



Morse Micro
reaching farther™

Platform IO + MM IoT SDK User Guide

November 2023

Author: Matthew Forgie

© 2023 Morse Micro | morsemicro.com

Table of Contents

1. Scope	3
2. Pre-requisites	3
3. Installation	3
3.1 Getting started with Visual Studio Code	3
3.2 PlatformIO extension	5
3.3 Install the MM IoT-SDK for PlatformIO	6
4. Basic Usage	8
4.1 Connecting the evaluation board to PC	8
4.2 Opening an example project	11
4.3 Building and Uploading	13
4.4 Preparing the config store partition	15
4.5 Programming the config store partition	17
4.6 Using serial monitor	17
5. Creating a custom project	19
5.1 Creating a new project	19
5.2 Opening additional projects	20
6. Advanced features	22
6.1 Debugging	22
6.2 Inspection tool	23
6.1.1 Static code analysis	25
6.1.2 Memory analysis	26
7. Troubleshooting	28
7.1 I can't identify my serial port!	28
7.2 Upload failed	28
7.3 Error: jtag status contains invalid mode value	28
7.4. Debug session doesn't launch	29
8. Revision History	29

1. Scope

An important use case for HaLow is in embedded systems, and Morse Micro provides the MM-IoT-SDK as a starting point for developing FreeRTOS based embedded software with HaLow support. This can present a steep learning curve for new users, and PlatformIO is aimed at making it easier to get started. PlatformIO is an open source multi-platform build system with integration available for various IDEs. It has powerful built-in features such as debugging, unit testing, and static code analysis and it supports a multitude of hardware platforms. PlatformIO also has a vast amount of libraries and example code available for various sensors and actuators used in IoT applications. The MM IoT SDK provides an integration for PlatformIO that supports multiple development boards from ST Microelectronics combined with Morse Micro extension board for developing Wi-Fi HaLow enabled applications.

This user guide covers installation and basic setup of PlatformIO on a development system, using Visual Studio Code (also referred to as VS Code). It also includes usage of PlatformIO Morse Micro IoT framework (MM-IoT-framework).

2. Pre-requisites

1. Computer running Windows/Linux/macOS.
2. MM-IoT SDK release 2.0.x or later
3. Visual Studio Code installed
4. One of the Morse Micro IOT evaluation kits, e.g. EKH08-U575, EKH08-F429, or EKH08-WB55.
5. An internet connection.

3. Installation

3.1 Getting started with Visual Studio Code

This user guide will assume the user's editor of choice is Visual Studio Code (VS Code).

Install Visual Studio Code for your platform from <https://code.visualstudio.com/>.

VS Code offers a rich user interface, and is highly extensible. Familiarity with VS Code is recommended when using the MM IoT SDK as PlatformIO heavily leverages UI features of the editor in which it is installed.

See <https://code.visualstudio.com/docs/getstarted/userinterface> for a full description of the UI elements. Notable elements relevant to PlatformIO, and referenced throughout this User Guide are shown in Figure 1.

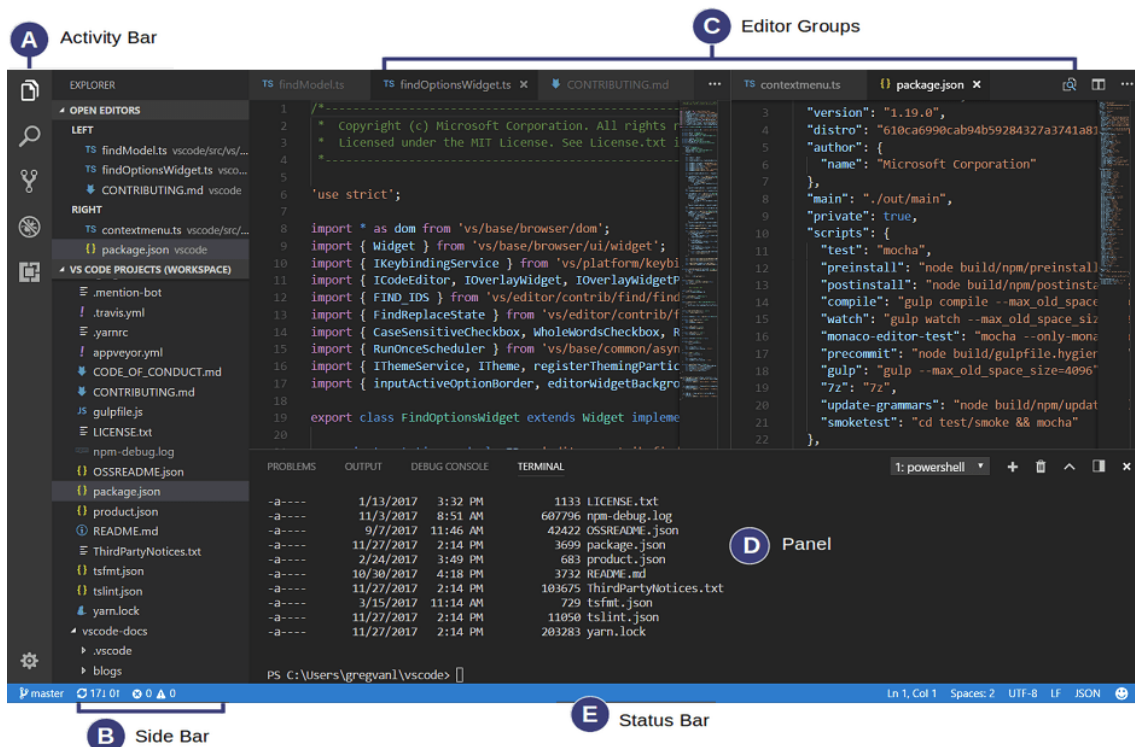


Figure 1. The VS Code UI

In addition, the Command Palette (shown in Figure 2) can be opened with CTRL+SHIFT+p. This may also be opened by PlatformIO features that require additional user input.

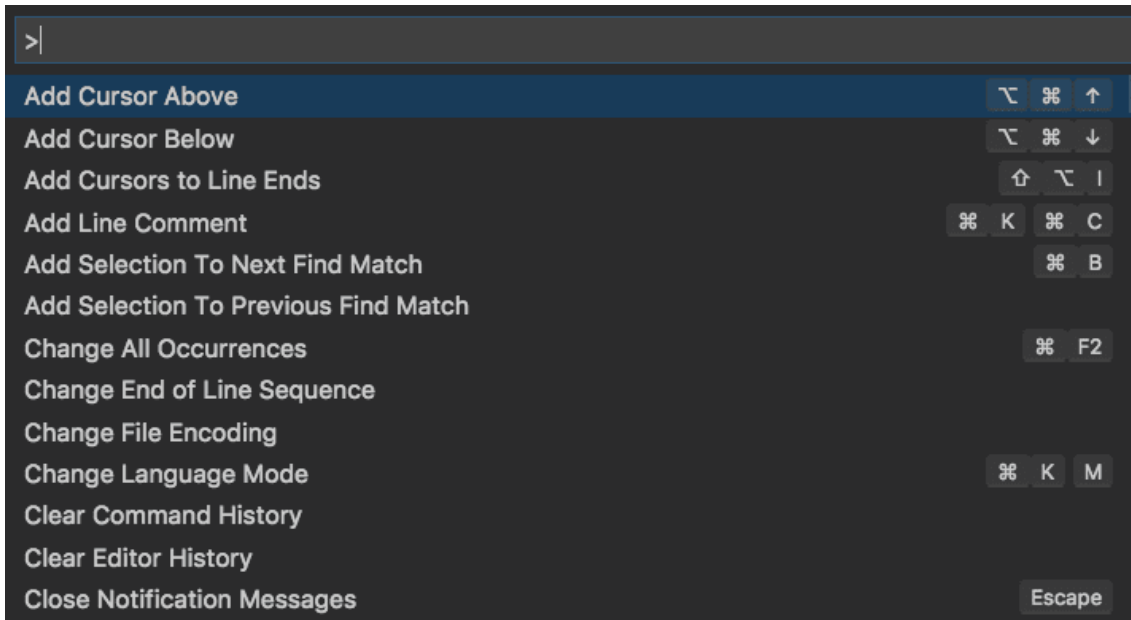


Figure 2. The VS Code command palette

3.2 Install PlatformIO extension for VS Code

After installing VS Code, and launching it, follow the following steps to install the PlatformIO IDE extension (see Figure 3 below)

1. Open Extensions from the activity bar (on the left).
2. Type platformio to the search box and the PlatformIO IDE extension will appear)
3. Click install and wait for the installation to complete. You should see a notification box in the lower right corner when this occurs.

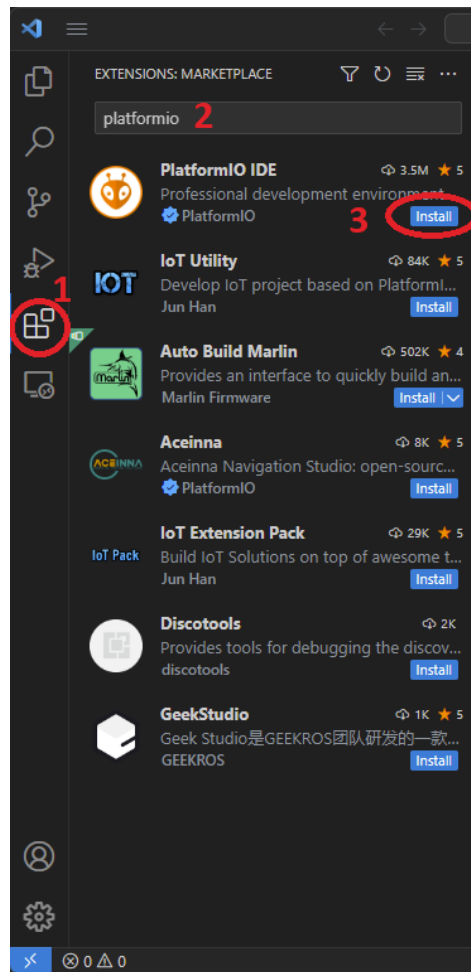


Figure 3. Installing the PlatformIO extension in VS Code

Optionally, a user may find it useful to install the following extensions for VS Code :

- Python - language support package
- hjson - syntax highlighting for the configstore file
- C/C++ - syntax highlighting for C/C++.

