions: `setUp`, `tearDown`, `setup` and `loop`. Functions `setUp` and `tearDown` are used to initialize and finalize test conditions. Implementations of these functions are not required for running tests, but if you need to initialize some variables before you run a test, use the `setUp` function. Likewise, if you need to clean up variables, use `tearDown` function. In our example we will use these functions to respectively initialize and deinitialize LED states. The `setup` and `loop` functions act as a simple Arduino program where we describe our test plan.

Let's create a `test` folder in the root of the project and add a new file, `test_main.cpp`, to this folder. Next, basic tests for `String` class will be implemented in this file:

- `test_string_concat` tests the concatenation of two strings
- `test_string_substring` tests the correctness of the substring extraction
- `test_string_index_of` ensures that the string returns the correct index of the specified symbol
- `test_string_equal_ignore_case` tests case-insensitive comparison of two strings
- `test_string_to_upper_case` tests conversion of the string to upper-case
- `test_string_replace` tests the correctness of the replacing operation

```
#include <Arduino.h>
#include <unity.h>
```

(continues on next page)

(continued from previous page)

```

String STR_TO_TEST;

void setUp(void) {
    // set stuff up here
    STR_TO_TEST = "Hello, world!";
}

void tearDown(void) {
    // clean stuff up here
    STR_TO_TEST = "";
}

void test_string_concat(void) {
    String hello = "Hello, ";
    String world = "world!";
    TEST_ASSERT_EQUAL_STRING(STR_TO_TEST.c_str(), (hello + world).c_str());
}

void test_string_substring(void) {
    TEST_ASSERT_EQUAL_STRING("Hello", STR_TO_TEST.substring(0, 5).c_str());
}

void test_string_index_of(void) {
    TEST_ASSERT_EQUAL(7, STR_TO_TEST.indexOf('w'));
}

void test_string_equal_ignore_case(void) {
    TEST_ASSERT_TRUE(STR_TO_TEST.equalsIgnoreCase("HELLO, WORLD!"));
}

void test_string_to_upper_case(void) {
    STR_TO_TEST.toUpperCase();
    TEST_ASSERT_EQUAL_STRING("HELLO, WORLD!", STR_TO_TEST.c_str());
}

void test_string_replace(void) {
    STR_TO_TEST.replace('!', '?');
    TEST_ASSERT_EQUAL_STRING("Hello, world?", STR_TO_TEST.c_str());
}

void setup()
{
    delay(2000); // service delay
    UNITY_BEGIN();

    RUN_TEST(test_string_concat);
    RUN_TEST(test_string_substring);
    RUN_TEST(test_string_index_of);
    RUN_TEST(test_string_equal_ignore_case);
    RUN_TEST(test_string_to_upper_case);
    RUN_TEST(test_string_replace);

    UNITY_END(); // stop unit testing
}

void loop()

```

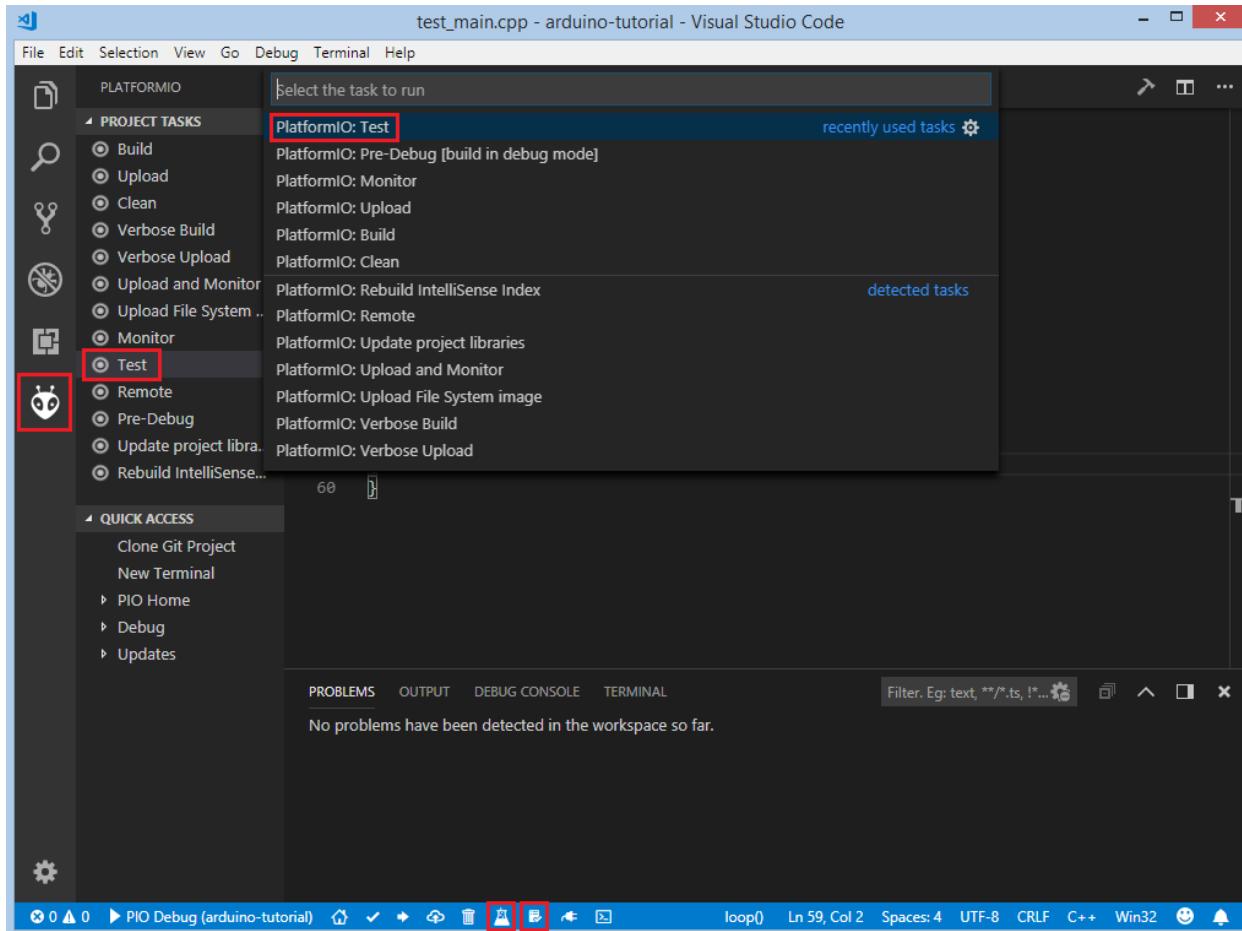
(continues on next page)

(continued from previous page)

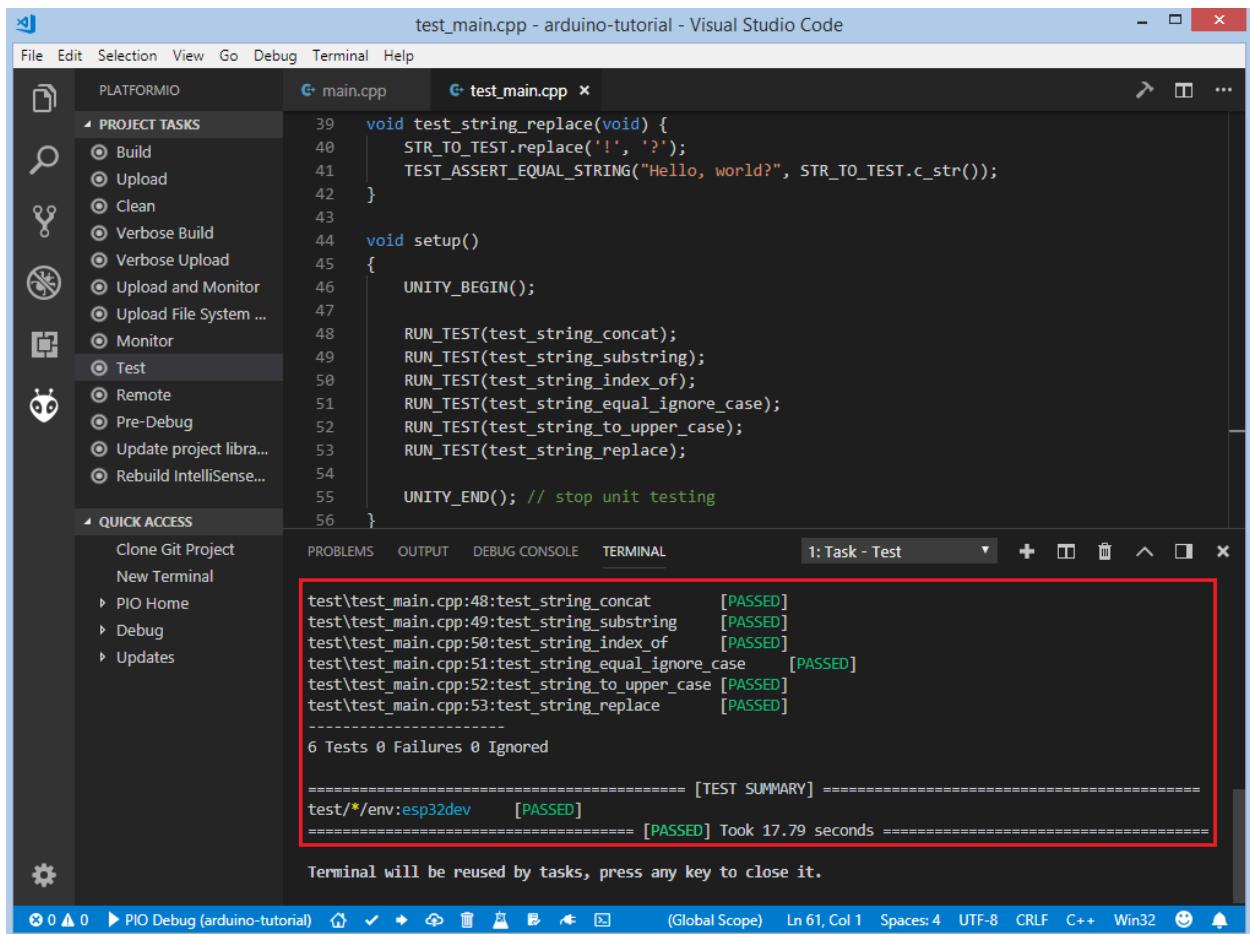
{
}

Now we are ready to upload tests to the board. To do this we can use the following:

- Test button on *PlatformIO Toolbar*,
- Test option in the Project Tasks menu, or
- Tasks: Run Task... > PlatformIO Test in the top menu:



After processing, we should see a detailed report about the testing results:



As we can see from the report, all our tests were successful!

Adding Bluetooth LE features

Now let's create a basic application that can interact with other BLE devices (e.g phones). For example, the following code declares a BLE characteristic whose value can be printed to the serial port:

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

#define SERVICE_UUID          "4fafc201-1fb5-459e-8fcc-c5c9c331914b"
#define CHARACTERISTIC_UUID   "beb5483e-36e1-4688-b7f5-ea07361b26a8"

class MyCallbacks: public BLECharacteristicCallbacks {
    void onWrite(BLECharacteristic *pCharacteristic) {
        std::string value = pCharacteristic->getValue();
        if (value.length() > 0) {
            Serial.print("\r\nNew value: ");
            for (int i = 0; i < value.length(); i++)
                Serial.print(value[i]);
            Serial.println();
        }
    }
}
```

(continues on next page)

(continued from previous page)

```
    }

};

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

    BLEDevice::init("ESP32 BLE example");
    BLEServer *pServer = BLEDevice::createServer();
    BLEService *pService = pServer->createService(SERVICE_UUID);
    BLECharacteristic *pCharacteristic = pService->createCharacteristic(
        CHARACTERISTIC_UUID,
        BLECharacteristic::PROPERTY_READ | BLECharacteristic::PROPERTY_WRITE
    );

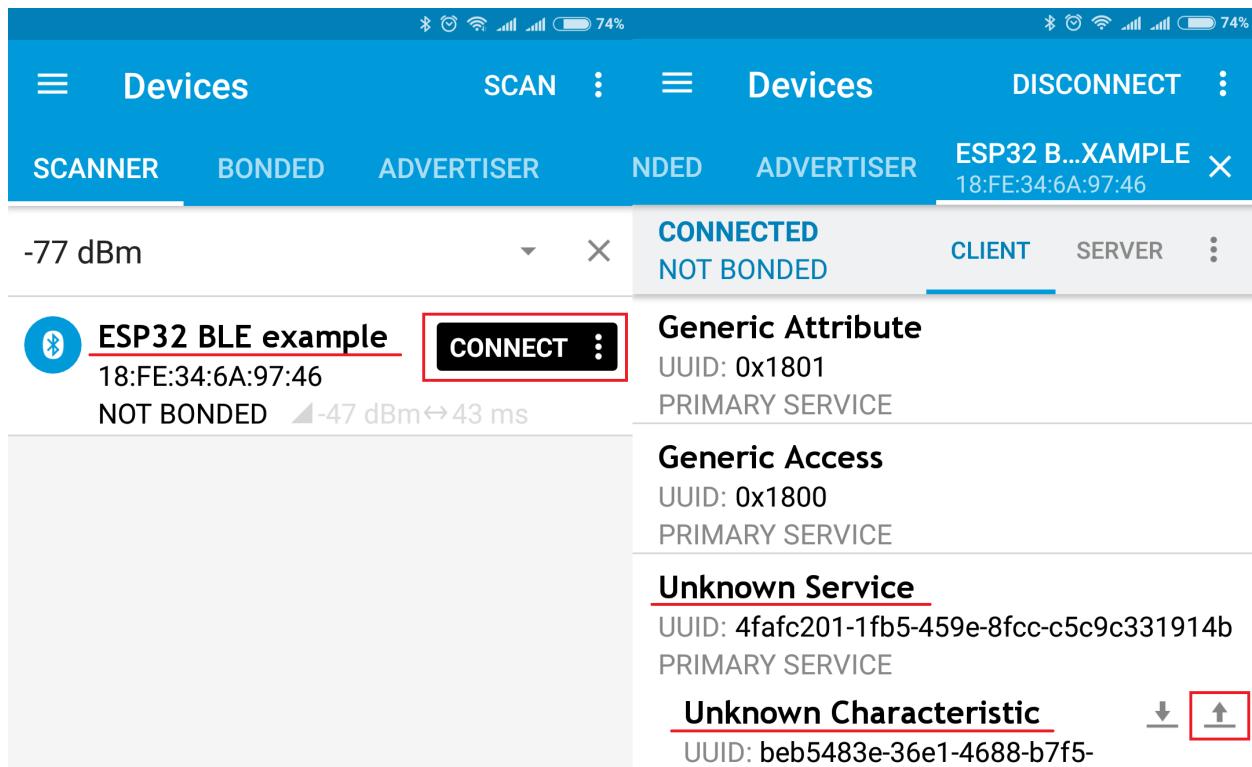
    pCharacteristic->setCallbacks(new MyCallbacks());
    pCharacteristic->setValue("Hello World");
    pService->start();

    BLEAdvertising *pAdvertising = pServer->getAdvertising();
    pAdvertising->start();
}

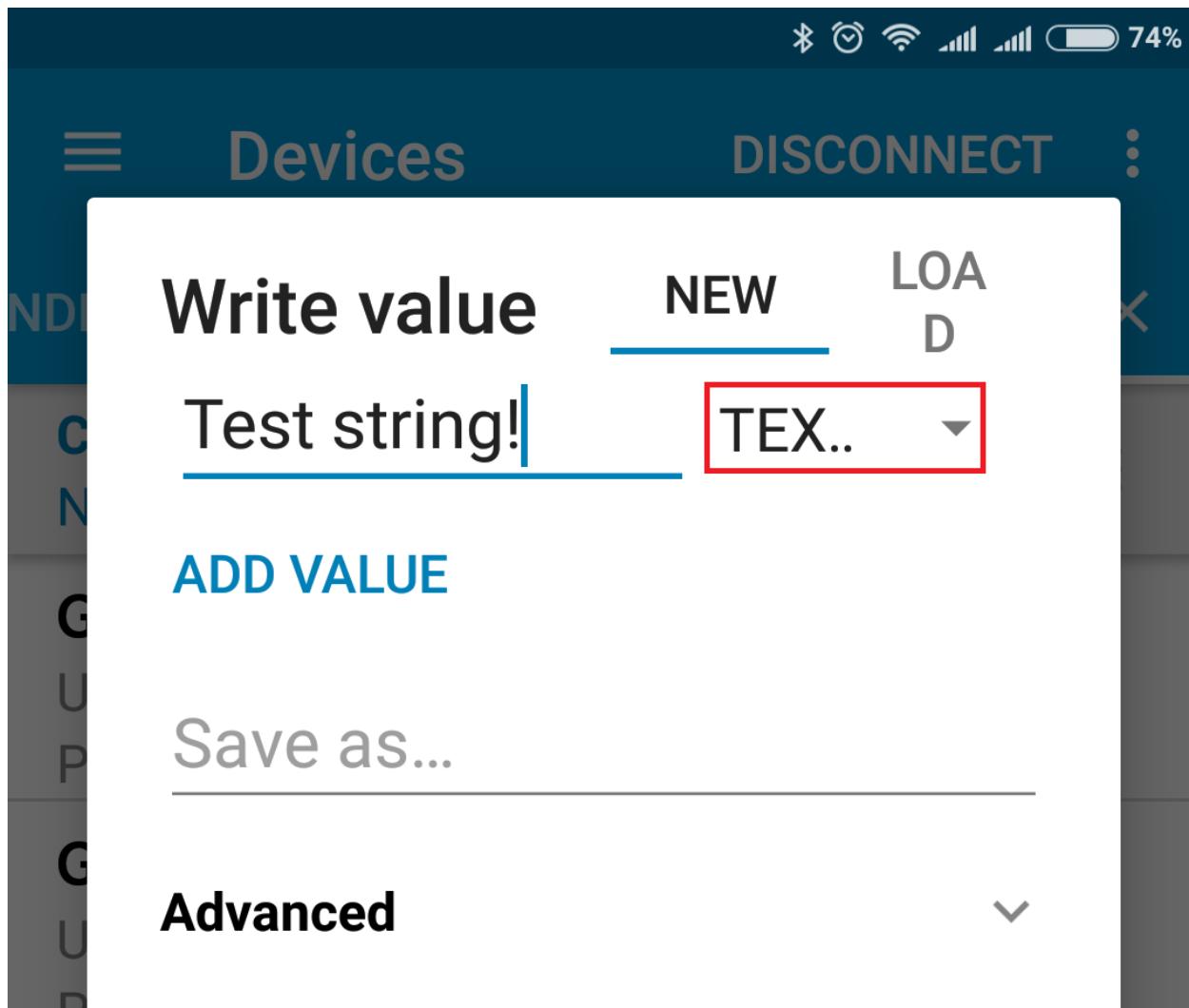
void loop() {
    delay(2000);
}
```

Now we can compile and upload this program to the board as described in the previous sections. To verify that our application works as expected, we can use any Android smartphone with the BLE feature and [Nordic nRF Connect tool](#).

At first, we need to scan all advertising BLE devices and connect to the device called `ESP32 BLE example`. After successful connection to the board, we should see one “Unknown Service” with one “Unknown Characteristic” field:



To set the value, we need to send new text to the BLE characteristic:



The change of the value is printed to the serial monitor:

```

main.cpp - arduino-tutorial - Visual Studio Code
File Edit Selection View Go Debug Terminal Help
PLATFORMIO
PROJECT TASKS
Build
Upload
Clean
Verbose Build
Verbose Upload
Upload and Monitor
Upload File System ...
Monitor
Test
Remote
Pre-Debug
Update project libra...
Rebuild IntelliSense...
QUICK ACCESS
Clone Git Project
New Terminal
PIO Home
Debug
Updates
main.cpp
21 void setup() {
22     Serial.begin(9600);
23
24     BLEDevice::init("ESP32 BLE example");
25     BLEServer *pServer = BLEDevice::createServer();
26     BLEService *pService = pServer->createService(SERVICE_UUID);
27     BLECharacteristic *pCharacteristic = pService->createCharacteristic(
28         CHARACTERISTIC_UUID,
29         BLECharacteristic::PROPERTY_READ |
30         BLECharacteristic::PROPERTY_WRITE
31     );
32
33     pCharacteristic->setCallbacks(new MyCallbacks());
34
35     pCharacteristic->setValue("Hello World");
36     pService->start();
37
38     BLEAdvertising *pAdvertising = pServer->getAdvertising();
39     pAdvertising->start();
40 }
41
42 void loop() {
43     delay(2000);
44 }
45

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 1: Task - Monitor

--- Miniterm on COM19 9600,8,N,1 ---
--- Quit: Ctrl+C | Menu: Ctrl+F | Help: Ctrl+T followed by Ctrl+H ---

New value: Test string!

Conclusion

Now we have a project template for the ESP32-DevKitC board that we can use as boilerplate for later projects.

STM32Cube HAL and Nucleo-F401RE: debugging and unit testing

The goal of this tutorial is to demonstrate how simple it is to use *PlatformIO IDE for Atom* to develop, run and debug a basic blink project with *STM32Cube* framework for STM32 Nucleo-F401RE board.

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

Requirements:

- Downloaded and installed *PlatformIO IDE for Atom*
- Install drivers for *ST-LINK* debug tool
- Nucleo-F401RE development board

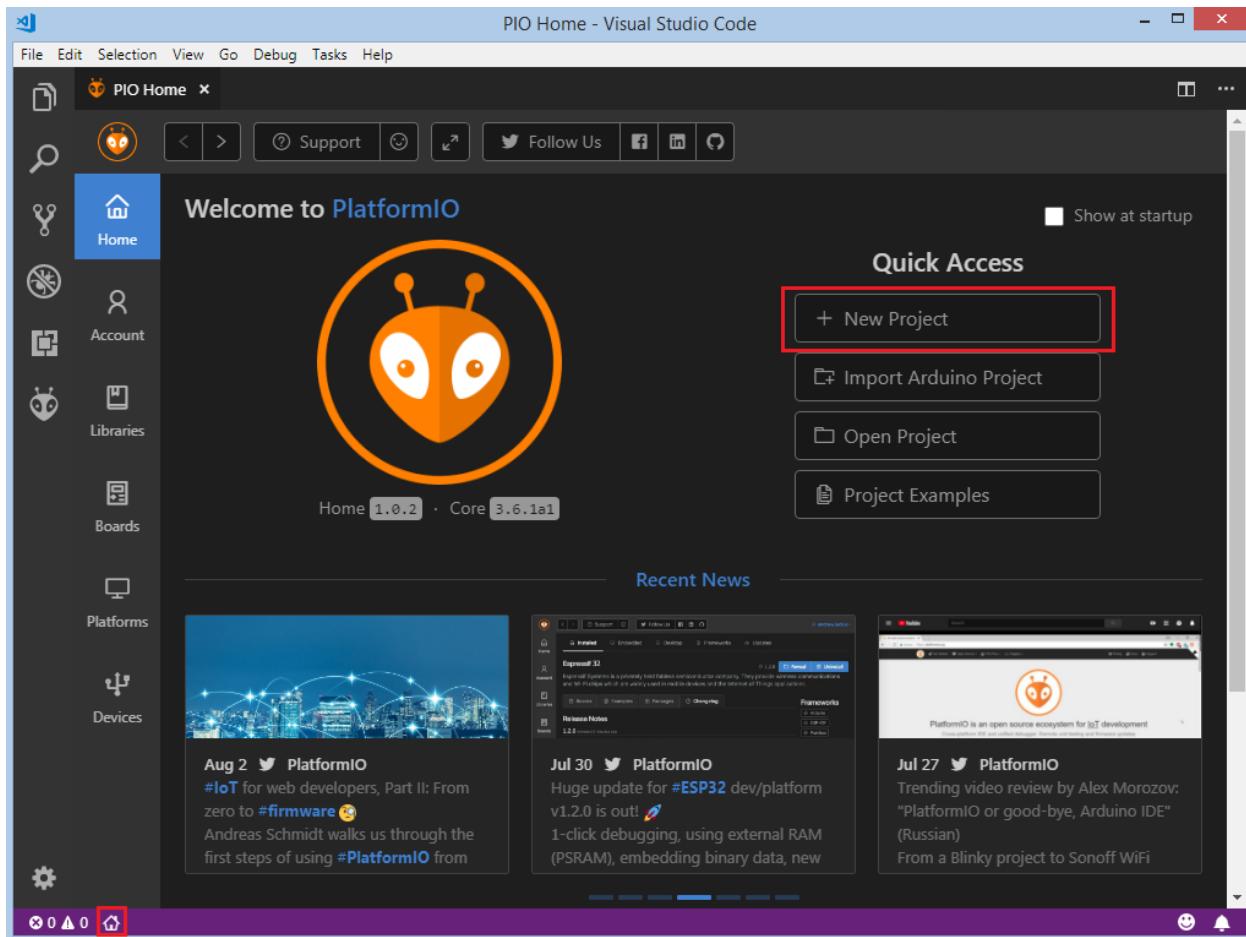
Contents

- *Setting Up the Project*

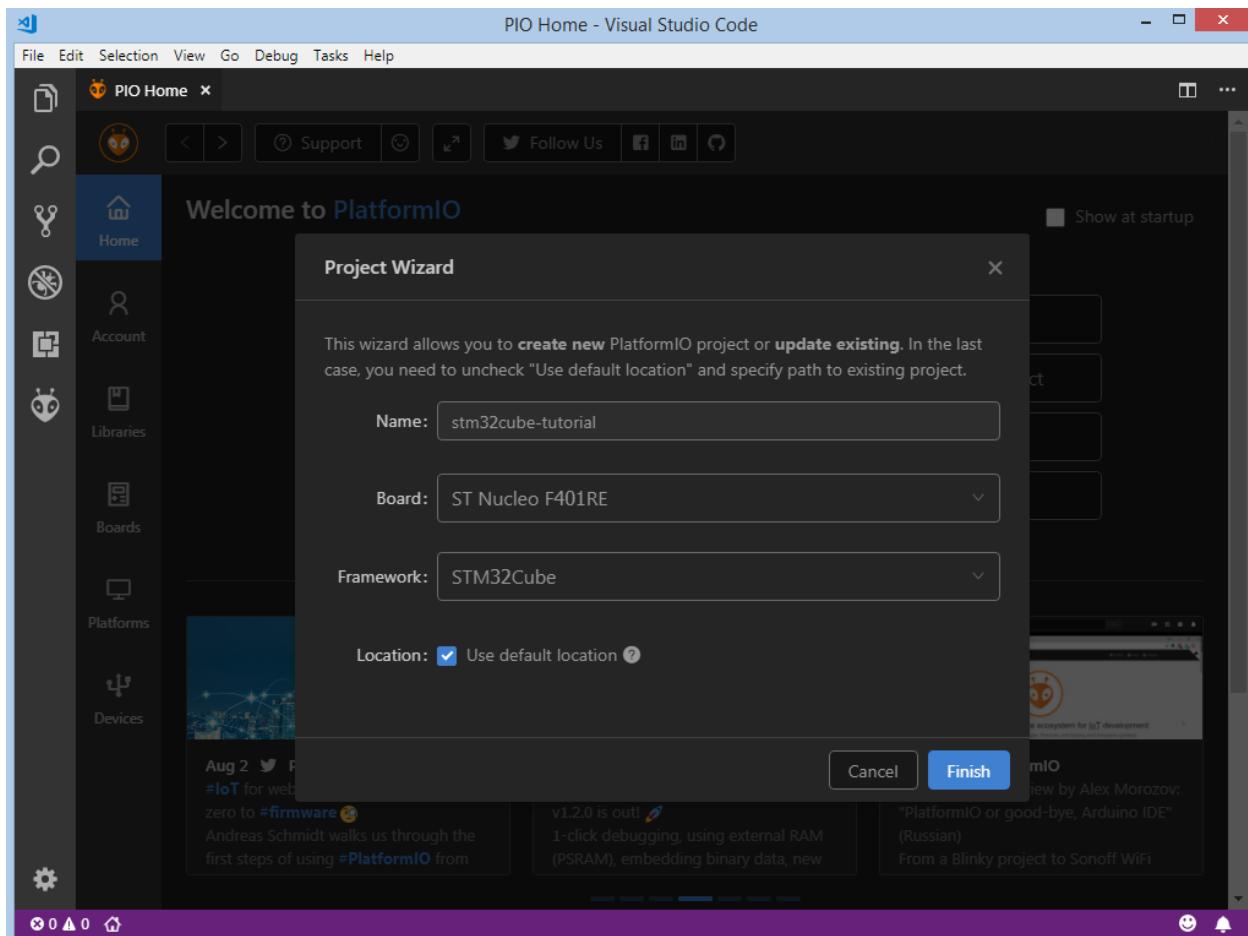
- [Adding Code to the Generated Project](#)
- [Compiling and Uploading the Firmware](#)
- [Debugging the Firmware](#)
- [Writing Unit Tests](#)
- [Conclusion](#)
- [Project Source Code](#)

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):



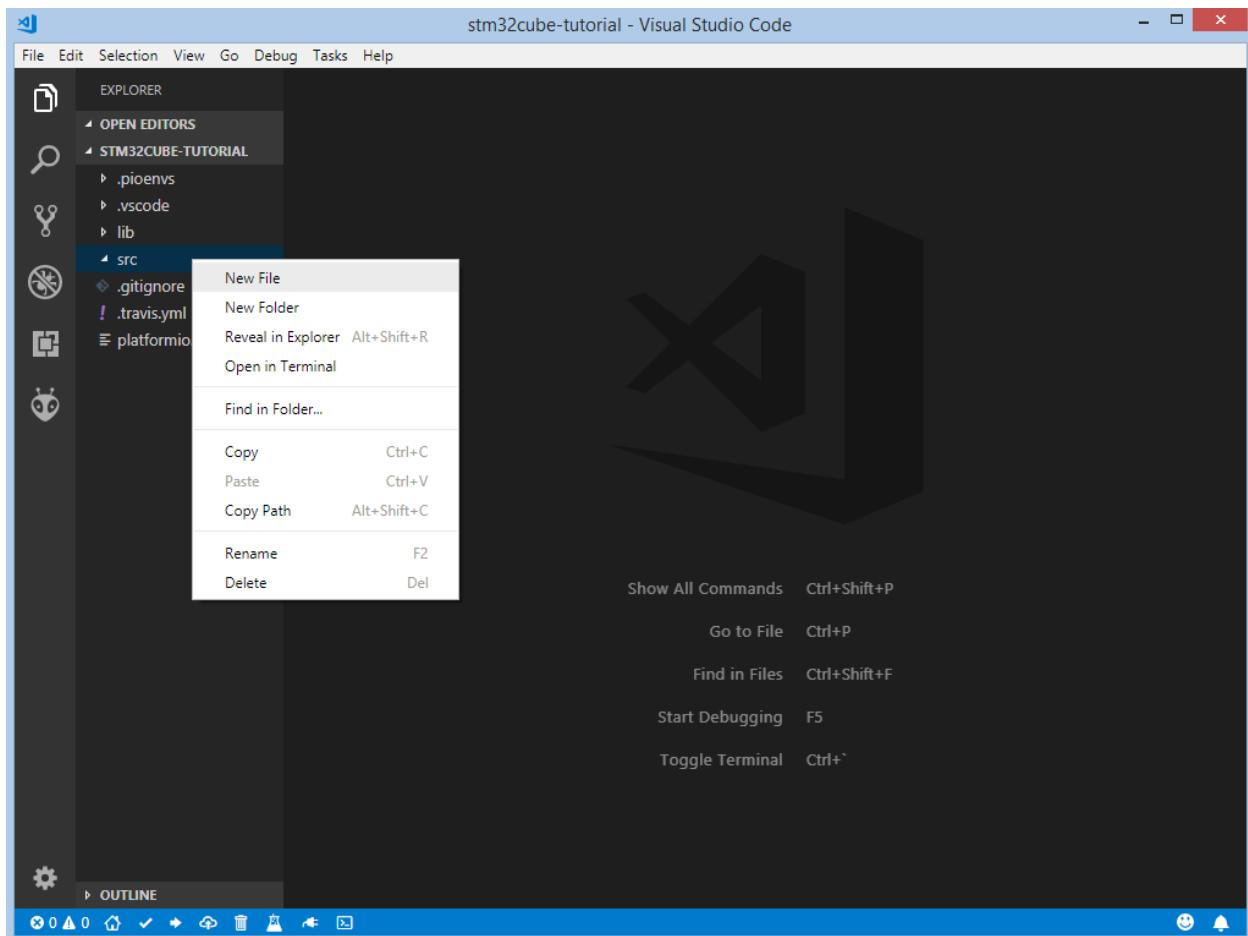
On the next step, we need to select ST Nucleo-F401RE as a development board, *STM32Cube* as a framework and a path to the project location (or use the default one):



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:

```
#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:

```
#include "main.h"

void LED_Init();

int main(void) {
    HAL_Init();
    LED_Init();

    while (1)
    {
```

(continues on next page)

(continued from previous page)

```
    HAL_GPIO_TogglePin(LED_GPIO_PORT, LED_PIN);
    HAL_Delay(1000);
}

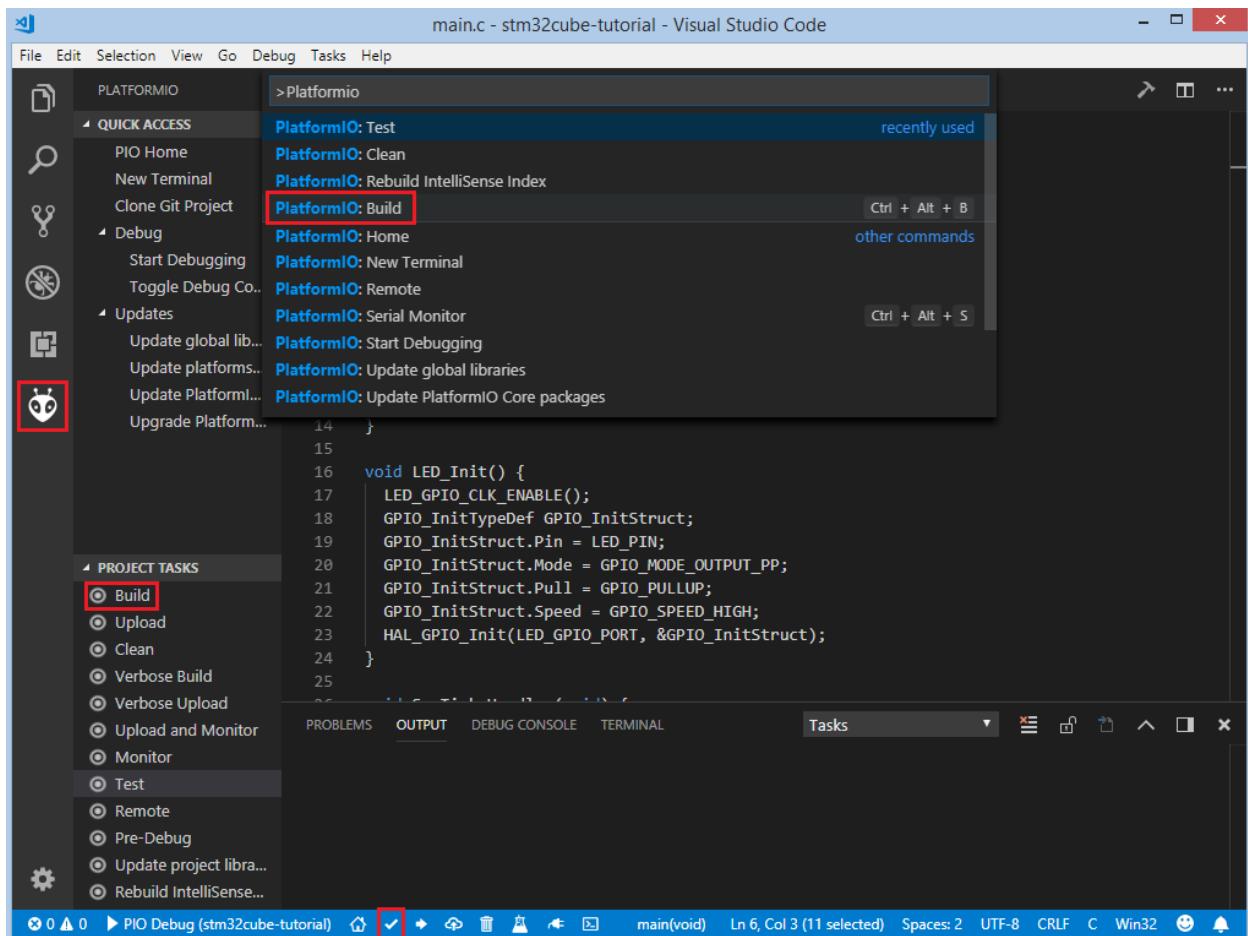
void LED_Init() {
    LED_GPIO_CLK_ENABLE();
    GPIO_InitTypeDef GPIO_InitStruct;
    GPIO_InitStruct.Pin = LED_PIN;
    GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
    GPIO_InitStruct.Pull = GPIO_PULLUP;
    GPIO_InitStruct.Speed = GPIO_SPEED_HIGH;
    HAL_GPIO_Init(LED_GPIO_PORT, &GPIO_InitStruct);
}

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:

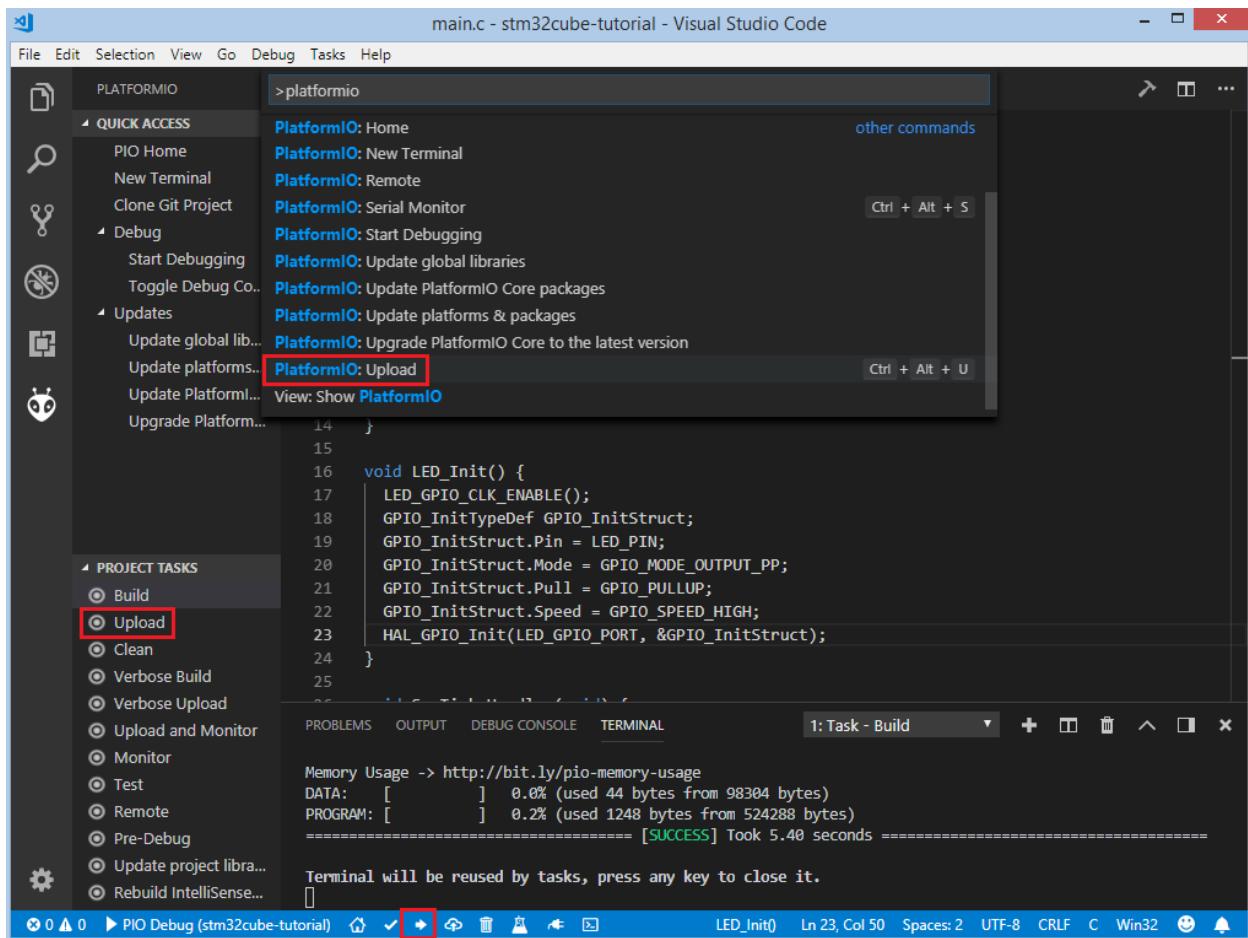
The screenshot shows the Visual Studio Code interface with a PlatformIO project named "stm32cube-tutorial". The Explorer sidebar shows the project structure with files main.c and main.h under the src folder. The main.c file contains C code for initializing an LED using HAL and GPIO functions. The terminal window displays the build logs, which include memory usage statistics and a success message indicating the task took 7.13 seconds.

```

main.c - stm32cube-tutorial - Visual Studio Code
File Edit Selection View Go Debug Tasks Help
EXPLORER
OPEN EDITORS
STM32CUBE-TUTORIAL
  .pioenvs
  .vscode
  lib
  src
    main.c
    main.h
    .gitignore
    .travis.yml
    platformio.ini
C main.c x C main.h
1 #include "main.h"
2
3 void LED_Init();
4
5 int main(void) {
6     HAL_Init();
7     LED_Init();
8
9     while (1)
10    {
11        HAL_GPIO_TogglePin(LED_GPIO_PORT, LED_PIN);
12        HAL_Delay(1000);
13    }
14}
15
16 void LED_Init()
17 {
18     LED_GPIO_CLK_ENABLE();
19     GPIO_InitTypeDef GPIO_InitStruct;
20     GPIO_InitStruct.Pin = LED_PIN;
21     GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
22     GPIO_InitStruct.Pull = GPIO_PULLUP;
23     GPIO_InitStruct.Speed = GPIO_SPEED_HIGH;
24     HAL_GPIO_Init(LED_GPIO_PORT, &GPIO_InitStruct);
}
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
1: Task - Build
Checking size .pioenvs\nucleo_f401re\firmware.elf
Building .pioenvs\nucleo_f401re\firmware.bin
Memory Usage -> http://bit.ly/pio-memory-usage
DATA: [ ] 0.0% (used 44 bytes from 98304 bytes)
PROGRAM: [ ] 0.2% (used 1248 bytes from 524288 bytes)
===== [SUCCESS] Took 7.13 seconds =====
Terminal will be reused by tasks, press any key to close it.
(Global Scope) Ln 28, Col 2 Spaces: 4 UTF-8 CRLF C Win32 ☺

```

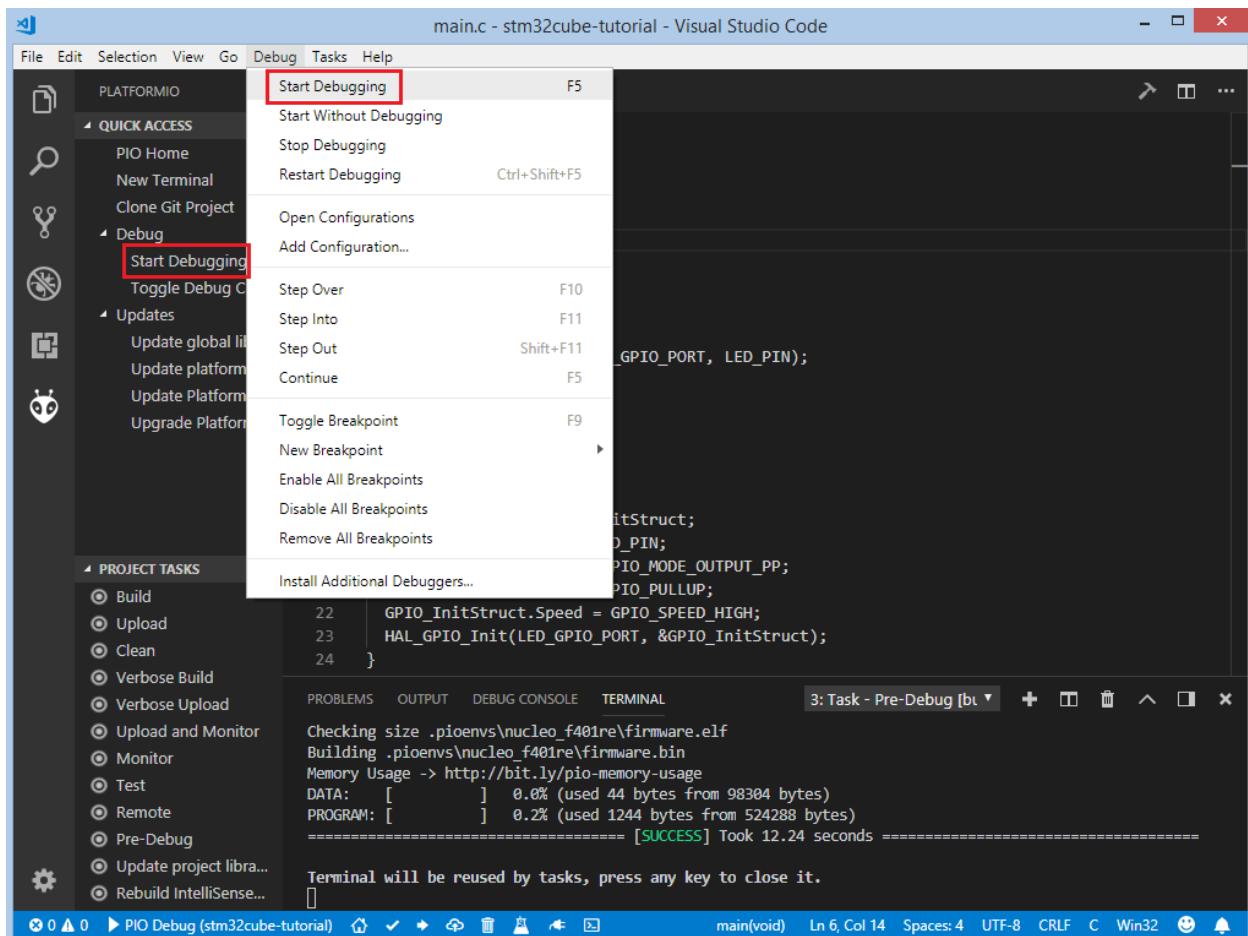
To upload the firmware to the board we can use next options: Upload option on the Project Tasks menu, Upload button on *PlatformIO Toolbar*, using Command Palette View: Command Palette > PlatformIO: Upload, using Task Menu Tasks: Run Task... > PlatformIO: Upload or via hotkeys cmd+alt+u / ctrl+alt+u:



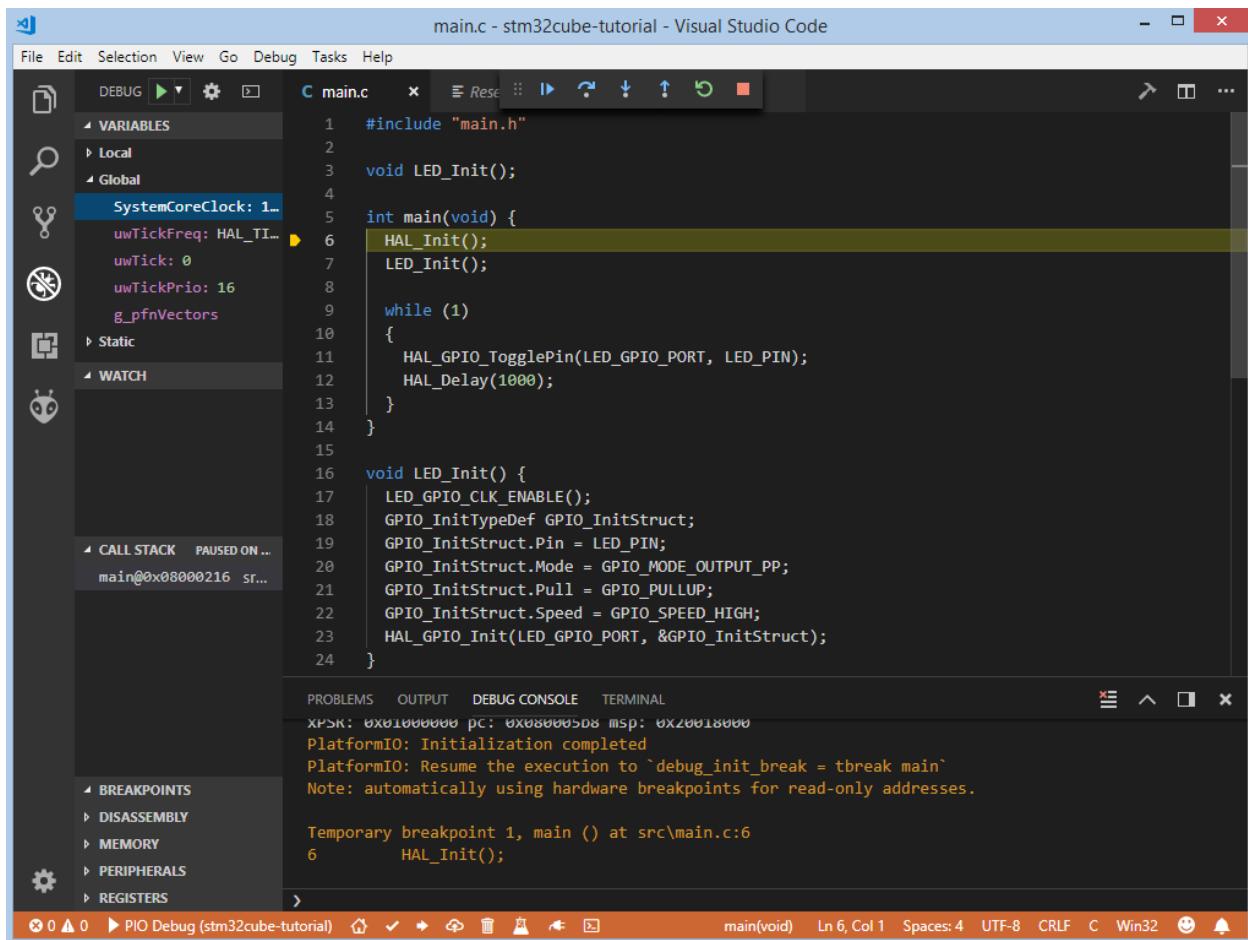
After successful uploading, the green LED2 should start blinking.

Debugging the Firmware

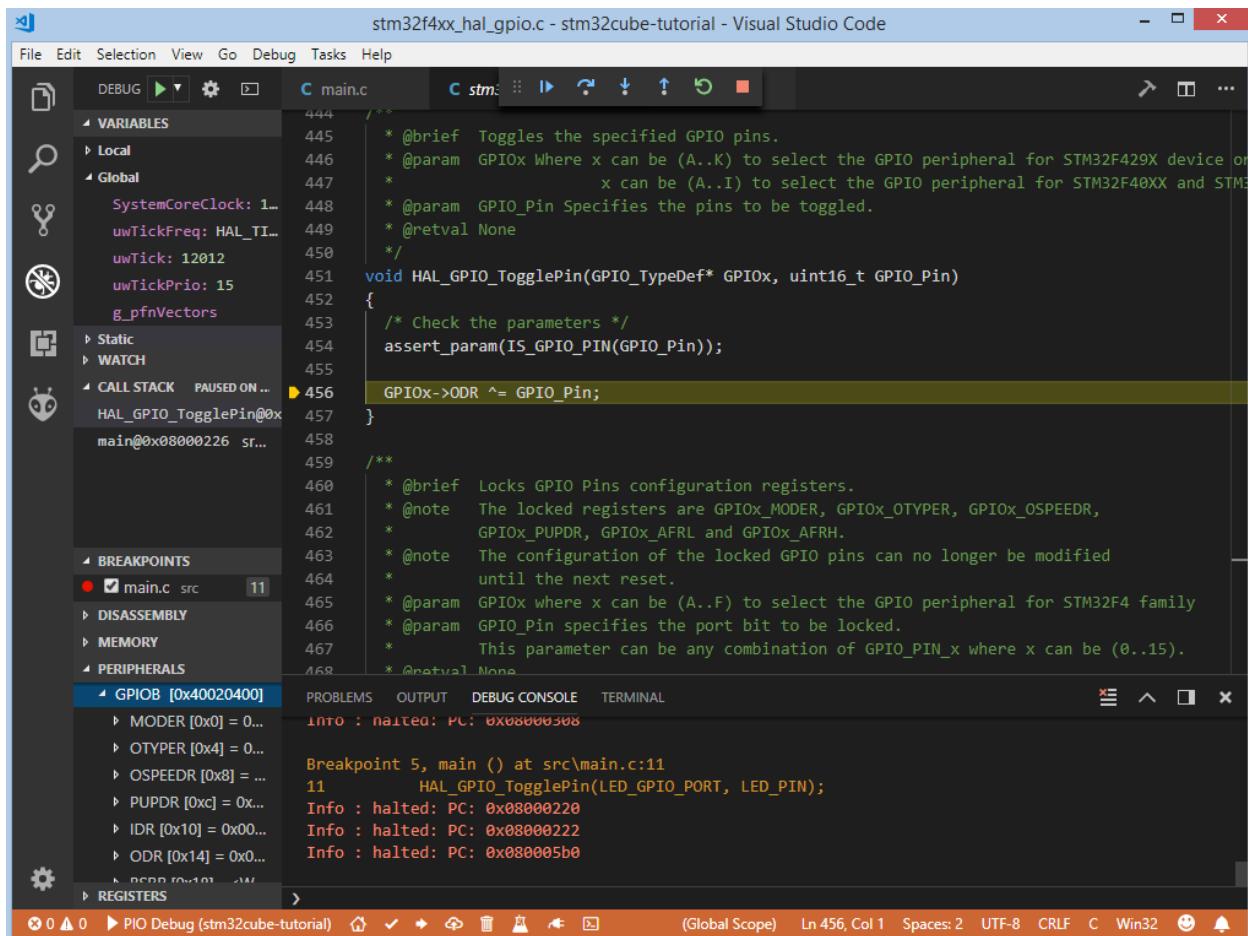
PIO Unified Debugger offers the easiest way to debug your board. To start debugging session you can use Start debugging option in PlatformIO Quick Access menu, Debug: Start debugging from the top menu or hotkey button F5:



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:

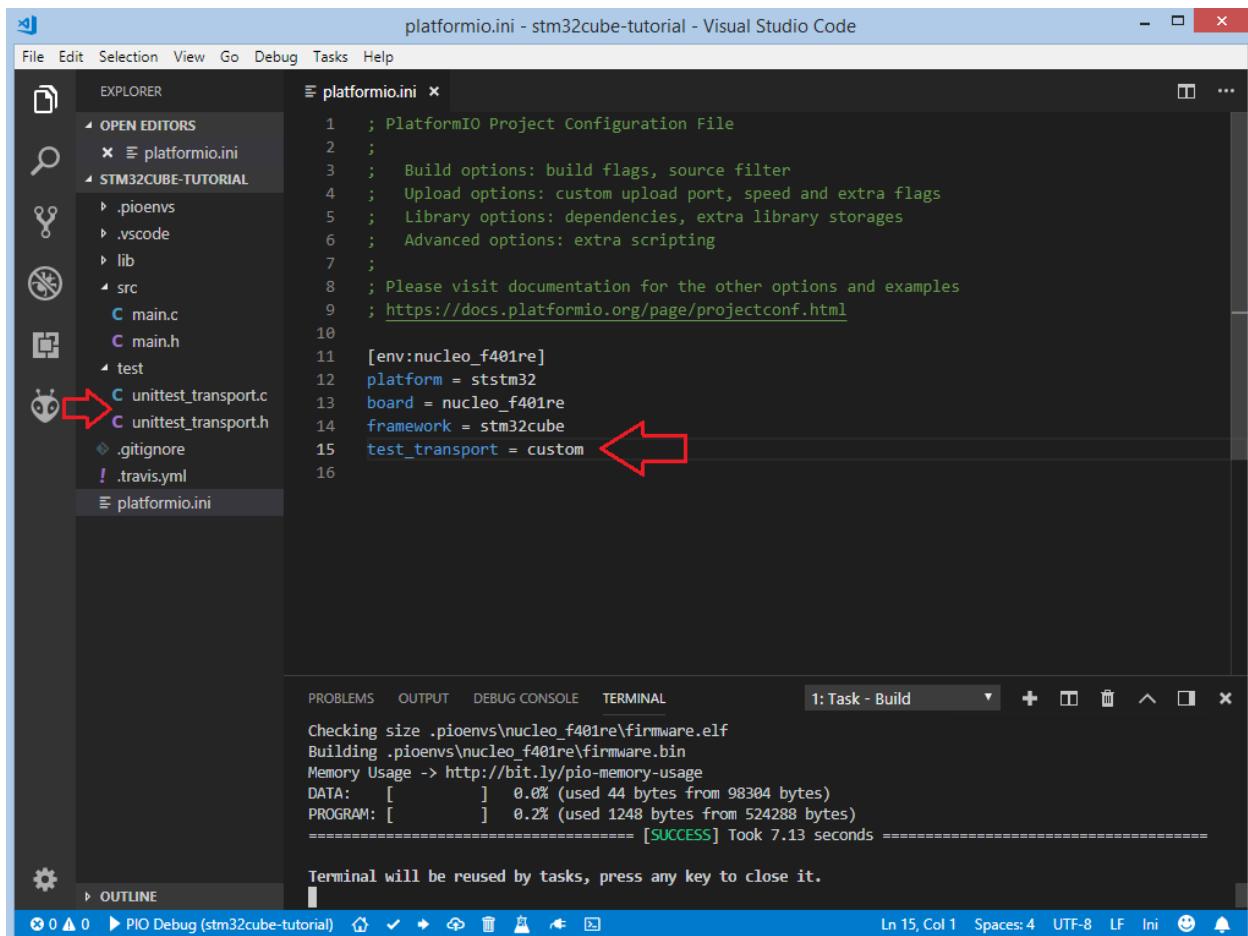


Writing Unit Tests

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](#)):

```
[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:

```
void unittest_uart_begin();
void unittest_uart_putchar(char c);
void unittest_uart_flush();
void unittest_uart_end();
```

Implementation of `unittest_transport.h`:

```
#ifndef UNITTEST_TRANSPORT_H
#define UNITTEST_TRANSPORT_H

#ifndef __cplusplus
extern "C" {
#endif

void unittest_uart_begin();
void unittest_uart_putchar(char c);
void unittest_uart_flush();
void unittest_uart_end();
```

(continues on next page)

(continued from previous page)

```
#ifdef __cplusplus
}
#endif

#endif // UNITEST_TRANSPORT_H
```

Implementation of unittest_transport.c:

```
#include "unittest_transport.h"
#include "stm32f4xx_hal.h"

#define USARTx
#define USARTx_CLK_ENABLE()
#define USARTx_CLK_DISABLE()
#define USARTx_RX_GPIO_CLK_ENABLE()
#define USARTx_TX_GPIO_CLK_ENABLE()
#define USARTx_RX_GPIO_CLK_DISABLE()
#define USARTx_TX_GPIO_CLK_DISABLE()

#define USARTx_FORCE_RESET()
#define USARTx_RELEASE_RESET()

#define USARTx_TX_PIN
#define USARTx_TX_GPIO_PORT
#define USARTx_TX_AF
#define USARTx_RX_PIN
#define USARTx_RX_GPIO_PORT
#define USARTx_RX_AF

static UART_HandleTypeDef UartHandle;

void unittest_uart_begin()
{
    GPIO_InitTypeDef GPIO_InitStruct;

    USARTx_TX_GPIO_CLK_ENABLE();
    USARTx_RX_GPIO_CLK_ENABLE();

    USARTx_CLK_ENABLE();

    GPIO_InitStruct.Pin      = USARTx_TX_PIN;
    GPIO_InitStruct.Mode     = GPIO_MODE_AF_PP;
    GPIO_InitStruct.Pull     = GPIO_PULLUP;
    GPIO_InitStruct.Speed    = GPIO_SPEED_FAST;
    GPIO_InitStruct.Alternate = USARTx_TX_AF;

    HAL_GPIO_Init(USARTx_TX_GPIO_PORT, &GPIO_InitStruct);

    GPIO_InitStruct.Pin = USARTx_RX_PIN;
    GPIO_InitStruct.Alternate = USARTx_RX_AF;

    HAL_GPIO_Init(USARTx_RX_GPIO_PORT, &GPIO_InitStruct);
    UartHandle.Instance = USARTx;

    UartHandle.Init.BaudRate = 115200;
    UartHandle.Init.WordLength = UART_WORDLENGTH_8B;
    UartHandle.Init.StopBits = UART_STOPBITS_1;
```

(continues on next page)

(continued from previous page)

```

UartHandle.Init.Parity      = UART_PARITY_NONE;
UartHandle.Init.HwFlowCtl   = UART_HWCONTROL_NONE;
UartHandle.Init.Mode        = UART_MODE_TX_RX;
UartHandle.Init.OverSampling = UART_OVERSAMPLING_16;

if(HAL_UART_Init(&UartHandle) != HAL_OK) {
    while(1) {}
}

void unittest_uart_putchar(char c)
{
    HAL_UART_Transmit(&UartHandle, (uint8_t*)(&c), 1, 1000);
}

void unittest_uart_flush() {}

void unittest_uart_end() {
    USARTx_CLK_DISABLE();
    USARTx_RX_GPIO_CLK_DISABLE();
    USARTx_TX_GPIO_CLK_DISABLE();
}

```

Now we need to add some test cases. Tests can be added to a single C file that may include multiple tests. First of all, we need to add three default functions: `setUp`, `tearDown` and `main`. `setUp` and `tearDown` are used to initialize and finalize test conditions. Implementations of these functions are not required for running tests but if you need to initialize some variables before you run a test, you use the `setUp` function and if you need to clean up variables you use `tearDown` function. In our example, we will use these functions to accordingly initialize and deinitialize LED. `main` function acts as a simple program where we describe our test plan.

Let's add a new file `test_main.c` to the folder `test`. Next basic tests for blinking routine will be implemented in this file:

- `test_led_builtin_pin_number` ensures that `LED_PIN` has the correct value
- `test_led_state_high` tests functions `HAL_GPIO_WritePin` and `HAL_GPIO_ReadPin` with `GPIO_PIN_SET` value
- `test_led_state_low` tests functions `HAL_GPIO_WritePin` and `HAL_GPIO_ReadPin` with `GPIO_PIN_RESET` value

Note:

- 2 sec delay is required since the board doesn't support software resetting via `Serial.DTR/RTS`

```

#include ".../src/main.h"
#include <unity.h>

void setUp(void) {
    LED_GPIO_CLK_ENABLE();
    GPIO_InitTypeDef GPIO_InitStruct;
    GPIO_InitStruct.Pin = LED_PIN;
    GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
    GPIO_InitStruct.Pull = GPIO_PULLUP;
    GPIO_InitStruct.Speed = GPIO_SPEED_HIGH;
}

```

(continues on next page)

(continued from previous 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);
    }

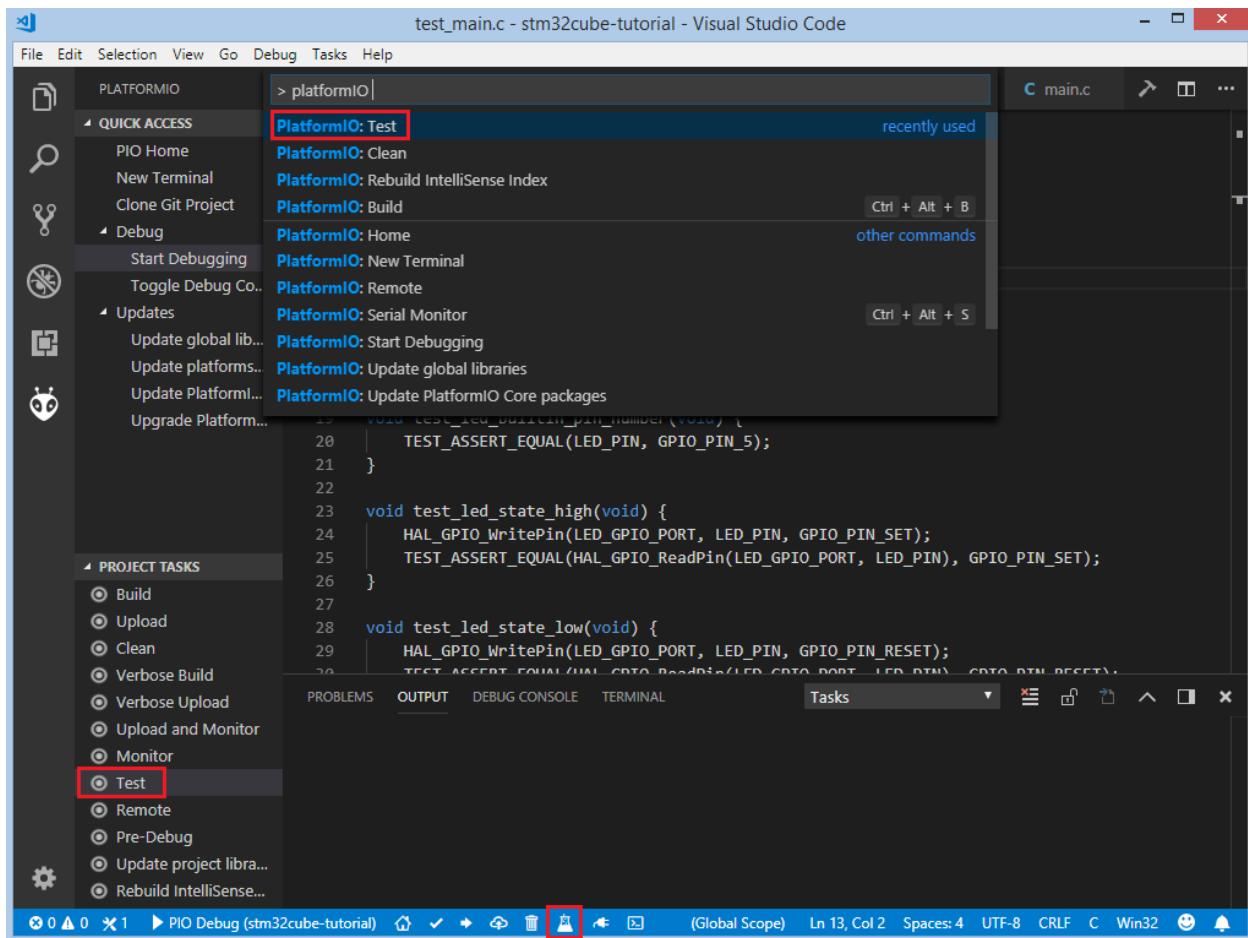
    UNITY_END(); // stop unit testing

    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 Toolbar*:



After processing we should see a detailed report about the testing results:

```

test\test_main.c:37:test_led_builtin_pin_number [PASSED]
test\test_main.c:41:test_led_state_high [PASSED]
test\test_main.c:43:test_led_state_low [PASSED]
-----
11 Tests 0 Failures 0 Ignored
=====
===== [TEST SUMMARY] =====
test\*/env:nucleo_f401re [PASSED]
===== [PASSED] Took 16.81 seconds =====

```

Congratulations! As we can see from the report, all our tests went successfully!

Conclusion

Now we have a decent template that we can improve for our next more complex projects.

Project Source Code

The source code of this tutorial is available at <https://github.com/platformio/platformio-examples/tree/develop/unit-testing/stm32cube>

Arduino and Nordic nRF52-DK: debugging and unit testing

The goal of this tutorial is to demonstrate how simple it is to use *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*

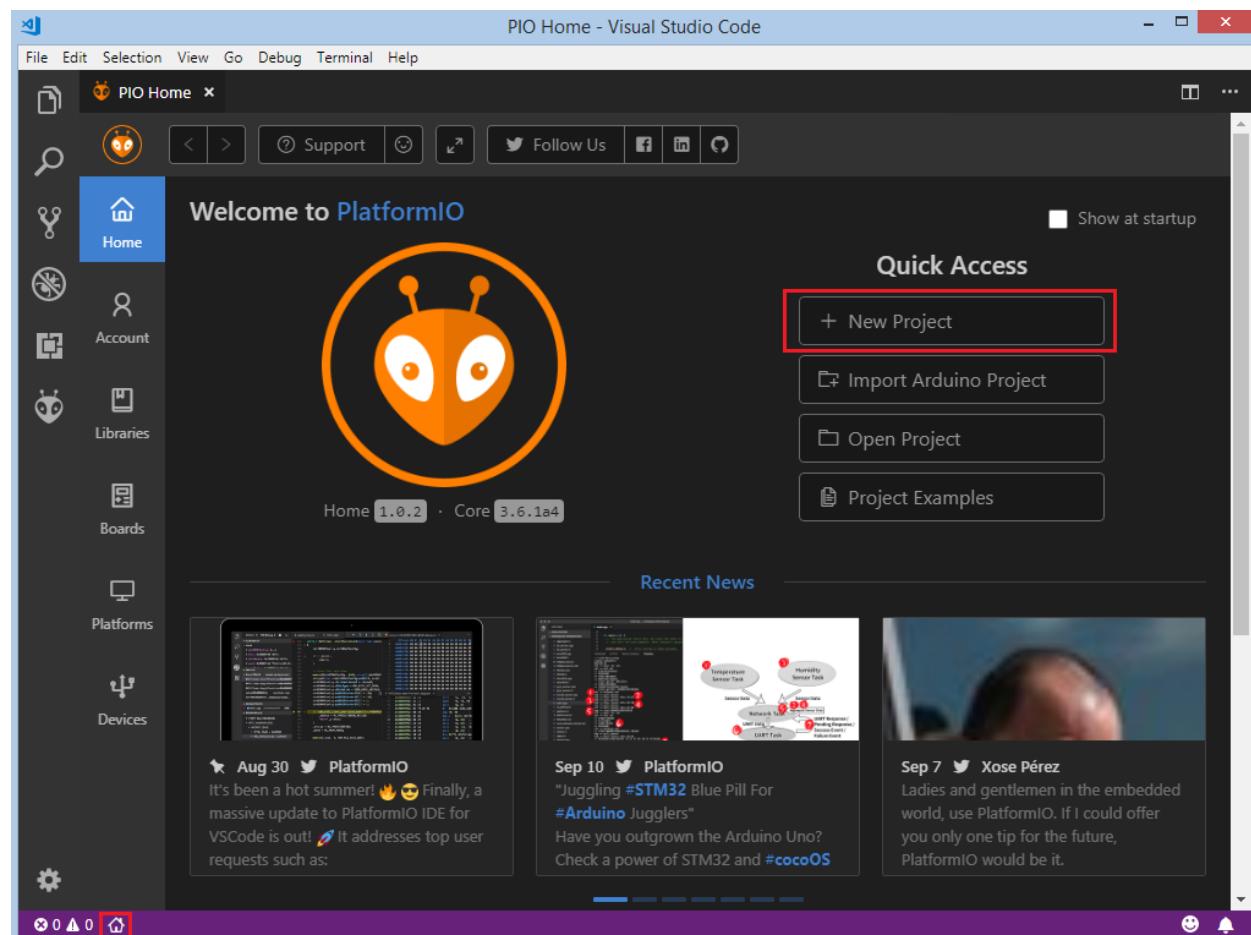
- Install drivers for [J-LINK](#) debug tool
- Nordic nRF52-DK development board

Contents

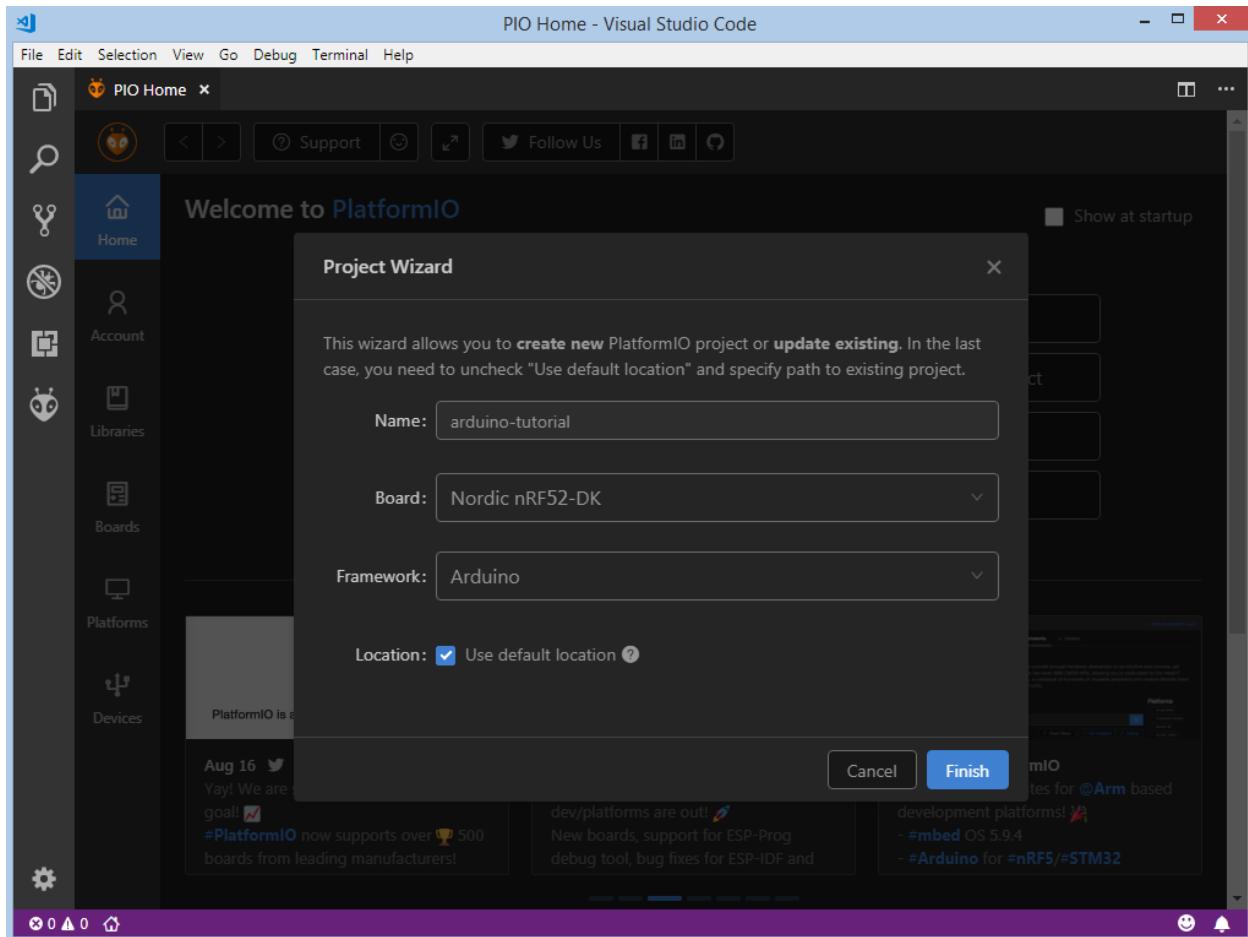
- [Setting Up the Project](#)
- [Adding Code to the Generated Project](#)
- [Compiling and Uploading the Firmware](#)
- [Debugging the Firmware](#)
- [Writing Unit Tests](#)
- [Adding Bluetooth LE features](#)
- [Conclusion](#)

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:

```
#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)

(continued from previous page)

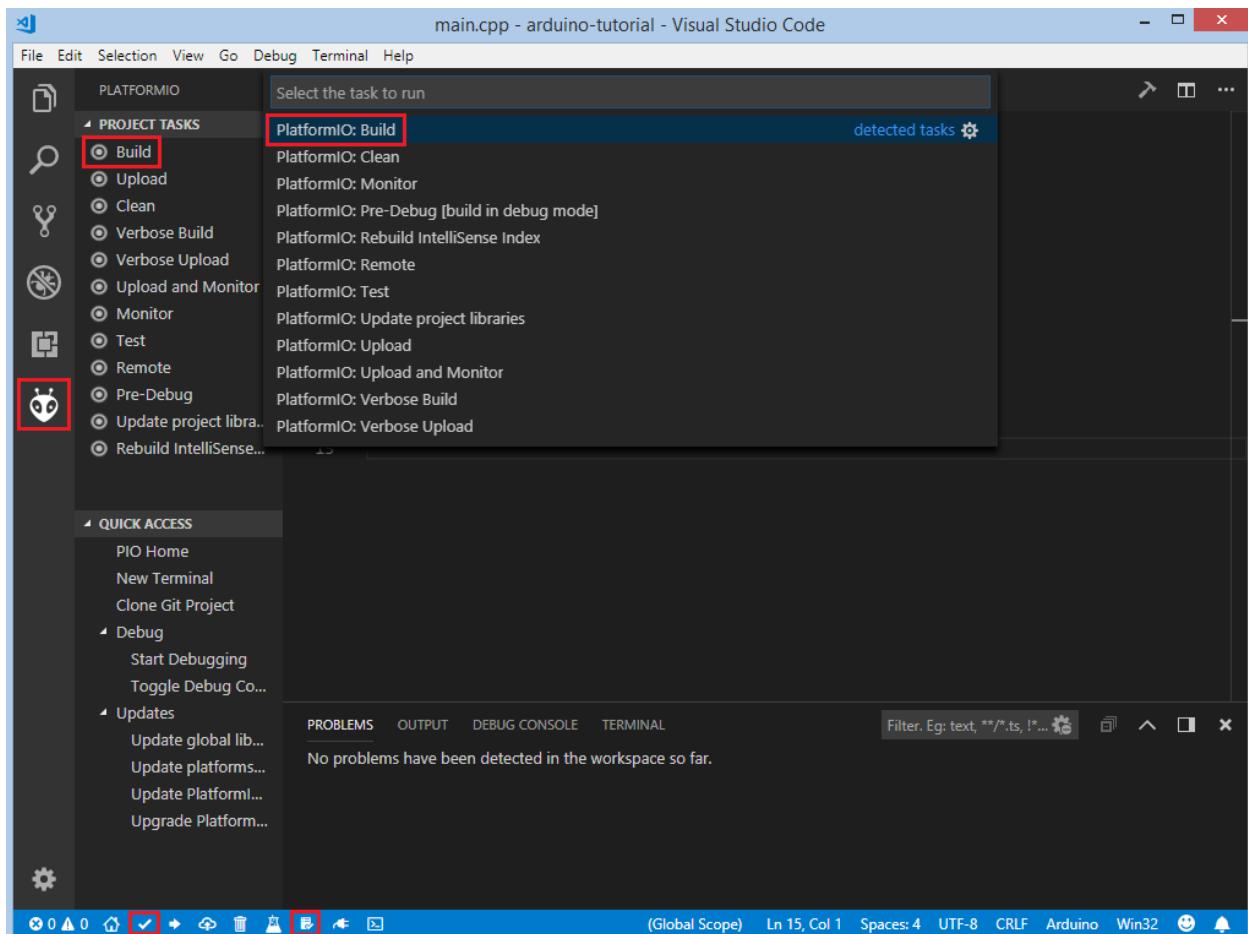
```
delay(100);
```

```
File Edit Selection View Go Debug Terminal Help  
EXPLORER main.cpp •  
OPEN EDITORS 1 UNSAVED  
ARDUINO-TUTORIAL  
.pioenvs  
.vscode  
lib  
src  
main.cpp  
.gitignore  
.travis.yml  
platformio.ini  
main.cpp  
1 #include <Arduino.h>  
2  
3 void setup()  
4 {  
5     pinMode(LED_BUILTIN, OUTPUT);  
6 }  
7  
8 void loop()  
9 {  
10    digitalWrite(LED_BUILTIN, HIGH);  
11    delay(100);  
12    digitalWrite(LED_BUILTIN, LOW);  
13    delay(100);  
14 }  
15  
OUTLINE  
(Global Scope) Ln 15, Col 1 Spaces: 4 UTF-8 CRLF Arduino Win32
```

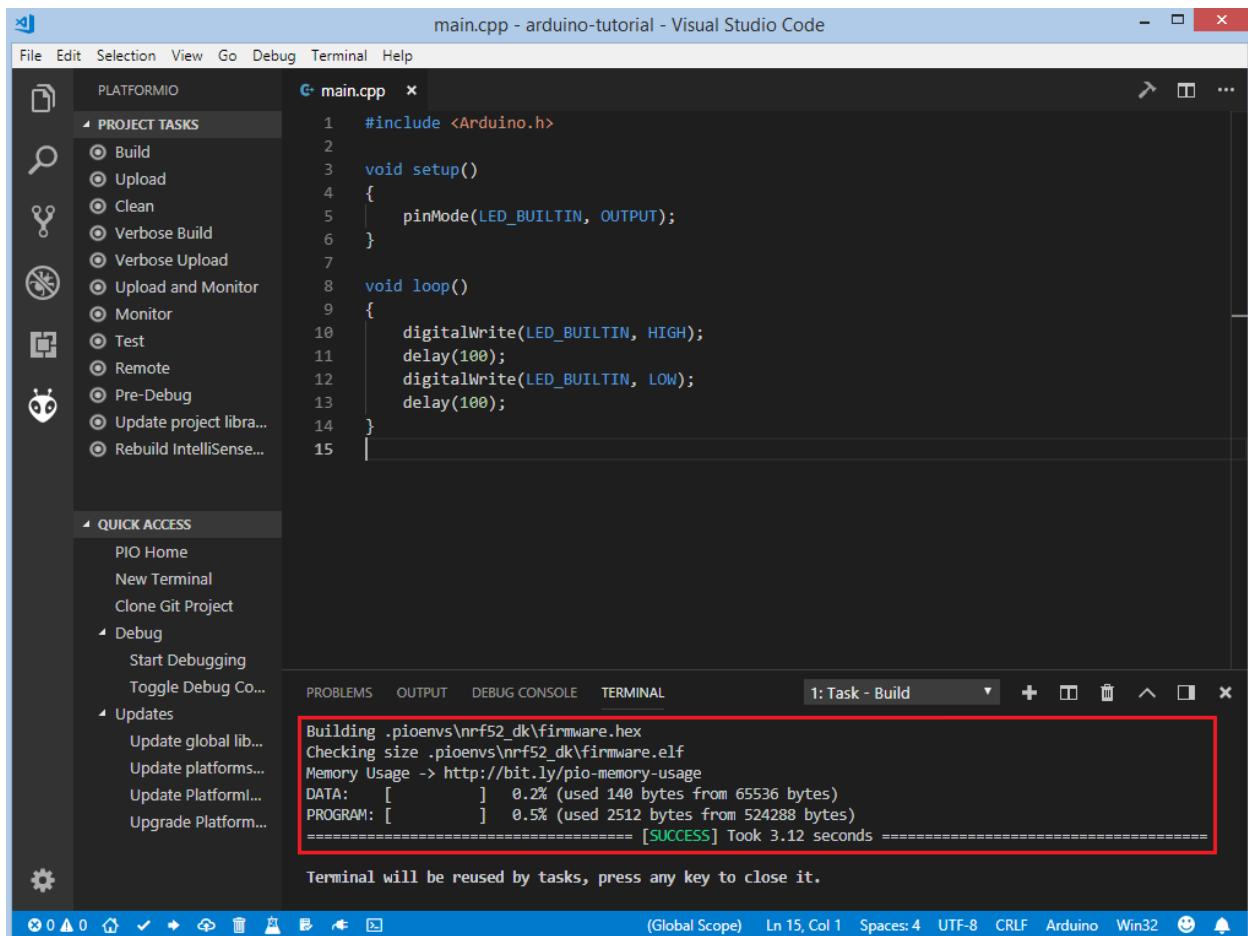
After this step, we created a basic blink project ready for compiling and uploading.

Compiling and Uploading the Firmware

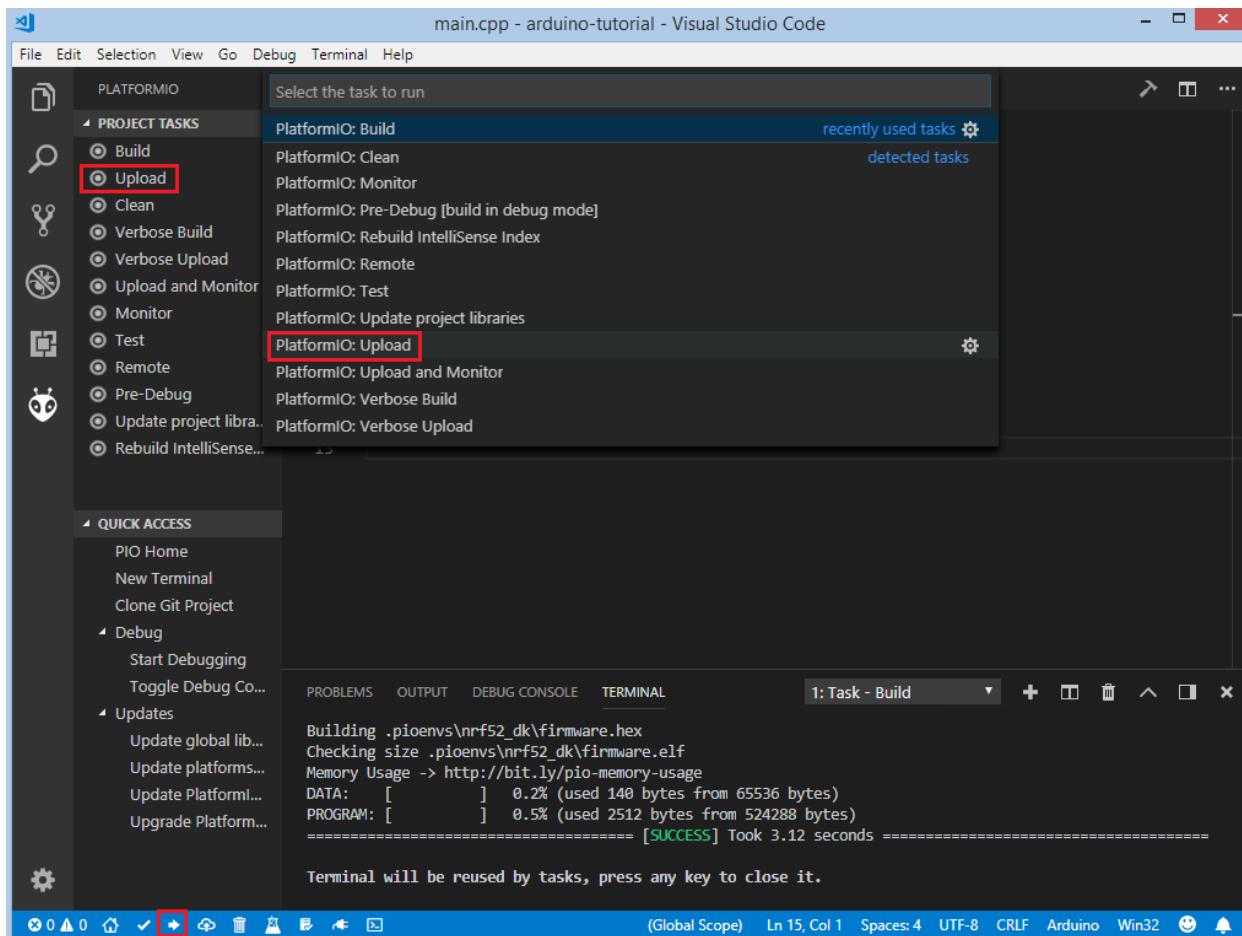
Now we can build the project. To compile firmware we can use next options: Build option from the Project Tasks menu, Build button in *PlatformIO Toolbar*, Task Menu Tasks: Run Task... > PlatformIO: Build or in *PlatformIO Toolbar*, Command Palette View: Command Palette > PlatformIO: Build or via hotkeys 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:



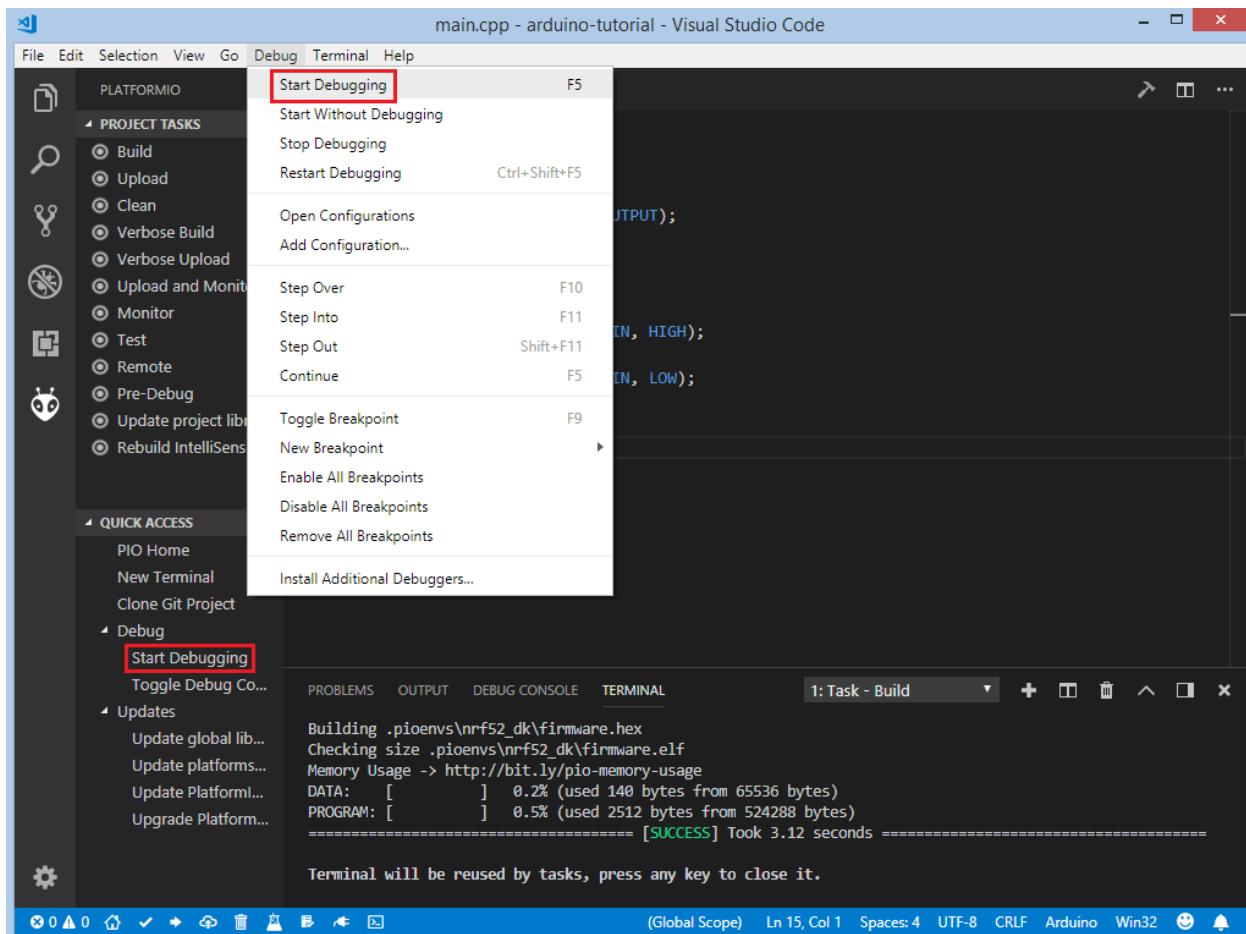
After successful uploading, the green LED1 should start blinking.

Debugging the Firmware

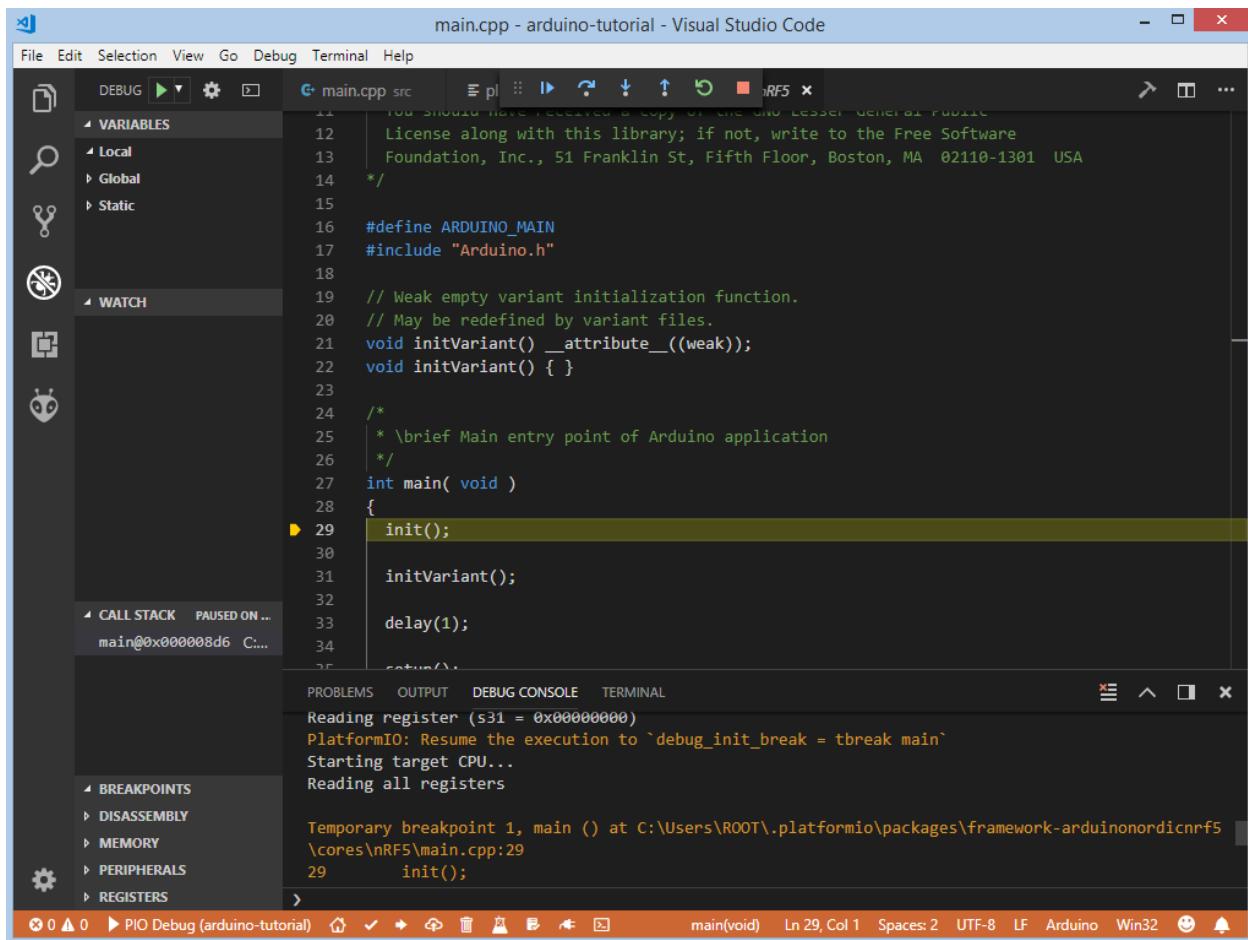
PIO Unified Debugger offers the easiest way to debug the board. Firstly, we need to specify `debug_tool` in “`platformio.ini`” (*Project Configuration File*). Since the board has an on-board JLink debug probe we can directly declare it in “`platformio.ini`” (*Project Configuration File*):

```
[env:nrf52_dk]
platform = nordicnrf52
board = nrf52_dk
framework = arduino
debug_tool = jlink
```

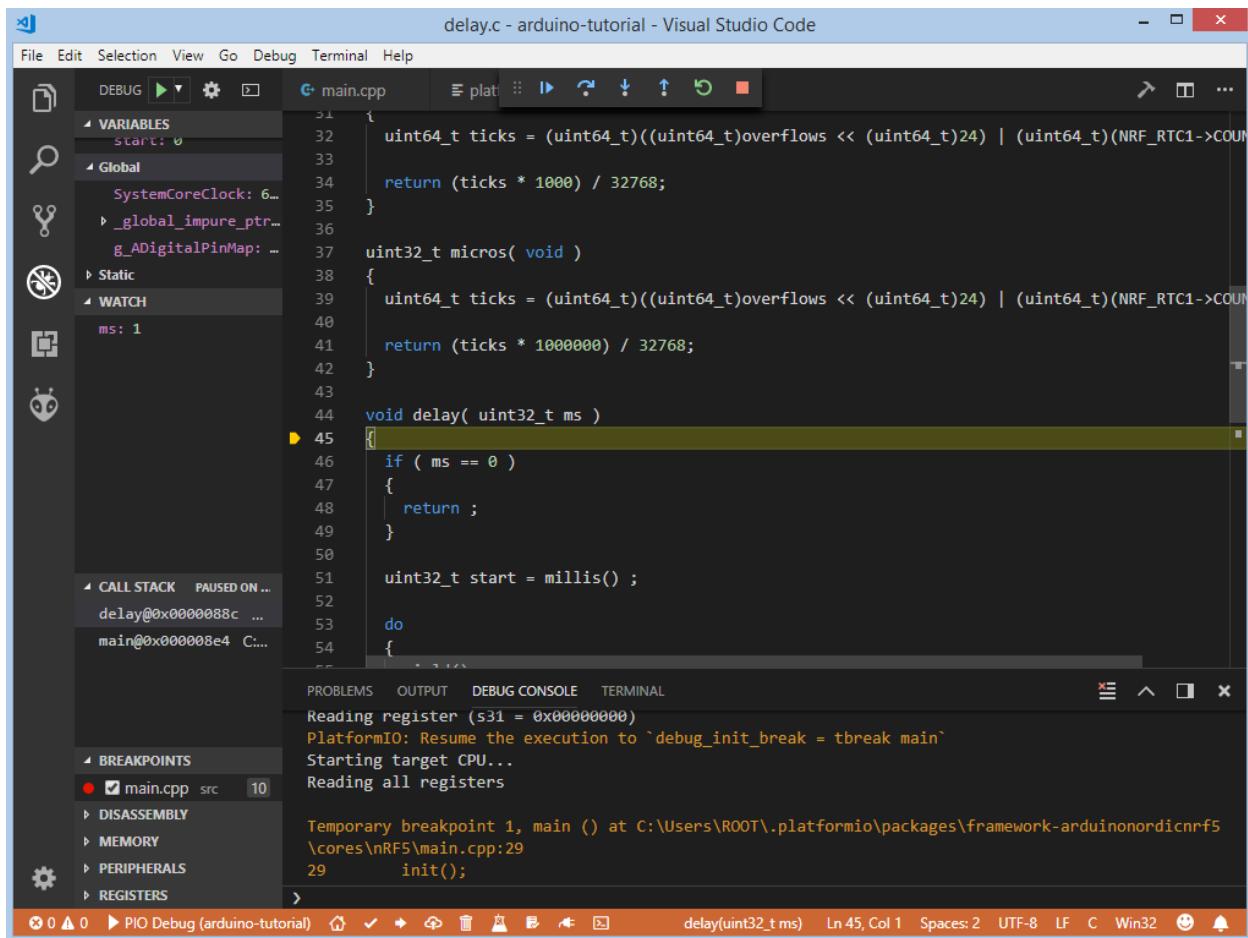
To start the debug session we can use next options: Debug: Start debugging from the top menu, Start Debugging option from Quick Access menu or hotkey button F5:



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`

```
#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)

(continued from previous page)

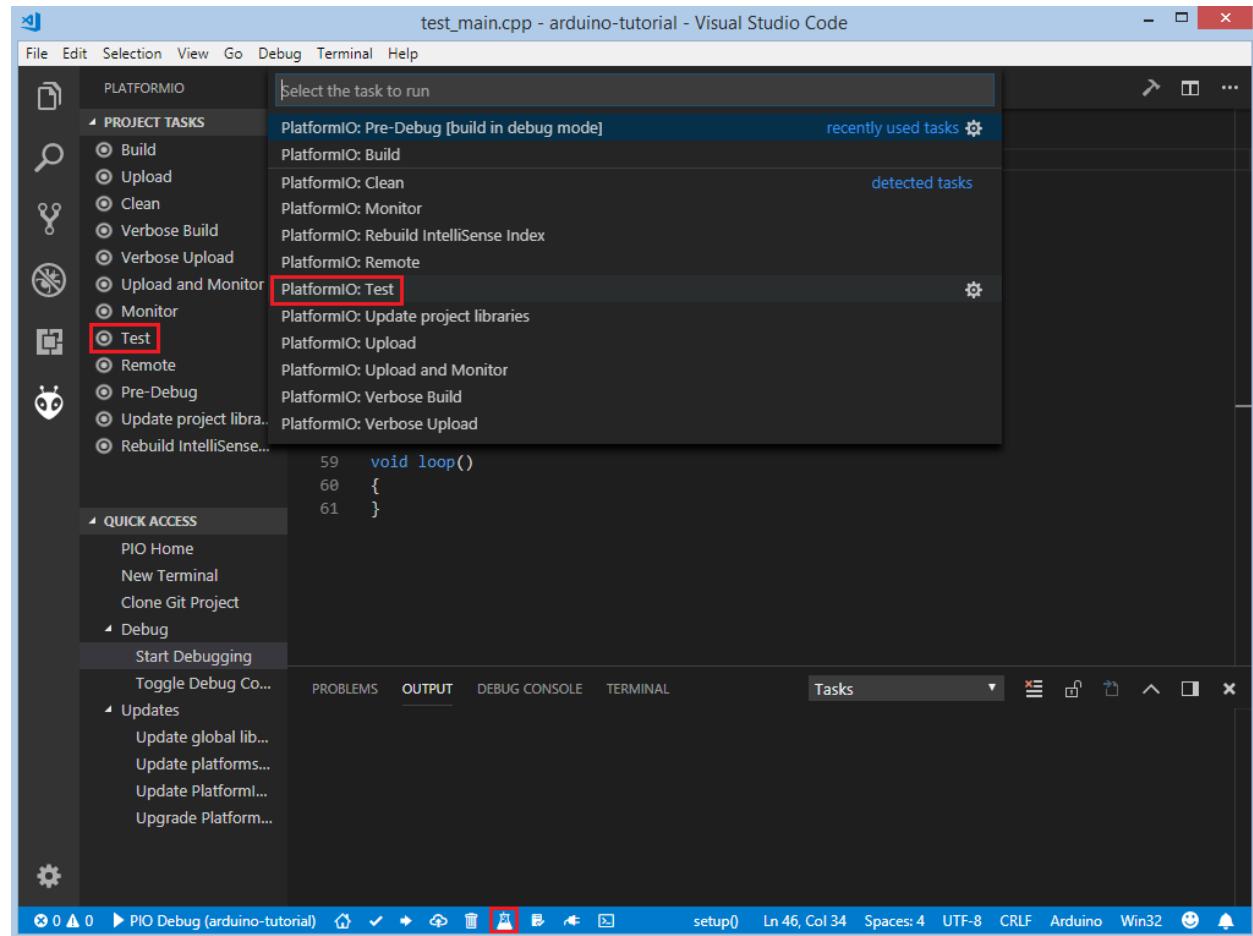
```

    UNITY_END(); // stop unit testing
}

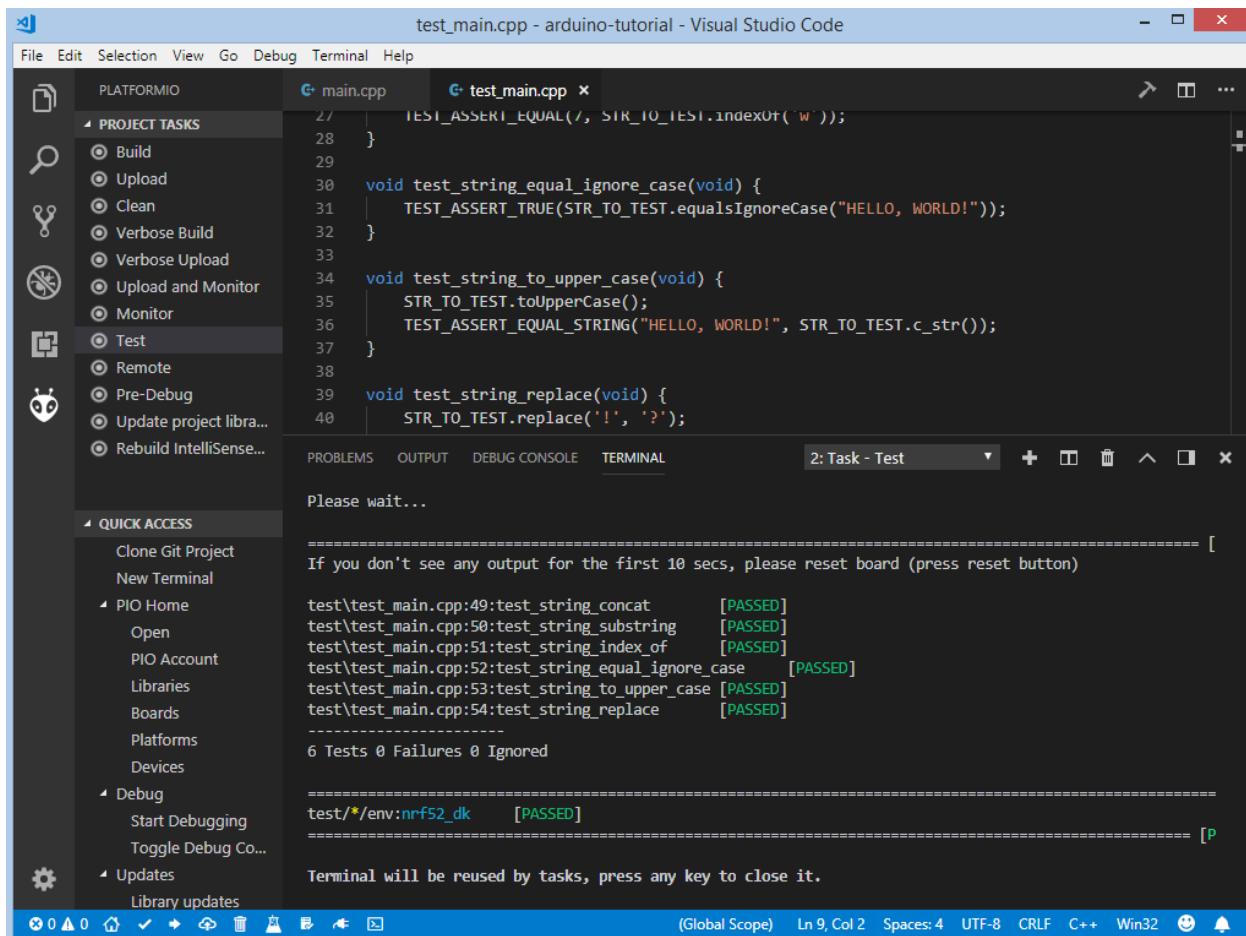
void loop()
{
}

```

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:



After processing we should see a detailed report about the testing results:



As we can see from the report, all our tests were successful!

Adding Bluetooth LE features

To add the basic BLE functionality to our project we need to define the SoftDevice version and install a library called `BLEPeripheral`. Both these modifications can be specified in “`platformio.ini`” (*Project Configuration File*):

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

```
#include <Arduino.h>
#include <SPI.h>
#include <BLEPeripheral.h>
```

(continues on next page)

(continued from previous page)

```

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();
}

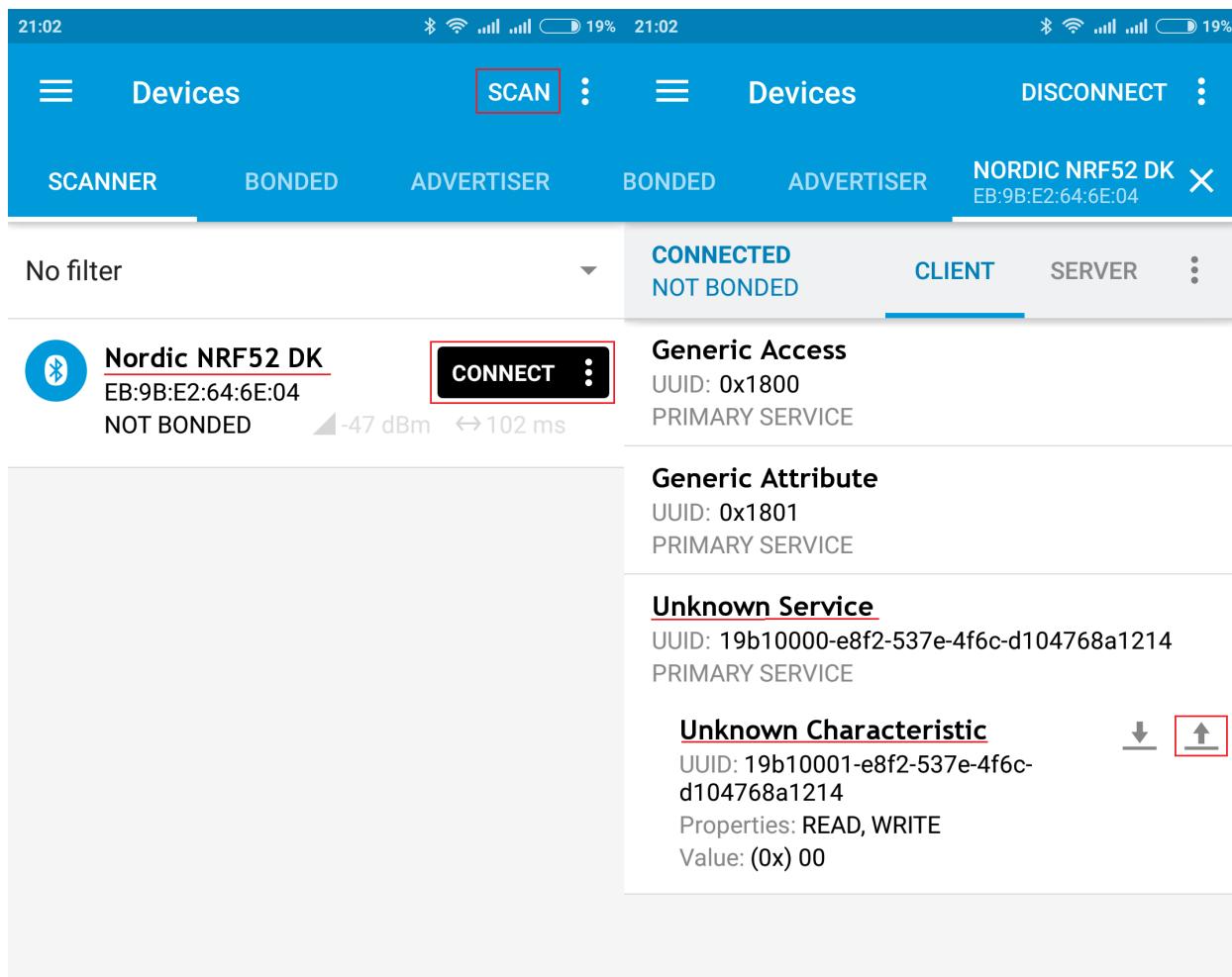
void loop()
{
    BLECentral central = ledPeripheral.central();

    if (central) {
        while (central.connected()) {
            if (ledCharacteristic.written()) {
                if (ledCharacteristic.value()) {
                    digitalWrite(LED_BUILTIN, HIGH);
                }
                else{
                    digitalWrite(LED_BUILTIN, LOW);
                }
            }
        }
    }
}

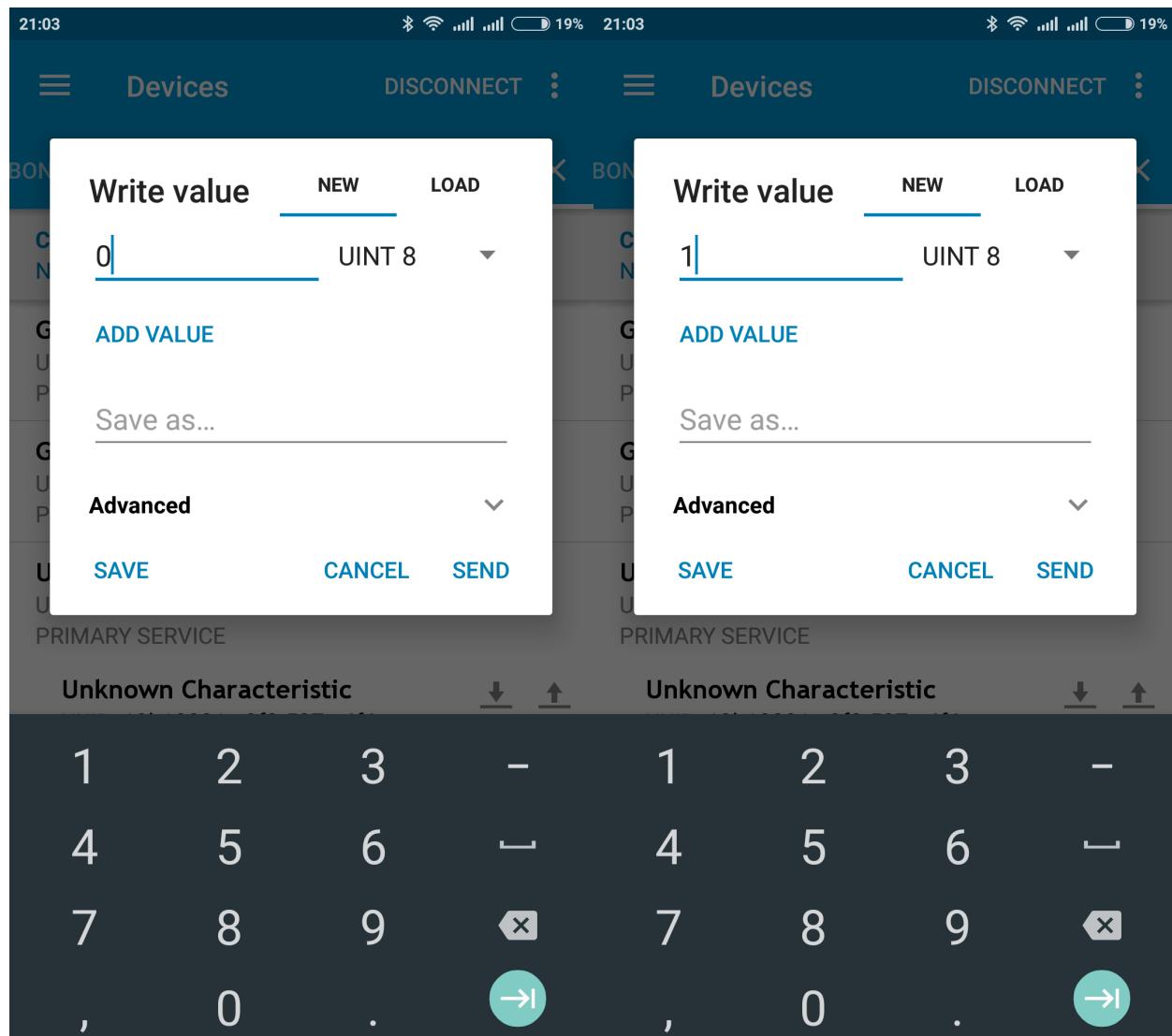
```

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 Tutorials

- Arduino In-circuit Debugging with PlatformIO
- ThingForward: First steps with PlatformIO's Unified Debugger

1.5.4 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
- workspace_dir
- build_dir
- libdeps_dir
- include_dir
- src_dir
- lib_dir
- data_dir
- test_dir
- boards_dir
- shared_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:

Pattern	Meaning
*	matches everything
?	matches any single character
[seq]	matches any character in seq
[!seq]	matches any character not in seq

Note: If you declare the same pair of “group” + “option” in an extra configuration file which was previously declared in a base “`platformio.ini`” (*Project Configuration File*), it will be overwritten with a value from extra configuration.

Example

Base “`platformio.ini`”

```
[platformio]
extra_configs =
    extra_envs.ini
    extra_debug.ini

; Global data for all [env:***]
[env]
platform = espressif32
framework = espidf

; Custom data group
; can be used in [env:***] via ${common.***}
[common]
debug_flags = -D RELEASE
lib_flags = -lc -lm

[env:esp-wrover-kit]
board = esp-wrover-kit
build_flags = ${common.debug_flags}
```

“extra_envs.ini”

```
[env:esp32dev]
board = esp32dev
build_flags = ${common.lib_flags} ${common.debug_flags}

[env:lolin32]
platform = espressif32
framework = espidf
board = lolin32
build_flags = ${common.debug_flags}
```

“extra_debug.ini”

```
# Override base "common.debug_flags"
[common]
debug_flags = -D DEBUG=1

[env:lolin32]
build_flags = -Og
```

After a parsing process, configuration state will be the next:

```
[common]
debug_flags = -D DEBUG=1
lib_flags = -lc -lm

[env:esp-wrover-kit]
platform = espressif32
framework = espidf
board = esp-wrover-kit
build_flags = ${common.debug_flags}

[env:esp32dev]
platform = espressif32
framework = espidf
board = esp32dev
```

(continues on next page)

(continued from previous page)

```
build_flags = ${common.lib_flags} ${common.debug_flags}

[env:lolin32]
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 PlatfrmIO 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
```

`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`.

Note: If you have any problems with building your project environments which are defined in “`platformio.ini`” (*Project Configuration File*), then **TRY TO DELETE** this folder. In this situation you will remove all cached files without any risk. Also, you can use “clean” target for `platformio run --target` command.

`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/include` 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:

```
[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:

```
[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

Platform 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
```

(continues on next page)

(continued from previous page)

```
[env:specific_major_and_minor_version]
platform = atmelavr@~1.2.3

[env:development_verion_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#921855a9c530082efddb5d48b44c3f4be0e2dfa2
```

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 =
; use GCC AVR 5.0+
toolchain-gccarmnoneabi@>1.50000.0

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

[env:external_package]
platform = ststm32
platform_packages =
; latest openOCD from PlatformIO Package Registry
tool-openocd

; source code of ST-Link
tool-stlink-source @ https://github.com/texane/stlink.git
```

framework

Type: String | Multiple: Yes

Frameworks name.

Board options

- *board*
- *board_build.mcu*
- *board_build.f_cpu*
- *More options*

board

Type: String | 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:

Format	Scope	Description
-D name	CPPDE-FINES	Predefine <i>name</i> as a macro, with definition 1.
-D name=definition	CPPDE-FINES	The contents of <i>definition</i> are tokenized and processed as if they appeared during translation phase three in a #define directive.
-U name	CPPDE-FINES	Cancel any previous definition of <i>name</i> , either built in or provided with a -D option.
-Wp,option	CPPFLAGS	Bypass the compiler driver and pass <i>option</i> directly through to the preprocessor
-Wall	CCFLAGS	Turns on all optional warnings which are desirable for normal code.
-Werror	CCFLAGS	Make all warnings into hard errors. Source code which triggers warnings will be rejected.
-w	CCFLAGS	Suppress all warnings, including those which GNU CPP issues by default.
-include file	CCFLAGS	Process <i>file</i> as if #include "file" appeared as the first line of the primary source file.
-I dir	CPPPATH	Add the directory <i>dir</i> to the list of directories to be searched for header files.
-Wa,option	ASFLAGS, CCFLAGS	Pass <i>option</i> as an option to the assembler. If <i>option</i> contains commas, it is split into multiple options at the commas.
-Wl,option	LINKFLAGS	Pass <i>option</i> as an option to the linker. If <i>option</i> contains commas, it is split into multiple options at the commas.
-l library	LIBS	Search the <i>library</i> named library when linking
-L dir	LIBPATH	Add directory <i>dir</i> to the list of directories to be searched for -l.

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:specificDefines]
build_flags =
    -DFOO=DBAR=1
    -D BUILD_ENV_NAME=$PIOENV
    -D CURRENT_TIME=$UNIX_TIME
    -DFLOAT_VALUE=1.23457e+07

[env:stringDefines]
build_flags =
    -DHELLO="World!"
    '-DWIFI_PASS="My password"'
    ; Password with special chars: My pass'word
    -DWIFI_PASS=\"My\ pass\'word\"

[env:specificInclibs]
```

(continues on next page)

(continued from previous page)

```

build_flags =
    -I/opt/include
    -L/opt/lib
    -lfoo

[env:specific_ld_script]
build_flags = -Wl,-T/path/to/ld_script.ld

[env:ignore_incremental_builds]
; We dynamically change the value of "LAST_BUILD_TIME" macro,
; PlatformIO will not cache objects
build_flags = -DLAST_BUILD_TIME=$UNIX_TIME

```

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:

```
[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`:

```
[env:git_revision_macro]
build_flags = !python git_rev_macro.py
```

`git_rev_macro.py`:

```
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.

```
[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 theirs 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:

Pattern	Meaning
*	matches everything
?	matches any single character
[seq]	matches any character in seq
[!seq]	matches any character not in seq

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>

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

- 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 `$UPLOADERFLAGS` build variable. A classic default upload command for *Development Platforms* may look as `some-flash-bin-tool $UPLOADERFLAGS $SOURCE`, where `$SOURCE` will be replaced by a real program/firmware binary.

`$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)

(continued from previous 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
-Pusb
-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:

Pattern	Meaning
*	matches everything
?	matches any single character
[seq]	matches any character in seq
[!seq]	matches any character not in seq

Example:

```
[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:

```
[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:

```
[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

```
...
Library Dependency Finder -> http://bit.ly/configure-pio-ldf
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)

(continued from previous page)

```
|-- <DNSServer> v1.1.0
|   |-- <ESP8266WiFi> v1.0
|-- <Ticker> v1.0
....
```

`platformio.ini`

```
[env:myenv]
; Single line
lib_ignore = AsyncMqttClient, DNSServer

; Multi-line
lib_ignore =
  AsyncMqttClient
  ESP Async WebServer
```

`lib_extra_dirs`**See also:**

Please make sure to read [Library Dependency Finder \(LDF\)](#) guide first.

Type: DirPath | Multiple: Yes

A list with extra directories/storages where [Library Dependency Finder \(LDF\)](#) will look for dependencies.

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

Warning: This is a not direct path to a library with source code. It should be a path to storage that contains libraries grouped by folders. For example, `D:\PlatformIO\extra\libraries` but not `D:\PlatformIO\extra\libraries\FooLibrary`.

Example:

```
[env:myenv]
lib_extra_dirs =
  /common/libraries
  /iot/libraries
```

`lib_ldf_mode`**See also:**

Please make sure to read [Library Dependency Finder \(LDF\)](#) guide first.

Type: String | Multiple: No | Default: chain

This option specifies how does Library Dependency Finder should analyze dependencies (#include directives). See [Dependency Finder Mode](#) for details and available options.

Example:

```
[env:myenv]
; evaluate C/C++ Preprocessor conditional syntax
lib_ldf_mode = chain+
```

lib_compat_mode

See also:

Please make sure to read [Library Dependency Finder \(LDF\)](#) guide first.

Type: String | Multiple: No | Default: soft

Library compatibility mode allows one to control strictness of Library Dependency Finder. See [Compatibility Mode](#) for details and available options..

By default, this value is set to `lib_compat_mode = soft` and means that LDF will check only for framework compatibility.

Example:

```
[env:myenv]
; Checks for the compatibility with frameworks and dev/platforms
lib_compat_mode = strict
```

lib_archive

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

Create an archive (*.a, static library) from the object files and link it into a firmware (program). This is default behavior of PlatformIO Build System (`lib_archive = yes`).

Setting `lib_archive = no` will instruct 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:

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

Check options

See also:

Please make sure to read [PIO Check](#) guide first.

- `check_tool`
- `check_filter`
- `check_flags`
- `check_severity`

check_tool

Type: String | Multiple: Yes | Default: `cppcheck`

A name of the check tool used for analysis. This option is useful when you want to check source code with two or more tools.

See available tools in [Check tools](#).

Example

```
[env:myenv]
platform = ...
board = ...
check_tool = cppcheck, clangtidy
```

check_filter

Type: String (Pattern) | Multiple: Yes

This option allows specifying which source files should be included/excluded from the check process. The filter supports 2 templates:

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

PATH MUST BE related from a project root directory. All patterns will be applied in their order. GLOB Patterns are allowed.

Another option for filtering source files is `platformio check --filter` command.

Example

```
[env:custom_check_filter]
platform = ...
board = ...
check_tool = clangtidy
check_filter = -<*> +<src/module_to_check>
```

check_flags

Type: String | Multiple: No

Additional flags to be passed to the tool command line. This option is useful when you want to adjust the check process to fit your project requirements. By default, the flags are passed to all tools specified in `check_tool` section. To set individual flags, define tool name at the beginning of the line.

Another option for adding flags is `platformio check --flags` command.

Example

```
[env:extra_check_flags]
platform = ...
board = ...
check_tool = cppcheck, clangtidy
check_flags =
--common-flag
```

(continues on next page)

(continued from previous page)

```
cppcheck: --enable=performance --inline-suppr
clangtidy: -fix-errors -format-style=mozilla
```

`check_severity`

Type: String | Multiple: Yes | Default: low, medium, high

This option allows specifying the *Defect severity* types which will be reported by the *Check tools*.

Another option for filtering source files is `platformio check --severity` command.

Example

```
[env:detect_only_medium_or_high_defects]
platform = ...
board = ...
check_severity = medium, high
```

Test options

See also:

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

- `test_filter`
- `test_ignore`
- `test_port`
- `test_speed`
- `test_transport`
- `test_build_project_src`

`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.

Pattern	Meaning
*	matches everything
?	matches any single character
[seq]	matches any character in seq
[!seq]	matches any character not in seq

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.

Pattern	Meaning
*	matches everything
?	matches any single character
[seq]	matches any character in seq
[!seq]	matches any character not in seq

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);
}

void unittest_uart_flush() {
    fflush(stdout);
}

void unittest_uart_end() {

}

#endif
```

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 = ...
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

```
[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

```
[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:

```
[env:debug]
platform = ...
board = ...
debug_init_cmds =
    target extended-remote $DEBUG_PORT
    $INIT_BREAK
    monitor reset halt
    $LOAD_CMDS
    monitor init
    monitor reset halt
```

debug_extra_cmds

Type: String | Multiple: Yes

Extra commands that will be passed to back-end debugger after *debug_init_cmds*. For example, add custom breakpoint and load .gdbinit from a project directory for GDB:

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

(continues on next page)

(continued from previous page)

```
debug_extra_cmds =
  break main.cpp:13
  break foo.cpp:100
  source .gdbinit
```

Note: Initial Project Breakpoints: Use `break path/to/file:LINE_NUMBER` to define initial breakpoints for debug environment. Multiple breakpoints are allowed.

To save session breakpoints, please use `save breakpoints [filename]` command in Debug Console. For example, `save breakpoints .gdbinit`. Later, this file could be loaded via `source [filename]` command. See above.

debug_load_cmds

New in version 4.0.

Type: String | Multiple: Yes | Default: `load`

Specify a command which will be used to load program/firmware to a target device. Possible options:

- `load - default` option
- `load [address]` - load program at specified address, where “[address]” should be a valid number
- `preload` - some embedded devices have locked Flash Memory (a few Freescale Kinetis and NXP LPC boards). In this case, firmware loading using debugging client is disabled. `preload` command instructs [PlatformIO Core \(CLI\)](#) to load program/firmware using development platform “upload” method (via bootloader, media disk, etc)
- (empty value, `debug_load_cmds =`), disables program loading at all.
- `custom commands` - pass any debugging client command (GDB, etc.)

Sometimes you need to run extra monitor commands (on debug server side) before program/firmware loading, such as flash unlocking or erasing. In this case we can combine service commands with loading and run them before. See example:

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

debug_load_mode

Type: String | Multiple: No | Default: `always`

Allows one to control when PlatformIO should load debugging firmware to the end target. Possible options:

- `always` - load for 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.

debug_server

Type: String | Multiple: Yes

Allows one to setup a custom debugging server. By default, boards are pre-configured with a debugging server that is compatible with “on-board” debugging tool (adapter, probe). Also, this option is useful for a [Custom](#) debugging tool.

Option format (multi-line):

- First line is an executable path of debugging server
- 2-nd and the next lines are arguments for executable file

Example:

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

debug_port

Type: String | Multiple: No

A debugging port of a remote target. Could be a serial device or network address. PlatformIO detects it automatically if is not specified.

For example:

- /dev/ttyUSB0 - Unix-based OS
- COM3 - Windows OS
- localhost:3333

debug_svd_path

Type: FilePath | Multiple: No

A custom path to [SVD file](#) which contains information about device peripherals.

Advanced options**extends**

New in version 4.1.

Type: String | Multiple: Yes

This option allows to inherit configuration from other sections or build environments in “[platformio.ini](#)” ([Project Configuration File](#)). Multiple items are allowed, split them with , or with a new line.

If you need to extend only a few options from some section, please take a look at [Dynamic variables](#).

Example:

```
[strict_ldf]
lib_ldf_mode = chain+
lib_compat_mode = strict

[espressif32_base]
platform = espressif32
framework = arduino

[env:release]
extends = espressif32_base, strict_ldf
board = esp32dev
build_flags = -D RELEASE

[env:debug]
extends = env:release
build_type = debug
build_flags = -D DEBUG
```

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](#):

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

[env:myrelease]
some_extra_options = ...

[env:mydebug]
build_type = debug
some_extra_options = ...
```

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 build environments, please take a look at “**Global scope**” in [Section \[env\]](#) or `extends` option which allows extending of other sections.

Example:

```
[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
```

(continues on next page)

(continued from previous page)

```

board = uno
build_flags = ${extra.build_flags}
lib_deps =
    ${extra.lib_deps_builtin}
    ${extra.lib_deps_external}

[env:nodemcuv2]
platform = espressif8266
framework = arduino
board = nodemcuv2
build_flags = ${extra.build_flags} -DSSID_NAME=HELLO -DSSID_PASSWORD=WORLD
lib_deps =
    ${extra.lib_deps_builtin}
    ${extra.lib_deps_external}
    PubSubClient@2.6
    OneWire

[env:esp32dev]
extends = env:nodemcuv2
platform = espressif32
board = esp32dev

```

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

```

(continues on next page)

(continued from previous page)

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

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

; Library options
lib_deps =
    ${common_env_data.lib_deps_builtin}
    ${common_env_data.lib_deps_external}
    https://github.com/me-no-dev/ESPAsyncTCP.git
    PubSubClient@2.6
    OneWire

; Serial Monitor options
monitor_speed = 115200
monitor_flags =
    --encoding
    hexlify

; Unit Testing options
test_ignore = test_desktop

[env:bluepill_f103c8]
platform = ststm32
framework = arduino
board = bluepill_f103c8

; Build options
build_flags = ${common_env_data.build_flags}

; Library options
lib_deps =
    ${common_env_data.lib_deps_external}

; Debug options
debug_tool = custom
debug_server =
    JLinkGDBServer
    -singlerun
    -if
    SWD
    -select
    USB
    -port
    2331
    -device
    STM32F103C8

; Unit Testing options
test_ignore = test_desktop
```

1.7 Environment variables

Environment variables are a set of dynamic named values that can affect the way running processes will behave on a computer. PlatformIO handles variables which start with `PLATFORMIO_` prefix.

How to set environment variable?

```
# Windows  
set VARIABLE_NAME=VALUE  
  
# Windows GUI -> https://www.youtube.com/watch?v=bEroNNzqlF4  
  
# Unix (bash, zsh)  
export VARIABLE_NAME=VALUE  
  
# Unix (fish)  
set -x VARIABLE_NAME VALUE
```

Contents

- [General](#)
- [Directories](#)
- [Building](#)
- [Uploading](#)
- [Settings](#)

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_ANSI`

Force to output ANSI control character even if the output is a pipe (not a `tty`). The possible values are `true` and `false`. Default is `PLATFORMIO_FORCE_ANSI=false`.

`PLATFORMIO_NO_ANSI`

Do not print ANSI control characters. The possible values are `true` and `false`. Default is `PLATFORMIO_NO_ANSI=false`.

You can also use `platformio --no-ansi` flag for *PlatformIO Core (CLI)*.

PLATFORMIO_DISABLE_PROGRESSBAR

Disable progress bar for package/library downloader and uploader. This is useful when calling PlatformIO from subprocess and output is a pipe (not a `tty`). The possible values are `true` and `false`. Default is `PLATFORMIO_DISABLE_PROGRESSBAR=false`.

1.7.2 Directories

PLATFORMIO_CORE_DIR

Allows one to override “`platformio.ini`” (*Project Configuration File*) option `core_dir`.

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*.

1.7.3 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*.

Contents

- *Advanced Scripting*
 - *Launch types*
 - *Construction Environments*

- *Before/Pre and After/Post actions*
- *Custom target*
 - * *Command shortcut*
 - * *Dependent target*
 - * *Target with options*
- *Examples*
 - * *Custom options in platformio.ini*
 - * *Split C/C++ build flags*
 - * *Extra Linker Flags without -Wl, prefix*
 - * *Custom upload tool*
 - * *Upload to Cloud (OTA)*
 - * *Custom firmware/program name*
 - * *Override package files*
 - * *Override Board Configuration*

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 can not run/debug these scripts directly with Python interpreter. They will be loaded automatically when you processing 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*)

```
[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_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.

`my_pre_extra_script.py`:

```
Import("env")

# access to global construction environment
print(env)

# Dump construction environment (for debug purpose)
print(env.Dump())

# append extra flags to global build environment
# which later will be used to build:
# - project source code
# - frameworks
# - dependent libraries
env.Append(CPPDEFINES=[
    "MACRO_1_NAME",
    ("MACRO_2_NAME", "MACRO_2_VALUE")
])
```

`my_post_extra_script.py`:

```
Import("env", "projenv")

# access to global construction environment
print(env)

# access to project construction environment
print(projenv)

# Dump construction environments (for debug purpose)
print(env.Dump())
print(projenv.Dump())

# append extra flags to global build environment
# which later will be used to build:
# - frameworks
# - dependent libraries
env.Append(CPPDEFINES=[
    "MACRO_1_NAME",
```

(continues on next page)

(continued from previous page)

```
( "MACRO_2_NAME", "MACRO_2_VALUE")
])

# append extra flags to only project build environment
projenv.Append(CPPDEFINES=[
    "PROJECT_EXTRA_MACRO_1_NAME",
    ("PROJECT_EXTRA_MACRO_2_NAME", "PROJECT_EXTRA_MACRO_2_VALUE")
])
```

See examples below how to import construction environments and modify existing data or add new.

1.8.3 Before/Pre and After/Post actions

PlatformIO Build System has rich API that allows one to attach different pre-/post actions (hooks) using `env.AddPreAction(target, callback)` or `env.AddPostAction(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
```

(continues on next page)

(continued from previous page)

```

# call Node.JS or other script
env.Execute("node --version")

def after_upload(source, target, env):
    print("after_upload")
    # do some actions

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

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

#
# Custom actions when building program/firmware
#

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

#
# Custom actions for specific files/objects
#

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

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

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

# Custom HEX from ELF
env.AddPostAction(
    "${BUILD_DIR}/${PROGNAME}.elf",
    env.VerboseAction(" ".join([
        "$OBJCOPY", "-O", "ihex", "-R", ".eprom",
        "${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

platformio.ini:

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

extra_script.py:

```
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:

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

extra_script.py:

```
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:

```
[env:env_custom_target]
platform = ...
...
extra_scripts = extra_script.py
custom_ping_host = google.com
```

extra_script.py:

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

Import("env")
```

(continues on next page)

(continued from previous page)

```

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

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

# General options that are passed to the C++ compiler
env.Append(CXXFLAGS=["flag1", "flag2"])
```

Extra Linker Flags without -Wl, prefix

Sometimes you need to pass extra flags to GCC linker without Wl,. You could use `build_flags` option but it will not work. PlatformIO will not parse these flags to `LINKFLAGS` scope. In this case, simple extra script will help:

`platformio.ini`:

```
[env:env_extra_link_flags]
platform = windows_x86
extra_scripts = extra_script.py
```

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

```
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 `UPLOADEEPCMD` (EEPROM data).

See examples below:

Template

`platformio.ini`:

```
[env:my_custom_upload_tool]
platform = ...
; place it into the root of project or use full path
```

(continues on next page)

(continued from previous page)

```
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`):

```
Import("env")

# please keep $SOURCE variable, it will be replaced with a path to firmware

# Generic
env.Replace(
    UPLOADER="executable or path to executable",
    UPLOADCMD="$UPLOADER $UPLOADERFLAGS $SOURCE"
)

# In-line command with arguments
env.Replace(
    UPLOADCMD="executable -arg1 -arg2 $SOURCE"
)

# Python callback
def on_upload(source, target, env):
    print(source, target)
    firmware_path = str(source[0])
    # do something
    env.Execute("executable arg1 arg2")

env.Replace(UPLOADCMD=on_upload)
```

Custom openOCD command

`platformio.ini`:

```
[env:disco_f407vg]
platform = ststm32
board = disco_f407vg
framework = mbed

extra_scripts = extra_script.py
upload_protocol = custom
; each flag in a new line
upload_flags =
    -f
    scripts/interface/stlink.cfg
    -f
    scripts/target/stm32f4x.cfg
```

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

```
Import("env")

platform = env.PioPlatform()
```

(continues on next page)

(continued from previous page)

```

env.Prepend(
    UPLOADERFLAGS=[ "-s ", platform.get_package_dir("tool-openocd") or "" ]
)
env.Append(
    UPLOADERFLAGS=[ "-c ", "program {$SOURCE} verify reset; shutdown" ]
)
env.Replace(
    UPLOADER="openocd",
    UPLOADCMD="$UPLOADER $UPLOADERFLAGS"
)

```

Upload to Cloud (OTA)

See project example <https://github.com/platformio/bintray-secure-ota>

Custom firmware/program name

Sometimes is useful to have a different firmware/program name in *build_dir*.

`platformio.ini`:

```

[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`:

```

Import("env")

my_flags = env.ParseFlags(env['BUILD_FLAGS'])
defines = {k: v for (k, v) in my_flags.get("CPPDEFINES")}
# print(defines)

env.Replace(PROGNAME="firmware_%s" % defines.get("VERSION"))

```

Override package files

PlatformIO Package Manager automatically installs pre-built packages (*Frameworks*, toolchains, libraries) required by development *Development Platforms* and build process. Sometimes you need to override original files with own versions: configure custom GPIO, do changes to built-in LD scripts, or some patching to installed library dependency.

The simplest way is using **Diff** and **Patch** technique. How does it work?

1. Modify original source files
2. Generate patches
3. Apply patches via PlatformIO extra script before build process.

Example

We need to patch the original standard/pins_arduino.h variant from [Arduino](#) framework and add extra macro #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:

```
63a64
> #define PIN_A8      (99)
112c113
< // 14-21 PA0-PA7 works
---
> // 14-21 PA0-PA7 works
```

Using extra scripting we can apply patching before a build process. The final result of “*platformio.ini*” (*Project Configuration File*) and “PRE” extra script named `apply_patches.py`:

`platformio.ini`:

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

`apply_patches.py`:

```
from os.path import join, isfile

Import("env")

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:

`platformio.ini`:

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

`custom_hwids.py`:

```
Import ("env")

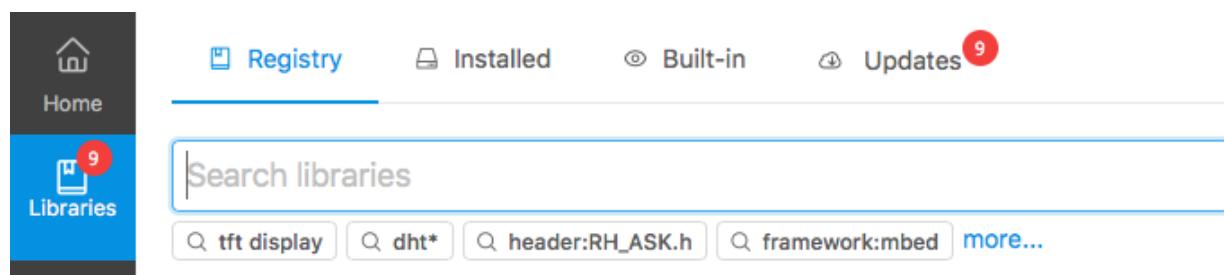
board_config = env.BoardConfig()
board_config.update("build.hwids", [
    ["0x2341", "0x0243"],
    ["0x2A03", "0x0043"]
])
```

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:

- [PlatformIO Home](#)
- [Web Library Search](#)
- PIO Core [Command Line Interface](#)

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@~5.6.1!=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

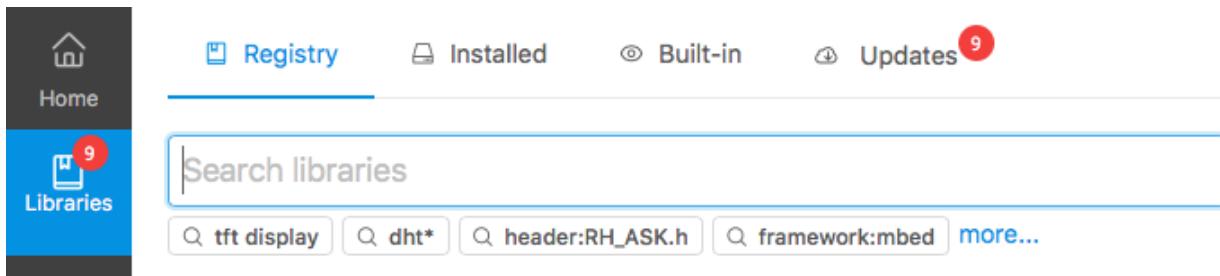
- **Atom:** “Menu: PlatformIO > Rebuild C/C++ Project Index (Autocomplete, Linter)”
- **VSCode:** “Menu: View > Command Palette... > PlatformIO: Rebuild C/C++ Project Index”

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.



PlatformIO Core

CLI Guide

1.9.2 Library Dependency Finder (LDF)

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

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. `lib_extra_dirs` - extra storages per build environment
2. `lib_dir` - own/private library storage per project

3. *libdeps_dir* - project dependency storage used by *Library Manager*
4. “*core_dir/lib*” - global storage per all projects.
5. Library storages built into frameworks, SDKs.

Dependency Finder Mode

Library Dependency Finder starts work from analyzing source files of the project (*src_dir*) and can work in the next modes:

off “Manual mode”, does not process source files of a project and dependencies. Builds only the libraries that are specified in manifests (*library.json*, *module.json*) or using *lib_deps* option.

chain [DEFAULT] Parses ALL C/C++ source files of the project and follows only by nested includes (#include . . . , chain...) from the libraries. It also parses C, CC, CPP files from libraries which have the same name as included header file. **Does not evaluate C/C++ Preprocessor conditional syntax**.

deep Parses ALL C/C++ source files of the project and parses ALL C/C++ source files of the each found dependency (recursively). **Does not evaluate C/C++ Preprocessor conditional syntax**.

chain+ The same behavior as for the **chain** but **evaluates C/C++ Preprocessor conditional syntax**.

deep+ The same behavior as for the **deep** but **evaluates C/C++ Preprocessor conditional syntax**.

The mode can be changed using *lib_ldf_mode* option in “*platformio.ini*” (*Project Configuration File*). Default value is set to **chain**.

Note: Usually, when the LDF appears to fail to identify a dependency of a library, it is because the dependency is only referenced from 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/extracpp

- 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,

`platformio.ini`

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

`mylib.h`

```
#ifdef 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

Pattern	Meaning
*	matches everything
?	matches any single character
[seq]	matches any character in seq
[!seq]	matches any character not in seq

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:

```
"dependencies": [
    {
        "name": "Library-Foo",
        "authors": [
            "Jhon Smith",
            "Andrew Smith"
        ],
        "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:

```
"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:

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

(continues on next page)

(continued from previous page)

```
[ " [Ee]xamples/*/*/*.pde"  
]
```

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`

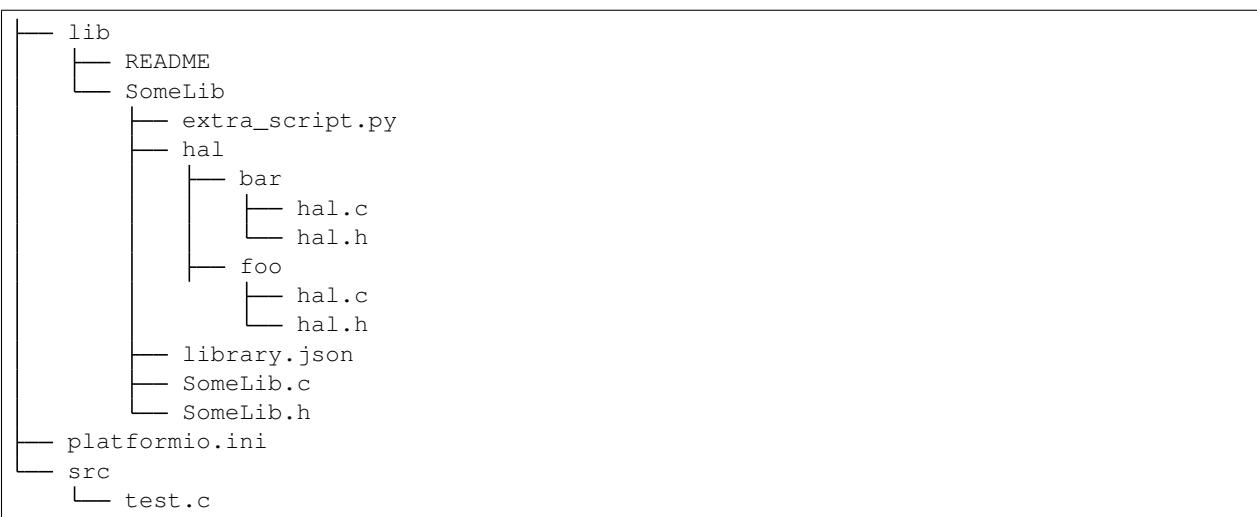
Optional | Type: String

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



`platformio.ini`

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

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

`library.json`

```
{  
    "name": "SomeLib",  
    "version": "0.0.0",  
    "build": {  
        "extraScript": "extra_script.py"  
    }  
}
```

extra_script.py

```
Import('env')  
from os.path import join, realpath  
  
# private library flags  
for item in env.get("CPPDEFINES", []):  
    if isinstance(item, tuple) and item[0] == "HAL":  
        env.Append(CPPPATH=[realpath(join("hal", item[1]))])  
        env.Replace(SRC_FILTER=["+<*>", "-<hal>", "+<%s>" % join("hal", item[1])])  
        break  
  
# pass flags to a global build environment (for all libraries, etc)  
global_env = DefaultEnvironment()  
global_env.Append(  
    CPPDEFINES=[  
        ("MQTT_MAX_PACKET_SIZE", 512),  
        "ARDUINOJSON_ENABLE_STD_STRING",  
        ("BUFFER_LENGTH", 32)  
    ]  
)
```

libArchive

Optional | Type: Boolean

Create an archive (*.a, static library) from the object files and link it into a firmware (program). This is default behavior of PlatformIO Build System ("libArchive": true).

Setting "libArchive": false will instruct 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

```
"build": {
    "flags": "-D MYLIB_REV=1.2.3 -DRELEASE"
```

2. Extra includes for C preprocessor

```
"build": {
    "flags": [
        "-I inc",
        "-I inc/target_x13"
    ]
}
```

3. Force to use C99 standard instead of C11

```
"build": {
    "unflags": "-std=gnu++11",
    "flags": "-std=c99"
}
```

4. Build source files (c, cpp, h) at the top level of the library

```
"build": {
    "srcFilter": [
        "+<*.c>",
        "+<*.cpp>",
        "+<*.h>"
    ]
}
```

5. Extend PlatformIO Build System with own extra script

```
"build": {
    "extraScript": "generate_headers.py"
}
```

`generate_headers.py`

```
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`:

```
{  
    "name": "DallasTemperature",  
    "keywords": "onewire, 1-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":  
    [  
        {  
            "name": "Miles Burton",  
            "email": "milesburton@users.noreply.github.com",  
            "url": "https://github.com/milesburton"  
        }  
    ]  
}
```

(continues on next page)

(continued from previous 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": "gfbbarros@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:

```
{
    "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": ...
}
```

(continues on next page)

(continued from previous page)

```
{
    "type": "git",
    "url": "https://code.google.com/p/xbee-arduino/"
},
"frameworks": "arduino",
"platforms": "atmelavr"
}
```

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*

```
{
    "name": "OneWire",
    "keywords": "onewire, 1-wire, bus, sensor, temperature, ibutton",
    "description": "Control devices (from Dallas Semiconductor) that use the One Wire protocol (DS18S20, DS18B20, DS24...)",
    "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:

```
$ 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](#).

Contents

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

Examples

Examples are listed from Aceinna IMU development platform repository:

- [OpenIMU300RI](#)
- [OpenIMU300ZI](#)
- [OpenIMU330BI](#)

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

Bands listed below have on-board debug probe and **ARE READY** for debugging! You do not need to use/buy external debug probe.

Name	MCU	Frequency	Flash	RAM
Aceinna Low Cost RTK	STM32F469NIH6	180MHz	1MB	384KB

External Debug Tools

Bands 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.

Name	MCU	Frequency	Flash	RAM
Aceinna OpenIMU 300ZA	STM32F405RG	120MHz	1MB	128KB
Aceinna OpenIMU 300ZA	STM32F405RG	120MHz	1MB	128KB
Aceinna OpenIMU 330	STM32L431CB	80MHz	128KB	64KB

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

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

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

Upstream

```
[env:upstream_develop]
platform = https://github.com/aceinna/platform-aceinna_imu.git
board = ...
```

Packages

Name	Description
tool-jlink	SEGGER J-Link Software and Documentation Pack
tool-openocd	OpenOCD
toolchain-gccarmnoneabi	gcc-arm-embedded

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

Name	Debug	MCU	Frequency	Flash	RAM
<i>Aceinna Low Cost RTK</i>	On-board	STM32F469NIH6	180MHz	1MB	384KB
<i>Aceinna OpenIMU 300ZA</i>	External	STM32F405RG	120MHz	1MB	128KB
<i>Aceinna OpenIMU 300ZA</i>	External	STM32F405RG	120MHz	1MB	128KB
<i>Aceinna OpenIMU 330</i>	External	STM32L431CB	80MHz	128KB	64KB

Atmel AVR

Configuration *platform* = 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 programming*
 - *Custom fuses*
 - *Mini, Mega, Mighty cores*
- *Bootloader programming*
 - *Custom bootloader*
 - *Mini, Mega, Mighty cores*

Upload using Programmer

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

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

Note: List of avrdude supported programmers are accessible with `avrdude -c ?`

Configuration for the programmers:

- AVR ISP

```
[env:myenv]
platform = atmelavr
framework = arduino
upload_protocol = stk500v1
; each flag in a new line
upload_flags =
    -P$UPLOAD_PORT

; edit this line with valid upload port
upload_port = SERIAL_PORT_HERE
```

- AVRISP mkII

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

- USBtinyISP

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

- ArduinoISP

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

- USBasp

```
[env:myenv]
platform = atmelavr
```

(continues on next page)

(continued from previous page)

```
framework = arduino
upload_protocol = usbasp
; each flag in a new line
upload_flags =
    -Pusb
```

- Parallel Programmer

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

- Arduino as ISP

```
[env:myenv]
platform = atmelavr
framework = arduino
upload_protocol = stk500v1
; each flag in a new line
upload_flags =
    -P$UPLOAD_PORT
    -b$UPLOAD_SPEED

; edit these lines
upload_port = SERIAL_PORT_HERE
upload_speed = 19200
```

- Bus Pirate as ISP

```
[env:myenv]
platform = atmelavr
framework = arduino
upload_protocol = buspirate
; each flag in a new line
upload_flags =
    -P$UPLOAD_PORT
    -b$UPLOAD_SPEED

; edit these lines
upload_port = SERIAL_PORT_HERE
upload_speed = 115200
```

Upload EEPROM data

To upload EEPROM data (from EEMEM directive) you need to use `uploaddeep` target instead `upload` for `platformio run --target` command. For example, `platformio run -t uploaddeep`.

Fuses programming

PlatformIO has a built-in target named `fuses` for setting fuse bits. The default fuse bits are predefined in board manifest file in `fuses` section. For example, [fuses section for Arduino Uno board](#). To set fuse bits you need to use target `fuses` with `platformio run --target` command.

Custom fuses

Custom fuse values and upload flags (based on upload protocol) should be specified in “*platformio.ini*” (*Project Configuration File*). `lfuse` and `hfuse` bits are mandatory, `efuse` is optional and not supported by all targets. An example of setting custom fuses for uno board:

```
[env:custom_fuses]
platform = atmelavr
framework = arduino
board = uno
upload_protocol = stk500v1
upload_speed = 19200
board_fuses.lfuse = 0xAA
board_fuses.hfuse = 0xBB
board_fuses.efuse = 0xCC
upload_flags =
    -PCOM15
    -b$UPLOAD_SPEED
    -e
```

Mini, Mega, Mighty cores

Mini, Mega, Mighty cores support dynamic fuses generation. Generated values are based on the next parameters:

Pa-ram-eter	Description	De-fault value
<code>f_cpu</code>	Specifies the clock frequencies in Hz. Used to determine what oscillator option to choose. A capital L has to be added to the end of the frequency number.	16000000L
<code>oscill</code>	Specifies which oscillator is used internal or external. Internal oscillator only works with <code>f_cpu</code> values <code>8000000L</code> and <code>1000000L</code>	external
<code>uart</code>	Specifies the hardware UART port used for serial upload. can be <code>uart0</code> , <code>uart1</code> , <code>uart2</code> or <code>uart3</code> depending on the target. Use <code>no_bootloader</code> if you're not using a bootloader for serial upload.	<code>uart0</code>
<code>bod</code>	Specifies the hardware brown-out detection. Use <code>disabled</code> to disable brown-out detection.	2.7v
<code>eesave</code>	Specifies if the EEPROM memory should be retained when uploading using a programmer. Use <code>no</code> to disable	yes

Valid BOD values:

ATmega8, ATmega8535/16/32, ATmega64/128	AT90CAN32/64/128	Other targets
4.0v	4.1v	4.3v
2.7v	4.0v	2.7v
disabled	3.9v	1.8v
	3.8v	disabled
	2.7v	
	2.6v	
	2.5v	
	disabled	

Hardware configuration example:

```
[env:custom_fuses]
platform = atmelavr
framework = arduino
board = ATmega32

board_build.f_cpu = 1000000L
board_hardware.uart = uart0
board_hardware.oscillator = internal
board_hardware.bod = 2.7v
board_hardware.eesave = no

upload_protocol = usbsp
upload_flags =
    -Pusb
```

Bootloader programming

PlatformIO has a built-in target named `bootloader` for flashing bootloaders. The default bootloader image and corresponding fuse bits are predefined in board manifest file in `bootloader` section, for example, `Arduino Uno`. To upload bootloader image you need to use target `bootloader` with `platformio run --target` command.

Custom bootloader

Custom bootloader and corresponding fuses should be specified in “`platformio.ini`” (*Project Configuration File*). If `lock_bits` and `unlock_bits` are not set then the default values `0x0F` and `0x3F` are used accordingly. An example of setting custom bootloader for uno board:

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

board_bootloader.file = /path/to/custom/bootloader.hex
board_bootloader.low_fuses = 0xFF
board_bootloader.high_fuses = 0xDE
board_bootloader.extended_fuses = 0xFD
board_bootloader.lock_bits = 0x0F
board_bootloader.unlock_bits = 0x3F
```

Mini, Mega, Mighty cores

Mini, Mega, Mighty cores have a wide variety of precompiled bootloaders. Bootloader binary is dynamically selected according to the hardware parameters: `f_cpu`, `oscillator`, `upload_speed`:

Frequency	Oscillator	Upload Speed
20000000L	external	115200
18432000L	external	115200
16000000L	external	115200
14745600L	external	115200
12000000L	external	57600
11059200L	external	115200
8000000L	external/internal	57600/38400
7372800L	external	115200
3686400L	external	115200
1843200L	external	115200
1000000L	external/internal	9600

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

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

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

Upstream

```
[env:upstream_develop]
platform = https://github.com/platformio/platform-atmelavr.git
board = ...
```

Packages

Name	Description
framework-arduinoavr	Arduino Wiring-based Framework (AVR Core)
framework-simba	Simba Framework
tool-avrdude	AVRDUDE
tool-micronucleus	Micronucleus
toolchain-atmelavr	avr-gcc

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

Name	Description
Arduin	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
Simba	Simba is an RTOS and build framework. It aims to make embedded programming easy and portable.

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

Name	Debug	MCU	Frequency	Flash	RAM
<i>Adafruit Bluefruit Micro</i>	No	ATMEGA32U4	8MHz	28KB	2.50KB
<i>Adafruit Circuit Playground Classic</i>	No	ATMEGA32U4	8MHz	28KB	2.50KB
<i>Adafruit Feather 328P</i>	No	ATMEGA328P	8MHz	31.50KB	2KB
<i>Adafruit Feather 32u4</i>	No	ATMEGA32U4	8MHz	28KB	2.50KB
<i>Adafruit Flora</i>	No	ATMEGA32U4	8MHz	28KB	2.50KB
<i>Adafruit Gemma</i>	No	ATTINY85	8MHz	8KB	512B
<i>Adafruit ItsyBitsy 3V/8MHz</i>	No	ATMEGA32U4	8MHz	28KB	2.50KB
<i>Adafruit ItsyBitsy 5V/16MHz</i>	No	ATMEGA32U4	16MHz	28KB	2.50KB
<i>Adafruit Metro</i>	No	ATMEGA328P	16MHz	31.50KB	2KB
<i>Adafruit Pro Trinket 3V/12MHz (FTDI)</i>	No	ATMEGA328P	12MHz	28KB	2KB
<i>Adafruit Pro Trinket 3V/12MHz (USB)</i>	No	ATMEGA328P	12MHz	28KB	2KB
<i>Adafruit Pro Trinket 5V/16MHz (FTDI)</i>	No	ATMEGA328P	16MHz	28KB	2KB
<i>Adafruit Pro Trinket 5V/16MHz (USB)</i>	No	ATMEGA328P	16MHz	28KB	2KB
<i>Adafruit Trinket 3V/8MHz</i>	No	ATTINY85	8MHz	8KB	512B
<i>Adafruit Trinket 5V/16MHz</i>	No	ATTINY85	16MHz	8KB	512B

Alorium Technology

Name	Debug	MCU	Frequency	Flash	RAM
<i>Alorium Hinj</i>	No	ATMEGA328P	16MHz	31.50KB	2KB
<i>Alorium Sno</i>	No	ATMEGA328P	16MHz	31.50KB	2KB
<i>Alorium XLR8</i>	No	ATMEGA328P	16MHz	31.50KB	2KB

Anarduino

Name	Debug	MCU	Frequency	Flash	RAM
<i>Anarduino MiniWireless</i>	No	ATMEGA328P	16MHz	31.50KB	2KB

Arduboy

Name	Debug	MCU	Frequency	Flash	RAM
<i>Arduboy</i>	No	ATMEGA32U4	16MHz	28KB	2.50KB
<i>Arduboy DevKit</i>	No	ATMEGA32U4	16MHz	28KB	2.50KB

Arduino

Name	Debug	MCU	Frequency	Flash	RAM
<i>Arduino BT ATmega168</i>	No	ATMEGA168	16MHz	14KB	1KB
<i>Arduino BT ATmega328</i>	No	ATMEGA328P	16MHz	28KB	2KB
<i>Arduino Duemilanove or Diecimila ATmega168</i>	No	ATMEGA168	16MHz	14KB	1KB

Continued on next page

Table 1 – continued from previous page

Name	Debug	MCU	Frequency	Flash	RAM
<i>Arduino Duemilanove or Diecimila ATmega328</i>	No	ATMEGA328P	16MHz	30KB	2KB
<i>Arduino Esplora</i>	No	ATMEGA32U4	16MHz	28KB	2.50KB
<i>Arduino Ethernet</i>	No	ATMEGA328P	16MHz	31.50KB	2KB
<i>Arduino Fio</i>	No	ATMEGA328P	8MHz	30KB	2KB
<i>Arduino Industrial 101</i>	No	ATMEGA32U4	16MHz	28KB	2.50KB
<i>Arduino Leonardo</i>	No	ATMEGA32U4	16MHz	28KB	2.50KB
<i>Arduino Leonardo ETH</i>	No	ATMEGA32U4	16MHz	28KB	2.50KB
<i>Arduino LilyPad ATmega168</i>	No	ATMEGA168	8MHz	14KB	1KB
<i>Arduino LilyPad ATmega328</i>	No	ATMEGA328P	8MHz	30KB	2KB
<i>Arduino LilyPad USB</i>	No	ATMEGA32U4	8MHz	28KB	2.50KB
<i>Arduino Mega ADK</i>	No	ATMEGA2560	16MHz	248KB	8KB
<i>Arduino Mega or Mega 2560 ATmega1280</i>	No	ATMEGA1280	16MHz	124KB	8KB
<i>Arduino Mega or Mega 2560 ATmega2560 (Mega 2560)</i>	No	ATMEGA2560	16MHz	248KB	8KB
<i>Arduino Micro</i>	No	ATMEGA32U4	16MHz	28KB	2.50KB
<i>Arduino Mini ATmega168</i>	No	ATMEGA168	16MHz	14KB	1KB
<i>Arduino Mini ATmega328</i>	No	ATMEGA328P	16MHz	28KB	2KB
<i>Arduino NG or older ATmega168</i>	No	ATMEGA168	16MHz	14KB	1KB
<i>Arduino NG or older ATmega8</i>	No	ATMEGA8	16MHz	7KB	1KB
<i>Arduino Nano ATmega168</i>	No	ATMEGA168	16MHz	14KB	1KB
<i>Arduino Nano ATmega328</i>	No	ATMEGA328P	16MHz	30KB	2KB
<i>Arduino Nano ATmega328 (New Bootloader)</i>	No	ATMEGA328P	16MHz	30KB	2KB
<i>Arduino Pro or Pro Mini ATmega168 (3.3V, 8 MHz)</i>	No	ATMEGA168	8MHz	14KB	1KB
<i>Arduino Pro or Pro Mini ATmega168 (5V, 16 MHz)</i>	No	ATMEGA168	16MHz	14KB	1KB
<i>Arduino Pro or Pro Mini ATmega328 (3.3V, 8 MHz)</i>	No	ATMEGA328P	8MHz	30KB	2KB
<i>Arduino Pro or Pro Mini ATmega328 (5V, 16 MHz)</i>	No	ATMEGA328P	16MHz	30KB	2KB
<i>Arduino Robot Control</i>	No	ATMEGA32U4	16MHz	28KB	2.50KB
<i>Arduino Robot Motor</i>	No	ATMEGA32U4	16MHz	28KB	2.50KB
<i>Arduino Uno</i>	No	ATMEGA328P	16MHz	31.50KB	2KB
<i>Arduino Yun</i>	No	ATMEGA32U4	16MHz	28KB	2.50KB
<i>Arduino Yun Mini</i>	No	ATMEGA32U4	16MHz	28KB	2.50KB

Atmel

Name	Debug	MCU	Frequency	Flash	RAM
<i>Generic ATtiny13</i>	No	ATTINY13	1MHz	1KB	64B
<i>Generic ATtiny13A</i>	No	ATTINY13A	1MHz	1KB	64B
<i>Generic ATtiny1634</i>	No	ATTINY1634	8MHz	16KB	1KB
<i>Generic ATtiny167</i>	No	ATTINY167	8MHz	16KB	512B
<i>Generic ATtiny2313</i>	No	ATTINY2313	8MHz	2KB	128B
<i>Generic ATtiny24</i>	No	ATTINY24	8MHz	2KB	128B
<i>Generic ATtiny25</i>	No	ATTINY25	8MHz	2KB	128B
<i>Generic ATtiny261</i>	No	ATTINY261	8MHz	2KB	128B
<i>Generic ATtiny4313</i>	No	ATTINY4313	8MHz	4KB	256B
<i>Generic ATtiny43U</i>	No	ATTINY43U	8MHz	4KB	256B
<i>Generic ATtiny44</i>	No	ATTINY44	8MHz	4KB	256B
<i>Generic ATtiny441</i>	No	ATTINY441	8MHz	4KB	256B
<i>Generic ATtiny45</i>	No	ATTINY45	8MHz	4KB	256B
<i>Generic ATtiny461</i>	No	ATTINY461	8MHz	4KB	256B
<i>Generic ATtiny48</i>	No	ATTINY48	8MHz	4KB	256B
<i>Generic ATtiny828</i>	No	ATTINY828	8MHz	8KB	512B
<i>Generic ATtiny84</i>	No	ATTINY84	8MHz	8KB	512B
<i>Generic ATtiny841</i>	No	ATTINY841	8MHz	8KB	512B
<i>Generic ATtiny85</i>	No	ATTINY85	8MHz	8KB	512B
<i>Generic ATtiny861</i>	No	ATTINY861	8MHz	8KB	512B
<i>Generic ATtiny87</i>	No	ATTINY87	8MHz	8KB	512B
<i>Generic ATtiny88</i>	No	ATTINY88	8MHz	8KB	512B
<i>USBasp stick</i>	No	ATMEGA8	12MHz	8KB	1KB

BQ

Name	Debug	MCU	Frequency	Flash	RAM
<i>BQ ZUM BT-328</i>	No	ATMEGA328P	16MHz	28KB	2KB

BSFrance

Name	Debug	MCU	Frequency	Flash	RAM
<i>LoRa32u4II (868-915MHz)</i>	No	ATMEGA32U4	8MHz	28KB	2.50KB

BitWizard

Name	Debug	MCU	Frequency	Flash	RAM
<i>BitWizard Raspduino</i>	No	ATMEGA328P	16MHz	30KB	2KB

Controllino

Name	Debug	MCU	Frequency	Flash	RAM
<i>Controllino Maxi</i>	No	ATMEGA2560	16MHz	248KB	8KB
<i>Controllino Maxi Automation</i>	No	ATMEGA2560	16MHz	248KB	8KB
<i>Controllino Mega</i>	No	ATMEGA2560	16MHz	248KB	8KB
<i>Controllino Mini</i>	No	ATMEGA328P	16MHz	31.50KB	2KB

Digistump

Name	Debug	MCU	Frequency	Flash	RAM
<i>Digispark Pro</i>	No	ATTINY167	16MHz	14.50KB	512B
<i>Digispark Pro (16 MHz) (64 byte buffer)</i>	No	ATTINY167	16MHz	14.50KB	512B
<i>Digispark Pro (32 byte buffer)</i>	No	ATTINY167	16MHz	14.50KB	512B
<i>Digispark USB</i>	No	ATTINY85	16MHz	5.87KB	512B

Dwengo

Name	Debug	MCU	Frequency	Flash	RAM
<i>Dwenguino</i>	No	AT90USB646	16MHz	60KB	2KB

Elektor

Name	Debug	MCU	Frequency	Flash	RAM
<i>Elektor Uno R4</i>	No	ATMEGA328PB	16MHz	31.50KB	2KB

Engduino

Name	Debug	MCU	Frequency	Flash	RAM
<i>Engduino 3</i>	No	ATMEGA32U4	8MHz	28KB	2.50KB

EnviroDIY

Name	Debug	MCU	Frequency	Flash	RAM
<i>EnviroDIY Mayfly</i>	No	ATMEGA1284P	8MHz	127KB	16KB

FYSETC

Name	Debug	MCU	Frequency	Flash	RAM
<i>FYSETC F6 V1.3</i>	No	ATMEGA2560	16MHz	252KB	8KB

LightUp

Name	Debug	MCU	Frequency	Flash	RAM
<i>LightUp</i>	No	ATMEGA32U4	8MHz	28KB	2.50KB

Linino

Name	Debug	MCU	Frequency	Flash	RAM
<i>Linino One</i>	No	ATMEGA32U4	16MHz	28KB	2.50KB

LowPowerLab

Name	Debug	MCU	Frequency	Flash	RAM
<i>LowPowerLab MightyHat</i>	No	ATMEGA328P	16MHz	31KB	2KB
<i>LowPowerLab Moteino</i>	No	ATMEGA328P	16MHz	31.50KB	2KB
<i>LowPowerLab Moteino (8MHz)</i>	No	ATMEGA328P	8MHz	31.50KB	2KB
<i>LowPowerLab MoteinoMEGA</i>	No	ATMEGA1284P	16MHz	127KB	16KB

MediaTek Labs

Name	Debug	MCU	Frequency	Flash	RAM
<i>LinkIt Smart 7688 Duo</i>	No	ATMEGA32U4	8MHz	28KB	2.50KB

Microchip

Name	Debug	MCU	Frequency	Flash	RAM
<i>AT90CAN128</i>	No	AT90CAN128	16MHz	127KB	4KB
<i>AT90CAN32</i>	No	AT90CAN32	16MHz	31KB	2KB
<i>AT90CAN64</i>	No	AT90CAN64	16MHz	63KB	4KB
<i>ATmega128/A</i>	No	ATMEGA128	16MHz	127KB	4KB
<i>ATmega1280</i>	No	ATMEGA1280	16MHz	127KB	8KB
<i>ATmega1281</i>	No	ATMEGA1281	16MHz	127KB	8KB
<i>ATmega1284</i>	No	ATMEGA1284	16MHz	127KB	16KB
<i>ATmega1284P</i>	No	ATMEGA1284P	16MHz	127KB	16KB
<i>ATmega16</i>	No	ATMEGA16	16MHz	15.50KB	1KB
<i>ATmega164A</i>	No	ATMEGA164A	16MHz	15.50KB	1KB
<i>ATmega164P/PA</i>	No	ATMEGA164P	16MHz	15.50KB	1KB
<i>ATmega168/A</i>	No	ATMEGA168	16MHz	15.50KB	1KB
<i>ATmega168P/PA</i>	No	ATMEGA168P	16MHz	15.50KB	1KB
<i>ATmega168PB</i>	No	ATMEGA168PB	16MHz	15.50KB	1KB
<i>ATmega2560</i>	No	ATMEGA2560	16MHz	255KB	8KB
<i>ATmega2561</i>	No	ATMEGA2561	16MHz	255KB	8KB
<i>ATmega32</i>	No	ATMEGA32	16MHz	31.50KB	2KB

Continued on next page

Table 2 – continued from previous page

Name	Debug	MCU	Frequency	Flash	RAM
<i>ATmega324A</i>	No	ATMEGA324A	16MHz	31.50KB	2KB
<i>ATmega324P</i>	No	ATMEGA324P	16MHz	31.50KB	2KB
<i>ATmega324PA</i>	No	ATMEGA324PA	16MHz	31.50KB	2KB
<i>ATmega324PB</i>	No	ATMEGA324PB	16MHz	31.50KB	2KB
<i>ATmega328</i>	No	ATMEGA328	16MHz	31.50KB	2KB
<i>ATmega328P/PA</i>	No	ATMEGA328P	16MHz	31.50KB	2KB
<i>ATmega328PB</i>	No	ATMEGA328PB	16MHz	31.50KB	2KB
<i>ATmega48/A</i>	No	ATMEGA48	16MHz	4KB	512B
<i>ATmega48PB</i>	No	ATMEGA48PB	16MHz	4KB	512B
<i>ATmega64/A</i>	No	ATMEGA64	16MHz	63KB	4KB
<i>ATmega640</i>	No	ATMEGA640	16MHz	63KB	8KB
<i>ATmega644/A</i>	No	ATMEGA644A	16MHz	63KB	4KB
<i>ATmega644P/PA</i>	No	ATMEGA644P	16MHz	63KB	4KB
<i>ATmega8/A</i>	No	ATMEGA8	16MHz	7.50KB	1KB
<i>ATmega8535</i>	No	ATMEGA8535	16MHz	7.50KB	512B
<i>ATmega88/A</i>	No	ATMEGA88	16MHz	7.50KB	1KB
<i>ATmega88P/PA</i>	No	ATMEGA88P	16MHz	7.50KB	1KB
<i>ATmega88PB</i>	No	ATMEGA88PB	16MHz	7.50KB	1KB
<i>ATmega8P/PA</i>	No	ATMEGA48P	16MHz	4KB	512B
<i>Atmel AT90PWM216</i>	No	AT90PWM216	16MHz	16KB	1KB
<i>Atmel AT90PWM316</i>	No	AT90PWM316	16MHz	16KB	1KB

Microduino

Name	De-bug	MCU	Fre-quency	Flash	RAM
<i>Microduino Core (Atmega168PA@16M,5V)</i>	No	ATMEGA168P	16MHz	15.50KB	1KB
<i>Microduino Core (Atmega168PA@8M,3.3V)</i>	No	ATMEGA168P	8MHz	15.50KB	1KB
<i>Microduino Core (Atmega328P@16M,5V)</i>	No	ATMEGA328P	16MHz	31.50KB	2KB
<i>Microduino Core (Atmega328P@8M,3.3V)</i>	No	ATMEGA328P	8MHz	31.50KB	2KB
<i>Microduino Core USB (AT-mega32U4@16M,5V)</i>	No	ATMEGA32U4	16MHz	28KB	2.50KB
<i>Microduino Core+ (ATmega1284P@16M,5V)</i>	No	AT-MEGA1284P	16MHz	127KB	16KB
<i>Microduino Core+ (ATmega1284P@8M,3.3V)</i>	No	AT-MEGA1284P	8MHz	127KB	16KB
<i>Microduino Core+ (Atmega644PA@16M,5V)</i>	No	ATMEGA644P	16MHz	63KB	4KB
<i>Microduino Core+ (Atmega644PA@8M,3.3V)</i>	No	ATMEGA644P	8MHz	63KB	4KB

OpenEnergyMonitor

Name	Debug	MCU	Frequency	Flash	RAM
<i>OpenEnergyMonitor emonPi</i>	No	ATMEGA328P	16MHz	30KB	2KB

PanStamp

Name	Debug	MCU	Frequency	Flash	RAM
<i>PanStamp AVR</i>	No	ATMEGA328P	8MHz	31.50KB	2KB

Pinoccio

Name	Debug	MCU	Frequency	Flash	RAM
<i>Pinoccio Scout</i>	No	ATMEGA256RFR2	16MHz	248KB	32KB

Pololu Corporation

Name	Debug	MCU	Frequency	Flash	RAM
<i>Pololu A-Star 32U4</i>	No	ATMEGA32U4	16MHz	28KB	2.50KB

Punch Through

Name	Debug	MCU	Frequency	Flash	RAM
<i>LightBlue Bean</i>	No	ATMEGA328P	8MHz	31.50KB	2KB
<i>LightBlue Bean+</i>	No	ATMEGA328P	16MHz	31.50KB	2KB

Quirkbot

Name	Debug	MCU	Frequency	Flash	RAM
<i>Quirkbot</i>	No	ATMEGA32U4	8MHz	28KB	2.50KB

RedBearLab

Name	De-bug	MCU	Fre-quency	Flash	RAM
<i>RedBearLab Blend</i>	No	AT-MEGA32U4	16MHz	28KB	2.50KB
<i>RedBearLab Blend Micro 3.3V/16MHz (overclock)</i>	No	AT-MEGA32U4	16MHz	28KB	2.50KB
<i>RedBearLab Blend Micro 3.3V/8MHz</i>	No	AT-MEGA32U4	8MHz	28KB	2.50KB

RepRap

Name	Debug	MCU	Frequency	Flash	RAM
<i>RepRap RAMBo</i>	No	ATMEGA2560	16MHz	252KB	8KB

SODAQ

Name	Debug	MCU	Frequency	Flash	RAM
<i>SODAQ GaLoRa</i>	No	ATMEGA1284P	8MHz	127KB	16KB
<i>SODAQ Mbili</i>	No	ATMEGA1284P	8MHz	127KB	16KB
<i>SODAQ Moja</i>	No	ATMEGA328P	8MHz	31.50KB	2KB
<i>SODAQ Ndogo</i>	No	ATMEGA1284P	8MHz	127KB	16KB
<i>SODAQ Tatu</i>	No	ATMEGA1284P	8MHz	127KB	16KB

Sanguino

Name	De-bug	MCU	Fre-quency	Flash	RAM
<i>Sanguino ATmega1284p (16MHz)</i>	No	AT-MEGA1284P	16MHz	127KB	16KB
<i>Sanguino ATmega1284p (8MHz)</i>	No	AT-MEGA1284P	8MHz	127KB	16KB
<i>Sanguino ATmega644 or ATmega644A (16 MHz)</i>	No	ATMEGA644	16MHz	63KB	4KB
<i>Sanguino ATmega644 or ATmega644A (8 MHz)</i>	No	ATMEGA644	8MHz	63KB	4KB
<i>Sanguino ATmega644P or ATmega644PA (16 MHz)</i>	No	ATMEGA644P	16MHz	63KB	4KB
<i>Sanguino ATmega644P or ATmega644PA (8 MHz)</i>	No	ATMEGA644P	8MHz	63KB	4KB

SeeedStudio

Name	Debug	MCU	Frequency	Flash	RAM
<i>Seeeduino</i>	No	ATMEGA328P	16MHz	31.50KB	2KB

SparkFun

Name	Debug	MCU	Frequency	Flash	RAM
<i>SparkFun ATmega128RFA1 Dev Board</i>	No	ATMEGA128RFA1	16MHz	16KB	124KB
<i>SparkFun Digital Sandbox</i>	No	ATMEGA328P	8MHz	31.50KB	2KB
<i>SparkFun Fio V3 3.3V/8MHz</i>	No	ATMEGA32U4	8MHz	28KB	2.50KB
<i>SparkFun Makey Makey</i>	No	ATMEGA32U4	16MHz	28KB	2.50KB
<i>SparkFun Mega Pro 3.3V/8MHz</i>	No	ATMEGA2560	8MHz	252KB	8KB
<i>SparkFun Mega Pro 5V/16MHz</i>	No	ATMEGA2560	16MHz	248KB	8KB
<i>SparkFun Mega Pro Mini 3.3V</i>	No	ATMEGA2560	8MHz	252KB	8KB
<i>SparkFun MicroView</i>	No	ATMEGA328P	16MHz	31.50KB	2KB
<i>SparkFun Pro Micro 3.3V/8MHz</i>	No	ATMEGA32U4	8MHz	28KB	2.50KB
<i>SparkFun Pro Micro 5V/16MHz</i>	No	ATMEGA32U4	16MHz	28KB	2.50KB
<i>SparkFun Qduino Mini</i>	No	ATMEGA32U4	8MHz	28KB	2.50KB
<i>SparkFun RedBoard</i>	No	ATMEGA328P	16MHz	31.50KB	2KB
<i>SparkFun Serial 7-Segment Display</i>	No	ATMEGA328P	8MHz	31.50KB	2KB

SpellFoundry

Name	Debug	MCU	Frequency	Flash	RAM
<i>SpellFoundry Sleepy Pi 2</i>	No	ATMEGA328P	8MHz	30KB	2KB

The Things Network

Name	Debug	MCU	Frequency	Flash	RAM
<i>The Things Uno</i>	No	ATMEGA32U4	16MHz	28KB	2.50KB

Till Harbaum

Name	Debug	MCU	Frequency	Flash	RAM
<i>ftDuino</i>	No	ATMEGA32U4	16MHz	28KB	2.50KB

TinyCircuits

Name	Debug	MCU	Frequency	Flash	RAM
<i>TinyCircuits TinyDuino Processor Board</i>	No	ATMEGA328P	8MHz	30KB	2KB
<i>TinyCircuits TinyLily Mini Processor</i>	No	ATMEGA328P	8MHz	30KB	2KB

Wicked Device

Name	Debug	MCU	Frequency	Flash	RAM
<i>Wicked Device WildFire V2</i>	No	ATMEGA1284P	16MHz	120.00KB	16KB
<i>Wicked Device WildFire V3</i>	No	ATMEGA1284P	16MHz	127KB	16KB

Wisen

Name	Debug	MCU	Frequency	Flash	RAM
<i>Talk2 Whisper Node</i>	No	ATMEGA328P	16MHz	31.50KB	2KB

makerlab.mx

Name	Debug	MCU	Frequency	Flash	RAM
<i>Altair</i>	No	ATMEGA256RFR2	16MHz	248KB	32KB

nicai-systems

Name	Debug	MCU	Frequency	Flash	RAM
<i>nicai-systems BOB3 coding bot</i>	No	ATMEGA88	8MHz	8KB	1KB
<i>nicai-systems NIBO 2 robot</i>	No	ATMEGA128	16MHz	128KB	4KB
<i>nicai-systems NIBO burger robot</i>	No	ATMEGA16	15MHz	16KB	1KB
<i>nicai-systems NIBO burger robot with Tuning Kit</i>	No	ATMEGA1284P	20MHz	128KB	16KB
<i>nicai-systems NIBObee robot</i>	No	ATMEGA16	15MHz	16KB	1KB
<i>nicai-systems NIBObee robot with Tuning Kit</i>	No	ATMEGA1284P	20MHz	128KB	16KB

ublQio

Name	Debug	MCU	Frequency	Flash	RAM
<i>ublQio Ardhat</i>	No	ATMEGA328P	16MHz	31.50KB	2KB

Atmel SAM

Configuration *platform* = 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

Bands listed below have on-board debug probe and **ARE READY** for debugging! You do not need to use/buy external debug probe.

Name	MCU	Frequency	Flash	RAM
<i>Arduino M0 Pro (Programming/Debug Port)</i>	SAMD21G18A	48MHz	256KB	32KB
<i>Arduino Zero (Programming/Debug Port)</i>	SAMD21G18A	48MHz	256KB	32KB
<i>Atmel ATSAMR21-XPRO</i>	SAMR21G18A	48MHz	256KB	32KB
<i>Atmel ATSAMW25-XPRO</i>	SAMD21G18A	48MHz	256KB	32KB
<i>Atmel SAMD21-XPRO</i>	SAMD21J18A	48MHz	256KB	32KB
<i>Atmel SAML21-XPRO-B</i>	SAML21J18B	48MHz	256KB	32KB

External Debug Tools

Bands 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.

Name	MCU	Frequency	Flash	RAM
<i>Adafruit Circuit Playground Express</i>	SAMD21G18A	48MHz	256KB	32KB
<i>Adafruit Crickit M0</i>	SAMD21G18A	48MHz	256KB	32KB
<i>Adafruit Feather M0</i>	SAMD21G18A	48MHz	256KB	32KB

Continued on next page

Table 3 – continued from previous page

Name	MCU	Frequency	Flash	RAM
<i>Adafruit Feather M0 Express</i>	SAMD21G18A	48MHz	256KB	32KB
<i>Adafruit Feather M4 Express</i>	SAMD51J19A	120MHz	512KB	192KB
<i>Adafruit Gemma M0</i>	SAMD21E18A	48MHz	256KB	32KB
<i>Adafruit Grand Central M4</i>	SAMD51P20A	120MHz	1MB	256KB
<i>Adafruit Hallowing M0</i>	SAMD21G18A	48MHz	256KB	32KB
<i>Adafruit Hallowing M4</i>	SAMD51J19A	120MHz	496KB	192KB
<i>Adafruit ItsyBitsy M0</i>	SAMD21G18A	48MHz	256KB	32KB
<i>Adafruit ItsyBitsy M4</i>	SAMD51G19A	120MHz	512KB	192KB
<i>Adafruit MONSTER M4SK</i>	SAMD51J19A	120MHz	496KB	192KB
<i>Adafruit Metro M0 Expresss</i>	SAMD21G18A	48MHz	256KB	32KB
<i>Adafruit Metro M4</i>	SAMD51J19A	120MHz	512KB	192KB
<i>Adafruit Metro M4 AirLift Lite</i>	SAMD51J19A	120MHz	512KB	192KB
<i>Adafruit PyGamer Advance M4</i>	SAMD51J20A	120MHz	1MB	256KB
<i>Adafruit PyGamer M4 Express</i>	SAMD51J19A	120MHz	512KB	192KB
<i>Adafruit PyPortal M4</i>	SAMD51J20A	120MHz	1MB	256KB
<i>Adafruit Trellis M4</i>	SAMD51J19A	120MHz	512KB	192KB
<i>Adafruit Trinket M0</i>	SAMD21E18A	48MHz	256KB	32KB
<i>Adafruit pIRkey</i>	SAMD21E18A	48MHz	256KB	32KB
<i>Adafruit pyBadge AirLift M4</i>	SAMD51J20A	120MHz	1008KB	192KB
<i>Adafruit pyBadge M4 Express</i>	SAMD51J19A	120MHz	512KB	192KB
<i>Arduino Due (Programming Port)</i>	AT91SAM3X8E	84MHz	512KB	96KB
<i>Arduino Due (USB Native Port)</i>	AT91SAM3X8E	84MHz	512KB	96KB
<i>Arduino M0</i>	SAMD21G18A	48MHz	256KB	32KB
<i>Arduino M0 Pro (Native USB Port)</i>	SAMD21G18A	48MHz	256KB	32KB
<i>Arduino MKR FOX 1200</i>	SAMD21G18A	48MHz	256KB	32KB
<i>Arduino MKR GSM 1400</i>	SAMD21G18A	48MHz	256KB	32KB
<i>Arduino MKR NB 1500</i>	SAMD21G18A	48MHz	256KB	32KB
<i>Arduino MKR WAN 1300</i>	SAMD21G18A	48MHz	256KB	32KB
<i>Arduino MKR WiFi 1010</i>	SAMD21G18A	48MHz	256KB	32KB
<i>Arduino MKR1000</i>	SAMD21G18A	48MHz	256KB	32KB
<i>Arduino MKRZERO</i>	SAMD21G18A	48MHz	256KB	32KB
<i>Arduino Tian</i>	SAMD21G18A	48MHz	256KB	32KB
<i>Arduino Zero (USB Native Port)</i>	SAMD21G18A	48MHz	256KB	32KB
<i>Digistump DigiX</i>	AT91SAM3X8E	84MHz	512KB	96KB
<i>MKR Vidor 4000</i>	SAMD21G18A	48MHz	256KB	32KB
<i>Macchina M2</i>	AT91SAM3X8E	84MHz	512KB	96KB
<i>Minitronics v2.0</i>	SAMD21J18A	48MHz	256KB	32KB
<i>Moteino M0</i>	SAMD21G18A	48MHz	256KB	32KB
<i>NANO 33 IoT</i>	SAMD21G18A	48MHz	256KB	32KB
<i>SODAQ Autonomo</i>	SAMD21J18A	48MHz	256KB	32KB
<i>SODAQ ExpLoRer</i>	SAMD21J18A	48MHz	256KB	32KB
<i>SODAQ ONE</i>	SAMD21G18A	48MHz	256KB	32KB
<i>SODAQ SARA</i>	SAMD21J18A	48MHz	256KB	32KB
<i>SODAQ SFF</i>	SAMD21G18A	48MHz	256KB	32KB
<i>SainSmart Due (Programming Port)</i>	AT91SAM3X8E	84MHz	512KB	96KB
<i>SainSmart Due (USB Native Port)</i>	AT91SAM3X8E	84MHz	512KB	96KB
<i>Seeeduino LoRaWAN</i>	SAMD21G18A	48MHz	256KB	32KB
<i>SparkFun SAMD21 Dev Breakout</i>	SAMD21G18A	48MHz	256KB	32KB
<i>SparkFun SAMD21 Mini Breakout</i>	SAMD21G18A	48MHz	256KB	32KB

Continued on next page

Table 3 – continued from previous page

Name	MCU	Frequency	Flash	RAM
Tuino 096	SAMD21G18A	48MHz	256KB	32KB

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

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

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

Upstream

```
[env:upstream_develop]
platform = https://github.com/platformio/platform-atmelsam.git
board = ...
```

Packages

Name	Description
framework-arduinosam	Arduino Wiring-based Framework (SAM Core)
framework-mbed	mbed Framework
framework-simba	Simba Framework
tool-avrdude	AVRDUDE
tool-bossac	BOSSA CLI
tool-jlink	SEGGER J-Link Software and Documentation Pack
tool-openocd	OpenOCD
toolchain-gccarmnoneeabi	gcc-arm-embedded

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

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.
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.
Simba	Simba is an RTOS and build framework. It aims to make embedded programming easy and portable.

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

Name	Debug	MCU	Frequency	Flash	RAM
Adafruit Circuit Playground Express	External	SAMD21G18A	48MHz	256KB	32KB
Adafruit Crickit M0	External	SAMD21G18A	48MHz	256KB	32KB
Adafruit Feather M0	External	SAMD21G18A	48MHz	256KB	32KB
Adafruit Feather M0 Express	External	SAMD21G18A	48MHz	256KB	32KB
Adafruit Feather M4 Express	External	SAMD51J19A	120MHz	512KB	192KB
Adafruit Gemma M0	External	SAMD21E18A	48MHz	256KB	32KB
Adafruit Grand Central M4	External	SAMD51P20A	120MHz	1MB	256KB
Adafruit Hallowing M0	External	SAMD21G18A	48MHz	256KB	32KB
Adafruit Hallowing M4	External	SAMD51J19A	120MHz	496KB	192KB
Adafruit ItsyBitsy M0	External	SAMD21G18A	48MHz	256KB	32KB
Adafruit ItsyBitsy M4	External	SAMD51G19A	120MHz	512KB	192KB
Adafruit MONSTER M4SK	External	SAMD51J19A	120MHz	496KB	192KB
Adafruit Metro M0 Expresss	External	SAMD21G18A	48MHz	256KB	32KB
Adafruit Metro M4	External	SAMD51J19A	120MHz	512KB	192KB
Adafruit Metro M4 AirLift Lite	External	SAMD51J19A	120MHz	512KB	192KB
Adafruit PyGamer Advance M4	External	SAMD51J20A	120MHz	1MB	256KB
Adafruit PyGamer M4 Express	External	SAMD51J19A	120MHz	512KB	192KB
Adafruit PyPortal M4	External	SAMD51J20A	120MHz	1MB	256KB
Adafruit Trellis M4	External	SAMD51J19A	120MHz	512KB	192KB
Adafruit Trinket M0	External	SAMD21E18A	48MHz	256KB	32KB
Adafruit pIRkey	External	SAMD21E18A	48MHz	256KB	32KB
Adafruit pyBadge AirLift M4	External	SAMD51J20A	120MHz	1008KB	192KB
Adafruit pyBadge M4 Express	External	SAMD51J19A	120MHz	512KB	192KB

Arduino

Name	Debug	MCU	Frequency	Flash	RAM
<i>Arduino Due (Programming Port)</i>	External	AT91SAM3X8E	84MHz	512KB	96KB
<i>Arduino Due (USB Native Port)</i>	External	AT91SAM3X8E	84MHz	512KB	96KB
<i>Arduino M0</i>	External	SAMD21G18A	48MHz	256KB	32KB
<i>Arduino M0 Pro (Native USB Port)</i>	External	SAMD21G18A	48MHz	256KB	32KB
<i>Arduino M0 Pro (Programming/Debug Port)</i>	On-board	SAMD21G18A	48MHz	256KB	32KB
<i>Arduino MKR FOX 1200</i>	External	SAMD21G18A	48MHz	256KB	32KB
<i>Arduino MKR GSM 1400</i>	External	SAMD21G18A	48MHz	256KB	32KB
<i>Arduino MKR NB 1500</i>	External	SAMD21G18A	48MHz	256KB	32KB
<i>Arduino MKR WAN 1300</i>	External	SAMD21G18A	48MHz	256KB	32KB
<i>Arduino MKR WiFi 1010</i>	External	SAMD21G18A	48MHz	256KB	32KB
<i>Arduino MKR1000</i>	External	SAMD21G18A	48MHz	256KB	32KB
<i>Arduino MKRZERO</i>	External	SAMD21G18A	48MHz	256KB	32KB
<i>Arduino Tian</i>	External	SAMD21G18A	48MHz	256KB	32KB
<i>Arduino Zero (Programming/Debug Port)</i>	On-board	SAMD21G18A	48MHz	256KB	32KB
<i>Arduino Zero (USB Native Port)</i>	External	SAMD21G18A	48MHz	256KB	32KB
<i>MKR Vidor 4000</i>	External	SAMD21G18A	48MHz	256KB	32KB
<i>NANO 33 IoT</i>	External	SAMD21G18A	48MHz	256KB	32KB

Atmel

Name	Debug	MCU	Frequency	Flash	RAM
<i>Atmel ATSAMR21-XPRO</i>	On-board	SAMR21G18A	48MHz	256KB	32KB
<i>Atmel ATSAMW25-XPRO</i>	On-board	SAMD21G18A	48MHz	256KB	32KB
<i>Atmel SAMD21-XPRO</i>	On-board	SAMD21J18A	48MHz	256KB	32KB
<i>Atmel SAML21-XPRO-B</i>	On-board	SAML21J18B	48MHz	256KB	32KB

Digistump

Name	Debug	MCU	Frequency	Flash	RAM
<i>Digistump DigiX</i>	External	AT91SAM3X8E	84MHz	512KB	96KB

Gimasi

Name	Debug	MCU	Frequency	Flash	RAM
<i>Tuino 096</i>	External	SAMD21G18A	48MHz	256KB	32KB

LowPowerLab

Name	Debug	MCU	Frequency	Flash	RAM
<i>LowPowerLab CurrentRanger</i>	No	SAMD21G18A	48MHz	256KB	32KB
<i>Moteino M0</i>	External	SAMD21G18A	48MHz	256KB	32KB

Macchina

Name	Debug	MCU	Frequency	Flash	RAM
<i>Macchina M2</i>	External	AT91SAM3X8E	84MHz	512KB	96KB

ReprapWorld

Name	Debug	MCU	Frequency	Flash	RAM
<i>Minitronics v2.0</i>	External	SAMD21J18A	48MHz	256KB	32KB

SODAQ

Name	Debug	MCU	Frequency	Flash	RAM
<i>SODAQ Autonomo</i>	External	SAMD21J18A	48MHz	256KB	32KB
<i>SODAQ ExpLoRer</i>	External	SAMD21J18A	48MHz	256KB	32KB
<i>SODAQ ONE</i>	External	SAMD21G18A	48MHz	256KB	32KB
<i>SODAQ SARA</i>	External	SAMD21J18A	48MHz	256KB	32KB
<i>SODAQ SFF</i>	External	SAMD21G18A	48MHz	256KB	32KB

SainSmart

Name	Debug	MCU	Frequency	Flash	RAM
<i>SainSmart Due (Programming Port)</i>	External	AT91SAM3X8E	84MHz	512KB	96KB
<i>SainSmart Due (USB Native Port)</i>	External	AT91SAM3X8E	84MHz	512KB	96KB

Seeed

Name	Debug	MCU	Frequency	Flash	RAM
<i>Seeeduino LoRaWAN</i>	External	SAMD21G18A	48MHz	256KB	32KB

SparkFun

Name	Debug	MCU	Frequency	Flash	RAM
<i>SparkFun SAMD21 Dev Breakout</i>	External	SAMD21G18A	48MHz	256KB	32KB
<i>SparkFun SAMD21 Mini Breakout</i>	External	SAMD21G18A	48MHz	256KB	32KB

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*)

```
[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

```
[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
```

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](#):

```
[env:myenv]
platform = espressif32
framework = arduino
board = ...
build_flags =
    -DBOARD_HAS_PSRAM
    -mfix-esp32-psram-cache-issue
```

Framework [ESP-IDF](#):

```
[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.

```
[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*)

```
[env:myenv]
upload_speed = 9600
```

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:

```
; 1) A "partitions_custom.csv" in the root of project directory
[env:custom_table]
board_build.partitions = partitions_custom.csv

; 2) Switch between built-in tables
; https://github.com/espressif/arduino-esp32/tree/master/tools/partitions
; https://github.com/espressif/esp-idf/tree/master/components/partition_table
[env:custom_builtin_table]
board_build.partitions = no_ota.csv
```

Embedding Binary Data

Sometimes you have a file with some binary or text data that you'd like to make available to your program - but you don't want to reformat the file as C source.

You can set a macro (define) COMPONENT_EMBED_TXTFILES using *build_flags* in “*platformio.ini*” (*Project Configuration 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
└── lib
    └── README
```

(continues on next page)

(continued from previous 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` command with `uploadfs` target.

To upload SPIFFS image using OTA update please specify `upload_port` / `--upload-port` as IP address or mDNS host name (ending with the `*.local`).

