Visual Studio Code for ModusToolbox™ user guide

About this document

Scope and purpose

This document provides information and instructions for using Visual Studio Code (VS Code) with ModusToolbox™ software.

ModusToolbox™ software is a set of tools and libraries that support device configuration and application development. These tools enable you to integrate our devices into your existing development methodology.

Document conventions

Convention	Explanation
Bold	Emphasizes heading levels, column headings, menus and sub-menus.
Italics	Denotes file names and paths.
`Monospace`	Denotes APIs, functions, interrupt handlers, events, data types, error handlers, file/folder names, directories, command line inputs, code snippets.
File > New	Indicates that a cascading sub-menu opens when you select a menu item.

Reference documents

Refer to the following documents for more information as needed:

- ModusToolbox™ software installation guide – Provides information and instructions about installing the tools package on Windows, Linux, and macOS.
- ModusToolbox™ tools package user guide – Provides information about all the tools included with ModusToolbox™ tools package.
- Debugging in Visual Studio Code
- GitHub - Marus /cortex-debug: Visual Studio Code extension for enhancing debug capabilities for Cortex-M Microcontrollers

Table of contents

About this document . . . 1

Table of contents . . . 2

1 Download/install software . . . 3

1.1 ModusToolbox™ software and tools . . . 4

1.2 J-Link . . . 4

2 Using the extension . . . 5

2.1 Main page . . . 5

2.2 Getting Started . . . 5

2.3 AI Assistant . . . 7

2.4 Application status and actions . . . 8

2.5 Device kits . . . 9

2.6 Recently opened . . . 9

2.7 Local content storage . . . 9

2.8 Settings . . . 9

2.9 Documentation and help . . . 10

2.10 Commands, menus, and shortcuts . . . 10

3 Not using the extension . . . 11

3.1 Export existing application . . . 11

3.2 Open workspace in VS Code . . . 11

3.3 Add/modify application code . . . 12

3.4 Command line . . . 13

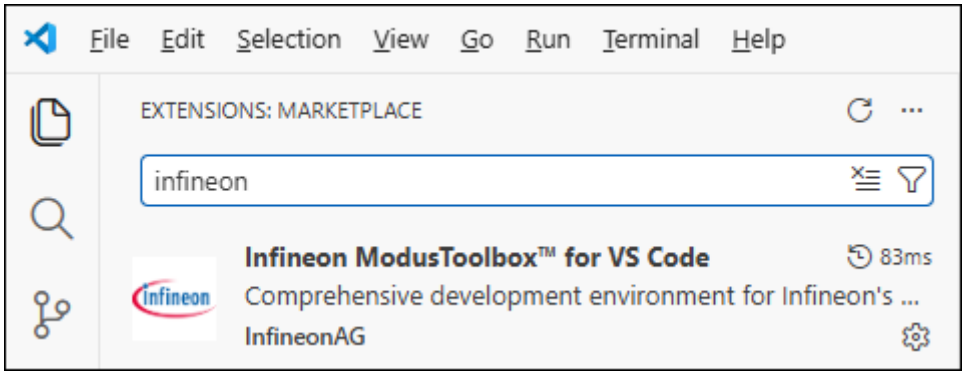
3.5 Build the Application . . . 14

3.6 Program/debug . . . 16

Disclaimer . . . 38

1 Download/install software

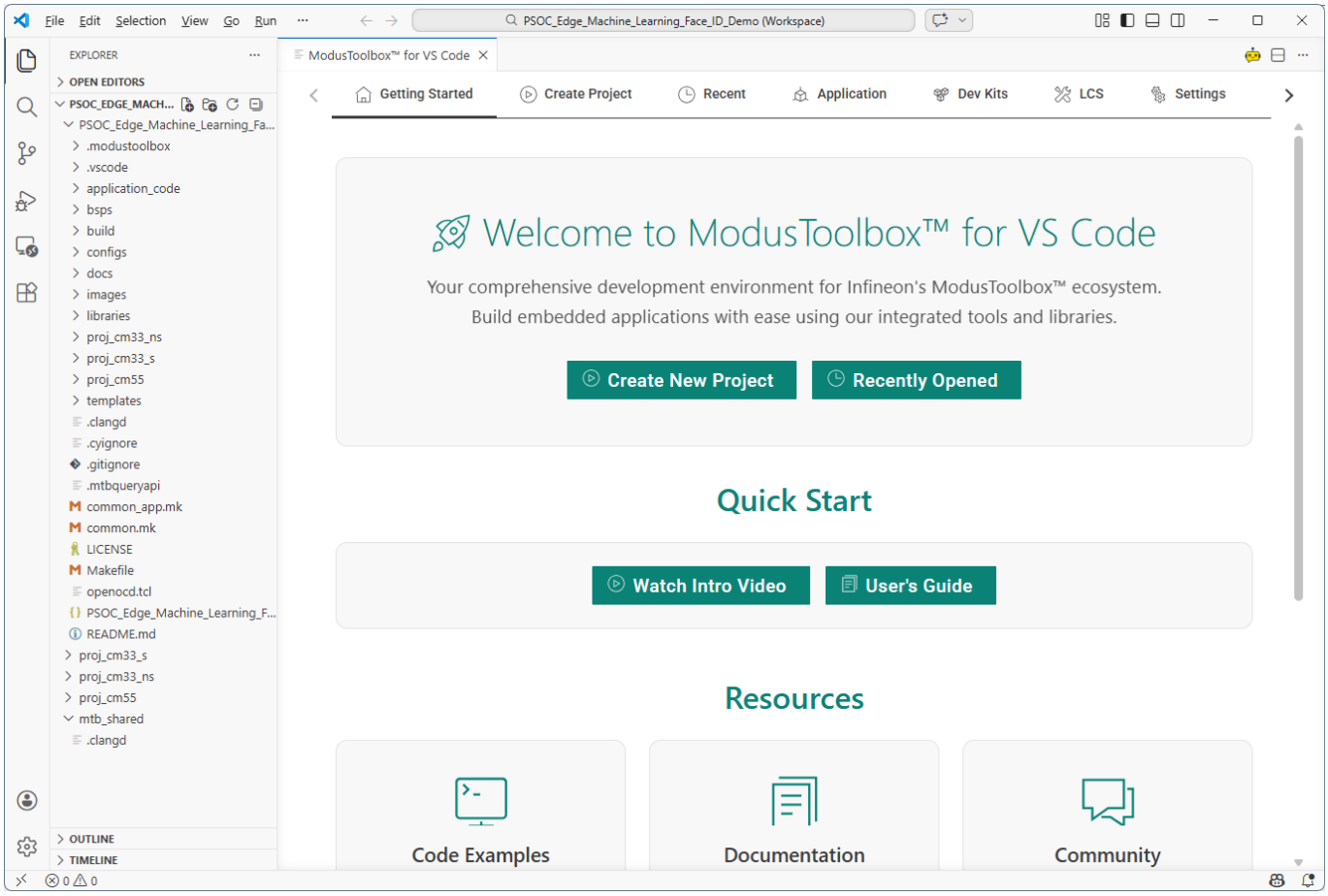
If you already have VS Code installed, then use the extensions marketplace to find and install the Infineon VS Code for ModusToolbox™ extension.



If you do not have VS Code already installed, you can download it from the website:

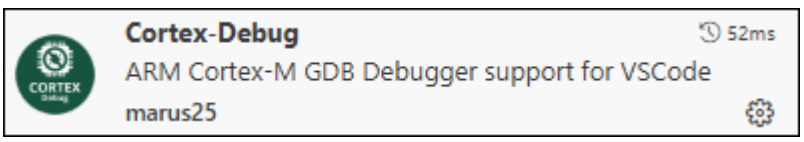
https://code.visualstudio.com/

Once the extension is installed, you can use it to create new projects and work of them in VS Code. See Getting Started for more details.



The extension will prompt to install other recommended extensions, including the following:

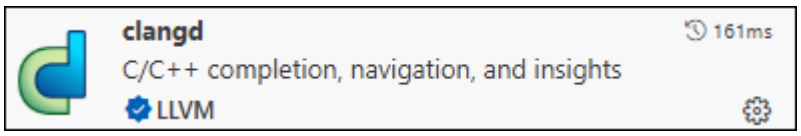
- Cortex-Debug



- Arm® Assembly



- clangd



1.1 ModusToolbox™ software and tools

If the ModusToolbox™ software is not yet installed when you open the extension, the extension displays a software installer so you can get everything you need without leaving VS Code.

- Choose where to install the software. You can use the recommended location in your home directory, or browse to a custom path. The installer reports any problems with a location, such as insufficient disk space or missing permissions.
- Review the required tools and select any optional tools you want to install. Tools that are already installed show whether an upgrade is available.
- Download and install the selected tools. A progress bar is shown for each tool, and you can continue to the main page when the installation completes.

The extension can also install the LLVM compiler. Run the **Infineon ModusToolbox™: Install the LLVM Compiler** command, choose an LLVM version and installation path, and the extension downloads and installs it. When you open a PSoC™ Edge application without LLVM installed, the extension offers to install it for you; you can disable this prompt in the settings.

1.2 J-Link

For J-Link debugging, download and install J-Link software:

https://www.segger.com/downloads/J-Link

2 Using the extension

This section describes how to work with ModusToolbox™ applications using the ModusToolbox™ Assistant extension for VS Code. The extension provides a graphical interface for creating projects, building, programming, debugging, and managing tools and content, without having to use the command line.

The easiest way to open various ModusToolbox™ tools with VS Code is by installing the ModusToolbox™ for VS Code extension, which provides access to tools, configurators, and documentation.

When you have a ModusToolbox™ application loaded in VS Code, you can use various tabs and commands to Build, Program, and Debug the application. You can also open additional tools such as Device Configurator to configure your application with device settings and firmware, as well as the Library Manager to add and remove firmware libraries.

- Main page
- Getting Started
- AI Assistant
- Application status and actions
- Device kits
- Recently opened
- Local content storage
- Settings
- Documentation and help
- Commands, menus, and shortcuts

2.1 Main page

To open the extension's main page, run the **Infineon ModusToolbox™: Show Main Page** command from the Command Palette, or click the ModusToolbox™ Assistant button in the editor title bar. By default, the main page appears automatically when a ModusToolbox™ application is loaded; you can change this behavior in the settings so that it always opens, or never opens automatically. The page uses a tabbed layout and follows the VS Code dark or light theme.

The main page provides the following tabs:

- **Getting Started** – orientation, quick actions, and a feature overview.
- **Create Project** – a guided wizard for creating new applications.
- **Recently Opened** – reopen applications you have used recently.
- **Application** – project health, build, program, debug, and memory usage for the loaded application.
- **Device Kits** – connected kits and BSP association.
- **Local Content Storage** – manage BSP and middleware content for offline use.
- **Settings** – extension and workspace preferences.
- **Glossary** – ModusToolbox™ terminology and definitions.
- **User Guide** – this built-in documentation.

2.2 Getting Started

This section covers how to get started building ModusToolbox™ applications using the ModusToolbox™ extension. The quickest way to start is to create a new application directly in VS Code with the extension's Create New Project wizard:

- Create new application

If you already have a ModusToolbox™ application, you can open it in VS Code.

- If it already contains a VS Code workspace, then a message will display to ask if you want to open the workspace.
- If it is not a VS Code application, a message will display asking to run `make vscode` and create the necessary files.

Once an application is open, you can also add AI capabilities to it using the **Add AI Enhancements** action on the **Application** tab to generate GitHub Copilot support files for the project.

2.2.1 Create new application

You can create a new ModusToolbox™ application directly inside VS Code using the **Create Project** tab provided by the ModusToolbox™ Assistant extension. If the ModusToolbox™ software is not yet installed, the extension first prompts you to install it.

Open the extension's main page by running the **Infineon ModusToolbox™: Show Main Page** command from the Command Palette, or by clicking the ModusToolbox™ Assistant button in the editor title bar. On the **Getting Started** tab, click **Create New Project** (or select the **Create Project** tab).

Choosing how to create your project

The **Create Project** tab offers two paths:

- **Guided Project Creation** – the guided three-step wizard described below. This path is best when you already know the board and code example you want to start from.
- **AI-Assisted Creation** (Preview) – describe what you want to build and let the AI Assistant plan and scaffold the project for you. This path is best when you are starting from a requirement or an idea rather than a specific board or example. It becomes available only when the AI Assistant is ready; see AI Assistant.

Guided Project Creation

Choose **Guided Project Creation**. The wizard guides you through the following steps:

1. **Project Information** – Enter a **Project Name** (letters, numbers, hyphens, and underscores only; spaces are not allowed) and a **Project Location**. Use the **Browse** button to select a folder. A preview shows the full path where the project will be created. The name and location fields become available after the manifest data finishes loading. Click **Next: Select BSP**.
2. **Board Support Package** – Choose a **BSP Category** and then a specific **Board Support Package**. Any connected development kits are listed at the top of this step, showing the kit name, BSP, status, and serial number; select an available kit to target it. The panel on the right displays the selected board's details, including device, connectivity, and a summary. Use the refresh button to re-scan for connected kits. Click **Next: Select Example**.
3. **Code Example** – Choose an **Example Category** and a specific **Example Code** to use as the starting point for your application. The panel on the right shows a description of the selected example. Click **Review & Create**.
4. **Review & Create** – Review the **Project Summary**, which lists the project name, location, full path, BSP category, board, device, connectivity, and selected code example. Click **Create Project** to begin. A progress bar reports the creation status, including the progress of any assets that are downloaded. Do not close the editor until the process completes.