3.3 Install the MM IoT-SDK for PlatformIO

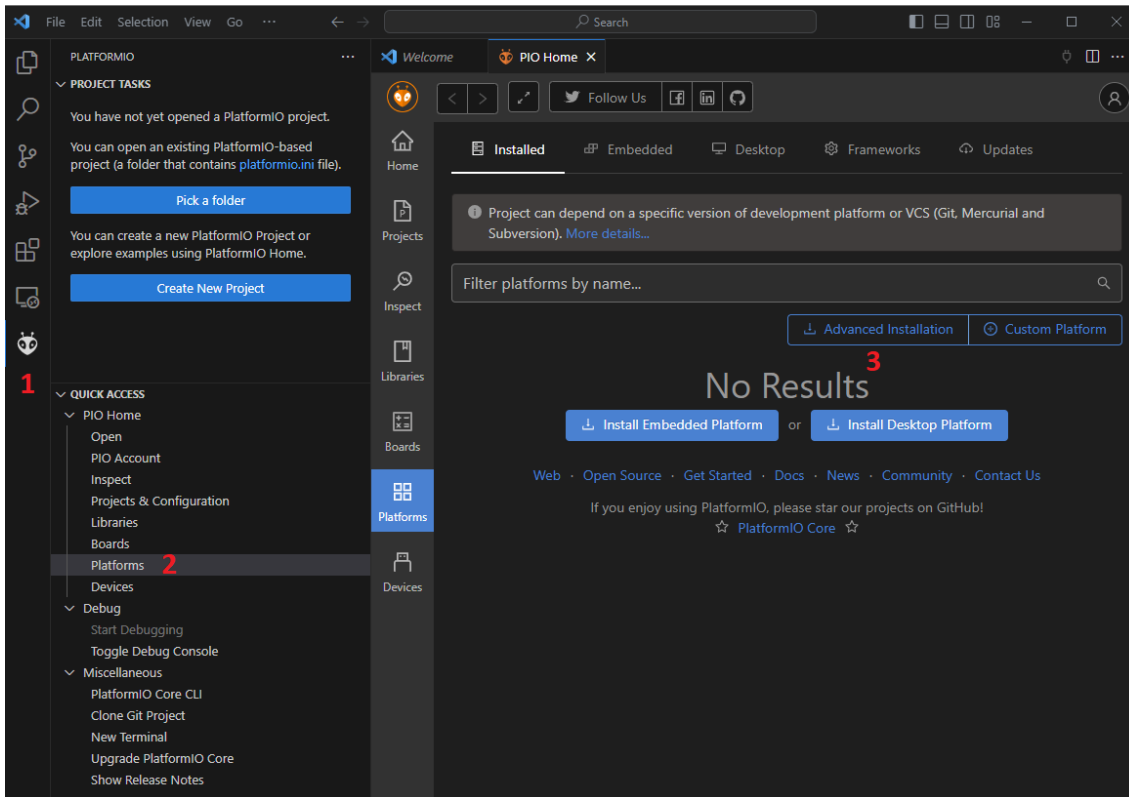


Figure 4. Installation of external platform - navigation

1. Click the PlatformIO icon from the Visual Studio Code activity bar on the left, and wait until you can see a quick access menu (see Figure 4 above). If the installation has not yet completed you may see "Initializing PlatformIO Core..." in the section where Quick Access is seen in Figure 3.
2. Under Quick Access → PIO Home, select Platforms
3. In the Platforms view, click Advanced Installation
4. In the Window which pops up (see Figure 5), the user will need to enter the full path to the MM-IoT-SDK.zip file, prefixed with the file:// pseudo protocol. Note, different host platforms have different formats for file paths. For example, if the mm-iot-sdk zip archive is located in the Users download folder, they could look as follows
 - a. Windows:
file://C:/Users/MorseMicro/Downloads/mm-iot-sdk-2.0.0.zip
 - b. macOS:
file:///Users/MorseMicro/Downloads/mm-iot-sdk-2.0.0.zip
 - c. Ubuntu:
file:///home/MorseMicro/Downloads/mm-iot-sdk-2.0.0.zip

5. An internet connection is required here, as the platform installation needs to download the appropriate arm toolchain. This process can take some time. Once completed, a "Platform has been successfully installed" message will appear.

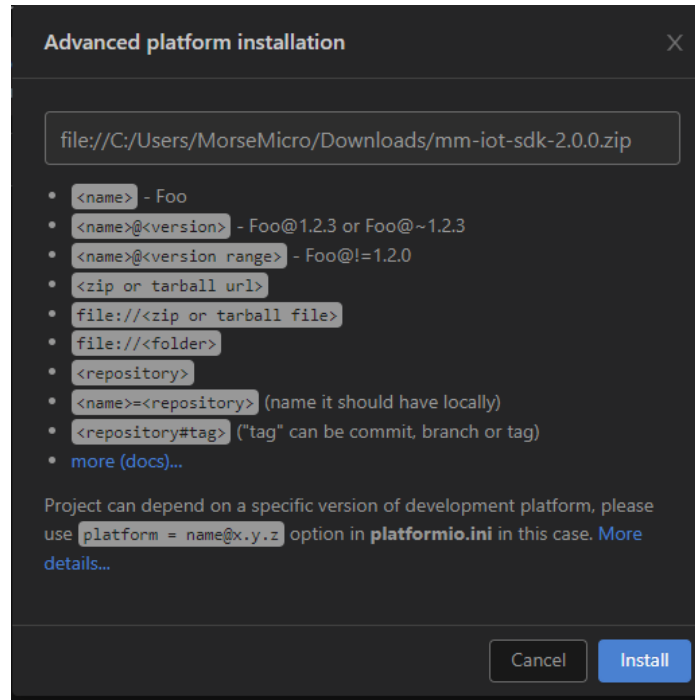


Figure 5. Installation of external platform - zip selection

Unless otherwise specified, PlatformIO will use the installed MM-IoT-SDK with the highest version number. Though it is possible to maintain multiple versions of the MM IoT-SDK in PlatformIO, it can often lead to confusion regarding which version a project is configured to use. In addition, having multiple installations of the SDK can result in file names that exceed the maximum path length on Windows. As a result, it is recommended to remove previous installations. This can be done by selecting "Uninstall" on old versions in the Platforms view.

For macOS, you may need to give VS Code permission to the folder where the MM-IoT-SDK.zip package resides. This can be done by opening Settings → Privacy & Security → Files & Folders and Enable folder access to VS Code (see Figure 6)

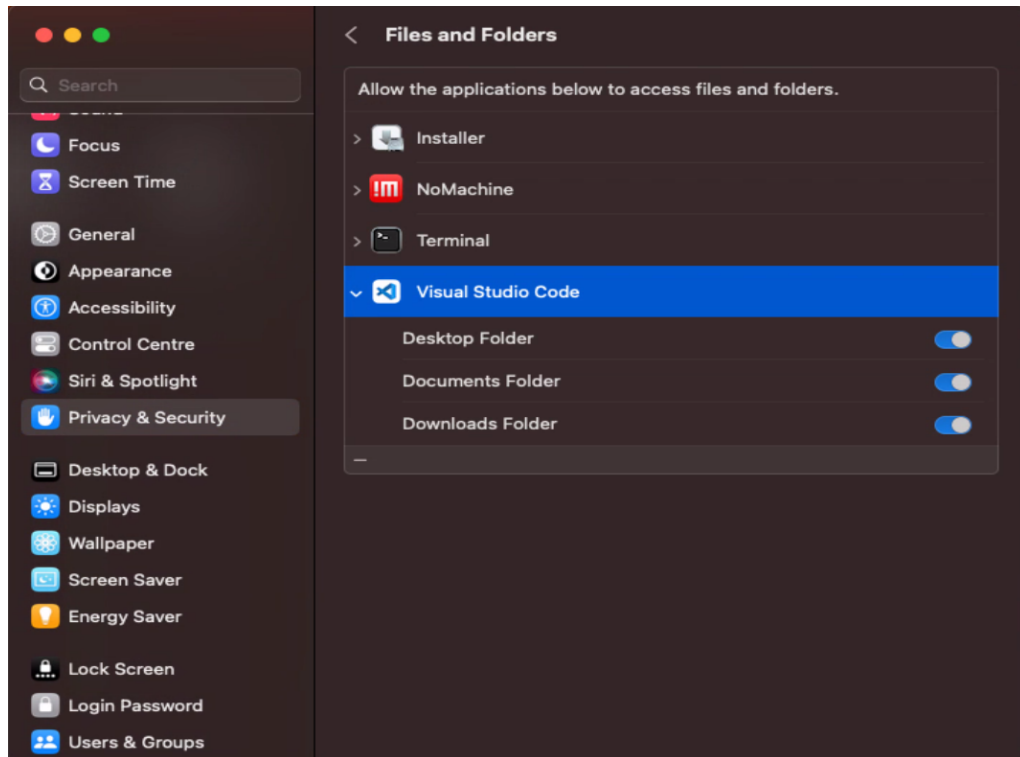


Figure 6. Allowing Visual Studio Code access to folders

4. Basic Usage

This section walks you through how to select one of the example applications and configure, compile, upload, and run it on Morse Micro evaluation hardware.

4.1 Connecting the evaluation board to PC

The evaluation boards have an integrated STLINK debugger/programmer that is connected to the PC via a USB cable. If this is the first time an STLINK has been attached to the host machine, you may need to install the necessary drivers or configure rules to enable the debugger.

- For Windows, drivers can be found at [STSW-LINK009](#), and should be installed via device manager.
- For Linux, configuration of udev rules may be required, instructions on [PlatformIO](#) should be followed.
- For Mac, no additional software or configuration should be required.

The power to the board is supplied via USB. Connect the micro-USB end to the evaluation board (see "Micro-USB to PC" in Figures 6 and 7) and the USB-A connector to your PC.

