Visual Studio Code for ModusToolbox™ user guide

ModusToolbox™ tools package version 3.1.0

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.
`Courier New`	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™ tools package 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

1 Download/install software

1.1 ModusToolbox™ tools package

Refer to the instructions in the ModusToolbox™ tools package installation guide for how to download and install the ModusToolbox™ tools package.

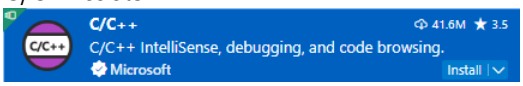
1.2 VS Code

The ModusToolbox™ tools package includes various tools to create and manage applications, but it does **not** include VS Code. If you do not already have VS Code installed on your computer, you can download it from the website:

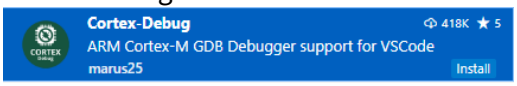
https://code.visualstudio.com/

After opening an application in VS Code, it will recommend several extensions. The C/C++ tools and Cortex-Debug extensions are required for build and debug. Other extensions such as the ModusToolbox Assistant and Arm Assembly improve the development and debug experience.

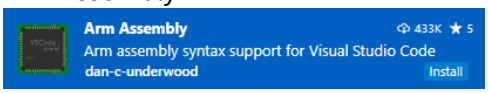
- C/C++ tools



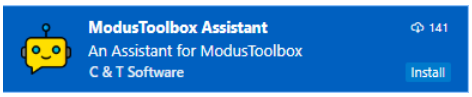
- Cortex-Debug



- Arm Assembly



- ModusToolbox™ Assistant



1.3 J-Link

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

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

2 Getting Started

This section covers the ways to get started using VS Code with ModusToolbox™ software

- Create new application
- Exporting existing application
- Open workspace in VS Code

2.1 Create new application

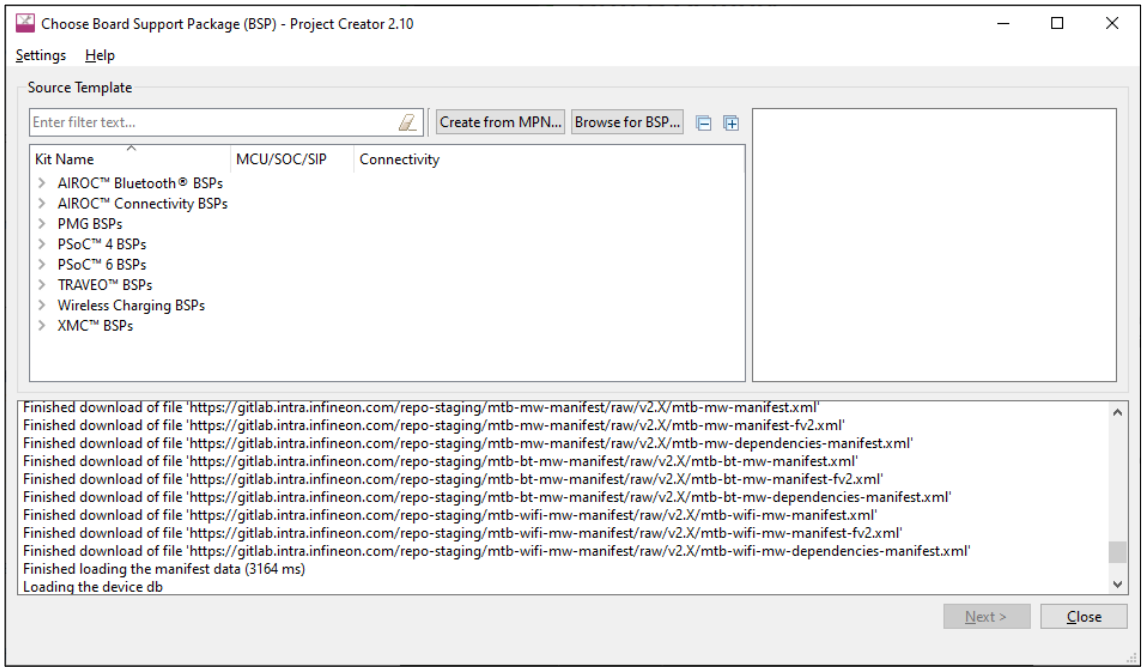
Creating an application includes several steps, as follows:

2.1.1 Step 1: Open Project Creator tool

The ModusToolbox™ Project Creator tool is used to create applications based on code examples and template applications. The tool is provided in GUI form and as a command line interface. For more details, refer to the Project Creator user guide. By default, the tool is installed in the following directory:

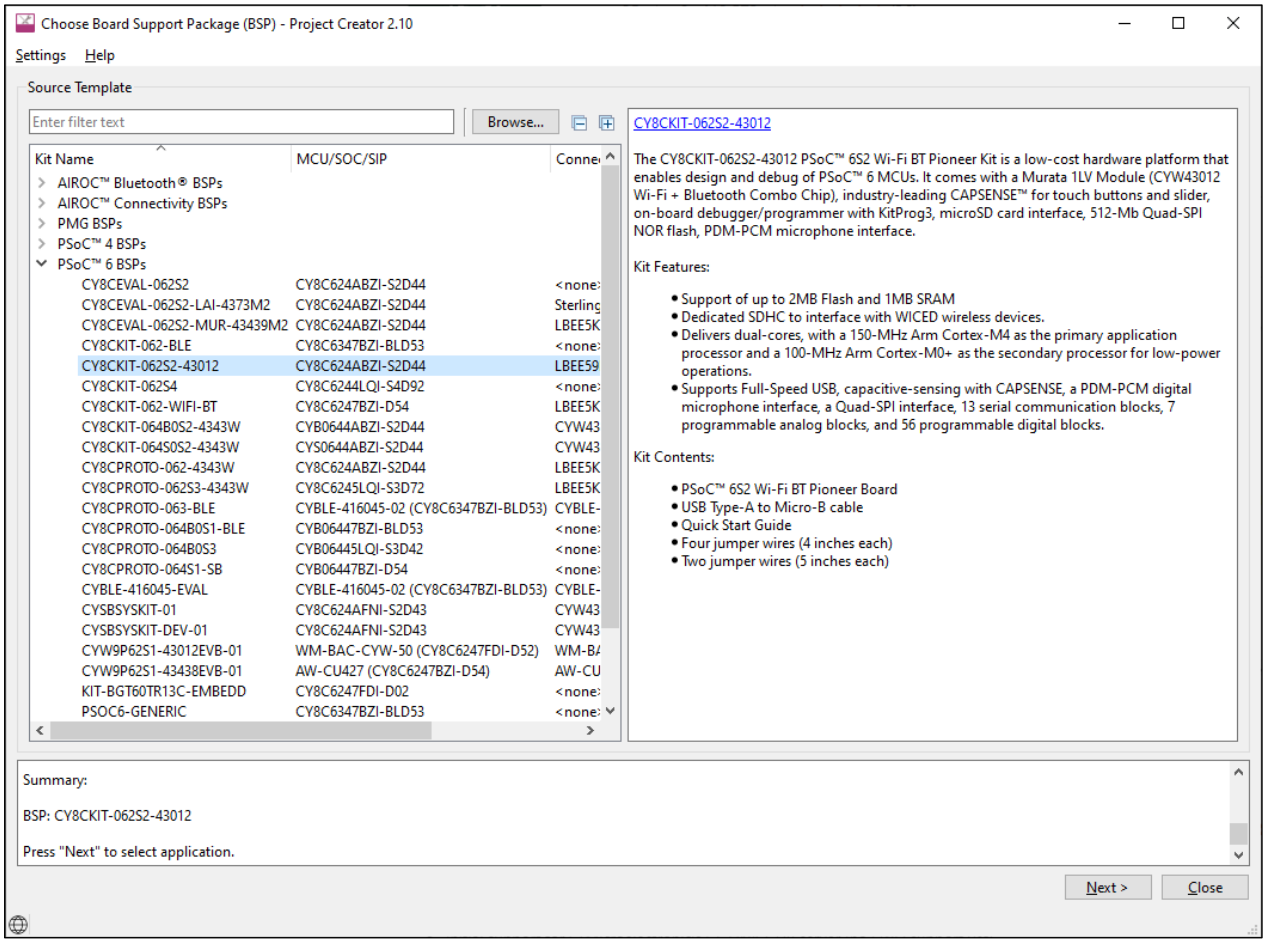
<user_home>/ModusToolbox/tools_<version>/project-creator

Open the Project Creator the tool as applicable for your operating system. You can launch it from the ModusToolbox™ Dashboard, and you can launch it from the VS Code ModusToolbox™ Assistant extension.