Examples:

- SPIFFS for Arduino
- SPIFFS for ESP-IDF

Over-the-Air (OTA) update

Using JFrog Bintray (free and secure Cloud solution)

- Video and presentation - swampUP: Over-The-Air (OTA) firmware upgrades for Internet of Things devices with PlatformIO and JFrog Bintray
- Demo source code: <https://github.com/platformio/bintray-secure-ota>

Using built-in Local solution

Demo code for:

- Arduino
- ESP-IDF

There are 2 options:

- Directly specify `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)

(continued from previous 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-esp32@feature/stage`. For example,

```
[env:esp32dev]
platform = https://github.com/platformio/platform-esp32@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

Tips, tricks and common problems: <http://desire.giesecke.tk/index.php/2018/01/30/esp32-wiki-entries/>

Examples

Examples are listed from Espressif 32 development platform repository:

- `arduino-blink`
- `arduino-wifiscan`
- `esp32-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

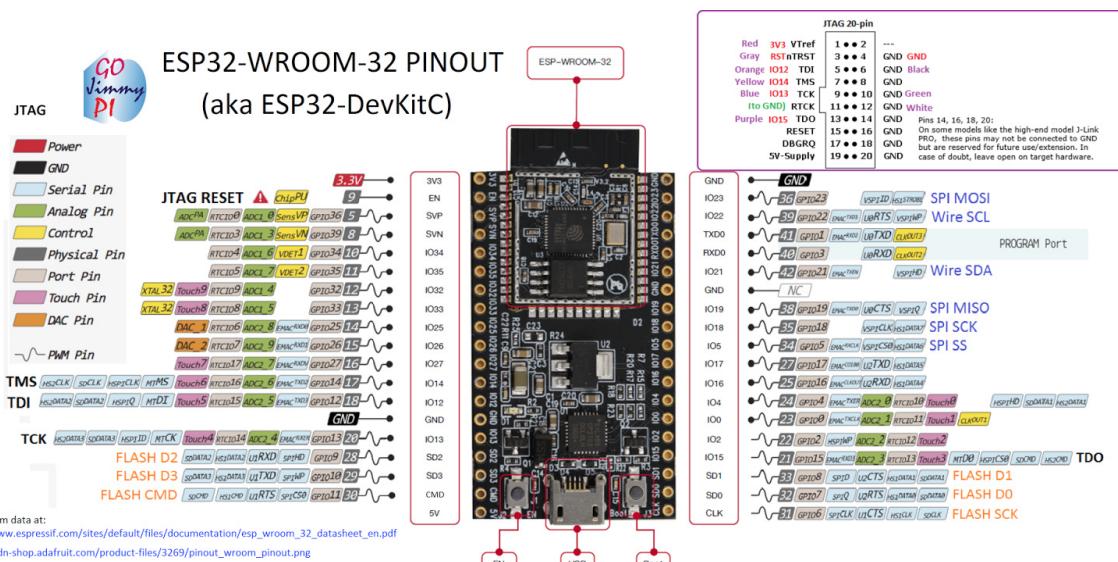
PIO Unified Debugger - “1-click” solution for debugging with a zero configuration.

- *Pinout Diagram*
- *Tools & Debug Probes*
 - *On-Board Debug Tools*
 - *External Debug Tools*

Pinout Diagram

JTAG Wiring Connections

Board Pin	JTAG Tool Pin
IO13	TCK
IO12	TDI
IO15	TDO
IO14	TMS
EN	RST
GND	GND



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

Bands listed below have on-board debug probe and **ARE READY** for debugging! You do not need to use/buy external debug probe.

Name	MCU	Frequency	Flash	RAM
<i>Espressif ESP-WROVER-KIT</i>	ESP32	240MHz	4MB	320KB

External Debug Tools

Bands 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.

Name	MCU	Frequency	Flash	RAM
<i>AI Thinker ESP32-CAM</i>	ESP32	240MHz	4MB	320KB
<i>ALKS ESP32</i>	ESP32	240MHz	4MB	320KB
<i>Adafruit ESP32 Feather</i>	ESP32	240MHz	4MB	320KB
<i>D-duino-32</i>	ESP32	240MHz	4MB	320KB

Continued on next page

Table 4 – continued from previous page

Name	MCU	Frequency	Flash	RAM
<i>DOIT ESP32 DEVKIT V1</i>	ESP32	240MHz	4MB	320KB
<i>Dongsen Tech Pocket 32</i>	ESP32	240MHz	4MB	320KB
<i>ESP32 FM DevKit</i>	ESP32	240MHz	4MB	320KB
<i>ESP32vn IoT Uno</i>	ESP32	240MHz	4MB	320KB
<i>ESPectro32</i>	ESP32	240MHz	4MB	320KB
<i>ESPino32</i>	ESP32	240MHz	4MB	320KB
<i>Espressif ESP32 Dev Module</i>	ESP32	240MHz	4MB	320KB
<i>FireBeetle-ESP32</i>	ESP32	240MHz	4MB	320KB
<i>Frog Board ESP32</i>	ESP32	240MHz	4MB	320KB
<i>Heltec WiFi LoRa 32</i>	ESP32	240MHz	4MB	320KB
<i>Heltec WiFi LoRa 32 (V2)</i>	ESP32	240MHz	8MB	320KB
<i>Heltec Wireless Stick</i>	ESP32	240MHz	8MB	320KB
<i>Hornbill ESP32 Dev</i>	ESP32	240MHz	4MB	320KB
<i>Hornbill ESP32 Minima</i>	ESP32	240MHz	4MB	320KB
<i>IoTaaP Magnolia</i>	ESP32	240MHz	4MB	320KB
<i>MH ET LIVE ESP32DevKIT</i>	ESP32	240MHz	4MB	320KB
<i>MH ET LIVE ESP32MiniKit</i>	ESP32	240MHz	4MB	320KB
<i>Node32s</i>	ESP32	240MHz	4MB	320KB
<i>NodeMCU-32S</i>	ESP32	240MHz	4MB	320KB
<i>OLIMEX ESP32-DevKit-LiPo</i>	ESP32	240MHz	4MB	320KB
<i>OLIMEX ESP32-EVB</i>	ESP32	240MHz	4MB	320KB
<i>OLIMEX ESP32-GATEWAY</i>	ESP32	240MHz	4MB	320KB
<i>Pycom LoPy</i>	ESP32	240MHz	4MB	320KB
<i>Pycom LoPy4</i>	ESP32	240MHz	4MB	1.25MB
<i>Silicognition wESP32</i>	ESP32	240MHz	4MB	320KB
<i>SparkFun ESP32 Thing</i>	ESP32	240MHz	4MB	320KB
<i>SparkFun LoRa Gateway 1-Channel</i>	ESP32	240MHz	4MB	320KB
<i>TTGO LoRa32-OLED VI</i>	ESP32	240MHz	4MB	320KB
<i>TTGO T-Beam</i>	ESP32	240MHz	4MB	1.25MB
<i>TTGO T1</i>	ESP32	240MHz	4MB	320KB
<i>VintLabs ESP32 Devkit</i>	ESP32	240MHz	4MB	320KB
<i>WEMOS LOLIN D32</i>	ESP32	240MHz	4MB	320KB
<i>WEMOS LOLIN D32 PRO</i>	ESP32	240MHz	4MB	320KB
<i>WEMOS LOLIN32</i>	ESP32	240MHz	4MB	320KB
<i>WeMos D1 MINI ESP32</i>	ESP32	240MHz	4MB	320KB
<i>WeMos WiFi and Bluetooth Battery</i>	ESP32	240MHz	4MB	320KB
<i>XinaBox CW02</i>	ESP32	240MHz	4MB	320KB
<i>oddWires IoT-Bus Io</i>	ESP32	240MHz	4MB	320KB
<i>oddWires IoT-Bus Proteus</i>	ESP32	240MHz	4MB	320KB

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

```
; Latest stable version
[env:latest_stable]
platform = espressif32
board = ...

; Custom stable version
[env:custom_stable]
platform = espressif32@x.y.z
board = ...
```

Upstream

```
[env:upstream_develop]
platform = https://github.com/platformio/platform-espressif32.git
board = ...
```

Packages

Name	Description
framework-arduinoespressif32	Arduino Wiring-based Framework (ESP32 Core)
framework-espidf	Espressif IoT Development Framework
framework-pumbaa	Pumbaa Framework
framework-simba	Simba Framework
tool-esptoolpy	ESP8266 and ESP32 serial bootloader utility
tool-mkspiffs	Tool to build and unpack SPIFFS images
tool-openocd-esp32	OpenOCD for Espressif 32
toolchain-esp32ulp	Binutils fork with support for the ESP32 ULP co-processor
toolchain-xtensa32	xtensa32-gcc

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

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP-IDF</i>	Espressif IoT Development Framework. Official development framework for ESP32.
<i>Pumba</i>	Pumba 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.
<i>Simba</i>	Simba is an RTOS and build framework. It aims to make embedded programming easy and portable.

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

Name	Debug	MCU	Frequency	Flash	RAM
<i>AI Thinker ESP32-CAM</i>	External	ESP32	240MHz	4MB	320KB

Adafruit

Name	Debug	MCU	Frequency	Flash	RAM
<i>Adafruit ESP32 Feather</i>	External	ESP32	240MHz	4MB	320KB

Aiyarafun

Name	Debug	MCU	Frequency	Flash	RAM
<i>Node32s</i>	External	ESP32	240MHz	4MB	320KB

April Brother

Name	Debug	MCU	Frequency	Flash	RAM
<i>April Brother ESPea32</i>	No	ESP32	240MHz	4MB	320KB

BPI Tech

Name	Debug	MCU	Frequency	Flash	RAM
<i>BPI-Bit</i>	No	ESP32	160MHz	4MB	320KB

DFRobot

Name	Debug	MCU	Frequency	Flash	RAM
<i>FireBeetle-ESP32</i>	External	ESP32	240MHz	4MB	320KB

DOIT

Name	Debug	MCU	Frequency	Flash	RAM
<i>DOIT ESP32 DEVKIT V1</i>	External	ESP32	240MHz	4MB	320KB

DSTIKE

Name	Debug	MCU	Frequency	Flash	RAM
<i>D-duino-32</i>	External	ESP32	240MHz	4MB	320KB

Dongsen Technology

Name	Debug	MCU	Frequency	Flash	RAM
<i>Dongsen Tech Pocket 32</i>	External	ESP32	240MHz	4MB	320KB

DycodeX

Name	Debug	MCU	Frequency	Flash	RAM
<i>ESPectro32</i>	External	ESP32	240MHz	4MB	320KB

ESP32vn

Name	Debug	MCU	Frequency	Flash	RAM
<i>ESP32vn IoT Uno</i>	External	ESP32	240MHz	4MB	320KB

Electronic SweetPeas

Name	Debug	MCU	Frequency	Flash	RAM
<i>Electronic SweetPeas ESP320</i>	No	ESP32	240MHz	4MB	320KB

Espressif

Name	Debug	MCU	Frequency	Flash	RAM
<i>ESP32 Pico Kit</i>	No	ESP32	240MHz	4MB	320KB
<i>Espressif ESP-WROVER-KIT</i>	On-board	ESP32	240MHz	4MB	320KB
<i>Espressif ESP32 Dev Module</i>	External	ESP32	240MHz	4MB	320KB

Fred

Name	Debug	MCU	Frequency	Flash	RAM
<i>Frog Board ESP32</i>	External	ESP32	240MHz	4MB	320KB

Hardkernel

Name	Debug	MCU	Frequency	Flash	RAM
<i>ODROID-GO</i>	No	ESP32	240MHz	16MB	320KB

Heltec Automation

Name	Debug	MCU	Frequency	Flash	RAM
<i>Heltec WiFi Kit 32</i>	No	ESP32	240MHz	4MB	320KB
<i>Heltec WiFi LoRa 32</i>	External	ESP32	240MHz	4MB	320KB
<i>Heltec WiFi LoRa 32 (V2)</i>	External	ESP32	240MHz	8MB	320KB
<i>Heltec Wireless Stick</i>	External	ESP32	240MHz	8MB	320KB

Hornbill

Name	Debug	MCU	Frequency	Flash	RAM
<i>Hornbill ESP32 Dev</i>	External	ESP32	240MHz	4MB	320KB
<i>Hornbill ESP32 Minima</i>	External	ESP32	240MHz	4MB	320KB

IntoRobot

Name	Debug	MCU	Frequency	Flash	RAM
<i>IntoRobot Fig</i>	No	ESP32	240MHz	4MB	320KB

M5Stack

Name	Debug	MCU	Frequency	Flash	RAM
<i>M5Stack Core ESP32</i>	No	ESP32	240MHz	4MB	320KB
<i>M5Stack FIRE</i>	No	ESP32	240MHz	16MB	6.25MB
<i>M5Stack GREY ESP32</i>	No	ESP32	240MHz	16MB	520KB
<i>M5Stick-C</i>	No	ESP32	240MHz	4MB	320KB

MH-ET Live

Name	Debug	MCU	Frequency	Flash	RAM
<i>MH ET LIVE ESP32DevKIT</i>	External	ESP32	240MHz	4MB	320KB
<i>MH ET LIVE ESP32MiniKit</i>	External	ESP32	240MHz	4MB	320KB

MVT Solutions

Name	Debug	MCU	Frequency	Flash	RAM
<i>IoTaaP Magnolia</i>	External	ESP32	240MHz	4MB	320KB

Magicblocks.io

Name	Debug	MCU	Frequency	Flash	RAM
<i>MagicBit</i>	No	ESP32	240MHz	4MB	320KB

MakerAsia

Name	Debug	MCU	Frequency	Flash	RAM
<i>MakerAsia Nano32</i>	No	ESP32	240MHz	4MB	320KB

Micoduino

Name	Debug	MCU	Frequency	Flash	RAM
<i>Micoduino Core ESP32</i>	No	ESP32	240MHz	4MB	320KB

NodeMCU

Name	Debug	MCU	Frequency	Flash	RAM
<i>NodeMCU-32S</i>	External	ESP32	240MHz	4MB	320KB

Noduino

Name	Debug	MCU	Frequency	Flash	RAM
<i>Noduino Quantum</i>	No	ESP32	240MHz	16MB	320KB

OLIMEX

Name	Debug	MCU	Frequency	Flash	RAM
<i>OLIMEX ESP32-DevKit-LiPo</i>	External	ESP32	240MHz	4MB	320KB
<i>OLIMEX ESP32-EVB</i>	External	ESP32	240MHz	4MB	320KB
<i>OLIMEX ESP32-GATEWAY</i>	External	ESP32	240MHz	4MB	320KB
<i>OLIMEX ESP32-PRO</i>	No	ESP32	240MHz	4MB	320KB
<i>OLIMEX ESP32-PoE</i>	No	ESP32	240MHz	4MB	320KB
<i>OLIMEX ESP32-PoE-ISO</i>	No	ESP32	240MHz	4MB	320KB

OROCA

Name	Debug	MCU	Frequency	Flash	RAM
<i>OROCA EduBot</i>	No	ESP32	240MHz	4MB	320KB

Onehorse

Name	Debug	MCU	Frequency	Flash	RAM
<i>Onehorse ESP32 Dev Module</i>	No	ESP32	240MHz	4MB	320KB

Pycom Ltd.

Name	Debug	MCU	Frequency	Flash	RAM
<i>Pycom GPy</i>	No	ESP32	240MHz	4MB	320KB
<i>Pycom LoPy</i>	External	ESP32	240MHz	4MB	320KB
<i>Pycom LoPy4</i>	External	ESP32	240MHz	4MB	1.25MB

RoboticsBrno

Name	Debug	MCU	Frequency	Flash	RAM
<i>ALKS ESP32</i>	External	ESP32	240MHz	4MB	320KB

Silicognition

Name	Debug	MCU	Frequency	Flash	RAM
<i>Silicognition wESP32</i>	External	ESP32	240MHz	4MB	320KB

SparkFun

Name	Debug	MCU	Frequency	Flash	RAM
SparkFun LoRa Gateway 1-Channel	External	ESP32	240MHz	4MB	320KB

SparkFun Electronics

Name	Debug	MCU	Frequency	Flash	RAM
SparkFun ESP32 Thing	External	ESP32	240MHz	4MB	320KB

TTGO

Name	Debug	MCU	Frequency	Flash	RAM
TTGO LoRa32-OLED V1	External	ESP32	240MHz	4MB	320KB
TTGO T-Beam	External	ESP32	240MHz	4MB	1.25MB
TTGO T-Watch	No	ESP32	240MHz	16MB	320KB
TTGO T1	External	ESP32	240MHz	4MB	320KB

ThaiEasyElec

Name	Debug	MCU	Frequency	Flash	RAM
ESPino32	External	ESP32	240MHz	4MB	320KB

TinyPICO

Name	Debug	MCU	Frequency	Flash	RAM
TinyPICO	No	ESP32	240MHz	4MB	320KB

Turta

Name	Debug	MCU	Frequency	Flash	RAM
Turta IoT Node	No	ESP32	240MHz	4MB	320KB

Unknown

Name	Debug	MCU	Frequency	Flash	RAM
ESP32 FM DevKit	External	ESP32	240MHz	4MB	320KB

VintLabs

Name	Debug	MCU	Frequency	Flash	RAM
<i>VintLabs ESP32 Devkit</i>	External	ESP32	240MHz	4MB	320KB

WEMOS

Name	Debug	MCU	Frequency	Flash	RAM
<i>WEMOS LOLIN D32</i>	External	ESP32	240MHz	4MB	320KB
<i>WEMOS LOLIN D32 PRO</i>	External	ESP32	240MHz	4MB	320KB
<i>WEMOS LOLIN32</i>	External	ESP32	240MHz	4MB	320KB
<i>WeMos D1 MINI ESP32</i>	External	ESP32	240MHz	4MB	320KB
<i>WeMos WiFi and Bluetooth Battery</i>	External	ESP32	240MHz	4MB	320KB

Widora

Name	Debug	MCU	Frequency	Flash	RAM
<i>Widora AIR</i>	No	ESP32	240MHz	16MB	320KB

XinaBox

Name	Debug	MCU	Frequency	Flash	RAM
<i>XinaBox CW02</i>	External	ESP32	240MHz	4MB	320KB

oddWires

Name	Debug	MCU	Frequency	Flash	RAM
<i>oddWires IoT-Bus Io</i>	External	ESP32	240MHz	4MB	320KB
<i>oddWires IoT-Bus Proteus</i>	External	ESP32	240MHz	4MB	320KB

u-blox

Name	Debug	MCU	Frequency	Flash	RAM
<i>u-blox NINA-W10 series</i>	No	ESP32	240MHz	2MB	320KB

Espressif 8266

Configuration *platform* = espressif8266

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

For more detailed information please visit [vendor site](#).

Contents

- [Configuration](#)
- [Examples](#)
- [Stable and upstream versions](#)
- [Packages](#)
- [Frameworks](#)
- [Boards](#)

Configuration

- [CPU Frequency](#)
- [FLASH Frequency](#)
- [FLASH Mode](#)
- [Reset Method](#)
- [Flash Size](#)
- [Upload Speed](#)
- [lwIP Variant](#)
- [SSL Support](#)
- [Serial Debug](#)
- [Debug Level](#)
- [VTables](#)
- [Exceptions](#)
- [Uploading files to file system SPIFFS](#)
- [Over-the-Air \(OTA\) update](#)
 - [Authentication and upload options](#)
- [Demo](#)
- [Using Arduino Framework with Staging version](#)

CPU Frequency

See `board_build.f_cpu` option from “`platformio.ini`” (*Project Configuration File*)

```
[env:myenv]
; set frequency to 160MHz
board_build.f_cpu = 160000000L
```

FLASH Frequency

Please use `board_build.f_flash` option from “*platformio.ini*” (*Project Configuration File*) to change a value. Possible values:

- 20000000L
- 26000000L
- 40000000L (default)
- 80000000L

```
[env:myenv]
; set frequency to 80MHz
board_build.f_flash = 80000000L
```

FLASH Mode

Flash chip interface mode. This parameter is stored in the binary image header, along with the flash size and flash frequency. The ROM bootloader in the ESP chip uses the value of these parameters in order to know how to talk to the flash chip.

Please use `board_build.flash_mode` option from “*platformio.ini*” (*Project Configuration File*) to change a value. Possible values:

- qio
- qout
- dio
- dout

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

Reset Method

You can set custom reset method using `upload_resetmethod` option from “*platformio.ini*” (*Project Configuration File*).

The possible values are:

- ck - RTS controls RESET or CH_PD, DTR controls GPIO0
- wifi0 - 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

; 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
```

(continues on next page)

(continued from previous page)

```
; SSL+HTTP_SERVER
build_flags =
-DDEBUG_ESP_SSL
-DDEBUG_ESP_HTTP_SERVER

; TLS_MEM+HTTP_CLIENT
build_flags =
-DDEBUG_ESP_TLS_MEM
-DDEBUG_ESP_HTTP_CLIENT

; TLS_MEM+HTTP_SERVER
build_flags =
-DDEBUG_ESP_TLS_MEM
-DDEBUG_ESP_HTTP_SERVER

; HTTP_CLIENT+HTTP_SERVER
build_flags =
-DDEBUG_ESP_HTTP_CLIENT
-DDEBUG_ESP_HTTP_SERVER

; SSL+TLS_MEM+HTTP_CLIENT
build_flags =
-DDEBUG_ESP_SSL
-DDEBUG_ESP_TLS_MEM
-DDEBUG_ESP_HTTP_CLIENT

; SSL+TLS_MEM+HTTP_SERVER
build_flags =
-DDEBUG_ESP_SSL
-DDEBUG_ESP_TLS_MEM
-DDEBUG_ESP_HTTP_SERVER

; SSL+HTTP_CLIENT+HTTP_SERVER
build_flags =
-DDEBUG_ESP_SSL
-DDEBUG_ESP_HTTP_CLIENT
-DDEBUG_ESP_HTTP_SERVER

; TLS_MEM+HTTP_CLIENT+HTTP_SERVER
build_flags =
-DDEBUG_ESP_TLS_MEM
-DDEBUG_ESP_HTTP_CLIENT
-DDEBUG_ESP_HTTP_SERVER

; SSL+TLS_MEM+HTTP_CLIENT+HTTP_SERVER
build_flags =
-DDEBUG_ESP_SSL
-DDEBUG_ESP_TLS_MEM
-DDEBUG_ESP_HTTP_CLIENT
-DDEBUG_ESP_HTTP_SERVER

; CORE
build_flags = -DDEBUG_ESP_CORE

; WIFI
build_flags = -DDEBUG_ESP_WIFI
```

(continues on next page)

(continued from previous page)

```

; HTTP_UPDATE
build_flags = -DDEBUG_ESP_HTTP_UPDATE

; UPDATER
build_flags = -DDEBUG_ESP_UPDATER

; OTA
build_flags = -DDEBUG_ESP_OTA

; OOM
build_flags =
  -DDEBUG_ESP_OOM
  -include "umm_malloc/umm_malloc_cfg.h"

; CORE+WIFI+HTTP_UPDATE+UPDATER+OTA+OOM
build_flags =
  -DDEBUG_ESP_CORE
  -DDEBUG_ESP_WIFI
  -DDEBUG_ESP_HTTP_UPDATE
  -DDEBUG_ESP_UPDATER
  -DDEBUG_ESP_OTA
  -DDEBUG_ESP_OOM -include "umm_malloc/umm_malloc_cfg.h"

; SSL+TLS_MEM+HTTP_CLIENT+HTTP_SERVER+CORE+WIFI+HTTP_UPDATE+UPDATER+OTA+OOM
build_flags =
  -DDEBUG_ESP_SSL
  -DDEBUG_ESP_TLS_MEM
  -DDEBUG_ESP_HTTP_CLIENT
  -DDEBUG_ESP_HTTP_SERVER
  -DDEBUG_ESP_CORE
  -DDEBUG_ESP_WIFI
  -DDEBUG_ESP_HTTP_UPDATE
  -DDEBUG_ESP_UPDATER
  -DDEBUG_ESP_OTA
  -DDEBUG_ESP_OOM -include "umm_malloc/umm_malloc_cfg.h"

; NoAssert-NDEBUG
build_flags = -DNDEBUG

```

VTables

Please use one of the next *build_flags*:

```

[env:myenv]
...
; Flash (default)
build_flags = -DVTABLES_IN_FLASH

; Heap
build_flags = -DVTABLES_IN_DRAM

; IRAM
build_flags = -DVTABLES_IN_IRAM

```

Exceptions

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

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

; Enable exceptions
build_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
; each flag in a new line
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

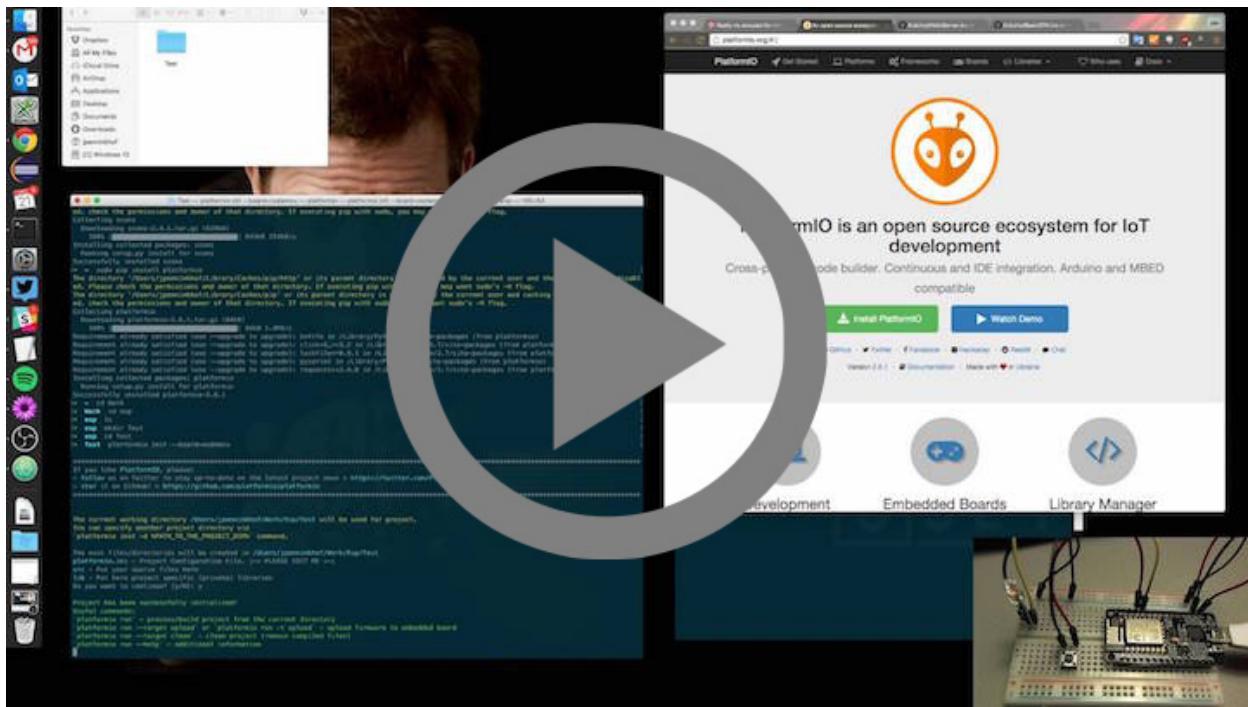
Destination:
-i ESP_IP, --ip=ESP_IP
                  ESP8266 IP Address.
-I HOST_IP, --host_ip=HOST_IP
                  Host IP Address.
-p ESP_PORT, --port=ESP_PORT
                  ESP8266 ota Port. Default 8266
-P HOST_PORT, --host_port=HOST_PORT
                  Host server ota Port. Default random 10000-60000

Authentication:
-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,

```
[env:nodemcuv2]
platform = https://github.com/platformio/platform-espressif8266.git#feature/stage
board = nodemcuv2
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>)

Examples

Examples are listed from Espressif 8266 development platform repository:

- arduino-asyncudp
- 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

```
; Latest stable version
[env:latest_stable]
platform = espressif8266
board = ...

; Custom stable version
[env:custom_stable]
platform = espressif8266@x.y.z
board = ...
```

Upstream

```
[env:upstream_develop]
platform = https://github.com/platformio/platform-espressif8266.git
board = ...
```

Packages

Name	Description
framework-arduinoespressif8266	Arduino Wiring-based Framework (ESP8266 Core)
framework-esp8266-nonos-sdk	ESP8266 Non-OS SDK
framework-esp8266-rtos-sdk	ESP8266 SDK based on FreeRTOS
framework-simba	Simba Framework
tool-esptool	esptool-ck
tool-esptoolpy	ESP8266 and ESP32 serial bootloader utility
tool-mkspiffs	Tool to build and unpack SPIFFS images
toolchain-xtensa	xtensa-gcc

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

Name	Description
<i>Arduino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP8266 Non-OS SDK</i>	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.
<i>ESP8266 RTOS SDK</i>	ESP8266 SDK based on FreeRTOS, a truly free professional grade RTOS for microcontrollers
<i>Simba</i>	Simba is an RTOS and build framework. It aims to make embedded programming easy and portable.

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

Name	Debug	MCU	Frequency	Flash	RAM
<i>4D Systems gen4 IoD Range</i>	No	ESP8266	80MHz	512KB	80KB

Adafruit

Name	Debug	MCU	Frequency	Flash	RAM
<i>Adafruit HUZZAH ESP8266</i>	No	ESP8266	80MHz	4MB	80KB

Amperka

Name	Debug	MCU	Frequency	Flash	RAM
<i>WiFi Slot</i>	No	ESP8266	80MHz	4MB	80KB

DigiStump

Name	Debug	MCU	Frequency	Flash	RAM
<i>DigiStump Oak</i>	No	ESP8266	80MHz	4MB	80KB

Doit

Name	Debug	MCU	Frequency	Flash	RAM
<i>ESPDuino (ESP-13 Module)</i>	No	ESP8266	80MHz	4MB	80KB

DycodeX

Name	Debug	MCU	Frequency	Flash	RAM
<i>ESPectro Core</i>	No	ESP8266	80MHz	4MB	80KB

ESPert

Name	Debug	MCU	Frequency	Flash	RAM
<i>ESPRESSO Lite 1.0</i>	No	ESP8266	80MHz	4MB	80KB
<i>ESPRESSO Lite 2.0</i>	No	ESP8266	80MHz	4MB	80KB

ESPino

Name	Debug	MCU	Frequency	Flash	RAM
<i>ESPino</i>	No	ESP8266	80MHz	4MB	80KB

Espressif

Name	Debug	MCU	Frequency	Flash	RAM
<i>ESP-WROOM-02</i>	No	ESP8266	80MHz	2MB	80KB
<i>Espressif ESP8266 ESP-12E</i>	No	ESP8266	80MHz	4MB	80KB
<i>Espressif Generic ESP8266 ESP-01 1M</i>	No	ESP8266	80MHz	1MB	80KB
<i>Espressif Generic ESP8266 ESP-01 512k</i>	No	ESP8266	80MHz	512KB	80KB
<i>Espressif Generic ESP8266 ESP-07</i>	No	ESP8266	80MHz	4MB	80KB
<i>Generic ESP8285 Module</i>	No	ESP8266	80MHz	1MB	80KB
<i>Phoenix 1.0</i>	No	ESP8266	80MHz	4MB	80KB
<i>Phoenix 2.0</i>	No	ESP8266	80MHz	4MB	80KB
<i>WifInfo</i>	No	ESP8266	80MHz	1MB	80KB

Heltec

Name	Debug	MCU	Frequency	Flash	RAM
<i>Heltec WiFi kit 8</i>	No	ESP8266	80MHz	4MB	80KB

Invent One

Name	Debug	MCU	Frequency	Flash	RAM
<i>Invent One</i>	No	ESP8266	80MHz	4MB	80KB

NodeMCU

Name	Debug	MCU	Frequency	Flash	RAM
<i>NodeMCU 0.9 (ESP-12 Module)</i>	No	ESP8266	80MHz	4MB	80KB
<i>NodeMCU 1.0 (ESP-12E Module)</i>	No	ESP8266	80MHz	4MB	80KB

Olimex

Name	Debug	MCU	Frequency	Flash	RAM
<i>Olimex MOD-WIFI-ESP8266(-DEV)</i>	No	ESP8266	80MHz	2MB	80KB

SeeedStudio

Name	Debug	MCU	Frequency	Flash	RAM
<i>Wio Link</i>	No	ESP8266	80MHz	4MB	80KB
<i>Wio Node</i>	No	ESP8266	80MHz	4MB	80KB

SparkFun

Name	Debug	MCU	Frequency	Flash	RAM
<i>SparkFun Blynk Board</i>	No	ESP8266	80MHz	4MB	80KB
<i>SparkFun ESP8266 Thing</i>	No	ESP8266	80MHz	512KB	80KB
<i>SparkFun ESP8266 Thing Dev</i>	No	ESP8266	80MHz	512KB	80KB

SweetPea

Name	Debug	MCU	Frequency	Flash	RAM
<i>SweetPea ESP-210</i>	No	ESP8266	80MHz	4MB	80KB

ThaiEasyElec

Name	Debug	MCU	Frequency	Flash	RAM
<i>ThaiEasyElec ESPino</i>	No	ESP8266	80MHz	4MB	80KB

WEMOS

Name	Debug	MCU	Frequency	Flash	RAM
<i>WEMOS D1 R1</i>	No	ESP8266	80MHz	4MB	80KB
<i>WeMos D1 R2 and mini</i>	No	ESP8266	80MHz	4MB	80KB
<i>WeMos D1 mini Lite</i>	No	ESP8266	80MHz	1MB	80KB
<i>WeMos D1 mini Pro</i>	No	ESP8266	80MHz	16MB	80KB

WifiDuino

Name	Debug	MCU	Frequency	Flash	RAM
<i>WiFiduino</i>	No	ESP8266	80MHz	4MB	80KB

XinaBox

Name	Debug	MCU	Frequency	Flash	RAM
<i>XinaBox CW01</i>	No	ESP8266	80MHz	4MB	80KB

Freescale Kinetis

Configuration `platform = freescalekinetics`

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](#)
 - [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

Bands listed below have on-board debug probe and **ARE READY** for debugging! You do not need to use/buy external debug probe.

Name	MCU	Frequency	Flash	RAM
<i>Ethernet IoT Starter Kit</i>	MK64FN1M0VLL12	120MHz	1MB	256KB
<i>Freescale Kinetis FRDM-K20D50M</i>	MK20DX128VLH5	48MHz	128KB	16KB
<i>Freescale Kinetis FRDM-K22F</i>	MK22FN512VLH12	120MHz	512KB	128KB
<i>Freescale Kinetis FRDM-K64F</i>	MK64FN1M0VLL12	120MHz	1MB	256KB
<i>Freescale Kinetis FRDM-K66F</i>	MK66FN2M0VMD18	180MHz	2MB	256KB
<i>Freescale Kinetis FRDM-K82F</i>	MK82FN256VLL15	150MHz	256KB	256KB
<i>Freescale Kinetis FRDM-KL05Z</i>	MKL05Z32VFM4	48MHz	32KB	4KB
<i>Freescale Kinetis FRDM-KL25Z</i>	MKL25Z128VLK4	48MHz	128KB	16KB
<i>Freescale Kinetis FRDM-KL27Z</i>	MKL27Z64VLH4	48MHz	64KB	16KB
<i>Freescale Kinetis FRDM-KL43Z</i>	MKL43Z256VLH4	48MHz	256KB	32KB
<i>Freescale Kinetis FRDM-KL46Z</i>	MKL46Z256VLL4	48MHz	256KB	32KB
<i>Freescale Kinetis FRDM-KW41Z</i>	MKW41Z512VHT4	48MHz	512KB	128KB

External Debug Tools

Bands 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.

Name	MCU	Frequency	Flash	RAM
<i>Freescale Kinetis FRDM-KL82Z</i>	MKL82Z128VLK7	96MHz	128KB	96KB
<i>Freescale Kinetis FRDM-KW24D512</i>	MKW24D512	50MHz	512KB	64KB
<i>Hexiwear</i>	MK64FN1M0VDC12	120MHz	1MB	256KB

Stable and upstream versions

You can switch between stable releases of Freescale Kinetis development platform and the latest upstream version using *platform* option in “*platformio.ini*” (*Project Configuration File*) as described below.

Stable

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

; 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

Name	Description
framework-mbed	mbed Framework
tool-jlink	SEGGER J-Link Software and Documentation Pack
tool-pyocd	Open source python library for programming and debugging ARM Cortex-M microcontrollers using CMSIS-DAP
toolchain-gccarmnoneabi	gcc-arm-embedded

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

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.

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

Name	Debug	MCU	Frequency	Flash	RAM
<i>Ethernet IoT Starter Kit</i>	On-board	MK64FN1M0VLL12	120MHz	1MB	256KB
<i>Freescale Kinetis FRDM-K20D50M</i>	On-board	MK20DX128VLH5	48MHz	128KB	16KB
<i>Freescale Kinetis FRDM-K22F</i>	On-board	MK22FN512VLH12	120MHz	512KB	128KB
<i>Freescale Kinetis FRDM-K64F</i>	On-board	MK64FN1M0VLL12	120MHz	1MB	256KB
<i>Freescale Kinetis FRDM-K66F</i>	On-board	MK66FN2M0VMD18	180MHz	2MB	256KB
<i>Freescale Kinetis FRDM-K82F</i>	On-board	MK82FN256VLL15	150MHz	256KB	256KB
<i>Freescale Kinetis FRDM-KL05Z</i>	On-board	MKL05Z32VFM4	48MHz	32KB	4KB
<i>Freescale Kinetis FRDM-KL25Z</i>	On-board	MKL25Z128VLK4	48MHz	128KB	16KB
<i>Freescale Kinetis FRDM-KL27Z</i>	On-board	MKL27Z64VLH4	48MHz	64KB	16KB
<i>Freescale Kinetis FRDM-KL43Z</i>	On-board	MKL43Z256VLH4	48MHz	256KB	32KB
<i>Freescale Kinetis FRDM-KL46Z</i>	On-board	MKL46Z256VLL4	48MHz	256KB	32KB
<i>Freescale Kinetis FRDM-KL82Z</i>	External	MKL82Z128VLK7	96MHz	128KB	96KB
<i>Freescale Kinetis FRDM-KW24D512</i>	External	MKW24D512	50MHz	512KB	64KB
<i>Freescale Kinetis FRDM-KW41Z</i>	On-board	MKW41Z512VHT4	48MHz	512KB	128KB

MikroElektronika

Name	Debug	MCU	Frequency	Flash	RAM
<i>Hexiwear</i>	External	MK64FN1M0VDC12	120MHz	1MB	256KB

GigaDevice GD32V

Configuration *platform* = gd32v

The GigaDevice GD32V device is a 32-bit general-purpose microcontroller based on the RISC-V core with an impressive balance of processing power, reduced power consumption and peripheral set.

For more detailed information please visit [vendor site](#).

Contents

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

Examples

Examples are listed from [GigaDevice GD32V development platform repository](#):

- [arduino-blink](#)

- eval-link
- longan-nano-link

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

Banks 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.

Name	MCU	Frequency	Flash	RAM
<i>GD32VF103V-EVAL</i>	GD32VF103VBT6	108MHz	128KB	32KB
<i>Sipeed Longan Nano</i>	GD32VF103CBT6	108MHz	128KB	32KB
<i>Wio Lite RISC-V</i>	GD32VF103CBT6	108MHz	128KB	32KB

Stable and upstream versions

You can switch between stable releases of GigaDevice GD32V development platform and the latest upstream version using *platform* option in “*platformio.ini*” (*Project Configuration File*) as described below.

Stable

```
; Latest stable version
[env:latest_stable]
platform = gd32v
board = ...

; Custom stable version
[env:custom_stable]
```

(continues on next page)

(continued from previous page)

```
platform = gd32v@x.y.z
board = ...
```

Upstream

```
[env:upstream_develop]
platform = https://github.com/sipeed/platform-gd32v.git
board = ...
```

Packages

Name	Description
framework-arduino-gd32v	Arduino Wiring-based Framework (GigaDevice GD32V Core)
framework-gd32vf103-sdk	GigaDevice GD32V SDK
tool-gd32vflash	GD32V FLASH TOOLS
tool-openocd-gd32v	OpenOCD for RISC-V GigaDevice GD32V
toolchain-gd32v	GCC for GigaDevice GD32V

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

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.
GigaDevice GD32V SDK	GigaDevice GD32VF103 Firmware Library (SDK)

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.

SeeedStudio

Name	Debug	MCU	Frequency	Flash	RAM
Wio Lite RISC-V	External	GD32VF103CBT6	108MHz	128KB	32KB

Sipeed

Name	Debug	MCU	Frequency	Flash	RAM
GD32VF103V-EVAL	External	GD32VF103VBT6	108MHz	128KB	32KB
Sipeed Longan Nano	External	GD32VF103CBT6	108MHz	128KB	32KB

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](#).

Contents

- [Examples](#)
- [Debugging](#)
- [Stable and upstream versions](#)
- [Packages](#)
- [Frameworks](#)
- [Boards](#)

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

Banks listed below have on-board debug probe and **ARE READY** for debugging! You do not need to use/buy external debug probe.

Name	MCU	Frequency	Flash	RAM
<i>XMC1100 Boot Kit</i>	XMC1100	32MHz	64KB	16KB
<i>XMC1100 H-Bridge 2Go</i>	XMC1100	32MHz	64KB	16KB
<i>XMC1100 XMC2Go</i>	XMC1100	32MHz	64KB	16KB
<i>XMC1300 Boot Kit</i>	XMC1300	32MHz	64KB	16KB
<i>XMC1300 Sense2GoL</i>	XMC1300	32MHz	32KB	16KB
<i>XMC1400 Boot Kit</i>	XMC1400	48MHz	1.95MB	16KB
<i>XMC4200 Distance2Go</i>	XMC4200	80MHz	256KB	40KB
<i>XMC4700 Relax Kit</i>	XMC4700	144MHz	2.00MB	1.95MB

Stable and upstream versions

You can switch between stable releases of Infineon XMC development platform and the latest upstream version using *platform* option in “*platformio.ini*” (*Project Configuration File*) as described below.

Stable

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

(continues on next page)

(continued from previous page)

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

Upstream

```
[env:upstream_develop]
platform = https://github.com/Infineon/platformio-infineonxmc.git
board = ...
```

Packages

Name	Description
framework-arduinoxmc	Arduino Wiring-based Framework (Infineon XMC Core)
tool-jlink	SEGGER J-Link Software and Documentation Pack
toolchain-gccarmnoneeabi	gcc-arm-embedded

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

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.

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

Name	Debug	MCU	Frequency	Flash	RAM
XMC1100 Boot Kit	On-board	XMC1100	32MHz	64KB	16KB
XMC1100 H-Bridge 2Go	On-board	XMC1100	32MHz	64KB	16KB
XMC1100 XMC2Go	On-board	XMC1100	32MHz	64KB	16KB
XMC1300 Boot Kit	On-board	XMC1300	32MHz	64KB	16KB
XMC1300 Sense2GoL	On-board	XMC1300	32MHz	32KB	16KB
XMC1400 Boot Kit	On-board	XMC1400	48MHz	1.95MB	16KB
XMC4200 Distance2Go	On-board	XMC4200	80MHz	256KB	40KB
XMC4700 Relax Kit	On-board	XMC4700	144MHz	2.00MB	1.95MB

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.int`” (*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

Name	Description
framework-arduinointel	Arduino Wiring-based Framework (Intel ARC Core)
tool-arduino101load	Genuino101 uploader
toolchain-intelarc32	GCC for Intel ARC

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

Name	Description
Arduinointel	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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

Name	Debug	MCU	Frequency	Flash	RAM
<i>Arduino/Genuino 101</i>	No	ARCV2EM	32MHz	152KB	80KB

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

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

; Custom stable version
```

(continues on next page)

(continued from previous page)

```
[env:custom_stable]
platform = intel_mcs51@x.y.z
board = ...
```

Upstream

```
[env:upstream_develop]
platform = https://github.com/platformio/platform-intel_mcs51.git
board = ...
```

Packages

Name	Description
tool-stcgal	Open Source STC MCU ISP flash tool
toolchain-sdcc	Small Device C Compiler

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

Name	Debug	MCU	Frequency	Flash	RAM
Generic N79E8432	No	N79E8432	22MHz	4KB	512B
Generic N79E844	No	N79E844	22MHz	8KB	512B
Generic N79E845	No	N79E845	22MHz	16KB	512B
Generic N79E854	No	N79E854	22MHz	8KB	512B
Generic N79E855	No	N79E855	22MHz	16KB	512B

STC

Name	Debug	MCU	Frequency	Flash	RAM
<i>Generic STC15F204EA</i>	No	STC15F204EA	11MHz	4KB	256B
<i>Generic STC15F2K60S2</i>	No	STC15F2K60S2	6MHz	60KB	2KB
<i>Generic STC15W204S</i>	No	STC15W204S	11MHz	4KB	256B
<i>Generic STC15W404AS</i>	No	STC15W404AS	11MHz	4KB	512B
<i>Generic STC15W408AS</i>	No	STC15W408AS	11MHz	8KB	512B
<i>Generic STC89C52RC</i>	No	STC89C52RC	11MHz	8KB	512B

Kendryte K210

Configuration `platform = kendryte210`

Kendryte K210 is an AI capable RISCV64 dual core SoC.

For more detailed information please visit [vendor site](#).

Contents

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

Examples

Examples are listed from Kendryte K210 development platform repository:

- arduino-blink
- kendryte-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

Banks 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.

Name	MCU	Frequency	Flash	RAM
<i>Sipeed MAIX BiT</i>	K210	400MHz	16MB	6MB
<i>Sipeed MAIX BiT with Mic</i>	K210	400MHz	16MB	6MB
<i>Sipeed MAIX GO</i>	K210	400MHz	16MB	6MB
<i>Sipeed MAIX ONE DOCK</i>	K210	400MHz	16MB	6MB
<i>Sipeed MAIXDUINO</i>	K210	400MHz	16MB	6MB

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

Name	Description
framework-kendryte-freertos-sdk	Kendryte SDK with FreeRTOS support
framework-kendryte-standalone-sdk	Kendryte standalone SDK without OS support
framework-maixduino	Arduino Wiring-based Framework (K210 Core)
tool-kflash-kendryte210	kflash, A Python-based Kendryte K210 UART ISP Utility
tool-openocd-kendryte	OpenOCD for RISC-V Kendryte
toolchain-kendryte210	RISC-V GCC toolchain for Kendryte 210

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

Name	Description
<i>Arduino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>Kendryte Standalone SDK</i>	Kendryte Standalone SDK without OS support
<i>Kendryte FreeRTOS SDK</i>	Kendryte SDK with FreeRTOS support

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

Name	Debug	MCU	Frequency	Flash	RAM
<i>Sipeed MAIX BiT</i>	External	K210	400MHz	16MB	6MB
<i>Sipeed MAIX BiT with Mic</i>	External	K210	400MHz	16MB	6MB
<i>Sipeed MAIX GO</i>	External	K210	400MHz	16MB	6MB
<i>Sipeed MAIX ONE DOCK</i>	External	K210	400MHz	16MB	6MB
<i>Sipeed MAIXDUINO</i>	External	K210	400MHz	16MB	6MB

Lattice iCE40

Configuration `platform = lattice_ice40`

The iCE40 family of ultra-low power, non-volatile FPGAs has five devices with densities ranging from 384 to 7680 Look-Up Tables (LUTs). In addition to LUT-based, low-cost programmable logic, these devices feature Embedded Block RAM (EBR), Non-volatile Configuration Memory (NVCM) and Phase Locked Loops (PLLs). These features allow the devices to be used in low-cost, high-volume consumer and system applications.

For more detailed information please visit [vendor site](#).

Contents

- *Examples*
- *Stable and upstream versions*
- *Packages*
- *Boards*

Examples

Examples are listed from [Lattice iCE40 development platform repository](#):

- `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.

Stable

```
; Latest stable version
[env:latest_stable]
platform = lattice_ice40
board = ...
```

(continues on next page)

(continued from previous 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

Name	Description
toolchain-icestorm	Tools for analyzing and creating bitstream files for FPGA IceStorm
toolchain-verilog	Verilog simulation and synthesis tool

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

Name	Debug	MCU	Frequency	Flash	RAM
IceZUM Alhambra FPGA	No	ICE40-HX1K-TQ144	12MHz	32KB	32KB

Lattice

Name	Debug	MCU	Frequency	Flash	RAM
Lattice iCEstick FPGA Evaluation Kit	No	ICE40-HX1K-TQ144	12MHz	32KB	32KB

Maxim 32

Configuration `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

Banks listed below have on-board debug probe and **ARE READY** for debugging! You do not need to use/buy external debug probe.

Name	MCU	Frequency	Flash	RAM
<i>Maxim ARM mbed Enabled Development Platform for MAX32600</i>	MAX32600	24MHz	256KB	32KB

External Debug Tools

Banks 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.

Name	MCU	Frequency	Flash	RAM
<i>MAX32620FTHR</i>	MAX32620FTHR	96MHz	2MB	256KB
<i>Maxim Health Sensor Platform</i>	MAX32620	96MHz	2MB	256KB
<i>Maxim Wireless Sensor Node Demonstrator</i>	MAX32610	24MHz	256KB	32KB

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

Name	Description
framework-mbed	mbed Framework
tool-jlink	SEGGER J-Link Software and Documentation Pack
tool-pyocd	Open source python library for programming and debugging ARM Cortex-M microcontrollers using CMSIS-DAP
toolchain-gccarmnoneabi	gcc-arm-embedded

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

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.

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

Name	Debug	MCU	Frequency	Flash	RAM
<i>MAX32620FTHR</i>	External	MAX32620FTHR	96MHz	2MB	256KB
<i>MAX32625MBED</i>	No	MAX32625	96MHz	512KB	160KB
<i>MAX32625NEXPAQ</i>	No	MAX32625	96MHz	512KB	160KB
<i>MAX32625PICO</i>	No	MAX32625	96MHz	512KB	160KB
<i>Maxim ARM mbed Enabled Development Platform for MAX32600</i>	On-board	MAX32600	24MHz	256KB	32KB
<i>Maxim Health Sensor Platform</i>	External	MAX32620	96MHz	2MB	256KB
<i>Maxim MAX32630FTHR Application Platform</i>	No	MAX32630	96MHz	2MB	512KB
<i>Maxim Wireless Sensor Node Demonstrator</i>	External	MAX32610	24MHz	256KB	32KB

Sigma Delta Technologies

Name	Debug	MCU	Frequency	Flash	RAM
<i>SDT32620B</i>	No	MAX32620IWG	96MHz	2MB	256KB
<i>SDT32625B</i>	No	MAX32625ITK	96MHz	512KB	160KB

Microchip PIC32

Configuration *platform* = microchippic32

Microchip's 32-bit portfolio with the MIPS microAptiv or M4K core offer high performance microcontrollers, and all the tools needed to develop your embedded projects. PIC32 MCUs gives your application the processing power, memory and peripherals your design needs!

For more detailed information please visit [vendor site](#).

Contents

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

Examples

Examples are listed from Microchip PIC32 development platform repository:

- [arduino-blink](#)

- arduino-internal-libs

Stable and upstream versions

You can switch between stable releases of Microchip PIC32 development platform and the latest upstream version using `platform` option in “*platformio.ini*” (*Project Configuration File*) as described below.

Stable

```
; Latest stable version
[env:latest_stable]
platform = microchippic32
board = ...

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

Upstream

```
[env:upstream_develop]
platform = https://github.com/platformio/platform-microchippic32.git
board = ...
```

Packages

Name	Description
framework-arduinomicrochippic32	Arduino Wiring-based Framework (PIC32 Core)
tool-pic32prog	pic32prog
toolchain-microchippic32	GCC for Microchip PIC32

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

Name	Description
Arduin	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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

Name	Debug	MCU	Frequency	Flash	RAM
4DSystems PICadillo 35T	No	32MX795F512L	80MHz	508KB	128KB

BOXTEC

Name	Debug	MCU	Frequency	Flash	RAM
HelvePic32	No	32MX250F128B	48MHz	120KB	32KB
HelvePic32	No	32MX250F128B	48MHz	120KB	32KB
HelvePic32	No	32MX250F128D	48MHz	120KB	32KB
HelvePic32 MX270	No	32MX270F256B	48MHz	244KB	62KB
HelvePic32 Robot	No	32MX270F256D	48MHz	244KB	62KB
HelvePic32 SMD MX270	No	32MX270F256D	48MHz	244KB	62KB

ChipKIT

Name	Debug	MCU	Frequency	Flash	RAM
RGB Station	No	32MX270F256D	48MHz	240KB	62KB

Digilent

Name	Debug	MCU	Frequency	Flash	RAM
<i>Digilent Cerebot 32MX4</i>	No	32MX460F512L	80MHz	508KB	32KB
<i>Digilent Cerebot 32MX7</i>	No	32MX795F512L	80MHz	508KB	128KB
<i>Digilent OpenScope</i>	No	32MZ2048EFG124	200MHz	1.98MB	512KB
<i>Digilent chipKIT Cmod</i>	No	32MX150F128D	40MHz	124KB	32KB
<i>Digilent chipKIT DP32</i>	No	32MX250F128B	40MHz	120KB	32KB
<i>Digilent chipKIT MAX32</i>	No	32MX795F512L	80MHz	508KB	128KB
<i>Digilent chipKIT MX3</i>	No	32MX320F128H	80MHz	124KB	16KB
<i>Digilent chipKIT Pro MX4</i>	No	32MX460F512L	80MHz	508KB	32KB
<i>Digilent chipKIT Pro MX7</i>	No	32MX795F512L	80MHz	508KB	128KB
<i>Digilent chipKIT UNO32</i>	No	32MX320F128H	80MHz	124KB	16KB
<i>Digilent chipKIT WF32</i>	No	32MX695F512L	80MHz	508KB	128KB
<i>Digilent chipKIT WiFire</i>	No	32MZ2048ECG100	200MHz	1.98MB	512KB
<i>Digilent chipKIT uC32</i>	No	32MX340F512H	80MHz	508KB	32KB
<i>chipKIT WiFire rev. C</i>	No	32MZ2048EFG100	200MHz	1.98MB	512KB

Fubarino

Name	Debug	MCU	Frequency	Flash	RAM
<i>Fubarino Mini</i>	No	32MX250F128D	48MHz	120KB	32KB
<i>Fubarino SD (1.5)</i>	No	32MX795F512H	80MHz	508KB	128KB
<i>Mini 2.0</i>	No	32MX270F256D	48MHz	240KB	62KB

Makerology

Name	Debug	MCU	Frequency	Flash	RAM
<i>DataStation Mini</i>	No	32MX150F128C	40MHz	120KB	32KB

MikroElektronika

Name	Debug	MCU	Frequency	Flash	RAM
<i>MikroElektronika Clicker 2</i>	No	32MX460F512L	80MHz	508KB	32KB
<i>MikroElektronika Flip N Click MZ</i>	No	32MZ2048EFH100	252MHz	1.98MB	512KB

Olimex

Name	Debug	MCU	Frequency	Flash	RAM
<i>Olimex PIC32-PINGUINO</i>	No	32MX440F256H	80MHz	252KB	32KB

OpenBCI

Name	Debug	MCU	Frequency	Flash	RAM
<i>OpenBCI 32bit</i>	No	32MX250F128B	40MHz	120KB	32KB

PONTECH

Name	Debug	MCU	Frequency	Flash	RAM
<i>PONTECH UAV100</i>	No	32MX440F512H	80MHz	508KB	32KB

Pontech

Name	Debug	MCU	Frequency	Flash	RAM
<i>Pontech NoFire</i>	No	32MZ2048EFG100	200MHz	1.98MB	512KB
<i>Pontech Quick240</i>	No	32MX795F512L	80MHz	508KB	128KB

SeeedStudio

Name	Debug	MCU	Frequency	Flash	RAM
<i>SeeedStudio CUI32stem</i>	No	32MX795F512H	80MHz	508KB	128KB

SparkFun

Name	Debug	MCU	Frequency	Flash	RAM
<i>Pic32 CUI32-Development Stick</i>	No	32MX440F512H	80MHz	508KB	32KB

UBW32

Name	Debug	MCU	Frequency	Flash	RAM
<i>UBW32 MX460</i>	No	32MX460F512L	80MHz	508KB	32KB
<i>UBW32 MX795</i>	No	32MX795F512L	80MHz	508KB	128KB

chipKIT

Name	Debug	MCU	Frequency	Flash	RAM
<i>chipKIT Lenny</i>	No	32MX270F256D	40MHz	120KB	32KB

element14

Name	Debug	MCU	Frequency	Flash	RAM
<i>Element14 chipKIT Pi</i>	No	32MX250F128B	40MHz	120KB	32KB

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-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

Bands listed below have on-board debug probe and **ARE READY** for debugging! You do not need to use/buy external debug probe.

Name	MCU	Frequency	Flash	RAM
<i>BBC micro:bit</i>	NRF51822	16MHz	256KB	16KB
<i>Calliope mini</i>	NRF51822	16MHz	256KB	16KB
<i>Delta DFCM-NNN40</i>	NRF51822	32MHz	256KB	32KB
<i>Delta DFCM-NNN50</i>	NRF51822	32MHz	256KB	16KB
<i>JKSoft Wallbot BLE</i>	NRF51822	16MHz	128KB	16KB
<i>Nordic Beacon Kit (PCA20006)</i>	NRF51822	32MHz	256KB	32KB
<i>Nordic nRF51 Dongle (PCA10031)</i>	NRF51822	32MHz	256KB	32KB
<i>Nordic nRF51822-mKIT</i>	NRF51822	16MHz	128KB	16KB
<i>Nordic nRF51X22 Development Kit(PCA1000X)</i>	NRF51822	32MHz	256KB	32KB
<i>RedBearLab BLE Nano 1.5</i>	NRF51822	16MHz	256KB	32KB
<i>RedBearLab nRF51822</i>	NRF51822	16MHz	256KB	16KB
<i>Seeed Arch BLE</i>	NRF51822	16MHz	128KB	16KB
<i>Seeed Arch Link</i>	NRF51822	16MHz	256KB	16KB
<i>Seeed Tiny BLE</i>	NRF51822	16MHz	256KB	16KB
<i>Switch Science mbed HRM1017</i>	NRF51822	16MHz	256KB	16KB
<i>Switch Science mbed TY51822r3</i>	NRF51822	32MHz	256KB	32KB
<i>VNG VBLUNO51</i>	NRF51822	16MHz	128KB	32KB
<i>y5 nRF51822 mbug</i>	NRF51822	16MHz	256KB	16KB

External Debug Tools

Bands 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.

Name	MCU	Frequency	Flash	RAM
<i>BluzDK</i>	NRF51822	32MHz	256KB	32KB
<i>OSHChip</i>	NRF51822	32MHz	256KB	32KB
<i>Sino:Bit</i>	NRF51822	32MHz	256KB	32KB
<i>Waveshare BLE400</i>	NRF51822	32MHz	256KB	32KB
<i>ng-beacon</i>	NRF51822	16MHz	256KB	32KB

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

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

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

Upstream

```
[env:upstream_develop]
platform = https://github.com/platformio/platform-nordicnrf51.git
board = ...
```

Packages

Name	Description
framework-arduinonordicnrf5	Arduino Wiring-based Framework (Nordic NRF5 Core)
framework-mbed	mbed Framework
tool-jlink	SEGGER J-Link Software and Documentation Pack
tool-nrfjprog	nRF5x command line tool
tool-openocd	OpenOCD
tool-srecat	Merging tool
toolchain-gccarmnoneabi	gcc-arm-embedded

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

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>mbed</i>	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.

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

Name	Debug	MCU	Frequency	Flash	RAM
<i>BBC micro:bit</i>	On-board	NRF51822	16MHz	256KB	16KB

BluzDK

Name	Debug	MCU	Frequency	Flash	RAM
<i>BluzDK</i>	External	NRF51822	32MHz	256KB	32KB

Calliope

Name	Debug	MCU	Frequency	Flash	RAM
<i>Calliope mini</i>	On-board	NRF51822	16MHz	256KB	16KB

Delta

Name	Debug	MCU	Frequency	Flash	RAM
<i>Delta DFCM-NNN40</i>	On-board	NRF51822	32MHz	256KB	32KB
<i>Delta DFCM-NNN50</i>	On-board	NRF51822	32MHz	256KB	16KB

JKSoft

Name	Debug	MCU	Frequency	Flash	RAM
<i>JKSoft Wallbot BLE</i>	On-board	NRF51822	16MHz	128KB	16KB

Nordic

Name	Debug	MCU	Frequency	Flash	RAM
<i>Nordic Beacon Kit (PCA20006)</i>	On-board	NRF51822	32MHz	256KB	32KB
<i>Nordic nRF51 Dongle (PCA10031)</i>	On-board	NRF51822	32MHz	256KB	32KB
<i>Nordic nRF51822-mKIT</i>	On-board	NRF51822	16MHz	128KB	16KB
<i>Nordic nRF51X22 Development Kit(PCA1000X)</i>	On-board	NRF51822	32MHz	256KB	32KB

OSHChip

Name	Debug	MCU	Frequency	Flash	RAM
<i>OSHChip</i>	External	NRF51822	32MHz	256KB	32KB

RedBearLab

Name	Debug	MCU	Frequency	Flash	RAM
<i>RedBearLab BLE Nano 1.5</i>	On-board	NRF51822	16MHz	256KB	32KB
<i>RedBearLab nRF51822</i>	On-board	NRF51822	16MHz	256KB	16KB

SeeedStudio

Name	Debug	MCU	Frequency	Flash	RAM
<i>Seeed Arch BLE</i>	On-board	NRF51822	16MHz	128KB	16KB
<i>Seeed Arch Link</i>	On-board	NRF51822	16MHz	256KB	16KB
<i>Seeed Tiny BLE</i>	On-board	NRF51822	16MHz	256KB	16KB

Switch Science

Name	Debug	MCU	Frequency	Flash	RAM
<i>Switch Science mbed HRM1017</i>	On-board	NRF51822	16MHz	256KB	16KB
<i>Switch Science mbed TY51822r3</i>	On-board	NRF51822	32MHz	256KB	32KB

VNG

Name	Debug	MCU	Frequency	Flash	RAM
<i>VNG VBLUNO51</i>	On-board	NRF51822	16MHz	128KB	32KB

Waveshare

Name	Debug	MCU	Frequency	Flash	RAM
<i>Waveshare BLE400</i>	External	NRF51822	32MHz	256KB	32KB

ng-beacon

Name	Debug	MCU	Frequency	Flash	RAM
<i>ng-beacon</i>	External	NRF51822	16MHz	256KB	32KB

sino:bit

Name	Debug	MCU	Frequency	Flash	RAM
<i>Sino:Bit</i>	External	NRF51822	32MHz	256KB	32KB

y5 design

Name	Debug	MCU	Frequency	Flash	RAM
<i>y5 nRF51822 mbug</i>	On-board	NRF51822	16MHz	256KB	16KB

Nordic nRF52

Configuration *platform* = nordicnrf52

The nRF52 Series are built for speed to carry out increasingly complex tasks in the shortest possible time and return to sleep, conserving precious battery power. They have a Cortex-M4F processor and are the most capable Bluetooth Smart SoCs on the market.

For more detailed information please visit [vendor site](#).

Contents

- [*Tutorials*](#)
- [*Examples*](#)
- [*Debugging*](#)
- [*Stable and upstream versions*](#)
- [*Packages*](#)
- [*Frameworks*](#)
- [*Boards*](#)

Tutorials

- [Arduino and Nordic nRF52-DK: debugging and unit testing](#)

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

Bands listed below have on-board debug probe and **ARE READY** for debugging! You do not need to use/buy external debug probe.

Name	MCU	Frequency	Flash	RAM
<i>Adafruit Bluefruit nRF52832 Feather</i>	NRF52832	64MHz	512KB	64KB
<i>Adafruit Feather nRF52840 Express</i>	NRF52840	64MHz	796KB	243KB
<i>Delta DFBM-NQ620</i>	NRF52832	64MHz	512KB	64KB
<i>Metro nRF52840 Express</i>	NRF52840	64MHz	796KB	243KB
<i>Nordic nRF52-DK</i>	NRF52832	64MHz	512KB	64KB
<i>Nordic nRF52840-DK</i>	NRF52840	64MHz	1MB	256KB
<i>Nordic nRF52840-DK (Adafruit BSP)</i>	NRF52840	64MHz	796KB	243KB
<i>RedBearLab BLE Nano 2</i>	NRF52832	64MHz	512KB	64KB
<i>RedBearLab Blend 2</i>	NRF52832	64MHz	512KB	64KB
<i>u-blox EVK-NINA-B1</i>	NRF52832	64MHz	512KB	64KB

External Debug Tools

Bands 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.

Name	MCU	Frequency	Flash	RAM
<i>Bluey nRF52832 IoT</i>	NRF52832	64MHz	512KB	64KB
<i>Circuit Playground Bluefruit</i>	NRF52840	64MHz	796KB	243KB
<i>SDT52832B</i>	NRF52832	64MHz	512KB	64KB
<i>Taida Century nRF52 mini board</i>	NRF52832	64MHz	512KB	64KB
<i>Xenon</i>	NRF52840	64MHz	796KB	243KB
<i>hackaBLE</i>	NRF52832	64MHz	512KB	64KB

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

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

Name	Description
framework-arduinoadafruitnrf52	Arduino Wiring-based Framework (Nordic nRF52 BLE SoC)
framework-arduinonordicnrf5	Arduino Wiring-based Framework (Nordic NRF5 Core)
framework-mbed	mbed Framework
tool-jlink	SEGGER J-Link Software and Documentation Pack
tool-nrfjprog	nRF5x command line tool
tool-openocd	OpenOCD
tool-srecat	Merging tool
toolchain-gccarmnoneabi	gcc-arm-embedded

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

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>mbed</i>	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.

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

Name	Debug	MCU	Frequency	Flash	RAM
<i>Adafruit Bluefruit nRF52832 Feather</i>	On-board	NRF52832	64MHz	512KB	64KB
<i>Adafruit Feather nRF52840 Express</i>	On-board	NRF52840	64MHz	796KB	243KB
<i>Circuit Playground Bluefruit</i>	External	NRF52840	64MHz	796KB	243KB
<i>Metro nRF52840 Express</i>	On-board	NRF52840	64MHz	796KB	243KB

Delta

Name	Debug	MCU	Frequency	Flash	RAM
<i>Delta DFIM-NQ620</i>	On-board	NRF52832	64MHz	512KB	64KB

Electronut Labs

Name	Debug	MCU	Frequency	Flash	RAM
<i>Bluey nRF52832 IoT</i>	External	NRF52832	64MHz	512KB	64KB
<i>hackable</i>	External	NRF52832	64MHz	512KB	64KB

Nordic

Name	Debug	MCU	Frequency	Flash	RAM
<i>Nordic nRF52-DK</i>	On-board	NRF52832	64MHz	512KB	64KB
<i>Nordic nRF52840-DK</i>	On-board	NRF52840	64MHz	1MB	256KB
<i>Nordic nRF52840-DK (Adafruit BSP)</i>	On-board	NRF52840	64MHz	796KB	243KB

Particle

Name	Debug	MCU	Frequency	Flash	RAM
<i>Xenon</i>	External	NRF52840	64MHz	796KB	243KB

RedBearLab

Name	Debug	MCU	Frequency	Flash	RAM
<i>RedBearLab BLE Nano 2</i>	On-board	NRF52832	64MHz	512KB	64KB
<i>RedBearLab Blend 2</i>	On-board	NRF52832	64MHz	512KB	64KB

Sigma Delta Technologies

Name	Debug	MCU	Frequency	Flash	RAM
<i>SDT52832B</i>	External	NRF52832	64MHz	512KB	64KB

Taida Century

Name	Debug	MCU	Frequency	Flash	RAM
<i>Taida Century nRF52 mini board</i>	External	NRF52832	64MHz	512KB	64KB

u-blox

Name	Debug	MCU	Frequency	Flash	RAM
<i>u-blox EVK-NINA-B1</i>	On-board	NRF52832	64MHz	512KB	64KB

NXP LPC

Configuration *platform* = nxplpc

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

For more detailed information please visit [vendor site](#).

Contents

- [*Examples*](#)
- [*Debugging*](#)
- [*Stable and upstream versions*](#)
- [*Packages*](#)
- [*Frameworks*](#)
- [*Boards*](#)

Examples

Examples are listed from NXP LPC development platform repository:

- [mbed-blink](#)
- [mbed-custom-target](#)
- [mbed-dsp](#)
- [mbed-events](#)
- [mbed-rtos](#)
- [mbed-rtos-ethernet](#)
- [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.

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

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.

Name	MCU	Frequency	Flash	RAM
<i>CQ Publishing TG-LPC11U35-501</i>	LPC11U35	48MHz	64KB	10KB
<i>DipCortex M3</i>	LPC1347	72MHz	64KB	12KB
<i>EA LPC11U35 QuickStart Board</i>	LPC11U35	48MHz	64KB	10KB
<i>NGX Technologies BlueBoard-LPC11U24</i>	LPC11U24	48MHz	32KB	8KB
<i>NXP LPC11C24</i>	LPC11C24	48MHz	32KB	8KB
<i>NXP LPC11U34</i>	LPC11U34	48MHz	40KB	8KB
<i>NXP LPC11U37</i>	LPC11U37	48MHz	128KB	10KB
<i>NXP LPCXpresso1549</i>	LPC1549	72MHz	256KB	36KB
<i>Solder Splash Labs DipCortex M0</i>	LPC11U24	50MHz	32KB	8KB
<i>y5 LPC11U35 mbug</i>	LPC11U35	48MHz	64KB	10KB

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

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

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

Upstream

```
[env:upstream_develop]
platform = https://github.com/platformio/platform-nxplpc.git
board = ...
```

Packages

Name	Description
framework-mbed	mbed Framework
tool-jlink	SEGGER J-Link Software and Documentation Pack
tool-openocd	OpenOCD
tool-pyocd	Open source python library for programming and debugging ARM Cortex-M microcontrollers using CMSIS-DAP
toolchain-gccarmnoneabi	gcc-arm-embedded

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

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.

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

Name	Debug	MCU	Frequency	Flash	RAM
MicroNFCBoard	No	LPC11U34	48MHz	48KB	10KB

CQ Publishing

Name	Debug	MCU	Frequency	Flash	RAM
CQ Publishing TG-LPC11U35-501	External	LPC11U35	48MHz	64KB	10KB

Elektor Labs

Name	Debug	MCU	Frequency	Flash	RAM
CoCo-ri-Co!	On-board	LPC812	30MHz	16KB	4KB

Embedded Artists

Name	Debug	MCU	Frequency	Flash	RAM
EA LPC11U35 QuickStart Board	External	LPC11U35	48MHz	64KB	10KB
Embedded Artists LPC4088 Display Module	On-board	LPC4088	120MHz	512KB	96KB
Embedded Artists LPC4088 QuickStart Board	On-board	LPC4088	120MHz	512KB	96KB

GHI Electronics

Name	Debug	MCU	Frequency	Flash	RAM
mBuino	No	LPC11U24	50MHz	32KB	10KB

Micromint

Name	Debug	MCU	Frequency	Flash	RAM
Bambino-210E	On-board	LPC4330	204MHz	8MB	264KB

NGX Technologies

Name	Debug	MCU	Frequency	Flash	RAM
NGX Technologies BlueBoard-LPC11U24	External	LPC11U24	48MHz	32KB	8KB

NXP

Name	Debug	MCU	Frequency	Flash	RAM
ARM mbed LPC11U24 (+CAN)	On-board	LPC11U24	48MHz	32KB	8KB
LPCXpresso11U68	On-board	LPC11U68	50MHz	256KB	36KB
LPCXpresso824-MAX	On-board	LPC824	30MHz	32KB	8KB
NXP LPC11C24	External	LPC11C24	48MHz	32KB	8KB
NXP LPC11U34	External	LPC11U34	48MHz	40KB	8KB
NXP LPC11U37	External	LPC11U37	48MHz	128KB	10KB
NXP LPC800-MAX	On-board	LPC812	30MHz	16KB	4KB
NXP LPCXpresso1549	External	LPC1549	72MHz	256KB	36KB
NXP LPCXpresso54114	On-board	LPC54114J256BD64	100MHz	256KB	192KB
NXP LPCXpresso54608	On-board	LPC54608ET512	180MHz	512KB	200KB
NXP mbed LPC11U24	On-board	LPC11U24	48MHz	32KB	8KB
NXP mbed LPC1768	On-board	LPC1768	96MHz	512KB	64KB

Outrageous Circuits

Name	Debug	MCU	Frequency	Flash	RAM
Outrageous Circuits mBuino	No	LPC11U24	48MHz	32KB	8KB

SeeedStudio

Name	Debug	MCU	Frequency	Flash	RAM
<i>Seeed Arch GPRS V2</i>	No	LPC11U37	48MHz	128KB	10KB
<i>Seeed Arch Pro</i>	On-board	LPC1768	96MHz	512KB	64KB
<i>Seeed Xadow M0</i>	No	LPC11U35	48MHz	64KB	10KB

Smeshlink

Name	Debug	MCU	Frequency	Flash	RAM
<i>Smeshlink xbed LPC1768</i>	No	LPC1768	96MHz	512KB	32KB

Solder Splash Labs

Name	Debug	MCU	Frequency	Flash	RAM
<i>DipCortex M3</i>	External	LPC1347	72MHz	64KB	12KB
<i>Solder Splash Labs DipCortex M0</i>	External	LPC11U24	50MHz	32KB	8KB

Switch Science

Name	Debug	MCU	Frequency	Flash	RAM
<i>Switch Science mbed LPC1114FN28</i>	On-board	LPC1114FN28	48MHz	32KB	4KB
<i>Switch Science mbed LPC824</i>	On-board	LPC824	30MHz	32KB	8KB

u-blox

Name	Debug	MCU	Frequency	Flash	RAM
<i>u-blox C027</i>	On-board	LPC1768	96MHz	512KB	64KB

y5 design

Name	Debug	MCU	Frequency	Flash	RAM
<i>y5 LPC11U35 mbug</i>	External	LPC11U35	48MHz	64KB	10KB

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

```
[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

```
[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

```
[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

```
[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` 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` 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-cpp-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*
 - *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

Banks listed below have on-board debug probe and **ARE READY** for debugging! You do not need to use/buy external debug probe.

Name	MCU	Frequency	Flash	RAM
GAPuino GAP8	GAP8	250MHz	64MB	8MB

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

```
; 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

Name	Description
framework-gap_sdk	The GAP8 SDK allows you to compile and execute applications on the GAP8 IoT Application Processor.
tool-pulp_tools	Top project for building PULP development tools
toolchain-riscv-pulp	RISC-V GCC toolchain for PULP platform

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

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.
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.

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

Name	Debug	MCU	Frequency	Flash	RAM
GAPuino GAP8	On-board	GAP8	250MHz	64MB	8MB

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*
 - *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

Bands listed below have on-board debug probe and **ARE READY** for debugging! You do not need to use/buy external debug probe.

Name	MCU	Frequency	Flash	RAM
Samsung ARTIK053	S5JT200	320MHz	8MB	1.25MB

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

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

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

Upstream

```
[env:upstream_develop]
platform = https://github.com/platformio/platform-samsung_artik.git
board = ...
```

Packages

Name	Description
framework-tizenrt	TizenRT RTOS with library
tool-artik-openocd	OpenOCD for ARTIK
toolchain-gccarmnoneabi	gcc-arm-embedded

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

Name	Description
Tizen RT	Tizen RT is a lightweight RTOS-based platform to support low-end IoT devices

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

Name	Debug	MCU	Frequency	Flash	RAM
Samsung ARTIK053	On-board	S5JT200	320MHz	8MB	1.25MB

Shakti

Configuration `platform = shakti`

Shakti is an open-source initiative by the RISE group at IIT-Madras, which is not only building open source, production grade processors, but also associated components like interconnect fabrics, verification tools, storage controllers, peripheral IPs and SOC tools.

For more detailed information please visit [vendor site](#).

Contents

- [Examples](#)
- [Debugging](#)
- [Stable and upstream versions](#)
- [Packages](#)
- [Frameworks](#)
- [Boards](#)

Examples

Examples are listed from Shakti development platform repository:

- [shakti-sdk_gpio-keypad](#)
- [shakti-sdk_i2c-lm75](#)
- [shakti-sdk_uart-hello](#)

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

Bands listed below have on-board debug probe and **ARE READY** for debugging! You do not need to use/buy external debug probe.

Name	MCU	Frequency	Flash	RAM
<i>Artix-7 35T Arty FPGA Evaluation Kit</i>	E-CLASS	50MHz	0B	128KB
<i>Arty A7-100: Artix-7 FPGA Development Board</i>	C-CLASS	50MHz	0B	128MB

Stable and upstream versions

You can switch between stable releases of Shakti development platform and the latest upstream version using *platform* option in “*platformio.ini*” (*Project Configuration File*) as described below.

Stable

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

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

Upstream

```
[env:upstream_develop]
platform = https://github.com/platformio/platform-shakti.git
board = ...
```

Packages

Name	Description
framework-shakti-sdk	A software development kit for developing applications on Shakti class of processors
tool-jlink	SEGGER J-Link Software and Documentation Pack
tool-openocd-riscv	OpenOCD for RISC-V
tool-qemu-riscv	Open source machine emulator and virtualizer
toolchain-riscv	GNU toolchain for RISC-V, including GCC

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

Name	Description
Shakti SDK	A software development kit for developing applications on Shakti class of processors

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.

Xilinx

Name	Debug	MCU	Frequency	Flash	RAM
Artix-7 35T Arty FPGA Evaluation Kit	On-board	E-CLASS	50MHz	0B	128KB
Arty A7-100: Artix-7 FPGA Development Board	On-board	C-CLASS	50MHz	0B	128MB

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*
 - *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

Bands listed below have on-board debug probe and **ARE READY** for debugging! You do not need to use/buy external debug probe.

Name	MCU	Frequency	Flash	RAM
<i>Aarty FPGA Dev Kit</i>	FE310	450MHz	16MB	256MB
<i>HiFive Unleashed</i>	FU540	1500MHz	32MB	8GB
<i>HiFive1</i>	FE310	320MHz	16MB	16KB
<i>HiFive1 Rev B</i>	FE310	320MHz	16MB	16KB

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

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

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

Upstream

```
[env:upstream_develop]
platform = https://github.com/platformio/platform-sifive.git
board = ...
```

Packages

Name	Description
framework-freedom-e-sdk	Open Source Software for Developing on the SiFive Freedom E Platform
tool-jlink	SEGGER J-Link Software and Documentation Pack
tool-openocd-riscv	OpenOCD for RISC-V
tool-qemu-riscv	Open source machine emulator and virtualizer
toolchain-riscv	GNU toolchain for RISC-V, including GCC

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

Name	Description
Freedom E SDK	Open Source Software for Developing on the SiFive Freedom E Platform

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

Name	Debug	MCU	Frequency	Flash	RAM
<i>HiFive Unleashed</i>	On-board	FU540	1500MHz	32MB	8GB
<i>HiFive1</i>	On-board	FE310	320MHz	16MB	16KB
<i>HiFive1 Rev B</i>	On-board	FE310	320MHz	16MB	16KB

Xilinx

Name	Debug	MCU	Frequency	Flash	RAM
<i>Ary FPGA Dev Kit</i>	On-board	FE310	450MHz	16MB	256MB

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`](#)

- 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

Bands listed below have on-board debug probe and **ARE READY** for debugging! You do not need to use/buy external debug probe.

Name	MCU	Fre-quency	Flash	RAM
<i>EFM32GG-STK3700 Giant Gecko</i>	EFM32GG990F1024	48MHz	1MB	128KB
<i>EFM32LG-STK3600 Leopard Gecko</i>	EFM32LG990F256	48MHz	256KB	32KB
<i>EFM32WG-STK3800 Wonder Gecko</i>	EFM32WG990F256	48MHz	256KB	32KB
<i>EFM32ZG-STK3200 Zero Gecko</i>	EFM32ZG222F32	24MHz	32KB	4KB
<i>SLSTK3400A USB-enabled Happy Gecko</i>	EFM32HG322F64	25MHz	64KB	8KB
<i>SLSTK3401A Pearl Gecko PG1</i>	EFM32PG1B200F256GM48	40MHz	256KB	32KB
<i>Thunderboard Sense 2 Sensor-to-Cloud Advanced IoT</i>	EFR32MG12P432F1024	40MHz	1MB	256KB

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

Name	Description
framework-mbed	mbed Framework
tool-jlink	SEGGER J-Link Software and Documentation Pack
toolchain-gccarmnoneeabi	gcc-arm-embedded

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

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.

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.

Silicon Labs

Name	Debug	MCU	Fre-quency	Flash	RAM
<i>EFM32GG-STK3700 Giant Gecko</i>	On-board	EFM32GG990F1024	48MHz	1MB	128KB
<i>EFM32LG-STK3600 Leopard Gecko</i>	On-board	EFM32LG990F256	48MHz	256KB	32KB
<i>EFM32WG-STK3800 Wonder Gecko</i>	On-board	EFM32WG990F256	48MHz	256KB	32KB
<i>EFM32ZG-STK3200 Zero Gecko</i>	On-board	EFM32ZG222F32	24MHz	32KB	4KB
<i>SLSTK3400A USB-enabled Happy Gecko</i>	On-board	EFM32HG322F64	25MHz	64KB	8KB
<i>SLSTK3401A Pearl Gecko PG1</i>	On-board	EFM32PG1B200F256GM	40MHz	256KB	32KB
<i>Thunderboard Sense 2 Sensor-to-Cloud Advanced IoT</i>	On-board	EFR32MG12P432F1024	40MHz	1MB	256KB

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](#).

Contents

- *Tutorials*
- *Configuration*
- *Examples*
- *Debugging*
- *Stable and upstream versions*
- *Packages*
- *Frameworks*
- *Boards*

Tutorials

- STM32Cube HAL and Nucleo-F401RE: debugging and unit testing

Configuration

Switching between Arduino cores

There are 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 to be added to `build_flags`:

An example of “`platformio.ini`” (*Project Configuration File*) with `maple` core

```
[env:hy_tinystm103tb]
platform = ststm32
framework = arduino
board = hy_tinystm103tb
board_build.core = maple
```

STM32Duino configuration system

STM32Duino core has several options that can be configured using the next configuration flags in `build_flags` section of “`platformio.ini`” (*Project Configuration File*):

Table 5: C/C++ standard library configuration

Name	Description
<code>PIO_FRAMEWORK_ARDUINO_STANDARD_LIB</code>	Disable Newlib Nano library
<code>PIO_FRAMEWORK_ARDUINO_NANOLIB_FLOAT_PRINTF</code>	Newlib Nano + float printf support
<code>PIO_FRAMEWORK_ARDUINO_NANOLIB_FLOAT_SCANF</code>	Newlib Nano + float scanf support

Table 6: USART Configuration

Name	Description
<code>PIO_FRAMEWORK_ARDUINO_SERIAL_WITHOUT_GENERIC</code>	Enabled (no generic Serial)
<code>PIO_FRAMEWORK_ARDUINO_SERIAL_DISABLED</code>	Disabled (no Serial support)

Table 7: USB Configuration

Name	Description
<code>PIO_FRAMEWORK_ARDUINO_ENABLE_CDC</code>	CDC (generic Serial supersede U(S)ART)
<code>PIO_FRAMEWORK_ARDUINO_ENABLE_CDC_WITHOUT_SERIAL</code>	CDC (no generic Serial)
<code>PIO_FRAMEWORK_ARDUINO_ENABLE_HID</code>	HID (keyboard and mouse)

Table 8: USB Speed Configuration

Name	Description
<code>PIO_FRAMEWORK_ARDUINO_USB_HIGHSPEED</code>	High Speed mode
<code>PIO_FRAMEWORK_ARDUINO_USB_HIGHSPEED_FULLMODE</code>	High Speed in Full Speed mode

Example:

```
[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_FULLSCREEN
```

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 9: USB Configuration for STM32F4 boards

Name	Description
ENABLE_USB_SERIAL	USB serial (CDC)
ENABLE_USB_MASS_STORAGE	USB Mass Storage (MSC)

Example:

```
[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*
 - *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

Bands listed below have on-board debug probe and **ARE READY** for debugging! You do not need to use/buy external debug probe.

Name	MCU	Frequency	Flash	RAM
32F723EDISCOVERY	STM32F723IEK6	216MHz	512KB	192KB
3D printer controller	STM32F765VIT6	216MHz	2MB	512KB
3DP001VI Evaluation board for 3D printer	STM32F401VGT6	84MHz	512KB	96KB
96Boards B96B-F446VE	STM32F446VET6	168MHz	512KB	128KB

Continued on next page

Table 10 – continued from previous page

Name	MCU	Frequency	Flash	RAM
<i>L476DMWIK</i>	STM32L476VGT6	80MHz	1MB	128KB
<i>Mbed Connect Cloud</i>	STM32F439ZIY6	168MHz	2MB	256KB
<i>Microsoft Azure IoT Development Kit (MXChip AZ3166)</i>	STM32F412ZGT6	100MHz	1MB	256KB
<i>Nucleo G071RB</i>	STM32G071RBT6	24MHz	2MB	128KB
<i>P-Nucleo WB55RG</i>	STM32WB55RG	64MHz	512KB	192.00KB
<i>RushUp Cloud-JAM</i>	STM32F401RET6	84MHz	512KB	96KB
<i>RushUp Cloud-JAM L4</i>	STM32L476RGT6	80MHz	1MB	128KB
<i>ST 32F3348DISCOVERY</i>	STM32F334C8T6	72MHz	64KB	12KB
<i>ST 32F401CDISCOVERY</i>	STM32F401VCT6	84MHz	256KB	64KB
<i>ST 32F411EDISCOVERY</i>	STM32F411VET6	100MHz	512KB	128KB
<i>ST 32F413HDISCOVERY</i>	STM32F413ZHT6	100MHz	512KB	128KB
<i>ST 32F429IDISCOVERY</i>	STM32F429ZIT6	180MHz	2MB	256KB
<i>ST 32F469IDISCOVERY</i>	STM32F469NIH6	180MHz	1MB	384KB
<i>ST 32F746GDISCOVERY</i>	STM32F746NGH6	216MHz	1MB	320KB
<i>ST 32F769IDISCOVERY</i>	STM32F769NIH6	216MHz	1MB	512KB
<i>ST 32L0538DISCOVERY</i>	STM32L053C8T6	32MHz	64KB	8KB
<i>ST 32L100DISCOVERY</i>	STM32L100RCT6	32MHz	256KB	16KB
<i>ST 32L476GDISCOVERY</i>	STM32L476VGT6	80MHz	1MB	128KB
<i>ST 32L496GDISCOVERY</i>	STM32L496AGI6	80MHz	1MB	320KB
<i>ST DISCO-L072CZ-LRWAN1</i>	STM32L072CZ	32MHz	192KB	20KB
<i>ST DISCO-L475VG-IOT01A</i>	STM32L475VGT6	80MHz	1MB	128KB
<i>ST Discovery F072RB</i>	STM32F072RBT6	48MHz	128KB	16KB
<i>ST Nucleo F030R8</i>	STM32F030R8T6	48MHz	64KB	8KB
<i>ST Nucleo F031K6</i>	STM32F031K6T6	48MHz	32KB	4KB
<i>ST Nucleo F042K6</i>	STM32F042K6T6	48MHz	32KB	6KB
<i>ST Nucleo F070RB</i>	STM32F070RBT6	48MHz	128KB	16KB
<i>ST Nucleo F072RB</i>	STM32F072RBT6	48MHz	128KB	16KB
<i>ST Nucleo F091RC</i>	STM32F091RCT6	48MHz	256KB	32KB
<i>ST Nucleo F103RB</i>	STM32F103RBT6	72MHz	128KB	20KB
<i>ST Nucleo F207ZG</i>	STM32F207ZGT6	120MHz	1MB	128KB
<i>ST Nucleo F302R8</i>	STM32F302R8T6	72MHz	64KB	16KB
<i>ST Nucleo F303K8</i>	STM32F303K8T6	72MHz	64KB	12KB
<i>ST Nucleo F303RE</i>	STM32F303RET6	72MHz	512KB	64KB
<i>ST Nucleo F303ZE</i>	STM32F303ZET6	72MHz	512KB	64KB
<i>ST Nucleo F334R8</i>	STM32F334R8T6	72MHz	64KB	16KB
<i>ST Nucleo F401RE</i>	STM32F401RET6	84MHz	512KB	96KB
<i>ST Nucleo F410RB</i>	STM32F410RBT6	100MHz	128KB	32KB
<i>ST Nucleo F411RE</i>	STM32F411RET6	100MHz	512KB	128KB
<i>ST Nucleo F412ZG</i>	STM32F412ZGT6	100MHz	1MB	256KB
<i>ST Nucleo F413ZH</i>	STM32F413ZHT6	100MHz	512KB	128KB
<i>ST Nucleo F429ZI</i>	STM32F429ZIT6	180MHz	2MB	256KB
<i>ST Nucleo F439ZI</i>	STM32F439ZIT6	180MHz	2MB	256KB
<i>ST Nucleo F446RE</i>	STM32F446RET6	180MHz	512KB	128KB
<i>ST Nucleo F446ZE</i>	STM32F446ZET6	180MHz	512KB	128KB
<i>ST Nucleo F722ZE</i>	STM32F722ZET6	216MHz	512KB	256KB
<i>ST Nucleo F746ZG</i>	STM32F746ZGT6	216MHz	1MB	320KB
<i>ST Nucleo F756ZG</i>	STM32F756ZG	216MHz	1MB	320KB
<i>ST Nucleo F767ZI</i>	STM32F767ZIT6	216MHz	2MB	512KB
<i>ST Nucleo H743ZI</i>	STM32H743ZIT6	400MHz	2MB	1MB

Continued on next page

Table 10 – continued from previous page

Name	MCU	Frequency	Flash	RAM
<i>ST Nucleo L011K4</i>	STM32L011K4T6	32MHz	16KB	2KB
<i>ST Nucleo L031K6</i>	STM32L031K6T6	32MHz	32KB	8KB
<i>ST Nucleo L053R8</i>	STM32L053R8T6	32MHz	64KB	8KB
<i>ST Nucleo L073RZ</i>	STM32L073RZ	32MHz	192KB	20KB
<i>ST Nucleo L152RE</i>	STM32L152RET6	32MHz	512KB	80KB
<i>ST Nucleo L412KB</i>	STM32L412KBU6	80MHz	128KB	40KB
<i>ST Nucleo L432KC</i>	STM32L432KCU6	80MHz	256KB	64KB
<i>ST Nucleo L433RC-P</i>	STM32L433RC	80MHz	256KB	64KB
<i>ST Nucleo L452RE</i>	STM32L452RET6	80MHz	256KB	64KB
<i>ST Nucleo L476RG</i>	STM32L476RGT6	80MHz	1MB	128KB
<i>ST Nucleo L486RG</i>	STM32L486RGT6	80MHz	1MB	128KB
<i>ST Nucleo L496ZG</i>	STM32L496ZGT6	80MHz	1MB	128KB
<i>ST Nucleo L496ZG-P</i>	STM32L496ZGT6P	80MHz	1MB	320KB
<i>ST Nucleo L4R5ZI</i>	STM32L4R5ZIT6	120MHz	2MB	640KB
<i>ST STM32F0308DISCOVERY</i>	STM32F030R8T6	48MHz	64KB	8KB
<i>ST STM32F0DISCOVERY</i>	STM32F051R8T6	48MHz	64KB	8KB
<i>ST STM32F3DISCOVERY</i>	STM32F303VCT6	72MHz	256KB	48KB
<i>ST STM32F4DISCOVERY</i>	STM32F407VGT6	168MHz	1MB	128KB
<i>ST STM32L073Z-EVAL</i>	STM32L073VZT6	32MHz	192KB	20KB
<i>ST STM32LDISCOVERY</i>	STM32L152RBT6	32MHz	128KB	16KB
<i>ST STM32VLDISCOVERY</i>	STM32F100RBT6	24MHz	128KB	8KB
<i>ST Sensor Node</i>	STM32L476JG	80MHz	1MB	128KB
<i>STM32F7508-DK</i>	STM32F750N8H6	216MHz	64KB	340KB
<i>Seeed Arch Max</i>	STM32F407VET6	168MHz	512KB	192KB
<i>Seeed Wio 3G</i>	STM32F439VI	180MHz	2MB	256KB
<i>sakura.io Evaluation Board</i>	STM32F411RET6	100MHz	1MB	128KB
<i>u-blox C030-R410M IoT</i>	STM32F437VG	180MHz	1MB	256KB

External Debug Tools

Banks 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.

Name	MCU	Frequency	Flash	RAM
<i>1Bitsy</i>	STM32F415RGT	168MHz	1MB	128KB
<i>3D Printer Controller</i>	STM32F407VET6	168MHz	512KB	192KB
<i>3D Printer control board</i>	STM32F446RET6	180MHz	512KB	128KB
<i>Armstrap Eagle 1024</i>	STM32F417VGT6	168MHz	1MB	192KB
<i>Armstrap Eagle 2048</i>	STM32F427VIT6	168MHz	1.99MB	256KB
<i>Armstrap Eagle 512</i>	STM32F407VET6	168MHz	512KB	192KB
<i>Black STM32F407VE</i>	STM32F407VET6	168MHz	512KB	128KB
<i>Black STM32F407VG</i>	STM32F407VGT6	168MHz	512KB	128KB
<i>Black STM32F407ZE</i>	STM32F407ZET6	168MHz	512KB	128KB
<i>Black STM32F407ZE</i>	STM32F407ZGT6	168MHz	1MB	128KB
<i>BlackPill F103C8</i>	STM32F103C8T6	72MHz	64KB	20KB
<i>BlackPill F103C8 (128k)</i>	STM32F103C8T6	72MHz	128KB	20KB
<i>BlackPill F303CC</i>	STM32F303CCT6	72MHz	256KB	40KB
<i>Blue STM32F407VE Mini</i>	STM32F407VET6	168MHz	512KB	128KB

Continued on next page

Table 11 – continued from previous page

Name	MCU	Frequency	Flash	RAM
<i>BluePill F103C6</i>	STM32F103C6T6	72MHz	32KB	10KB
<i>BluePill F103C8</i>	STM32F103C8T6	72MHz	64KB	20KB
<i>BluePill F103C8 (128k)</i>	STM32F103C8T6	72MHz	128KB	20KB
<i>Demo F030F4</i>	STM32F030F4P6	48MHz	16KB	4KB
<i>Espotel LoRa Module</i>	STM32F411RET6	100MHz	512KB	128KB
<i>F407VG</i>	STM32F407VGT6	168MHz	512KB	128KB
<i>FK407M1</i>	STM32F407VET6	168MHz	512KB	128KB
<i>M200 V2</i>	STM32F070CBT6	48MHz	120KB	14.81KB
<i>MTS Dragonfly</i>	STM32F411RET6	100MHz	512KB	128KB
<i>Malyan M200 V1</i>	STM32F103CBT6	72MHz	120KB	20KB
<i>Maple</i>	STM32F103RBT6	72MHz	108KB	17KB
<i>Maple (RET6)</i>	STM32F103RET6	72MHz	256KB	48KB
<i>Maple Mini Bootloader 2.0</i>	STM32F103CBT6	72MHz	120KB	20KB
<i>Maple Mini Original</i>	STM32F103CBT6	72MHz	108KB	17KB
<i>Microduino Core STM32 to Flash</i>	STM32F103CBT6	72MHz	105.47KB	16.60KB
<i>MultiTech mDot</i>	STM32F411RET6	100MHz	512KB	128KB
<i>MultiTech mDot F411</i>	STM32F411RET6	100MHz	512KB	128KB
<i>MultiTech xDot</i>	STM32L151CCU6	32MHz	256KB	32KB
<i>N2+</i>	STM32F405RGT6	168MHz	1MB	192KB
<i>NAMote72</i>	STM32L152RC	32MHz	256KB	32KB
<i>RAK811 LoRa Tracker</i>	STM32L151RBT6	32MHz	128KB	16KB
<i>RAK811 LoRa Tracker</i>	STM32L151RBT6	32MHz	128KB	32KB
<i>STM32F103C8 (20k RAM. 64k Flash)</i>	STM32F103C8T6	72MHz	64KB	20KB
<i>STM32F103CB (20k RAM. 128k Flash)</i>	STM32F103CBT6	72MHz	128KB	20KB
<i>STM32F103R8 (20k RAM. 64 Flash)</i>	STM32F103R8T6	72MHz	64KB	20KB
<i>STM32F103RB (20k RAM. 128k Flash)</i>	STM32F103RBT6	72MHz	128KB	20KB
<i>STM32F103RC (48k RAM. 256k Flash)</i>	STM32F103RCT6	72MHz	256KB	48KB
<i>STM32F103RE (64k RAM. 512k Flash)</i>	STM32F103RET6	72MHz	512KB	64KB
<i>STM32F103T8 (20k RAM. 64k Flash)</i>	STM32F103T8T6	72MHz	20KB	64KB
<i>STM32F103TB (20k RAM. 128k Flash)</i>	STM32F103TBT6	72MHz	128KB	20KB
<i>STM32F103VB (20k RAM. 128k Flash)</i>	STM32F103VBT6	72MHz	128KB	20KB
<i>STM32F103VC (48k RAM. 256k Flash)</i>	STM32F103VCT6	72MHz	256KB	48KB
<i>STM32F103VD (64k RAM. 384k Flash)</i>	STM32F103VDT6	72MHz	384KB	64KB
<i>STM32F103VE (64k RAM. 512k Flash)</i>	STM32F103VET6	72MHz	512KB	64KB
<i>STM32F103ZC (48k RAM. 256k Flash)</i>	STM32F103ZCT6	72MHz	256KB	48KB
<i>STM32F103ZD (64k RAM. 384k Flash)</i>	STM32F103ZDT6	72MHz	384KB	64KB
<i>STM32F103ZE (64k RAM. 512k Flash)</i>	STM32F103ZET6	72MHz	512KB	64KB
<i>STM32F303CB (32k RAM. 128k Flash)</i>	STM32F303CBT6	72MHz	128KB	32KB
<i>STM32F407VE (192k RAM. 512k Flash)</i>	STM32F407VET6	168MHz	502.23KB	128KB
<i>STM32F407VG (192k RAM. 1024k Flash)</i>	STM32F407VGT6	168MHz	1MB	192KB
<i>STM32F4Stamp F405</i>	STM32F405RGT6	168MHz	1MB	192KB
<i>Sparky V1 F303</i>	STM32F303CCT6	72MHz	256KB	40KB
<i>Tiny STM103T</i>	STM32F103TBU6	72MHz	128KB	20KB
<i>VakE v1.0</i>	STM32F446RET6	180MHz	512KB	128KB
<i>u-blox C030-N211 IoT Starter Kit</i>	STM32F437VG	180MHz	1MB	256KB
<i>u-blox C030-U201 IoT Starter Kit</i>	STM32F437VG	180MHz	1MB	256KB
<i>u-blox EVK-ODIN-W2</i>	STM32F439ZIY6	168MHz	2MB	256KB
<i>u-blox ODIN-W2</i>	STM32F439ZIY6	168MHz	2MB	256KB

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

```
; Latest stable version
[env:latest_stable]
platform = ststm32
board = ...

; Custom stable version
[env:custom_stable]
platform = ststm32@x.y.z
board = ...
```

Upstream

```
[env:upstream_develop]
platform = https://github.com/platformio/platform-ststm32.git
board = ...
```

Packages

Name	Description
framework-arduinostm32mxchip	Arduino Wiring-based Framework (ST STM32 MXChip Core)
framework-arduinoststm32	Arduino Wiring-based Framework (STM32 Core)
framework-arduinoststm32-maple	Arduino Wiring-based Framework (ST STM32 Maple Core)
framework-cmsis	Vendor-independent hardware abstraction layer for the Cortex-M processor series
framework-libopencm3	libOpenCM3 Framework
framework-mbed	mbed Framework
framework-spl	Standard Peripheral Library for STM32 MCUs
framework-stm32cube	STM32Cube embedded software libraries
tool-dfuutil	Host side implementation of the DFU 1.0 and DFU 1.1 specifications
tool-jlink	SEGGER J-Link Software and Documentation Pack
tool-openocd	OpenOCD
tool-stm32duino	STM32Duino Tools
toolchain-gccarmnoneabi	gcc-arm-embedded

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

Name	Description
Arduin	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
CM-SIS	The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.
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.
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.
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.
STM32	STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.

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

Name	Debug	MCU	Frequency	Flash	RAM
1Bitsy	External	STM32F415RG	168MHz	1MB	128KB

96Boards

Name	Debug	MCU	Frequency	Flash	RAM
<i>96Boards B96B-F446VE</i>	On-board	STM32F446VET6	168MHz	512KB	128KB

Armed

Name	Debug	MCU	Frequency	Flash	RAM
<i>3D Printer Controller</i>	External	STM32F407VET6	168MHz	512KB	192KB

Armstrap

Name	Debug	MCU	Frequency	Flash	RAM
<i>Armstrap Eagle 1024</i>	External	STM32F417VGT6	168MHz	1MB	192KB
<i>Armstrap Eagle 2048</i>	External	STM32F427VIT6	168MHz	1.99MB	256KB
<i>Armstrap Eagle 512</i>	External	STM32F407VET6	168MHz	512KB	192KB

Avnet Silica

Name	Debug	MCU	Frequency	Flash	RAM
<i>ST Sensor Node</i>	On-board	STM32L476JG	80MHz	1MB	128KB

Diymore

Name	Debug	MCU	Frequency	Flash	RAM
<i>F407VG</i>	External	STM32F407VGT6	168MHz	512KB	128KB

Espotel

Name	Debug	MCU	Frequency	Flash	RAM
<i>Espotel LoRa Module</i>	External	STM32F411RET6	100MHz	512KB	128KB

Generic

Name	Debug	MCU	Fre-quency	Flash	RAM
<i>BlackPill F103C8</i>	External	STM32F103C8T6	72MHz	64KB	20KB
<i>BlackPill F103C8 (128k)</i>	External	STM32F103C8T6	72MHz	128KB	20KB
<i>BluePill F103C6</i>	External	STM32F103C6T6	72MHz	32KB	10KB
<i>BluePill F103C8</i>	External	STM32F103C8T6	72MHz	64KB	20KB
<i>BluePill F103C8 (128k)</i>	External	STM32F103C8T6	72MHz	128KB	20KB
<i>Demo F030F4</i>	External	STM32F030F4P6	48MHz	16KB	4KB
<i>FK407M1</i>	External	STM32F407VET6	168MHz	512KB	128KB
<i>STM32F103C8 (20k RAM. 64k Flash)</i>	External	STM32F103C8T6	72MHz	64KB	20KB
<i>STM32F103CB (20k RAM. 128k Flash)</i>	External	STM32F103CBT6	72MHz	128KB	20KB
<i>STM32F103R8 (20k RAM. 64 Flash)</i>	External	STM32F103R8T6	72MHz	64KB	20KB
<i>STM32F103RB (20k RAM. 128k Flash)</i>	External	STM32F103RBT6	72MHz	128KB	20KB
<i>STM32F103RC (48k RAM. 256k Flash)</i>	External	STM32F103RCT6	72MHz	256KB	48KB
<i>STM32F103RE (64k RAM. 512k Flash)</i>	External	STM32F103RET6	72MHz	512KB	64KB
<i>STM32F103T8 (20k RAM. 64k Flash)</i>	External	STM32F103T8T6	72MHz	20KB	64KB
<i>STM32F103TB (20k RAM. 128k Flash)</i>	External	STM32F103TBT6	72MHz	128KB	20KB
<i>STM32F103VB (20k RAM. 128k Flash)</i>	External	STM32F103VBT6	72MHz	128KB	20KB
<i>STM32F103VC (48k RAM. 256k Flash)</i>	External	STM32F103VCT6	72MHz	256KB	48KB
<i>STM32F103VD (64k RAM. 384k Flash)</i>	External	STM32F103VDT6	72MHz	384KB	64KB
<i>STM32F103VE (64k RAM. 512k Flash)</i>	External	STM32F103VET6	72MHz	512KB	64KB
<i>STM32F103ZC (48k RAM. 256k Flash)</i>	External	STM32F103ZCT6	72MHz	256KB	48KB
<i>STM32F103ZD (64k RAM. 384k Flash)</i>	External	STM32F103ZDT6	72MHz	384KB	64KB
<i>STM32F103ZE (64k RAM. 512k Flash)</i>	External	STM32F103ZET6	72MHz	512KB	64KB
<i>STM32F303CB (32k RAM. 128k Flash)</i>	External	STM32F303CBT6	72MHz	128KB	32KB
<i>STM32F407VE (192k RAM. 512k Flash)</i>	External	STM32F407VET6	168MHz	502.23KB	128KB
<i>STM32F407VG (192k RAM. 1024k Flash)</i>	External	STM32F407VGT6	168MHz	1MB	192KB
1.10M3 Development Platforms	External	STM32F405RGT6	168MHz	1MB	192KB

HY

Name	Debug	MCU	Frequency	Flash	RAM
<i>Tiny STM103T</i>	External	STM32F103TBU6	72MHz	128KB	20KB

LeafLabs

Name	Debug	MCU	Frequency	Flash	RAM
<i>Maple</i>	External	STM32F103RBT6	72MHz	108KB	17KB
<i>Maple (RET6)</i>	External	STM32F103RET6	72MHz	256KB	48KB
<i>Maple Mini Bootloader 2.0</i>	External	STM32F103CBT6	72MHz	120KB	20KB
<i>Maple Mini Original</i>	External	STM32F103CBT6	72MHz	108KB	17KB

MXChip

Name	Debug	MCU	Frequency	Flash	RAM
<i>Microsoft Azure IoT Development Kit (MXChip AZ3166)</i>	On-board	STM32F412ZGT6	100MHz	1MB	256KB

Malyan

Name	Debug	MCU	Frequency	Flash	RAM
<i>M200 V2</i>	External	STM32F070CBT6	48MHz	120KB	14.81KB
<i>Malyan M200 V1</i>	External	STM32F103CBT6	72MHz	120KB	20KB

Microduino

Name	Debug	MCU	Frequency	Flash	RAM
<i>Microduino Core STM32 to Flash</i>	External	STM32F103CBT6	72MHz	105.47KB	16.60KB

MultiTech

Name	Debug	MCU	Frequency	Flash	RAM
<i>MTS Dragonfly</i>	External	STM32F411RET6	100MHz	512KB	128KB
<i>MultiTech mDot</i>	External	STM32F411RET6	100MHz	512KB	128KB
<i>MultiTech mDot F411</i>	External	STM32F411RET6	100MHz	512KB	128KB
<i>MultiTech xDot</i>	External	STM32L151CCU6	32MHz	256KB	32KB

Netduino

Name	Debug	MCU	Frequency	Flash	RAM
N2+	External	STM32F405RGT6	168MHz	1MB	192KB

RAK

Name	Debug	MCU	Frequency	Flash	RAM
RAK811 LoRa Tracker	External	STM32L151RBT6	32MHz	128KB	16KB
RAK811 LoRa Tracker	External	STM32L151RBT6	32MHz	128KB	32KB

RUMBA

Name	Debug	MCU	Frequency	Flash	RAM
3D Printer control board	External	STM32F446RET6	180MHz	512KB	128KB

RemRam

Name	Debug	MCU	Frequency	Flash	RAM
3D printer controller	On-board	STM32F765VIT6	216MHz	2MB	512KB

RobotDyn

Name	Debug	MCU	Frequency	Flash	RAM
BlackPill F303CC	External	STM32F303CCT6	72MHz	256KB	40KB

RushUp

Name	Debug	MCU	Frequency	Flash	RAM
RushUp Cloud-JAM	On-board	STM32F401RET6	84MHz	512KB	96KB
RushUp Cloud-JAM L4	On-board	STM32L476RGT6	80MHz	1MB	128KB

ST

Name	Debug	MCU	Frequency	Flash	RAM
32F723EDISCOVERY	On-board	STM32F723IEK6	216MHz	512KB	192KB
3DP001V1 Evaluation board for 3D printer	On-board	STM32F401VGT6	84MHz	512KB	96KB
Black STM32F407VE	External	STM32F407VET6	168MHz	512KB	128KB
Black STM32F407VG	External	STM32F407VGT6	168MHz	512KB	128KB

Continued on next page

Table 12 – continued from previous page

Name	Debug	MCU	Frequency	Flash	RAM
<i>Black STM32F407ZE</i>	External	STM32F407ZET6	168MHz	512KB	128KB
<i>Black STM32F407ZE</i>	External	STM32F407ZGT6	168MHz	1MB	128KB
<i>Blue STM32F407VE Mini</i>	External	STM32F407VET6	168MHz	512KB	128KB
<i>Nucleo G071RB</i>	On-board	STM32G071RBT6	24MHz	2MB	128KB
<i>P-Nucleo WB55RG</i>	On-board	STM32WB55RG	64MHz	512KB	192.00KB
<i>ST 32F3348DISCOVERY</i>	On-board	STM32F334C8T6	72MHz	64KB	12KB
<i>ST 32F401CDISCOVERY</i>	On-board	STM32F401VCT6	84MHz	256KB	64KB
<i>ST 32F411EDISCOVERY</i>	On-board	STM32F411VET6	100MHz	512KB	128KB
<i>ST 32F413HDISCOVERY</i>	On-board	STM32F413ZHT6	100MHz	512KB	128KB
<i>ST 32F429IDISCOVERY</i>	On-board	STM32F429ZIT6	180MHz	2MB	256KB
<i>ST 32F469IDISCOVERY</i>	On-board	STM32F469NIH6	180MHz	1MB	384KB
<i>ST 32F746GDISCOVERY</i>	On-board	STM32F746NGH6	216MHz	1MB	320KB
<i>ST 32F769IDISCOVERY</i>	On-board	STM32F769NIH6	216MHz	1MB	512KB
<i>ST 32L0538DISCOVERY</i>	On-board	STM32L053C8T6	32MHz	64KB	8KB
<i>ST 32L100DISCOVERY</i>	On-board	STM32L100RCT6	32MHz	256KB	16KB
<i>ST 32L476GDISCOVERY</i>	On-board	STM32L476VGT6	80MHz	1MB	128KB
<i>ST 32L496GDISCOVERY</i>	On-board	STM32L496AGI6	80MHz	1MB	320KB
<i>ST DISCO-L072CZ-LRWANI</i>	On-board	STM32L072CZ	32MHz	192KB	20KB
<i>ST DISCO-L475VG-IOT01A</i>	On-board	STM32L475VGT6	80MHz	1MB	128KB
<i>ST Discovery F072RB</i>	On-board	STM32F072RBT6	48MHz	128KB	16KB
<i>ST Nucleo F030R8</i>	On-board	STM32F030R8T6	48MHz	64KB	8KB
<i>ST Nucleo F031K6</i>	On-board	STM32F031K6T6	48MHz	32KB	4KB
<i>ST Nucleo F042K6</i>	On-board	STM32F042K6T6	48MHz	32KB	6KB
<i>ST Nucleo F070RB</i>	On-board	STM32F070RBT6	48MHz	128KB	16KB
<i>ST Nucleo F072RB</i>	On-board	STM32F072RBT6	48MHz	128KB	16KB
<i>ST Nucleo F091RC</i>	On-board	STM32F091RCT6	48MHz	256KB	32KB
<i>ST Nucleo F103RB</i>	On-board	STM32F103RBT6	72MHz	128KB	20KB
<i>ST Nucleo F207ZG</i>	On-board	STM32F207ZGT6	120MHz	1MB	128KB
<i>ST Nucleo F302R8</i>	On-board	STM32F302R8T6	72MHz	64KB	16KB
<i>ST Nucleo F303K8</i>	On-board	STM32F303K8T6	72MHz	64KB	12KB
<i>ST Nucleo F303RE</i>	On-board	STM32F303RET6	72MHz	512KB	64KB
<i>ST Nucleo F303ZE</i>	On-board	STM32F303ZET6	72MHz	512KB	64KB
<i>ST Nucleo F334R8</i>	On-board	STM32F334R8T6	72MHz	64KB	16KB
<i>ST Nucleo F401RE</i>	On-board	STM32F401RET6	84MHz	512KB	96KB
<i>ST Nucleo F410RB</i>	On-board	STM32F410RBT6	100MHz	128KB	32KB
<i>ST Nucleo F411RE</i>	On-board	STM32F411RET6	100MHz	512KB	128KB
<i>ST Nucleo F412ZG</i>	On-board	STM32F412ZGT6	100MHz	1MB	256KB
<i>ST Nucleo F413ZH</i>	On-board	STM32F413ZHT6	100MHz	512KB	128KB
<i>ST Nucleo F429ZI</i>	On-board	STM32F429ZIT6	180MHz	2MB	256KB
<i>ST Nucleo F439ZI</i>	On-board	STM32F439ZIT6	180MHz	2MB	256KB
<i>ST Nucleo F446RE</i>	On-board	STM32F446RET6	180MHz	512KB	128KB
<i>ST Nucleo F446ZE</i>	On-board	STM32F446ZET6	180MHz	512KB	128KB
<i>ST Nucleo F722ZE</i>	On-board	STM32F722ZET6	216MHz	512KB	256KB
<i>ST Nucleo F746ZG</i>	On-board	STM32F746ZGT6	216MHz	1MB	320KB
<i>ST Nucleo F756ZG</i>	On-board	STM32F756ZG	216MHz	1MB	320KB
<i>ST Nucleo F767ZI</i>	On-board	STM32F767ZIT6	216MHz	2MB	512KB
<i>ST Nucleo H743ZI</i>	On-board	STM32H743ZIT6	400MHz	2MB	1MB
<i>ST Nucleo L011K4</i>	On-board	STM32L011K4T6	32MHz	16KB	2KB
<i>ST Nucleo L031K6</i>	On-board	STM32L031K6T6	32MHz	32KB	8KB

Continued on next page

Table 12 – continued from previous page

Name	Debug	MCU	Frequency	Flash	RAM
<i>ST Nucleo L053R8</i>	On-board	STM32L053R8T6	32MHz	64KB	8KB
<i>ST Nucleo L073RZ</i>	On-board	STM32L073RZ	32MHz	192KB	20KB
<i>ST Nucleo L152RE</i>	On-board	STM32L152RET6	32MHz	512KB	80KB
<i>ST Nucleo L412KB</i>	On-board	STM32L412KBU6	80MHz	128KB	40KB
<i>ST Nucleo L432KC</i>	On-board	STM32L432KCU6	80MHz	256KB	64KB
<i>ST Nucleo L433RC-P</i>	On-board	STM32L433RC	80MHz	256KB	64KB
<i>ST Nucleo L452RE</i>	On-board	STM32L452RET6	80MHz	256KB	64KB
<i>ST Nucleo L476RG</i>	On-board	STM32L476RGT6	80MHz	1MB	128KB
<i>ST Nucleo L486RG</i>	On-board	STM32L486RGT6	80MHz	1MB	128KB
<i>ST Nucleo L496ZG</i>	On-board	STM32L496ZGT6	80MHz	1MB	128KB
<i>ST Nucleo L496ZG-P</i>	On-board	STM32L496ZGT6P	80MHz	1MB	320KB
<i>ST Nucleo L4R5ZI</i>	On-board	STM32L4R5ZIT6	120MHz	2MB	640KB
<i>ST STM32F0308DISCOVERY</i>	On-board	STM32F030R8T6	48MHz	64KB	8KB
<i>ST STM32F0DISCOVERY</i>	On-board	STM32F051R8T6	48MHz	64KB	8KB
<i>ST STM32F3DISCOVERY</i>	On-board	STM32F303VCT6	72MHz	256KB	48KB
<i>ST STM32F4DISCOVERY</i>	On-board	STM32F407VGT6	168MHz	1MB	128KB
<i>ST STM32L073Z-EVAL</i>	On-board	STM32L073VZT6	32MHz	192KB	20KB
<i>ST STM32LDISCOVERY</i>	On-board	STM32L152RBT6	32MHz	128KB	16KB
<i>ST STM32VLDISCOVERY</i>	On-board	STM32F100RBT6	24MHz	128KB	8KB
<i>STM32F7508-DK</i>	On-board	STM32F750N8H6	216MHz	64KB	340KB

SeeedStudio

Name	Debug	MCU	Frequency	Flash	RAM
<i>Seeed Arch Max</i>	On-board	STM32F407VET6	168MHz	512KB	192KB
<i>Seeed Wio 3G</i>	On-board	STM32F439VI	180MHz	2MB	256KB

Semtech

Name	Debug	MCU	Frequency	Flash	RAM
<i>NAMote72</i>	External	STM32L152RC	32MHz	256KB	32KB

TauLabs

Name	Debug	MCU	Frequency	Flash	RAM
<i>Sparky V1 F303</i>	External	STM32F303CCT6	72MHz	256KB	40KB

VAE

Name	Debug	MCU	Frequency	Flash	RAM
<i>VAKe v1.0</i>	External	STM32F446RET6	180MHz	512KB	128KB

rhomb.io

Name	Debug	MCU	Frequency	Flash	RAM
L476DMWIK	On-board	STM32L476VGT6	80MHz	1MB	128KB

sakura.io

Name	Debug	MCU	Frequency	Flash	RAM
sakura.io Evaluation Board	On-board	STM32F411RET6	100MHz	1MB	128KB

u-blox

Name	Debug	MCU	Frequency	Flash	RAM
Mbed Connect Cloud	On-board	STM32F439ZIY6	168MHz	2MB	256KB
u-blox C030-N211 IoT Starter Kit	External	STM32F437VG	180MHz	1MB	256KB
u-blox C030-R410M IoT	On-board	STM32F437VG	180MHz	1MB	256KB
u-blox C030-U201 IoT Starter Kit	External	STM32F437VG	180MHz	1MB	256KB
u-blox EVK-ODIN-W2	External	STM32F439ZIY6	168MHz	2MB	256KB
u-blox ODIN-W2	External	STM32F439ZIY6	168MHz	2MB	256KB

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*
 - *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

Bands listed below have on-board debug probe and **ARE READY** for debugging! You do not need to use/buy external debug probe.

Name	MCU	Frequency	Flash	RAM
<i>ST STM8S-DISCOVERY</i>	STM8S105C6T6	16MHz	32KB	2KB

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

```
; Latest stable version
[env:latest_stable]
platform = ststm8
board = ...

; Custom stable version
[env:custom_stable]
```

(continues on next page)

(continued from previous page)

```
platform = ststm8@x.y.z
board = ...
```

Upstream

```
[env:upstream_develop]
platform = https://github.com/platformio/platform-ststm8.git
board = ...
```

Packages

Name	Description
framework-arduinoststm8	Arduino Wiring-based Framework (STM8 Core)
framework-ststm8spl	STM8S/A Standard peripheral library
tool-openocd	OpenOCD
tool-stm8binutils	STM8 GNU binutils
tool-stm8tools	STM8 upload tools
toolchain-sdcc	Small Device C Compiler

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

Name	Description
<i>Arduin</i> o	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>SPL</i>	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.

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.

ST

Name	Debug	MCU	Frequency	Flash	RAM
<i>ST STM8S-DISCOVERY</i>	On-board	STM8S105C6T6	16MHz	32KB	2KB
<i>ST STM8S103F3 Breakout Board</i>	No	STM8S103F3P6	16MHz	8KB	1KB
<i>ST STM8S105K4T6 Breakout Board</i>	No	STM8S105K4T6	16MHz	16KB	2KB

sduino

Name	Debug	MCU	Frequency	Flash	RAM
<i>sduino MB (STM8S208MBT6B)</i>	No	STM8S208MBT6	16MHz	128KB	6KB
<i>sduino UNO (STM8S105K6)</i>	No	STM8S105K6T6	16MHz	32KB	2KB

Teensy

Configuration *platform* = teensy

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

For more detailed information please visit [vendor site](#).

Contents

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

Configuration

- *Optimization*
- *USB Features*

Optimization

(valid only for Teensy LC, Teensy 3.0-3.6)

You can control firmware optimization via special macro/define using *build_flags* in “*platformio.ini*” (*Project Configuration File*):

- -D TEENSY_OPT_FASTER, **default**
- -D TEENSY_OPT_FASTER_LTO
- -D TEENSY_OPT_FAST
- -D TEENSY_OPT_FAST_LTO
- -D TEENSY_OPT_FASTEST
- -D TEENSY_OPT_FASTEST_LTO
- -D TEENSY_OPT_FASTEST_PURE_CODE, valid only for Teensy 3.5-3.6
- -D TEENSY_OPT_FASTEST_PURE_CODE_LTO, valid only for Teensy 3.5-3.6
- -D TEENSY_OPT_DEBUG
- -D TEENSY_OPT_DEBUG_LTO
- -D TEENSY_OPT_SMALLEST_CODE
- -D TEENSY_OPT_SMALLEST_CODE_LTO

The only one macro can be used in per one build environment. Also, you can see verbose build using `-v`, `--verbose` option for *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:

```
[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-serial

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

Banks 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.

Name	MCU	Frequency	Flash	RAM
Teensy 3.1 / 3.2	MK20DX256	72MHz	256KB	64KB
Teensy 3.5	MK64FX512	120MHz	512KB	255.99KB
Teensy 3.6	MK66FX1M0	180MHz	1MB	256KB
Teensy 4.0	IMXRT1062	600MHz	1.94MB	1MB
Teensy LC	MKL26Z64	48MHz	62KB	8KB

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 = ...
```

Packages

Name	Description
framework-arduinoteensy	Arduino Wiring-based Framework
framework-mbed	mbed Framework
tool-jlink	SEGGER J-Link Software and Documentation Pack
tool-teensy	Teensy Loader
toolchain-atmelavr	avr-gcc
toolchain-gccarmnoneabi	gcc-arm-embedded

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

Name	Description
Arduin	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
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.

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

Name	Debug	MCU	Frequency	Flash	RAM
Teensy 2.0	No	ATMEGA32U4	16MHz	31.50KB	2.50KB
Teensy 3.0	No	MK20DX128	48MHz	128KB	16KB
Teensy 3.1 / 3.2	External	MK20DX256	72MHz	256KB	64KB
Teensy 3.5	External	MK64FX512	120MHz	512KB	255.99KB
Teensy 3.6	External	MK66FX1M0	180MHz	1MB	256KB
Teensy 4.0	External	IMXRT1062	600MHz	1.94MB	1MB
Teensy LC	External	MKL26Z64	48MHz	62KB	8KB
Teensy++ 2.0	No	AT90USB1286	16MHz	127KB	8KB

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](#).

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

Bands listed below have on-board debug probe and **ARE READY** for debugging! You do not need to use/buy external debug probe.

Name	MCU	Frequency	Flash	RAM
<i>TI FraunchPad MSP-EXP430FR5739LP</i>	MSP430FR5739	16MHz	15.37KB	1KB
<i>TI LaunchPad MSP-EXP430F5529LP</i>	MSP430F5529	25MHz	47KB	8KB
<i>TI LaunchPad MSP-EXP430FR2311LP</i>	MSP430FR2311	16MHz	3.75KB	1KB
<i>TI LaunchPad MSP-EXP430FR2433LP</i>	MSP430FR2433	8MHz	15KB	4KB
<i>TI LaunchPad MSP-EXP430FR4133LP</i>	MSP430FR4133	8MHz	15KB	2KB
<i>TI LaunchPad MSP-EXP430FR5969LP</i>	MSP430FR5969	8MHz	47KB	2KB
<i>TI LaunchPad MSP-EXP430FR5994LP</i>	MSP430FR5994	16MHz	256KB	4KB
<i>TI LaunchPad MSP-EXP430FR6989LP</i>	MSP430FR6989	8MHz	47KB	2KB
<i>TI LaunchPad MSP-EXP430G2 w/ MSP430G2231</i>	MSP430G2231	1MHz	2KB	256B
<i>TI LaunchPad MSP-EXP430G2 w/ MSP430G2452</i>	MSP430G2452	16MHz	8KB	256B
<i>TI LaunchPad MSP-EXP430G2553LP</i>	MSP430G2553	16MHz	16KB	512B

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

```
; Latest stable version
[env:latest_stable]
platform = timsp430
board = ...

; Custom stable version
[env:custom_stable]
platform = timsp430@x.y.z
board = ...
```

Upstream

```
[env:upstream_develop]
platform = https://github.com/platformio/platform-timsp430.git
board = ...
```

Packages

Name	Description
framework-energiamsp430	Energia Wiring-based Framework (MSP430 Core)
tool-dslite	Uniflash Standalone Flash Tool for TI Microcontrollers, Sitara Processors & SimpleLink devices
tool-mspdebug	MSPDebug
toolchain-timsp430	msp-gcc

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

Name	Description
Arduin	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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

Name	Debug	MCU	Frequency	Flash	RAM
<i>TI FraunchPad MSP-EXP430FR5739LP</i>	On-board	MSP430FR5739	16MHz	15.37KB	1KB
<i>TI LaunchPad MSP-EXP430F5529LP</i>	On-board	MSP430F5529	25MHz	47KB	8KB
<i>TI LaunchPad MSP-EXP430FR2311LP</i>	On-board	MSP430FR2311	16MHz	3.75KB	1KB
<i>TI LaunchPad MSP-EXP430FR2433LP</i>	On-board	MSP430FR2433	8MHz	15KB	4KB
<i>TI LaunchPad MSP-EXP430FR4133LP</i>	On-board	MSP430FR4133	8MHz	15KB	2KB
<i>TI LaunchPad MSP-EXP430FR5969LP</i>	On-board	MSP430FR5969	8MHz	47KB	2KB
<i>TI LaunchPad MSP-EXP430FR5994LP</i>	On-board	MSP430FR5994	16MHz	256KB	4KB
<i>TI LaunchPad MSP-EXP430FR6989LP</i>	On-board	MSP430FR6989	8MHz	47KB	2KB
<i>TI LaunchPad MSP-EXP430G2 w/ MSP430G2231</i>	On-board	MSP430G2231	1MHz	2KB	256B
<i>TI LaunchPad MSP-EXP430G2 w/ MSP430G2452</i>	On-board	MSP430G2452	16MHz	8KB	256B
<i>TI LaunchPad MSP-EXP430G2553LP</i>	On-board	MSP430G2553	16MHz	16KB	512B

TI TIVA

Configuration `platform = titiva`

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

For more detailed information please visit [vendor site](#).

Contents

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

Examples

Examples are listed from [TI TIVA development platform repository](#):

- 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*
 - *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

Bands listed below have on-board debug probe and **ARE READY** for debugging! You do not need to use/buy external debug probe.

Name	MCU	Frequency	Flash	RAM
TI LaunchPad (Stellaris) w/ lm4f120 (80MHz)	LPLM4F120H5QR	80MHz	256KB	32KB
TI LaunchPad (Tiva C) w/ tm4c123 (80MHz)	LPTM4C1230C3PM	80MHz	256KB	32KB
TI LaunchPad (Tiva C) w/ tm4c129 (120MHz)	LPTM4C1294NCPDT	120MHz	1MB	256KB

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

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

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

Upstream

```
[env:upstream_develop]
platform = https://github.com/platformio/platform-titiva.git
board = ...
```

Packages

Name	Description
framework-energiativa	Energia Wiring-based Framework (LM4F Core)
framework-libopencm3	libOpenCM3 Framework
tool-openocd	OpenOCD
toolchain-gccarmnoneabi	gcc-arm-embedded

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

Name	Description
<i>Ar-duino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>libOpenCM3</i>	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.

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

Name	Debug	MCU	Fre-quency	Flash	RAM
<i>TI LaunchPad (Stellaris) w/ lm4f120 (80MHz)</i>	On-board	LPLM4F120H5QR	80MHz	256KB	32KB
<i>TI LaunchPad (Tiva C) w/ tm4c123 (80MHz)</i>	On-board	LPTM4C1230C3PM	80MHz	256KB	32KB
<i>TI LaunchPad (Tiva C) w/ tm4c129 (120MHz)</i>	On-board	LPTM4C1294NCPDT	120MHz	1MB	256KB

WIZNet W7500

Configuration `platform = wiznet7500`

The IOP (Internet Offload Processor) W7500 is the one-chip solution which integrates an ARM Cortex-M0, 128KB Flash and hardwired TCP/IP core for various embedded application platform especially requiring Internet of things

For more detailed information please visit [vendor site](#).

Contents

- [Examples](#)
- [Debugging](#)
- [Stable and upstream versions](#)
- [Packages](#)
- [Frameworks](#)
- [Boards](#)

Examples

Examples are listed from WIZNet W7500 development platform repository:

- [mbed-blink](#)
- [mbed-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*

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

Banks listed below have on-board debug probe and **ARE READY** for debugging! You do not need to use/buy external debug probe.

Name	MCU	Frequency	Flash	RAM
<i>WIZwiki-W7500</i>	WIZNET7500	48MHz	128KB	48KB
<i>WIZwiki-W7500ECO</i>	WIZNET7500ECO	48MHz	128KB	48KB
<i>WIZwiki-W7500P</i>	WIZNET7500P	48MHz	128KB	48KB

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

Name	Description
framework-mbed	mbed Framework
tool-jlink	SEGGER J-Link Software and Documentation Pack
tool-pyocd	Open source python library for programming and debugging ARM Cortex-M microcontrollers using CMSIS-DAP
toolchain-gccarmnoneabi	gcc-arm-embedded

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

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.

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

Name	Debug	MCU	Frequency	Flash	RAM
WIZwiki-W7500	On-board	WIZNET7500	48MHz	128KB	48KB
WIZwiki-W7500ECO	On-board	WIZNET7500ECO	48MHz	128KB	48KB
WIZwiki-W7500P	On-board	WIZNET7500P	48MHz	128KB	48KB

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

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

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

Upstream

```
[env:upstream_develop]
platform = https://github.com/platformio/platform-linux_arm.git
board = ...
```

Packages

Name	Description
framework-artik-sdk	ARTIK SDK is a C/C++ SDK targeting Samsung ARTIK platforms
framework-wiringpi	GPIO Interface library for the Raspberry Pi
toolchain-gccarmlinuxgnueabi	GCC for Linux ARM GNU EABI

Frameworks

Name	Description
<i>ARTIK SDK</i>	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.
<i>Wiring</i>	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.

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

Name	Debug	MCU	Frequency	Flash	RAM
<i>Raspberry Pi 1 Model B</i>	No	BCM2835	700MHz	512MB	512MB
<i>Raspberry Pi 2 Model B</i>	No	BCM2836	900MHz	1GB	1GB
<i>Raspberry Pi 3 Model B</i>	No	BCM2837	1200MHz	1GB	1GB
<i>Raspberry Pi Zero</i>	No	BCM2835	1000MHz	512MB	512MB

RushUp

Name	Debug	MCU	Frequency	Flash	RAM
<i>RushUp Kitra 520</i>	No	EXYNOS3250	1000MHz	4GB	512MB

Samsung

Name	Debug	MCU	Frequency	Flash	RAM
<i>Samsung ARTIK 1020</i>	No	EXYNOS5422	1500MHz	16GB	2GB
<i>Samsung ARTIK 520</i>	No	EXYNOS3250	1000MHz	4GB	512MB
<i>Samsung ARTIK 530</i>	No	S5P4418	1200MHz	4GB	512MB
<i>Samsung ARTIK 710</i>	No	S5P6818	1400MHz	4GB	1GB

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

Name	Description
toolchain-gcclinux32	GCC for Linux i686

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

```
; Latest stable version
[env:latest_stable]
platform = linux_x86_64
board = ...

; Custom stable version
```

(continues on next page)

(continued from previous page)

```
[env:custom_stable]
platform = linux_x86_64@x.y.z
board = ...
```

Upstream

```
[env:upstream_develop]
platform = https://github.com/platformio/platform-linux_x86_64.git
board = ...
```

Packages

Name	Description
toolchain-gcclinux64	GCC for Linux x86_64

Windows x86

Configuration `platform = windows_x86`

Windows x86 (32-bit) is a metafamily of graphical operating systems developed and marketed by Microsoft. Using host OS (Windows, Linux 32/64 or Mac OS X) you can build native application for Windows x86 platform.

For more detailed information please visit [vendor site](#).

Contents

- [Examples](#)
- [Stable and upstream versions](#)
- [Packages](#)

Examples

Examples are listed from Windows x86 development platform repository:

- [hello-world](#)

Stable and upstream versions

You can switch between stable releases of Windows x86 development platform and the latest upstream version using `platform` option in “*platformio.ini*” (*Project Configuration File*) as described below.

Stable

```
; Latest stable version
[env:latest_stable]
platform = windows_x86
board = ...

; Custom stable version
[env:custom_stable]
platform = windows_x86@x.y.z
board = ...
```

Upstream