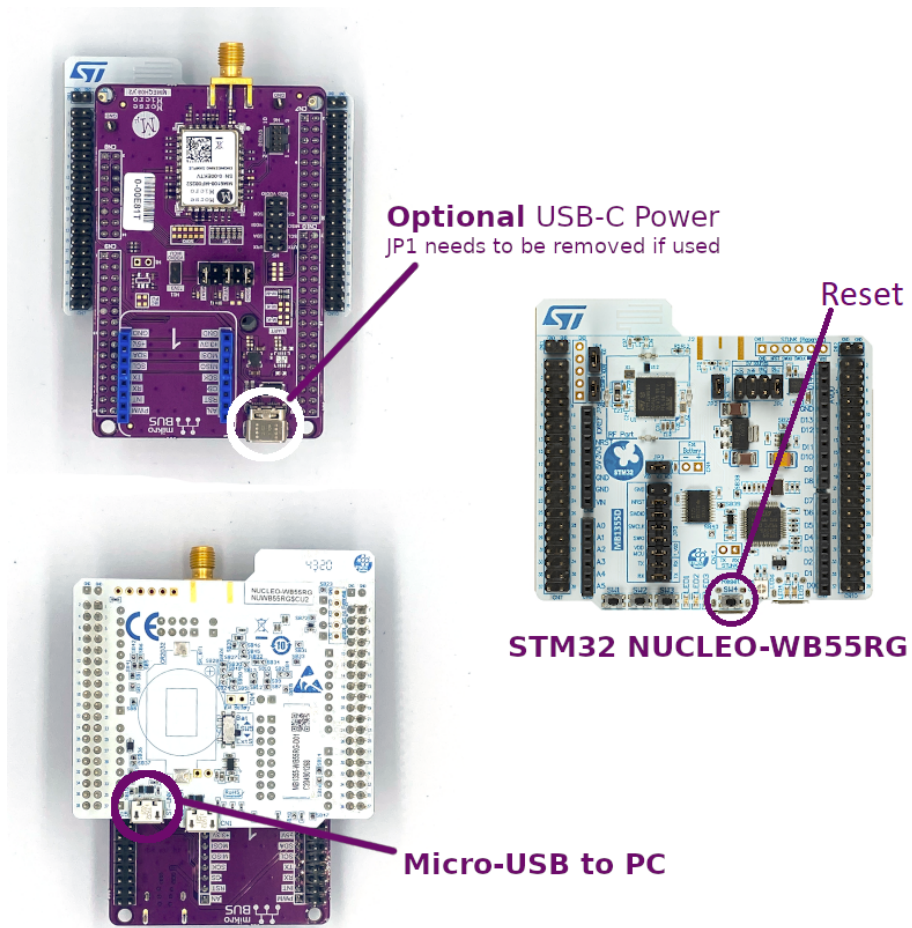


Figure 7. EKH08-WB55 evaluation board

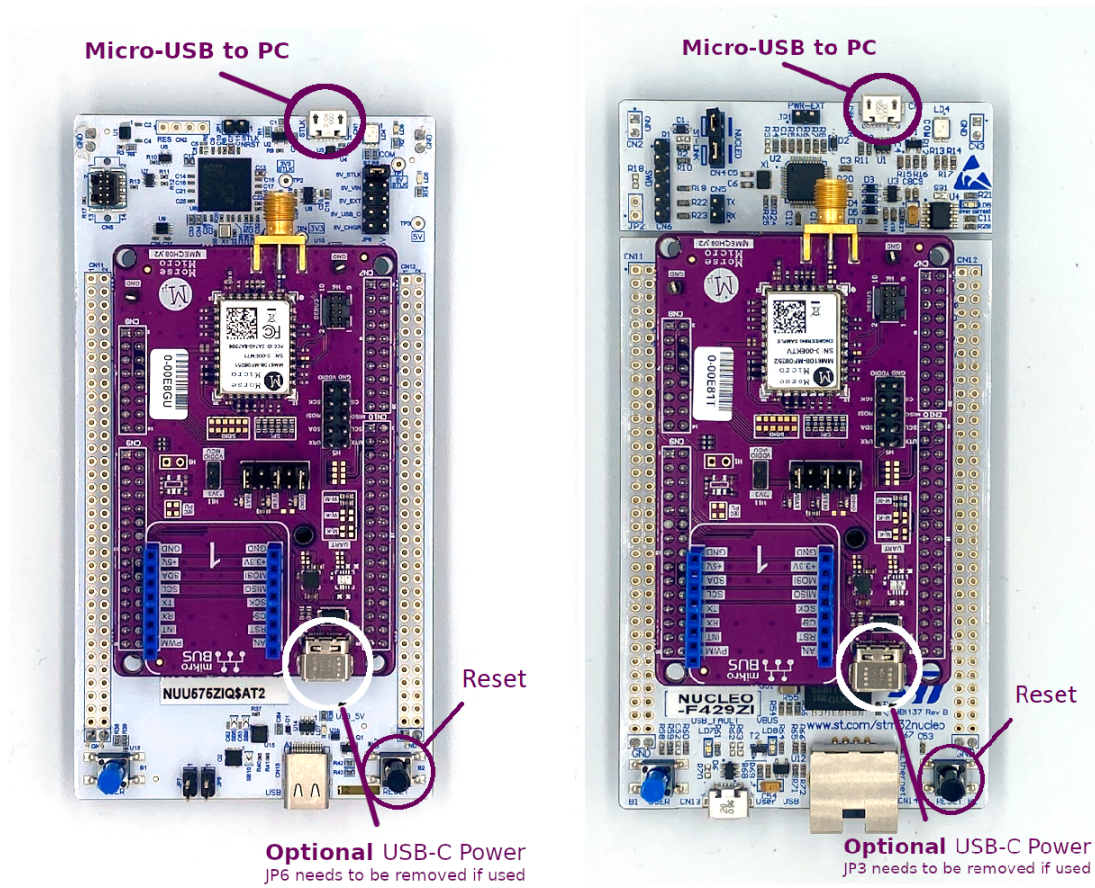


Figure 8. EKH08-U575 (left) and EKH08-F429 (right) evaluation boards

4.2 Opening an example project

To get started with one of the examples supplied in the MM-IoT-SDK:

1. Open Visual Studio Code and click the PlatformIO icon on the activity bar (Figure 9).
2. Click *Create New Project*, or alternatively, select *Quick Access* → *PIO Home* → *Open*.
3. From the opened dialog click *Project Examples* (Figure 10).
4. The example projects for “Morse Micro IoT Toolkit” will be visible (Figure 11). Select one of the projects (for example, ‘iperf’) and click *Import*. A copy of the example project is created and it can be freely modified.