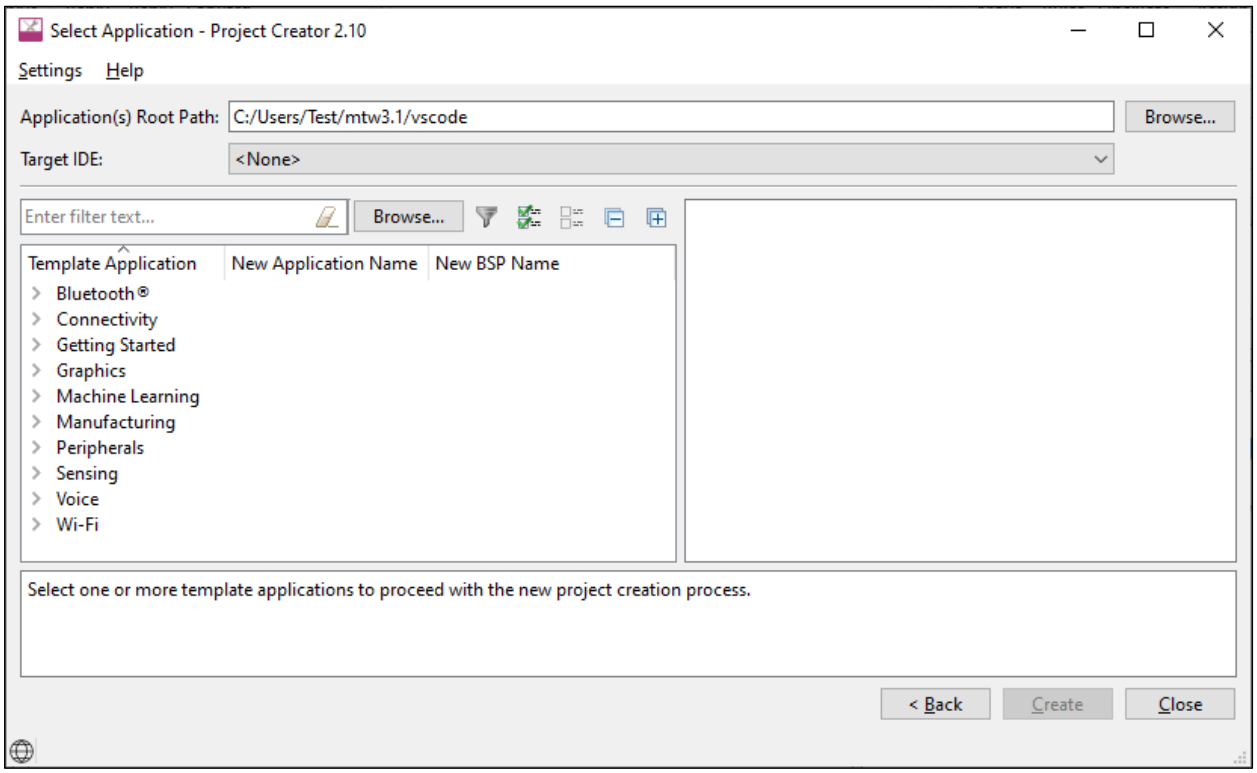
2.1.2 Step 2: Choose Board Support Package (BSP)

When the Project Creator tool opens, expand one of the BSP categories under **Kit Name** and select an appropriate kit; see the description for it on the right. For this example, select the **CY8CKIT-062S2-43012** kit. The following image is an example; the precise list of boards available in this version will reflect the platforms available for development.



2.1.3 Step 3: Select application

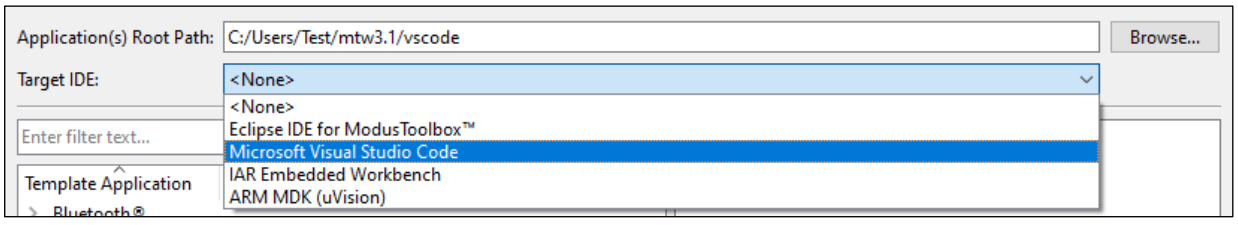
Click **Next >** to open the Select Application page.



This page displays example applications, which demonstrate different features available on the selected BSP. In this case, the CY8CKIT-062S2-43012 provides the PSoC™ 62 MCU and the AIROC™ CYW43012 Wi-Fi & Bluetooth® combo chip. You can create examples for PSoC™ 6 MCU resources such as CAPSENSE™ and QSPI, as well as numerous examples for other capabilities.

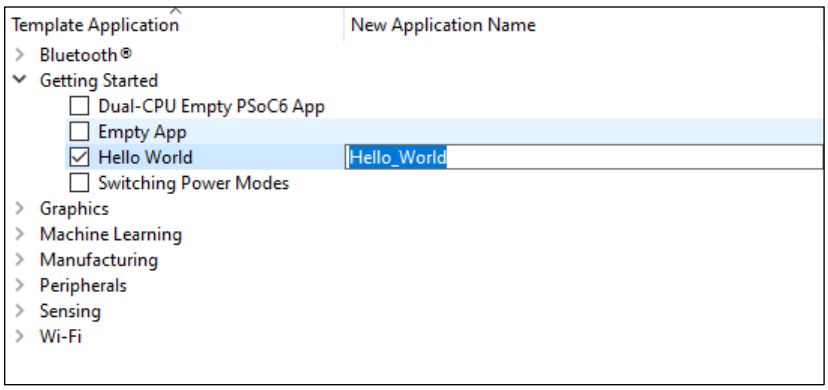
Click **Browse…** next to **Application(s) Root Path** to create or specify a folder where the application will be created.

Pull down the **Target IDE** menu, and select **Microsoft Visual Studio Code**.



Getting Started

Under the **Template Application** column, expand **Getting Started** and select **Hello World** from the list. This example exercises the PSoC™ 6 MCU to blink an LED.