When the project has been created, a **Project Creation Complete!** confirmation appears and shows the location of the new project. Click **Load Project** to open the new application in VS Code.

AI-assisted project creation

Choose **AI-Assisted Creation** to start from an idea instead of a board and example. This option requires the AI Assistant to be ready; see AI Assistant for the prerequisites and how to satisfy them.

1. Select **AI-Assisted Creation**. The extension prepares a dedicated planning workspace and reopens VS Code in it, then opens the **Create Project** tab with the **Create Your Project with Copilot** panel.

2. Describe what you want to build. You can start from one of the suggested example prompts by clicking **Try it**, or click **Open Copilot Chat** to describe your own idea.
3. Copilot uses the ModusToolbox™ support files in your workspace — including any custom instructions, agents, and skills — to plan and scaffold the project on your behalf. Review the plan and follow Copilot's guidance to complete project creation.

***Note**: For best results, use Claude Opus or a comparable high-capability reasoning model in Copilot Chat.*

***Note**: Only Edge devices are currently supported and verified for the AI-assisted creation flow.*

A standalone Project Creator tool is also available for use outside of VS Code. For more details, refer to the Project Creator user guide.

2.3 AI Assistant

The ModusToolbox™ Assistant extension integrates with the Infineon ModusToolbox™ AI Assistant to provide AI-assisted project creation and AI enhancements for your applications. These features are powered by GitHub Copilot Chat, so both the Infineon ModusToolbox™ AI Assistant and GitHub Copilot Chat extensions must be installed and ready before the AI features become available.

Installation, discovery, and dependency detection

The AI features rely on two companion extensions and a supporting runtime:

- The **Infineon ModusToolbox™ AI Assistant** extension.
- The **GitHub Copilot Chat** extension.
- Python 3.9 or later, which the AI Assistant uses at runtime.

When you open a ModusToolbox™ application and the AI Assistant is not yet installed, the extension shows a notification offering to install it. Choose **Install** to add it, **Later** to dismiss the prompt for the current session, or **Don't ask again** to stop future prompts.

The extension automatically discovers the companion extensions and detects the supporting dependencies. Discovery re-runs whenever an extension is installed or updated and whenever the VS Code window regains focus, so newly satisfied prerequisites are picked up without a manual refresh. In some cases you are prompted to reload the window so the change takes effect.

AI Assistant readiness

Before the AI features are enabled, every prerequisite must be satisfied. The extension continuously evaluates readiness and surfaces exactly what is missing so you are never left guessing. On the **Create Project** tab, the **AI-Assisted Creation** card stays greyed out until the AI Assistant is ready, and it lists every outstanding prerequisite with a one-click action to resolve it:

- AI Assistant not installed – **Install AI Assistant**.
- GitHub Copilot Chat not installed – **Install Copilot Chat**.
- Python 3.9 or later not found – **Install Python**.
- AI Assistant out of date – **Update AI Assistant**.

After you resolve the listed items, use the **Reload** button on the card to reload the window so the extension can re-evaluate readiness. The **Add AI Enhancements** action on the **Application** tab is likewise disabled while the AI Assistant is not ready, with a tooltip that explains which prerequisite is missing.

Version compatibility validation

The extension validates the installed AI Assistant version against the range of versions that this build of ModusToolbox™ for VS Code supports:

- If the installed AI Assistant is older than the supported range, the extension prompts you to update it (**Update AI Assistant**).
- If the installed AI Assistant is newer than the supported range, the extension prompts you to update ModusToolbox™ for VS Code to a build that supports it.

This validation keeps the two extensions compatible and prevents AI features from running against an unverified combination.

Once the AI Assistant is ready, you can create projects with AI assistance (see Create new application) and add AI enhancements to an existing application (see Application status and actions).

2.4 Application status and actions

When a ModusToolbox™ application is open, the **Application** tab gives a complete view of the application and provides one-click actions to build, program, debug, and maintain it.

What you see

- The application name and directory, along with each project in the application.
- Missing-asset indicators that flag content the application needs in order to build.
- The tools and middleware libraries detected for each project, including any available upgrades.
- Documentation links for each project and for the application as a whole.
- The current build configuration and a snapshot of device memory usage (see Memory usage view).
- A **Plan** tab that displays the generated application plan when the AI Assistant is available and the application includes a plan file (`mtb-application-plan.md`).

Actions

- Prepare the application for VS Code by creating the required files if they are missing, and update the VS Code tasks and settings as needed.
- Open the project folder or the project README.
- Run build actions: **Build**, **Rebuild**, and **Clean**.
- Program the device using **Program** or **Quick Program**.
- Erase the device using **Erase** (internal memory) or **Erase All** (internal and external memory).
- Launch the **Device Configurator** or the **Library Manager**.
- Run curated Quick Tasks, and refresh the application to rescan the environment.
- Add AI enhancements to the application with **Add AI Enhancements**, which generates GitHub Copilot support files (instructions, agents, and skills) for the project so Copilot can work more effectively with it. This action is available when the AI Assistant is ready; otherwise it is disabled with a tooltip that explains which prerequisite is missing (see AI Assistant).

***Note**: For the build, program, and erase buttons, left-click to run the current choice. Right-click to open a menu and switch modes, for example **Program** versus **Quick Program**, or **Erase** versus **Erase All**. If assets are missing, a Missing Assets banner provides **Fix** actions to restore the files required for a successful build.*

2.4.1 Memory usage view

The memory usage view shows how your application uses device memory in a three-level hierarchy: physical memory, then regions, then segments.

- The top summary shows overall physical memory usage on the device, such as SRAM, flash, and any external memory.

- Each region shows its own usage.
- Segments show names, start addresses, and sizes so you can see how the application is laid out in memory.

By default the top-level sections are expanded and the regions are collapsed. Use **Expand All** and **Collapse All** to change how much detail is shown.

2.5 Device kits

The **Device Kits** tab lists the Infineon development kits connected to your computer and lets you associate each kit with a Board Support Package (BSP).

- For each kit, see the kit name, serial number, and firmware version.
- View board features, the programmer (KitProg) type, and the USB mode.
- See the current BSP and the BSPs available for the kit, along with a status indicator.

From this tab you can update a kit's firmware when it is out of date, change the BSP associated with a kit, and refresh the list to re-scan for connected kits.

***Note**: Connected kits also appear in the Create New Project wizard, so you can target a kit directly when you create an application.*

2.6 Recently opened

The **Recently Opened** tab lists the ModusToolbox™ applications you have opened recently, grouped by time, for example Today, Yesterday, This Week, and Older. Each entry shows the application name, its BSP, and when it was last opened. Click an entry to reopen that application.

2.7 Local content storage

Local Content Storage lets you keep BSP and middleware content on your computer for offline development. You select content by BSP, and all content associated with a BSP is included.

- Two lists are shown: **Not In Local Storage** and **In Local Storage**. Drag and drop BSPs between the lists to add or remove them.
- Use the bulk actions **Move All To Local**, **Remove All From Local**, and **Revert**.
- Use **Check For Updates** and **Update Existing Content** to refresh the content you have stored.

***Note**: The extension computes the in/not-in lists against the active manifest and indicates when updates are needed. In Local Content operating mode, the manifest is read from your local content storage path, which you set in Settings.*

2.8 Settings

The **Settings** tab provides a guided editor for the extension and workspace settings, with validation and inline tips.

- Settings are grouped by ownership: ModusToolbox™, the extension, or the workspace.
- Supported value types include text, number, checkbox, choice, URL, directory path, file path, and tools path.
- Choice settings show a short tip for each option, and some settings are enabled only when relevant, for example the proxy server field is disabled when the proxy mode is direct.
- Quick-fixes help you correct invalid paths and mismatched configuration.

Common settings include the manifest database URL mode (default, china, or custom), Local Content mode and the local content storage path for offline use, the toolchain and tools version (including a custom tools path), and the information or logging level.

***Note**: Use the `modusToolbox.autodisplay` setting to choose when the main page opens automatically on startup: always, only when a ModusToolbox™ application is loaded, or never.*

2.9 Documentation and help

The extension gives you quick access to documentation and help from inside VS Code.

- Open the Getting Started and Documentation links in your browser.
- Browse ready-to-use code examples on GitHub.
- Visit the ModusToolbox™ community forums.
- Open the built-in user guide (this document) on the **User Guide** tab.
- Look up embedded-development terms on the **Glossary** tab.

In a C source file, right-click and choose **Display documentation for symbol**, or press **Ctrl+Shift+F1**, to open the API documentation for the symbol under the cursor.

2.10 Commands, menus, and shortcuts

The extension contributes the following commands, available from the Command Palette:

- **Infineon ModusToolbox™: Show Main Page** – open the extension's main page.
- **Infineon ModusToolbox™: Display documentation for symbol** – show API documentation for the symbol under the cursor in a C file.
- **Infineon ModusToolbox™: Install the LLVM Compiler** – install the LLVM toolchain.
- **Infineon ModusToolbox™: Add Bootloader To PSOC Edge Application** – add bootloader support to a PSoC™ Edge application.
- **Infineon ModusToolbox™: Show Bootloader Documentation** – open the bootloader documentation.
- **Infineon ModusToolbox™: Set Output Channel Level** – set the verbosity of the extension's output channel.

In a C source file, you can also right-click and choose **Display documentation for symbol** from the context menu, or press **Ctrl+Shift+F1**.

The extension provides a **ModusToolbox™ Shell** terminal profile for running environment-aware ModusToolbox™ commands in the integrated terminal.

3 Not using the extension

This section describes how to work with ModusToolbox™ applications without the extension's graphical interface, using the ModusToolbox™ command line tools and standard VS Code features such as tasks and launch configurations. These workflows also apply when you prefer to drive ModusToolbox™ manually or to script your build, program, and debug steps.

- Export existing application
- Open workspace in VS Code
- Add/modify application code
- Command line
- Build the Application
- Program/debug

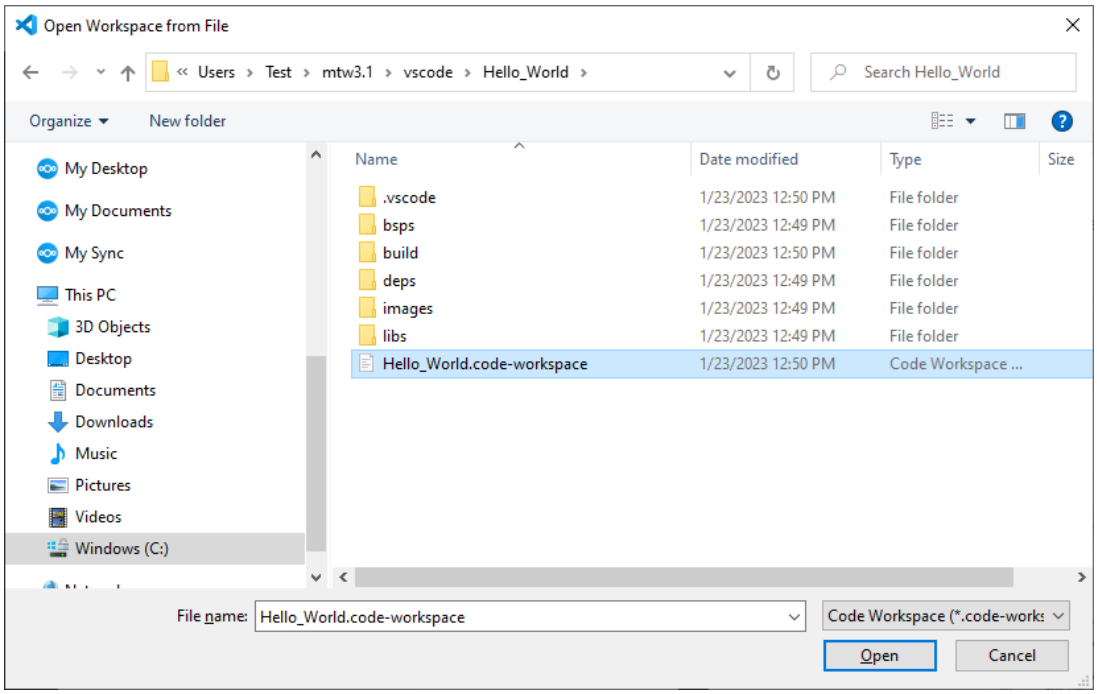
3.1 Export existing application

If you have a ModusToolbox™ application that was created for another IDE or the command line, you can export that application to be used in VS Code. Use the Open Folder command and point to the application directory containing the application *Makefile*.