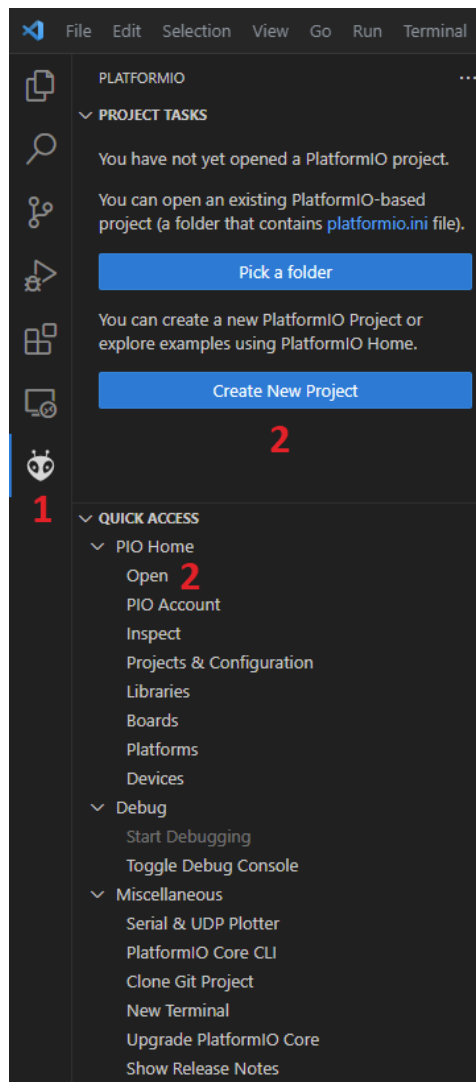


Figure 9. Creating new project

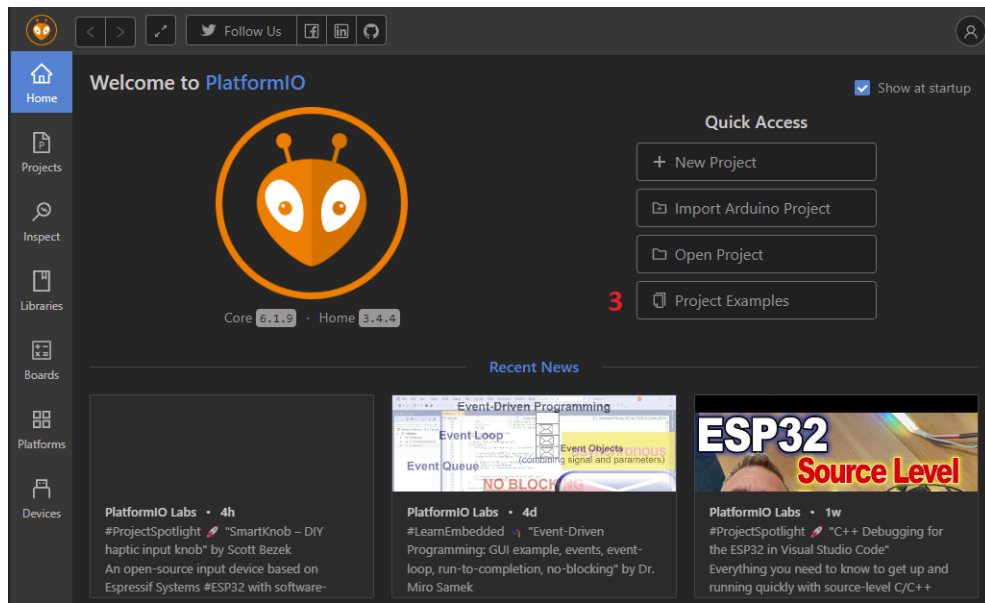


Figure 10. Project creation dialog

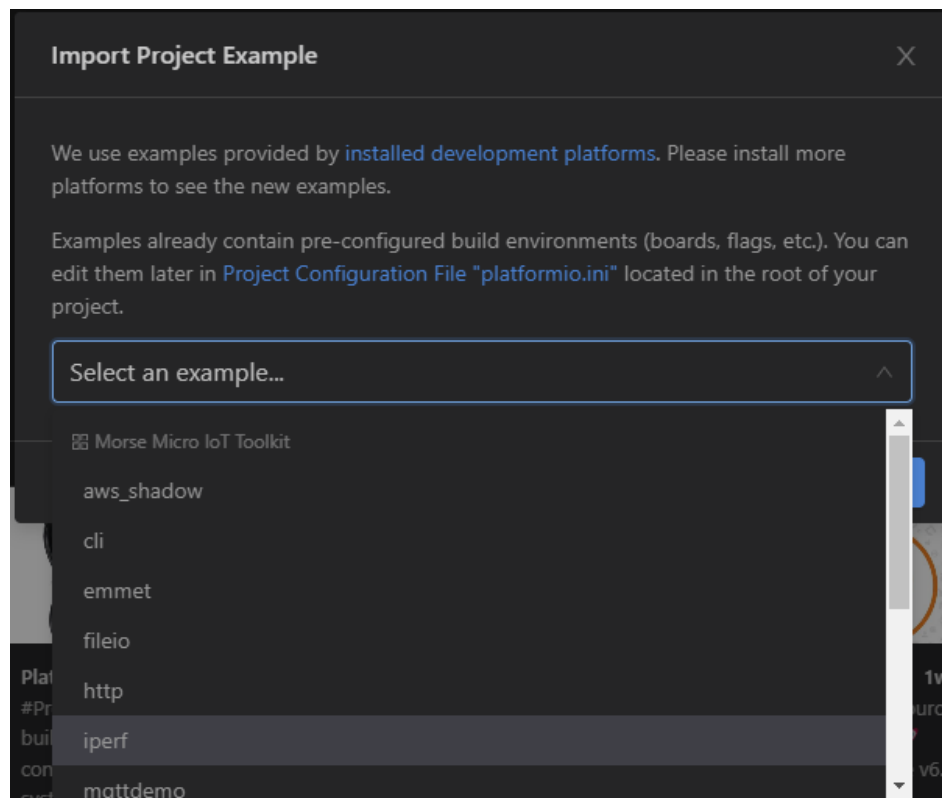


Figure 11. Importing mm-iot-framework examples

The project will take a few seconds to configure, and may ask you to ‘trust’ the authors of the file and folder. You will need to select “Yes” in order to proceed. Once this has completed, a list of project files on the left in the explorer window pane, and the source code for platform.ini in the main window pane.

At the bottom of Visual Studio Code, in the status bar, it is recommended to “Switch PlatformIO Project Environment” to target your desired board. The “Default” selected refers to the first environment specified in the platform.ini — typically the mm-ekh08-u575.

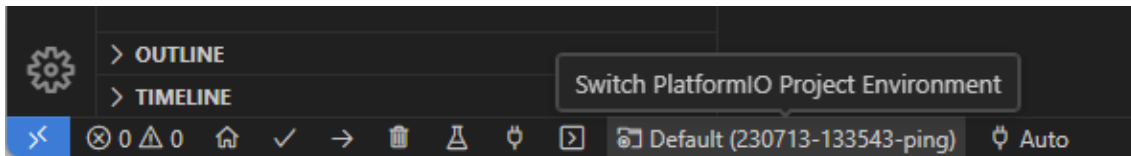


Figure 12. Switching from the default environment

Additionally, you may want to manually specify the serial port, particularly if there are multiple attached to your PC, so that the serial console works as expected. Click on “Auto” in the status bar and a drop-down at the top of Visual Studio Code will open to select from a list of available serial ports. See the Troubleshooting section for further information if your serial port does not show.

4.3 Building and Uploading

To build the project, select the PlatformIO icon from the activity bar, and under the Project Tasks for your platform, select Build (Figure 13).

Building the image can also be triggered by clicking the ‘tick’ in the status bar.

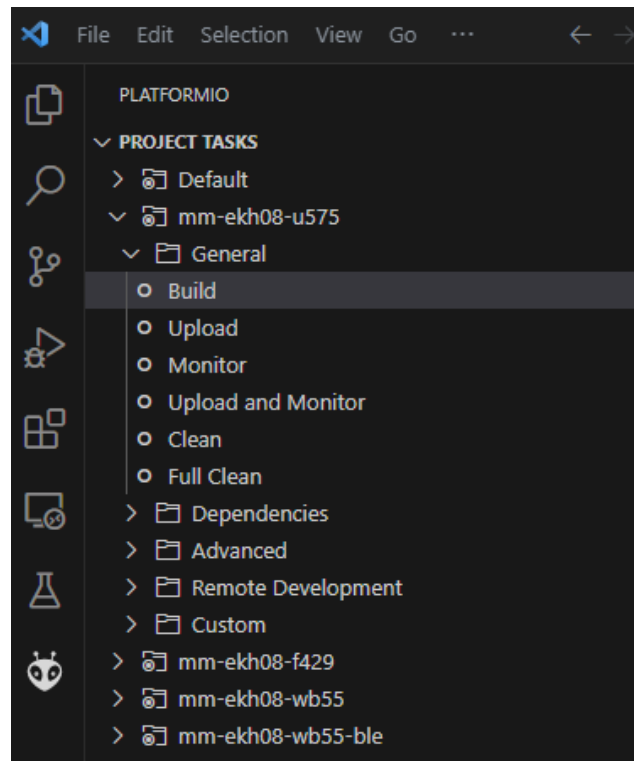


Figure 13. Building from Project Tasks.

The build task will execute, and show progress and status in the terminal output of VS Code (example shown in Figure 14).

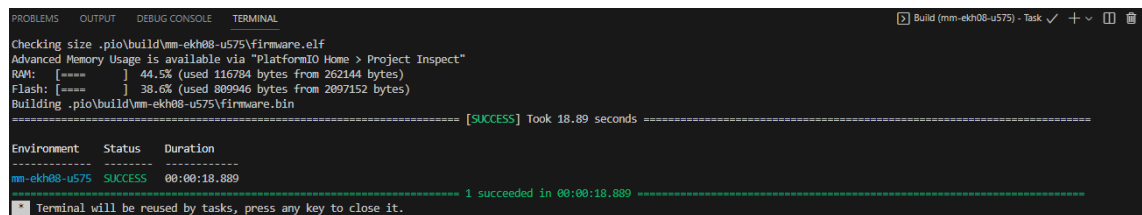
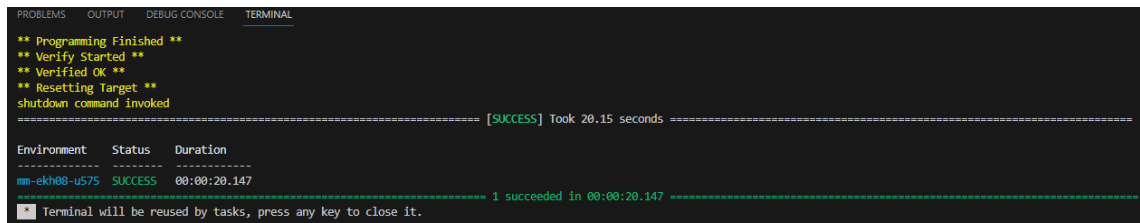


Figure 14. Build status terminal output

Once built successfully, the binary can be uploaded to the target platform (note configuration needs to be set up and loaded onto the board as well, see section 4.4 below). This can be executed by selecting Upload from Project Tasks. Alternatively, you can trigger an upload by clicking the right-arrow shortcut in the status bar.

Progress and status of the upload procedure will be shown in the terminal output (example shown in Figure 15).



```

PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL
** Programming Finished **
** Verify Started **
** Verified OK **
** Resetting Target **
shutdown command invoked
===== [SUCCESS] Took 20.15 seconds =====
Environment  Status  Duration
-----
mm-ekh08-u575  SUCCESS  00:00:20.147
===== 1 succeeded in 00:00:20.147 =====
Terminal will be reused by tasks, press any key to close it.

```

Figure 15. Upload status terminal output

4.4 Preparing the config store partition

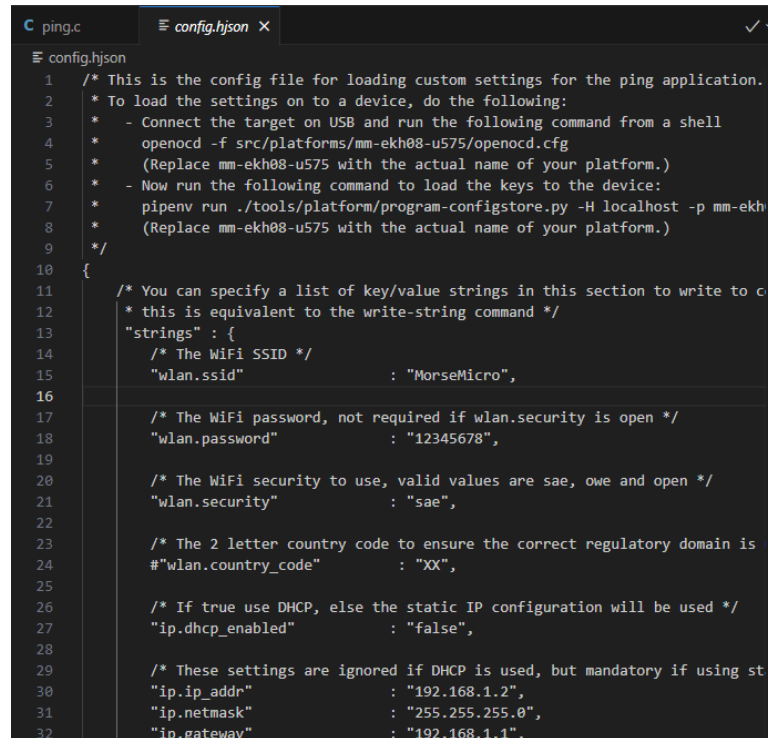
The MM-IoT-SDK example applications load configuration from the config store partition, if found on the device. Each example project includes a config.hjson which can be serialised and loaded onto the device.

For the applications demonstrating HaLow capability, they will need to be configured as appropriate for your application before they will run correctly. To modify the configuration, open the config.hjson file from the explorer pane on the left. An example config.hjson snippet is shown in Figure 16.