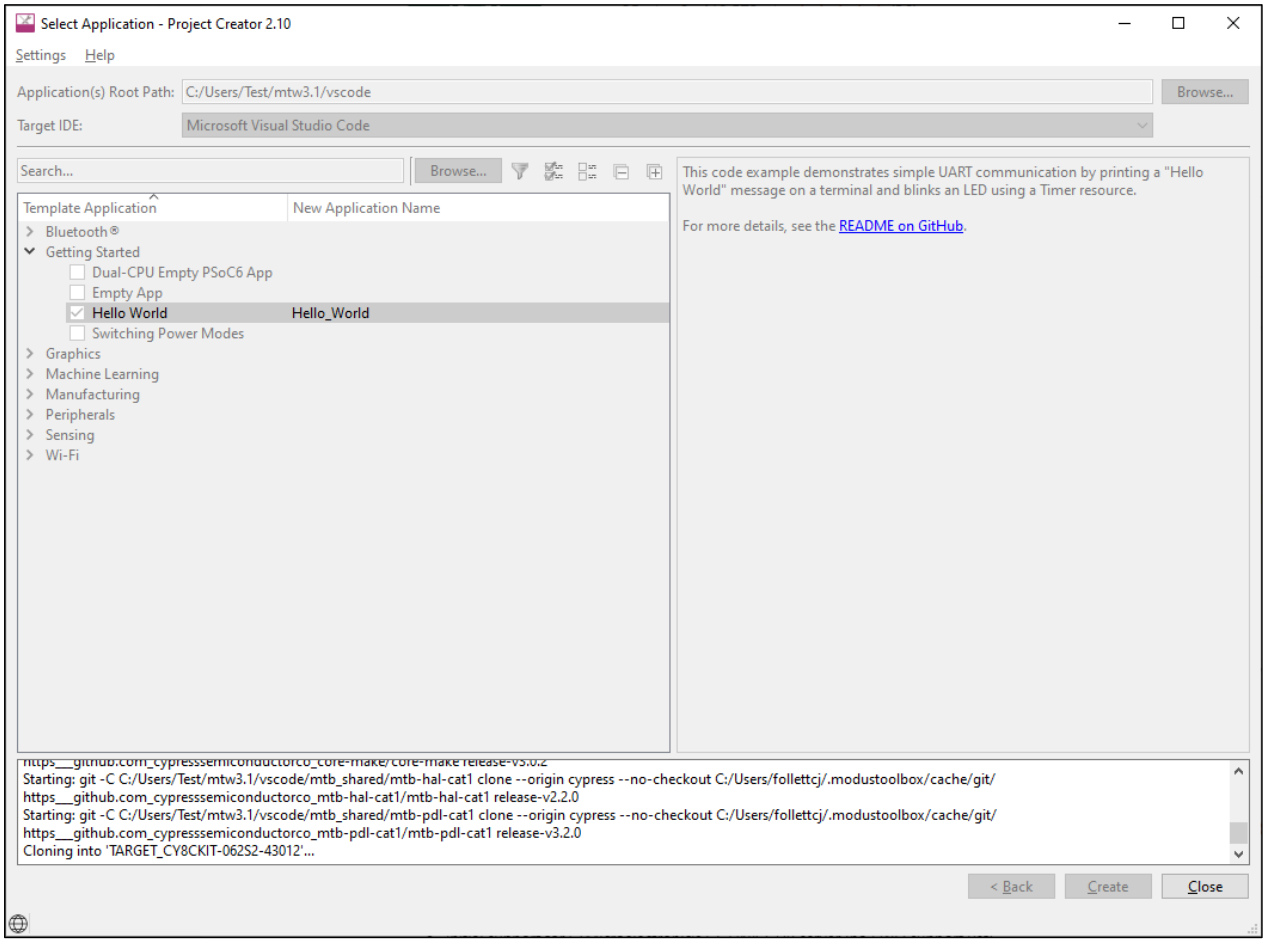
Note: The actual application names available might vary.

Type a name for your application or leave the default name. Do not use spaces in the application name. Also, do not use common illegal characters, such as:

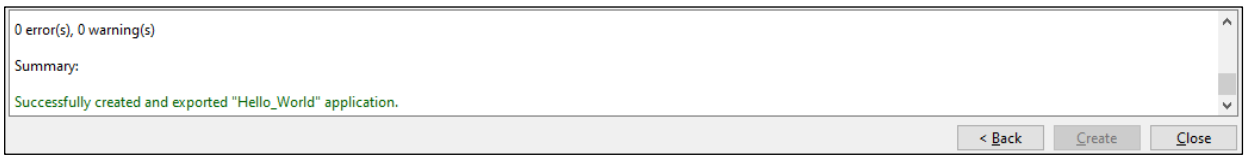
```
* . " ` / \ [ ] : ; | = ,
```

2.1.4 Step 4: Create application

Click **Create** to start creating the application. The tool displays various messages.



When the process completes, a message states that the application was created. Click **Close** to exit the Project Creator tool.

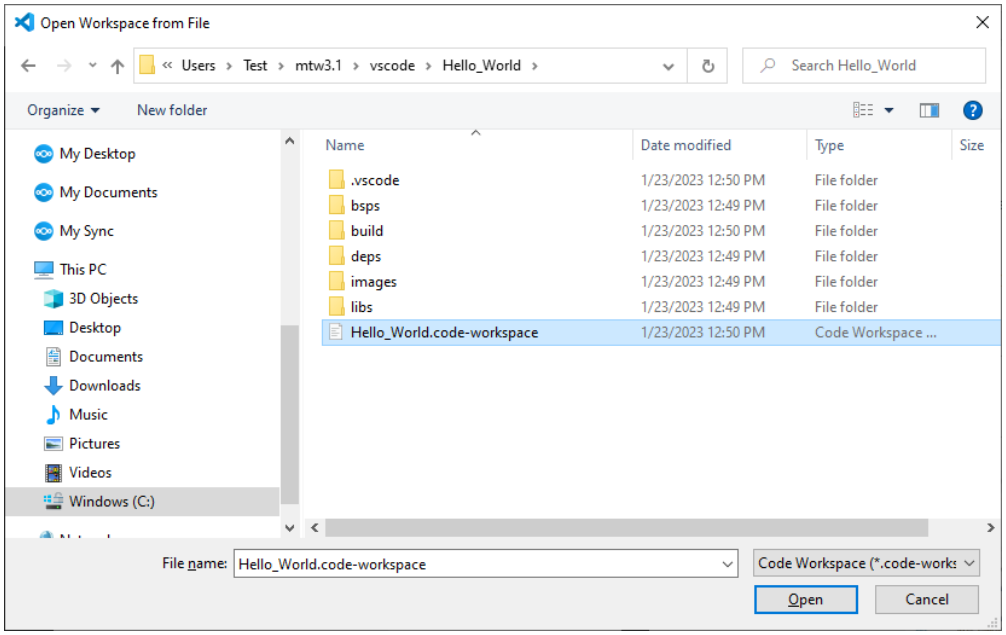


2.2 Export existing application

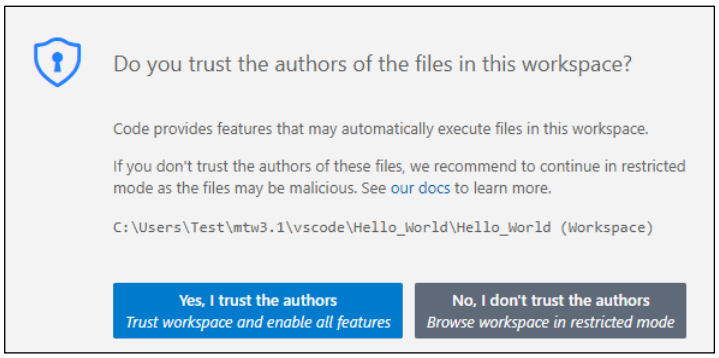
If you have a ModusToolbox™ application that was created for another IDE or for the command line, you can export that application to be used in VS Code. Open a terminal window in the application directory, and run the command `make vscode`.

2.3 Open workspace in VS Code

In VS Code, select **File > Open Workspace from File**, navigate to the location of the application that was just created, select the workspace file, and click **Open**.