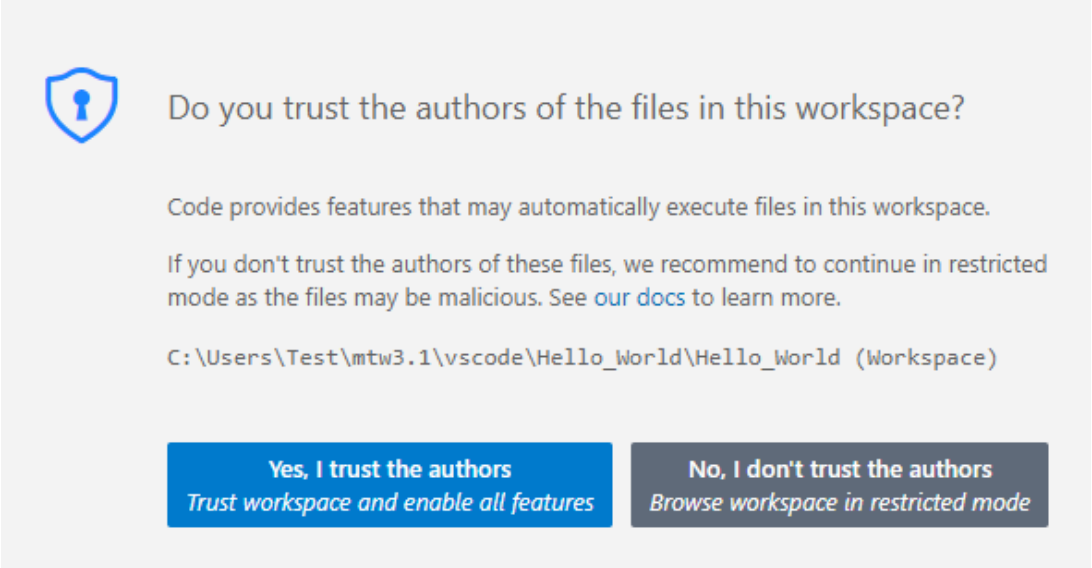
Open a ModusToolbox™ Shell terminal in the application directory and run the command `make vscode`. This will add VS Code specific files, including the workspace file, that will make the application compatible with VS Code.

3.2 Open workspace in VS Code

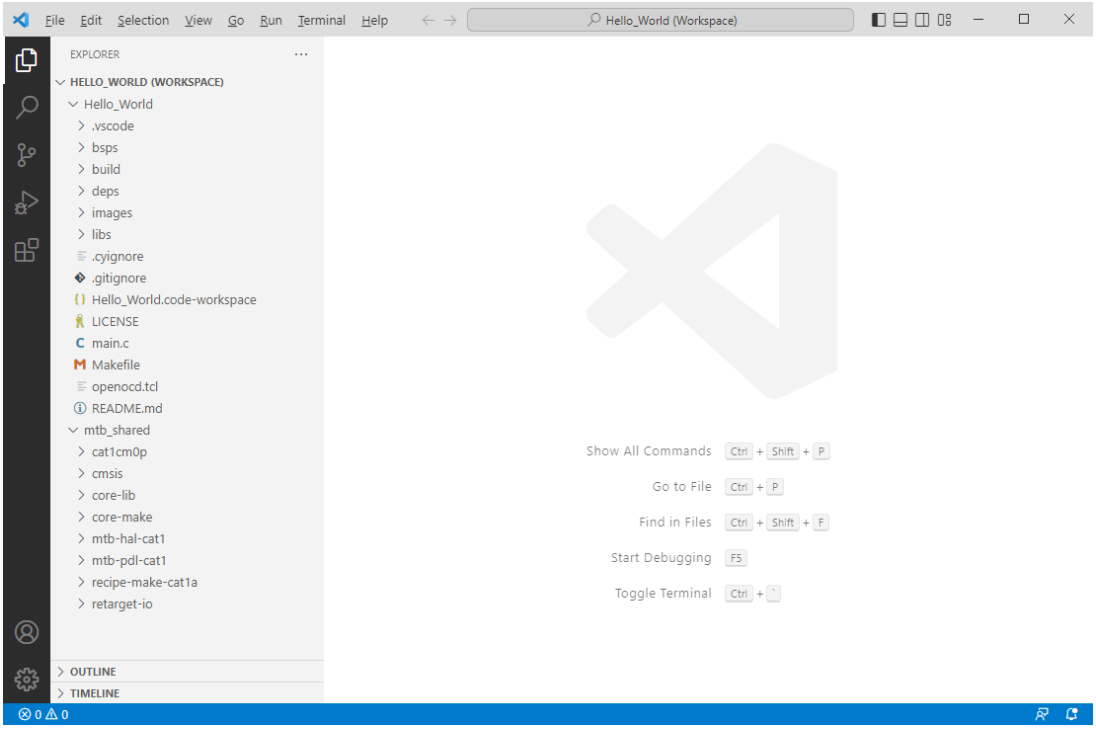
If you have a ModusToolbox™ application VS Code workspace on disk already, select **File > Open Workspace from File**, navigate to the location of the application, select the workspace file, and click **Open**.



Depending on your settings in VS Code, you may see a message about trusting the authors. If so, click **Yes, I trust the authors**.



VS Code opens with the Hello_World workspace in the EXPLORER view.



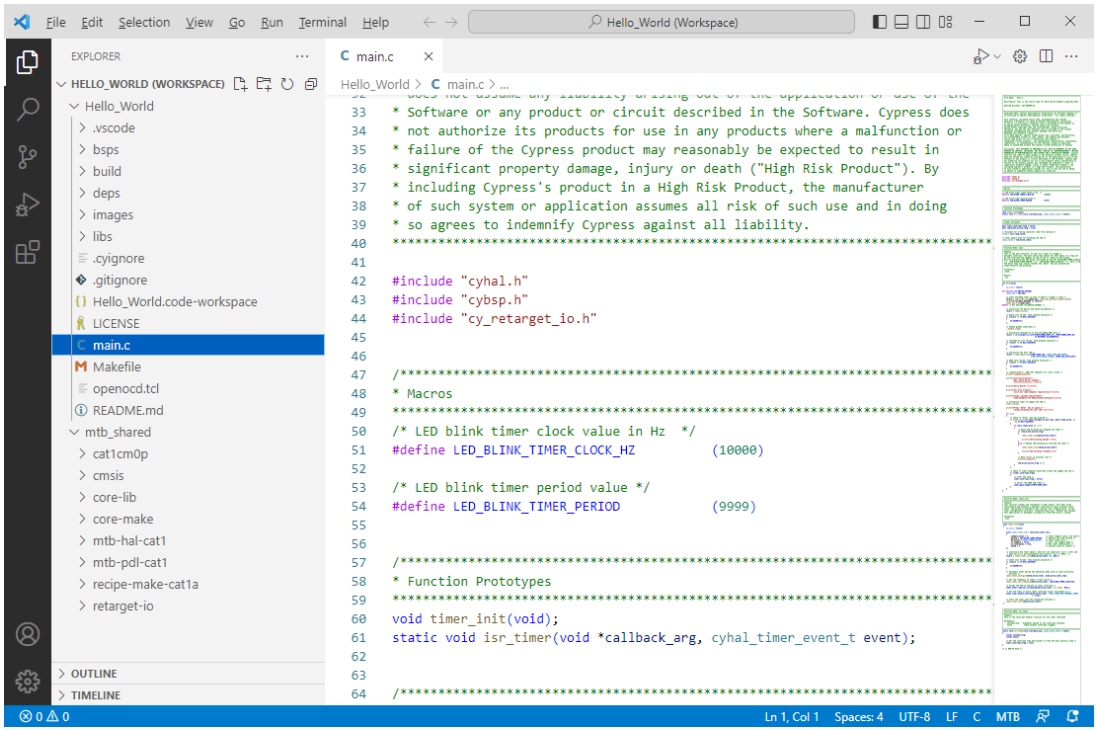
***Note**: If you open the application folder directly (**File > Open Folder**) instead of the workspace file, the ModusToolbox™ Assistant extension automatically detects an existing VS Code workspace file (.code-workspace) in the application directory and re-opens the application using it. If no workspace file exists, the extension creates one for you so the application is ready to build, program, and debug in VS Code.*

3.3 Add/modify application code

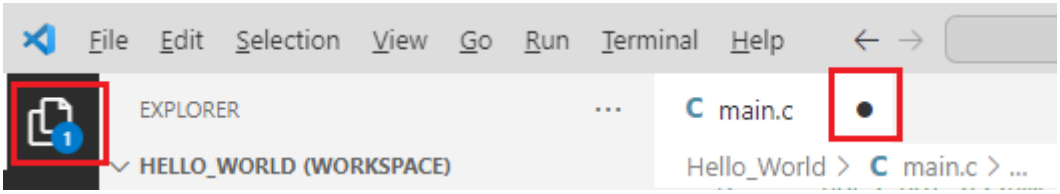
Code example applications work as they are, and there is no need to add or modify code in order to build or program them. However, if you want to update and change the application to do something else, open the appropriate file in the code editor.

Double-click the *main.c* file to open it.

3 Not using the extension

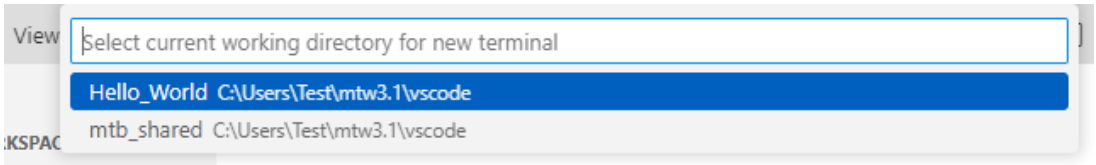


As you type into the file, a dot will appear in the file's tab to indicate changes were made. The file icon will also indicate that there are unsaved changes.



3.4 Command line

Alternatively, you can open various ModusToolbox™ tools using make commands in the terminal. Select **Terminal > New Terminal**, then select the main project folder for your application (in this case, Hello_World):



***Note**: On Windows, use the modus-shell (Cygwin) terminal.*

This section covers a few of the tools you might open more frequently. For a complete list of the tools available, refer to the ModusToolbox™ tools package user guide.

3.4.1 Library Manager

To add, remove, or modify libraries, open the Library Manager using the following command:

```
make library-manager
```

Refer to the Library Manager user guide for details about that tool.

3.4.2 BSP Assistant

To create or modify a BSP, open the BSP Assistant using the following command:

```
make bsp-assistant
```

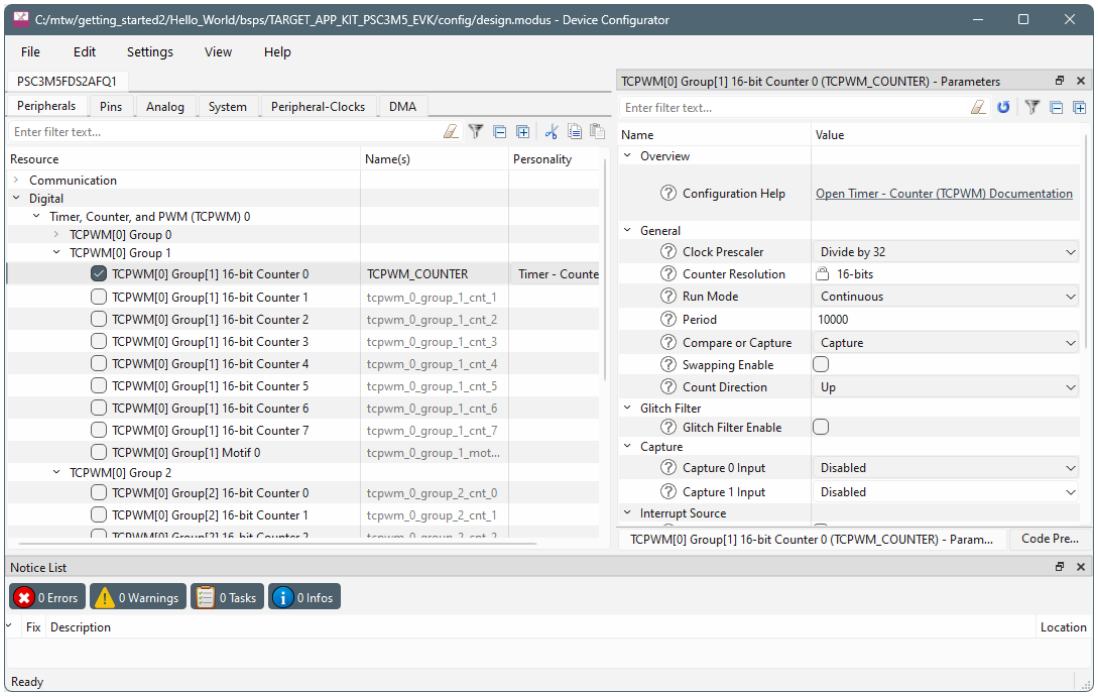
Refer to the BSP Assistant user guide for details about that tool.