Some notable configuration fields typically required for the example applications include:

wlan.ssid	SSID of the HaLow network to connect to
wlan.password	Password of the HaLow network if security type is SAE
wlan.security	Security type of the HaLow network (sae, owe, or open)
wlan.country_code	Two character country code for the applicable regulatory domain
ip.dhcp_enabled	true to enable DHCP, or false to use a static IP address
ip.ip_addr	Static IP address to use if DHCP is disabled
ip.netmask	Netmask to use if DHCP is disabled
ip.gateway	Gateway IP address to use if DHCP is disabled
bcf_file	Board Configuration File

In addition, there may be application specific configuration, such as `ping.target` to specify the target IP address for the ping example application.



```

1  /* This is the config file for loading custom settings for the ping application.
2  * To load the settings on to a device, do the following:
3  *   - Connect the target on USB and run the following command from a shell
4  *   openocd -f src/platforms/mm-ekh08-u575/openocd.cfg
5  *   (Replace mm-ekh08-u575 with the actual name of your platform.)
6  *   - Now run the following command to load the keys to the device:
7  *   pipenv run ./tools/platform/program-configstore.py -H localhost -p mm-ekh
8  *   (Replace mm-ekh08-u575 with the actual name of your platform.)
9  */
10 {
11     /* You can specify a list of key/value strings in this section to write to c
12     * this is equivalent to the write-string command */
13     "strings" : {
14         /* The WiFi SSID */
15         "wlan.ssid" : "MorseMicro",
16
17         /* The WiFi password, not required if wlan.security is open */
18         "wlan.password" : "12345678",
19
20         /* The WiFi security to use, valid values are sae, owe and open */
21         "wlan.security" : "sae",
22
23         /* The 2 letter country code to ensure the correct regulatory domain is
24         #wlan.country_code" : "XX",
25
26         /* If true use DHCP, else the static IP configuration will be used */
27         "ip.dhcp_enabled" : "false",
28
29         /* These settings are ignored if DHCP is used, but mandatory if using st
30         "ip.ip_addr" : "192.168.1.2",
31         "ip.netmask" : "255.255.255.0",
32         "ip.gateway" : "192.168.1.1",

```

Figure 16. *config.hjson snippet*

Most config store fields have default values specified, but there are two important fields that must be specified explicitly. The `wlan.country_code` field must be set to the 2 letter code for your applicable regulatory domain and the `bcf_file` field must specify the path to the appropriate Board Configuration File.

The Board Configuration File (BCF) is a configuration file for the Morse Micro chip that provides board-specific information and is required for the system to function properly. BCFs for Morse Micro modules are included in the MM-IoT-SDK package. Note that these files use a special format that differs from regular Linux BCFs and they are not compatible. These modules can be identified by the part numbers printed on their labels, which take the form MM6108-MFxxxxxx.

BCF files for Morse Micro evaluation boards are provided as part of the SDK in the following locations:

- Windows:
 - `C:/Users/<user>/platformio/platforms/mm-iottoolkit/framework/morsefirmware`
- macOS

- /Users/<user>/.platformio/platforms/mm_iottoolkit/framework/morsefirmware
- Linux
 - /home/<user>/.platformio/platforms/mm_iottoolkit/framework/morsefirmware

For MM-IoT-SDK versions 2.1.x and newer, the variable \$MMIOT_ROOT can be specified as part of the file path and this will be substituted with the path to the *framework* directory. Thus, for the module with part number MM6108-MF082551, the `bcf_file` config field should be set as follows:

```
"bcf_file": "$MMIOT_ROOT/morsefirmware/bcf_mf08551.mbin"
```

4.5 Programming the config store partition

Note: for 2.0.x, before programming the config store, it is required to run an Upload, as in Section 4.3, to properly synchronise your system's environment. This is not necessary with MM-IoT-SDK 2.1.x and newer.

After editing the `config.hjson`, the updated configuration can be pushed to the device by executing the *Program configstore* custom task. This can be found by navigating to the PlatformIO icon in the VS Code activity bar, and selecting *Custom* → *Program configstore* for the desired platform.

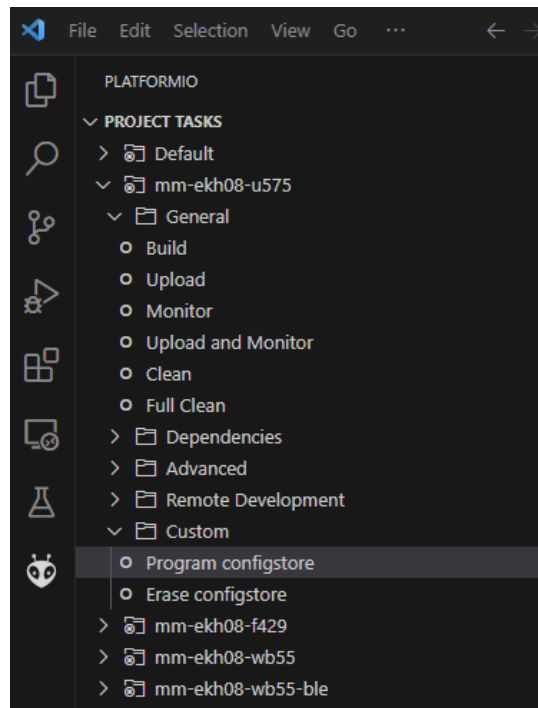


Figure 17. Navigation to Custom tasks

The config store is implicitly erased before programming. It can also be explicitly erased by selecting *Erase configstore*.

4.6 Using serial monitor

Open the serial monitor by pressing the power plug icon on the VS Code status bar (Figure 18), or by selecting the *Monitor* task from the list of *Project Tasks*. If successful, a long-lived Monitor task will open in a terminal in the VS Code panel. Press the reset button on the evaluation board to restart the application and program output will be displayed in the terminal window.

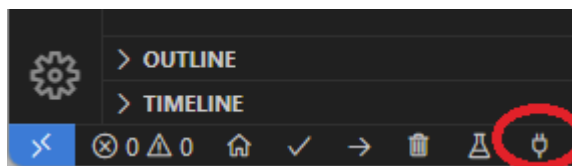


Figure 18. Serial monitor button

Note that monitor tasks are long-lived. If the monitor task was previously launched the following dialog will pop up:

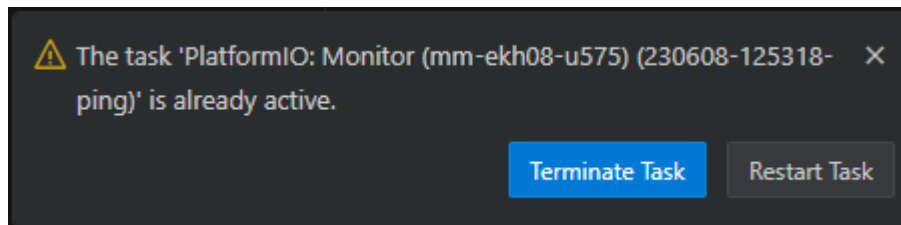


Figure 19. Duplicate monitoring

As mentioned in Section 4.2, you may want to explicitly select the relevant serial port for your evaluation board. This is particularly important if you have multiple serial ports attached to the host system.

1. Click on “Auto” in the status bar (Figure 20)
2. This will open a command palette at the top of VS Code containing a list of detected serial ports (Figure 21). Choose the one required for your evaluation board.

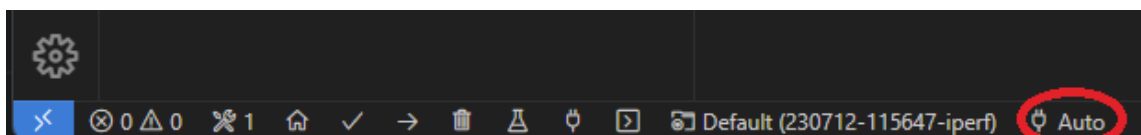


Figure 20. Serial Monitor selection

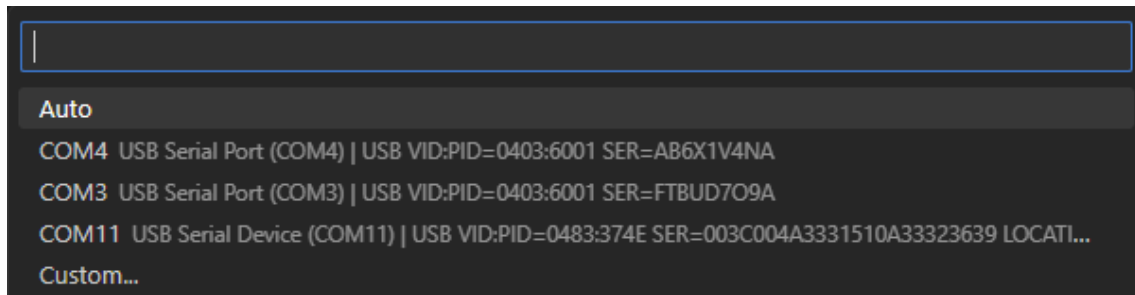


Figure 21. Command Palette showing detected serial ports

5. Creating a custom project

In this section is explained how to create a new custom project in PlatformIO using the MM-IoT-SDK.

5.1 Creating a new project

A new project can be created using the *Create New Project* dialog shown in the [Opening an example project](#) section.

1. Click the *New Project* icon. Project wizard will pop up (Figure 22).
2. Type *morse* to the board selection box and you will see the available Morse Micro development kits.
3. Select the board that you are using. The MM-IoT-SDK is selected automatically as the framework.
4. Click *Finish* and the new project is created. The src folder of the new project will be empty and you will have to write the application yourself.

In order for the serial monitor to work, you must specify the baud rate in the platformio.ini file, you can use the platformio.ini file of one of the example applications as reference.