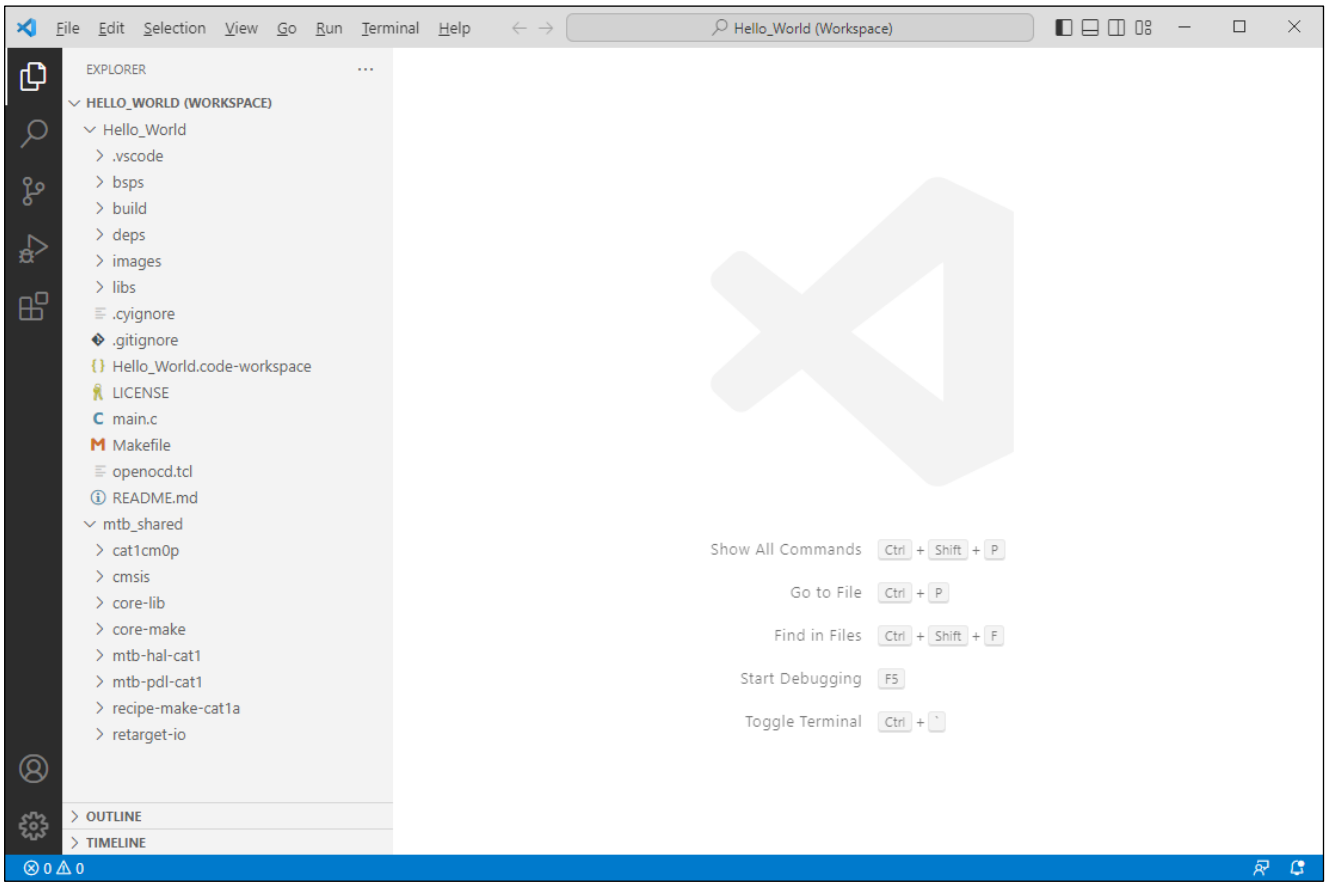


Depending on your settings in VS Code, you may see a message about trusting the authors. If so, click **Yes, I trust the authors**.



Getting Started

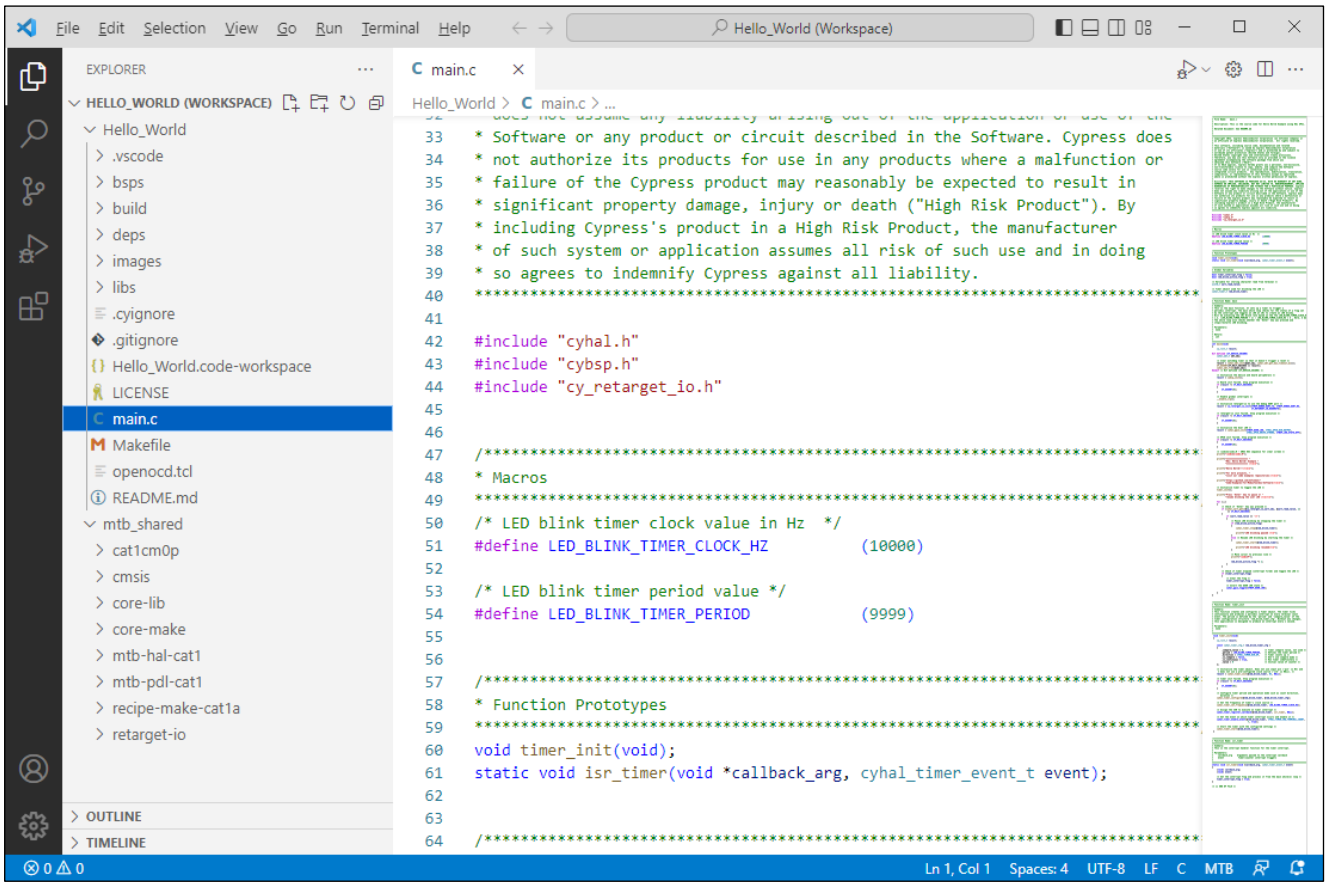
VS Code opens with the Hello_World workspace in the EXPLORER view.



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.