```
[env:upstream_develop]
platform = https://github.com/platformio/platform-windows_x86.git
board = ...
```

Packages

Name	Description
toolchain-gccmingw32	MinGW

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](#).

Contents

- *Tutorials*
- *Configuration*
- *Debugging*
- *Examples*
- *Platforms*
- *Boards*

Tutorials

- [Get started with Arduino and ESP32-DevKitC: debugging and unit testing](#)
- [Arduino and Nordic nRF52-DK: debugging and unit testing](#)

Configuration

MiniCore, MightyCore, MegaCore

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](#)
 - [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

Bands listed below have on-board debug probe and **ARE READY** for debugging! You do not need to use/buy external debug probe.

Name	Platform	MCU	Frequency	Flash	RAM
3D printer controller	<i>ST STM32</i>	STM32F765VIT6	216MHz	2MB	512KB
3DP001VI Evaluation board for 3D printer	<i>ST STM32</i>	STM32F401VGT6	84MHz	512KB	96KB
Adafruit Bluefruit nRF52832 Feather	<i>Nordic nRF52</i>	NRF52832	64MHz	512KB	64KB
Adafruit Feather nRF52840 Express	<i>Nordic nRF52</i>	NRF52840	64MHz	796KB	243KB
Arduino M0 Pro (Programming/Debug Port)	<i>Atmel SAM</i>	SAMD21G18A	48MHz	256KB	32KB

Continued on next page

Table 13 – continued from previous page

Name	Platform	MCU	Frequency	Flash	RAM
<i>Arduino Zero (Programming/Debug Port)</i>	<i>Atmel SAM</i>	SAMD21G18A	48MHz	256KB	32KB
<i>Atmel ATSAMW25-XPRO</i>	<i>Atmel SAM</i>	SAMD21G18A	48MHz	256KB	32KB
<i>BBC micro:bit</i>	<i>Nordic nRF51</i>	NRF51822	16MHz	256KB	16KB
<i>Calliope mini</i>	<i>Nordic nRF51</i>	NRF51822	16MHz	256KB	16KB
<i>Delta DFBM-NQ620</i>	<i>Nordic nRF52</i>	NRF52832	64MHz	512KB	64KB
<i>Espressif ESP-WROVER-KIT</i>	<i>Espressif 32</i>	ESP32	240MHz	4MB	320KB
<i>Metro nRF52840 Express</i>	<i>Nordic nRF52</i>	NRF52840	64MHz	796KB	243KB
<i>Microsoft Azure IoT Development Kit (MXChip AZ3166)</i>	<i>ST STM32</i>	STM32F412ZGT6	100MHz	1MB	256KB
<i>Nordic Beacon Kit (PCA20006)</i>	<i>Nordic nRF51</i>	NRF51822	32MHz	256KB	32KB
<i>Nordic nRF51 Dongle (PCA10031)</i>	<i>Nordic nRF51</i>	NRF51822	32MHz	256KB	32KB
<i>Nordic nRF51X22 Development Kit(PCA1000X)</i>	<i>Nordic nRF51</i>	NRF51822	32MHz	256KB	32KB
<i>Nordic nRF52-DK</i>	<i>Nordic nRF52</i>	NRF52832	64MHz	512KB	64KB
<i>Nordic nRF52840-DK</i>	<i>Nordic nRF52</i>	NRF52840	64MHz	1MB	256KB
<i>Nordic nRF52840-DK (Adafruit BSP)</i>	<i>Nordic nRF52</i>	NRF52840	64MHz	796KB	243KB
<i>Nucleo G071RB</i>	<i>ST STM32</i>	STM32G071RBT6	24MHz	2MB	128KB
<i>P-Nucleo WB55RG</i>	<i>ST STM32</i>	STM32WB55RG	64MHz	512KB	192KB
<i>RedBearLab BLE Nano 1.5</i>	<i>Nordic nRF51</i>	NRF51822	16MHz	256KB	32KB
<i>RedBearLab BLE Nano 2</i>	<i>Nordic nRF52</i>	NRF52832	64MHz	512KB	64KB
<i>RedBearLab Blend 2</i>	<i>Nordic nRF52</i>	NRF52832	64MHz	512KB	64KB
<i>RedBearLab nRF51822</i>	<i>Nordic nRF51</i>	NRF51822	16MHz	256KB	16KB
<i>ST 32F746DISCOVERY</i>	<i>ST STM32</i>	STM32F746NGH6	216MHz	1MB	320KB
<i>ST DISCO-L072CZ-LRWAN1</i>	<i>ST STM32</i>	STM32L072CZ	32MHz	192KB	20KB
<i>ST DISCO-L475VG-IOT01A</i>	<i>ST STM32</i>	STM32L475VGT6	80MHz	1MB	128KB
<i>ST Discovery F072RB</i>	<i>ST STM32</i>	STM32F072RBT6	48MHz	128KB	16KB
<i>ST Nucleo F030R8</i>	<i>ST STM32</i>	STM32F030R8T6	48MHz	64KB	8KB
<i>ST Nucleo F091RC</i>	<i>ST STM32</i>	STM32F091RCT6	48MHz	256KB	32KB
<i>ST Nucleo F103RB</i>	<i>ST STM32</i>	STM32F103RBT6	72MHz	128KB	20KB
<i>ST Nucleo F207ZG</i>	<i>ST STM32</i>	STM32F207ZGT6	120MHz	1MB	128KB
<i>ST Nucleo F302R8</i>	<i>ST STM32</i>	STM32F302R8T6	72MHz	64KB	16KB
<i>ST Nucleo F303K8</i>	<i>ST STM32</i>	STM32F303K8T6	72MHz	64KB	12KB
<i>ST Nucleo F303RE</i>	<i>ST STM32</i>	STM32F303RET6	72MHz	512KB	64KB
<i>ST Nucleo F401RE</i>	<i>ST STM32</i>	STM32F401RET6	84MHz	512KB	96KB
<i>ST Nucleo F411RE</i>	<i>ST STM32</i>	STM32F411RET6	100MHz	512KB	128KB
<i>ST Nucleo F429ZI</i>	<i>ST STM32</i>	STM32F429ZIT6	180MHz	2MB	256KB
<i>ST Nucleo F446RE</i>	<i>ST STM32</i>	STM32F446RET6	180MHz	512KB	128KB
<i>ST Nucleo F767ZI</i>	<i>ST STM32</i>	STM32F767ZIT6	216MHz	2MB	512KB
<i>ST Nucleo H743ZI</i>	<i>ST STM32</i>	STM32H743ZIT6	400MHz	2MB	1MB
<i>ST Nucleo L031K6</i>	<i>ST STM32</i>	STM32L031K6T6	32MHz	32KB	8KB
<i>ST Nucleo L053R8</i>	<i>ST STM32</i>	STM32L053R8T6	32MHz	64KB	8KB
<i>ST Nucleo L073RZ</i>	<i>ST STM32</i>	STM32L073RZ	32MHz	192KB	20KB
<i>ST Nucleo L152RE</i>	<i>ST STM32</i>	STM32L152RET6	32MHz	512KB	80KB
<i>ST Nucleo L412KB</i>	<i>ST STM32</i>	STM32L412KBU6	80MHz	128KB	40KB
<i>ST Nucleo L432KC</i>	<i>ST STM32</i>	STM32L432KCU6	80MHz	256KB	64KB
<i>ST Nucleo L452RE</i>	<i>ST STM32</i>	STM32L452RET6	80MHz	256KB	64KB
<i>ST Nucleo L476RG</i>	<i>ST STM32</i>	STM32L476RGTE6	80MHz	1MB	128KB
<i>ST Nucleo L496ZG</i>	<i>ST STM32</i>	STM32L496ZGT6	80MHz	1MB	128KB
<i>ST Nucleo L496ZG-P</i>	<i>ST STM32</i>	STM32L496ZGT6P	80MHz	1MB	320KB
<i>ST Nucleo L4R5ZI</i>	<i>ST STM32</i>	STM32L4R5ZIT6	120MHz	2MB	640KB
<i>ST STM32F0308DISCOVERY</i>	<i>ST STM32</i>	STM32F030R8T6	48MHz	64KB	8KB

Continued on next page

Table 13 – continued from previous page

Name	Platform	MCU	Frequency	Flash	RAM
<i>ST STM32F4DISCOVERY</i>	<i>ST STM32</i>	STM32F407VGT6	168MHz	1MB	128KB
<i>ST STM32VLDISCOVERY</i>	<i>ST STM32</i>	STM32F100RBT6	24MHz	128KB	8KB
<i>ST STM8S-DISCOVERY</i>	<i>ST STM8</i>	STM8S105C6T6	16MHz	32KB	2KB
<i>Seeed Tiny BLE</i>	<i>Nordic nRF51</i>	NRF51822	16MHz	256KB	16KB
<i>TI FraunchPad MSP-EXP430FR5739LP</i>	<i>TI MSP430</i>	MSP430FR5739	16MHz	15.37KB	1KB
<i>TI LaunchPad (Stellaris) w/ lm4f120 (80MHz)</i>	<i>TI TIVA</i>	LPLM4F120H5QR	80MHz	256KB	32KB
<i>TI LaunchPad (Tiva C) w/ tm4c123 (80MHz)</i>	<i>TI TIVA</i>	LPTM4C1230C3PM	80MHz	256KB	32KB
<i>TI LaunchPad (Tiva C) w/ tm4c129 (120MHz)</i>	<i>TI TIVA</i>	LPTM4C1294NCPDT	120MHz	1MB	256KB
<i>TI LaunchPad MSP-EXP430F5529LP</i>	<i>TI MSP430</i>	MSP430F5529	25MHz	47KB	8KB
<i>TI LaunchPad MSP-EXP430FR2311LP</i>	<i>TI MSP430</i>	MSP430FR2311	16MHz	3.75KB	1KB
<i>TI LaunchPad MSP-EXP430FR2433LP</i>	<i>TI MSP430</i>	MSP430FR2433	8MHz	15KB	4KB
<i>TI LaunchPad MSP-EXP430FR4133LP</i>	<i>TI MSP430</i>	MSP430FR4133	8MHz	15KB	2KB
<i>TI LaunchPad MSP-EXP430FR5969LP</i>	<i>TI MSP430</i>	MSP430FR5969	8MHz	47KB	2KB
<i>TI LaunchPad MSP-EXP430FR5994LP</i>	<i>TI MSP430</i>	MSP430FR5994	16MHz	256KB	4KB
<i>TI LaunchPad MSP-EXP430FR6989LP</i>	<i>TI MSP430</i>	MSP430FR6989	8MHz	47KB	2KB
<i>TI LaunchPad MSP-EXP430G2 w/ MSP430G2231</i>	<i>TI MSP430</i>	MSP430G2231	1MHz	2KB	256KB
<i>TI LaunchPad MSP-EXP430G2 w/ MSP430G2452</i>	<i>TI MSP430</i>	MSP430G2452	16MHz	8KB	256KB
<i>TI LaunchPad MSP-EXP430G2553LP</i>	<i>TI MSP430</i>	MSP430G2553	16MHz	16KB	512KB
<i>XMC1100 Boot Kit</i>	<i>Infineon XMC</i>	XMC1100	32MHz	64KB	16KB
<i>XMC1100 H-Bridge 2Go</i>	<i>Infineon XMC</i>	XMC1100	32MHz	64KB	16KB
<i>XMC1100 XMC2Go</i>	<i>Infineon XMC</i>	XMC1100	32MHz	64KB	16KB
<i>XMC1300 Boot Kit</i>	<i>Infineon XMC</i>	XMC1300	32MHz	64KB	16KB
<i>XMC1300 Sense2GoL</i>	<i>Infineon XMC</i>	XMC1300	32MHz	32KB	16KB
<i>XMC1400 Boot Kit</i>	<i>Infineon XMC</i>	XMC1400	48MHz	1.95MB	16KB
<i>XMC4200 Distance2Go</i>	<i>Infineon XMC</i>	XMC4200	80MHz	256KB	40KB
<i>XMC4700 Relax Kit</i>	<i>Infineon XMC</i>	XMC4700	144MHz	2.00MB	1.95MB
<i>u-blox EVK-NINA-B1</i>	<i>Nordic nRF52</i>	NRF52832	64MHz	512KB	64KB

External Debug Tools

Banks 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.

Name	Platform	MCU	Frequency	Flash	RAM
<i>3D Printer Controller</i>	<i>ST STM32</i>	STM32F407VET6	168MHz	512KB	192KB
<i>3D Printer control board</i>	<i>ST STM32</i>	STM32F446RET6	180MHz	512KB	128KB
<i>AI Thinker ESP32-CAM</i>	<i>Espressif 32</i>	ESP32	240MHz	4MB	320KB
<i>ALKS ESP32</i>	<i>Espressif 32</i>	ESP32	240MHz	4MB	320KB
<i>Adafruit Circuit Playground Express</i>	<i>Atmel SAM</i>	SAMD21G18A	48MHz	256KB	32KB
<i>Adafruit Crickit M0</i>	<i>Atmel SAM</i>	SAMD21G18A	48MHz	256KB	32KB
<i>Adafruit ESP32 Feather</i>	<i>Espressif 32</i>	ESP32	240MHz	4MB	320KB
<i>Adafruit Feather M0</i>	<i>Atmel SAM</i>	SAMD21G18A	48MHz	256KB	32KB
<i>Adafruit Feather M0 Express</i>	<i>Atmel SAM</i>	SAMD21G18A	48MHz	256KB	32KB
<i>Adafruit Feather M4 Express</i>	<i>Atmel SAM</i>	SAMD51J19A	120MHz	512KB	192KB
<i>Adafruit Gemma M0</i>	<i>Atmel SAM</i>	SAMD21E18A	48MHz	256KB	32KB
<i>Adafruit Grand Central M4</i>	<i>Atmel SAM</i>	SAMD51P20A	120MHz	1MB	256KB
<i>Adafruit Hallowing M0</i>	<i>Atmel SAM</i>	SAMD21G18A	48MHz	256KB	32KB
<i>Adafruit Hallowing M4</i>	<i>Atmel SAM</i>	SAMD51J19A	120MHz	496KB	192KB

Continued on next page

Table 14 – continued from previous page

Name	Platform	MCU	Frequency	Flash	RAM
<i>Adafruit ItsyBitsy M0</i>	<i>Atmel SAM</i>	SAMD21G18A	48MHz	256KB	32KB
<i>Adafruit ItsyBitsy M4</i>	<i>Atmel SAM</i>	SAMD51G19A	120MHz	512KB	192KB
<i>Adafruit MONSTER M4SK</i>	<i>Atmel SAM</i>	SAMD51J19A	120MHz	496KB	192KB
<i>Adafruit Metro M0 Expresss</i>	<i>Atmel SAM</i>	SAMD21G18A	48MHz	256KB	32KB
<i>Adafruit Metro M4</i>	<i>Atmel SAM</i>	SAMD51J19A	120MHz	512KB	192KB
<i>Adafruit Metro M4 AirLift Lite</i>	<i>Atmel SAM</i>	SAMD51J19A	120MHz	512KB	192KB
<i>Adafruit PyGamer Advance M4</i>	<i>Atmel SAM</i>	SAMD51J20A	120MHz	1MB	256KB
<i>Adafruit PyGamer M4 Express</i>	<i>Atmel SAM</i>	SAMD51J19A	120MHz	512KB	192KB
<i>Adafruit PyPortal M4</i>	<i>Atmel SAM</i>	SAMD51J20A	120MHz	1MB	256KB
<i>Adafruit Trellis M4</i>	<i>Atmel SAM</i>	SAMD51J19A	120MHz	512KB	192KB
<i>Adafruit Trinket M0</i>	<i>Atmel SAM</i>	SAMD21E18A	48MHz	256KB	32KB
<i>Adafruit pIRkey</i>	<i>Atmel SAM</i>	SAMD21E18A	48MHz	256KB	32KB
<i>Adafruit pyBadge AirLift M4</i>	<i>Atmel SAM</i>	SAMD51J20A	120MHz	1008KB	192KB
<i>Adafruit pyBadge M4 Express</i>	<i>Atmel SAM</i>	SAMD51J19A	120MHz	512KB	192KB
<i>Arduino Due (Programming Port)</i>	<i>Atmel SAM</i>	AT91SAM3X8E	84MHz	512KB	96KB
<i>Arduino Due (USB Native Port)</i>	<i>Atmel SAM</i>	AT91SAM3X8E	84MHz	512KB	96KB
<i>Arduino M0</i>	<i>Atmel SAM</i>	SAMD21G18A	48MHz	256KB	32KB
<i>Arduino M0 Pro (Native USB Port)</i>	<i>Atmel SAM</i>	SAMD21G18A	48MHz	256KB	32KB
<i>Arduino MKR FOX 1200</i>	<i>Atmel SAM</i>	SAMD21G18A	48MHz	256KB	32KB
<i>Arduino MKR GSM 1400</i>	<i>Atmel SAM</i>	SAMD21G18A	48MHz	256KB	32KB
<i>Arduino MKR NB 1500</i>	<i>Atmel SAM</i>	SAMD21G18A	48MHz	256KB	32KB
<i>Arduino MKR WAN 1300</i>	<i>Atmel SAM</i>	SAMD21G18A	48MHz	256KB	32KB
<i>Arduino MKR WiFi 1010</i>	<i>Atmel SAM</i>	SAMD21G18A	48MHz	256KB	32KB
<i>Arduino MKR1000</i>	<i>Atmel SAM</i>	SAMD21G18A	48MHz	256KB	32KB
<i>Arduino MKRZERO</i>	<i>Atmel SAM</i>	SAMD21G18A	48MHz	256KB	32KB
<i>Arduino Tian</i>	<i>Atmel SAM</i>	SAMD21G18A	48MHz	256KB	32KB
<i>Arduino Zero (USB Native Port)</i>	<i>Atmel SAM</i>	SAMD21G18A	48MHz	256KB	32KB
<i>Black STM32F407VE</i>	<i>ST STM32</i>	STM32F407VET6	168MHz	512KB	128KB
<i>Black STM32F407VG</i>	<i>ST STM32</i>	STM32F407VGT6	168MHz	512KB	128KB
<i>Black STM32F407ZE</i>	<i>ST STM32</i>	STM32F407ZET6	168MHz	512KB	128KB
<i>Black STM32F407ZE</i>	<i>ST STM32</i>	STM32F407ZGT6	168MHz	1MB	128KB
<i>BlackPill F103C8</i>	<i>ST STM32</i>	STM32F103C8T6	72MHz	64KB	20KB
<i>BlackPill F103C8 (128k)</i>	<i>ST STM32</i>	STM32F103C8T6	72MHz	128KB	20KB
<i>BlackPill F303CC</i>	<i>ST STM32</i>	STM32F303CCT6	72MHz	256KB	40KB
<i>Blue STM32F407VE Mini</i>	<i>ST STM32</i>	STM32F407VET6	168MHz	512KB	128KB
<i>BluePill F103C6</i>	<i>ST STM32</i>	STM32F103C6T6	72MHz	32KB	10KB
<i>BluePill F103C8</i>	<i>ST STM32</i>	STM32F103C8T6	72MHz	64KB	20KB
<i>BluePill F103C8 (128k)</i>	<i>ST STM32</i>	STM32F103C8T6	72MHz	128KB	20KB
<i>Bluey nRF52832 IoT</i>	<i>Nordic nRF52</i>	NRF52832	64MHz	512KB	64KB
<i>BluzDK</i>	<i>Nordic nRF51</i>	NRF51822	32MHz	256KB	32KB
<i>Circuit Playground Bluefruit</i>	<i>Nordic nRF52</i>	NRF52840	64MHz	796KB	243KB
<i>D-duino-32</i>	<i>Espressif 32</i>	ESP32	240MHz	4MB	320KB
<i>DOIT ESP32 DEVKIT V1</i>	<i>Espressif 32</i>	ESP32	240MHz	4MB	320KB
<i>Demo F030F4</i>	<i>ST STM32</i>	STM32F030F4P6	48MHz	16KB	4KB
<i>Digistump DigiX</i>	<i>Atmel SAM</i>	AT91SAM3X8E	84MHz	512KB	96KB
<i>Dongsen Tech Pocket 32</i>	<i>Espressif 32</i>	ESP32	240MHz	4MB	320KB
<i>ESP32 FM DevKit</i>	<i>Espressif 32</i>	ESP32	240MHz	4MB	320KB
<i>ESP32vn IoT Uno</i>	<i>Espressif 32</i>	ESP32	240MHz	4MB	320KB
<i>ESPectro32</i>	<i>Espressif 32</i>	ESP32	240MHz	4MB	320KB

Continued on next page

Table 14 – continued from previous page

Name	Platform	MCU	Frequency	Flash	RAM
<i>ESPino32</i>	<i>Espressif 32</i>	ESP32	240MHz	4MB	320KB
<i>Espressif ESP32 Dev Module</i>	<i>Espressif 32</i>	ESP32	240MHz	4MB	320KB
<i>F407VG</i>	<i>ST STM32</i>	STM32F407VGT6	168MHz	512KB	128KB
<i>FK407M1</i>	<i>ST STM32</i>	STM32F407VET6	168MHz	512KB	128KB
<i>FireBeetle-ESP32</i>	<i>Espressif 32</i>	ESP32	240MHz	4MB	320KB
<i>Frog Board ESP32</i>	<i>Espressif 32</i>	ESP32	240MHz	4MB	320KB
<i>GD32VF103V-EVAL</i>	<i>GigaDevice GD32V</i>	GD32VF103VBT6	108MHz	128KB	32KB
<i>Heltec WiFi LoRa 32</i>	<i>Espressif 32</i>	ESP32	240MHz	4MB	320KB
<i>Heltec WiFi LoRa 32 (V2)</i>	<i>Espressif 32</i>	ESP32	240MHz	8MB	320KB
<i>Heltec Wireless Stick</i>	<i>Espressif 32</i>	ESP32	240MHz	8MB	320KB
<i>Hornbill ESP32 Dev</i>	<i>Espressif 32</i>	ESP32	240MHz	4MB	320KB
<i>Hornbill ESP32 Minima</i>	<i>Espressif 32</i>	ESP32	240MHz	4MB	320KB
<i>IoTaaP Magnolia</i>	<i>Espressif 32</i>	ESP32	240MHz	4MB	320KB
<i>M200 V2</i>	<i>ST STM32</i>	STM32F070CBT6	48MHz	120KB	14.81KB
<i>MH ET LIVE ESP32DevKIT</i>	<i>Espressif 32</i>	ESP32	240MHz	4MB	320KB
<i>MH ET LIVE ESP32MiniKit</i>	<i>Espressif 32</i>	ESP32	240MHz	4MB	320KB
<i>MKR Vidor 4000</i>	<i>Atmel SAM</i>	SAMD21G18A	48MHz	256KB	32KB
<i>Macchina M2</i>	<i>Atmel SAM</i>	AT91SAM3X8E	84MHz	512KB	96KB
<i>Malyan M200 VI</i>	<i>ST STM32</i>	STM32F103CBT6	72MHz	120KB	20KB
<i>Maple</i>	<i>ST STM32</i>	STM32F103RBT6	72MHz	108KB	17KB
<i>Maple (RET6)</i>	<i>ST STM32</i>	STM32F103RET6	72MHz	256KB	48KB
<i>Maple Mini Bootloader 2.0</i>	<i>ST STM32</i>	STM32F103CBT6	72MHz	120KB	20KB
<i>Maple Mini Original</i>	<i>ST STM32</i>	STM32F103CBT6	72MHz	108KB	17KB
<i>Microduino Core STM32 to Flash</i>	<i>ST STM32</i>	STM32F103CBT6	72MHz	105.47KB	16.60KB
<i>Minitronics v2.0</i>	<i>Atmel SAM</i>	SAMD21J18A	48MHz	256KB	32KB
<i>Moteino M0</i>	<i>Atmel SAM</i>	SAMD21G18A	48MHz	256KB	32KB
<i>N2+</i>	<i>ST STM32</i>	STM32F405RGT6	168MHz	1MB	192KB
<i>NANO 33 IoT</i>	<i>Atmel SAM</i>	SAMD21G18A	48MHz	256KB	32KB
<i>Node32s</i>	<i>Espressif 32</i>	ESP32	240MHz	4MB	320KB
<i>NodeMCU-32S</i>	<i>Espressif 32</i>	ESP32	240MHz	4MB	320KB
<i>OLIMEX ESP32-DevKit-LiPo</i>	<i>Espressif 32</i>	ESP32	240MHz	4MB	320KB
<i>OLIMEX ESP32-EVB</i>	<i>Espressif 32</i>	ESP32	240MHz	4MB	320KB
<i>OLIMEX ESP32-GATEWAY</i>	<i>Espressif 32</i>	ESP32	240MHz	4MB	320KB
<i>OSHChip</i>	<i>Nordic nRF51</i>	NRF51822	32MHz	256KB	32KB
<i>Pycom LoPy</i>	<i>Espressif 32</i>	ESP32	240MHz	4MB	320KB
<i>Pycom LoPy4</i>	<i>Espressif 32</i>	ESP32	240MHz	4MB	1.25MB
<i>RAK811 LoRa Tracker</i>	<i>ST STM32</i>	STM32L151RBT6	32MHz	128KB	16KB
<i>RAK811 LoRa Tracker</i>	<i>ST STM32</i>	STM32L151RBT6	32MHz	128KB	32KB
<i>SODAQ Autonomo</i>	<i>Atmel SAM</i>	SAMD21J18A	48MHz	256KB	32KB
<i>SODAQ ExpLoRer</i>	<i>Atmel SAM</i>	SAMD21J18A	48MHz	256KB	32KB
<i>SODAQ ONE</i>	<i>Atmel SAM</i>	SAMD21G18A	48MHz	256KB	32KB
<i>SODAQ SARA</i>	<i>Atmel SAM</i>	SAMD21J18A	48MHz	256KB	32KB
<i>SODAQ SFF</i>	<i>Atmel SAM</i>	SAMD21G18A	48MHz	256KB	32KB
<i>STM32F103C8 (20k RAM, 64k Flash)</i>	<i>ST STM32</i>	STM32F103C8T6	72MHz	64KB	20KB
<i>STM32F103CB (20k RAM, 128k Flash)</i>	<i>ST STM32</i>	STM32F103CBT6	72MHz	128KB	20KB
<i>STM32F103R8 (20k RAM, 64 Flash)</i>	<i>ST STM32</i>	STM32F103R8T6	72MHz	64KB	20KB
<i>STM32F103RB (20k RAM, 128k Flash)</i>	<i>ST STM32</i>	STM32F103RBT6	72MHz	128KB	20KB
<i>STM32F103RC (48k RAM, 256k Flash)</i>	<i>ST STM32</i>	STM32F103RCT6	72MHz	256KB	48KB
<i>STM32F103RE (64k RAM, 512k Flash)</i>	<i>ST STM32</i>	STM32F103RET6	72MHz	512KB	64KB

Continued on next page

Table 14 – continued from previous page

Name	Platform	MCU	Frequency	Flash	RAM
<i>STM32F103T8 (20k RAM, 64k Flash)</i>	<i>ST STM32</i>	STM32F103T8T6	72MHz	20KB	64KB
<i>STM32F103TB (20k RAM, 128k Flash)</i>	<i>ST STM32</i>	STM32F103TBT6	72MHz	128KB	20KB
<i>STM32F103VB (20k RAM, 128k Flash)</i>	<i>ST STM32</i>	STM32F103VBT6	72MHz	128KB	20KB
<i>STM32F103VC (48k RAM, 256k Flash)</i>	<i>ST STM32</i>	STM32F103VCT6	72MHz	256KB	48KB
<i>STM32F103VD (64k RAM, 384k Flash)</i>	<i>ST STM32</i>	STM32F103VDT6	72MHz	384KB	64KB
<i>STM32F103VE (64k RAM, 512k Flash)</i>	<i>ST STM32</i>	STM32F103VET6	72MHz	512KB	64KB
<i>STM32F103ZC (48k RAM, 256k Flash)</i>	<i>ST STM32</i>	STM32F103ZCT6	72MHz	256KB	48KB
<i>STM32F103ZD (64k RAM, 384k Flash)</i>	<i>ST STM32</i>	STM32F103ZDT6	72MHz	384KB	64KB
<i>STM32F103ZE (64k RAM, 512k Flash)</i>	<i>ST STM32</i>	STM32F103ZET6	72MHz	512KB	64KB
<i>STM32F407VE (192k RAM, 512k Flash)</i>	<i>ST STM32</i>	STM32F407VET6	168MHz	502.23KB	128KB
<i>STM32F407VG (192k RAM, 1024k Flash)</i>	<i>ST STM32</i>	STM32F407VGT6	168MHz	1MB	192KB
<i>STM32F4Stamp F405</i>	<i>ST STM32</i>	STM32F405RGT6	168MHz	1MB	192KB
<i>SainSmart Due (Programming Port)</i>	<i>Atmel SAM</i>	AT91SAM3X8E	84MHz	512KB	96KB
<i>SainSmart Due (USB Native Port)</i>	<i>Atmel SAM</i>	AT91SAM3X8E	84MHz	512KB	96KB
<i>Seeeduino LoRaWAN</i>	<i>Atmel SAM</i>	SAMD21G18A	48MHz	256KB	32KB
<i>Silicognition wESP32</i>	<i>Espressif 32</i>	ESP32	240MHz	4MB	320KB
<i>Sino:Bit</i>	<i>Nordic nRF51</i>	NRF51822	32MHz	256KB	32KB
<i>Sipeed Longan Nano</i>	<i>GigaDevice GD32V</i>	GD32VF103CBT6	108MHz	128KB	32KB
<i>Sipeed MAIX BiT</i>	<i>Kendryte K210</i>	K210	400MHz	16MB	6MB
<i>Sipeed MAIX BiT with Mic</i>	<i>Kendryte K210</i>	K210	400MHz	16MB	6MB
<i>Sipeed MAIX GO</i>	<i>Kendryte K210</i>	K210	400MHz	16MB	6MB
<i>Sipeed MAIX ONE DOCK</i>	<i>Kendryte K210</i>	K210	400MHz	16MB	6MB
<i>Sipeed MAIXDUINO</i>	<i>Kendryte K210</i>	K210	400MHz	16MB	6MB
<i>SparkFun ESP32 Thing</i>	<i>Espressif 32</i>	ESP32	240MHz	4MB	320KB
<i>SparkFun LoRa Gateway 1-Channel</i>	<i>Espressif 32</i>	ESP32	240MHz	4MB	320KB
<i>SparkFun SAMD21 Dev Breakout</i>	<i>Atmel SAM</i>	SAMD21G18A	48MHz	256KB	32KB
<i>SparkFun SAMD21 Mini Breakout</i>	<i>Atmel SAM</i>	SAMD21G18A	48MHz	256KB	32KB
<i>Sparky V1 F303</i>	<i>ST STM32</i>	STM32F303CCT6	72MHz	256KB	40KB
<i>TTGO LoRa32-OLED V1</i>	<i>Espressif 32</i>	ESP32	240MHz	4MB	320KB
<i>TTGO T-Beam</i>	<i>Espressif 32</i>	ESP32	240MHz	4MB	1.25MB
<i>TTGO T1</i>	<i>Espressif 32</i>	ESP32	240MHz	4MB	320KB
<i>Taida Century nRF52 mini board</i>	<i>Nordic nRF52</i>	NRF52832	64MHz	512KB	64KB
<i>Teensy 3.1 / 3.2</i>	<i>Teensy</i>	MK20DX256	72MHz	256KB	64KB
<i>Teensy 3.5</i>	<i>Teensy</i>	MK64FX512	120MHz	512KB	255.99KB
<i>Teensy 3.6</i>	<i>Teensy</i>	MK66FX1M0	180MHz	1MB	256KB
<i>Teensy 4.0</i>	<i>Teensy</i>	IMXRT1062	600MHz	1.94MB	1MB
<i>Teensy LC</i>	<i>Teensy</i>	MKL26Z64	48MHz	62KB	8KB
<i>Tiny STM103T</i>	<i>ST STM32</i>	STM32F103TBU6	72MHz	128KB	20KB
<i>Tuino 096</i>	<i>Atmel SAM</i>	SAMD21G18A	48MHz	256KB	32KB
<i>VAKe v1.0</i>	<i>ST STM32</i>	STM32F446RET6	180MHz	512KB	128KB
<i>VintLabs ESP32 Devkit</i>	<i>Espressif 32</i>	ESP32	240MHz	4MB	320KB
<i>WEMOS LOLIN D32</i>	<i>Espressif 32</i>	ESP32	240MHz	4MB	320KB
<i>WEMOS LOLIN D32 PRO</i>	<i>Espressif 32</i>	ESP32	240MHz	4MB	320KB
<i>WEMOS LOLIN32</i>	<i>Espressif 32</i>	ESP32	240MHz	4MB	320KB
<i>Waveshare BLE400</i>	<i>Nordic nRF51</i>	NRF51822	32MHz	256KB	32KB
<i>WeMos D1 MINI ESP32</i>	<i>Espressif 32</i>	ESP32	240MHz	4MB	320KB
<i>WeMos WiFi and Bluetooth Battery</i>	<i>Espressif 32</i>	ESP32	240MHz	4MB	320KB
<i>Wio Lite RISC-V</i>	<i>GigaDevice GD32V</i>	GD32VF103CBT6	108MHz	128KB	32KB
<i>Xenon</i>	<i>Nordic nRF52</i>	NRF52840	64MHz	796KB	243KB

Continued on next page

Table 14 – continued from previous page

Name	Platform	MCU	Frequency	Flash	RAM
<i>XinaBox CW02</i>	<i>Espressif 32</i>	ESP32	240MHz	4MB	320KB
<i>hackaBLE</i>	<i>Nordic nRF52</i>	NRF52832	64MHz	512KB	64KB
<i>ng-beacon</i>	<i>Nordic nRF51</i>	NRF51822	16MHz	256KB	32KB
<i>oddWires IoT-Bus Io</i>	<i>Espressif 32</i>	ESP32	240MHz	4MB	320KB
<i>oddWires IoT-Bus Proteus</i>	<i>Espressif 32</i>	ESP32	240MHz	4MB	320KB

Examples

- Arduino for Atmel AVR
- Arduino for Atmel SAM
- Arduino for Espressif 32
- Arduino for Espressif 8266
- Arduino for GigaDevice GD32V
- Arduino for Infineon XMC
- Arduino for Intel ARC32
- Arduino for Kendryte K210
- Arduino for Microchip PIC32
- Arduino for Nordic nRF51
- Arduino for Nordic nRF52
- Arduino for ST STM32
- Arduino for ST STM8
- Arduino for Teensy
- Arduino for TI MSP430
- Arduino for TI TIVA

Platforms

Name	Description
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.
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.
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.
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.
GigaDevice GD32V	The GigaDevice GD32V device is a 32-bit general-purpose microcontroller based on the RISC-V core with an impressive balance of processing power, reduced power consumption and peripheral set.
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
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.
Kendryte K210	Kendryte K210 is an AI capable RISCV64 dual core SoC.
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!
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.
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.
ST STM32	The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.
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.
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.
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.
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.

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

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>4D Systems gen4 IoD Range</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	512KB	80KB

4DSystems

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>4DSystems PICadillo 35T</i>	<i>Microchip PIC32</i>	No	32MX795F512L	80MHz	508KB	128KB

AI Thinker

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>AI Thinker ESP32-CAM</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB

Adafruit

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Adafruit Bluefruit Micro</i>	<i>Atmel AVR</i>	No	ATMEGA32U4	8MHz	28KB	2.50KB
<i>Adafruit Bluefruit nRF52832 Feather</i>	<i>Nordic nRF52</i>	On-board	NRF52832	64MHz	512KB	64KB
<i>Adafruit Circuit Playground Classic</i>	<i>Atmel AVR</i>	No	ATMEGA32U4	8MHz	28KB	2.50KB
<i>Adafruit Circuit Playground Express</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz	256KB	32KB
<i>Adafruit Crickit M0</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz	256KB	32KB
<i>Adafruit ESP32 Feather</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>Adafruit Feather 328P</i>	<i>Atmel AVR</i>	No	ATMEGA328P	8MHz	31.50KB	2KB
<i>Adafruit Feather 32u4</i>	<i>Atmel AVR</i>	No	ATMEGA32U4	8MHz	28KB	2.50KB
<i>Adafruit Feather M0</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz	256KB	32KB
<i>Adafruit Feather M0 Express</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz	256KB	32KB
<i>Adafruit Feather M4 Express</i>	<i>Atmel SAM</i>	External	SAMD51J19A	120MHz	512KB	192KB
<i>Adafruit Feather nRF52840 Express</i>	<i>Nordic nRF52</i>	On-board	NRF52840	64MHz	796KB	243KB
<i>Adafruit Flora</i>	<i>Atmel AVR</i>	No	ATMEGA32U4	8MHz	28KB	2.50KB
<i>Adafruit Gemma</i>	<i>Atmel AVR</i>	No	ATTINY85	8MHz	8KB	512B
<i>Adafruit Gemma M0</i>	<i>Atmel SAM</i>	External	SAMD21E18A	48MHz	256KB	32KB
<i>Adafruit Grand Central M4</i>	<i>Atmel SAM</i>	External	SAMD51P20A	120MHz	1MB	256KB
<i>Adafruit HUZZAH ESP8266</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	4MB	80KB

Continued on next page

Table 15 – continued from previous page

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Adafruit Hallowing M0</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz	256KB	32KB
<i>Adafruit Hallowing M4</i>	<i>Atmel SAM</i>	External	SAMD51J19A	120MHz	496KB	192KB
<i>Adafruit ItsyBitsy 3V/8MHz</i>	<i>Atmel AVR</i>	No	ATMEGA32U4	8MHz	28KB	2.50KB
<i>Adafruit ItsyBitsy 5V/16MHz</i>	<i>Atmel AVR</i>	No	ATMEGA32U4	16MHz	28KB	2.50KB
<i>Adafruit ItsyBitsy M0</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz	256KB	32KB
<i>Adafruit ItsyBitsy M4</i>	<i>Atmel SAM</i>	External	SAMD51J19A	120MHz	512KB	192KB
<i>Adafruit MONSTER M4SK</i>	<i>Atmel SAM</i>	External	SAMD51J19A	120MHz	496KB	192KB
<i>Adafruit Metro</i>	<i>Atmel AVR</i>	No	ATMEGA328P	16MHz	31.50KB	2KB
<i>Adafruit Metro M0 Expresss</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz	256KB	32KB
<i>Adafruit Metro M4</i>	<i>Atmel SAM</i>	External	SAMD51J19A	120MHz	512KB	192KB
<i>Adafruit Metro M4 AirLift Lite</i>	<i>Atmel SAM</i>	External	SAMD51J19A	120MHz	512KB	192KB
<i>Adafruit Pro Trinket 3V/12MHz (FTDI)</i>	<i>Atmel AVR</i>	No	ATMEGA328P	12MHz	28KB	2KB
<i>Adafruit Pro Trinket 3V/12MHz (USB)</i>	<i>Atmel AVR</i>	No	ATMEGA328P	12MHz	28KB	2KB
<i>Adafruit Pro Trinket 5V/16MHz (FTDI)</i>	<i>Atmel AVR</i>	No	ATMEGA328P	16MHz	28KB	2KB
<i>Adafruit Pro Trinket 5V/16MHz (USB)</i>	<i>Atmel AVR</i>	No	ATMEGA328P	16MHz	28KB	2KB
<i>Adafruit PyGamer Advance M4</i>	<i>Atmel SAM</i>	External	SAMD51J20A	120MHz	1MB	256KB
<i>Adafruit PyGamer M4 Express</i>	<i>Atmel SAM</i>	External	SAMD51J19A	120MHz	512KB	192KB
<i>Adafruit PyPortal M4</i>	<i>Atmel SAM</i>	External	SAMD51J20A	120MHz	1MB	256KB
<i>Adafruit Trellis M4</i>	<i>Atmel SAM</i>	External	SAMD51J19A	120MHz	512KB	192KB
<i>Adafruit Trinket 3V/8MHz</i>	<i>Atmel AVR</i>	No	ATTINY85	8MHz	8KB	512B
<i>Adafruit Trinket 5V/16MHz</i>	<i>Atmel AVR</i>	No	ATTINY85	16MHz	8KB	512B
<i>Adafruit Trinket M0</i>	<i>Atmel SAM</i>	External	SAMD21E18A	48MHz	256KB	32KB
<i>Adafruit pIRkey</i>	<i>Atmel SAM</i>	External	SAMD21E18A	48MHz	256KB	32KB
<i>Adafruit pyBadge AirLift M4</i>	<i>Atmel SAM</i>	External	SAMD51J20A	120MHz	1008KB	192KB
<i>Adafruit pyBadge M4 Express</i>	<i>Atmel SAM</i>	External	SAMD51J19A	120MHz	512KB	192KB
<i>Circuit Playground Bluefruit</i>	<i>Nordic nRF52</i>	External	NRF52840	64MHz	796KB	243KB
<i>Metro nRF52840 Express</i>	<i>Nordic nRF52</i>	On-board	NRF52840	64MHz	796KB	243KB

Aiyarafun

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Node32s</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB

Alorium Technology

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Alorium Hinj</i>	<i>Atmel AVR</i>	No	ATMEGA328P	16MHz	31.50KB	2KB
<i>Alorium Sno</i>	<i>Atmel AVR</i>	No	ATMEGA328P	16MHz	31.50KB	2KB
<i>Alorium XLR8</i>	<i>Atmel AVR</i>	No	ATMEGA328P	16MHz	31.50KB	2KB

Amperka

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>WiFi Slot</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	4MB	80KB

Anarduino

Name	Platform	Debug	MCU	Frequency	Flash	RAM
Anarduino MiniWireless	Atmel AVR	No	ATMEGA328P	16MHz	31.50KB	2KB

April Brother

Name	Platform	Debug	MCU	Frequency	Flash	RAM
April Brother ESPea32	Espressif 32	No	ESP32	240MHz	4MB	320KB

Arduboy

Name	Platform	Debug	MCU	Frequency	Flash	RAM
Arduboy	Atmel AVR	No	ATMEGA32U4	16MHz	28KB	2.50KB
Arduboy DevKit	Atmel AVR	No	ATMEGA32U4	16MHz	28KB	2.50KB

Arduino

Name	Platform	Debug	MCU	Frequency	Flash
Arduino BT ATmega168	Atmel AVR	No	ATMEGA168	16MHz	14KB
Arduino BT ATmega328	Atmel AVR	No	ATMEGA328P	16MHz	28KB
Arduino Due (Programming Port)	Atmel SAM	External	AT91SAM3X8E	84MHz	512KB
Arduino Due (USB Native Port)	Atmel SAM	External	AT91SAM3X8E	84MHz	512KB
Arduino Duemilanove or Diecimila ATmega168	Atmel AVR	No	ATMEGA168	16MHz	14KB
Arduino Duemilanove or Diecimila ATmega328	Atmel AVR	No	ATMEGA328P	16MHz	30KB
Arduino Esplora	Atmel AVR	No	ATMEGA32U4	16MHz	28KB
Arduino Ethernet	Atmel AVR	No	ATMEGA328P	16MHz	31.50KB
Arduino Fio	Atmel AVR	No	ATMEGA328P	8MHz	30KB
Arduino Industrial 101	Atmel AVR	No	ATMEGA32U4	16MHz	28KB
Arduino Leonardo	Atmel AVR	No	ATMEGA32U4	16MHz	28KB
Arduino Leonardo ETH	Atmel AVR	No	ATMEGA32U4	16MHz	28KB
Arduino LilyPad ATmega168	Atmel AVR	No	ATMEGA168	8MHz	14KB
Arduino LilyPad ATmega328	Atmel AVR	No	ATMEGA328P	8MHz	30KB
Arduino LilyPad USB	Atmel AVR	No	ATMEGA32U4	8MHz	28KB
Arduino M0	Atmel SAM	External	SAMD21G18A	48MHz	256KB
Arduino M0 Pro (Native USB Port)	Atmel SAM	External	SAMD21G18A	48MHz	256KB
Arduino M0 Pro (Programming/Debug Port)	Atmel SAM	On-board	SAMD21G18A	48MHz	256KB
Arduino MKR FOX 1200	Atmel SAM	External	SAMD21G18A	48MHz	256KB
Arduino MKR GSM 1400	Atmel SAM	External	SAMD21G18A	48MHz	256KB
Arduino MKR NB 1500	Atmel SAM	External	SAMD21G18A	48MHz	256KB
Arduino MKR WAN 1300	Atmel SAM	External	SAMD21G18A	48MHz	256KB
Arduino MKR WiFi 1010	Atmel SAM	External	SAMD21G18A	48MHz	256KB
Arduino MKR1000	Atmel SAM	External	SAMD21G18A	48MHz	256KB
Arduino MKRZERO	Atmel SAM	External	SAMD21G18A	48MHz	256KB
Arduino Mega ADK	Atmel AVR	No	ATMEGA2560	16MHz	248KB

Continued on next page

Table 16 – continued from previous page

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Arduino Mega or Mega 2560 ATmega1280</i>	<i>Atmel AVR</i>	No	ATMEGA1280	16MHz	124KB	128KB
<i>Arduino Mega or Mega 2560 ATmega2560 (Mega 2560)</i>	<i>Atmel AVR</i>	No	ATMEGA2560	16MHz	248KB	256KB
<i>Arduino Micro</i>	<i>Atmel AVR</i>	No	ATMEGA32U4	16MHz	28KB	28KB
<i>Arduino Mini ATmega168</i>	<i>Atmel AVR</i>	No	ATMEGA168	16MHz	14KB	14KB
<i>Arduino Mini ATmega328</i>	<i>Atmel AVR</i>	No	ATMEGA328P	16MHz	28KB	28KB
<i>Arduino NG or older ATmega168</i>	<i>Atmel AVR</i>	No	ATMEGA168	16MHz	14KB	14KB
<i>Arduino NG or older ATmega8</i>	<i>Atmel AVR</i>	No	ATMEGA8	16MHz	7KB	7KB
<i>Arduino Nano ATmega168</i>	<i>Atmel AVR</i>	No	ATMEGA168	16MHz	14KB	14KB
<i>Arduino Nano ATmega328</i>	<i>Atmel AVR</i>	No	ATMEGA328P	16MHz	30KB	28KB
<i>Arduino Nano ATmega328 (New Bootloader)</i>	<i>Atmel AVR</i>	No	ATMEGA328P	16MHz	30KB	28KB
<i>Arduino Pro or Pro Mini ATmega168 (3.3V, 8 MHz)</i>	<i>Atmel AVR</i>	No	ATMEGA168	8MHz	14KB	14KB
<i>Arduino Pro or Pro Mini ATmega168 (5V, 16 MHz)</i>	<i>Atmel AVR</i>	No	ATMEGA168	16MHz	14KB	14KB
<i>Arduino Pro or Pro Mini ATmega328 (3.3V, 8 MHz)</i>	<i>Atmel AVR</i>	No	ATMEGA328P	8MHz	30KB	28KB
<i>Arduino Pro or Pro Mini ATmega328 (5V, 16 MHz)</i>	<i>Atmel AVR</i>	No	ATMEGA328P	16MHz	30KB	28KB
<i>Arduino Robot Control</i>	<i>Atmel AVR</i>	No	ATMEGA32U4	16MHz	28KB	28KB
<i>Arduino Robot Motor</i>	<i>Atmel AVR</i>	No	ATMEGA32U4	16MHz	28KB	28KB
<i>Arduino Tian</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz	256KB	32KB
<i>Arduino Uno</i>	<i>Atmel AVR</i>	No	ATMEGA328P	16MHz	31.50KB	28KB
<i>Arduino Yun</i>	<i>Atmel AVR</i>	No	ATMEGA32U4	16MHz	28KB	28KB
<i>Arduino Yun Mini</i>	<i>Atmel AVR</i>	No	ATMEGA32U4	16MHz	28KB	28KB
<i>Arduino Zero (Programming/Debug Port)</i>	<i>Atmel SAM</i>	On-board	SAMD21G18A	48MHz	256KB	32KB
<i>Arduino Zero (USB Native Port)</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz	256KB	32KB
<i>MKR Vidor 4000</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz	256KB	32KB
<i>NANO 33 IoT</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz	256KB	32KB

Armed

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>3D Printer Controller</i>	<i>ST STM32</i>	External	STM32F407VET6	168MHz	512KB	192KB

Atmel

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Atmel ATSAMW25-XPRO</i>	<i>Atmel SAM</i>	On-board	SAMD21G18A	48MHz	256KB	32KB
<i>Generic ATtiny13</i>	<i>Atmel AVR</i>	No	ATTINY13	1MHz	1KB	64B
<i>Generic ATtiny13A</i>	<i>Atmel AVR</i>	No	ATTINY13A	1MHz	1KB	64B
<i>Generic ATtiny1634</i>	<i>Atmel AVR</i>	No	ATTINY1634	8MHz	16KB	1KB
<i>Generic ATtiny167</i>	<i>Atmel AVR</i>	No	ATTINY167	8MHz	16KB	512B
<i>Generic ATtiny2313</i>	<i>Atmel AVR</i>	No	ATTINY2313	8MHz	2KB	128B
<i>Generic ATtiny24</i>	<i>Atmel AVR</i>	No	ATTINY24	8MHz	2KB	128B
<i>Generic ATtiny25</i>	<i>Atmel AVR</i>	No	ATTINY25	8MHz	2KB	128B
<i>Generic ATtiny261</i>	<i>Atmel AVR</i>	No	ATTINY261	8MHz	2KB	128B
<i>Generic ATtiny4313</i>	<i>Atmel AVR</i>	No	ATTINY4313	8MHz	4KB	256B
<i>Generic ATtiny43U</i>	<i>Atmel AVR</i>	No	ATTINY43U	8MHz	4KB	256B
<i>Generic ATtiny44</i>	<i>Atmel AVR</i>	No	ATTINY44	8MHz	4KB	256B
<i>Generic ATtiny441</i>	<i>Atmel AVR</i>	No	ATTINY441	8MHz	4KB	256B
<i>Generic ATtiny45</i>	<i>Atmel AVR</i>	No	ATTINY45	8MHz	4KB	256B
<i>Generic ATtiny461</i>	<i>Atmel AVR</i>	No	ATTINY461	8MHz	4KB	256B
<i>Generic ATtiny48</i>	<i>Atmel AVR</i>	No	ATTINY48	8MHz	4KB	256B
<i>Generic ATtiny828</i>	<i>Atmel AVR</i>	No	ATTINY828	8MHz	8KB	512B
<i>Generic ATtiny84</i>	<i>Atmel AVR</i>	No	ATTINY84	8MHz	8KB	512B
<i>Generic ATtiny841</i>	<i>Atmel AVR</i>	No	ATTINY841	8MHz	8KB	512B
<i>Generic ATtiny85</i>	<i>Atmel AVR</i>	No	ATTINY85	8MHz	8KB	512B
<i>Generic ATtiny861</i>	<i>Atmel AVR</i>	No	ATTINY861	8MHz	8KB	512B
<i>Generic ATtiny87</i>	<i>Atmel AVR</i>	No	ATTINY87	8MHz	8KB	512B
<i>Generic ATtiny88</i>	<i>Atmel AVR</i>	No	ATTINY88	8MHz	8KB	512B
<i>USBasp stick</i>	<i>Atmel AVR</i>	No	ATMEGA8	12MHz	8KB	1KB

BBC

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>BBC micro:bit</i>	<i>Nordic nRF51</i>	On-board	NRF51822	16MHz	256KB	16KB

BOXTEC

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>HelvePic32</i>	<i>Microchip PIC32</i>	No	32MX250F128B	48MHz	120KB	32KB
<i>HelvePic32</i>	<i>Microchip PIC32</i>	No	32MX250F128B	48MHz	120KB	32KB
<i>HelvePic32</i>	<i>Microchip PIC32</i>	No	32MX250F128D	48MHz	120KB	32KB
<i>HelvePic32 MX270</i>	<i>Microchip PIC32</i>	No	32MX270F256B	48MHz	244KB	62KB
<i>HelvePic32 Robot</i>	<i>Microchip PIC32</i>	No	32MX270F256D	48MHz	244KB	62KB
<i>HelvePic32 SMD MX270</i>	<i>Microchip PIC32</i>	No	32MX270F256D	48MHz	244KB	62KB

BPI Tech

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>BPI-Bit</i>	<i>Espressif 32</i>	No	ESP32	160MHz	4MB	320KB

BQ

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>BQ ZUM BT-328</i>	<i>Atmel AVR</i>	No	ATMEGA328P	16MHz	28KB	2KB

BSFrance

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>LoRa32u4II (868-915MHz)</i>	<i>Atmel AVR</i>	No	ATMEGA32U4	8MHz	28KB	2.50KB

BitWizard

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>BitWizard Raspuino</i>	<i>Atmel AVR</i>	No	ATMEGA328P	16MHz	30KB	2KB

BluzDK

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>BluzDK</i>	<i>Nordic nRF51</i>	External	NRF51822	32MHz	256KB	32KB

Calliope

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Calliope mini</i>	<i>Nordic nRF51</i>	On-board	NRF51822	16MHz	256KB	16KB

ChipKIT

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>RGB Station</i>	<i>Microchip PIC32</i>	No	32MX270F256D	48MHz	240KB	62KB

Controllino

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Controllino Maxi</i>	<i>Atmel AVR</i>	No	ATMEGA2560	16MHz	248KB	8KB
<i>Controllino Maxi Automation</i>	<i>Atmel AVR</i>	No	ATMEGA2560	16MHz	248KB	8KB
<i>Controllino Mega</i>	<i>Atmel AVR</i>	No	ATMEGA2560	16MHz	248KB	8KB
<i>Controllino Mini</i>	<i>Atmel AVR</i>	No	ATMEGA328P	16MHz	31.50KB	2KB

DFRobot

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>FireBeetle-ESP32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB

DOIT

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>DOIT ESP32 DEVKIT V1</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB

DSTIKE

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>D-duino-32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB

Delta

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Delta DFBM-NQ620</i>	<i>Nordic nRF52</i>	On-board	NRF52832	64MHz	512KB	64KB

DigiStamp

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>DigiStamp Oak</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	4MB	80KB

Digilent

Name	Platform	De-bug	MCU	Fre-quency	Flash	RAM
Digilent Cerebot 32MX4	Microchip PIC32	No	32MX460F512L	80MHz	508KB	32KB
Digilent Cerebot 32MX7	Microchip PIC32	No	32MX795F512L	80MHz	508KB	128KB
Digilent OpenScope	Microchip PIC32	No	32MZ2048EFG124	200MHz	1.98MB	512KB
Digilent chipKIT Cmod	Microchip PIC32	No	32MX150F128D	40MHz	124KB	32KB
Digilent chipKIT DP32	Microchip PIC32	No	32MX250F128B	40MHz	120KB	32KB
Digilent chipKIT MAX32	Microchip PIC32	No	32MX795F512L	80MHz	508KB	128KB
Digilent chipKIT MX3	Microchip PIC32	No	32MX320F128H	80MHz	124KB	16KB
Digilent chipKIT Pro MX4	Microchip PIC32	No	32MX460F512L	80MHz	508KB	32KB
Digilent chipKIT Pro MX7	Microchip PIC32	No	32MX795F512L	80MHz	508KB	128KB
Digilent chipKIT UNO32	Microchip PIC32	No	32MX320F128H	80MHz	124KB	16KB
Digilent chipKIT WF32	Microchip PIC32	No	32MX695F512L	80MHz	508KB	128KB
Digilent chipKIT WiFire	Microchip PIC32	No	32MZ2048ECG100	200MHz	1.98MB	512KB
Digilent chipKIT uC32	Microchip PIC32	No	32MX340F512H	80MHz	508KB	32KB
chipKIT WiFire rev. C	Microchip PIC32	No	32MZ2048EFG100	200MHz	1.98MB	512KB

Digistump

Name	Platform	Debug	MCU	Fre-quency	Flash	RAM
Digispark Pro	Atmel AVR	No	ATTINY167	16MHz	14.50KB	512B
Digispark Pro (16 MHz) (64 byte buffer)	Atmel AVR	No	ATTINY167	16MHz	14.50KB	512B
Digispark Pro (32 byte buffer)	Atmel AVR	No	ATTINY167	16MHz	14.50KB	512B
Digispark USB	Atmel AVR	No	ATTINY85	16MHz	5.87KB	512B
Digistump DigiX	Atmel SAM	External	AT91SAM3X8E	84MHz	512KB	96KB

Diymore

Name	Platform	Debug	MCU	Frequency	Flash	RAM
F407VG	ST STM32	External	STM32F407VGT6	168MHz	512KB	128KB

Doit

Name	Platform	Debug	MCU	Frequency	Flash	RAM
ESPDuino (ESP-13 Module)	Espressif 8266	No	ESP8266	80MHz	4MB	80KB

Dongsen Technology

Name	Platform	Debug	MCU	Frequency	Flash	RAM
Dongsen Tech Pocket 32	Espressif 32	External	ESP32	240MHz	4MB	320KB

Dwengo

Name	Platform	Debug	MCU	Frequency	Flash	RAM
Dwenguino	Atmel AVR	No	AT90USB646	16MHz	60KB	2KB

DycodeX

Name	Platform	Debug	MCU	Frequency	Flash	RAM
ESPectro Core	Espressif 8266	No	ESP8266	80MHz	4MB	80KB
ESPectro32	Espressif 32	External	ESP32	240MHz	4MB	320KB

ESP32vn

Name	Platform	Debug	MCU	Frequency	Flash	RAM
ESP32vn IoT Uno	Espressif 32	External	ESP32	240MHz	4MB	320KB

ESpert

Name	Platform	Debug	MCU	Frequency	Flash	RAM
ESPRESSO Lite 1.0	Espressif 8266	No	ESP8266	80MHz	4MB	80KB
ESPRESSO Lite 2.0	Espressif 8266	No	ESP8266	80MHz	4MB	80KB

ESPino

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>ESPino</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	4MB	80KB

Electronic SweetPeas

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Electronic SweetPeas ESP320</i>	<i>Espressif 32</i>	No	ESP32	240MHz	4MB	320KB

Electronut Labs

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Bluey nRF52832 IoT</i>	<i>Nordic nRF52</i>	External	NRF52832	64MHz	512KB	64KB
<i>hackaBLE</i>	<i>Nordic nRF52</i>	External	NRF52832	64MHz	512KB	64KB

Elektor

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Elektor Uno R4</i>	<i>Atmel AVR</i>	No	ATMEGA328PB	16MHz	31.50KB	2KB

Engduino

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Engduino 3</i>	<i>Atmel AVR</i>	No	ATMEGA32U4	8MHz	28KB	2.50KB

EnviroDIY

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>EnviroDIY Mayfly</i>	<i>Atmel AVR</i>	No	ATMEGA1284P	8MHz	127KB	16KB

Espressif

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>ESP-WROOM-02</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	2MB	80KB
<i>ESP32 Pico Kit</i>	<i>Espressif 32</i>	No	ESP32	240MHz	4MB	320KB
<i>Espressif ESP-WROVER-KIT</i>	<i>Espressif 32</i>	On-board	ESP32	240MHz	4MB	320KB
<i>Espressif ESP32 Dev Module</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>Espressif ESP8266 ESP-12E</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	4MB	80KB
<i>Espressif Generic ESP8266 ESP-01 1M</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	1MB	80KB
<i>Espressif Generic ESP8266 ESP-01 512k</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	512KB	80KB
<i>Espressif Generic ESP8266 ESP-07</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	4MB	80KB
<i>Generic ESP8285 Module</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	1MB	80KB
<i>Phoenix 1.0</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	4MB	80KB
<i>Phoenix 2.0</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	4MB	80KB
<i>WifInfo</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	1MB	80KB

FYSETC

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>FYSETC F6 V1.3</i>	<i>Atmel AVR</i>	No	ATMEGA2560	16MHz	252KB	8KB

Fred

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Frog Board ESP32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB

Fubarino

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Fubarino Mini</i>	<i>Microchip PIC32</i>	No	32MX250F128D	48MHz	120KB	32KB
<i>Fubarino SD (1.5)</i>	<i>Microchip PIC32</i>	No	32MX795F512H	80MHz	508KB	128KB
<i>Mini 2.0</i>	<i>Microchip PIC32</i>	No	32MX270F256D	48MHz	240KB	62KB

Generic

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>BlackPill F103C8</i>	<i>ST STM32</i>	External	STM32F103C8T6	72MHz	64KB	20KB
<i>BlackPill F103C8 (128k)</i>	<i>ST STM32</i>	External	STM32F103C8T6	72MHz	128KB	20KB
<i>BluePill F103C6</i>	<i>ST STM32</i>	External	STM32F103C6T6	72MHz	32KB	10KB
<i>BluePill F103C8</i>	<i>ST STM32</i>	External	STM32F103C8T6	72MHz	64KB	20KB
<i>BluePill F103C8 (128k)</i>	<i>ST STM32</i>	External	STM32F103C8T6	72MHz	128KB	20KB
<i>Demo F030F4</i>	<i>ST STM32</i>	External	STM32F030F4P6	48MHz	16KB	4KB
<i>FK407M1</i>	<i>ST STM32</i>	External	STM32F407VET6	168MHz	512KB	128KB
<i>STM32F103C8 (20k RAM. 64k Flash)</i>	<i>ST STM32</i>	External	STM32F103C8T6	72MHz	64KB	20KB
<i>STM32F103CB (20k RAM. 128k Flash)</i>	<i>ST STM32</i>	External	STM32F103CBT6	72MHz	128KB	20KB
<i>STM32F103R8 (20k RAM. 64 Flash)</i>	<i>ST STM32</i>	External	STM32F103R8T6	72MHz	64KB	20KB
<i>STM32F103RB (20k RAM. 128k Flash)</i>	<i>ST STM32</i>	External	STM32F103RBT6	72MHz	128KB	20KB
<i>STM32F103RC (48k RAM. 256k Flash)</i>	<i>ST STM32</i>	External	STM32F103RCT6	72MHz	256KB	48KB
<i>STM32F103RE (64k RAM. 512k Flash)</i>	<i>ST STM32</i>	External	STM32F103RET6	72MHz	512KB	64KB
<i>STM32F103T8 (20k RAM. 64k Flash)</i>	<i>ST STM32</i>	External	STM32F103T8T6	72MHz	20KB	64KB
<i>STM32F103TB (20k RAM. 128k Flash)</i>	<i>ST STM32</i>	External	STM32F103TBT6	72MHz	128KB	20KB
<i>STM32F103VB (20k RAM. 128k Flash)</i>	<i>ST STM32</i>	External	STM32F103VBT6	72MHz	128KB	20KB
<i>STM32F103VC (48k RAM. 256k Flash)</i>	<i>ST STM32</i>	External	STM32F103VCT6	72MHz	256KB	48KB
<i>STM32F103VD (64k RAM. 384k Flash)</i>	<i>ST STM32</i>	External	STM32F103VDT6	72MHz	384KB	64KB
<i>STM32F103VE (64k RAM. 512k Flash)</i>	<i>ST STM32</i>	External	STM32F103VET6	72MHz	512KB	64KB
<i>STM32F103ZC (48k RAM. 256k Flash)</i>	<i>ST STM32</i>	External	STM32F103ZCT6	72MHz	256KB	48KB
<i>STM32F103ZD (64k RAM. 384k Flash)</i>	<i>ST STM32</i>	External	STM32F103ZDT6	72MHz	384KB	64KB
<i>STM32F103ZE (64k RAM. 512k Flash)</i>	<i>ST STM32</i>	External	STM32F103ZET6	72MHz	512KB	64KB
<i>STM32F407VE (192k RAM. 512k Flash)</i>	<i>ST STM32</i>	External	STM32F407VET6	168MHz	502.23KB	128KB
<i>STM32F407VG (192k RAM. 1024k Flash)</i>	<i>ST STM32</i>	External	STM32F407VGT6	168MHz	1MB	192KB
<i>STM32F4Stamp F405</i>	<i>ST STM32</i>	External	STM32F405RGT6	168MHz	1MB	192KB

Gimasi

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Tuino 096</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz	256KB	32KB

HY

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Tiny STM103T</i>	<i>ST STM32</i>	External	STM32F103TBU6	72MHz	128KB	20KB

Hardkernel

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>ODROID-GO</i>	<i>Espressif 32</i>	No	ESP32	240MHz	16MB	320KB

Heltec

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Heltec Wifi kit 8</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	4MB	80KB

Heltec Automation

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Heltec WiFi Kit 32</i>	<i>Espressif 32</i>	No	ESP32	240MHz	4MB	320KB
<i>Heltec WiFi LoRa 32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>Heltec WiFi LoRa 32 (V2)</i>	<i>Espressif 32</i>	External	ESP32	240MHz	8MB	320KB
<i>Heltec Wireless Stick</i>	<i>Espressif 32</i>	External	ESP32	240MHz	8MB	320KB

Hornbill

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Hornbill ESP32 Dev</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>Hornbill ESP32 Minima</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB

Infineon

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>XMC1100 Boot Kit</i>	<i>Infineon XMC</i>	On-board	XMC1100	32MHz	64KB	16KB
<i>XMC1100 H-Bridge 2Go</i>	<i>Infineon XMC</i>	On-board	XMC1100	32MHz	64KB	16KB
<i>XMC1100 XMC2Go</i>	<i>Infineon XMC</i>	On-board	XMC1100	32MHz	64KB	16KB
<i>XMC1300 Boot Kit</i>	<i>Infineon XMC</i>	On-board	XMC1300	32MHz	64KB	16KB
<i>XMC1300 Sense2GoL</i>	<i>Infineon XMC</i>	On-board	XMC1300	32MHz	32KB	16KB
<i>XMC1400 Boot Kit</i>	<i>Infineon XMC</i>	On-board	XMC1400	48MHz	1.95MB	16KB
<i>XMC4200 Distance2Go</i>	<i>Infineon XMC</i>	On-board	XMC4200	80MHz	256KB	40KB
<i>XMC4700 Relax Kit</i>	<i>Infineon XMC</i>	On-board	XMC4700	144MHz	2.00MB	1.95MB

Intel

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Arduino/Genuino 101</i>	<i>Intel ARC32</i>	No	ARCV2EM	32MHz	152KB	80KB

IntoRobot

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>IntoRobot Fig</i>	<i>Espressif 32</i>	No	ESP32	240MHz	4MB	320KB

Invent One

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Invent One</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	4MB	80KB

LeafLabs

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Maple</i>	<i>ST STM32</i>	External	STM32F103RBT6	72MHz	108KB	17KB
<i>Maple (RET6)</i>	<i>ST STM32</i>	External	STM32F103RET6	72MHz	256KB	48KB
<i>Maple Mini Bootloader 2.0</i>	<i>ST STM32</i>	External	STM32F103CBT6	72MHz	120KB	20KB
<i>Maple Mini Original</i>	<i>ST STM32</i>	External	STM32F103CBT6	72MHz	108KB	17KB

LightUp

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>LightUp</i>	<i>Atmel AVR</i>	No	ATMEGA32U4	8MHz	28KB	2.50KB

Linino

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Linino One</i>	<i>Atmel AVR</i>	No	ATMEGA32U4	16MHz	28KB	2.50KB

LowPowerLab

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>LowPowerLab CurrentRanger</i>	<i>Atmel SAM</i>	No	SAMD21G18A	48MHz	256KB	32KB
<i>LowPowerLab MightyHat</i>	<i>Atmel AVR</i>	No	ATMEGA328P	16MHz	31KB	2KB
<i>LowPowerLab Moteino</i>	<i>Atmel AVR</i>	No	ATMEGA328P	16MHz	31.50KB	2KB
<i>LowPowerLab Moteino (8MHz)</i>	<i>Atmel AVR</i>	No	ATMEGA328P	8MHz	31.50KB	2KB
<i>LowPowerLab MoteinoMEGA</i>	<i>Atmel AVR</i>	No	ATMEGA1284P	16MHz	127KB	16KB
<i>Moteino M0</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz	256KB	32KB

M5Stack

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>M5Stack Core ESP32</i>	<i>Espressif 32</i>	No	ESP32	240MHz	4MB	320KB
<i>M5Stack FIRE</i>	<i>Espressif 32</i>	No	ESP32	240MHz	16MB	6.25MB
<i>M5Stack GREY ESP32</i>	<i>Espressif 32</i>	No	ESP32	240MHz	16MB	520KB
<i>M5Stick-C</i>	<i>Espressif 32</i>	No	ESP32	240MHz	4MB	320KB

MH-ET Live

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>MH ET LIVE ESP32DevKIT</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>MH ET LIVE ESP32MiniKit</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB

MVT Solutions

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>IoTaaP Magnolia</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB

MXChip

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Microsoft Azure IoT Development Kit (MX-Chip AZ3166)</i>	<i>ST STM32</i>	On-board	STM32F412ZG	1600MHz	1MB	256KB

Macchina

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Macchina M2</i>	<i>Atmel SAM</i>	External	AT91SAM3X8E	84MHz	512KB	96KB

Magicblocks.io

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>MagicBit</i>	<i>Espressif 32</i>	No	ESP32	240MHz	4MB	320KB

MakerAsia

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>MakerAsia Nano32</i>	<i>Espressif 32</i>	No	ESP32	240MHz	4MB	320KB

Makerology

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>DataStation Mini</i>	<i>Microchip PIC32</i>	No	32MX150F128C	40MHz	120KB	32KB

Malyan

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>M200 V2</i>	<i>ST STM32</i>	External	STM32F070CBT6	48MHz	120KB	14.81KB
<i>Malyan M200 V1</i>	<i>ST STM32</i>	External	STM32F103CBT6	72MHz	120KB	20KB

MediaTek Labs

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>LinkIt Smart 7688 Duo</i>	<i>Atmel AVR</i>	No	ATMEGA32U4	8MHz	28KB	2.50KB

Microchip

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>AT90CAN128</i>	<i>Atmel AVR</i>	No	AT90CAN128	16MHz	127KB	4KB
<i>AT90CAN32</i>	<i>Atmel AVR</i>	No	AT90CAN32	16MHz	31KB	2KB
<i>AT90CAN64</i>	<i>Atmel AVR</i>	No	AT90CAN64	16MHz	63KB	4KB
<i>ATmega128/A</i>	<i>Atmel AVR</i>	No	ATMEGA128	16MHz	127KB	4KB
<i>ATmega1280</i>	<i>Atmel AVR</i>	No	ATMEGA1280	16MHz	127KB	8KB

Continued on next page

Table 17 – continued from previous page

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>ATmega1281</i>	<i>Atmel AVR</i>	No	ATMEGA1281	16MHz	127KB	8KB
<i>ATmega1284</i>	<i>Atmel AVR</i>	No	ATMEGA1284	16MHz	127KB	16KB
<i>ATmega1284P</i>	<i>Atmel AVR</i>	No	ATMEGA1284P	16MHz	127KB	16KB
<i>ATmega16</i>	<i>Atmel AVR</i>	No	ATMEGA16	16MHz	15.50KB	1KB
<i>ATmega164A</i>	<i>Atmel AVR</i>	No	ATMEGA164A	16MHz	15.50KB	1KB
<i>ATmega164P/PA</i>	<i>Atmel AVR</i>	No	ATMEGA164P	16MHz	15.50KB	1KB
<i>ATmega168/A</i>	<i>Atmel AVR</i>	No	ATMEGA168	16MHz	15.50KB	1KB
<i>ATmega168P/PA</i>	<i>Atmel AVR</i>	No	ATMEGA168P	16MHz	15.50KB	1KB
<i>ATmega168PB</i>	<i>Atmel AVR</i>	No	ATMEGA168PB	16MHz	15.50KB	1KB
<i>ATmega2560</i>	<i>Atmel AVR</i>	No	ATMEGA2560	16MHz	255KB	8KB
<i>ATmega2561</i>	<i>Atmel AVR</i>	No	ATMEGA2561	16MHz	255KB	8KB
<i>ATmega32</i>	<i>Atmel AVR</i>	No	ATMEGA32	16MHz	31.50KB	2KB
<i>ATmega324A</i>	<i>Atmel AVR</i>	No	ATMEGA324A	16MHz	31.50KB	2KB
<i>ATmega324P</i>	<i>Atmel AVR</i>	No	ATMEGA324P	16MHz	31.50KB	2KB
<i>ATmega324PA</i>	<i>Atmel AVR</i>	No	ATMEGA324PA	16MHz	31.50KB	2KB
<i>ATmega324PB</i>	<i>Atmel AVR</i>	No	ATMEGA324PB	16MHz	31.50KB	2KB
<i>ATmega328</i>	<i>Atmel AVR</i>	No	ATMEGA328	16MHz	31.50KB	2KB
<i>ATmega328P/PA</i>	<i>Atmel AVR</i>	No	ATMEGA328P	16MHz	31.50KB	2KB
<i>ATmega328PB</i>	<i>Atmel AVR</i>	No	ATMEGA328PB	16MHz	31.50KB	2KB
<i>ATmega48/A</i>	<i>Atmel AVR</i>	No	ATMEGA48	16MHz	4KB	512B
<i>ATmega48PB</i>	<i>Atmel AVR</i>	No	ATMEGA48PB	16MHz	4KB	512B
<i>ATmega64/A</i>	<i>Atmel AVR</i>	No	ATMEGA64	16MHz	63KB	4KB
<i>ATmega640</i>	<i>Atmel AVR</i>	No	ATMEGA640	16MHz	63KB	8KB
<i>ATmega644/A</i>	<i>Atmel AVR</i>	No	ATMEGA644A	16MHz	63KB	4KB
<i>ATmega644P/PA</i>	<i>Atmel AVR</i>	No	ATMEGA644P	16MHz	63KB	4KB
<i>ATmega8/A</i>	<i>Atmel AVR</i>	No	ATMEGA8	16MHz	7.50KB	1KB
<i>ATmega8535</i>	<i>Atmel AVR</i>	No	ATMEGA8535	16MHz	7.50KB	512B
<i>ATmega88/A</i>	<i>Atmel AVR</i>	No	ATMEGA88	16MHz	7.50KB	1KB
<i>ATmega88P/PA</i>	<i>Atmel AVR</i>	No	ATMEGA88P	16MHz	7.50KB	1KB
<i>ATmega88PB</i>	<i>Atmel AVR</i>	No	ATMEGA88PB	16MHz	7.50KB	1KB
<i>ATmega8P/PA</i>	<i>Atmel AVR</i>	No	ATMEGA48P	16MHz	4KB	512B

Micoduino

Name	Platform	De-bug	MCU	Fre-quency	Flash	RAM
<i>Micoduino Core mega168PA@16M,5V)</i>	<i>Atmel AVR</i>	No	AT-MEGA168P	16MHz	15.50KB	1KB
<i>Micoduino Core mega168PA@8M,3.3V)</i>	<i>Atmel AVR</i>	No	AT-MEGA168P	8MHz	15.50KB	1KB
<i>Micoduino Core mega328P@16M,5V)</i>	<i>Atmel AVR</i>	No	AT-MEGA328P	16MHz	31.50KB	2KB
<i>Micoduino Core mega328P@8M,3.3V)</i>	<i>Atmel AVR</i>	No	AT-MEGA328P	8MHz	31.50KB	2KB
<i>Micoduino Core ESP32</i>	<i>Espressif 32</i>	No	ESP32	240MHz	4MB	320KB
<i>Micoduino Core STM32 to Flash</i>	<i>ST STM32</i>	Exter-nal	STM32F103CBT6/2MHz	105.47KB	16.60KB	
<i>Micoduino Core USB (AT-mega32U4@16M,5V)</i>	<i>Atmel AVR</i>	No	AT-MEGA32U4	16MHz	28KB	2.50KB
<i>Micoduino Core+ mega1284P@16M,5V)</i>	<i>Atmel AVR</i>	No	AT-MEGA1284P	16MHz	127KB	16KB
<i>Micoduino Core+ mega1284P@8M,3.3V)</i>	<i>Atmel AVR</i>	No	AT-MEGA1284P	8MHz	127KB	16KB
<i>Micoduino Core+ mega644PA@16M,5V)</i>	<i>Atmel AVR</i>	No	AT-MEGA644P	16MHz	63KB	4KB
<i>Micoduino Core+ mega644PA@8M,3.3V)</i>	<i>Atmel AVR</i>	No	AT-MEGA644P	8MHz	63KB	4KB

MikroElektronika

Name	Platform	De-bug	MCU	Fre-quency	Flash	RAM
<i>MikroElektronika Clicker 2</i>	<i>Microchip PIC32</i>	No	32MX460F512L	80MHz	508KB	32KB
<i>MikroElektronika Flip N Click MZ</i>	<i>Microchip PIC32</i>	No	32MZ2048EFH100	252MHz	1.98MB	512KB

Netduino

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>N2+</i>	<i>ST STM32</i>	External	STM32F405RGT6	168MHz	1MB	192KB

NodeMCU

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>NodeMCU 0.9 (ESP-12 Module)</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	4MB	80KB
<i>NodeMCU 1.0 (ESP-12E Module)</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	4MB	80KB
<i>NodeMCU-32S</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB

Noduino

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Noduino Quantum</i>	<i>Espressif 32</i>	No	ESP32	240MHz	16MB	320KB

Nordic

Name	Platform	Debug	MCU	Fre- quency	Flash	RAM
<i>Nordic Beacon Kit (PCA20006)</i>	<i>Nordic nRF51</i>	On-board	NRF51822	32MHz	256KB	32KB
<i>Nordic nRF51 Dongle (PCA10031)</i>	<i>Nordic nRF51</i>	On-board	NRF51822	32MHz	256KB	32KB
<i>Nordic nRF51X22 Development Kit(PCA1000X)</i>	<i>Nordic nRF51</i>	On-board	NRF51822	32MHz	256KB	32KB
<i>Nordic nRF52-DK</i>	<i>Nordic nRF52</i>	On-board	NRF52832	64MHz	512KB	64KB
<i>Nordic nRF52840-DK</i>	<i>Nordic nRF52</i>	On-board	NRF52840	64MHz	1MB	256KB
<i>Nordic nRF52840-DK (Adafruit BSP)</i>	<i>Nordic nRF52</i>	On-board	NRF52840	64MHz	796KB	243KB

OLIMEX

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>OLIMEX ESP32-DevKit-LiPo</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>OLIMEX ESP32-EVB</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>OLIMEX ESP32-GATEWAY</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>OLIMEX ESP32-PRO</i>	<i>Espressif 32</i>	No	ESP32	240MHz	4MB	320KB
<i>OLIMEX ESP32-PoE</i>	<i>Espressif 32</i>	No	ESP32	240MHz	4MB	320KB
<i>OLIMEX ESP32-PoE-ISO</i>	<i>Espressif 32</i>	No	ESP32	240MHz	4MB	320KB

OROCA

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>OROCA EduBot</i>	<i>Espressif 32</i>	No	ESP32	240MHz	4MB	320KB

OSHChip

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>OSHChip</i>	<i>Nordic nRF51</i>	External	NRF51822	32MHz	256KB	32KB

Olimex

Name	Platform	De-bug	MCU	Fre-quency	Flash	RAM
<i>Olimex MOD-WIFI-ESP8266(-DEV)</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	2MB	80KB
<i>Olimex PIC32-PINGUINO</i>	<i>Microchip PIC32</i>	No	32MX440F256H	80MHz	252KB	32KB

Onehorse

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Onehorse ESP32 Dev Module</i>	<i>Espressif 32</i>	No	ESP32	240MHz	4MB	320KB

OpenBCI

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>OpenBCI 32bit</i>	<i>Microchip PIC32</i>	No	32MX250F128B	40MHz	120KB	32KB

OpenEnergyMonitor

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>OpenEnergyMonitor emonPi</i>	<i>Atmel AVR</i>	No	ATMEGA328P	16MHz	30KB	2KB

PONTECH

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>PONTECH UAV100</i>	<i>Microchip PIC32</i>	No	32MX440F512H	80MHz	508KB	32KB

PanStamp

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>PanStamp AVR</i>	<i>Atmel AVR</i>	No	ATMEGA328P	8MHz	31.50KB	2KB

Particle

Name	Platform	Debug	MCU	Frequency	Flash	RAM
Xenon	Nordic nRF52	External	NRF52840	64MHz	796KB	243KB

Pinoccio

Name	Platform	Debug	MCU	Frequency	Flash	RAM
Pinoccio Scout	Atmel AVR	No	ATMEGA256RFR2	16MHz	248KB	32KB

Pololu Corporation

Name	Platform	Debug	MCU	Frequency	Flash	RAM
Pololu A-Star 32U4	Atmel AVR	No	ATMEGA32U4	16MHz	28KB	2.50KB

Pontech

Name	Platform	Debug	MCU	Frequency	Flash	RAM
Pontech NoFire	Microchip PIC32	No	32MZ2048EFG100	200MHz	1.98MB	512KB
Pontech Quick240	Microchip PIC32	No	32MX795F512L	80MHz	508KB	128KB

Punch Through

Name	Platform	Debug	MCU	Frequency	Flash	RAM
LightBlue Bean	Atmel AVR	No	ATMEGA328P	8MHz	31.50KB	2KB
LightBlue Bean+	Atmel AVR	No	ATMEGA328P	16MHz	31.50KB	2KB

Pycom Ltd.

Name	Platform	Debug	MCU	Frequency	Flash	RAM
Pycom GPy	Espressif 32	No	ESP32	240MHz	4MB	320KB
Pycom LoPy	Espressif 32	External	ESP32	240MHz	4MB	320KB
Pycom LoPy4	Espressif 32	External	ESP32	240MHz	4MB	1.25MB

Quirkbot

Name	Platform	Debug	MCU	Frequency	Flash	RAM
Quirkbot	Atmel AVR	No	ATMEGA32U4	8MHz	28KB	2.50KB

RAK

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>RAK811 LoRa Tracker</i>	<i>ST STM32</i>	External	STM32L151RBT6	32MHz	128KB	16KB
<i>RAK811 LoRa Tracker</i>	<i>ST STM32</i>	External	STM32L151RBT6	32MHz	128KB	32KB

RUMBA

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>3D Printer control board</i>	<i>ST STM32</i>	External	STM32F446RET6	180MHz	512KB	128KB

RedBearLab

Name	Platform	Debug	MCU	Fre- quency	Flash	RAM
<i>RedBearLab BLE Nano 1.5</i>	<i>Nordic nRF51</i>	On-board	NRF51822	16MHz	256KB	32KB
<i>RedBearLab BLE Nano 2</i>	<i>Nordic nRF52</i>	On-board	NRF52832	64MHz	512KB	64KB
<i>RedBearLab Blend</i>	<i>Atmel AVR</i>	No	AT-MEGA32U4	16MHz	28KB	2.50KB
<i>RedBearLab Blend 2</i>	<i>Nordic nRF52</i>	On-board	NRF52832	64MHz	512KB	64KB
<i>RedBearLab Blend Micro 3.3V/16MHz (overclock)</i>	<i>Atmel AVR</i>	No	AT-MEGA32U4	16MHz	28KB	2.50KB
<i>RedBearLab Blend Micro 3.3V/8MHz</i>	<i>Atmel AVR</i>	No	AT-MEGA32U4	8MHz	28KB	2.50KB
<i>RedBearLab nRF51822</i>	<i>Nordic nRF51</i>	On-board	NRF51822	16MHz	256KB	16KB

RemRam

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>3D printer controller</i>	<i>ST STM32</i>	On-board	STM32F765VIT6	216MHz	2MB	512KB

RepRap

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>RepRap RAMBo</i>	<i>Atmel AVR</i>	No	ATMEGA2560	16MHz	252KB	8KB

ReprapWorld

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Minitronics v2.0</i>	<i>Atmel SAM</i>	External	SAMD21J18A	48MHz	256KB	32KB

RobotDyn

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>BlackPill F303CC</i>	<i>ST STM32</i>	External	STM32F303CCT6	72MHz	256KB	40KB

RoboticsBrno

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>ALKS ESP32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB

SODAQ

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>SODAQ Autonomo</i>	<i>Atmel SAM</i>	External	SAMD21J18A	48MHz	256KB	32KB
<i>SODAQ ExpLoRer</i>	<i>Atmel SAM</i>	External	SAMD21J18A	48MHz	256KB	32KB
<i>SODAQ GaLoRa</i>	<i>Atmel AVR</i>	No	ATMEGA1284P	8MHz	127KB	16KB
<i>SODAQ Mbili</i>	<i>Atmel AVR</i>	No	ATMEGA1284P	8MHz	127KB	16KB
<i>SODAQ Moja</i>	<i>Atmel AVR</i>	No	ATMEGA328P	8MHz	31.50KB	2KB
<i>SODAQ Ndogo</i>	<i>Atmel AVR</i>	No	ATMEGA1284P	8MHz	127KB	16KB
<i>SODAQ ONE</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz	256KB	32KB
<i>SODAQ SARA</i>	<i>Atmel SAM</i>	External	SAMD21J18A	48MHz	256KB	32KB
<i>SODAQ SFF</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz	256KB	32KB
<i>SODAQ Tatu</i>	<i>Atmel AVR</i>	No	ATMEGA1284P	8MHz	127KB	16KB

ST

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>3DP001V1 Evaluation board for 3D printer</i>	<i>ST STM32</i>	On-board	STM32F401VGT6	84MHz	512KB	96KB
<i>Black STM32F407VE</i>	<i>ST STM32</i>	External	STM32F407VET6	168MHz	512KB	128KB
<i>Black STM32F407VG</i>	<i>ST STM32</i>	External	STM32F407VGT6	168MHz	512KB	128KB
<i>Black STM32F407ZE</i>	<i>ST STM32</i>	External	STM32F407ZET6	168MHz	512KB	128KB
<i>Black STM32F407ZE</i>	<i>ST STM32</i>	External	STM32F407ZGT6	168MHz	1MB	128KB
<i>Blue STM32F407VE Mini</i>	<i>ST STM32</i>	External	STM32F407VET6	168MHz	512KB	128KB
<i>Nucleo G071RB</i>	<i>ST STM32</i>	On-board	STM32G071RBT6	24MHz	2MB	128KB
<i>P-Nucleo WB55RG</i>	<i>ST STM32</i>	On-board	STM32WB55RG	64MHz	512KB	192.00KB
<i>ST 32F746GDISCOVERY</i>	<i>ST STM32</i>	On-board	STM32F746NGH6	216MHz	1MB	320KB
<i>ST DISCO-L072CZ-LRWAN1</i>	<i>ST STM32</i>	On-board	STM32L072CZ	32MHz	192KB	20KB
<i>ST DISCO-L475VG-IOT01A</i>	<i>ST STM32</i>	On-board	STM32L475VGT6	80MHz	1MB	128KB

Continued on next page

Table 18 – continued from previous page

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>ST Discovery F072RB</i>	<i>ST STM32</i>	On-board	STM32F072RBT6	48MHz	128KB	16KB
<i>ST Nucleo F030R8</i>	<i>ST STM32</i>	On-board	STM32F030R8T6	48MHz	64KB	8KB
<i>ST Nucleo F091RC</i>	<i>ST STM32</i>	On-board	STM32F091RCT6	48MHz	256KB	32KB
<i>ST Nucleo F103RB</i>	<i>ST STM32</i>	On-board	STM32F103RBT6	72MHz	128KB	20KB
<i>ST Nucleo F207ZG</i>	<i>ST STM32</i>	On-board	STM32F207ZGT6	120MHz	1MB	128KB
<i>ST Nucleo F302R8</i>	<i>ST STM32</i>	On-board	STM32F302R8T6	72MHz	64KB	16KB
<i>ST Nucleo F303K8</i>	<i>ST STM32</i>	On-board	STM32F303K8T6	72MHz	64KB	12KB
<i>ST Nucleo F303RE</i>	<i>ST STM32</i>	On-board	STM32F303RET6	72MHz	512KB	64KB
<i>ST Nucleo F401RE</i>	<i>ST STM32</i>	On-board	STM32F401RET6	84MHz	512KB	96KB
<i>ST Nucleo F411RE</i>	<i>ST STM32</i>	On-board	STM32F411RET6	100MHz	512KB	128KB
<i>ST Nucleo F429ZI</i>	<i>ST STM32</i>	On-board	STM32F429ZIT6	180MHz	2MB	256KB
<i>ST Nucleo F446RE</i>	<i>ST STM32</i>	On-board	STM32F446RET6	180MHz	512KB	128KB
<i>ST Nucleo F767ZI</i>	<i>ST STM32</i>	On-board	STM32F767ZIT6	216MHz	2MB	512KB
<i>ST Nucleo H743ZI</i>	<i>ST STM32</i>	On-board	STM32H743ZIT6	400MHz	2MB	1MB
<i>ST Nucleo L031K6</i>	<i>ST STM32</i>	On-board	STM32L031K6T6	32MHz	32KB	8KB
<i>ST Nucleo L053R8</i>	<i>ST STM32</i>	On-board	STM32L053R8T6	32MHz	64KB	8KB
<i>ST Nucleo L073RZ</i>	<i>ST STM32</i>	On-board	STM32L073RZ	32MHz	192KB	20KB
<i>ST Nucleo L152RE</i>	<i>ST STM32</i>	On-board	STM32L152RET6	32MHz	512KB	80KB
<i>ST Nucleo L412KB</i>	<i>ST STM32</i>	On-board	STM32L412KBU6	80MHz	128KB	40KB
<i>ST Nucleo L432KC</i>	<i>ST STM32</i>	On-board	STM32L432KCU6	80MHz	256KB	64KB
<i>ST Nucleo L452RE</i>	<i>ST STM32</i>	On-board	STM32L452RET6	80MHz	256KB	64KB
<i>ST Nucleo L476RG</i>	<i>ST STM32</i>	On-board	STM32L476RGRT6	80MHz	1MB	128KB
<i>ST Nucleo L496ZG</i>	<i>ST STM32</i>	On-board	STM32L496ZGT6	80MHz	1MB	128KB
<i>ST Nucleo L496ZG-P</i>	<i>ST STM32</i>	On-board	STM32L496ZGT6P	80MHz	1MB	320KB
<i>ST Nucleo L4R5ZI</i>	<i>ST STM32</i>	On-board	STM32L4R5ZIT6	120MHz	2MB	640KB
<i>ST STM32F0308DISCOVERY</i>	<i>ST STM32</i>	On-board	STM32F030R8T6	48MHz	64KB	8KB
<i>ST STM32F4DISCOVERY</i>	<i>ST STM32</i>	On-board	STM32F407VGT6	168MHz	1MB	128KB
<i>ST STM32VLDISCOVERY</i>	<i>ST STM32</i>	On-board	STM32F100RBT6	24MHz	128KB	8KB
<i>ST STM8S-DISCOVERY</i>	<i>ST STM8</i>	On-board	STM8S105C6T6	16MHz	32KB	2KB
<i>ST STM8S103F3 Breakout Board</i>	<i>ST STM8</i>	No	STM8S103F3P6	16MHz	8KB	1KB
<i>ST STM8S105K4T6 Breakout Board</i>	<i>ST STM8</i>	No	STM8S105K4T6	16MHz	16KB	2KB

SainSmart

Name	Platform	Debug	MCU	Fre-quency	Flash	RAM
<i>SainSmart Due (Programming Port)</i>	<i>Atmel SAM</i>	External	AT91SAM3X8E	84MHz	512KB	96KB
<i>SainSmart Due (USB Native Port)</i>	<i>Atmel SAM</i>	External	AT91SAM3X8E	84MHz	512KB	96KB

Sanguino

Name	Platform	De-bug	MCU	Fre-quency	Flash	RAM
<i>Sanguino ATmega1284p (16MHz)</i>	<i>Atmel AVR</i>	No	AT-MEGA1284P	16MHz	127KB	16KB
<i>Sanguino ATmega1284p (8MHz)</i>	<i>Atmel AVR</i>	No	AT-MEGA1284P	8MHz	127KB	16KB
<i>Sanguino ATmega644 or ATmega644A (16 MHz)</i>	<i>Atmel AVR</i>	No	ATMEGA644	16MHz	63KB	4KB
<i>Sanguino ATmega644 or ATmega644A (8 MHz)</i>	<i>Atmel AVR</i>	No	ATMEGA644	8MHz	63KB	4KB
<i>Sanguino ATmega644P or ATmega644PA (16 MHz)</i>	<i>Atmel AVR</i>	No	AT-MEGA644P	16MHz	63KB	4KB
<i>Sanguino ATmega644P or ATmega644PA (8 MHz)</i>	<i>Atmel AVR</i>	No	AT-MEGA644P	8MHz	63KB	4KB

Seeed

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Seeeduino LoRaWAN</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz	256KB	32KB

SeeedStudio

Name	Platform	Debug	MCU	Fre-quency	Flash	RAM
<i>Seeed Tiny BLE</i>	<i>Nordic nRF51</i>	On-board	NRF51822	16MHz	256KB	16KB
<i>SeeedStudio CUI32stem</i>	<i>Microchip PIC32</i>	No	32MX795F512H	80MHz	508KB	128KB
<i>Seeeduino</i>	<i>Atmel AVR</i>	No	ATMEGA328P	16MHz	31.50KB	2KB
<i>Wio Link</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	4MB	80KB
<i>Wio Lite RISC-V</i>	<i>GigaDevice GD32V</i>	External	GD32VF103CBT6	108MHz	128KB	32KB
<i>Wio Node</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	4MB	80KB

Silicognition

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Silicognition wESP32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB

Sipeed

Name	Platform	Debug	MCU	Fre-quency	Flash	RAM
<i>GD32VF103V-EVAL</i>	<i>GigaDevice GD32V</i>	External	GD32VF103VBT6	108MHz	128KB	32KB
<i>Sipeed Longan Nano</i>	<i>GigaDevice GD32V</i>	External	GD32VF103CBT6	108MHz	128KB	32KB
<i>Sipeed MAIX BiT</i>	<i>Kendryte K210</i>	External	K210	400MHz	16MB	6MB
<i>Sipeed MAIX BiT with Mic</i>	<i>Kendryte K210</i>	External	K210	400MHz	16MB	6MB
<i>Sipeed MAIX GO</i>	<i>Kendryte K210</i>	External	K210	400MHz	16MB	6MB
<i>Sipeed MAIX ONE DOCK</i>	<i>Kendryte K210</i>	External	K210	400MHz	16MB	6MB
<i>Sipeed MAIXDUINO</i>	<i>Kendryte K210</i>	External	K210	400MHz	16MB	6MB

SparkFun

Name	Platform	De-bug	MCU	Fre-quency	Flash	RAM
<i>Pic32 CUI32-Development Stick</i>	<i>Microchip PIC32</i>	No	32MX440F512H	80MHz	508KB	32KB
<i>SparkFun ATmega128RFA1 Dev Board</i>	<i>Atmel AVR</i>	No	AT-MEGA128RFA1	16MHz	16KB	124KB
<i>SparkFun Blynk Board</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	4MB	80KB
<i>SparkFun Digital Sandbox</i>	<i>Atmel AVR</i>	No	ATMEGA328P	8MHz	31.50KB	2KB
<i>SparkFun ESP8266 Thing</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	512KB	80KB
<i>SparkFun ESP8266 Thing Dev</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	512KB	80KB
<i>SparkFun Fio V3 3.3V/8MHz</i>	<i>Atmel AVR</i>	No	ATMEGA32U4	8MHz	28KB	2.50KB
<i>SparkFun LoRa Gateway 1-Channel</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>SparkFun Makey Makey</i>	<i>Atmel AVR</i>	No	ATMEGA32U4	16MHz	28KB	2.50KB
<i>SparkFun Mega Pro 3.3V/8MHz</i>	<i>Atmel AVR</i>	No	ATMEGA2560	8MHz	252KB	8KB
<i>SparkFun Mega Pro 5V/16MHz</i>	<i>Atmel AVR</i>	No	ATMEGA2560	16MHz	248KB	8KB
<i>SparkFun Mega Pro Mini 3.3V</i>	<i>Atmel AVR</i>	No	ATMEGA2560	8MHz	252KB	8KB
<i>SparkFun MicroView</i>	<i>Atmel AVR</i>	No	ATMEGA328P	16MHz	31.50KB	2KB
<i>SparkFun Pro Micro 3.3V/8MHz</i>	<i>Atmel AVR</i>	No	ATMEGA32U4	8MHz	28KB	2.50KB
<i>SparkFun Pro Micro 5V/16MHz</i>	<i>Atmel AVR</i>	No	ATMEGA32U4	16MHz	28KB	2.50KB
<i>SparkFun Qduino Mini</i>	<i>Atmel AVR</i>	No	ATMEGA32U4	8MHz	28KB	2.50KB
<i>SparkFun RedBoard</i>	<i>Atmel AVR</i>	No	ATMEGA328P	16MHz	31.50KB	2KB
<i>SparkFun SAMD21 Dev Break-out</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz	256KB	32KB
<i>SparkFun SAMD21 Mini Break-out</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz	256KB	32KB
<i>SparkFun Serial 7-Segment Display</i>	<i>Atmel AVR</i>	No	ATMEGA328P	8MHz	31.50KB	2KB

SparkFun Electronics

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>SparkFun ESP32 Thing</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB

SpellFoundry

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>SpellFoundry Sleepy Pi 2</i>	<i>Atmel AVR</i>	No	ATMEGA328P	8MHz	30KB	2KB

SweetPea

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>SweetPea ESP-210</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	4MB	80KB

TI

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>TI FraunchPad MSP-EXP430FR5739LP</i>	<i>TI MSP430</i>	On-board	MSP430FR5739	16MHz	15.37KB	1KB
<i>TI LaunchPad (Stellaris) w/ lm4f120 (80MHz)</i>	<i>TI TIVA</i>	On-board	LPLM4F120H5QR	80MHz	256KB	32KB
<i>TI LaunchPad (Tiva C) w/ tm4c123 (80MHz)</i>	<i>TI TIVA</i>	On-board	LPTM4C1230C3PM	80MHz	256KB	32KB
<i>TI LaunchPad (Tiva C) w/ tm4c129 (120MHz)</i>	<i>TI TIVA</i>	On-board	LPTM4C1294NCPD	120MHz	1MB	256KB
<i>TI LaunchPad MSP-EXP430F5529LP</i>	<i>TI MSP430</i>	On-board	MSP430F5529	25MHz	47KB	8KB
<i>TI LaunchPad MSP-EXP430FR2311LP</i>	<i>TI MSP430</i>	On-board	MSP430FR2311	16MHz	3.75KB	1KB
<i>TI LaunchPad MSP-EXP430FR2433LP</i>	<i>TI MSP430</i>	On-board	MSP430FR2433	8MHz	15KB	4KB
<i>TI LaunchPad MSP-EXP430FR4133LP</i>	<i>TI MSP430</i>	On-board	MSP430FR4133	8MHz	15KB	2KB
<i>TI LaunchPad MSP-EXP430FR5969LP</i>	<i>TI MSP430</i>	On-board	MSP430FR5969	8MHz	47KB	2KB
<i>TI LaunchPad MSP-EXP430FR5994LP</i>	<i>TI MSP430</i>	On-board	MSP430FR5994	16MHz	256KB	4KB
<i>TI LaunchPad MSP-EXP430FR6989LP</i>	<i>TI MSP430</i>	On-board	MSP430FR6989	8MHz	47KB	2KB
<i>TI LaunchPad MSP-EXP430G2 w/ MSP430G2231</i>	<i>TI MSP430</i>	On-board	MSP430G2231	1MHz	2KB	256B
<i>TI LaunchPad MSP-EXP430G2 w/ MSP430G2452</i>	<i>TI MSP430</i>	On-board	MSP430G2452	16MHz	8KB	256B
<i>TI LaunchPad MSP-EXP430G2553LP</i>	<i>TI MSP430</i>	On-board	MSP430G2553	16MHz	16KB	512B

TTGO

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>TTGO LoRa32-OLED V1</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>TTGO T-Beam</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	1.25MB
<i>TTGO T-Watch</i>	<i>Espressif 32</i>	No	ESP32	240MHz	16MB	320KB
<i>TTGO TI</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB

Taida Century

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Taida Century nRF52 mini board</i>	<i>Nordic nRF52</i>	External	NRF52832	64MHz	512KB	64KB

TauLabs

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Sparky V1 F303</i>	<i>ST STM32</i>	External	STM32F303CCT6	72MHz	256KB	40KB

Teensy

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Teensy 2.0</i>	<i>Teensy</i>	No	ATMEGA32U4	16MHz	31.50KB	2.50KB
<i>Teensy 3.0</i>	<i>Teensy</i>	No	MK20DX128	48MHz	128KB	16KB
<i>Teensy 3.1 / 3.2</i>	<i>Teensy</i>	External	MK20DX256	72MHz	256KB	64KB
<i>Teensy 3.5</i>	<i>Teensy</i>	External	MK64FX512	120MHz	512KB	255.99KB
<i>Teensy 3.6</i>	<i>Teensy</i>	External	MK66FX1M0	180MHz	1MB	256KB
<i>Teensy 4.0</i>	<i>Teensy</i>	External	IMXRT1062	600MHz	1.94MB	1MB
<i>Teensy LC</i>	<i>Teensy</i>	External	MKL26Z64	48MHz	62KB	8KB
<i>Teensy++ 2.0</i>	<i>Teensy</i>	No	AT90USB1286	16MHz	127KB	8KB

ThaiEasyElec

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>ESPino32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>ThaiEasyElec ESPino</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	4MB	80KB

The Things Network

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>The Things Uno</i>	<i>Atmel AVR</i>	No	ATMEGA32U4	16MHz	28KB	2.50KB

Till Harbaum

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>ftDuino</i>	<i>Atmel AVR</i>	No	ATMEGA32U4	16MHz	28KB	2.50KB

TinyCircuits

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>TinyCircuits TinyDuino Processor Board</i>	<i>Atmel AVR</i>	No	AT-MEGA328P	8MHz	30KB	2KB
<i>TinyCircuits TinyLily Mini Processor</i>	<i>Atmel AVR</i>	No	AT-MEGA328P	8MHz	30KB	2KB

TinyPICO

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>TinyPICO</i>	<i>Espressif 32</i>	No	ESP32	240MHz	4MB	320KB

Turta

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Turta IoT Node</i>	<i>Espressif 32</i>	No	ESP32	240MHz	4MB	320KB

UBW32

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>UBW32 MX460</i>	<i>Microchip PIC32</i>	No	32MX460F512L	80MHz	508KB	32KB
<i>UBW32 MX795</i>	<i>Microchip PIC32</i>	No	32MX795F512L	80MHz	508KB	128KB

Unknown

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>ESP32 FM DevKit</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB

VAE

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>VaKE v1.0</i>	<i>ST STM32</i>	External	STM32F446RET6	180MHz	512KB	128KB

VintLabs

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>VintLabs ESP32 Devkit</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB

WEMOS

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>WEMOS D1 R1</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	4MB	80KB
<i>WEMOS LOLIN D32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>WEMOS LOLIN D32 PRO</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>WEMOS LOLIN32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>WeMos D1 MINI ESP32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>WeMos D1 R2 and mini</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	4MB	80KB
<i>WeMos D1 mini Lite</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	1MB	80KB
<i>WeMos D1 mini Pro</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	16MB	80KB
<i>WeMos WiFi and Bluetooth Battery</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB

Waveshare

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Waveshare BLE400</i>	<i>Nordic nRF51</i>	External	NRF51822	32MHz	256KB	32KB

Wicked Device

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Wicked Device WildFire V2</i>	<i>Atmel AVR</i>	No	ATMEGA1284P	16MHz	120.00KB	16KB
<i>Wicked Device WildFire V3</i>	<i>Atmel AVR</i>	No	ATMEGA1284P	16MHz	127KB	16KB

Widora

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Widora AIR</i>	<i>Espressif 32</i>	No	ESP32	240MHz	16MB	320KB

WifiDuino

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>WiFiduino</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	4MB	80KB

Wisen

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Talk2 Whisper Node</i>	<i>Atmel AVR</i>	No	ATMEGA328P	16MHz	31.50KB	2KB

XinaBox

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>XinaBox CW01</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	4MB	80KB
<i>XinaBox CW02</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB

chipKIT

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>chipKIT Lenny</i>	<i>Microchip PIC32</i>	No	32MX270F256D	40MHz	120KB	32KB

element14

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Element14 chipKIT Pi</i>	<i>Microchip PIC32</i>	No	32MX250F128B	40MHz	120KB	32KB

makerlab.mx

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Altair</i>	<i>Atmel AVR</i>	No	ATMEGA256RFR2	16MHz	248KB	32KB

ng-beacon

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>ng-beacon</i>	<i>Nordic nRF51</i>	External	NRF51822	16MHz	256KB	32KB

nicai-systems

Name	Platform	De- bug	MCU	Fre- quency	Flash	RAM
<i>nicai-systems BOB3 coding bot</i>	<i>Atmel AVR</i>	No	ATMEGA88	8MHz	8KB	1KB
<i>nicai-systems NIBO 2 robot</i>	<i>Atmel AVR</i>	No	ATMEGA128	16MHz	128KB	4KB
<i>nicai-systems NIBO burger robot</i>	<i>Atmel AVR</i>	No	ATMEGA16	15MHz	16KB	1KB
<i>nicai-systems NIBO burger robot with Tuning Kit</i>	<i>Atmel AVR</i>	No	ATMEGA1284P	20MHz	128KB	16KB
<i>nicai-systems NIBObee robot</i>	<i>Atmel AVR</i>	No	ATMEGA16	15MHz	16KB	1KB
<i>nicai-systems NIBObee robot with Tuning Kit</i>	<i>Atmel AVR</i>	No	ATMEGA1284P	20MHz	128KB	16KB

oddWires

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>oddWires IoT-Bus Io</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>oddWires IoT-Bus Proteus</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB

sduino

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>sduino MB (STM8S208MBT6B)</i>	<i>ST STM8</i>	No	STM8S208MBT6	16MHz	128KB	6KB
<i>sduino UNO (STM8S105K6)</i>	<i>ST STM8</i>	No	STM8S105K6T6	16MHz	32KB	2KB

sino:bit

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Sino:Bit</i>	<i>Nordic nRF51</i>	External	NRF51822	32MHz	256KB	32KB

u-blox

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>u-blox EVK-NINA-B1</i>	<i>Nordic nRF52</i>	On-board	NRF52832	64MHz	512KB	64KB
<i>u-blox NINA-W10 series</i>	<i>Espressif 32</i>	No	ESP32	240MHz	2MB	320KB

ubIQio

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>ubIQio Ardhhat</i>	<i>Atmel AVR</i>	No	ATMEGA328P	16MHz	31.50KB	2KB

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](#).

Contents

- *Examples*
- *Platforms*

- *Boards*

Examples

- ARTIK SDK for Linux ARM

Platforms

Name	Description
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.

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

Name	Platform	Debug	MCU	Frequency	Flash	RAM
RushUp Kitra 520	Linux ARM	No	EXYNOS3250	1000MHz	4GB	512MB

Samsung

Name	Platform	Debug	MCU	Frequency	Flash	RAM
Samsung ARTIK 1020	Linux ARM	No	EXYNOS5422	1500MHz	16GB	2GB
Samsung ARTIK 520	Linux ARM	No	EXYNOS3250	1000MHz	4GB	512MB
Samsung ARTIK 530	Linux ARM	No	S5P4418	1200MHz	4GB	512MB
Samsung ARTIK 710	Linux ARM	No	S5P6818	1400MHz	4GB	1GB

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.

Name	Platform	MCU	Frequency	Flash	RAM
RushUp Cloud-JAM	ST STM32	STM32F401RET6	84MHz	512KB	96KB
ST Nucleo F401RE	ST STM32	STM32F401RET6	84MHz	512KB	96KB
ST STM32F3DISCOVERY	ST STM32	STM32F303VCT6	72MHz	256KB	48KB
ST STM32F4DISCOVERY	ST STM32	STM32F407VGT6	168MHz	1MB	128KB
ST STM32LDiscovery	ST STM32	STM32L152RBT6	32MHz	128KB	16KB

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.

Name	Platform	MCU	Frequency	Flash	RAM
<i>1Bitsy</i>	<i>ST STM32</i>	STM32F415RGT	168MHz	1MB	128KB
<i>3D Printer Controller</i>	<i>ST STM32</i>	STM32F407VET6	168MHz	512KB	192KB
<i>Armstrap Eagle 1024</i>	<i>ST STM32</i>	STM32F417VGT6	168MHz	1MB	192KB
<i>Armstrap Eagle 2048</i>	<i>ST STM32</i>	STM32F427VIT6	168MHz	1.99MB	256KB
<i>Armstrap Eagle 512</i>	<i>ST STM32</i>	STM32F407VET6	168MHz	512KB	192KB
<i>Black STM32F407VE</i>	<i>ST STM32</i>	STM32F407VET6	168MHz	512KB	128KB
<i>Black STM32F407VG</i>	<i>ST STM32</i>	STM32F407VGT6	168MHz	512KB	128KB
<i>Black STM32F407ZE</i>	<i>ST STM32</i>	STM32F407ZET6	168MHz	512KB	128KB
<i>Black STM32F407ZE</i>	<i>ST STM32</i>	STM32F407ZGT6	168MHz	1MB	128KB
<i>BlackPill F103C8</i>	<i>ST STM32</i>	STM32F103C8T6	72MHz	64KB	20KB
<i>BlackPill F103C8 (128k)</i>	<i>ST STM32</i>	STM32F103C8T6	72MHz	128KB	20KB
<i>Blue STM32F407VE Mini</i>	<i>ST STM32</i>	STM32F407VET6	168MHz	512KB	128KB
<i>BluePill F103C6</i>	<i>ST STM32</i>	STM32F103C6T6	72MHz	32KB	10KB
<i>BluePill F103C8</i>	<i>ST STM32</i>	STM32F103C8T6	72MHz	64KB	20KB
<i>BluePill F103C8 (128k)</i>	<i>ST STM32</i>	STM32F103C8T6	72MHz	128KB	20KB
<i>F407VG</i>	<i>ST STM32</i>	STM32F407VGT6	168MHz	512KB	128KB
<i>FK407M1</i>	<i>ST STM32</i>	STM32F407VET6	168MHz	512KB	128KB
<i>STM32F103C8 (20k RAM, 64k Flash)</i>	<i>ST STM32</i>	STM32F103C8T6	72MHz	64KB	20KB
<i>STM32F103CB (20k RAM, 128k Flash)</i>	<i>ST STM32</i>	STM32F103CBT6	72MHz	128KB	20KB
<i>STM32F103R8 (20k RAM, 64 Flash)</i>	<i>ST STM32</i>	STM32F103R8T6	72MHz	64KB	20KB
<i>STM32F103RB (20k RAM, 128k Flash)</i>	<i>ST STM32</i>	STM32F103RBT6	72MHz	128KB	20KB
<i>STM32F103RC (48k RAM, 256k Flash)</i>	<i>ST STM32</i>	STM32F103RCT6	72MHz	256KB	48KB
<i>STM32F103RE (64k RAM, 512k Flash)</i>	<i>ST STM32</i>	STM32F103RET6	72MHz	512KB	64KB
<i>STM32F103T8 (20k RAM, 64k Flash)</i>	<i>ST STM32</i>	STM32F103T8T6	72MHz	20KB	64KB
<i>STM32F103TB (20k RAM, 128k Flash)</i>	<i>ST STM32</i>	STM32F103TBT6	72MHz	128KB	20KB
<i>STM32F103VB (20k RAM, 128k Flash)</i>	<i>ST STM32</i>	STM32F103VBT6	72MHz	128KB	20KB
<i>STM32F103VC (48k RAM, 256k Flash)</i>	<i>ST STM32</i>	STM32F103VCT6	72MHz	256KB	48KB
<i>STM32F103VD (64k RAM, 384k Flash)</i>	<i>ST STM32</i>	STM32F103VDT6	72MHz	384KB	64KB
<i>STM32F103VE (64k RAM, 512k Flash)</i>	<i>ST STM32</i>	STM32F103VET6	72MHz	512KB	64KB
<i>STM32F103ZC (48k RAM, 256k Flash)</i>	<i>ST STM32</i>	STM32F103ZCT6	72MHz	256KB	48KB
<i>STM32F103ZD (64k RAM, 384k Flash)</i>	<i>ST STM32</i>	STM32F103ZDT6	72MHz	384KB	64KB
<i>STM32F103ZE (64k RAM, 512k Flash)</i>	<i>ST STM32</i>	STM32F103ZET6	72MHz	512KB	64KB
<i>STM32F303CB (32k RAM, 128k Flash)</i>	<i>ST STM32</i>	STM32F303CBT6	72MHz	128KB	32KB
<i>STM32F407VE (192k RAM, 512k Flash)</i>	<i>ST STM32</i>	STM32F407VET6	168MHz	502.23KB	128KB
<i>STM32F407VG (192k RAM, 1024k Flash)</i>	<i>ST STM32</i>	STM32F407VGT6	168MHz	1MB	192KB

Examples

- CMSIS for ST STM32

Platforms

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

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

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>IBitsy</i>	<i>ST STM32</i>	External	STM32F415RGT	168MHz	1MB	128KB

Armed

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>3D Printer Controller</i>	<i>ST STM32</i>	External	STM32F407VET6	168MHz	512KB	192KB

Armstrap

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Armstrap Eagle 1024</i>	<i>ST STM32</i>	External	STM32F417VGT6	168MHz	1MB	192KB
<i>Armstrap Eagle 2048</i>	<i>ST STM32</i>	External	STM32F427VIT6	168MHz	1.99MB	256KB
<i>Armstrap Eagle 512</i>	<i>ST STM32</i>	External	STM32F407VET6	168MHz	512KB	192KB

Diymore

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>F407VG</i>	<i>ST STM32</i>	External	STM32F407VGT6	168MHz	512KB	128KB

Generic

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>BlackPill F103C8</i>	<i>ST STM32</i>	External	STM32F103C8T6	72MHz	64KB	20KB
<i>BlackPill F103C8 (128k)</i>	<i>ST STM32</i>	External	STM32F103C8T6	72MHz	128KB	20KB
<i>BluePill F103C6</i>	<i>ST STM32</i>	External	STM32F103C6T6	72MHz	32KB	10KB
<i>BluePill F103C8</i>	<i>ST STM32</i>	External	STM32F103C8T6	72MHz	64KB	20KB
<i>BluePill F103C8 (128k)</i>	<i>ST STM32</i>	External	STM32F103C8T6	72MHz	128KB	20KB
<i>FK407M1</i>	<i>ST STM32</i>	External	STM32F407VET6	168MHz	512KB	128KB
<i>STM32F103C8 (20k RAM. 64k Flash)</i>	<i>ST STM32</i>	External	STM32F103C8T6	72MHz	64KB	20KB
<i>STM32F103CB (20k RAM. 128k Flash)</i>	<i>ST STM32</i>	External	STM32F103CBT6	72MHz	128KB	20KB
<i>STM32F103R8 (20k RAM. 64 Flash)</i>	<i>ST STM32</i>	External	STM32F103R8T6	72MHz	64KB	20KB
<i>STM32F103RB (20k RAM. 128k Flash)</i>	<i>ST STM32</i>	External	STM32F103RBT6	72MHz	128KB	20KB
<i>STM32F103RC (48k RAM. 256k Flash)</i>	<i>ST STM32</i>	External	STM32F103RCT6	72MHz	256KB	48KB
<i>STM32F103RE (64k RAM. 512k Flash)</i>	<i>ST STM32</i>	External	STM32F103RET6	72MHz	512KB	64KB
<i>STM32F103T8 (20k RAM. 64k Flash)</i>	<i>ST STM32</i>	External	STM32F103T8T6	72MHz	20KB	64KB
<i>STM32F103TB (20k RAM. 128k Flash)</i>	<i>ST STM32</i>	External	STM32F103TBT6	72MHz	128KB	20KB
<i>STM32F103VB (20k RAM. 128k Flash)</i>	<i>ST STM32</i>	External	STM32F103VBT6	72MHz	128KB	20KB
<i>STM32F103VC (48k RAM. 256k Flash)</i>	<i>ST STM32</i>	External	STM32F103VCT6	72MHz	256KB	48KB
<i>STM32F103VD (64k RAM. 384k Flash)</i>	<i>ST STM32</i>	External	STM32F103VDT6	72MHz	384KB	64KB
<i>STM32F103VE (64k RAM. 512k Flash)</i>	<i>ST STM32</i>	External	STM32F103VET6	72MHz	512KB	64KB
<i>STM32F103ZC (48k RAM. 256k Flash)</i>	<i>ST STM32</i>	External	STM32F103ZCT6	72MHz	256KB	48KB
<i>STM32F103ZD (64k RAM. 384k Flash)</i>	<i>ST STM32</i>	External	STM32F103ZDT6	72MHz	384KB	64KB
<i>STM32F103ZE (64k RAM. 512k Flash)</i>	<i>ST STM32</i>	External	STM32F103ZET6	72MHz	512KB	64KB
<i>STM32F303CB (32k RAM. 128k Flash)</i>	<i>ST STM32</i>	External	STM32F303CBT6	72MHz	128KB	32KB
<i>STM32F407VE (192k RAM. 512k Flash)</i>	<i>ST STM32</i>	External	STM32F407VET6	168MHz	502.23KB	128KB
<i>STM32F407VG (192k RAM. 1024k Flash)</i>	<i>ST STM32</i>	External	STM32F407VGT6	168MHz	1MB	192KB

RushUp

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>RushUp Cloud-JAM</i>	<i>ST STM32</i>	On-board	STM32F401RET6	84MHz	512KB	96KB

ST

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Black STM32F407VE</i>	<i>ST STM32</i>	External	STM32F407VET6	168MHz	512KB	128KB
<i>Black STM32F407VG</i>	<i>ST STM32</i>	External	STM32F407VGT6	168MHz	512KB	128KB
<i>Black STM32F407ZE</i>	<i>ST STM32</i>	External	STM32F407ZET6	168MHz	512KB	128KB
<i>Black STM32F407ZE</i>	<i>ST STM32</i>	External	STM32F407ZGT6	168MHz	1MB	128KB
<i>Blue STM32F407VE Mini</i>	<i>ST STM32</i>	External	STM32F407VET6	168MHz	512KB	128KB
<i>ST Nucleo F401RE</i>	<i>ST STM32</i>	On-board	STM32F401RET6	84MHz	512KB	96KB
<i>ST STM32F3DISCOVERY</i>	<i>ST STM32</i>	On-board	STM32F303VCT6	72MHz	256KB	48KB
<i>ST STM32F4DISCOVERY</i>	<i>ST STM32</i>	On-board	STM32F407VGT6	168MHz	1MB	128KB
<i>ST STM32LDISCOVERY</i>	<i>ST STM32</i>	On-board	STM32L152RBT6	32MHz	128KB	16KB

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

Name	Description
<i>Espressif 8266</i>	Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

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

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>4D Systems gen4 IoD Range</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	512KB	80KB

Adafruit

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Adafruit HUZZAH ESP8266</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	4MB	80KB

Amperka

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>WiFi Slot</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	4MB	80KB

Doit

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>ESPDuino (ESP-13 Module)</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	4MB	80KB

DycodeX

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>ESPectro Core</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	4MB	80KB

ESPert

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>ESPRESSO Lite 1.0</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	4MB	80KB
<i>ESPRESSO Lite 2.0</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	4MB	80KB

ESPino

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>ESPino</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	4MB	80KB

Espressif

Name	Platform	De-bug	MCU	Fre-quency	Flash	RAM
<i>ESP-WROOM-02</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	2MB	80KB
<i>Espressif ESP8266 ESP-12E</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	4MB	80KB
<i>Espressif Generic ESP8266 ESP-01 1M</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	1MB	80KB
<i>Espressif Generic ESP8266 ESP-01 512k</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	512KB	80KB
<i>Espressif Generic ESP8266 ESP-07</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	4MB	80KB
<i>Generic ESP8285 Module</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	1MB	80KB
<i>Phoenix 1.0</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	4MB	80KB
<i>Phoenix 2.0</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	4MB	80KB
<i>WifInfo</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	1MB	80KB

Heltec

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Heltec Wifi kit 8</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	4MB	80KB

Invent One

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Invent One</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	4MB	80KB

NodeMCU

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>NodeMCU 0.9 (ESP-12 Module)</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	4MB	80KB
<i>NodeMCU 1.0 (ESP-12E Module)</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	4MB	80KB

Olimex

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Olimex MOD-WIFI-ESP8266(-DEV)</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	2MB	80KB

SeeedStudio

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Wio Link</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	4MB	80KB
<i>Wio Node</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	4MB	80KB

SparkFun

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>SparkFun Blynk Board</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	4MB	80KB
<i>SparkFun ESP8266 Thing</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	512KB	80KB
<i>SparkFun ESP8266 Thing Dev</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	512KB	80KB

SweetPea

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>SweetPea ESP-210</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	4MB	80KB

ThaiEasyElec

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>ThaiEasyElec ESPino</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	4MB	80KB

WEMOS

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>WEMOS D1 R1</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	4MB	80KB
<i>WeMos D1 R2 and mini</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	4MB	80KB
<i>WeMos D1 mini Pro</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	16MB	80KB

WifiDuino

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>WiFiduino</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	4MB	80KB

XinaBox

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>XinaBox CW01</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	4MB	80KB

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

Name	Description
<i>Espressif 8266</i>	Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

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

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>4D Systems gen4 IoD Range</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	512KB	80KB

Adafruit

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Adafruit HUZZAH ESP8266</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	4MB	80KB

Amperka

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>WiFi Slot</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	4MB	80KB

Doit

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>ESPDuino (ESP-13 Module)</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	4MB	80KB

DycodeX

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>ESPectro Core</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	4MB	80KB

ESPert

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>ESPRESSO Lite 1.0</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	4MB	80KB
<i>ESPRESSO Lite 2.0</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	4MB	80KB

ESPino

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>ESPino</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	4MB	80KB

Espressif

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>ESP-WROOM-02</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	2MB	80KB
<i>Espressif ESP8266 ESP-12E</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	4MB	80KB
<i>Espressif Generic ESP8266 ESP-01 1M</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	1MB	80KB
<i>Espressif Generic ESP8266 ESP-01 512k</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	512KB	80KB
<i>Espressif Generic ESP8266 ESP-07</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	4MB	80KB
<i>Generic ESP8285 Module</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	1MB	80KB
<i>Phoenix 1.0</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	4MB	80KB
<i>Phoenix 2.0</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	4MB	80KB
<i>WifInfo</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	1MB	80KB

Heltec

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Heltec Wifi kit 8</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	4MB	80KB

Invent One

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Invent One</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	4MB	80KB

NodeMCU

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>NodeMCU 0.9 (ESP-12 Module)</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	4MB	80KB
<i>NodeMCU 1.0 (ESP-12E Module)</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	4MB	80KB

Olimex

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Olimex MOD-WIFI-ESP8266(-DEV)</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	2MB	80KB

SeeedStudio

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Wio Link</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	4MB	80KB
<i>Wio Node</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	4MB	80KB

SparkFun

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>SparkFun Blynk Board</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	4MB	80KB
<i>SparkFun ESP8266 Thing</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	512KB	80KB
<i>SparkFun ESP8266 Thing Dev</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	512KB	80KB

SweetPea

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>SweetPea ESP-210</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	4MB	80KB

ThaiEasyElec

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>ThaiEasyElec ESPino</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	4MB	80KB

WEMOS

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>WEMOS D1 R1</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	4MB	80KB
<i>WeMos D1 R2 and mini</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	4MB	80KB
<i>WeMos D1 mini Pro</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	16MB	80KB

WifiDuino

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>WiFiduino</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	4MB	80KB

XinaBox

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>XinaBox CW01</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	4MB	80KB

1.11.6 ESP-IDF

Configuration `framework = espidf`

Espressif IoT Development Framework. Official development framework for ESP32.

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

Bands listed below have on-board debug probe and **ARE READY** for debugging! You do not need to use/buy external debug probe.

Name	Platform	MCU	Frequency	Flash	RAM
Espressif ESP-WROVER-KIT	Espressif 32	ESP32	240MHz	4MB	320KB

External Debug Tools

Bands 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.

Name	Platform	MCU	Frequency	Flash	RAM
<i>AI Thinker ESP32-CAM</i>	<i>Espressif 32</i>	ESP32	240MHz	4MB	320KB
<i>Adafruit ESP32 Feather</i>	<i>Espressif 32</i>	ESP32	240MHz	4MB	320KB
<i>D-duino-32</i>	<i>Espressif 32</i>	ESP32	240MHz	4MB	320KB
<i>DOIT ESP32 DEVKIT V1</i>	<i>Espressif 32</i>	ESP32	240MHz	4MB	320KB
<i>Dongsen Tech Pocket 32</i>	<i>Espressif 32</i>	ESP32	240MHz	4MB	320KB
<i>ESP32 FM DevKit</i>	<i>Espressif 32</i>	ESP32	240MHz	4MB	320KB
<i>ESP32vn IoT Uno</i>	<i>Espressif 32</i>	ESP32	240MHz	4MB	320KB
<i>ESPectro32</i>	<i>Espressif 32</i>	ESP32	240MHz	4MB	320KB
<i>ESPino32</i>	<i>Espressif 32</i>	ESP32	240MHz	4MB	320KB
<i>Espressif ESP32 Dev Module</i>	<i>Espressif 32</i>	ESP32	240MHz	4MB	320KB
<i>FireBeetle-ESP32</i>	<i>Espressif 32</i>	ESP32	240MHz	4MB	320KB
<i>Frog Board ESP32</i>	<i>Espressif 32</i>	ESP32	240MHz	4MB	320KB
<i>Heltec WiFi LoRa 32</i>	<i>Espressif 32</i>	ESP32	240MHz	4MB	320KB
<i>Heltec WiFi LoRa 32 (V2)</i>	<i>Espressif 32</i>	ESP32	240MHz	8MB	320KB
<i>Heltec Wireless Stick</i>	<i>Espressif 32</i>	ESP32	240MHz	8MB	320KB
<i>Hornbill ESP32 Dev</i>	<i>Espressif 32</i>	ESP32	240MHz	4MB	320KB
<i>Hornbill ESP32 Minima</i>	<i>Espressif 32</i>	ESP32	240MHz	4MB	320KB
<i>MH ET LIVE ESP32DevKIT</i>	<i>Espressif 32</i>	ESP32	240MHz	4MB	320KB
<i>MH ET LIVE ESP32MiniKit</i>	<i>Espressif 32</i>	ESP32	240MHz	4MB	320KB
<i>Node32s</i>	<i>Espressif 32</i>	ESP32	240MHz	4MB	320KB
<i>NodeMCU-32S</i>	<i>Espressif 32</i>	ESP32	240MHz	4MB	320KB
<i>OLIMEX ESP32-DevKit-LiPo</i>	<i>Espressif 32</i>	ESP32	240MHz	4MB	320KB
<i>OLIMEX ESP32-EVB</i>	<i>Espressif 32</i>	ESP32	240MHz	4MB	320KB
<i>OLIMEX ESP32-GATEWAY</i>	<i>Espressif 32</i>	ESP32	240MHz	4MB	320KB
<i>Pycom LoPy</i>	<i>Espressif 32</i>	ESP32	240MHz	4MB	320KB
<i>Pycom LoPy4</i>	<i>Espressif 32</i>	ESP32	240MHz	4MB	1.25MB
<i>Silicognition wESP32</i>	<i>Espressif 32</i>	ESP32	240MHz	4MB	320KB
<i>SparkFun ESP32 Thing</i>	<i>Espressif 32</i>	ESP32	240MHz	4MB	320KB
<i>SparkFun LoRa Gateway 1-Channel</i>	<i>Espressif 32</i>	ESP32	240MHz	4MB	320KB
<i>TTGO LoRa32-OLED V1</i>	<i>Espressif 32</i>	ESP32	240MHz	4MB	320KB
<i>TTGO T-Beam</i>	<i>Espressif 32</i>	ESP32	240MHz	4MB	1.25MB
<i>TTGO T1</i>	<i>Espressif 32</i>	ESP32	240MHz	4MB	320KB
<i>VintLabs ESP32 Devkit</i>	<i>Espressif 32</i>	ESP32	240MHz	4MB	320KB
<i>WEMOS LOLIN D32</i>	<i>Espressif 32</i>	ESP32	240MHz	4MB	320KB
<i>WEMOS LOLIN D32 PRO</i>	<i>Espressif 32</i>	ESP32	240MHz	4MB	320KB
<i>WEMOS LOLIN32</i>	<i>Espressif 32</i>	ESP32	240MHz	4MB	320KB
<i>WeMos D1 MINI ESP32</i>	<i>Espressif 32</i>	ESP32	240MHz	4MB	320KB
<i>WeMos WiFi and Bluetooth Battery</i>	<i>Espressif 32</i>	ESP32	240MHz	4MB	320KB
<i>XinaBox CW02</i>	<i>Espressif 32</i>	ESP32	240MHz	4MB	320KB
<i>oddWires IoT-Bus Io</i>	<i>Espressif 32</i>	ESP32	240MHz	4MB	320KB
<i>oddWires IoT-Bus Proteus</i>	<i>Espressif 32</i>	ESP32	240MHz	4MB	320KB

Examples

- ESP-IDF for Espressif 32

Platforms

Name	Description
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.

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

Name	Platform	Debug	MCU	Frequency	Flash	RAM
AI Thinker ESP32-CAM	Espressif 32	External	ESP32	240MHz	4MB	320KB

Adafruit

Name	Platform	Debug	MCU	Frequency	Flash	RAM
Adafruit ESP32 Feather	Espressif 32	External	ESP32	240MHz	4MB	320KB

Aiyarafun

Name	Platform	Debug	MCU	Frequency	Flash	RAM
Node32s	Espressif 32	External	ESP32	240MHz	4MB	320KB

April Brother

Name	Platform	Debug	MCU	Frequency	Flash	RAM
April Brother ESPea32	Espressif 32	No	ESP32	240MHz	4MB	320KB

BPI Tech

Name	Platform	Debug	MCU	Frequency	Flash	RAM
BPI-Bit	Espressif 32	No	ESP32	160MHz	4MB	320KB

DFRobot

Name	Platform	Debug	MCU	Frequency	Flash	RAM
FireBeetle-ESP32	Espressif 32	External	ESP32	240MHz	4MB	320KB

DOIT

Name	Platform	Debug	MCU	Frequency	Flash	RAM
DOIT ESP32 DEVKIT V1	Espressif 32	External	ESP32	240MHz	4MB	320KB

DSTIKE

Name	Platform	Debug	MCU	Frequency	Flash	RAM
D-duino-32	Espressif 32	External	ESP32	240MHz	4MB	320KB

Dongsen Technology

Name	Platform	Debug	MCU	Frequency	Flash	RAM
Dongsen Tech Pocket 32	Espressif 32	External	ESP32	240MHz	4MB	320KB

DycodeX

Name	Platform	Debug	MCU	Frequency	Flash	RAM
ESPectro32	Espressif 32	External	ESP32	240MHz	4MB	320KB

ESP32vn

Name	Platform	Debug	MCU	Frequency	Flash	RAM
ESP32vn IoT Uno	Espressif 32	External	ESP32	240MHz	4MB	320KB

Electronic SweetPeas

Name	Platform	Debug	MCU	Frequency	Flash	RAM
Electronic SweetPeas ESP320	Espressif 32	No	ESP32	240MHz	4MB	320KB

Espressif

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>ESP32 Pico Kit</i>	<i>Espressif 32</i>	No	ESP32	240MHz	4MB	320KB
<i>Espressif ESP-WROVER-KIT</i>	<i>Espressif 32</i>	On-board	ESP32	240MHz	4MB	320KB
<i>Espressif ESP32 Dev Module</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB

Fred

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Frog Board ESP32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB

Hardkernel

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>ODROID-GO</i>	<i>Espressif 32</i>	No	ESP32	240MHz	16MB	320KB

Heltec Automation

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Heltec WiFi Kit 32</i>	<i>Espressif 32</i>	No	ESP32	240MHz	4MB	320KB
<i>Heltec WiFi LoRa 32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>Heltec WiFi LoRa 32 (V2)</i>	<i>Espressif 32</i>	External	ESP32	240MHz	8MB	320KB
<i>Heltec Wireless Stick</i>	<i>Espressif 32</i>	External	ESP32	240MHz	8MB	320KB

Hornbill

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Hornbill ESP32 Dev</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>Hornbill ESP32 Minima</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB

IntoRobot

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>IntoRobot Fig</i>	<i>Espressif 32</i>	No	ESP32	240MHz	4MB	320KB

M5Stack

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>M5Stack Core ESP32</i>	<i>Espressif 32</i>	No	ESP32	240MHz	4MB	320KB
<i>M5Stack FIRE</i>	<i>Espressif 32</i>	No	ESP32	240MHz	16MB	6.25MB
<i>M5Stack GREY ESP32</i>	<i>Espressif 32</i>	No	ESP32	240MHz	16MB	520KB
<i>M5Stick-C</i>	<i>Espressif 32</i>	No	ESP32	240MHz	4MB	320KB

MH-ET Live

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>MH ET LIVE ESP32DevKIT</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>MH ET LIVE ESP32MiniKit</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB

Magicblocks.io

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>MagicBit</i>	<i>Espressif 32</i>	No	ESP32	240MHz	4MB	320KB

MakerAsia

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>MakerAsia Nano32</i>	<i>Espressif 32</i>	No	ESP32	240MHz	4MB	320KB

Microduino

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Microduino Core ESP32</i>	<i>Espressif 32</i>	No	ESP32	240MHz	4MB	320KB

NodeMCU

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>NodeMCU-32S</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB

Noduino

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Noduino Quantum</i>	<i>Espressif 32</i>	No	ESP32	240MHz	16MB	320KB

OLIMEX

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>OLIMEX ESP32-DevKit-LiPo</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>OLIMEX ESP32-EVB</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>OLIMEX ESP32-GATEWAY</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>OLIMEX ESP32-PRO</i>	<i>Espressif 32</i>	No	ESP32	240MHz	4MB	320KB
<i>OLIMEX ESP32-PoE</i>	<i>Espressif 32</i>	No	ESP32	240MHz	4MB	320KB
<i>OLIMEX ESP32-PoE-ISO</i>	<i>Espressif 32</i>	No	ESP32	240MHz	4MB	320KB

OROCA

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>OROCA EduBot</i>	<i>Espressif 32</i>	No	ESP32	240MHz	4MB	320KB

Onehorse

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Onehorse ESP32 Dev Module</i>	<i>Espressif 32</i>	No	ESP32	240MHz	4MB	320KB

Pycom Ltd.

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Pycom GPy</i>	<i>Espressif 32</i>	No	ESP32	240MHz	4MB	320KB
<i>Pycom LoPy</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>Pycom LoPy4</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	1.25MB

Silicognition

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Silicognition wESP32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB

SparkFun

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>SparkFun LoRa Gateway 1-Channel</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB

SparkFun Electronics

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>SparkFun ESP32 Thing</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB

TTGO

Name	Platform	Debug	MCU	Frequency	Flash	RAM
TTGO LoRa32-OLED V1	Espressif 32	External	ESP32	240MHz	4MB	320KB
TTGO T-Beam	Espressif 32	External	ESP32	240MHz	4MB	1.25MB
TTGO T-Watch	Espressif 32	No	ESP32	240MHz	16MB	320KB
TTGO T1	Espressif 32	External	ESP32	240MHz	4MB	320KB

ThaiEasyElec

Name	Platform	Debug	MCU	Frequency	Flash	RAM
ESPino32	Espressif 32	External	ESP32	240MHz	4MB	320KB

TinyPICO

Name	Platform	Debug	MCU	Frequency	Flash	RAM
TinyPICO	Espressif 32	No	ESP32	240MHz	4MB	320KB

Turta

Name	Platform	Debug	MCU	Frequency	Flash	RAM
Turta IoT Node	Espressif 32	No	ESP32	240MHz	4MB	320KB

Unknown

Name	Platform	Debug	MCU	Frequency	Flash	RAM
ESP32 FM DevKit	Espressif 32	External	ESP32	240MHz	4MB	320KB

VintLabs

Name	Platform	Debug	MCU	Frequency	Flash	RAM
VintLabs ESP32 Devkit	Espressif 32	External	ESP32	240MHz	4MB	320KB

WEMOS

Name	Platform	Debug	MCU	Frequency	Flash	RAM
WEMOS LOLIN D32	Espressif 32	External	ESP32	240MHz	4MB	320KB
WEMOS LOLIN D32 PRO	Espressif 32	External	ESP32	240MHz	4MB	320KB
WEMOS LOLIN32	Espressif 32	External	ESP32	240MHz	4MB	320KB
WeMos D1 MINI ESP32	Espressif 32	External	ESP32	240MHz	4MB	320KB
WeMos WiFi and Bluetooth Battery	Espressif 32	External	ESP32	240MHz	4MB	320KB

Widora

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Widora AIR</i>	<i>Espressif 32</i>	No	ESP32	240MHz	16MB	320KB

XinaBox

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>XinaBox CW02</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB

oddWires

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>oddWires IoT-Bus Io</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>oddWires IoT-Bus Proteus</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB

u-blox

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>u-blox NINA-W10 series</i>	<i>Espressif 32</i>	No	ESP32	240MHz	2MB	320KB

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*
 - *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

Banks listed below have on-board debug probe and **ARE READY** for debugging! You do not need to use/buy external debug probe.

Name	Platform	MCU	Frequency	Flash	RAM
<i>Aarty FPGA Dev Kit</i>	<i>SiFive</i>	FE310	450MHz	16MB	256MB
<i>HiFive Unleashed</i>	<i>SiFive</i>	FU540	1500MHz	32MB	8GB
<i>HiFive1</i>	<i>SiFive</i>	FE310	320MHz	16MB	16KB
<i>HiFive1 Rev B</i>	<i>SiFive</i>	FE310	320MHz	16MB	16KB

Examples

- Freedom E SDK for SiFive

Platforms

Name	Description
<i>SiFive</i>	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.

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

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>HiFive Unleashed</i>	<i>SiFive</i>	On-board	FU540	1500MHz	32MB	8GB
<i>HiFive1</i>	<i>SiFive</i>	On-board	FE310	320MHz	16MB	16KB
<i>HiFive1 Rev B</i>	<i>SiFive</i>	On-board	FE310	320MHz	16MB	16KB

Xilinx

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Aryt FPGA Dev Kit</i>	<i>SiFive</i>	On-board	FE310	450MHz	16MB	256MB

1.11.8 GigaDevice GD32V SDK

Configuration *framework* = gd32vf103-sdk

GigaDevice GD32VF103 Firmware Library (SDK)

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.

Name	Platform	MCU	Frequency	Flash	RAM
GD32VF103V-EVAL	GigaDevice GD32V	GD32VF103VBT6	108MHz	128KB	32KB
Sipeed Longan Nano	GigaDevice GD32V	GD32VF103CBT6	108MHz	128KB	32KB
Wio Lite RISC-V	GigaDevice GD32V	GD32VF103CBT6	108MHz	128KB	32KB

Examples

- GigaDevice GD32V SDK for GigaDevice GD32V

Platforms

Name	Description
GigaDevice GD32V	The GigaDevice GD32V device is a 32-bit general-purpose microcontroller based on the RISC-V core with an impressive balance of processing power, reduced power consumption and peripheral set.

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.
-

SeeedStudio

Name	Platform	Debug	MCU	Frequency	Flash	RAM
Wio Lite RISC-V	GigaDevice GD32V	External	GD32VF103CBT6	108MHz	128KB	32KB

Sipeed

Name	Platform	Debug	MCU	Frequency	Flash	RAM
GD32VF103V-EVAL	GigaDevice GD32V	External	GD32VF103VBT6	108MHz	128KB	32KB
Sipeed Longan Nano	GigaDevice GD32V	External	GD32VF103CBT6	108MHz	128KB	32KB

1.11.9 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](#)

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

Bands 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.

Name	Platform	MCU	Frequency	Flash	RAM
Sipeed MAIX BiT	Kendryte K210	K210	400MHz	16MB	6MB
Sipeed MAIX BiT with Mic	Kendryte K210	K210	400MHz	16MB	6MB
Sipeed MAIX GO	Kendryte K210	K210	400MHz	16MB	6MB
Sipeed MAIX ONE DOCK	Kendryte K210	K210	400MHz	16MB	6MB
Sipeed MAIXDUINO	Kendryte K210	K210	400MHz	16MB	6MB

Examples

- Kendryte FreeRTOS SDK for Kendryte K210

Platforms

Name	Description
<i>Kendryte K210</i>	Kendryte K210 is an AI capable RISC-V64 dual core SoC.

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

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Sipeed MAIX BiT</i>	<i>Kendryte K210</i>	External	K210	400MHz	16MB	6MB
<i>Sipeed MAIX BiT with Mic</i>	<i>Kendryte K210</i>	External	K210	400MHz	16MB	6MB
<i>Sipeed MAIX GO</i>	<i>Kendryte K210</i>	External	K210	400MHz	16MB	6MB
<i>Sipeed MAIX ONE DOCK</i>	<i>Kendryte K210</i>	External	K210	400MHz	16MB	6MB
<i>Sipeed MAIXDUINO</i>	<i>Kendryte K210</i>	External	K210	400MHz	16MB	6MB

1.11.10 Kendryte Standalone SDK

Configuration `framework = kendryte-standalone-sdk`

Kendryte Standalone SDK without OS support

For more detailed information please visit [vendor site](#).

Contents

- [Debugging](#)
- [Examples](#)
- [Platforms](#)
- [Boards](#)

Debugging

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.

Name	Platform	MCU	Frequency	Flash	RAM
Sipeed MAIX BiT	Kendryte K210	K210	400MHz	16MB	6MB
Sipeed MAIX BiT with Mic	Kendryte K210	K210	400MHz	16MB	6MB
Sipeed MAIX GO	Kendryte K210	K210	400MHz	16MB	6MB
Sipeed MAIX ONE DOCK	Kendryte K210	K210	400MHz	16MB	6MB
Sipeed MAIXDUINO	Kendryte K210	K210	400MHz	16MB	6MB

Examples

- Kendryte Standalone SDK for Kendryte K210

Platforms

Name	Description
Kendryte K210	Kendryte K210 is an AI capable RISC-V64 dual core SoC.

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

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Sipeed MAIX BiT</i>	<i>Kendryte K210</i>	External	K210	400MHz	16MB	6MB
<i>Sipeed MAIX BiT with Mic</i>	<i>Kendryte K210</i>	External	K210	400MHz	16MB	6MB
<i>Sipeed MAIX GO</i>	<i>Kendryte K210</i>	External	K210	400MHz	16MB	6MB
<i>Sipeed MAIX ONE DOCK</i>	<i>Kendryte K210</i>	External	K210	400MHz	16MB	6MB
<i>Sipeed MAIXDUINO</i>	<i>Kendryte K210</i>	External	K210	400MHz	16MB	6MB

1.11.11 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, TI 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](#).

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

Bands listed below have on-board debug probe and **ARE READY** for debugging! You do not need to use/buy external debug probe.

Name	Platform	MCU	Fre-quency	Flash	RAM
<i>ST Nucleo F103RB</i>	<i>ST STM32</i>	STM32F103RBT6	72MHz	128KB	20KB
<i>ST STM32F3DISCOVERY</i>	<i>ST STM32</i>	STM32F303VCT6	72MHz	256KB	48KB
<i>ST STM32F4DISCOVERY</i>	<i>ST STM32</i>	STM32F407VGT6	168MHz	1MB	128KB
<i>ST STM32LDiscovery</i>	<i>ST STM32</i>	STM32L152RBT6	32MHz	128KB	16KB
<i>TI LaunchPad (Stellaris) w/ lm4f120 (80MHz)</i>	<i>TI TIVA</i>	LPLM4F120H5QR	80MHz	256KB	32KB
<i>TI LaunchPad (Tiva C) w/ tm4c123 (80MHz)</i>	<i>TI TIVA</i>	LPTM4C1230C3PM	80MHz	256KB	32KB
<i>TI LaunchPad (Tiva C) w/ tm4c129 (120MHz)</i>	<i>TI TIVA</i>	LPTM4C1294NCPDT	120MHz	1MB	256KB

External Debug Tools

Bands 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.

Name	Platform	MCU	Frequency	Flash	RAM
<i>IBitsy</i>	<i>ST STM32</i>	STM32F415RGT	168MHz	1MB	128KB
<i>BlackPill F103C8</i>	<i>ST STM32</i>	STM32F103C8T6	72MHz	64KB	20KB
<i>BlackPill F103C8 (128k)</i>	<i>ST STM32</i>	STM32F103C8T6	72MHz	128KB	20KB
<i>BluePill F103C8</i>	<i>ST STM32</i>	STM32F103C8T6	72MHz	64KB	20KB
<i>BluePill F103C8 (128k)</i>	<i>ST STM32</i>	STM32F103C8T6	72MHz	128KB	20KB
<i>Maple</i>	<i>ST STM32</i>	STM32F103RBT6	72MHz	108KB	17KB
<i>Maple (RET6)</i>	<i>ST STM32</i>	STM32F103RET6	72MHz	256KB	48KB
<i>Maple Mini Bootloader 2.0</i>	<i>ST STM32</i>	STM32F103CBT6	72MHz	120KB	20KB
<i>Maple Mini Original</i>	<i>ST STM32</i>	STM32F103CBT6	72MHz	108KB	17KB
<i>STM32F103C8 (20k RAM, 64k Flash)</i>	<i>ST STM32</i>	STM32F103C8T6	72MHz	64KB	20KB
<i>STM32F103CB (20k RAM, 128k Flash)</i>	<i>ST STM32</i>	STM32F103CBT6	72MHz	128KB	20KB
<i>STM32F103R8 (20k RAM, 64 Flash)</i>	<i>ST STM32</i>	STM32F103R8T6	72MHz	64KB	20KB
<i>STM32F103RB (20k RAM, 128k Flash)</i>	<i>ST STM32</i>	STM32F103RBT6	72MHz	128KB	20KB
<i>STM32F103RC (48k RAM, 256k Flash)</i>	<i>ST STM32</i>	STM32F103RCT6	72MHz	256KB	48KB
<i>STM32F103RE (64k RAM, 512k Flash)</i>	<i>ST STM32</i>	STM32F103RET6	72MHz	512KB	64KB
<i>STM32F103T8 (20k RAM, 64k Flash)</i>	<i>ST STM32</i>	STM32F103T8T6	72MHz	20KB	64KB
<i>STM32F103TB (20k RAM, 128k Flash)</i>	<i>ST STM32</i>	STM32F103TBT6	72MHz	128KB	20KB
<i>STM32F103VB (20k RAM, 128k Flash)</i>	<i>ST STM32</i>	STM32F103VBT6	72MHz	128KB	20KB
<i>STM32F103VC (48k RAM, 256k Flash)</i>	<i>ST STM32</i>	STM32F103VCT6	72MHz	256KB	48KB
<i>STM32F103VD (64k RAM, 384k Flash)</i>	<i>ST STM32</i>	STM32F103VDT6	72MHz	384KB	64KB
<i>STM32F103VE (64k RAM, 512k Flash)</i>	<i>ST STM32</i>	STM32F103VET6	72MHz	512KB	64KB
<i>STM32F103ZC (48k RAM, 256k Flash)</i>	<i>ST STM32</i>	STM32F103ZCT6	72MHz	256KB	48KB
<i>STM32F103ZD (64k RAM, 384k Flash)</i>	<i>ST STM32</i>	STM32F103ZDT6	72MHz	384KB	64KB
<i>STM32F103ZE (64k RAM, 512k Flash)</i>	<i>ST STM32</i>	STM32F103ZET6	72MHz	512KB	64KB

Examples

- libOpenCM3 for ST STM32
- libOpenCM3 for TI TIVA

Platforms

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

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

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>1Bitsy</i>	<i>ST STM32</i>	External	STM32F415RGT	168MHz	1MB	128KB

Generic

Name	Platform	Debug	MCU	Fre-quency	Flash	RAM
<i>BlackPill F103C8</i>	<i>ST STM32</i>	External	STM32F103C8T6	72MHz	64KB	20KB
<i>BlackPill F103C8 (128k)</i>	<i>ST STM32</i>	External	STM32F103C8T6	72MHz	128KB	20KB
<i>BluePill F103C8</i>	<i>ST STM32</i>	External	STM32F103C8T6	72MHz	64KB	20KB
<i>BluePill F103C8 (128k)</i>	<i>ST STM32</i>	External	STM32F103C8T6	72MHz	128KB	20KB
<i>STM32F103C8 (20k RAM. 64k Flash)</i>	<i>ST STM32</i>	External	STM32F103C8T6	72MHz	64KB	20KB
<i>STM32F103CB (20k RAM. 128k Flash)</i>	<i>ST STM32</i>	External	STM32F103CBT6	72MHz	128KB	20KB
<i>STM32F103R8 (20k RAM. 64 Flash)</i>	<i>ST STM32</i>	External	STM32F103R8T6	72MHz	64KB	20KB
<i>STM32F103RB (20k RAM. 128k Flash)</i>	<i>ST STM32</i>	External	STM32F103RBT6	72MHz	128KB	20KB
<i>STM32F103RC (48k RAM. 256k Flash)</i>	<i>ST STM32</i>	External	STM32F103RCT6	72MHz	256KB	48KB
<i>STM32F103RE (64k RAM. 512k Flash)</i>	<i>ST STM32</i>	External	STM32F103RET6	72MHz	512KB	64KB
<i>STM32F103T8 (20k RAM. 64k Flash)</i>	<i>ST STM32</i>	External	STM32F103T8T6	72MHz	20KB	64KB
<i>STM32F103TB (20k RAM. 128k Flash)</i>	<i>ST STM32</i>	External	STM32F103TBT6	72MHz	128KB	20KB
<i>STM32F103VB (20k RAM. 128k Flash)</i>	<i>ST STM32</i>	External	STM32F103VBT6	72MHz	128KB	20KB
<i>STM32F103VC (48k RAM. 256k Flash)</i>	<i>ST STM32</i>	External	STM32F103VCT6	72MHz	256KB	48KB
<i>STM32F103VD (64k RAM. 384k Flash)</i>	<i>ST STM32</i>	External	STM32F103VDT6	72MHz	384KB	64KB
<i>STM32F103VE (64k RAM. 512k Flash)</i>	<i>ST STM32</i>	External	STM32F103VET6	72MHz	512KB	64KB
<i>STM32F103ZC (48k RAM. 256k Flash)</i>	<i>ST STM32</i>	External	STM32F103ZCT6	72MHz	256KB	48KB
<i>STM32F103ZD (64k RAM. 384k Flash)</i>	<i>ST STM32</i>	External	STM32F103ZDT6	72MHz	384KB	64KB
<i>STM32F103ZE (64k RAM. 512k Flash)</i>	<i>ST STM32</i>	External	STM32F103ZET6	72MHz	512KB	64KB

LeafLabs

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Maple</i>	<i>ST STM32</i>	External	STM32F103RBT6	72MHz	108KB	17KB
<i>Maple (RET6)</i>	<i>ST STM32</i>	External	STM32F103RET6	72MHz	256KB	48KB
<i>Maple Mini Bootloader 2.0</i>	<i>ST STM32</i>	External	STM32F103CBT6	72MHz	120KB	20KB
<i>Maple Mini Original</i>	<i>ST STM32</i>	External	STM32F103CBT6	72MHz	108KB	17KB

ST

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>ST Nucleo F103RB</i>	<i>ST STM32</i>	On-board	STM32F103RBT6	72MHz	128KB	20KB
<i>ST STM32F3DISCOVERY</i>	<i>ST STM32</i>	On-board	STM32F303VCT6	72MHz	256KB	48KB
<i>ST STM32F4DISCOVERY</i>	<i>ST STM32</i>	On-board	STM32F407VGT6	168MHz	1MB	128KB
<i>ST STM32LDISCOVERY</i>	<i>ST STM32</i>	On-board	STM32L152RBT6	32MHz	128KB	16KB

TI

Name	Platform	Debug	MCU	Fre- quency	Flash	RAM
<i>TI LaunchPad (Stellaris) w/ lm4f120 (80MHz)</i>	<i>TI TIVA</i>	On- board	LPLM4F120H5QR	80MHz	256KB	32KB
<i>TI LaunchPad (Tiva C) w/ tm4c123 (80MHz)</i>	<i>TI TIVA</i>	On- board	LPTM4C1230C3PM	80MHz	256KB	32KB
<i>TI LaunchPad (Tiva C) w/ tm4c129 (120MHz)</i>	<i>TI TIVA</i>	On- board	LPTM4C1294NCPDTI	120MHz	1MB	256KB

1.11.12 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](#).

Contents

- [Configuration](#)
- [Debugging](#)
- [Examples](#)
- [Platforms](#)
- [Boards](#)

Configuration

- Configuration system
- Mbed lib and Mbed OS 5
- Build profiles
- Ignoring particular components
- Custom Targets

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*):

Name	Description
<code>PIO_FRAMEWORK_MBED_RTOS_PRESENT</code>	Build the project with enabled <code>rtos</code>

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*):

Name	Description
<code>MBED_BUILD_PROFILE_RELEASE</code>	Release profile (smallest firmware, minimal error info)
<code>MBED_BUILD_PROFILE_DEBUG</code>	Debug profile (largest firmware, disabled sleep mode)

More information about differences between build profiles can be found on the official page [ARM Mbed OS Build Profiles](#).

Ignoring particular components

In case you don't need all parts of the framework or you want to reduce the compilation time, you can explicitly exclude folders with redundant sources. For example, to remove `cellular`, `mbedtls` and `nanostack` features from the build process, navigate to `packages_dir` and create a new file `framework-mbed/features/.mbedignore` with the following contents:

```
cellular/*
mbedtls/*
nanostack/*
```

If you want to exclude the entire folder, simply create `.mbedignore` file and add only one symbol `*` to this file.

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:

```
{
  "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"
    ]
  }
}
```

(continues on next page)

(continued from previous page)

```
],
"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
└── boards
    └── my_custom_board.json
├── src
    ├── main.cpp
    └── MY_CUSTOM_BOARD_TARGET
        ├── pinNames.h
        └── pinNames.c
└── custom_targets.json
    └── platformio.ini
```

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>

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

Banks listed below have on-board debug probe and **ARE READY** for debugging! You do not need to use/buy external debug probe.

Name	Platform	MCU	Frequency
<i>96Boards B96B-F446VE</i>	<i>ST STM32</i>	STM32F446VET6	168MHz
<i>ARM mbed LPC11U24 (+CAN)</i>	<i>NXP LPC</i>	LPC11U24	48MHz
<i>Atmel ATSAMR21-XPRO</i>	<i>Atmel SAM</i>	SAMR21G18A	48MHz
<i>Atmel ATSAMW25-XPRO</i>	<i>Atmel SAM</i>	SAMD21G18A	48MHz
<i>Atmel SAMD21-XPRO</i>	<i>Atmel SAM</i>	SAMD21J18A	48MHz
<i>Atmel SAML21-XPRO-B</i>	<i>Atmel SAM</i>	SAML21J18B	48MHz
<i>BBC micro:bit</i>	<i>Nordic nRF51</i>	NRF51822	16MHz
<i>Bambino-210E</i>	<i>NXP LPC</i>	LPC4330	204MHz
<i>CoCo-ri-Co!</i>	<i>NXP LPC</i>	LPC812	30MHz
<i>Delta DFBM-NQ620</i>	<i>Nordic nRF52</i>	NRF52832	64MHz
<i>Delta DFCM-NNN40</i>	<i>Nordic nRF51</i>	NRF51822	32MHz
<i>Delta DFCM-NNN50</i>	<i>Nordic nRF51</i>	NRF51822	32MHz
<i>EFM32GG-STK3700 Giant Gecko</i>	<i>Silicon Labs EFM32</i>	EFM32GG990F1024	48MHz
<i>EFM32LG-STK3600 Leopard Gecko</i>	<i>Silicon Labs EFM32</i>	EFM32LG990F256	48MHz
<i>EFM32WG-STK3800 Wonder Gecko</i>	<i>Silicon Labs EFM32</i>	EFM32WG990F256	48MHz
<i>EFM32ZG-STK3200 Zero Gecko</i>	<i>Silicon Labs EFM32</i>	EFM32ZG222F32	24MHz
<i>Embedded Artists LPC4088 Display Module</i>	<i>NXP LPC</i>	LPC4088	120MHz
<i>Embedded Artists LPC4088 QuickStart Board</i>	<i>NXP LPC</i>	LPC4088	120MHz
<i>Ethernet IoT Starter Kit</i>	<i>Freescale Kinetis</i>	MK64FN1M0VLL12	120MHz
<i>Freescale Kinetis FRDM-K20D50M</i>	<i>Freescale Kinetis</i>	MK20DX128VLH5	48MHz
<i>Freescale Kinetis FRDM-K22F</i>	<i>Freescale Kinetis</i>	MK22FN512VLH12	120MHz
<i>Freescale Kinetis FRDM-K64F</i>	<i>Freescale Kinetis</i>	MK64FN1M0VLL12	120MHz
<i>Freescale Kinetis FRDM-K66F</i>	<i>Freescale Kinetis</i>	MK66FN2M0VMD18	180MHz
<i>Freescale Kinetis FRDM-K82F</i>	<i>Freescale Kinetis</i>	MK82FN256VLL15	150MHz
<i>Freescale Kinetis FRDM-KL05Z</i>	<i>Freescale Kinetis</i>	MKL05Z32VFM4	48MHz
<i>Freescale Kinetis FRDM-KL25Z</i>	<i>Freescale Kinetis</i>	MKL25Z128VLK4	48MHz
<i>Freescale Kinetis FRDM-KL27Z</i>	<i>Freescale Kinetis</i>	MKL27Z64VLH4	48MHz
<i>Freescale Kinetis FRDM-KL43Z</i>	<i>Freescale Kinetis</i>	MKL43Z256VLH4	48MHz
<i>Freescale Kinetis FRDM-KL46Z</i>	<i>Freescale Kinetis</i>	MKL46Z256VLL4	48MHz
<i>Freescale Kinetis FRDM-KW41Z</i>	<i>Freescale Kinetis</i>	MKW41Z512VHT4	48MHz
<i>GAPuino GAP8</i>	<i>RISC-V GAP</i>	GAP8	250MHz
<i>JKSoft Wallbot BLE</i>	<i>Nordic nRF51</i>	NRF51822	16MHz
<i>L476DMWIK</i>	<i>ST STM32</i>	STM32L476VGT6	80MHz
<i>LPCXpresso11U68</i>	<i>NXP LPC</i>	LPC11U68	50MHz

Table 21 – continued from previous page

Name	Platform	MCU	Frequency
LPCXpresso824-MAX	NXP LPC	LPC824	30MHz
Maxim ARM mbed Enabled Development Platform for MAX32600	Maxim 32	MAX32600	24MHz
Mbed Connect Cloud	ST STM32	STM32F439ZIY6	168MHz
NXP LPC800-MAX	NXP LPC	LPC812	30MHz
NXP LPCXpresso54114	NXP LPC	LPC54114J256BD64	100MHz
NXP LPCXpresso54608	NXP LPC	LPC54608ET512	180MHz
NXP mbed LPC11U24	NXP LPC	LPC11U24	48MHz
NXP mbed LPC1768	NXP LPC	LPC1768	96MHz
Nordic nRF51 Dongle (PCA10031)	Nordic nRF51	NRF51822	32MHz
Nordic nRF51822-mKIT	Nordic nRF51	NRF51822	16MHz
Nordic nRF51X22 Development Kit(PCA1000X)	Nordic nRF51	NRF51822	32MHz
Nordic nRF52-DK	Nordic nRF52	NRF52832	64MHz
Nordic nRF52840-DK	Nordic nRF52	NRF52840	64MHz
RedBearLab BLE Nano 1.5	Nordic nRF51	NRF51822	16MHz
RedBearLab BLE Nano 2	Nordic nRF52	NRF52832	64MHz
RedBearLab nRF51822	Nordic nRF51	NRF51822	16MHz
RushUp Cloud-JAM	ST STM32	STM32F401RET6	84MHz
RushUp Cloud-JAM L4	ST STM32	STM32L476RGT6	80MHz
SLSTK3400A USB-enabled Happy Gecko	Silicon Labs EFM32	EFM32HG322F64	25MHz
SLSTK3401A Pearl Gecko PG1	Silicon Labs EFM32	EFM32PG1B200F256GM48	40MHz
ST 32F3348DISCOVERY	ST STM32	STM32F334C8T6	72MHz
ST 32F401CDISCOVERY	ST STM32	STM32F401VCT6	84MHz
ST 32F413HDISCOVERY	ST STM32	STM32F413ZHT6	100MHz
ST 32F429IDISCOVERY	ST STM32	STM32F429ZIT6	180MHz
ST 32F469IDISCOVERY	ST STM32	STM32F469NIH6	180MHz
ST 32F746GDISCOVERY	ST STM32	STM32F746NGH6	216MHz
ST 32F769IDISCOVERY	ST STM32	STM32F769NIH6	216MHz
ST 32L0538DISCOVERY	ST STM32	STM32L053C8T6	32MHz
ST 32L476GDISCOVERY	ST STM32	STM32L476VGT6	80MHz
ST 32L496GDISCOVERY	ST STM32	STM32L496AGI6	80MHz
ST DISCO-L072CZ-LRWAN1	ST STM32	STM32L072CZ	32MHz
ST DISCO-L475VG-IOT01A	ST STM32	STM32L475VGT6	80MHz
ST Discovery F072RB	ST STM32	STM32F072RBT6	48MHz
ST Nucleo F030R8	ST STM32	STM32F030R8T6	48MHz
ST Nucleo F031K6	ST STM32	STM32F031K6T6	48MHz
ST Nucleo F042K6	ST STM32	STM32F042K6T6	48MHz
ST Nucleo F070RB	ST STM32	STM32F070RBT6	48MHz
ST Nucleo F072RB	ST STM32	STM32F072RBT6	48MHz
ST Nucleo F091RC	ST STM32	STM32F091RCT6	48MHz
ST Nucleo F103RB	ST STM32	STM32F103RBT6	72MHz
ST Nucleo F207ZG	ST STM32	STM32F207ZGT6	120MHz
ST Nucleo F302R8	ST STM32	STM32F302R8T6	72MHz
ST Nucleo F303K8	ST STM32	STM32F303K8T6	72MHz
ST Nucleo F303RE	ST STM32	STM32F303RET6	72MHz
ST Nucleo F303ZE	ST STM32	STM32F303ZET6	72MHz
ST Nucleo F334R8	ST STM32	STM32F334R8T6	72MHz
ST Nucleo F401RE	ST STM32	STM32F401RET6	84MHz
ST Nucleo F410RB	ST STM32	STM32F410RBT6	100MHz
ST Nucleo F411RE	ST STM32	STM32F411RET6	100MHz

Table 21 – continued from previous page

Name	Platform	MCU	Frequency
<i>ST Nucleo F412ZG</i>	<i>ST STM32</i>	STM32F412ZGT6	100MHz
<i>ST Nucleo F413ZH</i>	<i>ST STM32</i>	STM32F413ZHT6	100MHz
<i>ST Nucleo F429ZI</i>	<i>ST STM32</i>	STM32F429ZIT6	180MHz
<i>ST Nucleo F439ZI</i>	<i>ST STM32</i>	STM32F439ZIT6	180MHz
<i>ST Nucleo F446RE</i>	<i>ST STM32</i>	STM32F446RET6	180MHz
<i>ST Nucleo F446ZE</i>	<i>ST STM32</i>	STM32F446ZET6	180MHz
<i>ST Nucleo F746ZG</i>	<i>ST STM32</i>	STM32F746ZGT6	216MHz
<i>ST Nucleo F756ZG</i>	<i>ST STM32</i>	STM32F756ZG	216MHz
<i>ST Nucleo F767ZI</i>	<i>ST STM32</i>	STM32F767ZIT6	216MHz
<i>ST Nucleo L031K6</i>	<i>ST STM32</i>	STM32L031K6T6	32MHz
<i>ST Nucleo L053R8</i>	<i>ST STM32</i>	STM32L053R8T6	32MHz
<i>ST Nucleo L073RZ</i>	<i>ST STM32</i>	STM32L073RZ	32MHz
<i>ST Nucleo L152RE</i>	<i>ST STM32</i>	STM32L152RET6	32MHz
<i>ST Nucleo L432KC</i>	<i>ST STM32</i>	STM32L432KCU6	80MHz
<i>ST Nucleo L433RC-P</i>	<i>ST STM32</i>	STM32L433RC	80MHz
<i>ST Nucleo L476RG</i>	<i>ST STM32</i>	STM32L476RGT6	80MHz
<i>ST Nucleo L486RG</i>	<i>ST STM32</i>	STM32L486RGT6	80MHz
<i>ST Nucleo L496ZG</i>	<i>ST STM32</i>	STM32L496ZGT6	80MHz
<i>ST Nucleo L496ZG-P</i>	<i>ST STM32</i>	STM32L496ZGT6P	80MHz
<i>ST Nucleo L4R5ZI</i>	<i>ST STM32</i>	STM32L4R5ZIT6	120MHz
<i>ST STM32F0308DISCOVERY</i>	<i>ST STM32</i>	STM32F030R8T6	48MHz
<i>ST STM32F0DISCOVERY</i>	<i>ST STM32</i>	STM32F051R8T6	48MHz
<i>ST STM32F3DISCOVERY</i>	<i>ST STM32</i>	STM32F303VCT6	72MHz
<i>ST STM32F4DISCOVERY</i>	<i>ST STM32</i>	STM32F407VGT6	168MHz
<i>ST STM32VLDISCOVERY</i>	<i>ST STM32</i>	STM32F100RBT6	24MHz
<i>ST Sensor Node</i>	<i>ST STM32</i>	STM32L476JG	80MHz
<i>Seeed Arch BLE</i>	<i>Nordic nRF51</i>	NRF51822	16MHz
<i>Seeed Arch Link</i>	<i>Nordic nRF51</i>	NRF51822	16MHz
<i>Seeed Arch Max</i>	<i>ST STM32</i>	STM32F407VET6	168MHz
<i>Seeed Arch Pro</i>	<i>NXP LPC</i>	LPC1768	96MHz
<i>Seeed Tiny BLE</i>	<i>Nordic nRF51</i>	NRF51822	16MHz
<i>Seeed Wio 3G</i>	<i>ST STM32</i>	STM32F439VI	180MHz
<i>Switch Science mbed HRM1017</i>	<i>Nordic nRF51</i>	NRF51822	16MHz
<i>Switch Science mbed LPC1114FN28</i>	<i>NXP LPC</i>	LPC1114FN28	48MHz
<i>Switch Science mbed LPC824</i>	<i>NXP LPC</i>	LPC824	30MHz
<i>Switch Science mbed TY51822r3</i>	<i>Nordic nRF51</i>	NRF51822	32MHz
<i>Thunderboard Sense 2 Sensor-to-Cloud Advanced IoT</i>	<i>Silicon Labs EFM32</i>	EFR32MG12P432F1024	40MHz
<i>VNG VBLUNO51</i>	<i>Nordic nRF51</i>	NRF51822	16MHz
<i>WIZwiki-W7500</i>	<i>WIZNet W7500</i>	WIZNET7500	48MHz
<i>WIZwiki-W7500ECO</i>	<i>WIZNet W7500</i>	WIZNET7500ECO	48MHz
<i>WIZwiki-W7500P</i>	<i>WIZNet W7500</i>	WIZNET7500P	48MHz
<i>sakura.io Evaluation Board</i>	<i>ST STM32</i>	STM32F411RET6	100MHz
<i>u-blox C027</i>	<i>NXP LPC</i>	LPC1768	96MHz
<i>u-blox C030-R410M IoT</i>	<i>ST STM32</i>	STM32F437VG	180MHz
<i>u-blox EVK-NINA-B1</i>	<i>Nordic nRF52</i>	NRF52832	64MHz
<i>y5 nRF51822 mbug</i>	<i>Nordic nRF51</i>	NRF51822	16MHz

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.

Name	Platform	MCU	Frequency	Flash	RAM
BluePill F103C8	ST STM32	STM32F103C8T6	72MHz	64KB	20KB
CQ Publishing TG-LPC11U35-501	NXP LPC	LPC11U35	48MHz	64KB	10KB
DipCortex M3	NXP LPC	LPC1347	72MHz	64KB	12KB
EA LPC11U35 QuickStart Board	NXP LPC	LPC11U35	48MHz	64KB	10KB
Espotel LoRa Module	ST STM32	STM32F411RET6	100MHz	512KB	128KB
Freescale Kinetis FRDM-KL82Z	Freescale Kinetis	MKL82Z128VLK7	96MHz	128KB	96KB
Freescale Kinetis FRDM-KW24D512	Freescale Kinetis	MKW24D512	50MHz	512KB	64KB
Hexiwear	Freescale Kinetis	MK64FN1M0VDC12	120MHz	1MB	256KB
MAX32620FTHR	Maxim 32	MAX32620FTHR	96MHz	2MB	256KB
MTS Dragonfly	ST STM32	STM32F411RET6	100MHz	512KB	128KB
Maxim Health Sensor Platform	Maxim 32	MAX32620	96MHz	2MB	256KB
Maxim Wireless Sensor Node Demonstrator	Maxim 32	MAX32610	24MHz	256KB	32KB
MultiTech mDot	ST STM32	STM32F411RET6	100MHz	512KB	128KB
MultiTech mDot F411	ST STM32	STM32F411RET6	100MHz	512KB	128KB
MultiTech xDot	ST STM32	STM32L151CCU6	32MHz	256KB	32KB
NAMote72	ST STM32	STM32L152RC	32MHz	256KB	32KB
NGX Technologies BlueBoard-LPC11U24	NXP LPC	LPC11U24	48MHz	32KB	8KB
NXP LPC11C24	NXP LPC	LPC11C24	48MHz	32KB	8KB
NXP LPC11U34	NXP LPC	LPC11U34	48MHz	40KB	8KB
NXP LPC11U37	NXP LPC	LPC11U37	48MHz	128KB	10KB
NXP LPCXpresso1549	NXP LPC	LPC1549	72MHz	256KB	36KB
SDT52832B	Nordic nRF52	NRF52832	64MHz	512KB	64KB
STM32F103C8 (20k RAM, 64k Flash)	ST STM32	STM32F103C8T6	72MHz	64KB	20KB
STM32F103RB (20k RAM, 128k Flash)	ST STM32	STM32F103RBT6	72MHz	128KB	20KB
Solder Splash Labs DipCortex M0	NXP LPC	LPC11U24	50MHz	32KB	8KB
Teensy 3.1 / 3.2	Teensy	MK20DX256	72MHz	256KB	64KB
u-blox C030-N211 IoT Starter Kit	ST STM32	STM32F437VG	180MHz	1MB	256KB
u-blox C030-U201 IoT Starter Kit	ST STM32	STM32F437VG	180MHz	1MB	256KB
u-blox EVK-ODIN-W2	ST STM32	STM32F439ZIY6	168MHz	2MB	256KB
u-blox ODIN-W2	ST STM32	STM32F439ZIY6	168MHz	2MB	256KB
y5 LPC11U35 mbug	NXP LPC	LPC11U35	48MHz	64KB	10KB

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

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

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.
-

96Boards

Name	Platform	Debug	MCU	Frequency	Flash	RAM
96Boards B96B-F446VE	ST STM32	On-board	STM32F446VET6	168MHz	512KB	128KB

AppNearMe

Name	Platform	Debug	MCU	Frequency	Flash	RAM
MicroNFCBoard	NXP LPC	No	LPC11U34	48MHz	48KB	10KB

Atmel

Name	Platform	Debug	MCU	Frequency	Flash	RAM
Atmel ATSAMR21-XPRO	Atmel SAM	On-board	SAMR21G18A	48MHz	256KB	32KB
Atmel ATSAMW25-XPRO	Atmel SAM	On-board	SAMD21G18A	48MHz	256KB	32KB
Atmel SAMD21-XPRO	Atmel SAM	On-board	SAMD21J18A	48MHz	256KB	32KB
Atmel SAML21-XPRO-B	Atmel SAM	On-board	SAML21J18B	48MHz	256KB	32KB

Avnet Silica

Name	Platform	Debug	MCU	Frequency	Flash	RAM
ST Sensor Node	ST STM32	On-board	STM32L476JG	80MHz	1MB	128KB

BBC

Name	Platform	Debug	MCU	Frequency	Flash	RAM
BBC micro:bit	Nordic nRF51	On-board	NRF51822	16MHz	256KB	16KB

CQ Publishing

Name	Platform	Debug	MCU	Frequency	Flash	RAM
CQ Publishing TG-LPC11U35-501	NXP LPC	External	LPC11U35	48MHz	64KB	10KB

Delta

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Delta DFBM-NQ620</i>	<i>Nordic nRF52</i>	On-board	NRF52832	64MHz	512KB	64KB
<i>Delta DFCM-NNN40</i>	<i>Nordic nRF51</i>	On-board	NRF51822	32MHz	256KB	32KB
<i>Delta DFCM-NNN50</i>	<i>Nordic nRF51</i>	On-board	NRF51822	32MHz	256KB	16KB

Elektor Labs

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>CoCo-ri-Co!</i>	<i>NXP LPC</i>	On-board	LPC812	30MHz	16KB	4KB

Embedded Artists

Name	Platform	Debug	MCU	Fre- quency	Flash	RAM
<i>EA LPC11U35 QuickStart Board</i>	<i>NXP LPC</i>	External	LPC11U35	48MHz	64KB	10KB
<i>Embedded Artists LPC4088 Display Mod- ule</i>	<i>NXP LPC</i>	On- board	LPC4088	120MHz	512KB	96KB
<i>Embedded Artists LPC4088 QuickStart Board</i>	<i>NXP LPC</i>	On- board	LPC4088	120MHz	512KB	96KB

Espotel

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Espotel LoRa Module</i>	<i>ST STM32</i>	External	STM32F411RET6	100MHz	512KB	128KB

Freescale

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Ethernet IoT Starter Kit</i>	<i>Freescale Kinetis</i>	On-board	MK64FN1M0VLL12	120MHz	1MB	256KB
<i>Freescale Kinetis FRDM-K20D50M</i>	<i>Freescale Kinetis</i>	On-board	MK20DX128VLH5	48MHz	128KB	16KB
<i>Freescale Kinetis FRDM-K22F</i>	<i>Freescale Kinetis</i>	On-board	MK22FN512VLH12	120MHz	512KB	128KB
<i>Freescale Kinetis FRDM-K64F</i>	<i>Freescale Kinetis</i>	On-board	MK64FN1M0VLL12	120MHz	1MB	256KB
<i>Freescale Kinetis FRDM-K66F</i>	<i>Freescale Kinetis</i>	On-board	MK66FN2M0VMD18I	80MHz	2MB	256KB
<i>Freescale Kinetis FRDM-K82F</i>	<i>Freescale Kinetis</i>	On-board	MK82FN256VLL15	150MHz	256KB	256KB
<i>Freescale Kinetis FRDM-KL05Z</i>	<i>Freescale Kinetis</i>	On-board	MKL05Z32VFM4	48MHz	32KB	4KB
<i>Freescale Kinetis FRDM-KL25Z</i>	<i>Freescale Kinetis</i>	On-board	MKL25Z128VLK4	48MHz	128KB	16KB
<i>Freescale Kinetis FRDM-KL27Z</i>	<i>Freescale Kinetis</i>	On-board	MKL27Z64VLH4	48MHz	64KB	16KB
<i>Freescale Kinetis FRDM-KL43Z</i>	<i>Freescale Kinetis</i>	On-board	MKL43Z256VLH4	48MHz	256KB	32KB
<i>Freescale Kinetis FRDM-KL46Z</i>	<i>Freescale Kinetis</i>	On-board	MKL46Z256VLL4	48MHz	256KB	32KB
<i>Freescale Kinetis FRDM-KL82Z</i>	<i>Freescale Kinetis</i>	External	MKL82Z128VLK7	96MHz	128KB	96KB
<i>Freescale Kinetis FRDM-KW24D512</i>	<i>Freescale Kinetis</i>	External	MKW24D512	50MHz	512KB	64KB
<i>Freescale Kinetis FRDM-KW41Z</i>	<i>Freescale Kinetis</i>	On-board	MKW41Z512VHT4	48MHz	512KB	128KB

GHI Electronics

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>mBuino</i>	<i>NXP LPC</i>	No	LPC11U24	50MHz	32KB	10KB

Generic

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>BluePill F103C8</i>	<i>ST STM32</i>	External	STM32F103C8T6	72MHz	64KB	20KB
<i>STM32F103C8 (20k RAM. 64k Flash)</i>	<i>ST STM32</i>	External	STM32F103C8T6	72MHz	64KB	20KB
<i>STM32F103RB (20k RAM. 128k Flash)</i>	<i>ST STM32</i>	External	STM32F103RBT6	72MHz	128KB	20KB

GreenWaves Technologies

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>GAPuino GAP8</i>	<i>RISC-V GAP</i>	On-board	GAP8	250MHz	64MB	8MB

JKSoft

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>JKSoft Wallbot BLE</i>	<i>Nordic nRF51</i>	On-board	NRF51822	16MHz	128KB	16KB

Maxim

Name	Platform	De-bug	MCU	Fre-quency	Flash	RAM
<i>MAX32620FTHR</i>	<i>Maxim 32</i>	External	MAX32620FT	96MHz	2MB	256KB
<i>MAX32625MBED</i>	<i>Maxim 32</i>	No	MAX32625	96MHz	512KB	160KB
<i>MAX32625NEXPAQ</i>	<i>Maxim 32</i>	No	MAX32625	96MHz	512KB	160KB
<i>MAX32625PICO</i>	<i>Maxim 32</i>	No	MAX32625	96MHz	512KB	160KB
<i>Maxim ARM mbed Enabled Development Platform for MAX32600</i>	<i>Maxim 32</i>	On-board	MAX32600	24MHz	256KB	32KB
<i>Maxim Health Sensor Platform</i>	<i>Maxim 32</i>	External	MAX32620	96MHz	2MB	256KB
<i>Maxim MAX32630FTHR Application Platform</i>	<i>Maxim 32</i>	No	MAX32630	96MHz	2MB	512KB
<i>Maxim Wireless Sensor Node Demonstrator</i>	<i>Maxim 32</i>	External	MAX32610	24MHz	256KB	32KB

Micromint

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Bambino-210E</i>	<i>NXP LPC</i>	On-board	LPC4330	204MHz	8MB	264KB

MikroElektronika

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Hexiwear</i>	<i>Freescale Kinetis</i>	External	MK64FN1M0VDC12	120MHz	1MB	256KB

Multitech

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>MTS Dragonfly</i>	<i>ST STM32</i>	External	STM32F411RET6	100MHz	512KB	128KB
<i>MultiTech mDot</i>	<i>ST STM32</i>	External	STM32F411RET6	100MHz	512KB	128KB
<i>MultiTech mDot F411</i>	<i>ST STM32</i>	External	STM32F411RET6	100MHz	512KB	128KB
<i>MultiTech xDot</i>	<i>ST STM32</i>	External	STM32L151CCU6	32MHz	256KB	32KB

NGX Technologies

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>NGX Technologies BlueBoard-LPC11U24</i>	<i>NXP LPC</i>	External	LPC11U24	48MHz	32KB	8KB

NXP

Name	Platform	Debug	MCU	Fre- quency	Flash	RAM
<i>ARM mbed LPC11U24 (+CAN)</i>	<i>NXP LPC</i>	On- board	LPC11U24	48MHz	32KB	8KB
<i>LPCXpresso11U68</i>	<i>NXP LPC</i>	On- board	LPC11U68	50MHz	256KB	36KB
<i>LPCXpresso824-MAX</i>	<i>NXP LPC</i>	On- board	LPC824	30MHz	32KB	8KB
<i>NXP LPC11C24</i>	<i>NXP LPC</i>	External	LPC11C24	48MHz	32KB	8KB
<i>NXP LPC11U34</i>	<i>NXP LPC</i>	External	LPC11U34	48MHz	40KB	8KB
<i>NXP LPC11U37</i>	<i>NXP LPC</i>	External	LPC11U37	48MHz	128KB	10KB
<i>NXP LPC800-MAX</i>	<i>NXP LPC</i>	On- board	LPC812	30MHz	16KB	4KB
<i>NXP LPCXpresso1549</i>	<i>NXP LPC</i>	External	LPC1549	72MHz	256KB	36KB
<i>NXP LPCXpresso54114</i>	<i>NXP LPC</i>	On- board	LPC54114J256BD64	100MHz	256KB	192KB
<i>NXP LPCXpresso54608</i>	<i>NXP LPC</i>	On- board	LPC54608ET512	180MHz	512KB	200KB
<i>NXP mbed LPC11U24</i>	<i>NXP LPC</i>	On- board	LPC11U24	48MHz	32KB	8KB
<i>NXP mbed LPC1768</i>	<i>NXP LPC</i>	On- board	LPC1768	96MHz	512KB	64KB

Nordic

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Nordic nRF51 Dongle (PCA10031)</i>	<i>Nordic nRF51</i>	On-board	NRF51822	32MHz	256KB	32KB
<i>Nordic nRF51822-mKIT</i>	<i>Nordic nRF51</i>	On-board	NRF51822	16MHz	128KB	16KB
<i>Nordic nRF51X22 Development Kit(PCA1000X)</i>	<i>Nordic nRF51</i>	On-board	NRF51822	32MHz	256KB	32KB
<i>Nordic nRF52-DK</i>	<i>Nordic nRF52</i>	On-board	NRF52832	64MHz	512KB	64KB
<i>Nordic nRF52840-DK</i>	<i>Nordic nRF52</i>	On-board	NRF52840	64MHz	1MB	256KB

Outrageous Circuits

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Outrageous Circuits mBuino</i>	<i>NXP LPC</i>	No	LPC11U24	48MHz	32KB	8KB

RedBearLab

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>RedBearLab BLE Nano 1.5</i>	<i>Nordic nRF51</i>	On-board	NRF51822	16MHz	256KB	32KB
<i>RedBearLab BLE Nano 2</i>	<i>Nordic nRF52</i>	On-board	NRF52832	64MHz	512KB	64KB
<i>RedBearLab nRF51822</i>	<i>Nordic nRF51</i>	On-board	NRF51822	16MHz	256KB	16KB

RushUp

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>RushUp Cloud-JAM</i>	<i>ST STM32</i>	On-board	STM32F401RET6	84MHz	512KB	96KB
<i>RushUp Cloud-JAM L4</i>	<i>ST STM32</i>	On-board	STM32L476RGTE6	80MHz	1MB	128KB

ST

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>ST 32F3348DISCOVERY</i>	<i>ST STM32</i>	On-board	STM32F334C8T6	72MHz	64KB	12KB
<i>ST 32F401CDISCOVERY</i>	<i>ST STM32</i>	On-board	STM32F401VCT6	84MHz	256KB	64KB
<i>ST 32F413HDISCOVERY</i>	<i>ST STM32</i>	On-board	STM32F413ZHT6	100MHz	512KB	128KB
<i>ST 32F429IDISCOVERY</i>	<i>ST STM32</i>	On-board	STM32F429ZIT6	180MHz	2MB	256KB
<i>ST 32F469IDISCOVERY</i>	<i>ST STM32</i>	On-board	STM32F469NIH6	180MHz	1MB	384KB
<i>ST 32F746GDISCOVERY</i>	<i>ST STM32</i>	On-board	STM32F746NGH6	216MHz	1MB	320KB
<i>ST 32F769IDISCOVERY</i>	<i>ST STM32</i>	On-board	STM32F769NIH6	216MHz	1MB	512KB

Continued on next page

Table 23 – continued from previous page

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>ST 32L0538DISCOVERY</i>	<i>ST STM32</i>	On-board	STM32L053C8T6	32MHz	64KB	8KB
<i>ST 32L476GDISCOVERY</i>	<i>ST STM32</i>	On-board	STM32L476VGT6	80MHz	1MB	128KB
<i>ST 32L496GDISCOVERY</i>	<i>ST STM32</i>	On-board	STM32L496AGI6	80MHz	1MB	320KB
<i>ST DISCO-L072CZ-LRWAN1</i>	<i>ST STM32</i>	On-board	STM32L072CZ	32MHz	192KB	20KB
<i>ST DISCO-L475VG-IOT01A</i>	<i>ST STM32</i>	On-board	STM32L475VGT6	80MHz	1MB	128KB
<i>ST Discovery F072RB</i>	<i>ST STM32</i>	On-board	STM32F072RBT6	48MHz	128KB	16KB
<i>ST Nucleo F030R8</i>	<i>ST STM32</i>	On-board	STM32F030R8T6	48MHz	64KB	8KB
<i>ST Nucleo F031K6</i>	<i>ST STM32</i>	On-board	STM32F031K6T6	48MHz	32KB	4KB
<i>ST Nucleo F042K6</i>	<i>ST STM32</i>	On-board	STM32F042K6T6	48MHz	32KB	6KB
<i>ST Nucleo F070RB</i>	<i>ST STM32</i>	On-board	STM32F070RBT6	48MHz	128KB	16KB
<i>ST Nucleo F072RB</i>	<i>ST STM32</i>	On-board	STM32F072RBT6	48MHz	128KB	16KB
<i>ST Nucleo F091RC</i>	<i>ST STM32</i>	On-board	STM32F091RCT6	48MHz	256KB	32KB
<i>ST Nucleo F103RB</i>	<i>ST STM32</i>	On-board	STM32F103RBT6	72MHz	128KB	20KB
<i>ST Nucleo F207ZG</i>	<i>ST STM32</i>	On-board	STM32F207ZGT6	120MHz	1MB	128KB
<i>ST Nucleo F302R8</i>	<i>ST STM32</i>	On-board	STM32F302R8T6	72MHz	64KB	16KB
<i>ST Nucleo F303K8</i>	<i>ST STM32</i>	On-board	STM32F303K8T6	72MHz	64KB	12KB
<i>ST Nucleo F303RE</i>	<i>ST STM32</i>	On-board	STM32F303RET6	72MHz	512KB	64KB
<i>ST Nucleo F303ZE</i>	<i>ST STM32</i>	On-board	STM32F303ZET6	72MHz	512KB	64KB
<i>ST Nucleo F334R8</i>	<i>ST STM32</i>	On-board	STM32F334R8T6	72MHz	64KB	16KB
<i>ST Nucleo F401RE</i>	<i>ST STM32</i>	On-board	STM32F401RET6	84MHz	512KB	96KB
<i>ST Nucleo F410RB</i>	<i>ST STM32</i>	On-board	STM32F410RBT6	100MHz	128KB	32KB
<i>ST Nucleo F411RE</i>	<i>ST STM32</i>	On-board	STM32F411RET6	100MHz	512KB	128KB
<i>ST Nucleo F412ZG</i>	<i>ST STM32</i>	On-board	STM32F412ZGT6	100MHz	1MB	256KB
<i>ST Nucleo F413ZH</i>	<i>ST STM32</i>	On-board	STM32F413ZH6	100MHz	512KB	128KB
<i>ST Nucleo F429ZI</i>	<i>ST STM32</i>	On-board	STM32F429ZIT6	180MHz	2MB	256KB
<i>ST Nucleo F439ZI</i>	<i>ST STM32</i>	On-board	STM32F439ZIT6	180MHz	2MB	256KB
<i>ST Nucleo F446RE</i>	<i>ST STM32</i>	On-board	STM32F446RET6	180MHz	512KB	128KB
<i>ST Nucleo F446ZE</i>	<i>ST STM32</i>	On-board	STM32F446ZET6	180MHz	512KB	128KB
<i>ST Nucleo F746ZG</i>	<i>ST STM32</i>	On-board	STM32F746ZGT6	216MHz	1MB	320KB
<i>ST Nucleo F756ZG</i>	<i>ST STM32</i>	On-board	STM32F756ZG	216MHz	1MB	320KB
<i>ST Nucleo F767ZI</i>	<i>ST STM32</i>	On-board	STM32F767ZIT6	216MHz	2MB	512KB
<i>ST Nucleo L031K6</i>	<i>ST STM32</i>	On-board	STM32L031K6T6	32MHz	32KB	8KB
<i>ST Nucleo L053R8</i>	<i>ST STM32</i>	On-board	STM32L053R8T6	32MHz	64KB	8KB
<i>ST Nucleo L073RZ</i>	<i>ST STM32</i>	On-board	STM32L073RZ	32MHz	192KB	20KB
<i>ST Nucleo L152RE</i>	<i>ST STM32</i>	On-board	STM32L152RET6	32MHz	512KB	80KB
<i>ST Nucleo L432KC</i>	<i>ST STM32</i>	On-board	STM32L432KCU6	80MHz	256KB	64KB
<i>ST Nucleo L433RC-P</i>	<i>ST STM32</i>	On-board	STM32L433RC	80MHz	256KB	64KB
<i>ST Nucleo L476RG</i>	<i>ST STM32</i>	On-board	STM32L476RGT6	80MHz	1MB	128KB
<i>ST Nucleo L486RG</i>	<i>ST STM32</i>	On-board	STM32L486RGT6	80MHz	1MB	128KB
<i>ST Nucleo L496ZG</i>	<i>ST STM32</i>	On-board	STM32L496ZGT6	80MHz	1MB	128KB
<i>ST Nucleo L496ZG-P</i>	<i>ST STM32</i>	On-board	STM32L496ZGT6P	80MHz	1MB	320KB
<i>ST Nucleo L4R5ZI</i>	<i>ST STM32</i>	On-board	STM32L4R5ZIT6	120MHz	2MB	640KB
<i>ST STM32F0308DISCOVERY</i>	<i>ST STM32</i>	On-board	STM32F030R8T6	48MHz	64KB	8KB
<i>ST STM32F0DISCOVERY</i>	<i>ST STM32</i>	On-board	STM32F051R8T6	48MHz	64KB	8KB
<i>ST STM32F3DISCOVERY</i>	<i>ST STM32</i>	On-board	STM32F303VCT6	72MHz	256KB	48KB
<i>ST STM32F4DISCOVERY</i>	<i>ST STM32</i>	On-board	STM32F407VGT6	168MHz	1MB	128KB
<i>ST STM32VLDiscovery</i>	<i>ST STM32</i>	On-board	STM32F100RBT6	24MHz	128KB	8KB

SeeedStudio

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Seeed Arch BLE</i>	<i>Nordic nRF51</i>	On-board	NRF51822	16MHz	128KB	16KB
<i>Seeed Arch GPRS V2</i>	<i>NXP LPC</i>	No	LPC11U37	48MHz	128KB	10KB
<i>Seeed Arch Link</i>	<i>Nordic nRF51</i>	On-board	NRF51822	16MHz	256KB	16KB
<i>Seeed Arch Max</i>	<i>ST STM32</i>	On-board	STM32F407VET6	168MHz	512KB	192KB
<i>Seeed Arch Pro</i>	<i>NXP LPC</i>	On-board	LPC1768	96MHz	512KB	64KB
<i>Seeed Tiny BLE</i>	<i>Nordic nRF51</i>	On-board	NRF51822	16MHz	256KB	16KB
<i>Seeed Wio 3G</i>	<i>ST STM32</i>	On-board	STM32F439VI	180MHz	2MB	256KB
<i>Seeed Xadow M0</i>	<i>NXP LPC</i>	No	LPC11U35	48MHz	64KB	10KB

Semtech

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>NAMote72</i>	<i>ST STM32</i>	External	STM32L152RC	32MHz	256KB	32KB

Sigma Delta Technologies

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>SDT32620B</i>	<i>Maxim 32</i>	No	MAX32620IWG	96MHz	2MB	256KB
<i>SDT32625B</i>	<i>Maxim 32</i>	No	MAX32625ITK	96MHz	512KB	160KB
<i>SDT52832B</i>	<i>Nordic nRF52</i>	External	NRF52832	64MHz	512KB	64KB

Silicon Labs

Name	Platform	De- bug	MCU	Fre- quency	Flash	RAM
<i>EFM32GG-STK3700 Giant Gecko</i>	<i>Silicon Labs EFM32</i>	On- board	EFM32GG990F1024	48MHz	1MB	128KB
<i>EFM32LG-STK3600 Leopard Gecko</i>	<i>Silicon Labs EFM32</i>	On- board	EFM32LG990F256	48MHz	256KB	32KB
<i>EFM32WG-STK3800 Wonder Gecko</i>	<i>Silicon Labs EFM32</i>	On- board	EFM32WG990F256	48MHz	256KB	32KB
<i>EFM32ZG-STK3200 Zero Gecko</i>	<i>Silicon Labs EFM32</i>	On- board	EFM32ZG222F32	24MHz	32KB	4KB
<i>SLSTK3400A USB-enabled Happy Gecko</i>	<i>Silicon Labs EFM32</i>	On- board	EFM32HG322F64	25MHz	64KB	8KB
<i>SLSTK3401A Pearl Gecko PG1</i>	<i>Silicon Labs EFM32</i>	On- board	EFM32PG1B200F256	40MHz	256KB	32KB
<i>Thunderboard Sense 2 Sensor-to-Cloud Advanced IoT</i>	<i>Silicon Labs EFM32</i>	On- board	EFR32MG12P432F10240MHz	10240MHz	1MB	256KB

Smeshlink

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Smeshlink xbed LPC1768</i>	<i>NXP LPC</i>	No	LPC1768	96MHz	512KB	32KB

Solder Splash Labs

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>DipCortex M3</i>	<i>NXP LPC</i>	External	LPC1347	72MHz	64KB	12KB
<i>Solder Splash Labs DipCortex M0</i>	<i>NXP LPC</i>	External	LPC11U24	50MHz	32KB	8KB

Switch Science

Name	Platform	Debug	MCU	Fre- quency	Flash	RAM
<i>Switch Science mbed HRM1017</i>	<i>Nordic nRF51</i>	On-board	NRF51822	16MHz	256KB	16KB
<i>Switch Science mbed LPC1114FN28</i>	<i>NXP LPC</i>	On-board	LPC1114FN28	48MHz	32KB	4KB
<i>Switch Science mbed LPC824</i>	<i>NXP LPC</i>	On-board	LPC824	30MHz	32KB	8KB
<i>Switch Science mbed TY51822r3</i>	<i>Nordic nRF51</i>	On-board	NRF51822	32MHz	256KB	32KB

Teensy

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Teensy 3.1 / 3.2</i>	<i>Teensy</i>	External	MK20DX256	72MHz	256KB	64KB

VNG

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>VNG VBLUNO51</i>	<i>Nordic nRF51</i>	On-board	NRF51822	16MHz	128KB	32KB

WIZNet

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>WIZwiki-W7500</i>	<i>WIZNet W7500</i>	On-board	WIZNET7500	48MHz	128KB	48KB
<i>WIZwiki-W7500ECO</i>	<i>WIZNet W7500</i>	On-board	WIZNET7500ECO	48MHz	128KB	48KB
<i>WIZwiki-W7500P</i>	<i>WIZNet W7500</i>	On-board	WIZNET7500P	48MHz	128KB	48KB

rhomb.io

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>L476DMWIK</i>	<i>ST STM32</i>	On-board	STM32L476VGT6	80MHz	1MB	128KB

sakura.io

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>sakura.io Evaluation Board</i>	<i>ST STM32</i>	On-board	STM32F411RET6	100MHz	1MB	128KB

u-blox

Name	Platform	Debug	MCU	Fre-quency	Flash	RAM
<i>Mbed Connect Cloud</i>	<i>ST STM32</i>	On-board	STM32F439ZIY6	168MHz	2MB	256KB
<i>u-blox C027</i>	<i>NXP LPC</i>	On-board	LPC1768	96MHz	512KB	64KB
<i>u-blox C030-N211 IoT Starter Kit</i>	<i>ST STM32</i>	External	STM32F437VG	180MHz	1MB	256KB
<i>u-blox C030-R410M IoT</i>	<i>ST STM32</i>	On-board	STM32F437VG	180MHz	1MB	256KB
<i>u-blox C030-U201 IoT Starter Kit</i>	<i>ST STM32</i>	External	STM32F437VG	180MHz	1MB	256KB
<i>u-blox EVK-NINA-B1</i>	<i>Nordic nRF52</i>	On-board	NRF52832	64MHz	512KB	64KB
<i>u-blox EVK-ODIN-W2</i>	<i>ST STM32</i>	External	STM32F439ZIY6	168MHz	2MB	256KB
<i>u-blox ODIN-W2</i>	<i>ST STM32</i>	External	STM32F439ZIY6	168MHz	2MB	256KB

y5 design

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>y5 LPC11U35 mbug</i>	<i>NXP LPC</i>	External	LPC11U35	48MHz	64KB	10KB
<i>y5 nRF51822 mbug</i>	<i>Nordic nRF51</i>	On-board	NRF51822	16MHz	256KB	16KB

1.11.13 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](#)
 - [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

Bboards listed below have on-board debug probe and **ARE READY** for debugging! You do not need to use/buy external debug probe.

Name	Platform	MCU	Frequency	Flash	RAM
GAPuino GAP8	RISC-V GAP	GAP8	250MHz	64MB	8MB

Examples

- PULP OS for RISC-V GAP

Platforms

Name	Description
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.

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

Name	Platform	Debug	MCU	Frequency	Flash	RAM
GAPuino GAP8	RISC-V GAP	On-board	GAP8	250MHz	64MB	8MB

1.11.14 Pumbaa

Configuration `framework` = pumbaa

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

For more detailed information please visit [vendor site](#).

Contents

- [Examples](#)
- [Platforms](#)
- [Boards](#)

Examples

- Pumbaa for Espressif 32

Platforms

Name	Description
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.

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.
-

MakerAsia

Name	Platform	Debug	MCU	Frequency	Flash	RAM
MakerAsia Nano32	Espressif 32	No	ESP32	240MHz	4MB	320KB

1.11.15 Shakti SDK

Configuration `framework = shakti-sdk`

A software development kit for developing applications on Shakti class of processors

For more detailed information please visit [vendor site](#).

Contents

- [Debugging](#)
- [Examples](#)
- [Platforms](#)
- [Boards](#)

Debugging

[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

Bboards listed below have on-board debug probe and **ARE READY** for debugging! You do not need to use/buy external debug probe.

Name	Platform	MCU	Frequency	Flash	RAM
<i>Artix-7 35T Arty FPGA Evaluation Kit</i>	<i>Shakti</i>	E-CLASS	50MHz	0B	128KB
<i>Arty A7-100: Artix-7 FPGA Development Board</i>	<i>Shakti</i>	C-CLASS	50MHz	0B	128MB

Examples

- Shakti SDK for Shakti

Platforms

Name	Description
<i>Shakti</i>	Shakti is an open-source initiative by the RISE group at IIT-Madras, which is not only building open source, production grade processors, but also associated components like interconnect fabrics, verification tools, storage controllers, peripheral IPs and SOC tools.

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.
-

Xilinx

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Artix-7 35T Arty FPGA Evaluation Kit</i>	<i>Shakti</i>	On-board	E-CLASS	50MHz	0B	128KB
<i>Arty A7-100: Artix-7 FPGA Development Board</i>	<i>Shakti</i>	On-board	C-CLASS	50MHz	0B	128MB

1.11.16 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*
 - *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

Banks 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.

Name	Platform	MCU	Frequency	Flash	RAM
<i>Arduino Due (Programming Port)</i>	<i>Atmel SAM</i>	AT91SAM3X8E	84MHz	512KB	96KB
<i>Arduino Due (USB Native Port)</i>	<i>Atmel SAM</i>	AT91SAM3X8E	84MHz	512KB	96KB

Examples

- Simba for Atmel AVR
- Simba for Atmel SAM
- Simba for Espressif 32
- Simba for Espressif 8266

Platforms

Name	Description
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.
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.
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.
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.

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

Name	Platform	Debug	MCU	Frequency	Flash	RAM
Adafruit HUZZAH ESP8266	Espressif 8266	No	ESP8266	80MHz	4MB	80KB

Arduino

Name	Platform	De- bug	MCU	Fre- quency	Flash	RAM
Arduino Due (Programming Port)	Atmel SAM	External	AT91SAM3X8E84MHz	512KB	96KB	
Arduino Due (USB Native Port)	Atmel SAM	External	AT91SAM3X8E84MHz	512KB	96KB	
Arduino Mega or Mega 2560 ATmega2560 (Mega 2560)	Atmel AVR	No	AT-MEGA2560	16MHz	248KB	8KB
Arduino Nano ATmega328	Atmel AVR	No	AT-MEGA328P	16MHz	30KB	2KB
Arduino Uno	Atmel AVR	No	AT-MEGA328P	16MHz	31.50KB	2KB

Espressif

Name	Platform	De- bug	MCU	Fre- quency	Flash	RAM
<i>ESP-WROOM-02</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	2MB	80KB
<i>Espressif ESP8266 ESP-12E</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	4MB	80KB
<i>Espressif Generic ESP8266 ESP-01 512k</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	512KB	80KB

Invent One

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Invent One</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	4MB	80KB

MakerAsia

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>MakerAsia Nano32</i>	<i>Espressif 32</i>	No	ESP32	240MHz	4MB	320KB

NodeMCU

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>NodeMCU 0.9 (ESP-12 Module)</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	4MB	80KB
<i>NodeMCU 1.0 (ESP-12E Module)</i>	<i>Espressif 8266</i>	No	ESP8266	80MHz	4MB	80KB

SeeedStudio

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Seeeduino</i>	<i>Atmel AVR</i>	No	ATMEGA328P	16MHz	31.50KB	2KB

1.11.17 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.

Name	Platform	MCU	Frequency	Flash	RAM
RushUp Cloud-JAM	ST STM32	STM32F401RET6	84MHz	512KB	96KB
ST Nucleo F401RE	ST STM32	STM32F401RET6	84MHz	512KB	96KB
ST STM32F3DISCOVERY	ST STM32	STM32F303VCT6	72MHz	256KB	48KB
ST STM32F4DISCOVERY	ST STM32	STM32F407VGT6	168MHz	1MB	128KB
ST STM32LDISCOVERY	ST STM32	STM32L152RBT6	32MHz	128KB	16KB
ST STM8S-DISCOVERY	ST STM8	STM8S105C6T6	16MHz	32KB	2KB

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.

Name	Platform	MCU	Frequency	Flash	RAM
1Bitsy	ST STM32	STM32F415RGT	168MHz	1MB	128KB
Armstrap Eagle 1024	ST STM32	STM32F417VGT6	168MHz	1MB	192KB
Armstrap Eagle 2048	ST STM32	STM32F427VIT6	168MHz	1.99MB	256KB
Armstrap Eagle 512	ST STM32	STM32F407VET6	168MHz	512KB	192KB

Examples

- SPL for ST STM32
- SPL for ST STM8

Platforms

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

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

Name	Platform	Debug	MCU	Frequency	Flash	RAM
1Bitsy	ST STM32	External	STM32F415RGT	168MHz	1MB	128KB

Armstrap

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Armstrap Eagle 1024</i>	<i>ST STM32</i>	External	STM32F417VGT6	168MHz	1MB	192KB
<i>Armstrap Eagle 2048</i>	<i>ST STM32</i>	External	STM32F427VIT6	168MHz	1.99MB	256KB
<i>Armstrap Eagle 512</i>	<i>ST STM32</i>	External	STM32F407VET6	168MHz	512KB	192KB

RushUp

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>RushUp Cloud-JAM</i>	<i>ST STM32</i>	On-board	STM32F401RET6	84MHz	512KB	96KB

ST

Name	Platform	Debug	MCU	Fre-quency	Flash	RAM
<i>ST Nucleo F401RE</i>	<i>ST STM32</i>	On-board	STM32F401RET6	84MHz	512KB	96KB
<i>ST STM32F3DISCOVERY</i>	<i>ST STM32</i>	On-board	STM32F303VCT6	72MHz	256KB	48KB
<i>ST STM32F4DISCOVERY</i>	<i>ST STM32</i>	On-board	STM32F407VGT6	168MHz	1MB	128KB
<i>ST STM32LDiscovery</i>	<i>ST STM32</i>	On-board	STM32L152RBT6	32MHz	128KB	16KB
<i>ST STM8S-DISCOVERY</i>	<i>ST STM8</i>	On-board	STM8S105C6T6	16MHz	32KB	2KB
<i>ST STM8S103F3 Breakout Board</i>	<i>ST STM8</i>	No	STM8S103F3P6	16MHz	8KB	1KB
<i>ST STM8S105K4T6 Breakout Board</i>	<i>ST STM8</i>	No	STM8S105K4T6	16MHz	16KB	2KB

sduino

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>sduino MB (STM8S208MBT6B)</i>	<i>ST STM8</i>	No	STM8S208MBT6	16MHz	128KB	6KB
<i>sduino UNO (STM8S105K6)</i>	<i>ST STM8</i>	No	STM8S105K6T6	16MHz	32KB	2KB

1.11.18 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*
 - *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

Bands listed below have on-board debug probe and **ARE READY** for debugging! You do not need to use/buy external debug probe.

Name	Platform	MCU	Frequency	Flash	RAM
<i>32F723EDISCOVERY</i>	<i>ST STM32</i>	<i>STM32F723IEK6</i>	216MHz	512KB	192KB
<i>3D printer controller</i>	<i>ST STM32</i>	<i>STM32F765VIT6</i>	216MHz	2MB	512KB
<i>3DP001VI Evaluation board for 3D printer</i>	<i>ST STM32</i>	<i>STM32F401VGT6</i>	84MHz	512KB	96KB
<i>96Boards B96B-F446VE</i>	<i>ST STM32</i>	<i>STM32F446VET6</i>	168MHz	512KB	128KB
<i>Aceinna Low Cost RTK</i>	<i>Aceinna IMU</i>	<i>STM32F469NIH6</i>	180MHz	1MB	384KB
<i>L476DMWIK</i>	<i>ST STM32</i>	<i>STM32L476VGT6</i>	80MHz	1MB	128KB

Continued on next page

Table 24 – continued from previous page

Name	Platform	MCU	Frequency	Flash	RAM
<i>Mbed Connect Cloud</i>	<i>ST STM32</i>	STM32F439ZIY6	168MHz	2MB	256KB
<i>Microsoft Azure IoT Development Kit (MXChip AZ3166)</i>	<i>ST STM32</i>	STM32F412ZGT6	100MHz	1MB	256KB
<i>RushUp Cloud-JAM</i>	<i>ST STM32</i>	STM32F401RET6	84MHz	512KB	96KB
<i>RushUp Cloud-JAM L4</i>	<i>ST STM32</i>	STM32L476RGT6	80MHz	1MB	128KB
<i>ST 32F3348DISCOVERY</i>	<i>ST STM32</i>	STM32F334C8T6	72MHz	64KB	12KB
<i>ST 32F401CDISCOVERY</i>	<i>ST STM32</i>	STM32F401VCT6	84MHz	256KB	64KB
<i>ST 32F411EDISCOVERY</i>	<i>ST STM32</i>	STM32F411VET6	100MHz	512KB	128KB
<i>ST 32F413HDISCOVERY</i>	<i>ST STM32</i>	STM32F413ZHT6	100MHz	512KB	128KB
<i>ST 32F429IDISCOVERY</i>	<i>ST STM32</i>	STM32F429ZIT6	180MHz	2MB	256KB
<i>ST 32F469IDISCOVERY</i>	<i>ST STM32</i>	STM32F469NIH6	180MHz	1MB	384KB
<i>ST 32F746GDISCOVERY</i>	<i>ST STM32</i>	STM32F746NGH6	216MHz	1MB	320KB
<i>ST 32F769IDISCOVERY</i>	<i>ST STM32</i>	STM32F769NIH6	216MHz	1MB	512KB
<i>ST 32L0538DISCOVERY</i>	<i>ST STM32</i>	STM32L053C8T6	32MHz	64KB	8KB
<i>ST 32L100DISCOVERY</i>	<i>ST STM32</i>	STM32L100RCT6	32MHz	256KB	16KB
<i>ST 32L476GDISCOVERY</i>	<i>ST STM32</i>	STM32L476VGT6	80MHz	1MB	128KB
<i>ST 32L496GDISCOVERY</i>	<i>ST STM32</i>	STM32L496AGI6	80MHz	1MB	320KB
<i>ST DISCO-L072CZ-LRWAN1</i>	<i>ST STM32</i>	STM32L072CZ	32MHz	192KB	20KB
<i>ST DISCO-L475VG-IOT01A</i>	<i>ST STM32</i>	STM32L475VGT6	80MHz	1MB	128KB
<i>ST Discovery F072RB</i>	<i>ST STM32</i>	STM32F072RBT6	48MHz	128KB	16KB
<i>ST Nucleo F030R8</i>	<i>ST STM32</i>	STM32F030R8T6	48MHz	64KB	8KB
<i>ST Nucleo F031K6</i>	<i>ST STM32</i>	STM32F031K6T6	48MHz	32KB	4KB
<i>ST Nucleo F042K6</i>	<i>ST STM32</i>	STM32F042K6T6	48MHz	32KB	6KB
<i>ST Nucleo F070RB</i>	<i>ST STM32</i>	STM32F070RBT6	48MHz	128KB	16KB
<i>ST Nucleo F072RB</i>	<i>ST STM32</i>	STM32F072RBT6	48MHz	128KB	16KB
<i>ST Nucleo F091RC</i>	<i>ST STM32</i>	STM32F091RCT6	48MHz	256KB	32KB
<i>ST Nucleo F103RB</i>	<i>ST STM32</i>	STM32F103RBT6	72MHz	128KB	20KB
<i>ST Nucleo F207ZG</i>	<i>ST STM32</i>	STM32F207ZGT6	120MHz	1MB	128KB
<i>ST Nucleo F302R8</i>	<i>ST STM32</i>	STM32F302R8T6	72MHz	64KB	16KB
<i>ST Nucleo F303K8</i>	<i>ST STM32</i>	STM32F303K8T6	72MHz	64KB	12KB
<i>ST Nucleo F303RE</i>	<i>ST STM32</i>	STM32F303RET6	72MHz	512KB	64KB
<i>ST Nucleo F303ZE</i>	<i>ST STM32</i>	STM32F303ZET6	72MHz	512KB	64KB
<i>ST Nucleo F334R8</i>	<i>ST STM32</i>	STM32F334R8T6	72MHz	64KB	16KB
<i>ST Nucleo F401RE</i>	<i>ST STM32</i>	STM32F401RET6	84MHz	512KB	96KB
<i>ST Nucleo F410RB</i>	<i>ST STM32</i>	STM32F410RBT6	100MHz	128KB	32KB
<i>ST Nucleo F411RE</i>	<i>ST STM32</i>	STM32F411RET6	100MHz	512KB	128KB
<i>ST Nucleo F412ZG</i>	<i>ST STM32</i>	STM32F412ZGT6	100MHz	1MB	256KB
<i>ST Nucleo F413ZH</i>	<i>ST STM32</i>	STM32F413ZHT6	100MHz	512KB	128KB
<i>ST Nucleo F429ZI</i>	<i>ST STM32</i>	STM32F429ZIT6	180MHz	2MB	256KB
<i>ST Nucleo F439ZI</i>	<i>ST STM32</i>	STM32F439ZIT6	180MHz	2MB	256KB
<i>ST Nucleo F446RE</i>	<i>ST STM32</i>	STM32F446RET6	180MHz	512KB	128KB
<i>ST Nucleo F446ZE</i>	<i>ST STM32</i>	STM32F446ZET6	180MHz	512KB	128KB
<i>ST Nucleo F722ZE</i>	<i>ST STM32</i>	STM32F722ZET6	216MHz	512KB	256KB
<i>ST Nucleo F746ZG</i>	<i>ST STM32</i>	STM32F746ZGT6	216MHz	1MB	320KB
<i>ST Nucleo F756ZG</i>	<i>ST STM32</i>	STM32F756ZG	216MHz	1MB	320KB
<i>ST Nucleo F767ZI</i>	<i>ST STM32</i>	STM32F767ZIT6	216MHz	2MB	512KB
<i>ST Nucleo H743ZI</i>	<i>ST STM32</i>	STM32H743ZIT6	400MHz	2MB	1MB
<i>ST Nucleo L011K4</i>	<i>ST STM32</i>	STM32L011K4T6	32MHz	16KB	2KB
<i>ST Nucleo L031K6</i>	<i>ST STM32</i>	STM32L031K6T6	32MHz	32KB	8KB
<i>ST Nucleo L053R8</i>	<i>ST STM32</i>	STM32L053R8T6	32MHz	64KB	8KB

Continued on next page

Table 24 – continued from previous page

Name	Platform	MCU	Frequency	Flash	RAM
<i>ST Nucleo L073RZ</i>	<i>ST STM32</i>	STM32L073RZ	32MHz	192KB	20KB
<i>ST Nucleo L152RE</i>	<i>ST STM32</i>	STM32L152RET6	32MHz	512KB	80KB
<i>ST Nucleo L412KB</i>	<i>ST STM32</i>	STM32L412KBU6	80MHz	128KB	40KB
<i>ST Nucleo L432KC</i>	<i>ST STM32</i>	STM32L432KCU6	80MHz	256KB	64KB
<i>ST Nucleo L433RC-P</i>	<i>ST STM32</i>	STM32L433RC	80MHz	256KB	64KB
<i>ST Nucleo L452RE</i>	<i>ST STM32</i>	STM32L452RET6	80MHz	256KB	64KB
<i>ST Nucleo L476RG</i>	<i>ST STM32</i>	STM32L476RGT6	80MHz	1MB	128KB
<i>ST Nucleo L486RG</i>	<i>ST STM32</i>	STM32L486RGT6	80MHz	1MB	128KB
<i>ST Nucleo L496ZG</i>	<i>ST STM32</i>	STM32L496ZGT6	80MHz	1MB	128KB
<i>ST Nucleo L496ZG-P</i>	<i>ST STM32</i>	STM32L496ZGT6P	80MHz	1MB	320KB
<i>ST Nucleo L4R5ZI</i>	<i>ST STM32</i>	STM32L4R5ZIT6	120MHz	2MB	640KB
<i>ST STM32F0308DISCOVERY</i>	<i>ST STM32</i>	STM32F030R8T6	48MHz	64KB	8KB
<i>ST STM32F0DISCOVERY</i>	<i>ST STM32</i>	STM32F051R8T6	48MHz	64KB	8KB
<i>ST STM32F3DISCOVERY</i>	<i>ST STM32</i>	STM32F303VCT6	72MHz	256KB	48KB
<i>ST STM32F4DISCOVERY</i>	<i>ST STM32</i>	STM32F407VGT6	168MHz	1MB	128KB
<i>ST STM32L073Z-EVAL</i>	<i>ST STM32</i>	STM32L073VZT6	32MHz	192KB	20KB
<i>ST STM32LDISCOVERY</i>	<i>ST STM32</i>	STM32L152RBT6	32MHz	128KB	16KB
<i>ST STM32VLDiscovery</i>	<i>ST STM32</i>	STM32F100RBT6	24MHz	128KB	8KB
<i>ST Sensor Node</i>	<i>ST STM32</i>	STM32L476JG	80MHz	1MB	128KB
<i>STM32F7508-DK</i>	<i>ST STM32</i>	STM32F750N8H6	216MHz	64KB	340KB
<i>Seeed Arch Max</i>	<i>ST STM32</i>	STM32F407VET6	168MHz	512KB	192KB
<i>Seeed Wio 3G</i>	<i>ST STM32</i>	STM32F439VI	180MHz	2MB	256KB
<i>sakura.io Evaluation Board</i>	<i>ST STM32</i>	STM32F411RET6	100MHz	1MB	128KB
<i>u-blox C030-R410M IoT</i>	<i>ST STM32</i>	STM32F437VG	180MHz	1MB	256KB

External Debug Tools

Banks 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.

Name	Platform	MCU	Frequency	Flash	RAM
<i>1Bitsy</i>	<i>ST STM32</i>	STM32F415RGT	168MHz	1MB	128KB
<i>3D Printer Controller</i>	<i>ST STM32</i>	STM32F407VET6	168MHz	512KB	192KB
<i>3D Printer control board</i>	<i>ST STM32</i>	STM32F446RET6	180MHz	512KB	128KB
<i>Aceinna OpenIMU 330</i>	<i>Aceinna IMU</i>	STM32L431CB	80MHz	128KB	64KB
<i>Armstrap Eagle 1024</i>	<i>ST STM32</i>	STM32F417VGT6	168MHz	1MB	192KB
<i>Armstrap Eagle 2048</i>	<i>ST STM32</i>	STM32F427VIT6	168MHz	1.99MB	256KB
<i>Armstrap Eagle 512</i>	<i>ST STM32</i>	STM32F407VET6	168MHz	512KB	192KB
<i>Black STM32F407VE</i>	<i>ST STM32</i>	STM32F407VET6	168MHz	512KB	128KB
<i>Black STM32F407VG</i>	<i>ST STM32</i>	STM32F407VGT6	168MHz	512KB	128KB
<i>Black STM32F407ZE</i>	<i>ST STM32</i>	STM32F407ZET6	168MHz	512KB	128KB
<i>Black STM32F407Z</i>	<i>ST STM32</i>	STM32F407ZGT6	168MHz	1MB	128KB
<i>BlackPill F103C8</i>	<i>ST STM32</i>	STM32F103C8T6	72MHz	64KB	20KB
<i>BlackPill F103C8 (128k)</i>	<i>ST STM32</i>	STM32F103C8T6	72MHz	128KB	20KB
<i>BlackPill F303CC</i>	<i>ST STM32</i>	STM32F303CCT6	72MHz	256KB	40KB
<i>Blue STM32F407VE Mini</i>	<i>ST STM32</i>	STM32F407VET6	168MHz	512KB	128KB
<i>BluePill F103C6</i>	<i>ST STM32</i>	STM32F103C6T6	72MHz	32KB	10KB
<i>BluePill F103C8</i>	<i>ST STM32</i>	STM32F103C8T6	72MHz	64KB	20KB

Continued on next page

Table 25 – continued from previous page

Name	Platform	MCU	Frequency	Flash	RAM
<i>BluePill F103C8 (128k)</i>	<i>ST STM32</i>	STM32F103C8T6	72MHz	128KB	20KB
<i>Demo F030F4</i>	<i>ST STM32</i>	STM32F030F4P6	48MHz	16KB	4KB
<i>Espotel LoRa Module</i>	<i>ST STM32</i>	STM32F411RET6	100MHz	512KB	128KB
<i>F407VG</i>	<i>ST STM32</i>	STM32F407VGT6	168MHz	512KB	128KB
<i>FK407M1</i>	<i>ST STM32</i>	STM32F407VET6	168MHz	512KB	128KB
<i>M200 V2</i>	<i>ST STM32</i>	STM32F070CBT6	48MHz	120KB	14.81KB
<i>MTS Dragonfly</i>	<i>ST STM32</i>	STM32F411RET6	100MHz	512KB	128KB
<i>Malyan M200 V1</i>	<i>ST STM32</i>	STM32F103CBT6	72MHz	120KB	20KB
<i>Maple</i>	<i>ST STM32</i>	STM32F103RBT6	72MHz	108KB	17KB
<i>Maple (RET6)</i>	<i>ST STM32</i>	STM32F103RET6	72MHz	256KB	48KB
<i>Maple Mini Bootloader 2.0</i>	<i>ST STM32</i>	STM32F103CBT6	72MHz	120KB	20KB
<i>Maple Mini Original</i>	<i>ST STM32</i>	STM32F103CBT6	72MHz	108KB	17KB
<i>Microduino Core STM32 to Flash</i>	<i>ST STM32</i>	STM32F103CBT6	72MHz	105.47KB	16.60KB
<i>MultiTech mDot</i>	<i>ST STM32</i>	STM32F411RET6	100MHz	512KB	128KB
<i>MultiTech mDot F411</i>	<i>ST STM32</i>	STM32F411RET6	100MHz	512KB	128KB
<i>MultiTech xDot</i>	<i>ST STM32</i>	STM32L151CCU6	32MHz	256KB	32KB
<i>N2+</i>	<i>ST STM32</i>	STM32F405RGT6	168MHz	1MB	192KB
<i>NAMote72</i>	<i>ST STM32</i>	STM32L152RC	32MHz	256KB	32KB
<i>RAK811 LoRa Tracker</i>	<i>ST STM32</i>	STM32L151RBT6	32MHz	128KB	16KB
<i>RAK811 LoRa Tracker</i>	<i>ST STM32</i>	STM32L151RBT6	32MHz	128KB	32KB
<i>STM32F103C8 (20k RAM. 64k Flash)</i>	<i>ST STM32</i>	STM32F103C8T6	72MHz	64KB	20KB
<i>STM32F103CB (20k RAM. 128k Flash)</i>	<i>ST STM32</i>	STM32F103CBT6	72MHz	128KB	20KB
<i>STM32F103R8 (20k RAM. 64 Flash)</i>	<i>ST STM32</i>	STM32F103R8T6	72MHz	64KB	20KB
<i>STM32F103RB (20k RAM. 128k Flash)</i>	<i>ST STM32</i>	STM32F103RBT6	72MHz	128KB	20KB
<i>STM32F103RC (48k RAM. 256k Flash)</i>	<i>ST STM32</i>	STM32F103RCT6	72MHz	256KB	48KB
<i>STM32F103RE (64k RAM. 512k Flash)</i>	<i>ST STM32</i>	STM32F103RET6	72MHz	512KB	64KB
<i>STM32F103T8 (20k RAM. 64k Flash)</i>	<i>ST STM32</i>	STM32F103T8T6	72MHz	20KB	64KB
<i>STM32F103TB (20k RAM. 128k Flash)</i>	<i>ST STM32</i>	STM32F103TBT6	72MHz	128KB	20KB
<i>STM32F103VB (20k RAM. 128k Flash)</i>	<i>ST STM32</i>	STM32F103VBT6	72MHz	128KB	20KB
<i>STM32F103VC (48k RAM. 256k Flash)</i>	<i>ST STM32</i>	STM32F103VCT6	72MHz	256KB	48KB
<i>STM32F103VD (64k RAM. 384k Flash)</i>	<i>ST STM32</i>	STM32F103VDT6	72MHz	384KB	64KB
<i>STM32F103VE (64k RAM. 512k Flash)</i>	<i>ST STM32</i>	STM32F103VET6	72MHz	512KB	64KB
<i>STM32F103ZC (48k RAM. 256k Flash)</i>	<i>ST STM32</i>	STM32F103ZCT6	72MHz	256KB	48KB
<i>STM32F103ZD (64k RAM. 384k Flash)</i>	<i>ST STM32</i>	STM32F103ZDT6	72MHz	384KB	64KB
<i>STM32F103ZE (64k RAM. 512k Flash)</i>	<i>ST STM32</i>	STM32F103ZET6	72MHz	512KB	64KB
<i>STM32F303CB (32k RAM. 128k Flash)</i>	<i>ST STM32</i>	STM32F303CBT6	72MHz	128KB	32KB
<i>STM32F407VE (192k RAM. 512k Flash)</i>	<i>ST STM32</i>	STM32F407VET6	168MHz	502.23KB	128KB
<i>STM32F407VG (192k RAM. 1024k Flash)</i>	<i>ST STM32</i>	STM32F407VGT6	168MHz	1MB	192KB
<i>STM32F4Stamp F405</i>	<i>ST STM32</i>	STM32F405RGT6	168MHz	1MB	192KB
<i>Sparky V1 F303</i>	<i>ST STM32</i>	STM32F303CCT6	72MHz	256KB	40KB
<i>Tiny STM103T</i>	<i>ST STM32</i>	STM32F103TBU6	72MHz	128KB	20KB
<i>VAKe v1.0</i>	<i>ST STM32</i>	STM32F446RET6	180MHz	512KB	128KB
<i>u-blox C030-N211 IoT Starter Kit</i>	<i>ST STM32</i>	STM32F437VG	180MHz	1MB	256KB
<i>u-blox C030-U201 IoT Starter Kit</i>	<i>ST STM32</i>	STM32F437VG	180MHz	1MB	256KB
<i>u-blox EVK-ODIN-W2</i>	<i>ST STM32</i>	STM32F439ZIY6	168MHz	2MB	256KB
<i>u-blox ODIN-W2</i>	<i>ST STM32</i>	STM32F439ZIY6	168MHz	2MB	256KB

Examples

- STM32Cube for ST STM32

Platforms

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

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

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>1Bitsy</i>	<i>ST STM32</i>	External	STM32F415RGT	168MHz	1MB	128KB

96Boards

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>96Boards B96B-F446VE</i>	<i>ST STM32</i>	On-board	STM32F446VET6	168MHz	512KB	128KB

Aceinna

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Aceinna Low Cost RTK</i>	<i>Aceinna IMU</i>	On-board	STM32F469NIH6	180MHz	1MB	384KB
<i>Aceinna OpenIMU 330</i>	<i>Aceinna IMU</i>	External	STM32L431CB	80MHz	128KB	64KB

Armed

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>3D Printer Controller</i>	<i>ST STM32</i>	External	STM32F407VET6	168MHz	512KB	192KB

Armstrap

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Armstrap Eagle 1024</i>	<i>ST STM32</i>	External	STM32F417VGT6	168MHz	1MB	192KB
<i>Armstrap Eagle 2048</i>	<i>ST STM32</i>	External	STM32F427VIT6	168MHz	1.99MB	256KB
<i>Armstrap Eagle 512</i>	<i>ST STM32</i>	External	STM32F407VET6	168MHz	512KB	192KB

Avnet Silica

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>ST Sensor Node</i>	<i>ST STM32</i>	On-board	STM32L476JG	80MHz	1MB	128KB

Diymore

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>F407VG</i>	<i>ST STM32</i>	External	STM32F407VGT6	168MHz	512KB	128KB

Espotel

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Espotel LoRa Module</i>	<i>ST STM32</i>	External	STM32F411RET6	100MHz	512KB	128KB

Generic

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>BlackPill F103C8</i>	<i>ST STM32</i>	External	STM32F103C8T6	72MHz	64KB	20KB
<i>BlackPill F103C8 (128k)</i>	<i>ST STM32</i>	External	STM32F103C8T6	72MHz	128KB	20KB
<i>BluePill F103C6</i>	<i>ST STM32</i>	External	STM32F103C6T6	72MHz	32KB	10KB
<i>BluePill F103C8</i>	<i>ST STM32</i>	External	STM32F103C8T6	72MHz	64KB	20KB
<i>BluePill F103C8 (128k)</i>	<i>ST STM32</i>	External	STM32F103C8T6	72MHz	128KB	20KB
<i>Demo F030F4</i>	<i>ST STM32</i>	External	STM32F030F4P6	48MHz	16KB	4KB
<i>FK407M1</i>	<i>ST STM32</i>	External	STM32F407VET6	168MHz	512KB	128KB
<i>STM32F103C8 (20k RAM. 64k Flash)</i>	<i>ST STM32</i>	External	STM32F103C8T6	72MHz	64KB	20KB
<i>STM32F103CB (20k RAM. 128k Flash)</i>	<i>ST STM32</i>	External	STM32F103CBT6	72MHz	128KB	20KB
<i>STM32F103R8 (20k RAM. 64 Flash)</i>	<i>ST STM32</i>	External	STM32F103R8T6	72MHz	64KB	20KB
<i>STM32F103RB (20k RAM. 128k Flash)</i>	<i>ST STM32</i>	External	STM32F103RBT6	72MHz	128KB	20KB
<i>STM32F103RC (48k RAM. 256k Flash)</i>	<i>ST STM32</i>	External	STM32F103RCT6	72MHz	256KB	48KB
<i>STM32F103RE (64k RAM. 512k Flash)</i>	<i>ST STM32</i>	External	STM32F103RET6	72MHz	512KB	64KB
<i>STM32F103T8 (20k RAM. 64k Flash)</i>	<i>ST STM32</i>	External	STM32F103T8T6	72MHz	20KB	64KB
<i>STM32F103TB (20k RAM. 128k Flash)</i>	<i>ST STM32</i>	External	STM32F103TBT6	72MHz	128KB	20KB
<i>STM32F103VB (20k RAM. 128k Flash)</i>	<i>ST STM32</i>	External	STM32F103VBT6	72MHz	128KB	20KB
<i>STM32F103VC (48k RAM. 256k Flash)</i>	<i>ST STM32</i>	External	STM32F103VCT6	72MHz	256KB	48KB
<i>STM32F103VD (64k RAM. 384k Flash)</i>	<i>ST STM32</i>	External	STM32F103VDT6	72MHz	384KB	64KB
<i>STM32F103VE (64k RAM. 512k Flash)</i>	<i>ST STM32</i>	External	STM32F103VET6	72MHz	512KB	64KB
<i>STM32F103ZC (48k RAM. 256k Flash)</i>	<i>ST STM32</i>	External	STM32F103ZCT6	72MHz	256KB	48KB
<i>STM32F103ZD (64k RAM. 384k Flash)</i>	<i>ST STM32</i>	External	STM32F103ZDT6	72MHz	384KB	64KB
<i>STM32F103ZE (64k RAM. 512k Flash)</i>	<i>ST STM32</i>	External	STM32F103ZET6	72MHz	512KB	64KB
<i>STM32F303CB (32k RAM. 128k Flash)</i>	<i>ST STM32</i>	External	STM32F303CBT6	72MHz	128KB	32KB
<i>STM32F407VE (192k RAM. 512k Flash)</i>	<i>ST STM32</i>	External	STM32F407VET6	168MHz	502.23KB	128KB
<i>STM32F407VG (192k RAM. 1024k Flash)</i>	<i>ST STM32</i>	External	STM32F407VGT6	168MHz	1MB	192KB
1.1 STM Frameworks	<i>ST STM32</i>	External	STM32F405RGT6	168MHz	1MB	192KB

HY

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Tiny STM103T</i>	<i>ST STM32</i>	External	STM32F103TBU6	72MHz	128KB	20KB

LeafLabs

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Maple</i>	<i>ST STM32</i>	External	STM32F103RBT6	72MHz	108KB	17KB
<i>Maple (RET6)</i>	<i>ST STM32</i>	External	STM32F103RET6	72MHz	256KB	48KB
<i>Maple Mini Bootloader 2.0</i>	<i>ST STM32</i>	External	STM32F103CBT6	72MHz	120KB	20KB
<i>Maple Mini Original</i>	<i>ST STM32</i>	External	STM32F103CBT6	72MHz	108KB	17KB

MXChip

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Microsoft Azure IoT Development Kit (MX-Chip AZ3166)</i>	<i>ST STM32</i>	On-board	STM32F412ZG	100MHz	1MB	256KB

Malyan

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>M200 V2</i>	<i>ST STM32</i>	External	STM32F070CBT6	48MHz	120KB	14.81KB
<i>Malyan M200 V1</i>	<i>ST STM32</i>	External	STM32F103CBT6	72MHz	120KB	20KB

Micoduino

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Microduino Core STM32 to Flash</i>	<i>ST STM32</i>	External	STM32F103CBT6	72MHz	105.47KB	16.60KB

MultiTech

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>MTS Dragonfly</i>	<i>ST STM32</i>	External	STM32F411RET6	100MHz	512KB	128KB
<i>MultiTech mDot</i>	<i>ST STM32</i>	External	STM32F411RET6	100MHz	512KB	128KB
<i>MultiTech mDot F411</i>	<i>ST STM32</i>	External	STM32F411RET6	100MHz	512KB	128KB
<i>MultiTech xDot</i>	<i>ST STM32</i>	External	STM32L151CCU6	32MHz	256KB	32KB

Netduino

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>N2+</i>	<i>ST STM32</i>	External	STM32F405RGT6	168MHz	1MB	192KB

RAK

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>RAK811 LoRa Tracker</i>	<i>ST STM32</i>	External	STM32L151RBT6	32MHz	128KB	16KB
<i>RAK811 LoRa Tracker</i>	<i>ST STM32</i>	External	STM32L151RBT6	32MHz	128KB	32KB

RUMBA

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>3D Printer control board</i>	<i>ST STM32</i>	External	STM32F446RET6	180MHz	512KB	128KB

RemRam

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>3D printer controller</i>	<i>ST STM32</i>	On-board	STM32F765VIT6	216MHz	2MB	512KB

RobotDyn

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>BlackPill F303CC</i>	<i>ST STM32</i>	External	STM32F303CCT6	72MHz	256KB	40KB

RushUp

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>RushUp Cloud-JAM</i>	<i>ST STM32</i>	On-board	STM32F401RET6	84MHz	512KB	96KB
<i>RushUp Cloud-JAM L4</i>	<i>ST STM32</i>	On-board	STM32L476RGT6	80MHz	1MB	128KB

ST

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>32F723EDISCOVERY</i>	<i>ST STM32</i>	On-board	STM32F723IEK6	216MHz	512KB	192KB
<i>3DP001V1 Evaluation board for 3D printer</i>	<i>ST STM32</i>	On-board	STM32F401VGT6	84MHz	512KB	96KB
<i>Black STM32F407VE</i>	<i>ST STM32</i>	External	STM32F407VET6	168MHz	512KB	128KB
<i>Black STM32F407VG</i>	<i>ST STM32</i>	External	STM32F407VGT6	168MHz	512KB	128KB

Continued on next page

Table 26 – continued from previous page

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Black STM32F407ZE</i>	<i>ST STM32</i>	External	STM32F407ZET6	168MHz	512KB	128KB
<i>Black STM32F407ZE</i>	<i>ST STM32</i>	External	STM32F407ZGT6	168MHz	1MB	128KB
<i>Blue STM32F407VE Mini</i>	<i>ST STM32</i>	External	STM32F407VET6	168MHz	512KB	128KB
<i>ST 32F3348DISCOVERY</i>	<i>ST STM32</i>	On-board	STM32F334C8T6	72MHz	64KB	12KB
<i>ST 32F401CDISCOVERY</i>	<i>ST STM32</i>	On-board	STM32F401VCT6	84MHz	256KB	64KB
<i>ST 32F411EDISCOVERY</i>	<i>ST STM32</i>	On-board	STM32F411VET6	100MHz	512KB	128KB
<i>ST 32F413HDISCOVERY</i>	<i>ST STM32</i>	On-board	STM32F413ZHT6	100MHz	512KB	128KB
<i>ST 32F429IDISCOVERY</i>	<i>ST STM32</i>	On-board	STM32F429ZIT6	180MHz	2MB	256KB
<i>ST 32F469IDISCOVERY</i>	<i>ST STM32</i>	On-board	STM32F469NIH6	180MHz	1MB	384KB
<i>ST 32F746GDISCOVERY</i>	<i>ST STM32</i>	On-board	STM32F746NGH6	216MHz	1MB	320KB
<i>ST 32F769IDISCOVERY</i>	<i>ST STM32</i>	On-board	STM32F769NIH6	216MHz	1MB	512KB
<i>ST 32L0538DISCOVERY</i>	<i>ST STM32</i>	On-board	STM32L053C8T6	32MHz	64KB	8KB
<i>ST 32L100DISCOVERY</i>	<i>ST STM32</i>	On-board	STM32L100RCT6	32MHz	256KB	16KB
<i>ST 32L476GDISCOVERY</i>	<i>ST STM32</i>	On-board	STM32L476VGT6	80MHz	1MB	128KB
<i>ST 32L496GDISCOVERY</i>	<i>ST STM32</i>	On-board	STM32L496AGI6	80MHz	1MB	320KB
<i>ST DISCO-L072CZ-LRWAN1</i>	<i>ST STM32</i>	On-board	STM32L072CZ	32MHz	192KB	20KB
<i>ST DISCO-L475VG-IOT01A</i>	<i>ST STM32</i>	On-board	STM32L475VGT6	80MHz	1MB	128KB
<i>ST Discovery F072RB</i>	<i>ST STM32</i>	On-board	STM32F072RBT6	48MHz	128KB	16KB
<i>ST Nucleo F030R8</i>	<i>ST STM32</i>	On-board	STM32F030R8T6	48MHz	64KB	8KB
<i>ST Nucleo F031K6</i>	<i>ST STM32</i>	On-board	STM32F031K6T6	48MHz	32KB	4KB
<i>ST Nucleo F042K6</i>	<i>ST STM32</i>	On-board	STM32F042K6T6	48MHz	32KB	6KB
<i>ST Nucleo F070RB</i>	<i>ST STM32</i>	On-board	STM32F070RBT6	48MHz	128KB	16KB
<i>ST Nucleo F072RB</i>	<i>ST STM32</i>	On-board	STM32F072RBT6	48MHz	128KB	16KB
<i>ST Nucleo F091RC</i>	<i>ST STM32</i>	On-board	STM32F091RCT6	48MHz	256KB	32KB
<i>ST Nucleo F103RB</i>	<i>ST STM32</i>	On-board	STM32F103RBT6	72MHz	128KB	20KB
<i>ST Nucleo F207ZG</i>	<i>ST STM32</i>	On-board	STM32F207ZGT6	120MHz	1MB	128KB
<i>ST Nucleo F302R8</i>	<i>ST STM32</i>	On-board	STM32F302R8T6	72MHz	64KB	16KB
<i>ST Nucleo F303K8</i>	<i>ST STM32</i>	On-board	STM32F303K8T6	72MHz	64KB	12KB
<i>ST Nucleo F303RE</i>	<i>ST STM32</i>	On-board	STM32F303RET6	72MHz	512KB	64KB
<i>ST Nucleo F303ZE</i>	<i>ST STM32</i>	On-board	STM32F303ZET6	72MHz	512KB	64KB
<i>ST Nucleo F334R8</i>	<i>ST STM32</i>	On-board	STM32F334R8T6	72MHz	64KB	16KB
<i>ST Nucleo F401RE</i>	<i>ST STM32</i>	On-board	STM32F401RET6	84MHz	512KB	96KB
<i>ST Nucleo F410RB</i>	<i>ST STM32</i>	On-board	STM32F410RBT6	100MHz	128KB	32KB
<i>ST Nucleo F411RE</i>	<i>ST STM32</i>	On-board	STM32F411RET6	100MHz	512KB	128KB
<i>ST Nucleo F412ZG</i>	<i>ST STM32</i>	On-board	STM32F412ZGT6	100MHz	1MB	256KB
<i>ST Nucleo F413ZH</i>	<i>ST STM32</i>	On-board	STM32F413ZHT6	100MHz	512KB	128KB
<i>ST Nucleo F429ZI</i>	<i>ST STM32</i>	On-board	STM32F429ZIT6	180MHz	2MB	256KB
<i>ST Nucleo F439ZI</i>	<i>ST STM32</i>	On-board	STM32F439ZIT6	180MHz	2MB	256KB
<i>ST Nucleo F446RE</i>	<i>ST STM32</i>	On-board	STM32F446RET6	180MHz	512KB	128KB
<i>ST Nucleo F446ZE</i>	<i>ST STM32</i>	On-board	STM32F446ZET6	180MHz	512KB	128KB
<i>ST Nucleo F722ZE</i>	<i>ST STM32</i>	On-board	STM32F722ZET6	216MHz	512KB	256KB
<i>ST Nucleo F746ZG</i>	<i>ST STM32</i>	On-board	STM32F746ZGT6	216MHz	1MB	320KB
<i>ST Nucleo F756ZG</i>	<i>ST STM32</i>	On-board	STM32F756ZG	216MHz	1MB	320KB
<i>ST Nucleo F767ZI</i>	<i>ST STM32</i>	On-board	STM32F767ZIT6	216MHz	2MB	512KB
<i>ST Nucleo H743ZI</i>	<i>ST STM32</i>	On-board	STM32H743ZIT6	400MHz	2MB	1MB
<i>ST Nucleo L011K4</i>	<i>ST STM32</i>	On-board	STM32L011K4T6	32MHz	16KB	2KB
<i>ST Nucleo L031K6</i>	<i>ST STM32</i>	On-board	STM32L031K6T6	32MHz	32KB	8KB
<i>ST Nucleo L053R8</i>	<i>ST STM32</i>	On-board	STM32L053R8T6	32MHz	64KB	8KB
<i>ST Nucleo L073RZ</i>	<i>ST STM32</i>	On-board	STM32L073RZ	32MHz	192KB	20KB

Continued on next page

Table 26 – continued from previous page

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>ST Nucleo L152RE</i>	<i>ST STM32</i>	On-board	STM32L152RET6	32MHz	512KB	80KB
<i>ST Nucleo L412KB</i>	<i>ST STM32</i>	On-board	STM32L412KBU6	80MHz	128KB	40KB
<i>ST Nucleo L432KC</i>	<i>ST STM32</i>	On-board	STM32L432KCU6	80MHz	256KB	64KB
<i>ST Nucleo L433RC-P</i>	<i>ST STM32</i>	On-board	STM32L433RC	80MHz	256KB	64KB
<i>ST Nucleo L452RE</i>	<i>ST STM32</i>	On-board	STM32L452RET6	80MHz	256KB	64KB
<i>ST Nucleo L476RG</i>	<i>ST STM32</i>	On-board	STM32L476RGT6	80MHz	1MB	128KB
<i>ST Nucleo L486RG</i>	<i>ST STM32</i>	On-board	STM32L486RGT6	80MHz	1MB	128KB
<i>ST Nucleo L496ZG</i>	<i>ST STM32</i>	On-board	STM32L496ZGT6	80MHz	1MB	128KB
<i>ST Nucleo L496ZG-P</i>	<i>ST STM32</i>	On-board	STM32L496ZGT6P	80MHz	1MB	320KB
<i>ST Nucleo L4R5ZI</i>	<i>ST STM32</i>	On-board	STM32L4R5ZIT6	120MHz	2MB	640KB
<i>ST STM32F0308DISCOVERY</i>	<i>ST STM32</i>	On-board	STM32F030R8T6	48MHz	64KB	8KB
<i>ST STM32F0DISCOVERY</i>	<i>ST STM32</i>	On-board	STM32F051R8T6	48MHz	64KB	8KB
<i>ST STM32F3DISCOVERY</i>	<i>ST STM32</i>	On-board	STM32F303VCT6	72MHz	256KB	48KB
<i>ST STM32F4DISCOVERY</i>	<i>ST STM32</i>	On-board	STM32F407VGT6	168MHz	1MB	128KB
<i>ST STM32L073Z-EVAL</i>	<i>ST STM32</i>	On-board	STM32L073VZT6	32MHz	192KB	20KB
<i>ST STM32LDISCOVERY</i>	<i>ST STM32</i>	On-board	STM32L152RBT6	32MHz	128KB	16KB
<i>ST STM32VLDISCOVERY</i>	<i>ST STM32</i>	On-board	STM32F100RBT6	24MHz	128KB	8KB
<i>STM32F7508-DK</i>	<i>ST STM32</i>	On-board	STM32F750N8H6	216MHz	64KB	340KB

SeeedStudio

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Seeed Arch Max</i>	<i>ST STM32</i>	On-board	STM32F407VET6	168MHz	512KB	192KB
<i>Seeed Wio 3G</i>	<i>ST STM32</i>	On-board	STM32F439VI	180MHz	2MB	256KB

Semtech

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>NAMote72</i>	<i>ST STM32</i>	External	STM32L152RC	32MHz	256KB	32KB

TauLabs

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Sparky V1 F303</i>	<i>ST STM32</i>	External	STM32F303CCT6	72MHz	256KB	40KB

VAE

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>VAkE v1.0</i>	<i>ST STM32</i>	External	STM32F446RET6	180MHz	512KB	128KB

rhomb.io

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>L476DMWIK</i>	<i>ST STM32</i>	On-board	STM32L476VGT6	80MHz	1MB	128KB

sakura.io

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>sakura.io Evaluation Board</i>	<i>ST STM32</i>	On-board	STM32F411RET6	100MHz	1MB	128KB

u-blox

Name	Platform	Debug	MCU	Fre-quency	Flash	RAM
<i>Mbed Connect Cloud</i>	<i>ST STM32</i>	On-board	STM32F439ZIY6	168MHz	2MB	256KB
<i>u-blox C030-N211 IoT Starter Kit</i>	<i>ST STM32</i>	External	STM32F437VG	180MHz	1MB	256KB
<i>u-blox C030-R410M IoT</i>	<i>ST STM32</i>	On-board	STM32F437VG	180MHz	1MB	256KB
<i>u-blox C030-U201 IoT Starter Kit</i>	<i>ST STM32</i>	External	STM32F437VG	180MHz	1MB	256KB
<i>u-blox EVK-ODIN-W2</i>	<i>ST STM32</i>	External	STM32F439ZIY6	168MHz	2MB	256KB
<i>u-blox ODIN-W2</i>	<i>ST STM32</i>	External	STM32F439ZIY6	168MHz	2MB	256KB

1.11.19 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*
 - *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

Banks listed below have on-board debug probe and **ARE READY** for debugging! You do not need to use/buy external debug probe.

Name	Platform	MCU	Frequency	Flash	RAM
Samsung ARTIK053	Samsung ARTIK	S5JT200	320MHz	8MB	1.25MB

Examples

- Tizen RT for Samsung ARTIK

Platforms

Name	Description
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.

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

Name	Platform	Debug	MCU	Frequency	Flash	RAM
Samsung ARTIK053	Samsung ARTIK	On-board	S5JT200	320MHz	8MB	1.25MB

1.11.20 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

Name	Description
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.

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

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Raspberry Pi 1 Model B</i>	Linux ARM	No	BCM2835	700MHz	512MB	512MB
<i>Raspberry Pi 2 Model B</i>	Linux ARM	No	BCM2836	900MHz	1GB	1GB
<i>Raspberry Pi 3 Model B</i>	Linux ARM	No	BCM2837	1200MHz	1GB	1GB
<i>Raspberry Pi Zero</i>	Linux ARM	No	BCM2835	1000MHz	512MB	512MB

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 Low Cost RTK

Contents

- *Aceinna Low Cost RTK*
 - *Hardware*
 - *Configuration*
 - *Uploading*
 - *Debugging*
 - *Frameworks*

Hardware

Platform [*Aceinna IMU*](#): Open-source, embedded development platform for Aceinna IMU hardware. Run custom algorithms and navigation code on Aceinna IMU/INS hardware.

Microcontroller	STM32F469NIH6
Frequency	180MHz
Flash	1MB
RAM	384KB
Vendor	Aceinna

Configuration

Please use LowCostRTK ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:LowCostRTK]
platform = aceinna_imu
board = LowCostRTK
```

You can override default Aceinna Low Cost RTK settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `LowCostRTK.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:LowCostRTK]
platform = aceinna_imu
board = LowCostRTK

; change microcontroller
board_build.mcu = stm32f469nih6

; change MCU frequency
board_build.f_cpu = 180000000L
```

Uploading

Aceinna Low Cost RTK supports the next uploading protocols:

- blackmagic
- jlink
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```
[env:LowCostRTK]
platform = aceinna_imu
board = LowCostRTK

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 Low Cost RTK 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
<i>Black Magic Probe</i>		
<i>J-LINK</i>		
<i>ST-LINK</i>	Yes	Yes

Frameworks

Name	Description
<i>STM32</i>	STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.

Aceinna OpenIMU 300ZA

Contents

- Aceinna OpenIMU 300ZA
 - *Hardware*
 - *Configuration*
 - *Uploading*
 - *Debugging*

Hardware

Platform *Aceinna IMU*: Open-source, embedded development platform for Aceinna IMU hardware. Run custom algorithms and navigation code on Aceinna IMU/INS hardware.

Microcontroller	STM32F405RG
Frequency	120MHz
Flash	1MB
RAM	128KB
Vendor	Aceinna

Configuration

Please use OpenIMU300 ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[env:OpenIMU300]
platform = aceinna_imu
board = OpenIMU300
```

You can override default Aceinna OpenIMU 300ZA settings per build environment using *board_**** option, where *** is a JSON object path from board manifest *OpenIMU300.json*. For example, *board_build.mcu*, *board_build.f_cpu*, etc.

```
[env:OpenIMU300]
platform = aceinna_imu
board = OpenIMU300

; 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:

```
[env:OpenIMU300]
platform = aceinna_imu
board = OpenIMU300

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
<i>Black Magic Probe</i>		
<i>J-LINK</i>		
<i>ST-LINK</i>		Yes

Aceinna OpenIMU 300ZA

Contents

- Aceinna OpenIMU 300ZA
 - *Hardware*
 - *Configuration*
 - *Uploading*
 - *Debugging*

Hardware

Platform *Aceinna IMU*: Open-source, embedded development platform for Aceinna IMU hardware. Run custom algorithms and navigation code on Aceinna IMU/INS hardware.

Microcontroller	STM32F405RG
Frequency	120MHz
Flash	1MB
RAM	128KB
Vendor	Aceinna

Configuration

Please use OpenIMU300ZA ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[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.

```
[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:

```
[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
<i>Black Magic Probe</i>		
<i>J-LINK</i>		
<i>ST-LINK</i>		Yes

Aceinna OpenIMU 330

Contents

- *Aceinna OpenIMU 330*
 - *Hardware*
 - *Configuration*
 - *Uploading*
 - *Debugging*
 - *Frameworks*

Hardware

Platform *Aceinna IMU*: Open-source, embedded development platform for Aceinna IMU hardware. Run custom algorithms and navigation code on Aceinna IMU/INS hardware.

Microcontroller	STM32L431CB
Frequency	80MHz
Flash	128KB
RAM	64KB
Vendor	Aceinna

Configuration

Please use OpenIMU330 ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:OpenIMU330]
platform = aceinna_imu
board = OpenIMU330
```

You can override default Aceinna OpenIMU 330 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `OpenIMU330.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:OpenIMU330]
platform = aceinna_imu
board = OpenIMU330

; change microcontroller
board_build.mcu = stm32l431cb

; change MCU frequency
board_build.f_cpu = 80000000L
```

Uploading

Aceinna OpenIMU 330 supports the next uploading protocols:

- blackmagic
- jlink
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```
[env:OpenIMU330]
platform = aceinna_imu
board = OpenIMU330

upload_protocol = stlink
```

Debugging

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

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using *debug_tool* option in “*platformio.ini*” (*Project Configuration File*).

Aceinna OpenIMU 330 does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

Compatible Tools	On-board	Default
Black Magic Probe		
J-LINK		
ST-LINK		Yes

Frameworks

Name	Description
STM32	STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.

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.

Microcontroller	AT90CAN128
Frequency	16MHz
Flash	127KB
RAM	4KB
Vendor	Microchip

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
board_build.mcu = at90can128

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

PIO Unified Debugger currently does not support AT90CAN128 board.

Frameworks

Name	Description
<code>Ar-duino</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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 industry's most code-efficient architecture for C and assembly programming.

Microcontroller	AT90CAN32
Frequency	16MHz
Flash	31KB
RAM	2KB
Vendor	Microchip

Configuration

Please use AT90CAN32 ID for `board` option in “*platformio.ini*” (*Project Configuration File*):

```
[env:AT90CAN32]
platform = atmelavr
board = AT90CAN32
```

You can override default AT90CAN32 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest [AT90CAN32.json](#). For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[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

Name	Description
Arduin	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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 industry's most code-efficient architecture for C and assembly programming.

Microcontroller	AT90CAN64
Frequency	16MHz
Flash	63KB
RAM	4KB
Vendor	Microchip

Configuration

Please use AT90CAN64 ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[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.

```
[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

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.

ATmega128/A

Contents

- *ATmega128/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.

Microcontroller	ATMEGA128
Frequency	16MHz
Flash	127KB
RAM	4KB
Vendor	Microchip

Configuration

Please use ATmega128 ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[env:ATmega128]
platform = atmelavr
board = ATmega128
```

You can override default ATmega128/A settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `ATmega128.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:ATmega128]
platform = atmelavr
board = ATmega128

; change microcontroller
```

(continues on next page)

(continued from previous page)

```
board_build.mcu = atmega128
; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

PIO Unified Debugger currently does not support ATmega128/A board.

Frameworks

Name	Description
Arduin	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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 industrys most code-efficient architecture for C and assembly programming.

Microcontroller	ATMEGA1280
Frequency	16MHz
Flash	127KB
RAM	8KB
Vendor	Microchip

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

Name	Description
<code>Ar-duino</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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.

Microcontroller	ATMEGA1281
Frequency	16MHz
Flash	127KB
RAM	8KB
Vendor	Microchip

Configuration

Please use ATmega1281 ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[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.

```
[env:ATmega1281]
platform = atmelavr
board = ATmega1281

; change microcontroller
board_build.mcu = atmega1281

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

PIO Unified Debugger currently does not support ATmega1281 board.

Frameworks

Name	Description
<code>Ar- duino</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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.

Microcontroller	ATMEGA1284
Frequency	16MHz
Flash	127KB
RAM	16KB
Vendor	Microchip

Configuration

Please use ATmega1284 ID for *board* option in “*platformio.ini*” (Project Configuration File):

```
[env:ATmega1284]
platform = atmelavr
board = ATmega1284
```

You can override default ATmega1284 settings per build environment using *board_**** option, where *** is a JSON object path from board manifest [ATmega1284.json](#). For example, *board_build.mcu*, *board_build.f_cpu*, etc.

```
[env:ATmega1284]
platform = atmelavr
board = ATmega1284

; change microcontroller
board_build.mcu = atmega1284

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

PIO Unified Debugger currently does not support ATmega1284 board.

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.

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 industrys most code-efficient architecture for C and assembly programming.

Microcontroller	ATMEGA1284P
Frequency	16MHz
Flash	127KB
RAM	16KB
Vendor	Microchip

Configuration

Please use ATmega1284P ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[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.

```
[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

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.

ATmega16

Contents

- [*ATmega16*](#)
 - [*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	ATMEGA16
Frequency	16MHz
Flash	15.50KB
RAM	1KB
Vendor	Microchip

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
```

(continues on next page)

(continued from previous page)

```
; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

PIO Unified Debugger currently does not support ATmega16 board.

Frameworks

Name	Description
Arduin	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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 industry's most code-efficient architecture for C and assembly programming.

Microcontroller	ATMEGA164A
Frequency	16MHz
Flash	15.50KB
RAM	1KB
Vendor	Microchip

Configuration

Please use ATmega164A ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[env:ATmega164A]
platform = atmelavr
board = ATmega164A
```

You can override default ATmega164A settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `ATmega164A.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:ATmega164A]
platform = atmelavr
board = ATmega164A

; change microcontroller
board_build.mcu = atmega164a

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

PIO Unified Debugger currently does not support ATmega164A board.

Frameworks

Name	Description
<code>Ar-duino</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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.

Microcontroller	ATMEGA164P
Frequency	16MHz
Flash	15.50KB
RAM	1KB
Vendor	Microchip

Configuration

Please use ATmega164P ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[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.

```
[env:ATmega164P]
platform = atmelavr
board = ATmega164P

; change microcontroller
board_build.mcu = atmega164p

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

PIO Unified Debugger currently does not support ATmega164P/PA board.

Frameworks

Name	Description
<code>Ardu</code> <code>ino</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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 industry's most code-efficient architecture for C and assembly programming.

Microcontroller	ATMEGA168
Frequency	16MHz
Flash	15.50KB
RAM	1KB
Vendor	Microchip

Configuration

Please use ATmega168 ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[env:ATmega168]
platform = atmelavr
board = ATmega168
```

You can override default ATmega168/A settings per build environment using *board_**** option, where *** is a JSON object path from board manifest [ATmega168.json](#). For example, *board_build.mcu*, *board_build.f_cpu*, etc.

```
[env:ATmega168]
platform = atmelavr
board = ATmega168

; change microcontroller
board_build.mcu = atmega168

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

PIO Unified Debugger currently does not support ATmega168/A board.

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.

ATmega168P/PA

Contents

- ATmega168P/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.

Microcontroller	ATMEGA168P
Frequency	16MHz
Flash	15.50KB
RAM	1KB
Vendor	Microchip

Configuration

Please use ATmega168P ID for `board` option in “`platformio.ini`” (Project Configuration File):

```
[env:ATmega168P]
platform = atmelavr
board = ATmega168P
```

You can override default ATmega168P/PA settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `ATmega168P.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:ATmega168P]
platform = atmelavr
board = ATmega168P

; change microcontroller
board_build.mcu = atmega168p

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

`PIO Unified Debugger` currently does not support ATmega168P/PA board.

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.

ATmega168PB

Contents

- [ATmega168PB](#)
 - [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	ATMEGA168PB
Frequency	16MHz
Flash	15.50KB
RAM	1KB
Vendor	Microchip

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
```

(continues on next page)

(continued from previous page)

```
board_build.mcu = atmega168pb
; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

PIO Unified Debugger currently does not support ATmega168PB board.

Frameworks

Name	Description
Arduin	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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 industrys most code-efficient architecture for C and assembly programming.

Microcontroller	ATMEGA2560
Frequency	16MHz
Flash	255KB
RAM	8KB
Vendor	Microchip

Configuration

Please use ATmega2560 ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:ATmega2560]
platform = atmelavr
board = ATmega2560
```

You can override default ATmega2560 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `ATmega2560.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:ATmega2560]
platform = atmelavr
board = ATmega2560

; change microcontroller
board_build.mcu = atmega2560

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

PIO Unified Debugger currently does not support ATmega2560 board.

Frameworks

Name	Description
<code>Ar-duino</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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.

Microcontroller	ATMEGA2561
Frequency	16MHz
Flash	255KB
RAM	8KB
Vendor	Microchip

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
board_build.mcu = atmega2561

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

PIO Unified Debugger currently does not support ATmega2561 board.

Frameworks

Name	Description
<code>Ardu</code> <code>ino</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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.

Microcontroller	ATMEGA32
Frequency	16MHz
Flash	31.50KB
RAM	2KB
Vendor	Microchip

Configuration

Please use ATmega32 ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[env:ATmega32]
platform = atmelavr
board = ATmega32
```

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

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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.

Microcontroller	ATMEGA324A
Frequency	16MHz
Flash	31.50KB
RAM	2KB
Vendor	Microchip

Configuration

Please use ATmega324A ID for `board` option in “`platformio.ini`” (Project Configuration File):

```
[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.

```
[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

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.

ATmega324P

Contents

- [ATmega324P](#)
 - [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	ATMEGA324P
Frequency	16MHz
Flash	31.50KB
RAM	2KB
Vendor	Microchip

Configuration

Please use ATmega324P ID for `board` option in “`platformio.ini`” (Project Configuration File):

```
[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.

```
[env:ATmega324P]
platform = atmelavr
board = ATmega324P

; change microcontroller
```

(continues on next page)

(continued from previous page)

```
board_build.mcu = atmega324p
; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

PIO Unified Debugger currently does not support ATmega324P board.

Frameworks

Name	Description
Arduin	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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 industrys most code-efficient architecture for C and assembly programming.

Microcontroller	ATMEGA324PA
Frequency	16MHz
Flash	31.50KB
RAM	2KB
Vendor	Microchip

Configuration

Please use ATmega324PA ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:ATmega324PA]
platform = atmelavr
board = ATmega324PA
```

You can override default ATmega324PA settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest [ATmega324PA.json](#). For example, `board_build.mcu`, `board_build.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

Name	Description
Arduin	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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.

Microcontroller	ATMEGA324PB
Frequency	16MHz
Flash	31.50KB
RAM	2KB
Vendor	Microchip

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
board_build.mcu = atmega324pb

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

PIO Unified Debugger currently does not support ATmega324PB board.

Frameworks

Name	Description
<code>Ardu</code> <code>ino</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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.

Microcontroller	ATMEGA328
Frequency	16MHz
Flash	31.50KB
RAM	2KB
Vendor	Microchip

Configuration

Please use ATmega328 ID for `board` option in “*platformio.ini*” (*Project Configuration File*):

```
[env:ATmega328]
platform = atmelavr
board = ATmega328
```

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.

```
[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

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.

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.

Microcontroller	ATMEGA328P
Frequency	16MHz
Flash	31.50KB
RAM	2KB
Vendor	Microchip

Configuration

Please use ATmega328P ID for `board` option in “`platformio.ini`” (Project Configuration File):

```
[env:ATmega328P]
platform = atmelavr
board = ATmega328P
```

You can override default ATmega328P/PA settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `ATmega328P.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:ATmega328P]
platform = atmelavr
board = ATmega328P

; change microcontroller
board_build.mcu = atmega328p

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

`PIO Unified Debugger` currently does not support ATmega328P/PA board.

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.

ATmega328PB

Contents

- ATmega328PB
 - 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	ATMEGA328PB
Frequency	16MHz
Flash	31.50KB
RAM	2KB
Vendor	Microchip

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
```

(continues on next page)

(continued from previous page)

```
board_build.mcu = atmega328pb
; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

PIO Unified Debugger currently does not support ATmega328PB board.

Frameworks

Name	Description
Arduin	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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 industrys most code-efficient architecture for C and assembly programming.

Microcontroller	ATMEGA48
Frequency	16MHz
Flash	4KB
RAM	512B
Vendor	Microchip

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

Name	Description
<code>Arduin</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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 industry's most code-efficient architecture for C and assembly programming.

Microcontroller	ATMEGA48PB
Frequency	16MHz
Flash	4KB
RAM	512B
Vendor	Microchip

Configuration

Please use ATmega48PB ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[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.

```
[env:ATmega48PB]
platform = atmelavr
board = ATmega48PB

; change microcontroller
board_build.mcu = atmega48pb

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

PIO Unified Debugger currently does not support ATmega48PB board.

Frameworks

Name	Description
<code>Ar- duino</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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 industry's most code-efficient architecture for C and assembly programming.

Microcontroller	ATMEGA64
Frequency	16MHz
Flash	63KB
RAM	4KB
Vendor	Microchip

Configuration

Please use ATmega64 ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[env:ATmega64]
platform = atmelavr
board = ATmega64
```

You can override default ATmega64/A settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest [ATmega64.json](#). For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[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

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.

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 industrys most code-efficient architecture for C and assembly programming.

Microcontroller	ATMEGA640
Frequency	16MHz
Flash	63KB
RAM	8KB
Vendor	Microchip

Configuration

Please use ATmega640 ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:ATmega640]
platform = atmelavr
board = ATmega640
```

You can override default ATmega640 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `ATmega640.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:ATmega640]
platform = atmelavr
board = ATmega640

; change microcontroller
board_build.mcu = atmega640

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

PIO Unified Debugger currently does not support ATmega640 board.

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.

ATmega644/A

Contents

- ATmega644/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.

Microcontroller	ATMEGA644A
Frequency	16MHz
Flash	63KB
RAM	4KB
Vendor	Microchip

Configuration

Please use ATmega644A ID for `board` option in “*platformio.ini*” (Project Configuration File):

```
[env:ATmega644A]
platform = atmelavr
board = ATmega644A
```

You can override default ATmega644/A settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest [ATmega644A.json](#). For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:ATmega644A]
platform = atmelavr
board = ATmega644A

; change microcontroller
```

(continues on next page)

(continued from previous page)

```
board_build.mcu = atmega644a
; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

PIO Unified Debugger currently does not support ATmega644/A board.

Frameworks

Name	Description
Arduin	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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 industrys most code-efficient architecture for C and assembly programming.

Microcontroller	ATMEGA644P
Frequency	16MHz
Flash	63KB
RAM	4KB
Vendor	Microchip

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

Name	Description
<code>Ar-duino</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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 industry's most code-efficient architecture for C and assembly programming.

Microcontroller	ATMEGA8
Frequency	16MHz
Flash	7.50KB
RAM	1KB
Vendor	Microchip

Configuration

Please use ATmega8 ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:ATmega8]
platform = atmelavr
board = ATmega8
```

You can override default ATmega8/A settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `ATmega8.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:ATmega8]
platform = atmelavr
board = ATmega8

; change microcontroller
board_build.mcu = atmega8

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

PIO Unified Debugger currently does not support ATmega8/A board.

Frameworks

Name	Description
<code>Ar-duino</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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.

Microcontroller	ATMEGA8535
Frequency	16MHz
Flash	7.50KB
RAM	512B
Vendor	Microchip

Configuration

Please use ATmega8535 ID for `board` option in “*platformio.ini*” (*Project Configuration File*):

```
[env:ATmega8535]
platform = atmelavr
board = ATmega8535
```

You can override default ATmega8535 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `ATmega8535.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:ATmega8535]
platform = atmelavr
board = ATmega8535

; change microcontroller
board_build.mcu = atmega8535

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

PIO Unified Debugger currently does not support ATmega8535 board.