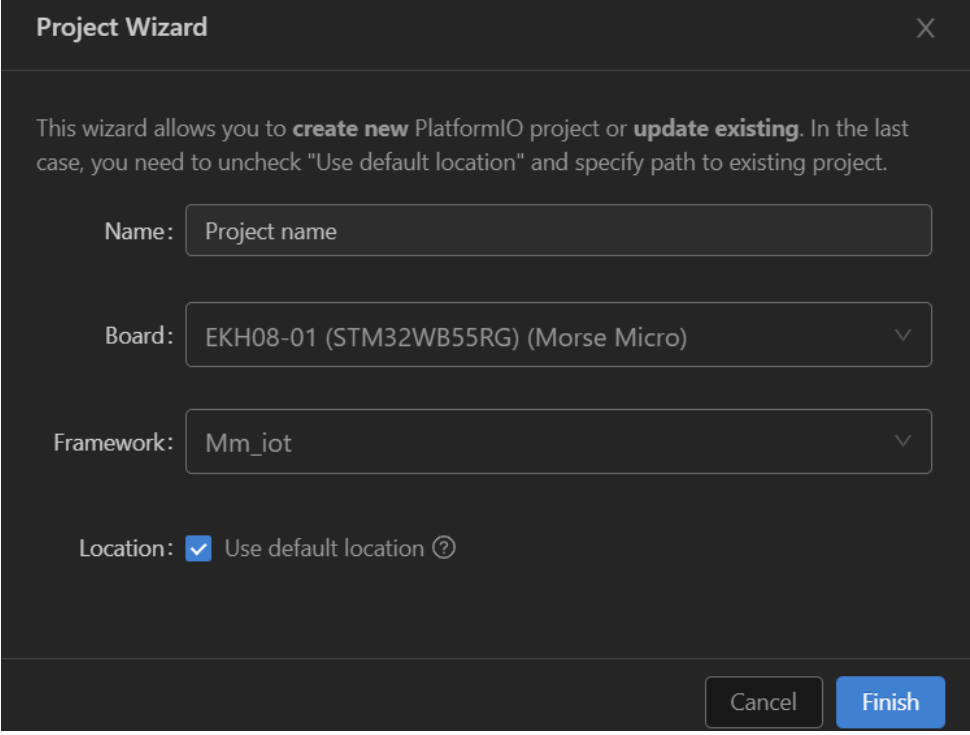
The image shows a 'Project Wizard' dialog box with a dark background. At the top, the title 'Project Wizard' is on the left and a close button 'X' is on the right. Below the title, a paragraph of text reads: 'This wizard allows you to **create new** PlatformIO project or **update existing**. In the last case, you need to uncheck "Use default location" and specify path to existing project.' There are three input fields: 'Name:' with a text box containing 'Project name'; 'Board:' with a dropdown menu showing 'EKH08-01 (STM32WB55RG) (Morse Micro)'; and 'Framework:' with a dropdown menu showing 'Mm_iot'. Below these is a 'Location:' section with a checked checkbox and the text 'Use default location' followed by a help icon '?'. At the bottom right, there are two buttons: 'Cancel' and 'Finish'.

Figure 22. Creating a new project

5.2 Returning to PlatformIO Home

Once you've opened a project, it may not be obvious how to navigate back to the PlatformIO home screen to, for example, start a new project. To return to the home screen:

1. Click on the PlatformIO icon in the activity bar (Figure 23)

2. Under the quick access menu, click on *Open* under the *PIO Home* section.

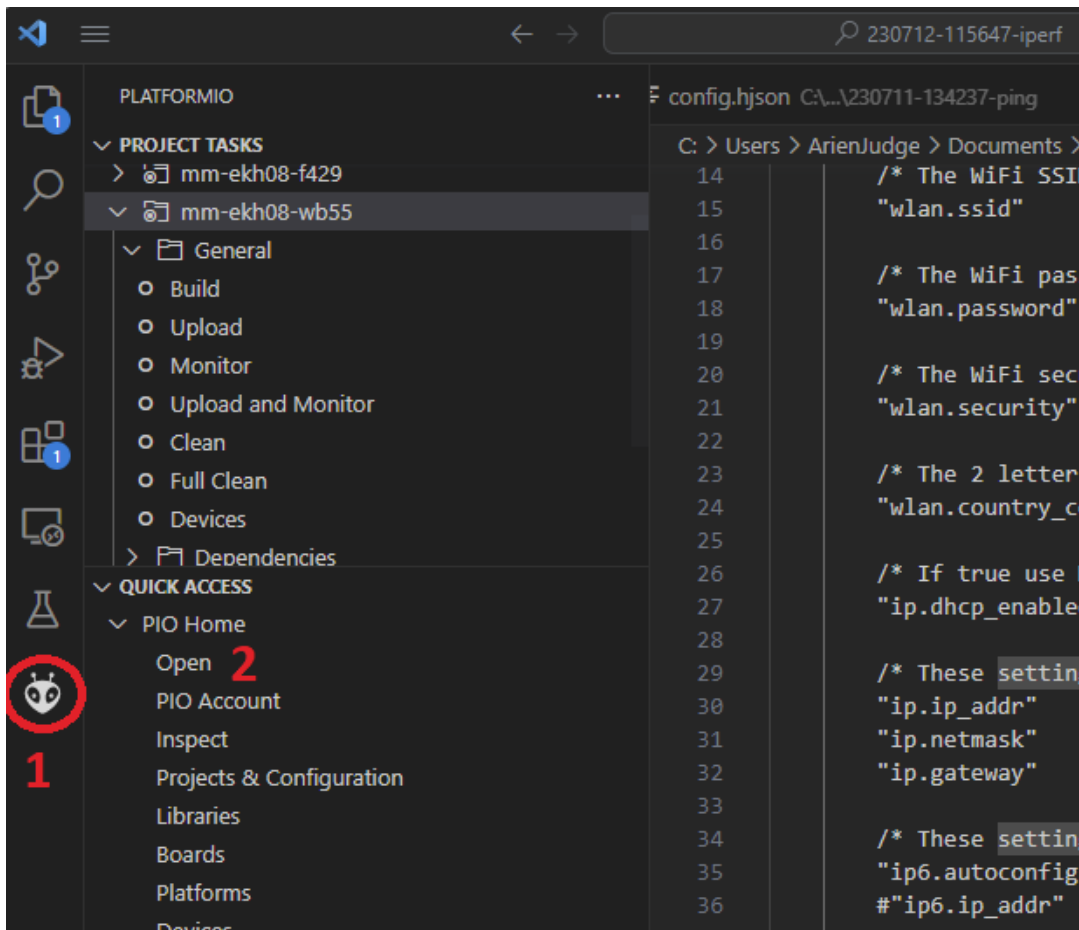


Figure 23. PlatformIO quick access

3. This should now display the home screen of PlatformIO where you can now start a new project or example (Figure 24).

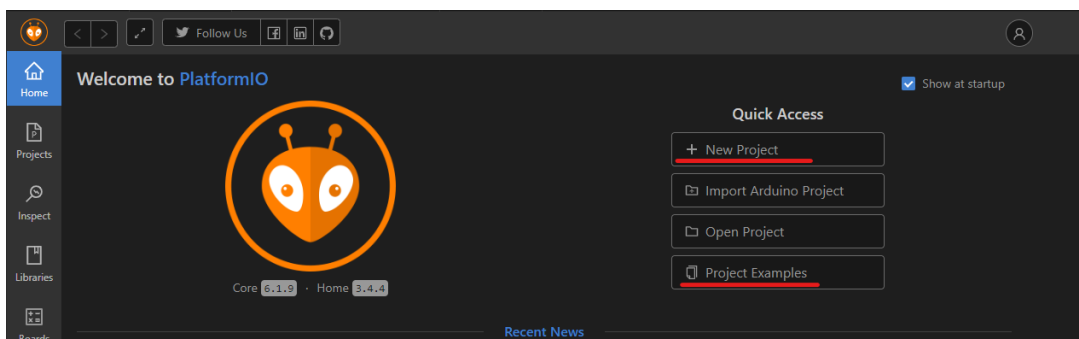


Figure 24. PlatformIO Home

6. Advanced features

In the following section more advanced features of PlatformIO are explained.

6.1 Debugging

Note: the debug feature is not supported by MM-IoT-SDK2.0.x. Please update to MM-IoT-SDK 2.1.x or newer.

The Debug view can be opened by clicking the *Run and Debug* button on the toolbar on the left side (Figure 25). Click on the green play button on the top to initiate a debug session. The project will be built in debug configuration and uploaded on the board automatically. Once this completes the debug session is started and the program is halted at the entry point.

It is important that the correct environment is chosen to match the board under evaluation so that PlatformIO can configure the debug session correctly. In the status bar, at the bottom of Visual Studio Code ensure your Project Environment is selected. *Default* refers to the first environment in the platformio.ini, which is typically *mm-ekh08-u575*.

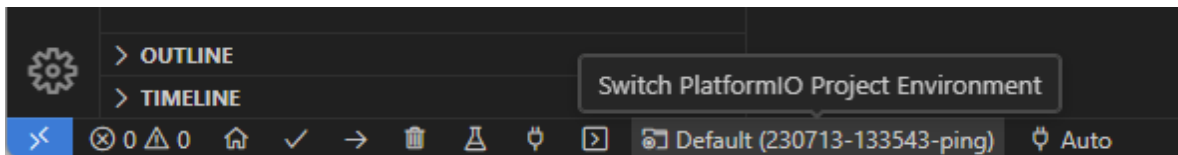


Figure 25. PlatformIO project environment selection

You can find more information about the debugging features on the following links:

<https://docs.platformio.org/en/stable/integration/ide/vscode.html#debugging>

<https://code.visualstudio.com/docs/editor/debugging>

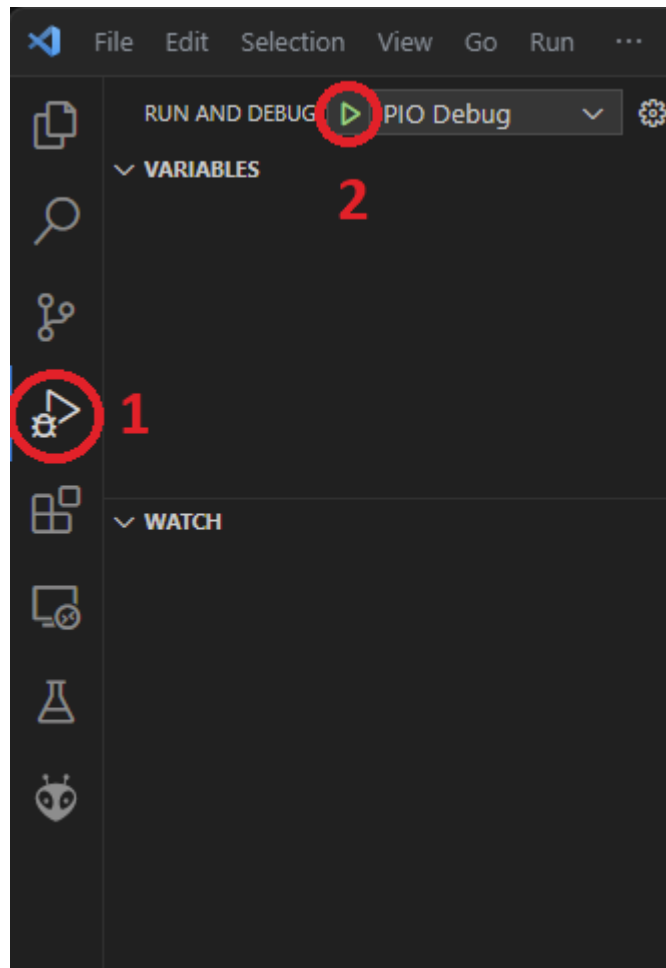


Figure 26. Run and Debug

6.2 Inspection tool

The PlatformIO inspection tool provides more insight into the composition of the program and its resource usage. It provides static code and memory usage analysis of the application. The inspection tool can be started by clicking *Inspect* in the Quick access menu (Figure 27). On the Project Inspection dialogue, select the project and the environment that you want to inspect. Click on the Inspect button and wait for the process to complete.