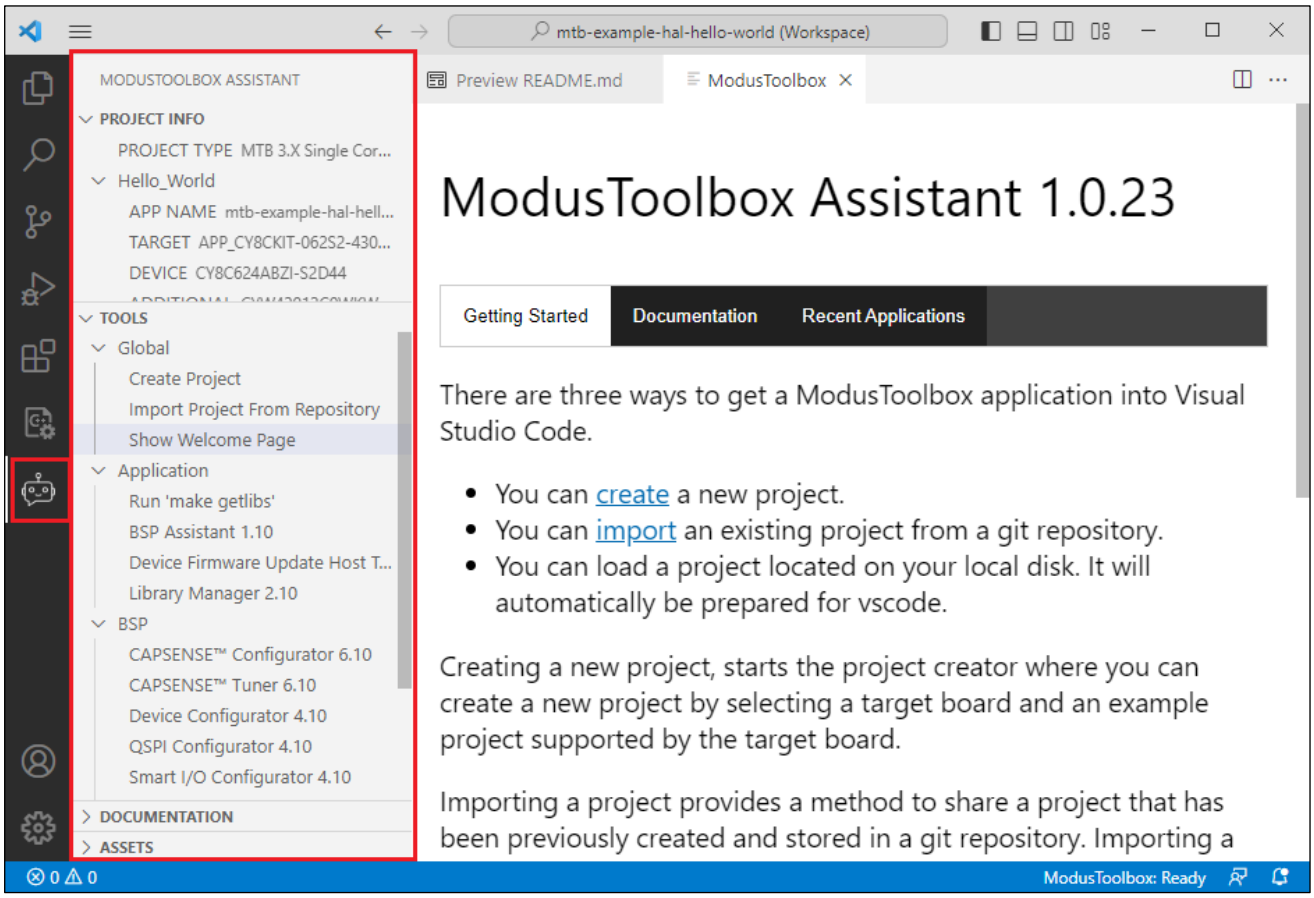


As you type into the file, a dot will appear in the file's tab to indicate changes were made. The file icon will also indicate that there are unsaved changes.

4 Using ModusToolbox™ tools

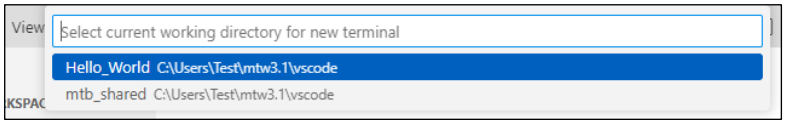
4.1 ModusToolbox™ Assistant extension

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



4.2 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.

4.2.1 Library Manager

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

```
make library-manager
```

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

4.2.2 BSP Assistant

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

```
make bsp-assistant
```

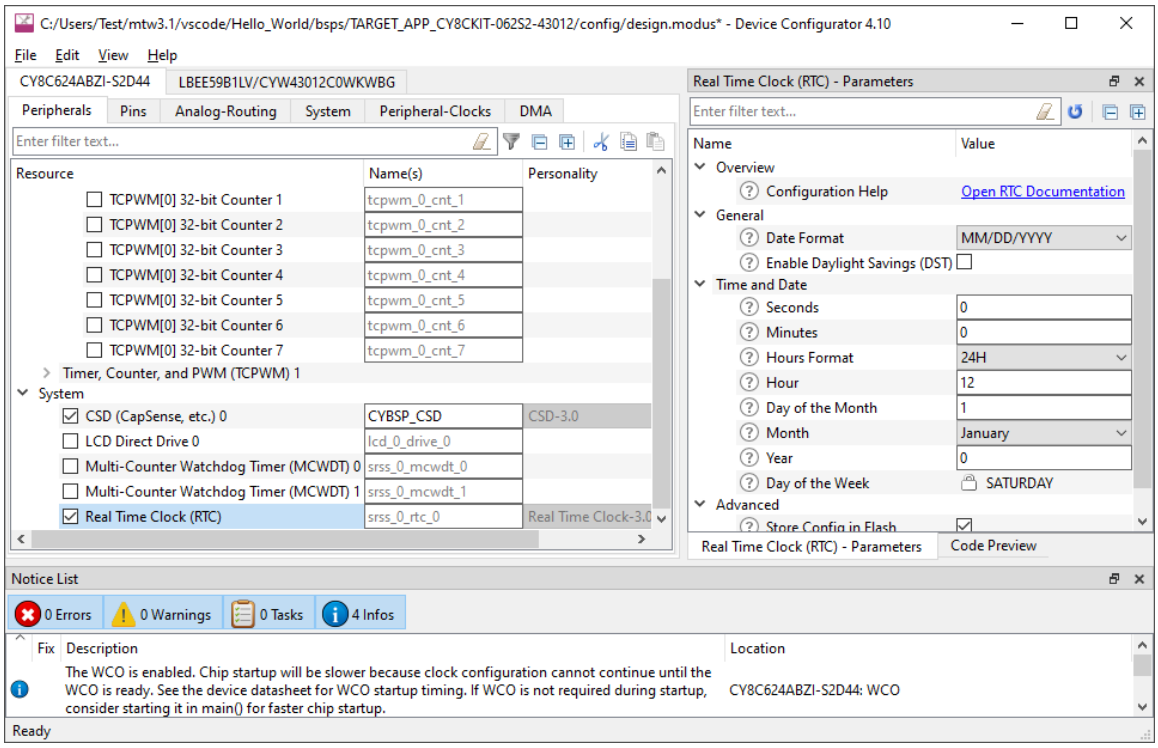
Refer to the ModusToolbox™ BSP Assistant user guide for details about that tool.

4.2.3 Device Configrator

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.

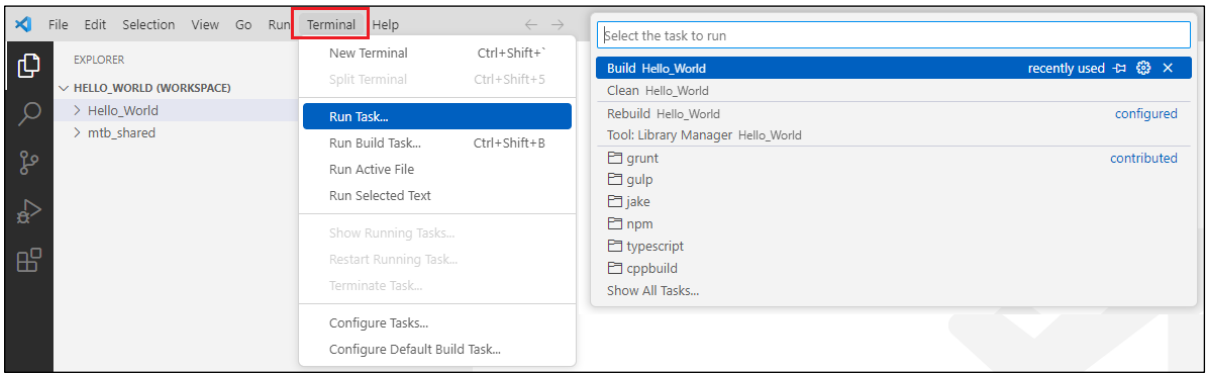


Note: The Device Configurator cannot be used to open Library Configurators, such as Bluetooth®.