3.4.3 Device Configurator

To view peripherals, pins, clocks, etc., open the Device Configurator using the following command:

```
make device-configurator
```

The Device Configurator provides access to the BSP resources and settings. Each enabled resource contains one or more links to the related API documentation. There are also buttons to open other configurators for CAPSENSE™, QSPI, Smart I/O, etc. For more information, refer to the Device Configurator user guide, which is also available by selecting **View Help** from the tool's **Help** menu.



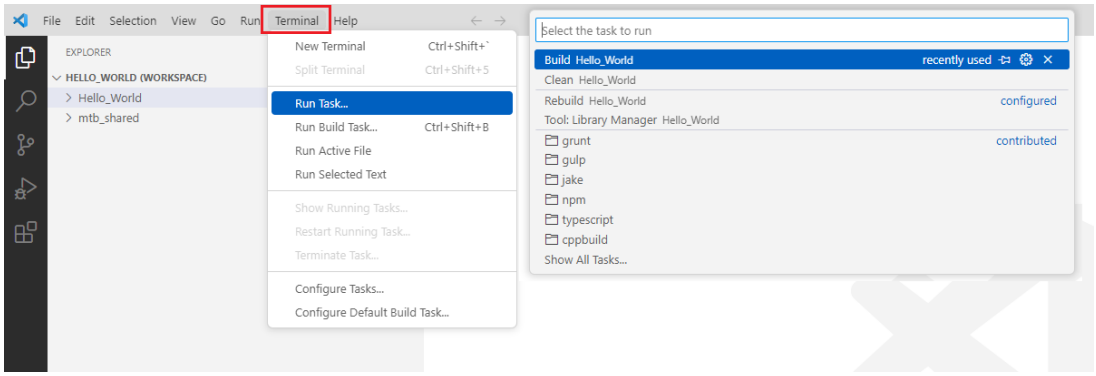
***Note**: The Device Configurator cannot be used to open Library Configurators, such as Bluetooth®.*

3.5 Build the Application

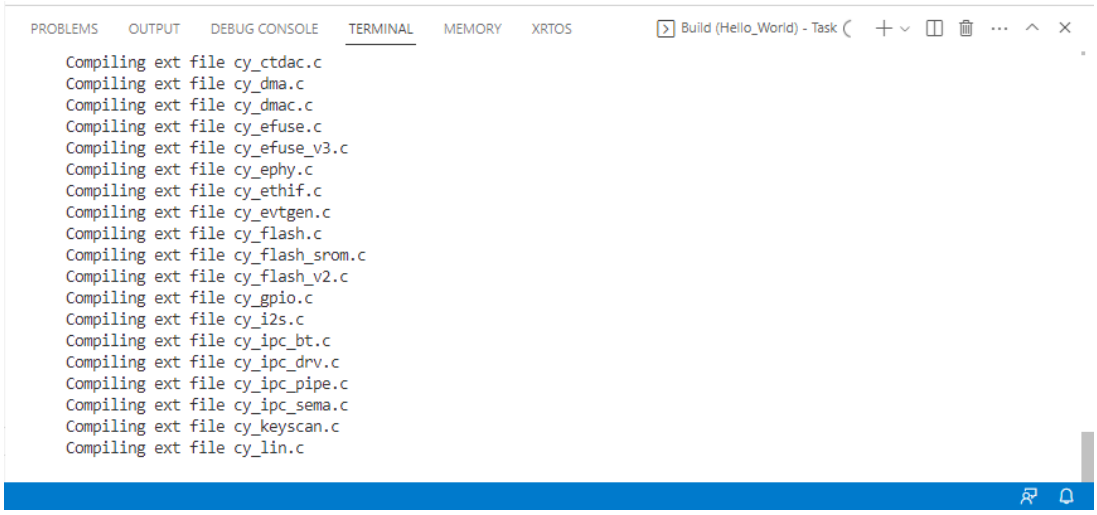
Building the application is not specifically required, because building will be performed as part of the programming and debugging process. However, if you are running VS Code without any hardware attached you may wish to build your application to ensure all the code is correct.

Select **Terminal > Run Task**. Then select **Build Hello_World**.

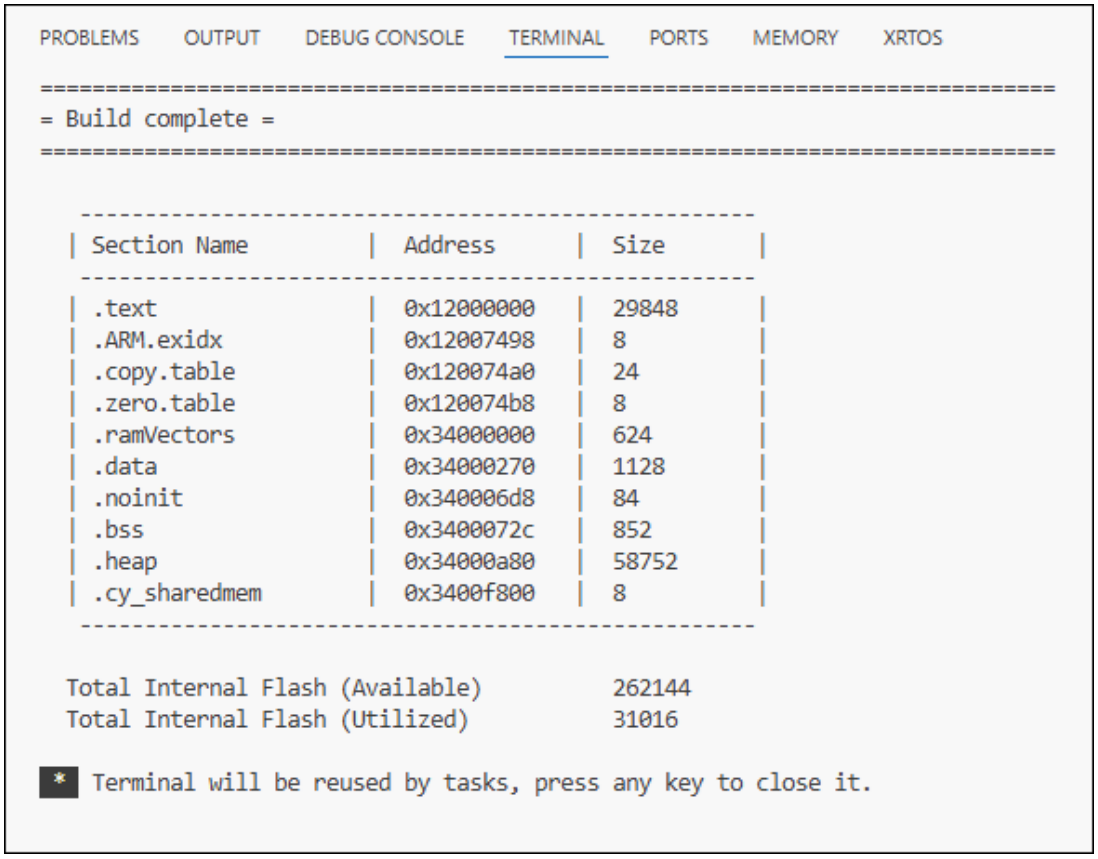
3 Not using the extension



Build information will display in the Terminal.



The build should complete successfully with messages similar to the following:



3.6 Program/debug

This section covers various aspects of programming and debugging using VS Code:

- Program/debug common
- Program/debug using KitProg3/MiniProg4
- Program/debug using J-Link
- Multi-core debugging
- Program/debug secure configuration devices

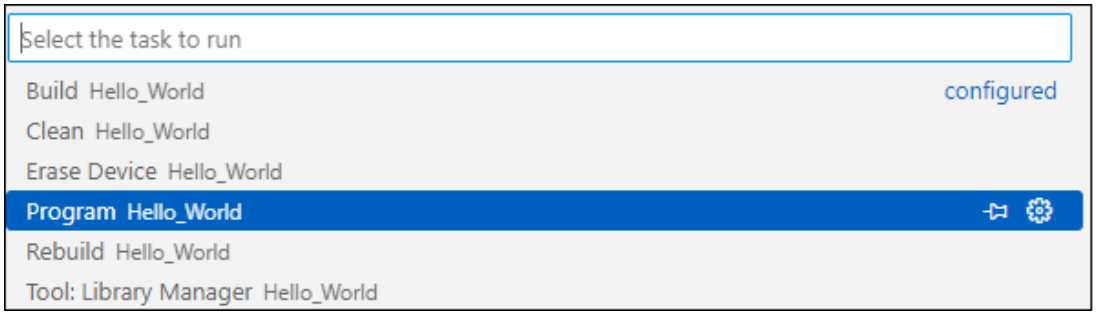
3.6.1 Program/debug common

The VS Code GUI shows these launch configurations by default:

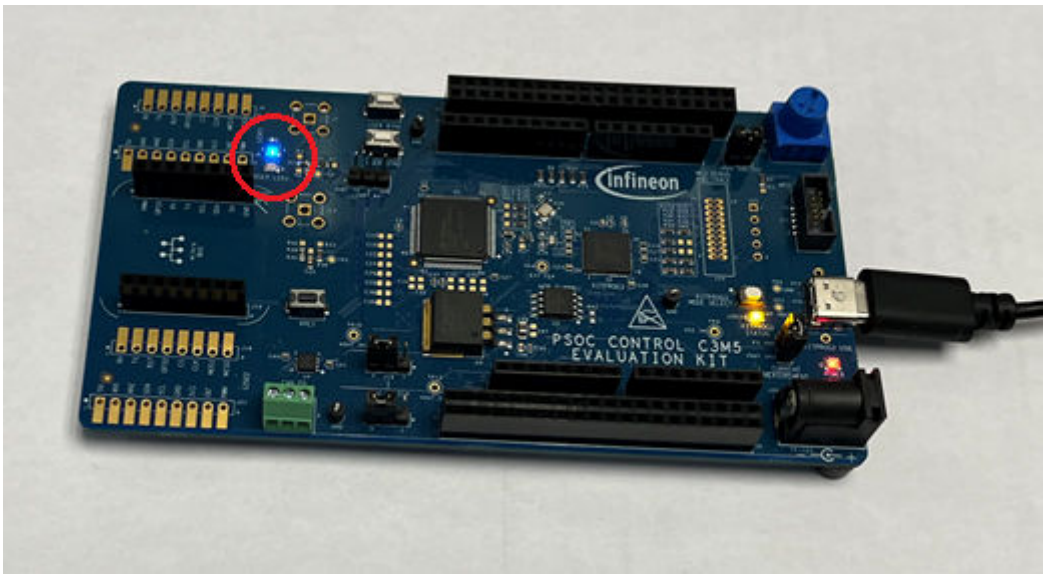
- **Launch**: This builds the associated project, programs project-specific output file, and then starts a debugging session.
- **Attach**: This starts a debugging session attaching to a running target without programming or resetting.
- **Erase Device**: This erases all internal memories.
- **Erase All**: If present, erases all internal and external memories.
- **Program**: This builds the associated project, programs project-specific output file, and then runs the program.

3.6.1.1 Program

Open the main menu, select **Terminal > Run Task**. On the selection menu, select the "program" task.

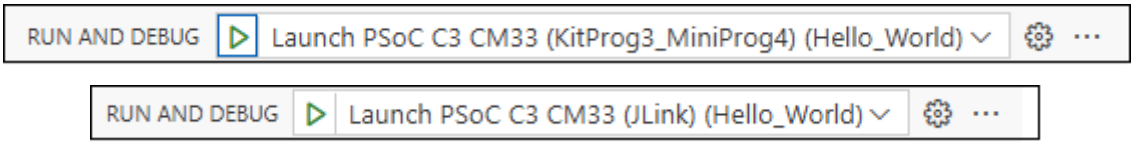


If needed, VS Code builds the application and messages display in the Terminal. If the build is successful, device programming starts immediately. If there are build errors, then error messages will indicate as such. When programming completes successfully, the LED will start blinking.