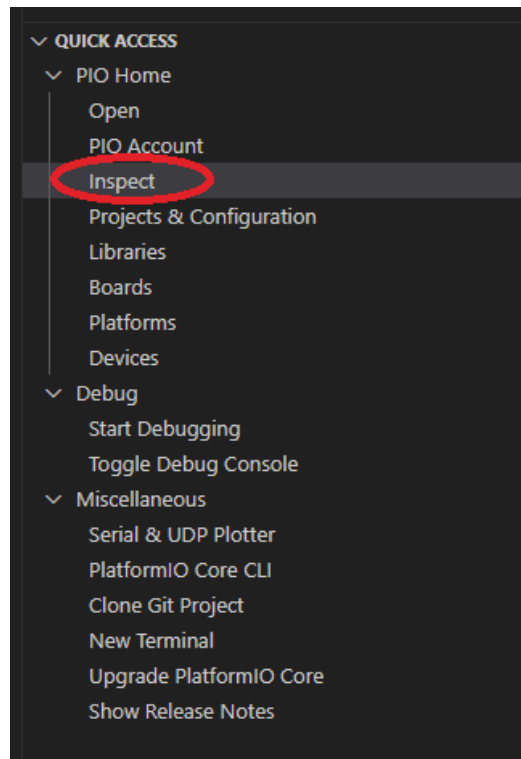


Figure 27. Quick access menu

After the tool completes the analysis, you can see the overview page (Figure 28). On this you will see the summary of the memory usage (RAM and Flash) and any defects that are found during the static code analysis.

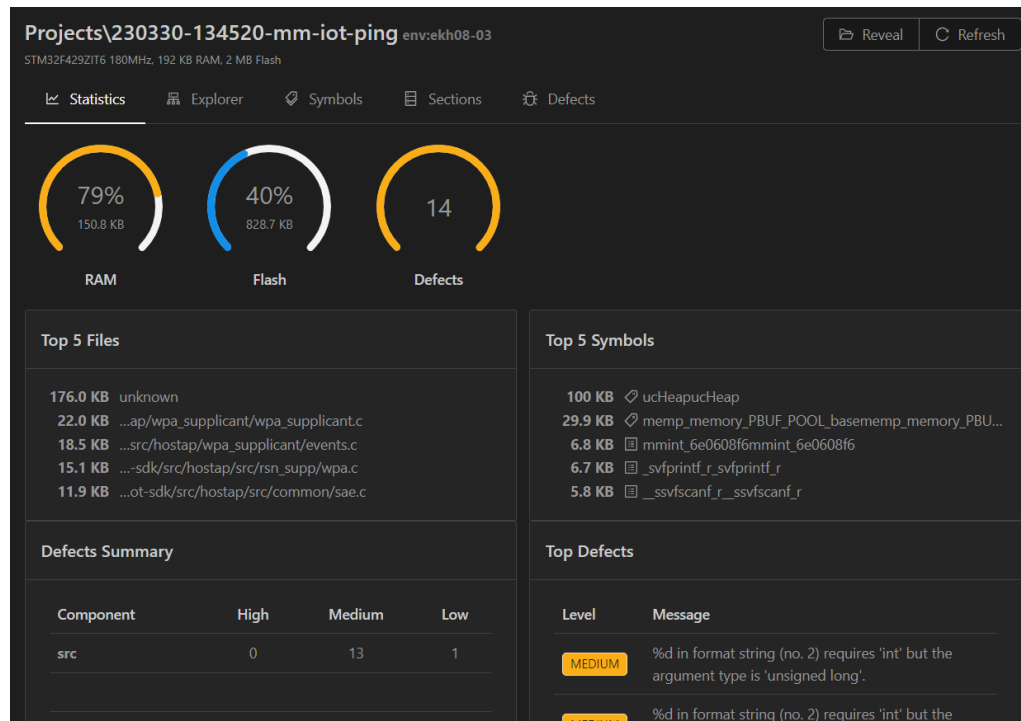
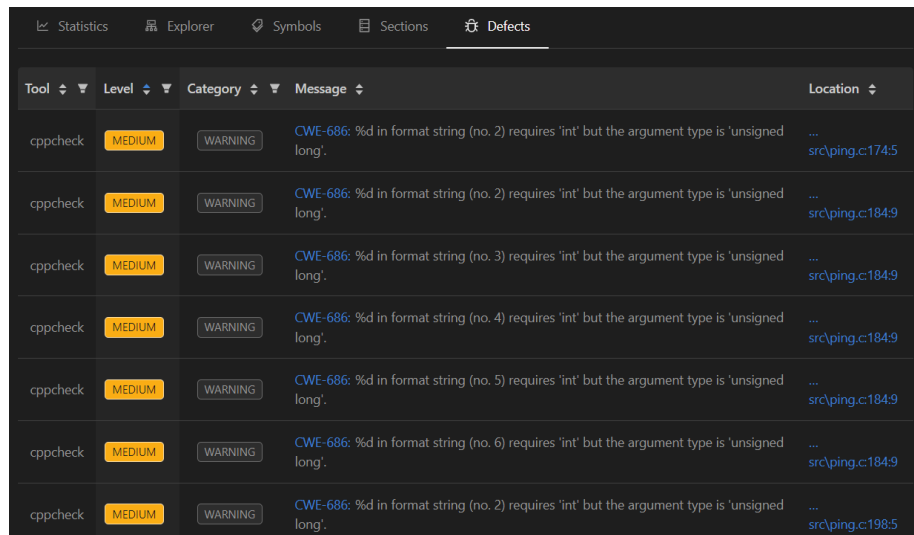


Figure 28. Overview page of the inspection tool

6.1.1 Static code analysis

The static code analysis is by default performed by a tool called Cppcheck. By default the tool is run with `--enable=all` flags and runs all the available checks. The result of the analysis can be seen on the *Defects* page (Figure 29). The location of the defect in the source code is provided with a link on the *Location* column which can be clicked to view the defect. For more information on static code analysis in PlatformIO, refer to:

<https://docs.platformio.org/en/stable/advanced/static-code-analysis/index.html>

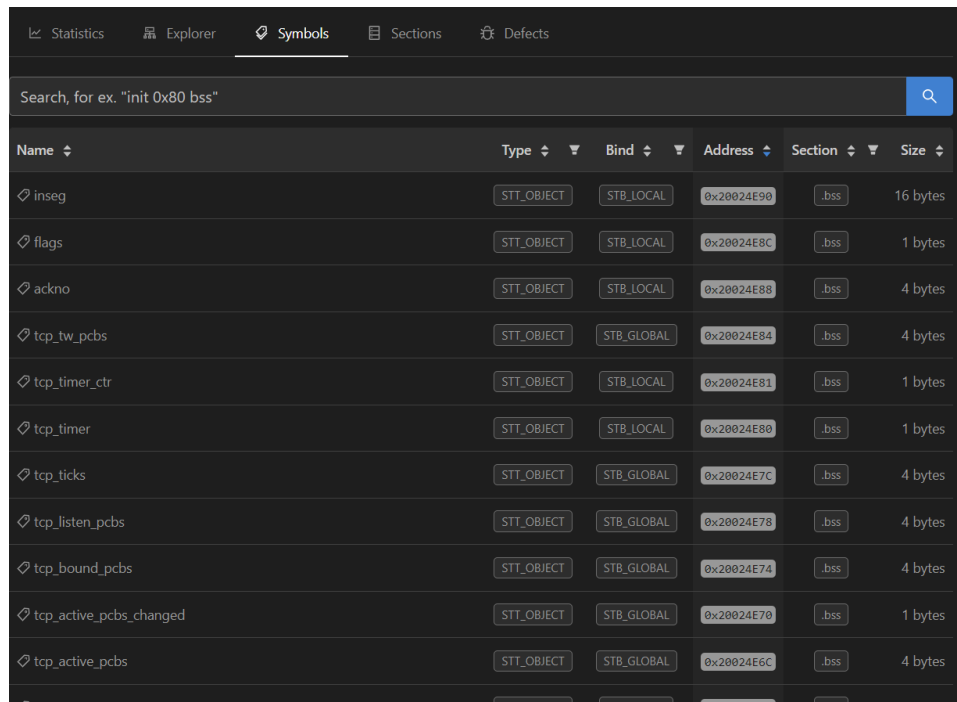


Tool	Level	Category	Message	Location
cppcheck	MEDIUM	WARNING	CWE-686: %d in format string (no. 2) requires 'int' but the argument type is 'unsigned long'.	... src\ping.c:174:5
cppcheck	MEDIUM	WARNING	CWE-686: %d in format string (no. 2) requires 'int' but the argument type is 'unsigned long'.	... src\ping.c:184:9
cppcheck	MEDIUM	WARNING	CWE-686: %d in format string (no. 3) requires 'int' but the argument type is 'unsigned long'.	... src\ping.c:184:9
cppcheck	MEDIUM	WARNING	CWE-686: %d in format string (no. 4) requires 'int' but the argument type is 'unsigned long'.	... src\ping.c:184:9
cppcheck	MEDIUM	WARNING	CWE-686: %d in format string (no. 5) requires 'int' but the argument type is 'unsigned long'.	... src\ping.c:184:9
cppcheck	MEDIUM	WARNING	CWE-686: %d in format string (no. 6) requires 'int' but the argument type is 'unsigned long'.	... src\ping.c:184:9
cppcheck	MEDIUM	WARNING	CWE-686: %d in format string (no. 2) requires 'int' but the argument type is 'unsigned long'.	... src\ping.c:198:5