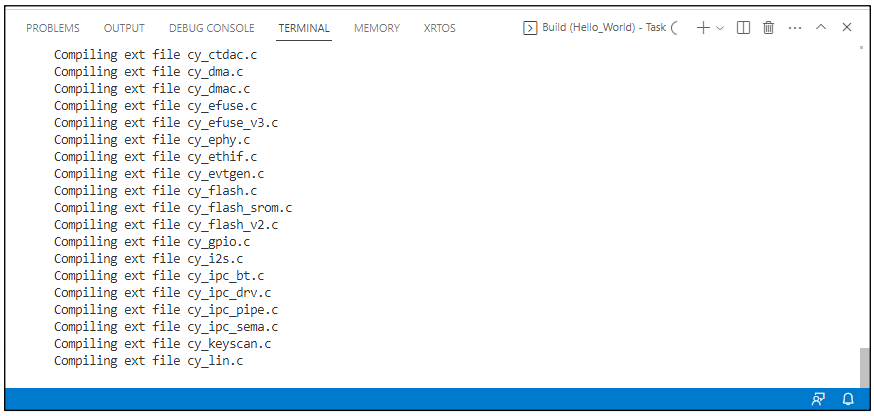
5 Build the Application

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**.



Build information will display in the Terminal.

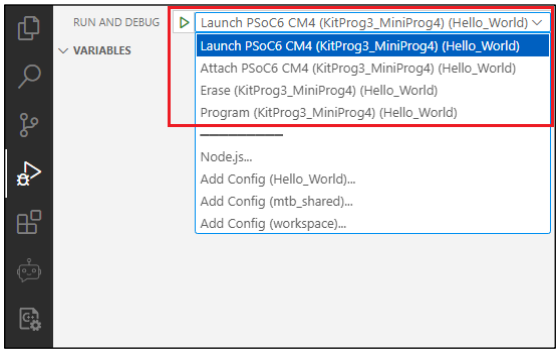


The build should complete successfully with messages similar to the following:

```
================================================================================
= Build complete =
================================================================================

Calculating memory consumption: CY8C624ABZI-S2D44 GCC_ARM

  ------------------------------------------------
 | Section Name         | Address     | Size      |
  ------------------------------------------------
 | .cy_m0p_image        | 0x10000000  | 6224      |
 | .text                | 0x10002000  | 43664     |
 | .ARM.exidx           | 0x1000ca90  | 8         |
 | .copy.table          | 0x1000ca98  | 24        |
 | .zero.table          | 0x1000cab0  | 8         |
 | .data                | 0x080022e0  | 1592      |
 | .cy_sharedmem        | 0x08002918  | 8         |
 | .noinit              | 0x08002920  | 224       |
 | .bss                 | 0x08002a00  | 1532      |
 | .heap                | 0x08003000  | 1030144   |
  ------------------------------------------------

 Total Internal Flash (Available)          2097152
 Total Internal Flash (Utilized)           53504


 Terminal will be reused by tasks, press any key to close it.
```

6 Program/debug using KitProg3/MiniProg4

Most PSoC™-based kits use KitProg3/MiniProg4 as the default programmer/debugger. The VS Code GUI shows these configurations by default:



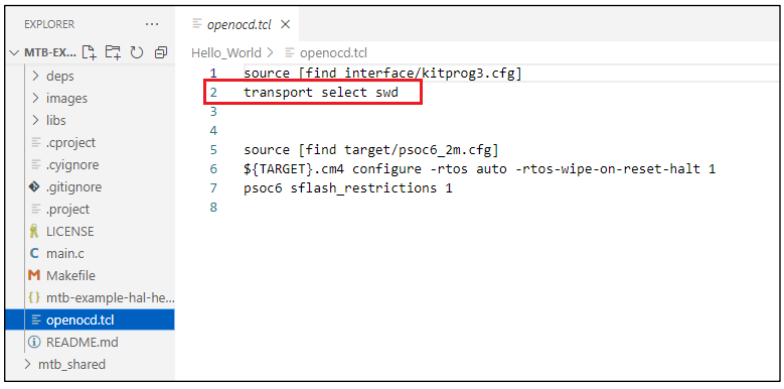
- **Launch:** This builds the entire application on both cores, programs all the device's memories, and then starts a Cortex-M4 debugging session.
- **Attach:** This starts a Cortex-M4 debugging session attaching to a running PSoC™ 6 target without programming or reset.
- **Erase:** This erases all internal memories.
- **Program:** This builds the entire application on both cores, programs all the device's memories, and then runs the program.

6.1 Connect the Kit

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

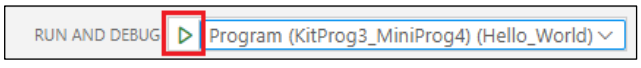
6.2 Changing programming interface SWD/JTAG

To change the target interface for KitProg3_MiniProg4, edit the *openocd.tcl* file in the project root directory. This file contains the OpenOCD command allowing you to select the debugging interface: "transport select". Set this to either swd or jtag.



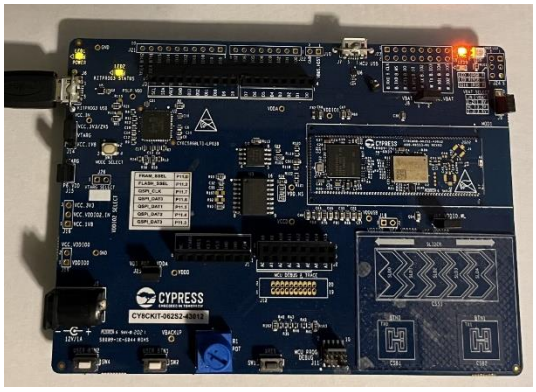
6.3 Program

Select the **Run And Debug** icon in the VS Code Activity Bar, select the **Program (KitProg3_MiniProg4)** Launch Configuration, and click **Start Debugging** icon or press **F5**.



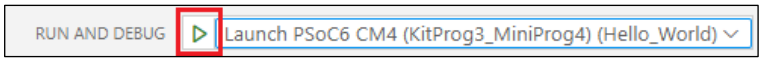
Program/debug using KitProg3/MiniProg4

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.