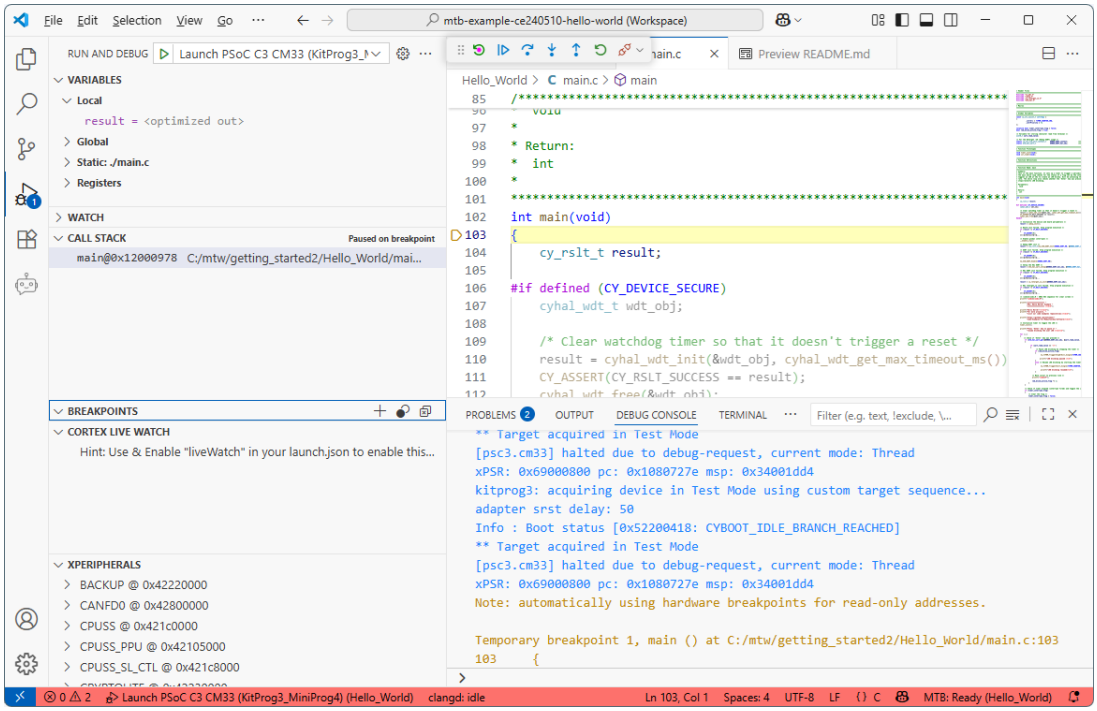
3.6.1.2 Debug

Select the **Run and Debug** icon in the VS Code Activity Bar, select the **Launch PSoC C3 CM33 (KitProg3_Miniprog4)** or **Launch PSoC C3 CM33 (JLink)** Launch Configuration, and click **Start Debugging** icon or press **F5**.



If needed, VS Code builds the application and messages display in the Console. If the build is successful, VS Code switches to debug mode automatically. If there are build errors, then error messages will indicate as such.

3 Not using the extension



3.6.1.3 Changing programming interface SWD/JTAG

To change the target interface, update the application's *bsp.mk* file by adding a make variable as shown (possible values are 'swd' and 'jtag').

```
MTB_PROBE_INTERFACE=swd
```

Then, regenerate launch configurations:

3.6.1.4 Update debugger serial number

If there are two or more debugger probes connected to your computer, the first detected probe will be used by default. There should not be more than one probe with the same serial number. Use this method if you want to use only one specific device. Use OS-specific tools to determine the serial number of connected USB devices.

Update application's *bsp.mk* file by adding variable below with the serial number specified, and regenerate launch configurations:

```
MTB_PROBE_SERIAL=0B0B0F9701047400
```

3.6.1.5 Add Live Watch

While debugging an application in VS Code, it is possible to add a Live Watch variable. This topic provides an example using the Hello World application.

***Note**: Live Watch is not supported for multi-core applications.*

1. Open the application's *launch.json* file, and locate the launch configuration. In this case "Launch PSoC C3 CM33 (KitProg3_MiniProg4)".
2. Scroll to the end of the configuration and add the following:

```
"liveWatch": {
                "enabled": true,
                "samplesPerSecond": 4
}
```

3. Open the *main.c* file and declare a global variable:

```
volatile int count = 0;
```

4. Add the code to increment the variable every time the LED is blinking:

```
count++;
```

```
        /* Check if timer elapsed (interrupt fired) and toggle the LED */
        if (timer_interrupt_flag)
        {
            /* Clear the flag */
            timer_interrupt_flag = false;

            /* Invert the USER LED state */
            Cy_GPIO_Inv(CYBSP_USER_LED_PORT, CYBSP_USER_LED_PIN);

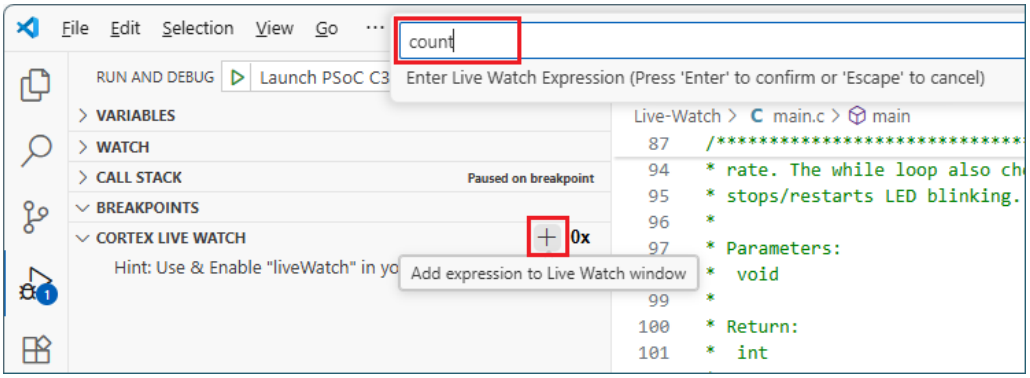
            /* Increment the count in Cortex Watch */
            count++;

        } // if timer_interrupt_flag
    } // for
} // main
```

5. Launch a serial terminal such as PuTTY to monitor the output.

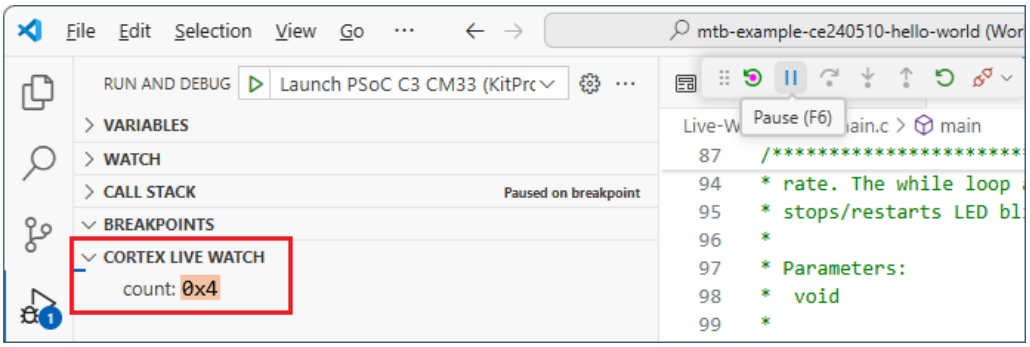
   ***Note**: This step may only be required for applications that have output, such as Hello World.*

6. Start the debugger. When it stops at `main()`, add the `count` variable to the **CORTEX LIVE WATCH** section and press the [**Enter**] key.



7. Check that the variable was added to the list and click on **Run/Continue** to proceed with debugging.

8. Observe that the variable's value increments as the code is running.



3.6.1.6 Add SEGGER SWO/RTT Grapher

You can use the SEGGER real-time transfer (RTT) to visualize the output of the target performed via the SWO pin. This section provides an example:

1. Start by adding the live watch described in the previous topic.

2. In the *main.c* file, add an include for the SEGGER RTT header file:

```
#include "SEGGER_RTT.h"
```

3. Also, change the "count" global variable to `uint32_t`:

```
static uint32_t count = 0;
```

4. Next, locate the main "for" loop. Insert these two lines directly before the loop:

```
SEGGER_RTT_Init();
SEGGER_RTT_ConfigUpBuffer(0, NULL, NULL, 0, SEGGER_RTT_MODE_BLOCK_IF_FIFO_FULL);
```

5. Then, inside the loop, update the `count=count+1;` line to add `CySysLib_Delay();` and `SEGGER_RTT_Write();` commands, as follows:

```
    for (;;)
    {
        Cy_SysLib_Delay(100);
        count=count+1;
        if(count%256==0)
        {
            count=0;
        }
        SEGGER_RTT_Write(0, &count, sizeof(count));

/* code continues */

    }
```

6. Manually add a new file in the *deps* directory named, *segger-rtt.mtb*, with the following content:

```
https://github.com/SEGGERMicro/RTT#main#$$ASSET_REPO$$/RTT/main
```

7. Edit the **.***cyignore* file with the following content:

```
# Segger RTT
$(SEARCH_RTT)/main/Examples
$(SEARCH_RTT)/main/Syscalls/SEGGER_RTT_Syscalls_IAR.c
```

8. Open the Library Manager and click **Update**, or run `make getlibs` to update the application and acquire the RTT library.

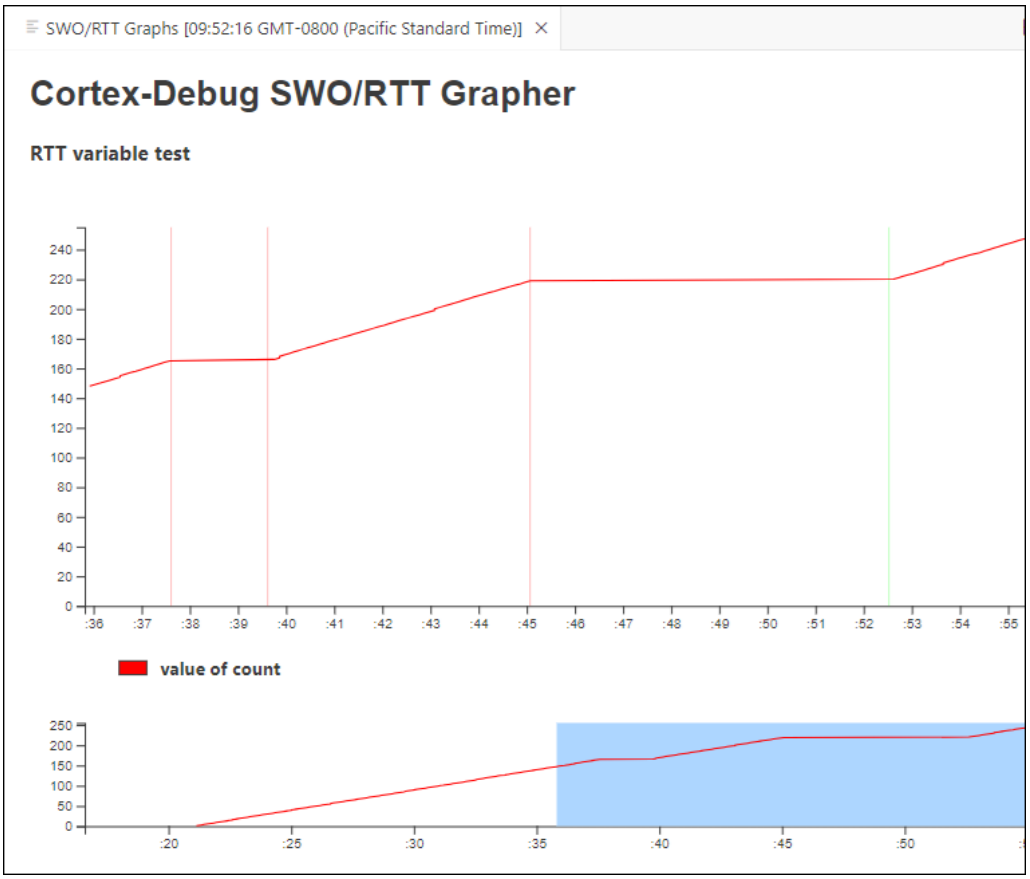
9. Open the *launch.json* file and, in the same "Launch" configuration where you added the live watch, and add the `rttConfig` configuration, as follows:

```
"liveWatch": {
    "enabled": true,
},
"rttConfig": {
    "enabled": true,
    "address": "auto",
    "decoders": [
        {
            "label": "",
            "port": 0,
            "type": "graph",
            "encoding": "unsigned",
            "graphId": "count",
            "scale": 1
        }
    ]
},
```

10. Then, add the `graphConfig` configuration:

```
"graphConfig": [
    {
        "label": "RTT variable test",
        "timespan": 20,
        "type": "realtime",
        "annotate": true,
        "maximum": 255,
        "minimum": 0,
        "plots": [
            {
                "graphId": "count",
                "label": "value of count",
                "color": "#FF0000"
            }
        ]
    }
]
```

11. Save all files, start the debugger, and then click **Continue**.
VS Code will display the RTT grapher.



See https://community.infineon.com/t5/ModusToolbox/Using-Segger-RTT-library-in-ModusToolbox/td-p/965434 for expanding instructions for this example. To learn more about SEGGER RTT support, review these links:

- https://github.com/Marus/cortex-debug/wiki/SEGGER-RTT-support
- https://github.com/Marus/cortex-debug/wiki/SWO-Output#output-graphing-graphing
- https://github.com/SEGGERMicro/RTT.git

3.6.1.7 PSOC™ Control: Setting the OpenOCD probe frequency

The `MTB_PROBE_FREQUENCY_KHZ` variable lets you override the OpenOCD adapter clock speed used by every debug-probe operation (program, erase, debug). It applies to all KitProg3- / MiniProg4-based programming and debugging flows on every supported PSOC™ Control family device, as well as the other recipes that share the common program/IDE infrastructure.

Use it when:

- The default adapter speed is unstable on your board (long flying leads, bus extenders, level shifters).
- You need to slow the probe down for compatibility with custom hardware, or speed it up to reduce program/debug latency on a clean setup.

How to:

Define it once in the application *Makefile*:

```
MTB_PROBE_FREQUENCY_KHZ=2000
```

Regenerate the launch configs by running `make vscode`.