Frameworks

Name	Description
Arduin	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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 industrys most code-efficient architecture for C and assembly programming.

Microcontroller	ATMEGA88
Frequency	16MHz
Flash	7.50KB
RAM	1KB
Vendor	Microchip

Configuration

Please use ATmega88 ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:ATmega88]
platform = atmelavr
board = ATmega88
```

You can override default ATmega88/A settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `ATmega88.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:ATmega88]
platform = atmelavr
board = ATmega88

; change microcontroller
board_build.mcu = atmega88

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

`PIO Unified Debugger` currently does not support ATmega88/A board.

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.

ATmega88P/PA

Contents

- ATmega88P/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.

Microcontroller	ATMEGA88P
Frequency	16MHz
Flash	7.50KB
RAM	1KB
Vendor	Microchip

Configuration

Please use ATmega88P ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:ATmega88P]
platform = atmelavr
board = ATmega88P
```

You can override default ATmega88P/PA settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `ATmega88P.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:ATmega88P]
platform = atmelavr
board = ATmega88P

; change microcontroller
```

(continues on next page)

(continued from previous page)

```
board_build.mcu = atmega88p
; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

PIO Unified Debugger currently does not support ATmega88P/PA board.

Frameworks

Name	Description
Arduin	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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.

Microcontroller	ATMEGA88PB
Frequency	16MHz
Flash	7.50KB
RAM	1KB
Vendor	Microchip

Configuration

Please use ATmega88PB ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[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.

```
[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

Name	Description
Arduin	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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.

Microcontroller	ATMEGA48P
Frequency	16MHz
Flash	4KB
RAM	512B
Vendor	Microchip

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
board_build.mcu = atmega48p

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

PIO Unified Debugger currently does not support ATmega8P/PA board.

Frameworks

Name	Description
<code>Arduin</code>	Arduino Wiring-based Framework allows writing cross-platform software 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 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.

Microcontroller	ATMEGA32U4
Frequency	8MHz
Flash	28KB
RAM	2.50KB
Vendor	Adafruit

Configuration

Please use `bluefruitmicro` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:bluefruitmicro]
platform = atmelavr
board = bluefruitmicro
```

You can override default Adafruit Bluefruit Micro settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `bluefruitmicro.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[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

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 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	ATMEGA32U4
Frequency	8MHz
Flash	28KB
RAM	2.50KB
Vendor	Adafruit

Configuration

Please use `circuitplay_classic` ID for `board` option in “`platformio.ini`” (Project Configuration File):

```
[env:circuitplay_classic]
platform = atmelavr
board = circuitplay_classic
```

You can override default Adafruit Circuit Playground Classic settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `circuitplay_classic.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:circuitplay_classic]
platform = atmelavr
board = circuitplay_classic

; change microcontroller
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

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 Feather 328P

Contents

- *Adafruit Feather 328P*
 - *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	ATMEGA328P
Frequency	8MHz
Flash	31.50KB
RAM	2KB
Vendor	Adafruit

Configuration

Please use `feather328p` ID for `board` option in “`platformio.ini`” (Project Configuration File):

```
[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.

```
[env:feather328p]
platform = atmelavr
board = feather328p

; change microcontroller
```

(continues on next page)

(continued from previous page)

```
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

Name	Description
Arduin	Arduino Wiring-based Framework allows writing cross-platform software 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 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 industrys most code-efficient architecture for C and assembly programming.

Microcontroller	ATMEGA32U4
Frequency	8MHz
Flash	28KB
RAM	2.50KB
Vendor	Adafruit

Configuration

Please use `feather32u4` ID for `board` option in “`platformio.ini`” (Project Configuration File):

```
[env:feather32u4]
platform = atmelavr
board = feather32u4
```

You can override default Adafruit Feather 32u4 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `feather32u4.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:feather32u4]
platform = atmelavr
board = feather32u4

; change microcontroller
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

Name	Description
<code>Ar-duino</code>	Arduino Wiring-based Framework allows writing cross-platform software 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 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.

Microcontroller	ATMEGA32U4
Frequency	8MHz
Flash	28KB
RAM	2.50KB
Vendor	Adafruit

Configuration

Please use `flora8` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:flora8]
platform = atmelavr
board = flora8
```

You can override default Adafruit Flora settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `flora8.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:flora8]
platform = atmelavr
board = flora8

; change microcontroller
board_build.mcu = atmega32u4

; change MCU frequency
board_build.f_cpu = 8000000L
```

Debugging

PIO Unified Debugger currently does not support Adafruit Flora board.

Frameworks

Name	Description
<code>Ar- duino</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

Adafruit Gemma

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 industry's most code-efficient architecture for C and assembly programming.

Microcontroller	ATTINY85
Frequency	8MHz
Flash	8KB
RAM	512B
Vendor	Adafruit

Configuration

Please use gemma ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[env:gemma]
platform = atmelavr
board = gemma
```

You can override default Adafruit Gemma settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `gemma.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[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

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 3V/8MHz

Contents

- Adafruit ItsyBitsy 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.

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

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 5V/16MHz

Contents

- [*Adafruit ItsyBitsy 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.

Microcontroller	ATMEGA32U4
Frequency	16MHz
Flash	28KB
RAM	2.50KB
Vendor	Adafruit

Configuration

Please use `itsybitsy32u4_5V` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[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
```

(continues on next page)

(continued from previous page)

```
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

Name	Description
Arduin	Arduino Wiring-based Framework allows writing cross-platform software 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 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 industrys most code-efficient architecture for C and assembly programming.

Microcontroller	ATMEGA328P
Frequency	16MHz
Flash	31.50KB
RAM	2KB
Vendor	Adafruit

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.

Frameworks

Name	Description
<code>Ar-duino</code>	Arduino Wiring-based Framework allows writing cross-platform software 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 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.

Microcontroller	ATMEGA328P
Frequency	12MHz
Flash	28KB
RAM	2KB
Vendor	Adafruit

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
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

Name	Description
<code>Arduin</code>	Arduino Wiring-based Framework allows writing cross-platform software 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 Pro Trinket 3V/12MHz (USB)

Contents

- *Adafruit Pro Trinket 3V/12MHz (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 industry's most code-efficient architecture for C and assembly programming.

Microcontroller	ATMEGA328P
Frequency	12MHz
Flash	28KB
RAM	2KB
Vendor	Adafruit

Configuration

Please use protrinket3 ID for `board` option in “`platformio.ini`” (Project Configuration File):

```
[env:protrinket3]
platform = atmelavr
board = protrinket3
```

You can override default Adafruit Pro Trinket 3V/12MHz (USB) settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `protrinket3.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:protrinket3]
platform = atmelavr
board = protrinket3

; change microcontroller
board_build.mcu = atmega328p

; change MCU frequency
board_build.f_cpu = 12000000L
```

Debugging

PIO Unified Debugger currently does not support Adafruit Pro Trinket 3V/12MHz (USB) board.

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 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.

Microcontroller	ATMEGA328P
Frequency	16MHz
Flash	28KB
RAM	2KB
Vendor	Adafruit

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

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 Pro Trinket 5V/16MHz (USB)

Contents

- [*Adafruit Pro Trinket 5V/16MHz \(USB\)*](#)
 - [*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 industry's most code-efficient architecture for C and assembly programming.

Microcontroller	ATMEGA328P
Frequency	16MHz
Flash	28KB
RAM	2KB
Vendor	Adafruit

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
```

(continues on next page)

(continued from previous page)

```
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

Name	Description
Arduin	Arduino Wiring-based Framework allows writing cross-platform software 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 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 industrys most code-efficient architecture for C and assembly programming.

Microcontroller	ATTINY85
Frequency	8MHz
Flash	8KB
RAM	512B
Vendor	Adafruit

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

Name	Description
<code>Arduin</code>	Arduino Wiring-based Framework allows writing cross-platform software 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 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.

Microcontroller	ATTINY85
Frequency	16MHz
Flash	8KB
RAM	512B
Vendor	Adafruit

Configuration

Please use `trinket5` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:trinket5]
platform = atmelavr
board = trinket5
```

You can override default Adafruit Trinket 5V/16MHz settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `trinket5.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:trinket5]
platform = atmelavr
board = trinket5

; change microcontroller
board_build.mcu = attiny85

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

PIO Unified Debugger currently does not support Adafruit Trinket 5V/16MHz board.

Frameworks

Name	Description
<code>Arduin</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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.

Microcontroller	ATMEGA328P
Frequency	16MHz
Flash	31.50KB
RAM	2KB
Vendor	Alorium Technology

Configuration

Please use `alorium_hinj` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:alorium_hinj]
platform = atmelavr
board = alorium_hinj
```

You can override default Alorium Hinj settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `alorium_hinj.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:alorium_hinj]
platform = atmelavr
board = alorium_hinj

; change microcontroller
board_build.mcu = atmega328p

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

PIO Unified Debugger currently does not support Alorium Hinj board.

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.

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	ATMEGA328P
Frequency	16MHz
Flash	31.50KB
RAM	2KB
Vendor	Alorium Technology

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

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.

Alorium XLR8

Contents

- [Alorium XLR8](#)
 - [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	ATMEGA328P
Frequency	16MHz
Flash	31.50KB
RAM	2KB
Vendor	Alorium Technology

Configuration

Please use `alorium_xlr8` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:alorium_xlr8]
platform = atmelavr
board = alorium_xlr8
```

You can override default Alorium XLR8 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `alorium_xlr8.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:alorium_xlr8]
platform = atmelavr
board = alorium_xlr8

; change microcontroller
```

(continues on next page)

(continued from previous page)

```
board_build.mcu = atmega328p
; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

PIO Unified Debugger currently does not support Alorium XLR8 board.

Frameworks

Name	Description
Arduin	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

Altair

Contents

- [Altair](#)
 - [Hardware](#)
 - [Configuration](#)
 - [Debugging](#)
 - [Frameworks](#)

Hardware

Platform [Atmel AVR](#): Atmel AVR 8- 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	ATMEGA256RFR2
Frequency	16MHz
Flash	248KB
RAM	32KB
Vendor	makerlab.mx

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

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.

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 industrys most code-efficient architecture for C and assembly programming.

Microcontroller	ATMEGA328P
Frequency	16MHz
Flash	31.50KB
RAM	2KB
Vendor	Anarduino

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
board_build.mcu = atmega328p

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

PIO Unified Debugger currently does not support Anarduino MiniWireless board.

Frameworks

Name	Description
Arduino	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

Arduboy

Contents

- [Arduboy](#)
 - [Hardware](#)
 - [Configuration](#)

- *Debugging*
- *Frameworks*

Hardware

Platform [Atmel AVR](#): Atmel AVR 8- 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	ATMEGA32U4
Frequency	16MHz
Flash	28KB
RAM	2.50KB
Vendor	Arduboy

Configuration

Please use `arduboy` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:arduboy]
platform = atmelavr
board = arduboy
```

You can override default Arduboy settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `arduboy.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[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

Name	Description
<code>Arduin</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

Arduboy 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 industrys most code-efficient architecture for C and assembly programming.

Microcontroller	ATMEGA32U4
Frequency	16MHz
Flash	28KB
RAM	2.50KB
Vendor	Arduboy

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

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.

Arduino BT ATmega168

Contents

- *Arduino BT 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.

Microcontroller	ATMEGA168
Frequency	16MHz
Flash	14KB
RAM	1KB
Vendor	Arduino

Configuration

Please use `bstatmega168` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:bstatmega168]
platform = atmelavr
board = bstatmega168
```

You can override default Arduino BT ATmega168 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `bstatmega168.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:bstatmega168]
platform = atmelavr
board = bstatmega168

; change microcontroller
```

(continues on next page)

(continued from previous page)

```
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

Name	Description
Arduin	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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 industrys most code-efficient architecture for C and assembly programming.

Microcontroller	ATMEGA328P
Frequency	16MHz
Flash	28KB
RAM	2KB
Vendor	Arduino

Configuration

Please use `btagm328` ID for `board` option in “`platformio.ini`” (Project Configuration File):

```
[env:btatmega328]
platform = atmelavr
board = btatmega328
```

You can override default Arduino BT ATmega328 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `btatmega328.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:btatmega328]
platform = atmelavr
board = btatmega328

; change microcontroller
board_build.mcu = atmega328p

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

PIO Unified Debugger currently does not support Arduino BT ATmega328 board.

Frameworks

Name	Description
<code>Ar-duino</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

Arduino Duemilanove or Diecimila ATmega168

Contents

- *Arduino Duemilanove or Diecimila 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.

Microcontroller	ATMEGA168
Frequency	16MHz
Flash	14KB
RAM	1KB
Vendor	Arduino

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
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

Name	Description
<code>Arduin</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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.

Microcontroller	ATMEGA328P
Frequency	16MHz
Flash	30KB
RAM	2KB
Vendor	Arduino

Configuration

Please use `diecimilaatmega328` ID for `board` option in “`platformio.ini`” (Project Configuration File):

```
[env:diecimilaatmega328]
platform = atmelavr
board = diecimilaatmega328
```

You can override default Arduino Duemilanove or Diecimila ATmega328 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `diecimilaatmega328.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[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

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.

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.

Microcontroller	ATMEGA32U4
Frequency	16MHz
Flash	28KB
RAM	2.50KB
Vendor	Arduino

Configuration

Please use `esplora` ID for *board* option in “`platformio.ini`” (*Project Configuration File*):

```
[env:esplora]
platform = atmelavr
board = esplora
```

You can override default Arduino Esplora settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `esplora.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:esplora]
platform = atmelavr
board = esplora

; change microcontroller
board_build.mcu = atmega32u4

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

PIO Unified Debugger currently does not support Arduino Esplora board.

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.

Arduino Ethernet

Contents

- [Arduino Ethernet](#)
 - [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	ATMEGA328P
Frequency	16MHz
Flash	31.50KB
RAM	2KB
Vendor	Arduino

Configuration

Please use `ethernet` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:ethernet]
platform = atmelavr
board = ethernet
```

You can override default Arduino Ethernet settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `ethernet.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:ethernet]
platform = atmelavr
board = ethernet

; change microcontroller
```

(continues on next page)

(continued from previous page)

```
board_build.mcu = atmega328p
; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

PIO Unified Debugger currently does not support Arduino Ethernet board.

Frameworks

Name	Description
Arduin	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

Arduino Fio

Contents

- [Arduino Fio](#)
 - [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	ATMEGA328P
Frequency	8MHz
Flash	30KB
RAM	2KB
Vendor	Arduino

Configuration

Please use `fio` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:fio]
platform = atmelavr
board = fio
```

You can override default Arduino Fio settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `fio.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:fio]
platform = atmelavr
board = fio

; change microcontroller
board_build.mcu = atmega328p

; change MCU frequency
board_build.f_cpu = 8000000L
```

Debugging

PIO Unified Debugger currently does not support Arduino Fio board.

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.

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.

Microcontroller	ATMEGA32U4
Frequency	16MHz
Flash	28KB
RAM	2.50KB
Vendor	Arduino

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
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

Name	Description
<code>Arduin</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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.

Microcontroller	ATMEGA32U4
Frequency	16MHz
Flash	28KB
RAM	2.50KB
Vendor	Arduino

Configuration

Please use `leonardo` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:leonardo]
platform = atmelavr
board = leonardo
```

You can override default Arduino Leonardo settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `leonardo.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:leonardo]
platform = atmelavr
board = leonardo

; change microcontroller
board_build.mcu = atmega32u4

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

PIO Unified Debugger currently does not support Arduino Leonardo board.

Frameworks

Name	Description
<code>Arduin</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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.

Microcontroller	ATMEGA32U4
Frequency	16MHz
Flash	28KB
RAM	2.50KB
Vendor	Arduino

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

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.

Arduino LilyPad ATmega168

Contents

- [Arduino LilyPad 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.

Microcontroller	ATMEGA168
Frequency	8MHz
Flash	14KB
RAM	1KB
Vendor	Arduino

Configuration

Please use `lilypadatmega168` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:lilypadatmega168]
platform = atmelavr
board = lilypadatmega168
```

You can override default Arduino LilyPad ATmega168 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `lilypadatmega168.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:lilypadatmega168]
platform = atmelavr
board = lilypadatmega168

; change microcontroller
```

(continues on next page)

(continued from previous page)

```
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

Name	Description
Arduin	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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 industrys most code-efficient architecture for C and assembly programming.

Microcontroller	ATMEGA328P
Frequency	8MHz
Flash	30KB
RAM	2KB
Vendor	Arduino

Configuration

Please use `lilyepadatmega328` 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

Name	Description
<code>Arduin</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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 industry's most code-efficient architecture for C and assembly programming.

Microcontroller	ATMEGA32U4
Frequency	8MHz
Flash	28KB
RAM	2.50KB
Vendor	Arduino

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
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

Name	Description
<code>Ardu</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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 industry's most code-efficient architecture for C and assembly programming.

Microcontroller	ATMEGA2560
Frequency	16MHz
Flash	248KB
RAM	8KB
Vendor	Arduino

Configuration

Please use megaADK ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[env:megaADK]
platform = atmelavr
board = megaADK
```

You can override default Arduino Mega ADK settings per build environment using *board_**** option, where *** is a JSON object path from board manifest [megaADK.json](#). For example, *board_build.mcu*, *board_build.f_cpu*, etc.

```
[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

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.

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*):

```
[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.

```
[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

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.

Arduino Mega or Mega 2560 ATmega2560 (Mega 2560)

Contents

- *Arduino Mega or Mega 2560 ATmega2560 (Mega 2560)*
 - *Hardware*
 - *Configuration*
 - *Debugging*
 - *Frameworks*

Hardware

Platform [Atmel AVR](#): Atmel AVR 8- 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	ATMEGA2560
Frequency	16MHz
Flash	248KB
RAM	8KB
Vendor	Arduino

Configuration

Please use megaatmega2560 ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[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.

```
[env:megaatmega2560]
platform = atmelavr
board = megaatmega2560

; change microcontroller
```

(continues on next page)

(continued from previous page)

```
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

Name	Description
Ar-duino	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
Simba	Simba is an RTOS and build framework. It aims to make embedded programming easy and portable.

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.

Microcontroller	ATMEGA32U4
Frequency	16MHz
Flash	28KB
RAM	2.50KB
Vendor	Arduino

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

Name	Description
<code>Arduin</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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.

Microcontroller	ATMEGA168
Frequency	16MHz
Flash	14KB
RAM	1KB
Vendor	Arduino

Configuration

Please use `miniatmega168` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:miniatmega168]
platform = atmelavr
board = miniatmega168
```

You can override default Arduino Mini ATmega168 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `miniatmega168.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:miniatmega168]
platform = atmelavr
board = miniatmega168

; change microcontroller
board_build.mcu = atmega168

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

PIO Unified Debugger currently does not support Arduino Mini ATmega168 board.

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.

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.

Microcontroller	ATMEGA328P
Frequency	16MHz
Flash	28KB
RAM	2KB
Vendor	Arduino

Configuration

Please use `miniatmega328` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:miniatmega328]
platform = atmelavr
board = miniatmega328
```

You can override default Arduino Mini ATmega328 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `miniatmega328.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:miniatmega328]
platform = atmelavr
board = miniatmega328

; change microcontroller
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

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.

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.

Microcontroller	ATMEGA168
Frequency	16MHz
Flash	14KB
RAM	1KB
Vendor	Arduino

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

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.

Arduino NG or older ATmega8

Contents

- *Arduino NG or older ATmega8*
 - *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	ATMEGA8
Frequency	16MHz
Flash	7KB
RAM	1KB
Vendor	Arduino

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
```

(continues on next page)

(continued from previous page)

```
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

Name	Description
Arduin	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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 industrys most code-efficient architecture for C and assembly programming.

Microcontroller	ATMEGA168
Frequency	16MHz
Flash	14KB
RAM	1KB
Vendor	Arduino

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

Name	Description
<code>Arduin</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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.

Microcontroller	ATMEGA328P
Frequency	16MHz
Flash	30KB
RAM	2KB
Vendor	Arduino

Configuration

Please use `nanoatmega328` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:nanoatmega328]
platform = atmelavr
board = nanoatmega328
```

You can override default Arduino Nano ATmega328 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `nanoatmega328.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:nanoatmega328]
platform = atmelavr
board = nanoatmega328

; 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 board.

Frameworks

Name	Description
<code>Arduin</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<code>Simba</code>	Simba is an RTOS and build framework. It aims to make embedded programming easy and portable.

Arduino Nano ATmega328 (New Bootloader)

Contents

- *Arduino Nano ATmega328 (New Bootloader)*
 - *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	ATMEGA328P
Frequency	16MHz
Flash	30KB
RAM	2KB
Vendor	Arduino

Configuration

Please use `nanoatmega328new` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:nanoatmega328new]
platform = atmelavr
board = nanoatmega328new
```

You can override default Arduino Nano ATmega328 (New Bootloader) settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `nanoatmega328new.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[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

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.

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 time 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	ATMEGA168
Frequency	8MHz
Flash	14KB
RAM	1KB
Vendor	Arduino

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

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.

Arduino Pro or Pro Mini ATmega168 (5V, 16 MHz)

Contents

- *Arduino Pro or Pro Mini ATmega168 (5V, 16 MHz)*
 - *Hardware*
 - *Configuration*
 - *Debugging*
 - *Frameworks*

Hardware

Platform [Atmel AVR](#): Atmel AVR 8- 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	ATMEGA168
Frequency	16MHz
Flash	14KB
RAM	1KB
Vendor	Arduino

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
```

(continues on next page)

(continued from previous page)

```
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

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.

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 industrys most code-efficient architecture for C and assembly programming.

Microcontroller	ATMEGA328P
Frequency	8MHz
Flash	30KB
RAM	2KB
Vendor	Arduino

Configuration

Please use `pro8MHzatmega328` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[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.

```
[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

Name	Description
<code>Arduin</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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.

Microcontroller	ATMEGA328P
Frequency	16MHz
Flash	30KB
RAM	2KB
Vendor	Arduino

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
board_build.mcu = atmega328p

; 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

Name	Description
<code>Arduin</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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.

Microcontroller	ATMEGA32U4
Frequency	16MHz
Flash	28KB
RAM	2.50KB
Vendor	Arduino

Configuration

Please use `robotControl` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:robotControl]
platform = atmelavr
board = robotControl
```

You can override default Arduino Robot Control settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `robotControl.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:robotControl]
platform = atmelavr
board = robotControl

; change microcontroller
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

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.

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.

Microcontroller	ATMEGA32U4
Frequency	16MHz
Flash	28KB
RAM	2.50KB
Vendor	Arduino

Configuration

Please use `robotMotor` ID for `board` option in “`platformio.ini`” (Project Configuration File):

```
[env:robotMotor]
platform = atmelavr
board = robotMotor
```

You can override default Arduino Robot Motor settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `robotMotor.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:robotMotor]
platform = atmelavr
board = robotMotor

; change microcontroller
board_build.mcu = atmega32u4

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

`PIO Unified Debugger` currently does not support Arduino Robot Motor board.

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.

Arduino Uno

Contents

- *Arduino 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 industry's most code-efficient architecture for C and assembly programming.

Microcontroller	ATMEGA328P
Frequency	16MHz
Flash	31.50KB
RAM	2KB
Vendor	Arduino

Configuration

Please use uno ID for *board* option in “*platformio.ini*” (Project Configuration File):

```
[env:uno]
platform = atmelavr
board = uno
```

You can override default Arduino Uno settings per build environment using *board_**** option, where *** is a JSON object path from board manifest *uno.json*. For example, *board_build.mcu*, *board_build.f_cpu*, etc.

```
[env:uno]
platform = atmelavr
board = uno

; change microcontroller
board_build.mcu = atmega328p
```

(continues on next page)

(continued from previous page)

```
; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

PIO Unified Debugger currently does not support Arduino Uno board.

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>Simba</i>	Simba is an RTOS and build framework. It aims to make embedded programming easy and portable.

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.

Microcontroller	ATMEGA32U4
Frequency	16MHz
Flash	28KB
RAM	2.50KB
Vendor	Arduino

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

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.

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.

Microcontroller	ATMEGA32U4
Frequency	16MHz
Flash	28KB
RAM	2.50KB
Vendor	Arduino

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
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

Name	Description
Arduinio	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

Atmel AT90PWM216

Contents

- [Atmel AT90PWM216](#)
 - [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 industrys most code-efficient architecture for C and assembly programming.

Microcontroller	AT90PWM216
Frequency	16MHz
Flash	16KB
RAM	1KB
Vendor	Microchip

Configuration

Please use `at90pwm216` ID for `board` option in “*platformio.ini*” (*Project Configuration File*):

```
[env:at90pwm216]
platform = atmelavr
board = at90pwm216
```

You can override default Atmel AT90PWM216 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `at90pwm216.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[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 industrys most code-efficient architecture for C and assembly programming.

Microcontroller	AT90PWM316
Frequency	16MHz
Flash	16KB
RAM	1KB
Vendor	Microchip

Configuration

Please use at90pwm316 ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[env:at90pwm316]
platform = atmelavr
board = at90pwm316
```

You can override default Atmel AT90PWM316 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `at90pwm316.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[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 industry's most code-efficient architecture for C and assembly programming.

Microcontroller	ATMEGA328P
Frequency	16MHz
Flash	28KB
RAM	2KB
Vendor	BQ

Configuration

Please use `zumbt328` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[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.

```
[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

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.

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.

Microcontroller	ATMEGA328P
Frequency	16MHz
Flash	30KB
RAM	2KB
Vendor	BitWizard

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

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.

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 industry's most code-efficient architecture for C and assembly programming.

Microcontroller	ATMEGA2560
Frequency	16MHz
Flash	248KB
RAM	8KB
Vendor	Controllino

Configuration

Please use `controllino_maxi` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[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.

```
[env:controllino_maxi]
platform = atmelavr
board = controllino_maxi

; change microcontroller
```

(continues on next page)

(continued from previous page)

```
board_build.mcu = atmega2560
; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

PIO Unified Debugger currently does not support Controllino Maxi board.

Frameworks

Name	Description
Arduin	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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 industrys most code-efficient architecture for C and assembly programming.

Microcontroller	ATMEGA2560
Frequency	16MHz
Flash	248KB
RAM	8KB
Vendor	Controllino

Configuration

Please use `controllino_maxi_automation` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:controllino_maxi_automation]
platform = atmelavr
board = controllino_maxi_automation
```

You can override default Controllino Maxi Automation settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `controllino_maxi_automation.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:controllino_maxi_automation]
platform = atmelavr
board = controllino_maxi_automation

; change microcontroller
board_build.mcu = atmega2560

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

PIO Unified Debugger currently does not support Controllino Maxi Automation board.

Frameworks

Name	Description
<code>Arduin</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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 industry's most code-efficient architecture for C and assembly programming.

Microcontroller	ATMEGA2560
Frequency	16MHz
Flash	248KB
RAM	8KB
Vendor	Controllino

Configuration

Please use `controllino_mega` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:controllino_mega]
platform = atmelavr
board = controllino_mega
```

You can override default Controllino Mega settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `controllino_mega.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[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

Name	Description
<code>Ardu</code> <code>duino</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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 industry's most code-efficient architecture for C and assembly programming.

Microcontroller	ATMEGA328P
Frequency	16MHz
Flash	31.50KB
RAM	2KB
Vendor	Controllino

Configuration

Please use `controllino_mini` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[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.

```
[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

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.

Digispark Pro

Contents

- *Digispark Pro*
 - *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	ATTINY167
Frequency	16MHz
Flash	14.50KB
RAM	512B
Vendor	Digistump

Configuration

Please use `digispark-pro` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[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.

```
[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

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.

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.

Microcontroller	ATTINY167
Frequency	16MHz
Flash	14.50KB
RAM	512B
Vendor	Digistump

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
```

(continues on next page)

(continued from previous page)

```
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

Name	Description
Arduinio	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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 industrys most code-efficient architecture for C and assembly programming.

Microcontroller	ATTINY167
Frequency	16MHz
Flash	14.50KB
RAM	512B
Vendor	Digistump

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
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

Name	Description
<code>Ar-duino</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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 industry's most code-efficient architecture for C and assembly programming.

Microcontroller	ATTINY85
Frequency	16MHz
Flash	5.87KB
RAM	512B
Vendor	Digistump

Configuration

Please use `digispark-tiny` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:digispark-tiny]
platform = atmelavr
board = digispark-tiny
```

You can override default Digispark USB settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `digispark-tiny.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:digispark-tiny]
platform = atmelavr
board = digispark-tiny

; change microcontroller
board_build.mcu = attiny85

; change MCU frequency
board_build.f_cpu = 16500000L
```

Debugging

PIO Unified Debugger currently does not support Digispark USB board.

Frameworks

Name	Description
<code>Ar-duino</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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 industry's most code-efficient architecture for C and assembly programming.

Microcontroller	AT90USB646
Frequency	16MHz
Flash	60KB
RAM	2KB
Vendor	Dwengo

Configuration

Please use dwenguino ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[env:dwenguino]
platform = atmelavr
board = dwenguino
```

You can override default Dwenguino settings per build environment using *board_**** option, where *** is a JSON object path from board manifest [dwenguino.json](#). For example, *board_build.mcu*, *board_build.f_cpu*, etc.

```
[env:dwenguino]
platform = atmelavr
board = dwenguino

; change microcontroller
board_build.mcu = at90usb646

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

PIO Unified Debugger currently does not support Dwenguino board.

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

Elektor Uno R4

Contents

- *Elektor Uno R4*
 - *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	ATMEGA328PB
Frequency	16MHz
Flash	31.50KB
RAM	2KB
Vendor	Elektor

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

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.

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.

Microcontroller	ATMEGA32U4
Frequency	8MHz
Flash	28KB
RAM	2.50KB
Vendor	Engduino

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
```

(continues on next page)

(continued from previous page)

```
board_build.mcu = atmega32u4
; change MCU frequency
board_build.f_cpu = 8000000L
```

Debugging

PIO Unified Debugger currently does not support Engduino 3 board.

Frameworks

Name	Description
Ardu- duino	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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 industrys most code-efficient architecture for C and assembly programming.

Microcontroller	ATMEGA1284P
Frequency	8MHz
Flash	127KB
RAM	16KB
Vendor	EnviroDIY

Configuration

Please use `mayfly` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[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.

```
[env:mayfly]
platform = atmelavr
board = mayfly

; change microcontroller
board_build.mcu = atmega1284p

; change MCU frequency
board_build.f_cpu = 8000000L
```

Debugging

PIO Unified Debugger currently does not support EnviroDIY Mayfly board.

Frameworks

Name	Description
<code>Ar-duino</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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.

Microcontroller	ATMEGA2560
Frequency	16MHz
Flash	252KB
RAM	8KB
Vendor	FYSETC

Configuration

Please use `fysetc_f6_13` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:fysetc_f6_13]
platform = atmelavr
board = fysetc_f6_13
```

You can override default FYSETC F6 V1.3 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `fysetc_f6_13.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:fysetc_f6_13]
platform = atmelavr
board = fysetc_f6_13

; change microcontroller
board_build.mcu = atmega2560

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

PIO Unified Debugger currently does not support FYSETC F6 V1.3 board.

Frameworks

Name	Description
<code>Ardu</code> <code>ino</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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 industry's most code-efficient architecture for C and assembly programming.

Microcontroller	ATTINY13
Frequency	1MHz
Flash	1KB
RAM	64B
Vendor	Atmel

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

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.

Generic ATtiny13A

Contents

- *Generic ATtiny13A*
 - *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	ATTINY13A
Frequency	1MHz
Flash	1KB
RAM	64B
Vendor	Atmel

Configuration

Please use attiny13a ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[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.

```
[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

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.

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.

Microcontroller	ATTINY1634
Frequency	8MHz
Flash	16KB
RAM	1KB
Vendor	Atmel

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
```

(continues on next page)

(continued from previous page)

```
board_build.mcu = attiny1634
; change MCU frequency
board_build.f_cpu = 8000000L
```

Debugging

PIO Unified Debugger currently does not support Generic ATtiny1634 board.

Frameworks

Name	Description
Arduin	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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 industrys most code-efficient architecture for C and assembly programming.

Microcontroller	ATTINY167
Frequency	8MHz
Flash	16KB
RAM	512B
Vendor	Atmel

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 ATtiny167 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `attiny167.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:attiny167]
platform = atmelavr
board = attiny167

; change microcontroller
board_build.mcu = attiny167

; change MCU frequency
board_build.f_cpu = 8000000L
```

Debugging

PIO Unified Debugger currently does not support Generic ATtiny167 board.

Frameworks

Name	Description
<code>Ar-duino</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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.

Microcontroller	ATTINY2313
Frequency	8MHz
Flash	2KB
RAM	128B
Vendor	Atmel

Configuration

Please use attiny2313 ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:attiny2313]
platform = atmelavr
board = attiny2313
```

You can override default Generic ATtiny2313 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `attiny2313.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:attiny2313]
platform = atmelavr
board = attiny2313

; change microcontroller
board_build.mcu = attiny2313

; change MCU frequency
board_build.f_cpu = 8000000L
```

Debugging

PIO Unified Debugger currently does not support Generic ATtiny2313 board.

Frameworks

Name	Description
<code>Ardu</code> <code>ino</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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 industry's most code-efficient architecture for C and assembly programming.

Microcontroller	ATTINY24
Frequency	8MHz
Flash	2KB
RAM	128B
Vendor	Atmel

Configuration

Please use attiny24 ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[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.

```
[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

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.

Generic ATtiny25

Contents

- Generic ATtiny25
 - 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	ATTINY25
Frequency	8MHz
Flash	2KB
RAM	128B
Vendor	Atmel

Configuration

Please use attiny25 ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:attiny25]
platform = atmelavr
board = attiny25
```

You can override default Generic ATtiny25 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `attiny25.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:attiny25]
platform = atmelavr
board = attiny25

; change microcontroller
board_build.mcu = attiny25

; change MCU frequency
board_build.f_cpu = 8000000L
```

Debugging

`PIO Unified Debugger` currently does not support Generic ATtiny25 board.

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.

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 industry's most code-efficient architecture for C and assembly programming.

Microcontroller	ATTINY261
Frequency	8MHz
Flash	2KB
RAM	128B
Vendor	Atmel

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
```

(continues on next page)

(continued from previous page)

```
board_build.mcu = attiny261
; change MCU frequency
board_build.f_cpu = 8000000L
```

Debugging

PIO Unified Debugger currently does not support Generic ATtiny261 board.

Frameworks

Name	Description
Arduin	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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.

Microcontroller	ATTINY4313
Frequency	8MHz
Flash	4KB
RAM	256B
Vendor	Atmel

Configuration

Please use `attiny4313` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:attiny4313]
platform = atmelavr
board = attiny4313
```

You can override default Generic ATtiny4313 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `attiny4313.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:attiny4313]
platform = atmelavr
board = attiny4313

; change microcontroller
board_build.mcu = attiny4313

; change MCU frequency
board_build.f_cpu = 8000000L
```

Debugging

PIO Unified Debugger currently does not support Generic ATtiny4313 board.

Frameworks

Name	Description
<code>Arduin</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

Generic ATtiny43U

Contents

- *Generic ATtiny43U*
 - *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	ATTINY43U
Frequency	8MHz
Flash	4KB
RAM	256B
Vendor	Atmel

Configuration

Please use attiny43 ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:attiny43]
platform = atmelavr
board = attiny43
```

You can override default Generic ATtiny43U settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `attiny43.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:attiny43]
platform = atmelavr
board = attiny43

; change microcontroller
board_build.mcu = attiny43u

; change MCU frequency
board_build.f_cpu = 8000000L
```

Debugging

PIO Unified Debugger currently does not support Generic ATtiny43U board.

Frameworks

Name	Description
<code>Ar- duino</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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.

Microcontroller	ATTINY44
Frequency	8MHz
Flash	4KB
RAM	256B
Vendor	Atmel

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

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.

Generic ATtiny441

Contents

- *Generic ATtiny441*
 - *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	ATTINY441
Frequency	8MHz
Flash	4KB
RAM	256B
Vendor	Atmel

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

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.

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.

Microcontroller	ATTINY45
Frequency	8MHz
Flash	4KB
RAM	256B
Vendor	Atmel

Configuration

Please use attiny45 ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[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.

```
[env:attiny45]
platform = atmelavr
board = attiny45

; change microcontroller
```

(continues on next page)

(continued from previous page)

```
board_build.mcu = attiny45
; change MCU frequency
board_build.f_cpu = 8000000L
```

Debugging

PIO Unified Debugger currently does not support Generic ATtiny45 board.

Frameworks

Name	Description
Arduin	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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.

Microcontroller	ATTINY461
Frequency	8MHz
Flash	4KB
RAM	256B
Vendor	Atmel

Configuration

Please use `attiny461` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:attiny461]
platform = atmelavr
board = attiny461
```

You can override default Generic ATtiny461 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `attiny461.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:attiny461]
platform = atmelavr
board = attiny461

; change microcontroller
board_build.mcu = attiny461

; change MCU frequency
board_build.f_cpu = 8000000L
```

Debugging

PIO Unified Debugger currently does not support Generic ATtiny461 board.

Frameworks

Name	Description
<code>Ar-duino</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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.

Microcontroller	ATTINY48
Frequency	8MHz
Flash	4KB
RAM	256B
Vendor	Atmel

Configuration

Please use attiny48 ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:attiny48]
platform = atmelavr
board = attiny48
```

You can override default Generic ATtiny48 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `attiny48.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[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

Name	Description
<code>Ar-duino</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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 industry's 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*):

```
[env:attiny828]
platform = atmelavr
board = attiny828
```

You can override default ATtiny828 settings per build environment using *board_**** option, where *** is a JSON object path from board manifest [attiny828.json](#). For example, *board_build.mcu*, *board_build.f_cpu*, etc.

```
[env:attiny828]
platform = atmelavr
board = attiny828

; change microcontroller
board_build.mcu = attiny828

; change MCU frequency
board_build.f_cpu = 8000000L
```

Debugging

PIO Unified Debugger currently does not support Generic ATtiny828 board.

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.

Generic ATtiny84

Contents

- Generic ATtiny84
 - 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	ATTINY84
Frequency	8MHz
Flash	8KB
RAM	512B
Vendor	Atmel

Configuration

Please use attiny84 ID for `board` option in “*platformio.ini*” (*Project Configuration File*):

```
[env:attiny84]
platform = atmelavr
board = attiny84
```

You can override default Generic ATtiny84 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `attiny84.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:attiny84]
platform = atmelavr
board = attiny84

; change microcontroller
board_build.mcu = attiny84

; change MCU frequency
board_build.f_cpu = 8000000L
```

Debugging

PIO Unified Debugger currently does not support Generic ATtiny84 board.

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.

Generic ATtiny841

Contents

- *Generic ATtiny841*
 - *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	ATTINY841
Frequency	8MHz
Flash	8KB
RAM	512B
Vendor	Atmel

Configuration

Please use attiny841 ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[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.

```
[env:attiny841]
platform = atmelavr
board = attiny841

; change microcontroller
```

(continues on next page)

(continued from previous page)

```
board_build.mcu = attiny841
; change MCU frequency
board_build.f_cpu = 8000000L
```

Debugging

PIO Unified Debugger currently does not support Generic ATtiny841 board.

Frameworks

Name	Description
Arduin	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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.

Microcontroller	ATTINY85
Frequency	8MHz
Flash	8KB
RAM	512B
Vendor	Atmel

Configuration

Please use `attiny85` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:attiny85]
platform = atmelavr
board = attiny85
```

You can override default Generic ATtiny85 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `attiny85.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:attiny85]
platform = atmelavr
board = attiny85

; change microcontroller
board_build.mcu = attiny85

; change MCU frequency
board_build.f_cpu = 8000000L
```

Debugging

PIO Unified Debugger currently does not support Generic ATtiny85 board.

Frameworks

Name	Description
<code>Ar-duino</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

Generic ATtiny861

Contents

- *Generic ATtiny861*
 - *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	ATTINY861
Frequency	8MHz
Flash	8KB
RAM	512B
Vendor	Atmel

Configuration

Please use attiny861 ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:attiny861]
platform = atmelavr
board = attiny861
```

You can override default Generic ATtiny861 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `attiny861.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:attiny861]
platform = atmelavr
board = attiny861

; change microcontroller
board_build.mcu = attiny861

; change MCU frequency
board_build.f_cpu = 8000000L
```

Debugging

PIO Unified Debugger currently does not support Generic ATtiny861 board.

Frameworks

Name	Description
<code>Ar- duino</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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

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.

Generic ATtiny88

Contents

- Generic ATtiny88
 - 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	ATTINY88
Frequency	8MHz
Flash	8KB
RAM	512B
Vendor	Atmel

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

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.

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.

Microcontroller	ATMEGA328P
Frequency	8MHz
Flash	31.50KB
RAM	2KB
Vendor	Punch Through

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
```

(continues on next page)

(continued from previous page)

```
board_build.mcu = atmega328p
; change MCU frequency
board_build.f_cpu = 8000000L
```

Debugging

PIO Unified Debugger currently does not support LightBlue Bean board.

Frameworks

Name	Description
Arduin	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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.

Microcontroller	ATMEGA328P
Frequency	16MHz
Flash	31.50KB
RAM	2KB
Vendor	Punch Through

Configuration

Please use `lightblue-beanplus` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[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.

```
[env:lightblue-beanplus]
platform = atmelavr
board = lightblue-beanplus

; change microcontroller
board_build.mcu = atmega328p

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

PIO Unified Debugger currently does not support LightBlue Bean+ board.

Frameworks

Name	Description
<code>Ar-duino</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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 industry's most code-efficient architecture for C and assembly programming.

Microcontroller	ATMEGA32U4
Frequency	8MHz
Flash	28KB
RAM	2.50KB
Vendor	LightUp

Configuration

Please use `lightup` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:lightup]
platform = atmelavr
board = lightup
```

You can override default LightUp settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `lightup.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[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

Name	Description
<code>Ar-duino</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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.

Microcontroller	ATMEGA32U4
Frequency	16MHz
Flash	28KB
RAM	2.50KB
Vendor	Linino

Configuration

Please use one ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[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.

```
[env:one]
platform = atmelavr
board = one

; change microcontroller
board_build.mcu = atmega32u4

; change MCU frequency
board_build.f_cpu = 1600000L
```

Debugging

PIO Unified Debugger currently does not support Linino One board.

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.

LinkIt Smart 7688 Duo

Contents

- *LinkIt Smart 7688 Duo*
 - *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	ATMEGA32U4
Frequency	8MHz
Flash	28KB
RAM	2.50KB
Vendor	MediaTek Labs

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

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.

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.

Microcontroller	ATMEGA32U4
Frequency	8MHz
Flash	28KB
RAM	2.50KB
Vendor	BSFrance

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
```

(continues on next page)

(continued from previous page)

```
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

Name	Description
Arduin	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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 industrys most code-efficient architecture for C and assembly programming.

Microcontroller	ATMEGA328P
Frequency	16MHz
Flash	31KB
RAM	2KB
Vendor	LowPowerLab

Configuration

Please use `mightyhat` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:mightyhat]
platform = atmelavr
board = mightyhat
```

You can override default LowPowerLab MightyHat settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `mightyhat.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:mightyhat]
platform = atmelavr
board = mightyhat

; change microcontroller
board_build.mcu = atmega328p

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

PIO Unified Debugger currently does not support LowPowerLab MightyHat board.

Frameworks

Name	Description
<code>Arduin</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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.

Microcontroller	ATMEGA328P
Frequency	16MHz
Flash	31.50KB
RAM	2KB
Vendor	LowPowerLab

Configuration

Please use moteino ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[env:moteino]
platform = atmelavr
board = moteino
```

You can override default LowPowerLab Moteino settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `moteino.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:moteino]
platform = atmelavr
board = moteino

; change microcontroller
board_build.mcu = atmega328p

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

PIO Unified Debugger currently does not support LowPowerLab Moteino board.

Frameworks

Name	Description
<code>Ar-duino</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

LowPowerLab Moteino (8Mhz)

Contents

- *LowPowerLab Moteino (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.

Microcontroller	ATMEGA328P
Frequency	8MHz
Flash	31.50KB
RAM	2KB
Vendor	LowPowerLab

Configuration

Please use `moteino8mhz` ID for `board` option in “`platformio.ini`” (Project Configuration File):

```
[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.

```
[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

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.

LowPowerLab MoteinoMEGA

Contents

- *LowPowerLab MoteinoMEGA*
 - *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	ATMEGA1284P
Frequency	16MHz
Flash	127KB
RAM	16KB
Vendor	LowPowerLab

Configuration

Please use `moteinomega` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:moteinomega]
platform = atmelavr
board = moteinomega
```

You can override default LowPowerLab MoteinoMEGA settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `moteinomega.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:moteinomega]
platform = atmelavr
board = moteinomega

; change microcontroller
board_build.mcu = atmega1284p

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

PIO Unified Debugger currently does not support LowPowerLab MoteinoMEGA board.

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.

Micoduino Core (Atmega168PA@16M,5V)

Contents

- *Micoduino 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.

Microcontroller	ATMEGA168P
Frequency	16MHz
Flash	15.50KB
RAM	1KB
Vendor	Micoduino

Configuration

Please use `168pa16m` ID for `board` option in “`platformio.ini`” (Project Configuration File):

```
[env:168pa16m]
platform = atmelavr
board = 168pa16m
```

You can override default Micoduino 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
```

(continues on next page)

(continued from previous page)

```
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

Name	Description
Ar-duino	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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 industrys most code-efficient architecture for C and assembly programming.

Microcontroller	ATMEGA168P
Frequency	8MHz
Flash	15.50KB
RAM	1KB
Vendor	Microduino

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
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

Name	Description
<code>Ar-duino</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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.

Microcontroller	ATMEGA328P
Frequency	16MHz
Flash	31.50KB
RAM	2KB
Vendor	Micoduino

Configuration

Please use 328p16m ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[env:328p16m]
platform = atmelavr
board = 328p16m
```

You can override default Microduino Core (Atmega328P@16M,5V) settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `328p16m.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:328p16m]
platform = atmelavr
board = 328p16m

; change microcontroller
board_build.mcu = atmega328p

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

PIO Unified Debugger currently does not support Microduino Core (Atmega328P@16M,5V) board.

Frameworks

Name	Description
<code>Ar-duino</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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.

Microcontroller	ATMEGA328P
Frequency	8MHz
Flash	31.50KB
RAM	2KB
Vendor	Microduino

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

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.

Microduino Core USB (ATmega32U4@16M,5V)

Contents

- Microduino Core USB (ATmega32U4@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.

Microcontroller	ATMEGA32U4
Frequency	16MHz
Flash	28KB
RAM	2.50KB
Vendor	Microduino

Configuration

Please use 32u416m ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[env:32u416m]
platform = atmelavr
board = 32u416m
```

You can override default Microduino Core USB (ATmega32U4@16M,5V) settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `32u416m.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:32u416m]
platform = atmelavr
board = 32u416m

; change microcontroller
board_build.mcu = atmega32u4

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

PIO Unified Debugger currently does not support Microduino Core USB (ATmega32U4@16M,5V) board.

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.

Micoduino Core+ (ATmega1284P@16M,5V)

Contents

- *Micoduino 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 industry's most code-efficient architecture for C and assembly programming.

Microcontroller	ATMEGA1284P
Frequency	16MHz
Flash	127KB
RAM	16KB
Vendor	Micoduino

Configuration

Please use `1284p16m` ID for `board` option in “`platformio.ini`” (Project Configuration File):

```
[env:1284p16m]
platform = atmelavr
board = 1284p16m
```

You can override default Micoduino Core+ (ATmega1284P@16M,5V) settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `1284p16m.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:1284p16m]
platform = atmelavr
board = 1284p16m

; change microcontroller
```

(continues on next page)

(continued from previous page)

```
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

Name	Description
Ar-duino	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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.

Microcontroller	ATMEGA1284P
Frequency	8MHz
Flash	127KB
RAM	16KB
Vendor	Microduino

Configuration

Please use 1284p8m ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[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.

```
[env:1284p8m]
platform = atmelavr
board = 1284p8m

; change microcontroller
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

Name	Description
Ar-duino	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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.

Microcontroller	ATMEGA644P
Frequency	16MHz
Flash	63KB
RAM	4KB
Vendor	Micoduino

Configuration

Please use `644pa16m` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:644pa16m]
platform = atmelavr
board = 644pa16m
```

You can override default Microduino Core+ (`Atmega644PA@16M,5V`) settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `644pa16m.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[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

Name	Description
<code>Ar-duino</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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 industry's most code-efficient architecture for C and assembly programming.

Microcontroller	ATMEGA644P
Frequency	8MHz
Flash	63KB
RAM	4KB
Vendor	Microduino

Configuration

Please use 644pa8m ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[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.

```
[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

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.

OpenEnergyMonitor emonPi

Contents

- *OpenEnergyMonitor emonPi*
 - *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	ATMEGA328P
Frequency	16MHz
Flash	30KB
RAM	2KB
Vendor	OpenEnergyMonitor

Configuration

Please use `emonpi` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[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.

```
[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

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.

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 industry's most code-efficient architecture for C and assembly programming.

Microcontroller	ATMEGA328P
Frequency	8MHz
Flash	31.50KB
RAM	2KB
Vendor	PanStamp

Configuration

Please use `panStampAVR` ID for `board` option in “`platformio.ini`” (Project Configuration File):

```
[env:panStampAVR]
platform = atmelavr
board = panStampAVR
```

You can override default PanStamp AVR settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `panStampAVR.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:panStampAVR]
platform = atmelavr
board = panStampAVR

; change microcontroller
```

(continues on next page)

(continued from previous page)

```
board_build.mcu = atmega328p
; change MCU frequency
board_build.f_cpu = 8000000L
```

Debugging

PIO Unified Debugger currently does not support PanStamp AVR board.

Frameworks

Name	Description
Arduin	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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 industrys most code-efficient architecture for C and assembly programming.

Microcontroller	ATMEGA256RFR2
Frequency	16MHz
Flash	248KB
RAM	32KB
Vendor	Pinoccio

Configuration

Please use `pinoccio` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:pinoccio]
platform = atmelavr
board = pinoccio
```

You can override default Pinoccio Scout settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `pinoccio.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:pinoccio]
platform = atmelavr
board = pinoccio

; change microcontroller
board_build.mcu = atmega256rfr2

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

PIO Unified Debugger currently does not support Pinoccio Scout board.

Frameworks

Name	Description
<code>Ar-duino</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

Pololu A-Star 32U4

Contents

- *Pololu A-Star 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.

Microcontroller	ATMEGA32U4
Frequency	16MHz
Flash	28KB
RAM	2.50KB
Vendor	Pololu Corporation

Configuration

Please use `a-star32U4` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:a-star32U4]
platform = atmelavr
board = a-star32U4
```

You can override default Pololu A-Star 32U4 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `a-star32U4.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:a-star32U4]
platform = atmelavr
board = a-star32U4

; change microcontroller
board_build.mcu = atmega32u4

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

PIO Unified Debugger currently does not support Pololu A-Star 32U4 board.

Frameworks

Name	Description
<code>Ardu</code> <code>ino</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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 industry's most code-efficient architecture for C and assembly programming.

Microcontroller	ATMEGA32U4
Frequency	8MHz
Flash	28KB
RAM	2.50KB
Vendor	Quirkbot

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

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

RedBearLab Blend

Contents

- *RedBearLab Blend*
 - *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	RedBearLab

Configuration

Please use blend ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[env:blend]
platform = atmelavr
board = blend
```

You can override default RedBearLab Blend settings per build environment using *board_**** option, where *** is a JSON object path from board manifest *blend.json*. For example, *board_build.mcu*, *board_build.f_cpu*, etc.

```
[env:blend]
platform = atmelavr
board = blend

; change microcontroller
board_build.mcu = atmega32u4

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

PIO Unified Debugger currently does not support RedBearLab Blend board.

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.

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 industry's most code-efficient architecture for C and assembly programming.

Microcontroller	ATMEGA32U4
Frequency	16MHz
Flash	28KB
RAM	2.50KB
Vendor	RedBearLab

Configuration

Please use `blendmicro16` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:blendmicro16]
platform = atmelavr
board = blendmicro16
```

You can override default RedBearLab Blend Micro 3.3V/16MHz (overclock) settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `blendmicro16.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:blendmicro16]
platform = atmelavr
board = blendmicro16

; change microcontroller
```

(continues on next page)

(continued from previous page)

```
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

Name	Description
Arduin	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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 industrys most code-efficient architecture for C and assembly programming.

Microcontroller	ATMEGA32U4
Frequency	8MHz
Flash	28KB
RAM	2.50KB
Vendor	RedBearLab

Configuration

Please use `blendmicro8` ID for `board` option in “`platformio.ini`” (Project Configuration File):

```
[env:blendmicro8]
platform = atmelavr
board = blendmicro8
```

You can override default RedBearLab Blend Micro 3.3V/8MHz settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `blendmicro8.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:blendmicro8]
platform = atmelavr
board = blendmicro8

; change microcontroller
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

Name	Description
<code>Arduin</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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.

Microcontroller	ATMEGA2560
Frequency	16MHz
Flash	252KB
RAM	8KB
Vendor	RepRap

Configuration

Please use `reprap_rambo` ID for `board` option in “`platformio.ini`” (Project Configuration File):

```
[env:reprap_rambo]
platform = atmelavr
board = reprap_rambo
```

You can override default RepRap RAMBo settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `reprap_rambo.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:reprap_rambo]
platform = atmelavr
board = reprap_rambo

; 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

Name	Description
<code>Ardu</code> <code>ino</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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 industry's most code-efficient architecture for C and assembly programming.

Microcontroller	ATMEGA1284P
Frequency	8MHz
Flash	127KB
RAM	16KB
Vendor	SODAQ

Configuration

Please use `sodaq_galora` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[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.

```
[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

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.

SODAQ Mbili

Contents

- *SODAQ Mbili*
 - *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	ATMEGA1284P
Frequency	8MHz
Flash	127KB
RAM	16KB
Vendor	SODAQ

Configuration

Please use `sodaq_mbili` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:sodaq_mbili]
platform = atmelavr
board = sodaq_mbili
```

You can override default SODAQ Mbili settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `sodaq_mbili.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:sodaq_mbili]
platform = atmelavr
board = sodaq_mbili

; change microcontroller
board_build.mcu = atmega1284p

; change MCU frequency
board_build.f_cpu = 8000000L
```

Debugging

PIO Unified Debugger currently does not support SODAQ Mbili board.

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.

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.

Microcontroller	ATMEGA328P
Frequency	8MHz
Flash	31.50KB
RAM	2KB
Vendor	SODAQ

Configuration

Please use `sodaq_moja` ID for `board` option in “`platformio.ini`” (Project Configuration File):

```
[env:sodaq_moja]
platform = atmelavr
board = sodaq_moja
```

You can override default SODAQ Moja settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `sodaq_moja.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:sodaq_moja]
platform = atmelavr
board = sodaq_moja

; change microcontroller
```

(continues on next page)

(continued from previous page)

```
board_build.mcu = atmega328p
; change MCU frequency
board_build.f_cpu = 8000000L
```

Debugging

PIO Unified Debugger currently does not support SODAQ Moja board.

Frameworks

Name	Description
Arduin	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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 industrys most code-efficient architecture for C and assembly programming.

Microcontroller	ATMEGA1284P
Frequency	8MHz
Flash	127KB
RAM	16KB
Vendor	SODAQ

Configuration

Please use `sodaq_ndogo` ID for `board` option in “`platformio.ini`” (Project Configuration File):

```
[env:sodaq_ndogo]
platform = atmelavr
board = sodaq_ndogo
```

You can override default SODAQ Ndogo settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `sodaq_ndogo.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:sodaq_ndogo]
platform = atmelavr
board = sodaq_ndogo

; change microcontroller
board_build.mcu = atmega1284p

; change MCU frequency
board_build.f_cpu = 8000000L
```

Debugging

PIO Unified Debugger currently does not support SODAQ Ndogo board.

Frameworks

Name	Description
<code>Ar-duino</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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.

Microcontroller	ATMEGA1284P
Frequency	8MHz
Flash	127KB
RAM	16KB
Vendor	SODAQ

Configuration

Please use `sodaq_tatu` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:sodaq_tatu]
platform = atmelavr
board = sodaq_tatu
```

You can override default SODAQ Tatu settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `sodaq_tatu.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:sodaq_tatu]
platform = atmelavr
board = sodaq_tatu

; change microcontroller
board_build.mcu = atmega1284p

; change MCU frequency
board_build.f_cpu = 8000000L
```

Debugging

PIO Unified Debugger currently does not support SODAQ Tatu board.

Frameworks

Name	Description
<code>Arduin</code>	Arduino Wiring-based Framework allows writing cross-platform software 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 industry's most code-efficient architecture for C and assembly programming.

Microcontroller	ATMEGA1284P
Frequency	16MHz
Flash	127KB
RAM	16KB
Vendor	Sanguino

Configuration

Please use `sanguino_atmega1284p` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:sanguino_atmega1284p]
platform = atmelavr
board = sanguino_atmega1284p
```

You can override default Sanguino ATmega1284p (16MHz) settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `sanguino_atmega1284p.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:sanguino_atmega1284p]
platform = atmelavr
board = sanguino_atmega1284p

; change microcontroller
board_build.mcu = atmega1284p

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

PIO Unified Debugger currently does not support Sanguino ATmega1284p (16MHz) board.

Frameworks

Name	Description
Arduino	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

Sanguino ATmega1284p (8MHz)

Contents

- Sanguino ATmega1284p (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.

Microcontroller	ATMEGA1284P
Frequency	8MHz
Flash	127KB
RAM	16KB
Vendor	Sanguino

Configuration

Please use `sanguino_atmega1284_8m` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:sanguino_atmega1284_8m]
platform = atmelavr
board = sanguino_atmega1284_8m
```

You can override default Sanguino ATmega1284p (8MHz) settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `sanguino_atmega1284_8m.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:sanguino_atmega1284_8m]
platform = atmelavr
board = sanguino_atmega1284_8m

; change microcontroller
board_build.mcu = atmega1284p

; change MCU frequency
board_build.f_cpu = 8000000L
```

Debugging

PIO Unified Debugger currently does not support Sanguino ATmega1284p (8MHz) board.

Frameworks

Name	Description
Arduino	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

Sanguino ATmega644 or ATmega644A (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.

Microcontroller	ATMEGA644
Frequency	16MHz
Flash	63KB
RAM	4KB
Vendor	Sanguino

Configuration

Please use `sanguino_atmega644` ID for `board` option in “`platformio.ini`” (Project Configuration File):

```
[env:sanguino_atmega644]
platform = atmelavr
board = sanguino_atmega644
```

You can override default Sanguino ATmega644 or ATmega644A (16 MHz) settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `sanguino_atmega644.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:sanguino_atmega644]
platform = atmelavr
board = sanguino_atmega644

; change microcontroller
```

(continues on next page)

(continued from previous page)

```
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

Name	Description
Ardu- duino	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

Sanguino ATmega644 or ATmega644A (8 MHz)

Contents

- [Sanguino ATmega644 or ATmega644A \(8 MHz\)](#)
 - [Hardware](#)
 - [Configuration](#)
 - [Debugging](#)
 - [Frameworks](#)

Hardware

Platform [Atmel AVR](#): Atmel AVR 8- 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	ATMEGA644
Frequency	8MHz
Flash	63KB
RAM	4KB
Vendor	Sanguino

Configuration

Please use `sanguino_atmega644_8m` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[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.

```
[env:sanguino_atmega644_8m]
platform = atmelavr
board = sanguino_atmega644_8m

; change microcontroller
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

Name	Description
<code>Ar-duino</code>	Arduino Wiring-based Framework allows writing cross-platform software 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 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.

Microcontroller	ATMEGA644P
Frequency	16MHz
Flash	63KB
RAM	4KB
Vendor	Sanguino

Configuration

Please use `sanguino_atmega644p` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:sanguino_atmega644p]
platform = atmelavr
board = sanguino_atmega644p
```

You can override default Sanguino ATmega644P or ATmega644PA (16 MHz) settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `sanguino_atmega644p.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:sanguino_atmega644p]
platform = atmelavr
board = sanguino_atmega644p

; change microcontroller
board_build.mcu = atmega644p

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

PIO Unified Debugger currently does not support Sanguino ATmega644P or ATmega644PA (16 MHz) board.

Frameworks

Name	Description
<code>Ar-duino</code>	Arduino Wiring-based Framework allows writing cross-platform software 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 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.

Microcontroller	ATMEGA644P
Frequency	8MHz
Flash	63KB
RAM	4KB
Vendor	Sanguino

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

Name	Description
Arduino	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

Seeeduino

Contents

- *Seeeduino*
 - *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	ATMEGA328P
Frequency	16MHz
Flash	31.50KB
RAM	2KB
Vendor	SeeedStudio

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

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.
Simba	Simba is an RTOS and build framework. It aims to make embedded programming easy and portable.

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.

Microcontroller	ATMEGA128RFA1
Frequency	16MHz
Flash	16KB
RAM	124KB
Vendor	SparkFun

Configuration

Please use `sparkfun_satmega128rfa1` ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[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.

```
[env:sparkfun_satmega128rfa1]
platform = atmelavr
board = sparkfun_satmega128rfa1
```

(continues on next page)

(continued from previous page)

```
; 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

Name	Description
Arduin	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

SparkFun 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 industry's most code-efficient architecture for C and assembly programming.

Microcontroller	ATMEGA328P
Frequency	8MHz
Flash	31.50KB
RAM	2KB
Vendor	SparkFun

Configuration

Please use `sparkfun_digital sandbox` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:sparkfun_digital sandbox]
platform = atmelavr
board = sparkfun_digital sandbox
```

You can override default SparkFun Digital Sandbox settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `sparkfun_digital sandbox.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:sparkfun_digital sandbox]
platform = atmelavr
board = sparkfun_digital sandbox

; change microcontroller
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

Name	Description
<code>Ar duino</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

SparkFun 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 industry's most code-efficient architecture for C and assembly programming.

Microcontroller	ATMEGA32U4
Frequency	8MHz
Flash	28KB
RAM	2.50KB
Vendor	SparkFun

Configuration

Please use `sparkfun_fiov3` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:sparkfun_fiov3]
platform = atmelavr
board = sparkfun_fiov3
```

You can override default SparkFun Fio V3 3.3V/8MHz settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `sparkfun_fiov3.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[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

Name	Description
<code>Ardu</code> <code>ino</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

SparkFun 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 industry's 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*):

```
[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.

```
[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 board.

Frameworks

Name	Description
Arduino	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

SparkFun Mega Pro 3.3V/8MHz

Contents

- *SparkFun Mega Pro 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.

Microcontroller	ATMEGA2560
Frequency	8MHz
Flash	252KB
RAM	8KB
Vendor	SparkFun

Configuration

Please use `sparkfun_megapro8MHz` ID for `board` option in “`platformio.ini`” (Project Configuration File):

```
[env:sparkfun_megapro8MHz]
platform = atmelavr
board = sparkfun_megapro8MHz
```

You can override default SparkFun Mega Pro 3.3V/8MHz settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `sparkfun_megapro8MHz.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:sparkfun_megapro8MHz]
platform = atmelavr
board = sparkfun_megapro8MHz

; change microcontroller
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

Name	Description
Arduino	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

SparkFun Mega Pro 5V/16MHz

Contents

- *SparkFun Mega Pro 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.

Microcontroller	ATMEGA2560
Frequency	16MHz
Flash	248KB
RAM	8KB
Vendor	SparkFun

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
```

(continues on next page)

(continued from previous page)

```
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

Name	Description
Arduin	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

SparkFun 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.

Microcontroller	ATMEGA2560
Frequency	8MHz
Flash	252KB
RAM	8KB
Vendor	SparkFun

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
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

Name	Description
<code>Arduin</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

SparkFun 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.

Microcontroller	ATMEGA328P
Frequency	16MHz
Flash	31.50KB
RAM	2KB
Vendor	SparkFun

Configuration

Please use `uvview` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:uvview]
platform = atmelavr
board = uvview
```

You can override default SparkFun MicroView settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `uvview.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:uvview]
platform = atmelavr
board = uvview

; 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

Name	Description
<code>Ardu</code> <code>ino</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

SparkFun Pro Micro 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.

Microcontroller	ATMEGA32U4
Frequency	8MHz
Flash	28KB
RAM	2.50KB
Vendor	SparkFun

Configuration

Please use `sparkfun_promicro8` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:sparkfun_promicro8]
platform = atmelavr
board = sparkfun_promicro8
```

You can override default SparkFun Pro Micro 3.3V/8MHz settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `sparkfun_promicro8.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:sparkfun_promicro8]
platform = atmelavr
board = sparkfun_promicro8

; change microcontroller
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

Name	Description
Arduino	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

SparkFun Pro Micro 5V/16MHz

Contents

- *SparkFun Pro Micro 5V/16MHz*
 - *Hardware*
 - *Configuration*
 - *Debugging*
 - *Frameworks*

Hardware

Platform [Atmel AVR](#): Atmel AVR 8- 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_promicro16` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[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.

```
[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

Name	Description
Arduino	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

SparkFun 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.

Microcontroller	ATMEGA32U4
Frequency	8MHz
Flash	28KB
RAM	2.50KB
Vendor	SparkFun

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
```

(continues on next page)

(continued from previous page)

```
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

Name	Description
Arduin	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

SparkFun 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 industrys most code-efficient architecture for C and assembly programming.

Microcontroller	ATMEGA328P
Frequency	16MHz
Flash	31.50KB
RAM	2KB
Vendor	SparkFun

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
board_build.mcu = atmega328p

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

PIO Unified Debugger currently does not support SparkFun RedBoard board.

Frameworks

Name	Description
<code>Arduin</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

SparkFun Serial 7-Segment Display

Contents

- *SparkFun Serial 7-Segment Display*
 - *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	ATMEGA328P
Frequency	8MHz
Flash	31.50KB
RAM	2KB
Vendor	SparkFun

Configuration

Please use `sparkfun_serial7seg` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:sparkfun_serial7seg]
platform = atmelavr
board = sparkfun_serial7seg
```

You can override default SparkFun Serial 7-Segment Display settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `sparkfun_serial7seg.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:sparkfun_serial7seg]
platform = atmelavr
board = sparkfun_serial7seg

; change microcontroller
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

Name	Description
<code>Ardu</code> <code>duino</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

SpellFoundry Sleepy Pi 2

Contents

- *SpellFoundry Sleepy Pi 2*
 - *Hardware*
 - *Configuration*

- *Debugging*
- *Frameworks*

Hardware

Platform [Atmel AVR](#): Atmel AVR 8- 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	ATMEGA328P
Frequency	8MHz
Flash	30KB
RAM	2KB
Vendor	SpellFoundry

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

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.

Talk2 Whisper Node

Contents

- *Talk2 Whisper Node*
 - *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	ATMEGA328P
Frequency	16MHz
Flash	31.50KB
RAM	2KB
Vendor	Wisen

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

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.

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 industry's most code-efficient architecture for C and assembly programming.

Microcontroller	ATMEGA32U4
Frequency	16MHz
Flash	28KB
RAM	2.50KB
Vendor	The Things Network

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
```

(continues on next page)

(continued from previous page)

```
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

Name	Description
Ardu- duino	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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.

Microcontroller	ATMEGA328P
Frequency	8MHz
Flash	30KB
RAM	2KB
Vendor	TinyCircuits

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
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

Name	Description
<code>Ar-duino</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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 industry's most code-efficient architecture for C and assembly programming.

Microcontroller	ATMEGA328P
Frequency	8MHz
Flash	30KB
RAM	2KB
Vendor	TinyCircuits

Configuration

Please use `tinylily` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:tinylily]
platform = atmelavr
board = tinylily
```

You can override default TinyCircuits TinyLily Mini Processor settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `tinylily.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[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

Name	Description
<code>Arduin</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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 industry's most code-efficient architecture for C and assembly programming.

Microcontroller	ATMEGA8
Frequency	12MHz
Flash	8KB
RAM	1KB
Vendor	Atmel

Configuration

Please use `usbasp` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[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.

```
[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

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.

Wicked Device WildFire V2

Contents

- *Wicked Device WildFire V2*
 - *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	ATMEGA1284P
Frequency	16MHz
Flash	120.00KB
RAM	16KB
Vendor	Wicked Device

Configuration

Please use `wildfirev2` ID for `board` option in “`platformio.ini`” (Project Configuration File):

```
[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 = atmega1284p

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

`PIO Unified Debugger` currently does not support Wicked Device WildFire V2 board.

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.

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.

Microcontroller	ATMEGA1284P
Frequency	16MHz
Flash	127KB
RAM	16KB
Vendor	Wicked Device

Configuration

Please use `wildfirev3` ID for `board` option in “`platformio.ini`” (Project Configuration File):

```
[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.

```
[env:wildfirev3]
platform = atmelavr
board = wildfirev3

; change microcontroller
```

(continues on next page)

(continued from previous page)

```
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

Name	Description
<i>Ar-duino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

ftDuino

Contents

- *ftDuino*
 - *Hardware*
 - *Configuration*
 - *Debugging*
 - *Frameworks*

Hardware

Platform *Atmel AVR*: Atmel AVR 8- 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	ATMEGA32U4
Frequency	16MHz
Flash	28KB
RAM	2.50KB
Vendor	Till Harbaum

Configuration

Please use `ftduino` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:ftduino]
platform = atmelavr
board = ftduino
```

You can override default ftDuino settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `ftduino.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:ftduino]
platform = atmelavr
board = ftduino

; change microcontroller
board_build.mcu = atmega32u4

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

PIO Unified Debugger currently does not support ftDuino board.

Frameworks

Name	Description
Arduino	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

nicai-systems 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.

Microcontroller	ATMEGA88
Frequency	8MHz
Flash	8KB
RAM	1KB
Vendor	nicai-systems

Configuration

Please use `bob3` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:bob3]
platform = atmelavr
board = bob3
```

You can override default nicai-systems BOB3 coding bot settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `bob3.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:bob3]
platform = atmelavr
board = bob3

; change microcontroller
board_build.mcu = atmega88

; change MCU frequency
board_build.f_cpu = 8000000L
```

Debugging

PIO Unified Debugger currently does not support nicai-systems BOB3 coding bot board.

Frameworks

Name	Description
Arduin	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

nicai-systems NIBO 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.

Microcontroller	ATMEGA128
Frequency	16MHz
Flash	128KB
RAM	4KB
Vendor	nicai-systems

Configuration

Please use nibo2 ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[env:nibo2]
platform = atmelavr
board = nibo2
```

You can override default nicai-systems NIBO 2 robot settings per build environment using *board_**** option, where *** is a JSON object path from board manifest [nibo2.json](#). For example, *board_build.mcu*, *board_build.f_cpu*, etc.

```
[env:nibo2]
platform = atmelavr
board = nibo2

; change microcontroller
board_build.mcu = atmega128

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

PIO Unified Debugger currently does not support nicai-systems NIBO 2 robot board.

Frameworks

Name	Description
Arduino	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

nicai-systems NIBO burger robot

Contents

- *nicai-systems NIBO burger 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 industrys most code-efficient architecture for C and assembly programming.

Microcontroller	ATMEGA16
Frequency	15MHz
Flash	16KB
RAM	1KB
Vendor	nicai-systems

Configuration

Please use `niboburger` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:niboburger]
platform = atmelavr
board = niboburger
```

You can override default nicai-systems NIBO burger robot settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `niboburger.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:niboburger]
platform = atmelavr
board = niboburger

; change microcontroller
board_build.mcu = atmega16

; change MCU frequency
board_build.f_cpu = 15000000L
```

Debugging

PIO Unified Debugger currently does not support nicai-systems NIBO burger robot board.

Frameworks

Name	Description
Arduino	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

nicai-systems NIBO burger robot 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 industry's most code-efficient architecture for C and assembly programming.

Microcontroller	ATMEGA1284P
Frequency	20MHz
Flash	128KB
RAM	16KB
Vendor	nicai-systems

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
```

(continues on next page)

(continued from previous page)

```
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

Name	Description
Arduin	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

nicai-systems 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 industrys most code-efficient architecture for C and assembly programming.

Microcontroller	ATMEGA16
Frequency	15MHz
Flash	16KB
RAM	1KB
Vendor	nicai-systems

Configuration

Please use `nibobee` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:nibobee]
platform = atmelavr
board = nibobee
```

You can override default nicai-systems NIBObee robot settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `nibobee.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:nibobee]
platform = atmelavr
board = nibobee

; change microcontroller
board_build.mcu = atmega16

; change MCU frequency
board_build.f_cpu = 15000000L
```

Debugging

PIO Unified Debugger currently does not support nicai-systems NIBObee robot board.

Frameworks

Name	Description
<code>Ar-duino</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

nicai-systems 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 industry's most code-efficient architecture for C and assembly programming.

Microcontroller	ATMEGA1284P
Frequency	20MHz
Flash	128KB
RAM	16KB
Vendor	nicai-systems

Configuration

Please use nibobee_1284 ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[env:nibobee_1284]
platform = atmelavr
board = nibobee_1284
```

You can override default nicai-systems NIBObee robot with Tuning Kit settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `nibobee_1284.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:nibobee_1284]
platform = atmelavr
board = nibobee_1284

; change microcontroller
board_build.mcu = atmega1284p

; change MCU frequency
board_build.f_cpu = 20000000L
```

Debugging

PIO Unified Debugger currently does not support nicai-systems NIBObee robot with Tuning Kit board.

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.

ubIQio Ardhat

Contents

- [*ubIQio Ardhat*](#)
 - [*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	ATMEGA328P
Frequency	16MHz
Flash	31.50KB
RAM	2KB
Vendor	ubIQio

Configuration

Please use ardhat ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[env:ardhat]
platform = atmelavr
board = ardhat
```

You can override default ubIQio Ardhat settings per build environment using *board_**** option, where *** is a JSON object path from board manifest *ardhat.json*. For example, *board_build.mcu*, *board_build.f_cpu*, etc.

```
[env:ardhat]
platform = atmelavr
board = ardhat

; change microcontroller
board_build.mcu = atmega328p

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

PIO Unified Debugger currently does not support ubIQio Ardhat board.

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

1.12.3 Atmel SAM

Adafruit Circuit Playground Express

Contents

- Adafruit Circuit Playground Express
 - Hardware
 - Configuration
 - Uploading
 - Debugging
 - Frameworks

Hardware

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

Microcontroller	SAMD21G18A
Frequency	48MHz
Flash	256KB
RAM	32KB
Vendor	Adafruit

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:

```
[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.

Compatible Tools	On-board	Default
<i>Atmel-ICE</i>		Yes
<i>Black Magic Probe</i>		
<i>J-LINK</i>		

Frameworks

Name	Description
<i>Ar-duino</i>	Arduino Wiring-based Framework allows writing cross-platform software 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 Crickit M0

Contents

- Adafruit Crickit M0
 - Hardware
 - Configuration
 - Uploading
 - Debugging
 - Frameworks

Hardware

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

Microcontroller	SAMD21G18A
Frequency	48MHz
Flash	256KB
RAM	32KB
Vendor	Adafruit

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.

Compatible Tools	On-board	Default
Atmel-ICE		Yes
Black Magic Probe		
J-LINK		

Frameworks

Name	Description
Arduino	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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.

Microcontroller	SAMD21G18A
Frequency	48MHz
Flash	256KB
RAM	32KB
Vendor	Adafruit

Configuration

Please use `adafruit_feather_m0` ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[env:adafruit_feather_m0]
platform = atmelsam
board = adafruit_feather_m0
```

You can override default Adafruit Feather M0 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `adafruit_feather_m0.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:adafruit_feather_m0]
platform = atmelsam
board = adafruit_feather_m0

; 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:

```
[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 external probe listed below.

Compatible Tools	On-board	Default
<i>Atmel-ICE</i>		Yes
<i>Black Magic Probe</i>		
<i>J-LINK</i>		

Frameworks

Name	Description
<i>Ardu- ino</i>	Arduino Wiring-based Framework allows writing cross-platform software 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 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.

Microcontroller	SAMD21G18A
Frequency	48MHz
Flash	256KB
RAM	32KB
Vendor	Adafruit

Configuration

Please use `adafruit_feather_m0_express` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:adafruit_feather_m0_express]
platform = atmelsam
board = adafruit_feather_m0_express
```

You can override default Adafruit Feather M0 Express settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `adafruit_feather_m0_express.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:adafruit_feather_m0_express]
platform = atmelsam
board = adafruit_feather_m0_express

; change microcontroller
board_build.mcu = samd21g18a

; change MCU frequency
board_build.f_cpu = 48000000L
```

Uploading

Adafruit Feather M0 Express supports the next uploading protocols:

- atmel-ice
- blackmagic
- jlink
- sam-ba

Default protocol is `sam-ba`

You can change upload protocol using `upload_protocol` option:

```
[env:adafruit_feather_m0_express]
platform = atmelsam
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.

Compatible Tools	On-board	Default
<i>Atmel-ICE</i>		Yes
<i>Black Magic Probe</i>		
<i>J-LINK</i>		

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software 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 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.

Microcontroller	SAMD51J19A
Frequency	120MHz
Flash	512KB
RAM	192KB
Vendor	Adafruit

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

Name	Description
Ardu- duino	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

Adafruit Gemma M0

Contents

- *Adafruit Gemma M0*
 - *Hardware*
 - *Configuration*
 - *Uploading*
 - *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 = atmelsam
board = adafruit_gemma_m0
```

You can override default Adafruit Gemma M0 settings per build environment using *board_**** option, where *** is a JSON object path from board manifest *adafruit_gemma_m0.json*. For example, *board_build.mcu*, *board_build.f_cpu*, etc.

```
[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.

Compatible Tools	On-board	Default
<i>Atmel-ICE</i>		Yes
<i>Black Magic Probe</i>		
<i>J-LINK</i>		

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software 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 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.

Microcontroller	SAMD51P20A
Frequency	120MHz
Flash	1MB
RAM	256KB
Vendor	Adafruit

Configuration

Please use `adafruit_grandcentral_m4` ID for `board` option in “`platformio.ini`” (Project Configuration File):

```
[env:adafruit_grandcentral_m4]
platform = atmelsam
board = adafruit_grandcentral_m4
```

You can override default Adafruit Grand Central M4 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `adafruit_grandcentral_m4.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:adafruit_grandcentral_m4]
platform = atmelsam
board = adafruit_grandcentral_m4

; change microcontroller
board_build.mcu = samd51p20a

; change MCU frequency
board_build.f_cpu = 120000000L
```

Uploading

Adafruit Grand Central M4 supports the next uploading protocols:

- atmel-ice
- jlink
- sam-ba

Default protocol is sam-ba

You can change upload protocol using `upload_protocol` option:

```
[env:adafruit_grandcentral_m4]
platform = atmelsam
board = adafruit_grandcentral_m4

upload_protocol = sam-ba
```

Debugging

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

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for 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.

Compatible Tools	On-board	Default
Atmel-ICE		Yes
J-LINK		

Frameworks

Name	Description
Arduino	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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.

Microcontroller	SAMD21G18A
Frequency	48MHz
Flash	256KB
RAM	32KB
Vendor	Adafruit

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
```

(continues on next page)

(continued from previous page)

```
; 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:

```
[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	On-board	Default
<i>Atmel-ICE</i>		Yes
<i>Black Magic Probe</i>		
<i>J-LINK</i>		

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 Hallowing M4

Contents

- [Adafruit Hallowing M4](#)
 - [Hardware](#)
 - [Configuration](#)
 - [Uploading](#)
 - [Debugging](#)
 - [Frameworks](#)

Hardware

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

Microcontroller	SAMD51J19A
Frequency	120MHz
Flash	496KB
RAM	192KB
Vendor	Adafruit

Configuration

Please use `adafruit_hallowing_m4` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:adafruit_hallowing_m4]
platform = atmelsam
board = adafruit_hallowing_m4
```

You can override default Adafruit Hallowing M4 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `adafruit_hallowing_m4.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:adafruit_hallowing_m4]
platform = atmelsam
board = adafruit_hallowing_m4
```

(continues on next page)

(continued from previous page)

```
; change microcontroller
board_build.mcu = samd51j19a

; change MCU frequency
board_build.f_cpu = 120000000L
```

Uploading

Adafruit Hallowing M4 supports the next uploading protocols:

- atmel-ice
- jlink
- sam-ba

Default protocol is sam-ba

You can change upload protocol using *upload_protocol* option:

```
[env:adafruit_hallowing_m4]
platform = atmelsam
board = adafruit_hallowing_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 Hallowing M4 does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

Compatible Tools	On-board	Default
Atmel-ICE		Yes
J-LINK		

Frameworks

Name	Description
Arduino	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

Adafruit ItsyBitsy M0

Contents

- Adafruit ItsyBitsy M0
 - Hardware
 - Configuration
 - Uploading
 - Debugging
 - Frameworks

Hardware

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

Microcontroller	SAMD21G18A
Frequency	48MHz
Flash	256KB
RAM	32KB
Vendor	Adafruit

Configuration

Please use `adafruit_itsybitsy_m0` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:adafruit_itsybitsy_m0]
platform = atmelsam
board = adafruit_itsybitsy_m0
```

You can override default Adafruit ItsyBitsy M0 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `adafruit_itsybitsy_m0.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:adafruit_itsybitsy_m0]
platform = atmelsam
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:

```
[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.

Compatible Tools	On-board	Default
<i>Atmel-ICE</i>		Yes
<i>Black Magic Probe</i>		
<i>J-LINK</i>		

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software 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.

Microcontroller	SAMD51G19A
Frequency	120MHz
Flash	512KB
RAM	192KB
Vendor	Adafruit

Configuration

Please use `adafruit_itsybitsy_m4` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:adafruit_itsybitsy_m4]
platform = atmelsam
board = adafruit_itsybitsy_m4
```

You can override default Adafruit ItsyBitsy M4 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `adafruit_itsybitsy_m4.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[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:

```
[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.

Compatible Tools	On-board	Default
<i>Atmel-ICE</i>		Yes
<i>J-LINK</i>		

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software 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 MONSTER M4SK

Contents

- Adafruit MONSTER M4SK
 - *Hardware*
 - *Configuration*
 - *Uploading*
 - *Debugging*
 - *Frameworks*

Hardware

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

Microcontroller	SAMD51J19A
Frequency	120MHz
Flash	496KB
RAM	192KB
Vendor	Adafruit

Configuration

Please use `adafruit_monster_m4sk` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:adafruit_monster_m4sk]
platform = atmelsam
board = adafruit_monster_m4sk
```

You can override default Adafruit MONSTER M4SK settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `adafruit_monster_m4sk.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:adafruit_monster_m4sk]
platform = atmelsam
board = adafruit_monster_m4sk

; change microcontroller
board_build.mcu = samd51j19a

; change MCU frequency
board_build.f_cpu = 120000000L
```

Uploading

Adafruit MONSTER M4SK supports the next uploading protocols:

- `atmel-ice`
- `jlink`
- `sam-ba`

Default protocol is `sam-ba`

You can change upload protocol using `upload_protocol` option:

```
[env:adafruit_monster_m4sk]
platform = atmelsam
board = adafruit_monster_m4sk

upload_protocol = sam-ba
```

Debugging

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

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using *debug_tool* option in “*platformio.ini*” (*Project Configuration File*).

Adafruit MONSTER M4SK does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

Compatible Tools	On-board	Default
Atmel-ICE		Yes
J-LINK		

Frameworks

Name	Description
Arduino	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

Adafruit Metro M0 Express

Contents

- *Adafruit Metro 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.

Microcontroller	SAMD21G18A
Frequency	48MHz
Flash	256KB
RAM	32KB
Vendor	Adafruit

Configuration

Please use `adafruit_metro_m0` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[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.

```
[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:

```
[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.

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using *debug_tool* option in “*platformio.ini*” (*Project Configuration File*).

Adafruit Metro M0 Expresss does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

Compatible Tools	On-board	Default
<i>Atmel-ICE</i>		Yes
<i>Black Magic Probe</i>		
<i>J-LINK</i>		

Frameworks

Name	Description
<i>Ar-duino</i>	Arduino Wiring-based Framework allows writing cross-platform software 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 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.

Microcontroller	SAMD51J19A
Frequency	120MHz
Flash	512KB
RAM	192KB
Vendor	Adafruit

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	On-board	Default
Atmel-ICE		Yes
J-LINK		

Frameworks

Name	Description
Arduino	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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.

Microcontroller	SAMD51J19A
Frequency	120MHz
Flash	512KB
RAM	192KB
Vendor	Adafruit

Configuration

Please use `adafruit_metro_m4_airliftlite` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:adafruit_metro_m4_airliftlite]
platform = atmelsam
board = adafruit_metro_m4_airliftlite
```

You can override default Adafruit Metro M4 AirLift Lite settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `adafruit_metro_m4_airliftlite.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:adafruit_metro_m4_airliftlite]
platform = atmelsam
board = adafruit_metro_m4_airliftlite

; change microcontroller
board_build.mcu = samd51j19a

; change MCU frequency
board_build.f_cpu = 120000000L
```

Uploading

Adafruit Metro M4 AirLift Lite supports the next uploading protocols:

- atmel-ice
- jlink
- sam-ba

Default protocol is sam-ba

You can change upload protocol using `upload_protocol` option:

```
[env:adafruit_metro_m4_airliftlite]
platform = atmelsam
board = adafruit_metro_m4_airliftlite

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.

Compatible Tools	On-board	Default
Atmel-ICE		Yes
J-LINK		

Frameworks

Name	Description
Arduino	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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.

Microcontroller	SAMD51J20A
Frequency	120MHz
Flash	1MB
RAM	256KB
Vendor	Adafruit

Configuration

Please use `adafruit_pygamer_advance_m4` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[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.

```
[env:adafruit_pygamer_advance_m4]
platform = atmelsam
board = adafruit_pygamer_advance_m4
```

(continues on next page)

(continued from previous page)

```
; change microcontroller
board_build.mcu = samd51j20a

; change MCU frequency
board_build.f_cpu = 120000000L
```

Uploading

Adafruit PyGamer Advance M4 supports the next uploading protocols:

- atmel-ice
- jlink
- sam-ba

Default protocol is sam-ba

You can change upload protocol using *upload_protocol* option:

```
[env:adafruit_pygamer_advance_m4]
platform = atmelsam
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.

Compatible Tools	On-board	Default
Atmel-ICE		Yes
J-LINK		

Frameworks

Name	Description
Arduino	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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.

Microcontroller	SAMD51J19A
Frequency	120MHz
Flash	512KB
RAM	192KB
Vendor	Adafruit

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 = 12000000L
```

Uploading

Adafruit PyGamer M4 Express supports the next uploading protocols:

- atmel-ice
- jlink
- sam-ba

Default protocol is sam-ba

You can change upload protocol using *upload_protocol* option:

```
[env:adafruit_pygamer_m4]
platform = atmelsam
board = adafruit_pygamer_m4

upload_protocol = sam-ba
```

Debugging

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

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using *debug_tool* option in “*platformio.ini*” (*Project Configuration File*).

Adafruit PyGamer M4 Express does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

Compatible Tools	On-board	Default
Atmel-ICE		Yes
J-LINK		

Frameworks

Name	Description
Arduin	Arduino Wiring-based Framework allows writing cross-platform software 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 PyPortal M4

Contents

- *Adafruit PyPortal M4*
 - *Hardware*
 - *Configuration*
 - *Uploading*

- *Debugging*
- *Frameworks*

Hardware

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

Microcontroller	SAMD51J20A
Frequency	120MHz
Flash	1MB
RAM	256KB
Vendor	Adafruit

Configuration

Please use `adafruit_pyportal_m4` ID for `board` option in “*platformio.ini*” (*Project Configuration File*):

```
[env:adafruit_pyportal_m4]
platform = atmelsam
board = adafruit_pyportal_m4
```

You can override default Adafruit PyPortal M4 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `adafruit_pyportal_m4.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:adafruit_pyportal_m4]
platform = atmelsam
board = adafruit_pyportal_m4

; change microcontroller
board_build.mcu = samd51j20a

; change MCU frequency
board_build.f_cpu = 120000000L
```

Uploading

Adafruit PyPortal M4 supports the next uploading protocols:

- `atmel-ice`
- `jlink`
- `sam-ba`

Default protocol is `sam-ba`

You can change upload protocol using `upload_protocol` option:

```
[env:adafruit_pyportal_m4]
platform = atmelsam
board = adafruit_pyportal_m4

upload_protocol = sam-ba
```

Debugging

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

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for 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.

Compatible Tools	On-board	Default
<i>Atmel-ICE</i>		Yes
<i>J-LINK</i>		

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software 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 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.

Microcontroller	SAMD51J19A
Frequency	120MHz
Flash	512KB
RAM	192KB
Vendor	Adafruit

Configuration

Please use `adafruit_trellis_m4` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:adafruit_trellis_m4]
platform = atmelsam
board = adafruit_trellis_m4
```

You can override default Adafruit Trellis M4 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `adafruit_trellis_m4.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:adafruit_trellis_m4]
platform = atmelsam
board = adafruit_trellis_m4

; change microcontroller
board_build.mcu = samd51j19a

; change MCU frequency
board_build.f_cpu = 120000000L
```

Uploading

Adafruit Trellis M4 supports the next uploading protocols:

- `atmel-ice`
- `jlink`
- `sam-ba`

Default protocol is `sam-ba`

You can change upload protocol using `upload_protocol` option:

```
[env:adafruit_trellis_m4]
platform = atmelsam
board = adafruit_trellis_m4

upload_protocol = sam-ba
```

Debugging

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

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for 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.

Compatible Tools	On-board	Default
Atmel-ICE		Yes
J-LINK		

Frameworks

Name	Description
Arduino	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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.

Microcontroller	SAMD21E18A
Frequency	48MHz
Flash	256KB
RAM	32KB
Vendor	Adafruit

Configuration

Please use `adafruit_trinket_m0` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[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.

```
[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:

```
[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.

Compatible Tools	On-board	Default
<i>Atmel-ICE</i>		Yes
<i>Black Magic Probe</i>		
<i>J-LINK</i>		

Frameworks

Name	Description
<i>Ar-duino</i>	Arduino Wiring-based Framework allows writing cross-platform software 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 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.

Microcontroller	SAMD21E18A
Frequency	48MHz
Flash	256KB
RAM	32KB
Vendor	Adafruit

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.

Compatible Tools	On-board	Default
<i>Atmel-ICE</i>		Yes
<i>Black Magic Probe</i>		
<i>J-LINK</i>		

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software 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 pyBadge AirLift M4

Contents

- Adafruit pyBadge AirLift M4
 - Hardware
 - Configuration
 - Uploading
 - Debugging
 - Frameworks

Hardware

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

Microcontroller	SAMD51J20A
Frequency	120MHz
Flash	1008KB
RAM	192KB
Vendor	Adafruit

Configuration

Please use `adafruit_pybadge_airlift_m4` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:adafruit_pybadge_airlift_m4]
platform = atmelsam
board = adafruit_pybadge_airlift_m4
```

You can override default Adafruit pyBadge AirLift M4 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `adafruit_pybadge_airlift_m4.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:adafruit_pybadge_airlift_m4]
platform = atmelsam
board = adafruit_pybadge_airlift_m4

; change microcontroller
board_build.mcu = samd51j20a

; change MCU frequency
board_build.f_cpu = 120000000L
```

Uploading

Adafruit pyBadge AirLift M4 supports the next uploading protocols:

- atmel-ice
- jlink
- sam-ba

Default protocol is `sam-ba`

You can change upload protocol using `upload_protocol` option:

```
[env:adafruit_pybadge_airlift_m4]
platform = atmelsam
board = adafruit_pybadge_airlift_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 AirLift M4 does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

Compatible Tools	On-board	Default
Atmel-ICE		Yes
J-LINK		

Frameworks

Name	Description
Arduino	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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.

Microcontroller	SAMD51J19A
Frequency	120MHz
Flash	512KB
RAM	192KB
Vendor	Adafruit

Configuration

Please use `adafruit_pybadge_m4` ID for `board` option in “`platformio.ini`” (Project Configuration File):

```
[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
```

(continues on next page)

(continued from previous page)

```
; 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.

Compatible Tools	On-board	Default
Atmel-ICE		Yes
J-LINK		

Frameworks

Name	Description
Arduino	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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.

Microcontroller	AT91SAM3X8E
Frequency	84MHz
Flash	512KB
RAM	96KB
Vendor	Arduino

Configuration

Please use due ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[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.

```
[env:due]
platform = atmelsam
board = due

; change microcontroller
board_build.mcu = at91sam3x8e

; change MCU frequency
board_build.f_cpu = 8400000L
```

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:

```
[env:due]
platform = atmelsam
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.

Compatible Tools	On-board	Default
<i>Atmel-ICE</i>		Yes
<i>Black Magic Probe</i>		
<i>J-LINK</i>		
<i>ST-LINK</i>		

Frameworks

Name	Description
<i>Ar-duino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>Simba</i>	Simba is an RTOS and build framework. It aims to make embedded programming easy and portable.

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.

Microcontroller	AT91SAM3X8E
Frequency	84MHz
Flash	512KB
RAM	96KB
Vendor	Arduino

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

; 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.

Compatible Tools	On-board	Default
<i>Atmel-ICE</i>		Yes
<i>Black Magic Probe</i>		
<i>J-LINK</i>		
<i>ST-LINK</i>		

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>Simba</i>	Simba is an RTOS and build framework. It aims to make embedded programming easy and portable.

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.

Microcontroller	SAMD21G18A
Frequency	48MHz
Flash	256KB
RAM	32KB
Vendor	Arduino

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

; change microcontroller
board_build.mcu = samd21g18a

; change MCU frequency
board_build.f_cpu = 48000000L
```

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:

```
[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.

Compatible Tools	On-board	Default
<i>Atmel-ICE</i>		Yes
<i>Black Magic Probe</i>		
<i>J-LINK</i>		

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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.

Microcontroller	SAMD21G18A
Frequency	48MHz
Flash	256KB
RAM	32KB
Vendor	Arduino

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

; change microcontroller
board_build.mcu = samd21g18a

; change MCU frequency
board_build.f_cpu = 48000000L
```

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:

```
[env:mzeroproUSB]
platform = atmelsam
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.

Compatible Tools	On-board	Default
<i>Atmel-ICE</i>		Yes
<i>Black Magic Probe</i>		
<i>J-LINK</i>		

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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.

Microcontroller	SAMD21G18A
Frequency	48MHz
Flash	256KB
RAM	32KB
Vendor	Arduino

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

; change microcontroller
board_build.mcu = samd21g18a

; change MCU frequency
board_build.f_cpu = 48000000L
```

Uploading

Arduino M0 Pro (Programming/Debug Port) supports the next uploading protocols:

- atmel-ice
- blackmagic
- cmsis-dap
- jlink

Default protocol is cmsis-dap

You can change upload protocol using `upload_protocol` option:

```
[env:mzeropro]
platform = atmelsam
board = mzeropro

upload_protocol = cmsis-dap
```

Debugging

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

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for 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.

Compatible Tools	On-board	Default
<i>Atmel-ICE</i>		
<i>Black Magic Probe</i>		
<i>CMSIS-DAP</i>	Yes	Yes
<i>J-LINK</i>		

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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.

Microcontroller	SAMD21G18A
Frequency	48MHz
Flash	256KB
RAM	32KB
Vendor	Arduino

Configuration

Please use `mkrfox1200` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[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.

```
[env:mkrfox1200]
platform = atmelsam
board = mkrfox1200

; 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 = atmelsam
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.

Compatible Tools	On-board	Default
<i>Atmel-ICE</i>		Yes
<i>Black Magic Probe</i>		
<i>J-LINK</i>		

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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.

Microcontroller	SAMD21G18A
Frequency	48MHz
Flash	256KB
RAM	32KB
Vendor	Arduino

Configuration

Please use `mkrgsm1400` ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[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.

```
[env:mkrgsm1400]
platform = atmelsam
board = mkrgsm1400

; change microcontroller
board_build.mcu = samd21g18a

; change MCU frequency
board_build.f_cpu = 48000000L
```

Uploading

Arduino MKR GSM 1400 supports the next uploading protocols:

- atmel-ice
- blackmagic
- jlink
- sam-ba

Default protocol is sam-ba

You can change upload protocol using `upload_protocol` option:

```
[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.

Compatible Tools	On-board	Default
<i>Atmel-ICE</i>		Yes
<i>Black Magic Probe</i>		
<i>J-LINK</i>		

Frameworks

Name	Description
<i>Arduino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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.

Microcontroller	SAMD21G18A
Frequency	48MHz
Flash	256KB
RAM	32KB
Vendor	Arduino

Configuration

Please use `mkrnb1500` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[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.

```
[env:mkrnb1500]
platform = atmelsam
board = mkrnb1500
```

(continues on next page)

(continued from previous page)

```
; 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:

```
[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.

Compatible Tools	On-board	Default
<i>Atmel-ICE</i>		Yes
<i>Black Magic Probe</i>		
<i>J-LINK</i>		

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.

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.

Microcontroller	SAMD21G18A
Frequency	48MHz
Flash	256KB
RAM	32KB
Vendor	Arduino

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)

(continued from previous 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:

```
[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.

Compatible Tools	On-board	Default
<i>Atmel-ICE</i>		Yes
<i>Black Magic Probe</i>		
<i>J-LINK</i>		

Frameworks

Name	Description
<i>Arduino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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.

Microcontroller	SAMD21G18A
Frequency	48MHz
Flash	256KB
RAM	32KB
Vendor	Arduino

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
```

(continues on next page)

(continued from previous page)

```
; change microcontroller
board_build.mcu = samd21g18a

; change MCU frequency
board_build.f_cpu = 48000000L
```

Uploading

Arduino MKR WiFi 1010 supports the next uploading protocols:

- atmel-ice
- blackmagic
- jlink
- sam-ba

Default protocol is sam-ba

You can change upload protocol using *upload_protocol* option:

```
[env:mkrwifi1010]
platform = 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.

Compatible Tools	On-board	Default
<i>Atmel-ICE</i>		Yes
<i>Black Magic Probe</i>		
<i>J-LINK</i>		

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.

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.

Microcontroller	SAMD21G18A
Frequency	48MHz
Flash	256KB
RAM	32KB
Vendor	Arduino

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)

(continued from previous page)

```
; change microcontroller
board_build.mcu = samd21g18a

; change MCU frequency
board_build.f_cpu = 48000000L
```

Uploading

Arduino MKR1000 supports the next uploading protocols:

- atmel-ice
- blackmagic
- jlink
- sam-ba

Default protocol is sam-ba

You can change upload protocol using *upload_protocol* option:

```
[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.

Compatible Tools	On-board	Default
<i>Atmel-ICE</i>		Yes
<i>Black Magic Probe</i>		
<i>J-LINK</i>		

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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.

Microcontroller	SAMD21G18A
Frequency	48MHz
Flash	256KB
RAM	32KB
Vendor	Arduino

Configuration

Please use `mkrzero` ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[env:mkrzero]
platform = atmelsam
board = mkrzero
```

You can override default Arduino MKRZERO settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `mkrzero.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:mkrzero]
platform = atmelsam
board = mkrzero
```

(continues on next page)

(continued from previous page)

```
; change microcontroller
board_build.mcu = samd21g18a

; change MCU frequency
board_build.f_cpu = 48000000L
```

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:

```
[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.

Compatible Tools	On-board	Default
<i>Atmel-ICE</i>		Yes
<i>Black Magic Probe</i>		
<i>J-LINK</i>		

Frameworks

Name	Description
<i>Arduino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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.

Microcontroller	SAMD21G18A
Frequency	48MHz
Flash	256KB
RAM	32KB
Vendor	Arduino

Configuration

Please use `tian` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:tian]
platform = atmelsam
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 = atmelsam
board = tian

; change microcontroller
```

(continues on next page)

(continued from previous page)

```
board_build.mcu = samd21g18a
; change MCU frequency
board_build.f_cpu = 48000000L
```

Uploading

Arduino Tian supports the next uploading protocols:

- atmel-ice
- blackmagic
- jlink
- stk500v2

Default protocol is stk500v2

You can change upload protocol using *upload_protocol* option:

```
[env:tian]
platform = 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.

Compatible Tools	On-board	Default
<i>Atmel-ICE</i>		Yes
<i>Black Magic Probe</i>		
<i>J-LINK</i>		

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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.

Microcontroller	SAMD21G18A
Frequency	48MHz
Flash	256KB
RAM	32KB
Vendor	Arduino

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.

Compatible Tools	On-board	Default
<i>Atmel-ICE</i>		
<i>Black Magic Probe</i>		
<i>CMSIS-DAP</i>	Yes	Yes
<i>J-LINK</i>		

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

Arduino Zero (USB Native Port)

Contents

- *Arduino Zero (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.

Microcontroller	SAMD21G18A
Frequency	48MHz
Flash	256KB
RAM	32KB
Vendor	Arduino

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:

```
[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.

Compatible Tools	On-board	Default
<i>Atmel-ICE</i>		Yes
<i>Black Magic Probe</i>		
<i>J-LINK</i>		

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

Atmel ATSAMR21-XPRO

Contents

- *Atmel ATSAMR21-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	SAMR21G18A
Frequency	48MHz
Flash	256KB
RAM	32KB
Vendor	Atmel

Configuration

Please use `samr21_xpro` ID for `board` option in “`platformio.ini`” (Project Configuration File):

```
[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.

```
[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:

```
[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.

Compatible Tools	On-board	Default
<i>Atmel-ICE</i>		
<i>Black Magic Probe</i>		
<i>CMSIS-DAP</i>	Yes	Yes
<i>J-LINK</i>		

Frameworks

Name	Description
<i>mbed</i>	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.

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.

Microcontroller	SAMD21G18A
Frequency	48MHz
Flash	256KB
RAM	32KB
Vendor	Atmel

Configuration

Please use `samd21g18a` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[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.

```
[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:

```
[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.

Compatible Tools	On-board	Default
<i>Atmel-ICE</i>		
<i>Black Magic Probe</i>		
<i>CMSIS-DAP</i>	Yes	Yes
<i>J-LINK</i>		

Frameworks

Name	Description
<i>Arduino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>mbed</i>	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.

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*):

```
[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.

```
[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:

```
[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.

Compatible Tools	On-board	Default
<i>Atmel-ICE</i>		
<i>Black Magic Probe</i>		
<i>CMSIS-DAP</i>	Yes	Yes
<i>J-LINK</i>		

Frameworks

Name	Description
<i>mbed</i>	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.

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.

Microcontroller	SAML21J18B
Frequency	48MHz
Flash	256KB
RAM	32KB
Vendor	Atmel

Configuration

Please use `saml21_xpro_b` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:saml21_xpro_b]
platform = atmelsam
board = saml21_xpro_b
```

You can override default Atmel SAML21-XPRO-B settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `saml21_xpro_b.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:saml21_xpro_b]
platform = 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:

```
[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.

Compatible Tools	On-board	Default
<i>Atmel-ICE</i>		
<i>Black Magic Probe</i>		
<i>CMSIS-DAP</i>	Yes	Yes
<i>J-LINK</i>		

Frameworks

Name	Description
<i>mbed</i>	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.

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.

Microcontroller	AT91SAM3X8E
Frequency	84MHz
Flash	512KB
RAM	96KB
Vendor	Digistump

Configuration

Please use `digix` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:digix]
platform = atmelsam
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 = atmelsam
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 = atmelsam
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.

Compatible Tools	On-board	Default
<i>Atmel-ICE</i>		Yes
<i>Black Magic Probe</i>		
<i>J-LINK</i>		
<i>ST-LINK</i>		

Frameworks

Name	Description
<i>Ardu- ino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

LowPowerLab CurrentRanger

Contents

- *LowPowerLab CurrentRanger*
 - *Hardware*
 - *Configuration*
 - *Debugging*
 - *Frameworks*

Hardware

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

Microcontroller	SAMD21G18A
Frequency	48MHz
Flash	256KB
RAM	32KB
Vendor	LowPowerLab

Configuration

Please use `current_ranger` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[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.

```
[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

Name	Description
<code>Ar-duino</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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.

Microcontroller	SAMD21G18A
Frequency	48MHz
Flash	256KB
RAM	32KB
Vendor	Arduino

Configuration

Please use `mkrvidor4000` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[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.

```
[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:

```
[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.

Compatible Tools	On-board	Default
<i>Atmel-ICE</i>		Yes
<i>Black Magic Probe</i>		
<i>J-LINK</i>		

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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.

Microcontroller	AT91SAM3X8E
Frequency	84MHz
Flash	512KB
RAM	96KB
Vendor	Macchina

Configuration

Please use macchina2 ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[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.

```
[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:

```
[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.

Compatible Tools	On-board	Default
<i>Atmel-ICE</i>		Yes
<i>Black Magic Probe</i>		
<i>J-LINK</i>		
<i>ST-LINK</i>		

Frameworks

Name	Description
<i>Ardu- ino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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.

Microcontroller	SAMD21J18A
Frequency	48MHz
Flash	256KB
RAM	32KB
Vendor	ReprapWorld

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.

Compatible Tools	On-board	Default
<i>Atmel-ICE</i>		Yes
<i>Black Magic Probe</i>		
<i>J-LINK</i>		

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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.

Microcontroller	SAMD21G18A
Frequency	48MHz
Flash	256KB
RAM	32KB
Vendor	LowPowerLab

Configuration

Please use `moteino_zero` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[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.

```
[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:

```
[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	On-board	Default
Atmel-ICE		Yes
Black Magic Probe		
CMSIS-DAP		
J-LINK		

Frameworks

Name	Description
Arduino	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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.

Microcontroller	SAMD21G18A
Frequency	48MHz
Flash	256KB
RAM	32KB
Vendor	Arduino

Configuration

Please use `nano_33_iot` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:nano_33_iot]
platform = atmelsam
board = nano_33_iot
```

You can override default NANO 33 IoT settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `nano_33_iot.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:nano_33_iot]
platform = atmelsam
board = nano_33_iot

; 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:

```
[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.

Compatible Tools	On-board	Default
<i>Atmel-ICE</i>		Yes
<i>Black Magic Probe</i>		
<i>J-LINK</i>		

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.

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.

Microcontroller	SAMD21J18A
Frequency	48MHz
Flash	256KB
RAM	32KB
Vendor	SODAQ

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)

(continued from previous page)

```
; change microcontroller
board_build.mcu = samd21j18a

; change MCU frequency
board_build.f_cpu = 48000000L
```

Uploading

SODAQ Autonomo supports the next uploading protocols:

- atmel-ice
- blackmagic
- jlink
- sam-ba

Default protocol is sam-ba

You can change upload protocol using *upload_protocol* option:

```
[env:sodaq_autonomo]
platform = 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.

Compatible Tools	On-board	Default
<i>Atmel-ICE</i>		Yes
<i>Black Magic Probe</i>		
<i>J-LINK</i>		

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.

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.

Microcontroller	SAMD21J18A
Frequency	48MHz
Flash	256KB
RAM	32KB
Vendor	SODAQ

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)

(continued from previous 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:

```
[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.

Compatible Tools	On-board	Default
<i>Atmel-ICE</i>		Yes
<i>Black Magic Probe</i>		
<i>J-LINK</i>		

Frameworks

Name	Description
<i>Arduino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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.

Microcontroller	SAMD21G18A
Frequency	48MHz
Flash	256KB
RAM	32KB
Vendor	SODAQ

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)

(continued from previous page)

```
; change microcontroller
board_build.mcu = samd21g18a

; change MCU frequency
board_build.f_cpu = 48000000L
```

Uploading

SODAQ ONE supports the next uploading protocols:

- atmel-ice
- blackmagic
- jlink
- sam-ba

Default protocol is sam-ba

You can change upload protocol using *upload_protocol* option:

```
[env:sodaq_one]
platform = atmelsam
board = sodaq_one

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 ONE does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

Compatible Tools	On-board	Default
<i>Atmel-ICE</i>		Yes
<i>Black Magic Probe</i>		
<i>J-LINK</i>		

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.

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.

Microcontroller	SAMD21J18A
Frequency	48MHz
Flash	256KB
RAM	32KB
Vendor	SODAQ

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)

(continued from previous 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.

Compatible Tools	On-board	Default
<i>Atmel-ICE</i>		Yes
<i>Black Magic Probe</i>		
<i>J-LINK</i>		

Frameworks

Name	Description
<i>Arduino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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.

Microcontroller	SAMD21G18A
Frequency	48MHz
Flash	256KB
RAM	32KB
Vendor	SODAQ

Configuration

Please use `sodaq_sff` ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[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.

```
[env:sodaq_sff]
platform = atmelsam
board = sodaq_sff

; change microcontroller
```

(continues on next page)

(continued from previous page)

```
board_build.mcu = samd21g18a
; change MCU frequency
board_build.f_cpu = 48000000L
```

Uploading

SODAQ SFF supports the next uploading protocols:

- atmel-ice
- blackmagic
- jlink
- sam-ba

Default protocol is sam-ba

You can change upload protocol using *upload_protocol* option:

```
[env:sodaq_sff]
platform = atmelsam
board = sodaq_sff

upload_protocol = sam-ba
```

Debugging

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

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for 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.

Compatible Tools	On-board	Default
<i>Atmel-ICE</i>		Yes
<i>Black Magic Probe</i>		
<i>J-LINK</i>		

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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.

Microcontroller	AT91SAM3X8E
Frequency	84MHz
Flash	512KB
RAM	96KB
Vendor	SainSmart

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 = 8400000L
```

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:

```
[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.

Compatible Tools	On-board	Default
<i>Atmel-ICE</i>		Yes
<i>Black Magic Probe</i>		
<i>J-LINK</i>		
<i>ST-LINK</i>		

Frameworks

Name	Description
<i>Ar-duino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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.

Microcontroller	AT91SAM3X8E
Frequency	84MHz
Flash	512KB
RAM	96KB
Vendor	SainSmart

Configuration

Please use `sainSmartDueUSB` ID for `board` option in “*platformio.ini*” (*Project Configuration File*):

```
[env:sainSmartDueUSB]
platform = atmelsam
board = sainSmartDueUSB
```

You can override default SainSmart Due (USB Native Port) settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `sainSmartDueUSB.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:sainSmartDueUSB]
platform = atmelsam
board = sainSmartDueUSB

; change microcontroller
board_build.mcu = at91sam3x8e

; change MCU frequency
board_build.f_cpu = 84000000L
```

Uploading

SainSmart Due (USB Native Port) supports the next uploading protocols:

- `atmel-ice`
- `blackmagic`
- `jlink`

- sam-ba
- stlink

Default protocol is sam-ba

You can change upload protocol using *upload_protocol* option:

```
[env:sainSmartDueUSB]
platform = atmelsam
board = sainSmartDueUSB

upload_protocol = sam-ba
```

Debugging

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

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for 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.

Compatible Tools	On-board	Default
<i>Atmel-ICE</i>		Yes
<i>Black Magic Probe</i>		
<i>J-LINK</i>		
<i>ST-LINK</i>		

Frameworks

Name	Description
<i>Ardu</i> <i>duino</i>	Arduino Wiring-based Framework allows writing cross-platform software 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

Contents

- *Seeeduino LoRaWAN*
 - *Hardware*
 - *Configuration*

- *Uploading*
- *Debugging*
- *Frameworks*

Hardware

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

Microcontroller	SAMD21G18A
Frequency	48MHz
Flash	256KB
RAM	32KB
Vendor	Seeed

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.

Compatible Tools	On-board	Default
<i>Atmel-ICE</i>		Yes
<i>Black Magic Probe</i>		
<i>J-LINK</i>		

Frameworks

Name	Description
<i>Ardu- ino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

SparkFun 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.

Microcontroller	SAMD21G18A
Frequency	48MHz
Flash	256KB
RAM	32KB
Vendor	SparkFun

Configuration

Please use `sparkfun_samd21_dev_usb` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[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.

```
[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:

```
[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.

Compatible Tools	On-board	Default
<i>Atmel-ICE</i>		Yes
<i>Black Magic Probe</i>		
<i>J-LINK</i>		

Frameworks

Name	Description
<i>Ardu- ino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

SparkFun 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*):

```
[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.

```
[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:

```
[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.

Compatible Tools	On-board	Default
<i>Atmel-ICE</i>		Yes
<i>Black Magic Probe</i>		
<i>J-LINK</i>		

Frameworks

Name	Description
<i>Ar-duino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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.

Microcontroller	SAMD21G18A
Frequency	48MHz
Flash	256KB
RAM	32KB
Vendor	Gimasi

Configuration

Please use tuinozero96 ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:tuinozero96]
platform = atmelsam
board = tuinozero96
```

You can override default Tuino 096 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `tuinozero96.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:tuinozero96]
platform = atmelsam
board = tuinozero96

; change microcontroller
board_build.mcu = samd21g18a

; change MCU frequency
board_build.f_cpu = 48000000L
```

Uploading

Tuino 096 supports the next uploading protocols:

- atmel-ice
- blackmagic
- jlink
- stk500v2

Default protocol is `stk500v2`

You can change upload protocol using `upload_protocol` option:

```
[env:tuinozero96]
platform = 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.

Compatible Tools	On-board	Default
<i>Atmel-ICE</i>		Yes
<i>Black Magic Probe</i>		
<i>J-LINK</i>		

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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.

Microcontroller	ESP32
Frequency	240MHz
Flash	4MB
RAM	320KB
Vendor	AI Thinker

Configuration

Please use `esp32cam` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:esp32cam]
platform = espressif32
board = esp32cam
```

You can override default AI Thinker ESP32-CAM settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `esp32cam.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:esp32cam]
platform = espressif32
board = esp32cam

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

Uploading

AI Thinker ESP32-CAM supports the next uploading protocols:

- esp-prog
- espota
- esptool
- iot-bus-jtag
- jlink
- minimodule
- olimex-arm-usb-ocd
- olimex-arm-usb-ocd-h
- olimex-arm-usb-tiny-h
- olimex-jtag-tiny
- tumpa

Default protocol is `esptool`

You can change upload protocol using `upload_protocol` option:

```
[env:esp32cam]
platform = espressif32
board = esp32cam

upload_protocol = esptool
```

Debugging

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

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using *debug_tool* option in “*platformio.ini*” (*Project Configuration File*).

AI Thinker ESP32-CAM does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

Compatible Tools	On-board	Default
<i>ESP-Prog</i>		Yes
<i>oddWires IOT-Bus JTAG</i>		
<i>J-LINK</i>		
<i>Mini-Module FT2232H</i>		
<i>Olimex ARM-USB-OCD</i>		
<i>Olimex ARM-USB-OCD-H</i>		
<i>Olimex ARM-USB-TINY-H</i>		
<i>Olimex ARM-USB-TINY</i>		
<i>TIAO USB Multi-Protocol Adapter (TUMPA)</i>		

Frameworks

Name	Description
<i>Ar-duino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP-IDF</i>	Espressif IoT Development Framework. Official development framework for ESP32.

ALKS ESP32

Contents

- *ALKS ESP32*
 - *Hardware*
 - *Configuration*
 - *Uploading*
 - *Debugging*
 - *Frameworks*

Hardware

Platform *Espressif 32*: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

Microcontroller	ESP32
Frequency	240MHz
Flash	4MB
RAM	320KB
Vendor	RoboticsBrno

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 = alksesp32

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.

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

Name	Description
Arduin	Arduino Wiring-based Framework allows writing cross-platform software 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 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.

Microcontroller	ESP32
Frequency	240MHz
Flash	4MB
RAM	320KB
Vendor	Adafruit

Configuration

Please use `featheresp32` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:featheresp32]
platform = espressif32
board = featheresp32
```

You can override default Adafruit ESP32 Feather settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `featheresp32.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:featheresp32]
platform = espressif32
board = featheresp32

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

Uploading

Adafruit ESP32 Feather supports the next uploading protocols:

- esp-prog
- espota
- esptool
- iot-bus-jtag
- jlink
- minimodule
- olimex-arm-usb-ocd

- olimex-arm-usb-ocd-h
- olimex-arm-usb-tiny-h
- olimex-jtag-tiny
- tumpa

Default protocol is esptool

You can change upload protocol using *upload_protocol* option:

```
[env:featheresp32]
platform = espressif32
board = featheresp32

upload_protocol = esptool
```

Debugging

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

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for 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
<i>ESP-Prog</i>		Yes
<i>oddWires IOT-Bus JTAG</i>		
<i>J-LINK</i>		
<i>Mini-Module FT2232H</i>		
<i>Olimex ARM-USB-OCD</i>		
<i>Olimex ARM-USB-OCD-H</i>		
<i>Olimex ARM-USB-TINY-H</i>		
<i>Olimex ARM-USB-TINY</i>		
<i>TIAO USB Multi-Protocol Adapter (TUMPA)</i>		

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP-IDF</i>	Espressif IoT Development Framework. Official development framework for ESP32.

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.

Microcontroller	ESP32
Frequency	240MHz
Flash	4MB
RAM	320KB
Vendor	April Brother

Configuration

Please use espea32 ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[env:espea32]
platform = espressif32
board = espea32
```

You can override default April Brother ESPea32 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `espea32.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:espea32]
platform = espressif32
board = espea32

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 24000000L
```

Uploading

April Brother ESPea32 supports the next uploading protocols:

- espota
- esptool

Default protocol is esptool

You can change upload protocol using *upload_protocol* option:

```
[env:espea32]
platform = espressif32
board = espea32

upload_protocol = esptool
```

Debugging

PIO Unified Debugger currently does not support April Brother ESPeA32 board.

Frameworks

Name	Description
Arduin	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
ESP-IDF	Espressif IoT Development Framework. Official development framework for ESP32.

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.

Microcontroller	ESP32
Frequency	160MHz
Flash	4MB
RAM	320KB
Vendor	BPI Tech

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

Name	Description
<i>Ar-duino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP-IDF</i>	Espressif IoT Development Framework. Official development framework for ESP32.

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.

Microcontroller	ESP32
Frequency	240MHz
Flash	4MB
RAM	320KB
Vendor	DSTIKE

Configuration

Please use d-duino-32 ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[env:d-duino-32]
platform = espressif32
board = d-duino-32
```

You can override default D-duino-32 settings per build environment using *board_**** option, where *** is a JSON object path from board manifest *d-duino-32.json*. For example, *board_build.mcu*, *board_build.f_cpu*, etc.

```
[env:d-duino-32]
platform = espressif32
board = d-duino-32
```

(continues on next page)

(continued from previous page)

```
; 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:

```
[env:d-duino-32]
platform = espressif32
board = d-duino-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*).

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.

Compatible Tools	On-board	Default
<i>ESP-Prog</i>		Yes
<i>oddWires IOT-Bus JTAG</i>		
<i>J-LINK</i>		
<i>Mini-Module FT2232H</i>		
<i>Olimex ARM-USB-OCD</i>		
<i>Olimex ARM-USB-OCD-H</i>		
<i>Olimex ARM-USB-TINY-H</i>		
<i>Olimex ARM-USB-TINY</i>		
<i>TIAO USB Multi-Protocol Adapter (TUMPA)</i>		

Frameworks

Name	Description
<i>Ar-duino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP-IDF</i>	Espressif IoT Development Framework. Official development framework for ESP32.

DOIT ESP32 DEVKIT V1

Contents

- *DOIT ESP32 DEVKIT V1*
 - *Hardware*
 - *Configuration*
 - *Uploading*
 - *Debugging*
 - *Frameworks*

Hardware

Platform *Espressif 32*: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

Microcontroller	ESP32
Frequency	240MHz
Flash	4MB
RAM	320KB
Vendor	DOIT

Configuration

Please use esp32doit-devkit-v1 ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[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.

```
[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.

Compatible Tools	On-board	Default
<i>ESP-Prog</i>		Yes
<i>oddWires IOT-Bus JTAG</i>		
<i>J-LINK</i>		
<i>Mini-Module FT2232H</i>		
<i>Olimex ARM-USB-OCD</i>		
<i>Olimex ARM-USB-OCD-H</i>		
<i>Olimex ARM-USB-TINY-H</i>		
<i>Olimex ARM-USB-TINY</i>		
<i>TIAO USB Multi-Protocol Adapter (TUMPA)</i>		

Frameworks

Name	Description
<i>Ar-duino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP-IDF</i>	Espressif IoT Development Framework. Official development framework for ESP32.

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.

Microcontroller	ESP32
Frequency	240MHz
Flash	4MB
RAM	320KB
Vendor	Dongsen Technology

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.

Compatible Tools	On-board	Default
<i>ESP-Prog</i>		Yes
<i>oddWires IOT-Bus JTAG</i>		
<i>J-LINK</i>		
<i>Mini-Module FT2232H</i>		
<i>Olimex ARM-USB-OCD</i>		
<i>Olimex ARM-USB-OCD-H</i>		
<i>Olimex ARM-USB-TINY-H</i>		
<i>Olimex ARM-USB-TINY</i>		
<i>TIAO USB Multi-Protocol Adapter (TUMPA)</i>		

Frameworks

Name	Description
<i>Ar-duino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP-IDF</i>	Espressif IoT Development Framework. Official development framework for ESP32.

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.

Microcontroller	ESP32
Frequency	240MHz
Flash	4MB
RAM	320KB
Vendor	Unknown

Configuration

Please use `fm-devkit` ID for `board` option in “*platformio.ini*” (*Project Configuration File*):

```
[env:fm-devkit]
platform = espressif32
board = fm-devkit
```

You can override default ESP32 FM DevKit settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `fm-devkit.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:fm-devkit]
platform = espressif32
board = fm-devkit

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

Uploading

ESP32 FM DevKit supports the next uploading protocols:

- `esp-prog`
- `espota`

- esptool
- iot-bus-jtag
- jlink
- minimodule
- olimex-arm-usb-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: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.

Compatible Tools	On-board	Default
<i>ESP-Prog</i>		Yes
<i>oddWires IOT-Bus JTAG</i>		
<i>J-LINK</i>		
<i>Mini-Module FT2232H</i>		
<i>Olimex ARM-USB-OCD</i>		
<i>Olimex ARM-USB-OCD-H</i>		
<i>Olimex ARM-USB-TINY-H</i>		
<i>Olimex ARM-USB-TINY</i>		
<i>TIAO USB Multi-Protocol Adapter (TUMPA)</i>		

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP-IDF</i>	Espressif IoT Development Framework. Official development framework for ESP32.

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.

Microcontroller	ESP32
Frequency	240MHz
Flash	4MB
RAM	320KB
Vendor	Espressif

Configuration

Please use `pico32` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:pico32]
platform = espressif32
board = pico32
```

You can override default ESP32 Pico Kit settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `pico32.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:pico32]
platform = espressif32
board = pico32

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

Uploading

ESP32 Pico Kit supports the next uploading protocols:

- espota
- esptool

Default protocol is esptool

You can change upload protocol using *upload_protocol* option:

```
[env:pico32]
platform = espressif32
board = pico32

upload_protocol = esptool
```

Debugging

PIO Unified Debugger currently does not support ESP32 Pico Kit board.

Frameworks

Name	Description
<i>Ar-duino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP-IDF</i>	Espressif IoT Development Framework. Official development framework for ESP32.

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.

Microcontroller	ESP32
Frequency	240MHz
Flash	4MB
RAM	320KB
Vendor	ESP32vn

Configuration

Please use `esp32vn-iot-uno` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:esp32vn-iot-uno]
platform = espressif32
board = esp32vn-iot-uno
```

You can override default ESP32vn IoT Uno settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `esp32vn-iot-uno.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:esp32vn-iot-uno]
platform = espressif32
board = esp32vn-iot-uno

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

Uploading

ESP32vn IoT Uno supports the next uploading protocols:

- `esp-prog`
- `espota`
- `esptool`
- `iot-bus-jtag`
- `jlink`
- `minimodule`

- olimex-arm-usb-ocd
- olimex-arm-usb-ocd-h
- olimex-arm-usb-tiny-h
- olimex-jtag-tiny
- tumpa

Default protocol is esptool

You can change upload protocol using *upload_protocol* option:

```
[env:esp32vn-iot-uno]
platform = espressif32
board = esp32vn-iot-uno

upload_protocol = esptool
```

Debugging

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

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for 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.

Compatible Tools	On-board	Default
<i>ESP-Prog</i>		Yes
<i>oddWires IOT-Bus JTAG</i>		
<i>J-LINK</i>		
<i>Mini-Module FT2232H</i>		
<i>Olimex ARM-USB-OCD</i>		
<i>Olimex ARM-USB-OCD-H</i>		
<i>Olimex ARM-USB-TINY-H</i>		
<i>Olimex ARM-USB-TINY</i>		
<i>TIAO USB Multi-Protocol Adapter (TUMPA)</i>		

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP-IDF</i>	Espressif IoT Development Framework. Official development framework for ESP32.

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.

Compatible Tools	On-board	Default
<i>ESP-Prog</i>		Yes
<i>oddWires IOT-Bus JTAG</i>		
<i>J-LINK</i>		
<i>Mini-Module FT2232H</i>		
<i>Olimex ARM-USB-OCD</i>		
<i>Olimex ARM-USB-OCD-H</i>		
<i>Olimex ARM-USB-TINY-H</i>		
<i>Olimex ARM-USB-TINY</i>		
<i>TIAO USB Multi-Protocol Adapter (TUMPA)</i>		

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP-IDF</i>	Espressif IoT Development Framework. Official development framework for ESP32.

ESPino32

Contents

- *ESPino32*
 - *Hardware*
 - *Configuration*
 - *Uploading*
 - *Debugging*
 - *Frameworks*

Hardware

Platform *Espressif 32*: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

Microcontroller	ESP32
Frequency	240MHz
Flash	4MB
RAM	320KB
Vendor	ThaiEasyElec

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
```

(continues on next page)

(continued from previous page)

```
; 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.

Compatible Tools	On-board	Default
<i>ESP-Prog</i>		Yes
<i>oddWires IOT-Bus JTAG</i>		
<i>J-LINK</i>		
<i>Mini-Module FT2232H</i>		
<i>Olimex ARM-USB-OCD</i>		
<i>Olimex ARM-USB-OCD-H</i>		
<i>Olimex ARM-USB-TINY-H</i>		
<i>Olimex ARM-USB-TINY</i>		
<i>TIAO USB Multi-Protocol Adapter (TUMPA)</i>		

Frameworks

Name	Description
<i>Ar-duino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP-IDF</i>	Espressif IoT Development Framework. Official development framework for ESP32.

Electronic SweetPeas ESP320

Contents

- *Electronic SweetPeas ESP320*
 - *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	Electronic SweetPeas

Configuration

Please use esp320 ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:esp320]
platform = espressif32
board = esp320
```

You can override default Electronic SweetPeas ESP320 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `esp320.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:esp320]
platform = espressif32
board = esp320

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

Uploading

Electronic SweetPeas ESP320 supports the next uploading protocols:

- espota
- esptool

Default protocol is esptool

You can change upload protocol using `upload_protocol` option:

```
[env:esp320]
platform = espressif32
board = esp320

upload_protocol = esptool
```

Debugging

PIO Unified Debugger currently does not support Electronic SweetPeas ESP320 board.

Frameworks

Name	Description
<code>Ar-duino</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<code>ESP-IDF</code>	Espressif IoT Development Framework. Official development framework for ESP32.

Espressif ESP-WROVER-KIT

Contents

- *Espressif ESP-WROVER-KIT*
 - *Hardware*
 - *Configuration*
 - *Uploading*
 - *Debugging*
 - *Frameworks*

Hardware

Platform [Espressif 32](#): Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

Microcontroller	ESP32
Frequency	240MHz
Flash	4MB
RAM	320KB
Vendor	Espressif

Configuration

Please use `esp-wrover-kit` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[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.

```
[env:esp-wrover-kit]
platform = espressif32
board = esp-wrover-kit

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 24000000L
```

Uploading

Espressif ESP-WROVER-KIT supports the next uploading protocols:

- esp-prog
- espota
- esptool
- ftdi
- iot-bus-jtag
- jlink
- minimodule
- olimex-arm-usb-ocd
- olimex-arm-usb-ocd-h
- olimex-arm-usb-tiny-h
- olimex-jtag-tiny
- tumpa

Default protocol is esptool

You can change upload protocol using *upload_protocol* option:

```
[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.

Compatible Tools	On-board	Default
<i>ESP-Prog</i>		
<i>FTDI Chip</i>	Yes	Yes
<i>oddWires IOT-Bus JTAG</i>		
<i>J-LINK</i>		
<i>Mini-Module FT2232H</i>		
<i>Olimex ARM-USB-OCD</i>		
<i>Olimex ARM-USB-OCD-H</i>		
<i>Olimex ARM-USB-TINY-H</i>		
<i>Olimex ARM-USB-TINY</i>		
<i>TIAO USB Multi-Protocol Adapter (TUMPA)</i>		

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP-IDF</i>	Espressif IoT Development Framework. Official development framework for ESP32.

Espressif ESP32 Dev Module

Contents

- *Espressif 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.

Microcontroller	ESP32
Frequency	240MHz
Flash	4MB
RAM	320KB
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.

Compatible Tools	On-board	Default
<i>ESP-Prog</i>		Yes
<i>oddWires IOT-Bus JTAG</i>		
<i>J-LINK</i>		
<i>Mini-Module FT2232H</i>		
<i>Olimex ARM-USB-OCD</i>		
<i>Olimex ARM-USB-OCD-H</i>		
<i>Olimex ARM-USB-TINY-H</i>		
<i>Olimex ARM-USB-TINY</i>		
<i>TIAO USB Multi-Protocol Adapter (TUMPA)</i>		

Frameworks

Name	Description
<i>Arduinio</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP-IDF</i>	Espressif IoT Development Framework. Official development framework for ESP32.

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.

Microcontroller	ESP32
Frequency	240MHz
Flash	4MB
RAM	320KB
Vendor	DFRobot

Configuration

Please use `firebeetle32` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:firebeetle32]
platform = espressif32
board = firebeetle32
```

You can override default FireBeetle-ESP32 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `firebeetle32.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:firebeetle32]
platform = espressif32
board = firebeetle32

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

Uploading

FireBeetle-ESP32 supports the next uploading protocols:

- esp-prog
- espota
- esptool
- iot-bus-jtag
- jlink
- minimodule
- olimex-arm-usb-ocd
- olimex-arm-usb-ocd-h
- olimex-arm-usb-tiny-h
- olimex-jtag-tiny
- tumpa

Default protocol is `esptool`

You can change upload protocol using `upload_protocol` option:

```
[env:firebeetle32]
platform = espressif32
board = firebeetle32

upload_protocol = esptool
```

Debugging

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

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for 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.

Compatible Tools	On-board	Default
<i>ESP-Prog</i>		Yes
<i>oddWires IOT-Bus JTAG</i>		
<i>J-LINK</i>		
<i>Mini-Module FT2232H</i>		
<i>Olimex ARM-USB-OCD</i>		
<i>Olimex ARM-USB-OCD-H</i>		
<i>Olimex ARM-USB-TINY-H</i>		
<i>Olimex ARM-USB-TINY</i>		
<i>TIAO USB Multi-Protocol Adapter (TUMPA)</i>		

Frameworks

Name	Description
<i>Ar-duino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP-IDF</i>	Espressif IoT Development Framework. Official development framework for ESP32.

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.

Microcontroller	ESP32
Frequency	240MHz
Flash	4MB
RAM	320KB
Vendor	Fred

Configuration

Please use `frogboard` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:frogboard]
platform = espressif32
board = frogboard
```

You can override default Frog Board ESP32 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `frogboard.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[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:

- esp-prog
- espota
- esptool
- iot-bus-jtag
- jlink
- minimodule
- olimex-arm-usb-ocd
- olimex-arm-usb-ocd-h
- olimex-arm-usb-tiny-h
- olimex-jtag-tiny

- tumpa

Default protocol is esptool

You can change upload protocol using *upload_protocol* option:

```
[env:frogboard]
platform = espressif32
board = frogboard

upload_protocol = esptool
```

Debugging

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

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for 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.

Compatible Tools	On-board	Default
<i>ESP-Prog</i>		Yes
<i>oddWires IOT-Bus JTAG</i>		
<i>J-LINK</i>		
<i>Mini-Module FT2232H</i>		
<i>Olimex ARM-USB-OCD</i>		
<i>Olimex ARM-USB-OCD-H</i>		
<i>Olimex ARM-USB-TINY-H</i>		
<i>Olimex ARM-USB-TINY</i>		
<i>TIAO USB Multi-Protocol Adapter (TUMPA)</i>		

Frameworks

Name	Description
<i>Ar-duino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP-IDF</i>	Espressif IoT Development Framework. Official development framework for ESP32.

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.

Microcontroller	ESP32
Frequency	240MHz
Flash	4MB
RAM	320KB
Vendor	Heltec Automation

Configuration

Please use `heltec_wifi_kit_32` ID for `board` option in “*platformio.ini*” (*Project Configuration File*):

```
[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.

```
[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:

```
[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

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP-IDF</i>	Espressif IoT Development Framework. Official development framework for ESP32.

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.

Microcontroller	ESP32
Frequency	240MHz
Flash	4MB
RAM	320KB
Vendor	Heltec Automation

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.

Compatible Tools	On-board	Default
<i>ESP-Prog</i>		Yes
<i>oddWires IOT-Bus JTAG</i>		
<i>J-LINK</i>		
<i>Mini-Module FT2232H</i>		
<i>Olimex ARM-USB-OCD</i>		
<i>Olimex ARM-USB-OCD-H</i>		
<i>Olimex ARM-USB-TINY-H</i>		
<i>Olimex ARM-USB-TINY</i>		
<i>TIAO USB Multi-Protocol Adapter (TUMPA)</i>		

Frameworks

Name	Description
<i>Ar-duino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP-IDF</i>	Espressif IoT Development Framework. Official development framework for ESP32.

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.

Microcontroller	ESP32
Frequency	240MHz
Flash	8MB
RAM	320KB
Vendor	Heltec Automation

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.

Compatible Tools	On-board	Default
<i>ESP-Prog</i>		Yes
<i>oddWires IOT-Bus JTAG</i>		
<i>J-LINK</i>		
<i>Mini-Module FT2232H</i>		
<i>Olimex ARM-USB-OCD</i>		
<i>Olimex ARM-USB-OCD-H</i>		
<i>Olimex ARM-USB-TINY-H</i>		
<i>Olimex ARM-USB-TINY</i>		
<i>TIAO USB Multi-Protocol Adapter (TUMPA)</i>		

Frameworks

Name	Description
<i>Ar-duino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP-IDF</i>	Espressif IoT Development Framework. Official development framework for ESP32.

Heltec Wireless Stick

Contents

- *Heltec Wireless Stick*
 - *Hardware*
 - *Configuration*
 - *Uploading*
 - *Debugging*
 - *Frameworks*

Hardware

Platform *Espressif 32*: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

Microcontroller	ESP32
Frequency	240MHz
Flash	8MB
RAM	320KB
Vendor	Heltec Automation

Configuration

Please use `heltec_wireless_stick` ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[env:heltec_wireless_stick]
platform = espressif32
board = heltec_wireless_stick
```

You can override default Heltec Wireless Stick settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `heltec_wireless_stick.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:heltec_wireless_stick]
platform = espressif32
board = heltec_wireless_stick

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

Uploading

Heltec Wireless Stick supports the next uploading protocols:

- esp-prog
- espota

- esptool
- iot-bus-jtag
- jlink
- minimodule
- olimex-arm-usb-ocd
- olimex-arm-usb-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.

Compatible Tools	On-board	Default
<i>ESP-Prog</i>		Yes
<i>oddWires IOT-Bus JTAG</i>		
<i>J-LINK</i>		
<i>Mini-Module FT2232H</i>		
<i>Olimex ARM-USB-OCD</i>		
<i>Olimex ARM-USB-OCD-H</i>		
<i>Olimex ARM-USB-TINY-H</i>		
<i>Olimex ARM-USB-TINY</i>		
<i>TIAO USB Multi-Protocol Adapter (TUMPA)</i>		

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP-IDF</i>	Espressif IoT Development Framework. Official development framework for ESP32.

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.

Microcontroller	ESP32
Frequency	240MHz
Flash	4MB
RAM	320KB
Vendor	Hornbill

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.

Compatible Tools	On-board	Default
<i>ESP-Prog</i>		Yes
<i>oddWires IOT-Bus JTAG</i>		
<i>J-LINK</i>		
<i>Mini-Module FT2232H</i>		
<i>Olimex ARM-USB-OCD</i>		
<i>Olimex ARM-USB-OCD-H</i>		
<i>Olimex ARM-USB-TINY-H</i>		
<i>Olimex ARM-USB-TINY</i>		
<i>TIAO USB Multi-Protocol Adapter (TUMPA)</i>		

Frameworks

Name	Description
<i>Arduinio</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP-IDF</i>	Espressif IoT Development Framework. Official development framework for ESP32.

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.

Microcontroller	ESP32
Frequency	240MHz
Flash	4MB
RAM	320KB
Vendor	Hornbill

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.

Compatible Tools	On-board	Default
<i>ESP-Prog</i>		Yes
<i>oddWires IOT-Bus JTAG</i>		
<i>J-LINK</i>		
<i>Mini-Module FT2232H</i>		
<i>Olimex ARM-USB-OCD</i>		
<i>Olimex ARM-USB-OCD-H</i>		
<i>Olimex ARM-USB-TINY-H</i>		
<i>Olimex ARM-USB-TINY</i>		
<i>TIAO USB Multi-Protocol Adapter (TUMPA)</i>		

Frameworks

Name	Description
<i>Ar-duino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP-IDF</i>	Espressif IoT Development Framework. Official development framework for ESP32.

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.

Microcontroller	ESP32
Frequency	240MHz
Flash	4MB
RAM	320KB
Vendor	IntoRobot

Configuration

Please use `intorobot` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[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.

```
[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:

```
[env:intorobot]
platform = espressif32
board = intorobot

upload_protocol = esptool
```

Debugging

PIO Unified Debugger currently does not support IntoRobot Fig board.

Frameworks

Name	Description
<i>Ar-duino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP-IDF</i>	Espressif IoT Development Framework. Official development framework for ESP32.

IoTaaP Magnolia

Contents

- *IoTaaP Magnolia*
 - *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	MVT Solutions

Configuration

Please use `iotaap_magnolia` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:iotaap_magnolia]
platform = espressif32
board = iotaap_magnolia
```

You can override default IoTaaP Magnolia settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `iotaap_magnolia.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:iotaap_magnolia]
platform = espressif32
board = iotaap_magnolia

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

Uploading

IoTaaP Magnolia supports the next uploading protocols:

- esp-prog
- espota
- esptool
- iot-bus-jtag
- jlink
- minimodule
- olimex-arm-usb-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:iotaap_magnolia]
platform = espressif32
board = iotaap_magnolia

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*).

IoTaaP Magnolia does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

Compatible Tools	On-board	Default
<i>ESP-Prog</i>		Yes
<i>oddWires IOT-Bus JTAG</i>		
<i>J-LINK</i>		
<i>Mini-Module FT2232H</i>		
<i>Olimex ARM-USB-OCD</i>		
<i>Olimex ARM-USB-OCD-H</i>		
<i>Olimex ARM-USB-TINY-H</i>		
<i>Olimex ARM-USB-TINY</i>		
<i>TIAO USB Multi-Protocol Adapter (TUMPA)</i>		

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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.

Microcontroller	ESP32
Frequency	240MHz
Flash	4MB
RAM	320KB
Vendor	M5Stack

Configuration

Please use m5stack-core-esp32 ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[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.

```
[env:m5stack-core-esp32]
platform = espressif32
board = m5stack-core-esp32

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

Uploading

M5Stack Core ESP32 supports the next uploading protocols:

- espota
- esptool

Default protocol is esptool

You can change upload protocol using `upload_protocol` option:

```
[env:m5stack-core-esp32]
platform = espressif32
board = m5stack-core-esp32

upload_protocol = esptool
```

Debugging

PIO Unified Debugger currently does not support M5Stack Core ESP32 board.

Frameworks

Name	Description
<code>Ar-duino</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<code>ESP-IDF</code>	Espressif IoT Development Framework. Official development framework for ESP32.

M5Stack FIRE

Contents

- *M5Stack FIRE*
 - *Hardware*
 - *Configuration*
 - *Uploading*
 - *Debugging*
 - *Frameworks*

Hardware

Platform [Espressif 32](#): Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

Microcontroller	ESP32
Frequency	240MHz
Flash	16MB
RAM	6.25MB
Vendor	M5Stack

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 = 24000000L
```

Uploading

M5Stack FIRE supports the next uploading protocols:

- espota
- esptool

Default protocol is esptool

You can change upload protocol using *upload_protocol* option:

```
[env:m5stack-fire]
platform = espressif32
board = m5stack-fire

upload_protocol = esptool
```

Debugging

PIO Unified Debugger currently does not support M5Stack FIRE board.

Frameworks

Name	Description
<i>Arduin</i> o	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP-IDF</i>	Espressif IoT Development Framework. Official development framework for ESP32.

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.

Microcontroller	ESP32
Frequency	240MHz
Flash	16MB
RAM	520KB
Vendor	M5Stack

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

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP-IDF</i>	Espressif IoT Development Framework. Official development framework for ESP32.

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.

Microcontroller	ESP32
Frequency	240MHz
Flash	4MB
RAM	320KB
Vendor	M5Stack

Configuration

Please use `m5stick-c` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[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.

```
[env:m5stick-c]
platform = espressif32
board = m5stick-c
```

(continues on next page)

(continued from previous page)

```
; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

Uploading

M5Stick-C supports the next uploading protocols:

- espota
- esptool

Default protocol is esptool

You can change upload protocol using *upload_protocol* option:

```
[env:m5stick-c]
platform = espressif32
board = m5stick-c

upload_protocol = esptool
```

Debugging

PIO Unified Debugger currently does not support M5Stick-C board.

Frameworks

Name	Description
<i>Ardu</i> <i>duino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP-IDF</i>	Espressif IoT Development Framework. Official development framework for ESP32.

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.

Microcontroller	ESP32
Frequency	240MHz
Flash	4MB
RAM	320KB
Vendor	MH-ET Live

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.

Compatible Tools	On-board	Default
<i>ESP-Prog</i>		Yes
<i>oddWires IOT-Bus JTAG</i>		
<i>J-LINK</i>		
<i>Mini-Module FT2232H</i>		
<i>Olimex ARM-USB-OCD</i>		
<i>Olimex ARM-USB-OCD-H</i>		
<i>Olimex ARM-USB-TINY-H</i>		
<i>Olimex ARM-USB-TINY</i>		
<i>TIAO USB Multi-Protocol Adapter (TUMPA)</i>		

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP-IDF</i>	Espressif IoT Development Framework. Official development framework for ESP32.

MH ET LIVE ESP32MiniKit

Contents

- *MH ET LIVE ESP32MiniKit*
 - *Hardware*
 - *Configuration*
 - *Uploading*
 - *Debugging*
 - *Frameworks*

Hardware

Platform *Espressif 32*: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

Microcontroller	ESP32
Frequency	240MHz
Flash	4MB
RAM	320KB
Vendor	MH-ET Live

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 = 24000000L
```

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.

Compatible Tools	On-board	Default
<i>ESP-Prog</i>		Yes
<i>oddWires IOT-Bus JTAG</i>		
<i>J-LINK</i>		
<i>Mini-Module FT2232H</i>		
<i>Olimex ARM-USB-OCD</i>		
<i>Olimex ARM-USB-OCD-H</i>		
<i>Olimex ARM-USB-TINY-H</i>		
<i>Olimex ARM-USB-TINY</i>		
<i>TIAO USB Multi-Protocol Adapter (TUMPA)</i>		

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP-IDF</i>	Espressif IoT Development Framework. Official development framework for ESP32.

MagicBit

Contents

- *MagicBit*
 - *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	Magicblocks.io

Configuration

Please use `magicbit` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:magicbit]
platform = espressif32
board = magicbit
```

You can override default MagicBit settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `magicbit.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:magicbit]
platform = espressif32
board = magicbit
```

(continues on next page)

(continued from previous page)

```
; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

Uploading

MagicBit supports the next uploading protocols:

- espota
- esptool

Default protocol is esptool

You can change upload protocol using *upload_protocol* option:

```
[env:magicbit]
platform = espressif32
board = magicbit

upload_protocol = esptool
```

Debugging

PIO Unified Debugger currently does not support MagicBit board.

Frameworks

Name	Description
<i>Ardu</i> <i>duino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP-IDF</i>	Espressif IoT Development Framework. Official development framework for ESP32.

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.

Microcontroller	ESP32
Frequency	240MHz
Flash	4MB
RAM	320KB
Vendor	MakerAsia

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
```

(continues on next page)

(continued from previous page)

```
upload_protocol = esptool
```

Debugging

PIO Unified Debugger currently does not support MakerAsia Nano32 board.

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP-IDF</i>	Espressif IoT Development Framework. Official development framework for ESP32.
<i>Pumbaa</i>	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.
<i>Simba</i>	Simba is an RTOS and build framework. It aims to make embedded programming easy and portable.

Microduino Core ESP32

Contents

- *Microduino Core ESP32*
 - *Hardware*
 - *Configuration*
 - *Uploading*
 - *Debugging*
 - *Frameworks*

Hardware

Platform *Espressif 32*: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

Microcontroller	ESP32
Frequency	240MHz
Flash	4MB
RAM	320KB
Vendor	Microduino

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:

```
[env:microduino-core-esp32]
platform = espressif32
board = microduino-core-esp32

upload_protocol = esptool
```

Debugging

PIO Unified Debugger currently does not support Microduino Core ESP32 board.

Frameworks

Name	Description
<code>Ar-duino</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<code>ESP-IDF</code>	Espressif IoT Development Framework. Official development framework for ESP32.

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.

Microcontroller	ESP32
Frequency	240MHz
Flash	4MB
RAM	320KB
Vendor	Aiyarafun

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.

Compatible Tools	On-board	Default
<i>ESP-Prog</i>		Yes
<i>oddWires IOT-Bus JTAG</i>		
<i>J-LINK</i>		
<i>Mini-Module FT2232H</i>		
<i>Olimex ARM-USB-OCD</i>		
<i>Olimex ARM-USB-OCD-H</i>		
<i>Olimex ARM-USB-TINY-H</i>		
<i>Olimex ARM-USB-TINY</i>		
<i>TIAO USB Multi-Protocol Adapter (TUMPA)</i>		

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP-IDF</i>	Espressif IoT Development Framework. Official development framework for ESP32.

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.

Microcontroller	ESP32
Frequency	240MHz
Flash	4MB
RAM	320KB
Vendor	NodeMCU

Configuration

Please use nodemcu-32s ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[env:nodemcu-32s]
platform = espressif32
board = nodemcu-32s
```

You can override default NodeMCU-32S settings per build environment using *board_**** option, where *** is a JSON object path from board manifest *nodemcu-32s.json*. For example, *board_build.mcu*, *board_build.f_cpu*, etc.

```
[env:nodemcu-32s]
platform = espressif32
board = nodemcu-32s

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

Uploading

NodeMCU-32S supports the next uploading protocols:

- esp-prog
- espota
- esptool
- iot-bus-jtag
- jlink
- minimodule
- olimex-arm-usb-ocd
- olimex-arm-usb-ocd-h
- olimex-arm-usb-tiny-h
- olimex-jtag-tiny
- tumpa

Default protocol is esptool

You can change upload protocol using *upload_protocol* option:

```
[env:nodemcu-32s]
platform = espressif32
board = nodemcu-32s

upload_protocol = esptool
```

Debugging

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

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for 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.

Compatible Tools	On-board	Default
<i>ESP-Prog</i>		Yes
<i>oddWires IOT-Bus JTAG</i>		
<i>J-LINK</i>		
<i>Mini-Module FT2232H</i>		
<i>Olimex ARM-USB-OCD</i>		
<i>Olimex ARM-USB-OCD-H</i>		
<i>Olimex ARM-USB-TINY-H</i>		
<i>Olimex ARM-USB-TINY</i>		
<i>TIAO USB Multi-Protocol Adapter (TUMPA)</i>		

Frameworks

Name	Description
<i>Arduino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP-IDF</i>	Espressif IoT Development Framework. Official development framework for ESP32.

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.

Microcontroller	ESP32
Frequency	240MHz
Flash	16MB
RAM	320KB
Vendor	<i>Noduino</i>

Configuration

Please use quantum ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[env:quantum]
platform = espressif32
board = quantum
```

You can override default Noduino Quantum settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `quantum.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:quantum]
platform = espressif32
board = quantum

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

Uploading

Noduino Quantum supports the next uploading protocols:

- espota
- esptool

Default protocol is `esptool`

You can change upload protocol using `upload_protocol` option:

```
[env:quantum]
platform = espressif32
board = quantum

upload_protocol = esptool
```

Debugging

PIO Unified Debugger currently does not support Noduino Quantum board.

Frameworks

Name	Description
<i>Ardu</i> <i>duino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP-IDF</i>	Espressif IoT Development Framework. Official development framework for ESP32.

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.

Microcontroller	ESP32
Frequency	240MHz
Flash	16MB
RAM	320KB
Vendor	Hardkernel

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 = 24000000L
```

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

Name	Description
<i>Arduin</i> o	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP-IDF</i>	Espressif IoT Development Framework. Official development framework for ESP32.

OLIMEX ESP32-DevKit-LiPo

Contents

- *OLIMEX ESP32-DevKit-LiPo*
 - *Hardware*
 - *Configuration*
 - *Uploading*
 - *Debugging*
 - *Frameworks*

Hardware

Platform *Espressif 32*: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

Microcontroller	ESP32
Frequency	240MHz
Flash	4MB
RAM	320KB
Vendor	OLIMEX

Configuration

Please use `esp32-devkitlipo` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:esp32-devkitlipo]
platform = espressif32
board = esp32-devkitlipo
```

You can override default OLIMEX ESP32-DevKit-LiPo settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `esp32-devkitlipo.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:esp32-devkitlipo]
platform = espressif32
board = esp32-devkitlipo

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

Uploading

OLIMEX ESP32-DevKit-LiPo supports the next uploading protocols:

- `esp-prog`
- `espota`
- `esptool`
- `iot-bus-jtag`
- `jlink`
- `minimodule`
- `olimex-arm-usb-ocd`
- `olimex-arm-usb-ocd-h`
- `olimex-arm-usb-tiny-h`
- `olimex-jtag-tiny`
- `tumpa`

Default protocol is `esptool`

You can change upload protocol using `upload_protocol` option:

```
[env:esp32-devkitlipo]
platform = espressif32
board = esp32-devkitlipo

upload_protocol = esptool
```

Debugging

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

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using *debug_tool* option in “*platformio.ini*” (*Project Configuration File*).

OLIMEX ESP32-DevKit-LiPo does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

Compatible Tools	On-board	Default
<i>ESP-Prog</i>		Yes
<i>oddWires IOT-Bus JTAG</i>		
<i>J-LINK</i>		
<i>Mini-Module FT2232H</i>		
<i>Olimex ARM-USB-OCD</i>		
<i>Olimex ARM-USB-OCD-H</i>		
<i>Olimex ARM-USB-TINY-H</i>		
<i>Olimex ARM-USB-TINY</i>		
<i>TIAO USB Multi-Protocol Adapter (TUMPA)</i>		

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP-IDF</i>	Espressif IoT Development Framework. Official development framework for ESP32.

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.

Microcontroller	ESP32
Frequency	240MHz
Flash	4MB
RAM	320KB
Vendor	OLIMEX

Configuration

Please use esp32-evb ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[env:esp32-evb]
platform = espressif32
board = esp32-evb
```

You can override default OLIMEX ESP32-EVB settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `esp32-evb.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:esp32-evb]
platform = espressif32
board = esp32-evb

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

Uploading

OLIMEX ESP32-EVB supports the next uploading protocols:

- esp-prog
- espota
- esptool
- iot-bus-jtag
- jlink

- minimodule
- olimex-arm-usb-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-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.

Compatible Tools	On-board	Default
<i>ESP-Prog</i>		Yes
<i>oddWires IOT-Bus JTAG</i>		
<i>J-LINK</i>		
<i>Mini-Module FT2232H</i>		
<i>Olimex ARM-USB-OCD</i>		
<i>Olimex ARM-USB-OCD-H</i>		
<i>Olimex ARM-USB-TINY-H</i>		
<i>Olimex ARM-USB-TINY</i>		
<i>TIAO USB Multi-Protocol Adapter (TUMPA)</i>		

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP-IDF</i>	Espressif IoT Development Framework. Official development framework for ESP32.

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.

Microcontroller	ESP32
Frequency	240MHz
Flash	4MB
RAM	320KB
Vendor	OLIMEX

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.

Compatible Tools	On-board	Default
<i>ESP-Prog</i>		Yes
<i>oddWires IOT-Bus JTAG</i>		
<i>J-LINK</i>		
<i>Mini-Module FT2232H</i>		
<i>Olimex ARM-USB-OCD</i>		
<i>Olimex ARM-USB-OCD-H</i>		
<i>Olimex ARM-USB-TINY-H</i>		
<i>Olimex ARM-USB-TINY</i>		
<i>TIAO USB Multi-Protocol Adapter (TUMPA)</i>		

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP-IDF</i>	Espressif IoT Development Framework. Official development framework for ESP32.

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.

Microcontroller	ESP32
Frequency	240MHz
Flash	4MB
RAM	320KB
Vendor	OLIMEX

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

Name	Description
<code>Ar-duino</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<code>ESP-IDF</code>	Espressif IoT Development Framework. Official development framework for ESP32.

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.

Microcontroller	ESP32
Frequency	240MHz
Flash	4MB
RAM	320KB
Vendor	OLIMEX

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 = 24000000L
```

Uploading

OLIMEX ESP32-PoE supports the next uploading protocols:

- espota
- esptool

Default protocol is esptool

You can change upload protocol using *upload_protocol* option:

```
[env:esp32-poe]
platform = espressif32
board = esp32-poe

upload_protocol = esptool
```

Debugging

PIO Unified Debugger currently does not support OLIMEX ESP32-PoE board.

Frameworks

Name	Description
<i>Arduin</i> o	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP-IDF</i>	Espressif IoT Development Framework. Official development framework for ESP32.

OLIMEX ESP32-PoE-ISO

Contents

- *OLIMEX ESP32-PoE-ISO*
 - *Hardware*
 - *Configuration*
 - *Uploading*
 - *Debugging*
 - *Frameworks*

Hardware

Platform *Espressif 32*: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

Microcontroller	ESP32
Frequency	240MHz
Flash	4MB
RAM	320KB
Vendor	OLIMEX

Configuration

Please use `esp32-poe-iso` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:esp32-poe-iso]
platform = espressif32
board = esp32-poe-iso
```

You can override default OLIMEX ESP32-PoE-ISO settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `esp32-poe-iso.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:esp32-poe-iso]
platform = espressif32
board = esp32-poe-iso

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

Uploading

OLIMEX ESP32-PoE-ISO supports the next uploading protocols:

- `espota`
- `esptool`

Default protocol is `esptool`

You can change upload protocol using `upload_protocol` option:

```
[env:esp32-poe-iso]
platform = espressif32
board = esp32-poe-iso

upload_protocol = esptool
```

Debugging

PIO Unified Debugger currently does not support OLIMEX ESP32-PoE-ISO board.

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP-IDF</i>	Espressif IoT Development Framework. Official development framework for ESP32.

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.

Microcontroller	ESP32
Frequency	240MHz
Flash	4MB
RAM	320KB
Vendor	OROCA

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

Name	Description
<i>Ar-duino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP-IDF</i>	Espressif IoT Development Framework. Official development framework for ESP32.

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.

Microcontroller	ESP32
Frequency	240MHz
Flash	4MB
RAM	320KB
Vendor	Onehorse

Configuration

Please use `onehorse32dev` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:onehorse32dev]
platform = espressif32
board = onehorse32dev
```

You can override default Onehorse ESP32 Dev Module settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `onehorse32dev.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:onehorse32dev]
platform = espressif32
board = onehorse32dev

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

Uploading

Onehorse ESP32 Dev Module supports the next uploading protocols:

- `espota`
- `esptool`

Default protocol is `esptool`

You can change upload protocol using `upload_protocol` option:

```
[env:onehorse32dev]
platform = espressif32
board = onehorse32dev

upload_protocol = esptool
```

Debugging

PIO Unified Debugger currently does not support Onehorse ESP32 Dev Module board.

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP-IDF</i>	Espressif IoT Development Framework. Official development framework for ESP32.

Pycom GPy

Contents

- *Pycom GPy*
 - *Hardware*
 - *Configuration*
 - *Uploading*
 - *Debugging*
 - *Frameworks*

Hardware

Platform *Espressif 32*: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

Microcontroller	ESP32
Frequency	240MHz
Flash	4MB
RAM	320KB
Vendor	Pycom Ltd.

Configuration

Please use `pycom_gpy` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:pycom_gpy]
platform = espressif32
board = pycom_gpy
```

You can override default Pycom GPy settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `pycom_gpy.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:pycom_gpy]
platform = espressif32
board = pycom_gpy

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

Uploading

Pycom GPy supports the next uploading protocols:

- `espota`
- `esptool`

Default protocol is `esptool`

You can change upload protocol using `upload_protocol` option:

```
[env:pycom_gpy]
platform = espressif32
board = pycom_gpy

upload_protocol = esptool
```

Debugging

PIO Unified Debugger currently does not support Pycom GPy board.

Frameworks

Name	Description
<code>Ar-duino</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<code>ESP-IDF</code>	Espressif IoT Development Framework. Official development framework for ESP32.

Pycom LoPy

Contents

- *Pycom LoPy*
 - *Hardware*
 - *Configuration*
 - *Uploading*
 - *Debugging*
 - *Frameworks*

Hardware

Platform [Espressif 32](#): Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

Microcontroller	ESP32
Frequency	240MHz
Flash	4MB
RAM	320KB
Vendor	Pycom Ltd.

Configuration

Please use `lopy` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[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.

```
[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:

```
[env:lopyp]
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.

Compatible Tools	On-board	Default
<i>ESP-Prog</i>		Yes
<i>oddWires IOT-Bus JTAG</i>		
<i>J-LINK</i>		
<i>Mini-Module FT2232H</i>		
<i>Olimex ARM-USB-OCD</i>		
<i>Olimex ARM-USB-OCD-H</i>		
<i>Olimex ARM-USB-TINY-H</i>		
<i>Olimex ARM-USB-TINY</i>		
<i>TIAO USB Multi-Protocol Adapter (TUMPA)</i>		

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP-IDF</i>	Espressif IoT Development Framework. Official development framework for ESP32.

Pycom LoPy4

Contents

- *Pycom LoPy4*
 - *Hardware*
 - *Configuration*
 - *Uploading*
 - *Debugging*
 - *Frameworks*

Hardware

Platform *Espressif 32*: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

Microcontroller	ESP32
Frequency	240MHz
Flash	4MB
RAM	1.25MB
Vendor	Pycom Ltd.

Configuration

Please use `lopy4` ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[env:lopy4]
platform = espressif32
board = lopy4
```

You can override default Pycom LoPy4 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `lopy4.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[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.

Compatible Tools	On-board	Default
<i>ESP-Prog</i>		Yes
<i>oddWires IOT-Bus JTAG</i>		
<i>J-LINK</i>		
<i>Mini-Module FT2232H</i>		
<i>Olimex ARM-USB-OCD</i>		
<i>Olimex ARM-USB-OCD-H</i>		
<i>Olimex ARM-USB-TINY-H</i>		
<i>Olimex ARM-USB-TINY</i>		
<i>TIAO USB Multi-Protocol Adapter (TUMPA)</i>		

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP-IDF</i>	Espressif IoT Development Framework. Official development framework for ESP32.

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.

Microcontroller	ESP32
Frequency	240MHz
Flash	4MB
RAM	320KB
Vendor	Silicognition

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.

```
[env:wesp32]
platform = espressif32
board = wesp32

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

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:

```
[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.

Compatible Tools	On-board	Default
<i>ESP-Prog</i>		Yes
<i>oddWires IOT-Bus JTAG</i>		
<i>J-LINK</i>		
<i>Mini-Module FT2232H</i>		
<i>Olimex ARM-USB-OCD</i>		
<i>Olimex ARM-USB-OCD-H</i>		
<i>Olimex ARM-USB-TINY-H</i>		
<i>Olimex ARM-USB-TINY</i>		
<i>TIAO USB Multi-Protocol Adapter (TUMPA)</i>		

Frameworks

Name	Description
<i>Ar-duino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP-IDF</i>	Espressif IoT Development Framework. Official development framework for ESP32.

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.

Microcontroller	ESP32
Frequency	240MHz
Flash	4MB
RAM	320KB
Vendor	SparkFun Electronics

Configuration

Please use `esp32thing` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:esp32thing]
platform = espressif32
board = esp32thing
```

You can override default SparkFun ESP32 Thing settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `esp32thing.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:esp32thing]
platform = espressif32
board = esp32thing

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

Uploading

SparkFun ESP32 Thing supports the next uploading protocols:

- esp-prog
- espota
- esptool
- iot-bus-jtag
- jlink
- minimodule
- olimex-arm-usb-ocd
- olimex-arm-usb-ocd-h
- olimex-arm-usb-tiny-h
- olimex-jtag-tiny

- tumpa

Default protocol is esptool

You can change upload protocol using *upload_protocol* option:

```
[env:esp32thing]
platform = espressif32
board = esp32thing

upload_protocol = esptool
```

Debugging

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

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for 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.

Compatible Tools	On-board	Default
<i>ESP-Prog</i>		Yes
<i>oddWires IOT-Bus JTAG</i>		
<i>J-LINK</i>		
<i>Mini-Module FT2232H</i>		
<i>Olimex ARM-USB-OCD</i>		
<i>Olimex ARM-USB-OCD-H</i>		
<i>Olimex ARM-USB-TINY-H</i>		
<i>Olimex ARM-USB-TINY</i>		
<i>TIAO USB Multi-Protocol Adapter (TUMPA)</i>		

Frameworks

Name	Description
<i>Ar-duino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP-IDF</i>	Espressif IoT Development Framework. Official development framework for ESP32.

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.

Microcontroller	ESP32
Frequency	240MHz
Flash	4MB
RAM	320KB
Vendor	SparkFun

Configuration

Please use `sparkfun_lora_gateway_1-channel` ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[env:sparkfun_lora_gateway_1-channel]
platform = espressif32
board = sparkfun_lora_gateway_1-channel
```

You can override default SparkFun LoRa Gateway 1-Channel settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `sparkfun_lora_gateway_1-channel.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:sparkfun_lora_gateway_1-channel]
platform = espressif32
board = sparkfun_lora_gateway_1-channel

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

Uploading

SparkFun LoRa Gateway 1-Channel supports the next uploading protocols:

- `esp-prog`

- espota
- esptool
- iot-bus-jtag
- jlink
- minimodule
- olimex-arm-usb-ocd
- olimex-arm-usb-ocd-h
- olimex-arm-usb-tiny-h
- olimex-jtag-tiny
- tumpa

Default protocol is esptool

You can change upload protocol using *upload_protocol* option:

```
[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.

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

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP-IDF</i>	Espressif IoT Development Framework. Official development framework for ESP32.

TTGO LoRa32-OLED V1

Contents

- *TTGO LoRa32-OLED V1*
 - *Hardware*
 - *Configuration*
 - *Uploading*
 - *Debugging*
 - *Frameworks*

Hardware

Platform *Espressif 32*: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

Microcontroller	ESP32
Frequency	240MHz
Flash	4MB
RAM	320KB
Vendor	TTGO

Configuration

Please use `ttgo-lora32-v1` ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[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.

```
[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.

Compatible Tools	On-board	Default
<i>ESP-Prog</i>		Yes
<i>oddWires IOT-Bus JTAG</i>		
<i>J-LINK</i>		
<i>Mini-Module FT2232H</i>		
<i>Olimex ARM-USB-OCD</i>		
<i>Olimex ARM-USB-OCD-H</i>		
<i>Olimex ARM-USB-TINY-H</i>		
<i>Olimex ARM-USB-TINY</i>		
<i>TIAO USB Multi-Protocol Adapter (TUMPA)</i>		

Frameworks

Name	Description
<i>Arduinio</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP-IDF</i>	Espressif IoT Development Framework. Official development framework for ESP32.

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.

Microcontroller	ESP32
Frequency	240MHz
Flash	4MB
RAM	1.25MB
Vendor	TTGO

Configuration

Please use ttgo-t-beam ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[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.

```
[env:ttgo-t-beam]
platform = espressif32
board = ttgo-t-beam

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

Uploading

TTGO T-Beam supports the next uploading protocols:

- esp-prog
- espota
- esptool
- iot-bus-jtag
- jlink
- minimodule
- olimex-arm-usb-ocd
- olimex-arm-usb-ocd-h
- olimex-arm-usb-tiny-h
- olimex-jtag-tiny
- tumpa

Default protocol is esptool

You can change upload protocol using `upload_protocol` option:

```
[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.

Compatible Tools	On-board	Default
<i>ESP-Prog</i>		Yes
<i>oddWires IOT-Bus JTAG</i>		
<i>J-LINK</i>		
<i>Mini-Module FT2232H</i>		
<i>Olimex ARM-USB-OCD</i>		
<i>Olimex ARM-USB-OCD-H</i>		
<i>Olimex ARM-USB-TINY-H</i>		
<i>Olimex ARM-USB-TINY</i>		
<i>TIAO USB Multi-Protocol Adapter (TUMPA)</i>		

Frameworks

Name	Description
<i>Ar-duino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP-IDF</i>	Espressif IoT Development Framework. Official development framework for ESP32.

TTGO T-Watch

Contents

- *TTGO T-Watch*
 - *Hardware*
 - *Configuration*
 - *Uploading*
 - *Debugging*
 - *Frameworks*

Hardware

Platform *Espressif 32*: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

Microcontroller	ESP32
Frequency	240MHz
Flash	16MB
RAM	320KB
Vendor	TTGO

Configuration

Please use `ttgo-t-watch` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:ttgo-t-watch]
platform = espressif32
board = ttgo-t-watch
```

You can override default TTGO T-Watch settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `ttgo-t-watch.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:ttgo-t-watch]
platform = espressif32
board = ttgo-t-watch

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

Uploading

TTGO T-Watch supports the next uploading protocols:

- `espota`
- `esptool`

Default protocol is `esptool`

You can change upload protocol using `upload_protocol` option:

```
[env:ttgo-t-watch]
platform = espressif32
board = ttgo-t-watch

upload_protocol = esptool
```

Debugging

PIO Unified Debugger currently does not support TTGO T-Watch board.

Frameworks

Name	Description
<i>Ar-duino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP-IDF</i>	Espressif IoT Development Framework. Official development framework for ESP32.

TTGO T1

Contents

- *TTGO T1*
 - *Hardware*
 - *Configuration*
 - *Uploading*
 - *Debugging*
 - *Frameworks*

Hardware

Platform *Espressif 32*: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

Microcontroller	ESP32
Frequency	240MHz
Flash	4MB
RAM	320KB
Vendor	TTGO

Configuration

Please use `ttgo-t1` ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[env:ttgo-t1]
platform = espressif32
board = ttgo-t1
```

You can override default TTGO T1 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `ttgo-t1.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:ttgo-t1]
platform = espressif32
board = ttgo-t1

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 24000000L
```

Uploading

TTGO T1 supports the next uploading protocols:

- esp-prog
- espota
- esptool
- iot-bus-jtag
- jlink
- minimodule
- olimex-arm-usb-ocd
- olimex-arm-usb-ocd-h
- olimex-arm-usb-tiny-h
- olimex-jtag-tiny
- tumpa

Default protocol is esptool

You can change upload protocol using `upload_protocol` option:

```
[env:ttgo-t1]
platform = espressif32
board = ttgo-t1

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 T1 does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

Compatible Tools	On-board	Default
<i>ESP-Prog</i>		Yes
<i>oddWires IOT-Bus JTAG</i>		
<i>J-LINK</i>		
<i>Mini-Module FT2232H</i>		
<i>Olimex ARM-USB-OCD</i>		
<i>Olimex ARM-USB-OCD-H</i>		
<i>Olimex ARM-USB-TINY-H</i>		
<i>Olimex ARM-USB-TINY</i>		
<i>TIAO USB Multi-Protocol Adapter (TUMPA)</i>		

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP-IDF</i>	Espressif IoT Development Framework. Official development framework for ESP32.

TinyPICO

Contents

- *TinyPICO*
 - *Hardware*
 - *Configuration*
 - *Uploading*
 - *Debugging*
 - *Frameworks*

Hardware

Platform *Espressif 32*: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

Microcontroller	ESP32
Frequency	240MHz
Flash	4MB
RAM	320KB
Vendor	<i>TinyPICO</i>

Configuration

Please use `tinypico` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:tinypico]
platform = espressif32
board = tinypico
```

You can override default TinyPICO settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `tinypico.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:tinypico]
platform = espressif32
board = tinypico

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

Uploading

TinyPICO supports the next uploading protocols:

- `espota`
- `esptool`

Default protocol is `esptool`

You can change upload protocol using `upload_protocol` option:

```
[env:tinypico]
platform = espressif32
board = tinypico

upload_protocol = esptool
```

Debugging

PIO Unified Debugger currently does not support TinyPICO board.

Frameworks

Name	Description
<code>Ar-duino</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<code>ESP-IDF</code>	Espressif IoT Development Framework. Official development framework for ESP32.

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.

Microcontroller	ESP32
Frequency	240MHz
Flash	4MB
RAM	320KB
Vendor	Turta

Configuration

Please use `turta_iot_node` ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[env:turta_iot_node]
platform = espressif32
board = turta_iot_node
```

You can override default Turta IoT Node settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `turta_iot_node.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:turta_iot_node]
platform = espressif32
board = turta_iot_node

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 24000000L
```

Uploading

Turta IoT Node supports the next uploading protocols:

- espota
- esptool

Default protocol is esptool

You can change upload protocol using *upload_protocol* option:

```
[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

Name	Description
<i>Arduin</i> o	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP-IDF</i>	Espressif IoT Development Framework. Official development framework for ESP32.

VintLabs ESP32 Devkit

Contents

- *VintLabs ESP32 Devkit*
 - *Hardware*
 - *Configuration*
 - *Uploading*
 - *Debugging*
 - *Frameworks*

Hardware

Platform *Espressif 32*: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

Microcontroller	ESP32
Frequency	240MHz
Flash	4MB
RAM	320KB
Vendor	VintLabs

Configuration

Please use `vintlabs-devkit-v1` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:vintlabs-devkit-v1]
platform = espressif32
board = vintlabs-devkit-v1
```

You can override default VintLabs ESP32 Devkit settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `vintlabs-devkit-v1.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:vintlabs-devkit-v1]
platform = espressif32
board = vintlabs-devkit-v1

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

Uploading

VintLabs ESP32 Devkit supports the next uploading protocols:

- esp-prog
- espota
- esptool
- iot-bus-jtag
- jlink
- minimodule
- olimex-arm-usb-ocd
- olimex-arm-usb-ocd-h
- olimex-arm-usb-tiny-h
- olimex-jtag-tiny
- tumpa

Default protocol is `esptool`

You can change upload protocol using `upload_protocol` option:

```
[env:vintlabs-devkit-v1]
platform = espressif32
board = vintlabs-devkit-v1

upload_protocol = esptool
```

Debugging

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

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using *debug_tool* option in “*platformio.ini*” (*Project Configuration File*).

VintLabs ESP32 Devkit does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

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

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.
ESP-IDF	Espressif IoT Development Framework. Official development framework for ESP32.

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.

Microcontroller	ESP32
Frequency	240MHz
Flash	4MB
RAM	320KB
Vendor	WEMOS

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:

- esp-prog
- espota
- esptool
- iot-bus-jtag
- jlink

- minimodule
- olimex-arm-usb-ocd
- olimex-arm-usb-ocd-h
- olimex-arm-usb-tiny-h
- olimex-jtag-tiny
- tumpa

Default protocol is esptool

You can change upload protocol using *upload_protocol* option:

```
[env:lolin_d32]
platform = espressif32
board = lolin_d32

upload_protocol = esptool
```

Debugging

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

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for 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.

Compatible Tools	On-board	Default
<i>ESP-Prog</i>		Yes
<i>oddWires IOT-Bus JTAG</i>		
<i>J-LINK</i>		
<i>Mini-Module FT2232H</i>		
<i>Olimex ARM-USB-OCD</i>		
<i>Olimex ARM-USB-OCD-H</i>		
<i>Olimex ARM-USB-TINY-H</i>		
<i>Olimex ARM-USB-TINY</i>		
<i>TIAO USB Multi-Protocol Adapter (TUMPA)</i>		

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP-IDF</i>	Espressif IoT Development Framework. Official development framework for ESP32.

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.

Microcontroller	ESP32
Frequency	240MHz
Flash	4MB
RAM	320KB
Vendor	WEMOS

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.

Compatible Tools	On-board	Default
<i>ESP-Prog</i>		Yes
<i>oddWires IOT-Bus JTAG</i>		
<i>J-LINK</i>		
<i>Mini-Module FT2232H</i>		
<i>Olimex ARM-USB-OCD</i>		
<i>Olimex ARM-USB-OCD-H</i>		
<i>Olimex ARM-USB-TINY-H</i>		
<i>Olimex ARM-USB-TINY</i>		
<i>TIAO USB Multi-Protocol Adapter (TUMPA)</i>		

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP-IDF</i>	Espressif IoT Development Framework. Official development framework for ESP32.

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.

Microcontroller	ESP32
Frequency	240MHz
Flash	4MB
RAM	320KB
Vendor	WEMOS

Configuration

Please use `lolin32` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[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.

```
[env:lolin32]
platform = espressif32
board = lolin32

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 24000000L
```

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:

```
[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.

Compatible Tools	On-board	Default
<i>ESP-Prog</i>		Yes
<i>oddWires IOT-Bus JTAG</i>		
<i>J-LINK</i>		
<i>Mini-Module FT2232H</i>		
<i>Olimex ARM-USB-OCD</i>		
<i>Olimex ARM-USB-OCD-H</i>		
<i>Olimex ARM-USB-TINY-H</i>		
<i>Olimex ARM-USB-TINY</i>		
<i>TIAO USB Multi-Protocol Adapter (TUMPA)</i>		

Frameworks

Name	Description
<i>Ar-duino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP-IDF</i>	Espressif IoT Development Framework. Official development framework for ESP32.

WeMos D1 MINI ESP32

Contents

- *WeMos D1 MINI ESP32*
 - *Hardware*
 - *Configuration*
 - *Uploading*
 - *Debugging*
 - *Frameworks*

Hardware

Platform [Espressif 32](#): Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

Microcontroller	ESP32
Frequency	240MHz
Flash	4MB
RAM	320KB
Vendor	WEMOS

Configuration

Please use `wemos_d1_mini32` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:wemos_d1_mini32]
platform = espressif32
board = wemos_d1_mini32
```

You can override default WeMos D1 MINI ESP32 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `wemos_d1_mini32.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:wemos_d1_mini32]
platform = espressif32
board = wemos_d1_mini32

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

Uploading

WeMos D1 MINI ESP32 supports the next uploading protocols:

- esp-prog
- espota
- esptool
- iot-bus-jtag
- jlink
- minimodule
- olimex-arm-usb-ocd
- olimex-arm-usb-ocd-h
- olimex-arm-usb-tiny-h
- olimex-jtag-tiny

- tumpa

Default protocol is esptool

You can change upload protocol using *upload_protocol* option:

```
[env:wemos_d1_mini32]
platform = espressif32
board = wemos_d1_mini32

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 D1 MINI ESP32 does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

Compatible Tools	On-board	Default
<i>ESP-Prog</i>		Yes
<i>oddWires IOT-Bus JTAG</i>		
<i>J-LINK</i>		
<i>Mini-Module FT2232H</i>		
<i>Olimex ARM-USB-OCD</i>		
<i>Olimex ARM-USB-OCD-H</i>		
<i>Olimex ARM-USB-TINY-H</i>		
<i>Olimex ARM-USB-TINY</i>		
<i>TIAO USB Multi-Protocol Adapter (TUMPA)</i>		

Frameworks

Name	Description
<i>Ar-duino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP-IDF</i>	Espressif IoT Development Framework. Official development framework for ESP32.

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.

Microcontroller	ESP32
Frequency	240MHz
Flash	4MB
RAM	320KB
Vendor	WEMOS

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-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: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.

Compatible Tools	On-board	Default
<i>ESP-Prog</i>		Yes
<i>oddWires IOT-Bus JTAG</i>		
<i>J-LINK</i>		
<i>Mini-Module FT2232H</i>		
<i>Olimex ARM-USB-OCD</i>		
<i>Olimex ARM-USB-OCD-H</i>		
<i>Olimex ARM-USB-TINY-H</i>		
<i>Olimex ARM-USB-TINY</i>		
<i>TIAO USB Multi-Protocol Adapter (TUMPA)</i>		

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP-IDF</i>	Espressif IoT Development Framework. Official development framework for ESP32.

Widora AIR

Contents

- *Widora AIR*
 - *Hardware*
 - *Configuration*
 - *Uploading*
 - *Debugging*
 - *Frameworks*

Hardware

Platform *Espressif 32*: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

Microcontroller	ESP32
Frequency	240MHz
Flash	16MB
RAM	320KB
Vendor	Widora

Configuration

Please use `widora-air` ID for `board` option in “`platformio.ini`” (Project Configuration File):

```
[env:widora-air]
platform = espressif32
board = widora-air
```

You can override default Widora AIR settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `widora-air.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:widora-air]
platform = espressif32
board = widora-air
```

(continues on next page)

(continued from previous page)

```
; 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:

```
[env:widora-air]
platform = espressif32
board = widora-air

upload_protocol = esptool
```

Debugging

PIO Unified Debugger currently does not support Widora AIR board.

Frameworks

Name	Description
<i>Ardu</i> <i>duino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP-IDF</i>	Espressif IoT Development Framework. Official development framework for ESP32.

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.

Microcontroller	ESP32
Frequency	240MHz
Flash	4MB
RAM	320KB
Vendor	XinaBox

Configuration

Please use `xinabox_cw02` ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[env:xinabox_cw02]
platform = espressif32
board = xinabox_cw02
```

You can override default XinaBox CW02 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `xinabox_cw02.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:xinabox_cw02]
platform = espressif32
board = xinabox_cw02

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 240000000L
```

Uploading

XinaBox CW02 supports the next uploading protocols:

- esp-prog
- espota
- esptool
- iot-bus-jtag
- jlink
- minimodule
- olimex-arm-usb-ocd

- olimex-arm-usb-ocd-h
- olimex-arm-usb-tiny-h
- olimex-jtag-tiny
- tumpa

Default protocol is esptool

You can change upload protocol using *upload_protocol* option:

```
[env:xinabox_cw02]
platform = espressif32
board = xinabox_cw02

upload_protocol = esptool
```

Debugging

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

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for 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.

Compatible Tools	On-board	Default
<i>ESP-Prog</i>		Yes
<i>oddWires IOT-Bus JTAG</i>		
<i>J-LINK</i>		
<i>Mini-Module FT2232H</i>		
<i>Olimex ARM-USB-OCD</i>		
<i>Olimex ARM-USB-OCD-H</i>		
<i>Olimex ARM-USB-TINY-H</i>		
<i>Olimex ARM-USB-TINY</i>		
<i>TIAO USB Multi-Protocol Adapter (TUMPA)</i>		

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP-IDF</i>	Espressif IoT Development Framework. Official development framework for ESP32.

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.

Microcontroller	ESP32
Frequency	240MHz
Flash	4MB
RAM	320KB
Vendor	oddWires

Configuration

Please use `iotbusio` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:iotbusio]
platform = espressif32
board = iotbusio
```

You can override default oddWires IoT-Bus Io settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `iotbusio.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:iotbusio]
platform = espressif32
board = iotbusio

; change microcontroller
board_build.mcu = esp32

; change MCU frequency
board_build.f_cpu = 24000000L
```

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:

```
[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.

Compatible Tools	On-board	Default
<i>ESP-Prog</i>		
<i>oddWires IOT-Bus JTAG</i>		Yes
<i>J-LINK</i>		
<i>Mini-Module FT2232H</i>		
<i>Olimex ARM-USB-OCD</i>		
<i>Olimex ARM-USB-OCD-H</i>		
<i>Olimex ARM-USB-TINY-H</i>		
<i>Olimex ARM-USB-TINY</i>		
<i>TIAO USB Multi-Protocol Adapter (TUMPA)</i>		

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP-IDF</i>	Espressif IoT Development Framework. Official development framework for ESP32.

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.

Microcontroller	ESP32
Frequency	240MHz
Flash	4MB
RAM	320KB
Vendor	oddWires

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:

```
[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.

Compatible Tools	On-board	Default
<i>ESP-Prog</i>		
<i>oddWires IOT-Bus JTAG</i>		Yes
<i>J-LINK</i>		
<i>Mini-Module FT2232H</i>		
<i>Olimex ARM-USB-OCD</i>		
<i>Olimex ARM-USB-OCD-H</i>		
<i>Olimex ARM-USB-TINY-H</i>		
<i>Olimex ARM-USB-TINY</i>		
<i>TIAO USB Multi-Protocol Adapter (TUMPA)</i>		

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP-IDF</i>	Espressif IoT Development Framework. Official development framework for ESP32.

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.

Microcontroller	ESP32
Frequency	240MHz
Flash	2MB
RAM	320KB
Vendor	<i>u-blox</i>

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

Name	Description
<code>Ar-duino</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<code>ESP-IDF</code>	Espressif IoT Development Framework. Official development framework for ESP32.

1.12.5 Espressif 8266

4D Systems gen4 IoT Range

Contents

- *4D Systems gen4 IoT Range*
 - *Hardware*
 - *Configuration*
 - *Uploading*
 - *Debugging*
 - *Frameworks*

Hardware

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

Microcontroller	ESP8266
Frequency	80MHz
Flash	512KB
RAM	80KB
Vendor	4D Systems

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 IoT Range settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `gen4iod.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:gen4iod]
platform = espressif8266
board = gen4iod

; change microcontroller
board_build.mcu = esp8266

; change MCU frequency
board_build.f_cpu = 80000000L
```

Uploading

4D Systems gen4 IoD Range supports the next uploading protocols:

- espota
- esptool

Default protocol is esptool

You can change upload protocol using *upload_protocol* option:

```
[env:gen4iod]
platform = espressif8266
board = gen4iod

upload_protocol = esptool
```

Debugging

PIO Unified Debugger currently does not support 4D Systems gen4 IoD Range board.

Frameworks

Name	Description
<i>Arduino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP8266 Non-OS SDK</i>	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.
<i>ESP8266 RTOS SDK</i>	ESP8266 SDK based on FreeRTOS, a truly free professional grade RTOS for microcontrollers

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.

Microcontroller	ESP8266
Frequency	80MHz
Flash	4MB
RAM	80KB
Vendor	Adafruit

Configuration

Please use huzzah ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[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.

```
[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:

```
[env:huzzah]
platform = espressif8266
board = huzzah

upload_protocol = esptool
```

Debugging

PIO Unified Debugger currently does not support Adafruit HUZZAH ESP8266 board.

Frameworks

Name	Description
<i>Arduino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP8266 Non-OS SDK</i>	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.
<i>ESP8266 RTOS SDK</i>	ESP8266 SDK based on FreeRTOS, a truly free professional grade RTOS for microcontrollers
<i>Simba</i>	Simba is an RTOS and build framework. It aims to make embedded programming easy and portable.

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.

Microcontroller	ESP8266
Frequency	80MHz
Flash	4MB
RAM	80KB
Vendor	DigiStump

Configuration

Please use `oak` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[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.

```
[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:

```
[env:oak]
platform = espressif8266
board = oak

upload_protocol = esptool
```

Debugging

PIO Unified Debugger currently does not support DigiStump Oak board.

Frameworks

Name	Description
<code>Ar-duino</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

ESP-WROOM-02

Contents

- *ESP-WROOM-02*
 - *Hardware*
 - *Configuration*
 - *Uploading*
 - *Debugging*
 - *Frameworks*

Hardware

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

Microcontroller	ESP8266
Frequency	80MHz
Flash	2MB
RAM	80KB
Vendor	Espressif

Configuration

Please use `esp_wroom_02` ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[env:esp_wroom_02]
platform = espressif8266
board = esp_wroom_02
```

You can override default ESP-WROOM-02 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `esp_wroom_02.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:esp_wroom_02]
platform = espressif8266
board = esp_wroom_02

; change microcontroller
board_build.mcu = esp8266

; change MCU frequency
board_build.f_cpu = 80000000L
```

Uploading

ESP-WROOM-02 supports the next uploading protocols:

- `espota`
- `esptool`

Default protocol is `esptool`

You can change upload protocol using `upload_protocol` option:

```
[env:esp_wroom_02]
platform = espressif8266
board = esp_wroom_02

upload_protocol = esptool
```

Debugging

PIO Unified Debugger currently does not support ESP-WROOM-02 board.

Frameworks

Name	Description
<i>Arduino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP8266 Non-OS SDK</i>	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.
<i>ESP8266 RTOS SDK</i>	ESP8266 SDK based on FreeRTOS, a truly free professional grade RTOS for microcontrollers
<i>Simba</i>	Simba is an RTOS and build framework. It aims to make embedded programming easy and portable.

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.

Microcontroller	ESP8266
Frequency	80MHz
Flash	4MB
RAM	80KB
Vendor	Doit

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

Name	Description
<i>Arduino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP8266 Non-OS SDK</i>	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.
<i>ESP8266 RTOS SDK</i>	ESP8266 SDK based on FreeRTOS, a truly free professional grade RTOS for microcontrollers

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.

Microcontroller	ESP8266
Frequency	80MHz
Flash	4MB
RAM	80KB
Vendor	DycodeX

Configuration

Please use `espectro` ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[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

Name	Description
<code>Arduino</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<code>ESP8266 Non-OS SDK</code>	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.
<code>ESP8266 RTOS SDK</code>	ESP8266 SDK based on FreeRTOS, a truly free professional grade RTOS for microcontrollers

ESPino

Contents

- *ESPino*
 - *Hardware*
 - *Configuration*
 - *Uploading*
 - *Debugging*
 - *Frameworks*

Hardware

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

Microcontroller	ESP8266
Frequency	80MHz
Flash	4MB
RAM	80KB
Vendor	ESPino

Configuration

Please use `espino` ID for `board` option in “*platformio.ini*” (*Project Configuration File*):

```
[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.

```
[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

Name	Description
<i>Arduino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP8266 Non-OS SDK</i>	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.
<i>ESP8266 RTOS SDK</i>	ESP8266 SDK based on FreeRTOS, a truly free professional grade RTOS for microcontrollers

ESPresso Lite 1.0

Contents

- *ESPresso Lite 1.0*
 - *Hardware*
 - *Configuration*
 - *Uploading*
 - *Debugging*
 - *Frameworks*

Hardware

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

Microcontroller	ESP8266
Frequency	80MHz
Flash	4MB
RAM	80KB
Vendor	ESPert

Configuration

Please use `espresso_lite_v1` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:espresso_lite_v1]
platform = espressif8266
board = espresso_lite_v1
```

You can override default ESPRESSO Lite 1.0 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `espresso_lite_v1.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[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:

```
[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

Name	Description
<i>Arduino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP8266 Non-OS SDK</i>	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.
<i>ESP8266 RTOS SDK</i>	ESP8266 SDK based on FreeRTOS, a truly free professional grade RTOS for microcontrollers

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.

Microcontroller	ESP8266
Frequency	80MHz
Flash	4MB
RAM	80KB
Vendor	ESPert

Configuration

Please use `espresso_lite_v2` ID for `board` option in “*platformio.ini*” (*Project Configuration File*):

```
[env:espresso_lite_v2]
platform = espressif8266
board = espresso_lite_v2
```

You can override default ESPRESSO Lite 2.0 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `espresso_lite_v2.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:espresso_lite_v2]
platform = espressif8266
board = espresso_lite_v2

; change microcontroller
board_build.mcu = esp8266

; change MCU frequency
board_build.f_cpu = 80000000L
```

Uploading

ESPRESSO Lite 2.0 supports the next uploading protocols:

- `espota`
- `esptool`

Default protocol is `esptool`

You can change upload protocol using `upload_protocol` option:

```
[env:espresso_lite_v2]
platform = espressif8266
board = espresso_lite_v2

upload_protocol = esptool
```

Debugging

PIO Unified Debugger currently does not support ESPRESSO Lite 2.0 board.

Frameworks

Name	Description
<code>Arduino</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<code>ESP8266 Non-OS SDK</code>	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.
<code>ESP8266 RTOS SDK</code>	ESP8266 SDK based on FreeRTOS, a truly free professional grade RTOS for microcontrollers

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.

Microcontroller	ESP8266
Frequency	80MHz
Flash	4MB
RAM	80KB
Vendor	Espressif

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:

```
[env:esp12e]
platform = espressif8266
board = esp12e

upload_protocol = esptool
```

Debugging

PIO Unified Debugger currently does not support Espressif ESP8266 ESP-12E board.

Frameworks

Name	Description
<i>Arduino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP8266 Non-OS SDK</i>	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.
<i>ESP8266 RTOS SDK</i>	ESP8266 SDK based on FreeRTOS, a truly free professional grade RTOS for microcontrollers
<i>Simba</i>	Simba is an RTOS and build framework. It aims to make embedded programming easy and portable.

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.

Microcontroller	ESP8266
Frequency	80MHz
Flash	1MB
RAM	80KB
Vendor	Espressif

Configuration

Please use `esp01_1m` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:esp01_1m]
platform = espressif8266
board = esp01_1m
```

You can override default Espressif Generic ESP8266 ESP-01 1M settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `esp01_1m.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:esp01_1m]
platform = espressif8266
board = esp01_1m

; change microcontroller
board_build.mcu = esp8266

; change MCU frequency
board_build.f_cpu = 80000000L
```

Uploading

Espressif Generic ESP8266 ESP-01 1M supports the next uploading protocols:

- `espota`
- `esptool`

Default protocol is `esptool`

You can change upload protocol using `upload_protocol` option:

```
[env:esp01_1m]
platform = espressif8266
board = esp01_1m

upload_protocol = esptool
```

Debugging

PIO Unified Debugger currently does not support Espressif Generic ESP8266 ESP-01 1M board.

Frameworks

Name	Description
<i>Arduino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP8266 Non-OS SDK</i>	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.
<i>ESP8266 RTOS SDK</i>	ESP8266 SDK based on FreeRTOS, a truly free professional grade RTOS for microcontrollers

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.

Microcontroller	ESP8266
Frequency	80MHz
Flash	512KB
RAM	80KB
Vendor	Espressif

Configuration

Please use esp01 ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[env:esp01]
platform = espressif8266
board = esp01
```

You can override default Espressif Generic ESP8266 ESP-01 512k settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `esp01.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:esp01]
platform = espressif8266
board = esp01

; change microcontroller
board_build.mcu = esp8266

; change MCU frequency
board_build.f_cpu = 80000000L
```

Uploading

Espressif Generic ESP8266 ESP-01 512k supports the next uploading protocols:

- `espota`
- `esptool`

Default protocol is `esptool`

You can change upload protocol using `upload_protocol` option:

```
[env:esp01]
platform = espressif8266
board = esp01

upload_protocol = esptool
```

Debugging

PIO Unified Debugger currently does not support Espressif Generic ESP8266 ESP-01 512k board.

Frameworks

Name	Description
<code>Arduino</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<code>ESP8266 Non-OS SDK</code>	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.
<code>ESP8266 RTOS SDK</code>	ESP8266 SDK based on FreeRTOS, a truly free professional grade RTOS for microcontrollers
<code>Simba</code>	Simba is an RTOS and build framework. It aims to make embedded programming easy and portable.

Espressif Generic ESP8266 ESP-07

Contents

- *Espressif Generic ESP8266 ESP-07*
 - *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	ESP8266
Frequency	80MHz
Flash	4MB
RAM	80KB
Vendor	Espressif

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 = 8000000L
```

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:

```
[env:esp07]
platform = espressif8266
board = esp07

upload_protocol = esptool
```

Debugging

PIO Unified Debugger currently does not support Espressif Generic ESP8266 ESP-07 board.

Frameworks

Name	Description
<i>Arduino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP8266 Non-OS SDK</i>	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.
<i>ESP8266 RTOS SDK</i>	ESP8266 SDK based on FreeRTOS, a truly free professional grade RTOS for microcontrollers

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.

Microcontroller	ESP8266
Frequency	80MHz
Flash	1MB
RAM	80KB
Vendor	Espressif

Configuration

Please use `esp8285` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:esp8285]
platform = espressif8266
board = esp8285
```

You can override default Generic ESP8285 Module settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `esp8285.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:esp8285]
platform = espressif8266
board = esp8285

; change microcontroller
board_build.mcu = esp8266

; change MCU frequency
board_build.f_cpu = 80000000L
```

Uploading

Generic ESP8285 Module supports the next uploading protocols:

- `espota`
- `esptool`

Default protocol is `esptool`

You can change upload protocol using `upload_protocol` option:

```
[env:esp8285]
platform = espressif8266
board = esp8285

upload_protocol = esptool
```

Debugging

PIO Unified Debugger currently does not support Generic ESP8285 Module board.

Frameworks

Name	Description
<i>Arduino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP8266 Non-OS SDK</i>	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.
<i>ESP8266 RTOS SDK</i>	ESP8266 SDK based on FreeRTOS, a truly free professional grade RTOS for microcontrollers

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.

Microcontroller	ESP8266
Frequency	80MHz
Flash	4MB
RAM	80KB
Vendor	Heltec

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.

```
[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:

```
[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

Name	Description
<code>Arduino</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<code>ESP8266 Non-OS SDK</code>	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.
<code>ESP8266 RTOS SDK</code>	ESP8266 SDK based on FreeRTOS, a truly free professional grade RTOS for microcontrollers

Invent One

Contents

- *Invent One*
 - *Hardware*
 - *Configuration*
 - *Uploading*
 - *Debugging*
 - *Frameworks*

Hardware

Platform [Espressif 8266](#): Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

Microcontroller	ESP8266
Frequency	80MHz
Flash	4MB
RAM	80KB
Vendor	Invent One

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:

```
[env:inventone]
platform = espressif8266
board = inventone

upload_protocol = esptool
```

Debugging

PIO Unified Debugger currently does not support Invent One board.

Frameworks

Name	Description
<i>Arduino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP8266 Non-OS SDK</i>	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.
<i>ESP8266 RTOS SDK</i>	ESP8266 SDK based on FreeRTOS, a truly free professional grade RTOS for microcontrollers
<i>Simba</i>	Simba is an RTOS and build framework. It aims to make embedded programming easy and portable.

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.

Microcontroller	ESP8266
Frequency	80MHz
Flash	4MB
RAM	80KB
Vendor	NodeMCU

Configuration

Please use nodemcu ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[env:nodemcu]
platform = espressif8266
board = nodemcu
```

You can override default NodeMCU 0.9 (ESP-12 Module) settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `nodemcu.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:nodemcu]
platform = espressif8266
board = nodemcu

; change microcontroller
board_build.mcu = esp8266

; change MCU frequency
board_build.f_cpu = 80000000L
```

Uploading

NodeMCU 0.9 (ESP-12 Module) supports the next uploading protocols:

- espota
- esptool

Default protocol is esptool

You can change upload protocol using `upload_protocol` option:

```
[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

Name	Description
<i>Arduino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP8266 Non-OS SDK</i>	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.
<i>ESP8266 RTOS SDK</i>	ESP8266 SDK based on FreeRTOS, a truly free professional grade RTOS for microcontrollers
<i>Simba</i>	Simba is an RTOS and build framework. It aims to make embedded programming easy and portable.

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.

Microcontroller	ESP8266
Frequency	80MHz
Flash	4MB
RAM	80KB
Vendor	NodeMCU

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 `nodemcu2.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:nodemcu2]
platform = espressif8266
board = nodemcu2

; change microcontroller
board_build.mcu = esp8266

; change MCU frequency
board_build.f_cpu = 80000000L
```

Uploading

NodeMCU 1.0 (ESP-12E Module) supports the next uploading protocols:

- `espota`
- `esptool`

Default protocol is `esptool`

You can change upload protocol using `upload_protocol` option:

```
[env:nodemcu2]
platform = espressif8266
board = nodemcu2

upload_protocol = esptool
```

Debugging

PIO Unified Debugger currently does not support NodeMCU 1.0 (ESP-12E Module) board.

Frameworks

Name	Description
<code>Arduino</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<code>ESP8266 Non-OS SDK</code>	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.
<code>ESP8266 RTOS SDK</code>	ESP8266 SDK based on FreeRTOS, a truly free professional grade RTOS for microcontrollers
<code>Simba</code>	Simba is an RTOS and build framework. It aims to make embedded programming easy and portable.

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.

Microcontroller	ESP8266
Frequency	80MHz
Flash	2MB
RAM	80KB
Vendor	Olimex

Configuration

Please use `modwifi` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:modwifi]
platform = espressif8266
board = modwifi
```

You can override default Olimex MOD-WIFI-ESP8266(-DEV) settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `modwifi.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:modwifi]
platform = espressif8266
board = modwifi

; change microcontroller
board_build.mcu = esp8266

; change MCU frequency
board_build.f_cpu = 8000000L
```

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 = espressif8266
board = modwifi

upload_protocol = esptool
```

Debugging

PIO Unified Debugger currently does not support Olimex MOD-WIFI-ESP8266(-DEV) board.

Frameworks

Name	Description
<i>Arduino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP8266 Non-OS SDK</i>	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.
<i>ESP8266 RTOS SDK</i>	ESP8266 SDK based on FreeRTOS, a truly free professional grade RTOS for microcontrollers

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.