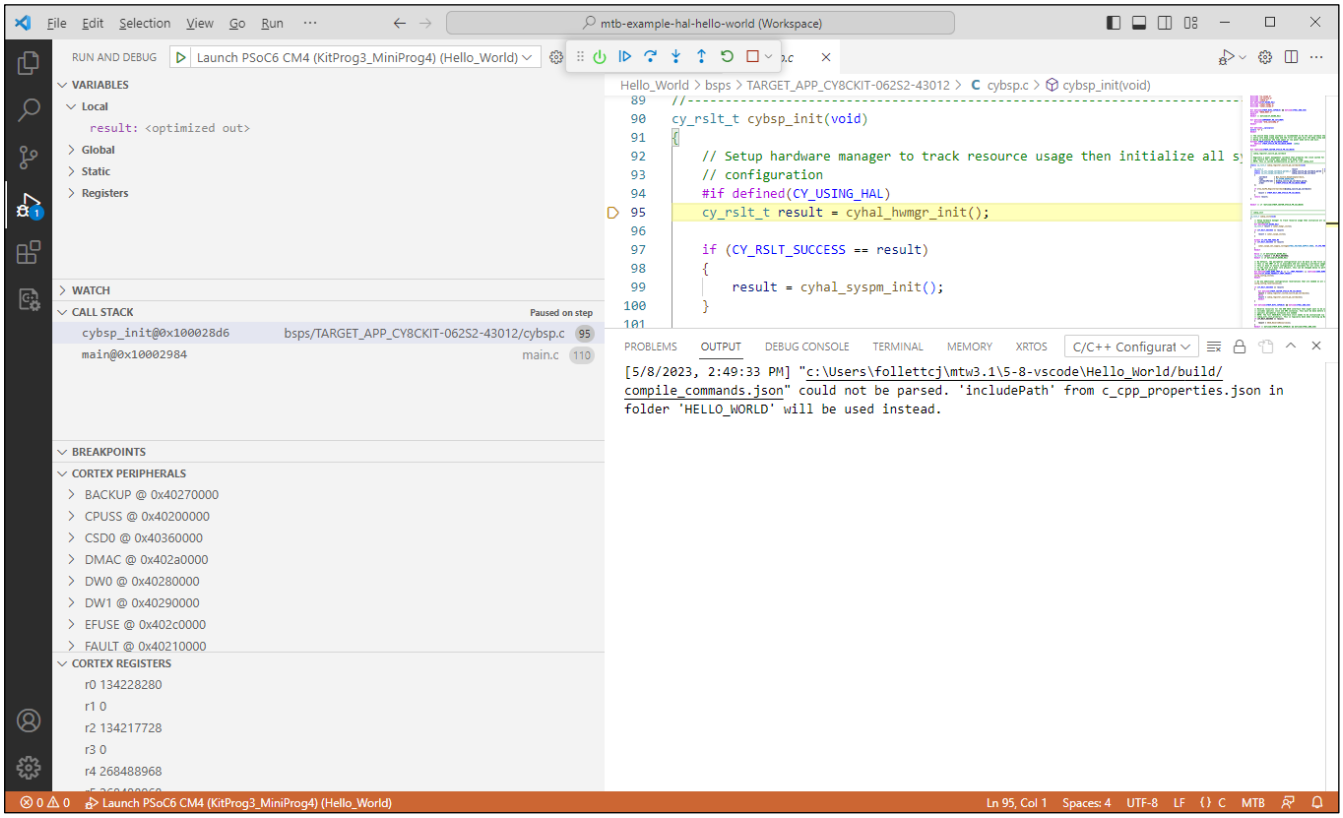


6.4 Debug

Select the **Run And Debug** icon in the VS Code **Activity Bar**, select the **Launch PSoC6 CM4 (KitProg3_Miniprog4)** 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.



7 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.

7.1 Configure J-Link programmer/debugger settings

1. Open your ModusToolbox™ single-core application's *Makefile*, or multi-core application's *common.mk* file, and enter the following variable:

   ```
   BSP_PROGRAM_INTERFACE=JLink
   ```

2. If you install J-Link software in a non-default location, or if you intend to use make commands, such as `make program` and `make qprogram`, also enter the following variable:

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

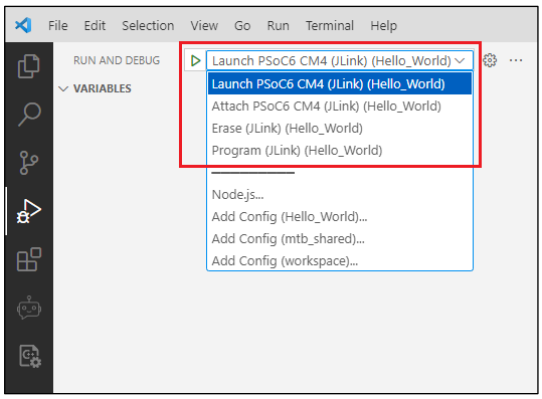
Note: *If you want all projects/applications for the BSP to use the J-Link programmer/debugger, you can specify the `BSP_PROGRAM_INTERFACE` and `MTB_JLINK_DIR` variables in the bsp.mk file instead.*

3. Save the *Makefile*.

4. In a bash Terminal run:

   ```
   make vscode
   ```

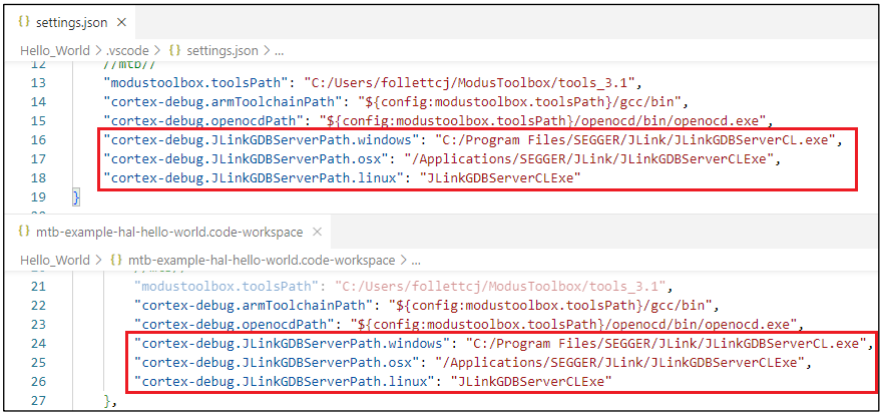
   When the command completes, J-Link configurations will be shown. These are the same configurations described in Program/debug using KitProg3/MiniProg4, but applicable to J-Link.