3.6.1.8 BMI for XMC1xxx/4xxx devices

XMC1xxx/4xxx devices use the Boot Mode Index (BMI) value to determine their boot mode and debug configuration after power-up. This value is stored in flash config sector 0.

The default boot mode is either:

- **ASC_BSL (Bootstrap Loader)** – for new, out-of-the-factory, devices. This mode allows erasing and programming flash via UART or SPI interface using the onboard J-Link Lite debugger and legacy Infineon tools, but does not allow programming and debugging in ModusToolbox™ and using the standalone J-Link debugger.
- **SWD0 (Serial Wire Debug via Channel 0)** – for Infineon kits, like XMC1100 Boot Kit. This mode allows any flash operations and debugging via the SWD interface.

There are several other modes, including using different channels (pins), SPD protocol, and flash protection scheme, which you may want to activate (switch to) during the development and production life cycles.

Refer to the following documentation for more details about BMI implementation and usage for XMC1xxx/4xxx devices.

- Boot Mode Index (BMI) XMC™ microcontrollers
- Tooling - Boot mode options XMC4000
- AN_201511_PL30_005 - Boot mode handling for XMC1000
- XMC1100 AB-Step Reference Manual
- SEGGER KBA - Infineon XMC1000: BMI - Boot Mode Index

Switching BMI with the onboard J-Link Lite debugger

With the on-board J-Link Lite debugger, you can switch the BMI using the SEGGER J-Link Commander tool, which is a part of the J-Link Software Pack that must be installed to use XMC1xxx/4xxx in ModusToolbox™.

1. Connect the XMC™ Kit to USB and start J-Link Commander.
2. In J-Link Commander, connect to the target device, providing its name and specifying the SWD interface.

```
SEGGER J-Link Commander V8.44a (Compiled Jun 18 2025 16:33:14)
DLL version V8.44a, compiled Jun 18 2025 16:32:23

Connecting to J-Link via USB...O.K.
Firmware: J-Link Lite-XMC4200 Rev.1 compiled Mar 22 2024 08:22:08
Hardware version: V1.00
J-Link uptime (since boot): 0d 00h 00m 04s
S/N: 591171370
USB speed mode: Full speed (12 MBit/s)
VTref=3.300V


Type "connect" to establish a target connection, '?' for help
J-Link>connect
Please specify device / core. <Default>: XMC1100-0064
Type '?' for selection dialog
Device>
Please specify target interface:
  J) JTAG (Default)
  S) SWD
  T) cJTAG
TIF>s
Specify target interface speed [kHz]. <Default>: 4000 kHz
Speed>
Device "XMC1100-0064" selected.
```

- Use GetBMI command to obtain the current BMI.
- Use SetBMI command without parameters to list all Boot Mode Indexes.
- Use SetBMI <index> command to set switch to desired BMI.

3 Not using the extension

```
Connecting to target via SWD
Trying to identify target via SPD
Could not identify target via SPD. Trying again via SWD.
Failed to attach to CPU. Trying connect under reset.
Trying to identify target via SPD
Could not identify target via SPD. Trying again via SWD.
Error occurred: Could not connect to the target device.
For troubleshooting steps visit: https://wiki.segger.com/J-Link_Tr
J-Link>GetBMI
Current BMI mode: 0.
J-Link>SetBMI
Syntax: SetBMI <Mode>
Valid values for <Mode>:
  0 ASC Bootstrap Load Mode (ASC_BSL)
  1 User Mode (Productive)
  2 User Mode (Debug) SWD0
  3 User Mode (Debug) SWD1
  4 User Mode (Debug) SPD0
  5 User Mode (Debug) SPD1
  6 User Mode (HAR) SWD0
  7 User Mode (HAR) SWD1
  8 User Mode (HAR) SPD0
  9 User Mode (HAR) SPD1
J-Link>SetBMI 2
Setting BMI mode 2...O.K.
J-Link>connect
Device "XMC1100-0064" selected.
```

The debugger is now able to connect and identify the target after switching the BMI.

```
Connecting to target via SWD
Trying to identify target via SPD
Could not identify target via SPD. Trying again via SWD.
Found SW-DP with ID 0x0BB11477
DPIDR: 0x0BB11477
CoreSight SoC-400 or earlier
Scanning AP map to find all available APs
AP[1]: Stopped AP scan as end of AP map has been reached
AP[0]: AHB-AP (IDR: 0x04770021, ADDR: 0x00000000)
Iterating through AP map to find AHB-AP to use
AP[0]: Core found
AP[0]: AHB-AP ROM base: 0xF0000000
CPUID register: 0x410CC200. Implementer code: 0x41 (ARM)
Found Cortex-M0 r0p0, Little endian.
```

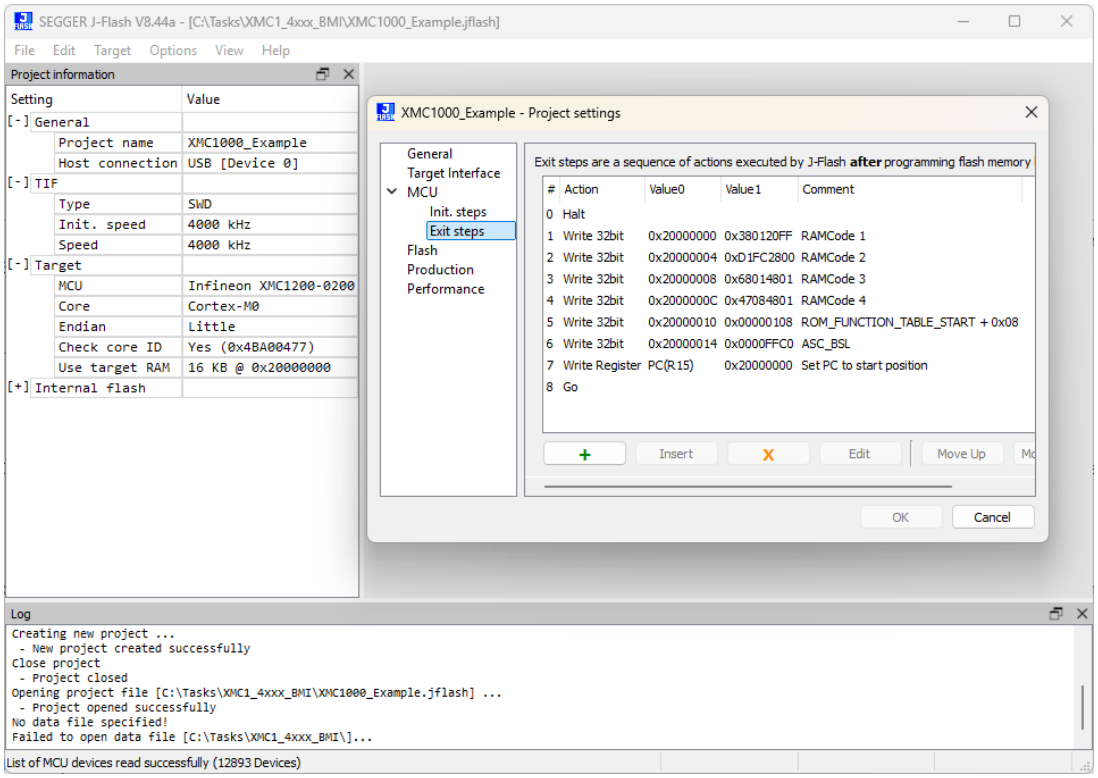
Switching BMI with stand-alone J-Link debugger

Stand-alone J-Link debuggers will automatically switch the BMI mode from ASC_BSL to SWD0 on target connection.

No additional steps required for programming and debugging in ModusToolbox™ if Channel 0 is used for debug pins (SWDIO - P0.14, SWCLK - P0.15).

If a BMI other than SWD0 is required, you can use one of three *.pex scripts (ASC_BSL to SWD1, ASC_BSL to SPD0, and ASC_BSL to SPD1) available in SEGGER's KBA. You can also use the J-Flash tool, where the desired BMI mode can be applied after programming, like in the example project file.

3 Not using the extension



3.6.2 Program/debug using KitProg3/MiniProg4

Most PSOC™-based kits use KitProg3/MiniProg4 as the default programmer/debugger, so there is nothing to configure for them.

3.6.2.1 Connect the Kit

Follow the instructions provided with the kit to connect it to the computer with the USB cable.

3.6.2.2 KitProg Firmware Loader

The PSOC™ MCU kits include on-board programmer/debug firmware, called KitProg. KitProg3 is the latest firmware version. However, some older kits come with KitProg2 firmware installed, which does not work with the ModusToolbox™ software and you must update them to KitProg3. KitProg3 provides the CMSIS-DAP (Bulk) protocol by default, which is up to ~2.5 times faster than the CMSIS-DAP (HID) protocol. Both modes can be used via OpenOCD.

To update it, use the ModusToolbox™ Programmer GUI or the fw-loader CLI tool. Both are provided with the ModusToolbox™ Programming tools package available from the Setup program.

For more details about these tools, refer to the ModusToolbox™ Programmer GUI user guide or the Firmware Loader user guide.

***Note**: On a Linux machine, you must run the udev_rules\install_rules.sh script before the first run of the fw-loader.*

3.6.2.2.1 Supplying power with KitProg3_MiniProg4

If using the KitProg3 connector on a kit, power is generally supplied by the host PC. When using a MiniProg4, power is not supplied via the MiniProg4 by default. It is expected that the target MCU will be powered externally. However, the MiniProg4 does provide the ability to supply power to the target MCU.

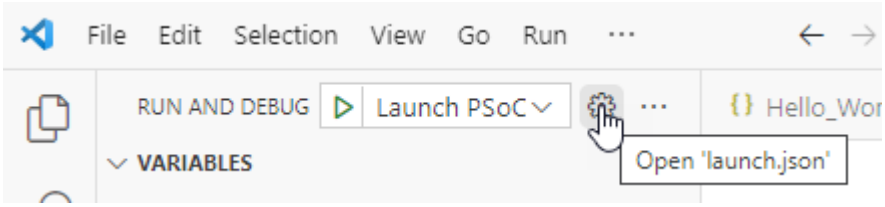
***Note**: Verify the voltage range supported by the target MCU, since it can be damaged by supplying unsupported voltage. Make sure that your MCU is not powered externally before supplying power via the KitProg3_MiniProg4 launch configuration. This supply is limited to approximately 200 mA and is protected against excess current draw. You can select 1.8 V, 2.5 V, 3.3 V, or 5 V.*

Turning power supply on

Debug session

To turn power supply on during a debug session, edit the Launch configurations:

1. Open the **Run and Debug** view, select the launch configuration to be modified, and click the gear icon that opens the selected launch configurations in *launch.json* file.



2. Look for the `openOCDPreConfigLaunchCommands` property. If it is not present, add it. Update the property to include the following value:

```
"set ENABLE_POWER_SUPPLY <mV>"
```

Where `<mV>` defines target voltage in millivolts. For example:

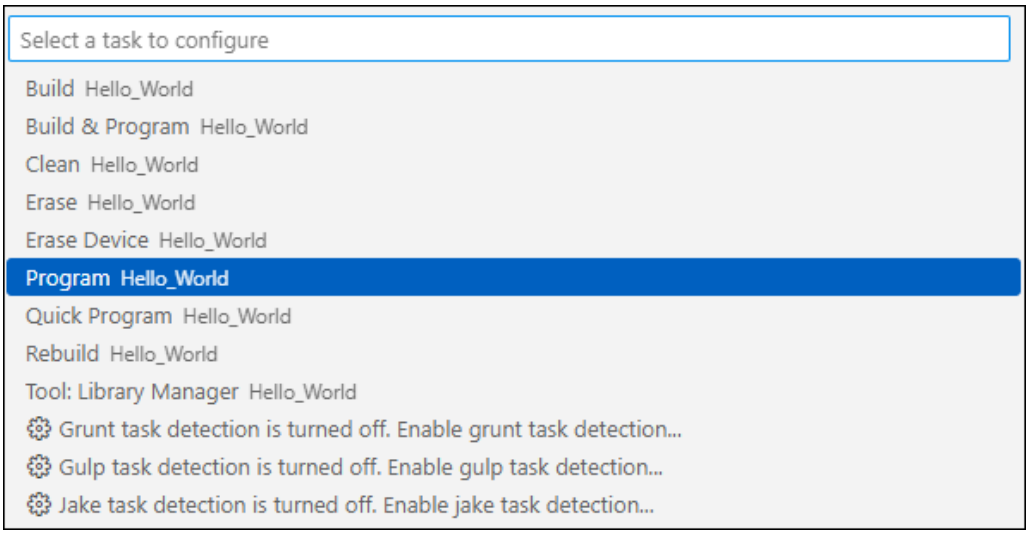


3. Save the changes to the file.

Programing or erasing

To turn power supply on when programing or erasing, edit the Task configurations:

1. On the main menu, select **Terminal > Configure Task...**.



2. On the dialog, select the task to be modified, which opens the *tasks.json* file to that task.
3. Update the task to add the following value to the `args` property:

```
_MTB_RECIPE__OPENOCD_INTERFACE=\"source [find interface/kitprog3.cfg]; set
ENABLE_POWER_SUPPLY <mV>";\
```

Where <mV> defines target voltage in millivolts. For example:

```
{
    "label": "Program",
            "type": "process",
    "command": "bash",
    "args": [
        "--norc",
        "-c",
        "make -j8 program _MTB_RECIPE__OPENOCD_INTERFACE=\"source [find interface/
kitprog3.cfg]; set ENABLE_POWER_SUPPLY 3300";\ --output-sync"
    ],
    "windows": {
        "command": "${config:modustoolbox.toolsPath}/modus-shell/bin/bash.exe",
        "args": [
            "--norc",
            "-c",
            "export PATH=/bin:/usr/bin:$PATH ; ${config:modustoolbox.toolsPath}/modus-
shell/bin/make.exe -j8 program _MTB_RECIPE__OPENOCD_INTERFACE=\"source [find interface/
kitprog3.cfg]; set ENABLE_POWER_SUPPLY 3300";\ --output-sync"
        ]
    },
    "problemMatcher": "$gcc",
    "group": {
        "kind": "build"
    }
},
```

4. Save the changes to the file.

3.6.2.2.2 Power cycle programming mode with KitProg3_MiniProg4

***Note**: This section is applicable to PSOC™ 6 and PSOC™ 4 only.*

By default, Launch Configurations use Reset mode to program the device. However, Reset mode is not available in all situations (for example, if the XRES pin is not available on the part's package). In these cases, Launch Configurations use an alternative reset with software. However, using the software reset type is not sufficient in cases in which access to the device's DAP is restricted (such as when set by security settings).