Figure 29. Defects page

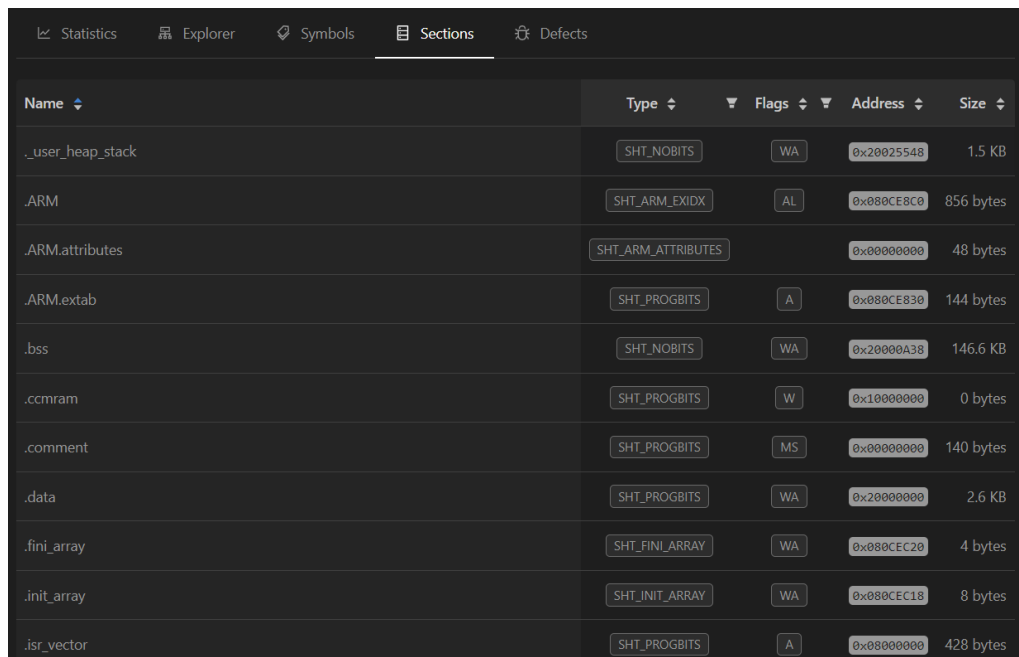
6.1.2 Memory analysis

The symbols view lists all the symbols of the program, the memory section on which they are located, the address, size, symbol type and binding attributes (Figure 30). Similar information could be obtained with a tool such as readelf of GNU binutils. This information can be useful for example in memory usage analysis or other advanced debugging scenarios. The *Sections* view lists all memory sections of the program (Figure 31). This information could be used when analysing memory consumption and its division between volatile and non-volatile memory. Similar information could be obtained with the readelf tool by running the dump section headers command.



Name	Type	Bind	Address	Section	Size
inseg	STT_OBJECT	STB_LOCAL	0x20024E90	.bss	16 bytes
flags	STT_OBJECT	STB_LOCAL	0x20024E8C	.bss	1 bytes
ackno	STT_OBJECT	STB_LOCAL	0x20024E88	.bss	4 bytes
tcp_tw_pcb	STT_OBJECT	STB_GLOBAL	0x20024E84	.bss	4 bytes
tcp_timer_ctr	STT_OBJECT	STB_LOCAL	0x20024E81	.bss	1 bytes
tcp_timer	STT_OBJECT	STB_LOCAL	0x20024E80	.bss	1 bytes
tcp_ticks	STT_OBJECT	STB_GLOBAL	0x20024E7C	.bss	4 bytes
tcp_listen_pcb	STT_OBJECT	STB_GLOBAL	0x20024E78	.bss	4 bytes
tcp_bound_pcb	STT_OBJECT	STB_GLOBAL	0x20024E74	.bss	4 bytes
tcp_active_pcb_changed	STT_OBJECT	STB_GLOBAL	0x20024E70	.bss	1 bytes
tcp_active_pcb	STT_OBJECT	STB_GLOBAL	0x20024E6C	.bss	4 bytes

Figure 30. Symbols view



Name	Type	Flags	Address	Size
_user_heap_stack	SHT_NOBITS	WA	0x20025548	1.5 KB
.ARM	SHT_ARM_EXIDX	AL	0x080CE8C0	856 bytes
.ARM.attributes	SHT_ARM_ATTRIBUTES		0x00000000	48 bytes
.ARM.extab	SHT_PROGBITS	A	0x080CE830	144 bytes
.bss	SHT_NOBITS	WA	0x20000A38	146.6 KB
.ccmram	SHT_PROGBITS	W	0x10000000	0 bytes
.comment	SHT_PROGBITS	MS	0x00000000	140 bytes
.data	SHT_PROGBITS	WA	0x20000000	2.6 KB
.fini_array	SHT_FINI_ARRAY	WA	0x080CEC20	4 bytes
.init_array	SHT_INIT_ARRAY	WA	0x080CEC18	8 bytes
.isr_vector	SHT_PROGBITS	A	0x08000000	428 bytes

Figure 31. Sections view

7. Troubleshooting

7.1 I can't identify my serial port!

For Windows this can be checked in the device manager under *Ports (COM & LPT)*, to identify the correct COM port. For Mac & Linux the available USB/serial ports are usually listed in */dev/*, e.g. */dev/ttyUSB0*.

A good technique to identify the right serial port is to list the attached ports before, then after attaching the COM port, and compare the difference. If no new port is identified then it is highly likely there is a hardware issue with the connection to the PC.

7.2 Upload failed

Upload failures can happen for a number of reasons. The most common is simply that PlatformIO's instance of OpenOCD can't find your uploader.

For the Morse Micro evaluation kits, the uploader is the STLINK debug probe built into the STM32 Nucleo evaluation boards. PlatformIO's instance of OpenOCD may not be able to determine which is the correct debug probe if you have multiple attached, so ensure only 1 debug probe is attached at a time.

In some cases, your host PC may put the USB port hosting the STLINK to sleep as part of its power save settings. An easy way to wake up the offending port is to select the serial port associated with the board. As these share a port, activity on this serial port will wake up the STLINK as well.

7.3 Error: JTAG status contains invalid mode value

Because the Nucleo-WB55 is a recent evaluation kit release from STMicroelectronics, OpenOCD support is still under development. You may see the following error in the Upload or Program configstore output, but the programming continues.

```
Error: jtag status contains invalid mode value - communication failure  
Polling target stm32wbx.cpu failed, trying to reexamine
```

This is normal and can be ignored for now.

7.4. Debug session doesn't launch

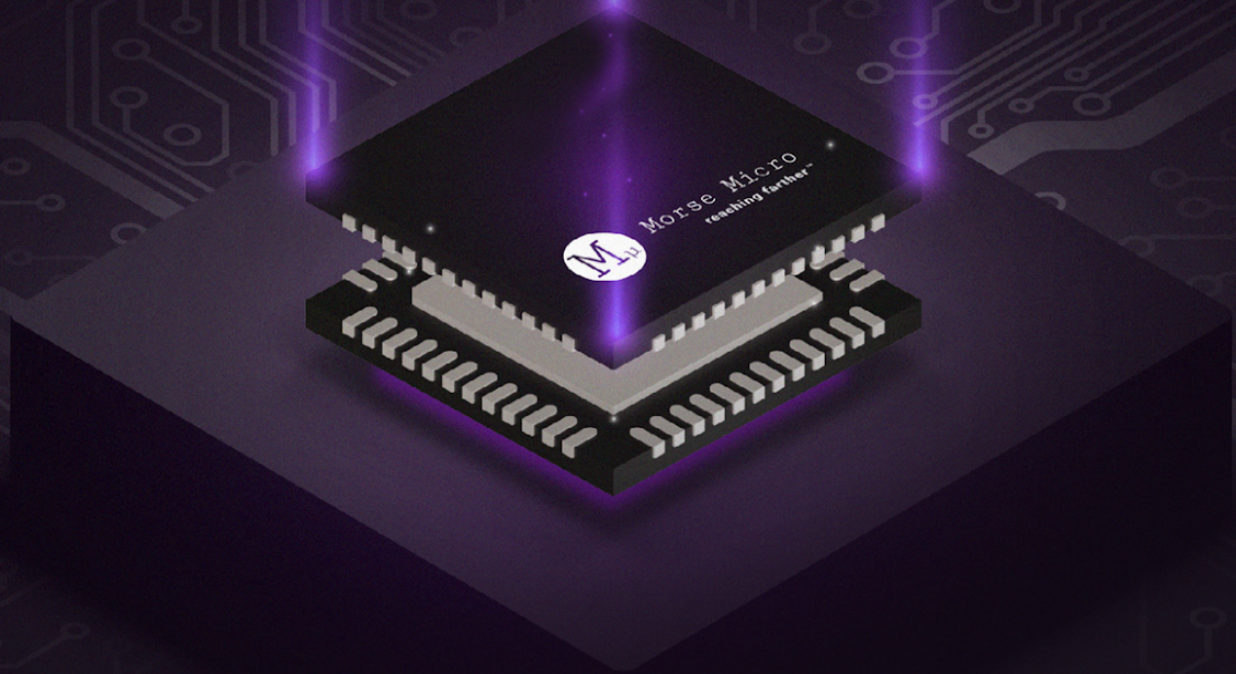
After pressing the Run and Debug the Terminal output for the Pre-Debug task shows *Status: SUCCESS*, but nothing happens!

For the 2.0.x release, there is a known bug with the way OpenOCD is configured for debugging. This is fixed in 2.1.x onwards.

External debugging tools are recommended for 2.0.x. The MM-IoT-SDK installs OpenOCD into the `.platformio/packages/tool-openocd` directory of the user's home directory, and can be used externally to run a gdb server to connect gdb to the device.

8. Revision History

Release Number	Release Date	Release Notes
01	02/11/2023	Initial release



Morse Micro
reaching farther™