Microcontroller	ESP8266
Frequency	80MHz
Flash	4MB
RAM	80KB
Vendor	Espressif

Configuration

Please use `phoenix_v1` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[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.

```
[env:phoenix_v1]
platform = espressif8266
board = phoenix_v1

; change microcontroller
board_build.mcu = esp8266

; change MCU frequency
board_build.f_cpu = 80000000L
```

Uploading

Phoenix 1.0 supports the next uploading protocols:

- `espota`
- `esptool`

Default protocol is `esptool`

You can change upload protocol using `upload_protocol` option:

```
[env:phoenix_v1]
platform = espressif8266
board = phoenix_v1

upload_protocol = esptool
```

Debugging

PIO Unified Debugger currently does not support Phoenix 1.0 board.

Frameworks

Name	Description
<i>Arduino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP8266 Non-OS SDK</i>	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.
<i>ESP8266 RTOS SDK</i>	ESP8266 SDK based on FreeRTOS, a truly free professional grade RTOS for microcontrollers

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.

Microcontroller	ESP8266
Frequency	80MHz
Flash	4MB
RAM	80KB
Vendor	Espressif

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

Name	Description
<code>Arduino</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<code>ESP8266 Non-OS SDK</code>	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.
<code>ESP8266 RTOS SDK</code>	ESP8266 SDK based on FreeRTOS, a truly free professional grade RTOS for microcontrollers

SparkFun Blynk Board

Contents

- *SparkFun Blynk Board*
 - *Hardware*
 - *Configuration*
 - *Uploading*
 - *Debugging*
 - *Frameworks*

Hardware

Platform *Espressif8266*: Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.

Microcontroller	ESP8266
Frequency	80MHz
Flash	4MB
RAM	80KB
Vendor	SparkFun

Configuration

Please use sparkfunBlynk ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[env:sparkfunBlynk]
platform = espressif8266
board = sparkfunBlynk
```

You can override default SparkFun Blynk Board settings per build environment using *board_**** option, where *** is a JSON object path from board manifest *sparkfunBlynk.json*. For example, *board_build.mcu*, *board_build.f_cpu*, etc.

```
[env:sparkfunBlynk]
platform = espressif8266
board = sparkfunBlynk

; change microcontroller
board_build.mcu = esp8266

; change MCU frequency
board_build.f_cpu = 80000000L
```

Uploading

SparkFun Blynk Board supports the next uploading protocols:

- espota
- esptool

Default protocol is `esptool`

You can change upload protocol using `upload_protocol` option:

```
[env:sparkfunBlynk]
platform = espressif8266
board = sparkfunBlynk

upload_protocol = esptool
```

Debugging

PIO Unified Debugger currently does not support SparkFun Blynk Board board.

Frameworks

Name	Description
<code>Arduino</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<code>ESP8266 Non-OS SDK</code>	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.
<code>ESP8266 RTOS SDK</code>	ESP8266 SDK based on FreeRTOS, a truly free professional grade RTOS for microcontrollers

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.

Microcontroller	ESP8266
Frequency	80MHz
Flash	512KB
RAM	80KB
Vendor	SparkFun

Configuration

Please use thing ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[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.

```
[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:

```
[env:thing]
platform = espressif8266
board = thing

upload_protocol = esptool
```

Debugging

PIO Unified Debugger currently does not support SparkFun ESP8266 Thing board.

Frameworks

Name	Description
<i>Arduino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP8266 Non-OS SDK</i>	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.
<i>ESP8266 RTOS SDK</i>	ESP8266 SDK based on FreeRTOS, a truly free professional grade RTOS for microcontrollers

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.

Microcontroller	ESP8266
Frequency	80MHz
Flash	512KB
RAM	80KB
Vendor	SparkFun

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.

```
[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:

```
[env:thingdev]
platform = espressif8266
board = thingdev

upload_protocol = esptool
```

Debugging

PIO Unified Debugger currently does not support SparkFun ESP8266 Thing Dev board.

Frameworks

Name	Description
<code>Arduino</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<code>ESP8266 Non-OS SDK</code>	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.
<code>ESP8266 RTOS SDK</code>	ESP8266 SDK based on FreeRTOS, a truly free professional grade RTOS for microcontrollers

SweetPea ESP-210

Contents

- *SweetPea ESP-210*
 - *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	ESP8266
Frequency	80MHz
Flash	4MB
RAM	80KB
Vendor	SweetPea

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

Name	Description
<i>Arduino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP8266 Non-OS SDK</i>	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.
<i>ESP8266 RTOS SDK</i>	ESP8266 SDK based on FreeRTOS, a truly free professional grade RTOS for microcontrollers

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.

Microcontroller	ESP8266
Frequency	80MHz
Flash	4MB
RAM	80KB
Vendor	ThaiEasyElec

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

Name	Description
<i>Arduino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP8266 Non-OS SDK</i>	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.
<i>ESP8266 RTOS SDK</i>	ESP8266 SDK based on FreeRTOS, a truly free professional grade RTOS for microcontrollers

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.

Microcontroller	ESP8266
Frequency	80MHz
Flash	4MB
RAM	80KB
Vendor	WEMOS

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.

```
[env:d1]
platform = espressif8266
board = d1

; change microcontroller
board_build.mcu = esp8266

; change MCU frequency
board_build.f_cpu = 80000000L
```

Uploading

WEMOS D1 R1 supports the next uploading protocols:

- espota
- esptool

Default protocol is esptool

You can change upload protocol using *upload_protocol* option:

```
[env:d1]
platform = espressif8266
board = d1

upload_protocol = esptool
```

Debugging

PIO Unified Debugger currently does not support WEMOS D1 R1 board.

Frameworks

Name	Description
<i>Arduino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP8266 Non-OS SDK</i>	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.
<i>ESP8266 RTOS SDK</i>	ESP8266 SDK based on FreeRTOS, a truly free professional grade RTOS for microcontrollers

WeMos D1 R2 and mini

Contents

- WeMos D1 R2 and mini
 - Hardware
 - Configuration
 - Uploading
 - Debugging
 - Frameworks

Hardware

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

Microcontroller	ESP8266
Frequency	80MHz
Flash	4MB
RAM	80KB
Vendor	WEMOS

Configuration

Please use d1_mini ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[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.

```
[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

Name	Description
<code>Arduino</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<code>ESP8266 Non-OS SDK</code>	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.
<code>ESP8266 RTOS SDK</code>	ESP8266 SDK based on FreeRTOS, a truly free professional grade RTOS for microcontrollers

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.

Microcontroller	ESP8266
Frequency	80MHz
Flash	1MB
RAM	80KB
Vendor	WEMOS

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

Name	Description
<i>Arduino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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.

Microcontroller	ESP8266
Frequency	80MHz
Flash	16MB
RAM	80KB
Vendor	WEMOS

Configuration

Please use `d1_mini_pro` ID for `board` option in “`platformio.ini`” (Project Configuration File):

```
[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.

```
[env:d1_mini_pro]
platform = espressif8266
board = d1_mini_pro
```

(continues on next page)

(continued from previous page)

```
; change microcontroller
board_build.mcu = esp8266

; change MCU frequency
board_build.f_cpu = 8000000L
```

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:

```
[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

Name	Description
<i>Arduino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP8266 Non-OS SDK</i>	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.
<i>ESP8266 RTOS SDK</i>	ESP8266 SDK based on FreeRTOS, a truly free professional grade RTOS for microcontrollers

WiFi Slot

Contents

- *WiFi Slot*
 - *Hardware*

- Configuration
- Uploading
- Debugging
- Frameworks

Hardware

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

Microcontroller	ESP8266
Frequency	80MHz
Flash	4MB
RAM	80KB
Vendor	Amperka

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

Name	Description
<i>Arduino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP8266 Non-OS SDK</i>	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.
<i>ESP8266 RTOS SDK</i>	ESP8266 SDK based on FreeRTOS, a truly free professional grade RTOS for microcontrollers

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.

Microcontroller	ESP8266
Frequency	80MHz
Flash	4MB
RAM	80KB
Vendor	WifiDuino

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

Name	Description
<i>Arduino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP8266 Non-OS SDK</i>	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.
<i>ESP8266 RTOS SDK</i>	ESP8266 SDK based on FreeRTOS, a truly free professional grade RTOS for microcontrollers

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.

Microcontroller	ESP8266
Frequency	80MHz
Flash	1MB
RAM	80KB
Vendor	Espressif

Configuration

Please use `wifinfo` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:wifinfo]
platform = espressif8266
board = wifinfo
```

You can override default WifInfo settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `wifinfo.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:wifinfo]
platform = espressif8266
board = wifinfo

; change microcontroller
board_build.mcu = esp8266

; change MCU frequency
board_build.f_cpu = 80000000L
```

Uploading

WifInfo supports the next uploading protocols:

- espota
- esptool

Default protocol is esptool

You can change upload protocol using *upload_protocol* option:

```
[env:wifinfo]
platform = espressif8266
board = wifinfo

upload_protocol = esptool
```

Debugging

PIO Unified Debugger currently does not support WifInfo board.

Frameworks

Name	Description
<i>Arduino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP8266 Non-OS SDK</i>	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.
<i>ESP8266 RTOS SDK</i>	ESP8266 SDK based on FreeRTOS, a truly free professional grade RTOS for microcontrollers

Wio Link

Contents

- *Wio Link*
 - *Hardware*
 - *Configuration*
 - *Uploading*
 - *Debugging*
 - *Frameworks*

Hardware

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

Microcontroller	ESP8266
Frequency	80MHz
Flash	4MB
RAM	80KB
Vendor	SeeedStudio

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:

```
[env:wio_link]
platform = espressif8266
board = wio_link

upload_protocol = esptool
```

Debugging

PIO Unified Debugger currently does not support Wio Link board.

Frameworks

Name	Description
<i>Arduino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP8266 Non-OS SDK</i>	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.
<i>ESP8266 RTOS SDK</i>	ESP8266 SDK based on FreeRTOS, a truly free professional grade RTOS for microcontrollers

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.

Microcontroller	ESP8266
Frequency	80MHz
Flash	4MB
RAM	80KB
Vendor	SeeedStudio

Configuration

Please use wio_node ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[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.

```
[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:

```
[env:wio_node]
platform = espressif8266
board = wio_node

upload_protocol = esptool
```

Debugging

PIO Unified Debugger currently does not support Wio Node board.

Frameworks

Name	Description
<i>Arduino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP8266 Non-OS SDK</i>	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.
<i>ESP8266 RTOS SDK</i>	ESP8266 SDK based on FreeRTOS, a truly free professional grade RTOS for microcontrollers

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.

Microcontroller	ESP8266
Frequency	80MHz
Flash	4MB
RAM	80KB
Vendor	XinaBox

Configuration

Please use `xinabox_cw01` ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[env:xinabox_cw01]
platform = espressif8266
board = xinabox_cw01
```

You can override default XinaBox CW01 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `xinabox_cw01.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:xinabox_cw01]
platform = espressif8266
board = xinabox_cw01

; change microcontroller
board_build.mcu = esp8266

; change MCU frequency
board_build.f_cpu = 80000000L
```

Uploading

XinaBox CW01 supports the next uploading protocols:

- `espota`
- `esptool`

Default protocol is `esptool`

You can change upload protocol using `upload_protocol` option:

```
[env:xinabox_cw01]
platform = espressif8266
board = xinabox_cw01

upload_protocol = esptool
```

Debugging

PIO Unified Debugger currently does not support XinaBox CW01 board.

Frameworks

Name	Description
<code>Arduino</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<code>ESP8266 Non-OS SDK</code>	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.
<code>ESP8266 RTOS SDK</code>	ESP8266 SDK based on FreeRTOS, a truly free professional grade RTOS for microcontrollers

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.

Microcontroller	MK64FN1M0VLL12
Frequency	120MHz
Flash	1MB
RAM	256KB
Vendor	Freescale

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:

```
[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.

Compatible Tools	On-board	Default
CMSIS-DAP	Yes	Yes
J-LINK		

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.

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.

Microcontroller	MK20DX128VLH5
Frequency	48MHz
Flash	128KB
RAM	16KB
Vendor	Freescale

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.

Compatible Tools	On-board	Default
CMSIS-DAP	Yes	Yes
J-LINK		

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.

Freescale Kinetis FRDM-K22F

Contents

- *Freescale Kinetis FRDM-K22F*
 - *Hardware*
 - *Configuration*

- *Uploading*
- *Debugging*
- *Frameworks*

Hardware

Platform *Freescale Kinetis*: Freescale Kinetis Microcontrollers is family of multiple hardware- and software-compatible ARM Cortex-M0+, Cortex-M4 and Cortex-M7-based MCU series. Kinetis MCUs offer exceptional low-power performance, scalability and feature integration.

Microcontroller	MK22FN512VLH12
Frequency	120MHz
Flash	512KB
RAM	128KB
Vendor	Freescale

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 = mk22fn512vlh12

; 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.

Compatible Tools	On-board	Default
CMSIS-DAP	Yes	Yes
J-LINK		

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.

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.

Microcontroller	MK64FN1M0VLL12
Frequency	120MHz
Flash	1MB
RAM	256KB
Vendor	Freescale

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.

Compatible Tools	On-board	Default
CMSIS-DAP	Yes	Yes
J-LINK		

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.

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.

Microcontroller	MK66FN2M0VMD18
Frequency	180MHz
Flash	2MB
RAM	256KB
Vendor	Freescale

Configuration

Please use `frdm_k66f` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:frdm_k66f]
platform = freescalekinetis
board = frdm_k66f
```

You can override default Freescale Kinetis FRDM-K66F settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `frdm_k66f.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:frdm_k66f]
platform = freescalekinetis
board = frdm_k66f

; change microcontroller
board_build.mcu = mk66fn2m0vmd18

; change MCU frequency
board_build.f_cpu = 180000000L
```

Uploading

Freescale Kinetis FRDM-K66F supports the next uploading protocols:

- cmsis-dap
- jlink
- mbed

Default protocol is `mbed`

You can change upload protocol using `upload_protocol` option:

```
[env:frdm_k66f]
platform = freescalekinetis
board = frdm_k66f

upload_protocol = mbed
```

Debugging

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

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for 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.

Compatible Tools	On-board	Default
CMSIS-DAP	Yes	Yes
J-LINK		

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.

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.

Microcontroller	MK82FN256VLL15
Frequency	150MHz
Flash	256KB
RAM	256KB
Vendor	Freescale

Configuration

Please use `frdm_k82f` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:frdm_k82f]
platform = freescalekinetis
board = frdm_k82f
```

You can override default Freescale Kinetis FRDM-K82F settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `frdm_k82f.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:frdm_k82f]
platform = freescalekinetis
board = frdm_k82f

; change microcontroller
board_build.mcu = mk82fn256v1115

; 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:

```
[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

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.

Freescale Kinetis FRDM-KL05Z

Contents

- *Freescale Kinetis FRDM-KL05Z*
 - *Hardware*
 - *Configuration*
 - *Uploading*
 - *Debugging*
 - *Frameworks*

Hardware

Platform *Freescale Kinetis*: Freescale Kinetis Microcontrollers is family of multiple hardware- and software-compatible ARM Cortex-M0+, Cortex-M4 and Cortex-M7-based MCU series. Kinetis MCUs offer exceptional low-power performance, scalability and feature integration.

Microcontroller	MKL05Z32VFM4
Frequency	48MHz
Flash	32KB
RAM	4KB
Vendor	Freescale

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.

```
[env:frdm_kl05z]
platform = freescalekinetis
board = frdm_kl05z

; change microcontroller
board_build.mcu = mkl05z32vfm4

; 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:

```
[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.

Compatible Tools	On-board	Default
CMSIS-DAP	Yes	Yes
J-LINK		

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.

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.

Microcontroller	MKL25Z128VLK4
Frequency	48MHz
Flash	128KB
RAM	16KB
Vendor	Freescale

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 = mk125z128vlk4

; 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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>CMSIS-DAP</i>	Yes	Yes
<i>J-LINK</i>		

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.

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.

Microcontroller	MKL27Z64VLH4
Frequency	48MHz
Flash	64KB
RAM	16KB
Vendor	Freescale

Configuration

Please use `frdm_kl27z` ID for `board` option in “*platformio.ini*” (*Project Configuration File*):

```
[env:frdm_kl27z]
platform = freescalekinetis
board = frdm_kl27z
```

You can override default Freescale Kinetis FRDM-KL27Z settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `frdm_kl27z.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:frdm_kl27z]
platform = freescalekinetis
board = frdm_kl27z

; change microcontroller
board_build.mcu = mkl27z64vlh4

; change MCU frequency
board_build.f_cpu = 48000000L
```

Uploading

Freescale Kinetis FRDM-KL27Z supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed

Default protocol is mbed

You can change upload protocol using *upload_protocol* option:

```
[env:frdm_kl27z]
platform = freescalekinetis
board = frdm_kl27z

upload_protocol = mbed
```

Debugging

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

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for 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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>CMSIS-DAP</i>	Yes	Yes
<i>J-LINK</i>		

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.

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.

Microcontroller	MKL43Z256VLH4
Frequency	48MHz
Flash	256KB
RAM	32KB
Vendor	Freescale

Configuration

Please use `frdm_kl43z` ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[env:frdm_kl43z]
platform = freescalekinetis
board = frdm_kl43z
```

You can override default Freescale Kinetis FRDM-KL43Z settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `frdm_kl43z.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:frdm_kl43z]
platform = freescalekinetis
board = frdm_kl43z

; change microcontroller
board_build.mcu = mk143z256vlh4

; change MCU frequency
board_build.f_cpu = 48000000L
```

Uploading

Freescale Kinetis FRDM-KL43Z supports the next uploading protocols:

- cmsis-dap
- jlink
- mbed

Default protocol is mbed

You can change upload protocol using *upload_protocol* option:

```
[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.

Compatible Tools	On-board	Default
CMSIS-DAP	Yes	Yes
J-LINK		

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.

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.

Microcontroller	MKL46Z256VLL4
Frequency	48MHz
Flash	256KB
RAM	32KB
Vendor	Freescale

Configuration

Please use `frdm_kl46z` ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[env:frdm_kl46z]
platform = freescalekinetis
board = frdm_kl46z
```

You can override default Freescale Kinetis FRDM-KL46Z settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `frdm_kl46z.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:frdm_kl46z]
platform = freescalekinetis
board = frdm_kl46z

; change microcontroller
board_build.mcu = mk146z256v114

; change MCU frequency
board_build.f_cpu = 48000000L
```

Uploading

Freescale Kinetis FRDM-KL46Z supports the next uploading protocols:

- cmsis-dap
- jlink
- mbed

Default protocol is mbed

You can change upload protocol using *upload_protocol* option:

```
[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.

Compatible Tools	On-board	Default
CMSIS-DAP	Yes	Yes
J-LINK		

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.

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.

Microcontroller	MKL82Z128VLK7
Frequency	96MHz
Flash	128KB
RAM	96KB
Vendor	Freescale

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.

```
[env:frdm_kl82z]
platform = freescalekinetis
board = frdm_kl82z

; change microcontroller
board_build.mcu = mk182z128vlk7

; change MCU frequency
board_build.f_cpu = 96000000L
```

Uploading

Freescale Kinetis FRDM-KL82Z supports the next uploading protocols:

- jlink
- mbed

Default protocol is mbed

You can change upload protocol using *upload_protocol* option:

```
[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.

Compatible Tools	On-board	Default
J-LINK		Yes

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.

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.

Microcontroller	MKW24D512
Frequency	50MHz
Flash	512KB
RAM	64KB
Vendor	Freescale

Configuration

Please use `frdm_kw24d` ID for `board` option in “*platformio.ini*” (*Project Configuration File*):

```
[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.

```
[env:frdm_kw24d]
platform = freescalekinetis
board = frdm_kw24d

; change microcontroller
board_build.mcu = mkw24d512

; change MCU frequency
board_build.f_cpu = 50000000L
```

Uploading

Freescale Kinetis FRDM-KW24D512 supports the next uploading protocols:

- jlink
- mbed

Default protocol is mbed

You can change upload protocol using *upload_protocol* option:

```
[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.

Compatible Tools	On-board	Default
J-LINK		Yes

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.

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.

Microcontroller	MKW41Z512VHT4
Frequency	48MHz
Flash	512KB
RAM	128KB
Vendor	Freescale

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.

```
[env:frdm_kw41z]
platform = freescalekinetis
board = frdm_kw41z

; change microcontroller
board_build.mcu = mkw41z512vht4

; change MCU frequency
board_build.f_cpu = 48000000L
```

Uploading

Freescale Kinetis FRDM-KW41Z supports the next uploading protocols:

- cmsis-dap
- jlink
- mbed

Default protocol is mbed

You can change upload protocol using *upload_protocol* option:

```
[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.

Compatible Tools	On-board	Default
CMSIS-DAP	Yes	Yes
J-LINK		

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.

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.

Microcontroller	MK64FN1M0VDC12
Frequency	120MHz
Flash	1MB
RAM	256KB
Vendor	MikroElektronika

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.

Compatible Tools	On-board	Default
CMSIS-DAP		Yes
J-LINK		

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.

1.12.7 GigaDevice GD32V

GD32VF103V-EVAL

Contents

- [GD32VF103V-EVAL](#)
 - [Hardware](#)
 - [Configuration](#)
 - [Uploading](#)
 - [Debugging](#)
 - [Frameworks](#)

Hardware

Platform [GigaDevice GD32V](#): The GigaDevice GD32V device is a 32-bit general-purpose microcontroller based on the RISC-V core with an impressive balance of processing power, reduced power consumption and peripheral set.

Microcontroller	GD32VF103VBT6
Frequency	108MHz
Flash	128KB
RAM	32KB
Vendor	Sipeed

Configuration

Please use gd32vf103v-eval ID for [board](#) option in “*platformio.ini*” (*Project Configuration File*):

```
[env:gd32vf103v-eval]
platform = gd32v
board = gd32vf103v-eval
```

You can override default GD32VF103V-EVAL settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `gd32vf103v-eval.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:gd32vf103v-eval]
platform = gd32v
board = gd32vf103v-eval

; change microcontroller
board_build.mcu = GD32VF103VBT6

; change MCU frequency
board_build.f_cpu = 108000000L
```

Uploading

GD32VF103V-EVAL supports the next uploading protocols:

- altera-usb-blaster
- gd-link
- jlink
- rv-link
- serial
- sipeed-rv-debugger
- um232h

Default protocol is gd-link

You can change upload protocol using *upload_protocol* option:

```
[env:gd32vf103v-eval]
platform = gd32v
board = gd32vf103v-eval

upload_protocol = gd-link
```

Debugging

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

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using *debug_tool* option in “*platformio.ini*” (*Project Configuration File*).

GD32VF103V-EVAL does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

Compatible Tools	On-board	Default
<i>Altera / Intel USB-Blaster Download Cable</i>		Yes
<i>GD-LINK</i>		
<i>J-LINK</i>		
<i>RV-LINK</i>		
<i>Sipeed RV Debugger</i>		
<i>UM232H</i>		

Frameworks

Name	Description
<i>Arduino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>GigaDevice GD32V SDK</i>	GigaDevice GD32VF103 Firmware Library (SDK)

Sipeed Longan Nano

Contents

- *Sipeed Longan Nano*
 - *Hardware*
 - *Configuration*
 - *Uploading*
 - *Debugging*
 - *Frameworks*

Hardware

Platform *GigaDevice GD32V*: The GigaDevice GD32V device is a 32-bit general-purpose microcontroller based on the RISC-V core with an impressive balance of processing power, reduced power consumption and peripheral set.

Microcontroller	GD32VF103CBT6
Frequency	108MHz
Flash	128KB
RAM	32KB
Vendor	Sipeed

Configuration

Please use sipeed-longan-nano ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[env:sipeed-longan-nano]
platform = gd32v
board = sipeed-longan-nano
```

You can override default Sipeed Longan Nano settings per build environment using *board_**** option, where *** is a JSON object path from board manifest *sipeed-longan-nano.json*. For example, *board_build.mcu*, *board_build.f_cpu*, etc.

```
[env:sipeed-longan-nano]
platform = gd32v
board = sipeed-longan-nano

; change microcontroller
board_build.mcu = GD32VF103CBT6

; change MCU frequency
board_build.f_cpu = 108000000L
```

Uploading

Sipeed Longan Nano supports the next uploading protocols:

- altera-usb-blaster
- gd-link
- jlink
- rv-link
- serial
- sipeed-rv-debugger
- um232h

Default protocol is *serial*

You can change upload protocol using *upload_protocol* option:

```
[env:sipeed-longan-nano]
platform = gd32v
board = sipeed-longan-nano

upload_protocol = serial
```

Debugging

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

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “`platformio.ini`” (*Project Configuration File*).

Sipeed Longan Nano does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

Compatible Tools	On-board	Default
<i>Altera / Intel USB-Blaster Download Cable</i>		Yes
<i>GD-LINK</i>		
<i>J-LINK</i>		
<i>RV-LINK</i>		
<i>Sipeed RV Debugger</i>		
<i>UM232H</i>		

Frameworks

Name	Description
<i>Arduino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>GigaDevice GD32V SDK</i>	GigaDevice GD32VF103 Firmware Library (SDK)

Wio Lite RISC-V

Contents

- *Wio Lite RISC-V*
 - *Hardware*
 - *Configuration*
 - *Uploading*
 - *Debugging*
 - *Frameworks*

Hardware

Platform *GigaDevice GD32V*: The GigaDevice GD32V device is a 32-bit general-purpose microcontroller based on the RISC-V core with an impressive balance of processing power, reduced power consumption and peripheral set.

Microcontroller	GD32VF103CBT6
Frequency	108MHz
Flash	128KB
RAM	32KB
Vendor	SeeedStudio

Configuration

Please use `wio_lite_risc-v` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:wio_lite_risc-v]
platform = gd32v
board = wio_lite_risc-v
```

You can override default Wio Lite RISC-V settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `wio_lite_risc-v.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:wio_lite_risc-v]
platform = gd32v
board = wio_lite_risc-v

; change microcontroller
board_build.mcu = GD32VF103CBT6

; change MCU frequency
board_build.f_cpu = 108000000L
```

Uploading

Wio Lite RISC-V supports the next uploading protocols:

- altera-usb-blaster
- gd-link
- jlink
- rv-link
- serial
- sipeed-rv-debugger
- um232h

Default protocol is `serial`

You can change upload protocol using `upload_protocol` option:

```
[env:wio_lite_risc-v]
platform = gd32v
board = wio_lite_risc-v

upload_protocol = serial
```

Debugging

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

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using *debug_tool* option in “*platformio.ini*” (*Project Configuration File*).

Wio Lite RISC-V does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

Compatible Tools	On-board	Default
<i>Altera / Intel USB-Blaster Download Cable</i>		Yes
<i>GD-LINK</i>		
<i>J-LINK</i>		
<i>RV-LINK</i>		
<i>Sipeed RV Debugger</i>		
<i>UM232H</i>		

Frameworks

Name	Description
<i>Arduino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>GigaDevice GD32VF103 Firmware Library (SDK)</i>	GigaDevice GD32VF103 Firmware Library (SDK)

1.12.8 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

Microcontroller	XMC1100
Frequency	32MHz
Flash	64KB
RAM	16KB
Vendor	Infineon

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.

Compatible Tools	On-board	Default
<i>J-Link</i>	Yes	Yes

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.

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

Microcontroller	XMC1100
Frequency	32MHz
Flash	64KB
RAM	16KB
Vendor	Infineon

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
```

(continues on next page)

(continued from previous page)

```
; 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.

Compatible Tools	On-board	Default
<i>J-LINK</i>	Yes	Yes

Frameworks

Name	Description
<i>Ardu- ino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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

Microcontroller	XMC1100
Frequency	32MHz
Flash	64KB
RAM	16KB
Vendor	Infineon

Configuration

Please use `xmc1100_xmc2go` ID for *board* option in “`platformio.ini`” (*Project Configuration File*):

```
[env:xmc1100_xmc2go]
platform = infineonxmc
board = xmc1100_xmc2go
```

You can override default XMC1100 XMC2Go settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `xmc1100_xmc2go.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:xmc1100_xmc2go]
platform = infineonxmc
board = xmc1100_xmc2go

; change microcontroller
board_build.mcu = XMC1100

; change MCU frequency
board_build.f_cpu = 32000000L
```

Debugging

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

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for 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.

Compatible Tools	On-board	Default
J-LINK	Yes	Yes

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.

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

Microcontroller	XMC1300
Frequency	32MHz
Flash	64KB
RAM	16KB
Vendor	Infineon

Configuration

Please use `xmc1300_boot_kit` ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[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.

```
[env:xmc1300_boot_kit]
platform = infineonxmc
board = xmc1300_boot_kit

; change microcontroller
board_build.mcu = XMC1300
```

(continues on next page)

(continued from previous page)

```
; 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.

Compatible Tools	On-board	Default
<i>J-LINK</i>	Yes	Yes

Frameworks

Name	Description
<i>Ardu- ino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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

Microcontroller	XMC1300
Frequency	32MHz
Flash	32KB
RAM	16KB
Vendor	Infineon

Configuration

Please use `xmc1300_sense2gol` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:xmc1300_sense2gol]
platform = infineonxmc
board = xmc1300_sense2gol
```

You can override default XMC1300 Sense2GoL settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `xmc1300_sense2gol.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:xmc1300_sense2gol]
platform = infineonxmc
board = xmc1300_sense2gol

; 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.

Compatible Tools	On-board	Default
<i>J-LINK</i>	Yes	Yes

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.

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

Microcontroller	XMC1400
Frequency	48MHz
Flash	1.95MB
RAM	16KB
Vendor	Infineon

Configuration

Please use `xmc1400_boot_kit` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:xmc1400_boot_kit]
platform = infineonxmc
board = xmc1400_boot_kit
```

You can override default XMC1400 Boot Kit settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `xmc1400_boot_kit.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:xmc1400_boot_kit]
platform = infineonxmc
board = xmc1400_boot_kit

; change microcontroller
board_build.mcu = XMC1400
```

(continues on next page)

(continued from previous page)

```
; 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.

Compatible Tools	On-board	Default
J-LINK	Yes	Yes

Frameworks

Name	Description
Ardu- ino	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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

Microcontroller	XMC4200
Frequency	80MHz
Flash	256KB
RAM	40KB
Vendor	Infineon

Configuration

Please use `xmc4200_distance2go` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:xmc4200_distance2go]
platform = infineonxmc
board = xmc4200_distance2go
```

You can override default XMC4200 Distance2Go settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `xmc4200_distance2go.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:xmc4200_distance2go]
platform = infineonxmc
board = xmc4200_distance2go

; change microcontroller
board_build.mcu = XMC4200

; change MCU frequency
board_build.f_cpu = 80000000L
```

Debugging

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

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for 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.

Compatible Tools	On-board	Default
J-LINK	Yes	Yes

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.

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

Microcontroller	XMC4700
Frequency	144MHz
Flash	2.00MB
RAM	1.95MB
Vendor	Infineon

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
```

(continues on next page)

(continued from previous page)

```
; 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.

Compatible Tools	On-board	Default
J-LINK	Yes	Yes

Frameworks

Name	Description
Ardu- ino	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

1.12.9 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.

Microcontroller	ARCV2EM
Frequency	32MHz
Flash	152KB
RAM	80KB
Vendor	Intel

Configuration

Please use `genuino101` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[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.

```
[env:genuino101]
platform = intel_arc32
board = genuino101

; change microcontroller
board_build.mcu = ARCV2EM

; change MCU frequency
board_build.f_cpu = 32000000L
```

Debugging

PIO Unified Debugger currently does not support Arduino/Genuino 101 board.

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.

1.12.10 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.

Microcontroller	N79E8432
Frequency	22MHz
Flash	4KB
RAM	512B
Vendor	Nuvoton

Configuration

Please use n79e8432 ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[env:n79e8432]
platform = intel_mcs51
board = n79e8432
```

You can override default Generic N79E8432 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `n79e8432.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:n79e8432]
platform = intel_mcs51
board = n79e8432

; change microcontroller
board_build.mcu = n79e8432

; change MCU frequency
board_build.f_cpu = 22118400L
```

Debugging

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.

Microcontroller	N79E844
Frequency	22MHz
Flash	8KB
RAM	512B
Vendor	Nuvoton

Configuration

Please use n79e844 ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[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.

```
[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.

Microcontroller	N79E845
Frequency	22MHz
Flash	16KB
RAM	512B
Vendor	Nuvoton

Configuration

Please use n79e845 ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[env:n79e845]
platform = intel_mcs51
board = n79e845
```

You can override default Generic N79E845 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `n79e845.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:n79e845]
platform = intel_mcs51
board = n79e845

; change microcontroller
board_build.mcu = n79e845

; change MCU frequency
board_build.f_cpu = 22118400L
```

Debugging

PIO Unified Debugger currently does not support Generic N79E845 board.

Generic N79E854

Contents

- *Generic N79E854*
 - *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.

Microcontroller	N79E854
Frequency	22MHz
Flash	8KB
RAM	512B
Vendor	Nuvoton

Configuration

Please use n79e854 ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[env:n79e854]
platform = intel_mcs51
board = n79e854
```

You can override default Generic N79E854 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `n79e854.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:n79e854]
platform = intel_mcs51
board = n79e854

; change microcontroller
board_build.mcu = n79e854

; change MCU frequency
board_build.f_cpu = 22118400L
```

Debugging

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.

Microcontroller	N79E855
Frequency	22MHz
Flash	16KB
RAM	512B
Vendor	Nuvoton

Configuration

Please use n79e855 ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[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.

```
[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.

Microcontroller	STC15F204EA
Frequency	11MHz
Flash	4KB
RAM	256B
Vendor	STC

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.

Microcontroller	STC15F2K60S2
Frequency	6MHz
Flash	60KB
RAM	2KB
Vendor	STC

Configuration

Please use `stc15f2k60s2` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:stc15f2k60s2]
platform = intel_mcs51
board = stc15f2k60s2
```

You can override default Generic STC15F2K60S2 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `stc15f2k60s2.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:stc15f2k60s2]
platform = intel_mcs51
board = stc15f2k60s2

; change microcontroller
board_build.mcu = stc15f2k60s2

; change MCU frequency
board_build.f_cpu = 6000000L
```

Debugging

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.

Microcontroller	STC15W204S
Frequency	11MHz
Flash	4KB
RAM	256B
Vendor	STC

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.

Microcontroller	STC15W404AS
Frequency	11MHz
Flash	4KB
RAM	512B
Vendor	STC

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.

Microcontroller	STC15W408AS
Frequency	11MHz
Flash	8KB
RAM	512B
Vendor	STC

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.

Microcontroller	STC89C52RC
Frequency	11MHz
Flash	8KB
RAM	512B
Vendor	STC

Configuration

Please use `stc89c52rc` ID for `board` option in “*platformio.ini*” (*Project Configuration File*):

```
[env:stc89c52rc]
platform = intel_mcs51
board = stc89c52rc
```

You can override default Generic STC89C52RC settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `stc89c52rc.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:stc89c52rc]
platform = intel_mcs51
board = stc89c52rc

; change microcontroller
board_build.mcu = stc89c52rc

; change MCU frequency
board_build.f_cpu = 11059200L
```

Debugging

PIO Unified Debugger currently does not support Generic STC89C52RC board.

1.12.11 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.

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-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]
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.

Compatible Tools	On-board	Default
<i>oddWires IOT-Bus JTAG</i>		Yes
<i>J-LINK</i>		
<i>Mini-Module FT2232H</i>		
<i>Olimex ARM-USB-OCD</i>		
<i>Olimex ARM-USB-OCD-H</i>		
<i>Olimex ARM-USB-TINY-H</i>		
<i>Olimex ARM-USB-TINY</i>		
<i>Sipeed RV Debugger</i>		
<i>TIAO USB Multi-Protocol Adapter (TUMPA)</i>		

Frameworks

Name	Description
<i>Arduino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>Kendryte Standalone SDK</i>	Kendryte Standalone SDK without OS support
<i>Kendryte FreeRTOS SDK</i>	Kendryte SDK with FreeRTOS support

Sipeed MAIX BiT with Mic

Contents

- *Sipeed MAIX BiT with Mic*
 - *Hardware*
 - *Configuration*
 - *Uploading*
 - *Debugging*
 - *Frameworks*

Hardware

Platform *Kendryte K210*: Kendryte K210 is an AI capable RISCV64 dual core SoC.

Microcontroller	K210
Frequency	400MHz
Flash	16MB
RAM	6MB
Vendor	Sipeed

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.

Compatible Tools	On-board	Default
<i>oddWires IOT-Bus JTAG</i>		Yes
<i>J-LINK</i>		
<i>Mini-Module FT2232H</i>		
<i>Olimex ARM-USB-OCD</i>		
<i>Olimex ARM-USB-OCD-H</i>		
<i>Olimex ARM-USB-TINY-H</i>		
<i>Olimex ARM-USB-TINY</i>		
<i>Sipeed RV Debugger</i>		
<i>TIAO USB Multi-Protocol Adapter (TUMPA)</i>		

Frameworks

Name	Description
<i>Arduino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>Kendryte Standalone SDK</i>	Kendryte Standalone SDK without OS support
<i>Kendryte FreeRTOS SDK</i>	Kendryte SDK with FreeRTOS support

Sipeed MAIX GO

Contents

- *Sipeed MAIX GO*
 - *Hardware*
 - *Configuration*
 - *Uploading*
 - *Debugging*
 - *Frameworks*

Hardware

Platform *Kendryte K210*: Kendryte K210 is an AI capable RISC-V64 dual core SoC.

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.

```
[env:sipeed-maix-go]
platform = kendryte210
board = sipeed-maix-go

; change microcontroller
board_build.mcu = K210

; change MCU frequency
board_build.f_cpu = 400000000L
```

Uploading

Sipeed MAIX GO supports the next uploading protocols:

- iot-bus-jtag
- jlink
- kflash
- minimodule
- olimex-arm-usb-ocd
- olimex-arm-usb-ocd-h
- olimex-arm-usb-tiny-h
- olimex-jtag-tiny
- sipeed-rv-debugger
- tumpa

Default protocol is `kflash`

You can change upload protocol using `upload_protocol` option:

```
[env:sipeed-maix-go]
platform = kendryte210
board = sipeed-maix-go

upload_protocol = kflash
```

Debugging

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

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using *debug_tool* option in “*platformio.ini*” (*Project Configuration File*).

Sipeed MAIX GO does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

Compatible Tools	On-board	Default
<i>oddWires IOT-Bus JTAG</i>		Yes
<i>J-LINK</i>		
<i>Mini-Module FT2232H</i>		
<i>Olimex ARM-USB-OCD</i>		
<i>Olimex ARM-USB-OCD-H</i>		
<i>Olimex ARM-USB-TINY-H</i>		
<i>Olimex ARM-USB-TINY</i>		
<i>Sipeed RV Debugger</i>		
<i>TIAO USB Multi-Protocol Adapter (TUMPA)</i>		

Frameworks

Name	Description
<i>Arduino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>Kendryte Standalone SDK</i>	Kendryte Standalone SDK without OS support
<i>Kendryte FreeRTOS SDK</i>	Kendryte SDK with FreeRTOS support

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.

Microcontroller	K210
Frequency	400MHz
Flash	16MB
RAM	6MB
Vendor	Sipeed

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.

Compatible Tools	On-board	Default
<i>oddWires IOT-Bus JTAG</i>		Yes
<i>J-LINK</i>		
<i>Mini-Module FT2232H</i>		
<i>Olimex ARM-USB-OCD</i>		
<i>Olimex ARM-USB-OCD-H</i>		
<i>Olimex ARM-USB-TINY-H</i>		
<i>Olimex ARM-USB-TINY</i>		
<i>Sipeed RV Debugger</i>		
<i>TIAO USB Multi-Protocol Adapter (TUMPA)</i>		

Frameworks

Name	Description
<i>Arduino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>Kendryte Standalone SDK</i>	Kendryte Standalone SDK without OS support
<i>Kendryte FreeRTOS SDK</i>	Kendryte SDK with FreeRTOS support

Sipeed MAIXDUINO

Contents

- *Sipeed MAIXDUINO*
 - *Hardware*
 - *Configuration*
 - *Uploading*
 - *Debugging*
 - *Frameworks*

Hardware

Platform *Kendryte K210*: Kendryte K210 is an AI capable RISCV64 dual core SoC.

Microcontroller	K210
Frequency	400MHz
Flash	16MB
RAM	6MB
Vendor	Sipeed

Configuration

Please use `sipeed-maixduino` ID for `board` option in “*platformio.ini*” (*Project Configuration File*):

```
[env:sipeed-maixduino]
platform = kendryte210
board = sipeed-maixduino
```

You can override default Sipeed MAIXDUINO settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `sipeed-maixduino.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:sipeed-maixduino]
platform = kendryte210
board = sipeed-maixduino

; change microcontroller
board_build.mcu = K210

; change MCU frequency
board_build.f_cpu = 400000000L
```

Uploading

Sipeed MAIXDUINO supports the next uploading protocols:

- 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.

Compatible Tools	On-board	Default
<i>oddWires IOT-Bus JTAG</i>		Yes
<i>J-LINK</i>		
<i>Mini-Module FT2232H</i>		
<i>Olimex ARM-USB-OCD</i>		
<i>Olimex ARM-USB-OCD-H</i>		
<i>Olimex ARM-USB-TINY-H</i>		
<i>Olimex ARM-USB-TINY</i>		
<i>Sipeed RV Debugger</i>		
<i>TIAO USB Multi-Protocol Adapter (TUMPA)</i>		

Frameworks

Name	Description
<i>Arduino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>Kendryte Standalone SDK</i>	Kendryte Standalone SDK without OS support
<i>Kendryte FreeRTOS SDK</i>	Kendryte SDK with FreeRTOS support

1.12.12 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.

Microcontroller	ICE40-HX1K-TQ144
Frequency	12MHz
Flash	32KB
RAM	32KB
Vendor	FPGAwars

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.

Microcontroller	ICE40-HX1K-TQ144
Frequency	12MHz
Flash	32KB
RAM	32KB
Vendor	Lattice

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.

1.12.13 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.

Microcontroller	BCM2835
Frequency	700MHz
Flash	512MB
RAM	512MB
Vendor	Raspberry Pi

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

Name	Description
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.

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.

Microcontroller	BCM2836
Frequency	900MHz
Flash	1GB
RAM	1GB
Vendor	Raspberry Pi

Configuration

Please use `raspberrypi_2b` ID for *board* option in “`platformio.ini`” (*Project Configuration File*):

```
[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.

```
[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

Name	Description
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.

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.

Microcontroller	BCM2837
Frequency	1200MHz
Flash	1GB
RAM	1GB
Vendor	Raspberry Pi

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

Name	Description
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.

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.

Microcontroller	BCM2835
Frequency	1000MHz
Flash	512MB
RAM	512MB
Vendor	Raspberry Pi

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

Name	Description
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.

RushUp Kitra 520

Contents

- *RushUp Kitra 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.

Microcontroller	EXYNOS3250
Frequency	1000MHz
Flash	4GB
RAM	512MB
Vendor	RushUp

Configuration

Please use `kitra_520` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[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.

```
[env:kitra_520]
platform = linux_arm
board = kitra_520

; change microcontroller
board_build.mcu = exynos3250
```

(continues on next page)

(continued from previous page)

```
; change MCU frequency
board_build.f_cpu = 1000000000L
```

Debugging

PIO Unified Debugger currently does not support RushUp Kitra 520 board.

Frameworks

Name	Description
AR-TIK 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.

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.

Microcontroller	EXYNOS5422
Frequency	1500MHz
Flash	16GB
RAM	2GB
Vendor	Samsung

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

Name	Description
<code>ART- TIK SDK</code>	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.

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.

Microcontroller	EXYNOS3250
Frequency	1000MHz
Flash	4GB
RAM	512MB
Vendor	Samsung

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
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

Name	Description
<code>ARTIK SDK</code>	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.

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.

Microcontroller	S5P4418
Frequency	1200MHz
Flash	4GB
RAM	512MB
Vendor	Samsung

Configuration

Please use `artik_530` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:artik_530]
platform = linux_arm
board = artik_530
```

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.

```
[env:artik_530]
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

Name	Description
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.

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.

Microcontroller	S5P6818
Frequency	1400MHz
Flash	4GB
RAM	1GB
Vendor	Samsung

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

Name	Description
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.

1.12.14 Maxim 32

MAX32620FTHR

Contents

- [*MAX32620FTHR*](#)
 - *Hardware*
 - *Configuration*
 - *Uploading*
 - *Debugging*
 - *Frameworks*

Hardware

Platform [*Maxim 32*](#): Maxim's microcontrollers provide low-power, efficient, and secure solutions for challenging embedded applications. Maxim's processors embed cutting-edge technologies to secure data and intellectual property, proven analog circuitry for real-world applications, and battery-conserving low power operation.

Microcontroller	MAX32620FTHR
Frequency	96MHz
Flash	2MB
RAM	256KB
Vendor	Maxim

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.

Compatible Tools	On-board	Default
J-LINK		Yes

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.

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.

Microcontroller	MAX32625
Frequency	96MHz
Flash	512KB
RAM	160KB
Vendor	Maxim

Configuration

Please use `max32625mbed` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[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.

```
[env:max32625mbed]
platform = maxim32
```

(continues on next page)

(continued from previous page)

```
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

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.

MAX32625NEXPAQ

Contents

- [*MAX32625NEXPAQ*](#)
 - [*Hardware*](#)
 - [*Configuration*](#)
 - [*Debugging*](#)
 - [*Frameworks*](#)

Hardware

Platform [*Maxim 32*](#): Maxim's microcontrollers provide low-power, efficient, and secure solutions for challenging embedded applications. Maxim's processors embed cutting-edge technologies to secure data and intellectual property, proven analog circuitry for real-world applications, and battery-conserving low power operation.

Microcontroller	MAX32625
Frequency	96MHz
Flash	512KB
RAM	160KB
Vendor	Maxim

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

Name	Description
<code>mbed</code>	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.

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.

Microcontroller	MAX32625
Frequency	96MHz
Flash	512KB
RAM	160KB
Vendor	Maxim

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
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

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.

Maxim ARM mbed Enabled Development Platform for MAX32600

Contents

- *Maxim ARM mbed Enabled Development Platform for MAX32600*
 - *Hardware*
 - *Configuration*
 - *Uploading*
 - *Debugging*
 - *Frameworks*

Hardware

Platform *Maxim 32*: Maxim's microcontrollers provide low-power, efficient, and secure solutions for challenging embedded applications. Maxim's processors embed cutting-edge technologies to secure data and intellectual property, proven analog circuitry for real-world applications, and battery-conserving low power operation.

Microcontroller	MAX32600
Frequency	24MHz
Flash	256KB
RAM	32KB
Vendor	Maxim

Configuration

Please use `max32600mbed` ID for `board` option in “*platformio.ini*” (*Project Configuration File*):

```
[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	On-board	Default
<i>CMSIS-DAP</i>	Yes	Yes

Frameworks

Name	Description
<i>mbed</i>	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.

Microcontroller	MAX32620
Frequency	96MHz
Flash	2MB
RAM	256KB
Vendor	Maxim

Configuration

Please use `max32620hsp` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[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.

```
[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:

```
[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.

Compatible Tools	On-board	Default
J-LINK		Yes

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.

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.

Microcontroller	MAX32630
Frequency	96MHz
Flash	2MB
RAM	512KB
Vendor	Maxim

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
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

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.

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.

Microcontroller	MAX32610
Frequency	24MHz
Flash	256KB
RAM	32KB
Vendor	Maxim

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

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.

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.

Microcontroller	MAX32620IWG
Frequency	96MHz
Flash	2MB
RAM	256KB
Vendor	Sigma Delta Technologies

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

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.

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.

Microcontroller	MAX32625ITK
Frequency	96MHz
Flash	512KB
RAM	160KB
Vendor	Sigma Delta Technologies

Configuration

Please use `sdt32625b` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[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.

```
[env:sdt32625b]
platform = maxim32
board = sdt32625b
```

(continues on next page)

(continued from previous page)

```
; 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

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.

1.12.15 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!

Microcontroller	32MX795F512L
Frequency	80MHz
Flash	508KB
RAM	128KB
Vendor	4DSystems

Configuration

Please use `picadillo_35t` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[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.

```
[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

Name	Description
<code>Ar-duino</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

DataStation Mini

Contents

- *DataStation Mini*
 - *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	32MX150F128C
Frequency	40MHz
Flash	120KB
RAM	32KB
Vendor	Makerology

Configuration

Please use `dsmini` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:dsmini]
platform = microchippic32
board = dsmini
```

You can override default DataStation Mini settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `dsmini.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:dsmini]
platform = microchippic32
board = dsmini

; change microcontroller
board_build.mcu = 32MX150F128C

; change MCU frequency
board_build.f_cpu = 40000000L
```

Debugging

PIO Unified Debugger currently does not support DataStation Mini board.

Frameworks

Name	Description
Arduin	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

Digilent Cerebot 32MX4

Contents

- *Diligent 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!

Microcontroller	32MX460F512L
Frequency	80MHz
Flash	508KB
RAM	32KB
Vendor	Diligent

Configuration

Please use `cerebot32mx4` ID for `board` option in “*platformio.ini*” (*Project Configuration File*):

```
[env:cerebot32mx4]
platform = microchippic32
board = cerebot32mx4
```

You can override default Diligent 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 Diligent Cerebot 32MX4 board.

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.

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!

Microcontroller	32MX795F512L
Frequency	80MHz
Flash	508KB
RAM	128KB
Vendor	Digilent

Configuration

Please use `cerebot32mx7` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:cerebot32mx7]
platform = microchippic32
board = cerebot32mx7
```

You can override default Digilent Cerebot 32MX7 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `cerebot32mx7.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:cerebot32mx7]
platform = microchippic32
board = cerebot32mx7

; change microcontroller
```

(continues on next page)

(continued from previous 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

Name	Description
Arduin	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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!

Microcontroller	32MZ2048EFG124
Frequency	200MHz
Flash	1.98MB
RAM	512KB
Vendor	Digilent

Configuration

Please use `openscope` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:openscope]
platform = microchippic32
board = openscope
```

You can override default Digilent OpenScope settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `openscope.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:openscope]
platform = microchippic32
board = openscope

; change microcontroller
board_build.mcu = 32MZ2048EFG124

; change MCU frequency
board_build.f_cpu = 200000000L
```

Debugging

PIO Unified Debugger currently does not support Digilent OpenScope board.

Frameworks

Name	Description
<code>Ar-duino</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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!

Microcontroller	32MX150F128D
Frequency	40MHz
Flash	124KB
RAM	32KB
Vendor	Digilent

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

Name	Description
<code>Arduin</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

Digilent chipKIT DP32

Contents

- *Digilent chipKIT DP32*
 - *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	32MX250F128B
Frequency	40MHz
Flash	120KB
RAM	32KB
Vendor	Digilent

Configuration

Please use `chipkit_dp32` ID for `board` option in “*platformio.ini*” (*Project Configuration File*):

```
[env:chipkit_dp32]
platform = microchippic32
board = chipkit_dp32
```

You can override default Digilent chipKIT DP32 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `chipkit_dp32.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:chipkit_dp32]
platform = microchippic32
board = chipkit_dp32

; change microcontroller
board_build.mcu = 32MX250F128B

; change MCU frequency
board_build.f_cpu = 40000000L
```

Debugging

PIO Unified Debugger currently does not support Digilent chipKIT DP32 board.

Frameworks

Name	Description
<code>Arduino</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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!

Microcontroller	32MX795F512L
Frequency	80MHz
Flash	508KB
RAM	128KB
Vendor	Digilent

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

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.

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!

Microcontroller	32MX320F128H
Frequency	80MHz
Flash	124KB
RAM	16KB
Vendor	Digilent

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)

(continued from previous 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

Name	Description
Arduin	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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!

Microcontroller	32MX460F512L
Frequency	80MHz
Flash	508KB
RAM	32KB
Vendor	Digilent

Configuration

Please use `chipkit_pro_mx4` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:chipkit_pro_mx4]
platform = microchippic32
board = chipkit_pro_mx4
```

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.

```
[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

Name	Description
<code>Arduin</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

Digilent chipKIT Pro MX7

Contents

- *Digilent chipKIT Pro MX7*
 - *Hardware*
 - *Configuration*
 - *Debugging*
 - *Frameworks*

Hardware

Platform [Microchip PIC32](#): Microchip's 32-bit portfolio with the MIPS microAptiv or M4K core offer high performance microcontrollers, and all the tools needed to develop your embedded projects. PIC32 MCUs gives your application the processing power, memory and peripherals your design needs!

Microcontroller	32MX795F512L
Frequency	80MHz
Flash	508KB
RAM	128KB
Vendor	Digilent

Configuration

Please use `chipkit_pro_mx7` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[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

Name	Description
<code>Arduino</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

Digilent chipKIT UNO32

Contents

- *Digilent chipKIT UNO32*
 - *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	32MX320F128H
Frequency	80MHz
Flash	124KB
RAM	16KB
Vendor	Digilent

Configuration

Please use uno_pic32 ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[env:uno_pic32]
platform = microchippic32
board = uno_pic32
```

You can override default Digilent chipKIT UNO32 settings per build environment using *board_**** option, where *** is a JSON object path from board manifest *uno_pic32.json*. For example, *board_build.mcu*, *board_build.f_cpu*, etc.

```
[env:uno_pic32]
platform = microchippic32
board = uno_pic32

; change microcontroller
board_build.mcu = 32MX320F128H

; change MCU frequency
board_build.f_cpu = 80000000L
```

Debugging

PIO Unified Debugger currently does not support Digilent chipKIT UNO32 board.

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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!

Microcontroller	32MX695F512L
Frequency	80MHz
Flash	508KB
RAM	128KB
Vendor	Digilent

Configuration

Please use `chipkit_wf32` ID for `board` option in “*platformio.ini*” (*Project Configuration File*):

```
[env:chipkit_wf32]
platform = microchippic32
board = chipkit_wf32
```

You can override default Digilent chipKIT WF32 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `chipkit_wf32.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:chipkit_wf32]
platform = microchippic32
board = chipkit_wf32

; change microcontroller
board_build.mcu = 32MX695F512L

; change MCU frequency
board_build.f_cpu = 80000000L
```

Debugging

PIO Unified Debugger currently does not support Digilent chipKIT WF32 board.

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.

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!

Microcontroller	32MZ2048ECG100
Frequency	200MHz
Flash	1.98MB
RAM	512KB
Vendor	Digilent

Configuration

Please use `chipkit_wifire` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:chipkit_wifire]
platform = microchippic32
board = chipkit_wifire
```

You can override default Digilent chipKIT WiFire settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `chipkit_wifire.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:chipkit_wifire]
platform = microchippic32
board = chipkit_wifire

; change microcontroller
```

(continues on next page)

(continued from previous 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

Name	Description
Arduin	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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!

Microcontroller	32MX340F512H
Frequency	80MHz
Flash	508KB
RAM	32KB
Vendor	Digilent

Configuration

Please use `chipkit_uc32` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:chipkit_uc32]
platform = microchippic32
board = chipkit_uc32
```

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.

```
[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

Name	Description
<code>Ar-duino</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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!

Microcontroller	32MX250F128B
Frequency	40MHz
Flash	120KB
RAM	32KB
Vendor	element14

Configuration

Please use `chipkit_pi` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:chipkit_pi]
platform = microchippic32
board = chipkit_pi
```

You can override default Element14 chipKIT Pi settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `chipkit_pi.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:chipkit_pi]
platform = microchippic32
board = chipkit_pi

; change microcontroller
board_build.mcu = 32MX250F128B

; change MCU frequency
board_build.f_cpu = 40000000L
```

Debugging

PIO Unified Debugger currently does not support Element14 chipKIT Pi board.

Frameworks

Name	Description
<code>Ardu</code> <code>duino</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

Fubarino Mini

Contents

- *Fubarino Mini*
 - *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	32MX250F128D
Frequency	48MHz
Flash	120KB
RAM	32KB
Vendor	Fubarino

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

Name	Description
<code>Arduino</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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!

Microcontroller	32MX795F512H
Frequency	80MHz
Flash	508KB
RAM	128KB
Vendor	Fubarino

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

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.

HelvePic32

Contents

- [HelvePic32](#)
 - [Hardware](#)
 - [Configuration](#)
 - [Debugging](#)
 - [Frameworks](#)

Hardware

Platform [Microchip PIC32](#): Microchip's 32-bit portfolio with the MIPS microAptiv or M4K core offer high performance microcontrollers, and all the tools needed to develop your embedded projects. PIC32 MCUs gives your application the processing power, memory and peripherals your design needs!

Microcontroller	32MX250F128B
Frequency	48MHz
Flash	120KB
RAM	32KB
Vendor	BOXTEC

Configuration

Please use helvepic32 ID for `board` option in “`platformio.ini`” (Project Configuration File):

```
[env:helvepic32]
platform = microchippic32
board = helvepic32
```

You can override default HelvePic32 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `helvepic32.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:helvepic32]
platform = microchippic32
board = helvepic32

; change microcontroller
board_build.mcu = 32MX250F128B
```

(continues on next page)

(continued from previous page)

```
; change MCU frequency
board_build.f_cpu = 48000000L
```

Debugging

PIO Unified Debugger currently does not support HelvePic32 board.

Frameworks

Name	Description
Ar-duino	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

HelvePic32

Contents

- [HelvePic32](#)
 - [Hardware](#)
 - [Configuration](#)
 - [Debugging](#)
 - [Frameworks](#)

Hardware

Platform [Microchip PIC32](#): Microchip's 32-bit portfolio with the MIPS microAptiv or M4K core offer high performance microcontrollers, and all the tools needed to develop your embedded projects. PIC32 MCUs gives your application the processing power, memory and peripherals your design needs!

Microcontroller	32MX250F128B
Frequency	48MHz
Flash	120KB
RAM	32KB
Vendor	BOXTEC

Configuration

Please use `helvepic32_breadboardside` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:helvepic32_breadboardside]
platform = microchippic32
board = helvepic32_breadboardside
```

You can override default HelvePic32 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `helvepic32_breadboardside.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:helvepic32_breadboardside]
platform = microchippic32
board = helvepic32_breadboardside

; change microcontroller
board_build.mcu = 32MX250F128B

; change MCU frequency
board_build.f_cpu = 48000000L
```

Debugging

PIO Unified Debugger currently does not support HelvePic32 board.

Frameworks

Name	Description
<code>Ar-duino</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

HelvePic32

Contents

- *HelvePic32*
 - *Hardware*
 - *Configuration*
 - *Debugging*
 - *Frameworks*

Hardware

Platform [Microchip PIC32](#): Microchip's 32-bit portfolio with the MIPS microAptiv or M4K core offer high performance microcontrollers, and all the tools needed to develop your embedded projects. PIC32 MCUs gives your application the processing power, memory and peripherals your design needs!

Microcontroller	32MX250F128D
Frequency	48MHz
Flash	120KB
RAM	32KB
Vendor	BOXTEC

Configuration

Please use helvepic32_smd ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[env:helvepic32_smd]
platform = microchippic32
board = helvepic32_smd
```

You can override default HelvePic32 settings per build environment using *board_**** option, where *** is a JSON object path from board manifest *helvepic32_smd.json*. For example, *board_build.mcu*, *board_build.f_cpu*, etc.

```
[env:helvepic32_smd]
platform = microchippic32
board = helvepic32_smd

; change microcontroller
board_build.mcu = 32MX250F128D

; change MCU frequency
board_build.f_cpu = 48000000L
```

Debugging

PIO Unified Debugger currently does not support HelvePic32 board.

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.

HelvePic32 MX270

Contents

- [HelvePic32 MX270](#)
 - [Hardware](#)
 - [Configuration](#)

- Debugging
- Frameworks

Hardware

Platform *Microchip PIC32*: Microchip's 32-bit portfolio with the MIPS microAptiv or M4K core offer high performance microcontrollers, and all the tools needed to develop your embedded projects. PIC32 MCUs gives your application the processing power, memory and peripherals your design needs!

Microcontroller	32MX270F256B
Frequency	48MHz
Flash	244KB
RAM	62KB
Vendor	BOXTEC

Configuration

Please use helvepic32_mx270 ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[env:helvepic32_mx270]
platform = microchippic32
board = helvepic32_mx270
```

You can override default HelvePic32 MX270 settings per build environment using *board_**** option, where *** is a JSON object path from board manifest *helvepic32_mx270.json*. For example, *board_build.mcu*, *board_build.f_cpu*, etc.

```
[env:helvepic32_mx270]
platform = microchippic32
board = helvepic32_mx270

; change microcontroller
board_build.mcu = 32MX270F256B

; change MCU frequency
board_build.f_cpu = 48000000L
```

Debugging

PIO Unified Debugger currently does not support HelvePic32 MX270 board.

Frameworks

Name	Description
<i>Arduino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

HelvePic32 Robot

Contents

- *HelvePic32 Robot*
 - *Hardware*
 - *Configuration*
 - *Debugging*
 - *Frameworks*

Hardware

Platform *Microchip PIC32*: Microchip's 32-bit portfolio with the MIPS microAptiv or M4K core offer high performance microcontrollers, and all the tools needed to develop your embedded projects. PIC32 MCUs gives your application the processing power, memory and peripherals your design needs!

Microcontroller	32MX270F256D
Frequency	48MHz
Flash	244KB
RAM	62KB
Vendor	BOXTEC

Configuration

Please use `helvepic32_robot` ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[env:helvepic32_robot]
platform = microchippic32
board = helvepic32_robot
```

You can override default HelvePic32 Robot settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `helvepic32_robot.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:helvepic32_robot]
platform = microchippic32
board = helvepic32_robot

; change microcontroller
board_build.mcu = 32MX270F256D

; change MCU frequency
board_build.f_cpu = 48000000L
```

Debugging

PIO Unified Debugger currently does not support HelvePic32 Robot board.

Frameworks

Name	Description
<i>Arduino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

HelvePic32 SMD MX270

Contents

- *HelvePic32 SMD MX270*
 - *Hardware*
 - *Configuration*
 - *Debugging*
 - *Frameworks*

Hardware

Platform *Microchip PIC32*: Microchip's 32-bit portfolio with the MIPS microAptiv or M4K core offer high performance microcontrollers, and all the tools needed to develop your embedded projects. PIC32 MCUs gives your application the processing power, memory and peripherals your design needs!

Microcontroller	32MX270F256D
Frequency	48MHz
Flash	244KB
RAM	62KB
Vendor	BOXTEC

Configuration

Please use `helvepic32_smd_mx270` ID for `board` option in “*platformio.ini*” (*Project Configuration File*):

```
[env:helvepic32_smd_mx270]
platform = microchippic32
board = helvepic32_smd_mx270
```

You can override default HelvePic32 SMD MX270 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `helvepic32_smd_mx270.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:helvepic32_smd_mx270]
platform = microchippic32
board = helvepic32_smd_mx270

; change microcontroller
```

(continues on next page)

(continued from previous page)

```
board_build.mcu = 32MX270F256D  
  
; change MCU frequency  
board_build.f_cpu = 48000000L
```

Debugging

PIO Unified Debugger currently does not support HelvePic32 SMD MX270 board.

Frameworks

Name	Description
Arduin	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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!

Microcontroller	32MX460F512L
Frequency	80MHz
Flash	508KB
RAM	32KB
Vendor	MikroElektronika

Configuration

Please use `clicker2` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[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.

```
[env:clicker2]
platform = microchippic32
board = clicker2

; change microcontroller
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

Name	Description
<code>Ar-duino</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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!

Microcontroller	32MZ2048EFH100
Frequency	252MHz
Flash	1.98MB
RAM	512KB
Vendor	MikroElektronika

Configuration

Please use `flipnclkzmz` ID for `board` option in “`platformio.ini`” (Project Configuration File):

```
[env:flipnclkzmz]
platform = microchippic32
board = flipnclkzmz
```

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 `flipnclkzmz.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:flipnclkzmz]
platform = microchippic32
board = flipnclkzmz

; 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

Name	Description
<code>Arduino</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

Mini 2.0

Contents

- *Mini 2.0*
 - *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	32MX270F256D
Frequency	48MHz
Flash	240KB
RAM	62KB
Vendor	Fubarino

Configuration

Please use fubarino_mini_20 ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[env:fubarino_mini_20]
platform = microchippic32
board = fubarino_mini_20
```

You can override default Mini 2.0 settings per build environment using *board_**** option, where *** is a JSON object path from board manifest *fubarino_mini_20.json*. For example, *board_build.mcu*, *board_build.f_cpu*, etc.

```
[env:fubarino_mini_20]
platform = microchippic32
board = fubarino_mini_20

; change microcontroller
board_build.mcu = 32MX270F256D

; change MCU frequency
board_build.f_cpu = 48000000L
```

Debugging

PIO Unified Debugger currently does not support Mini 2.0 board.

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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!

Microcontroller	32MX440F256H
Frequency	80MHz
Flash	252KB
RAM	32KB
Vendor	Olimex

Configuration

Please use pinguino32 ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[env:pinguino32]
platform = microchippic32
board = pinguino32
```

You can override default Olimex PIC32-PINGUINO settings per build environment using *board_**** option, where *** is a JSON object path from board manifest *pinguino32.json*. For example, *board_build.mcu*, *board_build.f_cpu*, etc.

```
[env:pinguino32]
platform = microchippic32
board = pinguino32

; change microcontroller
board_build.mcu = 32MX440F256H

; change MCU frequency
board_build.f_cpu = 80000000L
```

Debugging

PIO Unified Debugger currently does not support Olimex PIC32-PINGUINO board.

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.

OpenBCI 32bit

Contents

- [OpenBCI 32bit](#)
 - [Hardware](#)
 - [Configuration](#)
 - [Debugging](#)
 - [Frameworks](#)

Hardware

Platform [Microchip PIC32](#): Microchip's 32-bit portfolio with the MIPS microAptiv or M4K core offer high performance microcontrollers, and all the tools needed to develop your embedded projects. PIC32 MCUs gives your application the processing power, memory and peripherals your design needs!

Microcontroller	32MX250F128B
Frequency	40MHz
Flash	120KB
RAM	32KB
Vendor	OpenBCI

Configuration

Please use `openbci` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:openbci]
platform = microchippic32
board = openbci
```

You can override default OpenBCI 32bit settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `openbci.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:openbci]
platform = microchippic32
board = openbci

; change microcontroller
```

(continues on next page)

(continued from previous page)

```
board_build.mcu = 32MX250F128B  
;  
; change MCU frequency  
board_build.f_cpu = 40000000L
```

Debugging

PIO Unified Debugger currently does not support OpenBCI 32bit board.

Frameworks

Name	Description
Arduin	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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!

Microcontroller	32MX440F512H
Frequency	80MHz
Flash	508KB
RAM	32KB
Vendor	PONTECH

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

Name	Description
<code>Ar-duino</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

Pic32 CUI32-Development Stick

Contents

- *Pic32 CUI32-Development Stick*
 - *Hardware*
 - *Configuration*
 - *Debugging*
 - *Frameworks*

Hardware

Platform *Microchip PIC32*: Microchip's 32-bit portfolio with the MIPS microAptiv or M4K core offer high performance microcontrollers, and all the tools needed to develop your embedded projects. PIC32 MCUs gives your application the processing power, memory and peripherals your design needs!

Microcontroller	32MX440F512H
Frequency	80MHz
Flash	508KB
RAM	32KB
Vendor	SparkFun

Configuration

Please use `cui32` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:cui32]
platform = microchippic32
board = cui32
```

You can override default Pic32 CUI32-Development Stick settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `cui32.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:cui32]
platform = microchippic32
board = cui32

; change microcontroller
board_build.mcu = 32MX440F512H

; change MCU frequency
board_build.f_cpu = 80000000L
```

Debugging

PIO Unified Debugger currently does not support Pic32 CUI32-Development Stick board.

Frameworks

Name	Description
<code>Ardu</code> <code>duino</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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!

Microcontroller	32MZ2048EFG100
Frequency	200MHz
Flash	1.98MB
RAM	512KB
Vendor	Pontech

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
board_build.mcu = 32MZ2048EFG100

; change MCU frequency
board_build.f_cpu = 200000000L
```

Debugging

PIO Unified Debugger currently does not support Pontech NoFire board.

Frameworks

Name	Description
<i>Arduino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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!

Microcontroller	32MX795F512L
Frequency	80MHz
Flash	508KB
RAM	128KB
Vendor	Pontech

Configuration

Please use `quick240_usb` ID for `board` option in “*platformio.ini*” (*Project Configuration File*):

```
[env:quick240_usb]
platform = microchippic32
board = quick240_usb
```

You can override default Pontech Quick240 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `quick240_usb.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[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

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.

RGB Station

Contents

- [RGB Station](#)
 - [Hardware](#)
 - [Configuration](#)
 - [Debugging](#)
 - [Frameworks](#)

Hardware

Platform [Microchip PIC32](#): Microchip's 32-bit portfolio with the MIPS microAptiv or M4K core offer high performance microcontrollers, and all the tools needed to develop your embedded projects. PIC32 MCUs gives your application the processing power, memory and peripherals your design needs!

Microcontroller	32MX270F256D
Frequency	48MHz
Flash	240KB
RAM	62KB
Vendor	ChipKIT

Configuration

Please use `rgb_station` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:rgb_station]
platform = microchippic32
board = rgb_station
```

You can override default RGB Station settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `rgb_station.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:rgb_station]
platform = microchippic32
board = rgb_station

; change microcontroller
board_build.mcu = 32MX270F256D
```

(continues on next page)

(continued from previous page)

```
; change MCU frequency
board_build.f_cpu = 48000000L
```

Debugging

PIO Unified Debugger currently does not support RGB Station board.

Frameworks

Name	Description
Arduin	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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*):

```
[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.

```
[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

Name	Description
<code>Ar-duino</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

UBW32 MX460

Contents

- *UBW32 MX460*
 - *Hardware*
 - *Configuration*
 - *Debugging*
 - *Frameworks*

Hardware

Platform *Microchip PIC32*: Microchip's 32-bit portfolio with the MIPS microAptiv or M4K core offer high performance microcontrollers, and all the tools needed to develop your embedded projects. PIC32 MCUs gives your application the processing power, memory and peripherals your design needs!

Microcontroller	32MX460F512L
Frequency	80MHz
Flash	508KB
RAM	32KB
Vendor	UBW32

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

Name	Description
<code>Arduin</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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!

Microcontroller	32MX795F512L
Frequency	80MHz
Flash	508KB
RAM	128KB
Vendor	UBW32

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

Name	Description
<i>Arduino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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!

Microcontroller	32MX270F256D
Frequency	40MHz
Flash	120KB
RAM	32KB
Vendor	chipKIT

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
board_build.mcu = 32MX270F256D

; change MCU frequency
board_build.f_cpu = 40000000L
```

Debugging

PIO Unified Debugger currently does not support chipKIT Lenny board.

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.

chipKIT WiFire rev. C

Contents

- *chipKIT WiFire rev. C*
 - *Hardware*
 - *Configuration*
 - *Debugging*
 - *Frameworks*

Hardware

Platform [Microchip PIC32](#): Microchip's 32-bit portfolio with the MIPS microAptiv or M4K core offer high performance microcontrollers, and all the tools needed to develop your embedded projects. PIC32 MCUs gives your application the processing power, memory and peripherals your design needs!

Microcontroller	32MZ2048EFG100
Frequency	200MHz
Flash	1.98MB
RAM	512KB
Vendor	Digilent

Configuration

Please use `chipkit_wifire_revC` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:chipkit_wifire_revC]
platform = microchippic32
board = chipkit_wifire_revC
```

You can override default chipKIT WiFire rev. C settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `chipkit_wifire_revC.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:chipkit_wifire_revC]
platform = microchippic32
board = chipkit_wifire_revC

; change microcontroller
```

(continues on next page)

(continued from previous page)

```
board_build.mcu = 32MZ2048EFG100  
;  
; change MCU frequency  
board_build.f_cpu = 200000000L
```

Debugging

PIO Unified Debugger currently does not support chipKIT WiFire rev. C board.

Frameworks

Name	Description
Arduin	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

1.12.16 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.

Microcontroller	NRF51822
Frequency	16MHz
Flash	256KB
RAM	16KB
Vendor	BBC

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	On-board	Default
CMSIS-DAP	Yes	Yes
J-LINK		

Frameworks

Name	Description
Arduino	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
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.

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.

Microcontroller	NRF51822
Frequency	32MHz
Flash	256KB
RAM	32KB
Vendor	BluzDK

Configuration

Please use `bluz_dk` ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[env:bluz_dk]
platform = nordicnrf51
board = bluz_dk
```

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.

```
[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:

```
[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
<i>Black Magic Probe</i>		Yes
<i>J-LINK</i>		
<i>ST-LINK</i>		

Frameworks

Name	Description
<i>Ardu- duino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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.

Microcontroller	NRF51822
Frequency	16MHz
Flash	256KB
RAM	16KB
Vendor	Calliope

Configuration

Please use `calliope_mini` ID for `board` option in “*platformio.ini*” (*Project Configuration File*):

```
[env:calliope_mini]
platform = nordicnrf51
board = calliope_mini
```

You can override default Calliope mini settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `calliope_mini.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[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:

```
[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.

Compatible Tools	On-board	Default
CMSIS-DAP	Yes	Yes
J-LINK		

Frameworks

Name	Description
Arduino	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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.

Microcontroller	NRF51822
Frequency	32MHz
Flash	256KB
RAM	32KB
Vendor	Delta

Configuration

Please use `dfcm_nnn40` ID for `board` option in “*platformio.ini*” (Project Configuration File):

```
[env:dfcm_nnn40]
platform = nordicnrf51
board = dfcm_nnn40
```

You can override default Delta DFCM-NNN40 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `dfcm_nnn40.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:dfcm_nnn40]
platform = nordicnrf51
board = dfcm_nnn40
```

(continues on next page)

(continued from previous 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:

```
[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.

Compatible Tools	On-board	Default
CMSIS-DAP	Yes	Yes

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.

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.

Microcontroller	NRF51822
Frequency	32MHz
Flash	256KB
RAM	16KB
Vendor	Delta

Configuration

Please use `delta_dfcm_nnn50` ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[env:delta_dfcm_nnn50]
platform = nordicnrf51
board = delta_dfcm_nnn50
```

You can override default Delta DFCM-NNN50 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `delta_dfcm_nnn50.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:delta_dfcm_nnn50]
platform = nordicnrf51
board = delta_dfcm_nnn50

; change microcontroller
board_build.mcu = nrf51822

; change MCU frequency
board_build.f_cpu = 32000000L
```

Uploading

Delta DFCM-NNN50 supports the next uploading protocols:

- cmsis-dap
- mbed

Default protocol is cmsis-dap

You can change upload protocol using *upload_protocol* option:

```
[env:delta_dfcn_nnn50]
platform = nordicnrf51
board = delta_dfcn_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.

Compatible Tools	On-board	Default
CMSIS-DAP	Yes	Yes

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.

JKSoft Wallbot BLE

Contents

- [JKSoft Wallbot 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.

Microcontroller	NRF51822
Frequency	16MHz
Flash	128KB
RAM	16KB
Vendor	JKSoft

Configuration

Please use `wallbot_ble` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:wallbot_ble]
platform = nordicnrf51
board = wallbot_ble
```

You can override default JKSoft Wallbot BLE settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `wallbot_ble.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:wallbot_ble]
platform = nordicnrf51
board = wallbot_ble

; change microcontroller
board_build.mcu = nrf51822

; change MCU frequency
board_build.f_cpu = 16000000L
```

Uploading

JKSoft Wallbot BLE supports the next uploading protocols:

- cmsis-dap
- mbed

Default protocol is cmsis-dap

You can change upload protocol using `upload_protocol` option:

```
[env:wallbot_ble]
platform = nordicnrf51
board = wallbot_ble

upload_protocol = cmsis-dap
```

Debugging

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

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for 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.

Compatible Tools	On-board	Default
CMSIS-DAP	Yes	Yes

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.

Microcontroller	NRF51822
Frequency	32MHz
Flash	256KB
RAM	32KB
Vendor	Nordic

Configuration

Please use `nrf51_beacon` ID for `board` option in “*platformio.ini*” (*Project Configuration File*):

```
[env:nrf51_beacon]
platform = nordicnrf51
board = nrf51_beacon
```

You can override default Nordic Beacon Kit (PCA20006) settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `nrf51_beacon.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:nrf51_beacon]
platform = nordicnrf51
board = nrf51_beacon

; change microcontroller
board_build.mcu = nrf51822

; change MCU frequency
board_build.f_cpu = 32000000L
```

Uploading

Nordic Beacon Kit (PCA20006) supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- nrfjprog
- stlink

Default protocol is `jlink`

You can change upload protocol using `upload_protocol` option:

```
[env:nrf51_beacon]
platform = nordicnrf51
board = nrf51_beacon
```

(continues on next page)

(continued from previous 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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>CMSIS-DAP</i>	Yes	Yes
<i>J-LINK</i>	Yes	
<i>ST-LINK</i>		

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

Nordic nRF51 Dongle (PCA10031)

Contents

- *Nordic nRF51 Dongle (PCA10031)*
 - *Hardware*
 - *Configuration*
 - *Uploading*
 - *Debugging*
 - *Frameworks*

Hardware

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

Microcontroller	NRF51822
Frequency	32MHz
Flash	256KB
RAM	32KB
Vendor	Nordic

Configuration

Please use `nrf51_dongle` ID for `board` option in “`platformio.ini`” (Project Configuration File):

```
[env:nrf51_dongle]
platform = nordicnrf51
board = nrf51_dongle
```

You can override default Nordic nRF51 Dongle (PCA10031) settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `nrf51_dongle.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:nrf51_dongle]
platform = nordicnrf51
board = nrf51_dongle

; change microcontroller
board_build.mcu = nrf51822

; change MCU frequency
board_build.f_cpu = 32000000L
```

Uploading

Nordic nRF51 Dongle (PCA10031) supports the next uploading protocols:

- cmsis-dap
- jlink
- mbed
- nrfjprog

Default protocol is `jlink`

You can change upload protocol using `upload_protocol` option:

```
[env:nrf51_dongle]
platform = nordicnrf51
board = nrf51_dongle

upload_protocol = jlink
```

Debugging

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

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for 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.

Compatible Tools	On-board	Default
CMSIS-DAP	Yes	Yes
J-LINK	Yes	

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.
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 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.

Microcontroller	NRF51822
Frequency	16MHz
Flash	128KB
RAM	16KB
Vendor	Nordic

Configuration

Please use nrf51_mkit ID for `board` option in “*platformio.ini*” (*Project Configuration File*):

```
[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.

```
[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:

```
[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.

Compatible Tools	On-board	Default
CMSIS-DAP	Yes	Yes

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 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.

Microcontroller	NRF51822
Frequency	32MHz
Flash	256KB
RAM	32KB
Vendor	Nordic

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(PC1000X) 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(PC1000X) 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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>CMSIS-DAP</i>	Yes	Yes
<i>J-LINK</i>	Yes	
<i>ST-LINK</i>		

Frameworks

Name	Description
<i>Arduino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>mbed</i>	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.

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.

Microcontroller	NRF51822
Frequency	32MHz
Flash	256KB
RAM	32KB
Vendor	OSHChip

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
<i>Black Magic Probe</i>		Yes
<i>J-LINK</i>		
<i>ST-LINK</i>		

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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.

Microcontroller	NRF51822
Frequency	16MHz
Flash	256KB
RAM	32KB
Vendor	RedBearLab

Configuration

Please use `redBearLabBLENano` ID for `board` option in “*platformio.ini*” (*Project Configuration File*):

```
[env:redBearLabBLENano]
platform = nordicnrf51
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	On-board	Default
<i>Black Magic Probe</i>		
<i>CMSIS-DAP</i>	Yes	Yes
<i>J-LINK</i>		
<i>ST-LINK</i>		

Frameworks

Name	Description
<i>Arduino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>mbed</i>	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.

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.

Microcontroller	NRF51822
Frequency	16MHz
Flash	256KB
RAM	16KB
Vendor	RedBearLab

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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>CMSIS-DAP</i>	Yes	Yes
<i>J-LINK</i>		
<i>ST-LINK</i>		

Frameworks

Name	Description
<i>Arduino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>mbed</i>	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 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.

Microcontroller	NRF51822
Frequency	16MHz
Flash	128KB
RAM	16KB
Vendor	SeeedStudio

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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>CMSIS-DAP</i>	Yes	Yes
<i>J-LINK</i>		
<i>ST-LINK</i>		

Frameworks

Name	Description
<i>mbed</i>	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 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.

Microcontroller	NRF51822
Frequency	16MHz
Flash	256KB
RAM	16KB
Vendor	SeeedStudio

Configuration

Please use `seeedArchLink` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[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.

```
[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:

```
[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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>CMSIS-DAP</i>	Yes	Yes
<i>J-LINK</i>		
<i>ST-LINK</i>		

Frameworks

Name	Description
<i>mbed</i>	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 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.

Microcontroller	NRF51822
Frequency	16MHz
Flash	256KB
RAM	16KB
Vendor	SeeedStudio

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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>CMSIS-DAP</i>	Yes	Yes
<i>J-LINK</i>		
<i>ST-LINK</i>		

Frameworks

Name	Description
<i>Arduino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>mbed</i>	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.

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.

Microcontroller	NRF51822
Frequency	32MHz
Flash	256KB
RAM	32KB
Vendor	sino:bit

Configuration

Please use Sinobit ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[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.

```
[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:

```
[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	On-board	Default
<i>Black Magic Probe</i>		Yes
<i>J-LINK</i>		
<i>ST-LINK</i>		

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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.

Microcontroller	NRF51822
Frequency	16MHz
Flash	256KB
RAM	16KB
Vendor	Switch Science

Configuration

Please use `hrm1017` ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[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.

Compatible Tools	On-board	Default
CMSIS-DAP	Yes	Yes

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.

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.

Microcontroller	NRF51822
Frequency	32MHz
Flash	256KB
RAM	32KB
Vendor	Switch Science

Configuration

Please use `ty51822r3` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:ty51822r3]
platform = nordicnrf51
board = ty51822r3
```

You can override default Switch Science mbed TY51822r3 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `ty51822r3.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:ty51822r3]
platform = nordicnrf51
board = ty51822r3

; change microcontroller
board_build.mcu = nrf51822

; change MCU frequency
board_build.f_cpu = 32000000L
```

Uploading

Switch Science mbed TY51822r3 supports the next uploading protocols:

- cmsis-dap
- mbed

Default protocol is cmsis-dap

You can change upload protocol using *upload_protocol* option:

```
[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.

Compatible Tools	On-board	Default
<i>CMSIS-DAP</i>	Yes	Yes

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.

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.

Microcontroller	NRF51822
Frequency	16MHz
Flash	128KB
RAM	32KB
Vendor	VNG

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.

Compatible Tools	On-board	Default
<i>CMSIS-DAP</i>	Yes	Yes

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.

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.

Microcontroller	NRF51822
Frequency	32MHz
Flash	256KB
RAM	32KB
Vendor	Waveshare

Configuration

Please use `waveshare_ble400` ID for `board` option in “*platformio.ini*” (*Project Configuration File*):

```
[env:waveshare_ble400]
platform = nordicnrf51
board = waveshare_ble400
```

You can override default Waveshare BLE400 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `waveshare_ble400.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:waveshare_ble400]
platform = nordicnrf51
board = waveshare_ble400

; change microcontroller
board_build.mcu = nrf51822

; change MCU frequency
board_build.f_cpu = 32000000L
```

Uploading

Waveshare BLE400 supports the next uploading protocols:

- blackmagic
- jlink
- nrfjprog
- stlink

Default protocol is jlink

You can change upload protocol using *upload_protocol* option:

```
[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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		Yes
<i>J-LINK</i>		
<i>ST-LINK</i>		

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.

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.

Microcontroller	NRF51822
Frequency	16MHz
Flash	256KB
RAM	32KB
Vendor	ng-beacon

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)

(continued from previous 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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		Yes
<i>J-LINK</i>		
<i>ST-LINK</i>		

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.

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.

Microcontroller	NRF51822
Frequency	16MHz
Flash	256KB
RAM	16KB
Vendor	y5 design

Configuration

Please use `nrf51822_y5_mbug` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:nrf51822_y5_mbug]
platform = nordicnrf51
board = nrf51822_y5_mbug
```

You can override default y5 nRF51822 mbug settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `nrf51822_y5_mbug.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:nrf51822_y5_mbug]
platform = nordicnrf51
board = nrf51822_y5_mbug
```

(continues on next page)

(continued from previous page)

```
; change microcontroller
board_build.mcu = nrf51822

; change MCU frequency
board_build.f_cpu = 16000000L
```

Uploading

y5 nRF51822 mbug supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed
- nrfjprog
- stlink

Default protocol is cmsis-dap

You can change upload protocol using *upload_protocol* option:

```
[env:nrf51822_y5_mbug]
platform = nordicnrf51
board = nrf51822_y5_mbug

upload_protocol = cmsis-dap
```

Debugging

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

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for 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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>CMSIS-DAP</i>	Yes	Yes
<i>J-LINK</i>		
<i>ST-LINK</i>		

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.

1.12.17 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.

Microcontroller	NRF52832
Frequency	64MHz
Flash	512KB
RAM	64KB
Vendor	Adafruit

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.

```
[env:adafruit_feather_nrf52832]
platform = nordicnrf52
board = adafruit_feather_nrf52832

; change microcontroller
board_build.mcu = nrf52832

; change MCU frequency
board_build.f_cpu = 64000000L
```

Uploading

Adafruit Bluefruit nRF52832 Feather supports the next uploading protocols:

- jlink
- nrfjprog
- nrfutil

Default protocol is nrfutil

You can change upload protocol using *upload_protocol* option:

```
[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.

Compatible Tools	On-board	Default
J-LINK	Yes	Yes

Frameworks

Name	Description
<i>Arduino</i>	Arduino Wiring-based Framework allows writing cross-platform software 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 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.

Microcontroller	NRF52840
Frequency	64MHz
Flash	796KB
RAM	243KB
Vendor	Adafruit

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
```

(continues on next page)

(continued from previous page)

```
; 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.

Compatible Tools	On-board	Default
<i>J-LINK</i>	Yes	Yes

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

Bluey nRF52832 IoT

Contents

- *Bluey nRF52832 IoT*
 - *Hardware*
 - *Configuration*
 - *Uploading*
 - *Debugging*
 - *Frameworks*

Hardware

Platform *Nordic nRF52*: The nRF52 Series are built for speed to carry out increasingly complex tasks in the shortest possible time and return to sleep, conserving precious battery power. They have a Cortex-M4F processor and are the most capable Bluetooth Smart SoCs on the market.

Microcontroller	NRF52832
Frequency	64MHz
Flash	512KB
RAM	64KB
Vendor	Electronut Labs

Configuration

Please use `bluey` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:bluey]
platform = nordicnrf52
board = bluey
```

You can override default Bluey nRF52832 IoT settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `bluey.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:bluey]
platform = nordicnrf52
board = bluey

; change microcontroller
board_build.mcu = nrf52832

; change MCU frequency
board_build.f_cpu = 64000000L
```

Uploading

Bluey nRF52832 IoT supports the next uploading protocols:

- blackmagic
- jlink
- nrfjprog
- stlink

Default protocol is `jlink`

You can change upload protocol using `upload_protocol` option:

```
[env:bluey]
platform = nordicnrf52
board = bluey

upload_protocol = jlink
```

Debugging

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

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for 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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		Yes
<i>J-LINK</i>		
<i>ST-LINK</i>		

Frameworks

Name	Description
<i>Ar-duino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

Circuit Playground Bluefruit

Contents

- *Circuit Playground Bluefruit*
 - *Hardware*
 - *Configuration*
 - *Uploading*
 - *Debugging*
 - *Frameworks*

Hardware

Platform *Nordic nRF52*: The nRF52 Series are built for speed to carry out increasingly complex tasks in the shortest possible time and return to sleep, conserving precious battery power. They have a Cortex-M4F processor and are the most capable Bluetooth Smart SoCs on the market.

Microcontroller	NRF52840
Frequency	64MHz
Flash	796KB
RAM	243KB
Vendor	Adafruit

Configuration

Please use `adafruit_cplaynrf52840` ID for `board` option in “*platformio.ini*” (*Project Configuration File*):

```
[env:adafruit_cplaynrf52840]
platform = nordicnrf52
board = adafruit_cplaynrf52840
```

You can override default Circuit Playground Bluefruit settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `adafruit_cplaynrf52840.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:adafruit_cplaynrf52840]
platform = nordicnrf52
board = adafruit_cplaynrf52840

; change microcontroller
board_build.mcu = nrf52840

; change MCU frequency
board_build.f_cpu = 64000000L
```

Uploading

Circuit Playground Bluefruit supports the next uploading protocols:

- `jlink`

- nrfjprog
- nrfutil

Default protocol is nrfutil

You can change upload protocol using *upload_protocol* option:

```
[env:adafruit_cplaynrf52840]
platform = nordicnrf52
board = adafruit_cplaynrf52840

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*).

Circuit Playground Bluefruit does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

Compatible Tools	On-board	Default
J-LINK		Yes

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.

Delta DFBM-NQ620

Contents

- *Delta DFBM-NQ620*
 - *Hardware*
 - *Configuration*
 - *Uploading*
 - *Debugging*

– *Frameworks*

Hardware

Platform *Nordic nRF52*: The nRF52 Series are built for speed to carry out increasingly complex tasks in the shortest possible time and return to sleep, conserving precious battery power. They have a Cortex-M4F processor and are the most capable Bluetooth Smart SoCs on the market.

Microcontroller	NRF52832
Frequency	64MHz
Flash	512KB
RAM	64KB
Vendor	Delta

Configuration

Please use `delta_dfbm_nq620` ID for `board` option in “*platformio.ini*” (*Project Configuration File*):

```
[env:delta_dfbm_nq620]
platform = nordicnrf52
board = delta_dfbm_nq620
```

You can override default Delta DFBM-NQ620 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `delta_dfbm_nq620.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:delta_dfbm_nq620]
platform = nordicnrf52
board = delta_dfbm_nq620

; change microcontroller
board_build.mcu = nrf52832

; change MCU frequency
board_build.f_cpu = 64000000L
```

Uploading

Delta DFBM-NQ620 supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed
- nrfjprog
- stlink

Default protocol is `cmsis-dap`

You can change upload protocol using `upload_protocol` option:

```
[env:delta_dfbm_nq620]
platform = nordicnrf52
board = delta_dfbm_nq620

upload_protocol = cmsis-dap
```

Debugging

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

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for 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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>CMSIS-DAP</i>	Yes	Yes
<i>J-LINK</i>		
<i>ST-LINK</i>		

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>mbed</i>	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.

Metro nRF52840 Express

Contents

- *Metro nRF52840 Express*
 - *Hardware*

- Configuration
- Uploading
- Debugging
- Frameworks

Hardware

Platform *Nordic nRF52*: The nRF52 Series are built for speed to carry out increasingly complex tasks in the shortest possible time and return to sleep, conserving precious battery power. They have a Cortex-M4F processor and are the most capable Bluetooth Smart SoCs on the market.

Microcontroller	NRF52840
Frequency	64MHz
Flash	796KB
RAM	243KB
Vendor	Adafruit

Configuration

Please use adafruit_metro_nrf52840 ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[env:adafruit_metro_nrf52840]
platform = nordicnrf52
board = adafruit_metro_nrf52840
```

You can override default Metro nRF52840 Express settings per build environment using *board_**** option, where *** is a JSON object path from board manifest *adafruit_metro_nrf52840.json*. For example, *board_build.mcu*, *board_build.f_cpu*, etc.

```
[env:adafruit_metro_nrf52840]
platform = nordicnrf52
board = adafruit_metro_nrf52840

; change microcontroller
board_build.mcu = nrf52840

; change MCU frequency
board_build.f_cpu = 64000000L
```

Uploading

Metro nRF52840 Express supports the next uploading protocols:

- jlink
- nrfjprog
- nrfutil

Default protocol is *nrfutil*

You can change upload protocol using *upload_protocol* option:

```
[env:adafruit_metro_nrf52840]
platform = nordicnrf52
board = adafruit_metro_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*).

Metro nRF52840 Express 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

Name	Description
Arduin	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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.

Microcontroller	NRF52832
Frequency	64MHz
Flash	512KB
RAM	64KB
Vendor	Nordic

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
```

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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>CMSIS-DAP</i>	Yes	Yes
<i>J-LINK</i>	Yes	
<i>ST-LINK</i>		

Frameworks

Name	Description
<i>Arduino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>mbed</i>	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 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.

Microcontroller	NRF52840
Frequency	64MHz
Flash	1MB
RAM	256KB
Vendor	Nordic

Configuration

Please use `nrf52840_dk` ID for `board` option in “*platformio.ini*” (Project Configuration File):

```
[env:nrf52840_dk]
platform = nordicnrf52
board = nrf52840_dk
```

You can override default Nordic nRF52840-DK settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `nrf52840_dk.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:nrf52840_dk]
platform = nordicnrf52
board = nrf52840_dk

; change microcontroller
board_build.mcu = nrf52840

; change MCU frequency
board_build.f_cpu = 64000000L
```

Uploading

Nordic nRF52840-DK supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed
- nrfjprog
- stlink

Default protocol is `jlink`

You can change upload protocol using `upload_protocol` option:

```
[env:nrf52840_dk]
platform = nordicnrf52
board = nrf52840_dk

upload_protocol = jlink
```

Debugging

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

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for 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
<i>Black Magic Probe</i>		
<i>CMSIS-DAP</i>	Yes	Yes
<i>J-LINK</i>	Yes	
<i>ST-LINK</i>		

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>mbed</i>	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 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.

Microcontroller	NRF52840
Frequency	64MHz
Flash	796KB
RAM	243KB
Vendor	Nordic

Configuration

Please use `nrf52840_dk_adafruit` ID for `board` option in “*platformio.ini*” (*Project Configuration File*):

```
[env:nrf52840_dk_adafruit]
platform = nordicnrf52
board = nrf52840_dk_adafruit
```

You can override default Nordic nRF52840-DK (Adafruit BSP) settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `nrf52840_dk_adafruit.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:nrf52840_dk_adafruit]
platform = nordicnrf52
board = nrf52840_dk_adafruit

; change microcontroller
board_build.mcu = nrf52840

; change MCU frequency
board_build.f_cpu = 64000000L
```

Uploading

Nordic nRF52840-DK (Adafruit BSP) supports the next uploading protocols:

- jlink
- nrfjprog

Default protocol is `jlink`

You can change upload protocol using `upload_protocol` option:

```
[env:nrf52840_dk_adafruit]
platform = nordicnrf52
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

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.

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.

Microcontroller	NRF52832
Frequency	64MHz
Flash	512KB
RAM	64KB
Vendor	RedBearLab

Configuration

Please use `redbear_blenano2` ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[env:redbear_blenano2]
platform = nordicnrf52
board = redbear_blenano2
```

You can override default RedBearLab BLE Nano 2 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `redbear_blenano2.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:redbear_blenano2]
platform = nordicnrf52
board = redbear_blenano2

; change microcontroller
board_build.mcu = nrf52832

; change MCU frequency
board_build.f_cpu = 64000000L
```

Uploading

RedBearLab BLE Nano 2 supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- nrfjprog
- stlink

Default protocol is `cmsis-dap`

You can change upload protocol using `upload_protocol` option:

```
[env:redbear_blenano2]
platform = nordicnrf52
board = redbear_blenano2
```

(continues on next page)

(continued from previous page)

```
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 2 has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>CMSIS-DAP</i>	Yes	Yes
<i>J-LINK</i>		
<i>ST-LINK</i>		

Frameworks

Name	Description
<i>Arduino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>mbed</i>	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.

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.

Microcontroller	NRF52832
Frequency	64MHz
Flash	512KB
RAM	64KB
Vendor	RedBearLab

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	On-board	Default
<i>Black Magic Probe</i>		
<i>CMSIS-DAP</i>	Yes	Yes
<i>J-LINK</i>		
<i>ST-LINK</i>		

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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.

Microcontroller	NRF52832
Frequency	64MHz
Flash	512KB
RAM	64KB
Vendor	Sigma Delta Technologies

Configuration

Please use `sdt52832b` ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[env:sdt52832b]
platform = nordicnrf52
board = sdt52832b
```

You can override default SDT52832B settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `sdt52832b.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:sdt52832b]
platform = nordicnrf52
board = sdt52832b

; change microcontroller
board_build.mcu = nrf52832

; change MCU frequency
board_build.f_cpu = 64000000L
```

Uploading

SDT52832B supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed
- nrfjprog
- stlink

Default protocol is `mbed`

You can change upload protocol using `upload_protocol` option:

```
[env:sdt52832b]
platform = nordicnrf52
board = sdt52832b

upload_protocol = mbed
```

Debugging

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

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for 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
<i>Black Magic Probe</i>		Yes
<i>CMSIS-DAP</i>		
<i>J-LINK</i>		
<i>ST-LINK</i>		

Frameworks

Name	Description
<i>mbed</i>	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.

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.

Microcontroller	NRF52832
Frequency	64MHz
Flash	512KB
RAM	64KB
Vendor	Taida Century

Configuration

Please use `stct_nrf52_minidev` ID for `board` option in “*platformio.ini*” (*Project Configuration File*):

```
[env:stct_nrf52_minidev]
platform = nordicnrf52
board = stct_nrf52_minidev
```

You can override default Taida Century nRF52 mini board settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `stct_nrf52_minidev.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[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:

```
[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
<i>Black Magic Probe</i>		Yes
<i>J-LINK</i>		
<i>ST-LINK</i>		

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

Xenon

Contents

- *Xenon*
 - *Hardware*
 - *Configuration*
 - *Uploading*
 - *Debugging*
 - *Frameworks*

Hardware

Platform *Nordic nRF52*: The nRF52 Series are built for speed to carry out increasingly complex tasks in the shortest possible time and return to sleep, conserving precious battery power. They have a Cortex-M4F processor and are the most capable Bluetooth Smart SoCs on the market.

Microcontroller	NRF52840
Frequency	64MHz
Flash	796KB
RAM	243KB
Vendor	Particle

Configuration

Please use `particle_xenon` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:particle_xenon]
platform = nordicnrf52
board = particle_xenon
```

You can override default Xenon settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `particle_xenon.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:particle_xenon]
platform = nordicnrf52
board = particle_xenon

; change microcontroller
board_build.mcu = nrf52840

; change MCU frequency
board_build.f_cpu = 64000000L
```

Uploading

Xenon supports the next uploading protocols:

- jlink
- nrfjprog
- nrfutil

Default protocol is `nrfutil`

You can change upload protocol using `upload_protocol` option:

```
[env:particle_xenon]
platform = nordicnrf52
board = particle_xenon

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*).

Xenon does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

Compatible Tools	On-board	Default
J-LINK		Yes

Frameworks

Name	Description
Ar-duino	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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.

Microcontroller	NRF52832
Frequency	64MHz
Flash	512KB
RAM	64KB
Vendor	Electronut Labs

Configuration

Please use hackaBLE ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:hackaBLE]
platform = nordicnrf52
board = hackaBLE
```

You can override default hackaBLE settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `hackaBLE.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:hackaBLE]
platform = nordicnrf52
board = hackaBLE

; change microcontroller
board_build.mcu = nrf52832

; change MCU frequency
board_build.f_cpu = 64000000L
```

Uploading

hackaBLE supports the next uploading protocols:

- blackmagic
- jlink
- nrfjprog
- stlink

Default protocol is `jlink`

You can change upload protocol using `upload_protocol` option:

```
[env:hackaBLE]
platform = nordicnrf52
board = hackaBLE

upload_protocol = jlink
```

Debugging

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

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for 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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		Yes
<i>J-LINK</i>		
<i>ST-LINK</i>		

Frameworks

Name	Description
Arduin	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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.

Microcontroller	NRF52832
Frequency	64MHz
Flash	512KB
RAM	64KB
Vendor	u-blox

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

; 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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>J-LINK</i>	Yes	Yes
<i>ST-LINK</i>		

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>mbed</i>	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.

1.12.18 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.

Microcontroller	LPC11U24
Frequency	48MHz
Flash	32KB
RAM	8KB
Vendor	NXP

Configuration

Please use `lpc11u24_301` ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[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.

```
[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:

```
[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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>CMSIS-DAP</i>	Yes	Yes
<i>J-LINK</i>		

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.

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.

Microcontroller	LPC4330
Frequency	204MHz
Flash	8MB
RAM	264KB
Vendor	Micromint

Configuration

Please use `lpc4330_m4` ID for `board` option in “`platformio.ini`” (Project Configuration File):

```
[env:lpc4330_m4]
platform = nxplpc
board = lpc4330_m4
```

You can override default Bambino-210E settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `lpc4330_m4.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:lpc4330_m4]
platform = nxplpc
board = lpc4330_m4

; change microcontroller
board_build.mcu = lpc4330

; change MCU frequency
board_build.f_cpu = 204000000L
```

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_m4

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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>CMSIS-DAP</i>	Yes	Yes
<i>J-LINK</i>		

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.

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.

Microcontroller	LPC11U35
Frequency	48MHz
Flash	64KB
RAM	10KB
Vendor	CQ Publishing

Configuration

Please use `lpc11u35_501` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:lpc11u35_501]
platform = nxplpc
board = lpc11u35_501
```

You can override default CQ Publishing TG-LPC11U35-501 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `lpc11u35_501.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:lpc11u35_501]
platform = nxplpc
board = lpc11u35_501

; change microcontroller
board_build.mcu = lpc11u35

; change MCU frequency
board_build.f_cpu = 48000000L
```

Uploading

CQ Publishing TG-LPC11U35-501 supports the next uploading protocols:

- blackmagic
- jlink
- mbed

Default protocol is mbed

You can change upload protocol using *upload_protocol* option:

```
[env: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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		Yes
<i>J-LINK</i>		

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.

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.

Microcontroller	LPC812
Frequency	30MHz
Flash	16KB
RAM	4KB
Vendor	Elektor Labs

Configuration

Please use `elektor_cocorico` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:elektor_cocorico]
platform = nxplpc
board = elektor_cocorico
```

You can override default CoCo-ri-Co! settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `elektor_cocorico.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:elektor_cocorico]
platform = nxplpc
board = elektor_cocorico

; change microcontroller
board_build.mcu = lpc812

; change MCU frequency
board_build.f_cpu = 30000000L
```

Uploading

CoCo-ri-Co! supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed

Default protocol is mbed

You can change upload protocol using *upload_protocol* option:

```
[env:elektor_cocorico]
platform = nxplpc
board = elektor_cocorico

upload_protocol = mbed
```

Debugging

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

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for 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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>CMSIS-DAP</i>	Yes	Yes
<i>J-LINK</i>		

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.

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.

Microcontroller	LPC1347
Frequency	72MHz
Flash	64KB
RAM	12KB
Vendor	Solder Splash Labs

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.

```
[env:lpc1347]
platform = nxplpc
board = lpc1347

; change microcontroller
board_build.mcu = lpc1347

; change MCU frequency
board_build.f_cpu = 72000000L
```

Uploading

DipCortex M3 supports the next uploading protocols:

- jlink
- mbed

Default protocol is mbed

You can change upload protocol using *upload_protocol* option:

```
[env:lpc1347]
platform = nxplpc
board = lpc1347

upload_protocol = mbed
```

Debugging

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

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for 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.

Compatible Tools	On-board	Default
J-LINK		Yes

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.

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.

Microcontroller	LPC11U35
Frequency	48MHz
Flash	64KB
RAM	10KB
Vendor	Embedded Artists

Configuration

Please use `lpc11u35` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:lpc11u35]
platform = nxplpc
board = lpc11u35
```

You can override default EA LPC11U35 QuickStart Board settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `lpc11u35.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:lpc11u35]
platform = nxplpc
board = lpc11u35

; change microcontroller
board_build.mcu = lpc11u35

; change MCU frequency
board_build.f_cpu = 48000000L
```

Uploading

EA LPC11U35 QuickStart Board supports the next uploading protocols:

- blackmagic
- jlink
- mbed

Default protocol is mbed

You can change upload protocol using *upload_protocol* option:

```
[env:lpc11u35]
platform = nxplpc
board = lpc11u35

upload_protocol = mbed
```

Debugging

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

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for 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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		Yes
<i>J-LINK</i>		

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.

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.

Microcontroller	LPC4088
Frequency	120MHz
Flash	512KB
RAM	96KB
Vendor	Embedded Artists

Configuration

Please use `lpc4088_dm` ID for `board` option in “`platformio.ini`” (Project Configuration File):

```
[env:lpc4088_dm]
platform = nxplpc
board = lpc4088_dm
```

You can override default Embedded Artists LPC4088 Display Module settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `lpc4088_dm.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:lpc4088_dm]
platform = nxplpc
board = lpc4088_dm

; change microcontroller
board_build.mcu = lpc4088

; change MCU frequency
board_build.f_cpu = 120000000L
```

Uploading

Embedded Artists LPC4088 Display Module supports the next uploading protocols:

- cmsis-dap
- jlink
- mbed

Default protocol is mbed

You can change upload protocol using *upload_protocol* option:

```
[env:lpc4088_dm]
platform = nxplpc
board = lpc4088_dm

upload_protocol = mbed
```

Debugging

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

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for 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.

Compatible Tools	On-board	Default
CMSIS-DAP	Yes	Yes
J-LINK		

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.

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.

Microcontroller	LPC4088
Frequency	120MHz
Flash	512KB
RAM	96KB
Vendor	Embedded Artists

Configuration

Please use `lpc4088` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:lpc4088]
platform = nxplpc
board = lpc4088
```

You can override default Embedded Artists LPC4088 QuickStart Board settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `lpc4088.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:lpc4088]
platform = nxplpc
board = lpc4088

; change microcontroller
board_build.mcu = lpc4088

; change MCU frequency
board_build.f_cpu = 120000000L
```

Uploading

Embedded Artists LPC4088 QuickStart Board supports the next uploading protocols:

- cmsis-dap
- jlink
- mbed

Default protocol is mbed

You can change upload protocol using *upload_protocol* option:

```
[env:lpc4088]
platform = nxplpc
board = lpc4088

upload_protocol = mbed
```

Debugging

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

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for 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.

Compatible Tools	On-board	Default
CMSIS-DAP	Yes	Yes
J-LINK		

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.

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.

Microcontroller	LPC11U68
Frequency	50MHz
Flash	256KB
RAM	36KB
Vendor	NXP

Configuration

Please use `lpc11u68` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:lpc11u68]
platform = nxplpc
board = lpc11u68
```

You can override default LPCXpresso11U68 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `lpc11u68.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:lpc11u68]
platform = nxplpc
board = lpc11u68

; change microcontroller
board_build.mcu = lpc11u68

; change MCU frequency
board_build.f_cpu = 50000000L
```

Uploading

LPCXpresso11U68 supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed

Default protocol is mbed

You can change upload protocol using *upload_protocol* option:

```
[env:lpc11u68]
platform = nxplpc
board = lpc11u68

upload_protocol = mbed
```

Debugging

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

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for 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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>CMSIS-DAP</i>	Yes	Yes
<i>J-LINK</i>		

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.

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.

Microcontroller	LPC824
Frequency	30MHz
Flash	32KB
RAM	8KB
Vendor	NXP

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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>CMSIS-DAP</i>	Yes	Yes
<i>J-LINK</i>		

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.

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.

Microcontroller	LPC11U34
Frequency	48MHz
Flash	48KB
RAM	10KB
Vendor	AppNearMe

Configuration

Please use `micronfcboard` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:micronfcboard]
platform = nxplpc
board = micronfcboard
```

You can override default MicroNFCBoard settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `micronfcboard.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:micronfcboard]
platform = nxplpc
board = micronfcboard

; change microcontroller
board_build.mcu = lpc11u34

; change MCU frequency
board_build.f_cpu = 48000000L
```

Debugging

PIO Unified Debugger currently does not support MicroNFCBoard board.

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.

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.

Microcontroller	LPC11U24
Frequency	48MHz
Flash	32KB
RAM	8KB
Vendor	NGX Technologies

Configuration

Please use `blueboard_lpc11u24` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:blueboard_lpc11u24]
platform = nxplpc
board = blueboard_lpc11u24
```

You can override default NGX Technologies BlueBoard-LPC11U24 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `blueboard_lpc11u24.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:blueboard_lpc11u24]
platform = nxplpc
board = blueboard_lpc11u24

; change microcontroller
board_build.mcu = lpc11u24

; change MCU frequency
board_build.f_cpu = 48000000L
```

Uploading

NGX Technologies BlueBoard-LPC11U24 supports the next uploading protocols:

- blackmagic
- jlink
- mbed

Default protocol is `mbed`

You can change upload protocol using `upload_protocol` option:

```
[env:blueboard_lpc11u24]
platform = nxplpc
board = blueboard_lpc11u24

upload_protocol = mbed
```

Debugging

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

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for 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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		Yes
<i>J-LINK</i>		

Frameworks

Name	Description
<i>mbed</i>	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 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.

Microcontroller	LPC11C24
Frequency	48MHz
Flash	32KB
RAM	8KB
Vendor	NXP

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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		Yes
<i>J-LINK</i>		

Frameworks

Name	Description
<i>mbed</i>	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.

Microcontroller	LPC11U34
Frequency	48MHz
Flash	40KB
RAM	8KB
Vendor	NXP

Configuration

Please use `lpc11u34_421` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:lpc11u34_421]
platform = nxplpc
board = lpc11u34_421
```

You can override default NXP LPC11U34 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `lpc11u34_421.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:lpc11u34_421]
platform = nxplpc
board = lpc11u34_421

; change microcontroller
board_build.mcu = lpc11u34

; change MCU frequency
board_build.f_cpu = 48000000L
```

Uploading

NXP LPC11U34 supports the next uploading protocols:

- blackmagic
- jlink
- mbed

Default protocol is `mbed`

You can change upload protocol using `upload_protocol` option:

```
[env:lpc11u34_421]
platform = nxplpc
board = lpc11u34_421

upload_protocol = mbed
```

Debugging

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

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using *debug_tool* option in “*platformio.ini*” (*Project Configuration File*).

NXP LPC11U34 does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		Yes
<i>J-LINK</i>		

Frameworks

Name	Description
<i>mbed</i>	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 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.

Microcontroller	LPC11U37
Frequency	48MHz
Flash	128KB
RAM	10KB
Vendor	NXP

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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		Yes
<i>J-LINK</i>		

Frameworks

Name	Description
<i>mbed</i>	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 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.

Microcontroller	LPC812
Frequency	30MHz
Flash	16KB
RAM	4KB
Vendor	NXP

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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>CMSIS-DAP</i>	Yes	Yes
<i>J-LINK</i>		

Frameworks

Name	Description
<i>mbed</i>	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 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.

Microcontroller	LPC1549
Frequency	72MHz
Flash	256KB
RAM	36KB
Vendor	NXP

Configuration

Please use lpc1549 ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[env:lpc1549]
platform = nxplpc
board = lpc1549
```

You can override default NXP LPCXpresso1549 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `lpc1549.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:lpc1549]
platform = nxplpc
board = lpc1549

; change microcontroller
board_build.mcu = lpc1549

; change MCU frequency
board_build.f_cpu = 72000000L
```

Uploading

NXP LPCXpresso1549 supports the next uploading protocols:

- blackmagic
- jlink
- mbed

Default protocol is mbed

You can change upload protocol using `upload_protocol` option:

```
[env:lpc1549]
platform = nxplpc
board = lpc1549

upload_protocol = mbed
```

Debugging

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

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for 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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		Yes
<i>J-LINK</i>		

Frameworks

Name	Description
<i>mbed</i>	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 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.

Microcontroller	LPC54114J256BD64
Frequency	100MHz
Flash	256KB
RAM	192KB
Vendor	NXP

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.

Compatible Tools	On-board	Default
CMSIS-DAP	Yes	Yes
J-LINK		

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 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.

Microcontroller	LPC54608ET512
Frequency	180MHz
Flash	512KB
RAM	200KB
Vendor	NXP

Configuration

Please use lpc546xx ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[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.

```
[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:

```
[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.

Compatible Tools	On-board	Default
J-LINK	Yes	Yes

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 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.

Microcontroller	LPC11U24
Frequency	48MHz
Flash	32KB
RAM	8KB
Vendor	NXP

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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>CMSIS-DAP</i>	Yes	Yes
<i>J-LINK</i>		

Frameworks

Name	Description
<i>mbed</i>	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 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.

Microcontroller	LPC1768
Frequency	96MHz
Flash	512KB
RAM	64KB
Vendor	NXP

Configuration

Please use `lpc1768` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:lpc1768]
platform = nxplpc
board = lpc1768
```

You can override default NXP mbed LPC1768 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `lpc1768.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[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:

```
[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	On-board	Default
<i>Black Magic Probe</i>		
<i>CMSIS-DAP</i>	Yes	Yes
<i>J-LINK</i>		

Frameworks

Name	Description
<i>mbed</i>	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.

Outrageous Circuits mBuino

Contents

- *Outrageous Circuits mBuino*
 - *Hardware*
 - *Configuration*
 - *Debugging*
 - *Frameworks*

Hardware

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

Microcontroller	LPC11U24
Frequency	48MHz
Flash	32KB
RAM	8KB
Vendor	Outrageous Circuits

Configuration

Please use `mbuino` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:mbuino]
platform = nxplpc
board = mbuino
```

You can override default Outrageous Circuits mBuino settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `mbuino.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:mbuino]
platform = nxplpc
board = mbuino

; change microcontroller
board_build.mcu = lpc11u24

; change MCU frequency
board_build.f_cpu = 48000000L
```

Debugging

PIO Unified Debugger currently does not support Outrageous Circuits mBuino board.

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 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	LPC11U37
Frequency	48MHz
Flash	128KB
RAM	10KB
Vendor	SeeedStudio

Configuration

Please use `seeedArchGPRS` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[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.

```
[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

Name	Description
<code>mbed</code>	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 Arch Pro

Contents

- *Seeed Arch Pro*
 - *Hardware*

- Configuration
- Uploading
- Debugging
- Frameworks

Hardware

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

Microcontroller	LPC1768
Frequency	96MHz
Flash	512KB
RAM	64KB
Vendor	SeeedStudio

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:

```
[env:seeedArchPro]
platform = nxplpc
board = seeedArchPro

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 Pro has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

Compatible Tools	On-board	Default
CMSIS-DAP	Yes	Yes

Frameworks

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.

Microcontroller	LPC11U35
Frequency	48MHz
Flash	64KB
RAM	10KB
Vendor	SeeedStudio

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

Name	Description
<code>mbed</code>	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.

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.

Microcontroller	LPC1768
Frequency	96MHz
Flash	512KB
RAM	32KB
Vendor	Smeshlink

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

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.

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.

Microcontroller	LPC11U24
Frequency	50MHz
Flash	32KB
RAM	8KB
Vendor	Solder Splash Labs

Configuration

Please use `dipcortexm0` ID for `board` option in “`platformio.ini`” (Project Configuration File):

```
[env:dipcortexm0]
platform = nxplpc
board = dipcortexm0
```

You can override default Solder Splash Labs DipCortex M0 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `dipcortexm0.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:dipcortexm0]
platform = nxplpc
board = dipcortexm0

; change microcontroller
board_build.mcu = lpc11u24

; change MCU frequency
board_build.f_cpu = 50000000L
```

Uploading

Solder Splash Labs DipCortex M0 supports the next uploading protocols:

- blackmagic
- jlink
- mbed

Default protocol is mbed

You can change upload protocol using *upload_protocol* option:

```
[env:dipcortexm0]
platform = nxplpc
board = dipcortexm0

upload_protocol = mbed
```

Debugging

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

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for 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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		Yes
<i>J-LINK</i>		

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.

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.

Microcontroller	LPC1114FN28
Frequency	48MHz
Flash	32KB
RAM	4KB
Vendor	Switch Science

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.

```
[env:lpc1114fn28]
platform = nxplpc
board = lpc1114fn28

; change microcontroller
board_build.mcu = lpc1114fn28

; change MCU frequency
board_build.f_cpu = 48000000L
```

Uploading

Switch Science mbed LPC1114FN28 supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed

Default protocol is mbed

You can change upload protocol using *upload_protocol* option:

```
[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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>CMSIS-DAP</i>	Yes	Yes
<i>J-LINK</i>		

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.

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.

Microcontroller	LPC824
Frequency	30MHz
Flash	32KB
RAM	8KB
Vendor	Switch Science

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.

```
[env:ssci824]
platform = nxplpc
board = ssci824

; change microcontroller
board_build.mcu = lpc824

; change MCU frequency
board_build.f_cpu = 30000000L
```

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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>CMSIS-DAP</i>	Yes	Yes
<i>J-LINK</i>		

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.

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.

Microcontroller	LPC11U24
Frequency	50MHz
Flash	32KB
RAM	10KB
Vendor	GHI Electronics

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)

(continued from previous 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

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.

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.

Microcontroller	LPC1768
Frequency	96MHz
Flash	512KB
RAM	64KB
Vendor	u-blox

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	On-board	Default
<i>Black Magic Probe</i>		
<i>CMSIS-DAP</i>	Yes	Yes
<i>J-LINK</i>		

Frameworks

Name	Description
<i>mbed</i>	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.

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.

Microcontroller	LPC11U35
Frequency	48MHz
Flash	64KB
RAM	10KB
Vendor	y5 design

Configuration

Please use `lpc11u35_y5_mbug` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:lpc11u35_y5_mbug]
platform = nxplpc
board = lpc11u35_y5_mbug
```

You can override default y5 LPC11U35 mbug settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `lpc11u35_y5_mbug.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[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:

```
[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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		Yes
<i>J-LINK</i>		

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.

1.12.19 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.

Microcontroller	GAP8
Frequency	250MHz
Flash	64MB
RAM	8MB
Vendor	GreenWaves Technologies

Configuration

Please use gapuino ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[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.

Compatible Tools	On-board	Default
<i>FTDI Chip</i>	Yes	Yes

Frameworks

Name	Description
<i>mbed</i>	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.
<i>PULP OS</i>	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.

1.12.20 Samsung ARTIK

Samsung ARTIK053

Contents

- *Samsung ARTIK053*
 - *Hardware*

- Configuration
- Debugging
- Frameworks

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.

Microcontroller	S5JT200
Frequency	320MHz
Flash	8MB
RAM	1.25MB
Vendor	Samsung

Configuration

Please use `artik_053` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:artik_053]
platform = samsung_artik
board = artik_053
```

You can override default Samsung ARTIK053 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `artik_053.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[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.

Compatible Tools	On-board	Default
FTDI Chip	Yes	Yes

Frameworks

Name	Description
Tizen RT	Tizen RT is a lightweight RTOS-based platform to support low-end IoT devices

1.12.21 Shakti

Artix-7 35T Arty FPGA Evaluation Kit

Contents

- [Artix-7 35T Arty FPGA Evaluation Kit](#)
 - [Hardware](#)
 - [Configuration](#)
 - [Uploading](#)
 - [Debugging](#)
 - [Frameworks](#)

Hardware

Platform [Shakti](#): Shakti is an open-source initiative by the RISE group at IIT-Madras, which is not only building open source, production grade processors, but also associated components like interconnect fabrics, verification tools, storage controllers, peripheral IPs and SOC tools.

Microcontroller	E-CLASS
Frequency	50MHz
Flash	0B
RAM	128KB
Vendor	Xilinx

Configuration

Please use `artix7_35t` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:artix7_35t]
platform = shakti
board = artix7_35t
```

You can override default Artix-7 35T Arty FPGA Evaluation Kit settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `artix7_35t.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:artix7_35t]
platform = shakti
board = artix7_35t

; change microcontroller
board_build.mcu = E-Class

; change MCU frequency
board_build.f_cpu = 50000000L
```

Uploading

Artix-7 35T Arty FPGA Evaluation Kit supports the next uploading protocols:

- ftdi
- ftdi
- jlink
- jlink

Default protocol is `ftdi`

You can change upload protocol using `upload_protocol` option:

```
[env:artix7_35t]
platform = shakti
board = artix7_35t

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*).

Artix-7 35T Arty FPGA Evaluation Kit has on-board debug probe and **IS READY** for debugging. You don't need to use/buy external debug probe.

Compatible Tools	On-board	Default
FTDI Chip	Yes	Yes
J-LINK		

Frameworks

Name	Description
Shakti SDK	A software development kit for developing applications on Shakti class of processors

Arty A7-100: Artix-7 FPGA Development Board

Contents

- [Arty A7-100: Artix-7 FPGA Development Board](#)
 - [Hardware](#)
 - [Configuration](#)
 - [Uploading](#)
 - [Debugging](#)
 - [Frameworks](#)

Hardware

Platform [Shakti](#): Shakti is an open-source initiative by the RISE group at IIT-Madras, which is not only building open source, production grade processors, but also associated components like interconnect fabrics, verification tools, storage controllers, peripheral IPs and SOC tools.

Microcontroller	C-CLASS
Frequency	50MHz
Flash	0B
RAM	128MB
Vendor	Xilinx

Configuration

Please use `artix7_100t` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:artix7_100t]
platform = shakti
board = artix7_100t
```

You can override default Arty A7-100: Artix-7 FPGA Development Board settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `artix7_100t.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:artix7_100t]
platform = shakti
board = artix7_100t

; change microcontroller
```

(continues on next page)

(continued from previous page)

```
board_build.mcu = C-Class
; change MCU frequency
board_build.f_cpu = 50000000L
```

Uploading

Arty A7-100: Artix-7 FPGA Development Board supports the next uploading protocols:

- ftdi
- ftdi
- jlink
- jlink

Default protocol is ftdi

You can change upload protocol using *upload_protocol* option:

```
[env:artix7_100t]
platform = shakti
board = artix7_100t

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 A7-100: Artix-7 FPGA Development Board has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

Compatible Tools	On-board	Default
FTDI Chip	Yes	Yes
J-LINK		

Frameworks

Name	Description
Shakti SDK	A software development kit for developing applications on Shakti class of processors

1.12.22 SiFive

Aryt FPGA Dev Kit

Contents

- *Aryt 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.

Microcontroller	FE310
Frequency	450MHz
Flash	16MB
RAM	256MB
Vendor	Xilinx

Configuration

Please use e310-arty ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[env:e310-arty]
platform = sifive
board = e310-arty
```

You can override default Arty FPGA Dev Kit settings per build environment using *board_**** option, where *** is a JSON object path from board manifest *e310-arty.json*. For example, *board_build.mcu*, *board_build.f_cpu*, etc.

```
[env:e310-arty]
platform = sifive
board = e310-arty

; change microcontroller
board_build.mcu = fe310

; change MCU frequency
board_build.f_cpu = 450000000L
```

Uploading

Aarty 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*).

Aarty 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	On-board	Default
<i>FTDI Chip</i>	Yes	Yes
<i>J-LINK</i>		
<i>Mini-Module FT2232H</i>		
<i>Olimex ARM-USB-OCD</i>		
<i>Olimex ARM-USB-OCD-H</i>		
<i>Olimex ARM-USB-TINY-H</i>		
<i>Olimex ARM-USB-TINY</i>		
<i>QEMU</i>	Yes	
<i>TIAO USB Multi-Protocol Adapter (TUMPA)</i>		

Frameworks

Name	Description
Freedom E SDK	Open Source Software for Developing on the SiFive Freedom E Platform

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.

Microcontroller	FU540
Frequency	1500MHz
Flash	32MB
RAM	8GB
Vendor	SiFive

Configuration

Please use hifive-unleashed ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[env:hifive-unleashed]
platform = sifive
board = hifive-unleashed
```

You can override default HiFive Unleashed settings per build environment using *board_**** option, where *** is a JSON object path from board manifest *hifive-unleashed.json*. For example, *board_build.mcu*, *board_build.f_cpu*, etc.

```
[env:hifive-unleashed]
platform = sifive
board = hifive-unleashed

; change microcontroller
board_build.mcu = fu540
```

(continues on next page)

(continued from previous page)

```
; 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.

Compatible Tools	On-board	Default
FTDI Chip	Yes	Yes
QEMU	Yes	

Frameworks

Name	Description
Freedom E SDK	Open Source Software for Developing on the SiFive Freedom E Platform

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.

Microcontroller	FE310
Frequency	320MHz
Flash	16MB
RAM	16KB
Vendor	SiFive

Configuration

Please use `hifive1` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:hifive1]
platform = sifive
board = hifive1
```

You can override default HiFive1 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `hifive1.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:hifive1]
platform = sifive
board = hifive1

; 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.

Compatible Tools	On-board	Default
<i>FTDI Chip</i>	Yes	Yes
<i>QEMU</i>	Yes	

Frameworks

Name	Description
<i>Freedom E SDK</i>	Open Source Software for Developing on the SiFive Freedom E Platform

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.

Microcontroller	FE310
Frequency	320MHz
Flash	16MB
RAM	16KB
Vendor	SiFive

Configuration

Please use `hifive1-revb` ID for `board` option in “*platformio.ini*” (*Project Configuration File*):

```
[env:hifive1-revb]
platform = sifive
board = hifive1-revb
```

You can override default HiFive1 Rev B settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `hifive1-revb.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:hifive1-revb]
platform = sifive
board = hifive1-revb

; 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 Rev B 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

Name	Description
Freedom E SDK	Open Source Software for Developing on the SiFive Freedom E Platform

1.12.23 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 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	EFM32GG990F1024
Frequency	48MHz
Flash	1MB
RAM	128KB
Vendor	Silicon Labs

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.

Compatible Tools	On-board	Default
Black Magic Probe		
J-LINK	Yes	Yes

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.

EFM32LG-STK3600 Leopard Gecko

Contents

- *EFM32LG-STK3600 Leopard 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	EFM32LG990F256
Frequency	48MHz
Flash	256KB
RAM	32KB
Vendor	Silicon Labs

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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>J-LINK</i>	Yes	Yes

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.

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.

Microcontroller	EFM32WG990F256
Frequency	48MHz
Flash	256KB
RAM	32KB
Vendor	Silicon Labs

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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>J-LINK</i>	Yes	Yes

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.

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.

Microcontroller	EFM32ZG222F32
Frequency	24MHz
Flash	32KB
RAM	4KB
Vendor	Silicon Labs

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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>J-LINK</i>	Yes	Yes

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.

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*):

```
[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.

```
[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:

```
[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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>J-LINK</i>	Yes	Yes

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.

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.

Microcontroller	EFM32PG1B200F256GM48
Frequency	40MHz
Flash	256KB
RAM	32KB
Vendor	Silicon Labs

Configuration

Please use `efm32pg_stk3401` ID for `board` option in “*platformio.ini*” (*Project Configuration File*):

```
[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.

```
[env:efm32pg_stk3401]
platform = siliconlabsefm32
board = efm32pg_stk3401

; change microcontroller
board_build.mcu = efm32pg1b200f256gm48

; change MCU frequency
board_build.f_cpu = 40000000L
```

Uploading

SLSTK3401A Pearl Gecko PG1 supports the next uploading protocols:

- blackmagic
- jlink
- mbed

Default protocol is mbed

You can change upload protocol using *upload_protocol* option:

```
[env:efm32pg_stk3401]
platform = siliconlabsefm32
board = efm32pg_stk3401

upload_protocol = mbed
```

Debugging

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

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for 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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>J-LINK</i>	Yes	Yes

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.

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.

Microcontroller	EFR32MG12P432F1024
Frequency	40MHz
Flash	1MB
RAM	256KB
Vendor	Silicon Labs

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.

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using *debug_tool* option in “*platformio.ini*” (*Project Configuration File*).

Thunderboard Sense 2 Sensor-to-Cloud Advanced IoT has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>J-LINK</i>	Yes	Yes

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.

1.12.24 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.

Microcontroller	STM32F415RGT
Frequency	168MHz
Flash	1MB
RAM	128KB
Vendor	1BitSquared

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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		Yes
<i>J-LINK</i>		
<i>ST-LINK</i>		

Frameworks

Name	Description
<i>CMSIS</i>	The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.
<i>libOpenCM3</i>	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.
<i>SPL</i>	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.
<i>STM32Cube</i>	STM32Cube embedded 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.

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.

Microcontroller	STM32F723IEK6
Frequency	216MHz
Flash	512KB
RAM	192KB
Vendor	ST

Configuration

Please use `disco_f723ie` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:disco_f723ie]
platform = ststm32
board = disco_f723ie
```

You can override default 32F723EDISCOVERY settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `disco_f723ie.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:disco_f723ie]
platform = ststm32
board = disco_f723ie

; change microcontroller
board_build.mcu = stm32f723iek6

; change MCU frequency
board_build.f_cpu = 216000000L
```

Uploading

32F723EDISCOVERY supports the next uploading protocols:

- blackmagic
- jlink
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```
[env:disco_f723ie]
platform = ststm32
board = disco_f723ie

upload_protocol = stlink
```

Debugging

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

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for 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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>J-LINK</i>		
<i>ST-LINK</i>	Yes	Yes

Frameworks

Name	Description
<i>STM32</i>	STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.

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.

Microcontroller	STM32F407VET6
Frequency	168MHz
Flash	512KB
RAM	192KB
Vendor	Armed

Configuration

Please use `armed_v1` ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[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.

```
[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:

```
[env:armed_v1]
platform = ststm32
board = armed_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 does 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
<i>Black Magic Probe</i>		Yes
<i>J-LINK</i>		
<i>ST-LINK</i>		

Frameworks

Name	Description
Arduino	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
CMSIS	The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.
STM32	STM32Cube embedded 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.

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.

Microcontroller	STM32F446RET6
Frequency	180MHz
Flash	512KB
RAM	128KB
Vendor	RUMBA

Configuration

Please use rumba32_f446ve ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[env:rumba32_f446ve]
platform = ststm32
board = rumba32_f446ve
```

You can override default 3D Printer control board settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `rumba32_f446ve.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:rumba32_f446ve]
platform = ststm32
board = rumba32_f446ve

; change microcontroller
board_build.mcu = stm32f446ret6

; change MCU frequency
board_build.f_cpu = 180000000L
```

Uploading

3D Printer control board supports the next uploading protocols:

- blackmagic
- jlink
- serial
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```
[env:rumba32_f446ve]
platform = ststm32
board = rumba32_f446ve

upload_protocol = stlink
```

Debugging

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

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for 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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		Yes
<i>J-LINK</i>		
<i>ST-LINK</i>		

Frameworks

Name	Description
<i>Ar-duino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>STM32</i>	STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.

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.

Microcontroller	STM32F765VIT6
Frequency	216MHz
Flash	2MB
RAM	512KB
Vendor	RemRam

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 = stm32f765vit6

; change MCU frequency
board_build.f_cpu = 21600000L
```

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.

Compatible Tools	On-board	Default
Black Magic Probe		
J-LINK		
ST-LINK	Yes	Yes

Frameworks

Name	Description
Ar-duino	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
STM32	STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.

3DP001V1 Evaluation board for 3D printer

Contents

- [*3DP001V1 Evaluation board for 3D printer*](#)
 - [*Hardware*](#)
 - [*Configuration*](#)
 - [*Uploading*](#)
 - [*Debugging*](#)
 - [*Frameworks*](#)

Hardware

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

Microcontroller	STM32F401VGT6
Frequency	84MHz
Flash	512KB
RAM	96KB
Vendor	ST

Configuration

Please use `st3dp001_eval` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>J-LINK</i>		
<i>ST-LINK</i>	Yes	Yes

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>STM32</i>	STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.

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.

Microcontroller	STM32F446VET6
Frequency	168MHz
Flash	512KB
RAM	128KB
Vendor	96Boards

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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>J-LINK</i>		
<i>ST-LINK</i>	Yes	Yes

Frameworks

Name	Description
<i>mbed</i>	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.
<i>STM32</i>	STM32Cube embedded 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.

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.

Microcontroller	STM32F417VGT6
Frequency	168MHz
Flash	1MB
RAM	192KB
Vendor	Armstrap

Configuration

Please use `armstrap_eagle1024` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[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.

```
[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:

```
[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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		Yes
<i>J-LINK</i>		
<i>ST-LINK</i>		

Frameworks

Name	Description
<i>CM-SIS</i>	The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.
<i>SPL</i>	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.
<i>STM32Cube</i>	STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.

Armstrap Eagle 2048

Contents

- *Armstrap Eagle 2048*
 - *Hardware*
 - *Configuration*
 - *Uploading*
 - *Debugging*
 - *Frameworks*

Hardware

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

Microcontroller	STM32F427VIT6
Frequency	168MHz
Flash	1.99MB
RAM	256KB
Vendor	Armstrap

Configuration

Please use `armstrap_eagle2048` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:armstrap_eagle2048]
platform = ststm32
board = armstrap_eagle2048
```

You can override default Armstrap Eagle 2048 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `armstrap_eagle2048.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:armstrap_eagle2048]
platform = ststm32
board = armstrap_eagle2048

; change microcontroller
board_build.mcu = stm32f427vit6

; change MCU frequency
board_build.f_cpu = 168000000L
```

Uploading

Armstrap Eagle 2048 supports the next uploading protocols:

- blackmagic
- jlink
- stlink

Default protocol is `blackmagic`

You can change upload protocol using `upload_protocol` option:

```
[env:armstrap_eagle2048]
platform = ststm32
board = armstrap_eagle2048

upload_protocol = blackmagic
```

Debugging

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

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for 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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		Yes
<i>J-LINK</i>		
<i>ST-LINK</i>		

Frameworks

Name	Description
<i>CM-SIS</i>	The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.
<i>SPL</i>	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.
<i>STM32Cube</i>	STM32Cube embedded 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.

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.

Microcontroller	STM32F407VET6
Frequency	168MHz
Flash	512KB
RAM	192KB
Vendor	Armstrap

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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		Yes
<i>J-LINK</i>		
<i>ST-LINK</i>		

Frameworks

Name	Description
<i>CMSIS</i>	The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.
<i>SPL</i>	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.
<i>STM32Cube</i>	STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.

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.

Microcontroller	STM32F407VET6
Frequency	168MHz
Flash	512KB
RAM	128KB
Vendor	ST

Configuration

Please use `black_f407ve` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:black_f407ve]
platform = ststm32
board = black_f407ve
```

You can override default Black STM32F407VE settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `black_f407ve.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:black_f407ve]
platform = ststm32
board = black_f407ve

; change microcontroller
board_build.mcu = stm32f407vet6

; change MCU frequency
board_build.f_cpu = 168000000L
```

Uploading

Black STM32F407VE supports the next uploading protocols:

- blackmagic
- jlink
- serial
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```
[env:black_f407ve]
platform = ststm32
board = black_f407ve

upload_protocol = stlink
```

Debugging

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

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for 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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		Yes
<i>J-LINK</i>		
<i>ST-LINK</i>		

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>CMSIS</i>	The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.
<i>STM32</i>	STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.

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.

Microcontroller	STM32F407VGT6
Frequency	168MHz
Flash	512KB
RAM	128KB
Vendor	ST

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

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.

Compatible Tools	On-board	Default
Black Magic Probe		Yes
J-LINK		
ST-LINK		

Frameworks

Name	Description
Arduin	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
CM-SIS	The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.
STM32	STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.

Black STM32F407ZE

Contents

- [Black STM32F407ZE](#)
 - [Hardware](#)
 - [Configuration](#)
 - [Uploading](#)
 - [Debugging](#)
 - [Frameworks](#)

Hardware

Platform [ST STM32](#): The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-

time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

Microcontroller	STM32F407ZET6
Frequency	168MHz
Flash	512KB
RAM	128KB
Vendor	ST

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:

```
[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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		Yes
<i>J-LINK</i>		
<i>ST-LINK</i>		

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>CMSIS</i>	The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.
<i>STM32</i>	STM32Cube embedded 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.

Black STM32F407ZE

Contents

- *Black STM32F407ZE*
 - *Hardware*
 - *Configuration*
 - *Uploading*
 - *Debugging*
 - *Frameworks*

Hardware

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

Microcontroller	STM32F407ZGT6
Frequency	168MHz
Flash	1MB
RAM	128KB
Vendor	ST

Configuration

Please use `black_f407zg` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:black_f407zg]
platform = ststm32
board = black_f407zg
```

You can override default Black 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.

```
[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:

```
[env:black_f407zg]
platform = ststm32
board = black_f407zg
```

(continues on next page)

(continued from previous 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 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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		Yes
<i>J-LINK</i>		
<i>ST-LINK</i>		

Frameworks

Name	Description
<i>Arduino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>CMSIS</i>	The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.
<i>STM32</i>	STM32Cube embedded 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.

BlackPill F103C8

Contents

- *BlackPill 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.

Microcontroller	STM32F103C8T6
Frequency	72MHz
Flash	64KB
RAM	20KB
Vendor	Generic

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:

```
[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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>J-LINK</i>		
<i>ST-LINK</i>		Yes

Frameworks

Name	Description
<i>Arduino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>CMSIS</i>	The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.
<i>libOpenCM3</i>	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.
<i>STM32Cube</i>	STM32Cube embedded 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.

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.

Microcontroller	STM32F103C8T6
Frequency	72MHz
Flash	128KB
RAM	20KB
Vendor	Generic

Configuration

Please use `blackpill_f103c8_128` ID for `board` option in “*platformio.ini*” (Project Configuration File):

```
[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.

```
[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:

```
[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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>J-LINK</i>		
<i>ST-LINK</i>		Yes

Frameworks

Name	Description
<i>Arduino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>CMSIS</i>	The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.
<i>libOpenCM3</i>	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.
<i>STM32</i>	STM32Cube embedded 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.

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.

Microcontroller	STM32F303CCT6
Frequency	72MHz
Flash	256KB
RAM	40KB
Vendor	RobotDyn

Configuration

Please use `robotdyn_blackpill_f303cc` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:robotdyn_blackpill_f303cc]
platform = ststm32
board = robotdyn_blackpill_f303cc
```

You can override default BlackPill F303CC settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `robotdyn_blackpill_f303cc.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:robotdyn_blackpill_f303cc]
platform = ststm32
board = robotdyn_blackpill_f303cc

; change microcontroller
board_build.mcu = stm32f303cct6

; change MCU frequency
board_build.f_cpu = 72000000L
```

Uploading

BlackPill F303CC supports the next uploading protocols:

- blackmagic
- jlink
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```
[env:robotdyn_blackpill_f303cc]
platform = ststm32
board = robotdyn_blackpill_f303cc

upload_protocol = stlink
```

Debugging

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

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for 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
<i>Black Magic Probe</i>		Yes
<i>J-LINK</i>		
<i>ST-LINK</i>		

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>STM32</i>	STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.

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.

Microcontroller	STM32F407VET6
Frequency	168MHz
Flash	512KB
RAM	128KB
Vendor	ST

Configuration

Please use `blue_f407ve_mini` ID for `board` option in “*platformio.ini*” (*Project Configuration File*):

```
[env:blue_f407ve_mini]
platform = ststm32
board = blue_f407ve_mini
```

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.

```
[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:

```
[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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		Yes
<i>J-LINK</i>		
<i>ST-LINK</i>		

Frameworks

Name	Description
<i>Arduino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>CMSIS</i>	The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.
<i>STM32</i>	STM32Cube embedded 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.

BluePill F103C6

Contents

- *BluePill F103C6*
 - *Hardware*
 - *Configuration*
 - *Uploading*
 - *Debugging*
 - *Frameworks*

Hardware

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

Microcontroller	STM32F103C6T6
Frequency	72MHz
Flash	32KB
RAM	10KB
Vendor	Generic

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.

Compatible Tools	On-board	Default
Black Magic Probe		
J-LINK		
ST-LINK		Yes

Frameworks

Name	Description
Arduino	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
CMSIS	The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.
STM32	STM32Cube embedded 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.

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.

Microcontroller	STM32F103C8T6
Frequency	72MHz
Flash	64KB
RAM	20KB
Vendor	Generic

Configuration

Please use `bluepill_f103c8` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[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.

```
[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:

```
[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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>J-LINK</i>		
<i>ST-LINK</i>		Yes

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>CMSIS</i>	The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.
<i>libOpenCM3</i>	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.
<i>mbed</i>	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.
<i>STM32</i>	STM32Cube embedded 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.

BluePill F103C8 (128k)

Contents

- *BluePill F103C8 (128k)*
 - *Hardware*
 - *Configuration*
 - *Uploading*
 - *Debugging*
 - *Frameworks*

Hardware

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

Microcontroller	STM32F103C8T6
Frequency	72MHz
Flash	128KB
RAM	20KB
Vendor	Generic

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 = ststm32
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.

Compatible Tools	On-board	Default
Black Magic Probe		
J-LINK		
ST-LINK		Yes

Frameworks

Name	Description
Arduin	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
CM-SIS	The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.
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.
STM32	STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.

Demo F030F4

Contents

- [Demo F030F4](#)
 - [Hardware](#)
 - [Configuration](#)
 - [Uploading](#)
 - [Debugging](#)
 - [Frameworks](#)

Hardware

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

Microcontroller	STM32F030F4P6
Frequency	48MHz
Flash	16KB
RAM	4KB
Vendor	Generic

Configuration

Please use `demo_f030f4` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:demo_f030f4]
platform = ststm32
board = demo_f030f4
```

You can override default Demo F030F4 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `demo_f030f4.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:demo_f030f4]
platform = ststm32
board = demo_f030f4

; change microcontroller
board_build.mcu = stm32f030f4p6

; change MCU frequency
board_build.f_cpu = 48000000L
```

Uploading

Demo F030F4 supports the next uploading protocols:

- blackmagic
- jlink
- serial
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```
[env:demo_f030f4]
platform = ststm32
board = demo_f030f4
```

(continues on next page)

(continued from previous 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*).

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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		Yes
<i>J-LINK</i>		
<i>ST-LINK</i>		

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>STM32</i>	STM32Cube embedded 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.

Espotel LoRa Module

Contents

- *Espotel LoRa Module*
 - *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	STM32F411RET6
Frequency	100MHz
Flash	512KB
RAM	128KB
Vendor	Espotel

Configuration

Please use `elmo_f411re` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:elmo_f411re]
platform = ststm32
board = elmo_f411re
```

You can override default Espotel LoRa Module settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `elmo_f411re.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:elmo_f411re]
platform = ststm32
board = elmo_f411re

; change microcontroller
board_build.mcu = stm32f411ret6

; change MCU frequency
board_build.f_cpu = 100000000L
```

Uploading

Espotel LoRa Module supports the next uploading protocols:

- blackmagic
- jlink
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```
[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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>J-LINK</i>		
<i>ST-LINK</i>		Yes

Frameworks

Name	Description
<i>mbed</i>	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.
<i>STM32</i>	STM32Cube embedded 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.

F407VG

Contents

- *F407VG*
 - *Hardware*
 - *Configuration*
 - *Uploading*
 - *Debugging*
 - *Frameworks*

Hardware

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

Microcontroller	STM32F407VGT6
Frequency	168MHz
Flash	512KB
RAM	128KB
Vendor	Diymore

Configuration

Please use `diymore_f407vgt` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:diymore_f407vgt]
platform = ststm32
board = diymore_f407vgt
```

You can override default F407VG settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `diymore_f407vgt.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:diymore_f407vgt]
platform = ststm32
board = diymore_f407vgt

; change microcontroller
board_build.mcu = stm32f407vgt6

; change MCU frequency
board_build.f_cpu = 168000000L
```

Uploading

F407VG supports the next uploading protocols:

- blackmagic
- jlink
- serial
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```
[env:diymore_f407vgt]
platform = ststm32
board = diymore_f407vgt
```

(continues on next page)

(continued from previous 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*).

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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		Yes
<i>J-LINK</i>		
<i>ST-LINK</i>		

Frameworks

Name	Description
<i>Arduino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>CMSIS</i>	The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.
<i>STM32</i>	STM32Cube embedded 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.

FK407M1

Contents

- *FK407M1*
 - *Hardware*
 - *Configuration*
 - *Uploading*

- *Debugging*
- *Frameworks*

Hardware

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

Microcontroller	STM32F407VET6
Frequency	168MHz
Flash	512KB
RAM	128KB
Vendor	Generic

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 = ststm32
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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		Yes
<i>J-LINK</i>		
<i>ST-LINK</i>		

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>CMSIS</i>	The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.
<i>STM32</i>	STM32Cube embedded 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.

L476DMW1K

Contents

- *L476DMW1K*
 - *Hardware*

- Configuration
- Uploading
- Debugging
- Frameworks

Hardware

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

Microcontroller	STM32L476VGT6
Frequency	80MHz
Flash	1MB
RAM	128KB
Vendor	rhomb.io

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_1476dmw1k]
platform = ststm32
board = rhombio_1476dmw1k

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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>CMSIS-DAP</i>	Yes	Yes
<i>J-LINK</i>		
<i>ST-LINK</i>		

Frameworks

Name	Description
<i>mbed</i>	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.
<i>STM32</i>	STM32Cube embedded 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.

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.

Microcontroller	STM32F070CBT6
Frequency	48MHz
Flash	120KB
RAM	14.81KB
Vendor	Malyan

Configuration

Please use `malyanm200_f070cb` ID for `board` option in “*platformio.ini*” (*Project Configuration File*):

```
[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.

```
[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:

```
[env:malyanm200_f070cb]
platform = ststm32
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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		Yes
<i>J-LINK</i>		
<i>ST-LINK</i>		

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>STM32</i>	STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.

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.

Microcontroller	STM32F411RET6
Frequency	100MHz
Flash	512KB
RAM	128KB
Vendor	MultiTech

Configuration

Please use `mts_dragonfly_f411re` ID for `board` option in “*platformio.ini*” (Project Configuration File):

```
[env:mts_dragonfly_f411re]
platform = ststm32
board = mts_dragonfly_f411re
```

You can override default MTS Dragonfly settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `mts_dragonfly_f411re.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[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:

```
[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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		Yes
<i>J-LINK</i>		
<i>ST-LINK</i>		

Frameworks

Name	Description
<i>mbed</i>	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.
<i>STM32</i>	STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.

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.

Microcontroller	STM32F103CBT6
Frequency	72MHz
Flash	120KB
RAM	20KB
Vendor	Malyan

Configuration

Please use `malyanm200_f103cb` ID for `board` option in “*platformio.ini*” (*Project Configuration File*):

```
[env:malyanm200_f103cb]
platform = ststm32
board = malyanm200_f103cb
```

You can override default Malyan M200 V1 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `malyanm200_f103cb.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:malyanm200_f103cb]
platform = ststm32
board = malyanm200_f103cb

; change microcontroller
board_build.mcu = stm32f103cbt6

; change MCU frequency
board_build.f_cpu = 72000000L
```

Uploading

Malyan M200 V1 supports the next uploading protocols:

- blackmagic
- jlink
- serial
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```
[env:malyanm200_f103cb]
platform = ststm32
board = malyanm200_f103cb

upload_protocol = stlink
```

Debugging

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

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for 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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		Yes
<i>J-LINK</i>		
<i>ST-LINK</i>		

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>STM32</i>	STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.

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.

Microcontroller	STM32F103RBT6
Frequency	72MHz
Flash	108KB
RAM	17KB
Vendor	LeafLabs

Configuration

Please use `maple` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[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.

```
[env:maple]
platform = ststm32
board = maple

; change microcontroller
board_build.mcu = stm32f103rbt6

; change MCU frequency
board_build.f_cpu = 72000000L
```

Uploading

Maple supports the next uploading protocols:

- blackmagic
- dfu
- jlink
- stlink

Default protocol is `dfu`

You can change upload protocol using `upload_protocol` option:

```
[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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		Yes
<i>J-LINK</i>		
<i>ST-LINK</i>		

Frameworks

Name	Description
<i>Ar-duino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>libOpenCM3</i>	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.
<i>STM32Cube</i>	STM32Cube embedded 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.

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.

Microcontroller	STM32F103RET6
Frequency	72MHz
Flash	256KB
RAM	48KB
Vendor	LeafLabs

Configuration

Please use `maple_ret6` ID for `board` option in “*platformio.ini*” (*Project Configuration File*):

```
[env:maple_ret6]
platform = ststm32
board = maple_ret6
```

You can override default Maple (RET6) settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `maple_ret6.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:maple_ret6]
platform = ststm32
board = maple_ret6

; change microcontroller
board_build.mcu = stm32f103ret6

; change MCU frequency
board_build.f_cpu = 72000000L
```

Uploading

Maple (RET6) supports the next uploading protocols:

- blackmagic
- dfu
- jlink
- stlink

Default protocol is `dfu`

You can change upload protocol using `upload_protocol` option:

```
[env:maple_ret6]
platform = ststm32
board = maple_ret6
```

(continues on next page)

(continued from previous page)

```
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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		Yes
<i>J-LINK</i>		
<i>ST-LINK</i>		

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>li-bOpenCortex-M0(+)/M3/M4</i>	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.
<i>STM32</i>	STM32Cube embedded 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.

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.

Microcontroller	STM32F103CBT6
Frequency	72MHz
Flash	120KB
RAM	20KB
Vendor	LeafLabs

Configuration

Please use `maple_mini_b20` ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[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.

```
[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:

```
[env:maple_mini_b20]
platform = ststm32
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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		Yes
<i>J-LINK</i>		
<i>ST-LINK</i>		

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>li-bOpen</i>	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.
<i>STM32</i>	STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.

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.

Microcontroller	STM32F103CBT6
Frequency	72MHz
Flash	108KB
RAM	17KB
Vendor	LeafLabs

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:

```
[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 external probe listed below.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		Yes
<i>J-LINK</i>		
<i>ST-LINK</i>		

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>libOpenCM3</i>	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.
<i>STM32</i>	STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.

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.

Microcontroller	STM32F439ZIY6
Frequency	168MHz
Flash	2MB
RAM	256KB
Vendor	u-blox

Configuration

Please use `mbed_connect_odin` ID for `board` option in “*platformio.ini*” (*Project Configuration File*):

```
[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.

```
[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:

```
[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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>CMSIS-DAP</i>	Yes	Yes
<i>J-LINK</i>		
<i>ST-LINK</i>		

Frameworks

Name	Description
<code>mbed</code>	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.
<code>STM32</code>	STM32Cube embedded 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.

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.

Microcontroller	STM32F103CBT6
Frequency	72MHz
Flash	105.47KB
RAM	16.60KB
Vendor	Microduino

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.

```
[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:

```
[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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		Yes
<i>J-LINK</i>		
<i>ST-LINK</i>		

Frameworks

Name	Description
<i>Ar-duino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>STM32</i>	STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.

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.

Microcontroller	STM32F412ZGT6
Frequency	100MHz
Flash	1MB
RAM	256KB
Vendor	MXChip

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.

```
[env:mxchip_az3166]
platform = ststm32
board = mxchip_az3166

; change microcontroller
board_build.mcu = stm32f412zgt6

; change MCU frequency
board_build.f_cpu = 100000000L
```

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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>J-LINK</i>		
<i>ST-LINK</i>	Yes	Yes

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>STM32</i>	STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.

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.

Microcontroller	STM32F411RET6
Frequency	100MHz
Flash	512KB
RAM	128KB
Vendor	MultiTech

Configuration

Please use `mts_mdot_f405rg` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:mts_mdot_f405rg]
platform = ststm32
board = mts_mdot_f405rg
```

You can override default MultiTech mDot settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `mts_mdot_f405rg.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:mts_mdot_f405rg]
platform = ststm32
board = mts_mdot_f405rg

; change microcontroller
board_build.mcu = stm32f411ret6

; change MCU frequency
board_build.f_cpu = 100000000L
```

Uploading

MultiTech mDot supports the next uploading protocols:

- blackmagic
- jlink
- mbed
- stlink

Default protocol is mbed

You can change upload protocol using `upload_protocol` option:

```
[env:mts_mdot_f405rg]
platform = ststm32
board = mts_mdot_f405rg
```

(continues on next page)

(continued from previous 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*).

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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		Yes
<i>J-LINK</i>		
<i>ST-LINK</i>		

Frameworks

Name	Description
<i>mbed</i>	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.
<i>STM32</i>	STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.

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.

Microcontroller	STM32F411RET6
Frequency	100MHz
Flash	512KB
RAM	128KB
Vendor	MultiTech

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
```

(continues on next page)

(continued from previous 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*).

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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		Yes
<i>J-LINK</i>		
<i>ST-LINK</i>		

Frameworks

Name	Description
<i>mbed</i>	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.
<i>STM32</i>	STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.

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.

Microcontroller	STM32L151CCU6
Frequency	32MHz
Flash	256KB
RAM	32KB
Vendor	MultiTech

Configuration

Please use `xdot_1151cc` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[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_1151cc.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:xdot_1151cc]
platform = ststm32
board = xdot_1151cc

; 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:

```
[env:xdot_1151cc]
platform = ststm32
board = xdot_1151cc

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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		Yes
<i>J-LINK</i>		
<i>ST-LINK</i>		

Frameworks

Name	Description
<i>mbed</i>	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.
<i>STM32</i>	STM32Cube embedded 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.

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.

Microcontroller	STM32F405RG
Frequency	168MHz
Flash	1MB
RAM	192KB
Vendor	Netduino

Configuration

Please use `netduino2plus` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[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.

Compatible Tools	On-board	Default
J-LINK		
ST-LINK		Yes

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.
STM32	STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.

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.

Microcontroller	STM32L152RC
Frequency	32MHz
Flash	256KB
RAM	32KB
Vendor	Semtech

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_l152rc]
platform = ststm32
board = mote_l152rc

; change microcontroller
board_build.mcu = stm32l152rc

; change MCU frequency
board_build.f_cpu = 32000000L
```

Uploading

NAMote72 supports the next uploading protocols:

- blackmagic
- jlink
- mbed
- stlink

Default protocol is mbed

You can change upload protocol using `upload_protocol` option:

```
[env:mote_l152rc]
platform = ststm32
board = mote_l152rc

upload_protocol = mbed
```

Debugging

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

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using *debug_tool* option in “*platformio.ini*” (*Project Configuration File*).

NAMote72 does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		Yes
<i>J-LINK</i>		
<i>ST-LINK</i>		

Frameworks

Name	Description
<i>mbed</i>	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.
<i>STM32</i>	STM32Cube embedded 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.

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.

Microcontroller	STM32G071RBT6
Frequency	24MHz
Flash	2MB
RAM	128KB
Vendor	ST

Configuration

Please use nucleo_g071rb ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:nucleo_g071rb]
platform = ststm32
board = nucleo_g071rb
```

You can override default Nucleo G071RB settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `nucleo_g071rb.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:nucleo_g071rb]
platform = ststm32
board = nucleo_g071rb

; change microcontroller
board_build.mcu = stm32g071rbt6

; change MCU frequency
board_build.f_cpu = 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:

```
[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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>J-LINK</i>		
<i>ST-LINK</i>	Yes	Yes

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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.

Microcontroller	STM32WB55RG
Frequency	64MHz
Flash	512KB
RAM	192.00KB
Vendor	ST

Configuration

Please use nucleo_wb55rg_p ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[env:nucleo_wb55rg_p]
platform = ststm32
board = nucleo_wb55rg_p
```

You can override default P-Nucleo WB55RG settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `nucleo_wb55rg_p.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:nucleo_wb55rg_p]
platform = ststm32
board = nucleo_wb55rg_p

; change microcontroller
board_build.mcu = stm32wb55rg

; change MCU frequency
board_build.f_cpu = 64000000L
```

Uploading

P-Nucleo WB55RG supports the next uploading protocols:

- blackmagic
- 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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>J-LINK</i>		
<i>ST-LINK</i>	Yes	Yes

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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.

Microcontroller	STM32L151RBT6
Frequency	32MHz
Flash	128KB
RAM	16KB
Vendor	RAK

Configuration

Please use `rak811_tracker` ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[env:rak811_tracker]
platform = ststm32
board = rak811_tracker
```

You can override default RAK811 LoRa Tracker settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `rak811_tracker.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:rak811_tracker]
platform = ststm32
board = rak811_tracker

; change microcontroller
board_build.mcu = stm32l151rbt6

; change MCU frequency
board_build.f_cpu = 32000000L
```

Uploading

RAK811 LoRa Tracker supports the next uploading protocols:

- blackmagic
- 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.

Compatible Tools	On-board	Default
Black Magic Probe		Yes
J-LINK		
ST-LINK		

Frameworks

Name	Description
<i>Arduin</i> <i>o</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>STM32</i>	STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.

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.

Microcontroller	STM32L151RBT6
Frequency	32MHz
Flash	128KB
RAM	32KB
Vendor	RAK

Configuration

Please use `rak811_tracker_32` ID for `board` option in “*platformio.ini*” (*Project Configuration File*):

```
[env:rak811_tracker_32]
platform = ststm32
board = rak811_tracker_32
```

You can override default RAK811 LoRa Tracker settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `rak811_tracker_32.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:rak811_tracker_32]
platform = ststm32
board = rak811_tracker_32

; change microcontroller
board_build.mcu = stm32l151rbt6

; change MCU frequency
board_build.f_cpu = 32000000L
```

Uploading

RAK811 LoRa Tracker supports the next uploading protocols:

- blackmagic
- jlink
- serial
- stlink

Default protocol is stlink

You can change upload protocol using *upload_protocol* option:

```
[env:rak811_tracker_32]
platform = ststm32
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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		Yes
<i>J-LINK</i>		
<i>ST-LINK</i>		

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.
STM32	STM32Cube embedded 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.

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.

Microcontroller	STM32F401RET6
Frequency	84MHz
Flash	512KB
RAM	96KB
Vendor	RushUp

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.

```
[env:cloud_jam]
platform = ststm32
board = cloud_jam

; change microcontroller
board_build.mcu = stm32f401ret6

; change MCU frequency
board_build.f_cpu = 84000000L
```

Uploading

RushUp Cloud-JAM supports the next uploading protocols:

- blackmagic
- jlink
- mbed
- stlink

Default protocol is stlink

You can change upload protocol using *upload_protocol* option:

```
[env:cloud_jam]
platform = ststm32
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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>J-LINK</i>		
<i>ST-LINK</i>	Yes	Yes

Frameworks

Name	Description
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.
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.
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.
STM32	STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.

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.

Microcontroller	STM32L476RGT6
Frequency	80MHz
Flash	1MB
RAM	128KB
Vendor	RushUp

Configuration

Please use `cloud_jam_14` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:cloud_jam_14]
platform = ststm32
board = cloud_jam_14
```

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_14.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:cloud_jam_14]
platform = ststm32
board = cloud_jam_14

; change microcontroller
board_build.mcu = stm32147rgt6

; 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_14]
platform = ststm32
board = cloud_jam_14

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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>J-LINK</i>		
<i>ST-LINK</i>	Yes	Yes

Frameworks

Name	Description
<i>mbed</i>	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.
<i>STM32Cube</i>	STM32Cube embedded 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.

ST 32F3348DISCOVERY

Contents

- *ST 32F3348DISCOVERY*
 - *Hardware*
 - *Configuration*
 - *Uploading*
 - *Debugging*
 - *Frameworks*

Hardware

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

Microcontroller	STM32F334C8T6
Frequency	72MHz
Flash	64KB
RAM	12KB
Vendor	ST

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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>J-LINK</i>		
<i>ST-LINK</i>	Yes	Yes

Frameworks

Name	Description
<i>mbed</i>	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.
<i>STM32Cube</i>	STM32Cube embedded 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.

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.

Microcontroller	STM32F401VCT6
Frequency	84MHz
Flash	256KB
RAM	64KB
Vendor	ST

Configuration

Please use disco_f401vc ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[env:disco_f401vc]
platform = ststm32
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.

```
[env:disco_f401vc]
platform = ststm32
board = disco_f401vc

; change microcontroller
board_build.mcu = stm32f401vct6

; change MCU frequency
board_build.f_cpu = 84000000L
```

Uploading

ST 32F401CDISCOVERY supports the next uploading protocols:

- blackmagic
- jlink
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```
[env:disco_f401vc]
platform = ststm32
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
<i>Black Magic Probe</i>		
<i>J-LINK</i>		
<i>ST-LINK</i>	Yes	Yes

Frameworks

Name	Description
<i>mbed</i>	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.
<i>STM32Cube</i>	STM32Cube embedded 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.

ST 32F411EDISCOVERY

Contents

- *ST 32F411EDISCOVERY*
 - *Hardware*
 - *Configuration*
 - *Uploading*
 - *Debugging*
 - *Frameworks*

Hardware

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

Microcontroller	STM32F411VET6
Frequency	100MHz
Flash	512KB
RAM	128KB
Vendor	ST

Configuration

Please use disco_f411ve ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[env:disco_f411ve]
platform = ststm32
board = disco_f411ve
```

You can override default ST 32F411EDISCOVERY settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `disco_f411ve.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:disco_f411ve]
platform = ststm32
board = disco_f411ve

; change microcontroller
board_build.mcu = stm32f411vet6

; change MCU frequency
board_build.f_cpu = 100000000L
```

Uploading

ST 32F411EDISCOVERY supports the next uploading protocols:

- blackmagic
- jlink
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```
[env:disco_f411ve]
platform = ststm32
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	On-board	Default
<i>Black Magic Probe</i>		
<i>J-LINK</i>		
<i>ST-LINK</i>	Yes	Yes

Frameworks

Name	Description
<i>STM32</i>	STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.

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.

Microcontroller	STM32F413ZHT6
Frequency	100MHz
Flash	512KB
RAM	128KB
Vendor	ST

Configuration

Please use `disco_f413zh` ID for `board` option in “*platformio.ini*” (*Project Configuration File*):

```
[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.

```
[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:

```
[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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>J-LINK</i>		
<i>ST-LINK</i>	Yes	Yes

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.
STM32	STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.

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.

Microcontroller	STM32F429ZIT6
Frequency	180MHz
Flash	2MB
RAM	256KB
Vendor	ST

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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>J-LINK</i>		
<i>ST-LINK</i>	Yes	Yes

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.
STM32	STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.

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.

Microcontroller	STM32F469NIH6
Frequency	180MHz
Flash	1MB
RAM	384KB
Vendor	ST

Configuration

Please use disco_f469ni ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:disco_f469ni]
platform = ststm32
board = disco_f469ni
```

You can override default ST 32F469IDISCOVERY settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `disco_f469ni.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:disco_f469ni]
platform = ststm32
board = disco_f469ni

; change microcontroller
board_build.mcu = stm32f469nih6

; change MCU frequency
board_build.f_cpu = 180000000L
```

Uploading

ST 32F469IDISCOVERY supports the next uploading protocols:

- blackmagic
- jlink
- mbed
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```
[env:disco_f469ni]
platform = ststm32
board = disco_f469ni

upload_protocol = stlink
```

Debugging

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

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for 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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>J-LINK</i>		
<i>ST-LINK</i>	Yes	Yes

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.
STM32	STM32Cube embedded 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.

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.

Microcontroller	STM32F746NGH6
Frequency	216MHz
Flash	1MB
RAM	320KB
Vendor	ST

Configuration

Please use `disco_f746ng` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:disco_f746ng]
platform = ststm32
board = disco_f746ng
```

You can override default ST 32F746GDISCOVERY settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `disco_f746ng.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:disco_f746ng]
platform = ststm32
board = disco_f746ng

; change microcontroller
board_build.mcu = stm32f746ngh6

; change MCU frequency
board_build.f_cpu = 216000000L
```

Uploading

ST 32F746GDISCOVERY supports the next uploading protocols:

- blackmagic
- jlink
- mbed
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```
[env:disco_f746ng]
platform = ststm32
board = disco_f746ng

upload_protocol = stlink
```

Debugging

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

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for 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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>J-LINK</i>		
<i>ST-LINK</i>	Yes	Yes

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.
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.
STM32	STM32Cube embedded 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.

ST 32F769IDISCOVERY

Contents

- [ST 32F769IDISCOVERY](#)
 - [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	STM32F769NIH6
Frequency	216MHz
Flash	1MB
RAM	512KB
Vendor	ST

Configuration

Please use `disco_f769ni` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:disco_f769ni]
platform = ststm32
board = disco_f769ni
```

You can override default ST 32F769IDISCOVERY settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `disco_f769ni.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:disco_f769ni]
platform = ststm32
board = disco_f769ni

; change microcontroller
board_build.mcu = stm32f769nih6

; change MCU frequency
board_build.f_cpu = 216000000L
```

Uploading

ST 32F769IDISCOVERY supports the next uploading protocols:

- blackmagic
- jlink
- mbed
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```
[env:disco_f769ni]
platform = ststm32
board = disco_f769ni

upload_protocol = stlink
```

Debugging

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

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for 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	On-board	Default
<i>Black Magic Probe</i>		
<i>J-LINK</i>		
<i>ST-LINK</i>	Yes	Yes

Frameworks

Name	Description
<i>mbed</i>	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.
<i>STM32Cube</i>	STM32Cube embedded 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.

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.

Microcontroller	STM32L053C8T6
Frequency	32MHz
Flash	64KB
RAM	8KB
Vendor	ST

Configuration

Please use disco_1053c8 ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[env:disco_1053c8]
platform = ststm32
board = disco_1053c8
```

You can override default ST 32L0538DISCOVERY settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `disco_1053c8.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:disco_1053c8]
platform = ststm32
board = disco_1053c8

; 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_1053c8]
platform = ststm32
board = disco_1053c8

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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>J-LINK</i>		
<i>ST-LINK</i>	Yes	Yes

Frameworks

Name	Description
<i>mbed</i>	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.
<i>STM32Cube</i>	STM32Cube embedded 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.

ST 32L100DISCOVERY

Contents

- *ST 32L100DISCOVERY*
 - *Hardware*
 - *Configuration*
 - *Uploading*
 - *Debugging*
 - *Frameworks*

Hardware

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

Microcontroller	STM32L100RCT6
Frequency	32MHz
Flash	256KB
RAM	16KB
Vendor	ST

Configuration

Please use disco_l1100rc ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[env:disco_l1100rc]
platform = ststm32
board = disco_l1100rc
```

You can override default ST 32L100DISCOVERY settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `disco_l1100rc.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:disco_l1100rc]
platform = ststm32
board = disco_l1100rc

; change microcontroller
board_build.mcu = stm32l110rct6

; 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_l1100rc]
platform = ststm32
board = disco_l1100rc

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	On-board	Default
<i>Black Magic Probe</i>		
<i>J-LINK</i>		
<i>ST-LINK</i>	Yes	Yes

Frameworks

Name	Description
<i>STM32</i>	STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.

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.

Microcontroller	STM32L476VGT6
Frequency	80MHz
Flash	1MB
RAM	128KB
Vendor	ST

Configuration

Please use `disco_1476vg` ID for `board` option in “*platformio.ini*” (*Project Configuration File*):

```
[env:disco_1476vg]
platform = ststm32
board = disco_1476vg
```

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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>J-LINK</i>		
<i>ST-LINK</i>	Yes	Yes

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.
STM32	STM32Cube embedded 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.

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.

Microcontroller	STM32L496AGI6
Frequency	80MHz
Flash	1MB
RAM	320KB
Vendor	ST

Configuration

Please use `disco_1496ag` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:disco_1496ag]
platform = ststm32
board = disco_1496ag
```

You can override default ST 32L496GDISCOVERY settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `disco_l496ag.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:disco_1496ag]
platform = ststm32
board = disco_1496ag

; 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:

```
[env:disco_1496ag]
platform = ststm32
board = disco_1496ag

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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>J-LINK</i>		
<i>ST-LINK</i>	Yes	Yes

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.
STM32	STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.

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.

Microcontroller	STM32L072CZ
Frequency	32MHz
Flash	192KB
RAM	20KB
Vendor	ST

Configuration

Please use disco_l072cz_lrwan1 ID for `board` option in “`platformio.ini`” (Project Configuration File):

```
[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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>J-LINK</i>		
<i>ST-LINK</i>	Yes	Yes

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.
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.
STM32	STM32Cube embedded 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.

ST DISCO-L475VG-IOT01A

Contents

- [ST DISCO-L475VG-IOT01A](#)
 - [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	STM32L475VGT6
Frequency	80MHz
Flash	1MB
RAM	128KB
Vendor	ST

Configuration

Please use `disco_l475vg_iot01a` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:disco_1475vg_iot01a]
platform = ststm32
board = disco_1475vg_iot01a
```

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_1475vg_iot01a.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:disco_1475vg_iot01a]
platform = ststm32
board = disco_1475vg_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_1475vg_iot01a]
platform = ststm32
board = disco_1475vg_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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>J-LINK</i>		
<i>ST-LINK</i>	Yes	Yes

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>mbed</i>	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.
<i>STM32</i>	STM32Cube embedded 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.

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.

Microcontroller	STM32F072RBT6
Frequency	48MHz
Flash	128KB
RAM	16KB
Vendor	ST

Configuration

Please use disco_f072rb ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[env:disco_f072rb]
platform = ststm32
board = disco_f072rb
```

You can override default ST Discovery F072RB settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `disco_f072rb.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:disco_f072rb]
platform = ststm32
board = disco_f072rb

; change microcontroller
board_build.mcu = stm32f072rbt6

; change MCU frequency
board_build.f_cpu = 48000000L
```

Uploading

ST Discovery F072RB supports the next uploading protocols:

- blackmagic
- jlink
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```
[env:disco_f072rb]
platform = ststm32
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	On-board	Default
<i>Black Magic Probe</i>		
<i>J-LINK</i>		
<i>ST-LINK</i>	Yes	Yes

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>mbed</i>	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.
<i>STM32</i>	STM32Cube embedded 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.

ST Nucleo F030R8

Contents

- *ST Nucleo F030R8*
 - *Hardware*
 - *Configuration*
 - *Uploading*
 - *Debugging*
 - *Frameworks*

Hardware

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

Microcontroller	STM32F030R8T6
Frequency	48MHz
Flash	64KB
RAM	8KB
Vendor	ST

Configuration

Please use nucleo_f030r8 ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[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.

```
[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:

```
[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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>J-LINK</i>		
<i>ST-LINK</i>	Yes	Yes

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>mbed</i>	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.
<i>STM32</i>	STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.

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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>J-LINK</i>		
<i>ST-LINK</i>	Yes	Yes

Frameworks

Name	Description
<i>mbed</i>	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.
<i>STM32</i>	STM32Cube embedded 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.

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.

Microcontroller	STM32F042K6T6
Frequency	48MHz
Flash	32KB
RAM	6KB
Vendor	ST

Configuration

Please use nucleo_f042k6 ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:nucleo_f042k6]
platform = ststm32
board = nucleo_f042k6
```

You can override default ST Nucleo F042K6 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `nucleo_f042k6.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:nucleo_f042k6]
platform = ststm32
board = nucleo_f042k6

; change microcontroller
board_build.mcu = stm32f042k6t6

; change MCU frequency
board_build.f_cpu = 48000000L
```

Uploading

ST Nucleo F042K6 supports the next uploading protocols:

- blackmagic
- jlink
- mbed
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```
[env:nucleo_f042k6]
platform = ststm32
board = nucleo_f042k6

upload_protocol = stlink
```

Debugging

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

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for 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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>J-LINK</i>		
<i>ST-LINK</i>	Yes	Yes

Frameworks

Name	Description
<i>mbed</i>	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.
<i>STM32</i>	STM32Cube embedded 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.

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.

Microcontroller	STM32F070RBT6
Frequency	48MHz
Flash	128KB
RAM	16KB
Vendor	ST

Configuration

Please use nucleo_f070rb ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[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.

```
[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:

```
[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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>J-LINK</i>		
<i>ST-LINK</i>	Yes	Yes

Frameworks

Name	Description
<i>mbed</i>	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.
<i>STM32</i>	STM32Cube embedded 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.

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.

Microcontroller	STM32F072RBT6
Frequency	48MHz
Flash	128KB
RAM	16KB
Vendor	ST

Configuration

Please use nucleo_f072rb ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:nucleo_f072rb]
platform = ststm32
board = nucleo_f072rb
```

You can override default ST Nucleo F072RB settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `nucleo_f072rb.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:nucleo_f072rb]
platform = ststm32
board = nucleo_f072rb

; change microcontroller
board_build.mcu = stm32f072rbt6

; change MCU frequency
board_build.f_cpu = 48000000L
```

Uploading

ST Nucleo F072RB supports the next uploading protocols:

- blackmagic
- jlink
- mbed
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```
[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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>J-LINK</i>		
<i>ST-LINK</i>	Yes	Yes

Frameworks

Name	Description
<i>mbed</i>	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.
<i>STM32</i>	STM32Cube embedded 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.

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*):

```
[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.

```
[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:

```
[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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>J-LINK</i>		
<i>ST-LINK</i>	Yes	Yes

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>mbed</i>	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.
<i>STM32</i>	STM32Cube embedded 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.

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.

Microcontroller	STM32F103RBT6
Frequency	72MHz
Flash	128KB
RAM	20KB
Vendor	ST

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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>J-LINK</i>		
<i>ST-LINK</i>	Yes	Yes

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>libOpenCM3</i>	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.
<i>mbed</i>	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.
<i>STM32</i>	STM32Cube embedded 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.

ST Nucleo F207ZG

Contents

- *ST Nucleo F207ZG*
 - *Hardware*
 - *Configuration*
 - *Uploading*
 - *Debugging*

– *Frameworks*

Hardware

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

Microcontroller	STM32F207ZGT6
Frequency	120MHz
Flash	1MB
RAM	128KB
Vendor	ST

Configuration

Please use nucleo_f207zg ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[env:nucleo_f207zg]
platform = ststm32
board = nucleo_f207zg
```

You can override default ST Nucleo F207ZG settings per build environment using *board_**** option, where *** is a JSON object path from board manifest *nucleo_f207zg.json*. For example, *board_build.mcu*, *board_build.f_cpu*, etc.

```
[env:nucleo_f207zg]
platform = ststm32
board = nucleo_f207zg

; change microcontroller
board_build.mcu = stm32f207zgt6

; change MCU frequency
board_build.f_cpu = 120000000L
```

Uploading

ST Nucleo F207ZG supports the next uploading protocols:

- blackmagic
- jlink
- mbed
- stlink

Default protocol is *stlink*

You can change upload protocol using *upload_protocol* option:

```
[env:nucleo_f207zg]
platform = ststm32
board = nucleo_f207zg

upload_protocol = stlink
```

Debugging

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

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for 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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>J-LINK</i>		
<i>ST-LINK</i>	Yes	Yes

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>mbed</i>	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.
<i>STM32</i>	STM32Cube embedded 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.

ST Nucleo F302R8

Contents

- *ST Nucleo F302R8*
 - *Hardware*

- Configuration
- Uploading
- Debugging
- Frameworks

Hardware

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

Microcontroller	STM32F302R8T6
Frequency	72MHz
Flash	64KB
RAM	16KB
Vendor	ST

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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>J-LINK</i>		
<i>ST-LINK</i>	Yes	Yes

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>mbed</i>	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.
<i>STM32</i>	STM32Cube embedded 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.

ST Nucleo F303K8

Contents

- *ST Nucleo F303K8*

- [Hardware](#)
- [Configuration](#)
- [Uploading](#)
- [Debugging](#)
- [Frameworks](#)

Hardware

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

Microcontroller	STM32F303K8T6
Frequency	72MHz
Flash	64KB
RAM	12KB
Vendor	ST

Configuration

Please use nucleo_f303k8 ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[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.

```
[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:

```
[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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>J-LINK</i>		
<i>ST-LINK</i>	Yes	Yes

Frameworks

Name	Description
<i>Ar-duino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>mbed</i>	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.
<i>STM32</i>	STM32Cube embedded 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.

ST Nucleo F303RE

Contents

- ST Nucleo F303RE
 - Hardware
 - Configuration
 - Uploading
 - Debugging
 - Frameworks

Hardware

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

Microcontroller	STM32F303RET6
Frequency	72MHz
Flash	512KB
RAM	64KB
Vendor	ST

Configuration

Please use nucleo_f303re ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[env:nucleo_f303re]
platform = ststm32
board = nucleo_f303re
```

You can override default ST Nucleo F303RE settings per build environment using *board_**** option, where *** is a JSON object path from board manifest *nucleo_f303re.json*. For example, *board_build.mcu*, *board_build.f_cpu*, etc.

```
[env:nucleo_f303re]
platform = ststm32
board = nucleo_f303re

; change microcontroller
board_build.mcu = stm32f303ret6

; change MCU frequency
board_build.f_cpu = 72000000L
```

Uploading

ST Nucleo F303RE supports the next uploading protocols:

- blackmagic
- jlink
- mbed
- stlink

Default protocol is stlink

You can change upload protocol using *upload_protocol* option:

```
[env:nucleo_f303re]
platform = ststm32
board = nucleo_f303re

upload_protocol = stlink
```

Debugging

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

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for 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	On-board	Default
<i>Black Magic Probe</i>		
<i>J-LINK</i>		
<i>ST-LINK</i>	Yes	Yes

Frameworks

Name	Description
<i>Arduino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>mbed</i>	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.
<i>STM32</i>	STM32Cube embedded 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.

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.

Microcontroller	STM32F303ZET6
Frequency	72MHz
Flash	512KB
RAM	64KB
Vendor	ST

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 = ststm32
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.

Compatible Tools	On-board	Default
Black Magic Probe		
J-LINK		
ST-LINK	Yes	Yes

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.
STM32	STM32Cube embedded 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.

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.

Microcontroller	STM32F334R8T6
Frequency	72MHz
Flash	64KB
RAM	16KB
Vendor	ST

Configuration

Please use nucleo_f334r8 ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[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.

```
[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:

```
[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.

Compatible Tools	On-board	Default
Black Magic Probe		
J-LINK		
ST-LINK	Yes	Yes

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.
STM32	STM32Cube embedded 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.

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.

Microcontroller	STM32F401RET6
Frequency	84MHz
Flash	512KB
RAM	96KB
Vendor	ST

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.

Compatible Tools	On-board	Default
Black Magic Probe		
J-LINK		
ST-LINK	Yes	Yes

Frameworks

Name	Description
Arduino	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
CMSIS	The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.
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.
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.
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.

ST Nucleo F410RB

Contents

- [ST Nucleo F410RB](#)
 - [Hardware](#)
 - [Configuration](#)
 - [Uploading](#)
 - [Debugging](#)
 - [Frameworks](#)

Hardware

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

Microcontroller	STM32F410RBT6
Frequency	100MHz
Flash	128KB
RAM	32KB
Vendor	ST

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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>J-LINK</i>		
<i>ST-LINK</i>	Yes	Yes

Frameworks

Name	Description
<i>mbed</i>	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.
<i>STM32Cube</i>	STM32Cube embedded 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.

ST Nucleo F411RE

Contents

- *ST Nucleo F411RE*
 - *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	STM32F411RET6
Frequency	100MHz
Flash	512KB
RAM	128KB
Vendor	ST

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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>J-LINK</i>		
<i>ST-LINK</i>	Yes	Yes

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>mbed</i>	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.
<i>STM32</i>	STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.

ST Nucleo F412ZG

Contents

- *ST Nucleo F412ZG*
 - *Hardware*
 - *Configuration*
 - *Uploading*
 - *Debugging*
 - *Frameworks*

Hardware

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

Microcontroller	STM32F412ZGT6
Frequency	100MHz
Flash	1MB
RAM	256KB
Vendor	ST

Configuration

Please use nucleo_f412zg ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[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.

```
[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:

```
[env:nucleo_f412zg]
platform = ststm32
board = nucleo_f412zg

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 F412ZG 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
<i>Black Magic Probe</i>		
<i>J-LINK</i>		
<i>ST-LINK</i>	Yes	Yes

Frameworks

Name	Description
<i>mbed</i>	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.
<i>STM32</i>	STM32Cube embedded 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.

ST Nucleo F413ZH

Contents

- *ST Nucleo F413ZH*
 - *Hardware*
 - *Configuration*
 - *Uploading*
 - *Debugging*
 - *Frameworks*

Hardware

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

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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>J-LINK</i>		
<i>ST-LINK</i>	Yes	Yes

Frameworks

Name	Description
<i>mbed</i>	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.
<i>STM32</i>	STM32Cube embedded 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.

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.

Microcontroller	STM32F429ZIT6
Frequency	180MHz
Flash	2MB
RAM	256KB
Vendor	ST

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.

Compatible Tools	On-board	Default
Black Magic Probe		
J-LINK		
ST-LINK	Yes	Yes

Frameworks

Name	Description
Arduin	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
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.
STM32	STM32Cube embedded 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.

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.

Microcontroller	STM32F439ZIT6
Frequency	180MHz
Flash	2MB
RAM	256KB
Vendor	ST

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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>J-LINK</i>		
<i>ST-LINK</i>	Yes	Yes

Frameworks

Name	Description
<i>mbed</i>	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.
<i>STM32</i>	STM32Cube embedded 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.

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.

Microcontroller	STM32F446RET6
Frequency	180MHz
Flash	512KB
RAM	128KB
Vendor	ST

Configuration

Please use nucleo_f446re ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[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.

```
[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:

```
[env:nucleo_f446re]
platform = ststm32
board = nucleo_f446re
```

(continues on next page)

(continued from previous 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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>J-LINK</i>		
<i>ST-LINK</i>	Yes	Yes

Frameworks

Name	Description
<i>Arduino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>mbed</i>	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.
<i>STM32</i>	STM32Cube embedded 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.

ST Nucleo F446ZE

Contents

- *ST Nucleo F446ZE*
 - *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	STM32F446ZET6
Frequency	180MHz
Flash	512KB
RAM	128KB
Vendor	ST

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 = ststm32
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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>J-LINK</i>		
<i>ST-LINK</i>	Yes	Yes

Frameworks

Name	Description
<i>mbed</i>	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.
<i>STM32</i>	STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.

ST Nucleo F722ZE

Contents

- *ST Nucleo F722ZE*
 - *Hardware*
 - *Configuration*
 - *Uploading*

- Debugging
- Frameworks

Hardware

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

Microcontroller	STM32F722ZET6
Frequency	216MHz
Flash	512KB
RAM	256KB
Vendor	ST

Configuration

Please use nucleo_f722ze ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[env:nucleo_f722ze]
platform = ststm32
board = nucleo_f722ze
```

You can override default ST Nucleo F722ZE settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `nucleo_f722ze.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:nucleo_f722ze]
platform = ststm32
board = nucleo_f722ze

; change microcontroller
board_build.mcu = stm32f722zet6

; change MCU frequency
board_build.f_cpu = 216000000L
```

Uploading

ST Nucleo F722ZE supports the next uploading protocols:

- blackmagic
- jlink
- mbed
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```
[env:nucleo_f722ze]
platform = ststm32
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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>J-LINK</i>		
<i>ST-LINK</i>	Yes	Yes

Frameworks

Name	Description
<i>STM32</i>	STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.

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.

Microcontroller	STM32F746ZGT6
Frequency	216MHz
Flash	1MB
RAM	320KB
Vendor	ST

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)

(continued from previous 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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>J-LINK</i>		
<i>ST-LINK</i>	Yes	Yes

Frameworks

Name	Description
<i>mbed</i>	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.
<i>STM32</i>	STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.

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.

Microcontroller	STM32F756ZG
Frequency	216MHz
Flash	1MB
RAM	320KB
Vendor	ST

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)

(continued from previous 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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>J-LINK</i>		
<i>ST-LINK</i>	Yes	Yes

Frameworks

Name	Description
<i>mbed</i>	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.
<i>STM32</i>	STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.

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.

Microcontroller	STM32F767ZIT6
Frequency	216MHz
Flash	2MB
RAM	512KB
Vendor	ST

Configuration

Please use nucleo_f767zi ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[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.

```
[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:

```
[env:nucleo_f767zi]
platform = ststm32
board = nucleo_f767zi
```

(continues on next page)

(continued from previous 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 F767ZI 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
<i>Black Magic Probe</i>		
<i>J-LINK</i>		
<i>ST-LINK</i>	Yes	Yes

Frameworks

Name	Description
<i>Arduino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>mbed</i>	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.
<i>STM32</i>	STM32Cube embedded 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.

ST Nucleo H743ZI

Contents

- *ST Nucleo H743ZI*
 - *Hardware*
 - *Configuration*
 - *Uploading*

- Debugging
- Frameworks

Hardware

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

Microcontroller	STM32H743ZIT6
Frequency	400MHz
Flash	2MB
RAM	1MB
Vendor	ST

Configuration

Please use nucleo_h743zi ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[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.

```
[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
<i>Black Magic Probe</i>		
<i>J-LINK</i>		
<i>ST-LINK</i>	Yes	Yes

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>STM32</i>	STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.

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.

Microcontroller	STM32L011K4T6
Frequency	32MHz
Flash	16KB
RAM	2KB
Vendor	ST

Configuration

Please use nucleo_1011k4 ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[env:nucleo_1011k4]
platform = ststm32
board = nucleo_1011k4
```

You can override default ST Nucleo L011K4 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `nucleo_1011k4.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:nucleo_1011k4]
platform = ststm32
board = nucleo_1011k4

; 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:

```
[env:nucleo_1011k4]
platform = ststm32
board = nucleo_1011k4

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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>J-LINK</i>		
<i>ST-LINK</i>	Yes	Yes

Frameworks

Name	Description
<i>STM32Cube</i>	STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.

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.

Microcontroller	STM32L031K6T6
Frequency	32MHz
Flash	32KB
RAM	8KB
Vendor	ST

Configuration

Please use nucleo_1031k6 ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:nucleo_1031k6]
platform = ststm32
board = nucleo_1031k6
```

You can override default ST Nucleo L031K6 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `nucleo_1031k6.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:nucleo_1031k6]
platform = ststm32
board = nucleo_1031k6

; 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:

```
[env:nucleo_1031k6]
platform = ststm32
board = nucleo_1031k6

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.

Compatible Tools	On-board	Default
Black Magic Probe		
J-LINK		
ST-LINK	Yes	Yes

Frameworks

Name	Description
Arduin	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
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.
STM32	STM32Cube embedded 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.

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.

Microcontroller	STM32L053R8T6
Frequency	32MHz
Flash	64KB
RAM	8KB
Vendor	ST

Configuration

Please use nucleo_1053r8 ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:nucleo_1053r8]
platform = ststm32
board = nucleo_1053r8
```

You can override default ST Nucleo L053R8 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `nucleo_1053r8.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:nucleo_1053r8]
platform = ststm32
board = nucleo_1053r8

; 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:

```
[env:nucleo_1053r8]
platform = ststm32
board = nucleo_1053r8

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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>J-LINK</i>		
<i>ST-LINK</i>	Yes	Yes

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>mbed</i>	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.
<i>STM32</i>	STM32Cube embedded 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.

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.

Microcontroller	STM32L073RZ
Frequency	32MHz
Flash	192KB
RAM	20KB
Vendor	ST

Configuration

Please use nucleo_1073rz ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[env:nucleo_1073rz]
platform = ststm32
board = nucleo_1073rz
```

You can override default ST Nucleo L073RZ settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `nucleo_1073rz.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:nucleo_1073rz]
platform = ststm32
board = nucleo_1073rz

; change microcontroller
board_build.mcu = stm32l073rz

; change MCU frequency
board_build.f_cpu = 32000000L
```

Uploading

ST Nucleo L073RZ supports the next uploading protocols:

- blackmagic
- jlink
- mbed
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```
[env:nucleo_1073rz]
platform = ststm32
board = nucleo_1073rz
```

(continues on next page)

(continued from previous 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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>J-LINK</i>		
<i>ST-LINK</i>	Yes	Yes

Frameworks

Name	Description
<i>Arduino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>mbed</i>	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.
<i>STM32</i>	STM32Cube embedded 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.

ST Nucleo L152RE

Contents

- *ST Nucleo L152RE*
 - *Hardware*
 - *Configuration*
 - *Uploading*

- *Debugging*
- *Frameworks*

Hardware

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

Microcontroller	STM32L152RET6
Frequency	32MHz
Flash	512KB
RAM	80KB
Vendor	ST

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:

```
[env:nucleo_1152re]
platform = ststm32
board = nucleo_1152re

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 L152RE 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
<i>Black Magic Probe</i>		
<i>J-LINK</i>		
<i>ST-LINK</i>	Yes	Yes

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>mbed</i>	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.
<i>STM32</i>	STM32Cube embedded 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.

ST Nucleo L412KB

Contents

- *ST Nucleo L412KB*
 - *Hardware*

- Configuration
- Uploading
- Debugging
- Frameworks

Hardware

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

Microcontroller	STM32L412KBU6
Frequency	80MHz
Flash	128KB
RAM	40KB
Vendor	ST

Configuration

Please use nucleo_l412kb ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[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.

```
[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:

```
[env:nucleo_1412kb]
platform = ststm32
board = nucleo_1412kb

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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>J-LINK</i>		
<i>ST-LINK</i>	Yes	Yes

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>STM32</i>	STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.

ST Nucleo L432KC

Contents

- *ST Nucleo L432KC*
 - *Hardware*
 - *Configuration*
 - *Uploading*

- *Debugging*
- *Frameworks*

Hardware

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

Microcontroller	STM32L432KC <u>U6</u>
Frequency	80MHz
Flash	256KB
RAM	64KB
Vendor	ST

Configuration

Please use nucleo_1432kc ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[env:nucleo_1432kc]
platform = ststm32
board = nucleo_1432kc
```

You can override default ST Nucleo L432KC settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `nucleo_1432kc.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:nucleo_1432kc]
platform = ststm32
board = nucleo_1432kc

; 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_1432kc]
platform = ststm32
board = nucleo_1432kc

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 L432KC has on-board debug probe and **IS READY** for debugging. You don’t need to use/buy external debug probe.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>J-LINK</i>		
<i>ST-LINK</i>	Yes	Yes

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>mbed</i>	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.
<i>STM32</i>	STM32Cube embedded 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.

ST Nucleo L433RC-P

Contents

- *ST Nucleo L433RC-P*
 - *Hardware*

- Configuration
- Uploading
- Debugging
- Frameworks

Hardware

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

Microcontroller	STM32L433RC
Frequency	80MHz
Flash	256KB
RAM	64KB
Vendor	ST

Configuration

Please use nucleo_l433rc_p ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[env:nucleo_l433rc_p]
platform = ststm32
board = nucleo_l433rc_p
```

You can override default ST Nucleo L433RC-P settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `nucleo_l433rc_p.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:nucleo_l433rc_p]
platform = ststm32
board = nucleo_l433rc_p

; change microcontroller
board_build.mcu = stm32l433rc

; change MCU frequency
board_build.f_cpu = 80000000L
```

Uploading

ST Nucleo L433RC-P supports the next uploading protocols:

- blackmagic
- jlink
- mbed
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```
[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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>J-LINK</i>		
<i>ST-LINK</i>	Yes	Yes

Frameworks

Name	Description
<i>mbed</i>	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.
<i>STM32Cube</i>	STM32Cube embedded 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.

ST Nucleo L452RE

Contents

- *ST Nucleo L452RE*
 - *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	STM32L452RET6
Frequency	80MHz
Flash	256KB
RAM	64KB
Vendor	ST

Configuration

Please use nucleo_l452re ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[env:nucleo_1452re]
platform = ststm32
board = nucleo_1452re
```

You can override default ST Nucleo L452RE settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `nucleo_1452re.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:nucleo_1452re]
platform = ststm32
board = nucleo_1452re

; 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:

```
[env:nucleo_1452re]
platform = ststm32
board = nucleo_1452re

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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>J-LINK</i>		
<i>ST-LINK</i>	Yes	Yes

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>STM32</i>	STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.

ST Nucleo L476RG

Contents

- *ST Nucleo L476RG*
 - *Hardware*
 - *Configuration*
 - *Uploading*

- *Debugging*
- *Frameworks*

Hardware

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

Microcontroller	STM32L476RGT6
Frequency	80MHz
Flash	1MB
RAM	128KB
Vendor	ST

Configuration

Please use nucleo_1476rg ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[env:nucleo_1476rg]
platform = ststm32
board = nucleo_1476rg
```

You can override default ST Nucleo L476RG settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `nucleo_1476rg.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:nucleo_1476rg]
platform = ststm32
board = nucleo_1476rg

; 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:

```
[env:nucleo_1476rg]
platform = ststm32
board = nucleo_1476rg

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 L476RG 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
<i>Black Magic Probe</i>		
<i>J-LINK</i>		
<i>ST-LINK</i>	Yes	Yes

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>mbed</i>	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.
<i>STM32</i>	STM32Cube embedded 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.

ST Nucleo L486RG

Contents

- *ST Nucleo L486RG*
 - *Hardware*

- Configuration
- Uploading
- Debugging
- Frameworks

Hardware

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

Microcontroller	STM32L486RGT6
Frequency	80MHz
Flash	1MB
RAM	128KB
Vendor	ST

Configuration

Please use nucleo_l486rg ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[env:nucleo_l486rg]
platform = ststm32
board = nucleo_l486rg
```

You can override default ST Nucleo L486RG settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `nucleo_l486rg.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:nucleo_l486rg]
platform = ststm32
board = nucleo_l486rg

; change microcontroller
board_build.mcu = stm32l486rgt6

; change MCU frequency
board_build.f_cpu = 80000000L
```

Uploading

ST Nucleo L486RG supports the next uploading protocols:

- blackmagic
- jlink
- mbed
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```
[env:nucleo_1486rg]
platform = ststm32
board = nucleo_1486rg

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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>J-LINK</i>		
<i>ST-LINK</i>	Yes	Yes

Frameworks

Name	Description
<code>mbed</code>	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.
<code>STM32</code>	STM32Cube embedded 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.

ST Nucleo L496ZG

Contents

- *ST Nucleo L496ZG*
 - *Hardware*
 - *Configuration*

- *Uploading*
- *Debugging*
- *Frameworks*

Hardware

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

Microcontroller	STM32L496ZGT6
Frequency	80MHz
Flash	1MB
RAM	128KB
Vendor	ST

Configuration

Please use nucleo_l496zg ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[env:nucleo_1496zg]
platform = ststm32
board = nucleo_1496zg
```

You can override default ST Nucleo L496ZG settings per build environment using *board_**** option, where *** is a JSON object path from board manifest *nucleo_1496zg.json*. For example, *board_build.mcu*, *board_build.f_cpu*, etc.

```
[env:nucleo_1496zg]
platform = ststm32
board = nucleo_1496zg

; 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:

```
[env:nucleo_1496zg]
platform = ststm32
board = nucleo_1496zg

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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>J-LINK</i>		
<i>ST-LINK</i>	Yes	Yes

Frameworks

Name	Description
<i>Arduino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>mbed</i>	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.
<i>STM32</i>	STM32Cube embedded 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.

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.

Microcontroller	STM32L496ZGT6P
Frequency	80MHz
Flash	1MB
RAM	320KB
Vendor	ST

Configuration

Please use nucleo_l496zg_p ID for `board` option in “*platformio.ini*” (*Project Configuration File*):

```
[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.

```
[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:

```
[env:nucleo_l496zg_p]
platform = ststm32
board = nucleo_l496zg_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.

Compatible Tools	On-board	Default
Black Magic Probe		
J-LINK		
ST-LINK	Yes	Yes

Frameworks

Name	Description
Arduin	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
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.
STM32	STM32Cube embedded 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.

ST Nucleo L4R5ZI

Contents

- *ST Nucleo L4R5ZI*
 - *Hardware*
 - *Configuration*
 - *Uploading*
 - *Debugging*
 - *Frameworks*

Hardware

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

Microcontroller	STM32L4R5ZIT6
Frequency	120MHz
Flash	2MB
RAM	640KB
Vendor	ST

Configuration

Please use `nucleo_l4r5zi` ID for `board` option in “*platformio.ini*” (*Project Configuration File*):

```
[env:nucleo_l4r5zi]
platform = ststm32
board = nucleo_l4r5zi
```

You can override default ST Nucleo L4R5ZI settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `nucleo_l4r5zi.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:nucleo_l4r5zi]
platform = ststm32
board = nucleo_l4r5zi

; change microcontroller
board_build.mcu = stm32l4r5zit6

; change MCU frequency
board_build.f_cpu = 120000000L
```

Uploading

ST Nucleo L4R5ZI supports the next uploading protocols:

- blackmagic
- jlink
- mbed
- stlink

Default protocol is stlink

You can change upload protocol using *upload_protocol* option:

```
[env:nucleo_14r5zi]
platform = ststm32
board = nucleo_14r5zi

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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>J-LINK</i>		
<i>ST-LINK</i>	Yes	Yes

Frameworks

Name	Description
<i>Arduino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>mbed</i>	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.
<i>STM32</i>	STM32Cube embedded 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.

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.

Microcontroller	STM32F030R8T6
Frequency	48MHz
Flash	64KB
RAM	8KB
Vendor	ST

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:

```
[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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>J-LINK</i>		
<i>ST-LINK</i>	Yes	Yes

Frameworks

Name	Description
<i>Arduino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>mbed</i>	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.
<i>STM32</i>	STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.

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.

Microcontroller	STM32F051R8T6
Frequency	48MHz
Flash	64KB
RAM	8KB
Vendor	ST

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:

```
[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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>J-LINK</i>		
<i>ST-LINK</i>	Yes	Yes

Frameworks

Name	Description
<i>mbed</i>	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.
<i>STM32</i>	STM32Cube embedded 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.

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.

Microcontroller	STM32F303VCT6
Frequency	72MHz
Flash	256KB
RAM	48KB
Vendor	ST

Configuration

Please use `disco_f303vc` ID for `board` option in “*platformio.ini*” (*Project Configuration File*):

```
[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.

```
[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:

```
[env:disco_f303vc]
platform = ststm32
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.

Compatible Tools	On-board	Default
Black Magic Probe		
J-LINK		
ST-LINK	Yes	Yes

Frameworks

Name	Description
<i>CMSIS</i>	The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.
<i>libOpenCM3</i>	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.
<i>mbed</i>	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.
<i>SPL</i>	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.
<i>STM32Cube</i>	STM32Cube embedded 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.

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.

Microcontroller	STM32F407VGT6
Frequency	168MHz
Flash	1MB
RAM	128KB
Vendor	ST

Configuration

Please use disco_f407vg ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[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.