If there is no XRES pin available and DAP access is restricted, the only way to reset a part is to use Power Cycle mode. Follow these instructions to add commands to the launch configuration and switch to Power Cycle mode.

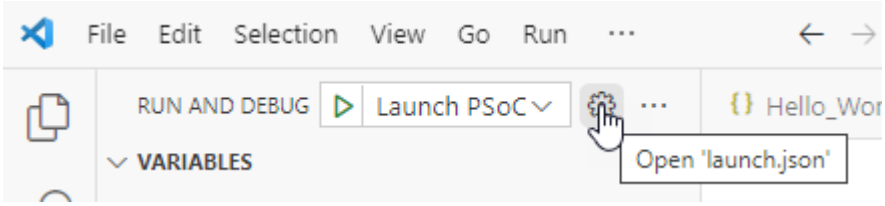
***Note**: Verify the voltage range supported by the target MCU, since it can be damaged by supplying unsupported voltage. Make sure that your MCU is not powered externally before supplying power via the KitProg3_MiniProg4.*

Debug session

To enable power cycle during a debug session, edit the Launch configurations:

1. Open the **Run and Debug** view, select the launch configuration to be modified, and click the gear icon that opens the selected launch configuration in the *launch.json* file.

3 Not using the extension



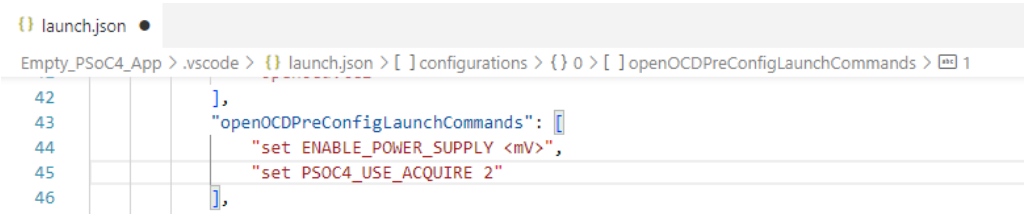
2. Look for the `openOCDPreConfigLaunchCommands` property. If it is not present, add it. Update the property to include the following value:
   - For PSOC™ 6:

     ```
     "set ENABLE_POWER_SUPPLY <mV>",
     "set ENABLE_ACQUIRE 2"
     ```

   - For PSOC™ 4:

     ```
     "set ENABLE_POWER_SUPPLY <mV>",
     "set PSOC4_USE_ACQUIRE 2"
     ```

   Where `<mV>` defines target voltage in millivolts. For example:

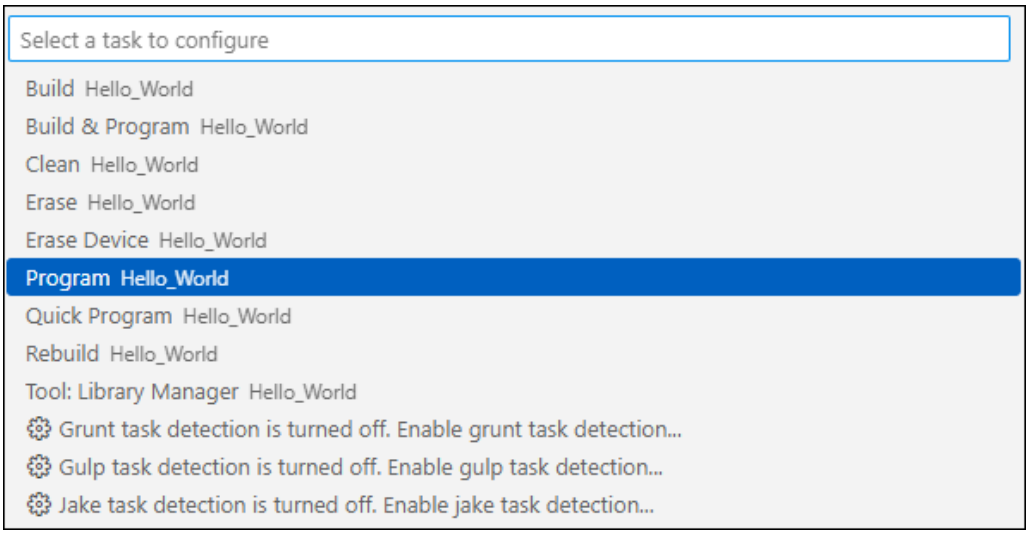


3. Save the changes to the file.

Programing or erasing

To enable power cycle when programing or erasing, edit the Task configurations:

1. On the main menu, select **Terminal > Configure Task...**.



2. On the dialog, select the task to be modified, which opens the *tasks.json* file to that task.

3. Update the task to add the following value to the `args` property:
   - For PSOC™ 6:

     ```
     _MTB_RECIPE__OPENOCD_INTERFACE=\"source [find interface/kitprog3.cfg]; set
     ENABLE_POWER_SUPPLY <mV>; set ENABLE_ACQUIRE 2;\"
     ```

   - For PSOC™ 4:

     ```
     _MTB_RECIPE__OPENOCD_INTERFACE=\"source [find interface/kitprog3.cfg]; set
     ENABLE_POWER_SUPPLY  <mV>; set PSOC4_USE_ACQUIRE 2;\"
     ```

   Where `<mV>` defines target voltage in millivolts. For example:
4. Save the changes to the file.

3.6.3 Program/debug using J-Link

Most PSOC™-based BSPs default to using the KitProg3/MiniProg4 programmer/debugger launch configurations. This section covers how to use J-Link.

3.6.3.1 Configure J-Link programmer/debugger settings

1. Open your ModusToolbox™ application's *bsp.mk* file and enter the following variable:

   ```
   BSP_PROGRAM_INTERFACE=JLink
   ```

2. Also enter the following variable to specify the path to the J-Link install directory:

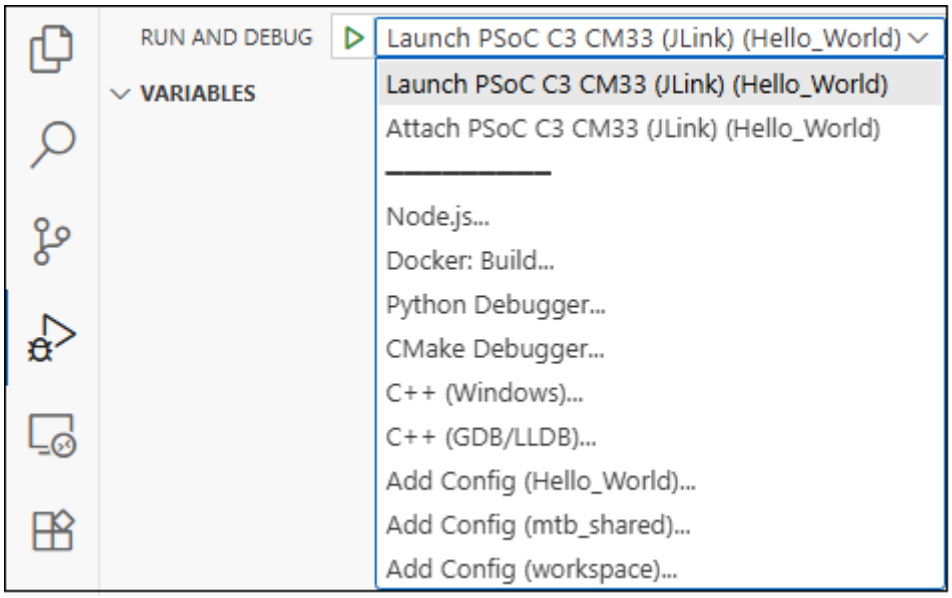
   ```
   MTB_JLINK_DIR=<path to J-Link>
   ```

3. Save the *bsp.mk* file.

4. In a bash Terminal run:

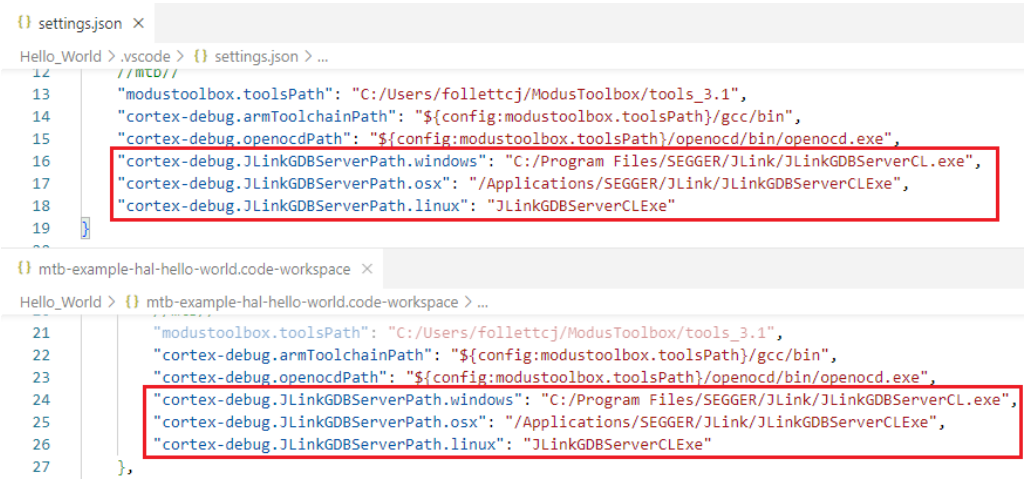
   ```
   make vscode
   ```

   When the command completes, J-Link configurations will be shown. These are the same configurations described in Program/debug common, but applicable to J-Link.



5. Open the *settings.json* file and *<app>.code-workpace* file to verify the path to the J-Link GDB server. For example, the default on Windows is:

```
"cortex-debug.JLinkGDBServerPath": "C:/Program Files/SEGGER/JLink/JLinkGDBServerCL.exe"
```



3.6.3.2 Connect the Kit

Follow the instructions provided with the kit and from SEGGER to connect it to the computer with the J-Link probe.

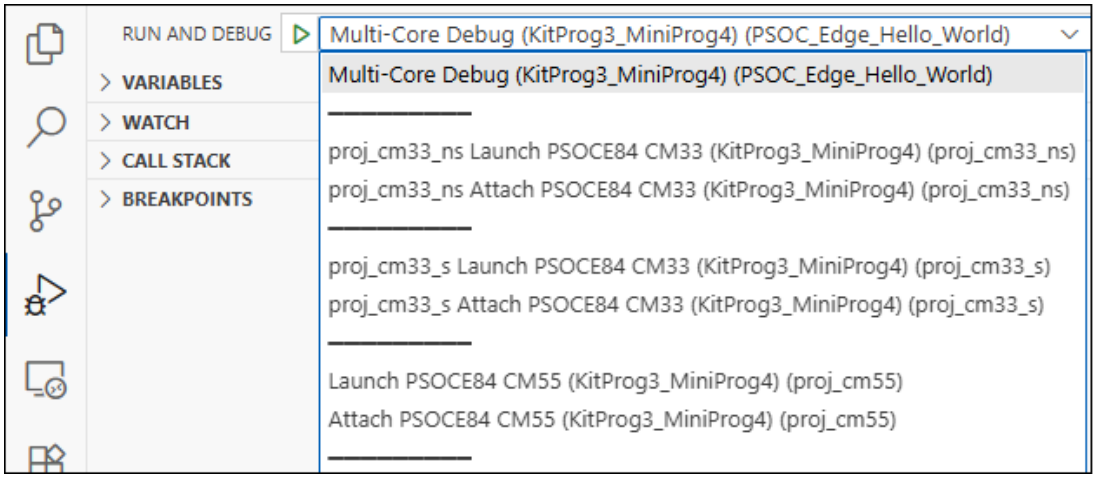
3.6.4 Multi-core debugging

Projects created for VS Code also provide debug configurations for multi-core applications. They support these probes:

- KitProg3 onboard programmer
- MiniProg4
- J-Link (See Configure J-Link programmer/debugger settings)

3.6.4.1 Configurations

The configurations support debugging one core at a time and multiple cores as well. After the application has opened, there will be several configurations available for use in the **Run and Debug** tab of Activity Bar as shown.