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"
   ```

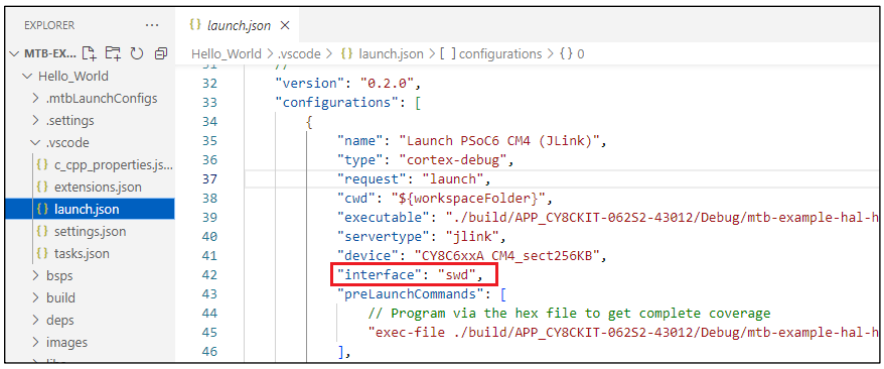


7.2 Connect the Kit

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

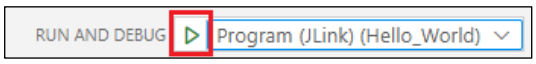
7.3 Changing programming interface SWD/JTAG

To change the target interface for J-Link, edit the *launch.json* file to specify the applicable "interface": swd or jtag. Do this for all the applicable configurations (e.g., Launch, Attach, Erase, etc.)



7.4 Program

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

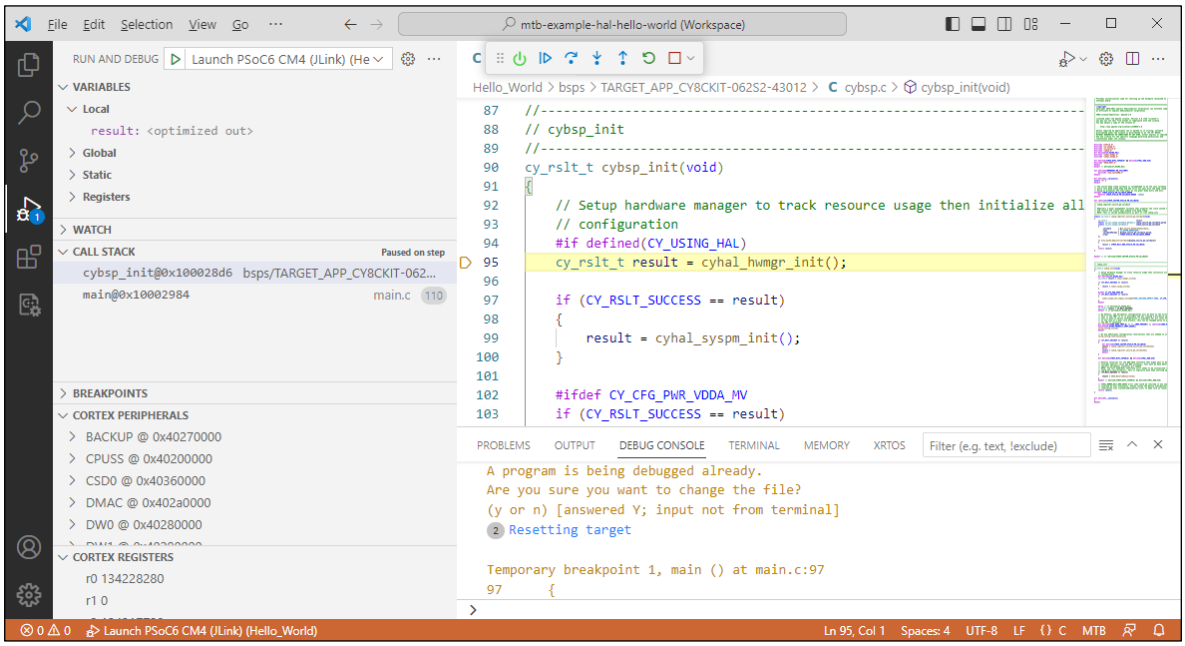


If needed, VS Code builds the application and messages display in the Terminal. If the build is successful, device programming starts immediately. If there are build errors, then error messages will indicate as such. When programming completes successfully, the LED will start blinking.

7.5 Debug

Click the **Run and Debug** icon, select **Launch PSoC6 CM4 (JLink)** config, and click **Start Debugging** icon or press **F5**.

Program/debug using J-Link



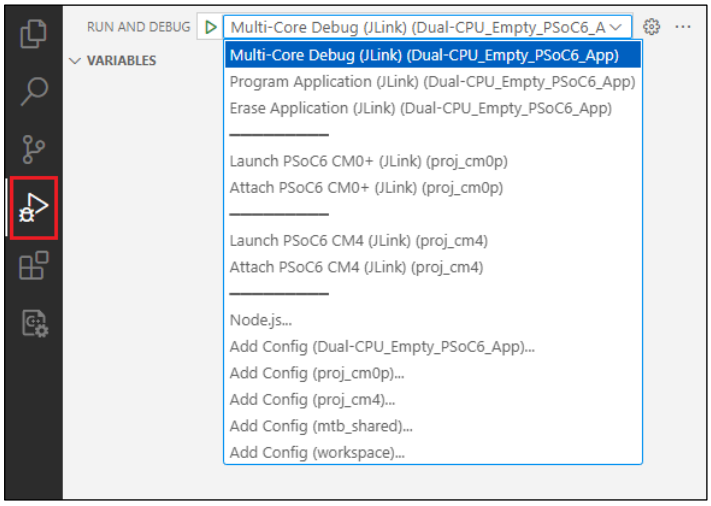
8 Multi-core debugging

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)

8.1 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
- Program Application: downloads combined hex file into the flash
- Erase Application: erases all internal memory banks
- Launch <device>: launches debug session on the chosen core
- Attach <device>: attaches to the running core

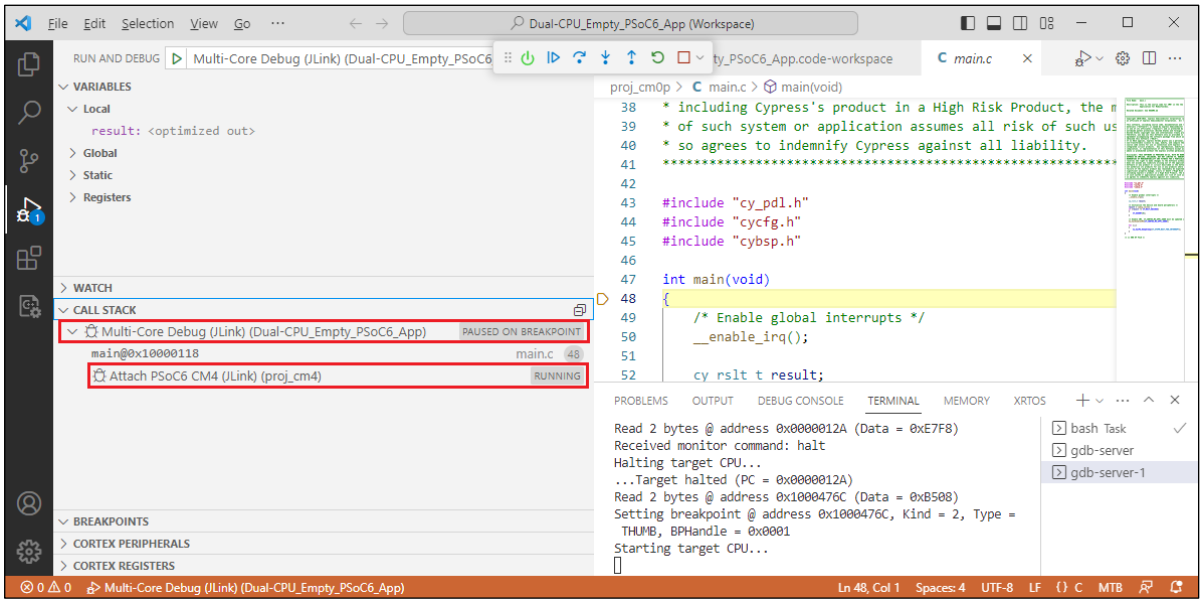
8.2 Changing programming interface SWD/JTAG

For multi-core debugging, there are more than one separate projects, and each project defines its own debugging interface. If changing from SWD to JTAG, or vice versa, make sure to set the same interface for all projects.

- For **KitProg3_MiniProg4**, set the same interface in all *openocd.tcl* files (see Changing programming interface SWD/JTAG for KitProg3/MiniProg4). There should be one *openocd.tcl* file per project and one for the application located at the application root directory.
- For **J-Link**, set the required interface for "interface" property (see Changing programming interface SWD/JTAG for J-Link). The multi-core debugging configuration also involves launching Attach configurations; define the same interface for each of them.

8.3 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.*

Revision history

Revision	Date	Description
**	2023-05-16	New document.

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

Edition 2023-05-16

Published by

Infineon Technologies AG

81726 Munich, Germany

Do you have a question about this document?

Email: erratum@infineon.com

Document reference

002-37543 Rev. **

Important notice

The information given in this document shall in no event be regarded as a guarantee of conditions or characteristics ("Beschaffenheitsgarantie")

With respect to any examples, hints or any typical values stated herein and/or any information regarding the application of the product, Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind, including without limitation warranties of non-infringement of intellectual property rights of any third party.

In addition, any information given in this document is subject to customer's compliance with its obligations stated in this document and any applicable legal requirements, norms and standards concerning customer's products and any use of the product of Infineon Technologies in customer's applications.

The data contained in this document is exclusively intended for technically trained staff. It is the responsibility of customer's technical departments to evaluate the suitability of the product for the intended application and the completeness of the product information given in this document with respect to such application.

Warnings

Due to technical requirements products may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies office.

Except as otherwise explicitly approved by Infineon Technologies in a written document signed by authorized representatives of Infineon Technologies, Infineon Technologies' products may not be used in any applications where a failure of the product or any consequences of the use thereof can reasonably be expected to result in personal injury.