```
[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:

```
[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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>J-LINK</i>		
<i>ST-LINK</i>	Yes	Yes

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>CMSIS</i>	The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.
<i>libOpenCM3</i>	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.
<i>mbed</i>	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.
<i>SPL</i>	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.
<i>STM32Cube</i>	STM32Cube embedded 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.

ST STM32L073Z-EVAL

Contents

- *ST STM32L073Z-EVAL*
 - *Hardware*
 - *Configuration*
 - *Uploading*
 - *Debugging*
 - *Frameworks*

Hardware

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

Microcontroller	STM32L073VZT6
Frequency	32MHz
Flash	192KB
RAM	20KB
Vendor	ST

Configuration

Please use eval_1073z ID for `board` option in “*platformio.ini*” (*Project Configuration File*):

```
[env:eval_1073z]
platform = ststm32
board = eval_1073z
```

You can override default ST STM32L073Z-EVAL settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `eval_1073z.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:eval_1073z]
platform = ststm32
board = eval_1073z

; 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:

```
[env:eval_1073z]
platform = ststm32
board = eval_1073z

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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>J-LINK</i>		
<i>ST-LINK</i>	Yes	Yes

Frameworks

Name	Description
<i>STM32Cube</i>	STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.

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.

Microcontroller	STM32L152RBT6
Frequency	32MHz
Flash	128KB
RAM	16KB
Vendor	ST

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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>J-LINK</i>		
<i>ST-LINK</i>	Yes	Yes

Frameworks

Name	Description
<i>CMSIS</i>	The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.
<i>libOpenCM3</i>	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.
<i>SPL</i>	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.
<i>STM32Cube</i>	STM32Cube embedded 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.

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.

Microcontroller	STM32F100RBT6
Frequency	24MHz
Flash	128KB
RAM	8KB
Vendor	ST

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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>J-LINK</i>		
<i>ST-LINK</i>	Yes	Yes

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>mbed</i>	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.
<i>STM32</i>	STM32Cube embedded 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.

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.

Microcontroller	STM32L476JG
Frequency	80MHz
Flash	1MB
RAM	128KB
Vendor	Avnet Silica

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)

(continued from previous 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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>J-LINK</i>		
<i>ST-LINK</i>	Yes	Yes

Frameworks

Name	Description
<i>mbed</i>	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.
<i>STM32</i>	STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.

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.

Microcontroller	STM32F103C8T6
Frequency	72MHz
Flash	64KB
RAM	20KB
Vendor	Generic

Configuration

Please use genericSTM32F103C8 ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[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.

```
[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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		Yes
<i>J-LINK</i>		
<i>ST-LINK</i>		

Frameworks

Name	Description
<i>Arduino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>CMSIS</i>	The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.
<i>libOpenCM3</i>	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.
<i>mbed</i>	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.
<i>STM32Cube</i>	STM32Cube embedded 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.

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.

Microcontroller	STM32F103CBT6
Frequency	72MHz
Flash	128KB
RAM	20KB
Vendor	Generic

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 = ststm32
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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		Yes
<i>J-LINK</i>		
<i>ST-LINK</i>		

Frameworks

Name	Description
Arduino	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
CMSIS	The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.
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.
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.

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.

Microcontroller	STM32F103R8T6
Frequency	72MHz
Flash	64KB
RAM	20KB
Vendor	Generic

Configuration

Please use genericSTM32F103R8 ID for `board` option in “`platformio.ini`” (Project Configuration File):

```
[env:genericSTM32F103R8]
platform = ststm32
board = genericSTM32F103R8
```

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.

```
[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:

```
[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.

Compatible Tools	On-board	Default
Black Magic Probe		Yes
J-LINK		
ST-LINK		

Frameworks

Name	Description
Arduino	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
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.
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.
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.

STM32F103RB (20k RAM. 128k Flash)

Contents

- [*STM32F103RB \(20k RAM. 128k Flash\)*](#)
 - [*Hardware*](#)
 - [*Configuration*](#)
 - [*Uploading*](#)
 - [*Debugging*](#)
 - [*Frameworks*](#)

Hardware

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

Microcontroller	STM32F103RBT6
Frequency	72MHz
Flash	128KB
RAM	20KB
Vendor	Generic

Configuration

Please use `genericSTM32F103RB` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[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.

```
[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:

```
[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.

Compatible Tools	On-board	Default
Black Magic Probe		Yes
J-LINK		
ST-LINK		

Frameworks

Name	Description
Arduin	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
CM-SIS	The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.
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.
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.
STM32	STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.

STM32F103RC (48k RAM. 256k Flash)

Contents

- [STM32F103RC \(48k RAM. 256k Flash\)](#)
 - [Hardware](#)
 - [Configuration](#)
 - [Uploading](#)
 - [Debugging](#)

– *Frameworks*

Hardware

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

Microcontroller	STM32F103RCT6
Frequency	72MHz
Flash	256KB
RAM	48KB
Vendor	Generic

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 = ststm32
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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		Yes
<i>J-LINK</i>		
<i>ST-LINK</i>		

Frameworks

Name	Description
<i>Arduino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>CMSIS</i>	The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.
<i>libOpenCM3</i>	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.
<i>STM32Cube</i>	STM32Cube embedded 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.

STM32F103RE (64k RAM, 512k Flash)

Contents

- *STM32F103RE (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.

Microcontroller	STM32F103RET6
Frequency	72MHz
Flash	512KB
RAM	64KB
Vendor	Generic

Configuration

Please use `genericSTM32F103RE` ID for `board` option in “`platformio.ini`” (Project Configuration File):

```
[env:genericSTM32F103RE]
platform = ststm32
board = genericSTM32F103RE
```

You can override default STM32F103RE (64k RAM. 512k Flash) settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `genericSTM32F103RE.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:genericSTM32F103RE]
platform = ststm32
board = genericSTM32F103RE

; change microcontroller
board_build.mcu = stm32f103ret6

; change MCU frequency
board_build.f_cpu = 72000000L
```

Uploading

STM32F103RE (64k RAM. 512k Flash) supports the next uploading protocols:

- blackmagic
- 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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		Yes
<i>J-LINK</i>		
<i>ST-LINK</i>		

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>CMSIS</i>	The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.
<i>libOpenCM3</i>	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.
<i>STM32Cube</i>	STM32Cube embedded 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.

STM32F103T8 (20k RAM. 64k Flash)

Contents

- STM32F103T8 (20k RAM. 64k Flash)
 - Hardware
 - Configuration
 - Uploading
 - Debugging
 - Frameworks

Hardware

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

Microcontroller	STM32F103T8T6
Frequency	72MHz
Flash	20KB
RAM	64KB
Vendor	Generic

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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		Yes
<i>J-LINK</i>		
<i>ST-LINK</i>		

Frameworks

Name	Description
Arduino	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
CMSIS	The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.
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.
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.

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.

Microcontroller	STM32F103TBT6
Frequency	72MHz
Flash	128KB
RAM	20KB
Vendor	Generic

Configuration

Please use genericSTM32F103TB ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[env:genericSTM32F103TB]
platform = ststm32
board = genericSTM32F103TB
```

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	On-board	Default
<i>Black Magic Probe</i>		Yes
<i>J-LINK</i>		
<i>ST-LINK</i>		

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>CMSIS</i>	The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.
<i>libOpenCM3</i>	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.
<i>STM32Cube</i>	STM32Cube embedded 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.

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.

Microcontroller	STM32F103VBT6
Frequency	72MHz
Flash	128KB
RAM	20KB
Vendor	Generic

Configuration

Please use `genericSTM32F103VB` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[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.

```
[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:

```
[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.

Compatible Tools	On-board	Default
Black Magic Probe		Yes
J-LINK		
ST-LINK		

Frameworks

Name	Description
Arduin	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
CM-SIS	The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.
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.
STM32	STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.

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.

Microcontroller	STM32F103VCT6
Frequency	72MHz
Flash	256KB
RAM	48KB
Vendor	Generic

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:

```
[env:genericSTM32F103VC]
platform = ststm32
board = genericSTM32F103VC

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*).

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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		Yes
<i>J-LINK</i>		
<i>ST-LINK</i>		

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>CMSIS</i>	The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.
<i>libOpenCM3</i>	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.
<i>STM32</i>	STM32Cube embedded 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.

STM32F103VD (64k RAM, 384k Flash)

Contents

- *STM32F103VD (64k RAM. 384k Flash)*
 - *Hardware*
 - *Configuration*
 - *Uploading*
 - *Debugging*
 - *Frameworks*

Hardware

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

Microcontroller	STM32F103VDT6
Frequency	72MHz
Flash	384KB
RAM	64KB
Vendor	Generic

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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		Yes
<i>J-LINK</i>		
<i>ST-LINK</i>		

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>CMSIS</i>	The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.
<i>libOpenCM3</i>	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.
<i>STM32Cube</i>	STM32Cube embedded 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.

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.

Microcontroller	STM32F103VET6
Frequency	72MHz
Flash	512KB
RAM	64KB
Vendor	Generic

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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		Yes
<i>J-LINK</i>		
<i>ST-LINK</i>		

Frameworks

Name	Description
Arduino	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
CMSIS	The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.
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.
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.

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.

Microcontroller	STM32F103ZCT6
Frequency	72MHz
Flash	256KB
RAM	48KB
Vendor	Generic

Configuration

Please use genericSTM32F103ZC ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[env:genericSTM32F103ZC]
platform = ststm32
board = genericSTM32F103ZC
```

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.

```
[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:

```
[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	On-board	Default
<i>Black Magic Probe</i>		Yes
<i>J-LINK</i>		
<i>ST-LINK</i>		

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>CMSIS</i>	The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.
<i>libOpenCM3</i>	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.
<i>STM32Cube</i>	STM32Cube embedded 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.

STM32F103ZD (64k RAM. 384k Flash)

Contents

- *STM32F103ZD (64k RAM. 384k Flash)*
 - *Hardware*
 - *Configuration*
 - *Uploading*
 - *Debugging*
 - *Frameworks*

Hardware

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

Microcontroller	STM32F103ZDT6
Frequency	72MHz
Flash	384KB
RAM	64KB
Vendor	Generic

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.

Compatible Tools	On-board	Default
Black Magic Probe		Yes
J-LINK		
ST-LINK		

Frameworks

Name	Description
Arduin	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
CM-SIS	The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.
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.
STM32	STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.

STM32F103ZE (64k RAM. 512k Flash)

Contents

- [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.

Microcontroller	STM32F103ZET6
Frequency	72MHz
Flash	512KB
RAM	64KB
Vendor	Generic

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:

```
[env:genericSTM32F103ZE]
platform = ststm32
board = genericSTM32F103ZE

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*).

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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		Yes
<i>J-LINK</i>		
<i>ST-LINK</i>		

Frameworks

Name	Description
<i>Arduino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>CMSIS</i>	The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.
<i>libOpenCM3</i>	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.
<i>STM32Cube</i>	STM32Cube embedded 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.

STM32F303CB (32k RAM, 128k Flash)

Contents

- *STM32F303CB (32k 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.

Microcontroller	STM32F303CBT6
Frequency	72MHz
Flash	128KB
RAM	32KB
Vendor	Generic

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:

```
[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.

Compatible Tools	On-board	Default
Black Magic Probe		Yes
J-LINK		
ST-LINK		

Frameworks

Name	Description
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.
STM32	STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.

STM32F407VE (192k RAM. 512k Flash)

Contents

- *STM32F407VE (192k 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.

Microcontroller	STM32F407VET6
Frequency	168MHz
Flash	502.23KB
RAM	128KB
Vendor	Generic

Configuration

Please use `genericSTM32F407VET6` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[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.

```
[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.

Compatible Tools	On-board	Default
J-LINK		
ST-LINK		Yes

Frameworks

Name	Description
Arduino	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
CMSIS	The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.
STM32	STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.

STM32F407VG (192k RAM, 1024k Flash)

Contents

- [STM32F407VG \(192k RAM, 1024k Flash\)](#)

- [Hardware](#)
- [Configuration](#)
- [Uploading](#)
- [Debugging](#)
- [Frameworks](#)

Hardware

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

Microcontroller	STM32F407VGT6
Frequency	168MHz
Flash	1MB
RAM	192KB
Vendor	Generic

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.

Compatible Tools	On-board	Default
J-LINK		
ST-LINK		Yes

Frameworks

Name	Description
Arduino	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
CMSIS	The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.
STM32	STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.

STM32F4Stamp F405

Contents

- [STM32F4Stamp F405](#)

- [Hardware](#)
- [Configuration](#)
- [Uploading](#)
- [Debugging](#)
- [Frameworks](#)

Hardware

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

Microcontroller	STM32F405RG
Frequency	168MHz
Flash	1MB
RAM	192KB
Vendor	Generic

Configuration

Please use `stm32f4stamp` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:stm32f4stamp]
platform = ststm32
board = stm32f4stamp
```

You can override default STM32F4Stamp F405 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `stm32f4stamp.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:stm32f4stamp]
platform = ststm32
board = stm32f4stamp

; change microcontroller
board_build.mcu = stm32f405rgt6

; change MCU frequency
board_build.f_cpu = 168000000L
```

Uploading

STM32F4Stamp F405 supports the next uploading protocols:

- dfu
- jlink
- stlink

Default protocol is `dfu`

You can change upload protocol using `upload_protocol` option:

```
[env:stm32f4stamp]
platform = 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.

Compatible Tools	On-board	Default
<i>J-LINK</i>		
<i>ST-LINK</i>		Yes

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>STM32</i>	STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.

STM32F7508-DK

Contents

- *STM32F7508-DK*
 - *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	STM32F750N8H6
Frequency	216MHz
Flash	64KB
RAM	340KB
Vendor	ST

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:

```
[env:disco_f750n8]
platform = ststm32
board = disco_f750n8

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*).

STM32F7508-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
<i>Black Magic Probe</i>		
<i>J-LINK</i>		
<i>ST-LINK</i>	Yes	Yes

Frameworks

Name	Description
<i>STM32</i>	STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.

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.

Microcontroller	STM32F407VET6
Frequency	168MHz
Flash	512KB
RAM	192KB
Vendor	SeeedStudio

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)

(continued from previous 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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>J-LINK</i>		
<i>ST-LINK</i>	Yes	Yes

Frameworks

Name	Description
<i>mbed</i>	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.
<i>STM32</i>	STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.

Seeed Wio 3G

Contents

- *Seeed Wio 3G*
 - *Hardware*
 - *Configuration*
 - *Uploading*
 - *Debugging*
 - *Frameworks*

Hardware

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

Microcontroller	STM32F439VI
Frequency	180MHz
Flash	2MB
RAM	256KB
Vendor	SeeedStudio

Configuration

Please use `wio_3g` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:wio_3g]
platform = ststm32
board = wio_3g
```

You can override default Seeed Wio 3G settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `wio_3g.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:wio_3g]
platform = ststm32
board = wio_3g

; change microcontroller
board_build.mcu = stm32f439vi

; change MCU frequency
board_build.f_cpu = 180000000L
```

Uploading

Seeed Wio 3G supports the next uploading protocols:

- blackmagic
- jlink
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```
[env:wio_3g]
platform = ststm32
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*).

Seed Wio 3G 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
<i>Black Magic Probe</i>		
<i>J-LINK</i>		
<i>ST-LINK</i>	Yes	Yes

Frameworks

Name	Description
<i>mbed</i>	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.
<i>STM32</i>	STM32Cube embedded 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.

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.

Microcontroller	STM32F303CCT6
Frequency	72MHz
Flash	256KB
RAM	40KB
Vendor	TauLabs

Configuration

Please use sparky_v1 ID for `board` option in “*platformio.ini*” (*Project Configuration File*):

```
[env:sparky_v1]
platform = ststm32
board = sparky_v1
```

You can override default Sparky V1 F303 settings per build environment using `board_***` option, where *** is a JSON object path from board manifest `sparky_v1.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:sparky_v1]
platform = ststm32
board = sparky_v1

; change microcontroller
board_build.mcu = stm32f303cct6

; change MCU frequency
board_build.f_cpu = 72000000L
```

Uploading

Sparky V1 F303 supports the next uploading protocols:

- blackmagic
- jlink
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```
[env:sparky_v1]
platform = ststm32
board = sparky_v1

upload_protocol = stlink
```

Debugging

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

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for 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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		Yes
<i>J-LINK</i>		
<i>ST-LINK</i>		

Frameworks

Name	Description
<i>Ar-duino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>STM32</i>	STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.

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.

Microcontroller	STM32F103TBU6
Frequency	72MHz
Flash	128KB
RAM	20KB
Vendor	HY

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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		Yes
<i>J-LINK</i>		
<i>ST-LINK</i>		

Frameworks

Name	Description
<i>Ar-duino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>STM32</i>	STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.

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.

Microcontroller	STM32F446RET6
Frequency	180MHz
Flash	512KB
RAM	128KB
Vendor	VAE

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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		Yes
<i>J-LINK</i>		
<i>ST-LINK</i>		

Frameworks

Name	Description
<i>Ar-duino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>STM32</i>	STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.

sakura.io Evaluation Board

Contents

- *sakura.io Evaluation Board*
 - *Hardware*
 - *Configuration*
 - *Uploading*
 - *Debugging*
 - *Frameworks*

Hardware

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

Microcontroller	STM32F411RET6
Frequency	100MHz
Flash	1MB
RAM	128KB
Vendor	sakura.io

Configuration

Please use `sakuraio_evb_01` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[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.

```
[env:sakuraio_evb_01]
platform = ststm32
board = sakuraio_evb_01

; change microcontroller
board_build.mcu = stm32f411ret6

; change MCU frequency
board_build.f_cpu = 100000000L
```

Uploading

`sakura.io` Evaluation Board supports the next uploading protocols:

- blackmagic
- cmsis-dap
- jlink
- mbed
- stlink

Default protocol is `mbed`

You can change upload protocol using `upload_protocol` option:

```
[env:sakuraio_evb_01]
platform = ststm32
board = sakuraio_evb_01

upload_protocol = mbed
```

Debugging

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

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “`platformio.ini`” (*Project Configuration File*).

sakura.io Evaluation Board has on-board debug probe and **IS READY** for debugging. You don't need to use/buy external debug probe.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>CMSIS-DAP</i>	Yes	Yes
<i>J-LINK</i>		
<i>ST-LINK</i>		

Frameworks

Name	Description
<i>mbed</i>	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.
<i>STM32</i>	STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.

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.

Microcontroller	STM32F437VG
Frequency	180MHz
Flash	1MB
RAM	256KB
Vendor	u-blox

Configuration

Please use ublox_c030_n211 ID for *board* option in “*platformio.ini*” (*Project Configuration File*):

```
[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.

```
[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:

```
[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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		Yes
<i>CMSIS-DAP</i>		
<i>J-LINK</i>		
<i>ST-LINK</i>		

Frameworks

Name	Description
<i>mbed</i>	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.
<i>STM32</i>	STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.

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.

Microcontroller	STM32F437VG
Frequency	180MHz
Flash	1MB
RAM	256KB
Vendor	u-blox

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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		
<i>J-LINK</i>		
<i>ST-LINK</i>	Yes	Yes

Frameworks

Name	Description
<i>mbed</i>	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.
<i>STM32</i>	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 light-weight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.

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.

Microcontroller	STM32F437VG
Frequency	180MHz
Flash	1MB
RAM	256KB
Vendor	u-blox

Configuration

Please use `ublox_c030_u201` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[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.

```
[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:

```
[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.

Compatible Tools	On-board	Default
<i>Black Magic Probe</i>		Yes
<i>CMSIS-DAP</i>		
<i>J-LINK</i>		
<i>ST-LINK</i>		

Frameworks

Name	Description
<i>mbed</i>	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.
<i>STM32</i>	STM32Cube embedded software libraries, including: The HAL hardware abstraction layer, enabling portability between different STM32 devices via standardized API calls; The Low-Layer (LL) APIs, a lightweight, optimized, expert oriented set of APIs designed for both performance and runtime efficiency.

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.

Microcontroller	STM32F439ZIY6
Frequency	168MHz
Flash	2MB
RAM	256KB
Vendor	u-blox

Configuration

Please use ublox_evk_odin_w2 ID for `board` option in “*platformio.ini*” (*Project Configuration File*):

```
[env:ublox_evk_odin_w2]
platform = ststm32
board = ublox_evk_odin_w2
```

You can override default u-blox EVK-ODIN-W2 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `ublox_evk_odin_w2.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:ublox_evk_odin_w2]
platform = ststm32
board = ublox_evk_odin_w2

; change microcontroller
board_build.mcu = stm32f439ziy6

; change MCU frequency
board_build.f_cpu = 168000000L
```

Uploading

u-blox EVK-ODIN-W2 supports the next uploading protocols:

- blackmagic
- jlink
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```
[env:ublox_evk_odin_w2]
platform = ststm32
board = ublox_evk_odin_w2

upload_protocol = stlink
```

Debugging

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

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for 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.

Compatible Tools	On-board	Default
Black Magic Probe		Yes
J-LINK		
ST-LINK		

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.
STM32	STM32Cube embedded 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.

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.

Microcontroller	STM32F439ZIY6
Frequency	168MHz
Flash	2MB
RAM	256KB
Vendor	u-blox

Configuration

Please use `mtb_ublox_odin_w2` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:mtb_ublox_odin_w2]
platform = ststm32
board = mtb_ublox_odin_w2
```

You can override default u-blox ODIN-W2 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `mtb_ublox_odin_w2.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:mtb_ublox_odin_w2]
platform = ststm32
board = mtb_ublox_odin_w2

; change microcontroller
board_build.mcu = stm32f439ziy6

; change MCU frequency
board_build.f_cpu = 168000000L
```

Uploading

u-blox ODIN-W2 supports the next uploading protocols:

- blackmagic
- jlink
- stlink

Default protocol is `stlink`

You can change upload protocol using `upload_protocol` option:

```
[env:mtb_ublox_odin_w2]
platform = ststm32
board = mtb_ublox_odin_w2

upload_protocol = stlink
```

Debugging

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

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for 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.

Compatible Tools	On-board	Default
Black Magic Probe		Yes
J-LINK		
ST-LINK		

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.
STM32	STM32Cube embedded 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.

1.12.25 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.

Microcontroller	STM8S105C6T6
Frequency	16MHz
Flash	32KB
RAM	2KB
Vendor	ST

Configuration

Please use `stm8sdisco` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[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.

```
[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:

```
[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.

Compatible Tools	On-board	Default
ST-LINK	Yes	Yes

Frameworks

Name	Description
Ar-duino	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
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.

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.

Microcontroller	STM8S103F3P6
Frequency	16MHz
Flash	8KB
RAM	1KB
Vendor	ST

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

Name	Description
<code>Arduin</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<code>SPL</code>	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.

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.

Microcontroller	STM8S105K4T6
Frequency	16MHz
Flash	16KB
RAM	2KB
Vendor	ST

Configuration

Please use `stm8sblack` ID for `board` option in “*platformio.ini*” (*Project Configuration File*):

```
[env:stm8sblack]
platform = ststm8
board = stm8sblack
```

You can override default ST STM8S105K4T6 Breakout Board settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `stm8sblack.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:stm8sblack]
platform = ststm8
board = stm8sblack

; change microcontroller
board_build.mcu = stm8s105k4t6

; change MCU frequency
board_build.f_cpu = 16000000L
```

Uploading

ST STM8S105K4T6 Breakout Board supports the next uploading protocols:

- serial
- stlinkv2

Default protocol is `serial`

You can change upload protocol using `upload_protocol` option:

```
[env:stm8sblack]
platform = ststm8
board = stm8sblack

upload_protocol = serial
```

Debugging

PIO Unified Debugger currently does not support ST STM8S105K4T6 Breakout Board board.

Frameworks

Name	Description
<i>Ar-duino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>SPL</i>	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.

sduino MB (STM8S208MBT6B)

Contents

- *sduino MB (STM8S208MBT6B)*
 - *Hardware*
 - *Configuration*
 - *Uploading*
 - *Debugging*
 - *Frameworks*

Hardware

Platform *ST STM8*: The STM8 is an 8-bit microcontroller family by STMicroelectronics an extended variant of the ST7 microcontroller architecture. STM8 microcontrollers are particularly low cost for a full-featured 8-bit microcontroller.

Microcontroller	STM8S208MBT6
Frequency	16MHz
Flash	128KB
RAM	6KB
Vendor	sduino

Configuration

Please use `mb208` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[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.

```
[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:

```
[env:mb208]
platform = ststm8
board = mb208

upload_protocol = serial
```

Debugging

PIO Unified Debugger currently does not support sduino MB (STM8S208MBT6B) board.

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>SPL</i>	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.

sduino UNO (STM8S105K6)

Contents

- *sduino UNO (STM8S105K6)*
 - *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.

Microcontroller	STM8S105K6T6
Frequency	16MHz
Flash	32KB
RAM	2KB
Vendor	sduino

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.

```
[env:s8uno]
platform = ststm8
board = s8uno

; change microcontroller
board_build.mcu = stm8s105k6t6

; change MCU frequency
board_build.f_cpu = 16000000L
```

Uploading

sduino UNO (STM8S105K6) supports the next uploading protocols:

- serial
- stlinkv2

Default protocol is `serial`

You can change upload protocol using `upload_protocol` option:

```
[env:s8uno]
platform = ststm8
board = s8uno

upload_protocol = serial
```

Debugging

PIO Unified Debugger currently does not support sduino UNO (STM8S105K6) board.

Frameworks

Name	Description
<code>Ar-duino</code>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<code>SPL</code>	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.

1.12.26 Teensy

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.

Microcontroller	ATMEGA32U4
Frequency	16MHz
Flash	31.50KB
RAM	2.50KB
Vendor	Teensy

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:

```
[env:teensy2]
platform = teensy
board = teensy2

upload_protocol = teensy-gui
```

Debugging

PIO Unified Debugger currently does not support Teensy 2.0 board.

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 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.

Microcontroller	MK20DX128
Frequency	48MHz
Flash	128KB
RAM	16KB
Vendor	Teensy

Configuration

Please use `teensy30` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:teensy30]
platform = teensy
board = teensy30
```

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

Name	Description
<code>Ar-duino</code>	Arduino Wiring-based Framework allows writing cross-platform software 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 3.1 / 3.2

Contents

- *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.

Microcontroller	MK20DX256
Frequency	72MHz
Flash	256KB
RAM	64KB
Vendor	Teensy

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.

Compatible Tools	On-board	Default
J-LINK		Yes

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.
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.

Teensy 3.5

Contents

- [Teensy 3.5](#)
 - [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.

Microcontroller	MK64FX512
Frequency	120MHz
Flash	512KB
RAM	255.99KB
Vendor	Teensy

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.

Compatible Tools	On-board	Default
J-LINK		Yes

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 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.

Microcontroller	MK66FX1M0
Frequency	180MHz
Flash	1MB
RAM	256KB
Vendor	Teensy

Configuration

Please use `teensy36` ID for `board` option in “*platformio.ini*” (*Project Configuration File*):

```
[env:teensy36]
platform = teensy
board = teensy36
```

You can override default Teensy 3.6 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `teensy36.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:teensy36]
platform = teensy
board = teensy36

; change microcontroller
board_build.mcu = mk66fx1m0

; change MCU frequency
board_build.f_cpu = 180000000L
```

Uploading

Teensy 3.6 supports the next uploading protocols:

- jlink
- teensy-cli
- teensy-gui

Default protocol is `teensy-gui`

You can change upload protocol using `upload_protocol` option:

```
[env:teensy36]
platform = teensy
board = teensy36

upload_protocol = teensy-gui
```

Debugging

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

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for 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.

Compatible Tools	On-board	Default
J-LINK		Yes

Frameworks

Name	Description
Ardu- ino	Arduino Wiring-based Framework allows writing cross-platform software 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 4.0

Contents

- *Teensy 4.0*
 - *Hardware*
 - *Configuration*
 - *Uploading*
 - *Debugging*
 - *Frameworks*

Hardware

Platform [*Teensy*](#): Teensy is a complete USB-based microcontroller development system, in a very small footprint, capable of implementing many types of projects. All programming is done via the USB port. No special programmer is needed, only a standard USB cable and a PC or Macintosh with a USB port.

Microcontroller	IMXRT1062
Frequency	600MHz
Flash	1.94MB
RAM	1MB
Vendor	Teensy

Configuration

Please use `teensy40` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:teensy40]
platform = teensy
board = teensy40
```

You can override default Teensy 4.0 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `teensy40.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:teensy40]
platform = teensy
board = teensy40

; change microcontroller
board_build.mcu = imxrt1062

; change MCU frequency
board_build.f_cpu = 600000000L
```

Uploading

Teensy 4.0 supports the next uploading protocols:

- jlink
- teensy-gui

Default protocol is `teensy-gui`

You can change upload protocol using `upload_protocol` option:

```
[env:teensy40]
platform = teensy
board = teensy40

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 4.0 does not have on-board debug probe and **IS NOT READY** for debugging. You will need to use/buy one of external probe listed below.

Compatible Tools	On-board	Default
J-LINK		Yes

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 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.

Microcontroller	MKL26Z64
Frequency	48MHz
Flash	62KB
RAM	8KB
Vendor	Teensy

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 = mk126z64

; 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.

Compatible Tools	On-board	Default
J-LINK		Yes

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.

Microcontroller	AT90USB1286
Frequency	16MHz
Flash	127KB
RAM	8KB
Vendor	Teensy

Configuration

Please use `teensy2pp` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:teensy2pp]
platform = teensy
board = teensy2pp
```

You can override default Teensy++ 2.0 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `teensy2pp.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:teensy2pp]
platform = teensy
board = teensy2pp
```

(continues on next page)

(continued from previous page)

```
; 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:

```
[env:teensy2pp]
platform = teensy
board = teensy2pp

upload_protocol = teensy-gui
```

Debugging

PIO Unified Debugger currently does not support Teensy++ 2.0 board.

Frameworks

Name	Description
<i>Ardu</i> <i>duino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

1.12.27 TI MSP430

TI FraunchPad MSP-EXP430FR5739LP

Contents

- *TI FraunchPad MSP-EXP430FR5739LP*
 - *Hardware*
 - *Configuration*
 - *Debugging*

- *Frameworks*

Hardware

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

Microcontroller	MSP430FR5739
Frequency	16MHz
Flash	15.37KB
RAM	1KB
Vendor	TI

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.

Compatible Tools	On-board	Default
MSP Debug	Yes	Yes

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.

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.

Microcontroller	MSP430F5529
Frequency	25MHz
Flash	47KB
RAM	8KB
Vendor	TI

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 = 25000000L
```

Debugging

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

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for 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.

Compatible Tools	On-board	Default
MSP Debug	Yes	Yes

Frameworks

Name	Description
Arduin	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

TI LaunchPad MSP-EXP430FR2311LP

Contents

- [TI LaunchPad MSP-EXP430FR2311LP](#)
 - [Hardware](#)
 - [Configuration](#)
 - [Debugging](#)
 - [Frameworks](#)

Hardware

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

Microcontroller	MSP430FR2311
Frequency	16MHz
Flash	3.75KB
RAM	1KB
Vendor	TI

Configuration

Please use `lpmsp430fr2311` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:lpmsp430fr2311]
platform = timsp430
board = lpmsp430fr2311
```

You can override default TI LaunchPad MSP-EXP430FR2311LP settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `lpmsp430fr2311.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:lpmsp430fr2311]
platform = timsp430
board = lpmsp430fr2311

; change microcontroller
board_build.mcu = msp430fr2311

; change MCU frequency
board_build.f_cpu = 16000000L
```

Debugging

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

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging *Tools & Debug Probes* using `debug_tool` option in “`platformio.ini`” (*Project Configuration File*).

TI LaunchPad MSP-EXP430FR2311LP has on-board debug probe and **IS READY** for debugging. You don't need to use/buy external debug probe.

Compatible Tools	On-board	Default
MSP Debug	Yes	Yes

Frameworks

Name	Description
<i>Arduino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

TI LaunchPad MSP-EXP430FR2433LP

Contents

- *TI LaunchPad MSP-EXP430FR2433LP*
 - *Hardware*
 - *Configuration*
 - *Debugging*
 - *Frameworks*

Hardware

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

Microcontroller	MSP430FR2433
Frequency	8MHz
Flash	15KB
RAM	4KB
Vendor	TI

Configuration

Please use `lpmsp430fr2433` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:lpmsp430fr2433]
platform = timsp430
board = lpmsp430fr2433
```

You can override default TI LaunchPad MSP-EXP430FR2433LP settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `lpmsp430fr2433.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:lpmsp430fr2433]
platform = timsp430
board = lpmsp430fr2433

; change microcontroller
```

(continues on next page)

(continued from previous page)

```
board_build.mcu = msp430fr2433
; 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-EXP430FR2433LP 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
MSP Debug	Yes	Yes

Frameworks

Name	Description
Ardu- ino	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

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.

Microcontroller	MSP430FR4133
Frequency	8MHz
Flash	15KB
RAM	2KB
Vendor	TI

Configuration

Please use `lpmsp430fr4133` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:lpmsp430fr4133]
platform = timsps430
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.

```
[env:lpmsp430fr4133]
platform = timsps430
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.

Compatible Tools	On-board	Default
<code>MSP Debug</code>	Yes	Yes

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.

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.

Microcontroller	MSP430FR5969
Frequency	8MHz
Flash	47KB
RAM	2KB
Vendor	TI

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
```

(continues on next page)

(continued from previous page)

```
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.

Compatible Tools	On-board	Default
MSP Debug	Yes	Yes

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.

TI LaunchPad MSP-EXP430FR5994LP

Contents

- [TI LaunchPad MSP-EXP430FR5994LP](#)
 - [Hardware](#)
 - [Configuration](#)
 - [Debugging](#)
 - [Frameworks](#)

Hardware

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

Microcontroller	MSP430FR5994
Frequency	16MHz
Flash	256KB
RAM	4KB
Vendor	TI

Configuration

Please use `lpmsp430fr5994` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:lpmsp430fr5994]
platform = timsp430
board = lpmsp430fr5994
```

You can override default TI LaunchPad MSP-EXP430FR5994LP settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `lpmsp430fr5994.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:lpmsp430fr5994]
platform = timsp430
board = lpmsp430fr5994

; change microcontroller
board_build.mcu = msp430fr5994

; 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-EXP430FR5994LP 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
<code>MSP Debug</code>	Yes	Yes

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.

TI LaunchPad MSP-EXP430FR6989LP

Contents

- [TI LaunchPad MSP-EXP430FR6989LP](#)
 - [Hardware](#)
 - [Configuration](#)
 - [Debugging](#)
 - [Frameworks](#)

Hardware

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

Microcontroller	MSP430FR6989
Frequency	8MHz
Flash	47KB
RAM	2KB
Vendor	TI

Configuration

Please use `lpmsp430fr6989` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:lpmsp430fr6989]
platform = timsp430
board = lpmsp430fr6989
```

You can override default TI LaunchPad MSP-EXP430FR6989LP settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `lpmsp430fr6989.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:lpmsp430fr6989]
platform = timsp430
board = lpmsp430fr6989

; change microcontroller
```

(continues on next page)

(continued from previous page)

```
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.

Compatible Tools	On-board	Default
MSP Debug	Yes	Yes

Frameworks

Name	Description
Ardu- ino	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

TI LaunchPad MSP-EXP430G2 w/ MSP430G2231

Contents

- [TI LaunchPad MSP-EXP430G2 w/ MSP430G2231](#)
 - [Hardware](#)
 - [Configuration](#)
 - [Debugging](#)
 - [Frameworks](#)

Hardware

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

Microcontroller	MSP430G2231
Frequency	1MHz
Flash	2KB
RAM	256B
Vendor	TI

Configuration

Please use `lpmsp430g2231` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:lpmsp430g2231]
platform = timsdp430
board = lpmsp430g2231
```

You can override default TI LaunchPad MSP-EXP430G2 w/ MSP430G2231 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `lpmsp430g2231.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:lpmsp430g2231]
platform = timsdp430
board = lpmsp430g2231

; change microcontroller
board_build.mcu = msp430g2231

; change MCU frequency
board_build.f_cpu = 1000000L
```

Debugging

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

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for the further instructions and configuration information.

You can switch between debugging `Tools & Debug Probes` using `debug_tool` option in “`platformio.ini`” (*Project Configuration File*).

TI LaunchPad MSP-EXP430G2 w/ MSP430G2231 has on-board debug probe and **IS READY** for debugging. You don't need to use/buy external debug probe.

Compatible Tools	On-board	Default
<code>MSP Debug</code>	Yes	Yes

Frameworks

Name	Description
<i>Arduino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

TI LaunchPad MSP-EXP430G2 w/ MSP430G2452

Contents

- *TI LaunchPad MSP-EXP430G2 w/ MSP430G2452*
 - *Hardware*
 - *Configuration*
 - *Debugging*
 - *Frameworks*

Hardware

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

Microcontroller	MSP430G2452
Frequency	16MHz
Flash	8KB
RAM	256B
Vendor	TI

Configuration

Please use `lmsp430g2452` ID for `board` option in “`platformio.ini`” (Project Configuration File):

```
[env:lmsp430g2452]
platform = timsp430
board = lmsp430g2452
```

You can override default TI LaunchPad MSP-EXP430G2 w/ MSP430G2452 settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `lmsp430g2452.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:lmsp430g2452]
platform = timsp430
board = lmsp430g2452

; change microcontroller
```

(continues on next page)

(continued from previous page)

```
board_build.mcu = msp430g2452
; 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-EXP430G2 w/ MSP430G2452 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
MSP Debug	Yes	Yes

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.

TI LaunchPad MSP-EXP430G2553LP

Contents

- [TI LaunchPad MSP-EXP430G2553LP](#)
 - [Hardware](#)
 - [Configuration](#)
 - [Debugging](#)
 - [Frameworks](#)

Hardware

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

Microcontroller	MSP430G2553
Frequency	16MHz
Flash	16KB
RAM	512B
Vendor	TI

Configuration

Please use `lpmsp430g2553` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:lpmsp430g2553]
platform = tmsp430
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 = tmsp430
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.

Compatible Tools	On-board	Default
<code>MSP Debug</code>	Yes	Yes

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.

1.12.28 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.

Microcontroller	LPLM4F120H5QR
Frequency	80MHz
Flash	256KB
RAM	32KB
Vendor	TI

Configuration

Please use `lplm4f120h5qr` ID for `board` option in “`platformio.ini`” (Project Configuration File):

```
[env:lplm4f120h5qr]
platform = titiva
board = lplm4f120h5qr
```

You can override default TI LaunchPad (Stellaris) w/ lm4f120 (80MHz) settings per build environment using `board_***` option, where `***` is a JSON object path from board manifest `lplm4f120h5qr.json`. For example, `board_build.mcu`, `board_build.f_cpu`, etc.

```
[env:lplm4f120h5qr]
platform = titiva
board = lplm4f120h5qr

; change microcontroller
board_build.mcu = lplm4f120h5qr

; change MCU frequency
board_build.f_cpu = 80000000L
```

Debugging

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

Warning: You will need to install debug tool drivers depending on your system. Please click on compatible debug tool below for 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.

Compatible Tools	On-board	Default
TI-ICDI	Yes	Yes

Frameworks

Name	Description
Ar-duino	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
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.

TI LaunchPad (Tiva C) w/ tm4c123 (80MHz)

Contents

- *TI LaunchPad (Tiva C) w/ tm4c123 (80MHz)*
 - *Hardware*
 - *Configuration*
 - *Debugging*
 - *Frameworks*

Hardware

Platform [TI TIVA](#): Texas Instruments TM4C12x MCUs offer the industry's most popular ARM Cortex-M4 core with scalable memory and package options, unparalleled connectivity peripherals, advanced application functions, industry-leading analog integration, and extensive software solutions.

Microcontroller	LPTM4C1230C3PM
Frequency	80MHz
Flash	256KB
RAM	32KB
Vendor	TI

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.

Compatible Tools	On-board	Default
TI-ICDI	Yes	Yes

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.
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.

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.

Microcontroller	LPTM4C1294NCPDT
Frequency	120MHz
Flash	1MB
RAM	256KB
Vendor	TI

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.

Compatible Tools	On-board	Default
TI-ICDI	Yes	Yes

Frameworks

Name	Description
Ar-duino	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
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.

1.12.29 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

Microcontroller	WIZNET7500
Frequency	48MHz
Flash	128KB
RAM	48KB
Vendor	WIZNet

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.

Compatible Tools	On-board	Default
CMSIS-DAP	Yes	Yes
J-LINK		

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.

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

Microcontroller	WIZNET7500ECO
Frequency	48MHz
Flash	128KB
RAM	48KB
Vendor	WIZNet

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	On-board	Default
CMSIS-DAP	Yes	Yes
J-LINK		

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.

WIZwiki-W7500P

Contents

- *WIZwiki-W7500P*
 - *Hardware*
 - *Configuration*
 - *Uploading*
 - *Debugging*
 - *Frameworks*

Hardware

Platform [WIZNet W7500](#): The IOP (Internet Offload Processor) W7500 is the one-chip solution which integrates an ARM Cortex-M0, 128KB Flash and hardwired TCP/IP core for various embedded application platform especially requiring Internet of things

Microcontroller	WIZNET7500P
Frequency	48MHz
Flash	128KB
RAM	48KB
Vendor	WIZNet

Configuration

Please use `wizwiki_w7500p` ID for `board` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:wizwiki_w7500]
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_w7500]
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_w7500]
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.

Compatible Tools	On-board	Default
CMSIS-DAP	Yes	Yes
J-LINK		

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.

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.

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 operating systems and you can use them in your manifest. These packages are stored in super-fast and reliable 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",
          "shai": "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",
          "shai": "127ddac56a314b5cb1926cc1790ae4de3a03e65c",
          "version": "1.2.3",
          "system": "linux_aarch64"
        }
      ],
      "framework-%FRAMEWORK_NAME_1%": [
        ...
      ]
    }
  ]
}
```

(continues on next page)

(continued from previous page)

```

        {
            "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%": {
        "package": "framework-%FRAMEWORK_NAME_1%",
        "script": "builder/frameworks/%FRAMEWORK_NAME_1%.py"
    },
    "%FRAMEWORK_NAME_N%": {
        "package": "framework-%FRAMEWORK_NAME_N%",
        "script": "builder/frameworks/%FRAMEWORK_NAME_N%.py"
    }
},
"packages": {
    "toolchain-gccarmnoneabi": {
        "type": "toolchain",
        "version": ">=1.40803.0,<1.40805.0"
    },
    "framework-%FRAMEWORK_NAME_1%": {
        "type": "framework",
        "optional": true,
        "version": "~1.10607.0"
    },
    "framework-%FRAMEWORK_NAME_N%": {
        "type": "framework",
        "optional": true,
        "version": "~1.117.0"
    },
    "tool-direct-vcs-url": {
        "type": "uploader",
        "optional": true,
        "version": "https://github.com/user/repo.git"
    }
}
}
}

```

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`.

```

"""
    Build script for test.py
    test-builder.py
"""

from os.path import join
from SCons.Script import AlwaysBuild, Builder, Default, DefaultEnvironment

```

(continues on next page)

(continued from previous page)

```

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)
#

```

(continues on next page)

(continued from previous page)

```
# 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 existing 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

```
{  
    "build": {  
        "extra_flags": "-DHELLO_PLATFORMIO",  
        "f_cpu": "16000000L",  
        "hwids": [  
            [  
                "0x1234",  
                "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.



i PIO Account

Having [PIO Account](#) allows you to use extra professional features:

- [PIO Remote](#)
- [Integration with Cloud IDEs](#)

[Forgot Password?](#)

Need an Account? [Create a new one.](#)

1.14.2 CLI Guide

1.15 PIO Check

New in version 4.1.

Automated code analysis without hassle!

Static analysis became an important part of software development cycle. It can identify potential bugs, vulnerabilities and security threats by doing an analysis on the source code level without having to test it on hardware or execute any code.

PIO Check helps reduce development cost by enabling engineers to detect the precise location of defects and eliminate issues more efficiently and earlier in the development cycle. It can also ensure compliance with internal or industry coding standards such as MISRA, CERT, etc.

Key features:

- Fully integrated within the PlatformIO ecosystem and easy to execute on the entire project.
- Straightforward integration with *Continuous Integration* services.
- Possibility to reuse the same setup on other projects.
- Easy and flexible rule configuration.
- Comprehensive and detailed error information
- Multiple architectures and development platforms.
- Cross-platform: Windows, MacOS, Linux.

PIO Check can detect a wide range of known defects in C/C++ code, including:

- Potential NULL pointer dereferences
- Possible indexing beyond array bounds
- Suspicious assignments
- Reads of potentially uninitialized objects
- Unused variables or functions
- Out of scope memory usage

Warning: Before performing a static analysis check, make sure your project builds without errors. For information about how to build a project, see the [platformio run](#) command or [PlatformIO IDE for VSCode](#) guide.

Contents

- [Configuration](#)
- [Check tools](#)
- [Defect severity](#)
- [CLI Guide](#)

1.15.1 Configuration

PIO Check allows selecting what tool is used for finding defects in the project, what source files are checked. **PIO Check** can be configured from “[platformio.ini](#)” (*Project Configuration File*) using the next options:

1.15.2 Check tools

You can switch between or specify multiple tools used for finding defects using *check_tool* option:

```
[env:myenv]
platform = ...
board = ...
check_tool = cppcheck, clangtidy
```

Detailed information about supported check tools and their configuration process can be found on these pages:

Cppcheck

Cppcheck is a static analysis tool for C/C++ code. It provides a unique code analysis to detect bugs and focuses on detecting undefined behavior and dangerous coding constructs. The goal is to detect only real errors in the code (i.e. have very few false positives). More information about this tool on [the official webpage](#).

Hint: Cppcheck is rarely wrong about reported errors. But there are many bugs that it doesn't detect. You will find more bugs in your software by testing your software carefully than by using Cppcheck.

Contents

- [*Features*](#)
- [*Additional checks*](#)
- [*Configuration*](#)
- [*Extra flags*](#)
- [*Addons \(MISRA, CERT\)*](#)
 - [*MISRA*](#)
 - [*CERT*](#)

Features

Cppcheck supports a wide variety of static checks that may not be covered by the compiler itself. These checks are static analysis checks that can be performed at a source code level. The program is directed towards static analysis checks that are rigorous, rather than heuristic in nature.

Some of the defects that might be detected include:

- Automatic variable checking
- Bounds checking for array overruns
- Classes checking (e.g. unused functions, variable initialization, and memory duplication)
- Usage of deprecated or superseded functions
- Exception safety checking, for example, usage of memory allocation and destructor checks
- Memory leaks, e.g. due to lost scope without deallocation
- Resource leaks, e.g. due to forgetting to close a file handle
- Invalid usage of Standard Template Library functions and idioms
- Miscellaneous stylistic and performance errors

Additional checks

Be default **Cppcheck** is configured to check the next additional defects:

- warning
- style
- performance
- portability
- unusedFunction

The full list of supported check with detailed description is located on [the official webpage](#).

Configuration

Cppcheck is implicitly used as the default check tool when `check_tool` option in “`platformio.ini`” (*Project Configuration File*) is not set. To be explicit, you can specify it in the configuration directly:

```
[env:myenv]
platform = ...
board = ...
check_tool = cppcheck
check_flags = --enable=all
```

Useful options that can be used used for adjusting check process:

Extra flags

Useful flags that can help more precisely configure **Cppcheck** to satisfy your project requirements:

Flag	Meaning
--enable=<id>	Enable additional checks. The available ids are: all, warning, style, performance, portability, information, unusedFunction, missingInclude
--std=<id>	Set standard. The available options are: c89, c99, c11, c++03, c++11, c++14, c++17, c++20 (default)
--language=<language>	Forces Cppcheck to check all files as the given language. Valid values are: c, c++
--inline-suppr	Enable inline suppressions. Use them by placing one or more comments, like: // cppcheck-suppress warningId on the lines before the warning to suppress.
--suppress=<spec>	Suppress warnings that match <spec>. The format of <spec> is: [error id] : [filename] : [line]
--platform=<platform>	Specifies platform-specific types and sizes. The available built-in platforms are: unix32, unix64, win32A, win32W, win64, avr8, native, unspecified (default)
--inconclusive	Allow reporting defects even though the analysis is inconclusive.
-D<ID>	Define a preprocessor symbol. Example: -DDEBUG=1
-U<ID>	Undefine preprocessor symbol. Use -U to explicitly hide certain #ifdef <ID> code paths from checking. Example: -UDEBUG
-I <dir>	Give a path to search for include files. Give several -I parameters to give several paths.
-j <jobs>	Start <jobs> threads to do the checking simultaneously.

Addons (MISRA, CERT)

Cppcheck provides several addon scripts that analyze dump files to check compatibility with secure coding standards and to locate various issues. Most useful addons for verifying compliance with popular guidelines are **MISRA** and **CERT**.

MISRA

MISRA is a proprietary set of software development guidelines for the C/C++ programming languages developed by MISRA (Motor Industry Software Reliability Association). It aims to facilitate code safety, security, portability, and reliability in the context of embedded systems, specifically those systems programmed in ISO C/C++.

Note: Since this standard is proprietary, **Cppcheck** does not display error text by specifying only the number of violated rules (for example, [c2012-21.3]). If you want to display full texts for violated rules, you will need to create a text file containing MISRA rules, which you will have to pass when calling the script with `--rule-texts` flag.

In order to use MISRA addon you will need to provide a special file with the description of MISRA rules. Usually, it has the next contents:

```
Appendix A Summary of guidelines
Rule 3.1 Required
R3.1 Rule description
Rule 4.1 Required
...
Rule 21.3 Required
R21.3 Rule description
Rule 21.4
R21.4 Rule description
```

Next, you need to instruct **Cppcheck** that you want to run an additional addon script. Since this script requires an additional file with rules, you can pass it via a special `json` file:

```
{
  "script": "addons/misra.py",
  "args": ["--rule-texts=misra-rules.txt"]
}
```

Finally, add new flag to `check_flags`:

```
[env:myenv]
platform = ...
board = ...
check_tool = cppcheck
check_flags =
  cppcheck: --addon=misra.json
```

The full list of implemented MISRA checks can be found on [the official webpage](#).

CERT

SEI CERT coding standard provides rules for secure coding in the C programming language. The goal of these rules and recommendations is to develop safe, reliable, and secure systems, for example by eliminating undefined behaviors that can lead to undefined program behaviors and exploitable vulnerabilities.

In order to use the CERT addon, simply specify it as an additional flag in `check_flags` section:

```
[env:myenv]
platform = ...
board = ...
check_tool = cppcheck
check_flags =
    cppcheck: --addon=cert.py
```

Clang-Tidy

Clang-Tidy is a clang-based C++ “linter” tool. Its purpose is to provide an extensible framework for diagnosing and fixing typical programming errors, like style violations, interface misuse, or bugs that can be deduced via static analysis. Official page can be found [here](#).

Contents

- *Features*
- *Configuration*
- *Supported checks*
- *Extra flags*

Features

Clang-Tidy supports a large variety of static checks that may not be covered by the compiler itself. These checks are static analysis checks that can be performed at a source code level.

Some of the defects that might be detected include:

- Buffer overflow
- Potential NULL pointer dereferences
- Use of memory that has already been deallocated
- Out of scope memory usage
- Failure to set a return value from a subroutine

Configuration

To enable **Clang-Tidy** tool simply add it to the `check_tool` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:myenv]
platform = ...
board = ...
check_tool = clangtidy
```

Useful options that can be used for adjusting check process:

Supported checks

There are currently the following groups of most used checks (By default all checks are enabled):

Check	Description
abseil-	Checks related to Abseil library.
boost-	Checks related to Boost library.
bugprone-	Checks that target bugprone code constructs.
cert-	Checks related to CERT Secure Coding Guidelines.
cppcoreguidelines	Checks related to C++ Core Guidelines.
clang-analyzer-	Clang Static Analyzer checks.
google-	Checks related to Google coding conventions.
hicpp-	Checks related to High Integrity C++ Coding Standard.
modernize-	Checks that advocate usage of modern (currently modern means C++11) language constructs.
performance-	Checks that target performance-related issues.
portability-	Checks that target portability-related issues that don't relate to any particular coding style.
readability-	Checks that target readability-related issues that don't relate to any particular coding style.

The full list of supported checks can be found on [the official webpage](#).

Extra flags

Useful flags that can help more precisely configure **Clang-Tidy** to satisfy your project requirements:

Flag	Meaning
--checks=<string>	Comma-separated list of enabled checks (* default)
--fix	Apply suggested fixes. Without <code>-fix-errors</code> clang-tidy will bail out if any compilation errors were found.
--fix-errors	Apply suggested fixes even if compilation errors were found. If compiler errors have attached fix-its, clang-tidy will apply them as well.
--format-style=<style>	Style for formatting code around applied fixes: llvm, google, webkit, mozilla, none (default)
--system-headers	Display the errors from system headers.

An example with enabling specific checks and fixing code on the fly:

```
[env:myenv]
platform = ...
board = ...
check_tool = clangtidy
check_flags =
    clangtidy: --check=*,cert-*,clang-analyzer-* --fix
```

1.15.3 Defect severity

Defect severity is a classification of software defect (bug, vulnerability, etc) that indicates the degree of negative impact on the quality of software. **PIO Check** uses the next classification of possible defects:

Severity	Meaning
high	Issues that are possibly bugs
medium	Suggestions about defensive programming in order to prevent potential bugs
low	Issues related to code cleanup and performance (unused functions, redundant code, const-ness, etc)

1.15.4 CLI Guide

PIO Check can be configured using command line commands. Detailed description of these commands can be found here:

platformio check

Helper command for *PIO Check*.

Contents

- *platformio check*
 - *Usage*
 - *Description*
 - *Options*
 - *Examples*

Usage

```
platformio check [OPTIONS]
pio check [OPTIONS]
```

Description

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

More details about PlatformIO *PIO Check*.

Options

-e, --environment

Process specified environments.

--filter

Normally a program has many source files. By default only *src_dir* and *include_dir* are checked. You can specify which source files should be included/excluded from check process. The paths in filter should be **relative to the root** of a project.

Multiple **--filter** options are allowed.

Example: `platformio check --filter "-<*> +<src/> +<tests/>"`

--flags

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

Flag	Meaning
<code>--addon=<addon></code>	Execute addon. i.e. cert.
<code>-D<ID></code>	Define preprocessor symbol.

Multiple `--flags` options are allowed.

Example: `platformio check --flags "-DDEBUG cppcheck: --std=c++11 --platform=avr8"`

--severity

Specify the [Defect severity](#) types which will be reported by the [Check tools](#).

Multiple `--severity` options are allowed.

-d, --project-dir

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

-c, --project-conf

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

--json-output

Return the output in [JSON](#) format.

-s, --silent

Suppress progress reporting and show only defects with high severity. See [Defect severity](#).

-v, --verbose

Show detailed information when processing environments.

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

Examples

For the examples please follow to [PIO Check](#) page.

1.16 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**.

Contents

- [PIO Remote](#)
 - [Features](#)
 - [Use Cases](#)
 - [Technology](#)
 - [Installation](#)
 - [Quick Start](#)
 - [CLI Guide](#)

1.16.1 Features

- [Remote Device Manager](#)
- [Remote Serial Port Monitor](#)
- [Remote Firmware Updates](#)
- **PIO Remote Share**
- Continuous Deployment
- Continuous Delivery
- Remote Unit Testing

1.16.2 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?

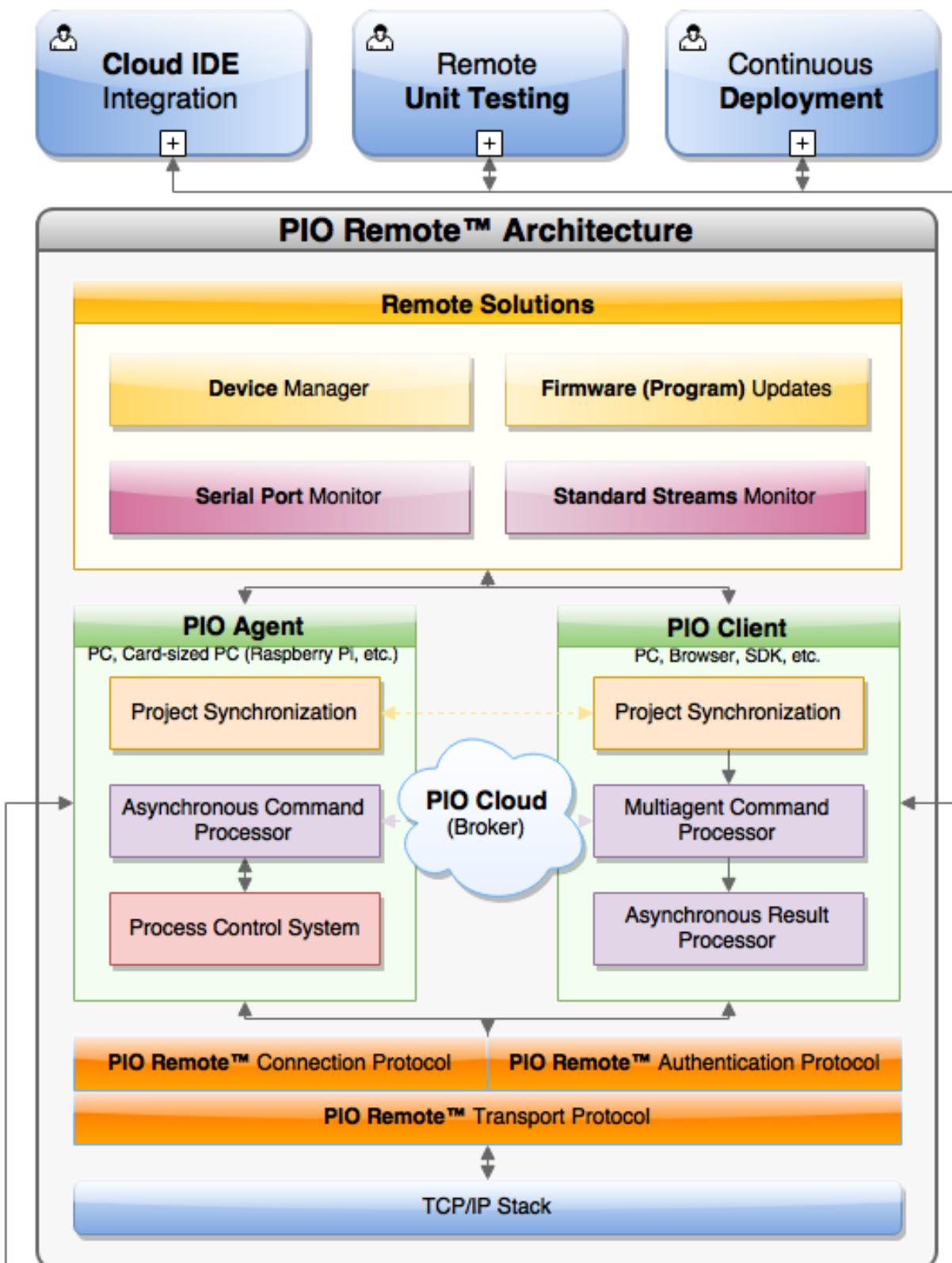
- You commit new changes to your source code repository
- [Continuous Integration](#) service deploys unit tests to a remote agent
- [PIO Unit Testing](#) engine runs tests on a physical device, process them, and send results
- [Continuous Integration](#) service prints results in human readable format
- 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.

1.16.3 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](#).

1.16.4 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 \(CLI\)](#).

1.16.5 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`.

1.16.6 CLI Guide

1.17 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

The screenshot shows the PlatformIO Debugger interface with the following components:

- Left Sidebar:** Includes sections for VARIABLES (Local, Global, Static), WATCH (mac: [6], strM2MAPConfig.au8DHCPServerIP[0..3]), CALL STACK (Paused on WiFiClass::startProvision@...), BREAKPOINTS (WiFi.cpp .piolib..., checked), PERIPHERALS (PORT [0x41000400], RTC [0x40001400], SERCOM0 [0x42000080], I2CM [0x0], I2CS [0x0], SPI [0x0]), and REGISTERS (pc = 0x00000726, xPSR = 0x21000000).
- Middle Left:** The `WiFi.cpp` file content, showing code for provisioning mode.
- Middle Right:** The `Memory [0x42000800+1]` view showing memory dump.
- Right Side:** The `WiFiClass::startProvision.dbgasm` view showing assembly code for the provisioning logic.
- Bottom:** A terminal window displaying the output of the task: `platformio device monitor`. The output includes the Miniterm configuration and the start of provisioning mode.

Contents

- *Tutorials*
- *Configuration*
- *Tools & Debug Probes*
- *CLI Guide*
- *Platforms*
- *Frameworks*
- *Boards*

1.17.1 Tutorials

- Arduino In-circuit Debugging with PlatformIO
- 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*

1.17.2 Configuration

PIO Unified Debugger can be configured from “*platformio.ini*” (*Project Configuration File*):

1.17.3 Tools & Debug Probes

You can switch between debugging tools using *debug_tool* option.

Warning: You will need to install debug tool drivers depending on your operating system. Please check “Drivers” section for debugging tool below.

Altera / Intel USB-Blaster Download Cable



USB Blaster Download Cable is designed for ALTERA FPGA, CPLD, Active Serial Configuration Devices and Enhanced Configuration Devices, USB 2.0 connection to the PC and JTAG, AS, PS to the target device. Official reference can be found [here](#).

Contents

- *Configuration*
- *Drivers*
- *Wiring Connections*

- *JTAG Interface*
- *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 = altera-usb-blaster
```

If you would like to use this tool for firmware uploading, please change upload protocol:

```
[env:myenv]  
platform = ...  
board = ...  
debug_tool = altera-usb-blaster  
upload_protocol = altera-usb-blaster
```

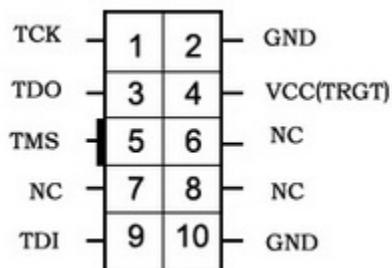
More options:

- *Debugging options*
- *Upload options*

Drivers

Please install official drivers.

Wiring Connections



JTAG Interface

USB-Blaster JTAG 10-Pin Connector	Board JTAG Pin	Description
1	TCK	JTAG Return Test Clock
2	GND	Digital ground
3	TDO	Test Data Out pin
4	VCC	Positive Supply Voltage — Power supply for JTAG interface drivers
5	TMS	Test Mode State pin
9	TDI	Test Data In pin

Platforms

Name	Description
GigaDevice GD32V	The GigaDevice GD32V device is a 32-bit general-purpose microcontroller based on the RISC-V core with an impressive balance of processing power, reduced power consumption and peripheral set.

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.
GigaDevice GD32V SDK	GigaDevice GD32VF103 Firmware Library (SDK)

Boards

Note: For more detailed board information please scroll tables below by horizontal.

Name	Platform	Debug	MCU	Frequency	Flash	RAM
GD32VF103V-EVAL	GigaDevice GD32V	External	GD32VF103VBT6	108MHz	128KB	32KB
Sipeed Longan Nano	GigaDevice GD32V	External	GD32VF103CBT6	108MHz	128KB	32KB
Wio Lite RISC-V	GigaDevice GD32V	External	GD32VF103CBT6	108MHz	128KB	32KB

Atmel-ICE



Atmel-ICE is a powerful development tool for debugging and programming ARM® Cortex®-M based SAM and AVR microcontrollers with on-chip debug capability. Official reference can be found [here](#).

Contents

- *Configuration*
- *Drivers*
- *Platforms*
- *Frameworks*
- *Boards*

Configuration

You can configure debugging tool using `debug_tool` option in “`platformio.ini`” (Project Configuration File):

```
[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

Name	Description
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.

Frameworks

Name	Description
Ar-duino	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
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.
Simba	Simba is an RTOS and build framework. It aims to make embedded programming easy and portable.

Boards

Note: For more detailed board information please scroll tables below by horizontal.

Name	Platform	Debug	MCU	Frequency	Flash	RAM
Adafruit Circuit Playground Express	Atmel SAM	External	SAMD21G18A	48MHz	256KB	32KB
Adafruit Crickit M0	Atmel SAM	External	SAMD21G18A	48MHz	256KB	32KB
Adafruit Feather M0	Atmel SAM	External	SAMD21G18A	48MHz	256KB	32KB
Adafruit Feather M0 Express	Atmel SAM	External	SAMD21G18A	48MHz	256KB	32KB
Adafruit Feather M4 Express	Atmel SAM	External	SAMD51J19A	120MHz	512KB	192KB
Adafruit Gemma M0	Atmel SAM	External	SAMD21E18A	48MHz	256KB	32KB
Adafruit Grand Central M4	Atmel SAM	External	SAMD51P20A	120MHz	1MB	256KB
Adafruit Hallowing M0	Atmel SAM	External	SAMD21G18A	48MHz	256KB	32KB
Adafruit Hallowing M4	Atmel SAM	External	SAMD51J19A	120MHz	496KB	192KB
Adafruit ItsyBitsy M0	Atmel SAM	External	SAMD21G18A	48MHz	256KB	32KB
Adafruit ItsyBitsy M4	Atmel SAM	External	SAMD51G19A	120MHz	512KB	192KB
Adafruit MONSTER M4SK	Atmel SAM	External	SAMD51J19A	120MHz	496KB	192KB

Continued on next page

Table 27 – continued from previous page

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Adafruit Metro M0 Expresss</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz	256KB	32KB
<i>Adafruit Metro M4</i>	<i>Atmel SAM</i>	External	SAMD51J19A	120MHz	512KB	192KB
<i>Adafruit Metro M4 AirLift Lite</i>	<i>Atmel SAM</i>	External	SAMD51J19A	120MHz	512KB	192KB
<i>Adafruit PyGamer Advance M4</i>	<i>Atmel SAM</i>	External	SAMD51J20A	120MHz	1MB	256KB
<i>Adafruit PyGamer M4 Express</i>	<i>Atmel SAM</i>	External	SAMD51J19A	120MHz	512KB	192KB
<i>Adafruit PyPortal M4</i>	<i>Atmel SAM</i>	External	SAMD51J20A	120MHz	1MB	256KB
<i>Adafruit Trellis M4</i>	<i>Atmel SAM</i>	External	SAMD51J19A	120MHz	512KB	192KB
<i>Adafruit Trinket M0</i>	<i>Atmel SAM</i>	External	SAMD21E18A	48MHz	256KB	32KB
<i>Adafruit pIRkey</i>	<i>Atmel SAM</i>	External	SAMD21E18A	48MHz	256KB	32KB
<i>Adafruit pyBadge AirLift M4</i>	<i>Atmel SAM</i>	External	SAMD51J20A	120MHz	1008KB	192KB
<i>Adafruit pyBadge M4 Express</i>	<i>Atmel SAM</i>	External	SAMD51J19A	120MHz	512KB	192KB
<i>Arduino Due (Programming Port)</i>	<i>Atmel SAM</i>	External	AT91SAM3X8E	84MHz	512KB	96KB
<i>Arduino Due (USB Native Port)</i>	<i>Atmel SAM</i>	External	AT91SAM3X8E	84MHz	512KB	96KB
<i>Arduino M0</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz	256KB	32KB
<i>Arduino M0 Pro (Native USB Port)</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz	256KB	32KB
<i>Arduino M0 Pro (Programming/Debug Port)</i>	<i>Atmel SAM</i>	On-board	SAMD21G18A	48MHz	256KB	32KB
<i>Arduino MKR FOX 1200</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz	256KB	32KB
<i>Arduino MKR GSM 1400</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz	256KB	32KB
<i>Arduino MKR NB 1500</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz	256KB	32KB
<i>Arduino MKR WAN 1300</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz	256KB	32KB
<i>Arduino MKR WiFi 1010</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz	256KB	32KB
<i>Arduino MKR1000</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz	256KB	32KB
<i>Arduino MKRZERO</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz	256KB	32KB
<i>Arduino Tian</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz	256KB	32KB
<i>Arduino Zero (Programming/Debug Port)</i>	<i>Atmel SAM</i>	On-board	SAMD21G18A	48MHz	256KB	32KB
<i>Arduino Zero (USB Native Port)</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz	256KB	32KB
<i>Atmel ATSAMR21-XPRO</i>	<i>Atmel SAM</i>	On-board	SAMR21G18A	48MHz	256KB	32KB
<i>Atmel ATSAMW25-XPRO</i>	<i>Atmel SAM</i>	On-board	SAMD21G18A	48MHz	256KB	32KB
<i>Atmel SAMD21-XPRO</i>	<i>Atmel SAM</i>	On-board	SAMD21J18A	48MHz	256KB	32KB
<i>Atmel SAML21-XPRO-B</i>	<i>Atmel SAM</i>	On-board	SAML21J18B	48MHz	256KB	32KB
<i>Digistump DigiX</i>	<i>Atmel SAM</i>	External	AT91SAM3X8E	84MHz	512KB	96KB
<i>MKR Vidor 4000</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz	256KB	32KB
<i>Macchina M2</i>	<i>Atmel SAM</i>	External	AT91SAM3X8E	84MHz	512KB	96KB
<i>Minitronics v2.0</i>	<i>Atmel SAM</i>	External	SAMD21J18A	48MHz	256KB	32KB
<i>Moteino M0</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz	256KB	32KB
<i>NANO 33 IoT</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz	256KB	32KB
<i>SODAQ Autonomo</i>	<i>Atmel SAM</i>	External	SAMD21J18A	48MHz	256KB	32KB
<i>SODAQ ExpLoRer</i>	<i>Atmel SAM</i>	External	SAMD21J18A	48MHz	256KB	32KB
<i>SODAQ ONE</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz	256KB	32KB
<i>SODAQ SARA</i>	<i>Atmel SAM</i>	External	SAMD21J18A	48MHz	256KB	32KB
<i>SODAQ SFF</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz	256KB	32KB
<i>SainSmart Due (Programming Port)</i>	<i>Atmel SAM</i>	External	AT91SAM3X8E	84MHz	512KB	96KB
<i>SainSmart Due (USB Native Port)</i>	<i>Atmel SAM</i>	External	AT91SAM3X8E	84MHz	512KB	96KB
<i>Seeeduino LoRaWAN</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz	256KB	32KB
<i>SparkFun SAMD21 Dev Breakout</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz	256KB	32KB
<i>SparkFun SAMD21 Mini Breakout</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz	256KB	32KB
<i>Tuino 096</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz	256KB	32KB

Black Magic Probe



The Black Magic Probe is a modern, in-application debugging tool for embedded microprocessors. It is able to control and examine the state of the target microprocessor using a JTAG or Serial Wire Debugging (SWD) port and on-chip debug logic provided by the microprocessor. The probe connects to a host computer using a standard USB interface. Official reference can be found [here](#).

Also, see [Custom](#) debugging configuration with Black Magic Probe.

Contents

- [Configuration](#)
- [Drivers](#)
- [Wiring Connections](#)
 - [JTAG Interface](#)
 - [Serial Wire Mode Interface \(SWD\)](#)
- [Platforms](#)
- [Frameworks](#)
- [Boards](#)

Configuration

You can configure debugging tool using `debug_tool` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:myenv]
platform = ...
board = ...
debug_tool = blackmagic
debug_port = <CONFIGURE GDB PORT>

;
; Debug Port Examples
;

; Linux
debug_port = /dev/ttyACM0
```

(continues on next page)

(continued from previous page)

```

; Windows for COM1-COM9
debug_port = COM3
; Windows for COM10-XXX
debug_port = \\.\COM13

; macOS
debug_port = /dev/cu.usbmodemE2C0C4C6

```

Black Magic Probe has 2 serial ports: UART and GDB. We will need “GDB” port. Please use [PlatformIO Home > Devices](#) or [PlatformIO Core \(CLI\)](#) and `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:

```

[env:myenv]
platform = ...
board = ...
debug_tool = blackmagic
debug_port = <CONFIGURE GDB PORT>

upload_port = <THE SAME AS DEBUG PORT>

; SWD interface
upload_protocol = blackmagic

; JTAG interface
upload_protocol = blackmagic-jtag

```

More options:

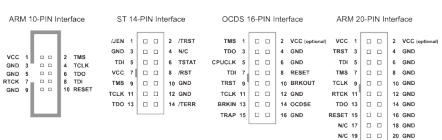
- *Debugging options*
- *Upload options*

Drivers

Not required.

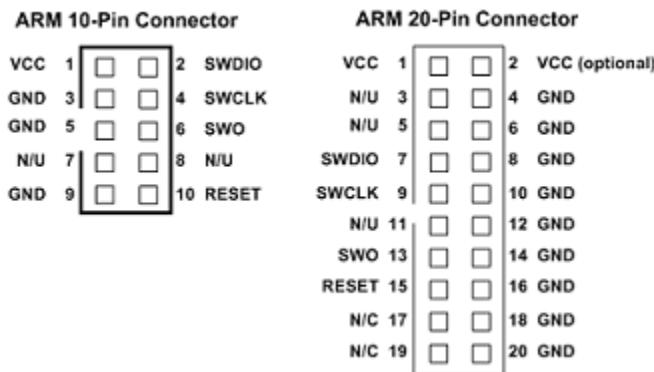
Wiring Connections

JTAG Interface



Black Magic Probe 10-Pin Connector	Board JTAG Pin	Description
1	VCC	Positive Supply Voltage — Power supply for JTAG interface drivers
3	GND	Digital ground
2	TMS	Test Mode State
4	TCLK	JTAG Return Test Clock
6	TDO	Test Data Out
8	TDI	Test Data In
10	RESET	Connect this pin to the (active low) reset input of the target CPU

Serial Wire Mode Interface (SWD)



Black Magic Probe 10-Pin Connector	Board SWD Pin	Description
1	VCC	Positive Supply Voltage — Power supply for JTAG interface drivers
3	GND	Digital ground
2	SWDIO	Data I/O
4	SWCLK	Clock
10	RESET	Connect this pin to the (active low) reset input of the target CPU

Platforms

Name	Description
<i>Aceinna IMU</i>	Open-source, embedded development platform for Aceinna IMU hardware. Run custom algorithms and navigation code on Aceinna IMU/INS hardware.
<i>Atmel SAM</i>	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.
<i>Freescale Kinetis</i>	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.
<i>Nordic nRF51</i>	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.
<i>Nordic nRF52</i>	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.
<i>NXP LPC</i>	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.
<i>Silicon Labs EFM32</i>	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.
<i>ST STM32</i>	The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.

Frameworks

Name	Description
Arduino	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
CMSIS	The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.
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.
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.
Simba	Simba is an RTOS and build framework. It aims to make embedded programming easy and portable.
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.
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.

Boards

Note: For more detailed board information please scroll tables below by horizontal.

Name	Platform	Debug	MCU	Freq
1Bitisy	ST STM32	External	STM32F415RGT	168MHz
32F723EDISCOVERY	ST STM32	On-board	STM32F723IEK6	216MHz
3D Printer Controller	ST STM32	External	STM32F407VET6	168MHz
3D Printer control board	ST STM32	External	STM32F446RET6	180MHz
3D printer controller	ST STM32	On-board	STM32F765VIT6	216MHz
3DP001V1 Evaluation board for 3D printer	ST STM32	On-board	STM32F401VGT6	84MHz
96Boards B96B-F446VE	ST STM32	On-board	STM32F446VET6	168MHz
ARM mbed LPC11U24 (+CAN)	NXP LPC	On-board	LPC11U24	48MHz
Aceinna Low Cost RTK	Aceinna IMU	On-board	STM32F469NIH6	180MHz
Aceinna OpenIMU 300ZA	Aceinna IMU	External	STM32F405RG	120MHz
Aceinna OpenIMU 300ZA	Aceinna IMU	External	STM32F405RG	120MHz
Aceinna OpenIMU 330	Aceinna IMU	External	STM32L431CB	80MHz
Adafruit Circuit Playground Express	Atmel SAM	External	SAMD21G18A	48MHz
Adafruit Crickit M0	Atmel SAM	External	SAMD21G18A	48MHz
Adafruit Feather M0	Atmel SAM	External	SAMD21G18A	48MHz
Adafruit Feather M0 Express	Atmel SAM	External	SAMD21G18A	48MHz

Table 28 – continued from previous page

Name	Platform	Debug	MCU	Freq
<i>Adafruit Gemma M0</i>	<i>Atmel SAM</i>	External	SAMD21E18A	48MHz
<i>Adafruit Hallowing M0</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz
<i>Adafruit ItsyBitsy M0</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz
<i>Adafruit Metro M0 Expresss</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz
<i>Adafruit Trinket M0</i>	<i>Atmel SAM</i>	External	SAMD21E18A	48MHz
<i>Adafruit pIRkey</i>	<i>Atmel SAM</i>	External	SAMD21E18A	48MHz
<i>Arduino Due (Programming Port)</i>	<i>Atmel SAM</i>	External	AT91SAM3X8E	84MHz
<i>Arduino Due (USB Native Port)</i>	<i>Atmel SAM</i>	External	AT91SAM3X8E	84MHz
<i>Arduino M0</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz
<i>Arduino M0 Pro (Native USB Port)</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz
<i>Arduino M0 Pro (Programming/Debug Port)</i>	<i>Atmel SAM</i>	On-board	SAMD21G18A	48MHz
<i>Arduino MKR FOX 1200</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz
<i>Arduino MKR GSM 1400</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz
<i>Arduino MKR NB 1500</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz
<i>Arduino MKR WAN 1300</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz
<i>Arduino MKR WiFi 1010</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz
<i>Arduino MKR1000</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz
<i>Arduino MKRZERO</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz
<i>Arduino Tian</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz
<i>Arduino Zero (Programming/Debug Port)</i>	<i>Atmel SAM</i>	On-board	SAMD21G18A	48MHz
<i>Arduino Zero (USB Native Port)</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz
<i>Armstrap Eagle 1024</i>	<i>ST STM32</i>	External	STM32F417VGT6	168MHz
<i>Armstrap Eagle 2048</i>	<i>ST STM32</i>	External	STM32F427VIT6	168MHz
<i>Armstrap Eagle 512</i>	<i>ST STM32</i>	External	STM32F407VET6	168MHz
<i>Atmel ATSAMR21-XPRO</i>	<i>Atmel SAM</i>	On-board	SAMR21G18A	48MHz
<i>Atmel ATSAMW25-XPRO</i>	<i>Atmel SAM</i>	On-board	SAMD21G18A	48MHz
<i>Atmel SAMD21-XPRO</i>	<i>Atmel SAM</i>	On-board	SAMD21J18A	48MHz
<i>Atmel SAML21-XPRO-B</i>	<i>Atmel SAM</i>	On-board	SAML21J18B	48MHz
<i>Bambino-210E</i>	<i>NXP LPC</i>	On-board	LPC4330	204MHz
<i>Black STM32F407VE</i>	<i>ST STM32</i>	External	STM32F407VET6	168MHz
<i>Black STM32F407VG</i>	<i>ST STM32</i>	External	STM32F407VGT6	168MHz
<i>Black STM32F407ZE</i>	<i>ST STM32</i>	External	STM32F407ZET6	168MHz
<i>Black STM32F407ZE</i>	<i>ST STM32</i>	External	STM32F407ZGT6	168MHz
<i>BlackPill F103C8</i>	<i>ST STM32</i>	External	STM32F103C8T6	72MHz
<i>BlackPill F103C8 (128k)</i>	<i>ST STM32</i>	External	STM32F103C8T6	72MHz
<i>BlackPill F303CC</i>	<i>ST STM32</i>	External	STM32F303CCT6	72MHz
<i>Blue STM32F407VE Mini</i>	<i>ST STM32</i>	External	STM32F407VET6	168MHz
<i>BluePill F103C6</i>	<i>ST STM32</i>	External	STM32F103C6T6	72MHz
<i>BluePill F103C8</i>	<i>ST STM32</i>	External	STM32F103C8T6	72MHz
<i>BluePill F103C8 (128k)</i>	<i>ST STM32</i>	External	STM32F103C8T6	72MHz
<i>Bluey nRF52832 IoT</i>	<i>Nordic nRF52</i>	External	NRF52832	64MHz
<i>BluzDK</i>	<i>Nordic nRF51</i>	External	NRF51822	32MHz
<i>CQ Publishing TG-LPC11U35-501</i>	<i>NXP LPC</i>	External	LPC11U35	48MHz
<i>CoCo-ri-Co!</i>	<i>NXP LPC</i>	On-board	LPC812	30MHz
<i>Delta DFBM-NQ620</i>	<i>Nordic nRF52</i>	On-board	NRF52832	64MHz
<i>Demo F030F4</i>	<i>ST STM32</i>	External	STM32F030F4P6	48MHz
<i>Digistump DigiX</i>	<i>Atmel SAM</i>	External	AT91SAM3X8E	84MHz
<i>EA LPC11U35 QuickStart Board</i>	<i>NXP LPC</i>	External	LPC11U35	48MHz
<i>EFM32GG-STK3700 Giant Gecko</i>	<i>Silicon Labs EFM32</i>	On-board	EFM32GG990F1024	48MHz

Table 28 – continued from previous page

Name	Platform	Debug	MCU	Freq
<i>EFM32LG-STK3600 Leopard Gecko</i>	<i>Silicon Labs EFM32</i>	On-board	EFM32LG990F256	48MHz
<i>EFM32WG-STK3800 Wonder Gecko</i>	<i>Silicon Labs EFM32</i>	On-board	EFM32WG990F256	48MHz
<i>EFM32ZG-STK3200 Zero Gecko</i>	<i>Silicon Labs EFM32</i>	On-board	EFM32ZG222F32	24MHz
<i>Espotel LoRa Module</i>	<i>ST STM32</i>	External	STM32F411RET6	100MHz
<i>F407VG</i>	<i>ST STM32</i>	External	STM32F407VGT6	168MHz
<i>FK407M1</i>	<i>ST STM32</i>	External	STM32F407VET6	168MHz
<i>Freescale Kinetis FRDM-KL25Z</i>	<i>Freescale Kinetis</i>	On-board	MKL25Z128VLK4	48MHz
<i>Freescale Kinetis FRDM-KL27Z</i>	<i>Freescale Kinetis</i>	On-board	MKL27Z64VLH4	48MHz
<i>L476DMWIK</i>	<i>ST STM32</i>	On-board	STM32L476VGT6	80MHz
<i>LPCXpresso11U68</i>	<i>NXP LPC</i>	On-board	LPC11U68	50MHz
<i>LPCXpresso824-MAX</i>	<i>NXP LPC</i>	On-board	LPC824	30MHz
<i>M200 V2</i>	<i>ST STM32</i>	External	STM32F070CBT6	48MHz
<i>MKR Vidor 4000</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz
<i>MTS Dragonfly</i>	<i>ST STM32</i>	External	STM32F411RET6	100MHz
<i>Macchina M2</i>	<i>Atmel SAM</i>	External	AT91SAM3X8E	84MHz
<i>Malyan M200 V1</i>	<i>ST STM32</i>	External	STM32F103CBT6	72MHz
<i>Maple</i>	<i>ST STM32</i>	External	STM32F103RBT6	72MHz
<i>Maple (RET6)</i>	<i>ST STM32</i>	External	STM32F103RET6	72MHz
<i>Maple Mini Bootloader 2.0</i>	<i>ST STM32</i>	External	STM32F103CBT6	72MHz
<i>Maple Mini Original</i>	<i>ST STM32</i>	External	STM32F103CBT6	72MHz
<i>Mbed Connect Cloud</i>	<i>ST STM32</i>	On-board	STM32F439ZIY6	168MHz
<i>Microduino Core STM32 to Flash</i>	<i>ST STM32</i>	External	STM32F103CBT6	72MHz
<i>Microsoft Azure IoT Development Kit (MXChip AZ3166)</i>	<i>ST STM32</i>	On-board	STM32F412ZGT6	100MHz
<i>Minitronics v2.0</i>	<i>Atmel SAM</i>	External	SAMD21J18A	48MHz
<i>Moteino M0</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz
<i>MultiTech mDot</i>	<i>ST STM32</i>	External	STM32F411RET6	100MHz
<i>MultiTech mDot F411</i>	<i>ST STM32</i>	External	STM32F411RET6	100MHz
<i>MultiTech xDot</i>	<i>ST STM32</i>	External	STM32L151CCU6	32MHz
<i>NAMote72</i>	<i>ST STM32</i>	External	STM32L152RC	32MHz
<i>NANO 33 IoT</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz
<i>NGX Technologies BlueBoard-LPC11U24</i>	<i>NXP LPC</i>	External	LPC11U24	48MHz
<i>NXP LPC11C24</i>	<i>NXP LPC</i>	External	LPC11C24	48MHz
<i>NXP LPC11U34</i>	<i>NXP LPC</i>	External	LPC11U34	48MHz
<i>NXP LPC11U37</i>	<i>NXP LPC</i>	External	LPC11U37	48MHz
<i>NXP LPC800-MAX</i>	<i>NXP LPC</i>	On-board	LPC812	30MHz
<i>NXP LPCXpresso1549</i>	<i>NXP LPC</i>	External	LPC1549	72MHz
<i>NXP mbed LPC11U24</i>	<i>NXP LPC</i>	On-board	LPC11U24	48MHz
<i>NXP mbed LPC1768</i>	<i>NXP LPC</i>	On-board	LPC1768	96MHz
<i>Nordic Beacon Kit (PCA20006)</i>	<i>Nordic nRF51</i>	On-board	NRF51822	32MHz
<i>Nordic nRF51X22 Development Kit(PCA1000X)</i>	<i>Nordic nRF51</i>	On-board	NRF51822	32MHz
<i>Nordic nRF52-DK</i>	<i>Nordic nRF52</i>	On-board	NRF52832	64MHz
<i>Nordic nRF52840-DK</i>	<i>Nordic nRF52</i>	On-board	NRF52840	64MHz
<i>Nucleo G071RB</i>	<i>ST STM32</i>	On-board	STM32G071RBT6	24MHz
<i>OSHChip</i>	<i>Nordic nRF51</i>	External	NRF51822	32MHz
<i>P-Nucleo WB55RG</i>	<i>ST STM32</i>	On-board	STM32WB55RG	64MHz
<i>RAK811 LoRa Tracker</i>	<i>ST STM32</i>	External	STM32L151RBT6	32MHz
<i>RAK811 LoRa Tracker</i>	<i>ST STM32</i>	External	STM32L151RBT6	32MHz
<i>RedBearLab BLE Nano 1.5</i>	<i>Nordic nRF51</i>	On-board	NRF51822	16MHz
<i>RedBearLab BLE Nano 2</i>	<i>Nordic nRF52</i>	On-board	NRF52832	64MHz

Table 28 – continued from previous page

Name	Platform	Debug	MCU	Freq
<i>RedBearLab Blend 2</i>	<i>Nordic nRF52</i>	On-board	NRF52832	64MHz
<i>RedBearLab nRF51822</i>	<i>Nordic nRF51</i>	On-board	NRF51822	16MHz
<i>RushUp Cloud-JAM</i>	<i>ST STM32</i>	On-board	STM32F401RET6	84MHz
<i>RushUp Cloud-JAM L4</i>	<i>ST STM32</i>	On-board	STM32L476RGT6	80MHz
<i>SDT52832B</i>	<i>Nordic nRF52</i>	External	NRF52832	64MHz
<i>SLSTK3400A USB-enabled Happy Gecko</i>	<i>Silicon Labs EFM32</i>	On-board	EFM32HG322F64	25MHz
<i>SLSTK3401A Pearl Gecko PG1</i>	<i>Silicon Labs EFM32</i>	On-board	EFM32PG1B200F256GM48	40MHz
<i>SODAQ Autonomo</i>	<i>Atmel SAM</i>	External	SAMD21J18A	48MHz
<i>SODAQ ExpLoRer</i>	<i>Atmel SAM</i>	External	SAMD21J18A	48MHz
<i>SODAQ ONE</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz
<i>SODAQ SARA</i>	<i>Atmel SAM</i>	External	SAMD21J18A	48MHz
<i>SODAQ SFF</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz
<i>ST 32F3348DISCOVERY</i>	<i>ST STM32</i>	On-board	STM32F334C8T6	72MHz
<i>ST 32F401CDISCOVERY</i>	<i>ST STM32</i>	On-board	STM32F401VCT6	84MHz
<i>ST 32F411EDISCOVERY</i>	<i>ST STM32</i>	On-board	STM32F411VET6	100MHz
<i>ST 32F413HDISCOVERY</i>	<i>ST STM32</i>	On-board	STM32F413ZHT6	100MHz
<i>ST 32F429IDISCOVERY</i>	<i>ST STM32</i>	On-board	STM32F429ZIT6	180MHz
<i>ST 32F469IDISCOVERY</i>	<i>ST STM32</i>	On-board	STM32F469NIH6	180MHz
<i>ST 32F746GDISCOVERY</i>	<i>ST STM32</i>	On-board	STM32F746NGH6	216MHz
<i>ST 32F769IDISCOVERY</i>	<i>ST STM32</i>	On-board	STM32F769NIH6	216MHz
<i>ST 32L0538DISCOVERY</i>	<i>ST STM32</i>	On-board	STM32L053C8T6	32MHz
<i>ST 32L100DISCOVERY</i>	<i>ST STM32</i>	On-board	STM32L100RCT6	32MHz
<i>ST 32L476GDISCOVERY</i>	<i>ST STM32</i>	On-board	STM32L476VGT6	80MHz
<i>ST 32L496GDISCOVERY</i>	<i>ST STM32</i>	On-board	STM32L496AGI6	80MHz
<i>ST DISCO-L072CZ-LRWAN1</i>	<i>ST STM32</i>	On-board	STM32L072CZ	32MHz
<i>ST DISCO-L475VG-IOT01A</i>	<i>ST STM32</i>	On-board	STM32L475VGT6	80MHz
<i>ST Discovery F072RB</i>	<i>ST STM32</i>	On-board	STM32F072RBT6	48MHz
<i>ST Nucleo F030R8</i>	<i>ST STM32</i>	On-board	STM32F030R8T6	48MHz
<i>ST Nucleo F031K6</i>	<i>ST STM32</i>	On-board	STM32F031K6T6	48MHz
<i>ST Nucleo F042K6</i>	<i>ST STM32</i>	On-board	STM32F042K6T6	48MHz
<i>ST Nucleo F070RB</i>	<i>ST STM32</i>	On-board	STM32F070RBT6	48MHz
<i>ST Nucleo F072RB</i>	<i>ST STM32</i>	On-board	STM32F072RBT6	48MHz
<i>ST Nucleo F091RC</i>	<i>ST STM32</i>	On-board	STM32F091RCT6	48MHz
<i>ST Nucleo F103RB</i>	<i>ST STM32</i>	On-board	STM32F103RBT6	72MHz
<i>ST Nucleo F207ZG</i>	<i>ST STM32</i>	On-board	STM32F207ZGT6	120MHz
<i>ST Nucleo F302R8</i>	<i>ST STM32</i>	On-board	STM32F302R8T6	72MHz
<i>ST Nucleo F303K8</i>	<i>ST STM32</i>	On-board	STM32F303K8T6	72MHz
<i>ST Nucleo F303RE</i>	<i>ST STM32</i>	On-board	STM32F303RET6	72MHz
<i>ST Nucleo F303ZE</i>	<i>ST STM32</i>	On-board	STM32F303ZET6	72MHz
<i>ST Nucleo F334R8</i>	<i>ST STM32</i>	On-board	STM32F334R8T6	72MHz
<i>ST Nucleo F401RE</i>	<i>ST STM32</i>	On-board	STM32F401RET6	84MHz
<i>ST Nucleo F410RB</i>	<i>ST STM32</i>	On-board	STM32F410RBT6	100MHz
<i>ST Nucleo F411RE</i>	<i>ST STM32</i>	On-board	STM32F411RET6	100MHz
<i>ST Nucleo F412ZG</i>	<i>ST STM32</i>	On-board	STM32F412ZGT6	100MHz
<i>ST Nucleo F413ZH</i>	<i>ST STM32</i>	On-board	STM32F413ZHT6	100MHz
<i>ST Nucleo F429ZI</i>	<i>ST STM32</i>	On-board	STM32F429ZIT6	180MHz
<i>ST Nucleo F439ZI</i>	<i>ST STM32</i>	On-board	STM32F439ZIT6	180MHz
<i>ST Nucleo F446RE</i>	<i>ST STM32</i>	On-board	STM32F446RET6	180MHz
<i>ST Nucleo F446ZE</i>	<i>ST STM32</i>	On-board	STM32F446ZET6	180MHz

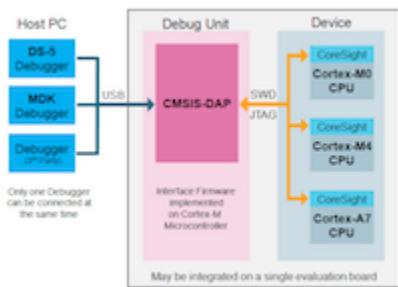
Table 28 – continued from previous page

Name	Platform	Debug	MCU	Freq
<i>ST Nucleo F722ZE</i>	<i>ST STM32</i>	On-board	STM32F722ZET6	216MHz
<i>ST Nucleo F746ZG</i>	<i>ST STM32</i>	On-board	STM32F746ZGT6	216MHz
<i>ST Nucleo F756ZG</i>	<i>ST STM32</i>	On-board	STM32F756ZG	216MHz
<i>ST Nucleo F767ZI</i>	<i>ST STM32</i>	On-board	STM32F767ZIT6	216MHz
<i>ST Nucleo H743ZI</i>	<i>ST STM32</i>	On-board	STM32H743ZIT6	400MHz
<i>ST Nucleo L011K4</i>	<i>ST STM32</i>	On-board	STM32L011K4T6	32MHz
<i>ST Nucleo L031K6</i>	<i>ST STM32</i>	On-board	STM32L031K6T6	32MHz
<i>ST Nucleo L053R8</i>	<i>ST STM32</i>	On-board	STM32L053R8T6	32MHz
<i>ST Nucleo L073RZ</i>	<i>ST STM32</i>	On-board	STM32L073RZ	32MHz
<i>ST Nucleo L152RE</i>	<i>ST STM32</i>	On-board	STM32L152RET6	32MHz
<i>ST Nucleo L412KB</i>	<i>ST STM32</i>	On-board	STM32L412KBU6	80MHz
<i>ST Nucleo L432KC</i>	<i>ST STM32</i>	On-board	STM32L432KCU6	80MHz
<i>ST Nucleo L433RC-P</i>	<i>ST STM32</i>	On-board	STM32L433RC	80MHz
<i>ST Nucleo L452RE</i>	<i>ST STM32</i>	On-board	STM32L452RET6	80MHz
<i>ST Nucleo L476RG</i>	<i>ST STM32</i>	On-board	STM32L476RGT6	80MHz
<i>ST Nucleo L486RG</i>	<i>ST STM32</i>	On-board	STM32L486RGT6	80MHz
<i>ST Nucleo L496ZG</i>	<i>ST STM32</i>	On-board	STM32L496ZGT6	80MHz
<i>ST Nucleo L496ZG-P</i>	<i>ST STM32</i>	On-board	STM32L496ZGT6P	80MHz
<i>ST Nucleo L4R5ZI</i>	<i>ST STM32</i>	On-board	STM32L4R5ZIT6	120MHz
<i>ST STM32F0308DISCOVERY</i>	<i>ST STM32</i>	On-board	STM32F030R8T6	48MHz
<i>ST STM32F0DISCOVERY</i>	<i>ST STM32</i>	On-board	STM32F051R8T6	48MHz
<i>ST STM32F3DISCOVERY</i>	<i>ST STM32</i>	On-board	STM32F303VCT6	72MHz
<i>ST STM32F4DISCOVERY</i>	<i>ST STM32</i>	On-board	STM32F407VGT6	168MHz
<i>ST STM32L073Z-EVAL</i>	<i>ST STM32</i>	On-board	STM32L073VZT6	32MHz
<i>ST STM32LDISCOVERY</i>	<i>ST STM32</i>	On-board	STM32L152RBT6	32MHz
<i>ST STM32VLDISCOVERY</i>	<i>ST STM32</i>	On-board	STM32F100RBT6	24MHz
<i>ST Sensor Node</i>	<i>ST STM32</i>	On-board	STM32L476JG	80MHz
<i>STM32F103C8 (20k RAM, 64k Flash)</i>	<i>ST STM32</i>	External	STM32F103C8T6	72MHz
<i>STM32F103CB (20k RAM, 128k Flash)</i>	<i>ST STM32</i>	External	STM32F103CBT6	72MHz
<i>STM32F103R8 (20k RAM, 64 Flash)</i>	<i>ST STM32</i>	External	STM32F103R8T6	72MHz
<i>STM32F103RB (20k RAM, 128k Flash)</i>	<i>ST STM32</i>	External	STM32F103RBT6	72MHz
<i>STM32F103RC (48k RAM, 256k Flash)</i>	<i>ST STM32</i>	External	STM32F103RCT6	72MHz
<i>STM32F103RE (64k RAM, 512k Flash)</i>	<i>ST STM32</i>	External	STM32F103RET6	72MHz
<i>STM32F103T8 (20k RAM, 64k Flash)</i>	<i>ST STM32</i>	External	STM32F103T8T6	72MHz
<i>STM32F103TB (20k RAM, 128k Flash)</i>	<i>ST STM32</i>	External	STM32F103TBT6	72MHz
<i>STM32F103VB (20k RAM, 128k Flash)</i>	<i>ST STM32</i>	External	STM32F103VBT6	72MHz
<i>STM32F103VC (48k RAM, 256k Flash)</i>	<i>ST STM32</i>	External	STM32F103VCT6	72MHz
<i>STM32F103VD (64k RAM, 384k Flash)</i>	<i>ST STM32</i>	External	STM32F103VDT6	72MHz
<i>STM32F103VE (64k RAM, 512k Flash)</i>	<i>ST STM32</i>	External	STM32F103VET6	72MHz
<i>STM32F103ZC (48k RAM, 256k Flash)</i>	<i>ST STM32</i>	External	STM32F103ZCT6	72MHz
<i>STM32F103ZD (64k RAM, 384k Flash)</i>	<i>ST STM32</i>	External	STM32F103ZDT6	72MHz
<i>STM32F103ZE (64k RAM, 512k Flash)</i>	<i>ST STM32</i>	External	STM32F103ZET6	72MHz
<i>STM32F303CB (32k RAM, 128k Flash)</i>	<i>ST STM32</i>	External	STM32F303CBT6	72MHz
<i>STM32F7508-DK</i>	<i>ST STM32</i>	On-board	STM32F750N8H6	216MHz
<i>SainSmart Due (Programming Port)</i>	<i>Atmel SAM</i>	External	AT91SAM3X8E	84MHz
<i>SainSmart Due (USB Native Port)</i>	<i>Atmel SAM</i>	External	AT91SAM3X8E	84MHz
<i>Seeed Arch BLE</i>	<i>Nordic nRF51</i>	On-board	NRF51822	16MHz
<i>Seeed Arch Link</i>	<i>Nordic nRF51</i>	On-board	NRF51822	16MHz
<i>Seeed Arch Max</i>	<i>ST STM32</i>	On-board	STM32F407VET6	168MHz

Table 28 – continued from previous page

Name	Platform	Debug	MCU	Freq
Seeed Tiny BLE	Nordic nRF51	On-board	NRF51822	16MHz
Seeed Wio 3G	ST STM32	On-board	STM32F439VI	180MHz
Seeeduino LoRaWAN	Atmel SAM	External	SAMD21G18A	48MHz
Sino:Bit	Nordic nRF51	External	NRF51822	32MHz
Solder Splash Labs DipCortex M0	NXP LPC	External	LPC11U24	50MHz
SparkFun SAMD21 Dev Breakout	Atmel SAM	External	SAMD21G18A	48MHz
SparkFun SAMD21 Mini Breakout	Atmel SAM	External	SAMD21G18A	48MHz
Sparky V1 F303	ST STM32	External	STM32F303CCT6	72MHz
Switch Science mbed LPC1114FN28	NXP LPC	On-board	LPC1114FN28	48MHz
Switch Science mbed LPC824	NXP LPC	On-board	LPC824	30MHz
Taida Century nRF52 mini board	Nordic nRF52	External	NRF52832	64MHz
Thunderboard Sense 2 Sensor-to-Cloud Advanced IoT	Silicon Labs EFM32	On-board	EFR32MG12P432F1024	40MHz
Tiny STM103T	ST STM32	External	STM32F103TBU6	72MHz
Tuino 096	Atmel SAM	External	SAMD21G18A	48MHz
VAkE v1.0	ST STM32	External	STM32F446RET6	180MHz
Waveshare BLE400	Nordic nRF51	External	NRF51822	32MHz
hackaBLE	Nordic nRF52	External	NRF52832	64MHz
ng-beacon	Nordic nRF51	External	NRF51822	16MHz
sakura.io Evaluation Board	ST STM32	On-board	STM32F411RET6	100MHz
u-blox C027	NXP LPC	On-board	LPC1768	96MHz
u-blox C030-N211 IoT Starter Kit	ST STM32	External	STM32F437VG	180MHz
u-blox C030-R410M IoT	ST STM32	On-board	STM32F437VG	180MHz
u-blox C030-U201 IoT Starter Kit	ST STM32	External	STM32F437VG	180MHz
u-blox EVK-NINA-B1	Nordic nRF52	On-board	NRF52832	64MHz
u-blox EVK-ODIN-W2	ST STM32	External	STM32F439ZIY6	168MHz
u-blox ODIN-W2	ST STM32	External	STM32F439ZIY6	168MHz
y5 LPC11U35 mbug	NXP LPC	External	LPC11U35	48MHz
y5 nRF51822 mbug	Nordic nRF51	On-board	NRF51822	16MHz

CMSIS-DAP



CMSIS-DAP is generally implemented as an on-board interface chip, providing direct USB connection from a development board to a debugger running on a host computer on one side, and over JTAG (Joint Test Action Group) or SWD (Serial Wire Debug) to the target device to access the Coresight DAP on the other. Official reference can be found [here](#).

Contents

- Configuration

- *Drivers*
- *Platforms*
- *Frameworks*
- *Boards*

Configuration

You can configure debugging tool using `debug_tool` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:myenv]
platform = ...
board = ...
debug_tool = cmsis-dap
```

If you would like to use this tool for firmware uploading, please change upload protocol:

```
[env:myenv]
platform = ...
board = ...
debug_tool = cmsis-dap
upload_protocol = cmsis-dap
```

More options:

- *Debugging options*
- *Upload options*

Drivers

Windows Please install [Windows serial driver](#) and check “USB Driver Installation” guide for your board.

Mac Not required.

Linux Please install “udev” rules `99-platformio-udev.rules`. If you already installed them before, please check that your rules are up-to-date or repeat steps.

Platforms

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

Frameworks

Name	Description
Arduino	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
CMSIS	The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.
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.
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.
Simba	Simba is an RTOS and build framework. It aims to make embedded programming easy and portable.
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.
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.

Boards

Note: For more detailed board information please scroll tables below by horizontal.

Name	Platform	Debug	MCU	Freq
ARM mbed LPC11U24 (+CAN)	NXP LPC	On-board	LPC11U24	48MHz
Arduino M0 Pro (Programming/Debug Port)	Atmel SAM	On-board	SAMD21G18A	48MHz
Arduino Zero (Programming/Debug Port)	Atmel SAM	On-board	SAMD21G18A	48MHz
Atmel ATSAMR21-XPRO	Atmel SAM	On-board	SAMR21G18A	48MHz
Atmel ATSAMW25-XPRO	Atmel SAM	On-board	SAMD21G18A	48MHz
Atmel SAMD21-XPRO	Atmel SAM	On-board	SAMD21J18A	48MHz
Atmel SAML21-XPRO-B	Atmel SAM	On-board	SAML21J18B	48MHz
BBC micro:bit	Nordic nRF51	On-board	NRF51822	16MHz
Bambino-210E	NXP LPC	On-board	LPC4330	204MHz
Calliope mini	Nordic nRF51	On-board	NRF51822	16MHz
CoCo-ri-Co!	NXP LPC	On-board	LPC812	30MHz
Delta DFBM-NQ620	Nordic nRF52	On-board	NRF52832	64MHz
Delta DFCM-NNN40	Nordic nRF51	On-board	NRF51822	32MHz
Delta DFCM-NNN50	Nordic nRF51	On-board	NRF51822	32MHz
Embedded Artists LPC4088 Display Module	NXP LPC	On-board	LPC4088	120MHz
Embedded Artists LPC4088 QuickStart Board	NXP LPC	On-board	LPC4088	120MHz

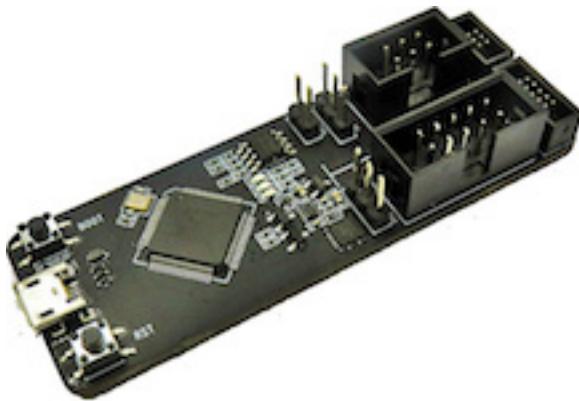
Table 29 – continued from previous page

Name	Platform	Debug	MCU	Freq
<i>Ethernet IoT Starter Kit</i>	<i>Freescale Kinetis</i>	On-board	MK64FN1M0VLL12	120MHz
<i>Freescale Kinetis FRDM-K20D50M</i>	<i>Freescale Kinetis</i>	On-board	MK20DX128VLH5	48MHz
<i>Freescale Kinetis FRDM-K22F</i>	<i>Freescale Kinetis</i>	On-board	MK22FN512VLH12	120MHz
<i>Freescale Kinetis FRDM-K64F</i>	<i>Freescale Kinetis</i>	On-board	MK64FN1M0VLL12	120MHz
<i>Freescale Kinetis FRDM-K66F</i>	<i>Freescale Kinetis</i>	On-board	MK66FN2M0VMD18	180MHz
<i>Freescale Kinetis FRDM-K82F</i>	<i>Freescale Kinetis</i>	On-board	MK82FN256VLL15	150MHz
<i>Freescale Kinetis FRDM-KL05Z</i>	<i>Freescale Kinetis</i>	On-board	MKL05Z32VFM4	48MHz
<i>Freescale Kinetis FRDM-KL25Z</i>	<i>Freescale Kinetis</i>	On-board	MKL25Z128VLK4	48MHz
<i>Freescale Kinetis FRDM-KL27Z</i>	<i>Freescale Kinetis</i>	On-board	MKL27Z64VLH4	48MHz
<i>Freescale Kinetis FRDM-KL43Z</i>	<i>Freescale Kinetis</i>	On-board	MKL43Z256VLH4	48MHz
<i>Freescale Kinetis FRDM-KL46Z</i>	<i>Freescale Kinetis</i>	On-board	MKL46Z256VLL4	48MHz
<i>Freescale Kinetis FRDM-KW41Z</i>	<i>Freescale Kinetis</i>	On-board	MKW41Z512VHT4	48MHz
<i>Hexiwear</i>	<i>Freescale Kinetis</i>	External	MK64FN1M0VDC12	120MHz
<i>JKSoft Wallbot BLE</i>	<i>Nordic nRF51</i>	On-board	NRF51822	16MHz
<i>L476DMWIK</i>	<i>ST STM32</i>	On-board	STM32L476VGT6	80MHz
<i>LPCXpresso11U68</i>	<i>NXP LPC</i>	On-board	LPC11U68	50MHz
<i>LPCXpresso824-MAX</i>	<i>NXP LPC</i>	On-board	LPC824	30MHz
<i>Maxim ARM mbed Enabled Development Platform for MAX32600</i>	<i>Maxim 32</i>	On-board	MAX32600	24MHz
<i>Maxim Wireless Sensor Node Demonstrator</i>	<i>Maxim 32</i>	External	MAX32610	24MHz
<i>Mbed Connect Cloud</i>	<i>ST STM32</i>	On-board	STM32F439ZIY6	168MHz
<i>Moteino M0</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz
<i>NXP LPC800-MAX</i>	<i>NXP LPC</i>	On-board	LPC812	30MHz
<i>NXP LPCXpresso54114</i>	<i>NXP LPC</i>	On-board	LPC54114J256BD64	100MHz
<i>NXP mbed LPC11U24</i>	<i>NXP LPC</i>	On-board	LPC11U24	48MHz
<i>NXP mbed LPC1768</i>	<i>NXP LPC</i>	On-board	LPC1768	96MHz
<i>Nordic Beacon Kit (PCA20006)</i>	<i>Nordic nRF51</i>	On-board	NRF51822	32MHz
<i>Nordic nRF51 Dongle (PCA10031)</i>	<i>Nordic nRF51</i>	On-board	NRF51822	32MHz
<i>Nordic nRF51822-mKIT</i>	<i>Nordic nRF51</i>	On-board	NRF51822	16MHz
<i>Nordic nRF51X22 Development Kit(PCA1000X)</i>	<i>Nordic nRF51</i>	On-board	NRF51822	32MHz
<i>Nordic nRF52-DK</i>	<i>Nordic nRF52</i>	On-board	NRF52832	64MHz
<i>Nordic nRF52840-DK</i>	<i>Nordic nRF52</i>	On-board	NRF52840	64MHz
<i>RedBearLab BLE Nano 1.5</i>	<i>Nordic nRF51</i>	On-board	NRF51822	16MHz
<i>RedBearLab BLE Nano 2</i>	<i>Nordic nRF52</i>	On-board	NRF52832	64MHz
<i>RedBearLab Blend 2</i>	<i>Nordic nRF52</i>	On-board	NRF52832	64MHz
<i>RedBearLab nRF51822</i>	<i>Nordic nRF51</i>	On-board	NRF51822	16MHz
<i>SDT52832B</i>	<i>Nordic nRF52</i>	External	NRF52832	64MHz
<i>Seeed Arch BLE</i>	<i>Nordic nRF51</i>	On-board	NRF51822	16MHz
<i>Seeed Arch Link</i>	<i>Nordic nRF51</i>	On-board	NRF51822	16MHz
<i>Seeed Arch Pro</i>	<i>NXP LPC</i>	On-board	LPC1768	96MHz
<i>Seeed Tiny BLE</i>	<i>Nordic nRF51</i>	On-board	NRF51822	16MHz
<i>Switch Science mbed HRM1017</i>	<i>Nordic nRF51</i>	On-board	NRF51822	16MHz
<i>Switch Science mbed LPC1114FN28</i>	<i>NXP LPC</i>	On-board	LPC1114FN28	48MHz
<i>Switch Science mbed LPC824</i>	<i>NXP LPC</i>	On-board	LPC824	30MHz
<i>Switch Science mbed TY51822r3</i>	<i>Nordic nRF51</i>	On-board	NRF51822	32MHz
<i>VNG VBLUNO51</i>	<i>Nordic nRF51</i>	On-board	NRF51822	16MHz
<i>WIZwiki-W7500</i>	<i>WIZNet W7500</i>	On-board	WIZNET7500	48MHz
<i>WIZwiki-W7500ECO</i>	<i>WIZNet W7500</i>	On-board	WIZNET7500ECO	48MHz
<i>WIZwiki-W7500P</i>	<i>WIZNet W7500</i>	On-board	WIZNET7500P	48MHz
<i>sakura.io Evaluation Board</i>	<i>ST STM32</i>	On-board	STM32F411RET6	100MHz

Table 29 – continued from previous page

Name	Platform	Debug	MCU	Freq
<i>u-blox C027</i>	<i>NXP LPC</i>	On-board	LPC1768	96MHz
<i>u-blox C030-N211 IoT Starter Kit</i>	<i>ST STM32</i>	External	STM32F437VG	180MHz
<i>u-blox C030-U201 IoT Starter Kit</i>	<i>ST STM32</i>	External	STM32F437VG	180MHz
<i>y5 nRF51822 mbug</i>	<i>Nordic nRF51</i>	On-board	NRF51822	16MHz

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 = ...
debug_tool = esp-prog
```

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*

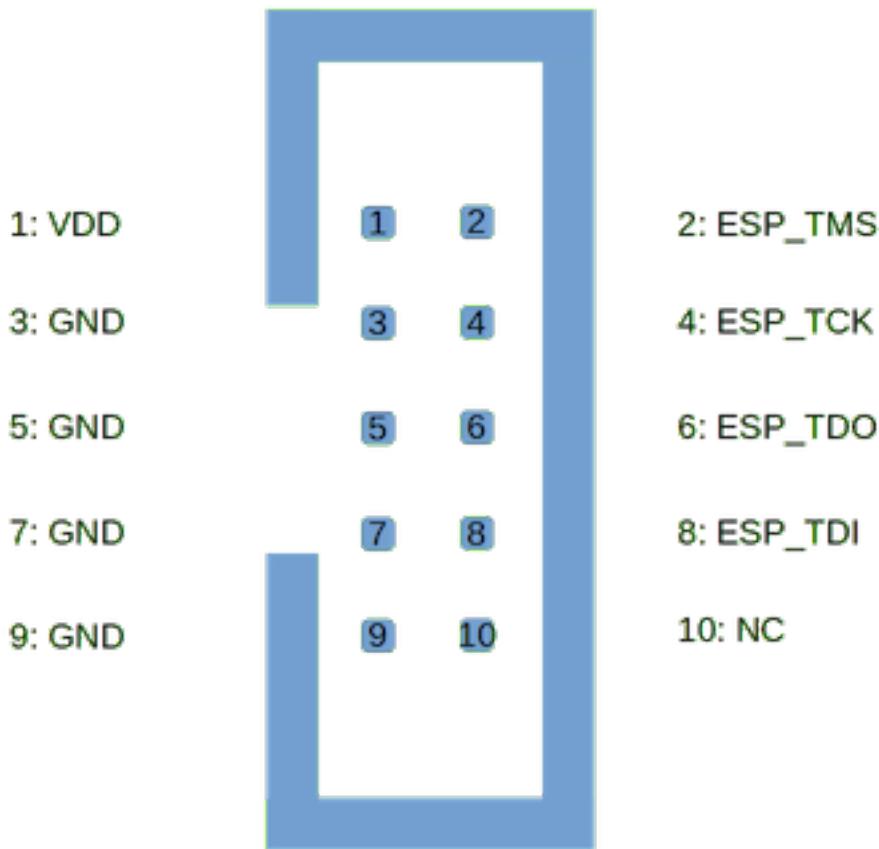
Drivers

Windows See <https://community.platformio.org/t/esp32-pio-unified-debugger/4541/20>

Mac macOS contains default FT232Serial 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



ESP-Prog JTAG 10-Pin Connector	Board JTAG Pin	Description
1	VDD	Positive Supply Voltage — Power supply for JTAG interface drivers
3	GND	Digital ground
2	ESP_TMS	Test Mode State
4	ESP_TCK	JTAG Return Test Clock
6	ESP_TDO	Test Data Out
8	ESP_TDI	Test Data In

Platforms

Name	Description
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.

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP-IDF</i>	Espressif IoT Development Framework. Official development framework for ESP32.
<i>Simba</i>	Simba is an RTOS and build framework. It aims to make embedded programming easy and portable.

Boards

Note: For more detailed board information please scroll tables below by horizontal.

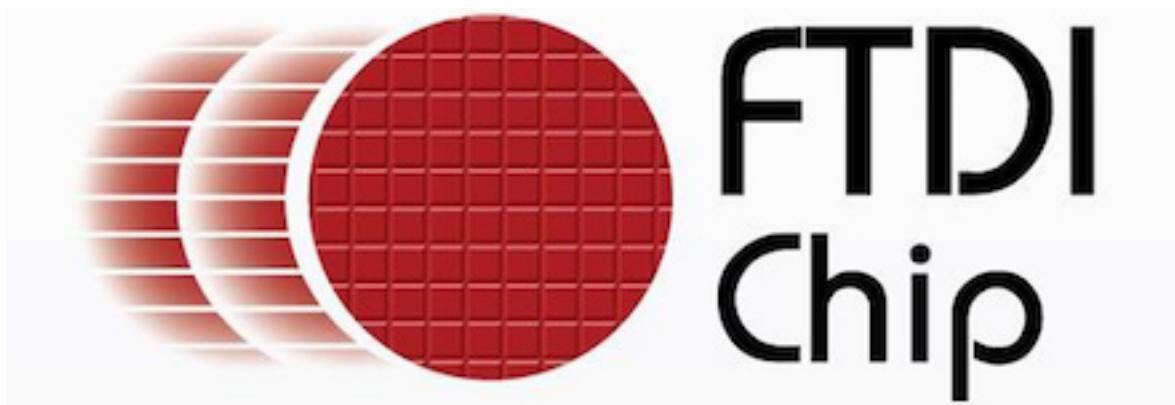
Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>AI Thinker ESP32-CAM</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>ALKS ESP32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>Adafruit ESP32 Feather</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>D-duino-32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>DOIT ESP32 DEVKIT V1</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>Dongsen Tech Pocket 32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>ESP32 FM DevKit</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>ESP32vn IoT Uno</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>ESPECTRO32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>ESPino32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>Espressif ESP-WROVER-KIT</i>	<i>Espressif 32</i>	On-board	ESP32	240MHz	4MB	320KB
<i>Espressif ESP32 Dev Module</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>FireBeetle-ESP32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>Frog Board ESP32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>Heltec WiFi LoRa 32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>Heltec WiFi LoRa 32 (V2)</i>	<i>Espressif 32</i>	External	ESP32	240MHz	8MB	320KB
<i>Heltec Wireless Stick</i>	<i>Espressif 32</i>	External	ESP32	240MHz	8MB	320KB
<i>Hornbill ESP32 Dev</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>Hornbill ESP32 Minima</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>IoTaaP Magnolia</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>MH ET LIVE ESP32DevKIT</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>MH ET LIVE ESP32MiniKit</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>Node32s</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>NodeMCU-32S</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>OLIMEX ESP32-DevKit-LiPo</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>OLIMEX ESP32-EVB</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>OLIMEX ESP32-GATEWAY</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>Pycom LoPy</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>Pycom LoPy4</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	1.25MB
<i>Silicognition wESP32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>SparkFun ESP32 Thing</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>SparkFun LoRa Gateway 1-Channel</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB

Continued on next page

Table 30 – continued from previous page

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>TTGO LoRa32-OLED V1</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>TTGO T-Beam</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	1.25MB
<i>TTGO T1</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>VintLabs ESP32 Devkit</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>WEMOS LOLIN D32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>WEMOS LOLIN D32 PRO</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>WEMOS LOLIN32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>WeMos D1 MINI ESP32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>WeMos WiFi and Bluetooth Battery</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>XinaBox CW02</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>oddWires IoT-Bus Io</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>oddWires IoT-Bus Proteus</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB

FTDI Chip



FTDI Chip develops innovative silicon solutions that enhance interaction with today's technology. When a designer needs to add a USB port, rest assured that FTDI Chip has a full range of USB solutions to get the job done. Official reference can be found [here](#).

Contents

- [Configuration](#)
- [Drivers](#)
- [Platforms](#)
- [Frameworks](#)
- [Boards](#)

Configuration

You can configure debugging tool using `debug_tool` option in “`platformio.ini`” (Project Configuration File):

```
[env:myenv]
platform = ...
board = ...
debug_tool = ftdi
```

If you would like to use this tool for firmware uploading, please change upload protocol:

```
[env:myenv]
platform = ...
board = ...
debug_tool = ftdi
upload_protocol = ftdi
```

More options:

- *Debugging options*
- *Upload options*

Drivers

Windows 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.

Platforms

Name	Description
Espresif 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.
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.
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.
Shakti	Shakti is an open-source initiative by the RISE group at IIT-Madras, which is not only building open source, production grade processors, but also associated components like interconnect fabrics, verification tools, storage controllers, peripheral IPs and SOC tools.
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.

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.
ESP-IDF	Espressif IoT Development Framework. Official development framework for ESP32.
Freedom E SDK	Open Source Software for Developing on the SiFive Freedom E Platform
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.
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.
Shakti SDK	A software development kit for developing applications on Shakti class of processors
Simba	Simba is an RTOS and build framework. It aims to make embedded programming easy and portable.
Tizen RT	Tizen RT is a lightweight RTOS-based platform to support low-end IoT devices

Boards

Note: For more detailed board information please scroll tables below by horizontal.

Name	Platform	Debug	MCU	Fre-quency	Flash	RAM
Artix-7 35T Arty FPGA Evaluation Kit	Shakti	On-board	E-CLASS	50MHz	0B	128KB
Arty A7-100: Artix-7 FPGA Development Board	Shakti	On-board	C-CLASS	50MHz	0B	128MB
Arty FPGA Dev Kit	SiFive	On-board	FE310	450MHz	16MB	256MB
Espressif ESP-WROVER-KIT	Espressif 32	On-board	ESP32	240MHz	4MB	320KB
GAPuino GAP8	RISC-V GAP	On-board	GAP8	250MHz	64MB	8MB
HiFive Unleashed	SiFive	On-board	FU540	1500MHz	32MB	8GB
HiFive1	SiFive	On-board	FE310	320MHz	16MB	16KB
Samsung ARTIK053	Samsung ARTIK	On-board	S5JT200	320MHz	8MB	1.25MB

GD-LINK



GD-Link adapter is a three-in-one multi-function development tool for GD32 series of MCUs. It provides CMSIS-DAP debugger port with JTAG/SWD interface. Official reference can be found [here](#).

Contents

- *Configuration*
- *Drivers*
- *Wiring Connections*
 - *JTAG Interface*
 - *Serial Wire Mode Interface (SWD)*
- *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 = gd-link
```

If you would like to use this tool for firmware uploading, please change upload protocol:

```
[env:myenv]
platform = ...
board = ...
debug_tool = gd-link
upload_protocol = gd-link
```

More options:

- *Debugging options*

- *Upload options*

Drivers

Windows Check vendor recommendations.

Mac Not required.

Linux Please install “udev” rules [99-platformio-udev.rules](#). If you already installed them before, please check that your rules are up-to-date or repeat steps.

Wiring Connections

JTAG Interface

GD-Link JTAG 20-Pin Connector	Board JTAG Pin	Description
+3V3	VCC	Positive Supply Voltage — Power supply for JTAG interface drivers
TMS/IO	TMS	Test Mode State pin
TCK/CLK	TCK	JTAG Return Test Clock
TDO/SWO	TDO	Test Data Out pin
TDI	TDI	Test Data In pin
GND	GND	Digital ground
TReset	RESET	Connect this pin to the (active low) reset input of the target CPU

Serial Wire Mode Interface (SWD)

GD-Link SWD 20-Pin Connector	Board SWD Pin	Description
+3V3	VCC	Positive Supply Voltage — Power supply for JTAG interface drivers
GND	GND	Digital ground
TMS/IO	SWDIO	Data I/O
TCK/CLK	SWCLK	Clock
TReset	RESET	Connect this pin to the (active low) reset input of the target CPU

Platforms

Name	Description
GigaDevice GD32V	The GigaDevice GD32V device is a 32-bit general-purpose microcontroller based on the RISC-V core with an impressive balance of processing power, reduced power consumption and peripheral set.

Frameworks

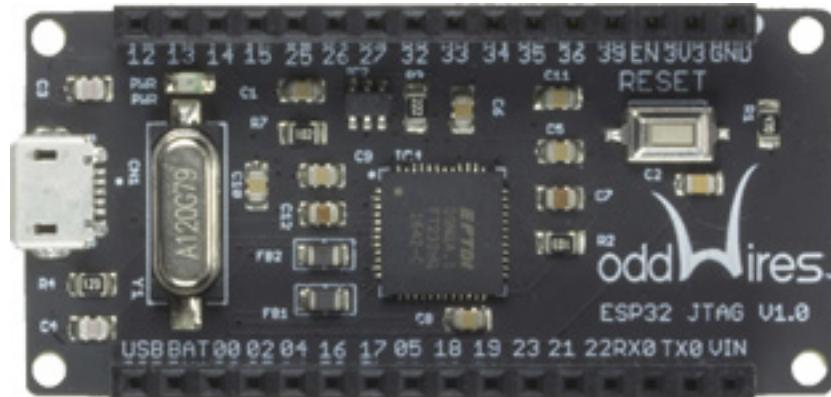
Name	Description
Arduino	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
GigaDevice GD32V SDK	GigaDevice GD32VF103 Firmware Library (SDK)

Boards

Note: For more detailed board information please scroll tables below by horizontal.

Name	Platform	Debug	MCU	Frequency	Flash	RAM
GD32VF103V-EVAL	GigaDevice GD32V	External	GD32VF103VBT6	108MHz	128KB	32KB
Sipeed Longan Nano	GigaDevice GD32V	External	GD32VF103CBT6	108MHz	128KB	32KB
Wio Lite RISC-V	GigaDevice GD32V	External	GD32VF103CBT6	108MHz	128KB	32KB

oddWires IOT-Bus JTAG



This IoT-Bus module provides JTAG debugging for the [oddWires IoT-Bus Io](#) and [oddWires IoT-Bus Proteus](#) boards (can be used with other boards too, see wiring connections below). The board uses the FT232H to provide a USB controller with JTAG support. Both debugging and flashing is possible using this port. Official reference can be found [here](#).

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 = iot-bus-jtag
```

If you would like to use this tool for firmware uploading, please change upload protocol:

```
[env:myenv]
platform = ...
board = ...
debug_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

IOT-Bus Pin	JTAG Pin	Board Pin	JTAG Description
3V3	VCC		Positive Supply Voltage — Power supply for JTAG interface drivers
GND	GND		Digital ground
12	TDI		Test Data In pin
14	TMS		Test Mode State pin
13	TCK		JTAG Return Test Clock
15	TDO		Test Data Out pin
EN	RESET		Connect this pin to the (active low) reset input of the target CPU (EN for ESP32)

Platforms

Name	Description
<i>Espressif 32</i>	Espressif Systems is a privately held fabless semiconductor company. They provide wireless communications and Wi-Fi chips which are widely used in mobile devices and the Internet of Things applications.
<i>Kendryte K210</i>	Kendryte K210 is an AI capable RISC-V64 dual core SoC.

Frameworks

Name	Description
<i>Arduino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>ESP-IDF</i>	Espressif IoT Development Framework. Official development framework for ESP32.
<i>Kendryte Standalone SDK</i>	Kendryte Standalone SDK without OS support
<i>Kendryte FreeRTOS SDK</i>	Kendryte SDK with FreeRTOS support
<i>Simba</i>	Simba is an RTOS and build framework. It aims to make embedded programming easy and portable.

Boards

Note: For more detailed board information please scroll tables below by horizontal.

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>AI Thinker ESP32-CAM</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>ALKS ESP32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>Adafruit ESP32 Feather</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>D-duino-32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>DOIT ESP32 DEVKIT V1</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>Dongsen Tech Pocket 32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>ESP32 FM DevKit</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>ESP32vn IoT Uno</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>ESPectro32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>ESPino32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>Espressif ESP-WROVER-KIT</i>	<i>Espressif 32</i>	On-board	ESP32	240MHz	4MB	320KB
<i>Espressif ESP32 Dev Module</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>FireBeetle-ESP32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>Frog Board ESP32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>Heltec WiFi LoRa 32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>Heltec WiFi LoRa 32 (V2)</i>	<i>Espressif 32</i>	External	ESP32	240MHz	8MB	320KB

Continued on next page

Table 31 – continued from previous page

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Heltec Wireless Stick</i>	<i>Espressif 32</i>	External	ESP32	240MHz	8MB	320KB
<i>Hornbill ESP32 Dev</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>Hornbill ESP32 Minima</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>IoTaaP Magnolia</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>MH ET LIVE ESP32DevKIT</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>MH ET LIVE ESP32MiniKit</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>Node32s</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>NodeMCU-32S</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>OLIMEX ESP32-DevKit-LiPo</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>OLIMEX ESP32-EVB</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>OLIMEX ESP32-GATEWAY</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>Pycom LoPy</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>Pycom LoPy4</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	1.25MB
<i>Silicognition wESP32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>Sipeed MAIX BiT</i>	<i>Kendryte K210</i>	External	K210	400MHz	16MB	6MB
<i>Sipeed MAIX BiT with Mic</i>	<i>Kendryte K210</i>	External	K210	400MHz	16MB	6MB
<i>Sipeed MAIX GO</i>	<i>Kendryte K210</i>	External	K210	400MHz	16MB	6MB
<i>Sipeed MAIX ONE DOCK</i>	<i>Kendryte K210</i>	External	K210	400MHz	16MB	6MB
<i>Sipeed MAIXDUINO</i>	<i>Kendryte K210</i>	External	K210	400MHz	16MB	6MB
<i>SparkFun ESP32 Thing</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>SparkFun LoRa Gateway 1-Channel</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>TTGO LoRa32-OLED V1</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>TTGO T-Beam</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	1.25MB
<i>TTGO T1</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>VintLabs ESP32 Devkit</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>WEMOS LOLIN D32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>WEMOS LOLIN D32 PRO</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>WEMOS LOLIN32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>WeMos D1 MINI ESP32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>WeMos WiFi and Bluetooth Battery</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>XinaBox CW02</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>oddWires IoT-Bus Io</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>oddWires IoT-Bus Proteus</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB

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 = ...
debug_tool = jlink

; SWD interface
upload_protocol = jlink

; JTAG interface
upload_protocol = jlink-jtag
```

More options:

- [Debugging options](#)
- [Upload options](#)

Drivers

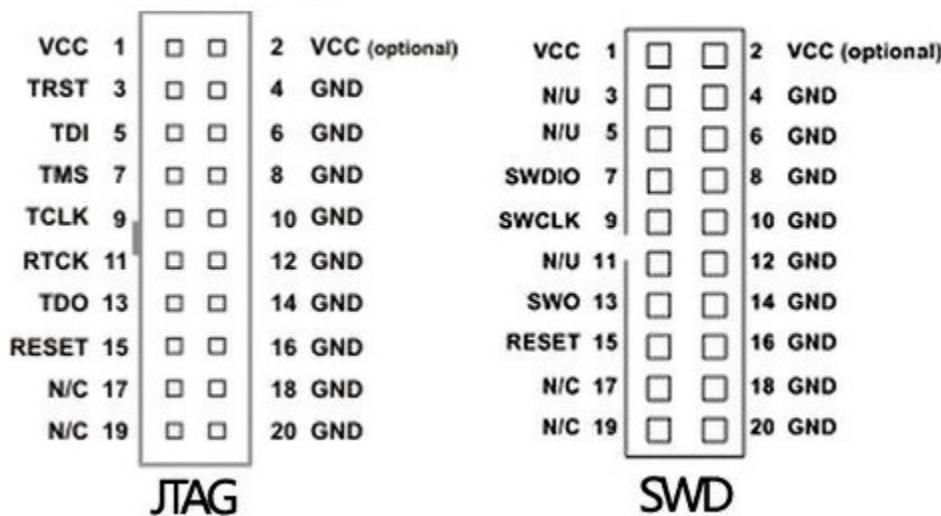
Windows

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

J-Link JTAG 20-Pin Connector	Board JTAG Pin	Description
1	VCC	Positive Supply Voltage — Power supply for JTAG interface drivers
4	GND	Digital ground
5	TDI	Test Data In pin
7	TMS	Test Mode State pin
9	TCK	JTAG Return Test Clock
13	TDO	Test Data Out pin
15	RESET	Connect this pin to the (active low) reset input of the target CPU (EN for ESP32)

Serial Wire Mode Interface (SWD)

J-Link SWD 20-Pin Connector	Board SWD Pin	Description
1	VCC	Positive Supply Voltage — Power supply for JTAG interface drivers
4	GND	Digital ground
7	SWDIO	Data I/O
9	SWCLK	Clock
15	RESET	Connect this pin to the (active low) reset input of the target CPU

Platforms

Name	Description
Aceinna IMU	Open-source, embedded development platform for Aceinna IMU hardware. Run custom algorithms and navigation code on Aceinna IMU/INS hardware.
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.
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.
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.
GigaDevice GD32V	The GigaDevice GD32V device is a 32-bit general-purpose microcontroller based on the RISC-V core with an impressive balance of processing power, reduced power consumption and peripheral set.
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
Kendryte K210	Kendryte K210 is an AI capable RISCV64 dual core SoC.
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.
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.
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.
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.
Shakti	Shakti is an open-source initiative by the RISE group at IIT-Madras, which is not only building open source, production grade processors, but also associated components like interconnect fabrics, verification tools, storage controllers, peripheral IPs and SOC tools.
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.
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.
ST STM32	The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.
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.
WIZ-Net W7500	The IOP (Internet Offload Processor) W7500 is the one-chip solution which integrates an ARM Cortex-M4 128KB Flash and hardwired TCP/IP core for various embedded application platform especially requiring Internet of things

Frameworks

Name	Description
Arduin	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
CM-SIS	The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.
ESP-IDF	Espressif IoT Development Framework. Official development framework for ESP32.
Free-dom E SDK	Open Source Software for Developing on the SiFive Freedom E Platform
Gi-gaDe-vic GD32V SDK	GigaDevice GD32VF103 Firmware Library (SDK)
Kendryte Stan-dalone SDK	Kendryte Standalone SDK without OS support
Kendryte FreeR-TOS SDK	Kendryte SDK with FreeRTOS support
li-bOpenCortex-M	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.
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.
Shakti SDK	A software development kit for developing applications on Shakti class of processors
Simba	Simba is an RTOS and build framework. It aims to make embedded programming easy and portable.
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.
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.

Boards

Note: For more detailed board information please scroll tables below by horizontal.

Name	Platform	Debug	MCU	Freq
1Bitsy	ST STM32	External	STM32F415RGT	168MHz
32F723EDISCOVERY	ST STM32	On-board	STM32F723IEK6	216MHz
3D Printer Controller	ST STM32	External	STM32F407VET6	168MHz
3D Printer control board	ST STM32	External	STM32F446RET6	180MHz
3D printer controller	ST STM32	On-board	STM32F765VIT6	216MHz
3DP001VI Evaluation board for 3D printer	ST STM32	On-board	STM32F401VGT6	84MHz
96Boards B96B-F446VE	ST STM32	On-board	STM32F446VET6	168MHz
AI Thinker ESP32-CAM	Espressif 32	External	ESP32	240MHz
ALKS ESP32	Espressif 32	External	ESP32	240MHz
ARM mbed LPC11U24 (+CAN)	NXP LPC	On-board	LPC11U24	48MHz
Aceinna Low Cost RTK	Aceinna IMU	On-board	STM32F469NIH6	180MHz
Aceinna OpenIMU 300ZA	Aceinna IMU	External	STM32F405RG	120MHz
Aceinna OpenIMU 300ZA	Aceinna IMU	External	STM32F405RG	120MHz
Aceinna OpenIMU 330	Aceinna IMU	External	STM32L431CB	80MHz
Adafruit Bluefruit nRF52832 Feather	Nordic nRF52	On-board	NRF52832	64MHz
Adafruit Circuit Playground Express	Atmel SAM	External	SAMD21G18A	48MHz
Adafruit Crickit M0	Atmel SAM	External	SAMD21G18A	48MHz
Adafruit ESP32 Feather	Espressif 32	External	ESP32	240MHz
Adafruit Feather M0	Atmel SAM	External	SAMD21G18A	48MHz
Adafruit Feather M0 Express	Atmel SAM	External	SAMD21G18A	48MHz
Adafruit Feather M4 Express	Atmel SAM	External	SAMD51J19A	120MHz
Adafruit Feather nRF52840 Express	Nordic nRF52	On-board	NRF52840	64MHz
Adafruit Gemma M0	Atmel SAM	External	SAMD21E18A	48MHz
Adafruit Grand Central M4	Atmel SAM	External	SAMD51P20A	120MHz
Adafruit Hallowing M0	Atmel SAM	External	SAMD21G18A	48MHz
Adafruit Hallowing M4	Atmel SAM	External	SAMD51J19A	120MHz
Adafruit ItsyBitsy M0	Atmel SAM	External	SAMD21G18A	48MHz
Adafruit ItsyBitsy M4	Atmel SAM	External	SAMD51G19A	120MHz
Adafruit MONSTER M4SK	Atmel SAM	External	SAMD51J19A	120MHz
Adafruit Metro M0 Expresss	Atmel SAM	External	SAMD21G18A	48MHz
Adafruit Metro M4	Atmel SAM	External	SAMD51J19A	120MHz
Adafruit Metro M4 AirLift Lite	Atmel SAM	External	SAMD51J19A	120MHz
Adafruit PyGamer Advance M4	Atmel SAM	External	SAMD51J20A	120MHz
Adafruit PyGamer M4 Express	Atmel SAM	External	SAMD51J19A	120MHz
Adafruit PyPortal M4	Atmel SAM	External	SAMD51J20A	120MHz
Adafruit Trellis M4	Atmel SAM	External	SAMD51J19A	120MHz
Adafruit Trinket M0	Atmel SAM	External	SAMD21E18A	48MHz
Adafruit pIRkey	Atmel SAM	External	SAMD21E18A	48MHz
Adafruit pyBadge AirLift M4	Atmel SAM	External	SAMD51J20A	120MHz
Adafruit pyBadge M4 Express	Atmel SAM	External	SAMD51J19A	120MHz
Arduino Due (Programming Port)	Atmel SAM	External	AT91SAM3X8E	84MHz
Arduino Due (USB Native Port)	Atmel SAM	External	AT91SAM3X8E	84MHz
Arduino M0	Atmel SAM	External	SAMD21G18A	48MHz
Arduino M0 Pro (Native USB Port)	Atmel SAM	External	SAMD21G18A	48MHz
Arduino M0 Pro (Programming/Debug Port)	Atmel SAM	On-board	SAMD21G18A	48MHz
Arduino MKR FOX 1200	Atmel SAM	External	SAMD21G18A	48MHz

Table 32 – continued from previous page

Name	Platform	Debug	MCU	Freq
<i>Arduino MKR GSM 1400</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz
<i>Arduino MKR NB 1500</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz
<i>Arduino MKR WAN 1300</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz
<i>Arduino MKR WiFi 1010</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz
<i>Arduino MKR1000</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz
<i>Arduino MKRZERO</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz
<i>Arduino Tian</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz
<i>Arduino Zero (Programming/Debug Port)</i>	<i>Atmel SAM</i>	On-board	SAMD21G18A	48MHz
<i>Arduino Zero (USB Native Port)</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz
<i>Armstrap Eagle 1024</i>	<i>ST STM32</i>	External	STM32F417VGT6	168MHz
<i>Armstrap Eagle 2048</i>	<i>ST STM32</i>	External	STM32F427VIT6	168MHz
<i>Armstrap Eagle 512</i>	<i>ST STM32</i>	External	STM32F407VET6	168MHz
<i>Artix-7 35T Arty FPGA Evaluation Kit</i>	<i>Shakti</i>	On-board	E-CLASS	50MHz
<i>Arty A7-100: Artix-7 FPGA Development Board</i>	<i>Shakti</i>	On-board	C-CLASS	50MHz
<i>Arty FPGA Dev Kit</i>	<i>SiFive</i>	On-board	FE310	450MHz
<i>Atmel ATSAMR21-XPRO</i>	<i>Atmel SAM</i>	On-board	SAMR21G18A	48MHz
<i>Atmel ATSAMW25-XPRO</i>	<i>Atmel SAM</i>	On-board	SAMD21G18A	48MHz
<i>Atmel SAMD21-XPRO</i>	<i>Atmel SAM</i>	On-board	SAMD21J18A	48MHz
<i>Atmel SAML21-XPRO-B</i>	<i>Atmel SAM</i>	On-board	SAML21J18B	48MHz
<i>BBC micro:bit</i>	<i>Nordic nRF51</i>	On-board	NRF51822	16MHz
<i>Bambino-210E</i>	<i>NXP LPC</i>	On-board	LPC4330	204MHz
<i>Black STM32F407VE</i>	<i>ST STM32</i>	External	STM32F407VET6	168MHz
<i>Black STM32F407VG</i>	<i>ST STM32</i>	External	STM32F407VGT6	168MHz
<i>Black STM32F407ZE</i>	<i>ST STM32</i>	External	STM32F407ZET6	168MHz
<i>Black STM32F407ZE</i>	<i>ST STM32</i>	External	STM32F407ZGT6	168MHz
<i>BlackPill F103C8</i>	<i>ST STM32</i>	External	STM32F103C8T6	72MHz
<i>BlackPill F103C8 (128k)</i>	<i>ST STM32</i>	External	STM32F103C8T6	72MHz
<i>BlackPill F303CC</i>	<i>ST STM32</i>	External	STM32F303CCT6	72MHz
<i>Blue STM32F407VE Mini</i>	<i>ST STM32</i>	External	STM32F407VET6	168MHz
<i>BluePill F103C6</i>	<i>ST STM32</i>	External	STM32F103C6T6	72MHz
<i>BluePill F103C8</i>	<i>ST STM32</i>	External	STM32F103C8T6	72MHz
<i>BluePill F103C8 (128k)</i>	<i>ST STM32</i>	External	STM32F103C8T6	72MHz
<i>Bluey nRF52832 IoT</i>	<i>Nordic nRF52</i>	External	NRF52832	64MHz
<i>BluzDK</i>	<i>Nordic nRF51</i>	External	NRF51822	32MHz
<i>CQ Publishing TG-LPC11U35-501</i>	<i>NXP LPC</i>	External	LPC11U35	48MHz
<i>Calliope mini</i>	<i>Nordic nRF51</i>	On-board	NRF51822	16MHz
<i>Circuit Playground Bluefruit</i>	<i>Nordic nRF52</i>	External	NRF52840	64MHz
<i>CoCo-ri-Co!</i>	<i>NXP LPC</i>	On-board	LPC812	30MHz
<i>D-duino-32</i>	<i>Espressif 32</i>	External	ESP32	240MHz
<i>DOIT ESP32 DEVKIT V1</i>	<i>Espressif 32</i>	External	ESP32	240MHz
<i>Delta DFBM-NQ620</i>	<i>Nordic nRF52</i>	On-board	NRF52832	64MHz
<i>Demo F030F4</i>	<i>ST STM32</i>	External	STM32F030F4P6	48MHz
<i>Digistump DigiX</i>	<i>Atmel SAM</i>	External	AT91SAM3X8E	84MHz
<i>DipCortex M3</i>	<i>NXP LPC</i>	External	LPC1347	72MHz
<i>Dongsen Tech Pocket 32</i>	<i>Espressif 32</i>	External	ESP32	240MHz
<i>EA LPC11U35 QuickStart Board</i>	<i>NXP LPC</i>	External	LPC11U35	48MHz
<i>EFM32GG-STK3700 Giant Gecko</i>	<i>Silicon Labs EFM32</i>	On-board	EFM32GG990F1024	48MHz
<i>EFM32LG-STK3600 Leopard Gecko</i>	<i>Silicon Labs EFM32</i>	On-board	EFM32LG990F256	48MHz
<i>EFM32WG-STK3800 Wonder Gecko</i>	<i>Silicon Labs EFM32</i>	On-board	EFM32WG990F256	48MHz

Table 32 – continued from previous page

Name	Platform	Debug	MCU	Freq
<i>EFM32ZG-STK3200 Zero Gecko</i>	<i>Silicon Labs EFM32</i>	On-board	EFM32ZG222F32	24MHz
<i>ESP32 FM DevKit</i>	<i>Espressif 32</i>	External	ESP32	240MHz
<i>ESP32vn IoT Uno</i>	<i>Espressif 32</i>	External	ESP32	240MHz
<i>ESPectro32</i>	<i>Espressif 32</i>	External	ESP32	240MHz
<i>ESPino32</i>	<i>Espressif 32</i>	External	ESP32	240MHz
<i>Embedded Artists LPC4088 Display Module</i>	<i>NXP LPC</i>	On-board	LPC4088	120MHz
<i>Embedded Artists LPC4088 QuickStart Board</i>	<i>NXP LPC</i>	On-board	LPC4088	120MHz
<i>Espotel LoRa Module</i>	<i>ST STM32</i>	External	STM32F411RET6	100MHz
<i>Espressif ESP-WROVER-KIT</i>	<i>Espressif 32</i>	On-board	ESP32	240MHz
<i>Espressif ESP32 Dev Module</i>	<i>Espressif 32</i>	External	ESP32	240MHz
<i>Ethernet IoT Starter Kit</i>	<i>Freescale Kinetis</i>	On-board	MK64FN1M0VLL12	120MHz
<i>F407VG</i>	<i>ST STM32</i>	External	STM32F407VGT6	168MHz
<i>FK407M1</i>	<i>ST STM32</i>	External	STM32F407VET6	168MHz
<i>FireBeetle-ESP32</i>	<i>Espressif 32</i>	External	ESP32	240MHz
<i>Freescale Kinetis FRDM-K20D50M</i>	<i>Freescale Kinetis</i>	On-board	MK20DX128VLH5	48MHz
<i>Freescale Kinetis FRDM-K22F</i>	<i>Freescale Kinetis</i>	On-board	MK22FN512VLH12	120MHz
<i>Freescale Kinetis FRDM-K64F</i>	<i>Freescale Kinetis</i>	On-board	MK64FN1M0VLL12	120MHz
<i>Freescale Kinetis FRDM-K66F</i>	<i>Freescale Kinetis</i>	On-board	MK66FN2M0VMD18	180MHz
<i>Freescale Kinetis FRDM-K82F</i>	<i>Freescale Kinetis</i>	On-board	MK82FN256VLL15	150MHz
<i>Freescale Kinetis FRDM-KL05Z</i>	<i>Freescale Kinetis</i>	On-board	MKL05Z32VFM4	48MHz
<i>Freescale Kinetis FRDM-KL25Z</i>	<i>Freescale Kinetis</i>	On-board	MKL25Z128VLK4	48MHz
<i>Freescale Kinetis FRDM-KL27Z</i>	<i>Freescale Kinetis</i>	On-board	MKL27Z64VLH4	48MHz
<i>Freescale Kinetis FRDM-KL43Z</i>	<i>Freescale Kinetis</i>	On-board	MKL43Z256VLH4	48MHz
<i>Freescale Kinetis FRDM-KL46Z</i>	<i>Freescale Kinetis</i>	On-board	MKL46Z256VLL4	48MHz
<i>Freescale Kinetis FRDM-KL82Z</i>	<i>Freescale Kinetis</i>	External	MKL82Z128VLK7	96MHz
<i>Freescale Kinetis FRDM-KW24D512</i>	<i>Freescale Kinetis</i>	External	MKW24D512	50MHz
<i>Freescale Kinetis FRDM-KW41Z</i>	<i>Freescale Kinetis</i>	On-board	MKW41Z512VHT4	48MHz
<i>Frog Board ESP32</i>	<i>Espressif 32</i>	External	ESP32	240MHz
<i>GD32VF103V-EVAL</i>	<i>GigaDevice GD32V</i>	External	GD32VF103VBT6	108MHz
<i>Heltec WiFi LoRa 32</i>	<i>Espressif 32</i>	External	ESP32	240MHz
<i>Heltec WiFi LoRa 32 (V2)</i>	<i>Espressif 32</i>	External	ESP32	240MHz
<i>Heltec Wireless Stick</i>	<i>Espressif 32</i>	External	ESP32	240MHz
<i>Hexiwear</i>	<i>Freescale Kinetis</i>	External	MK64FN1M0VDC12	120MHz
<i>HiFive1 Rev B</i>	<i>SiFive</i>	On-board	FE310	320MHz
<i>Hornbill ESP32 Dev</i>	<i>Espressif 32</i>	External	ESP32	240MHz
<i>Hornbill ESP32 Minima</i>	<i>Espressif 32</i>	External	ESP32	240MHz
<i>IoTaaP Magnolia</i>	<i>Espressif 32</i>	External	ESP32	240MHz
<i>L476DMWIK</i>	<i>ST STM32</i>	On-board	STM32L476VGT6	80MHz
<i>LPCXpresso11U68</i>	<i>NXP LPC</i>	On-board	LPC11U68	50MHz
<i>LPCXpresso824-MAX</i>	<i>NXP LPC</i>	On-board	LPC824	30MHz
<i>M200 V2</i>	<i>ST STM32</i>	External	STM32F070CBT6	48MHz
<i>MAX32620FTHR</i>	<i>Maxim 32</i>	External	MAX32620FTHR	96MHz
<i>MH ET LIVE ESP32DevKIT</i>	<i>Espressif 32</i>	External	ESP32	240MHz
<i>MH ET LIVE ESP32MiniKit</i>	<i>Espressif 32</i>	External	ESP32	240MHz
<i>MKR Vidor 4000</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz
<i>MTS Dragonfly</i>	<i>ST STM32</i>	External	STM32F411RET6	100MHz
<i>Macchina M2</i>	<i>Atmel SAM</i>	External	AT91SAM3X8E	84MHz
<i>Malyan M200 V1</i>	<i>ST STM32</i>	External	STM32F103CBT6	72MHz
<i>Maple</i>	<i>ST STM32</i>	External	STM32F103RBT6	72MHz

Table 32 – continued from previous page

Name	Platform	Debug	MCU	Freq
<i>Maple (RET6)</i>	<i>ST STM32</i>	External	STM32F103RET6	72MHz
<i>Maple Mini Bootloader 2.0</i>	<i>ST STM32</i>	External	STM32F103CBT6	72MHz
<i>Maple Mini Original</i>	<i>ST STM32</i>	External	STM32F103CBT6	72MHz
<i>Maxim Health Sensor Platform</i>	<i>Maxim 32</i>	External	MAX32620	96MHz
<i>Mbed Connect Cloud</i>	<i>ST STM32</i>	On-board	STM32F439ZIY6	168MHz
<i>Metro nRF52840 Express</i>	<i>Nordic nRF52</i>	On-board	NRF52840	64MHz
<i>Microduino Core STM32 to Flash</i>	<i>ST STM32</i>	External	STM32F103CBT6	72MHz
<i>Microsoft Azure IoT Development Kit (MXChip AZ3166)</i>	<i>ST STM32</i>	On-board	STM32F412ZGT6	100MHz
<i>Minitronics v2.0</i>	<i>Atmel SAM</i>	External	SAMD21J18A	48MHz
<i>Moteino M0</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz
<i>MultiTech mDot</i>	<i>ST STM32</i>	External	STM32F411RET6	100MHz
<i>MultiTech mDot F411</i>	<i>ST STM32</i>	External	STM32F411RET6	100MHz
<i>MultiTech xDot</i>	<i>ST STM32</i>	External	STM32L151CCU6	32MHz
<i>N2+</i>	<i>ST STM32</i>	External	STM32F405RGT6	168MHz
<i>NAMote72</i>	<i>ST STM32</i>	External	STM32L152RC	32MHz
<i>NANO 33 IoT</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz
<i>NGX Technologies BlueBoard-LPC11U24</i>	<i>NXP LPC</i>	External	LPC11U24	48MHz
<i>NXP LPC11C24</i>	<i>NXP LPC</i>	External	LPC11C24	48MHz
<i>NXP LPC11U34</i>	<i>NXP LPC</i>	External	LPC11U34	48MHz
<i>NXP LPC11U37</i>	<i>NXP LPC</i>	External	LPC11U37	48MHz
<i>NXP LPC800-MAX</i>	<i>NXP LPC</i>	On-board	LPC812	30MHz
<i>NXP LPCXpresso1549</i>	<i>NXP LPC</i>	External	LPC1549	72MHz
<i>NXP LPCXpresso54114</i>	<i>NXP LPC</i>	On-board	LPC54114J256BD64	100MHz
<i>NXP LPCXpresso54608</i>	<i>NXP LPC</i>	On-board	LPC54608ET512	180MHz
<i>NXP mbed LPC11U24</i>	<i>NXP LPC</i>	On-board	LPC11U24	48MHz
<i>NXP mbed LPC1768</i>	<i>NXP LPC</i>	On-board	LPC1768	96MHz
<i>Node32s</i>	<i>Espressif 32</i>	External	ESP32	240MHz
<i>NodeMCU-32S</i>	<i>Espressif 32</i>	External	ESP32	240MHz
<i>Nordic Beacon Kit (PCA20006)</i>	<i>Nordic nRF51</i>	On-board	NRF51822	32MHz
<i>Nordic nRF51 Dongle (PCA10031)</i>	<i>Nordic nRF51</i>	On-board	NRF51822	32MHz
<i>Nordic nRF51X22 Development Kit(PCA1000X)</i>	<i>Nordic nRF51</i>	On-board	NRF51822	32MHz
<i>Nordic nRF52-DK</i>	<i>Nordic nRF52</i>	On-board	NRF52832	64MHz
<i>Nordic nRF52840-DK</i>	<i>Nordic nRF52</i>	On-board	NRF52840	64MHz
<i>Nordic nRF52840-DK (Adafruit BSP)</i>	<i>Nordic nRF52</i>	On-board	NRF52840	64MHz
<i>Nucleo G071RB</i>	<i>ST STM32</i>	On-board	STM32G071RBT6	24MHz
<i>OLIMEX ESP32-DevKit-LiPo</i>	<i>Espressif 32</i>	External	ESP32	240MHz
<i>OLIMEX ESP32-EVB</i>	<i>Espressif 32</i>	External	ESP32	240MHz
<i>OLIMEX ESP32-GATEWAY</i>	<i>Espressif 32</i>	External	ESP32	240MHz
<i>OSHChip</i>	<i>Nordic nRF51</i>	External	NRF51822	32MHz
<i>P-Nucleo WB55RG</i>	<i>ST STM32</i>	On-board	STM32WB55RG	64MHz
<i>Pycom LoPy</i>	<i>Espressif 32</i>	External	ESP32	240MHz
<i>Pycom LoPy4</i>	<i>Espressif 32</i>	External	ESP32	240MHz
<i>RAK811 LoRa Tracker</i>	<i>ST STM32</i>	External	STM32L151RBT6	32MHz
<i>RAK811 LoRa Tracker</i>	<i>ST STM32</i>	External	STM32L151RBT6	32MHz
<i>RedBearLab BLE Nano 1.5</i>	<i>Nordic nRF51</i>	On-board	NRF51822	16MHz
<i>RedBearLab BLE Nano 2</i>	<i>Nordic nRF52</i>	On-board	NRF52832	64MHz
<i>RedBearLab Blend 2</i>	<i>Nordic nRF52</i>	On-board	NRF52832	64MHz
<i>RedBearLab nRF51822</i>	<i>Nordic nRF51</i>	On-board	NRF51822	16MHz
<i>RushUp Cloud-JAM</i>	<i>ST STM32</i>	On-board	STM32F401RET6	84MHz

Table 32 – continued from previous page

Name	Platform	Debug	MCU	Freq
<i>RushUp Cloud-JAM L4</i>	<i>ST STM32</i>	On-board	STM32L476RGT6	80MHz
<i>SDT52832B</i>	<i>Nordic nRF52</i>	External	NRF52832	64MHz
<i>SLSTK3400A USB-enabled Happy Gecko</i>	<i>Silicon Labs EFM32</i>	On-board	EFM32HG322F64	25MHz
<i>SLSTK3401A Pearl Gecko PG1</i>	<i>Silicon Labs EFM32</i>	On-board	EFM32PG1B200F256GM48	40MHz
<i>SODAQ Autonomo</i>	<i>Atmel SAM</i>	External	SAMD21J18A	48MHz
<i>SODAQ ExpLoRer</i>	<i>Atmel SAM</i>	External	SAMD21J18A	48MHz
<i>SODAQ ONE</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz
<i>SODAQ SARA</i>	<i>Atmel SAM</i>	External	SAMD21J18A	48MHz
<i>SODAQ SFF</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz
<i>ST 32F3348DISCOVERY</i>	<i>ST STM32</i>	On-board	STM32F334C8T6	72MHz
<i>ST 32F401CDISCOVERY</i>	<i>ST STM32</i>	On-board	STM32F401VCT6	84MHz
<i>ST 32F411EDISCOVERY</i>	<i>ST STM32</i>	On-board	STM32F411VET6	100MHz
<i>ST 32F413HDISCOVERY</i>	<i>ST STM32</i>	On-board	STM32F413ZHT6	100MHz
<i>ST 32F429IDISCOVERY</i>	<i>ST STM32</i>	On-board	STM32F429ZIT6	180MHz
<i>ST 32F469IDISCOVERY</i>	<i>ST STM32</i>	On-board	STM32F469NIH6	180MHz
<i>ST 32F746GDISCOVERY</i>	<i>ST STM32</i>	On-board	STM32F746NGH6	216MHz
<i>ST 32F769IDISCOVERY</i>	<i>ST STM32</i>	On-board	STM32F769NIH6	216MHz
<i>ST 32L0538DISCOVERY</i>	<i>ST STM32</i>	On-board	STM32L053C8T6	32MHz
<i>ST 32L100DISCOVERY</i>	<i>ST STM32</i>	On-board	STM32L100RCT6	32MHz
<i>ST 32L476GDISCOVERY</i>	<i>ST STM32</i>	On-board	STM32L476VGT6	80MHz
<i>ST 32L496GDISCOVERY</i>	<i>ST STM32</i>	On-board	STM32L496AGI6	80MHz
<i>ST DISCO-L072CZ-LRWAN1</i>	<i>ST STM32</i>	On-board	STM32L072CZ	32MHz
<i>ST DISCO-L475VG-IOT01A</i>	<i>ST STM32</i>	On-board	STM32L475VGT6	80MHz
<i>ST Discovery F072RB</i>	<i>ST STM32</i>	On-board	STM32F072RBT6	48MHz
<i>ST Nucleo F030R8</i>	<i>ST STM32</i>	On-board	STM32F030R8T6	48MHz
<i>ST Nucleo F031K6</i>	<i>ST STM32</i>	On-board	STM32F031K6T6	48MHz
<i>ST Nucleo F042K6</i>	<i>ST STM32</i>	On-board	STM32F042K6T6	48MHz
<i>ST Nucleo F070RB</i>	<i>ST STM32</i>	On-board	STM32F070RBT6	48MHz
<i>ST Nucleo F072RB</i>	<i>ST STM32</i>	On-board	STM32F072RBT6	48MHz
<i>ST Nucleo F091RC</i>	<i>ST STM32</i>	On-board	STM32F091RCT6	48MHz
<i>ST Nucleo F103RB</i>	<i>ST STM32</i>	On-board	STM32F103RBT6	72MHz
<i>ST Nucleo F207ZG</i>	<i>ST STM32</i>	On-board	STM32F207ZGT6	120MHz
<i>ST Nucleo F302R8</i>	<i>ST STM32</i>	On-board	STM32F302R8T6	72MHz
<i>ST Nucleo F303K8</i>	<i>ST STM32</i>	On-board	STM32F303K8T6	72MHz
<i>ST Nucleo F303RE</i>	<i>ST STM32</i>	On-board	STM32F303RET6	72MHz
<i>ST Nucleo F303ZE</i>	<i>ST STM32</i>	On-board	STM32F303ZET6	72MHz
<i>ST Nucleo F334R8</i>	<i>ST STM32</i>	On-board	STM32F334R8T6	72MHz
<i>ST Nucleo F401RE</i>	<i>ST STM32</i>	On-board	STM32F401RET6	84MHz
<i>ST Nucleo F410RB</i>	<i>ST STM32</i>	On-board	STM32F410RBT6	100MHz
<i>ST Nucleo F411RE</i>	<i>ST STM32</i>	On-board	STM32F411RET6	100MHz
<i>ST Nucleo F412ZG</i>	<i>ST STM32</i>	On-board	STM32F412ZGT6	100MHz
<i>ST Nucleo F413ZH</i>	<i>ST STM32</i>	On-board	STM32F413ZHT6	100MHz
<i>ST Nucleo F429ZI</i>	<i>ST STM32</i>	On-board	STM32F429ZIT6	180MHz
<i>ST Nucleo F439ZI</i>	<i>ST STM32</i>	On-board	STM32F439ZIT6	180MHz
<i>ST Nucleo F446RE</i>	<i>ST STM32</i>	On-board	STM32F446RET6	180MHz
<i>ST Nucleo F446ZE</i>	<i>ST STM32</i>	On-board	STM32F446ZET6	180MHz
<i>ST Nucleo F722ZE</i>	<i>ST STM32</i>	On-board	STM32F722ZET6	216MHz
<i>ST Nucleo F746ZG</i>	<i>ST STM32</i>	On-board	STM32F746ZGT6	216MHz
<i>ST Nucleo F756ZG</i>	<i>ST STM32</i>	On-board	STM32F756ZG	216MHz

Table 32 – continued from previous page

Name	Platform	Debug	MCU	Freq
<i>ST Nucleo F767ZI</i>	<i>ST STM32</i>	On-board	STM32F767ZIT6	216MHz
<i>ST Nucleo H743ZI</i>	<i>ST STM32</i>	On-board	STM32H743ZIT6	400MHz
<i>ST Nucleo L011K4</i>	<i>ST STM32</i>	On-board	STM32L011K4T6	32MHz
<i>ST Nucleo L031K6</i>	<i>ST STM32</i>	On-board	STM32L031K6T6	32MHz
<i>ST Nucleo L053R8</i>	<i>ST STM32</i>	On-board	STM32L053R8T6	32MHz
<i>ST Nucleo L073RZ</i>	<i>ST STM32</i>	On-board	STM32L073RZ	32MHz
<i>ST Nucleo L152RE</i>	<i>ST STM32</i>	On-board	STM32L152RET6	32MHz
<i>ST Nucleo L412KB</i>	<i>ST STM32</i>	On-board	STM32L412KBU6	80MHz
<i>ST Nucleo L432KC</i>	<i>ST STM32</i>	On-board	STM32L432KCU6	80MHz
<i>ST Nucleo L433RC-P</i>	<i>ST STM32</i>	On-board	STM32L433RC	80MHz
<i>ST Nucleo L452RE</i>	<i>ST STM32</i>	On-board	STM32L452RET6	80MHz
<i>ST Nucleo L476RG</i>	<i>ST STM32</i>	On-board	STM32L476RGT6	80MHz
<i>ST Nucleo L486RG</i>	<i>ST STM32</i>	On-board	STM32L486RGT6	80MHz
<i>ST Nucleo L496ZG</i>	<i>ST STM32</i>	On-board	STM32L496ZGT6	80MHz
<i>ST Nucleo L496ZG-P</i>	<i>ST STM32</i>	On-board	STM32L496ZGT6P	80MHz
<i>ST Nucleo L4R5ZI</i>	<i>ST STM32</i>	On-board	STM32L4R5ZIT6	120MHz
<i>ST STM32F0308DISCOVERY</i>	<i>ST STM32</i>	On-board	STM32F030R8T6	48MHz
<i>ST STM32F0DISCOVERY</i>	<i>ST STM32</i>	On-board	STM32F051R8T6	48MHz
<i>ST STM32F3DISCOVERY</i>	<i>ST STM32</i>	On-board	STM32F303VCT6	72MHz
<i>ST STM32F4DISCOVERY</i>	<i>ST STM32</i>	On-board	STM32F407VGT6	168MHz
<i>ST STM32L073Z-EVAL</i>	<i>ST STM32</i>	On-board	STM32L073VZT6	32MHz
<i>ST STM32LDISCOVERY</i>	<i>ST STM32</i>	On-board	STM32L152RBT6	32MHz
<i>ST STM32VLDISCOVERY</i>	<i>ST STM32</i>	On-board	STM32F100RBT6	24MHz
<i>ST Sensor Node</i>	<i>ST STM32</i>	On-board	STM32L476JG	80MHz
<i>STM32F103C8 (20k RAM, 64k Flash)</i>	<i>ST STM32</i>	External	STM32F103C8T6	72MHz
<i>STM32F103CB (20k RAM, 128k Flash)</i>	<i>ST STM32</i>	External	STM32F103CBT6	72MHz
<i>STM32F103R8 (20k RAM, 64 Flash)</i>	<i>ST STM32</i>	External	STM32F103R8T6	72MHz
<i>STM32F103RB (20k RAM, 128k Flash)</i>	<i>ST STM32</i>	External	STM32F103RBT6	72MHz
<i>STM32F103RC (48k RAM, 256k Flash)</i>	<i>ST STM32</i>	External	STM32F103RCT6	72MHz
<i>STM32F103RE (64k RAM, 512k Flash)</i>	<i>ST STM32</i>	External	STM32F103RET6	72MHz
<i>STM32F103T8 (20k RAM, 64k Flash)</i>	<i>ST STM32</i>	External	STM32F103T8T6	72MHz
<i>STM32F103TB (20k RAM, 128k Flash)</i>	<i>ST STM32</i>	External	STM32F103TBT6	72MHz
<i>STM32F103VB (20k RAM, 128k Flash)</i>	<i>ST STM32</i>	External	STM32F103VBT6	72MHz
<i>STM32F103VC (48k RAM, 256k Flash)</i>	<i>ST STM32</i>	External	STM32F103VCT6	72MHz
<i>STM32F103VD (64k RAM, 384k Flash)</i>	<i>ST STM32</i>	External	STM32F103VDT6	72MHz
<i>STM32F103VE (64k RAM, 512k Flash)</i>	<i>ST STM32</i>	External	STM32F103VET6	72MHz
<i>STM32F103ZC (48k RAM, 256k Flash)</i>	<i>ST STM32</i>	External	STM32F103ZCT6	72MHz
<i>STM32F103ZD (64k RAM, 384k Flash)</i>	<i>ST STM32</i>	External	STM32F103ZDT6	72MHz
<i>STM32F103ZE (64k RAM, 512k Flash)</i>	<i>ST STM32</i>	External	STM32F103ZET6	72MHz
<i>STM32F303CB (32k RAM, 128k Flash)</i>	<i>ST STM32</i>	External	STM32F303CBT6	72MHz
<i>STM32F407VE (192k RAM, 512k Flash)</i>	<i>ST STM32</i>	External	STM32F407VET6	168MHz
<i>STM32F407VG (192k RAM, 1024k Flash)</i>	<i>ST STM32</i>	External	STM32F407VGT6	168MHz
<i>STM32F4Stamp F405</i>	<i>ST STM32</i>	External	STM32F405RGT6	168MHz
<i>STM32F7508-DK</i>	<i>ST STM32</i>	On-board	STM32F750N8H6	216MHz
<i>SainSmart Due (Programming Port)</i>	<i>Atmel SAM</i>	External	AT91SAM3X8E	84MHz
<i>SainSmart Due (USB Native Port)</i>	<i>Atmel SAM</i>	External	AT91SAM3X8E	84MHz
<i>Seeed Arch BLE</i>	<i>Nordic nRF51</i>	On-board	NRF51822	16MHz
<i>Seeed Arch Link</i>	<i>Nordic nRF51</i>	On-board	NRF51822	16MHz
<i>Seeed Arch Max</i>	<i>ST STM32</i>	On-board	STM32F407VET6	168MHz

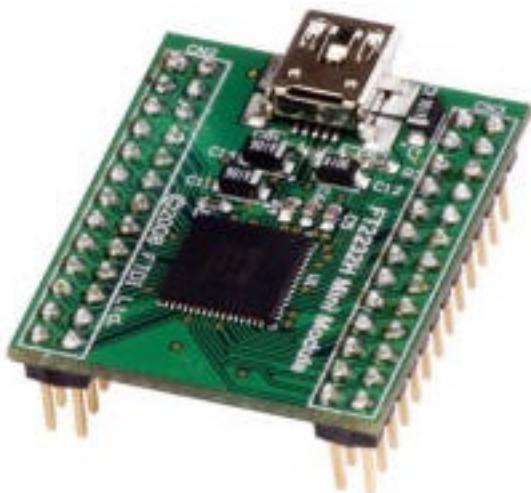
Table 32 – continued from previous page

Name	Platform	Debug	MCU	Freq
<i>Seeed Tiny BLE</i>	<i>Nordic nRF51</i>	On-board	NRF51822	16MHz
<i>Seeed Wio 3G</i>	<i>ST STM32</i>	On-board	STM32F439VI	180MHz
<i>Seeeduino LoRaWAN</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz
<i>Silicognition wESP32</i>	<i>Espressif 32</i>	External	ESP32	240MHz
<i>Sino:Bit</i>	<i>Nordic nRF51</i>	External	NRF51822	32MHz
<i>Sipeed Longan Nano</i>	<i>GigaDevice GD32V</i>	External	GD32VF103CBT6	108MHz
<i>Sipeed MAIX BiT</i>	<i>Kendryte K210</i>	External	K210	400MHz
<i>Sipeed MAIX BiT with Mic</i>	<i>Kendryte K210</i>	External	K210	400MHz
<i>Sipeed MAIX GO</i>	<i>Kendryte K210</i>	External	K210	400MHz
<i>Sipeed MAIX ONE DOCK</i>	<i>Kendryte K210</i>	External	K210	400MHz
<i>Sipeed MAIXDUINO</i>	<i>Kendryte K210</i>	External	K210	400MHz
<i>Solder Splash Labs DipCortex M0</i>	<i>NXP LPC</i>	External	LPC11U24	50MHz
<i>SparkFun ESP32 Thing</i>	<i>Espressif 32</i>	External	ESP32	240MHz
<i>SparkFun LoRa Gateway 1-Channel</i>	<i>Espressif 32</i>	External	ESP32	240MHz
<i>SparkFun SAMD21 Dev Breakout</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz
<i>SparkFun SAMD21 Mini Breakout</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz
<i>Sparky V1 F303</i>	<i>ST STM32</i>	External	STM32F303CCT6	72MHz
<i>Switch Science mbed LPC1114FN28</i>	<i>NXP LPC</i>	On-board	LPC1114FN28	48MHz
<i>Switch Science mbed LPC824</i>	<i>NXP LPC</i>	On-board	LPC824	30MHz
<i>TTGO LoRa32-OLED V1</i>	<i>Espressif 32</i>	External	ESP32	240MHz
<i>TTGO T-Beam</i>	<i>Espressif 32</i>	External	ESP32	240MHz
<i>TTGO T1</i>	<i>Espressif 32</i>	External	ESP32	240MHz
<i>Taida Century nRF52 mini board</i>	<i>Nordic nRF52</i>	External	NRF52832	64MHz
<i>Teensy 3.1 / 3.2</i>	<i>Teensy</i>	External	MK20DX256	72MHz
<i>Teensy 3.5</i>	<i>Teensy</i>	External	MK64FX512	120MHz
<i>Teensy 3.6</i>	<i>Teensy</i>	External	MK66FX1M0	180MHz
<i>Teensy 4.0</i>	<i>Teensy</i>	External	IMXRT1062	600MHz
<i>Teensy LC</i>	<i>Teensy</i>	External	MKL26Z64	48MHz
<i>Thunderboard Sense 2 Sensor-to-Cloud Advanced IoT</i>	<i>Silicon Labs EFM32</i>	On-board	EFR32MG12P432F1024	40MHz
<i>Tiny STM103T</i>	<i>ST STM32</i>	External	STM32F103TBU6	72MHz
<i>Tuino 096</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz
<i>VAkE v1.0</i>	<i>ST STM32</i>	External	STM32F446RET6	180MHz
<i>VintLabs ESP32 Devkit</i>	<i>Espressif 32</i>	External	ESP32	240MHz
<i>WEMOS LOLIN D32</i>	<i>Espressif 32</i>	External	ESP32	240MHz
<i>WEMOS LOLIN D32 PRO</i>	<i>Espressif 32</i>	External	ESP32	240MHz
<i>WEMOS LOLIN32</i>	<i>Espressif 32</i>	External	ESP32	240MHz
<i>WIZwiki-W7500</i>	<i>WIZNet W7500</i>	On-board	WIZNET7500	48MHz
<i>WIZwiki-W7500ECO</i>	<i>WIZNet W7500</i>	On-board	WIZNET7500ECO	48MHz
<i>WIZwiki-W7500P</i>	<i>WIZNet W7500</i>	On-board	WIZNET7500P	48MHz
<i>Waveshare BLE400</i>	<i>Nordic nRF51</i>	External	NRF51822	32MHz
<i>WeMos D1 MINI ESP32</i>	<i>Espressif 32</i>	External	ESP32	240MHz
<i>WeMos WiFi and Bluetooth Battery</i>	<i>Espressif 32</i>	External	ESP32	240MHz
<i>Wio Lite RISC-V</i>	<i>GigaDevice GD32V</i>	External	GD32VF103CBT6	108MHz
<i>XMC1100 Boot Kit</i>	<i>Infineon XMC</i>	On-board	XMC1100	32MHz
<i>XMC1100 H-Bridge 2Go</i>	<i>Infineon XMC</i>	On-board	XMC1100	32MHz
<i>XMC1100 XMC2Go</i>	<i>Infineon XMC</i>	On-board	XMC1100	32MHz
<i>XMC1300 Boot Kit</i>	<i>Infineon XMC</i>	On-board	XMC1300	32MHz
<i>XMC1300 Sense2GoL</i>	<i>Infineon XMC</i>	On-board	XMC1300	32MHz
<i>XMC1400 Boot Kit</i>	<i>Infineon XMC</i>	On-board	XMC1400	48MHz

Table 32 – continued from previous page

Name	Platform	Debug	MCU	Freq
XMC4200 Distance2Go	Infineon XMC	On-board	XMC4200	80MHz
XMC4700 Relax Kit	Infineon XMC	On-board	XMC4700	144MHz
Xenon	Nordic nRF52	External	NRF52840	64MHz
XinaBox CW02	Espressif 32	External	ESP32	240MHz
hackaBLE	Nordic nRF52	External	NRF52832	64MHz
ng-beacon	Nordic nRF51	External	NRF51822	16MHz
oddWires IoT-Bus Io	Espressif 32	External	ESP32	240MHz
oddWires IoT-Bus Proteus	Espressif 32	External	ESP32	240MHz
sakura.io Evaluation Board	ST STM32	On-board	STM32F411RET6	100MHz
u-blox C027	NXP LPC	On-board	LPC1768	96MHz
u-blox C030-N211 IoT Starter Kit	ST STM32	External	STM32F437VG	180MHz
u-blox C030-R410M IoT	ST STM32	On-board	STM32F437VG	180MHz
u-blox C030-U201 IoT Starter Kit	ST STM32	External	STM32F437VG	180MHz
u-blox EVK-NINA-B1	Nordic nRF52	On-board	NRF52832	64MHz
u-blox EVK-ODIN-W2	ST STM32	External	STM32F439ZIY6	168MHz
u-blox ODIN-W2	ST STM32	External	STM32F439ZIY6	168MHz
y5 LPC11U35 mbug	NXP LPC	External	LPC11U35	48MHz
y5 nRF51822 mbug	Nordic nRF51	On-board	NRF51822	16MHz

Mini-Module FT2232H



The FT2232H Mini Module is a USB to dual channel serial/MPSSE/FIFO interface converter module based on the FT2232H USB Hi-Speed IC. The FT2232H handles all the USB signalling and protocol handling. The module provides access to device I/O interfaces via 2 double row 0.1" pitch male connectors. The module is ideal for development purposes to quickly prove functionality of adding USB to a target design. Official reference can be found [here](#)

Contents

- [Configuration](#)
- [Drivers](#)
- [Wiring Connections](#)

- *Platforms*
- *Frameworks*
- *Boards*

Configuration

You can configure debugging tool using `debug_tool` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:myenv]
platform = ...
board = ...
debug_tool = minimodule
```

If you would like to use this tool for firmware uploading, please change upload protocol:

```
[env:myenv]
platform = ...
board = ...
debug_tool = minimodule
upload_protocol = minimodule
```

More options:

- *Debugging options*
- *Upload options*

Drivers

Windows See <https://community.platformio.org/t/esp32-pio-unified-debugger/4541/20>

Mac macOS contains default FT2232H Mini-Module 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

FT2232H Mini-Module Pin	Board JTAG Pin	Description
GND	GND	Digital ground
AD0	TCK	JTAG Return Test Clock
AD1	TDI	Test Data In
AD2	TDO	Test Data Out
AD3	TMS	Test Mode State
RESET#	RESET	Connect this pin to the (active low) reset input of the target CPU (EN for ESP32)

You will also need to connect Vbus [CN3-1] to Vcc [CN3-3] of FT2232H Mini-Module to power the FTDI chip. See [FT2232H Mini-Module Datasheet](#)

Platforms

Name	Description
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.
Kendryte K210	Kendryte K210 is an AI capable RISC-V64 dual core SoC.
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.

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.
ESP-IDF	Espressif IoT Development Framework. Official development framework for ESP32.
Freedom E SDK	Open Source Software for Developing on the SiFive Freedom E Platform
Kendryte Standalone SDK	Kendryte Standalone SDK without OS support
Kendryte FreeRTOS SDK	Kendryte SDK with FreeRTOS support
Simba	Simba is an RTOS and build framework. It aims to make embedded programming easy and portable.

Boards

Note: For more detailed board information please scroll tables below by horizontal.

Name	Platform	Debug	MCU	Frequency	Flash	RAM
AI Thinker ESP32-CAM	Espressif 32	External	ESP32	240MHz	4MB	320KB
ALKS ESP32	Espressif 32	External	ESP32	240MHz	4MB	320KB
Adafruit ESP32 Feather	Espressif 32	External	ESP32	240MHz	4MB	320KB
Arty FPGA Dev Kit	SiFive	On-board	FE310	450MHz	16MB	256MB
D-duino-32	Espressif 32	External	ESP32	240MHz	4MB	320KB
DOIT ESP32 DEVKIT V1	Espressif 32	External	ESP32	240MHz	4MB	320KB
Dongsen Tech Pocket 32	Espressif 32	External	ESP32	240MHz	4MB	320KB
ESP32 FM DevKit	Espressif 32	External	ESP32	240MHz	4MB	320KB
ESP32vn IoT Uno	Espressif 32	External	ESP32	240MHz	4MB	320KB

Continued on next page

Table 33 – continued from previous page

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>ESPectro32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>ESPino32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>Espressif ESP-WROVER-KIT</i>	<i>Espressif 32</i>	On-board	ESP32	240MHz	4MB	320KB
<i>Espressif ESP32 Dev Module</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>FireBeetle-ESP32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>Frog Board ESP32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>Heltec WiFi LoRa 32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>Heltec WiFi LoRa 32 (V2)</i>	<i>Espressif 32</i>	External	ESP32	240MHz	8MB	320KB
<i>Heltec Wireless Stick</i>	<i>Espressif 32</i>	External	ESP32	240MHz	8MB	320KB
<i>Hornbill ESP32 Dev</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>Hornbill ESP32 Minima</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>IoTaaP Magnolia</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>MH ET LIVE ESP32DevKIT</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>MH ET LIVE ESP32MiniKit</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>Node32s</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>NodeMCU-32S</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>OLIMEX ESP32-DevKit-LiPo</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>OLIMEX ESP32-EVB</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>OLIMEX ESP32-GATEWAY</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>Pycom LoPy</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>Pycom LoPy4</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	1.25MB
<i>Silicognition wESP32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>Sipeed MAIX BiT</i>	<i>Kendryte K210</i>	External	K210	400MHz	16MB	6MB
<i>Sipeed MAIX BiT with Mic</i>	<i>Kendryte K210</i>	External	K210	400MHz	16MB	6MB
<i>Sipeed MAIX GO</i>	<i>Kendryte K210</i>	External	K210	400MHz	16MB	6MB
<i>Sipeed MAIX ONE DOCK</i>	<i>Kendryte K210</i>	External	K210	400MHz	16MB	6MB
<i>Sipeed MAIXDUINO</i>	<i>Kendryte K210</i>	External	K210	400MHz	16MB	6MB
<i>SparkFun ESP32 Thing</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>SparkFun LoRa Gateway 1-Channel</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>TTGO LoRa32-OLED V1</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>TTGO T-Beam</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	1.25MB
<i>TTGO T1</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>VintLabs ESP32 Devkit</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>WEMOS LOLIN D32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>WEMOS LOLIN D32 PRO</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>WEMOS LOLIN32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>WeMos DI MINI ESP32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>WeMos WiFi and Bluetooth Battery</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>XinaBox CW02</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>oddWires IoT-Bus Io</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>oddWires IoT-Bus Proteus</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB

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*):

```
[env:myenv]
platform = ...
board = ...
debug_tool = mspdebug
```

If you would like to use this tool for firmware uploading, please change upload protocol:

```
[env:myenv]
platform = ...
board = ...
debug_tool = mspdebug
upload_protocol = mspdebug
```

More options:

- *Debugging options*
- *Upload options*

Drivers

Windows Please “USB Driver Installation” guide for your board.

Mac Not required.

Linux Please install “udev” rules [99-platformio-udev.rules](#). If you already installed them before, please check that your rules are up-to-date or repeat steps.

Platforms

Name	Description
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.

Frameworks

Name	Description
<i>Arduin</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.

Boards

Note: For more detailed board information please scroll tables below by horizontal.

Name	Platform	Debug	MCU	Fre-quency	Flash	RAM
<i>TI FraunchPad MSP-EXP430FR5739LP</i>	<i>TI MSP430</i>	On-board	MSP430FR5739	16MHz	15.37KB	1KB
<i>TI LaunchPad MSP-EXP430F5529LP</i>	<i>TI MSP430</i>	On-board	MSP430F5529	25MHz	47KB	8KB
<i>TI LaunchPad MSP-EXP430FR2311LP</i>	<i>TI MSP430</i>	On-board	MSP430FR2311	16MHz	3.75KB	1KB
<i>TI LaunchPad MSP-EXP430FR2433LP</i>	<i>TI MSP430</i>	On-board	MSP430FR2433	8MHz	15KB	4KB
<i>TI LaunchPad MSP-EXP430FR4133LP</i>	<i>TI MSP430</i>	On-board	MSP430FR4133	8MHz	15KB	2KB
<i>TI LaunchPad MSP-EXP430FR5969LP</i>	<i>TI MSP430</i>	On-board	MSP430FR5969	8MHz	47KB	2KB
<i>TI LaunchPad MSP-EXP430FR5994LP</i>	<i>TI MSP430</i>	On-board	MSP430FR5994	16MHz	256KB	4KB
<i>TI LaunchPad MSP-EXP430FR6989LP</i>	<i>TI MSP430</i>	On-board	MSP430FR6989	8MHz	47KB	2KB
<i>TI LaunchPad MSP-EXP430G2 w/ MSP430G2231</i>	<i>TI MSP430</i>	On-board	MSP430G2231	1MHz	2KB	256B
<i>TI LaunchPad MSP-EXP430G2 w/ MSP430G2452</i>	<i>TI MSP430</i>	On-board	MSP430G2452	16MHz	8KB	256B
<i>TI LaunchPad MSP-EXP430G2553LP</i>	<i>TI MSP430</i>	On-board	MSP430G2553	16MHz	16KB	512B

Olimex ARM-USB-OCD-H



High-speed 3-IN-1 fast USB ARM/ESP32 JTAG, USB-to-RS232 virtual port and power supply 5VDC device. Official reference can be found [here](#).

Contents

- [*Configuration*](#)
- [*Drivers*](#)
- [*Wiring Connections*](#)
- [*Platforms*](#)
- [*Frameworks*](#)
- [*Boards*](#)

Configuration

You can configure debugging tool using `debug_tool` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:myenv]
platform = ...
board = ...
debug_tool = olimex-arm-usb-ocd-h
```

If you would like to use this tool for firmware uploading, please change upload protocol:

```
[env:myenv]
platform = ...
board = ...
debug_tool = olimex-arm-usb-ocd-h
upload_protocol = olimex-arm-usb-ocd-h
```

More options:

- [*Debugging options*](#)
- [*Upload options*](#)

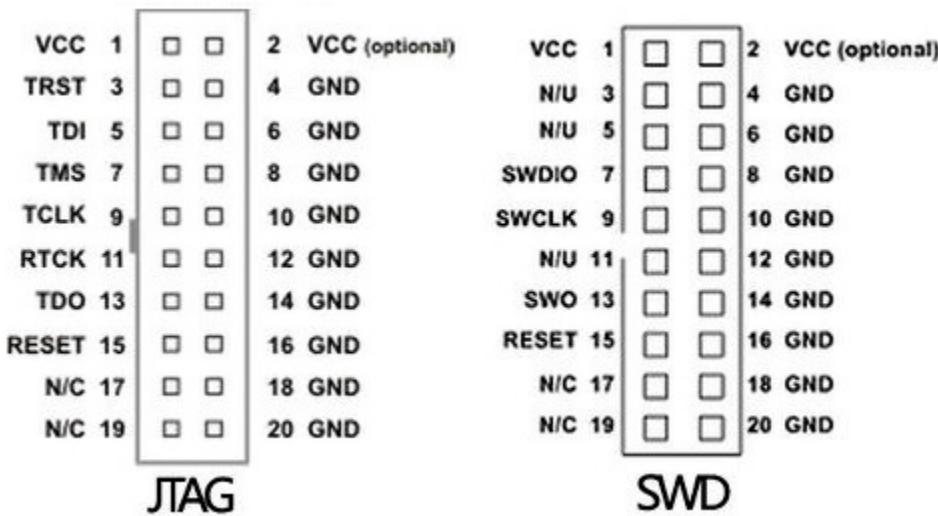
Drivers

Windows See <https://community.platformio.org/t/esp32-pio-unified-debugger/4541/20>

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-OCD-H JTAG 20-Pin Connector	Board JTAG Pin	Description
1	VCC	Positive Supply Voltage — Power supply for JTAG interface drivers
4	GND	Digital ground
5	TDI	Test Data In pin
7	TMS	Test Mode State pin
9	TCK	JTAG Return Test Clock
13	TDO	Test Data Out pin
3	RESET	Connect this pin to the (active low) reset input of the target CPU (EN for ESP32)

Platforms

Name	Description
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.
Kendryte K210	Kendryte K210 is an AI capable RISC-V dual core SoC.
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.

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.
ESP-IDF	Espressif IoT Development Framework. Official development framework for ESP32.
Freedom E SDK	Open Source Software for Developing on the SiFive Freedom E Platform
Kendryte Standalone SDK	Kendryte Standalone SDK without OS support
Kendryte FreeRTOS SDK	Kendryte SDK with FreeRTOS support
Simba	Simba is an RTOS and build framework. It aims to make embedded programming easy and portable.

Boards

Note: For more detailed board information please scroll tables below by horizontal.

Name	Platform	Debug	MCU	Frequency	Flash	RAM
AI Thinker ESP32-CAM	Espressif 32	External	ESP32	240MHz	4MB	320KB
ALKS ESP32	Espressif 32	External	ESP32	240MHz	4MB	320KB
Adafruit ESP32 Feather	Espressif 32	External	ESP32	240MHz	4MB	320KB
Arty FPGA Dev Kit	SiFive	On-board	FE310	450MHz	16MB	256MB
D-duino-32	Espressif 32	External	ESP32	240MHz	4MB	320KB
DOIT ESP32 DEVKIT V1	Espressif 32	External	ESP32	240MHz	4MB	320KB
Dongsen Tech Pocket 32	Espressif 32	External	ESP32	240MHz	4MB	320KB
ESP32 FM DevKit	Espressif 32	External	ESP32	240MHz	4MB	320KB
ESP32vn IoT Uno	Espressif 32	External	ESP32	240MHz	4MB	320KB
ESPectro32	Espressif 32	External	ESP32	240MHz	4MB	320KB
ESPino32	Espressif 32	External	ESP32	240MHz	4MB	320KB
Espressif ESP-WROVER-KIT	Espressif 32	On-board	ESP32	240MHz	4MB	320KB
Espressif ESP32 Dev Module	Espressif 32	External	ESP32	240MHz	4MB	320KB

Continued on next page

Table 34 – continued from previous page

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>FireBeetle-ESP32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>Frog Board ESP32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>Heltec WiFi LoRa 32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>Heltec WiFi LoRa 32 (V2)</i>	<i>Espressif 32</i>	External	ESP32	240MHz	8MB	320KB
<i>Heltec Wireless Stick</i>	<i>Espressif 32</i>	External	ESP32	240MHz	8MB	320KB
<i>Hornbill ESP32 Dev</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>Hornbill ESP32 Minima</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>IoTaaP Magnolia</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>MH ET LIVE ESP32DevKIT</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>MH ET LIVE ESP32MiniKit</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>Node32s</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>NodeMCU-32S</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>OLIMEX ESP32-DevKit-LiPo</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>OLIMEX ESP32-EVB</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>OLIMEX ESP32-GATEWAY</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>Pycom LoPy</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>Pycom LoPy4</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	1.25MB
<i>Silicognition wESP32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>Sipeed MAIX BiT</i>	<i>Kendryte K210</i>	External	K210	400MHz	16MB	6MB
<i>Sipeed MAIX BiT with Mic</i>	<i>Kendryte K210</i>	External	K210	400MHz	16MB	6MB
<i>Sipeed MAIX GO</i>	<i>Kendryte K210</i>	External	K210	400MHz	16MB	6MB
<i>Sipeed MAIX ONE DOCK</i>	<i>Kendryte K210</i>	External	K210	400MHz	16MB	6MB
<i>Sipeed MAIXDUINO</i>	<i>Kendryte K210</i>	External	K210	400MHz	16MB	6MB
<i>SparkFun ESP32 Thing</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>SparkFun LoRa Gateway 1-Channel</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>TTGO LoRa32-OLED V1</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>TTGO T-Beam</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	1.25MB
<i>TTGO T1</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>VintLabs ESP32 Devkit</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>WEMOS LOLIN D32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>WEMOS LOLIN D32 PRO</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>WEMOS LOLIN32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>WeMos D1 MINI ESP32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>WeMos WiFi and Bluetooth Battery</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>XinaBox CW02</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>oddWires IoT-Bus Io</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>oddWires IoT-Bus Proteus</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB

Olimex ARM-USB-OCD



3-IN-1 fast USB ARM/ESP32 JTAG, USB-to-RS232 virtual port and power supply 5-9-12VDC device (supported by OpenOCD ARM debugger software). Official reference can be found [here](#).

Contents

- *Configuration*
- *Drivers*
- *Wiring Connections*
- *Platforms*
- *Frameworks*
- *Boards*

Configuration

You can configure debugging tool using `debug_tool` option in “`platformio.ini`” (Project Configuration File):

```
[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:

```
[env:myenv]
platform = ...
board = ...
debug_tool = olimex-arm-usb-ocd
upload_protocol = olimex-arm-usb-ocd
```

More options:

- *Debugging options*
- *Upload options*

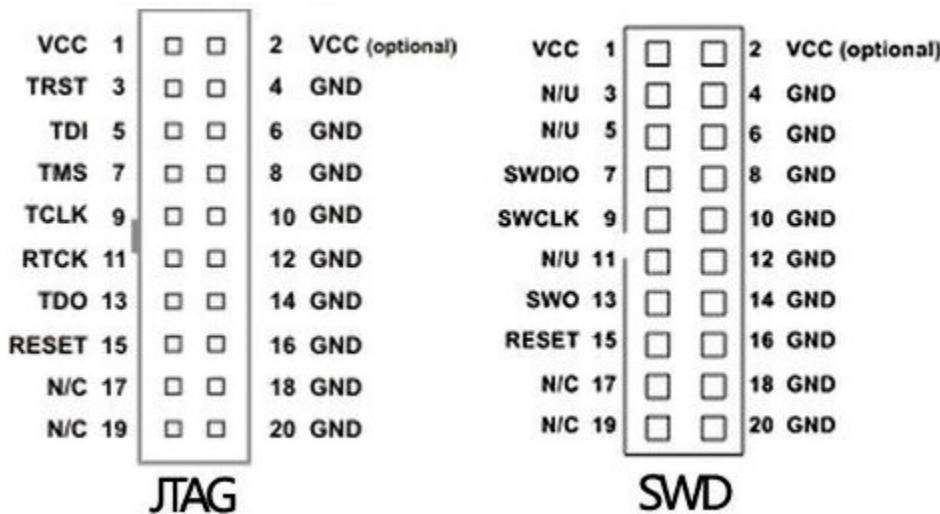
Drivers

Windows See <https://community.platformio.org/t/esp32-pio-unified-debugger/4541/20>

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-OCD JTAG 20-Pin Connector	Board JTAG Pin	Description
1	VCC	Positive Supply Voltage — Power supply for JTAG interface drivers
4	GND	Digital ground
5	TDI	Test Data In pin
7	TMS	Test Mode State pin
9	TCK	JTAG Return Test Clock
13	TDO	Test Data Out pin
3	RESET	Connect this pin to the (active low) reset input of the target CPU (EN for ESP32)

Platforms

Name	Description
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.
Kendryte K210	Kendryte K210 is an AI capable RISC-V dual core SoC.
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.

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.
ESP-IDF	Espressif IoT Development Framework. Official development framework for ESP32.
Freedom E SDK	Open Source Software for Developing on the SiFive Freedom E Platform
Kendryte Standalone SDK	Kendryte Standalone SDK without OS support
Kendryte FreeRTOS SDK	Kendryte SDK with FreeRTOS support
Simba	Simba is an RTOS and build framework. It aims to make embedded programming easy and portable.

Boards

Note: For more detailed board information please scroll tables below by horizontal.

Name	Platform	Debug	MCU	Frequency	Flash	RAM
AI Thinker ESP32-CAM	Espressif 32	External	ESP32	240MHz	4MB	320KB
ALKS ESP32	Espressif 32	External	ESP32	240MHz	4MB	320KB
Adafruit ESP32 Feather	Espressif 32	External	ESP32	240MHz	4MB	320KB
Arty FPGA Dev Kit	SiFive	On-board	FE310	450MHz	16MB	256MB
D-duino-32	Espressif 32	External	ESP32	240MHz	4MB	320KB
DOIT ESP32 DEVKIT V1	Espressif 32	External	ESP32	240MHz	4MB	320KB
Dongsen Tech Pocket 32	Espressif 32	External	ESP32	240MHz	4MB	320KB
ESP32 FM DevKit	Espressif 32	External	ESP32	240MHz	4MB	320KB
ESP32vn IoT Uno	Espressif 32	External	ESP32	240MHz	4MB	320KB
ESPectro32	Espressif 32	External	ESP32	240MHz	4MB	320KB
ESPino32	Espressif 32	External	ESP32	240MHz	4MB	320KB
Espressif ESP-WROVER-KIT	Espressif 32	On-board	ESP32	240MHz	4MB	320KB
Espressif ESP32 Dev Module	Espressif 32	External	ESP32	240MHz	4MB	320KB

Continued on next page

Table 35 – continued from previous page

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>FireBeetle-ESP32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>Frog Board ESP32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>Heltec WiFi LoRa 32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>Heltec WiFi LoRa 32 (V2)</i>	<i>Espressif 32</i>	External	ESP32	240MHz	8MB	320KB
<i>Heltec Wireless Stick</i>	<i>Espressif 32</i>	External	ESP32	240MHz	8MB	320KB
<i>Hornbill ESP32 Dev</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>Hornbill ESP32 Minima</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>IoTaaP Magnolia</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>MH ET LIVE ESP32DevKIT</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>MH ET LIVE ESP32MiniKit</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>Node32s</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>NodeMCU-32S</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>OLIMEX ESP32-DevKit-LiPo</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>OLIMEX ESP32-EVB</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>OLIMEX ESP32-GATEWAY</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>Pycom LoPy</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>Pycom LoPy4</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	1.25MB
<i>Silicognition wESP32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>Sipeed MAIX BiT</i>	<i>Kendryte K210</i>	External	K210	400MHz	16MB	6MB
<i>Sipeed MAIX BiT with Mic</i>	<i>Kendryte K210</i>	External	K210	400MHz	16MB	6MB
<i>Sipeed MAIX GO</i>	<i>Kendryte K210</i>	External	K210	400MHz	16MB	6MB
<i>Sipeed MAIX ONE DOCK</i>	<i>Kendryte K210</i>	External	K210	400MHz	16MB	6MB
<i>Sipeed MAIXDUINO</i>	<i>Kendryte K210</i>	External	K210	400MHz	16MB	6MB
<i>SparkFun ESP32 Thing</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>SparkFun LoRa Gateway 1-Channel</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>TTGO LoRa32-OLED V1</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>TTGO T-Beam</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	1.25MB
<i>TTGO T1</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>VintLabs ESP32 Devkit</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>WEMOS LOLIN D32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>WEMOS LOLIN D32 PRO</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>WEMOS LOLIN32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>WeMos D1 MINI ESP32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>WeMos WiFi and Bluetooth Battery</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>XinaBox CW02</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>oddWires IoT-Bus Io</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>oddWires IoT-Bus Proteus</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB

Olimex ARM-USB-TINY-H



Low-cost and high-speed ARM/ESP32 USB JTAG. Official reference can be found [here](#).