These include:

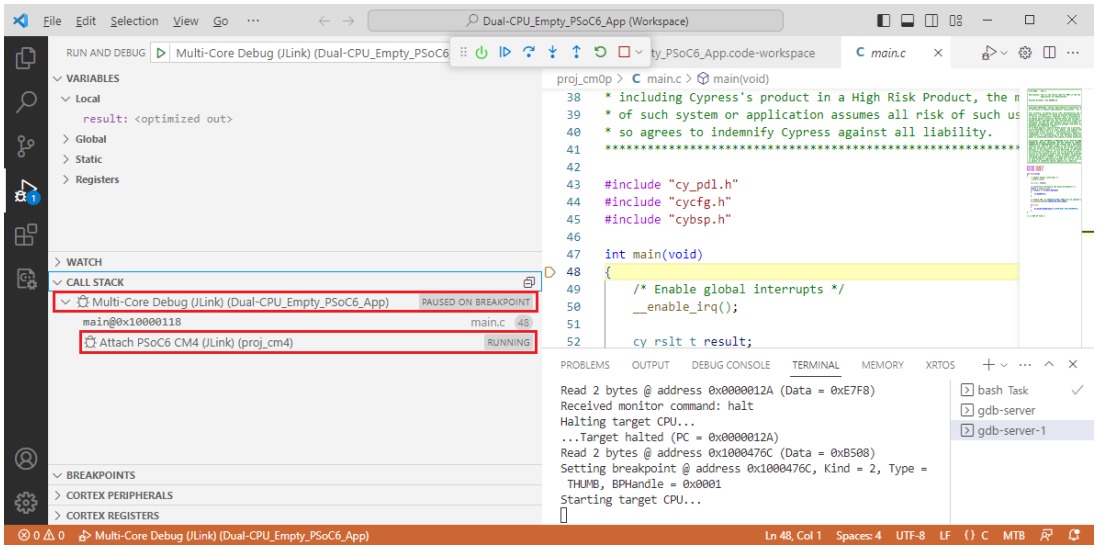
- **Multi-Core Debug**: programs multiple hex files, launches OpenOCD/J-Link GDB Server and starts multi-core debug session
- **Launch <device>**: launches debug session on the chosen core
- **Attach <device>**: attaches to the running core

In addition to these configurations, there is associated VS Code tasks available through the main menu **Terminal > Run Task**.

- **Erase Device**: erases all internal memory banks.
- **Program <application_name>**: downloads combined hex file into the flash.
- **Program <project_name>**: downloads project-specific hex file into the flash.
- **Erase All**: If present, erases all internal and external memories.

3.6.4.2 Launch the configuration

To launch multi-core debugging, run the **Multi-Core Debug** configuration. You will end up with a debug session containing two debug processes in CALL STACK view.



Once a session has started, the CM0+ core is halted at the beginning of `main()`, while the CM4 core is spinning in an endless loop in boot code, waiting for start. It will start and halt at `main()` as soon as the application running on the CM0+ executes the `Cy_SysEnableCM4()` function.

In the CALL STACK view you can observe two debug processes, each of them associated with a specific core. You can switch between the cores by selecting the appropriate process.

***Note**: There is one limitation for XMC7000 MCUs. Before launching a multi-core debug session, you must program the MCU by launching the **Program Application** configuration.*

3.6.4.3 Multi-core debug CM33 secure application booting from RRAM

When using multi-core debug CM33 secure applications booting from RRAM (for example PSOC_Edge_RTC_periodic_wakeup), you must configure the launch configuration to add the SMIF IP enabling commands.

3 Not using the extension

launch.json

PSOC_Edge_RTC_periodic_wakeup > .vscode > {} launch.json > Launch Targets > {} Add CM55 to CM33 PSOCE84 (KitProg3_MiniProg4)

```
"configurations": [
    {
        "name": "Multi-Core Debug (KitProg3_MiniProg4)",
        "type": "cortex-debug",
        "request": "launch",
        "cwd": "${workspaceFolder}",
        "executable": "proj_cm33_s/build/last_config/proj_cm33_s.elf",
        "servertype": "openocd",
        "searchDir": [
            "${workspaceFolder}",
            "${config:modustoolbox.toolsPath}/../packs/PSOC-E84-EA/tools/openocd/scripts/",
            "bsps/TARGET_APP_KIT_PSE84_EVAL/config/GeneratedSource"
        ],
        "configFiles": [
            "openocd.tcl"
        ],
        "preLaunchCommands": [
            "add-symbol-file proj_cm33_ns/build/last_config/proj_cm33_ns.elf"
        ],
        "openOCDPreConfigLaunchCommands": [
            "set DEBUG_CERTIFICATE ./packets/debug_token.bin"
        ],
        "overrideLaunchCommands": [
            "set mem inaccessible-by-default off",
            "-enable-pretty-printing",
            "set remotetimeout 90",
            "monitor reset init",
            // Comment this next lines out if you don't want to reload program
            "monitor flash write_image erase {build/app_combined.hex}",
            "monitor reset init",
            "monitor reset_halt cm33",
            "monitor mww 0x54004054 0",
            "monitor mww 0x54004050 4"
        ],
        "overrideRestartCommands": [
            "monitor reset_halt cm33",
            "monitor mww 0x54004054 0",
            "monitor mww 0x54004050 4"
        ],
```

Add the following to `overrideLaunchCommands` and `overrideRestartCommands` parameters in the "Multi-Core Debug" configuration:

For KitProg3/MiniProg4

```
monitor mww 0x54004054 0
monitor mww 0x54004050 4
```

For J-Link

```
monitor memU32 0x54004054=0
monitor memU32 0x54004050=4
```

3.6.5 Program/debug secure configuration devices

The BootROM on devices such as PSOC™ Control and PSOC™ Edge can temporarily disable access to the core's Access Port (AP). This behavior is determined by the security policy during device provisioning. However, access can be re-enabled using a certificate (debug token). If the debugger identifies that the AP is disabled, it will upload and verify the certificate located in the *./packets/debug_token.bin* file within the current project folder.

***Note**: For details about PSOC™ Control C3 and PSOC™ Edge E8 MCU security, refer to Application Notes:*

- AN240106 – Getting started with PSOC™ Control C3 security
- AN237849 – Getting started with PSOC™ Edge security

By default, the certificate is located in *./packets/debug_token.bin* in the application root folder for both OpenOCD and Segger J-Link.

3.6.5.1 OpenOCD

You can optionally make changes to the certificate's setting using make commands and OpenOCD commands.

Make variable (Program and Debug

You can optionally specify a different path and address for the certificate using the following OpenOCD commands:

To change the location and name of the certificate (address cannot be changed) for programming and debugging, use the make variable `CY_DBG_CERTIFICATE_PATH`.

You can use the variable in the *Makefile*. For example:

```
CY_DBG_CERTIFICATE_PATH=../new_location/new_name_4_token.bin
```

***Note**: If you use this variable, it will apply the new path to the certificate for all Launch Configurations (Debug, Program, Erase), but you must first regenerate the Launch Configuration using the `make vscode` command.*

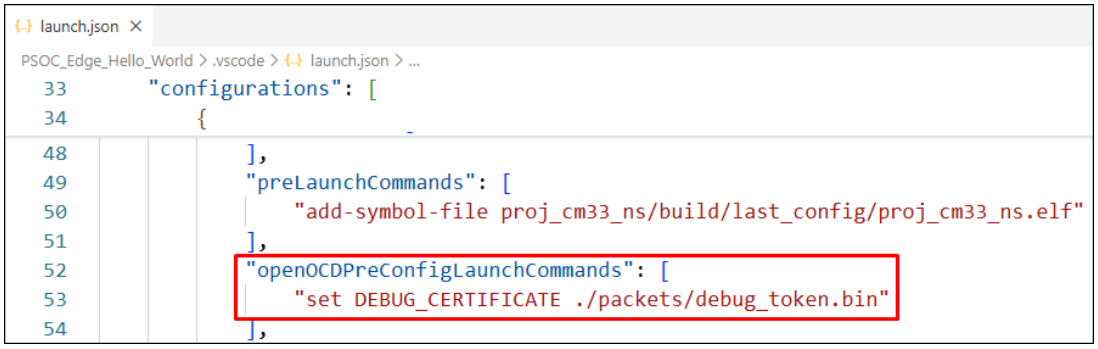
You can also run the make variable in a terminal, and it will only apply to the specific make command. For example:

```
make program CY_DBG_CERTIFICATE_PATH=../new_location/new_name_4_token.bin
make erase CY_DBG_CERTIFICATE_PATH=../new_location/new_name_4_token.bin
```

OpenOCD commands (Debug only)

Another option to specify a different path and address for the certificate is by using the following OpenOCD commands. These apply only during a debug session:

```
"set DEBUG_CERTIFICATE %PATH_TO_TOKEN%"
"set DEBUG_CERTIFICATE_ADDR %HEX_ADDRESS%"
```



3.6.5.2 J-Link

For single-core applications, the certificate must be located in *./packets/debug_token.bin* in the application root folder. Its name and location cannot be changed.

For multi-core applications, there must be multiple certificates located in *./packets/debug_token.bin* in the application root folder, as well as all sub-project folders. The names and locations cannot be changed.

Trademarks

All referenced product or service names and trademarks are the property of their respective owners.

Edition 2026-07-10

Published by

Infineon Technologies AG

81726 Munich, Germany

© 2026 Infineon Technologies AG

All Rights Reserved.

Do you have a question about any aspect of this document?

Email: erratum@infineon.com

Document reference

IFX-gal1782164011158

Important notice

Products which may also include samples and may be comprised of hardware or software or both ("Product(s)") are sold or provided and delivered by Infineon Technologies AG and its affiliates ("Infineon") subject to the terms and conditions of the frame supply contract or other written agreement(s) executed by a customer and Infineon or, in the absence of the foregoing, the applicable Sales Conditions of Infineon. General terms and conditions of a customer or deviations from applicable Sales Conditions of Infineon shall only be binding for Infineon if and to the extent Infineon has given its express written consent.

For the avoidance of doubt, Infineon disclaims all warranties of non-infringement of third-party rights and implied warranties such as warranties of fitness for a specific use/purpose or merchantability. Infineon shall not be responsible for any information with respect to samples, the application or customer's specific use of any Product or for any examples or typical values given in this document.

If this document is marked as 'Preliminary', 'Target', 'Draft' or 'Proposal', Infineon reserves the right to change all information given in this document at any time without notice. Subject to the development and release of a Product for series supply by Infineon, the technical specifications of the Product are set forth in the relevant datasheet provided by Infineon without such marking.

The data contained in this document is exclusively intended for technically qualified and skilled customer representatives. It is the responsibility of the customer to evaluate the suitability of the Product for the intended application and the customer's specific use and to verify all relevant technical data contained in this document in the intended application and the customer's specific use. The customer is responsible for properly designing, programming, and testing the functionality and safety of the intended application, as well as complying with any legal requirements related to its use.

Unless otherwise explicitly approved by Infineon, Products may not be used in any application where a failure of the Products or any consequences of the use thereof can reasonably be expected to result in personal injury.

However, the foregoing shall not prevent the customer from using any Product in such fields of use that Infineon has explicitly designed and sold it for, provided that the overall responsibility for the application lies with the customer. Infineon expressly reserves the right to use its content for commercial text and data mining (TDM) according to applicable laws, e.g. Section 44b of the German Copyright Act (UrhG). If the Product includes security features:

Because no computing device can be absolutely secure, and despite security measures implemented in the Product, Infineon does not guarantee that the Product will be free from intrusion, data theft or loss, or other breaches ("Security Breaches"), and Infineon shall have no liability arising out of any Security Breaches.

If this document includes or references software:

The software is owned by Infineon under the intellectual property laws and treaties of the United States, Germany, and other countries worldwide. All rights reserved. Therefore, you may use the software only as provided in the software license agreement accompanying the software.

If no software license agreement applies, Infineon hereby grants you a personal, non-exclusive, non-transferable license (without the right to sublicense) under its intellectual property rights in the software (a) for software provided in source code form, to modify and reproduce the software solely for use with Infineon hardware products, only internally within your organization, and (b) to distribute the software in binary code form externally to end users, solely for use on Infineon hardware products. Any other use, reproduction, modification, translation, or compilation of the software is prohibited.

For further information on the Product, technology, delivery terms and conditions, and prices, please contact your nearest Infineon office or visit www.infineon.com.