Contents

- [Configuration](#)
- [Drivers](#)
- [Wiring Connections](#)
- [Platforms](#)
- [Frameworks](#)
- [Boards](#)

Configuration

You can configure debugging tool using `debug_tool` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:myenv]
platform = ...
board = ...
debug_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 = ...
debug_tool = olimex-arm-usb-tiny-h
upload_protocol = olimex-arm-usb-tiny-h
```

More options:

- [Debugging options](#)
- [Upload options](#)

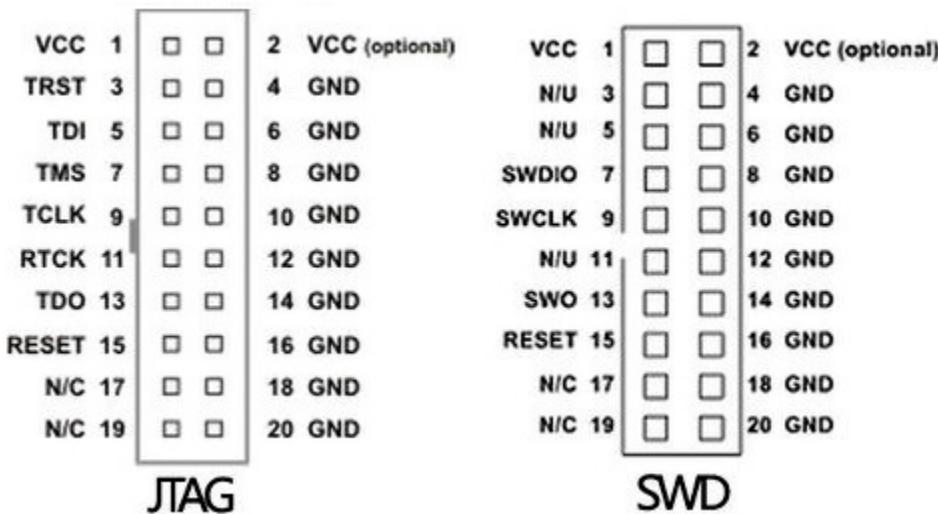
Drivers

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	Board JTAG Pin	Description
1	VCC	Positive Supply Voltage — Power supply for JTAG interface drivers
4	GND	Digital ground
5	TDI	Test Data In pin
7	TMS	Test Mode State pin
9	TCK	JTAG Return Test Clock
13	TDO	Test Data Out pin
3	RESET	Connect this pin to the (active low) reset input of the target CPU (EN for ESP32)

Platforms

Name	Description
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.
Kendryte K210	Kendryte K210 is an AI capable RISC-V dual core SoC.
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.

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.
ESP-IDF	Espressif IoT Development Framework. Official development framework for ESP32.
Freedom E SDK	Open Source Software for Developing on the SiFive Freedom E Platform
Kendryte Standalone SDK	Kendryte Standalone SDK without OS support
Kendryte FreeRTOS SDK	Kendryte SDK with FreeRTOS support
Simba	Simba is an RTOS and build framework. It aims to make embedded programming easy and portable.

Boards

Note: For more detailed board information please scroll tables below by horizontal.

Name	Platform	Debug	MCU	Frequency	Flash	RAM
AI Thinker ESP32-CAM	Espressif 32	External	ESP32	240MHz	4MB	320KB
ALKS ESP32	Espressif 32	External	ESP32	240MHz	4MB	320KB
Adafruit ESP32 Feather	Espressif 32	External	ESP32	240MHz	4MB	320KB
Arty FPGA Dev Kit	SiFive	On-board	FE310	450MHz	16MB	256MB
D-duino-32	Espressif 32	External	ESP32	240MHz	4MB	320KB
DOIT ESP32 DEVKIT V1	Espressif 32	External	ESP32	240MHz	4MB	320KB
Dongsen Tech Pocket 32	Espressif 32	External	ESP32	240MHz	4MB	320KB
ESP32 FM DevKit	Espressif 32	External	ESP32	240MHz	4MB	320KB
ESP32vn IoT Uno	Espressif 32	External	ESP32	240MHz	4MB	320KB
ESPectro32	Espressif 32	External	ESP32	240MHz	4MB	320KB
ESPino32	Espressif 32	External	ESP32	240MHz	4MB	320KB
Espressif ESP-WROVER-KIT	Espressif 32	On-board	ESP32	240MHz	4MB	320KB
Espressif ESP32 Dev Module	Espressif 32	External	ESP32	240MHz	4MB	320KB

Continued on next page

Table 36 – continued from previous page

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>FireBeetle-ESP32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>Frog Board ESP32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>Heltec WiFi LoRa 32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>Heltec WiFi LoRa 32 (V2)</i>	<i>Espressif 32</i>	External	ESP32	240MHz	8MB	320KB
<i>Heltec Wireless Stick</i>	<i>Espressif 32</i>	External	ESP32	240MHz	8MB	320KB
<i>Hornbill ESP32 Dev</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>Hornbill ESP32 Minima</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>IoTaaP Magnolia</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>MH ET LIVE ESP32DevKIT</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>MH ET LIVE ESP32MiniKit</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>Node32s</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>NodeMCU-32S</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>OLIMEX ESP32-DevKit-LiPo</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>OLIMEX ESP32-EVB</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>OLIMEX ESP32-GATEWAY</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>Pycom LoPy</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>Pycom LoPy4</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	1.25MB
<i>Silicognition wESP32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>Sipeed MAIX BiT</i>	<i>Kendryte K210</i>	External	K210	400MHz	16MB	6MB
<i>Sipeed MAIX BiT with Mic</i>	<i>Kendryte K210</i>	External	K210	400MHz	16MB	6MB
<i>Sipeed MAIX GO</i>	<i>Kendryte K210</i>	External	K210	400MHz	16MB	6MB
<i>Sipeed MAIX ONE DOCK</i>	<i>Kendryte K210</i>	External	K210	400MHz	16MB	6MB
<i>Sipeed MAIXDUINO</i>	<i>Kendryte K210</i>	External	K210	400MHz	16MB	6MB
<i>SparkFun ESP32 Thing</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>SparkFun LoRa Gateway 1-Channel</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>TTGO LoRa32-OLED V1</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>TTGO T-Beam</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	1.25MB
<i>TTGO T1</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>VintLabs ESP32 Devkit</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>WEMOS LOLIN D32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>WEMOS LOLIN D32 PRO</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>WEMOS LOLIN32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>WeMos D1 MINI ESP32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>WeMos WiFi and Bluetooth Battery</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>XinaBox CW02</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>oddWires IoT-Bus Io</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>oddWires IoT-Bus Proteus</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB

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](#)

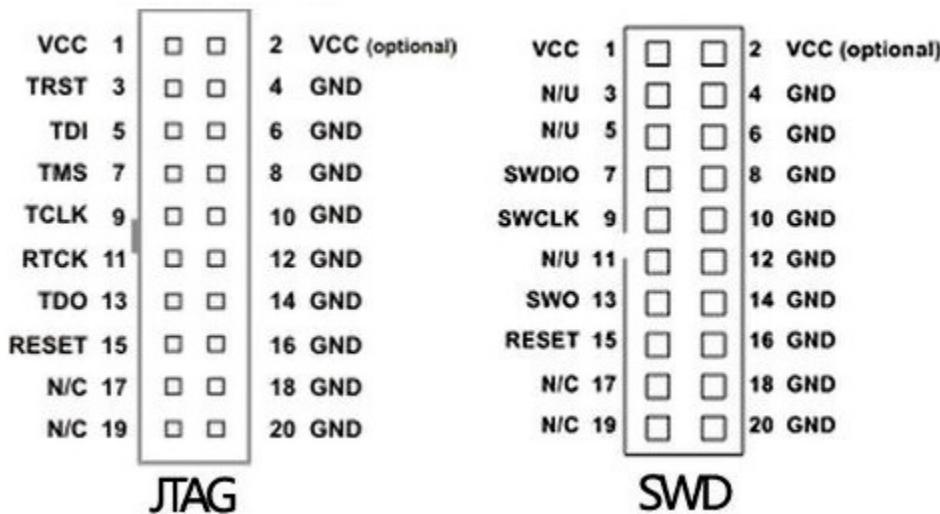
Drivers

Windows See <https://community.platformio.org/t/esp32-pio-unified-debugger/4541/20>

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 20-Pin Connector	Board JTAG Pin	Description
1	VCC	Positive Supply Voltage — Power supply for JTAG interface drivers
4	GND	Digital ground
5	TDI	Test Data In pin
7	TMS	Test Mode State pin
9	TCK	JTAG Return Test Clock
13	TDO	Test Data Out pin
3	RESET	Connect this pin to the (active low) reset input of the target CPU (EN for ESP32)

Platforms

Name	Description
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.
Kendryte K210	Kendryte K210 is an AI capable RISC-V dual core SoC.
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.

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.
ESP-IDF	Espressif IoT Development Framework. Official development framework for ESP32.
Freedom E SDK	Open Source Software for Developing on the SiFive Freedom E Platform
Kendryte Standalone SDK	Kendryte Standalone SDK without OS support
Kendryte FreeRTOS SDK	Kendryte SDK with FreeRTOS support
Simba	Simba is an RTOS and build framework. It aims to make embedded programming easy and portable.

Boards

Note: For more detailed board information please scroll tables below by horizontal.

Name	Platform	Debug	MCU	Frequency	Flash	RAM
AI Thinker ESP32-CAM	Espressif 32	External	ESP32	240MHz	4MB	320KB
ALKS ESP32	Espressif 32	External	ESP32	240MHz	4MB	320KB
Adafruit ESP32 Feather	Espressif 32	External	ESP32	240MHz	4MB	320KB
Arty FPGA Dev Kit	SiFive	On-board	FE310	450MHz	16MB	256MB
D-duino-32	Espressif 32	External	ESP32	240MHz	4MB	320KB
DOIT ESP32 DEVKIT V1	Espressif 32	External	ESP32	240MHz	4MB	320KB
Dongsen Tech Pocket 32	Espressif 32	External	ESP32	240MHz	4MB	320KB
ESP32 FM DevKit	Espressif 32	External	ESP32	240MHz	4MB	320KB
ESP32vn IoT Uno	Espressif 32	External	ESP32	240MHz	4MB	320KB
ESPectro32	Espressif 32	External	ESP32	240MHz	4MB	320KB
ESPino32	Espressif 32	External	ESP32	240MHz	4MB	320KB
Espressif ESP-WROVER-KIT	Espressif 32	On-board	ESP32	240MHz	4MB	320KB
Espressif ESP32 Dev Module	Espressif 32	External	ESP32	240MHz	4MB	320KB

Continued on next page

Table 37 – continued from previous page

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>FireBeetle-ESP32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>Frog Board ESP32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>Heltec WiFi LoRa 32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>Heltec WiFi LoRa 32 (V2)</i>	<i>Espressif 32</i>	External	ESP32	240MHz	8MB	320KB
<i>Heltec Wireless Stick</i>	<i>Espressif 32</i>	External	ESP32	240MHz	8MB	320KB
<i>Hornbill ESP32 Dev</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>Hornbill ESP32 Minima</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>IoTaaP Magnolia</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>MH ET LIVE ESP32DevKIT</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>MH ET LIVE ESP32MiniKit</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>Node32s</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>NodeMCU-32S</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>OLIMEX ESP32-DevKit-LiPo</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>OLIMEX ESP32-EVB</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>OLIMEX ESP32-GATEWAY</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>Pycom LoPy</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>Pycom LoPy4</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	1.25MB
<i>Silicognition wESP32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>Sipeed MAIX BiT</i>	<i>Kendryte K210</i>	External	K210	400MHz	16MB	6MB
<i>Sipeed MAIX BiT with Mic</i>	<i>Kendryte K210</i>	External	K210	400MHz	16MB	6MB
<i>Sipeed MAIX GO</i>	<i>Kendryte K210</i>	External	K210	400MHz	16MB	6MB
<i>Sipeed MAIX ONE DOCK</i>	<i>Kendryte K210</i>	External	K210	400MHz	16MB	6MB
<i>Sipeed MAIXDUINO</i>	<i>Kendryte K210</i>	External	K210	400MHz	16MB	6MB
<i>SparkFun ESP32 Thing</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>SparkFun LoRa Gateway 1-Channel</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>TTGO LoRa32-OLED V1</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>TTGO T-Beam</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	1.25MB
<i>TTGO T1</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>VintLabs ESP32 Devkit</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>WEMOS LOLIN D32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>WEMOS LOLIN D32 PRO</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>WEMOS LOLIN32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>WeMos D1 MINI ESP32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>WeMos WiFi and Bluetooth Battery</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>XinaBox CW02</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>oddWires IoT-Bus Io</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>oddWires IoT-Bus Proteus</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB

QEMU



QEMU is a free and open-source emulator that performs hardware virtualization. Official reference can be found [here](#).

Contents

- *Configuration*
- *Platforms*
- *Frameworks*
- *Boards*

Configuration

You can configure debugging tool using *debug_tool* option in “*platformio.ini*” (*Project Configuration File*):

```
[env:myenv]
platform = ...
board = ...
debug_tool = qemu
```

More options:

- *Debugging options*

Platforms

Name	Description
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.

Frameworks

Name	Description
Freedom E SDK	Open Source Software for Developing on the SiFive Freedom E Platform

Boards

Note: For more detailed board information please scroll tables below by horizontal.

Name	Platform	Debug	MCU	Frequency	Flash	RAM
Arty FPGA Dev Kit	SiFive	On-board	FE310	450MHz	16MB	256MB
HiFive Unleashed	SiFive	On-board	FU540	1500MHz	32MB	8GB
HiFive1	SiFive	On-board	FE310	320MHz	16MB	16KB

RV-LINK

RISC-V emulator implemented with RISC-V development board. Unlike other emulators: RV-LINK interacts directly with GDB via a USB serial port and does not require an intermediary such as OpenOCD. Official reference can be found [here](#).

Contents

- *Configuration*
- *Platforms*
- *Frameworks*
- *Boards*

Configuration

You can configure debugging tool using `debug_tool` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:myenv]
platform = ...
board = ...
debug_tool = rv-link
```

If you would like to use this tool for firmware uploading, please change upload protocol:

```
[env:myenv]
platform = ...
board = ...
debug_tool = rv-link
upload_protocol = rv-link
```

More options:

- *Debugging options*
- *Upload options*

Platforms

Name	Description
<i>GigaDevice GD32V</i>	The GigaDevice GD32V device is a 32-bit general-purpose microcontroller based on the RISC-V core with an impressive balance of processing power, reduced power consumption and peripheral set.

Frameworks

Name	Description
<i>Arduino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>GigaDevice GD32V SDK</i>	GigaDevice GD32VF103 Firmware Library (SDK)

Boards

Note: For more detailed board information please scroll tables below by horizontal.

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>GD32VF103V-EVAL</i>	<i>GigaDevice GD32V</i>	External	GD32VF103VBT6	108MHz	128KB	32KB
<i>Sipeed Longan Nano</i>	<i>GigaDevice GD32V</i>	External	GD32VF103CBT6	108MHz	128KB	32KB
<i>Wio Lite RISC-V</i>	<i>GigaDevice GD32V</i>	External	GD32VF103CBT6	108MHz	128KB	32KB

Sipeed RV Debugger



High-speed 3-IN-1 fast USB ARM/ESP32 JTAG, USB-to-RS232 virtual port and power supply 5VDC device. Official reference can be found [here](#).

Contents

- *Configuration*
- *Drivers*
- *Wiring Connections*
- *Platforms*

- *Frameworks*
- *Boards*

Configuration

You can configure debugging tool using `debug_tool` option in “`platformio.ini`” (*Project Configuration File*):

```
[env:myenv]
platform = ...
board = ...
debug_tool = sipeed-rv-debugger
```

If you would like to use this tool for firmware uploading, please change upload protocol:

```
[env:myenv]
platform = ...
board = ...
debug_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 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

Sipeed RV Debugger Connector	Board Pin	Description
1	GND	Digital ground
2	TDI	Test Data In pin
6	TMS	Test Mode State pin
10	TCK	JTAG Return Test Clock
8	TDO	Test Data Out pin
4	RST	Connect this pin to the (active low) reset input of the target CPU

Platforms

Name	Description
<i>GigaDevice GD32V</i>	The GigaDevice GD32V device is a 32-bit general-purpose microcontroller based on the RISC-V core with an impressive balance of processing power, reduced power consumption and peripheral set.
<i>Kendryte K210</i>	Kendryte K210 is an AI capable RISC-V64 dual core SoC.

Frameworks

Name	Description
<i>Arduino</i>	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
<i>GigaDevice GD32V SDK</i>	GigaDevice GD32VF103 Firmware Library (SDK)
<i>Kendryte Standalone SDK</i>	Kendryte Standalone SDK without OS support
<i>Kendryte FreeRTOS SDK</i>	Kendryte SDK with FreeRTOS support

Boards

Note: For more detailed board information please scroll tables below by horizontal.

Name	Platform	Debug	MCU	Fre-quency	Flash	RAM
<i>GD32VF103V-EVAL</i>	<i>GigaDevice GD32V</i>	External	GD32VF103VBT6	108MHz	128KB	32KB
<i>Sipeed Longan Nano</i>	<i>GigaDevice GD32V</i>	External	GD32VF103CBT6	108MHz	128KB	32KB
<i>Sipeed MAIX BiT</i>	<i>Kendryte K210</i>	External	K210	400MHz	16MB	6MB
<i>Sipeed MAIX BiT with Mic</i>	<i>Kendryte K210</i>	External	K210	400MHz	16MB	6MB
<i>Sipeed MAIX GO</i>	<i>Kendryte K210</i>	External	K210	400MHz	16MB	6MB
<i>Sipeed MAIX ONE DOCK</i>	<i>Kendryte K210</i>	External	K210	400MHz	16MB	6MB
<i>Sipeed MAIXDUINO</i>	<i>Kendryte K210</i>	External	K210	400MHz	16MB	6MB
<i>Wio Lite RISC-V</i>	<i>GigaDevice GD32V</i>	External	GD32VF103CBT6	108MHz	128KB	32KB

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*

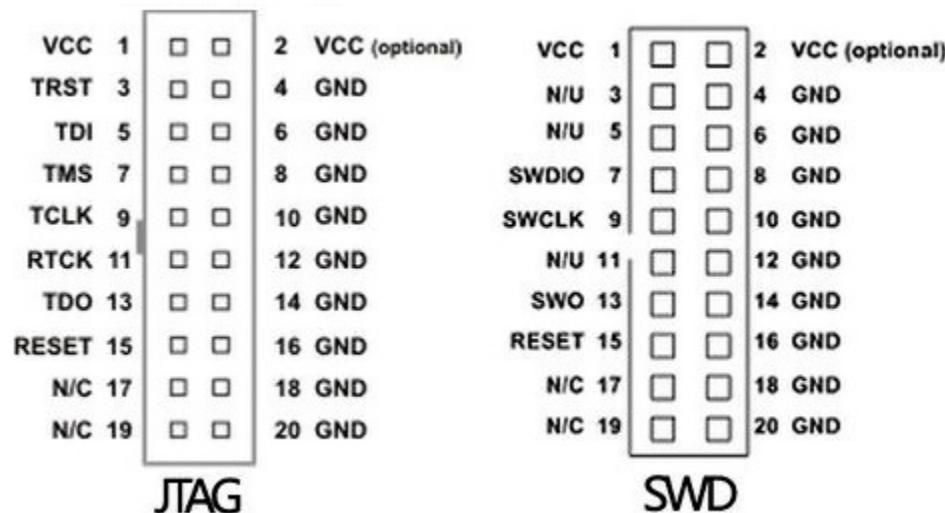
Drivers

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

ST-Link JTAG 20-Pin Connector	Board JTAG Pin	Description
1	VCC	Positive Supply Voltage — Power supply for JTAG interface drivers
4	GND	Digital ground
5	TDI	Test Data In pin
7	TMS	Test Mode State pin
9	TCK	JTAG Return Test Clock
13	TDO	Test Data Out pin
15	RESET	Connect this pin to the (active low) reset input of the target CPU

Serial Wire Mode Interface (SWD)

ST-Link SWD 20-Pin Connector	Board SWD Pin	Description
1	VCC	Positive Supply Voltage — Power supply for JTAG interface drivers
4	GND	Digital ground
7	SWDIO	Data I/O
9	SWCLK	Clock
15	RESET	Connect this pin to the (active low) reset input of the target CPU

Platforms

Name	Description
<i>Aceinna IMU</i>	Open-source, embedded development platform for Aceinna IMU hardware. Run custom algorithms and navigation code on Aceinna IMU/INS hardware.
<i>Atmel SAM</i>	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.
<i>Nordic nRF51</i>	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.
<i>Nordic nRF52</i>	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.
<i>ST STM32</i>	The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining full integration and ease of development.
<i>ST STM8</i>	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.

Frameworks

Name	Description
Arduino	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
CMSIS	The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.
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.
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.
Simba	Simba is an RTOS and build framework. It aims to make embedded programming easy and portable.
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.
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.

Boards

Note: For more detailed board information please scroll tables below by horizontal.

Name	Platform	Debug	MCU	Frequency	Flash
1Bitsy	<i>ST STM32</i>	External	STM32F415RGT	168MHz	1MB
32F723EDISCOVERY	<i>ST STM32</i>	On-board	STM32F723IEK6	216MHz	512K
3D Printer Controller	<i>ST STM32</i>	External	STM32F407VET6	168MHz	512K
3D Printer control board	<i>ST STM32</i>	External	STM32F446RET6	180MHz	512K
3D printer controller	<i>ST STM32</i>	On-board	STM32F765VIT6	216MHz	2MB
3DP001VI Evaluation board for 3D printer	<i>ST STM32</i>	On-board	STM32F401VGT6	84MHz	512K
96Boards B96B-F446VE	<i>ST STM32</i>	On-board	STM32F446VET6	168MHz	512K
Aceinna Low Cost RTK	<i>Aceinna IMU</i>	On-board	STM32F469NIH6	180MHz	1MB
Aceinna OpenIMU 300ZA	<i>Aceinna IMU</i>	External	STM32F405RG	120MHz	1MB
Aceinna OpenIMU 300ZA	<i>Aceinna IMU</i>	External	STM32F405RG	120MHz	1MB
Aceinna OpenIMU 330	<i>Aceinna IMU</i>	External	STM32L431CB	80MHz	128K
Arduino Due (Programming Port)	<i>Atmel SAM</i>	External	AT91SAM3X8E	84MHz	512K
Arduino Due (USB Native Port)	<i>Atmel SAM</i>	External	AT91SAM3X8E	84MHz	512K
Armstrap Eagle 1024	<i>ST STM32</i>	External	STM32F417VGT6	168MHz	1MB
Armstrap Eagle 2048	<i>ST STM32</i>	External	STM32F427VIT6	168MHz	1.99M
Armstrap Eagle 512	<i>ST STM32</i>	External	STM32F407VET6	168MHz	512K

Continu

Table 38 – continued from previous page

Name	Platform	Debug	MCU	Frequency	Flash
<i>Black STM32F407VE</i>	<i>ST STM32</i>	External	STM32F407VET6	168MHz	512K
<i>Black STM32F407VG</i>	<i>ST STM32</i>	External	STM32F407VGT6	168MHz	512K
<i>Black STM32F407ZE</i>	<i>ST STM32</i>	External	STM32F407ZET6	168MHz	512K
<i>Black STM32F407ZE</i>	<i>ST STM32</i>	External	STM32F407ZGT6	168MHz	1MB
<i>BlackPill F103C8</i>	<i>ST STM32</i>	External	STM32F103C8T6	72MHz	64KB
<i>BlackPill F103C8 (128k)</i>	<i>ST STM32</i>	External	STM32F103C8T6	72MHz	128K
<i>BlackPill F303CC</i>	<i>ST STM32</i>	External	STM32F303CCT6	72MHz	256K
<i>Blue STM32F407VE Mini</i>	<i>ST STM32</i>	External	STM32F407VET6	168MHz	512K
<i>BluePill F103C6</i>	<i>ST STM32</i>	External	STM32F103C6T6	72MHz	32KB
<i>BluePill F103C8</i>	<i>ST STM32</i>	External	STM32F103C8T6	72MHz	64KB
<i>BluePill F103C8 (128k)</i>	<i>ST STM32</i>	External	STM32F103C8T6	72MHz	128K
<i>Bluey nRF52832 IoT</i>	<i>Nordic nRF52</i>	External	NRF52832	64MHz	512K
<i>BluzDK</i>	<i>Nordic nRF51</i>	External	NRF51822	32MHz	256K
<i>Delta DFBM-NQ620</i>	<i>Nordic nRF52</i>	On-board	NRF52832	64MHz	512K
<i>Demo F030F4</i>	<i>ST STM32</i>	External	STM32F030F4P6	48MHz	16KB
<i>Digistump DigiX</i>	<i>Atmel SAM</i>	External	AT91SAM3X8E	84MHz	512K
<i>Espotel LoRa Module</i>	<i>ST STM32</i>	External	STM32F411RET6	100MHz	512K
<i>F407VG</i>	<i>ST STM32</i>	External	STM32F407VGT6	168MHz	512K
<i>FK407M1</i>	<i>ST STM32</i>	External	STM32F407VET6	168MHz	512K
<i>L476DMWIK</i>	<i>ST STM32</i>	On-board	STM32L476VGT6	80MHz	1MB
<i>M200 V2</i>	<i>ST STM32</i>	External	STM32F070CBT6	48MHz	120K
<i>MTS Dragonfly</i>	<i>ST STM32</i>	External	STM32F411RET6	100MHz	512K
<i>Macchina M2</i>	<i>Atmel SAM</i>	External	AT91SAM3X8E	84MHz	512K
<i>Malyan M200 V1</i>	<i>ST STM32</i>	External	STM32F103CBT6	72MHz	120K
<i>Maple</i>	<i>ST STM32</i>	External	STM32F103RBT6	72MHz	108K
<i>Maple (RET6)</i>	<i>ST STM32</i>	External	STM32F103RET6	72MHz	256K
<i>Maple Mini Bootloader 2.0</i>	<i>ST STM32</i>	External	STM32F103CBT6	72MHz	120K
<i>Maple Mini Original</i>	<i>ST STM32</i>	External	STM32F103CBT6	72MHz	108K
<i>Mbed Connect Cloud</i>	<i>ST STM32</i>	On-board	STM32F439ZIY6	168MHz	2MB
<i>Microduino Core STM32 to Flash</i>	<i>ST STM32</i>	External	STM32F103CBT6	72MHz	105.4
<i>Microsoft Azure IoT Development Kit (MXChip AZ3166)</i>	<i>ST STM32</i>	On-board	STM32F412ZGT6	100MHz	1MB
<i>MultiTech mDot</i>	<i>ST STM32</i>	External	STM32F411RET6	100MHz	512K
<i>MultiTech mDot F411</i>	<i>ST STM32</i>	External	STM32F411RET6	100MHz	512K
<i>MultiTech xDot</i>	<i>ST STM32</i>	External	STM32L151CCU6	32MHz	256K
<i>N2+</i>	<i>ST STM32</i>	External	STM32F405RGT6	168MHz	1MB
<i>NAMote72</i>	<i>ST STM32</i>	External	STM32L152RC	32MHz	256K
<i>Nordic Beacon Kit (PCA20006)</i>	<i>Nordic nRF51</i>	On-board	NRF51822	32MHz	256K
<i>Nordic nRF51X22 Development Kit(PCA1000X)</i>	<i>Nordic nRF51</i>	On-board	NRF51822	32MHz	256K
<i>Nordic nRF52-DK</i>	<i>Nordic nRF52</i>	On-board	NRF52832	64MHz	512K
<i>Nordic nRF52840-DK</i>	<i>Nordic nRF52</i>	On-board	NRF52840	64MHz	1MB
<i>Nucleo G071RB</i>	<i>ST STM32</i>	On-board	STM32G071RBT6	24MHz	2MB
<i>OSHChip</i>	<i>Nordic nRF51</i>	External	NRF51822	32MHz	256K
<i>P-Nucleo WB55RG</i>	<i>ST STM32</i>	On-board	STM32WB55RG	64MHz	512K
<i>RAK811 LoRa Tracker</i>	<i>ST STM32</i>	External	STM32L151RBT6	32MHz	128K
<i>RAK811 LoRa Tracker</i>	<i>ST STM32</i>	External	STM32L151RBT6	32MHz	128K
<i>RedBearLab BLE Nano 1.5</i>	<i>Nordic nRF51</i>	On-board	NRF51822	16MHz	256K
<i>RedBearLab BLE Nano 2</i>	<i>Nordic nRF52</i>	On-board	NRF52832	64MHz	512K
<i>RedBearLab Blend 2</i>	<i>Nordic nRF52</i>	On-board	NRF52832	64MHz	512K
<i>RedBearLab nRF51822</i>	<i>Nordic nRF51</i>	On-board	NRF51822	16MHz	256K

Continu

Table 38 – continued from previous page

Name	Platform	Debug	MCU	Frequency	Flash
<i>RushUp Cloud-JAM</i>	<i>ST STM32</i>	On-board	STM32F401RET6	84MHz	512K
<i>RushUp Cloud-JAM L4</i>	<i>ST STM32</i>	On-board	STM32L476RGT6	80MHz	1MB
<i>SDT52832B</i>	<i>Nordic nRF52</i>	External	NRF52832	64MHz	512K
<i>ST 32F3348DISCOVERY</i>	<i>ST STM32</i>	On-board	STM32F334C8T6	72MHz	64KB
<i>ST 32F401CDISCOVERY</i>	<i>ST STM32</i>	On-board	STM32F401VCT6	84MHz	256K
<i>ST 32F411EDISCOVERY</i>	<i>ST STM32</i>	On-board	STM32F411VET6	100MHz	512K
<i>ST 32F413HDISCOVERY</i>	<i>ST STM32</i>	On-board	STM32F413ZHT6	100MHz	512K
<i>ST 32F429IDISCOVERY</i>	<i>ST STM32</i>	On-board	STM32F429ZIT6	180MHz	2MB
<i>ST 32F469IDISCOVERY</i>	<i>ST STM32</i>	On-board	STM32F469NIH6	180MHz	1MB
<i>ST 32F746GDISCOVERY</i>	<i>ST STM32</i>	On-board	STM32F746NGH6	216MHz	1MB
<i>ST 32F769IDISCOVERY</i>	<i>ST STM32</i>	On-board	STM32F769NIH6	216MHz	1MB
<i>ST 32L0538DISCOVERY</i>	<i>ST STM32</i>	On-board	STM32L053C8T6	32MHz	64KB
<i>ST 32L100DISCOVERY</i>	<i>ST STM32</i>	On-board	STM32L100RCT6	32MHz	256K
<i>ST 32L476GDISCOVERY</i>	<i>ST STM32</i>	On-board	STM32L476VGT6	80MHz	1MB
<i>ST 32L496GDISCOVERY</i>	<i>ST STM32</i>	On-board	STM32L496AGI6	80MHz	1MB
<i>ST DISCO-L072CZ-LRWAN1</i>	<i>ST STM32</i>	On-board	STM32L072CZ	32MHz	192K
<i>ST DISCO-L475VG-IOT01A</i>	<i>ST STM32</i>	On-board	STM32L475VGT6	80MHz	1MB
<i>ST Discovery F072RB</i>	<i>ST STM32</i>	On-board	STM32F072RBT6	48MHz	128K
<i>ST Nucleo F030R8</i>	<i>ST STM32</i>	On-board	STM32F030R8T6	48MHz	64KB
<i>ST Nucleo F031K6</i>	<i>ST STM32</i>	On-board	STM32F031K6T6	48MHz	32KB
<i>ST Nucleo F042K6</i>	<i>ST STM32</i>	On-board	STM32F042K6T6	48MHz	32KB
<i>ST Nucleo F070RB</i>	<i>ST STM32</i>	On-board	STM32F070RBT6	48MHz	128K
<i>ST Nucleo F072RB</i>	<i>ST STM32</i>	On-board	STM32F072RBT6	48MHz	128K
<i>ST Nucleo F091RC</i>	<i>ST STM32</i>	On-board	STM32F091RCT6	48MHz	256K
<i>ST Nucleo F103RB</i>	<i>ST STM32</i>	On-board	STM32F103RBT6	72MHz	128K
<i>ST Nucleo F207ZG</i>	<i>ST STM32</i>	On-board	STM32F207ZGT6	120MHz	1MB
<i>ST Nucleo F302R8</i>	<i>ST STM32</i>	On-board	STM32F302R8T6	72MHz	64KB
<i>ST Nucleo F303K8</i>	<i>ST STM32</i>	On-board	STM32F303K8T6	72MHz	64KB
<i>ST Nucleo F303RE</i>	<i>ST STM32</i>	On-board	STM32F303RET6	72MHz	512K
<i>ST Nucleo F303ZE</i>	<i>ST STM32</i>	On-board	STM32F303ZET6	72MHz	512K
<i>ST Nucleo F334R8</i>	<i>ST STM32</i>	On-board	STM32F334R8T6	72MHz	64KB
<i>ST Nucleo F401RE</i>	<i>ST STM32</i>	On-board	STM32F401RET6	84MHz	512K
<i>ST Nucleo F410RB</i>	<i>ST STM32</i>	On-board	STM32F410RBT6	100MHz	128K
<i>ST Nucleo F411RE</i>	<i>ST STM32</i>	On-board	STM32F411RET6	100MHz	512K
<i>ST Nucleo F412ZG</i>	<i>ST STM32</i>	On-board	STM32F412ZGT6	100MHz	1MB
<i>ST Nucleo F413ZH</i>	<i>ST STM32</i>	On-board	STM32F413ZHT6	100MHz	512K
<i>ST Nucleo F429ZI</i>	<i>ST STM32</i>	On-board	STM32F429ZIT6	180MHz	2MB
<i>ST Nucleo F439ZI</i>	<i>ST STM32</i>	On-board	STM32F439ZIT6	180MHz	2MB
<i>ST Nucleo F446RE</i>	<i>ST STM32</i>	On-board	STM32F446RET6	180MHz	512K
<i>ST Nucleo F446ZE</i>	<i>ST STM32</i>	On-board	STM32F446ZET6	180MHz	512K
<i>ST Nucleo F722ZE</i>	<i>ST STM32</i>	On-board	STM32F722ZET6	216MHz	512K
<i>ST Nucleo F746ZG</i>	<i>ST STM32</i>	On-board	STM32F746ZGT6	216MHz	1MB
<i>ST Nucleo F756ZG</i>	<i>ST STM32</i>	On-board	STM32F756ZG	216MHz	1MB
<i>ST Nucleo F767ZI</i>	<i>ST STM32</i>	On-board	STM32F767ZIT6	216MHz	2MB
<i>ST Nucleo H743ZI</i>	<i>ST STM32</i>	On-board	STM32H743ZIT6	400MHz	2MB
<i>ST Nucleo L011K4</i>	<i>ST STM32</i>	On-board	STM32L011K4T6	32MHz	16KB
<i>ST Nucleo L031K6</i>	<i>ST STM32</i>	On-board	STM32L031K6T6	32MHz	32KB
<i>ST Nucleo L053R8</i>	<i>ST STM32</i>	On-board	STM32L053R8T6	32MHz	64KB
<i>ST Nucleo L073RZ</i>	<i>ST STM32</i>	On-board	STM32L073RZ	32MHz	192K

Continu

Table 38 – continued from previous page

Name	Platform	Debug	MCU	Frequency	Flash
<i>ST Nucleo L152RE</i>	<i>ST STM32</i>	On-board	STM32L152RET6	32MHz	512K
<i>ST Nucleo L412KB</i>	<i>ST STM32</i>	On-board	STM32L412KBU6	80MHz	128K
<i>ST Nucleo L432KC</i>	<i>ST STM32</i>	On-board	STM32L432KCU6	80MHz	256K
<i>ST Nucleo L433RC-P</i>	<i>ST STM32</i>	On-board	STM32L433RC	80MHz	256K
<i>ST Nucleo L452RE</i>	<i>ST STM32</i>	On-board	STM32L452RET6	80MHz	256K
<i>ST Nucleo L476RG</i>	<i>ST STM32</i>	On-board	STM32L476RGT6	80MHz	1MB
<i>ST Nucleo L486RG</i>	<i>ST STM32</i>	On-board	STM32L486RGT6	80MHz	1MB
<i>ST Nucleo L496ZG</i>	<i>ST STM32</i>	On-board	STM32L496ZGT6	80MHz	1MB
<i>ST Nucleo L496ZG-P</i>	<i>ST STM32</i>	On-board	STM32L496ZGT6P	80MHz	1MB
<i>ST Nucleo L4R5ZI</i>	<i>ST STM32</i>	On-board	STM32L4R5ZIT6	120MHz	2MB
<i>ST STM32F0308DISCOVERY</i>	<i>ST STM32</i>	On-board	STM32F030R8T6	48MHz	64KB
<i>ST STM32F0DISCOVERY</i>	<i>ST STM32</i>	On-board	STM32F051R8T6	48MHz	64KB
<i>ST STM32F3DISCOVERY</i>	<i>ST STM32</i>	On-board	STM32F303VCT6	72MHz	256K
<i>ST STM32F4DISCOVERY</i>	<i>ST STM32</i>	On-board	STM32F407VGT6	168MHz	1MB
<i>ST STM32L073Z-EVAL</i>	<i>ST STM32</i>	On-board	STM32L073VZT6	32MHz	192K
<i>ST STM32LDISCOVERY</i>	<i>ST STM32</i>	On-board	STM32L152RBT6	32MHz	128K
<i>ST STM32VLDISCOVERY</i>	<i>ST STM32</i>	On-board	STM32F100RBT6	24MHz	128K
<i>ST STM8S-DISCOVERY</i>	<i>ST STM8</i>	On-board	STM8S105C6T6	16MHz	32KB
<i>ST Sensor Node</i>	<i>ST STM32</i>	On-board	STM32L476JG	80MHz	1MB
<i>STM32F103C8 (20k RAM, 64k Flash)</i>	<i>ST STM32</i>	External	STM32F103C8T6	72MHz	64KB
<i>STM32F103CB (20k RAM, 128k Flash)</i>	<i>ST STM32</i>	External	STM32F103CBT6	72MHz	128K
<i>STM32F103R8 (20k RAM, 64 Flash)</i>	<i>ST STM32</i>	External	STM32F103R8T6	72MHz	64KB
<i>STM32F103RB (20k RAM, 128k Flash)</i>	<i>ST STM32</i>	External	STM32F103RBT6	72MHz	128K
<i>STM32F103RC (48k RAM, 256k Flash)</i>	<i>ST STM32</i>	External	STM32F103RCT6	72MHz	256K
<i>STM32F103RE (64k RAM, 512k Flash)</i>	<i>ST STM32</i>	External	STM32F103RET6	72MHz	512K
<i>STM32F103T8 (20k RAM, 64k Flash)</i>	<i>ST STM32</i>	External	STM32F103T8T6	72MHz	20KB
<i>STM32F103TB (20k RAM, 128k Flash)</i>	<i>ST STM32</i>	External	STM32F103TBT6	72MHz	128K
<i>STM32F103VB (20k RAM, 128k Flash)</i>	<i>ST STM32</i>	External	STM32F103VBT6	72MHz	128K
<i>STM32F103VC (48k RAM, 256k Flash)</i>	<i>ST STM32</i>	External	STM32F103VCT6	72MHz	256K
<i>STM32F103VD (64k RAM, 384k Flash)</i>	<i>ST STM32</i>	External	STM32F103VDT6	72MHz	384K
<i>STM32F103VE (64k RAM, 512k Flash)</i>	<i>ST STM32</i>	External	STM32F103VET6	72MHz	512K
<i>STM32F103ZC (48k RAM, 256k Flash)</i>	<i>ST STM32</i>	External	STM32F103ZCT6	72MHz	256K
<i>STM32F103ZD (64k RAM, 384k Flash)</i>	<i>ST STM32</i>	External	STM32F103ZDT6	72MHz	384K
<i>STM32F103ZE (64k RAM, 512k Flash)</i>	<i>ST STM32</i>	External	STM32F103ZET6	72MHz	512K
<i>STM32F303CB (32k RAM, 128k Flash)</i>	<i>ST STM32</i>	External	STM32F303CBT6	72MHz	128K
<i>STM32F407VE (192k RAM, 512k Flash)</i>	<i>ST STM32</i>	External	STM32F407VET6	168MHz	502.2
<i>STM32F407VG (192k RAM, 1024k Flash)</i>	<i>ST STM32</i>	External	STM32F407VGT6	168MHz	1MB
<i>STM32F4Stamp F405</i>	<i>ST STM32</i>	External	STM32F405RGT6	168MHz	1MB
<i>STM32F7508-DK</i>	<i>ST STM32</i>	On-board	STM32F750N8H6	216MHz	64KB
<i>SainSmart Due (Programming Port)</i>	<i>Atmel SAM</i>	External	AT91SAM3X8E	84MHz	512K
<i>SainSmart Due (USB Native Port)</i>	<i>Atmel SAM</i>	External	AT91SAM3X8E	84MHz	512K
<i>Seeed Arch BLE</i>	<i>Nordic nRF51</i>	On-board	NRF51822	16MHz	128K
<i>Seeed Arch Link</i>	<i>Nordic nRF51</i>	On-board	NRF51822	16MHz	256K
<i>Seeed Arch Max</i>	<i>ST STM32</i>	On-board	STM32F407VET6	168MHz	512K
<i>Seeed Tiny BLE</i>	<i>Nordic nRF51</i>	On-board	NRF51822	16MHz	256K
<i>Seeed Wio 3G</i>	<i>ST STM32</i>	On-board	STM32F439VI	180MHz	2MB
<i>Sino:Bit</i>	<i>Nordic nRF51</i>	External	NRF51822	32MHz	256K
<i>Sparky V1 F303</i>	<i>ST STM32</i>	External	STM32F303CCT6	72MHz	256K
<i>Taida Century nRF52 mini board</i>	<i>Nordic nRF52</i>	External	NRF52832	64MHz	512K

Continu

Table 38 – continued from previous page

Name	Platform	Debug	MCU	Frequency	Flash
Tiny STM103T	ST STM32	External	STM32F103TBU6	72MHz	128K
VAkE v1.0	ST STM32	External	STM32F446RET6	180MHz	512K
Waveshare BLE400	Nordic nRF51	External	NRF51822	32MHz	256K
hackaBLE	Nordic nRF52	External	NRF52832	64MHz	512K
ng-beacon	Nordic nRF51	External	NRF51822	16MHz	256K
sakura.io Evaluation Board	ST STM32	On-board	STM32F411RET6	100MHz	1MB
u-blox C030-N211 IoT Starter Kit	ST STM32	External	STM32F437VG	180MHz	1MB
u-blox C030-R410M IoT	ST STM32	On-board	STM32F437VG	180MHz	1MB
u-blox C030-U201 IoT Starter Kit	ST STM32	External	STM32F437VG	180MHz	1MB
u-blox EVK-NINA-B1	Nordic nRF52	On-board	NRF52832	64MHz	512K
u-blox EVK-ODIN-W2	ST STM32	External	STM32F439ZIY6	168MHz	2MB
u-blox ODIN-W2	ST STM32	External	STM32F439ZIY6	168MHz	2MB
y5 nRF51822 mbug	Nordic nRF51	On-board	NRF51822	16MHz	256K

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):

```
[env:myenv]
platform = ...
board = ...
debug_tool = ti-icdi
```

If you would like to use this tool for firmware uploading, please change upload protocol:

```
[env:myenv]
platform = ...
board = ...
debug_tool = ti-icdi
upload_protocol = ti-icdi
```

More options:

- *Debugging options*

- *Upload options*

Drivers

Windows Please “USB Driver Installation” guide for your board.

Mac Not required.

Linux Please install “udev” rules [99-platformio-udev.rules](#). If you already installed them before, please check that your rules are up-to-date or repeat steps.

Platforms

Name	Description
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.

Frameworks

Name	Description
Ardu- ino	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
li- bOpenCM3	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.

Boards

Note: For more detailed board information please scroll tables below by horizontal.

Name	Plat- form	Debug	MCU	Fre- quency	Flash	RAM
TI LaunchPad (Stellaris) w/ lm4f120 (80MHz)	TI TIVA	On- board	LPLM4F120H5QR	80MHz	256KB	32KB
TI LaunchPad (Tiva C) w/ tm4c123 (80MHz)	TI TIVA	On- board	LPTM4C1230C3PM	80MHz	256KB	32KB
TI LaunchPad (Tiva C) w/ tm4c129 (120MHz)	TI TIVA	On- board	LPTM4C1294NCPDTI20MHz		1MB	256KB

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 FTDI'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*

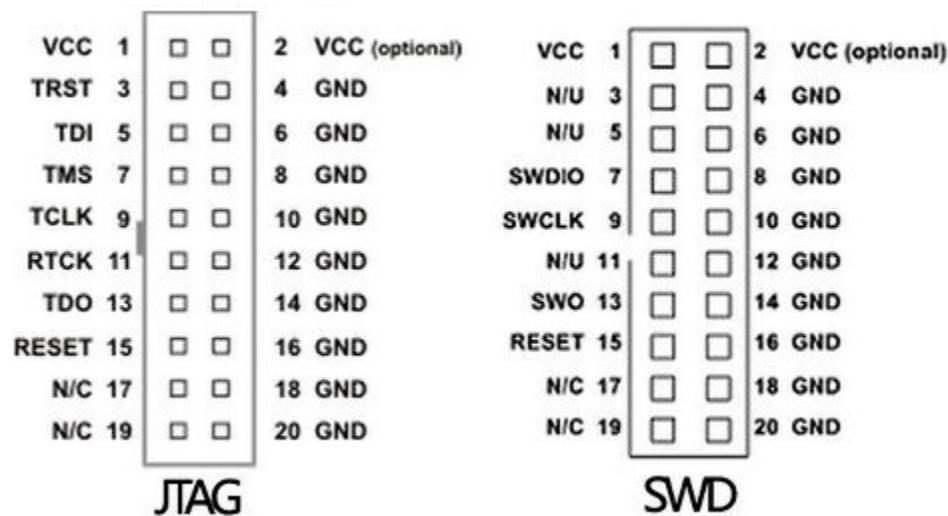
Drivers

Windows See <https://community.platformio.org/t/esp32-pio-unified-debugger/4541/20>

Mac macOS contains default FT232R USB Serial Driver 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

TUMPA JTAG 20-Pin Connector	Board JTAG Pin	Description
1	VCC	Positive Supply Voltage — Power supply for JTAG interface drivers
4	GND	Digital ground
5	TDI	Test Data In pin
7	TMS	Test Mode State pin
9	TCK	JTAG Return Test Clock
13	TDO	Test Data Out pin
15	RESET	Connect this pin to the (active low) reset input of the target CPU (EN for ESP32)

Serial Wire Mode Interface (SWD)

TUMPA SWD 20-Pin Connector	Board SWD Pin	Description
1	VCC	Positive Supply Voltage — Power supply for JTAG interface drivers
4	GND	Digital ground
7	SWDIO	Data I/O
9	SWCLK	Clock
15	RESET	Connect this pin to the (active low) reset input of the target CPU

Platforms

Name	Description
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.
Kendryte K210	Kendryte K210 is an AI capable RISC-V64 dual core SoC.
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.

Frameworks

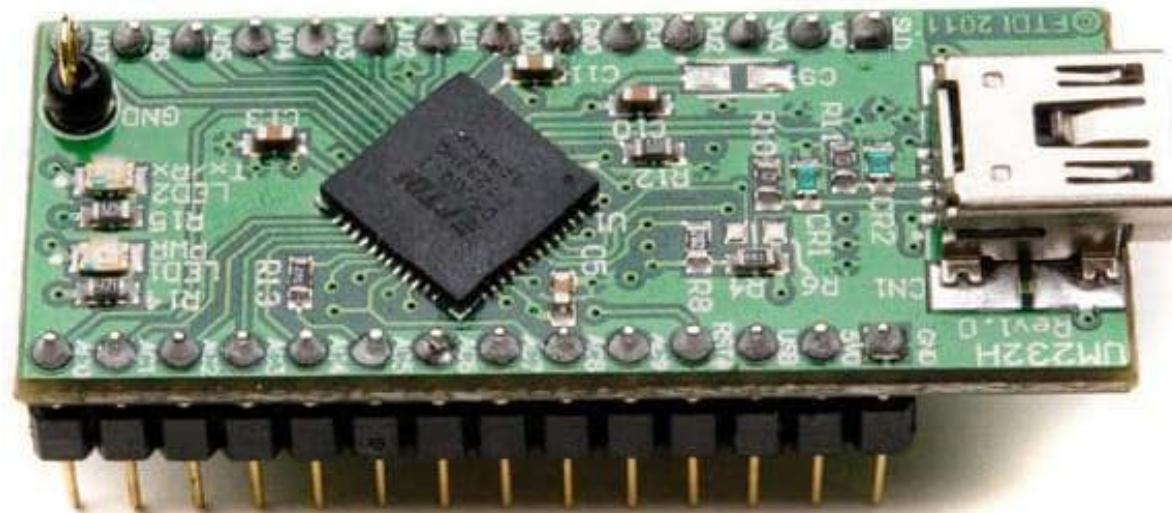
Name	Description
Arduino	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
ESP-IDF	Espressif IoT Development Framework. Official development framework for ESP32.
Freedom E SDK	Open Source Software for Developing on the SiFive Freedom E Platform
Kendryte Standalone SDK	Kendryte Standalone SDK without OS support
Kendryte FreeRTOS SDK	Kendryte SDK with FreeRTOS support
Simba	Simba is an RTOS and build framework. It aims to make embedded programming easy and portable.

Boards

Note: For more detailed board information please scroll tables below by horizontal.

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>AI Thinker ESP32-CAM</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>ALKS ESP32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>Adafruit ESP32 Feather</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>Arty FPGA Dev Kit</i>	<i>SiFive</i>	On-board	FE310	450MHz	16MB	256MB
<i>D-duino-32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>DOIT ESP32 DEVKIT V1</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>Dongsen Tech Pocket 32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>ESP32 FM DevKit</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>ESP32vn IoT Uno</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>ESPectro32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>ESPino32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>Espressif ESP-WROVER-KIT</i>	<i>Espressif 32</i>	On-board	ESP32	240MHz	4MB	320KB
<i>Espressif ESP32 Dev Module</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>FireBeetle-ESP32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>Frog Board ESP32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>Heltec WiFi LoRa 32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>Heltec WiFi LoRa 32 (V2)</i>	<i>Espressif 32</i>	External	ESP32	240MHz	8MB	320KB
<i>Heltec Wireless Stick</i>	<i>Espressif 32</i>	External	ESP32	240MHz	8MB	320KB
<i>Hornbill ESP32 Dev</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>Hornbill ESP32 Minima</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>IoTaaP Magnolia</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>MH ET LIVE ESP32DevKIT</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>MH ET LIVE ESP32MiniKit</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>Node32s</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>NodeMCU-32S</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>OLIMEX ESP32-DevKit-LiPo</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>OLIMEX ESP32-EVB</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>OLIMEX ESP32-GATEWAY</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>Pycom LoPy</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>Pycom LoPy4</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	1.25MB
<i>Silicognition wESP32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>Sipeed MAIX BiT</i>	<i>Kendryte K210</i>	External	K210	400MHz	16MB	6MB
<i>Sipeed MAIX BiT with Mic</i>	<i>Kendryte K210</i>	External	K210	400MHz	16MB	6MB
<i>Sipeed MAIX GO</i>	<i>Kendryte K210</i>	External	K210	400MHz	16MB	6MB
<i>Sipeed MAIX ONE DOCK</i>	<i>Kendryte K210</i>	External	K210	400MHz	16MB	6MB
<i>Sipeed MAIXDUINO</i>	<i>Kendryte K210</i>	External	K210	400MHz	16MB	6MB
<i>SparkFun ESP32 Thing</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>SparkFun LoRa Gateway 1-Channel</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>TTGO LoRa32-OLED V1</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>TTGO T-Beam</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	1.25MB
<i>TTGO T1</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>VintLabs ESP32 Devkit</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>WEMOS LOLIN D32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>WEMOS LOLIN D32 PRO</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>WEMOS LOLIN32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>WeMos D1 MINI ESP32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>WeMos WiFi and Bluetooth Battery</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>XinaBox CW02</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>oddWires IoT-Bus Io</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>oddWires IoT-Bus Proteus</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB

UM232H



The UM232H is a USB-to-serial/FIFO development module in the FTDI product range which utilizes the FT232H USB Hi-Speed (480Mb/s) single-port bridge chip to handle the USB signaling and protocols. Official reference can be found [here](#)

Contents

- [Configuration](#)
- [Drivers](#)
- [Wiring Connections](#)
- [Platforms](#)
- [Frameworks](#)
- [Boards](#)

Configuration

You can configure debugging tool using `debug_tool` option in “`platformio.ini`” (Project Configuration File):

```
[env:myenv]
platform = ...
board = ...
debug_tool = um232h
```

If you would like to use this tool for firmware uploading, please change upload protocol:

```
[env:myenv]
platform = ...
board = ...
debug_tool = um232h
upload_protocol = um232h
```

More options:

- *Debugging options*
- *Upload options*

Drivers

Windows See <https://community.platformio.org/t/esp32-pio-unified-debugger/4541/20>

Mac macOS contains default FT232USBSerialDriver driver which conflicts with debug tools which are based on this chip. FTDI Chip company recommends removing this default driver from a system. Everything should work after system rebooting. See detailed instruction in official application note (Page 16, Section 4: Uninstalling FTDI Drivers on OS X) [AN134: FTDI Drivers Installation guide for MAC OS X](#)

Linux Please install “udev” rules [99-platformio-udev.rules](#). If you already installed them before, please check that your rules are up-to-date or repeat steps.

Wiring Connections

Please read [4. UM232H Pin Out and Signal Descriptions](#) section for details.

UM232H Pin	Board Pin	JTAG Pin	Description
GND	GND		Digital ground
AD0	TCK		JTAG Return Test Clock
AD1	TDI		Test Data In
AD2	TDO		Test Data Out
AD3	TMS		Test Mode State
RST#	RESET		Connect this pin to the (active low) reset input of the target CPU (EN for ESP32)

You will also need to connect VIO to V3V and USB to 5V0 of UM232H to power the FTDI chip and board. See [UM232H Datasheet](#)

Platforms

Name	Description
GigaDevice GD32V	The GigaDevice GD32V device is a 32-bit general-purpose microcontroller based on the RISC-V core with an impressive balance of processing power, reduced power consumption and peripheral set.

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.
GigaDevice GD32V SDK	GigaDevice GD32VF103 Firmware Library (SDK)

Boards

Note: For more detailed board information please scroll tables below by horizontal.

Name	Platform	Debug	MCU	Frequency	Flash	RAM
GD32VF103V-EVAL	GigaDevice GD32V	External	GD32VF103VBT6	108MHz	128KB	32KB
Sipeed Longan Nano	GigaDevice GD32V	External	GD32VF103CBT6	108MHz	128KB	32KB
Wio Lite RISC-V	GigaDevice GD32V	External	GD32VF103CBT6	108MHz	128KB	32KB

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`)

```
[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](#).

```
[env:debug_jlink]
platform = ststm32
framework = mbed
board = nucleo_f446re
debug_tool = custom
debug_server =
    /full/path/to/JLinkGDBServerCL
    -singlerun
    -if
    SWD
    -select
    USB
    -port
    2331
    -device
    STM32F446RE
```

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
MK20DX256xxx7
```

extra_script.py

Place this file on the same level as “*platformio.ini*” (*Project Configuration File*).

```
from os import makedirs
from os.path import isdir, join
Import('env')

# Optional block, only for Teensy
env.AddPostAction(
    "$BUILD_DIR/firmware.hex",
    env.VerboseAction(" ".join([
        "sed", "-i.bak",
        "<",
        "s/:10040000FFFFFFFFFFFFFFFDEF9FFF23/:10040000FFFFFFFFFFFFFFFEFFFF/",
        ">",
        "$BUILD_DIR/firmware.hex"
    ]), "Fixing $BUILD_DIR/firmware.hex secure flash flags"))

def _jlink_cmd_script(env, source):
    build_dir = env.subst("$BUILD_DIR")
    if not isdir(build_dir):
        makedirs(build_dir)
    script_path = join(build_dir, "upload.jlink")
    commands = ["h", "loadbin %s,0x0" % source, "r", "q"]
    with open(script_path, "w") as fp:
        fp.write("\n".join(commands))
    return script_path

env.Replace(
    __jlink_cmd_script=_jlink_cmd_script,
    UPLOADER="/full/path/to/JLink",
    UPLOADERFLAGS=[
        "-device", "MK20DX256xxx7",
        "-speed", "4000",
        "-if", "swd",
        "-autoconnect", "1"])
```

(continues on next page)

(continued from previous page)

```
],
↳ UPLOADCMD='"$UPLOADER" $UPLOADERFLAGS -CommanderScript ${__jlink_cmd_script(__env__, SOURCE)}'
)
```

ST-Util and ST-Link

On-board ST-Link V2/V2-1 in pair with ST-Util GDB Server:

```
[env:debug]
platform = ststm32
framework = mbed
board = ...
debug_tool = custom
debug_port = :4242
debug_server = ${PLATFORMIO_CORE_DIR}/packages/tool-stlink/bin/st-util
```

OpenOCD and ST-Link

On-board ST-Link V2/V2-1 in pair with OpenOCD GDB Server:

```
[env:debug]
platform = ststm32
framework = mbed
board = ...
debug_tool = custom
debug_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.

```
[env:debug]
platform = ...
board = ...
framework = mbed
debug_tool = custom
debug_server = pyocd-gdbserver
```


1.17.4 CLI Guide

1.17.5 Platforms

Name	Description
Aceinna IMU	Open-source, embedded development platform for Aceinna IMU hardware. Run custom algorithms and navigation code on Aceinna IMU/INS hardware.
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.
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.
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.
GigaDevice GD32V	The GigaDevice GD32V device is a 32-bit general-purpose microcontroller based on the RISC-V core with an impressive balance of processing power, reduced power consumption and peripheral set.
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
Kendryte K210	Kendryte K210 is an AI capable RISCV64 dual core SoC.
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.
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.
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.
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.
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.
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.
Shakti	Shakti is an open-source initiative by the RISE group at IIT-Madras, which is not only building open source, production grade processors, but also associated components like interconnect fabrics, verification tools, storage controllers, peripheral IPs and SOC tools.
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.
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
1.17.5 PIO Unified Debugger	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.
ST STM32	The STM32 family of 32-bit Flash MCUs based on the ARM Cortex-M processor is designed to offer new degrees of freedom to MCU users. It offers a 32-bit product range that combines very high performance, real-time capabilities, digital signal processing, and low-power, low-voltage operation, while maintaining

1.17.6 Frameworks

Name	Description
Ar-duino	Arduino Wiring-based Framework allows writing cross-platform software to control devices attached to a wide range of Arduino boards to create all kinds of creative coding, interactive objects, spaces or physical experiences.
CM-SIS	The ARM Cortex Microcontroller Software Interface Standard (CMSIS) is a vendor-independent hardware abstraction layer for the Cortex-M processor series and specifies debugger interfaces. The CMSIS enables consistent and simple software interfaces to the processor for interface peripherals, real-time operating systems, and middleware. It simplifies software re-use, reducing the learning curve for new microcontroller developers and cutting the time-to-market for devices.
ESP-IDF	Espressif IoT Development Framework. Official development framework for ESP32.
Free-dom E SDK	Open Source Software for Developing on the SiFive Freedom E Platform
Gi-gaDevice GD32V SDK	GigaDevice GD32VF103 Firmware Library (SDK)
Kendryte Stanalone SDK	Kendryte Standalone SDK without OS support
Kendryte FreeRTOS SDK	Kendryte SDK with FreeRTOS support
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.
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.
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.
Shakti SDK	A software development kit for developing applications on Shakti class of processors
Simba	Simba is an RTOS and build framework. It aims to make embedded programming easy and portable.
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.
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.
Tizen RT	Tizen RT is a lightweight RTOS-based platform to support low-end IoT devices

1.17.7 Boards

Note: For more detailed board information please scroll tables below by horizontal.

1BitSquared

Name	Platform	Debug	MCU	Frequency	Flash	RAM
1Bitsy	ST STM32	External	STM32F415RGT	168MHz	1MB	128KB

96Boards

Name	Platform	Debug	MCU	Frequency	Flash	RAM
96Boards B96B-F446VE	ST STM32	On-board	STM32F446VET6	168MHz	512KB	128KB

AI Thinker

Name	Platform	Debug	MCU	Frequency	Flash	RAM
AI Thinker ESP32-CAM	Espressif 32	External	ESP32	240MHz	4MB	320KB

Aceinna

Name	Platform	Debug	MCU	Frequency	Flash	RAM
Aceinna Low Cost RTK	Aceinna IMU	On-board	STM32F469NIH6	180MHz	1MB	384KB
Aceinna OpenIMU 300ZA	Aceinna IMU	External	STM32F405RG	120MHz	1MB	128KB
Aceinna OpenIMU 300ZA	Aceinna IMU	External	STM32F405RG	120MHz	1MB	128KB
Aceinna OpenIMU 330	Aceinna IMU	External	STM32L431CB	80MHz	128KB	64KB

Adafruit

Name	Platform	Debug	MCU	Frequency	Flash	RAM
Adafruit Bluefruit nRF52832 Feather	Nordic nRF52	On-board	NRF52832	64MHz	512KB	64KB
Adafruit Circuit Playground Express	Atmel SAM	External	SAMD21G18A	48MHz	256KB	32KB
Adafruit Crickit M0	Atmel SAM	External	SAMD21G18A	48MHz	256KB	32KB
Adafruit ESP32 Feather	Espressif 32	External	ESP32	240MHz	4MB	320KB
Adafruit Feather M0	Atmel SAM	External	SAMD21G18A	48MHz	256KB	32KB
Adafruit Feather M0 Express	Atmel SAM	External	SAMD21G18A	48MHz	256KB	32KB
Adafruit Feather M4 Express	Atmel SAM	External	SAMD51J19A	120MHz	512KB	192KB
Adafruit Feather nRF52840 Express	Nordic nRF52	On-board	NRF52840	64MHz	796KB	243KB
Adafruit Gemma M0	Atmel SAM	External	SAMD21E18A	48MHz	256KB	32KB
Adafruit Grand Central M4	Atmel SAM	External	SAMD51P20A	120MHz	1MB	256KB
Adafruit Hallowing M0	Atmel SAM	External	SAMD21G18A	48MHz	256KB	32KB
Adafruit Hallowing M4	Atmel SAM	External	SAMD51J19A	120MHz	496KB	192KB
Adafruit ItsyBitsy M0	Atmel SAM	External	SAMD21G18A	48MHz	256KB	32KB
Adafruit ItsyBitsy M4	Atmel SAM	External	SAMD51G19A	120MHz	512KB	192KB
Adafruit MONSTER M4SK	Atmel SAM	External	SAMD51J19A	120MHz	496KB	192KB
Adafruit Metro M0 Expresss	Atmel SAM	External	SAMD21G18A	48MHz	256KB	32KB
Adafruit Metro M4	Atmel SAM	External	SAMD51J19A	120MHz	512KB	192KB
Adafruit Metro M4 AirLift Lite	Atmel SAM	External	SAMD51J19A	120MHz	512KB	192KB
Adafruit PyGamer Advance M4	Atmel SAM	External	SAMD51J20A	120MHz	1MB	256KB
Adafruit PyGamer M4 Express	Atmel SAM	External	SAMD51J19A	120MHz	512KB	192KB
Adafruit PyPortal M4	Atmel SAM	External	SAMD51J20A	120MHz	1MB	256KB
Adafruit Trellis M4	Atmel SAM	External	SAMD51J19A	120MHz	512KB	192KB
Adafruit Trinket M0	Atmel SAM	External	SAMD21E18A	48MHz	256KB	32KB
Adafruit pIRkey	Atmel SAM	External	SAMD21E18A	48MHz	256KB	32KB
Adafruit pyBadge AirLift M4	Atmel SAM	External	SAMD51J20A	120MHz	1008KB	192KB
Adafruit pyBadge M4 Express	Atmel SAM	External	SAMD51J19A	120MHz	512KB	192KB
Circuit Playground Bluefruit	Nordic nRF52	External	NRF52840	64MHz	796KB	243KB
Metro nRF52840 Express	Nordic nRF52	On-board	NRF52840	64MHz	796KB	243KB

Aiyarafun

Name	Platform	Debug	MCU	Frequency	Flash	RAM
Node32s	Espressif 32	External	ESP32	240MHz	4MB	320KB

Arduino

Name	Platform	Debug	MCU	Fre- quency	Flash	RAM
<i>Arduino Due (Programming Port)</i>	<i>Atmel SAM</i>	External	AT91SAM3X8E	84MHz	512KB	96KB
<i>Arduino Due (USB Native Port)</i>	<i>Atmel SAM</i>	External	AT91SAM3X8E	84MHz	512KB	96KB
<i>Arduino M0</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz	256KB	32KB
<i>Arduino M0 Pro (Native USB Port)</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz	256KB	32KB
<i>Arduino M0 Pro (Programming/Debug Port)</i>	<i>Atmel SAM</i>	On-board	SAMD21G18A	48MHz	256KB	32KB
<i>Arduino MKR FOX 1200</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz	256KB	32KB
<i>Arduino MKR GSM 1400</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz	256KB	32KB
<i>Arduino MKR NB 1500</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz	256KB	32KB
<i>Arduino MKR WAN 1300</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz	256KB	32KB
<i>Arduino MKR WiFi 1010</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz	256KB	32KB
<i>Arduino MKR1000</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz	256KB	32KB
<i>Arduino MKRZERO</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz	256KB	32KB
<i>Arduino Tian</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz	256KB	32KB
<i>Arduino Zero (Programming/Debug Port)</i>	<i>Atmel SAM</i>	On-board	SAMD21G18A	48MHz	256KB	32KB
<i>Arduino Zero (USB Native Port)</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz	256KB	32KB
<i>MKR Vidor 4000</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz	256KB	32KB
<i>NANO 33 IoT</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz	256KB	32KB

Armed

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>3D Printer Controller</i>	<i>ST STM32</i>	External	STM32F407VET6	168MHz	512KB	192KB

Armstrap

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Armstrap Eagle 1024</i>	<i>ST STM32</i>	External	STM32F417VGT6	168MHz	1MB	192KB
<i>Armstrap Eagle 2048</i>	<i>ST STM32</i>	External	STM32F427VIT6	168MHz	1.99MB	256KB
<i>Armstrap Eagle 512</i>	<i>ST STM32</i>	External	STM32F407VET6	168MHz	512KB	192KB

Atmel

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Atmel ATSAMR21-XPRO</i>	<i>Atmel SAM</i>	On-board	SAMR21G18A	48MHz	256KB	32KB
<i>Atmel ATSAMW25-XPRO</i>	<i>Atmel SAM</i>	On-board	SAMD21G18A	48MHz	256KB	32KB
<i>Atmel SAMD21-XPRO</i>	<i>Atmel SAM</i>	On-board	SAMD21J18A	48MHz	256KB	32KB
<i>Atmel SAML21-XPRO-B</i>	<i>Atmel SAM</i>	On-board	SAML21J18B	48MHz	256KB	32KB

Avnet Silica

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>ST Sensor Node</i>	<i>ST STM32</i>	On-board	STM32L476JG	80MHz	1MB	128KB

BBC

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>BBC micro:bit</i>	<i>Nordic nRF51</i>	On-board	NRF51822	16MHz	256KB	16KB

BluzDK

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>BluzDK</i>	<i>Nordic nRF51</i>	External	NRF51822	32MHz	256KB	32KB

CQ Publishing

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>CQ Publishing TG-LPC11U35-501</i>	<i>NXP LPC</i>	External	LPC11U35	48MHz	64KB	10KB

Calliope

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Calliope mini</i>	<i>Nordic nRF51</i>	On-board	NRF51822	16MHz	256KB	16KB

DFRobot

Name	Platform	Debug	MCU	Frequency	Flash	RAM
FireBeetle-ESP32	Espressif 32	External	ESP32	240MHz	4MB	320KB

DOIT

Name	Platform	Debug	MCU	Frequency	Flash	RAM
DOIT ESP32 DEVKIT V1	Espressif 32	External	ESP32	240MHz	4MB	320KB

DSTIKE

Name	Platform	Debug	MCU	Frequency	Flash	RAM
D-duino-32	Espressif 32	External	ESP32	240MHz	4MB	320KB

Delta

Name	Platform	Debug	MCU	Frequency	Flash	RAM
Delta DFBM-NQ620	Nordic nRF52	On-board	NRF52832	64MHz	512KB	64KB
Delta DFCM-NNN40	Nordic nRF51	On-board	NRF51822	32MHz	256KB	32KB
Delta DFCM-NNN50	Nordic nRF51	On-board	NRF51822	32MHz	256KB	16KB

Digistump

Name	Platform	Debug	MCU	Frequency	Flash	RAM
Digistump DigiX	Atmel SAM	External	AT91SAM3X8E	84MHz	512KB	96KB

Diymore

Name	Platform	Debug	MCU	Frequency	Flash	RAM
F407VG	ST STM32	External	STM32F407VGT6	168MHz	512KB	128KB

Dongsen Technology

Name	Platform	Debug	MCU	Frequency	Flash	RAM
Dongsen Tech Pocket 32	Espressif 32	External	ESP32	240MHz	4MB	320KB

DycodeX

Name	Platform	Debug	MCU	Frequency	Flash	RAM
ESPectro32	Espressif 32	External	ESP32	240MHz	4MB	320KB

ESP32vn

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>ESP32vn IoT Uno</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB

Electronut Labs

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Bluey nRF52832 IoT</i>	<i>Nordic nRF52</i>	External	NRF52832	64MHz	512KB	64KB
<i>hackaBLE</i>	<i>Nordic nRF52</i>	External	NRF52832	64MHz	512KB	64KB

Elektor Labs

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>CoCo-ri-Co!</i>	<i>NXP LPC</i>	On-board	LPC812	30MHz	16KB	4KB

Embedded Artists

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>EA LPC11U35 QuickStart Board</i>	<i>NXP LPC</i>	External	LPC11U35	48MHz	64KB	10KB
<i>Embedded Artists LPC4088 Display Module</i>	<i>NXP LPC</i>	On-board	LPC4088	120MHz	512KB	96KB
<i>Embedded Artists LPC4088 QuickStart Board</i>	<i>NXP LPC</i>	On-board	LPC4088	120MHz	512KB	96KB

Espotel

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Espotel LoRa Module</i>	<i>ST STM32</i>	External	STM32F411RET6	100MHz	512KB	128KB

Espressif

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Espressif ESP-WROVER-KIT</i>	<i>Espressif 32</i>	On-board	ESP32	240MHz	4MB	320KB
<i>Espressif ESP32 Dev Module</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB

Fred

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Frog Board ESP32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB

Freescale

Name	Platform	Debug	MCU	Fre-quency	Flash	RAM
<i>Ethernet IoT Starter Kit</i>	<i>Freescale Kinetis</i>	On-board	MK64FN1M0VLL12	120MHz	1MB	256KB
<i>Freescale Kinetis FRDM-K20D50M</i>	<i>Freescale Kinetis</i>	On-board	MK20DX128VLH5	48MHz	128KB	16KB
<i>Freescale Kinetis FRDM-K22F</i>	<i>Freescale Kinetis</i>	On-board	MK22FN512VLH12	120MHz	512KB	128KB
<i>Freescale Kinetis FRDM-K64F</i>	<i>Freescale Kinetis</i>	On-board	MK64FN1M0VLL12	120MHz	1MB	256KB
<i>Freescale Kinetis FRDM-K66F</i>	<i>Freescale Kinetis</i>	On-board	MK66FN2M0VMD18I	80MHz	2MB	256KB
<i>Freescale Kinetis FRDM-K82F</i>	<i>Freescale Kinetis</i>	On-board	MK82FN256VLL15	150MHz	256KB	256KB
<i>Freescale Kinetis FRDM-KL05Z</i>	<i>Freescale Kinetis</i>	On-board	MKL05Z32VFM4	48MHz	32KB	4KB
<i>Freescale Kinetis FRDM-KL25Z</i>	<i>Freescale Kinetis</i>	On-board	MKL25Z128VLK4	48MHz	128KB	16KB
<i>Freescale Kinetis FRDM-KL27Z</i>	<i>Freescale Kinetis</i>	On-board	MKL27Z64VLH4	48MHz	64KB	16KB
<i>Freescale Kinetis FRDM-KL43Z</i>	<i>Freescale Kinetis</i>	On-board	MKL43Z256VLH4	48MHz	256KB	32KB
<i>Freescale Kinetis FRDM-KL46Z</i>	<i>Freescale Kinetis</i>	On-board	MKL46Z256VLL4	48MHz	256KB	32KB
<i>Freescale Kinetis FRDM-KL82Z</i>	<i>Freescale Kinetis</i>	External	MKL82Z128VLK7	96MHz	128KB	96KB
<i>Freescale Kinetis FRDM-KW24D512</i>	<i>Freescale Kinetis</i>	External	MKW24D512	50MHz	512KB	64KB
<i>Freescale Kinetis FRDM-KW41Z</i>	<i>Freescale Kinetis</i>	On-board	MKW41Z512VHT4	48MHz	512KB	128KB

Generic

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>BlackPill F103C8</i>	<i>ST STM32</i>	External	STM32F103C8T6	72MHz	64KB	20KB
<i>BlackPill F103C8 (128k)</i>	<i>ST STM32</i>	External	STM32F103C8T6	72MHz	128KB	20KB
<i>BluePill F103C6</i>	<i>ST STM32</i>	External	STM32F103C6T6	72MHz	32KB	10KB
<i>BluePill F103C8</i>	<i>ST STM32</i>	External	STM32F103C8T6	72MHz	64KB	20KB
<i>BluePill F103C8 (128k)</i>	<i>ST STM32</i>	External	STM32F103C8T6	72MHz	128KB	20KB
<i>Demo F030F4</i>	<i>ST STM32</i>	External	STM32F030F4P6	48MHz	16KB	4KB
<i>FK407M1</i>	<i>ST STM32</i>	External	STM32F407VET6	168MHz	512KB	128KB
<i>STM32F103C8 (20k RAM. 64k Flash)</i>	<i>ST STM32</i>	External	STM32F103C8T6	72MHz	64KB	20KB
<i>STM32F103CB (20k RAM. 128k Flash)</i>	<i>ST STM32</i>	External	STM32F103CBT6	72MHz	128KB	20KB
<i>STM32F103R8 (20k RAM. 64 Flash)</i>	<i>ST STM32</i>	External	STM32F103R8T6	72MHz	64KB	20KB
<i>STM32F103RB (20k RAM. 128k Flash)</i>	<i>ST STM32</i>	External	STM32F103RBT6	72MHz	128KB	20KB
<i>STM32F103RC (48k RAM. 256k Flash)</i>	<i>ST STM32</i>	External	STM32F103RCT6	72MHz	256KB	48KB
<i>STM32F103RE (64k RAM. 512k Flash)</i>	<i>ST STM32</i>	External	STM32F103RET6	72MHz	512KB	64KB
<i>STM32F103T8 (20k RAM. 64k Flash)</i>	<i>ST STM32</i>	External	STM32F103T8T6	72MHz	20KB	64KB
<i>STM32F103TB (20k RAM. 128k Flash)</i>	<i>ST STM32</i>	External	STM32F103TBT6	72MHz	128KB	20KB
<i>STM32F103VB (20k RAM. 128k Flash)</i>	<i>ST STM32</i>	External	STM32F103VBT6	72MHz	128KB	20KB
<i>STM32F103VC (48k RAM. 256k Flash)</i>	<i>ST STM32</i>	External	STM32F103VCT6	72MHz	256KB	48KB
<i>STM32F103VD (64k RAM. 384k Flash)</i>	<i>ST STM32</i>	External	STM32F103VDT6	72MHz	384KB	64KB
<i>STM32F103VE (64k RAM. 512k Flash)</i>	<i>ST STM32</i>	External	STM32F103VET6	72MHz	512KB	64KB
<i>STM32F103ZC (48k RAM. 256k Flash)</i>	<i>ST STM32</i>	External	STM32F103ZCT6	72MHz	256KB	48KB
<i>STM32F103ZD (64k RAM. 384k Flash)</i>	<i>ST STM32</i>	External	STM32F103ZDT6	72MHz	384KB	64KB
<i>STM32F103ZE (64k RAM. 512k Flash)</i>	<i>ST STM32</i>	External	STM32F103ZET6	72MHz	512KB	64KB
<i>STM32F303CB (32k RAM. 128k Flash)</i>	<i>ST STM32</i>	External	STM32F303CBT6	72MHz	128KB	32KB
<i>STM32F407VE (192k RAM. 512k Flash)</i>	<i>ST STM32</i>	External	STM32F407VET6	168MHz	502.23KB	128KB
<i>STM32F407VG (192k RAM. 1024k Flash)</i>	<i>ST STM32</i>	External	STM32F407VGT6	168MHz	1MB	192KB
1856	<i>ST STM32</i>	External	STM32F405RGT6	168MHz	1MB	192KB

Gimasi

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Tuino 096</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz	256KB	32KB

GreenWaves Technologies

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>GAPuino GAP8</i>	<i>RISC-V GAP</i>	On-board	GAP8	250MHz	64MB	8MB

HY

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Tiny STM103T</i>	<i>ST STM32</i>	External	STM32F103TBU6	72MHz	128KB	20KB

Heltec Automation

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Heltec WiFi LoRa 32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>Heltec WiFi LoRa 32 (V2)</i>	<i>Espressif 32</i>	External	ESP32	240MHz	8MB	320KB
<i>Heltec Wireless Stick</i>	<i>Espressif 32</i>	External	ESP32	240MHz	8MB	320KB

Hornbill

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Hornbill ESP32 Dev</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>Hornbill ESP32 Minima</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB

Infineon

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>XMC1100 Boot Kit</i>	<i>Infineon XMC</i>	On-board	XMC1100	32MHz	64KB	16KB
<i>XMC1100 H-Bridge 2Go</i>	<i>Infineon XMC</i>	On-board	XMC1100	32MHz	64KB	16KB
<i>XMC1100 XMC2Go</i>	<i>Infineon XMC</i>	On-board	XMC1100	32MHz	64KB	16KB
<i>XMC1300 Boot Kit</i>	<i>Infineon XMC</i>	On-board	XMC1300	32MHz	64KB	16KB
<i>XMC1300 Sense2GoL</i>	<i>Infineon XMC</i>	On-board	XMC1300	32MHz	32KB	16KB
<i>XMC1400 Boot Kit</i>	<i>Infineon XMC</i>	On-board	XMC1400	48MHz	1.95MB	16KB
<i>XMC4200 Distance2Go</i>	<i>Infineon XMC</i>	On-board	XMC4200	80MHz	256KB	40KB
<i>XMC4700 Relax Kit</i>	<i>Infineon XMC</i>	On-board	XMC4700	144MHz	2.00MB	1.95MB

JKSoft

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>JKSoft Wallbot BLE</i>	<i>Nordic nRF51</i>	On-board	NRF51822	16MHz	128KB	16KB

LeafLabs

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Maple</i>	<i>ST STM32</i>	External	STM32F103RBT6	72MHz	108KB	17KB
<i>Maple (RET6)</i>	<i>ST STM32</i>	External	STM32F103RET6	72MHz	256KB	48KB
<i>Maple Mini Bootloader 2.0</i>	<i>ST STM32</i>	External	STM32F103CBT6	72MHz	120KB	20KB
<i>Maple Mini Original</i>	<i>ST STM32</i>	External	STM32F103CBT6	72MHz	108KB	17KB

LowPowerLab

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Moteino M0</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz	256KB	32KB

MH-ET Live

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>MH ET LIVE ESP32DevKIT</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>MH ET LIVE ESP32MiniKit</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB

MVT Solutions

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>IoTaaP Magnolia</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB

MXChip

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Microsoft Azure IoT Development Kit (MX-Chip AZ3166)</i>	<i>ST STM32</i>	On-board	STM32F412ZGT	140MHz	1MB	256KB

Macchina

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Macchina M2</i>	<i>Atmel SAM</i>	External	AT91SAM3X8E	84MHz	512KB	96KB

Malyan

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>M200 V2</i>	<i>ST STM32</i>	External	STM32F070CBT6	48MHz	120KB	14.81KB
<i>Malyan M200 V1</i>	<i>ST STM32</i>	External	STM32F103CBT6	72MHz	120KB	20KB

Maxim

Name	Platform	De- bug	MCU	Fre- quency	Flash	RAM
<i>MAX32620FTHR</i>	<i>Maxim 32</i>	Exter- nal	MAX32620FT	96MHz	2MB	256KB
<i>Maxim ARM mbed Enabled Development Platform for MAX32600</i>	<i>Maxim 32</i>	On- board	MAX32600	24MHz	256KB	32KB
<i>Maxim Health Sensor Platform</i>	<i>Maxim 32</i>	Exter- nal	MAX32620	96MHz	2MB	256KB
<i>Maxim Wireless Sensor Node Demonstrator</i>	<i>Maxim 32</i>	Exter- nal	MAX32610	24MHz	256KB	32KB

Microduino

Name	Platform	Debug	MCU	Fre- quency	Flash	RAM
<i>Microduino Core STM32 to Flash</i>	<i>ST STM32</i>	Exter- nal	STM32F103CBT6	72MHz	105.47KB	16.60KB

Micromint

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Bambino-210E</i>	<i>NXP LPC</i>	On-board	LPC4330	204MHz	8MB	264KB

MikroElektronika

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Hexiwear</i>	<i>Freescale Kinetis</i>	External	MK64FN1M0VDC12	120MHz	1MB	256KB

MultiTech

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>MTS Dragonfly</i>	<i>ST STM32</i>	External	STM32F411RET6	100MHz	512KB	128KB
<i>MultiTech mDot</i>	<i>ST STM32</i>	External	STM32F411RET6	100MHz	512KB	128KB
<i>MultiTech mDot F411</i>	<i>ST STM32</i>	External	STM32F411RET6	100MHz	512KB	128KB
<i>MultiTech xDot</i>	<i>ST STM32</i>	External	STM32L151CCU6	32MHz	256KB	32KB

NGX Technologies

Name	Platform	Debug	MCU	Frequency	Flash	RAM
NGX Technologies BlueBoard-LPC11U24	NXP LPC	External	LPC11U24	48MHz	32KB	8KB

NXP

Name	Platform	Debug	MCU	Fre-quency	Flash	RAM
ARM mbed LPC11U24 (+CAN)	NXP LPC	On-board	LPC11U24	48MHz	32KB	8KB
LPCXpresso11U68	NXP LPC	On-board	LPC11U68	50MHz	256KB	36KB
LPCXpresso824-MAX	NXP LPC	On-board	LPC824	30MHz	32KB	8KB
NXP LPC11C24	NXP LPC	External	LPC11C24	48MHz	32KB	8KB
NXP LPC11U34	NXP LPC	External	LPC11U34	48MHz	40KB	8KB
NXP LPC11U37	NXP LPC	External	LPC11U37	48MHz	128KB	10KB
NXP LPC800-MAX	NXP LPC	On-board	LPC812	30MHz	16KB	4KB
NXP LPCXpresso1549	NXP LPC	External	LPC1549	72MHz	256KB	36KB
NXP LPCXpresso54114	NXP LPC	On-board	LPC54114J256BD64	100MHz	256KB	192KB
NXP LPCXpresso54608	NXP LPC	On-board	LPC54608ET512	180MHz	512KB	200KB
NXP mbed LPC11U24	NXP LPC	On-board	LPC11U24	48MHz	32KB	8KB
NXP mbed LPC1768	NXP LPC	On-board	LPC1768	96MHz	512KB	64KB

Netduino

Name	Platform	Debug	MCU	Frequency	Flash	RAM
N2+	ST STM32	External	STM32F405RGT6	168MHz	1MB	192KB

NodeMCU

Name	Platform	Debug	MCU	Frequency	Flash	RAM
NodeMCU-32S	Espressif 32	External	ESP32	240MHz	4MB	320KB

Nordic

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Nordic Beacon Kit (PCA20006)</i>	<i>Nordic nRF51</i>	On-board	NRF51822	32MHz	256KB	32KB
<i>Nordic nRF51 Dongle (PCA10031)</i>	<i>Nordic nRF51</i>	On-board	NRF51822	32MHz	256KB	32KB
<i>Nordic nRF51822-mKIT</i>	<i>Nordic nRF51</i>	On-board	NRF51822	16MHz	128KB	16KB
<i>Nordic nRF51X22 Development Kit(PCA1000X)</i>	<i>Nordic nRF51</i>	On-board	NRF51822	32MHz	256KB	32KB
<i>Nordic nRF52-DK</i>	<i>Nordic nRF52</i>	On-board	NRF52832	64MHz	512KB	64KB
<i>Nordic nRF52840-DK</i>	<i>Nordic nRF52</i>	On-board	NRF52840	64MHz	1MB	256KB
<i>Nordic nRF52840-DK (Adafruit BSP)</i>	<i>Nordic nRF52</i>	On-board	NRF52840	64MHz	796KB	243KB

OLIMEX

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>OLIMEX ESP32-DevKit-LiPo</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>OLIMEX ESP32-EVB</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>OLIMEX ESP32-GATEWAY</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB

OSHChip

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>OSHChip</i>	<i>Nordic nRF51</i>	External	NRF51822	32MHz	256KB	32KB

Particle

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Xenon</i>	<i>Nordic nRF52</i>	External	NRF52840	64MHz	796KB	243KB

Pycom Ltd.

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Pycom LoPy</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>Pycom LoPy4</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	1.25MB

RAK

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>RAK811 LoRa Tracker</i>	<i>ST STM32</i>	External	STM32L151RBT6	32MHz	128KB	16KB
<i>RAK811 LoRa Tracker</i>	<i>ST STM32</i>	External	STM32L151RBT6	32MHz	128KB	32KB

RUMBA

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>3D Printer control board</i>	<i>ST STM32</i>	External	STM32F446RET6	180MHz	512KB	128KB

RedBearLab

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>RedBearLab BLE Nano 1.5</i>	<i>Nordic nRF51</i>	On-board	NRF51822	16MHz	256KB	32KB
<i>RedBearLab BLE Nano 2</i>	<i>Nordic nRF52</i>	On-board	NRF52832	64MHz	512KB	64KB
<i>RedBearLab Blend 2</i>	<i>Nordic nRF52</i>	On-board	NRF52832	64MHz	512KB	64KB
<i>RedBearLab nRF51822</i>	<i>Nordic nRF51</i>	On-board	NRF51822	16MHz	256KB	16KB

RemRam

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>3D printer controller</i>	<i>ST STM32</i>	On-board	STM32F765VIT6	216MHz	2MB	512KB

ReprapWorld

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Minitronics v2.0</i>	<i>Atmel SAM</i>	External	SAMD21J18A	48MHz	256KB	32KB

RobotDyn

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>BlackPill F303CC</i>	<i>ST STM32</i>	External	STM32F303CCT6	72MHz	256KB	40KB

RoboticsBrno

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>ALKS ESP32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB

RushUp

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>RushUp Cloud-JAM</i>	<i>ST STM32</i>	On-board	STM32F401RET6	84MHz	512KB	96KB
<i>RushUp Cloud-JAM L4</i>	<i>ST STM32</i>	On-board	STM32L476RGT6	80MHz	1MB	128KB

SODAQ

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>SODAQ Autonomo</i>	<i>Atmel SAM</i>	External	SAMD21J18A	48MHz	256KB	32KB
<i>SODAQ ExpLoRer</i>	<i>Atmel SAM</i>	External	SAMD21J18A	48MHz	256KB	32KB
<i>SODAQ ONE</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz	256KB	32KB
<i>SODAQ SARA</i>	<i>Atmel SAM</i>	External	SAMD21J18A	48MHz	256KB	32KB
<i>SODAQ SFF</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz	256KB	32KB

ST

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>32F723EDISCOVERY</i>	<i>ST STM32</i>	On-board	STM32F723IEK6	216MHz	512KB	192KB
<i>3DP001V1 Evaluation board for 3D printer</i>	<i>ST STM32</i>	On-board	STM32F401VGT6	84MHz	512KB	96KB
<i>Black STM32F407VE</i>	<i>ST STM32</i>	External	STM32F407VET6	168MHz	512KB	128KB
<i>Black STM32F407VG</i>	<i>ST STM32</i>	External	STM32F407VGT6	168MHz	512KB	128KB
<i>Black STM32F407ZE</i>	<i>ST STM32</i>	External	STM32F407ZET6	168MHz	512KB	128KB
<i>Black STM32F407ZE</i>	<i>ST STM32</i>	External	STM32F407ZGT6	168MHz	1MB	128KB
<i>Blue STM32F407VE Mini</i>	<i>ST STM32</i>	External	STM32F407VET6	168MHz	512KB	128KB
<i>Nucleo G071RB</i>	<i>ST STM32</i>	On-board	STM32G071RBT6	24MHz	2MB	128KB
<i>P-Nucleo WB55RG</i>	<i>ST STM32</i>	On-board	STM32WB55RG	64MHz	512KB	192.00KB
<i>ST 32F3348DISCOVERY</i>	<i>ST STM32</i>	On-board	STM32F334C8T6	72MHz	64KB	12KB
<i>ST 32F401CDISCOVERY</i>	<i>ST STM32</i>	On-board	STM32F401VCT6	84MHz	256KB	64KB
<i>ST 32F411EDISCOVERY</i>	<i>ST STM32</i>	On-board	STM32F411VET6	100MHz	512KB	128KB
<i>ST 32F413HDISCOVERY</i>	<i>ST STM32</i>	On-board	STM32F413ZHT6	100MHz	512KB	128KB
<i>ST 32F429IDISCOVERY</i>	<i>ST STM32</i>	On-board	STM32F429ZIT6	180MHz	2MB	256KB
<i>ST 32F469IDISCOVERY</i>	<i>ST STM32</i>	On-board	STM32F469NIH6	180MHz	1MB	384KB
<i>ST 32F746GDISCOVERY</i>	<i>ST STM32</i>	On-board	STM32F746NGH6	216MHz	1MB	320KB
<i>ST 32F769IDISCOVERY</i>	<i>ST STM32</i>	On-board	STM32F769NIH6	216MHz	1MB	512KB
<i>ST 32L0538DISCOVERY</i>	<i>ST STM32</i>	On-board	STM32L053C8T6	32MHz	64KB	8KB
<i>ST 32L100DISCOVERY</i>	<i>ST STM32</i>	On-board	STM32L100RCT6	32MHz	256KB	16KB
<i>ST 32L476GDISCOVERY</i>	<i>ST STM32</i>	On-board	STM32L476VGT6	80MHz	1MB	128KB
<i>ST 32L496GDISCOVERY</i>	<i>ST STM32</i>	On-board	STM32L496AGI6	80MHz	1MB	320KB
<i>ST DISCO-L072CZ-LRWANI</i>	<i>ST STM32</i>	On-board	STM32L072CZ	32MHz	192KB	20KB
<i>ST DISCO-L475VG-IOT01A</i>	<i>ST STM32</i>	On-board	STM32L475VGT6	80MHz	1MB	128KB
<i>ST Discovery F072RB</i>	<i>ST STM32</i>	On-board	STM32F072RBT6	48MHz	128KB	16KB
<i>ST Nucleo F030R8</i>	<i>ST STM32</i>	On-board	STM32F030R8T6	48MHz	64KB	8KB
<i>ST Nucleo F031K6</i>	<i>ST STM32</i>	On-board	STM32F031K6T6	48MHz	32KB	4KB
<i>ST Nucleo F042K6</i>	<i>ST STM32</i>	On-board	STM32F042K6T6	48MHz	32KB	6KB
<i>ST Nucleo F070RB</i>	<i>ST STM32</i>	On-board	STM32F070RBT6	48MHz	128KB	16KB
<i>ST Nucleo F072RB</i>	<i>ST STM32</i>	On-board	STM32F072RBT6	48MHz	128KB	16KB

Continued on next page

Table 40 – continued from previous page

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>ST Nucleo F091RC</i>	<i>ST STM32</i>	On-board	STM32F091RCT6	48MHz	256KB	32KB
<i>ST Nucleo F103RB</i>	<i>ST STM32</i>	On-board	STM32F103RBT6	72MHz	128KB	20KB
<i>ST Nucleo F207ZG</i>	<i>ST STM32</i>	On-board	STM32F207ZGT6	120MHz	1MB	128KB
<i>ST Nucleo F302R8</i>	<i>ST STM32</i>	On-board	STM32F302R8T6	72MHz	64KB	16KB
<i>ST Nucleo F303K8</i>	<i>ST STM32</i>	On-board	STM32F303K8T6	72MHz	64KB	12KB
<i>ST Nucleo F303RE</i>	<i>ST STM32</i>	On-board	STM32F303RET6	72MHz	512KB	64KB
<i>ST Nucleo F303ZE</i>	<i>ST STM32</i>	On-board	STM32F303ZET6	72MHz	512KB	64KB
<i>ST Nucleo F334R8</i>	<i>ST STM32</i>	On-board	STM32F334R8T6	72MHz	64KB	16KB
<i>ST Nucleo F401RE</i>	<i>ST STM32</i>	On-board	STM32F401RET6	84MHz	512KB	96KB
<i>ST Nucleo F410RB</i>	<i>ST STM32</i>	On-board	STM32F410RBT6	100MHz	128KB	32KB
<i>ST Nucleo F411RE</i>	<i>ST STM32</i>	On-board	STM32F411RET6	100MHz	512KB	128KB
<i>ST Nucleo F412ZG</i>	<i>ST STM32</i>	On-board	STM32F412ZGT6	100MHz	1MB	256KB
<i>ST Nucleo F413ZH</i>	<i>ST STM32</i>	On-board	STM32F413ZHT6	100MHz	512KB	128KB
<i>ST Nucleo F429ZI</i>	<i>ST STM32</i>	On-board	STM32F429ZIT6	180MHz	2MB	256KB
<i>ST Nucleo F439ZI</i>	<i>ST STM32</i>	On-board	STM32F439ZIT6	180MHz	2MB	256KB
<i>ST Nucleo F446RE</i>	<i>ST STM32</i>	On-board	STM32F446RET6	180MHz	512KB	128KB
<i>ST Nucleo F446ZE</i>	<i>ST STM32</i>	On-board	STM32F446ZET6	180MHz	512KB	128KB
<i>ST Nucleo F722ZE</i>	<i>ST STM32</i>	On-board	STM32F722ZET6	216MHz	512KB	256KB
<i>ST Nucleo F746ZG</i>	<i>ST STM32</i>	On-board	STM32F746ZGT6	216MHz	1MB	320KB
<i>ST Nucleo F756ZG</i>	<i>ST STM32</i>	On-board	STM32F756ZG	216MHz	1MB	320KB
<i>ST Nucleo F767ZI</i>	<i>ST STM32</i>	On-board	STM32F767ZIT6	216MHz	2MB	512KB
<i>ST Nucleo H743ZI</i>	<i>ST STM32</i>	On-board	STM32H743ZIT6	400MHz	2MB	1MB
<i>ST Nucleo L011K4</i>	<i>ST STM32</i>	On-board	STM32L011K4T6	32MHz	16KB	2KB
<i>ST Nucleo L031K6</i>	<i>ST STM32</i>	On-board	STM32L031K6T6	32MHz	32KB	8KB
<i>ST Nucleo L053R8</i>	<i>ST STM32</i>	On-board	STM32L053R8T6	32MHz	64KB	8KB
<i>ST Nucleo L073RZ</i>	<i>ST STM32</i>	On-board	STM32L073RZ	32MHz	192KB	20KB
<i>ST Nucleo L152RE</i>	<i>ST STM32</i>	On-board	STM32L152RET6	32MHz	512KB	80KB
<i>ST Nucleo L412KB</i>	<i>ST STM32</i>	On-board	STM32L412KBU6	80MHz	128KB	40KB
<i>ST Nucleo L432KC</i>	<i>ST STM32</i>	On-board	STM32L432KCU6	80MHz	256KB	64KB
<i>ST Nucleo L433RC-P</i>	<i>ST STM32</i>	On-board	STM32L433RC	80MHz	256KB	64KB
<i>ST Nucleo L452RE</i>	<i>ST STM32</i>	On-board	STM32L452RET6	80MHz	256KB	64KB
<i>ST Nucleo L476RG</i>	<i>ST STM32</i>	On-board	STM32L476RGT6	80MHz	1MB	128KB
<i>ST Nucleo L486RG</i>	<i>ST STM32</i>	On-board	STM32L486RGT6	80MHz	1MB	128KB
<i>ST Nucleo L496ZG</i>	<i>ST STM32</i>	On-board	STM32L496ZGT6	80MHz	1MB	128KB
<i>ST Nucleo L496ZG-P</i>	<i>ST STM32</i>	On-board	STM32L496ZGT6P	80MHz	1MB	320KB
<i>ST Nucleo L4R5ZI</i>	<i>ST STM32</i>	On-board	STM32L4R5ZIT6	120MHz	2MB	640KB
<i>ST STM32F0308DISCOVERY</i>	<i>ST STM32</i>	On-board	STM32F030R8T6	48MHz	64KB	8KB
<i>ST STM32F0DISCOVERY</i>	<i>ST STM32</i>	On-board	STM32F051R8T6	48MHz	64KB	8KB
<i>ST STM32F3DISCOVERY</i>	<i>ST STM32</i>	On-board	STM32F303VCT6	72MHz	256KB	48KB
<i>ST STM32F4DISCOVERY</i>	<i>ST STM32</i>	On-board	STM32F407VGT6	168MHz	1MB	128KB
<i>ST STM32L073Z-EVAL</i>	<i>ST STM32</i>	On-board	STM32L073VZT6	32MHz	192KB	20KB
<i>ST STM32LDISCOVERY</i>	<i>ST STM32</i>	On-board	STM32L152RBT6	32MHz	128KB	16KB
<i>ST STM32VLDISCOVERY</i>	<i>ST STM32</i>	On-board	STM32F100RBT6	24MHz	128KB	8KB
<i>ST STM8S-DISCOVERY</i>	<i>ST STM8</i>	On-board	STM8S105C6T6	16MHz	32KB	2KB
<i>STM32F7508-DK</i>	<i>ST STM32</i>	On-board	STM32F750N8H6	216MHz	64KB	340KB

SainSmart

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>SainSmart Due (Programming Port)</i>	<i>Atmel SAM</i>	External	AT91SAM3X8E	84MHz	512KB	96KB
<i>SainSmart Due (USB Native Port)</i>	<i>Atmel SAM</i>	External	AT91SAM3X8E	84MHz	512KB	96KB

Samsung

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Samsung ARTIK053</i>	<i>Samsung ARTIK</i>	On-board	S5JT200	320MHz	8MB	1.25MB

Seeed

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Seeeduino LoRaWAN</i>	<i>Atmel SAM</i>	External	SAMD21G18A	48MHz	256KB	32KB

SeeedStudio

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Seeed Arch BLE</i>	<i>Nordic nRF51</i>	On-board	NRF51822	16MHz	128KB	16KB
<i>Seeed Arch Link</i>	<i>Nordic nRF51</i>	On-board	NRF51822	16MHz	256KB	16KB
<i>Seeed Arch Max</i>	<i>ST STM32</i>	On-board	STM32F407VET6	168MHz	512KB	192KB
<i>Seeed Arch Pro</i>	<i>NXP LPC</i>	On-board	LPC1768	96MHz	512KB	64KB
<i>Seeed Tiny BLE</i>	<i>Nordic nRF51</i>	On-board	NRF51822	16MHz	256KB	16KB
<i>Seeed Wio 3G</i>	<i>ST STM32</i>	On-board	STM32F439VI	180MHz	2MB	256KB
<i>Wio Lite RISC-V</i>	<i>GigaDevice GD32V</i>	External	GD32VF103CBT6	108MHz	128KB	32KB

Semtech

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>NAMote72</i>	<i>ST STM32</i>	External	STM32L152RC	32MHz	256KB	32KB

SiFive

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>HiFive Unleashed</i>	<i>SiFive</i>	On-board	FU540	1500MHz	32MB	8GB
<i>HiFive1</i>	<i>SiFive</i>	On-board	FE310	320MHz	16MB	16KB
<i>HiFive1 Rev B</i>	<i>SiFive</i>	On-board	FE310	320MHz	16MB	16KB

Sigma Delta Technologies

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>SDT52832B</i>	<i>Nordic nRF52</i>	External	NRF52832	64MHz	512KB	64KB

Silicognition

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Silicognition wESP32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB

Silicon Labs

Name	Platform	De-bug	MCU	Fre-quency	Flash	RAM
<i>EFM32GG-STK3700 Giant Gecko</i>	<i>Silicon Labs EFM32</i>	On-board	EFM32GG990F1024	48MHz	1MB	128KB
<i>EFM32LG-STK3600 Leopard Gecko</i>	<i>Silicon Labs EFM32</i>	On-board	EFM32LG990F256	48MHz	256KB	32KB
<i>EFM32WG-STK3800 Wonder Gecko</i>	<i>Silicon Labs EFM32</i>	On-board	EFM32WG990F256	48MHz	256KB	32KB
<i>EFM32ZG-STK3200 Zero Gecko</i>	<i>Silicon Labs EFM32</i>	On-board	EFM32ZG222F32	24MHz	32KB	4KB
<i>SLSTK3400A USB-enabled Happy Gecko</i>	<i>Silicon Labs EFM32</i>	On-board	EFM32HG322F64	25MHz	64KB	8KB
<i>SLSTK3401A Pearl Gecko PG1</i>	<i>Silicon Labs EFM32</i>	On-board	EFM32PG1B200F256	40MHz	256KB	32KB
<i>Thunderboard Sense 2 Sensor-to-Cloud Advanced IoT</i>	<i>Silicon Labs EFM32</i>	On-board	EFR32MG12P432F1024	40MHz	1MB	256KB

Sipeed

Name	Platform	Debug	MCU	Fre-quency	Flash	RAM
<i>GD32VF103V-EVAL</i>	<i>GigaDevice GD32V</i>	External	GD32VF103VBT6	108MHz	128KB	32KB
<i>Sipeed Longan Nano</i>	<i>GigaDevice GD32V</i>	External	GD32VF103CBT6	108MHz	128KB	32KB
<i>Sipeed MAIX BiT</i>	<i>Kendryte K210</i>	External	K210	400MHz	16MB	6MB
<i>Sipeed MAIX BiT with Mic</i>	<i>Kendryte K210</i>	External	K210	400MHz	16MB	6MB
<i>Sipeed MAIX GO</i>	<i>Kendryte K210</i>	External	K210	400MHz	16MB	6MB
<i>Sipeed MAIX ONE DOCK</i>	<i>Kendryte K210</i>	External	K210	400MHz	16MB	6MB
<i>Sipeed MAIXDUINO</i>	<i>Kendryte K210</i>	External	K210	400MHz	16MB	6MB

Solder Splash Labs

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>DipCortex M3</i>	<i>NXP LPC</i>	External	LPC1347	72MHz	64KB	12KB
<i>Solder Splash Labs DipCortex M0</i>	<i>NXP LPC</i>	External	LPC11U24	50MHz	32KB	8KB

SparkFun

Name	Platform	Debug	MCU	Fre-quency	Flash	RAM
<i>SparkFun LoRa Gateway 1-Channel</i>	<i>Espressif 32</i>	Exter-nal	ESP32	240MHz	4MB	320KB
<i>SparkFun SAMD21 Dev Breakout</i>	<i>Atmel SAM</i>	Exter-nal	SAMD21G18A	48MHz	256KB	32KB
<i>SparkFun SAMD21 Mini Breakout</i>	<i>Atmel SAM</i>	Exter-nal	SAMD21G18A	48MHz	256KB	32KB

SparkFun Electronics

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>SparkFun ESP32 Thing</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB

Switch Science

Name	Platform	Debug	MCU	Fre-quency	Flash	RAM
<i>Switch Science mbed HRM1017</i>	<i>Nordic nRF51</i>	On-board	NRF51822	16MHz	256KB	16KB
<i>Switch Science mbed LPC1114FN28</i>	<i>NXP LPC</i>	On-board	LPC1114FN28	48MHz	32KB	4KB
<i>Switch Science mbed LPC824</i>	<i>NXP LPC</i>	On-board	LPC824	30MHz	32KB	8KB
<i>Switch Science mbed TY51822r3</i>	<i>Nordic nRF51</i>	On-board	NRF51822	32MHz	256KB	32KB

TI

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>TI FraunchPad MSP-EXP430FR5739LP</i>	<i>TI MSP430</i>	On-board	MSP430FR5739	16MHz	15.37KB	1KB
<i>TI LaunchPad (Stellaris) w/ lm4f120 (80MHz)</i>	<i>TI TIVA</i>	On-board	LPLM4F120H5QR	80MHz	256KB	32KB
<i>TI LaunchPad (Tiva C) w/ tm4c123 (80MHz)</i>	<i>TI TIVA</i>	On-board	LPTM4C1230C3PM80MHz	256KB	32KB	
<i>TI LaunchPad (Tiva C) w/ tm4c129 (120MHz)</i>	<i>TI TIVA</i>	On-board	LPTM4C1294NCPD120MHz	1MB	256KB	
<i>TI LaunchPad MSP-EXP430F5529LP</i>	<i>TI MSP430</i>	On-board	MSP430F5529	25MHz	47KB	8KB
<i>TI LaunchPad MSP-EXP430FR2311LP</i>	<i>TI MSP430</i>	On-board	MSP430FR2311	16MHz	3.75KB	1KB
<i>TI LaunchPad MSP-EXP430FR2433LP</i>	<i>TI MSP430</i>	On-board	MSP430FR2433	8MHz	15KB	4KB
<i>TI LaunchPad MSP-EXP430FR4133LP</i>	<i>TI MSP430</i>	On-board	MSP430FR4133	8MHz	15KB	2KB
<i>TI LaunchPad MSP-EXP430FR5969LP</i>	<i>TI MSP430</i>	On-board	MSP430FR5969	8MHz	47KB	2KB
<i>TI LaunchPad MSP-EXP430FR5994LP</i>	<i>TI MSP430</i>	On-board	MSP430FR5994	16MHz	256KB	4KB
<i>TI LaunchPad MSP-EXP430FR6989LP</i>	<i>TI MSP430</i>	On-board	MSP430FR6989	8MHz	47KB	2KB
<i>TI LaunchPad MSP-EXP430G2 w/ MSP430G2231</i>	<i>TI MSP430</i>	On-board	MSP430G2231	1MHz	2KB	256B
<i>TI LaunchPad MSP-EXP430G2 w/ MSP430G2452</i>	<i>TI MSP430</i>	On-board	MSP430G2452	16MHz	8KB	256B
<i>TI LaunchPad MSP-EXP430G2553LP</i>	<i>TI MSP430</i>	On-board	MSP430G2553	16MHz	16KB	512B

TTGO

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>TTGO LoRa32-OLED V1</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>TTGO T-Beam</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	1.25MB
<i>TTGO T1</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB

Taida Century

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Taida Century nRF52 mini board</i>	<i>Nordic nRF52</i>	External	NRF52832	64MHz	512KB	64KB

TauLabs

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Sparky V1 F303</i>	<i>ST STM32</i>	External	STM32F303CCT6	72MHz	256KB	40KB

Teensy

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Teensy 3.1 / 3.2</i>	<i>Teensy</i>	External	MK20DX256	72MHz	256KB	64KB
<i>Teensy 3.5</i>	<i>Teensy</i>	External	MK64FX512	120MHz	512KB	255.99KB
<i>Teensy 3.6</i>	<i>Teensy</i>	External	MK66FX1M0	180MHz	1MB	256KB
<i>Teensy 4.0</i>	<i>Teensy</i>	External	IMXRT1062	600MHz	1.94MB	1MB
<i>Teensy LC</i>	<i>Teensy</i>	External	MKL26Z64	48MHz	62KB	8KB

ThaiEasyElec

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>ESPino32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB

Unknown

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>ESP32 FM DevKit</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB

VAE

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>VAkE v1.0</i>	<i>ST STM32</i>	External	STM32F446RET6	180MHz	512KB	128KB

VNG

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>VNG VBLUNO51</i>	<i>Nordic nRF51</i>	On-board	NRF51822	16MHz	128KB	32KB

VintLabs

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>VintLabs ESP32 Devkit</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB

WEMOS

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>WEMOS LOLIN D32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>WEMOS LOLIN D32 PRO</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>WEMOS LOLIN32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>WeMos D1 MINI ESP32</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>WeMos WiFi and Bluetooth Battery</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB

WIZNet

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>WIZwiki-W7500</i>	<i>WIZNet W7500</i>	On-board	WIZNET7500	48MHz	128KB	48KB
<i>WIZwiki-W7500ECO</i>	<i>WIZNet W7500</i>	On-board	WIZNET7500ECO	48MHz	128KB	48KB
<i>WIZwiki-W7500P</i>	<i>WIZNet W7500</i>	On-board	WIZNET7500P	48MHz	128KB	48KB

Waveshare

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Waveshare BLE400</i>	<i>Nordic nRF51</i>	External	NRF51822	32MHz	256KB	32KB

Xilinx

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Artix-7 35T Arty FPGA Evaluation Kit</i>	<i>Shakti</i>	On-board	E-CLASS	50MHz	0B	128KB
<i>Arty A7-100: Artix-7 FPGA Development Board</i>	<i>Shakti</i>	On-board	C-CLASS	50MHz	0B	128MB
<i>Arty FPGA Dev Kit</i>	<i>SiFive</i>	On-board	FE310	450MHz	16MB	256MB

XinaBox

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>XinaBox CW02</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB

ng-beacon

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>ng-beacon</i>	<i>Nordic nRF51</i>	External	NRF51822	16MHz	256KB	32KB

oddWires

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>oddWires IoT-Bus Io</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB
<i>oddWires IoT-Bus Proteus</i>	<i>Espressif 32</i>	External	ESP32	240MHz	4MB	320KB

rhomb.io

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>L476DMWIK</i>	<i>ST STM32</i>	On-board	STM32L476VGT6	80MHz	1MB	128KB

sakura.io

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>sakura.io Evaluation Board</i>	<i>ST STM32</i>	On-board	STM32F411RET6	100MHz	1MB	128KB

sino:bit

Name	Platform	Debug	MCU	Frequency	Flash	RAM
<i>Sino:Bit</i>	<i>Nordic nRF51</i>	External	NRF51822	32MHz	256KB	32KB

u-blox

Name	Platform	Debug	MCU	Fre- quency	Flash	RAM
<i>Mbed Connect Cloud</i>	<i>ST STM32</i>	On- board	STM32F439ZIY6	168MHz	2MB	256KB
<i>u-blox C027</i>	<i>NXP LPC</i>	On- board	LPC1768	96MHz	512KB	64KB
<i>u-blox C030-N211 IoT Starter Kit</i>	<i>ST STM32</i>	External	STM32F437VG	180MHz	1MB	256KB
<i>u-blox C030-R410M IoT</i>	<i>ST STM32</i>	On- board	STM32F437VG	180MHz	1MB	256KB
<i>u-blox C030-U201 IoT Starter Kit</i>	<i>ST STM32</i>	External	STM32F437VG	180MHz	1MB	256KB
<i>u-blox EVK-NINA-B1</i>	<i>Nordic nRF52</i>	On- board	NRF52832	64MHz	512KB	64KB
<i>u-blox EVK-ODIN-W2</i>	<i>ST STM32</i>	External	STM32F439ZIY6	168MHz	2MB	256KB
<i>u-blox ODIN-W2</i>	<i>ST STM32</i>	External	STM32F439ZIY6	168MHz	2MB	256KB

y5 design

Name	Platform	Debug	MCU	Frequency	Flash	RAM
y5 LPC11U35 mbug	NXP LPC	External	LPC11U35	48MHz	64KB	10KB
y5 nRF51822 mbug	Nordic nRF51	On-board	NRF51822	16MHz	256KB	16KB

1.18 PIO Unit Testing

PIO Unit Testing allows segregating each part of the firmware/program and testing 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 on both. When testing both, PIO Plus builds firmware on the host machine, uploads into a target device, starts tests, and collects the test results into test reports. The final information will be shown on the host side with informative output and statistic.

Using [PIO Remote](#) you can start unit tests on the **Remote Device** from anywhere in the world or integrate with [Continuous Integration](#) systems.

Contents

- [Demo](#)
- [Tutorials and Examples](#)
 - [Tutorials](#)
 - [Project Examples](#)
- [Configuration](#)
- [Test Types](#)
 - [Desktop](#)
 - [Embedded](#)
- [Test Runner](#)
 - [Local](#)
 - [Remote](#)
- [Workflow](#)
 - [Shared Code](#)
- [API](#)
- [CLI Guide](#)

1.18.1 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.

```
(py27) /N/S/P/G/p/o/e/u/calculator (develop) $ platformio test -e uno -e native -f test_desktop -f test_embedded
Verbose mode can be enabled via `'-v, --verbose` option
Collected 3 items

----- [test/test_desktop] Building... (1/2) -----
Please wait...

----- [test/test_desktop] Testing... (2/2) -----
test/test_desktop/test_calculator.cpp:48:test_function_calculator_addition [PASSED]
test/test_desktop/test_calculator.cpp:49:test_function_calculator_subtraction [PASSED]
test/test_desktop/test_calculator.cpp:50:test_function_calculator_multiplication [PASSED]
test/test_desktop/test_calculator.cpp:43:test_function_calculator_division:FAIL: Expected 32 Was 33 [FAILED]

-----
4 Tests 1 Failures 0 Ignored
FAIL

----- [test/test_embedded] Building... (1/3) -----
Please wait...

----- [test/test_embedded] Uploading... (2/3) -----
Please wait...
Reading | #####| 100% 0.00s
Writing | #####| 100% 0.65s
Reading | #####| 100% 0.52s

----- [test/test_embedded] Testing... (3/3) -----
If you don't see any output for the first 10 secs, please reset board (press reset button)

test/test_embedded/test_calculator.cpp:53:test_function_calculator_addition [PASSED]
test/test_embedded/test_calculator.cpp:54:test_function_calculator_subtraction [PASSED]
test/test_embedded/test_calculator.cpp:55:test_function_calculator_multiplication [PASSED]
test/test_embedded/test_calculator.cpp:44:test_function_calculator_division:FAIL: Expected 32 Was 33 [FAILED]

-----
4 Tests 1 Failures 0 Ignored

----- [TEST SUMMARY] -----
test/test_common/env:uno [IGNORED]
test/test_common/env:nodemcu [IGNORED]
test/test_common/env:native [IGNORED]
test/test_desktop/env:uno [IGNORED]
test/test_desktop/env:nodemcu [IGNORED]
test/test_desktop/env:native [FAILED]
test/test_embedded/env:uno [FAILED]
test/test_embedded/env:nodemcu [IGNORED]
test/test_embedded/env:native [IGNORED]
----- [FAILED] Took 10.76 seconds -----
```

1.18.2 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.

1.18.3 Configuration

PIO Unit Testing Engine can be configured from “*platformio.ini*” (*Project Configuration File*)

1.18.4 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 programs it. Then, it connects to this device using configured Serial *test_port* and communicates via *test_transport*. Finally, it runs tests on the embedded side, collects results, analyzes them, and provides a summary on a host machine side (desktop).

Note: Please note that the **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 the base interface.

1.18.5 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 test on a remote machine or remote target device (board) without having to depend on OS software, extra software, SSH, VPN or opening network ports. Remote Unit Testing works in pair with [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](#).

1.18.6 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 = nodemcuv2

;
; 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 at least 2 seconds delay before `UNITY_BEGIN()`. That time is needed to establish a `Serial` connection between a host machine and a target device.

```
delay(2000); // for Arduino framework
wait(2); // for ARM mbed framework
UNITY_BEGIN();
```

4. Place a test in the `test` directory. If you have more than one test, split them into sub-folders. For example, `test/test_1/*. [c, cpp, h]`, `test_N/*.[c, cpp, h]`, etc. If there is no such directory in the `test` folder, then **PIO Unit Testing Engine** will treat the source code of `test` folder as SINGLE test.
5. Run tests 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 splitting the source code into multiple components and placing them into `lib_dir` (project’s private libraries and components). *Library Dependency Finder (LDF)* will find and include these libraries automatically in the build process. You can include any library/component header file in your test or program source code via `#include <MyComponent.h>`.
See [Local & Embedded: Calculator](#) for an example, where we have a “calculator” component in `lib_dir` folder and include it in tests and the main program using `#include <calculator.h>`.
2. Manually instruct PlatformIO to build source code from `src_dir` folder using `test_build_project_src` option in `platformio.ini` (*Project Configuration File*):

```
[env:myenv]
platform = ...
test_build_project_src = true
```

This is very useful if you unit test independent libraries where you can’t split source code.

Warning: Please note that you will need to use `#ifdef UNIT_TEST` and `#endif` guard to hide non-test related source code. For example, own `main()` or `setup()` / `loop()` functions.

1.18.7 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)

1.18.8 CLI Guide

1.19 Cloud & Desktop IDE

1.19.1 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

1.19.2 Cloud IDE

Cloud9

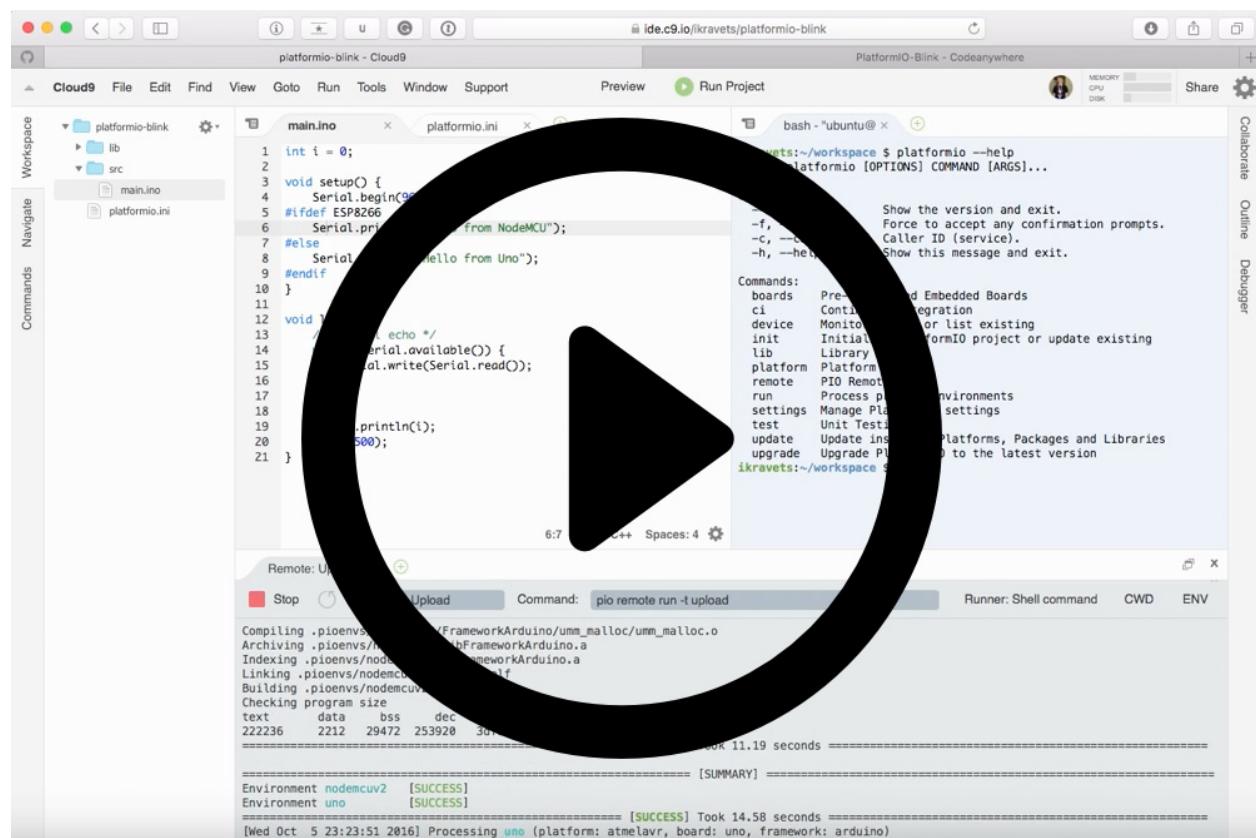
Cloud9 combines a powerful online code editor with a full Ubuntu workspace in the cloud. Workspaces are powered by Docker Ubuntu containers that give you full freedom over your environment, including sudo rights. Do a git push, compile SASS, see server output, and Run apps easily with the built-in Terminal and Runners.

Contents

- *Cloud9*
 - *Demo*
 - *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

Create a new workspace

Workspace name

platformio-workspace

Description

Workspace for private PlatformIO Projects

Hosted workspace

Clone workspace

Remote SSH workspace

Salesforce



Private

This is a workspace for your eyes only



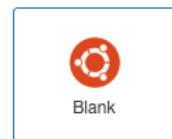
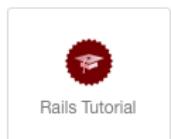
Public

This will create a workspace for everybody to see

Clone from Git or Mercurial URL (optional)

e.g. ajaxorg/ace or git@github.com:ajaxorg/ace.git

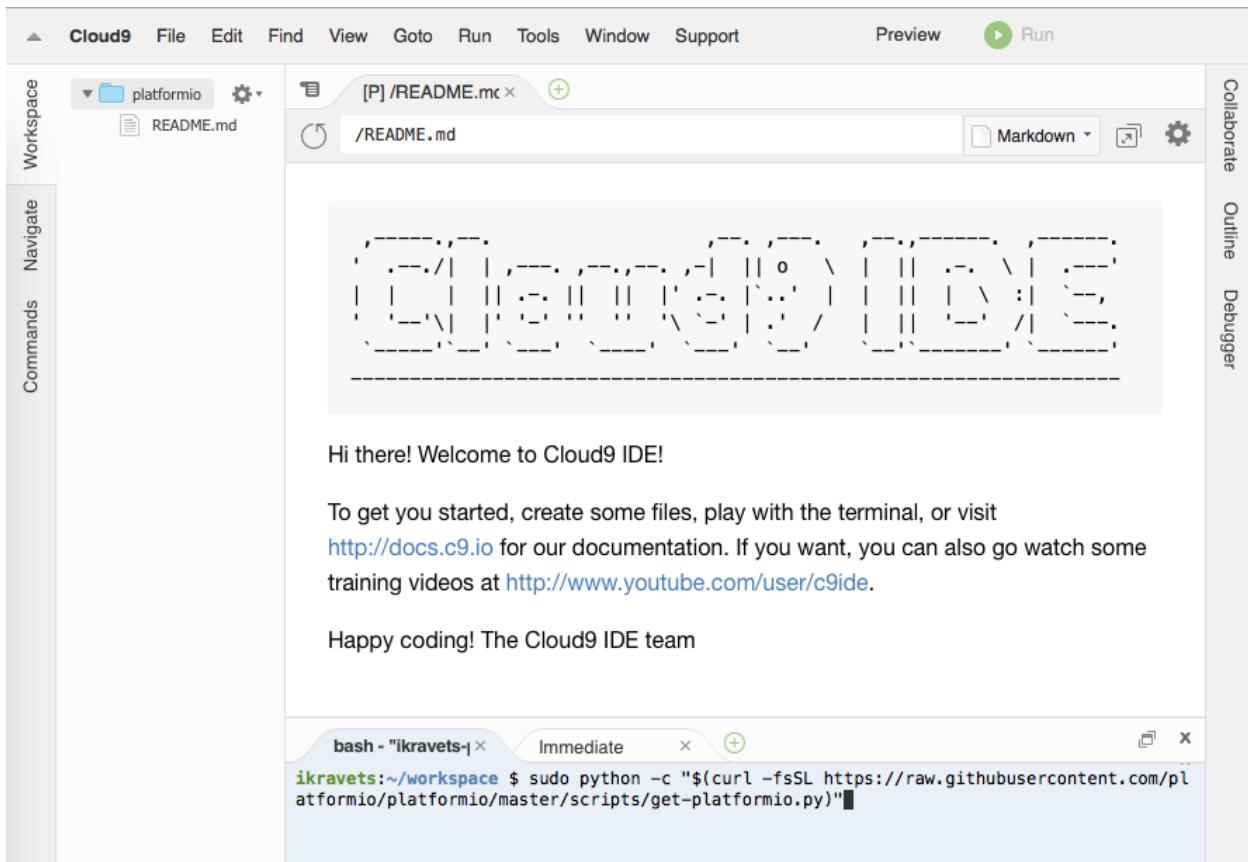
Choose a template



Create workspace

3. Install *PlatformIO Core (CLI)* using Cloud IDE Terminal. Paste a next command

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



4. Log in to *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:

```
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:

```
#include <Arduino.h>

int i = 0;

void setup() {
    Serial.begin(9600);
    Serial.println("Hello Cloud9!");
}
```

(continues on next page)

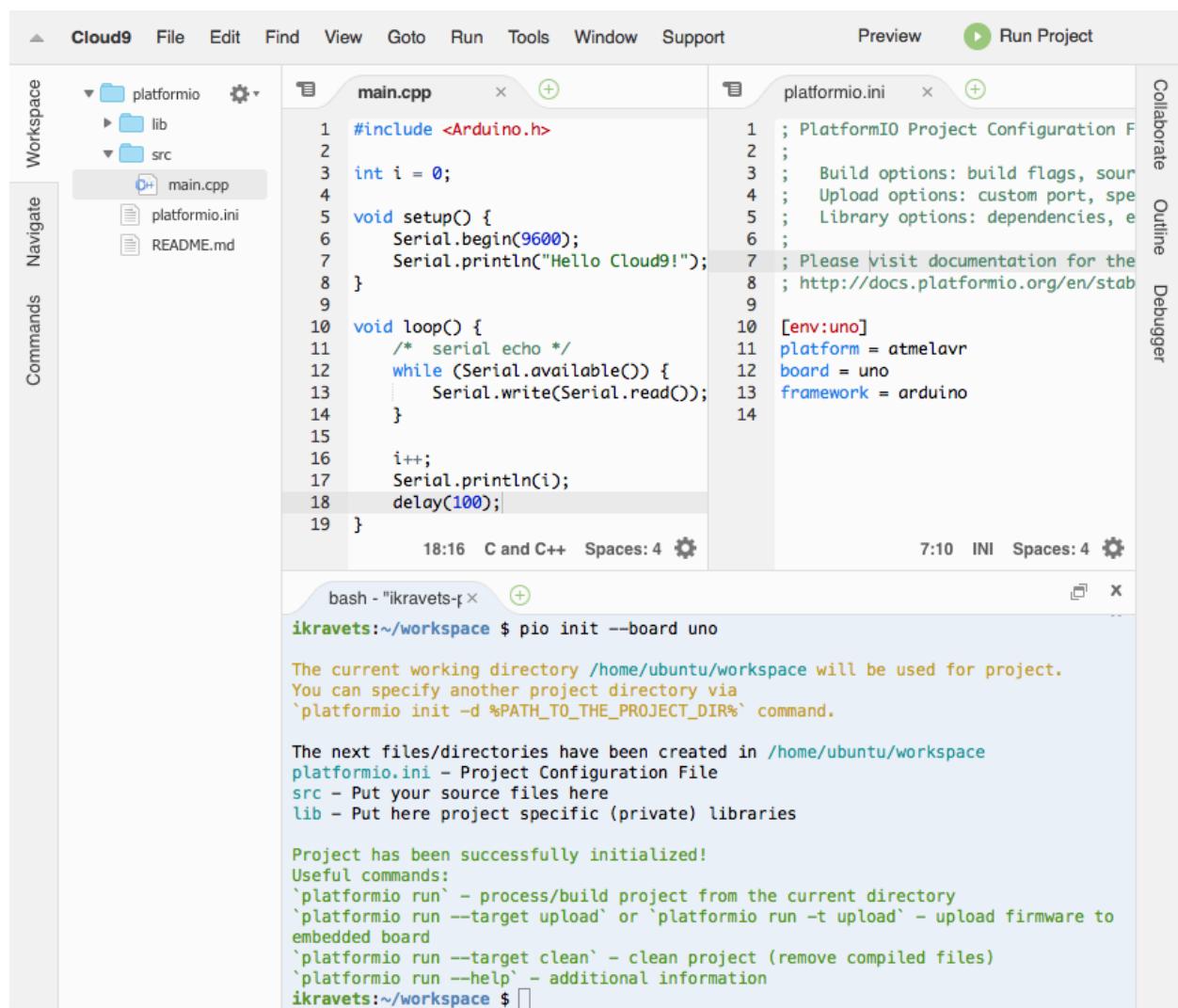
(continued from previous page)

```

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 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)$"
}
```

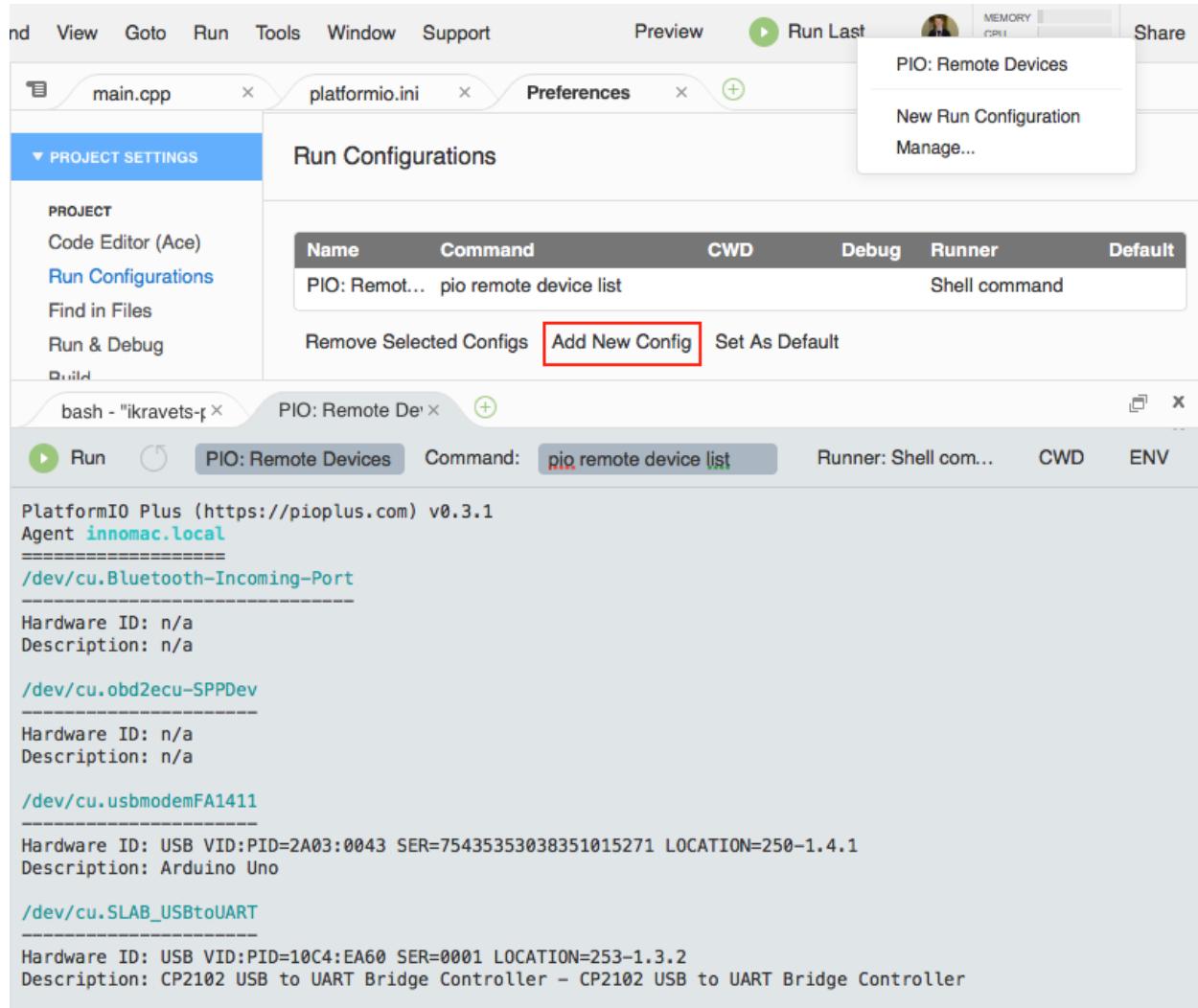
Save new Build System and give a name PIOBuilder. Now, you can select it as default Build System using Menu : Run > Build System > PIOBuilder.

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"

The screenshot shows the PlatformIO IDE interface. The top menu bar includes File, View, Goto, Run, Tools, Window, Support, Preview, and Run Project. A dropdown menu under Run Project shows options: PIO: Remote Devices and PIO: Remote Upload (which is selected). The main workspace has tabs for main.cpp, platformio.ini, and Preferences. On the left, a sidebar titled 'PROJECT SETTINGS' contains links for Code Editor (Ace), Run Configurations (which is currently selected), Find in Files, Run & Debug, and Build.

The central area displays 'Run Configurations' with a table:

Name	Command	CWD	Debug	Runner	Default
PIO: Remote Devices	pio remote device list			Shell command	
PIO: Remote Upload	pio remote run -t upload			Shell command	true

Buttons at the bottom of this section include Remove Selected Configs, Add New Config (which is highlighted with a red box), and Set As Default.

A terminal window titled 'PIO: Remote Up' is open, showing the output of a remote upload process:

```

bash - "ikravets-p ~" 10:35:49
PIO: Remote Up × + Run PIO: Remote Upload Command: pio remote run -t upload Runner: Shell com... CWD ENV
PlatformIO Plus (https://pioplus.com) v0.3.1
Building project locally
[Sat Oct 29 22:35:48 2016] Processing uno (platform: atmelavr, board: uno, framework: arduino)

Verbose mode can be enabled via `--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
[Sun Oct 30 01:35:49 2016] Processing uno (platform: atmelavr, board: uno, framework: arduino)

Verbose mode can be enabled via `--verbose` option
Looking for upload port...
Auto-detected: /dev/cu.usbmodemFA1411
Uploading .pioenvs/uno/firmware.hex

avrduude: AVR device initialized and ready to accept instructions
Reading | ##### | 100% 0.00s
avrduude: Device signature = 0x1e950f
avrduude: reading input file ".pioenvs/uno/firmware.hex"
avrduude: writing flash (2372 bytes):
Writing | ##### | 100% 0.39s
avrduude: 2372 bytes of flash written
avrduude: verifying flash memory against .pioenvs/uno/firmware.hex:
avrduude: load data flash data from input file .pioenvs/uno/firmware.hex:
avrduude: input file .pioenvs/uno/firmware.hex contains 2372 bytes
avrduude: reading on-chip flash data:
Reading | ##### | 100% 0.31s
avrduude: verifying ...
avrduude: 2372 bytes of flash verified

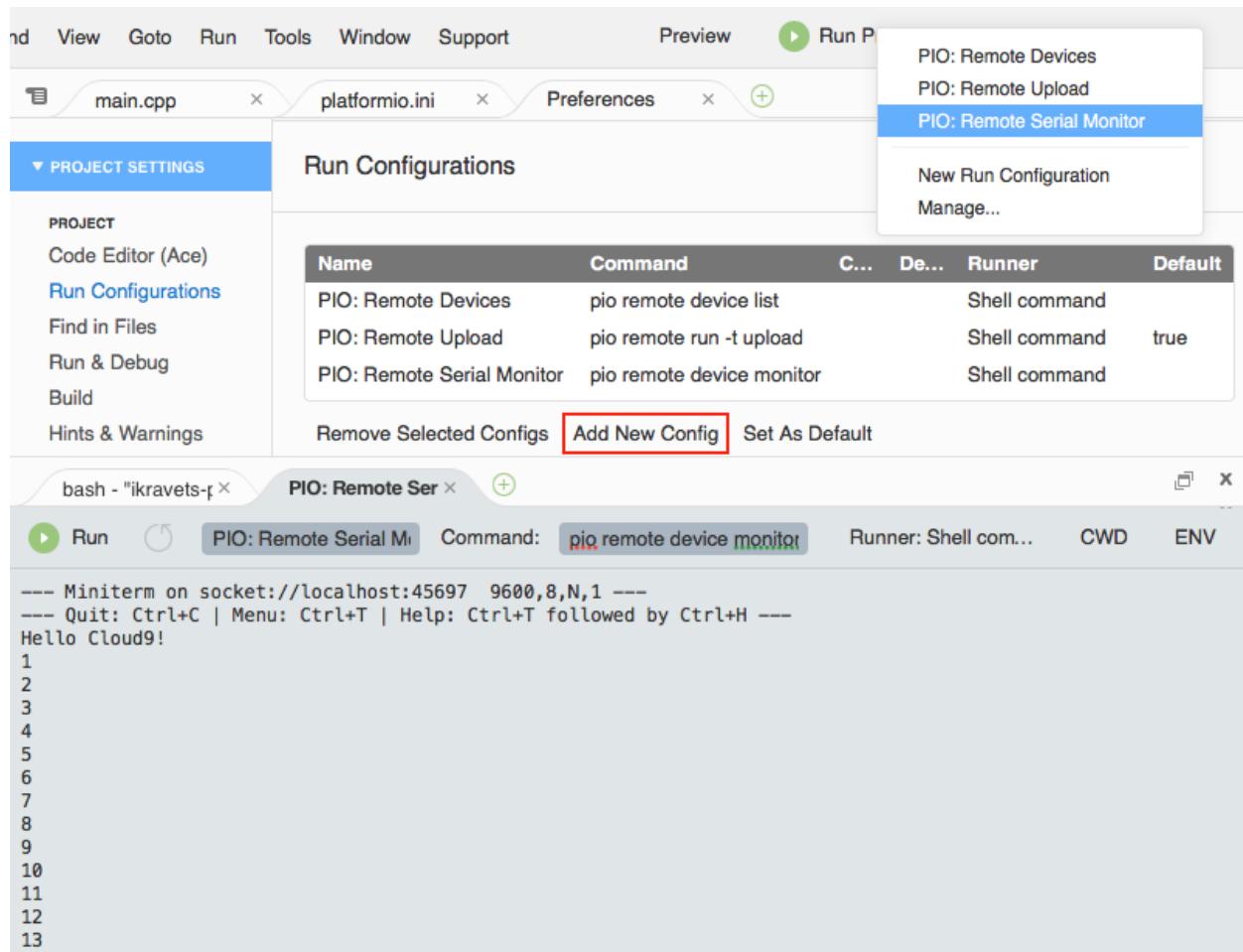
avrduude done. Thank you.
===== [SUCCESS] Took 3.19 seconds =====
  
```

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"

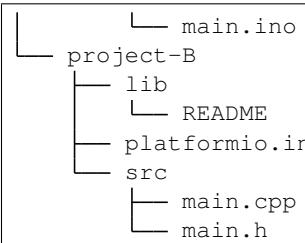


Multi-Project workspace

You can have multiple PlatformIO-based Projects in the same workspace. We recommend a next folders structure:



(continued from previous 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.

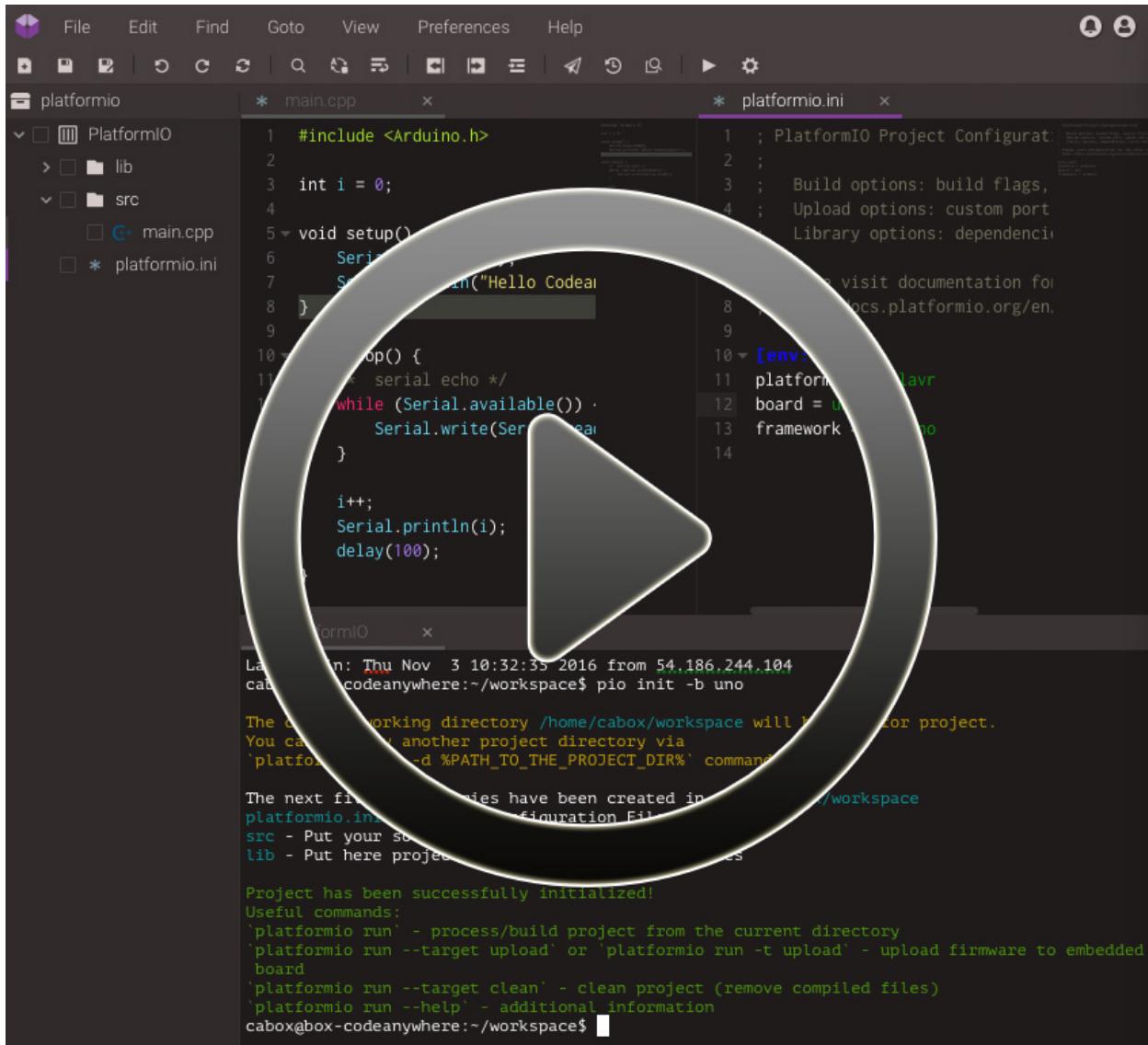
Contents

- *Codeanywhere*
 - *Demo*
 - *Integration*
 - *Quick Start*
 - *Run Button*
 - *Remote Device Manager*
 - *Remote Firmware Uploading*
 - *Remote Serial Port Monitor*
 - *Multi-Project workspace*

Note:

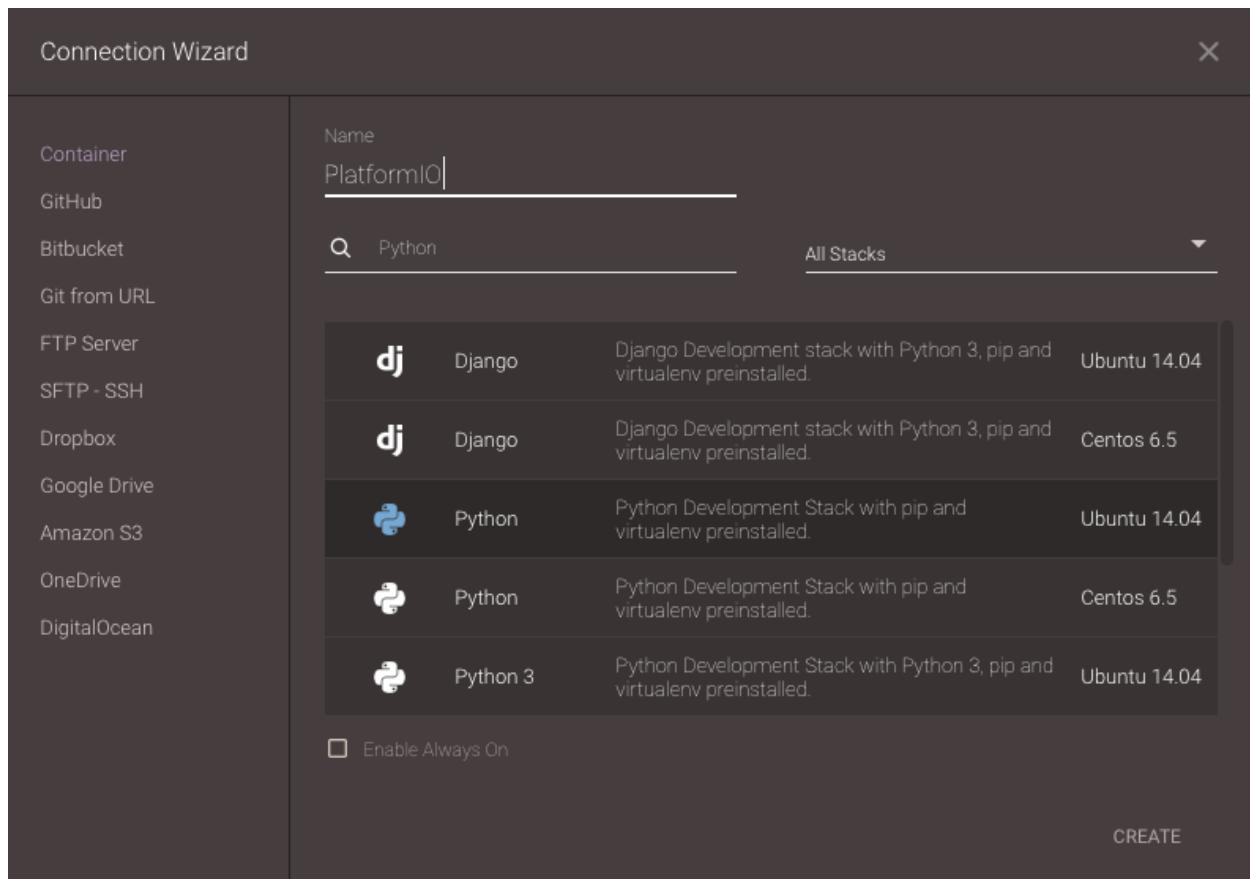
1. Please make sure to read [PIO Remote](#) guide first.
2. You need [PIO Account](#) if you don't have it. Registration is FREE.
3. You should have a running [PIO Remote Agent](#) on a remote machine where hardware devices are connected physically or accessible for the remote operations. See [PIO Remote Quick Start](#) for details.

Demo



Integration

1. Sign in to Codeanywhere. A registration is FREE and gives you unlimited private projects within the one Container.
2. Open Dashboard Projects
3. Create a new Project and open it. In Connection Wizard create new Container:
 - **Name** set to “PlatformIO”
 - **Stack** search for Python stack (not Python3) that is based on Ubuntu OS.
 - Click on “Create” button.



4. Open **SSH-Terminal** tab (right click on Container (PlatformIO) > SSH Terminal) and install *PlatformIO Core (CLI)* using a next command

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

1. Initialize new PlatformIO-based Project. Run a next command in a Cloud IDE SSH Terminal:

```

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:

```

#include <Arduino.h>

int i = 0;

void setup() {

```

(continues on next page)

(continued from previous page)

```

Serial.begin(9600);
Serial.println("Hello Codeanywhere!");
}

void loop() {
    /* serial echo */
    while (Serial.available()) {
        Serial.write(Serial.read());
    }

    i++;
    Serial.println(i);
    delay(100);
}

```

The screenshot shows the PlatformIO Cloud IDE interface. On the left, the project structure is displayed under the 'PlatformIO' workspace, containing 'lib', 'src', and 'main.cpp'. The 'main.cpp' file is open in the center editor, showing the provided C++ code. To the right, the 'platformio.ini' configuration file is also open. At the bottom, a terminal window shows the output of the 'pio init -b uno' command, which initializes the project with the specified board.

```

#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);
}

; PlatformIO Project Configuration File
; Build options: build flags, toolchain, dependencies
; Upload options: custom port, baud rate
; Library options: dependencies
; Please visit documentation for more details
; http://docs.platformio.org/en/latest/projectconf.html

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

```

```

Last login: Thu Nov  3 10:32:35 2016 from 54.186.244.104
cabox@box-codeanywhere:~/workspace$ pio init -b uno

The current working directory /home/cabox/workspace will be used for project.
You can specify another project directory via
`platformio init -d %PATH_TO_THE_PROJECT_DIR%` command.

The next files/directories have been created in /home/cabox/workspace
platformio.ini - Project Configuration File
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 embedded board
`platformio run --target clean` - clean project (remove compiled files)
`platformio run --help` - additional information
cabox@box-codeanywhere:~/workspace$ 

```

- If you prefer to work with *PlatformIO Core (CLI)* 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

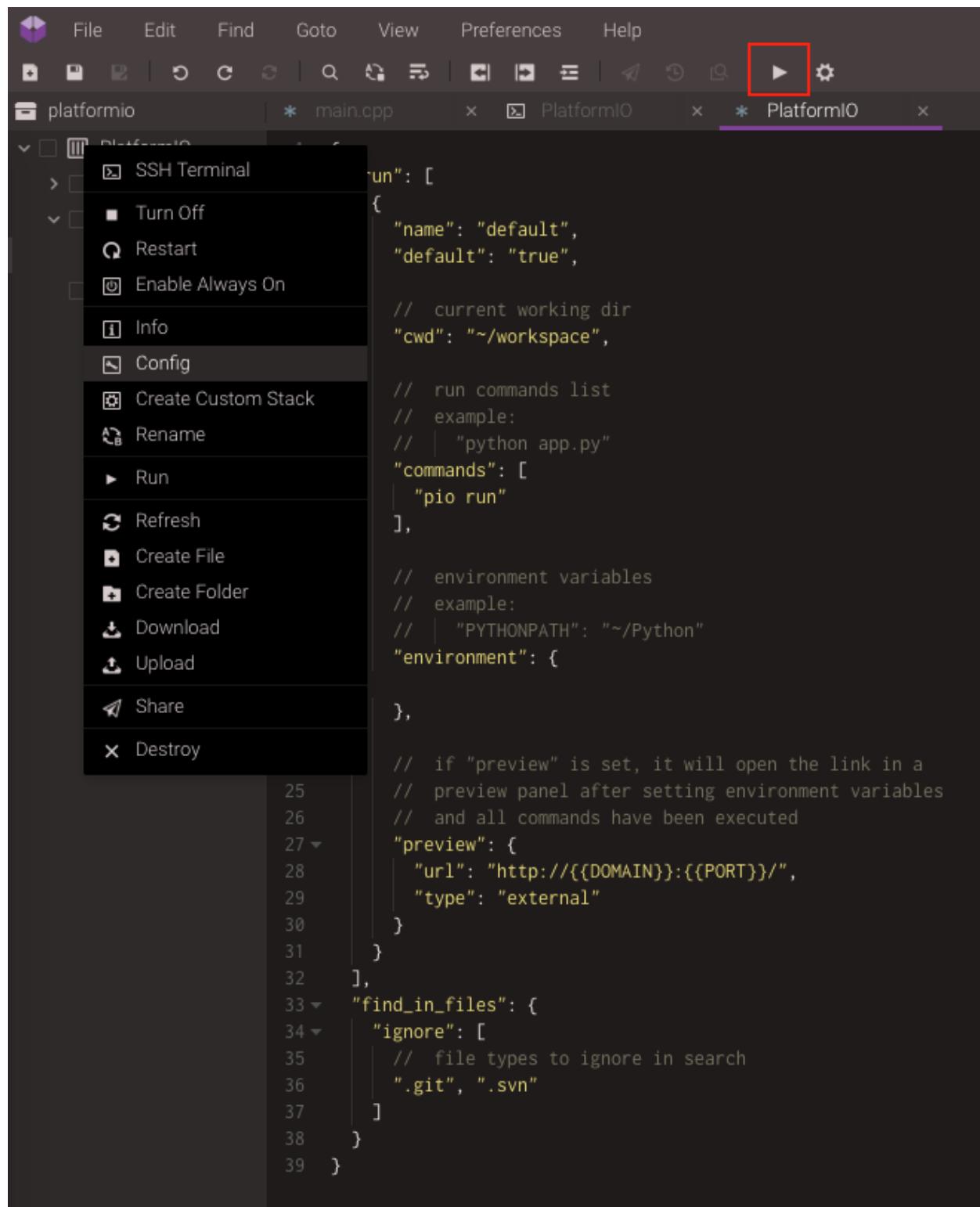
Codeanywhere provides a quick “Run Project” button where you can specify own command. Let’s add “PlatformIO Build Project” command:

1. Open “Project Config” via right click on Container Name (PlatformIO) > Config
2. Set commands field to

```
"commands": [  
    "pio run"  
]
```

3. Save configuration file.

Now, try to click on “Run Project” button. You can assign any PlatformIO command to this button.



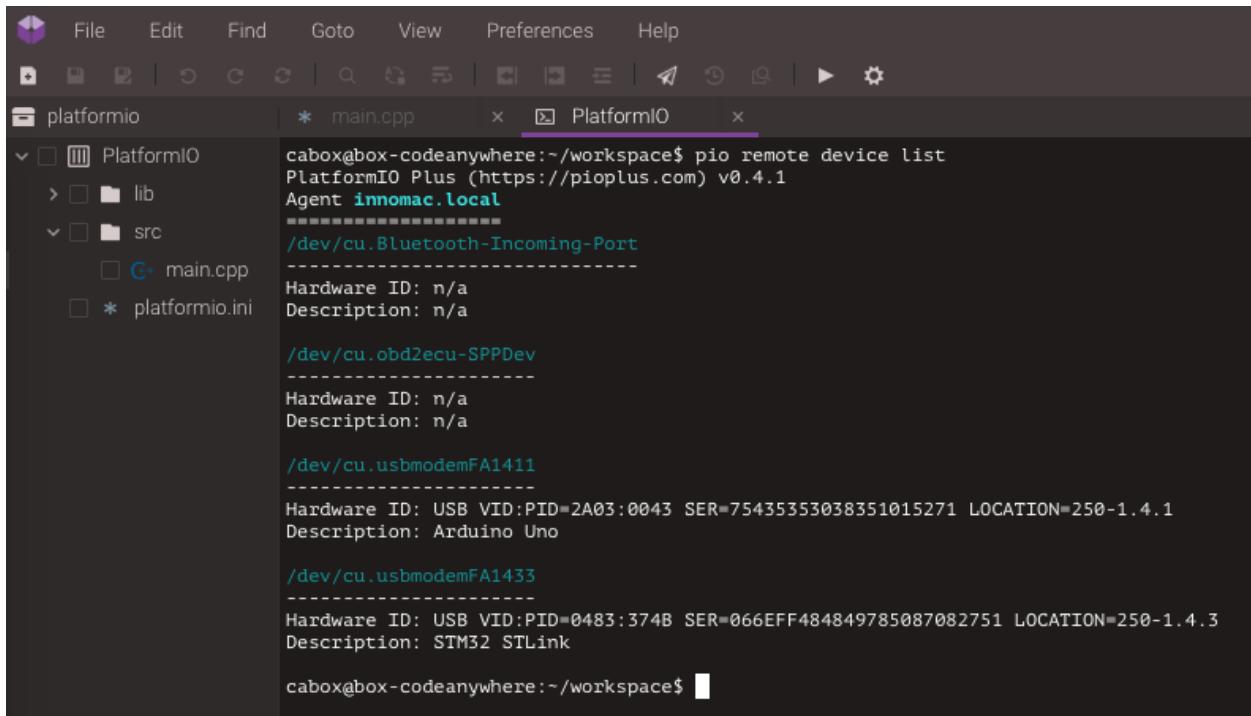
Remote Device Manager

Remote Device Manager works in pair with *PIO Remote*. You can list remote devices that are connected to host machine where *PIO Remote Agent* is started or are visible for it.

1. Open Cloud IDE SSH Terminal

2. Paste this command

```
pio remote device list
```



The screenshot shows the PlatformIO IDE interface. On the left is a file tree with a project named 'platformio'. The main area displays the output of the 'pio remote device list' command. The output shows a list of connected devices:

```
cabox@box-codeanywhere:~/workspace$ pio remote device list
PlatformIO Plus (https://pioplus.com) v0.4.1
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.usbmodemFA1411
-----
Hardware ID: USB VID:PID=2A03:0043 SER=75435353038351015271 LOCATION=250-1.4.1
Description: Arduino Uno

/dev/cu.usbmodemFA1433
-----
Hardware ID: USB VID:PID=0483:374B SER=066EFF484849785087082751 LOCATION=250-1.4.3
Description: STM32 STLink

cabox@box-codeanywhere:~/workspace$
```

Remote Firmware Uploading

Remote Firmware Uploading works in pair with *PIO Remote*. You can deploy firmware to any devices which are visible for *PIO Remote Agent*.

1. Open Cloud IDE SSH Terminal

2. Paste this command

```
pio remote run -t upload
```

```
* main.cpp      x PlatformIO      x
cabox@box-codeanywhere:~/workspace$ pio remote run -t upload
PlatformIO Plus (https://pioplus.com) v0.4.1
Building project locally
[Thu Nov  3 12:04:18 2016] Processing uno (platform: atmelavr, board: uno, framework: arduino)
-----
Verbose mode can be enabled via '-v, --verbose' option
Collected 25 compatible libraries
Looking for dependencies...
Project does not have dependencies
Checking program size
text      data      bss      dec      hex filename
2324        54     168    2546  9f2 .pioenvs/uno/firmware.elf
===== [SUCCESS] Took 0.64 seconds =====
Uploading firmware remotely
[Thu Nov  3 18:04:21 2016] Processing uno (platform: atmelavr, board: uno, framework: arduino)
-----
-----
Verbose mode can be enabled via '-v, --verbose' option
Looking for upload port...
Auto-detected: /dev/cu.usbmodemFA1411
Uploading .pioenvs/uno/firmware.hex

avrduude: AVR device initialized and ready to accept instructions

Reading | ##### | 100% 0.00s

avrduude: Device signature = 0x1e950f
avrduude: reading input file ".pioenvs/uno/firmware.hex"
avrduude: writing flash (2378 bytes):

Writing | ##### | 100% 0.39s

avrduude: 2378 bytes of flash written
avrduude: verifying flash memory against .pioenvs/uno/firmware.hex:
avrduude: load data flash data from input file .pioenvs/uno/firmware.hex:
avrduude: input file .pioenvs/uno/firmware.hex contains 2378 bytes
avrduude: reading on-chip flash data:

Reading | ##### | 100% 0.31s

avrduude: verifying ...
avrduude: 2378 bytes of flash verified

avrduude done. Thank you.
```

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

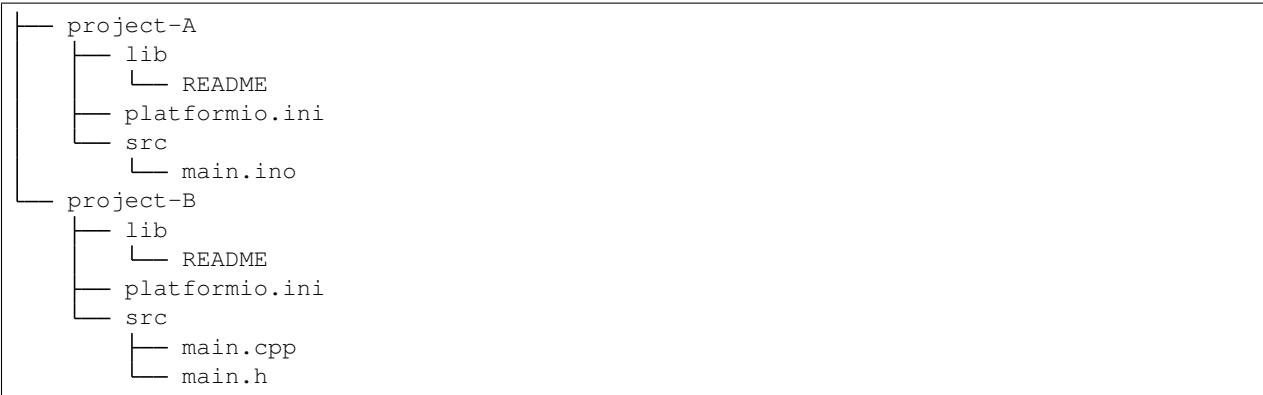
```
pio remote device monitor
```

```
* main.cpp      × PlatformIO      ×
cabox@box-codeanywhere:~/workspace$ pio remote device monitor
PlatformIO Plus (https://pioplus.com) v0.4.1
Starting Serial Monitor on innomac.local:/dev/cu.usbmodemFA1411
--- Miniterm on socket://localhost:44128 9600,8,N,1 ---
--- Quit: Ctrl+C | Menu: Ctrl+T | Help: Ctrl+T followed by Ctrl+H ---
Hello Codeanywhere!
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17

--- exit ---
1cabox@box-codeanywhere:~/workspace$
```

Multi-Project workspace

You can have multiple PlatformIO-based Projects in the same workspace. We recommend a next folders structure:



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.

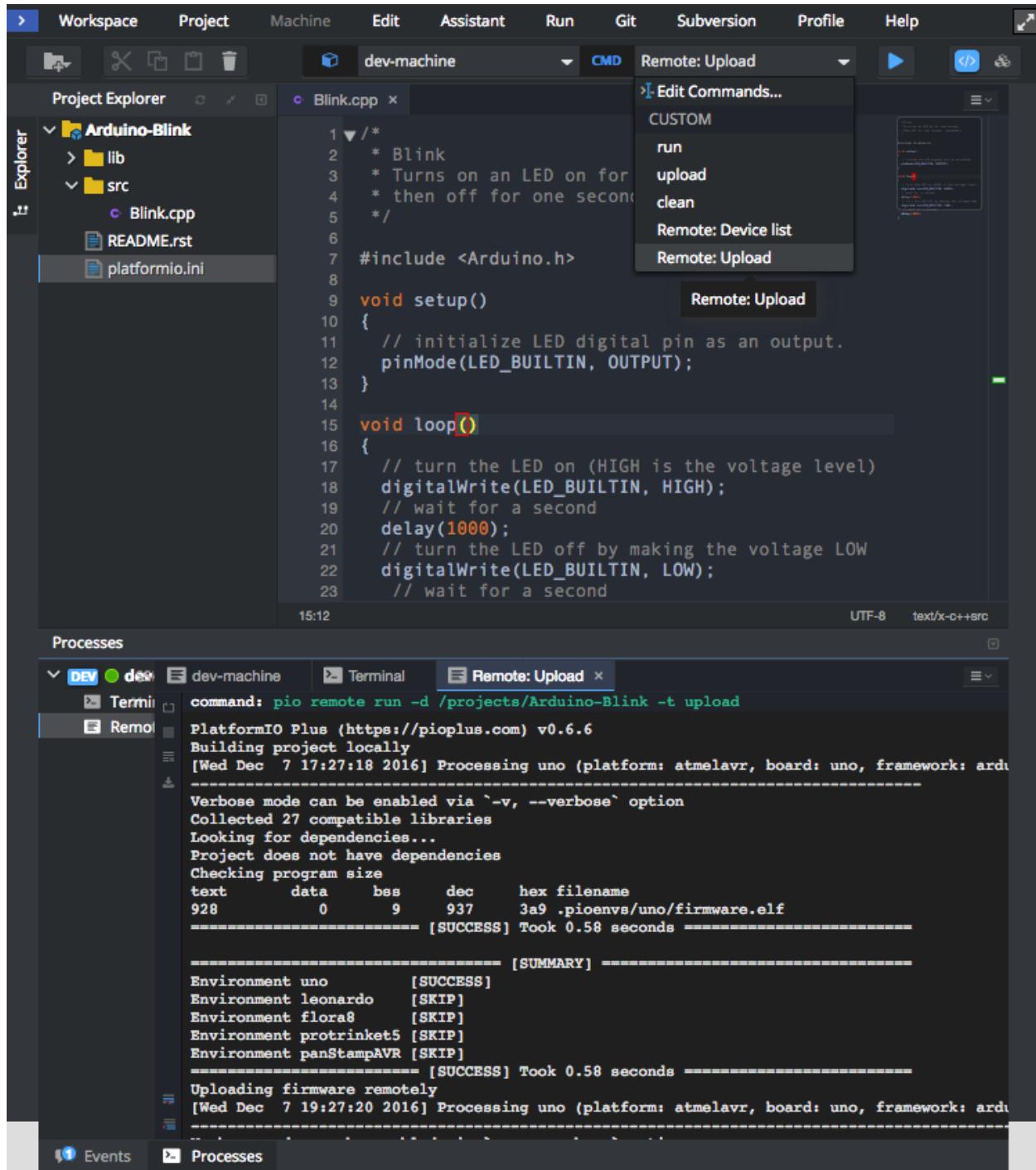
Contents

- *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



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”.

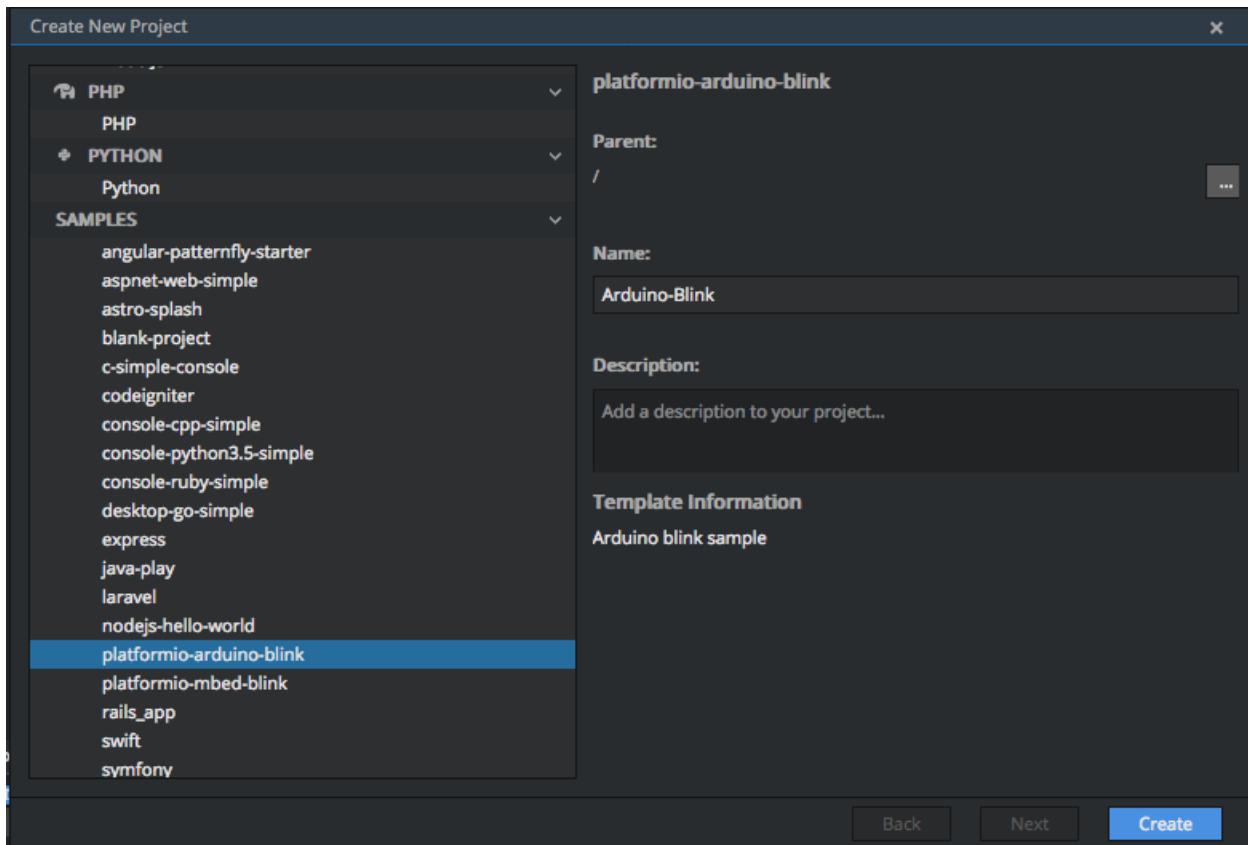
The screenshot shows the Codenvy interface. On the left, there's a sidebar with 'Dashboard', 'Workspaces' (selected), 'Stacks', and 'Factories'. The main area is titled 'New Workspace' with a 'CREATE' button. Below it, there are 'Settings' and 'Runtime' tabs, with 'Runtime' selected. A 'Name' field contains 'PlatformIO'. Under 'Select Stack', there's a description about stacks being recipes or images used to define your environment runtime. A 'Ready-to-go Stacks' section shows a single item: 'PLATFORMIO' (PlatformIO, IoT, embedded, arduino, mbed). A 'Filter:' section explains how to filter stacks by selecting technologies.

4. Using opened Terminal, please log in to *PIO Account* using *platformio account login* command.

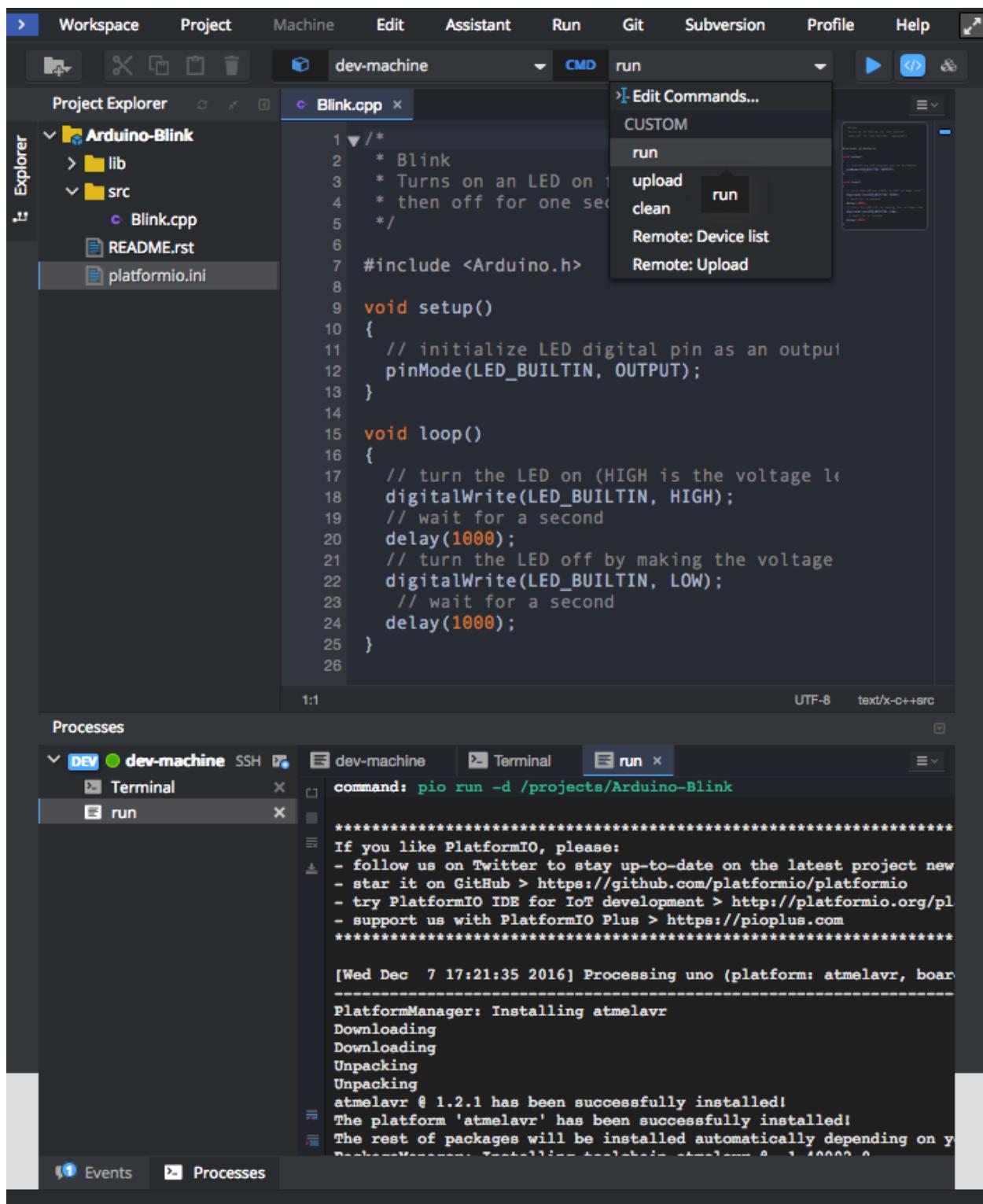
Quick Start

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



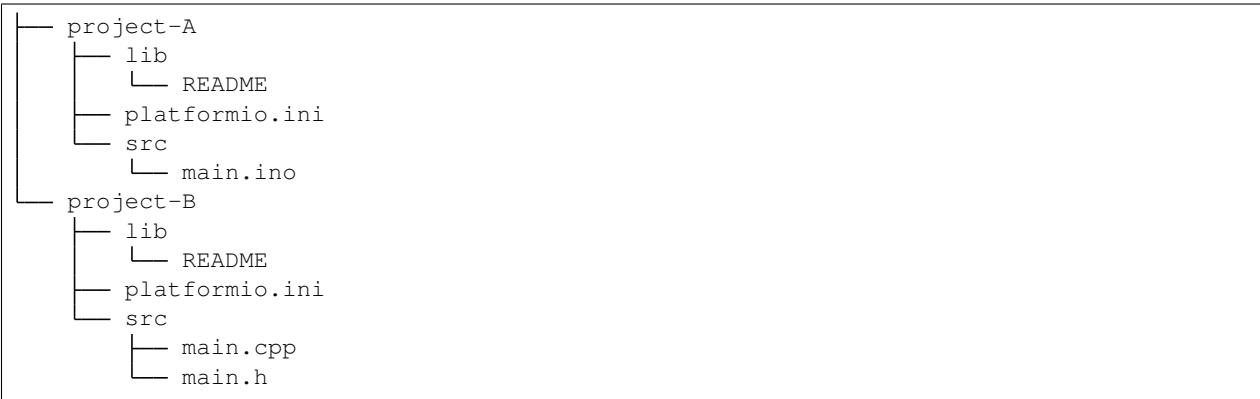
3. If you prefer to work with *PlatformIO Core (CLI)* CLI, then you can process project using Cloud IDE Terminal and the next commands:

- *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:



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.

1.19.3 Desktop IDE

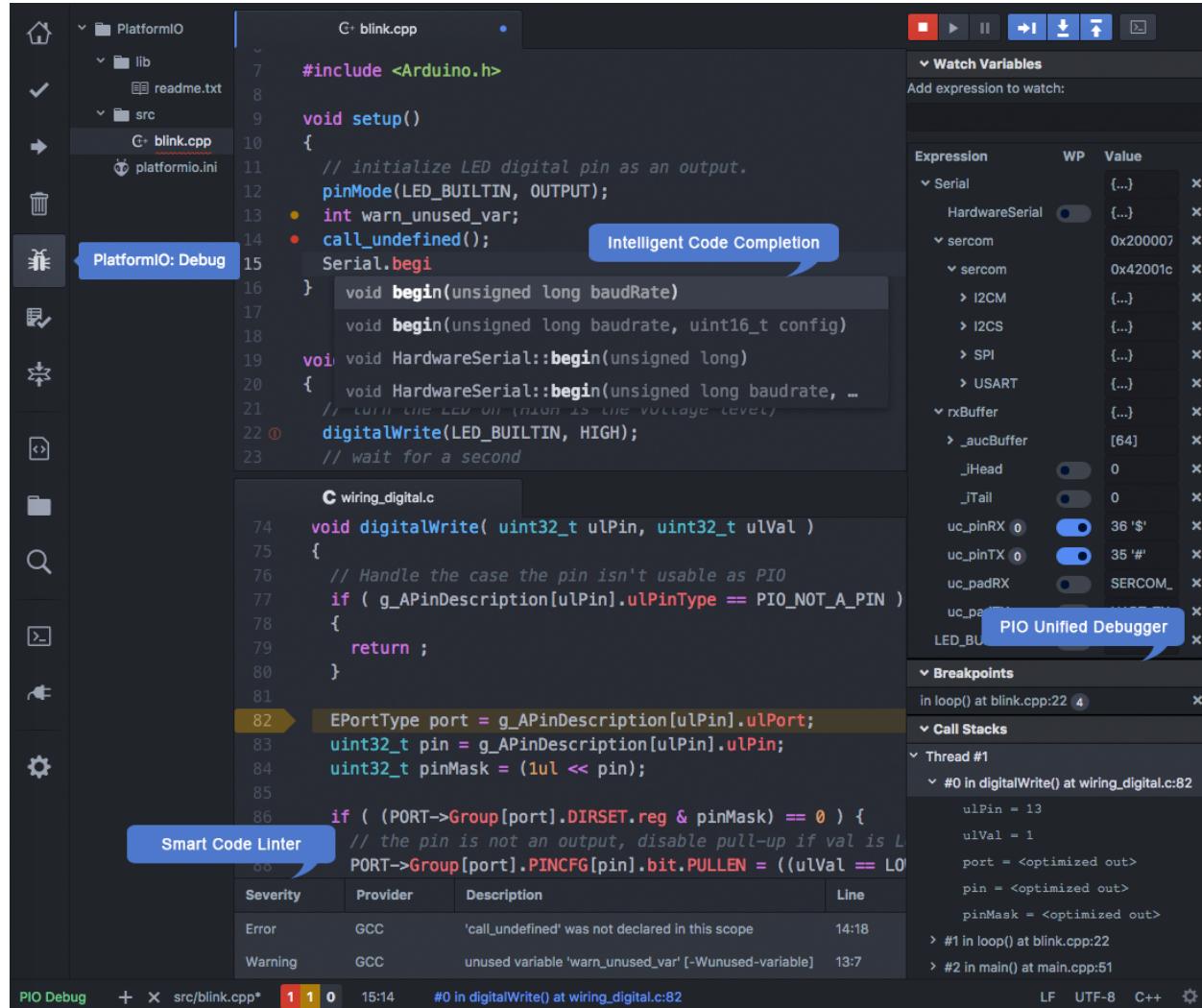
PlatformIO IDE for Atom

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

- Cross-platform build system without external dependencies to the OS software:
 - 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*

- [Launch](#)
- [Setting Up the Project](#)
- [Process Project](#)
- [Menu item PlatformIO](#)
- [PlatformIO Toolbar](#)
- [Building / Uploading / Targets](#)
- [Intelligent Code Completion](#)
- [Smart Code Linter](#)
- [Install Shell Commands](#)
- [Known issues](#)
 - [Smart Code Linter is disabled for Arduino files](#)
 - * [Convert Arduino file to C++ manually](#)
 - * [Force Arduino file as C++](#)
 - [Arch Linux: PlatformIO IDE Terminal issue](#)
- [Frequently Asked Questions](#)
 - [Keep build panel visible](#)
 - [Automatically save on build](#)
 - [Jump to Declaration](#)
 - [Code Formatting](#)
- [Uninstall Atom with PlatformIO IDE](#)
 - [Windows](#)
 - [macOS](#)
 - [Linux](#)
- [Articles / Manuals](#)
- [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 Atom](#). [PlatformIO Core \(CLI\)](#) is built into PlatformIO IDE and you will be able to use it within PlatformIO IDE Terminal.

Also, PlatformIO IDE allows one to install [PlatformIO Core \(CLI\)](#) Shell Commands (pio, platformio) globally to your system via Menu: PlatformIO > Install Shell Commands.

I. Atom

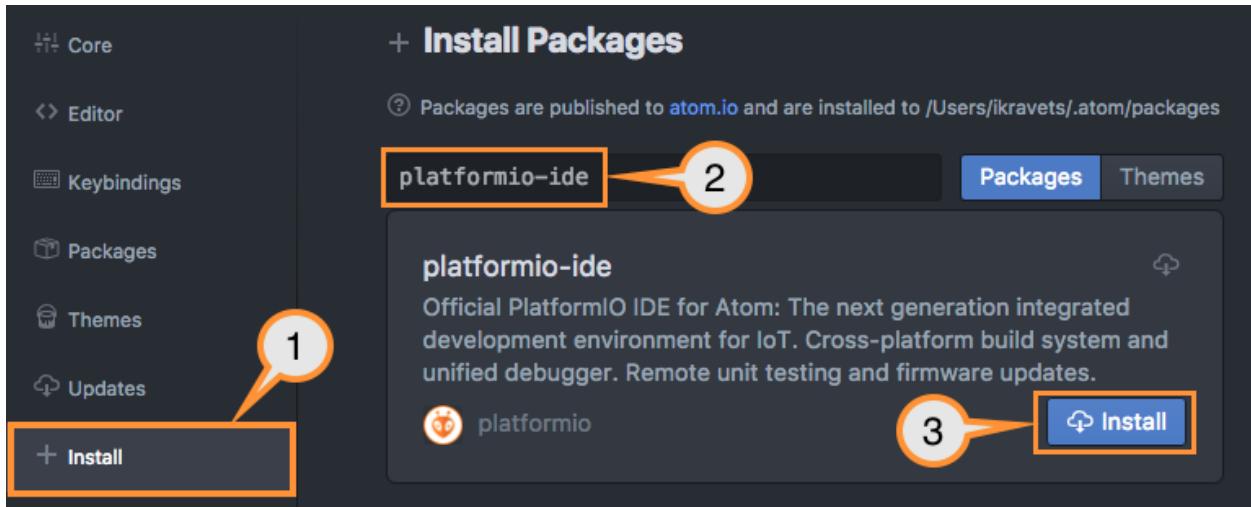
0. Download and install GitHub's official Atom text editor. PlatformIO IDE is built on top of it.

1. Open Atom Package Manager

- Mac OS X, Menu: Atom > Preferences > Install
- Windows, Menu: File > Settings > Install
- Linux, Menu: Edit > Preferences > Install

2. Search for the official platformio-ide package

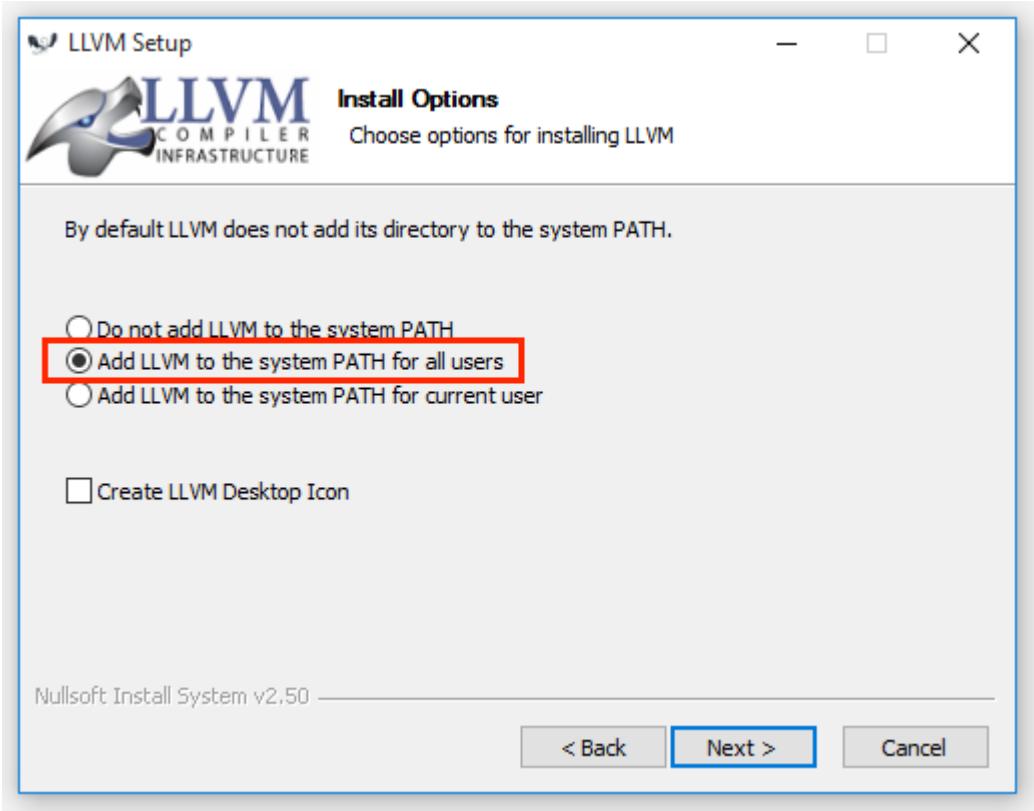
3. Install PlatformIO IDE.



II. Clang for Intelligent Code Completion

PlatformIO IDE uses Clang for the Intelligent Code Completion. To check that clang is available in your system, please open Terminal and run `clang --version`. If clang is not installed, then **install it and restart Atom**:

- **Mac OS X:** Install the latest Xcode along with the latest Command Line Tools (they are installed automatically when you run clang in Terminal for the first time, or manually by running `xcode-select --install`)
- **Windows:** Download Clang 3.9.1 for Windows. Please select “Add LLVM to the system PATH” option on the installation step.
 - Clang 3.9.1 for Windows (32-bit)
 - Clang 3.9.1 for Windows (64-bit)



Warning: PLEASE DO NOT INSTALL CLANG 4.0. TEMPORARILY, WE SUPPORT ONLY CLANG 3.9.

If you see a Failed to find MSBuild toolsets directory error in the installation console, please ignore it and press any key to close this window. PlatformIO IDE uses only the Clang completion engine, which should work after that without any problems.

- **Linux:** Using package managers: `apt-get install clang` or `yum install clang`.
- **Other Systems:** Download the latest [Clang](#) for the other systems.

Warning: If some libraries are not visible in [PlatformIO IDE for Atom](#) and Code Completion or Code Linting does not work properly, please perform Menu: PlatformIO > Rebuild C/C++ Project Index (Autocomplete, Linter)

Quick Start

This tutorial introduces you to the basics of PlatformIO IDE workflow and shows you the creation process for a simple “Blink” example. After finishing, you will have a general understanding of how to work with projects in the IDE.

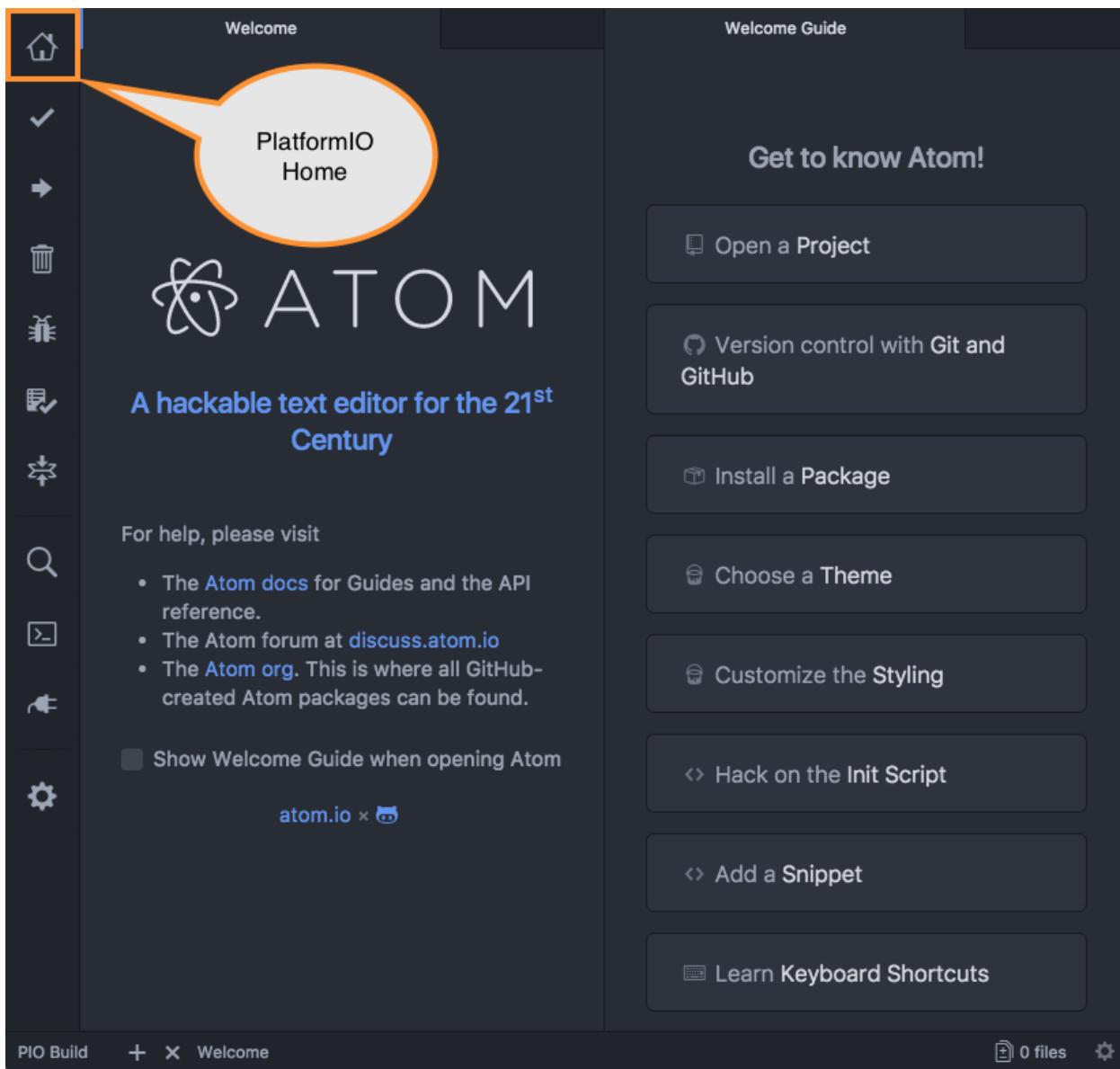
Launch

After installation, launch PlatformIO IDE by opening Atom. Once Atom is opened, the PlatformIO IDE auto installer will continue to install dependent packages and [PlatformIO Core \(CLI\)](#). Please be patient and let the installation

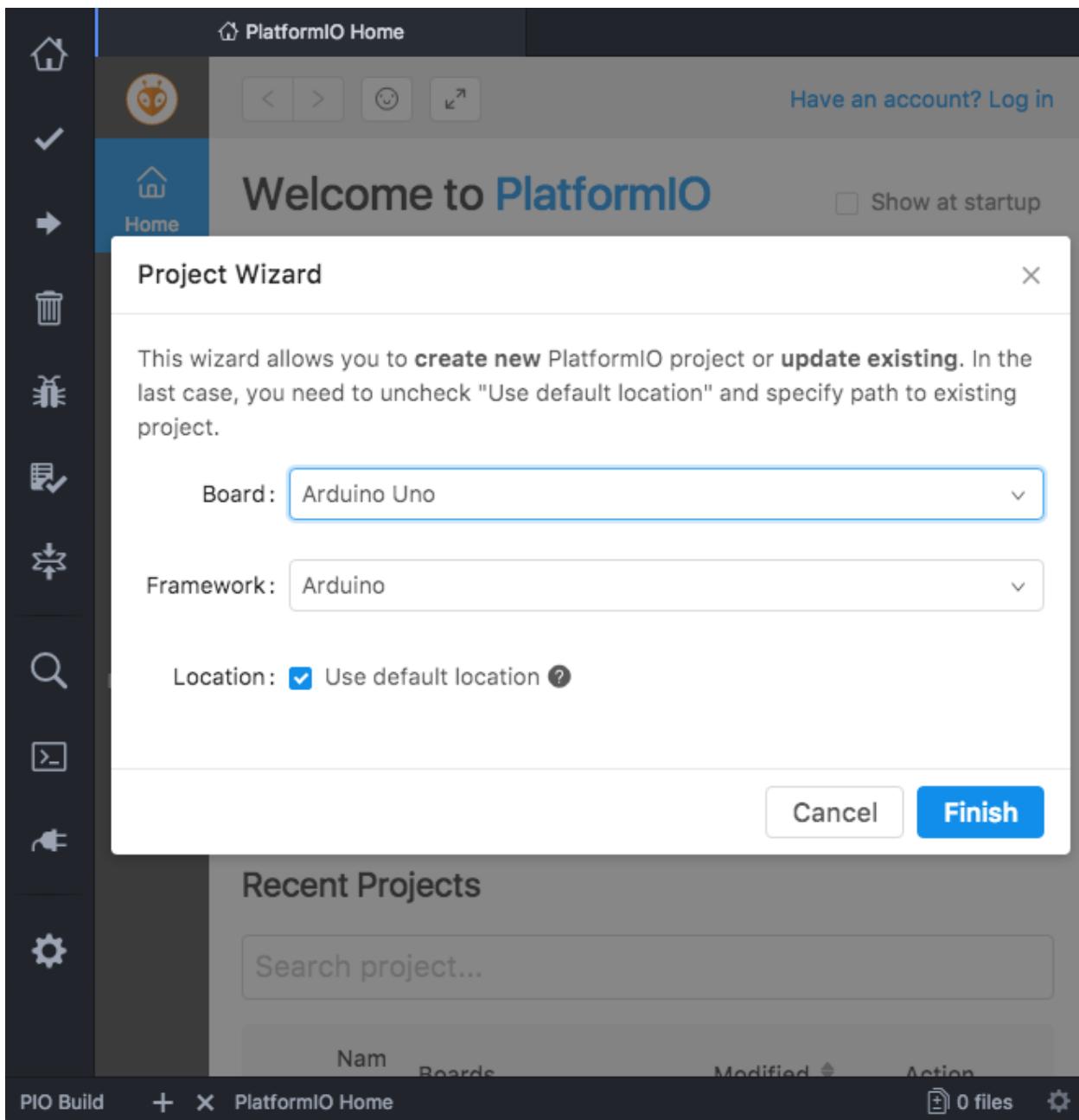
complete. Once finished, PlatformIO IDE will ask you to reload the Atom window to apply installed components. Please click on **Reload Now**. After that, PlatformIO IDE is ready for use. Happy coding!

Setting Up the Project

1. Click on the “PlatformIO Home” button on the *PlatformIO Toolbar*



2. Click on “New Project”, select a board and create a new PlatformIO Project



3. Open the `main.cpp` file in the `src` folder and replace its contents with the following:

Warning: The code below only works with Arduino-based boards. Please visit the [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)

(continued from previous page)

```
#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);
}
```

The screenshot shows the PlatformIO IDE interface. On the left is a toolbar with various icons: house, checkmark, right arrow, trash, bee, eye, download, magnifying glass, folder, plug, and gear. The main area has a title bar "Project" and a file tree under "170905-185203-un" containing "lib", "src" (with "main.cpp" and "platformio.ini"), and "platformio.ini". The code editor on the right displays the "main.cpp" file content:

```

1  /**
2  * Blink
3  *
4  * Turns on an LED on for one second,
5  * then off for one second, repeatedly.
6  */
7 #include "Arduino.h"
8
9 // Set LED_BUILTIN if it is not defined by Arduino.h
10 // #define LED_BUILTIN 13
11
12 void setup()
13 {
14     // initialize LED digital pin as an output.
15     pinMode(LED_BUILTIN, OUTPUT);
16 }
17
18 void loop()
19 {
20     // turn the LED on (HIGH is the voltage level)
21     digitalWrite(LED_BUILTIN, HIGH);
22
23     // wait for a second
24     delay(1000);
25
26     // turn the LED off by making the voltage LOW
27     digitalWrite(LED_BUILTIN, LOW);
28 }

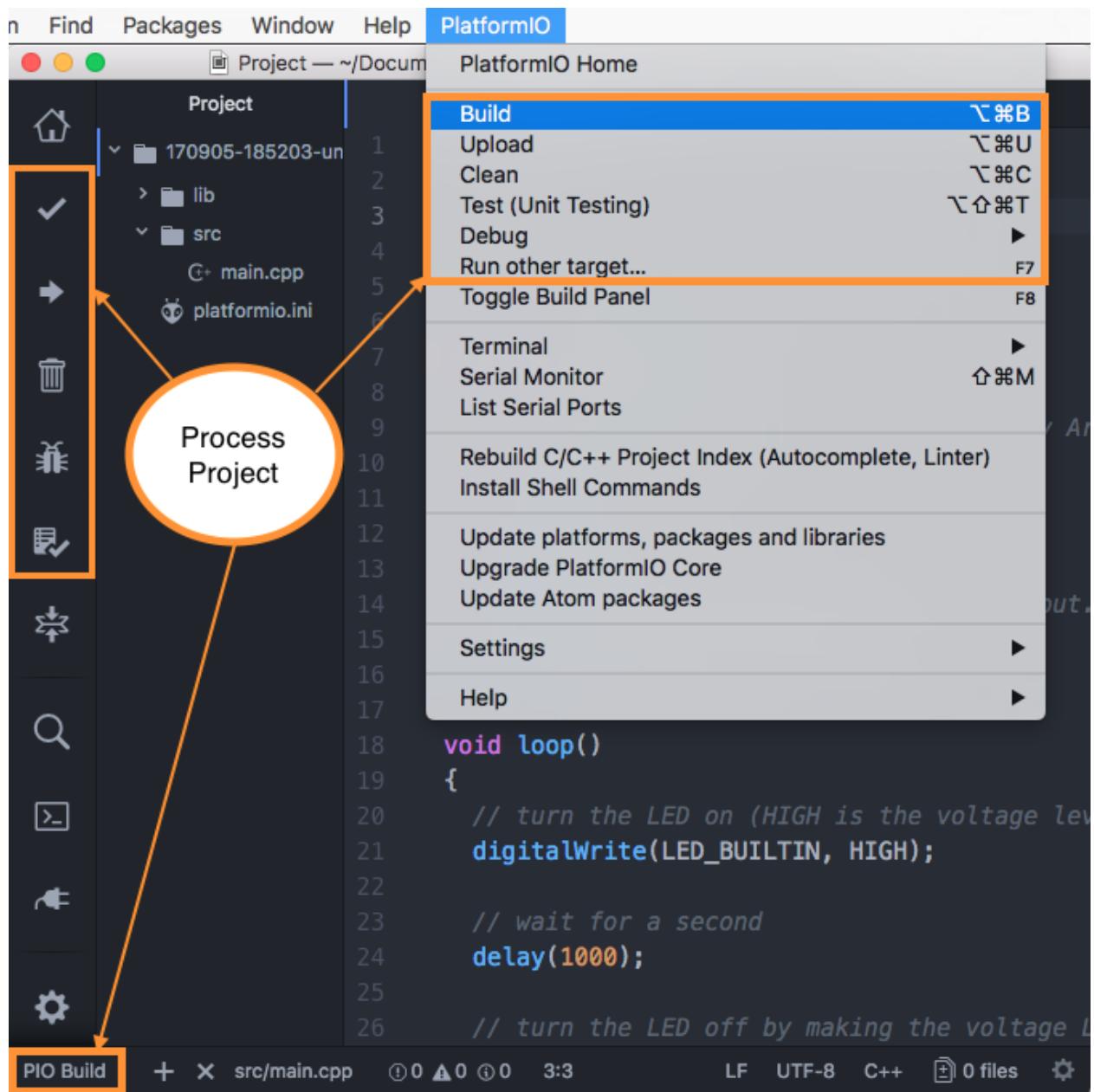
```

At the bottom, there are buttons for "PIO Build" (+), "src/main.cpp", build status (0 0 ▲ 0 0 0), file encoding (LF, UTF-8, C++), file count (0 files), and settings.

Process Project

PlatformIO IDE proposes different ways to process the project (build, clean, upload firmware, run other targets) using:

- *PlatformIO Toolbar*
- *Menu item PlatformIO*
- *Building / Uploading / Targets* and hotkeys



- Run Build and you should see a green “success” result in the build panel:

The screenshot shows the PlatformIO IDE interface. On the left is a toolbar with various icons: Home, Checkmark, Build (highlighted), Run, Delete, Find, Open, Save, and Settings. The central area is divided into sections: 'Project' (listing '170905-185203-un' with files 'src/main.cpp' and 'platformio.ini'), 'Code Editor' (displaying the content of 'src/main.cpp'), and a 'Terminal' or 'Output' window.

Code Editor:

```

1  /**
2  * Blink
3  *
4  * Turns on an LED on for one second,
5  * then off for one second, repeatedly.
6  */
7 #include "Arduino.h"
8
9 // Set LED_BUILTIN if it is not defined by Arduino
10 // #define LED_BUILTIN 13
11
12 void setup()
13 {
14
15 }

```

Output Window:

```

platformio run          5.0 s  ⚡  🗑  ✖
AVR Memory Usage
-----
Device: atmega328p

Program:    928 bytes (2.8% Full)
(.text + .data + .bootloader)

Data:        9 bytes (0.4% Full)
(.data + .bss + .noinit)

=====
[SUCCESS] Took 4.01 seconds

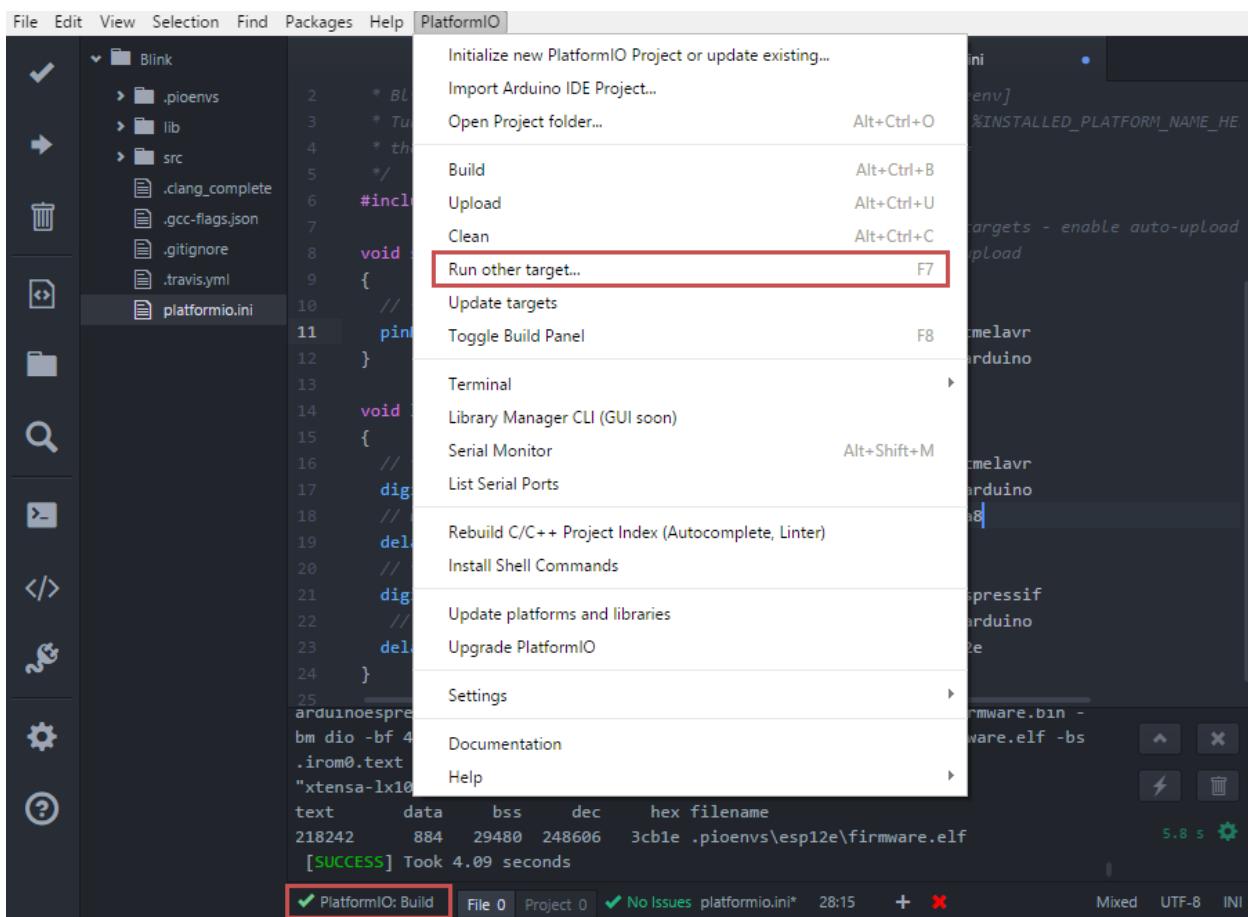
```

Status Bar:

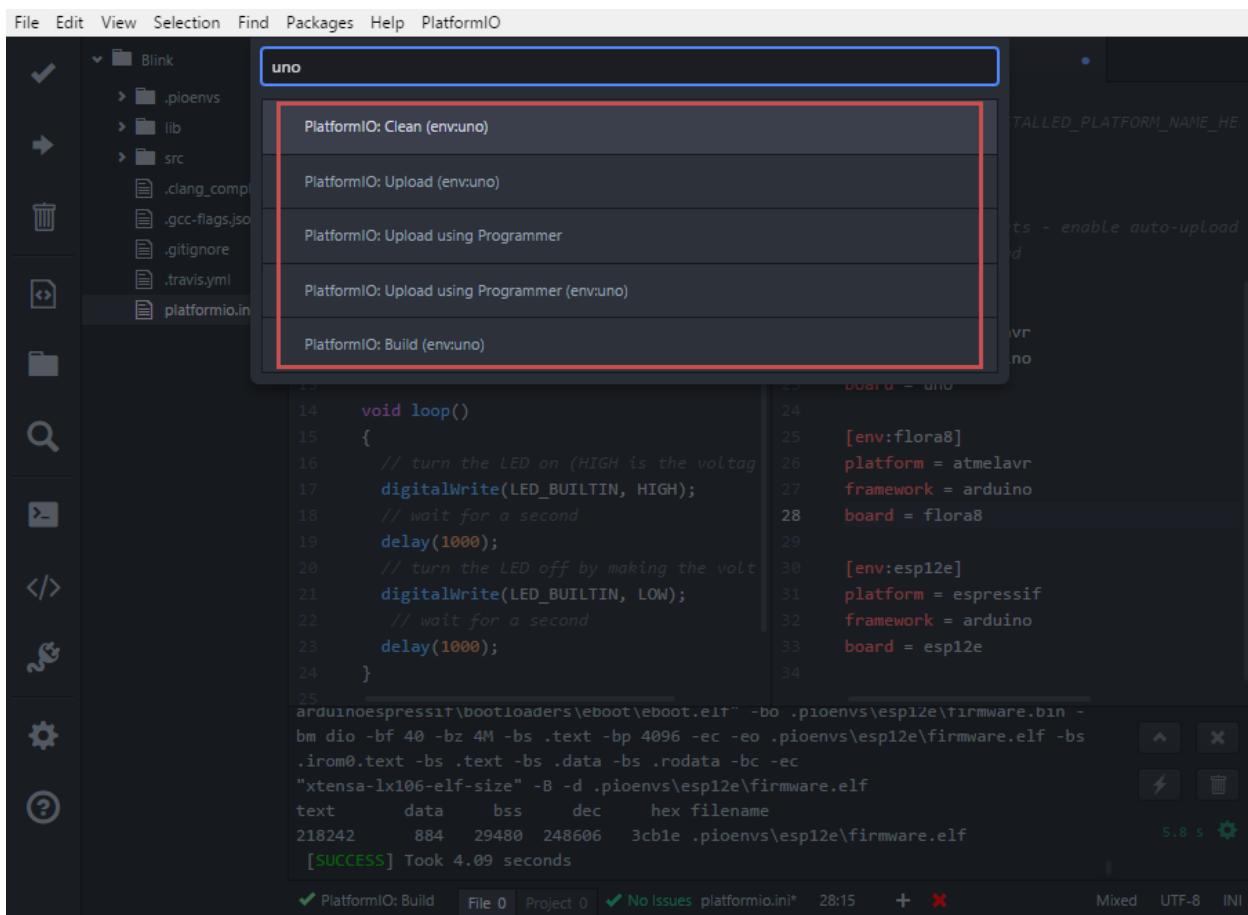
PIO Build + × src/main.cpp ⓘ 0 ▲ 0 ⓘ 0 3:3 LF UTF-8 C++ 0 files ⚙

To upload firmware to the board, run Upload.

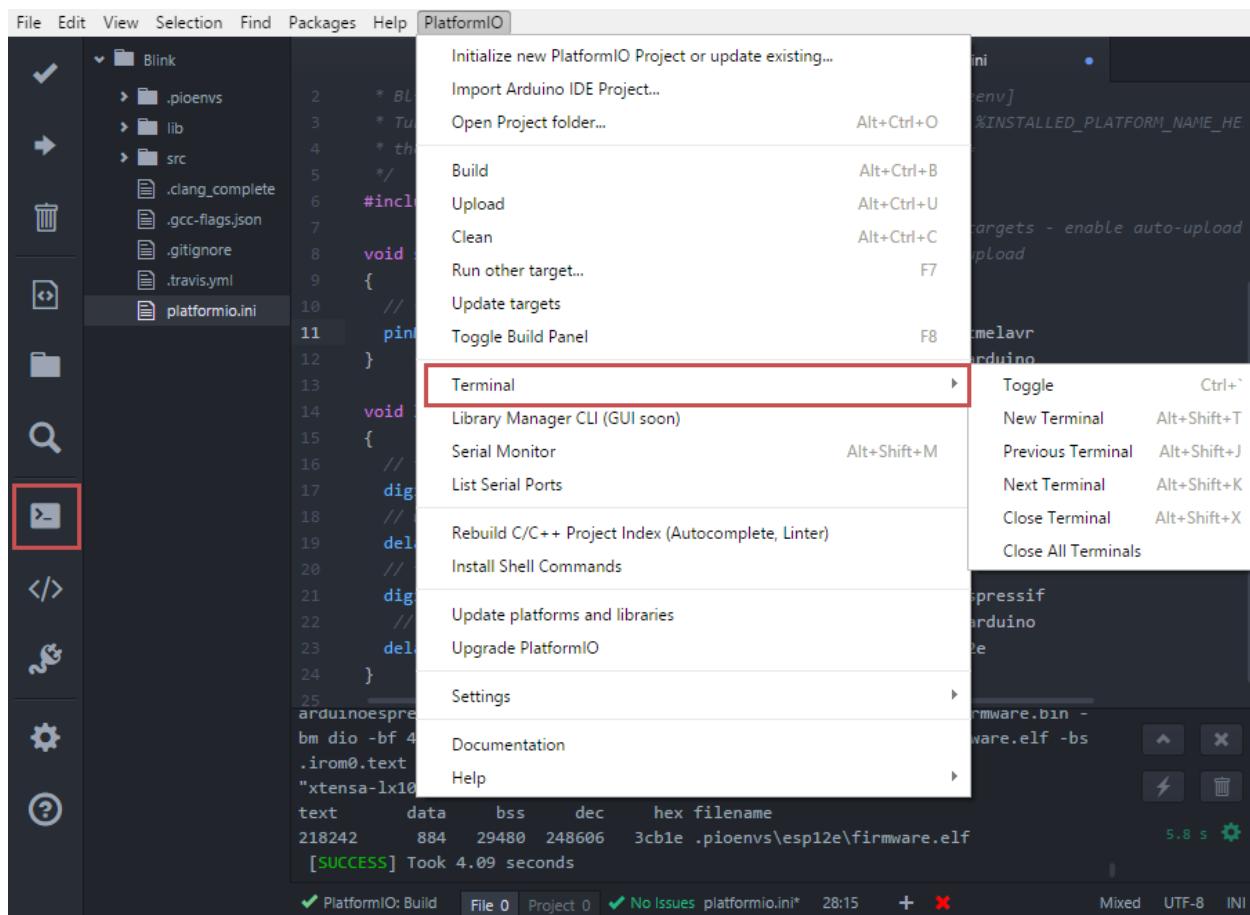
6. What is more, you can run specific target or process project environment using Menu: PlatformIO > Run other target... or call targets list from the status bar (bottom, left corner):



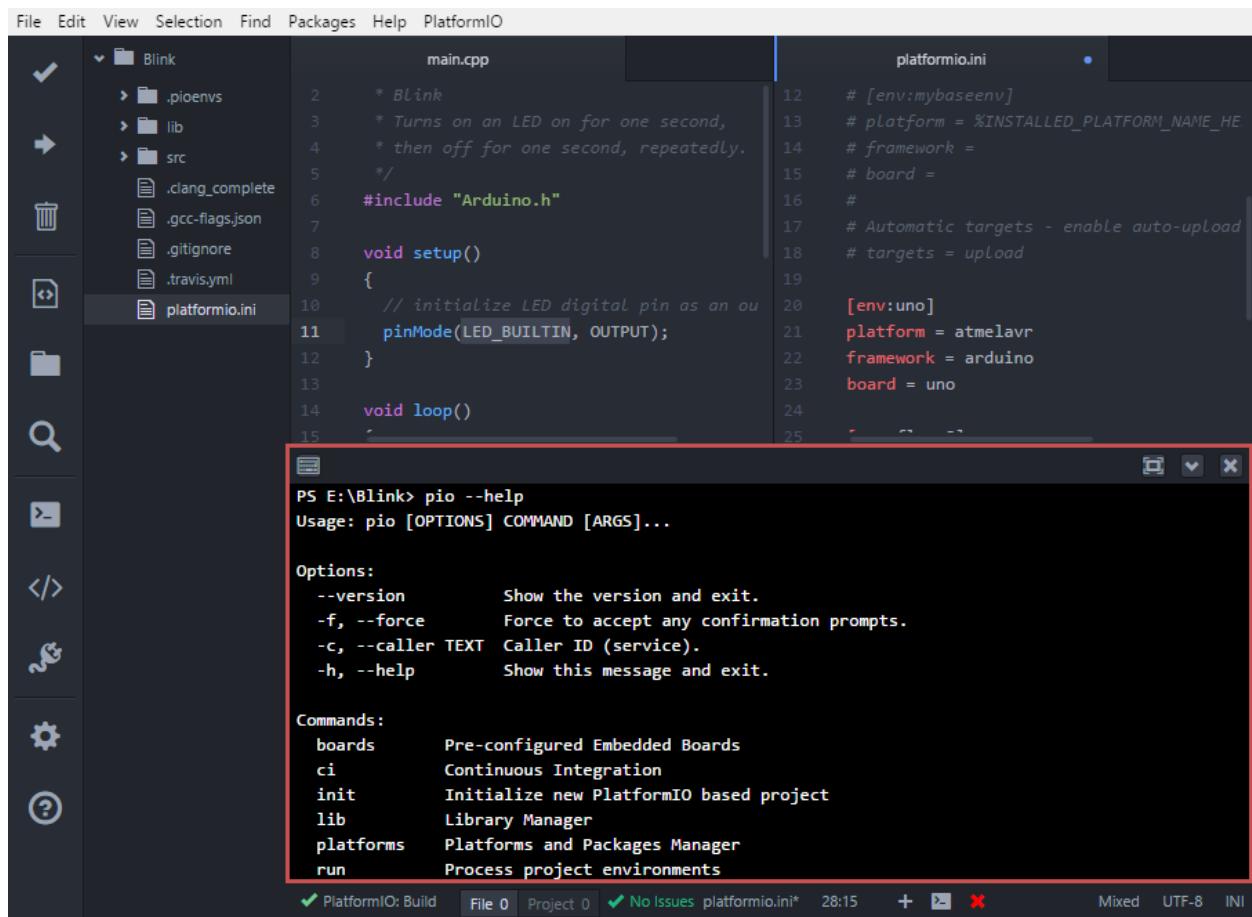
And select desired target:



- To launch the built-in terminal interface, choose Menu: PlatformIO > Terminal or press the corresponding icon in the PlatformIO toolbar:



This provides you fast access to a set of powerful *PlatformIO Core (CLI)* CLI commands:



The screenshot shows the PlatformIO IDE interface. On the left is a toolbar with various icons. The main area has a file tree on the left showing a project named 'Blink' with files like .pioenvs, lib, src, .clang_complete, .gcc-flags.json, .gitignore, .travis.yml, and platformio.ini. The central editor window displays the main.cpp file with code for a blinking LED. To the right is the platformio.ini configuration file. Below these is a terminal window showing the output of the command 'pio --help'. The status bar at the bottom indicates 'PlatformIO: Build' and 'File 0'.

```

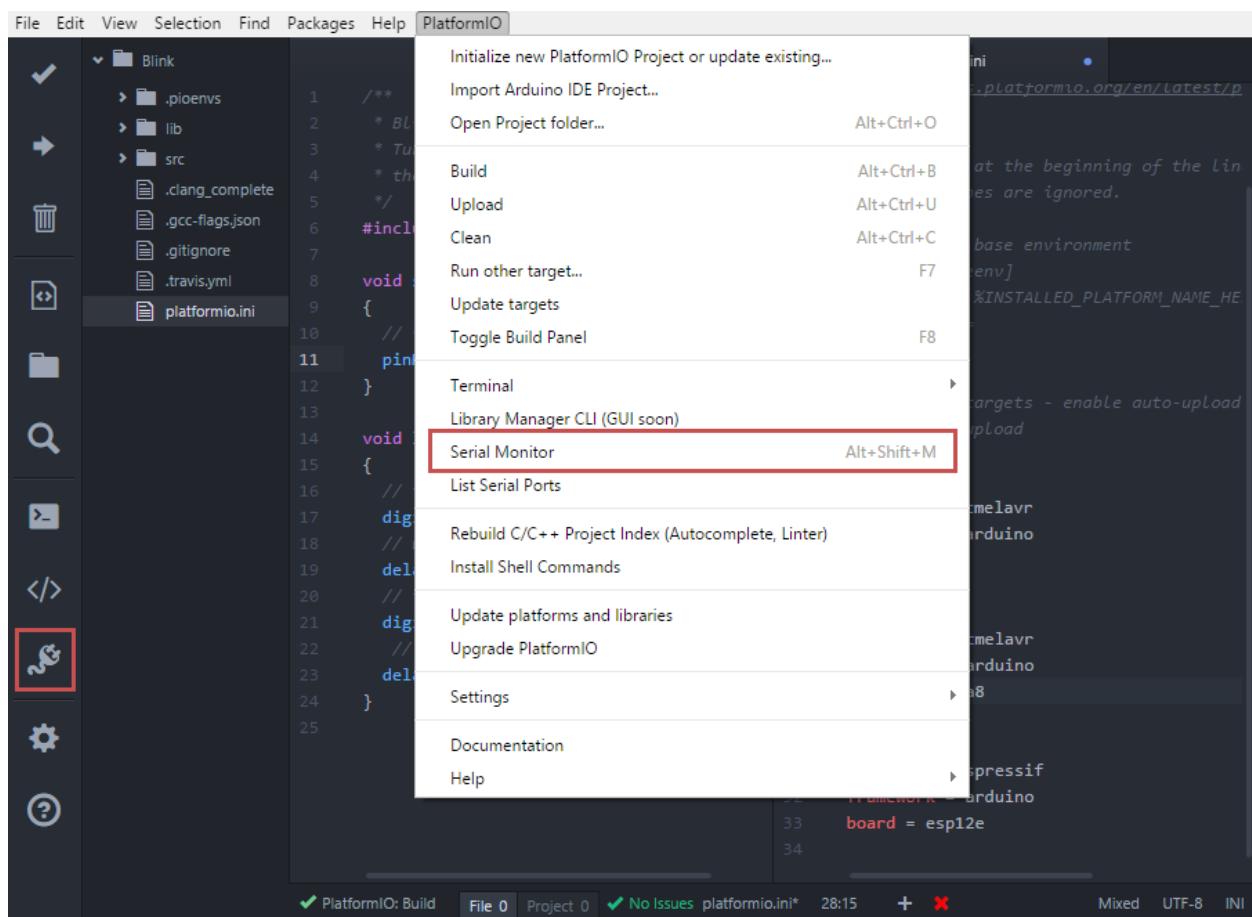
PS E:\Blink> pio --help
Usage: pio [OPTIONS] COMMAND [ARGS]...

Options:
  --version      Show the version and exit.
  -f, --force    Force to accept any confirmation prompts.
  -c, --caller TEXT Caller ID (service).
  -h, --help     Show this message and exit.

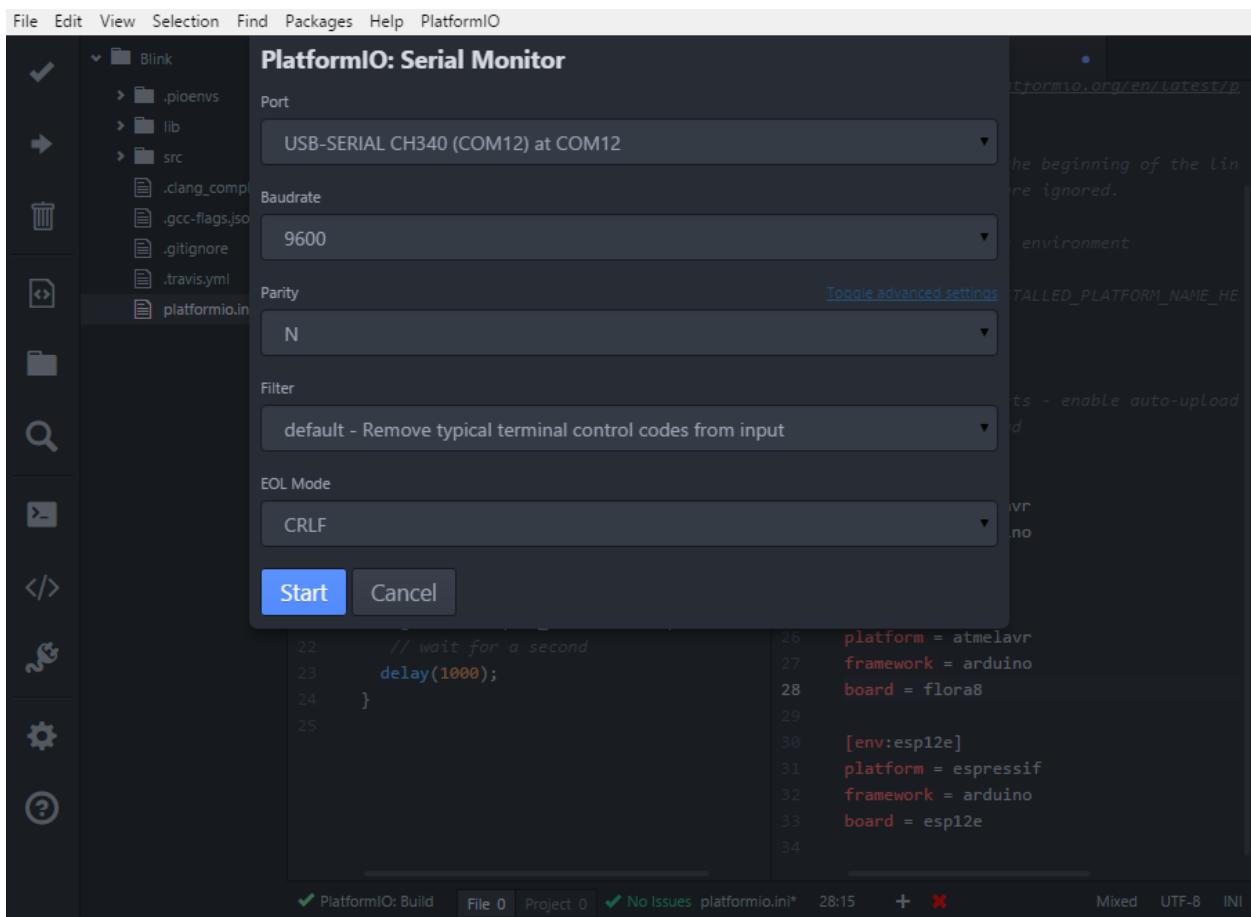
Commands:
  boards      Pre-configured Embedded Boards
  ci          Continuous Integration
  init        Initialize new PlatformIO based project
  lib         Library Manager
  platforms   Platforms and Packages Manager
  run         Process project environments

```

8. To run the built-in “Serial Monitor”, choose Menu: PlatformIO > Serial Monitor or press the corresponding icon in the PlatformIO toolbar:



The monitor has several settings to adjust your connection:



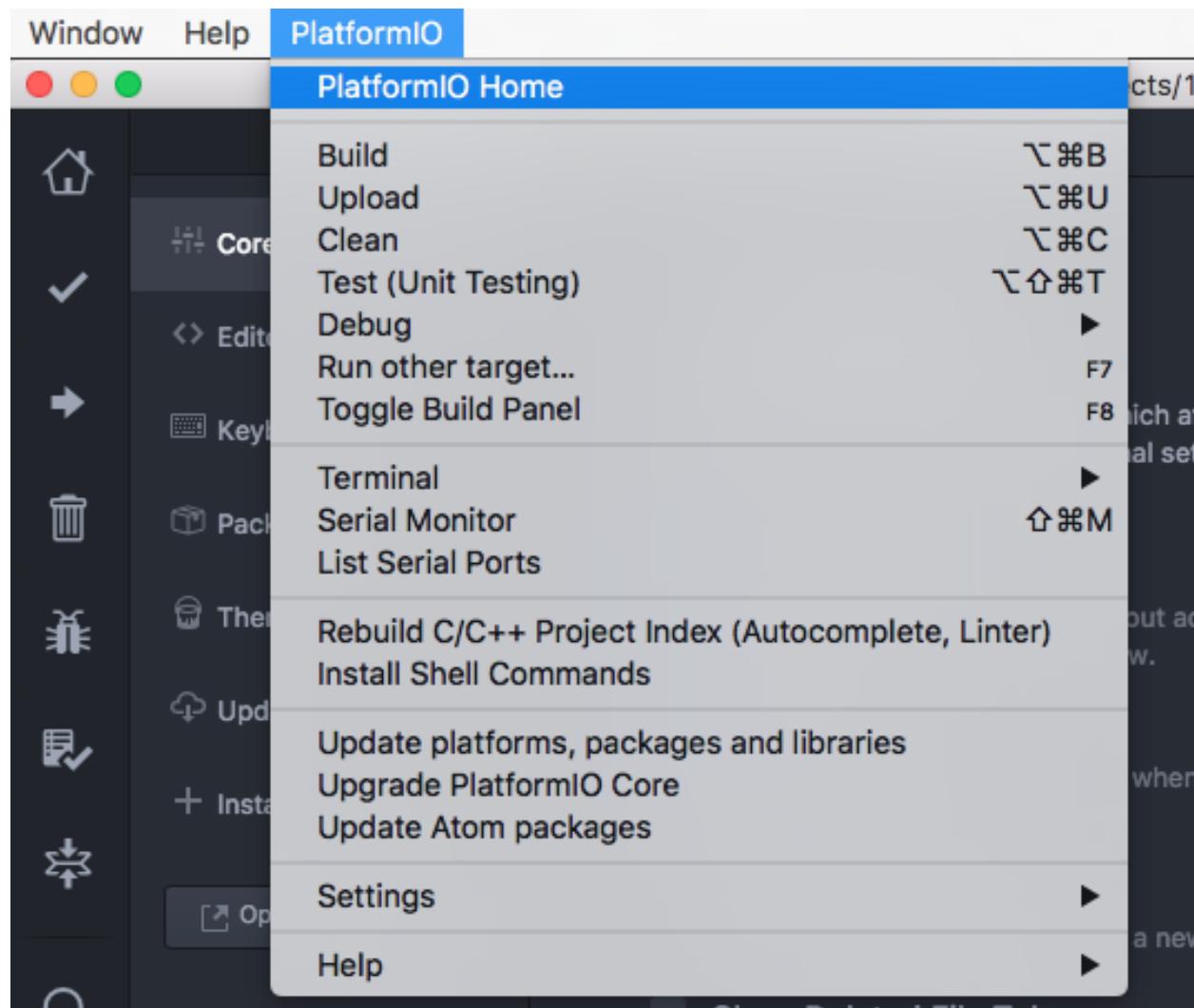
It also allows you to communicate with your board in an easy way:

The screenshot shows the PlatformIO IDE integrated into the Atom text editor. The interface includes:

- File Bar:** File, Edit, View, Selection, Find, Packages, Help, PlatformIO.
- Sidebar:** Project navigation with a tree view showing a project named "Blink" containing ".pioenvs", "lib", "src", ".clang_complete", ".gcc-flags.json", ".gitignore", ".travis.yml", and "platformio.ini".
- Code Editor:** Two tabs are open: "main.cpp" and "platformio.ini". The "main.cpp" tab contains Arduino code for a blinking LED. The "platformio.ini" tab contains the configuration file for the project.
- Terminal:** A central panel showing the output of a command-line session (PS E:\Blink> ...).
- Status Bar:** Shows "PlatformIO: Build", "File 0", "Project 0", "No Issues", "platformio.ini*", "28:15", and various icons for file operations.

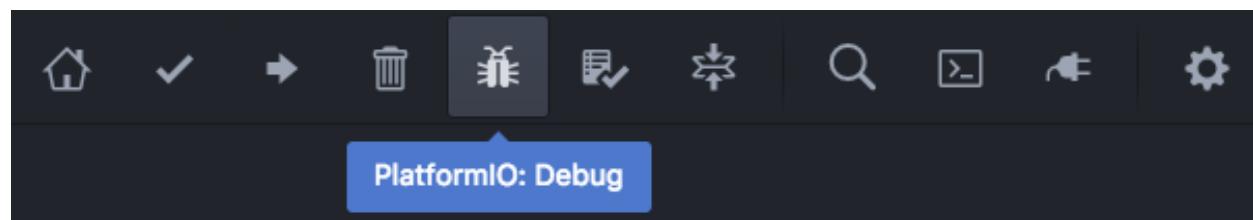
Menu item PlatformIO

platformio-ide package adds to Atom new menu item named **Menu: PlatformIO** (after **Menu: Help** item).



PlatformIO Toolbar

PlatformIO IDE Toolbar contains quick access buttons for the popular commands. Each button contains a hint (leave the mouse on it for a moment).



- [PlatformIO Home](#)
- PlatformIO: Build
- PlatformIO: Upload
- PlatformIO: Clean
- [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 see the step [II. Clang for Intelligent Code Completion](#) from Installation guide.

Warning: The libraries which are added/installed after the initializing process will not be reflected in the code linter. You need Menu: PlatformIO > Rebuild C/C++ Project Index (Autocomplete, Linter).

Smart Code Linter

PlatformIO IDE uses PlatformIO's pre-built GCC toolchains for Smart Code Linter and rapid professional development. The configuration data are located in `.gcc-flags.json`. This file will be automatically created and preconfigured when you initialize project using Menu: PlatformIO > Initialize new PlatformIO Project or update existing....

Warning: If some libraries are not visible in [PlatformIO IDE for Atom](#) and Code Completion or Code Linting does not work properly, please perform Menu: PlatformIO > Rebuild C/C++ Project Index (Autocomplete, Linter)

Install Shell Commands

Please navigate to 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:

```
{  
    "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/
```

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 - **AI 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 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 12, 2016 - **Peter Marks** - PlatformIO, the Arduino IDE for programmers
- 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)

See a full list with [Articles about us](#).

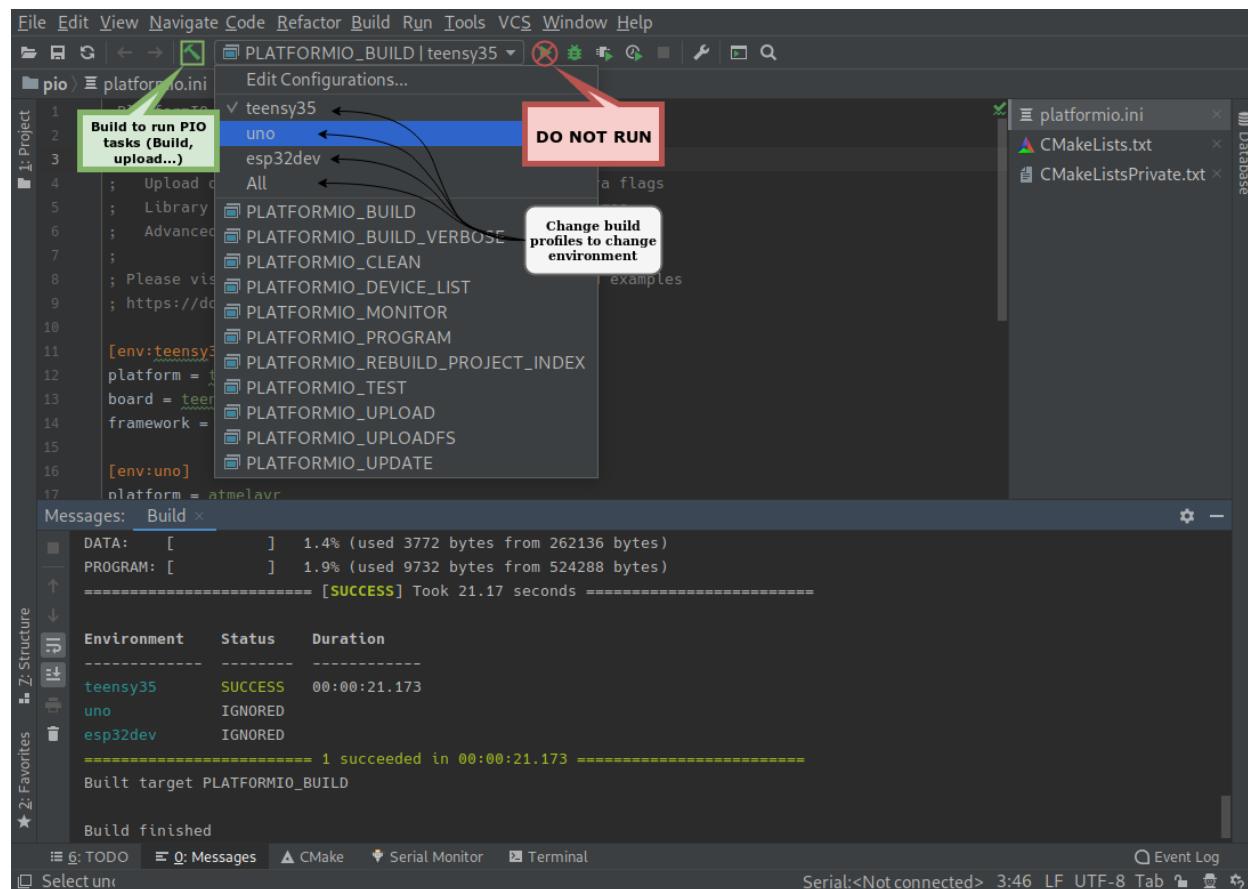
Changelog

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. Open 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 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. Open this project via Menu: File > Open... 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 11 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.
- PLATFORMIO_DEVICE_LIST - List connected devices.

If you have multiple environments, you can select which one the target is going to use by changing the build profile (See screenshot). Changing the build profile also updates defines and includes for code completion in the editor to those specified by the environment.

The profile All runs the target for all environments ; this was the previous behavior.

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

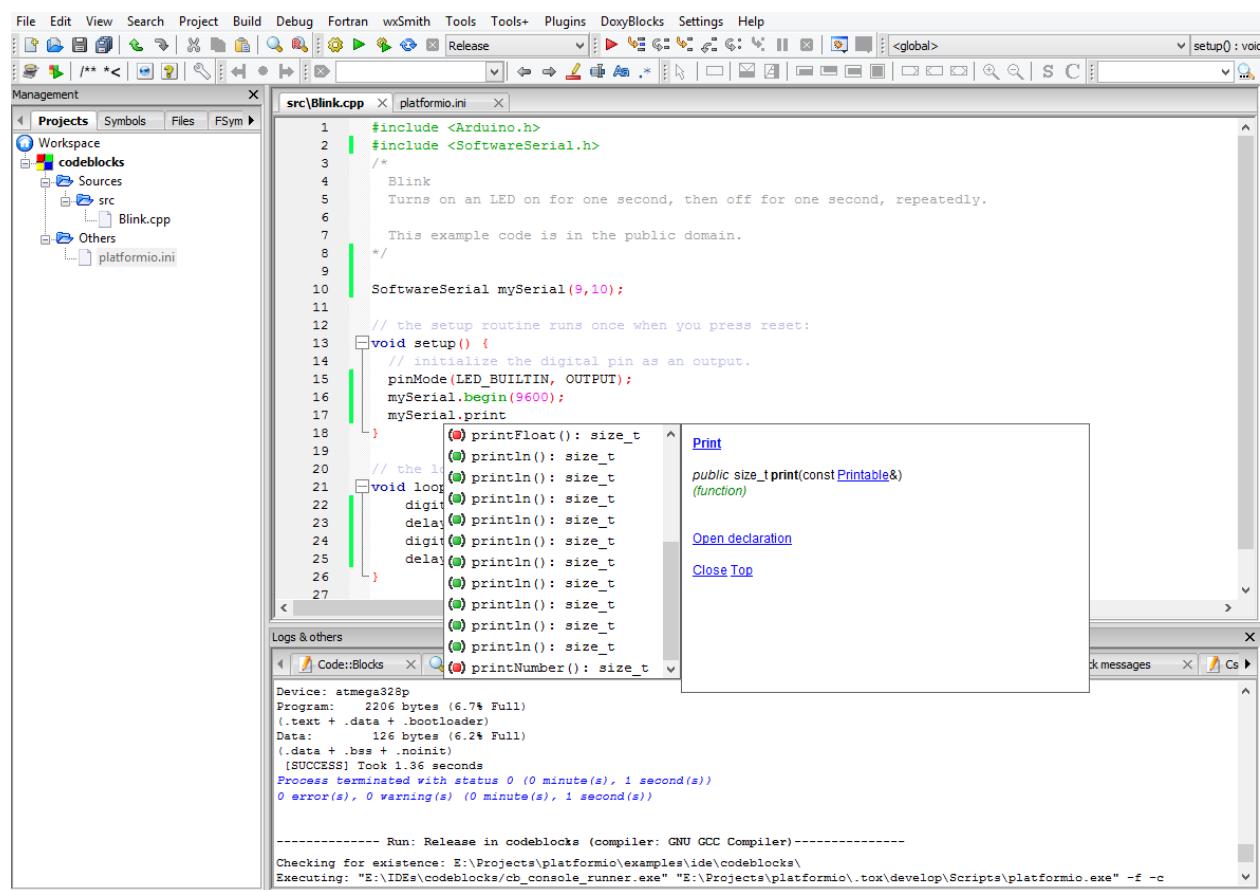
- Dec 01, 2015 - **JetBrains CLion Blog** - C++ Annotated: Fall 2015. Arduino Support in CLion using PlatformIO
- Nov 22, 2015 - **Michał Seroczyński** - Using PlatformIO to get started with Arduino in CLion IDE
- Nov 09, 2015 - **Álvaro García Gómez** - Programar con Arduino “The good way” (Programming 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:

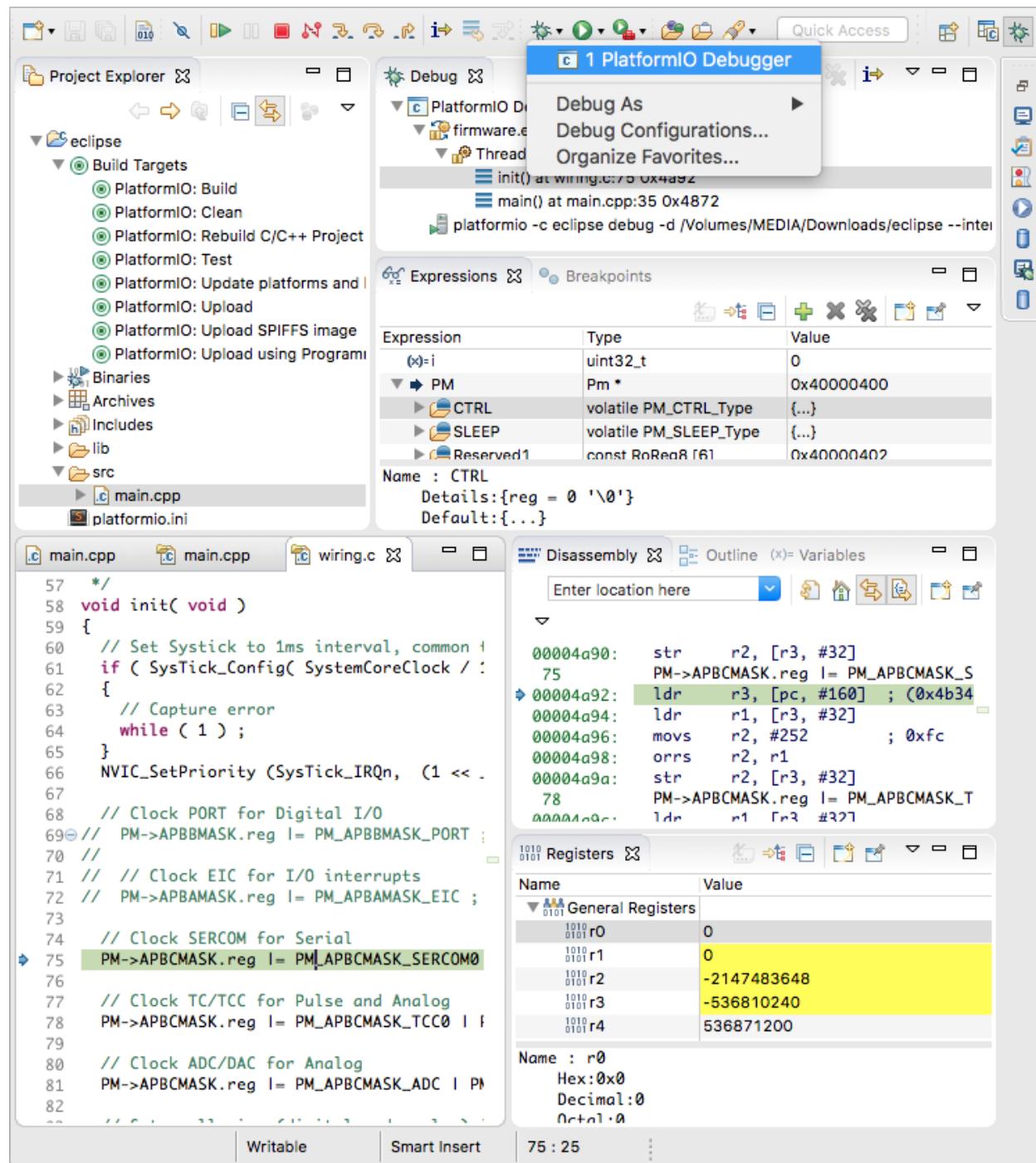
1. Open this project via Menu: File > Open...
2. Add new files to `src` directory (`*.c`, `*.cpp`, `*.ino`, etc.) via Menu: File > New > File..
3. Build project using Menu: Build > Build
4. Upload firmware using Menu: Build > Run

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.

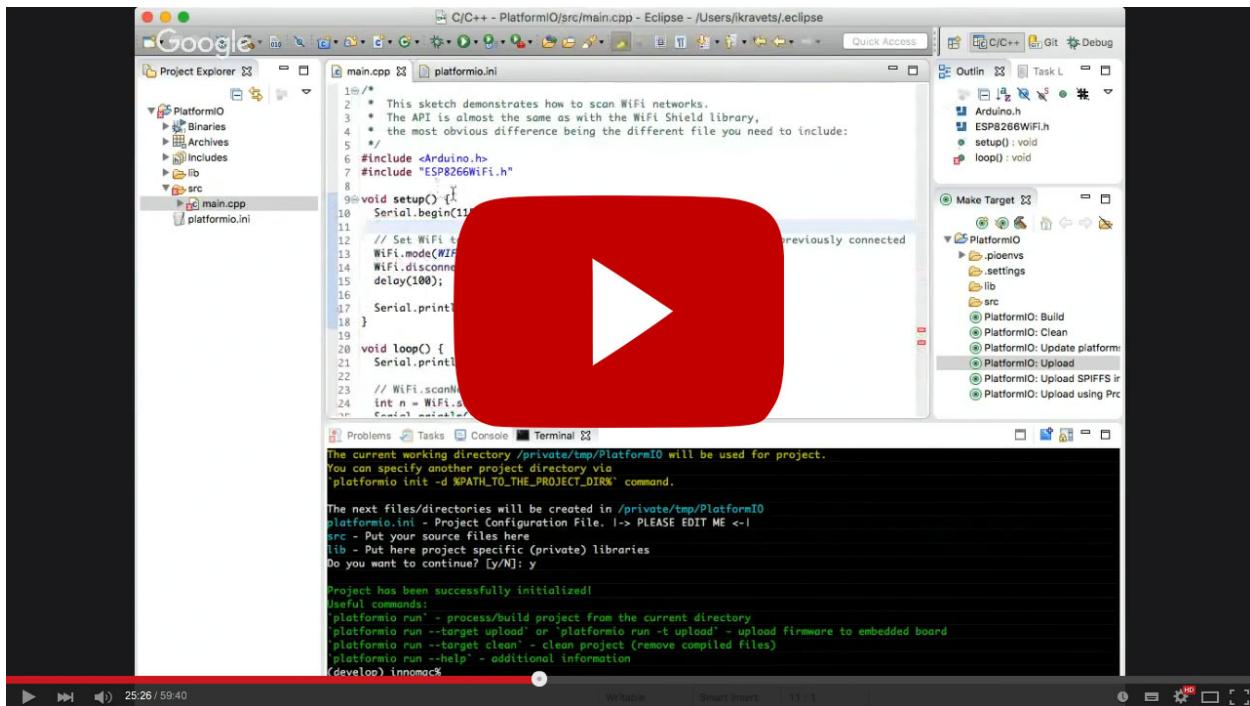
Warning: The libraries which are added, installed or used in the project after generating process won't be reflected in IDE. To fix it please run PlatformIO: Rebuild C/C++ Project Index target and right click on the project and Project > Refresh (F5).

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!](#)

- Sep 01, 2015 - **Thomas P. Weldon, Ph.D.** - Improvised MBED FRDM-K64F Eclipse/PlatformIO Setup and Software Installation
- Jul 11, 2015 - **TrojanC** - Learning Arduino GitHub Repository
- June 20, 2014 - **Ivan Kravets, Ph.D.** - Building and debugging Atmel AVR (Arduino-based) project using Eclipse IDE+PlatformIO

See a full list with [Articles about us](#).

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.

```

avr-gcc -c -Os -Wall -ffunction-sections -fdata-sections -MMD -mmcu=atmega328p -DF_CPU=16000000L -DARDUINO=10607 -I. -I.pioenvs/uno/FrameworkArduino/Wiring_digital.c -I.pioenvs/uno/FrameworkArduino/Wiring_pulse.c -I.pioenvs/uno/FrameworkArduino/Wiring_shift.c -I.pioenvs/uno/FrameworkArduino/Variant.c -I.pioenvs/uno/FrameworkArduino/HardwareSerial1.c -I.pioenvs/uno/FrameworkArduino/HardwareSerial2.c -I.pioenvs/uno/FrameworkArduino/PluggableUSB.o -I.pioenvs/uno/FrameworkArduino/Tone.o -I.pioenvs/uno/FrameworkArduino/Stream.o -I.pioenvs/uno/FrameworkArduino/Tone.o -I.pioenvs/uno/FrameworkArduino/USBCore.o -I.pioenvs/uno/FrameworkArduino/Interrupts.o -I.pioenvs/uno/FrameworkArduino/Math.o -I.pioenvs/uno/FrameworkArduino/String.o -I.pioenvs/uno/FrameworkArduino/_r�Arduino/_wiring_pulse.o -I.pioenvs/uno/FrameworkArduino/abi.o -I.pioenvs/uno/FrameworkArduino/main.o -I.pioenvs/uno/FrameworkArduino/new.o -I.pioenvs/uno/FrameworkArduino/wiring_digital.o -I.pioenvs/uno/FrameworkArduino/wiring_pulse.o -I.pioenvs/uno/FrameworkArduino/wiring_shift.o -I.pioenvs/uno/libFrameworkArduino.a -I.pioenvs/uno/fwfirmware.elf -Os -mmcu=atmega328p -Wl,--gc-sections,--relax -pie -I.pioenvs/uno/src/blink.o -L/Users/zach/.platformio/packages/ldscripts -L.pioenvs/uno -Wl,-start-group -lm -I.pioenvs/uno/libFrameworkArduinoVariant.a -I.pioenvs/uno/libFrameworkArduino.a -Wl,-end-group -avr-objcopy -O ihex -R .eeprom -I.pioenvs/uno/firmware.elf -I.pioenvs/uno/firmware.hex -avr-size" --mcu=atmega328p -C -d -I.pioenvs/uno/firmware.elf
AVR Memory Usage
Device: atmega328p
Program: 998 bytes (3.0% Full)
(.text + .data + .bootloader)
Data: 9 bytes (0.4% Full)
(.data + .bss + .noinit)

=====
[BUILD SUCCESS] Took 1.89 seconds
Built target platformio_build
Compilation finished at Fri Dec 25 23:08:44

```

13:12 U-[blink]src/blink.cpp All C++/LPlatformIO pair ws yas +
digitalWrite(uint8_t, uint8_t) = void 55: 0 -R-*compilation* Bot: Compilation:exit [0] Projectile

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.

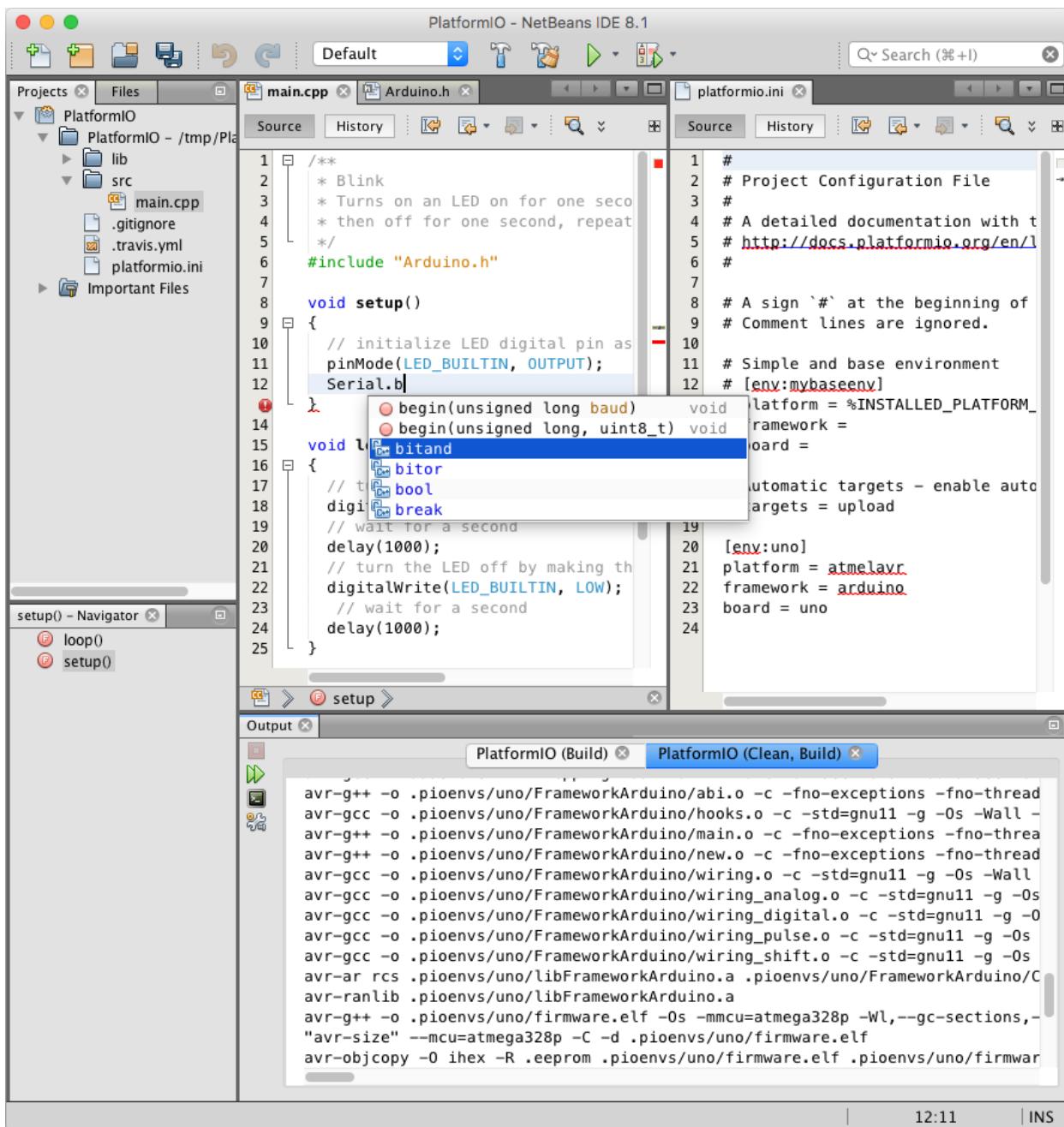
- `platformio_build` - Build project without auto-uploading. (`C-c i b`)
- `platformio_clean` - Clean compiled objects. (`C-c i c`)
- `platformio_upload` - Build and upload (if no errors). (`C-c i u`)
- `platformio_programmer_upload` - Build and upload using external programmer (if no errors, see [Upload 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

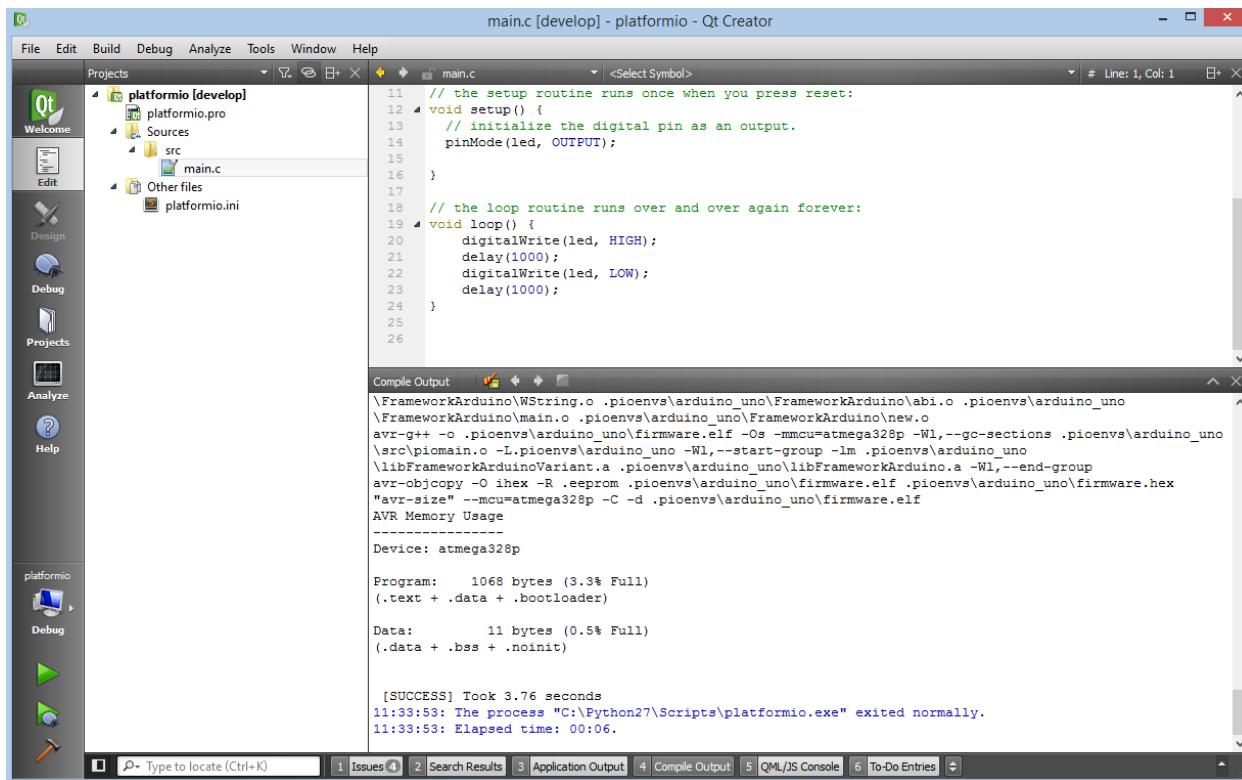
- Feb 22, 2016 - **Grzegorz Holdys** - How to Integrate PlatformIO with Netbeans

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.



Contents

- *Qt Creator*
 - *Integration*
 - * *Project Generator*
 - * *Manual Integration*
 - *Setup New Project*
 - *First program in Qt Creator*
 - *Conclusion*

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 qtcreator --board <ID>

# For example, generate project for Arduino Uno
platformio init --ide qtcreator --board uno
```

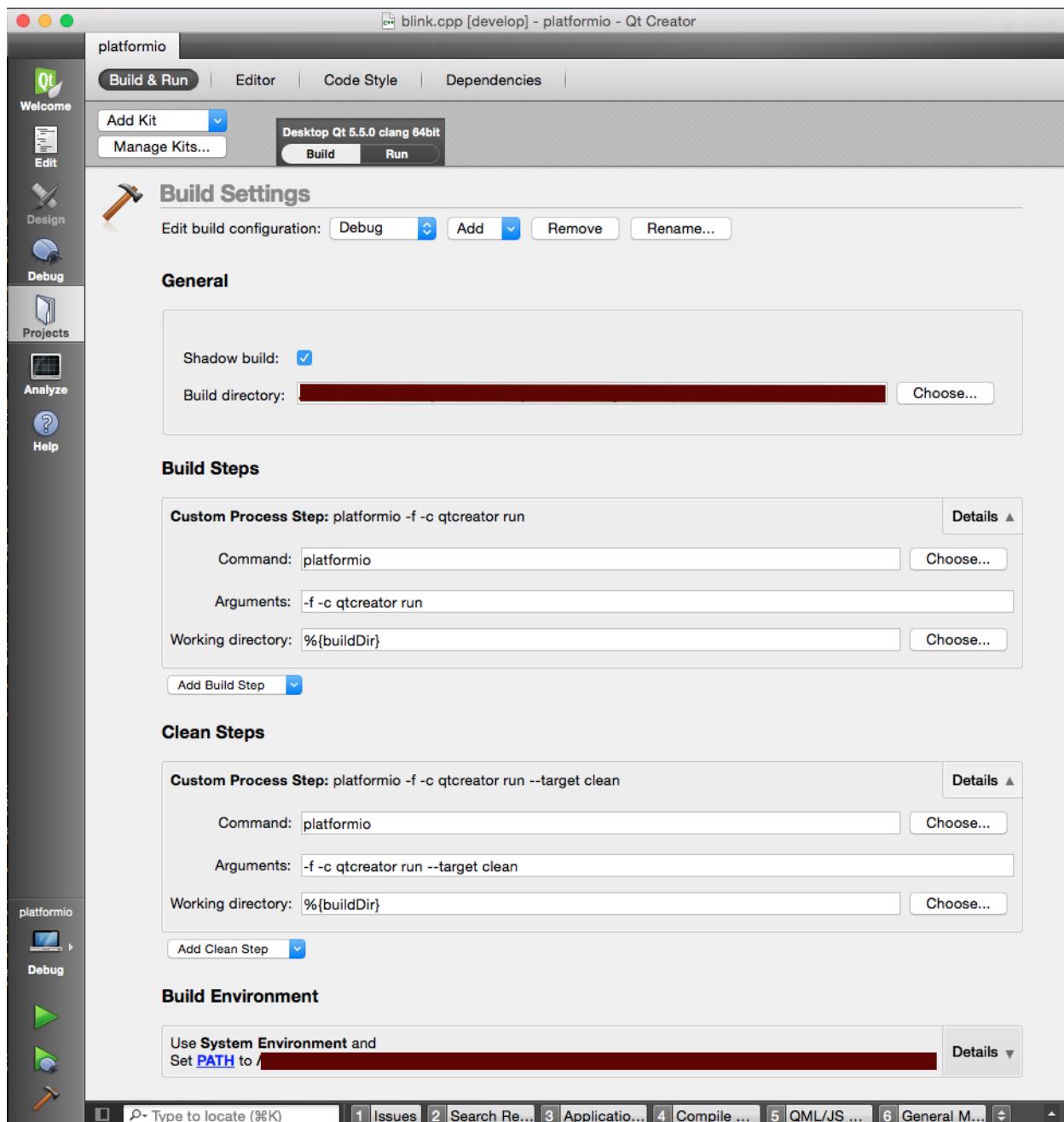
Then:

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):

```
# 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.



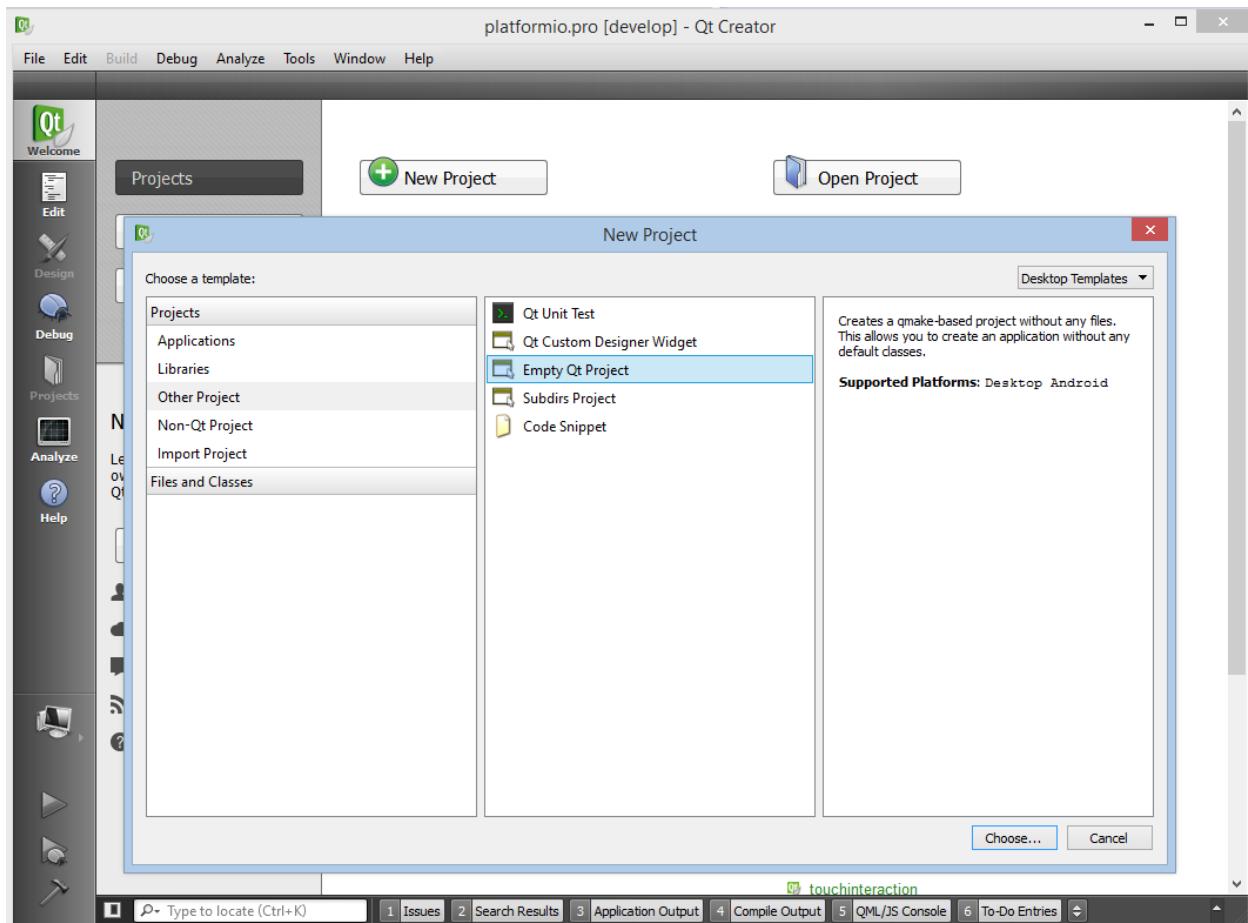
Warning: The libraries which are added, installed or used in the project after generating process won't be reflected in IDE. To fix it you need to reinitialize project using `platformio init` (repeat it).

Manual Integration

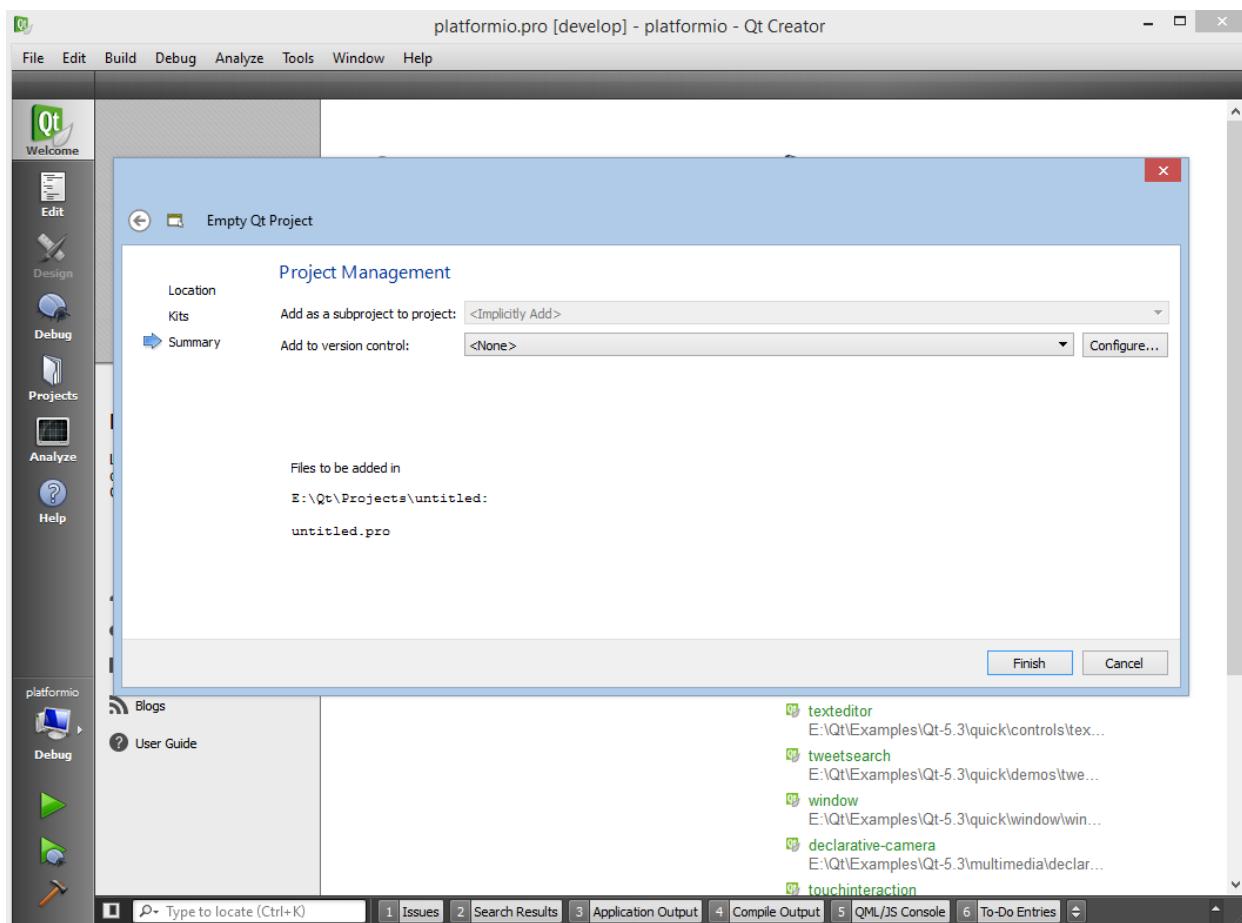
Setup New Project

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`

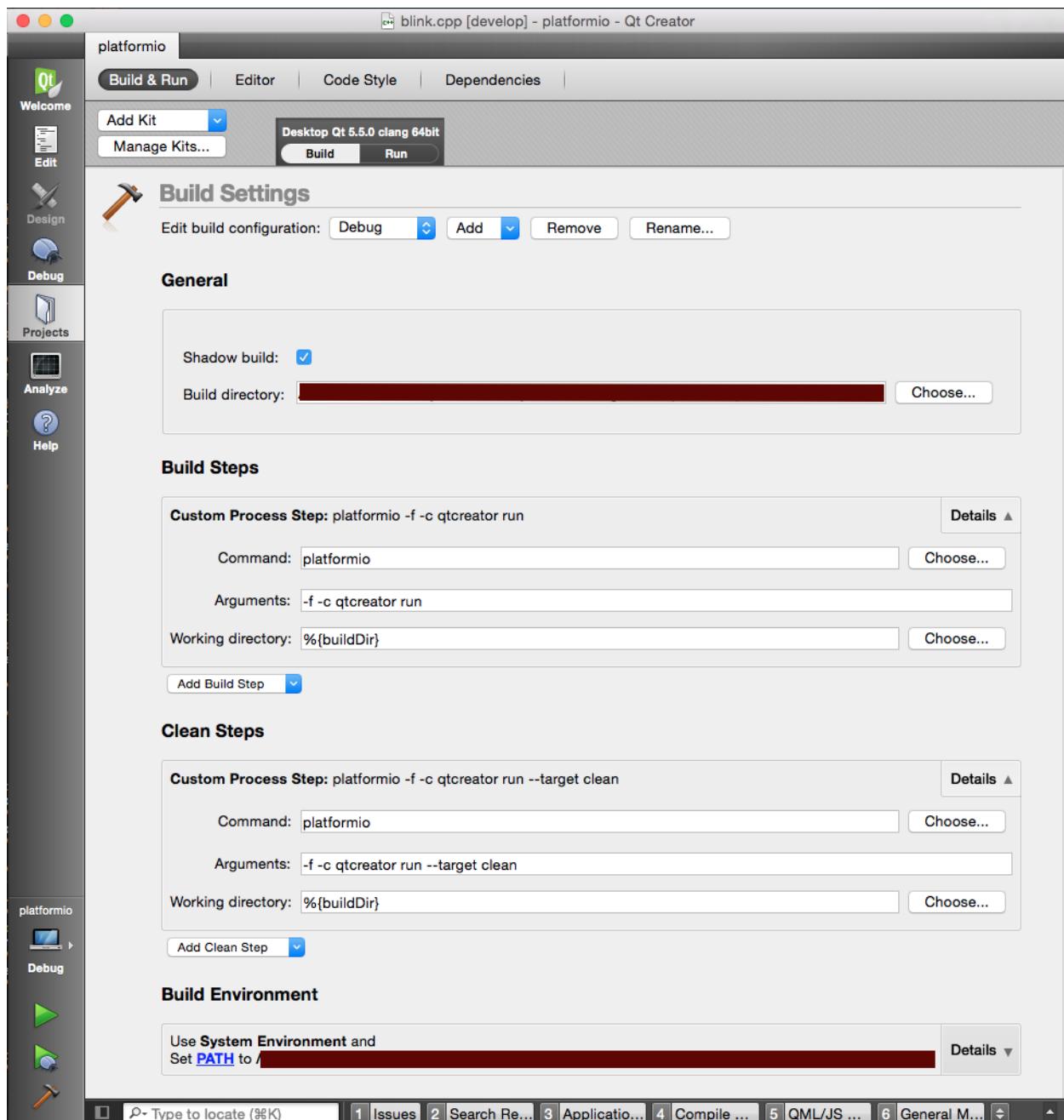
Project), fill Name, Create in.



On the next steps select any available kit and click Finish button.



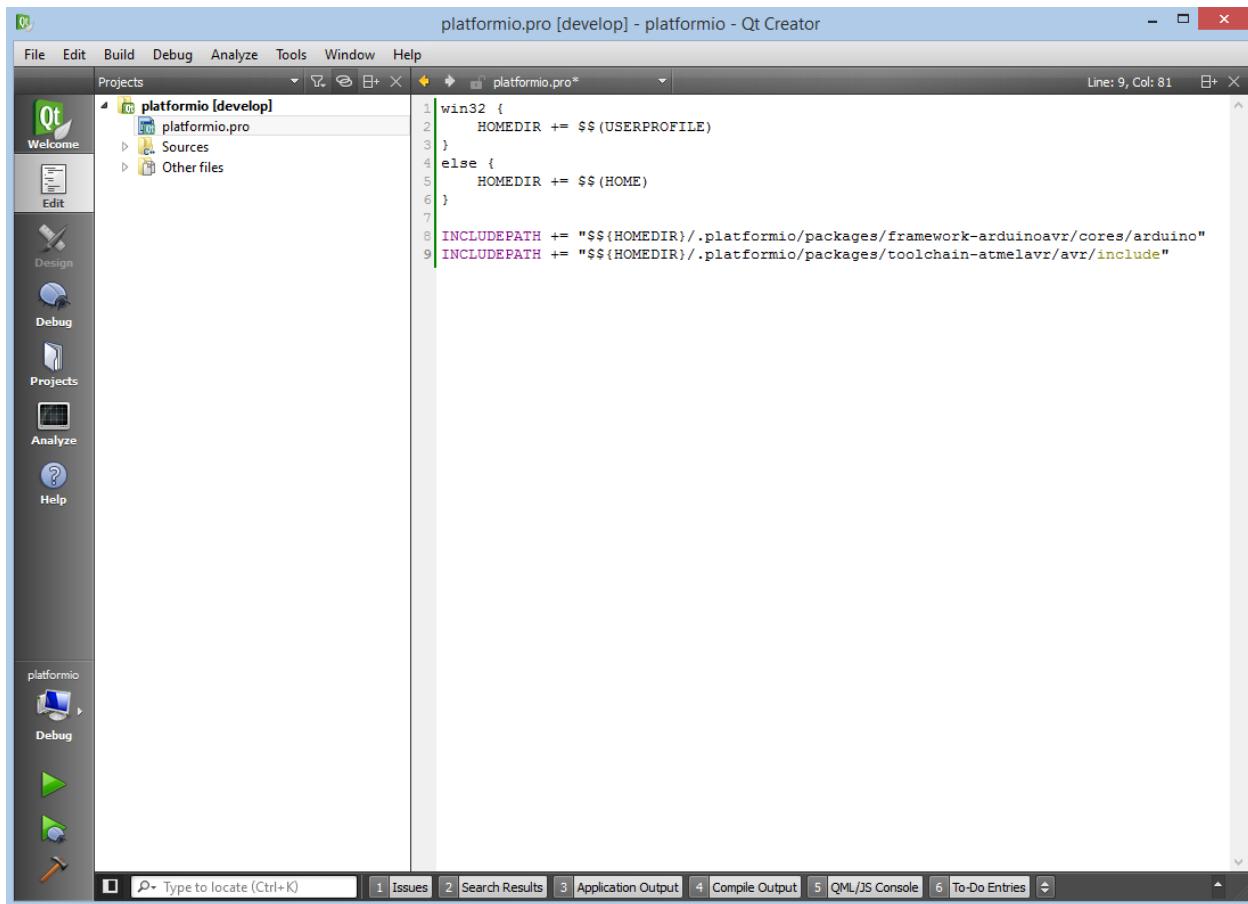
Secondly, we need to delete default build and clean steps and configure project with PlatformIO Build System (click on Projects label on left menu or **Ctrl+5** shortcut):



Thirdly, change project file by adding path to directories with header files. Please edit project file to match the following contents:

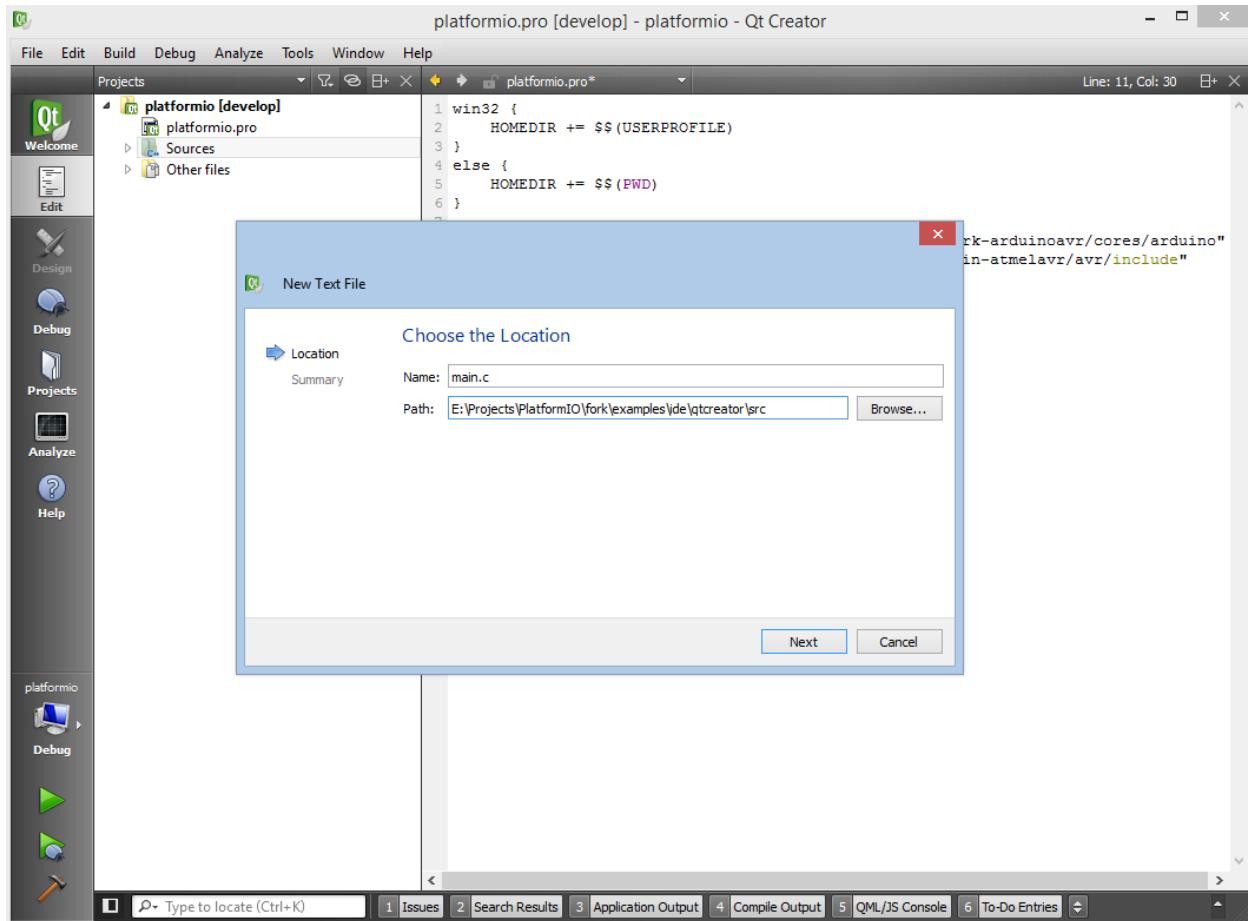
```
win32 {
    HOMEDIR += $$ (USERPROFILE)
}
else {
    HOMEDIR += $$ (HOME)
}

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:



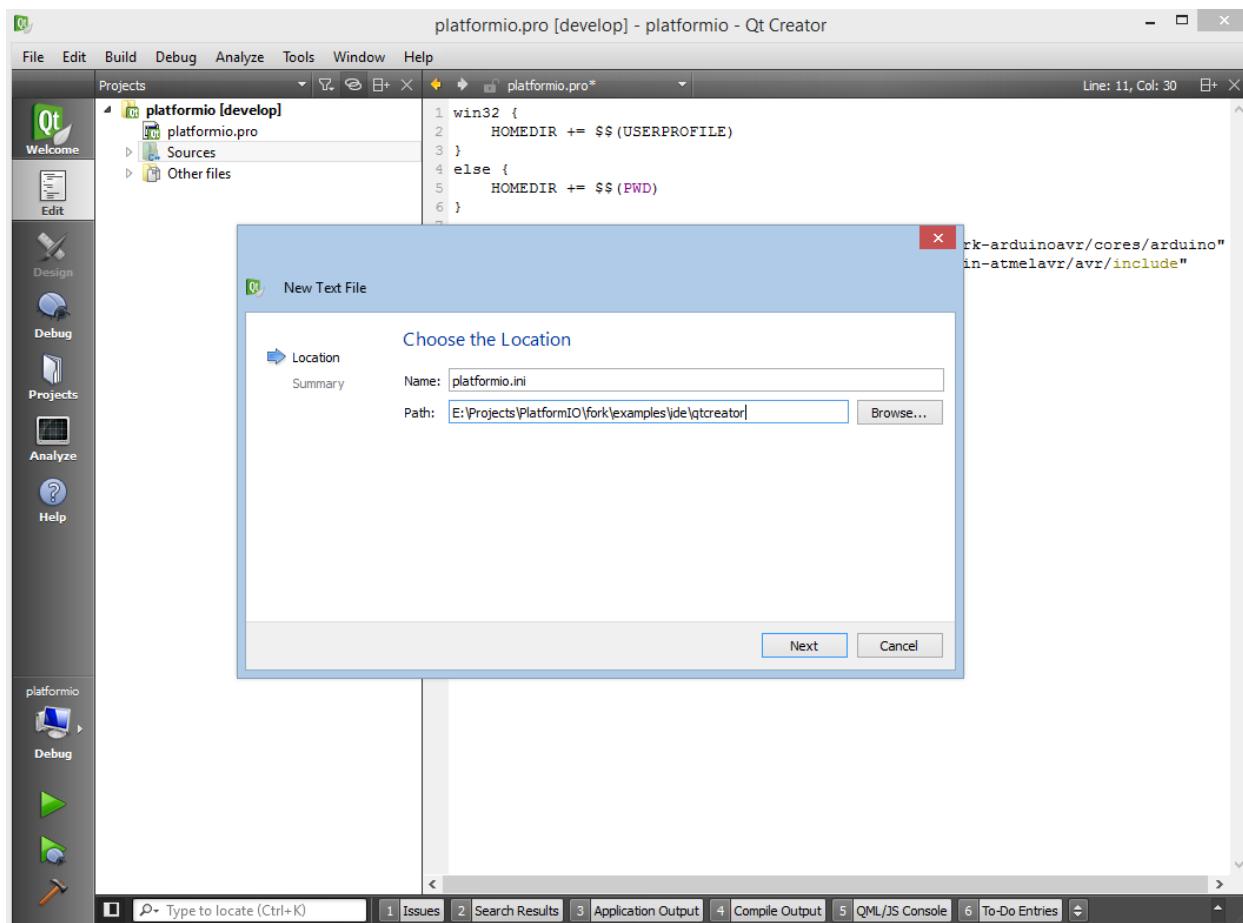
Copy the source code which is described below to file `main.c`.

```
#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.

```
; 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_ultimo]
platform = atmelavr
framework = arduino
board = uno
```

Conclusion

Taking everything into account, we can build project with shortcut **Ctrl+Shift+B** or using Menu: **Build > Build All**.

Sublime Text

The [Sublime Text](#) is a cross-platform text and source code editor, with a Python application programming interface (API). Sublime Text is proprietary software. Its functionality is extendable with plugins. Most of the extending packages have free-software licenses and are community-built and maintained. Sublime Text lacks graphical setting dialogues and is entirely configured by editing text files.

Refer to the [Sublime Text Documentation](#) page for more detailed information.

The screenshot shows a Sublime Text window with several open panes:

- File Browser (Left):** Shows project structure with files like main.cpp, USBCore.cpp, wiring_digital.c, and GDB-related files.
- Code Editor (Top Left):** Displays the wiring_digital.c source code, showing C code for digital pin control.
- GDB Disassembly (Top Right):** Shows assembly code corresponding to the C code in the editor.
- GDB Callstack (Bottom Left):** Shows the call stack of the current program.
- GDB Breakpoints (Bottom Middle):** Shows the current breakpoints set in the code.
- GDB Threads (Bottom Right):** Shows the list of threads running in the application.
- GDB Console (Bottom Far Right):** Shows the terminal output of the GDB session.
- Status Bar (Bottom):** Shows "Line 62, Column 12", "1 misspelled word", "Spaces: 2", and "C".

Contents

- [Deviot Plugin](#)
- [Integration](#)
 - [Project Generator](#)
 - [Manual Integration](#)
 - * [Initial configuration](#)
 - [Command Hotkeys](#)

- * *First program in Sublime Text*
- * *Conclusion*
- *Debugging*

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.)
4. Build project: Menu: Tools > Build.

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

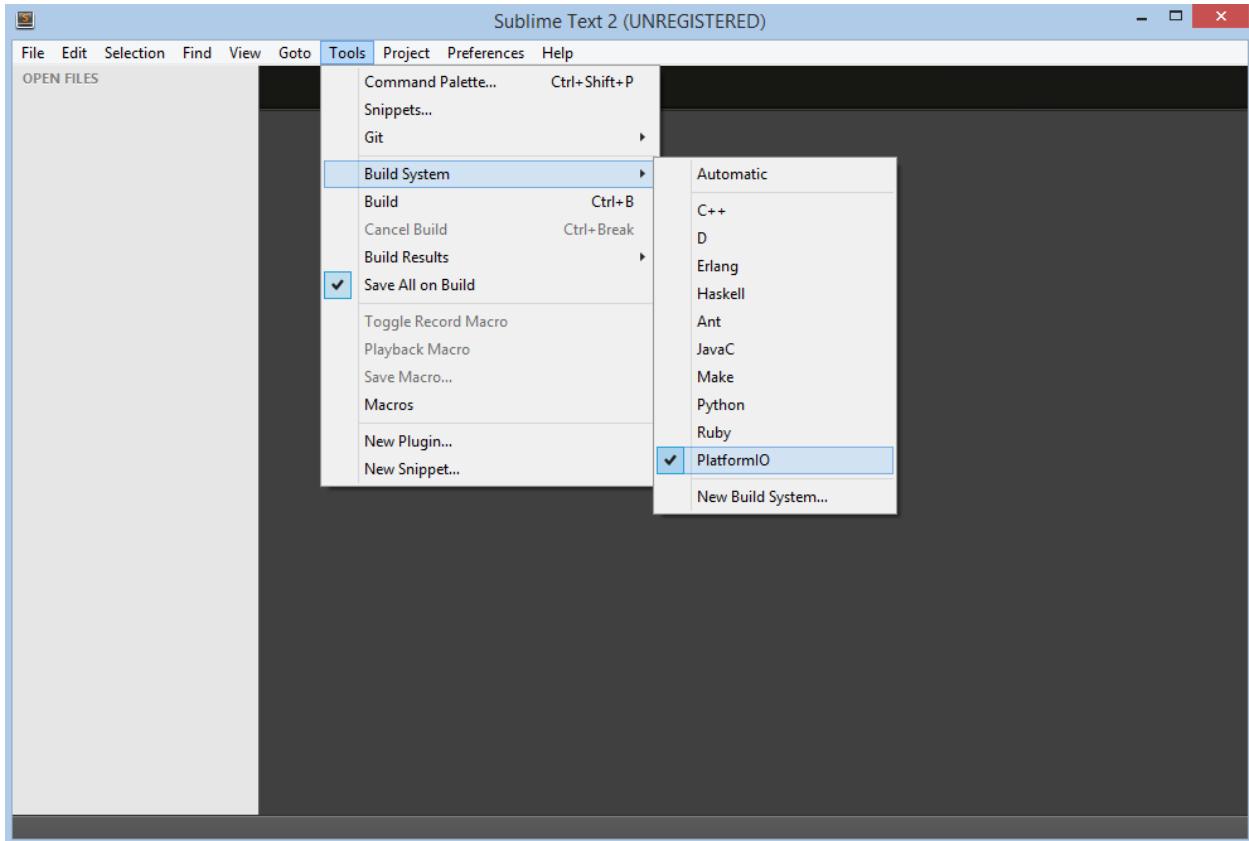
Note: Please verify that folder where is located platformio program is added to [PATH](#) (wiki) environment variable.

Initial configuration

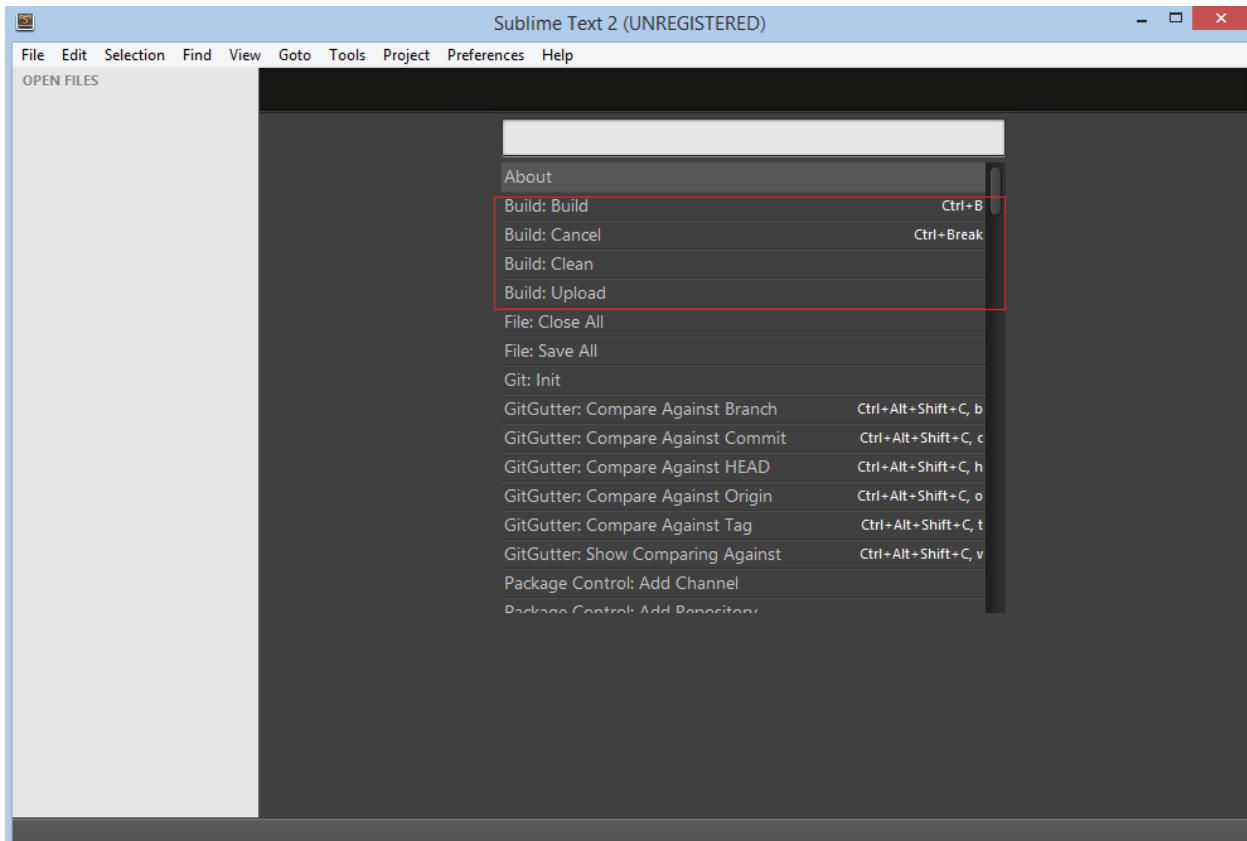
First of all, we need to create “New Build System” with name “PlatformIO” from Menu: Tools > Build System > New Build System and fill it like described below:

```
{
    "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:

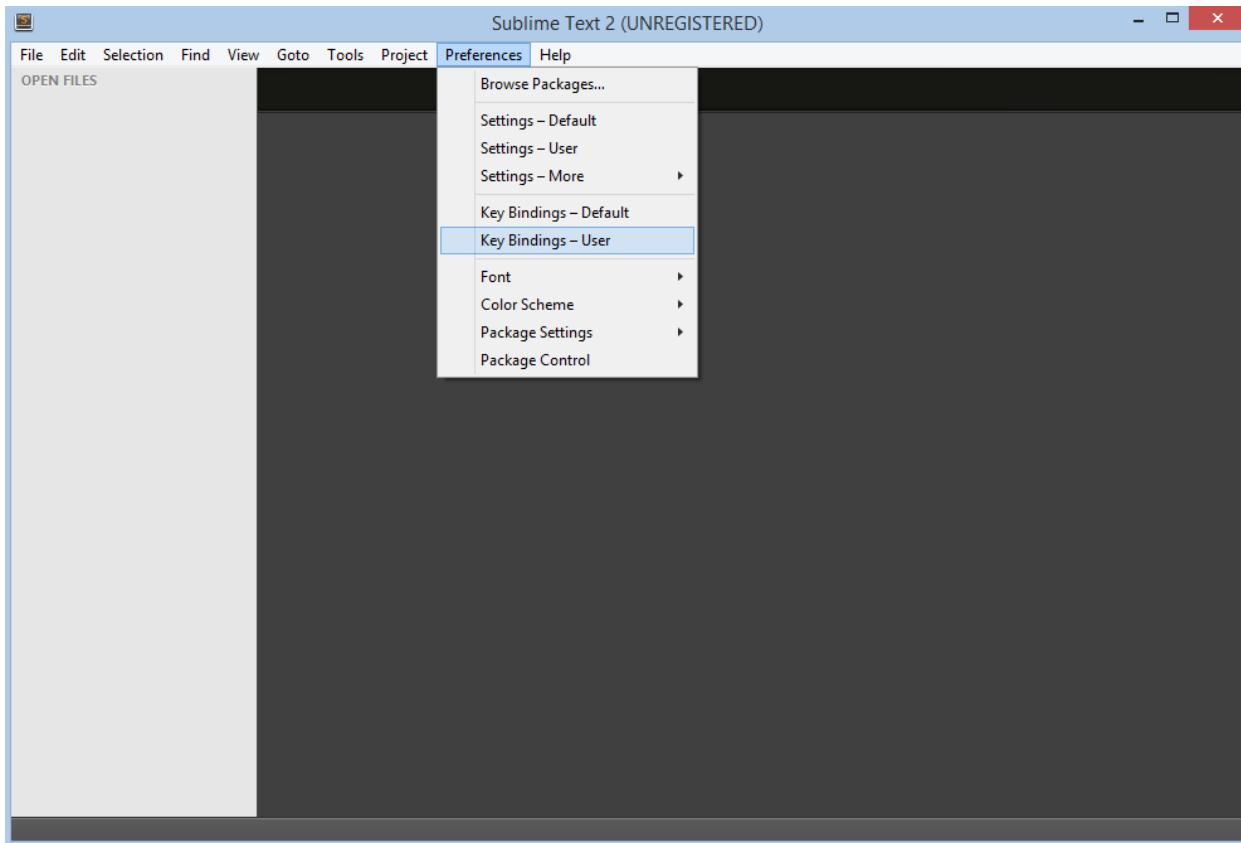


After that, we can use the necessary commands from Menu: Tools > Command Palette or with Ctrl+Shift+P (Windows/Linux) Cmd+Shift+P (Mac) shortcut.



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:

```
#include "Arduino.h"
#define WLED    13 // Most Arduino boards already have an LED attached to pin 13 on
                // the board itself

void setup()
{
    pinMode(WLED, OUTPUT); // set pin as output
}
```

(continues on next page)

(continued from previous page)

```
void loop()
{
    digitalWrite(WLED, HIGH); // set the LED on
    delay(1000); // wait for a second
    digitalWrite(WLED, LOW); // set the LED off
    delay(1000); // wait for a second
}
```

2. Project Configuration File named `platformio.ini` must be located in the project root directory. Copy the source code which is described below to it.

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

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

Conclusion

Taking everything into account, we can open project directory in Sublime Text using Menu: File > Open Folder and build it with shortcut Ctrl+B (Windows/Linux) or Cmd+B (Mac), clean project with shortcut F11 and upload firmware to target with shortcut F12.

Debugging

A debugging feature is provided by *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.

```

1 #include "Arduino.h"
2 #define WLED 13 // Most Arduino boards already have an LED attached to pin 13 on the board
3
4 void setup()
5 {
6   pinMode(WLED, OUTPUT); // set pin as output
7 }
8
9 void loop()
10 {
11   digitalWrite(WLED, HIGH); // set the LED on
12   delay(1000); // wait for a second
13   digitalWrite(WLED, LOW); // set the LED off
14   delay(1000); // wait for a second
15 }

~/Downloads/PlatformIO/main.c
1 [env:arduino_pro5v]
2 platform = atmelavr
3 framework = arduino
4 board = pro16MHzatmega168
5
6 # put here your device port..
7 # For Windows OS use COM1...COM9 ports
8 upload_port = /dev/ttyUSB0

~/Downloads/PlatformIO/platformio.ini
1 # Uncomment lines below if you have problems with $PATH
2 #SHELL := /bin/bash
3 #PATH := /usr/local/bin:$PATH
4
5 all:
6 > platformio run -t upload
7
8 clean:
9 > platformio run -t clean

NORMAL > ~/Downloads/PlatformIO/Makefile          make < utf-8[unix] < 66% 6: 26

```

Contents

- *Vim*
 - *Integration*
 - * “*Neomake-PlatformIO*” *Plugin*
 - * *Project Generator*
 - *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.

“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:

- C/C++/language server supporting cross references, hierarchies, completion and semantic highlighting - [CCLS: vim lsp](#)
- 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

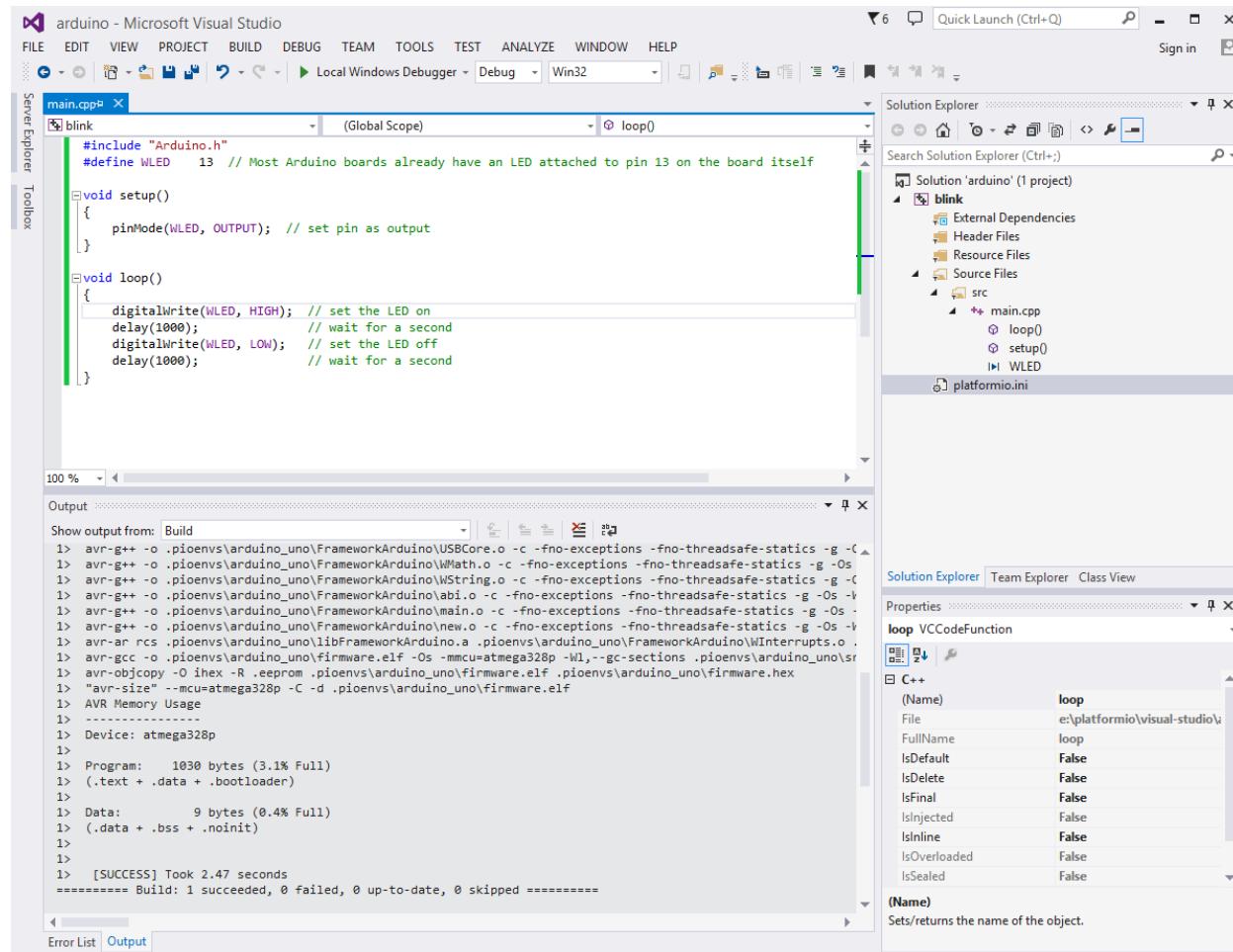
- Arduino : vim + platformio (Arduino development at the command line: VIM + PlatformIO, Japanese)

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*
 - * *Project Generator*
 - * *Manual Integration*
 - *Setup New Project*
 - *First program in Visual Studio*
 - *Conclusion*
 - *Known issues*
 - * *IntelliSense Errors*

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:

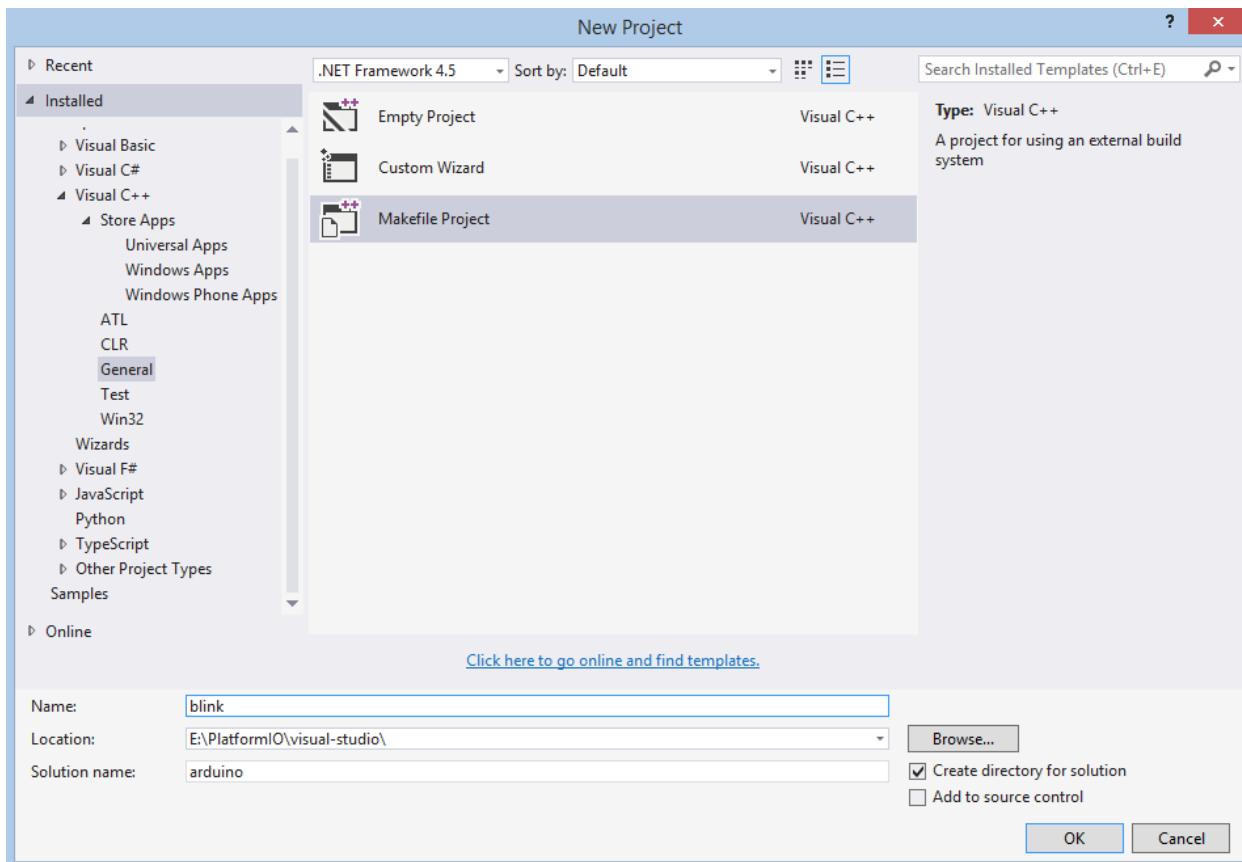
1. Import this project via Menu: File > Open > Project/Solution and specify root directory where is located “*platformio.ini*” (*Project Configuration File*)
2. Open source file from `src` directory (`*.c`, `*.cpp`, `*.ino`, etc.)
3. Build project: Menu: Build > Build Solution.

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).

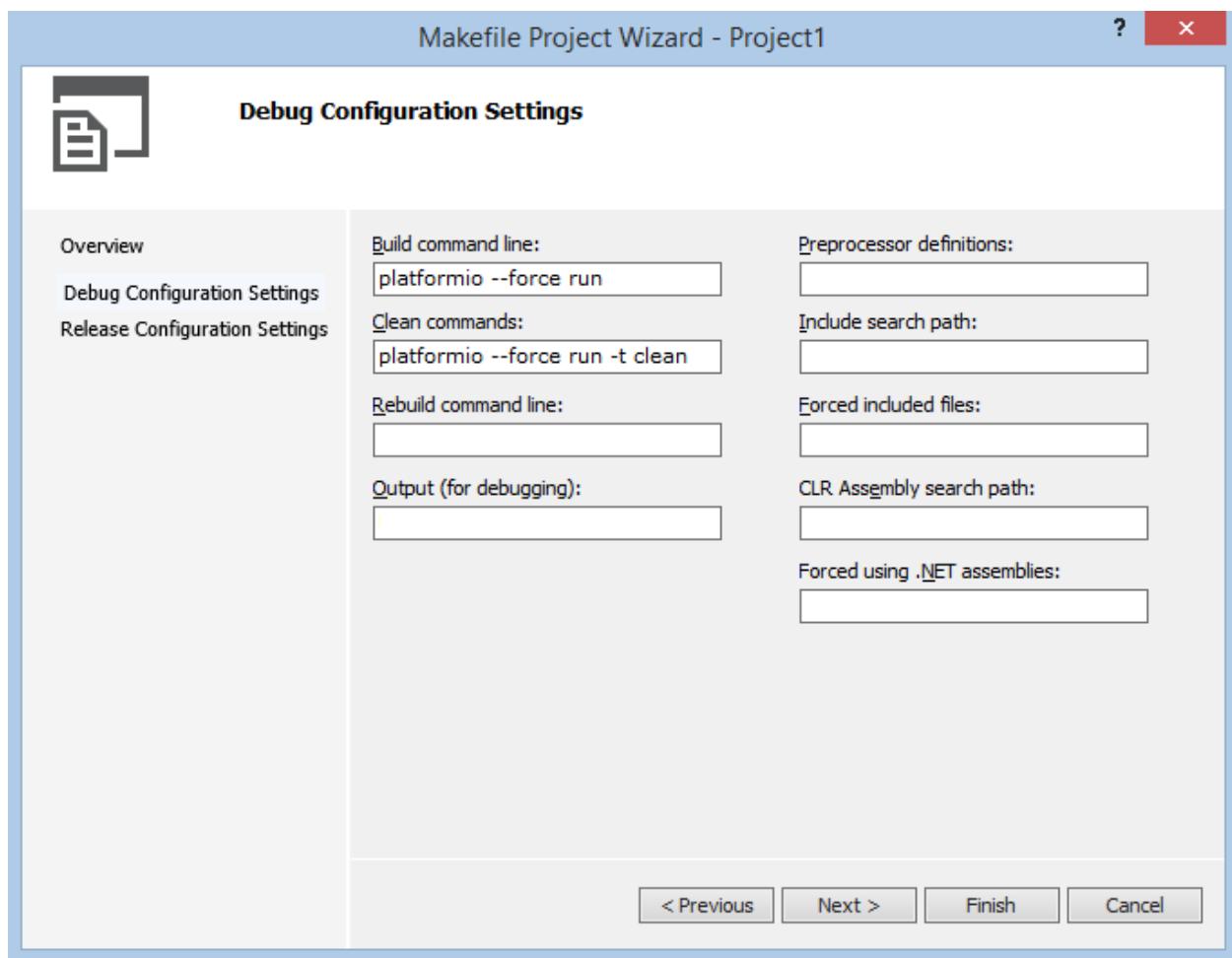
Manual Integration

Setup New Project

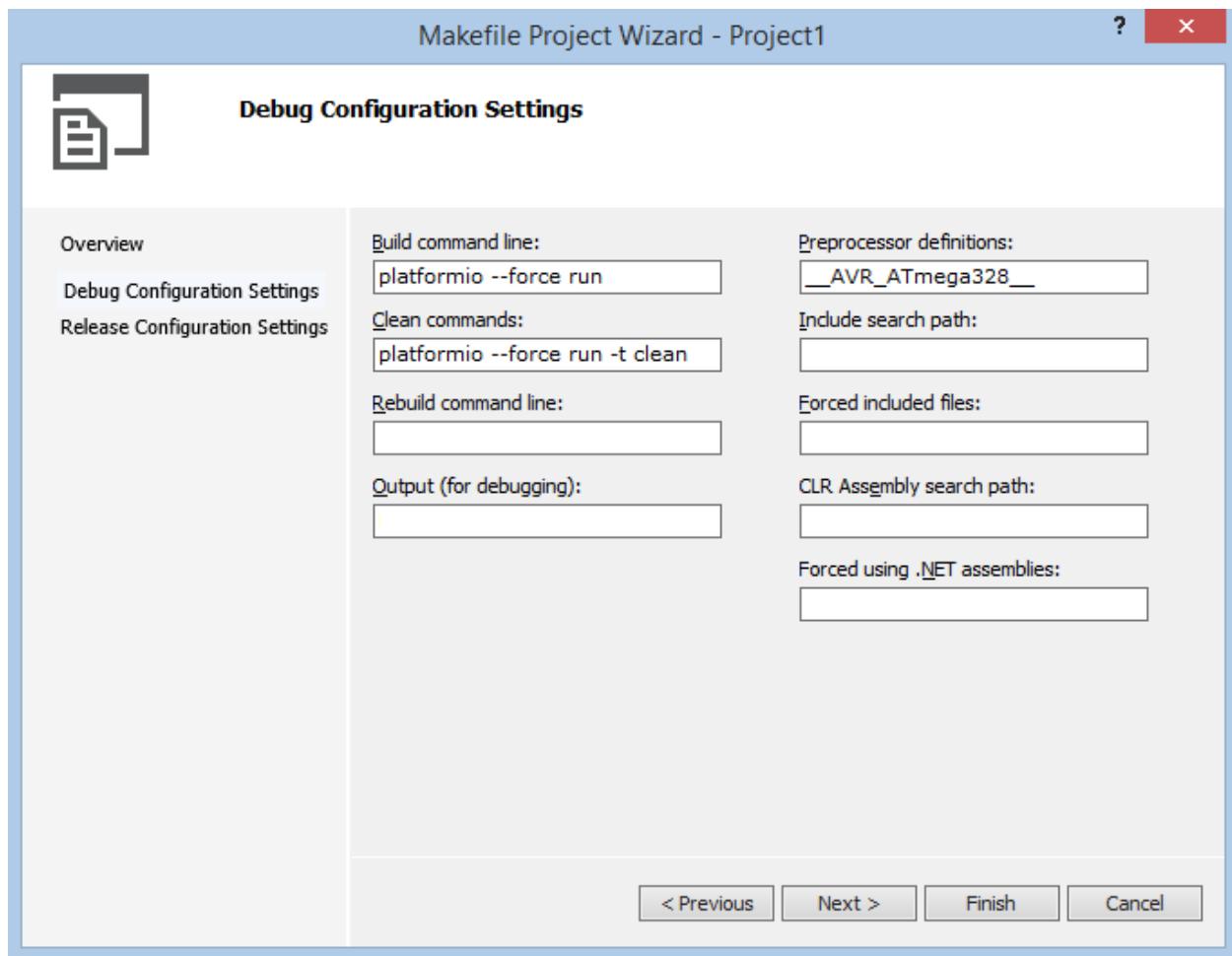
First of all, let's create new project from Visual Studio Start Page: Start > New Project or using Menu: File > New > Project, then select project with Makefile type (Visual C++ > General > Makefile Project), fill Project name, Solution name, Location fields and press OK button.



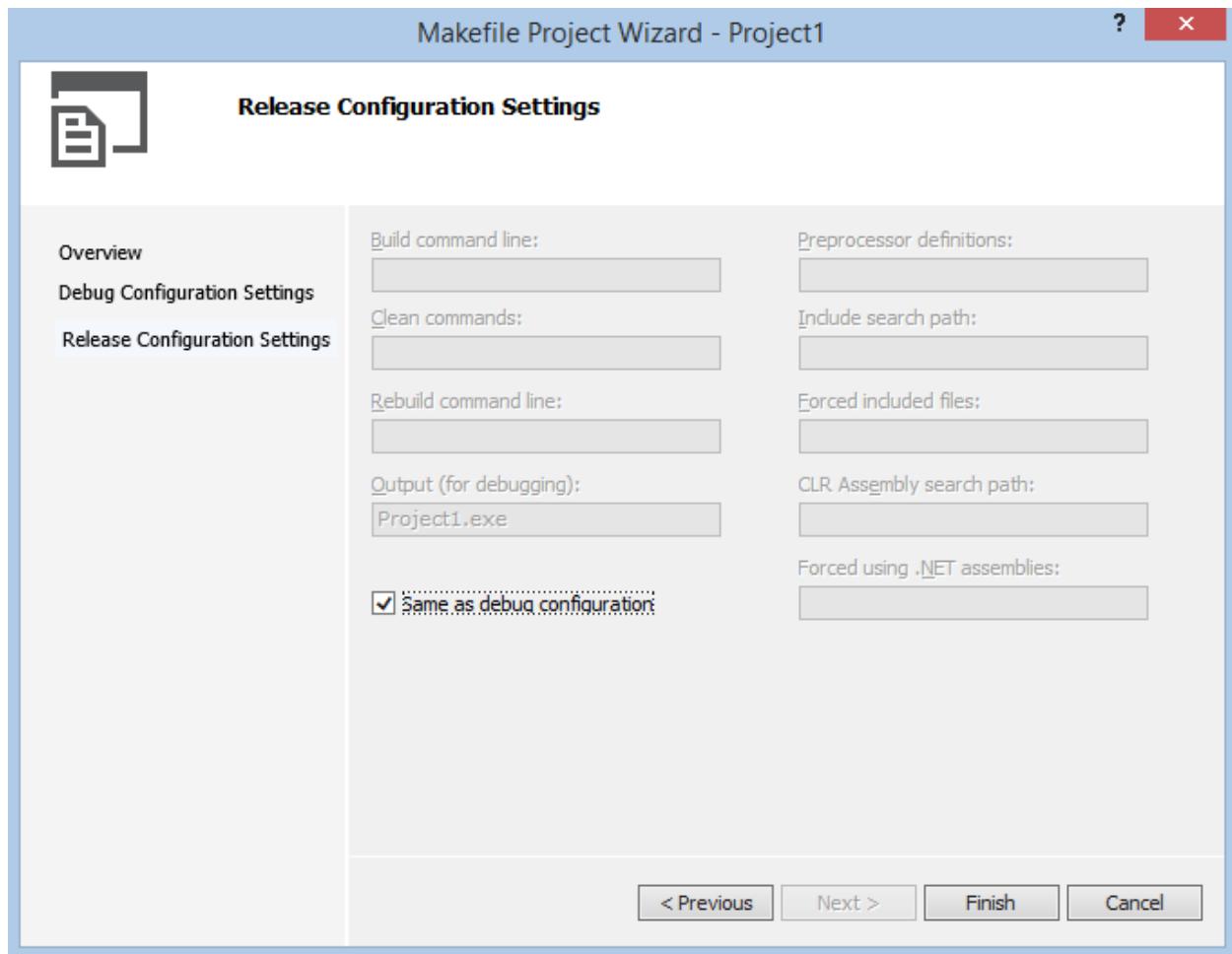
Secondly, we need to configure project with PlatformIO Build System:



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__`.

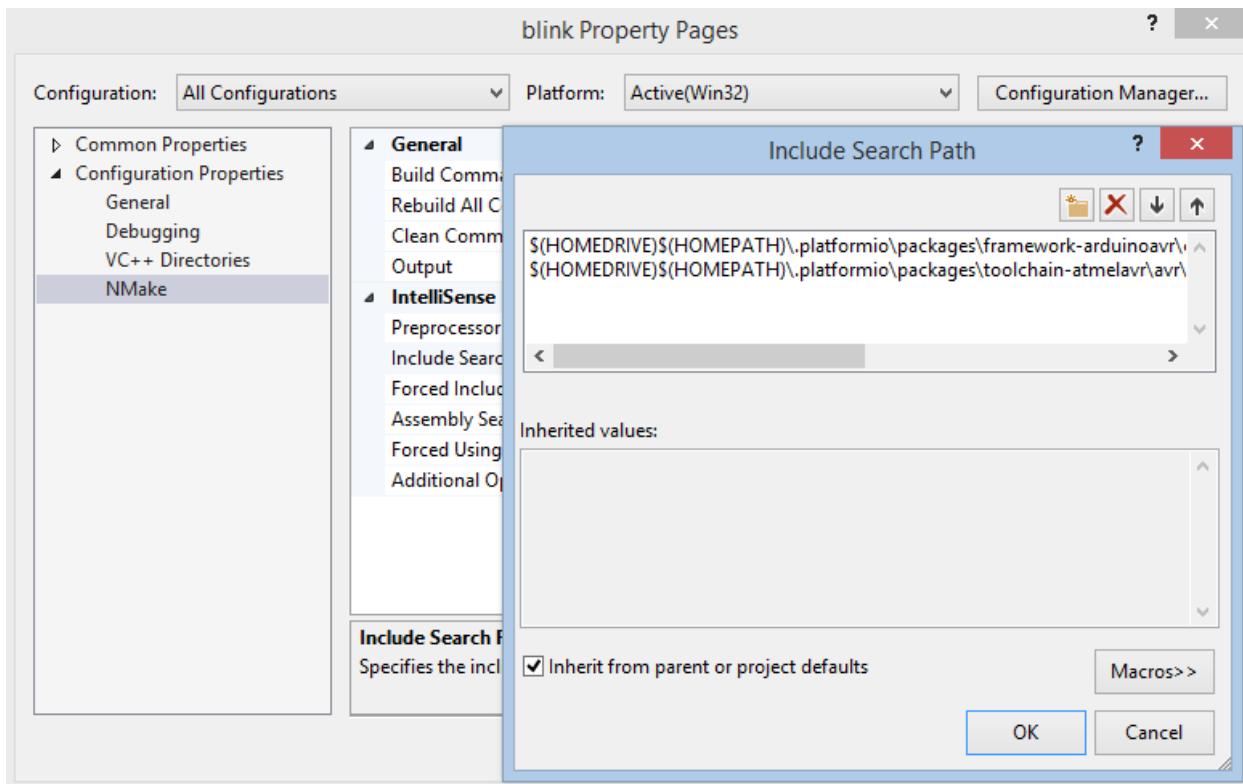


Release Configuration is the same as Debug, so on the next step we check “Same as Debug Configuration” and click “Finish” button.



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:

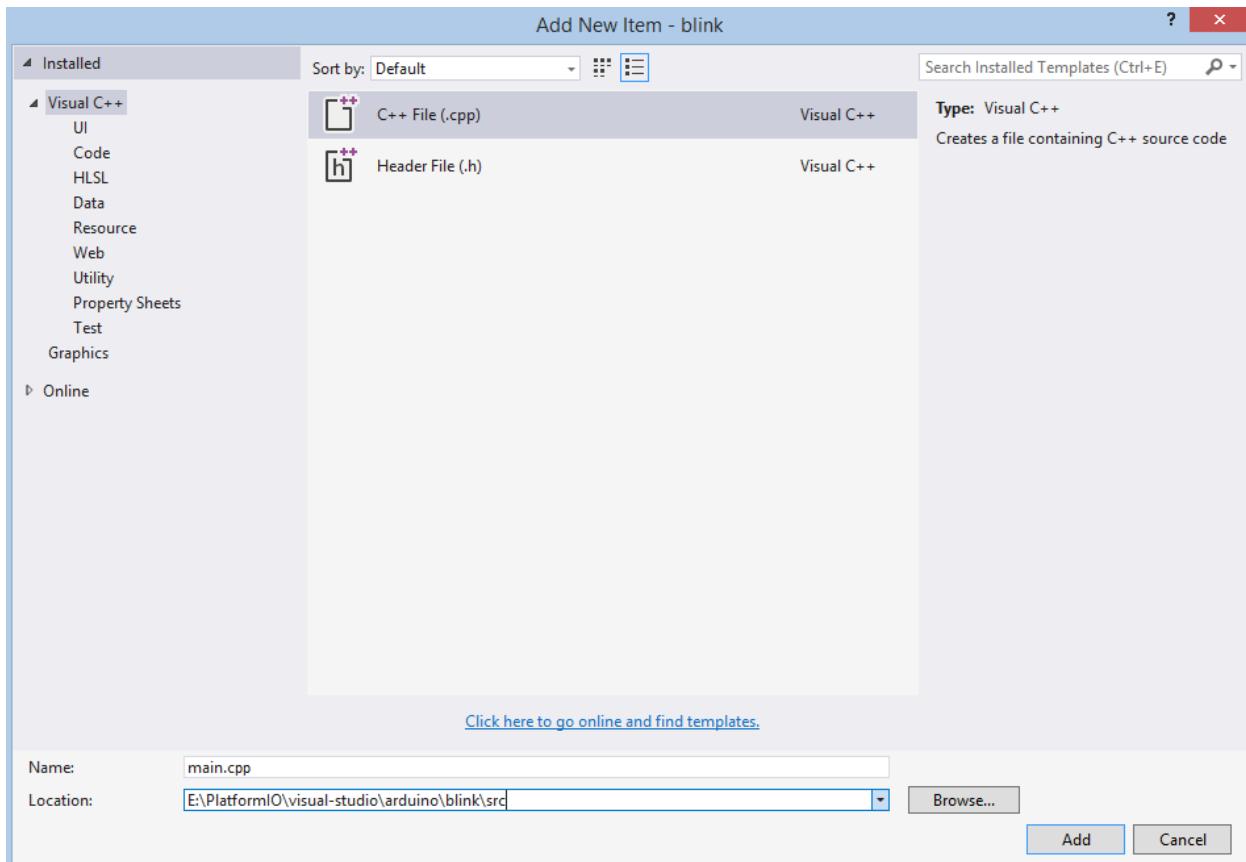
```
$ (HOMEDRIVE)$(HOMEPATH)\.platformio\packages\toolchain-atmelavr\avr\include  
$ (HOMEDRIVE)$(HOMEPATH)\.platformio\packages\framework-arduinoavr\cores\arduino
```



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`:



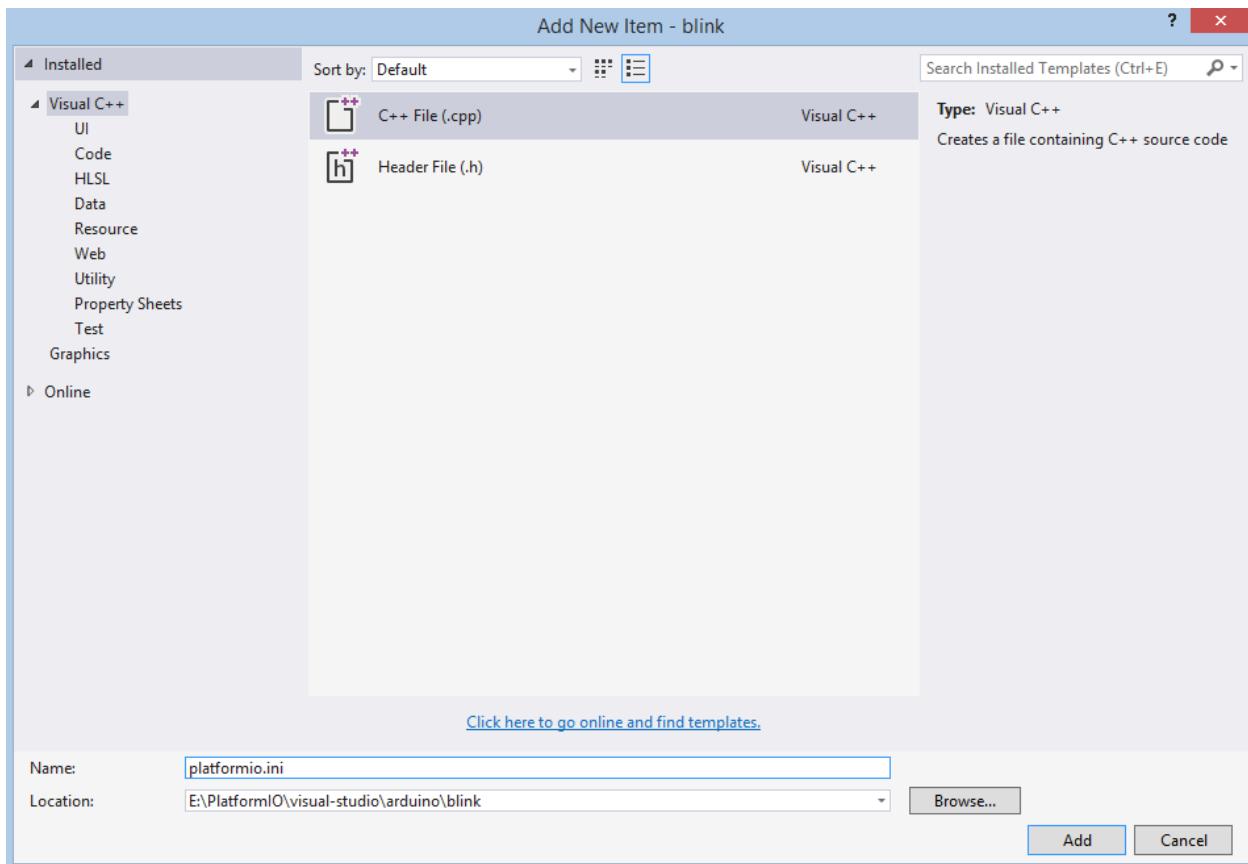
Copy the source code which is described below to file `main.cpp`.

```
#include "Arduino.h"

void setup()
{
    pinMode(LED_BUILTIN, OUTPUT); // set pin as output
}

void loop()
{
    digitalWrite(LED_BUILTIN, HIGH); // set the LED on
    delay(1000); // wait for a second
    digitalWrite(LED_BUILTIN, LOW); // set the LED off
    delay(1000); // wait for a second
}
```

2. Project Configuration File named `platformio.ini` must be located in the project root directory.



Copy the source code which is described below to it.

```
; 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**.

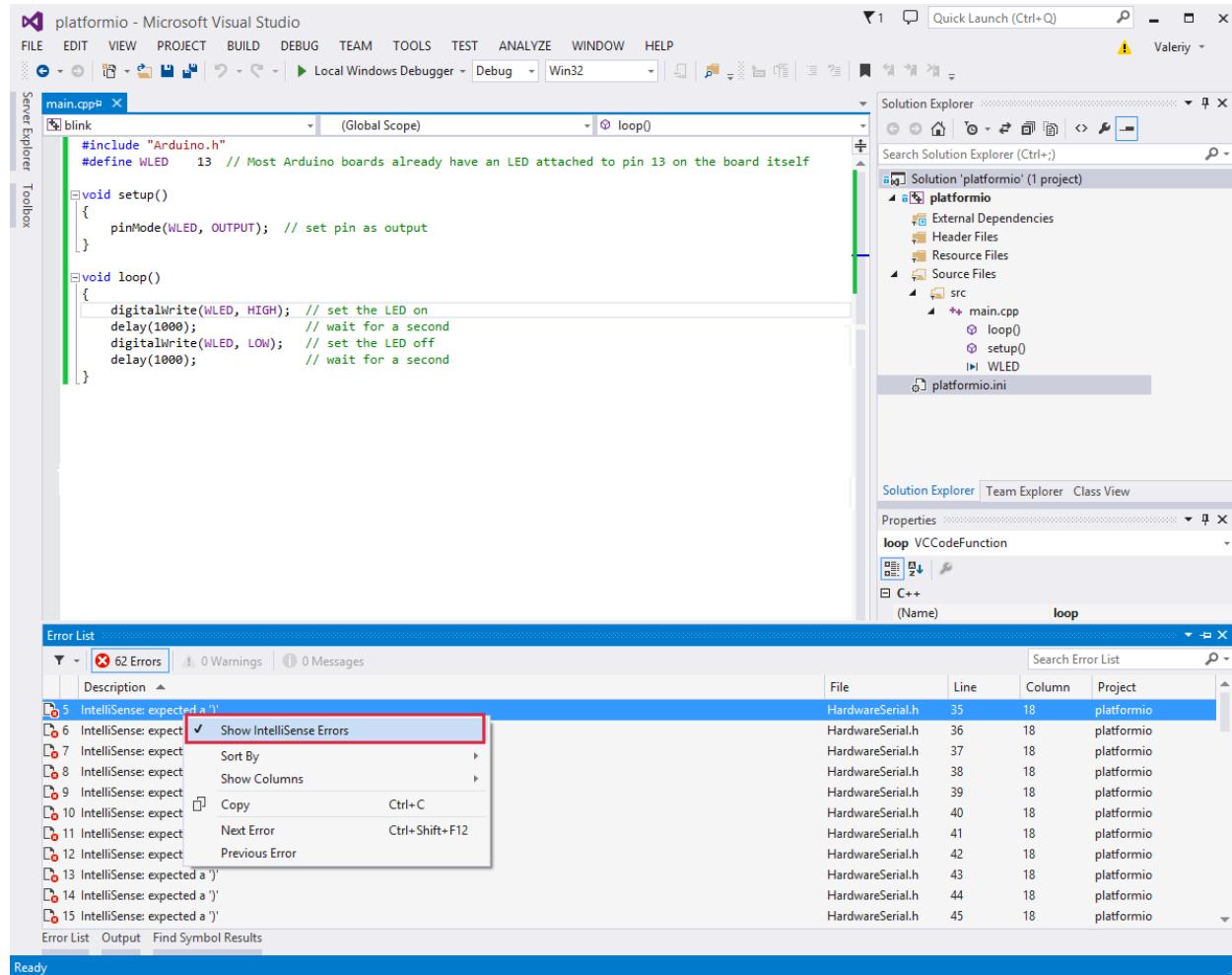
Known issues

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)

The screenshot shows the PlatformIO Debugger interface with the following components:

- Left Sidebar:** Includes sections for VARIABLES, WATCH, CALL STACK, BREAKPOINTS, PERIPHERALS, and REGISTERS.
- Middle Left:** Shows the `WiFi.cpp` file content with line numbers 579 to 621. It highlights a breakpoint at line 588 and shows memory dump for `mac` and `strM2MAPConfig.au8DHCPServerIP`.
- Middle Right:** Shows the assembly code for `WiFiClass::startProvision`, corresponding to the C code in the left pane.
- Bottom:** Includes tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, and TERMINAL. The TERMINAL tab displays the output of the task: "Executing task: platformio device monitor <". It shows the Miniterm setup and the command "Starting in provisioning mode...".

Contents

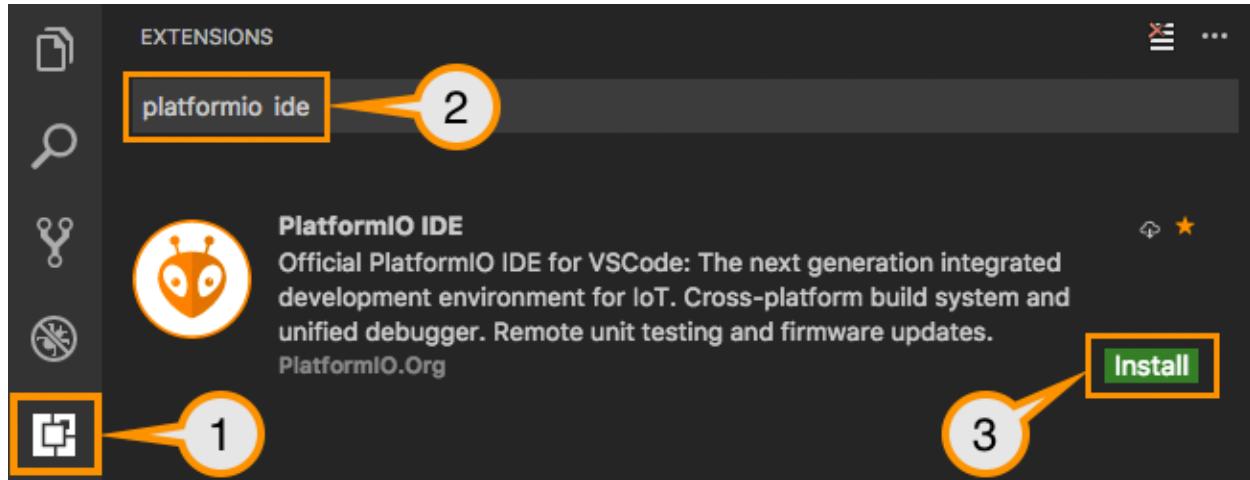
- *Installation*
- *Quick Start*
 - *Setting Up the Project*
- *PlatformIO Toolbar*
 - *Custom Build Task*
- *Key Bindings*
- *Project Tasks*
 - *Task Explorer*
 - *Task Runner*

- *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*
 - *PackageManager is unable to install tool*
- *Changelog*

Installation

Note: Please note that you do not need to install *PlatformIO Core (CLI)* separately if you are going to use *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.

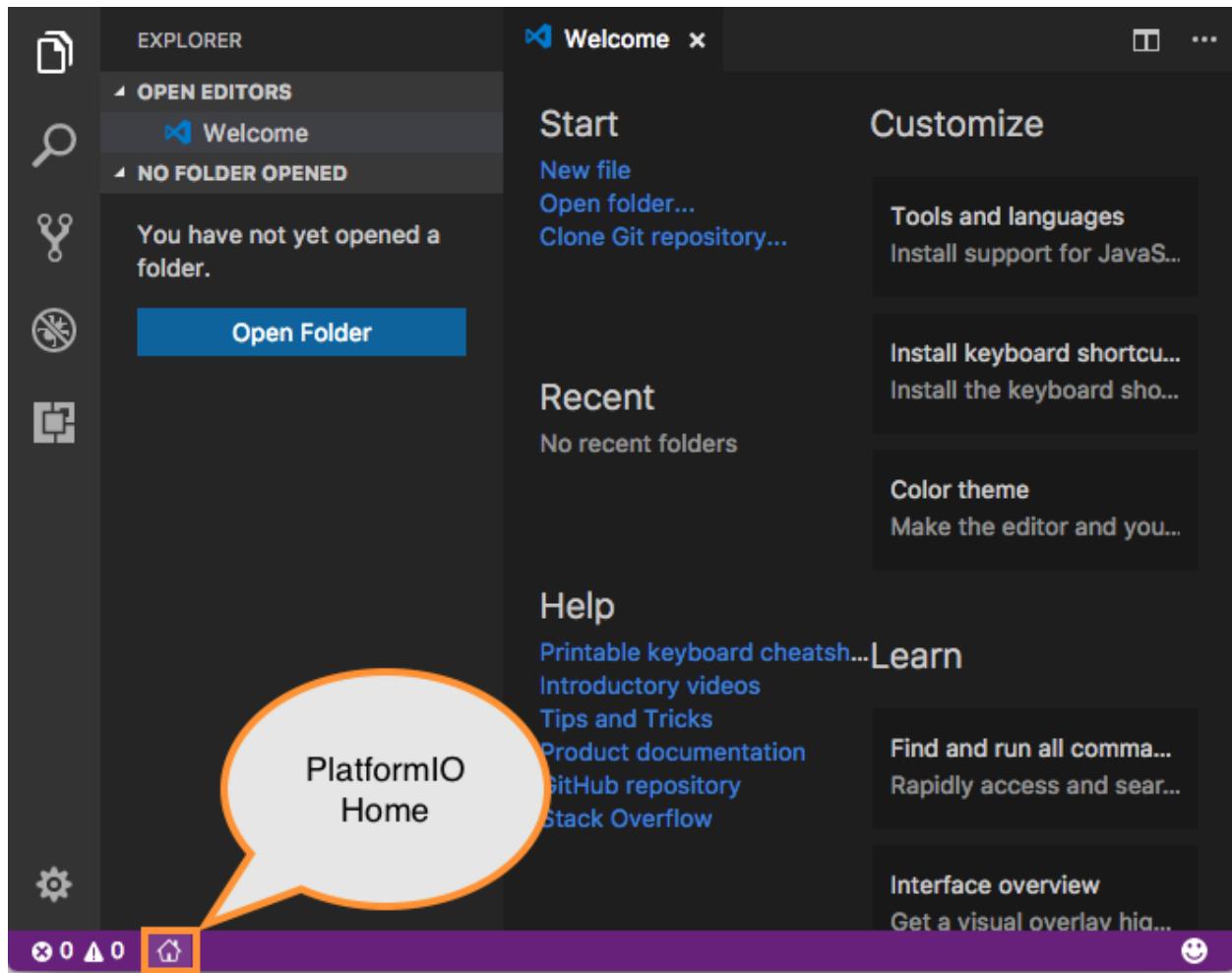


Quick Start

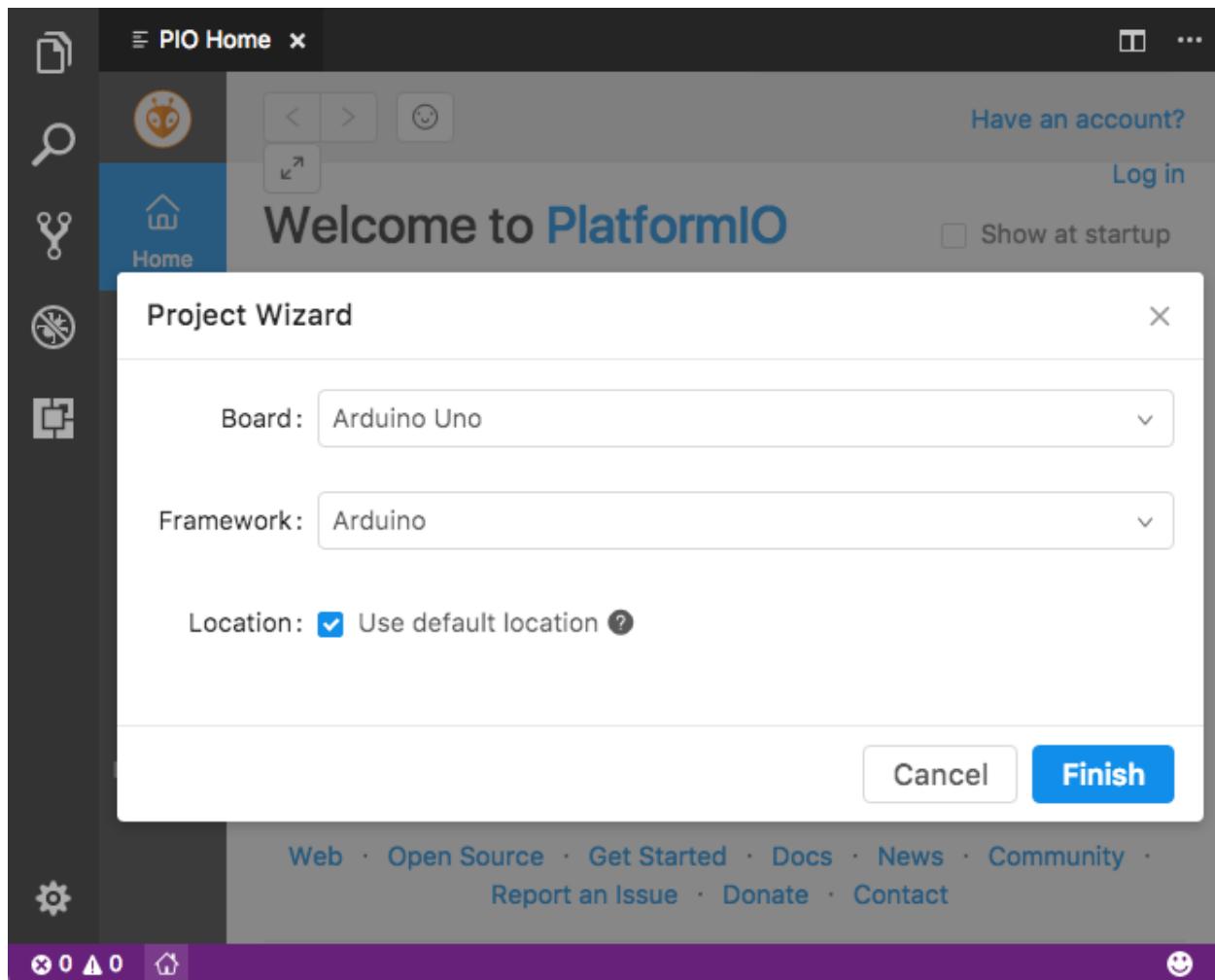
This tutorial introduces you to the basics of PlatformIO IDE workflow and shows you a creation process of a simple “Blink” example. After finishing you will have a general understanding of how to work with projects in the IDE.

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 from `src` folder and replace its contents with the next:

Warning: The code below works only in pair with Arduino-based boards. Please follow to [PlatformIO Project Examples](#) repository for other pre-configured projects.

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

// Set LED_BUILTIN if it is not defined by Arduino framework
// #define LED_BUILTIN 13

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

(continues on next page)

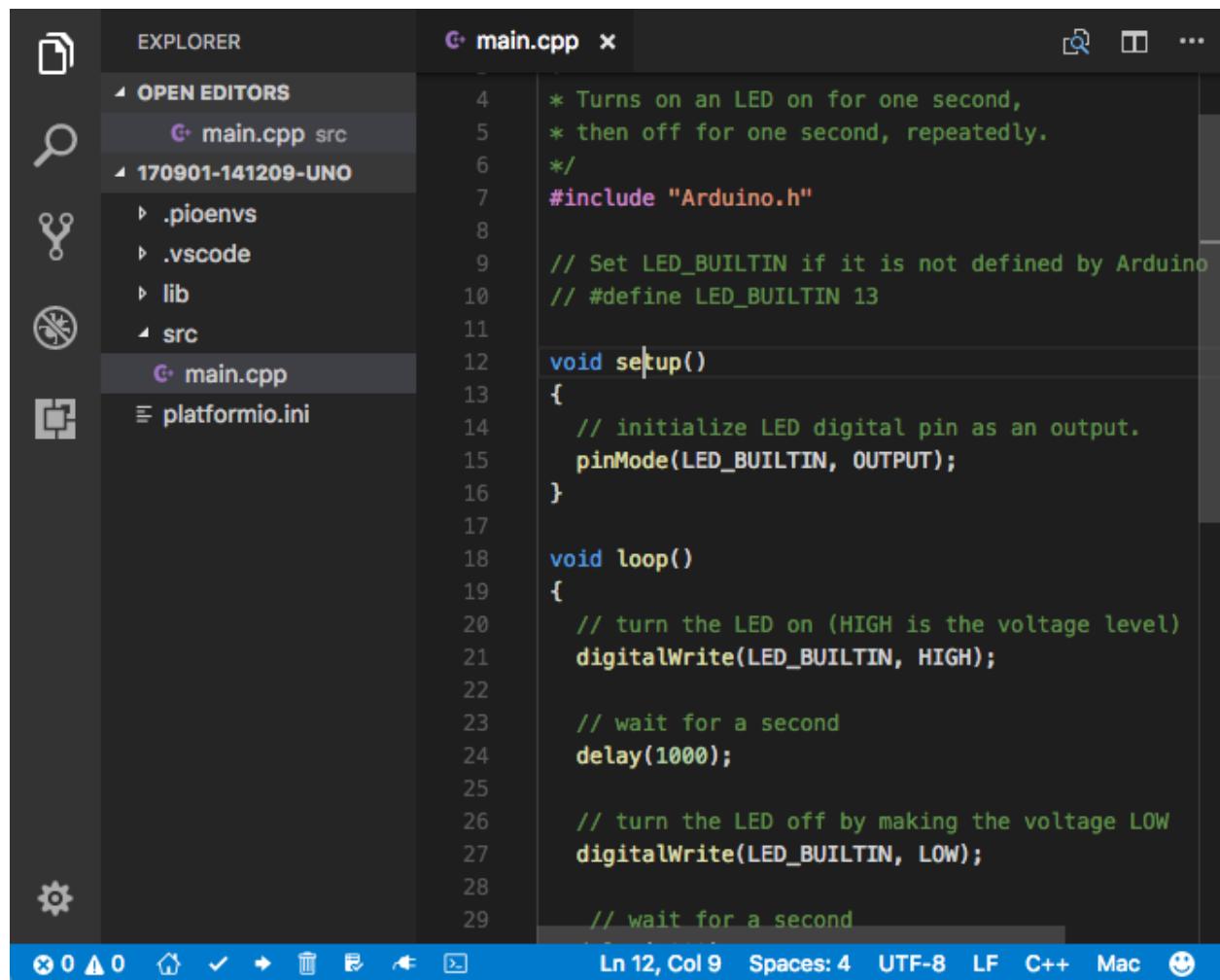
(continued from previous page)

```
void loop()
{
    // turn the LED on (HIGH is the voltage level)
    digitalWrite(LED_BUILTIN, HIGH);

    // wait for a second
    delay(1000);

    // turn the LED off by making the voltage LOW
    digitalWrite(LED_BUILTIN, LOW);

    // wait for a second
    delay(1000);
}
```



```
EXPLORER          main.cpp x
OPEN EDITORS
  main.cpp src
  170901-141209-UNO
    .pioenvs
    .vscode
    lib
    src
      main.cpp
platformio.ini

4  * Turns on an LED on for one second,
5  * then off for one second, repeatedly.
6  */
7  #include "Arduino.h"
8
9  // Set LED_BUILTIN if it is not defined by Arduino
10 // #define LED_BUILTIN 13
11
12 void setup()
13 {
14     // initialize LED digital pin as an output.
15     pinMode(LED_BUILTIN, OUTPUT);
16 }
17
18 void loop()
19 {
20     // turn the LED on (HIGH is the voltage level)
21     digitalWrite(LED_BUILTIN, HIGH);

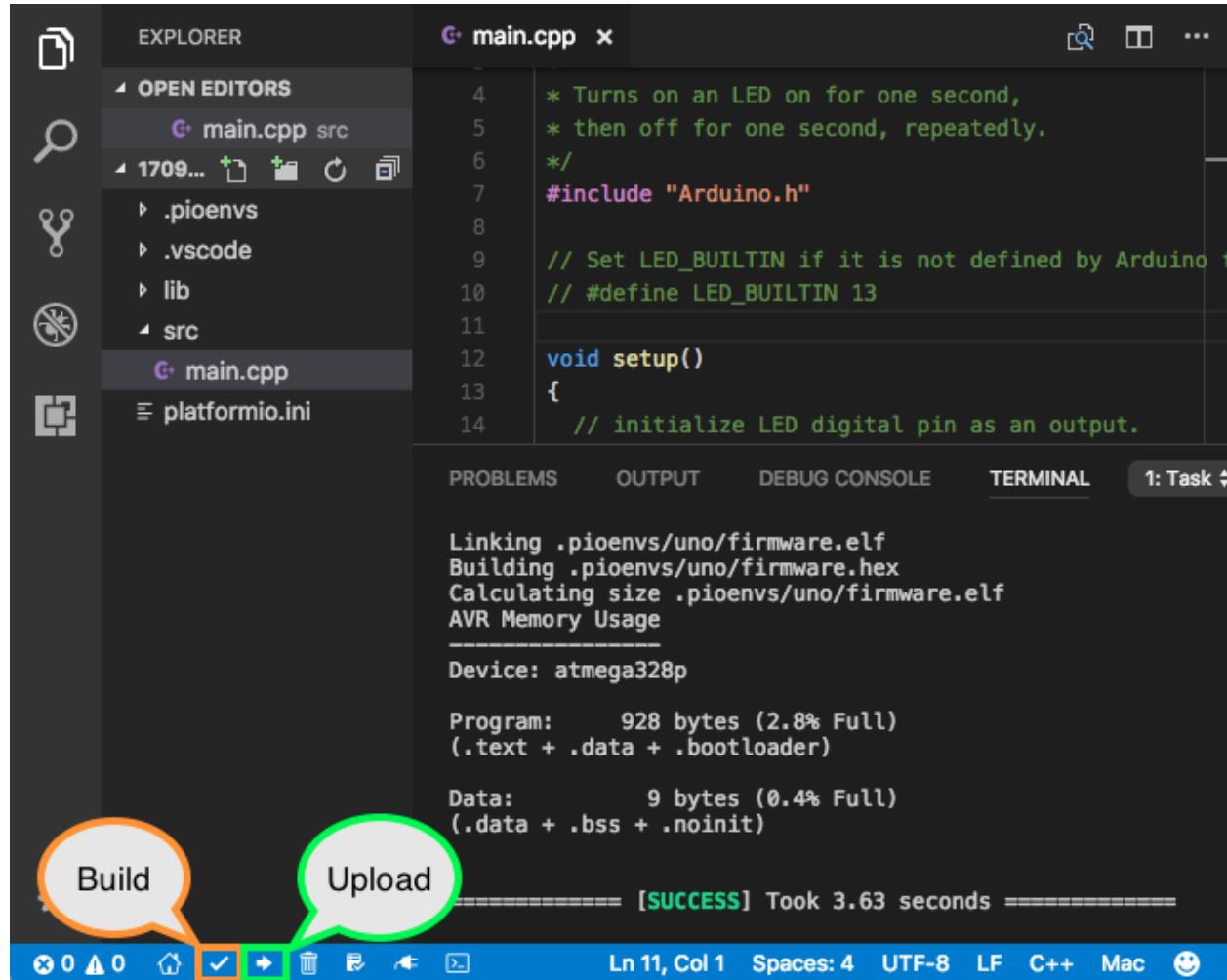
22     // wait for a second
23     delay(1000);

24     // turn the LED off by making the voltage LOW
25     digitalWrite(LED_BUILTIN, LOW);

26     // wait for a second
27     delay(1000);
28
29 }
```

Ln 12, Col 9 Spaces: 4 UTF-8 LF C++ Mac ☺

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

4. *PIO Remote*
5. PlatformIO: Clean
6. *PIO Unit Testing*
7. Run a task... (See “Task Runner” below)
8. *Serial Port Monitor*
9. PIO Terminal

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 *Settings* for more details.

Built-in PlatformIO tasks are available in “Menu > Terminal > Run Task...” list.

Key Bindings

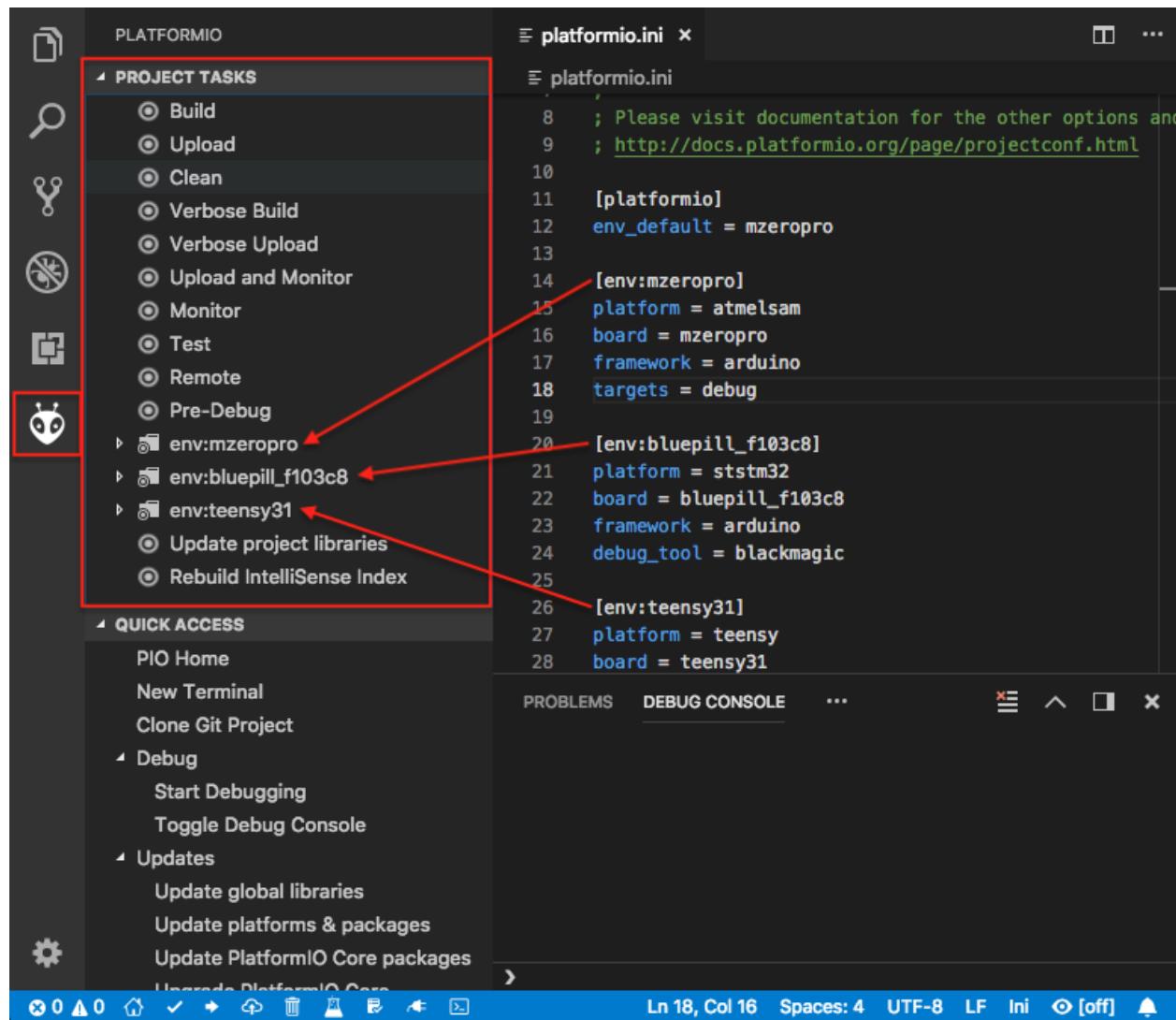
- `ctrl+alt+b / cmd-shift-b / ctrl-shift-b` Build Project
- `cmd-shift-d / ctrl-shift-d` Debug project
- `ctrl+alt+u` Upload Firmware
- `ctrl+alt+s` Open *Serial Port Monitor*

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”.



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

```
{
  "version": "2.0.0",
  "tasks": [
    {
      "name": "build",
      "action": "PIO Build"
    }
  ]
}
```

(continues on next page)

(continued from previous page)

```
{
    {
        "type": "PlatformIO",
        "task": "Monitor",
        "problemMatcher": [
            "$platformio"
        ],
        "presentation": {
            "panel": "new"
        }
    }
}
```

See more options in [official VSCode documentation](#).

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](#)).

```
{
    "version": "2.0.0",
    "tasks": [
        {
            "type": "shell",
            "command": "platformio",
            "args": [
                "device",
                "monitor",
                "--echo"
            ],
            "problemMatcher": [
                "$platformio"
            ],
            "label": "PlatformIO: Monitor (local echo)"
        }
    ]
}
```

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/pvenv/bin`”:

```
{
    "version": "2.0.0",
    "tasks": [
        {
            "type": "shell",
            "command": "platformio",
            "args": [
                "device",
                "monitor",
                "--echo"
            ],
            "options": {
                "cwd": "~/.platformio/pvenv/bin"
            }
        }
    ]
}
```

(continues on next page)

(continued from previous page)

```
        ],
        "problemMatcher": [
            "$platformio"
        ],
        "label": "PlatformIO: Monitor (local echo)",
        "options": {
            "env": { "PATH": "${env:HOME}/.platformio/pvenv/bin" }
        }
    }
}
```

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.

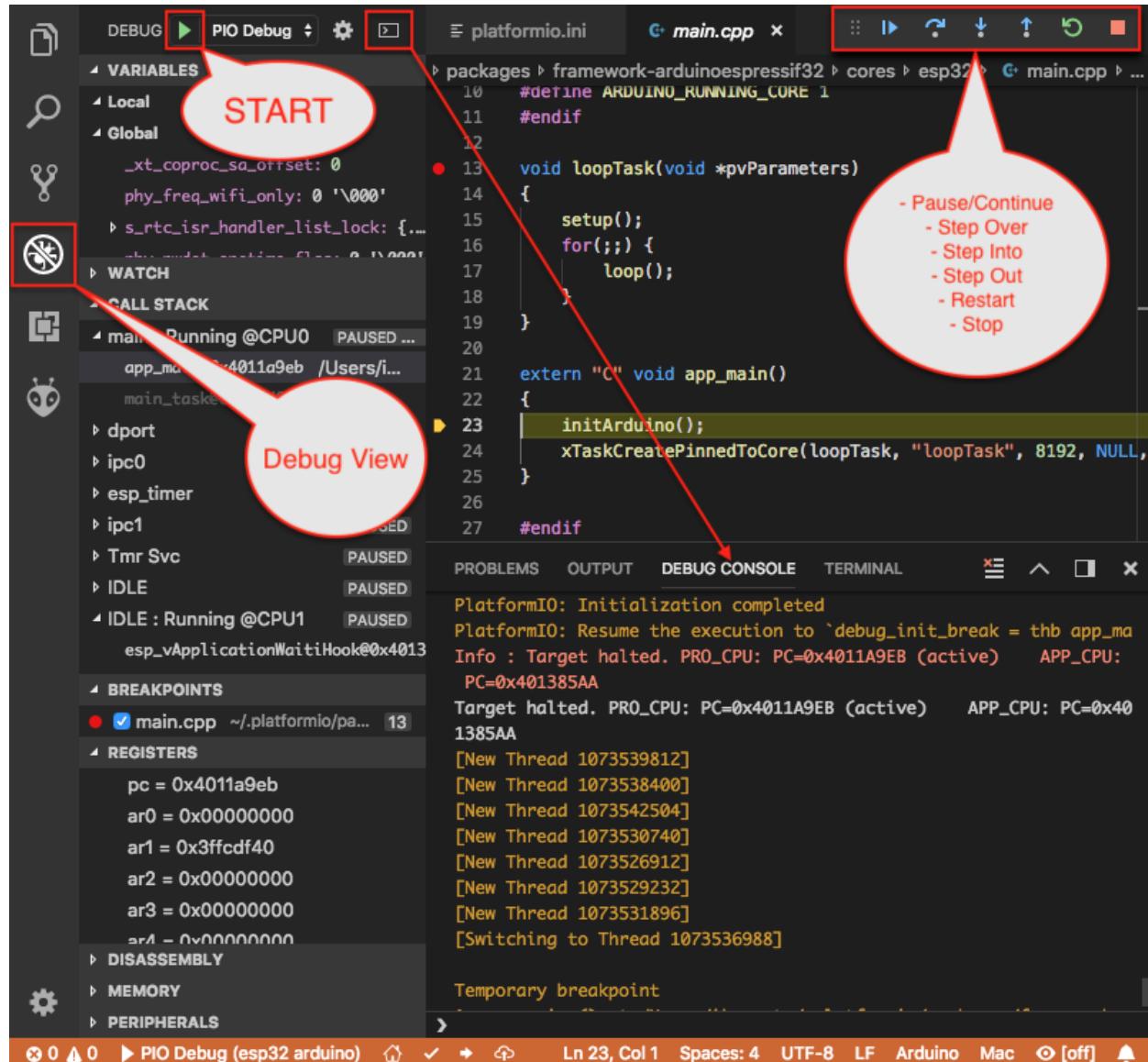
There are 2 pre-configured debugging configurations:

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](#).

1.20 Continuous Integration

Continuous Integration (CI, [wiki](#)) is the practice, in software engineering, of merging all developer working copies with a shared mainline several times a day.

`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.

1.20.1 AppVeyor

AppVeyor is an open-source hosted, distributed continuous integration service used to build and test projects hosted at [GitHub](#) on Windows family systems.

AppVeyor is configured by adding a file named `appveyor.yml`, which is a [YAML](#) format text file, to the root directory of the GitHub repository.

AppVeyor automatically detects when a commit has been made and pushed to a repository that is using AppVeyor, and each time this happens, it will try to build the project using `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.

```
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:

```
build: off
environment:

matrix:
  - PLATFORMIO_CI_SRC: "examples\\Bluetooth\\PS3SPP\\PS3SPP.ino"
  - PLATFORMIO_CI_SRC: "examples\\p12303\\p12303_gps\\p12303_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/xxxxjk/spi4teensy3.git C:\spi4teensy

test_script:
  - cmd: platformio ci --lib=". " --lib="C:\\spi4teensy" --board=uno --
  ↵board=teensy31 --board=due
```

1.20.2 CircleCI

CircleCI is a hosted cloud platform that provides hosted continuous integration, deployment, and testing to GitHub repositories.

CircleCI is configured by adding a file named `circle.yml`, which is a [YAML](#) format text file, to the root directory of the GitHub repository.

CircleCI automatically detects when a commit has been made and pushed to a repository that is using CircleCI, and each time this happens, it will try to build the project using `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.

```
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

```
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

```
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

```
dependencies:
  pre:
```

(continues on next page)

(continued from previous 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/extrra/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/platformio.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

```

(continues on next page)

(continued from previous page)

```

# 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 --
  ↵board=due --lib="."
      - platformio ci examples/adk/adk_barcode --board=uno --board=teensy31 --
  ↵board=due --lib="."
          - platformio ci examples/adk/ArduinoBlinkLED --board=uno --board=teensy31 --
  ↵board=due --lib="."
              - platformio ci examples/adk/demokit_20 --board=uno --board=teensy31 --
  ↵board=due --lib="."
                  #
                  ...
              - platformio ci examples/Xbox/XBOXUSB --board=uno --board=teensy31 --
  ↵board=due --lib="."

```

2. Dependency on external libraries

```

dependencies:
  pre:
    # Install the latest stable PlatformIO
    - sudo pip install -U platformio

    # download dependent libraries
    - wget https://github.com/jcw/jeelib/archive/master.zip -O /tmp/jeelib.zip
    - unzip /tmp/jeelib.zip -d /tmp

    - wget https://github.com/Rodot/Gamebuino/archive/master.zip -O /tmp/
  ↵gamebuino.zip
    - 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
                #
                ...
            - platformio ci examples/twitter/twitter.ino --lib=". --lib="/tmp/jeelib-
  ↵master" --lib="/tmp/Gamebuino-master/libraries/tinyFAT" --board=uno --
  ↵board=megaatmega2560
            - platformio ci examples/udpClientSendOnly/udpClientSendOnly.ino --lib=". --
  ↵lib="/tmp/jeelib-master" --lib="/tmp/Gamebuino-master/libraries/tinyFAT" --
  ↵board=uno --board=megaatmega2560
            - platformio ci examples/udpListener/udpListener.ino --lib=". --lib="/tmp/
  ↵jeelib-master" --lib="/tmp/Gamebuino-master/libraries/tinyFAT" --board=uno --
  ↵board=megaatmega2560
            - platformio ci examples/webClient/webClient.ino --lib=". --lib="/tmp/
  ↵jeelib-master" --lib="/tmp/Gamebuino-master/libraries/tinyFAT" --board=uno --
  ↵board=megaatmega2560

```

(continues on next page)

(continued from previous page)

-
- Configuration file: <https://github.com/ivankravets/ethercard/blob/master/circle.yaml>
 - Build History: <https://circleci.com/gh/ivankravets/ethercard/tree/master>

1.20.3 Drone

Drone is a hosted continuous integration service. It enables you to conveniently set up projects to automatically build, test, and deploy as you make changes to your code to [GitHub](#) and [BitBucket](#) repositories.

Drone is configured by modifying settings in your project control panel.

Drone automatically detects when a commit has been made and pushed to a repository that is using Drone, and each time this happens, it will try to build the project using `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:

```
PLATFROMIO_CI_SRC=path/to/source/file.c
PLATFROMIO_CI_SRC=path/to/source/file.ino
PLATFROMIO_CI_SRC=path/to/source/directory
```

Commands:

```
pip install -U platformio
platformio ci --board=<ID_1> --board=<ID_2> --board=<ID_N>
```

Build Now

Kickoff build request. Use this testing changes to your build script.
Make sure to save changes first (Save is at the bottom).

Language

Python 2.7

Databases

- MySQL
- PostgreSQL
- [more ...](#)

Environment Variables

Enter one per line (ex: FOO=bar)

Commands

Enter one command per line (ex: echo foo)

SaveFor 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/p12303/p12303_gps/p12303_gps.ino
```

Commands:

```
pip install -U platformio
wget https://github.com/xxxajk/spi4teensy3/archive/master.zip -O /tmp/spi4teensy3.zip
unzip /tmp/spi4teensy3.zip -d /tmp
platformio ci --lib="." --lib="/tmp/spi4teensy3-master" --board=uno --board=teensy31 --
→--board=due
```

Build Now

Kickoff build request. Use this testing changes to your build script.
Make sure to save changes first (Save is at the bottom).

Language

Python 2.7 ▾

Databases

- MySQL
- PostgreSQL
- [more ...](#)

Environment Variables

```
PLATFORMIO_CI_SRC=examples/Bluetooth/PS3SPP/PS3SPP.ino
PLATFORMIO_CI_SRC=examples/pl2303/pl2303_gps/pl2303_gps.ino
```

Enter one per line (ex: FOO=bar)

Commands

working directory `/home/ubuntu/src/github.com/valeros/USB_Host_Shield_2.0`

```
pip install --egg
http://sourceforge.net/projects/scons/files/latest/download
pip install
https://github.com/platformio/platformio/archive/develop.zip
wget https://github.com/xxxajk/spi4teensy3/archive/master.zip -O
/tmp/spi4teensy3.zip
unzip /tmp/spi4teensy3.zip -d /tmp
```

Enter one command per line (ex: echo foo)

Save

1.20.4 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.

```
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:

```
image: python:2.7

stages:
- test

.job_template: &pio_run
  script:
    - "platformio ci --lib='.' --board=uno --board=teensy31 --board=nodemcuv2_"
    ↴$PLATFORMIO_CI_EXTRA_ARGS"

before_script:
- "pip install -U platformio"

JsonGeneratorExample:
  <<: *pio_run
```

(continues on next page)

(continued from previous 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

```

1.20.5 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*.

Contents

- *Jenkins*
 - *Integration*

Integration

See step-by-step guide in ThingForward's blog post [Setting up a Jenkins CI engine for embedded projects](#).

1.20.6 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](#).

Contents

- *Shippable*
 - *Integration*
 - *Examples*

Integration

Please put `shippable.yml` or `.travis.yml` to the root directory of the GitHub repository.

```
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:

```
language: python
python:
  - "2.7"

env:
  - PLATFORMIO_CI_SRC=examples/Bluetooth/PS3SPP/PS3SPP.ino
  - PLATFORMIO_CI_SRC=examples/pl2303/pl2303_gps/pl2303_gps.ino

install:
  - pip install -U platformio
  - wget https://github.com/xxxajk/spi4teensy3/archive/master.zip -O /tmp/
  ↵spi4teensy3.zip
  - unzip /tmp/spi4teensy3.zip -d /tmp

script:
  - platformio ci --lib="." --lib="/tmp/spi4teensy3-master" --board=uno --
  ↵board=teensy31 --board=due
```

1.20.7 Travis CI



Travis CI officially supports [PlatformIO for Embedded Builds](#).

Travis CI is an open-source hosted, distributed continuous integration service used to build and test projects hosted at [GitHub](#).

Travis CI is configured by adding a file named `.travis.yml`, which is a [YAML](#) format text file, to the root directory of the GitHub repository.

Travis CI automatically detects when a commit has been made and pushed to a repository that is using Travis CI, and each time this happens, it will try to build the project using `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](#):

```
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

```
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

```
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

    # download library to the temporary directory
    - wget https://github.com/PaulStoffregen/OneWire/archive/master.zip -O /tmp/
    ↵onewire_source.zip
    - unzip /tmp/onewire_source.zip -d /tmp/

script:
    - 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*

```
script:
  - platformio ci --project-conf=/path/to/platformio.ini
```

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="--DWIICAMERA"
  - 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)

(continued from previous 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  
  
    - wget https://github.com/Rodot/Gamebuino/archive/master.zip -O /tmp/gamebuino.  
↪zip  
    - 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

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:
- PLATFROMIO_CI_SRC=examples/TimeArduinoDue PLATFROMIO_CI_EXTRA_ARGS="--board=due"
- PLATFROMIO_CI_SRC=examples/TimeGPS
- PLATFROMIO_CI_SRC=examples/TimeNTP
- PLATFROMIO_CI_SRC=examples/TimeTeensy3 PLATFROMIO_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)

(continued from previous page)

```
- platformio lib -g install 416 421 422

script:
- platformio ci --lib="." --board=uno --board=teensy20pp $PLATFORMIO_CI_EXTRA_ARGS
```

- Configuration file: <https://github.com/ivankravets/Time/blob/master/.travis.yml>
- Build History: <https://travis-ci.org/ivankravets/Time>

4. Advanced configuration with extra project options and libraries

```
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=genuino101"
- 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

- platformio lib -g install 17 28 91 418 419 420 65 44 19 299 259 177 https://
github.com/vshymanskyy/BlynkESP8266.git https://github.com/cmaglie/FlashStorage.git_
https://github.com/michael71/Timer5.git

script:
- make travis-build
```

- Configuration file: <https://github.com/blynkkk/blynk-library/blob/master/.travis.yml>
- Build History: <https://travis-ci.org/blynkkk/blynk-library>

1.21 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:

1.21.1 2019

- Aug 18, 2019 - **Manuel Bleichenbacher** - Arduino In-circuit Debugging with PlatformIO
- Aug 13, 2019 - **Tech Explorations** - 6 reasons why PlatformIO is perhaps the best programming environment for the ESP32
- Jul 04, 2019 - **Jean-Claude Wippler** - The PlatformIO command line
- Mar 08, 2019 - **Nathan Glover** - Amazon Alexa controlled IoT Traffic Lights

1.21.2 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

1.21.3 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-contrôleurs (Start with PlatformIO alternative IDE for Arduino, ESP8266, ESP32 and other micro-controllers, French)
- Apr 12, 2017 - **Jane Elizabeth** - Let's talk IoT: PlatformIO puts developers back in the driver's seat
- Apr 07, 2017 - **AI 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

1.21.4 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** - PlataformIO 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** - Mikrocontroller-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 Holdys** - How to Integrate PlatformIO with Netbeans
- Feb 19, 2016 - **Embedds** - Develop easier with PlatformIO ecosystem
- Feb 13, 2016 - **Robert Cudmore** - Programming an arduino with PlatformIO
- Jan 24, 2016 - **Sergey Prilukin** - How to use IntelliJ IDEA to develop and upload software for micro controllers like Arduino
- Jan 16, 2016 - **Dani Eichhorn** - ESP8266 Arduino IDE Alternative: PlatformIO
- Jan 11, 2016 - **David Mills, Ph.D.** - STM NUCLEOF401RE TIMER IO
- Jan 05, 2016 - **Julien Rodrigues** - Internet Of Things: The IDE scandal

1.21.5 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 LaunchPads
- 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 - **nocd5** - PlatformIOembedSTM32 Nucleomruby (Use mruby in the offline build for STM32 Nucleo board with mbed and PlatformIO, Japanese)
- Oct 21, 2015 - **Vittorio Zaccaria** - Using a cheap STM32 Nucleo to teach remote sensor monitoring
- Oct 18, 2015 - **Nico Coetzee** - First Arduino I2C Experience with PlatformIO

- Oct 10, 2015 - **Floyd Hilton** - Programming Arduino with Atom
- Oct 01, 2015 - **Mistan** - Compile and Upload Arduino Sketch with PlatformIO for Raspberry Pi Running Arch Linux
- Sep 30, 2015 - **Jay Wiggins** - PlatformIO Investigation
- Sep 01, 2015 - **Thomas P. Weldon, Ph.D.** - Improvised MBED FRDM-K64F Eclipse/PlatformIO Setup and Software Installation
- Aug 08, 2015 - **Josh Glendenning** - Armstrap Eagle and PlatformIO
- Aug 01, 2015 - **Russell Davis** - PlatformIO on the Raspberry Pi
- Jul 25, 2015 - **DinoTools** - Erste Schritte mit PlatformIO (Getting Started with PlatformIO, German)
- Jul 20, 2015 - **Eli Fatsi** - Arduino Development in Atom Editor
- Jul 14, 2015 - **ElbinarIO** - Programar para Arduino y otros microcontroladores desde la linea de comandos (Program Arguno 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

1.21.6 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

1.22 Frequently Asked Questions

Note: We have a big database with [Frequently Asked Questions](#) in our [Community Forums](#). Please have a look at it.

Contents

- *General*
 - *What is PlatformIO?*
 - *What is .pio directory*
 - *What is .pioenvs directory*
 - *Command completion in Terminal*
 - * *Bash completion*
 - * *ZSH completion*
- *Install Python Interpreter*
- *Convert Arduino file to C++ manually*
- *Program Memory Usage*
- *Troubleshooting*
 - *Installation*
 - * *Multiple PIO Cores in a system*
 - * *'platformio' is not recognized as an internal or external command*
 - * *99-platformio-udev.rules*
 - * *ImportError: cannot import name _remove_dead_weakref*
 - *Package Manager*
 - * *[Error 5] Access is denied*
 - *Solution 1: Remove folder*
 - *Solution 2: Antivirus*
 - *Solution 3: Run from Terminal*
 - *Solution 4: Manual*
 - *Building*
 - * *UnicodeWarning: Unicode equal comparison failed*
 - * *UnicodeDecodeError: Non-ASCII characters found in build environment*
 - * *ARM toolchain: cc1plus: error while loading shared libraries*
 - * *Archlinux: libncurses.so.5: cannot open shared object file*
 - * *Monitoring a serial port breaks upload*

1.22.1 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*:

```
eval "$(_PLATFORMIO_COMPLETE=source platformio)"
eval "$(_PIO_COMPLETE=source pio)"
```

ZSH completion

To enable zsh completion please run these commands:

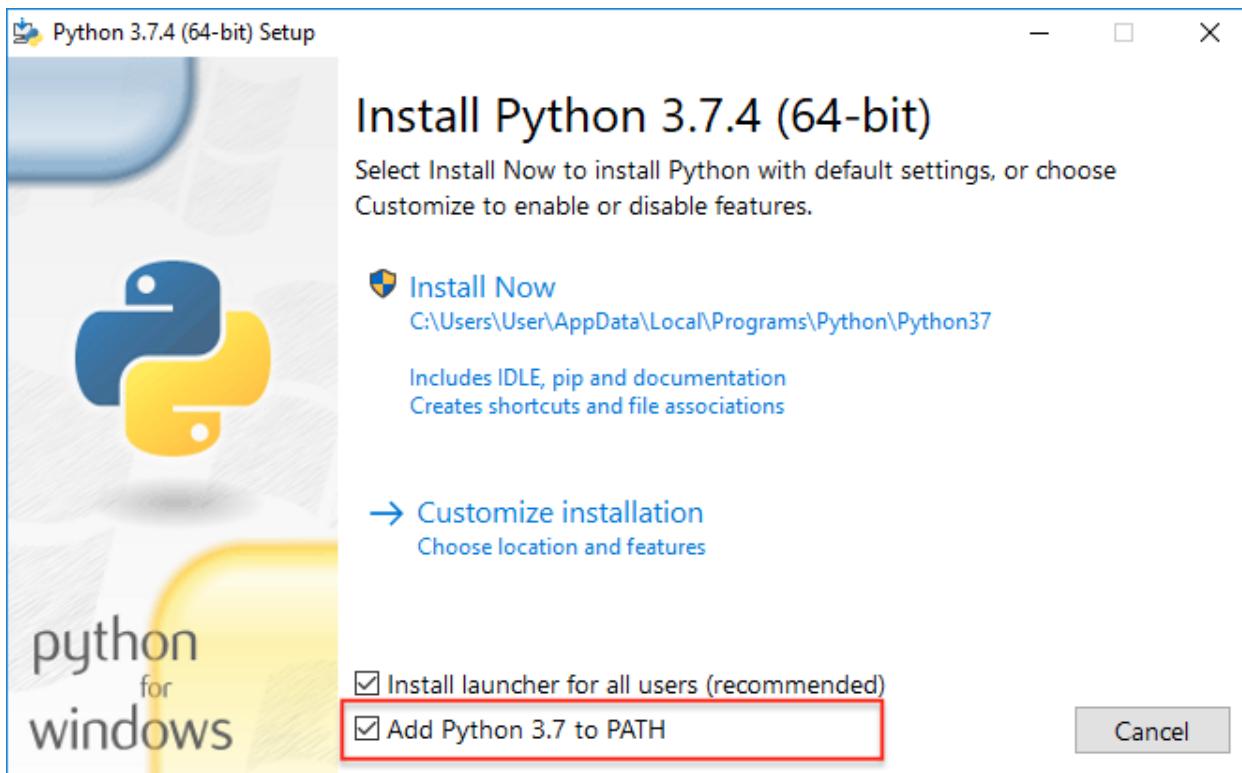
```
autoload bashcompinit && bashcompinit
eval "$(_PLATFORMIO_COMPLETE=source platformio)"
eval "$(_PIO_COMPLETE=source pio)"
```

Note: For permanent command completion you need to place commands above to `~/.bashrc` or `~/.zshrc` file.

1.22.2 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 to Path (see below), otherwise, `python` command will not be available.



1.22.3 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:

```
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`:

```
#include <Arduino.h>

void someFunction(int num);

void setup () {
    someFunction(13);
}

void loop() {
    delay(1000);
}

void someFunction(int num) {
```

Finish.

1.22.4 Program Memory Usage

PlatformIO calculates firmware/program memory usage based on the next segments:

.text The code segment, also known as a text segment or simply as text, is where a portion of an object file or the corresponding section of the program's virtual address space that contains executable instructions is stored and is generally read-only and fixed size.

.data The .data segment contains any global or static variables which have a pre-defined value and can be modified. The values for these variables are initially stored within the read-only memory (typically within .text) and are copied into the .data segment during the start-up routine of the program. Example,

```
int val = 3;
char string[] = "Hello World";
```

.bss Uninitialized data is usually adjacent to the data segment. The BSS segment contains all global variables and static variables that are initialized to zero or do not have explicit initialization in the source code. For instance, a variable defined as `static int i;` would be contained in the BSS segment.

The rough calculation could be done as:

- PROGRAM (Flash) = .text + .data
- DATA (RAM) = .bss + .data

If you need to print **all memory sections and addresses**, please use `platformio run --verbose` command.

Recommended for reading:

- https://en.wikipedia.org/wiki/Data_segment
- text, data and bss: Code and Data Size Explained

1.22.5 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* - Settings > Set `platformio-ide.useBuiltinPIOCore` to true.

Finally, if you have a standalone *PlatformIO Core (CLI)* in a system, please open system Terminal (not PlatformIO IDE Terminal) and uninstall obsolete PIO Core:

```
pip uninstall platformio

# if you used macOS "brew"
brew uninstall platformio
```

If you need to have *PlatformIO Core (CLI)* globally in a system, please [Install Shell Commands](#).

'platformio' is not recognized as an internal or external command

If you use *PlatformIO IDE*, please check in PlatformIO IDE Settings that “Use built-in 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

```
# Recommended
curl -fsSL https://raw.githubusercontent.com/platformio/platformio-core/master/
  ↵scripts/99-platformio-udev.rules | sudo tee /etc/udev/rules.d/99-platformio-udev.
  ↵rules

# OR, manually download and copy this file to destination folder
sudo cp 99-platformio-udev.rules /etc/udev/rules.d/99-platformio-udev.rules
```

Restart “udev” management tool:

```
sudo service udev restart

# or
```

(continues on next page)

(continued from previous page)

```
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
```

Similarly, Arch users may need to add their user to the “uucp” group

```
sudo usermod -a -G uucp $USER  
sudo usermod -a -G lock $USER
```

Note: You will need to log out and log back in again (or reboot) for the user group changes to take effect.

After this file is installed, physically unplug and reconnect your board.

ImportError: cannot import name _remove_dead_weakref

Windows users can experience this issue when multiple Python interpreters are installed in a system and conflict each other. The easy way to fix this problem is uninstalling all Python interpreters using Windows Programs Manager and installing them manually again.

1. “Windows > Start Menu > Settings > System > Apps & Features”, select Python interpreters and uninstall them.
2. Install the latest Python 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 Please replace {username} with a real user name
- Unix: ~/.platformio/penv/bin/platformio

Note: You can add platformio and pio commands to your system environment. See [Install Shell Commands](#).

Solution 4: Manual

If none of the solutions above do work for you, you can download and unpack all packages manually to “`core_dir/packages`”.

Please visit [PlatformIO Package Storage](#) and download a package for your platform. A correct package path is “`core_dir/packages/{package_name}/package.json`”.

Building

UnicodeWarning: Unicode equal comparison failed

Full warning message is “UnicodeWarning: Unicode equal comparison failed to convert both arguments to Unicode - interpreting them as being unequal”.

KNOWN ISSUE. Please move your project to a folder which full path does not contain non-ASCII chars.

UnicodeDecodeError: Non-ASCII characters found in build environment

KNOWN ISSUE. *PlatformIO Core (CLI)* currently does not support projects which contain non-ASCII characters (codes) in a full path or depend on the libraries which use non-ASCII characters in their names.

TEMPORARY SOLUTION

1. Use *PlatformIO IDE*, it will automatically install *PlatformIO Core (CLI)* in a root of system disk (%DISK%/.platformio) and avoid an issue when system User contains non-ASCII characters
2. Do not use non-ASCII characters in project folder name or its parent folders.

Also, if you want to place *PlatformIO Core (CLI)* in own location, see:

- Set `PLATFORMIO_CORE_DIR` environment variable with own path
- Configure custom location per project using `core_dir` option in “`platformio.ini`” (*Project Configuration File*).

ARM toolchain: cc1plus: error while loading shared libraries

See related answers for error while loading shared libraries.

Archlinux: libncurses.so.5: cannot open shared object file

Answered in issue #291.

Monitoring a serial port breaks upload

Answered in issue #384.

1.23 Release Notes

1.23.1 PlatformIO Core 4.0

4.1.0 (2019-??-??)

- PIO Check – automated code analysis without hassle:
 - Potential NULL pointer dereferences
 - Possible indexing beyond array bounds
 - Suspicious assignments
 - Reads of potentially uninitialized objects
 - Unused variables or functions
 - Out of scope memory usage.
- Extend project environment configuration in “`platformio.ini`” with other sections using a new `extends` option (issue #2953)
- Generate `.ccls` LSP file for Emacs cross references, hierarchies, completion and semantic highlighting
- Added `--no-ansi` flag for PIO Core to disable ANSI control characters

- Fixed an issue with project generator for [CLion IDE](#) when 2 environments were used ([issue #2824](#))
- Fixed default PIO Unified Debugger configuration for [J-Link probe](#)
- Fixed an issue when configuration file options partly ignored when using custom `--project-conf` ([issue #3034](#))
- Fixed an issue when installing a package using custom Git tag and submodules were not updated correctly ([issue #3060](#))
- Fixed an issue with linking process when `$LDSCRIPT` contains a space in path

4.0.3 (2019-08-30)

- Added support for multi-environment PlatformIO project for [CLion IDE](#) ([issue #2824](#))
- Generate `.ccls` LSP file for [Vim](#) cross references, hierarchies, completion and semantic highlighting ([issue #2952](#))
- Added support for `PLATFORMIO_DISABLE_COLOR` system environment variable which disables color ANSI-codes in a terminal output ([issue #2956](#))
- Updated SCons tool to 3.1.1
- Remove ProjectConfig cache when “platformio.ini” was modified outside
- Fixed an issue with PIO Unified Debugger on Windows OS when debug server is piped
- Fixed an issue when `-upload-port` CLI flag does not override declared `upload_port` option in “platformio.ini” ([Project Configuration File](#))

4.0.2 (2019-08-23)

- Fixed an issue with a broken [LDF](#) when checking for framework compatibility ([issue #2940](#))

4.0.1 (2019-08-22)

- 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
- Improved printing of tabulated results
- Automatically normalize file system paths to UNIX-style for Project Generator ([issue #2857](#))
- Ability to set “databaseFilename” for VSCode and C/C++ extension ([issue #2825](#))
- 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
- Fixed an issue with PIO Home’s “No JSON object could be decoded” ([issue #2823](#))
- Fixed an issue when `library.json` had priority over project configuration for [LDF](#) ([issue #2867](#))

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)

1.23.2 PlatformIO Core 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 `-f`, `--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 RemoteTM. 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

1.23.3 PlatformIO Core 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 Arduino.Org boards: Arduino Leonardo ETH, Arduino Yun Mini, Arduino Industrial 101 and Linino One ([issue #472](#))
- 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: TI LaunchPad 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 (ESP8266 Dev Board, ESP-WROOM-02, ESPRESSO 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 `uploaddeep` 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-ldscripts](#) 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 Ardhhat 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-teensy 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, Espres-sif 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 `-f, --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](#))

1.23.4 PlatformIO Core 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 `stm32` 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*, *RedBearLab*, *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

1.23.5 PlatformIO Core 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 no libs 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 *Microduino* 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

1.24 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)

1.24.1 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 *workspace_dir*.

Note: 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.

1.24.2 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 *Atmel AVR* “avrduude” programmer:

```
...
Looking for upload port...
Auto-detected: /dev/cu.usbmodemFA141
Uploading build/uno/firmware.hex

avrduude: AVR device initialized and ready to accept instructions

Reading | ##### | 100% 0.00s

avrduude: Device signature = 0x1e950f (probably m328p)
avrduude: reading input file "build/uno/firmware.hex"
avrduude: writing flash (930 bytes):

Writing | ##### |
```

1.24.3 What is new

In this section, we are going to highlight the most important changes and features introduced in *PlatformIO Core (CLI) 4.0*. Please visit [PlatformIO 4.0 Release Notes](#) for more detailed information.

“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 *Dependency Finder Mode* or non-existing *debug svd path*).

Global scope [env]

One of the most requested features was a “global” or “common” project environment (*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 *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_l1152re
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_l1152re
build_flags = -D RELEASE
lib_deps = Dep1, Dep2, Dep3

[env:debug]
platform = ststm32
framework = stm32cube
board = nucleo_l1152re
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)

(continued from previous 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.

Section	Option	Description
platformio	<i>extra_configs</i>	Extend base configuration with external “ <i>platformio.ini</i> ” (<i>Project Configuration File</i>)
platformio	<i>core_dir</i>	Directory where PlatformIO stores development platform packages (toolchains, frameworks, SDKs, upload and debug tools), global libraries for <i>Library Dependency Finder (LDF)</i> , and other PlatformIO Core service data
platformio	<i>global-lib_dir</i>	Global library storage for PlatfrmIO projects and <i>Library Manager</i> where <i>Library Dependency Finder (LDF)</i> looks for dependencies
platformio	<i>platforms_dir</i>	Global storage where PlatformIO Package Manager installs <i>Development Platforms</i>
platformio	<i>packages_dir</i>	Global storage where PlatformIO Package Manager installs <i>Development Platforms</i> dependencies (toolchains, <i>Frameworks</i> , SDKs, upload and debug tools)
platformio	<i>cache_dir</i>	<i>PlatformIO Core (CLI)</i> uses this folder to store caching information (requests to PlatformIO Registry, downloaded packages and other service information)
platformio	<i>workspace_dir</i>	Path to a project workspace directory where PlatformIO keeps by default compiled objects, static libraries, firmwares, and external library dependencies
platformio	<i>shared_dir</i>	<i>PIO Remote</i> uses this folder to synchronize extra files between remote machine
env	<i>build_type</i>	See extended documentation for <i>Build Configurations</i>
env	<i>monitor_flags</i>	Pass extra flags and options to <i>platformio device monitor</i> command
env	<i>upload_command</i>	Override default <i>Development Platforms</i> upload command with a custom one.

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 Core 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 Core 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 `--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: Library Dependency Finder -> http://bit.ly/configure-pio-ldf
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_toolchain]
platform = atmelavr
platform_packages =
; use GCC AVR 5.0+
toolchain-gccarmnoneabi@>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*.

1.24.4 What is changed or removed

Command Line Interface

The following commands have been changed in v4.0.

Command	Description
<code>platformio run</code>	Added <code>platformio run --jobs</code> option
<code>platformio update</code>	Replaced <code>-c</code> , <code>--only-check</code> with <code>platformio update --dry-run</code>
<code>platformio lib update</code>	Replaced <code>-c</code> , <code>--only-check</code> with <code>platformio lib update --dry-run</code>
<code>platformio platform update</code>	Replaced <code>-c</code> , <code>--only-check</code> with <code>platformio platform update --dry-run</code>
<code>platformio remote update</code>	Replaced <code>-c</code> , <code>--only-check</code> with <code>platformio remote update --dry-run</code>

“platformio.ini” (Project Configuration File)

The following options have been changed in v4.0.

Section	Option	Description
plat-formio	env_default	Renamed to <code>default_envs</code>
plat-formio	home_dir	Renamed to <code>core_dir</code>
env	debug_load_cmd	Renamed to <code>debug_load_cmds</code> and allowed to pass more than one load command

Bibliography

[Embedds] Embedds.com: Develop easier with PlatformIO ecosystem

Symbols

--quiet
 platformio-device-monitor command
 line option, 34
 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

-filter
 platformio-check command line
 option, 1746

-flags
 platformio-check command line
 option, 1747

-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-check command line
        option, 1747
    platformio-device-list command
        line option, 32
    platformio-lib-builtin 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-ansi
    platformio command line option, 19
-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
-rtscs
    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
-severity
    platformio-check command line
        option, 1747
-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
-username, -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
-a, --author
    platformio-lib-search command line
        option, 55
-b, --baud
    platformio-device-monitor command
        line option, 34
    platformio-remote-device-monitor
        command line option, 90
-b, --board
    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-check command line
        option, 1747
    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-check command line
        option, 1747
    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-run command line option,
        103
    platformio-test command line
        option, 115
-d, --storage-dir
    platformio-lib command line option,
        40
-d, --working-dir

```

```
platformio-remote-agent-start
    command line option, 87
-e, -environment
    platformio-check command line
        option, 1746
    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-check command line
        option, 1747
    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-check command line
        option, 1747
    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, 220
 PLATFORMIO_AUTH_TOKEN, 21, 24, 220, 1752, 1993
 PLATFORMIO_BOARDS_DIR, 190, 221
 PLATFORMIO_BUILD_CACHE_DIR, 187, 221
 PLATFORMIO_BUILD_DIR, 188, 221
 PLATFORMIO_BUILD_FLAGS, 196, 222, 1984, 1994
 PLATFORMIO_CACHE_DIR, 187, 221
 PLATFORMIO_CORE_DIR, 186, 221, 2010
 PLATFORMIO_DATA_DIR, 190, 221
 PLATFORMIO_DEFAULT_ENVS, 184, 222
 PLATFORMIO_DISABLE_PROGRESSBAR, 220
 PLATFORMIO_EXTRA_SCRIPTS, 222, 224
 PLATFORMIO_FORCE_ANSI, 19, 220
 PLATFORMIO_GLOBALLIB_DIR, 186, 221
 PLATFORMIO_INCLUDE_DIR, 188, 221
 PLATFORMIO_LIB_DIR, 189, 221
 PLATFORMIO_LIB_EXTRA_DIRS, 206, 222
 PLATFORMIO_LIBDEPS_DIR, 188, 221
 PLATFORMIO_NO_ANSI, 19, 220
 PLATFORMIO_PACKAGES_DIR, 187, 221
 PLATFORMIO_PLATFORMS_DIR, 186, 221
 PLATFORMIO_REMOTE_AGENT_DIR, 222
 PLATFORMIO_SETTING_AUTO_UPDATE_LIBRARIES, 222
 PLATFORMIO_SETTING_AUTO_UPDATE_PLATFORMS, 223
 PLATFORMIO_SETTING_CHECK_LIBRARIES_INTERVAL, 223
 PLATFORMIO_SETTING_CHECK_PLATFORMIO_INTERVAL, 223
 PLATFORMIO_SETTING_CHECK_PLATFORMS_INTERVAL, 223
 PLATFORMIO_SETTING_ENABLE_CACHE, 223
 PLATFORMIO_SETTING_ENABLE_TELEMETRY, 223
 PLATFORMIO_SETTING_FORCE_VERBOSE, 29, 30, 94, 97, 103, 116, 223, 1747
 PLATFORMIO_SETTING_PROJECTS_DIR, 223
 PLATFORMIO_SETTING_STRICT_SSL, 223
 PLATFORMIO_SHARED_DIR, 190, 221
 PLATFORMIO_SRC_BUILD_FLAGS, 198, 222
 PLATFORMIO_SRC_DIR, 189, 221
 PLATFORMIO_SRC_FILTER, 199, 222
 PLATFORMIO_TEST_DIR, 190, 221
 PLATFORMIO_UPLOAD_FLAGS, 200, 222
 PLATFORMIO_UPLOAD_PORT, 200, 222
 PLATFORMIO_WORKSPACE_DIR, 183, 188, 221

P

platformio command line option

-help, -h, 19
 -no-ansi, 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
 -json-output, 23
 platformio-account-token command line option
 -json-output, 24
 -regenerate, 24
 platformio-boards command line option
 -installed, 24
 -json-output, 25
 platformio-check command line option
 -filter, 1746
 -flags, 1747
 -json-output, 1747
 -severity, 1747
 -c, -project-conf, 1747
 -d, -project-dir, 1747
 -e, -environment, 1746
 -s, -silent, 1747
 -verbose, 1747
 platformio-ci command line option
 -build-dir, 28
 -exclude, 28
 -keep-build-dir, 28
 -O, -project-option, 28
 -b, -board, 28
 -c, -project-conf, 28
 -l, -lib, 28
 -v, -verbose, 28
 platformio-debug command line option
 -interface, 30
 -c, -project-conf, 29
 -d, -project-dir, 29
 -e, -environment, 29
 -v, -verbose, 30
 platformio-device-list command line option
 -json-output, 32
 -logical, 31
 -mdns, 32
 -serial, 31

```
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
    -env-prefix, 38
    -ide, 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, 1752, 1993
PLATFORMIO_BOARDS_DIR, 190
PLATFORMIO_BUILD_CACHE_DIR, 187
PLATFORMIO_BUILD_DIR, 188
PLATFORMIO_BUILD_FLAGS, 196, 1984, 1994
PLATFORMIO_CACHE_DIR, 187
PLATFORMIO_CORE_DIR, 186, 2010
PLATFORMIO_DATA_DIR, 190
PLATFORMIO_DEFAULT_ENVS, 184
PLATFORMIO_DISABLE_PROGRESSBAR, 220
PLATFORMIO_EXTRA_SCRIPTS, 224
PLATFORMIO_FORCE_ANSI, 19
PLATFORMIO_GLOBALLIB_DIR, 186
PLATFORMIO_INCLUDE_DIR, 188
PLATFORMIO_LIB_DIR, 189
PLATFORMIO_LIB_EXTRA_DIRS, 206
PLATFORMIO_LIBDEPS_DIR, 188
PLATFORMIO_NO_ANSI, 19
PLATFORMIO_PACKAGES_DIR, 187
PLATFORMIO_PLATFORMS_DIR, 186
PLATFORMIO_SETTING_FORCE_VERBOSE, 29, 30,
    94, 97, 103, 116, 1747
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

```