Visual Studio Code for ModusToolbox™ user guide

ModusToolbox™ tools package version 3.7.0

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.

A newer version of this document may be available on the web here.

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

1 Download/install software

1.1 ModusToolbox™ software

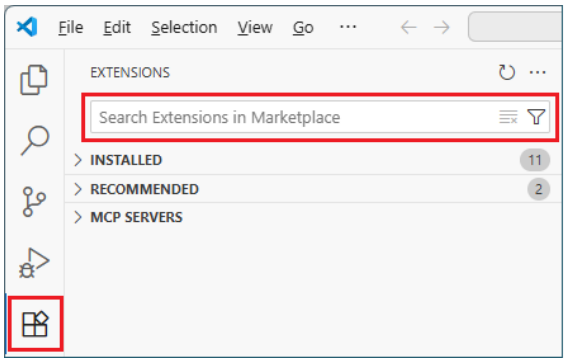
Download the ModusToolbox™ Setup program from https://softwaretools.infineon.com/tools/com.ifx.tb.tool.modustoolboxsetup. Refer to the instructions in the ModusToolbox™ software installation guide for how to install the necessary ModusToolbox™ tools and packages.

1.2 VS Code

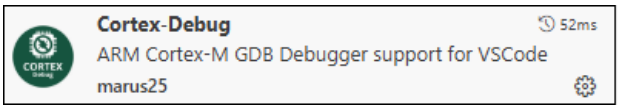
The ModusToolbox™ tools package includes various tools to create and manage applications, but it does not include VS Code. If you do not already have VS Code installed on your computer, you can download it from the website:

https://code.visualstudio.com/

After opening an application in VS Code, it will recommend several extensions. The Cortex-Debug extension is required for build and debug. Other extensions such as Arm® Assembly, ModusToolbox™ Assistant, and clangd improve the development and debug experience. Use the search under Extensions to locate and install them:

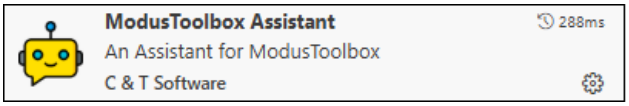


- Cortex-Debug

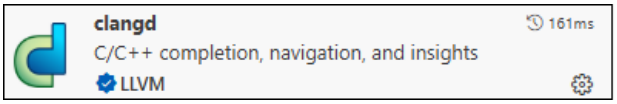


- Arm® Assembly

- ModusToolbox™ Assistant



- clangd



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
- Export existing application
- Open workspace in VS Code

2.1 Create new application

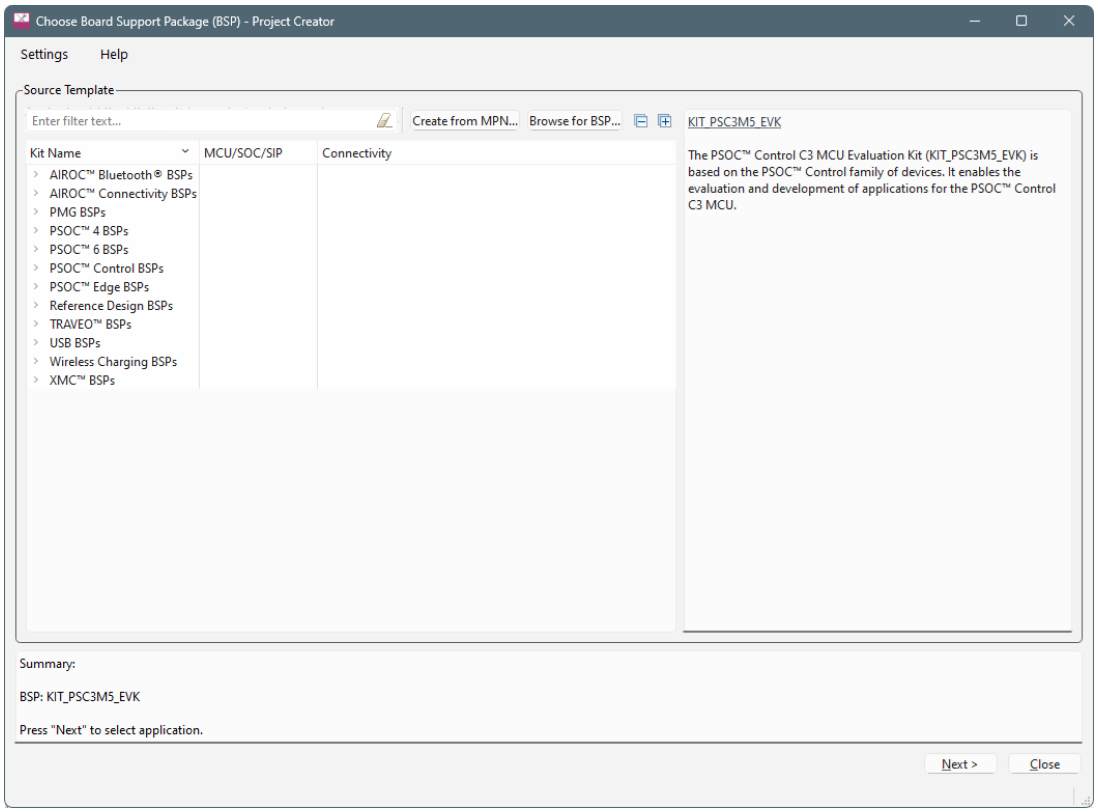
Creating an application includes several steps, as follows:

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:

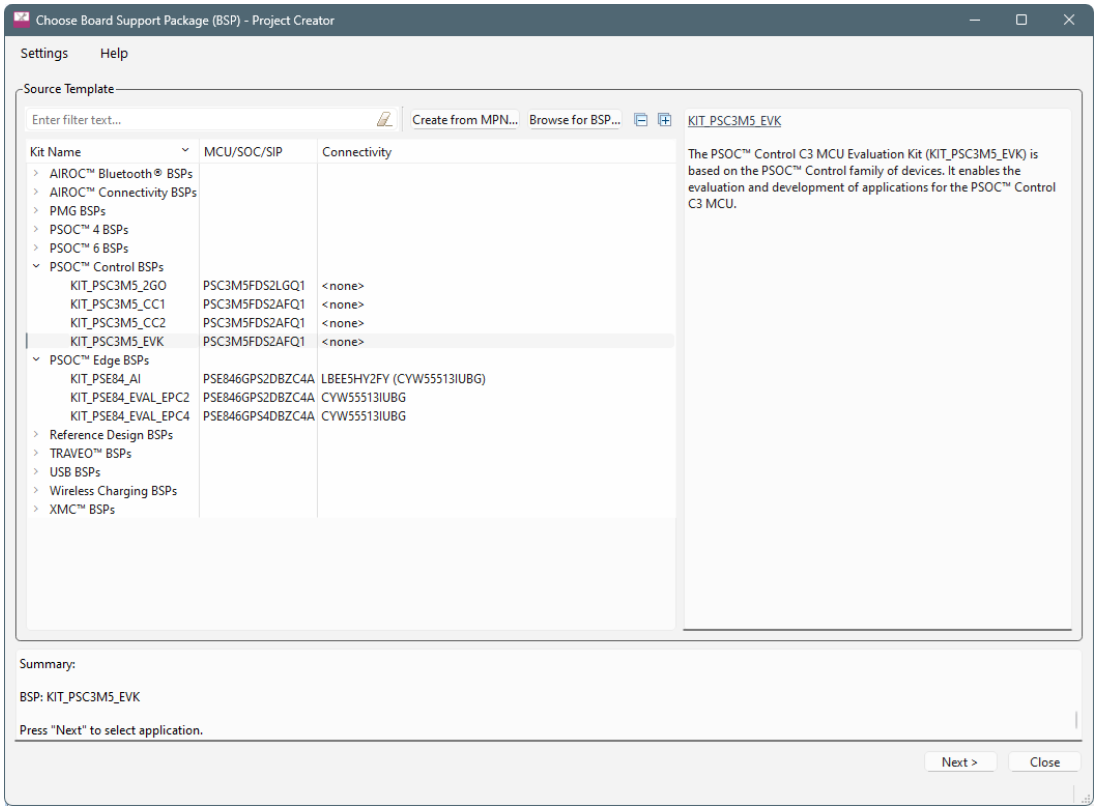
<user_home>/ModusToolbox/tools_<version>/project-creator

Open the Project Creator tool as applicable for your operating system. You can launch it from the ModusToolbox™ Dashboard or 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 **KIT-PSC3M5_EVK** kit. The following image is an example; the list of boards available in this version will reflect the platforms available for development.



2.1.3 Step 3: Select application

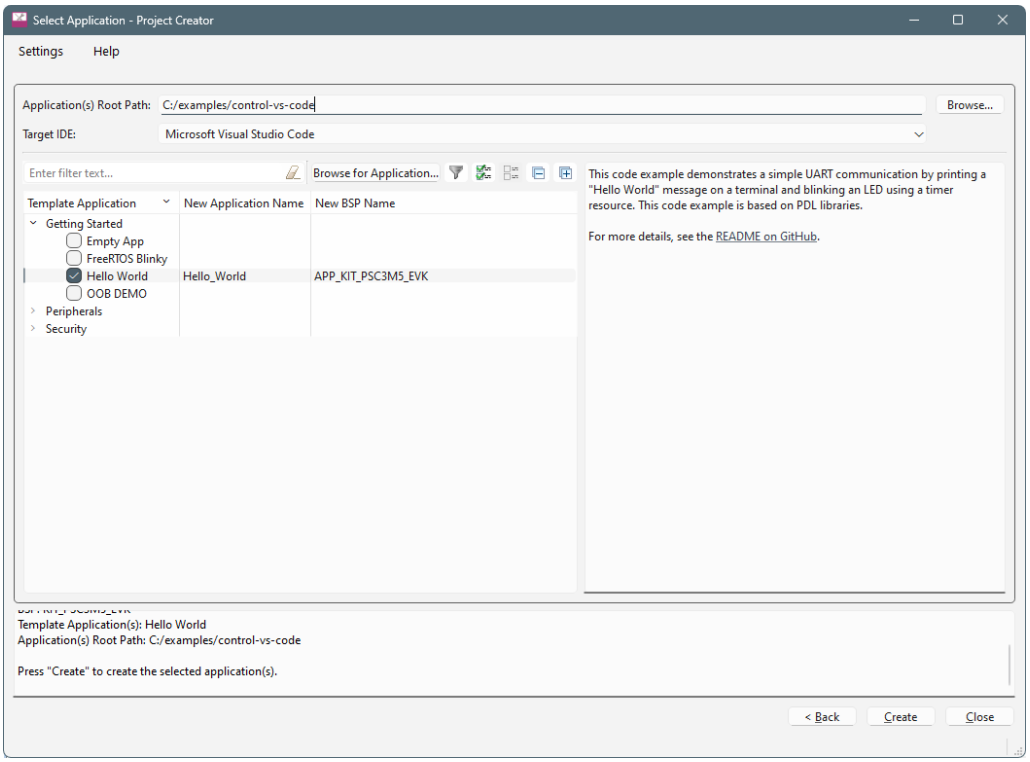
To select an application:

1. 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 KIT-PSC3M5_EVK provides the PSOC Control MCU. You can create examples for various peripherals and security.

2. Click **Browse…** next to **Application(s) Root Path** to create or specify a folder where the application will be created.
3. Pull down the **Target IDE** menu and select Microsoft Visual Studio Code.

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



***Note**: The actual application names available might vary.*

5. 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:

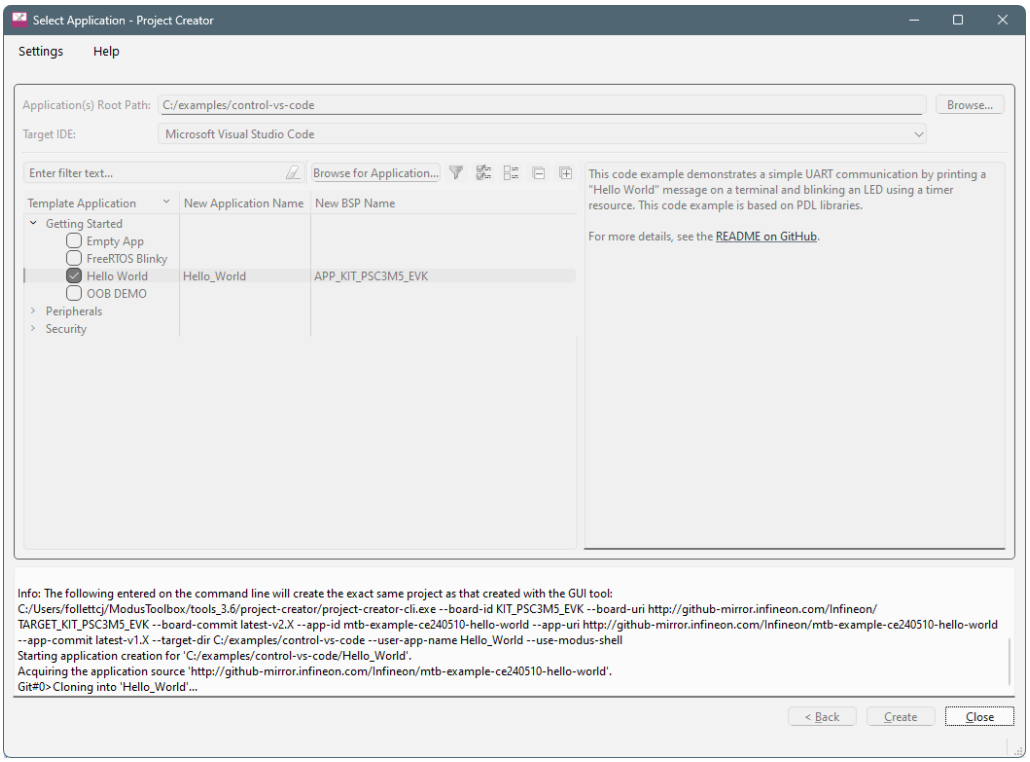
```
* . " ‘ / \ [ ] : ; | = ,
```

2.1.4 Step 4: Create application

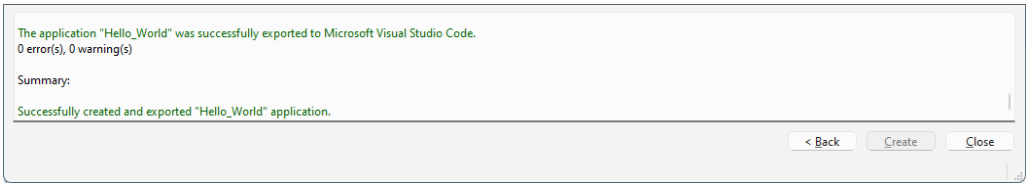
To create the application:

1. Click **Create**.

The tool displays various messages.



When the process completes, a message states that the application was created.



2. Click **Close** to exit the Project Creator tool.

***Note**: If you opened the Project Creator tool using the VS Code ModusToolbox Assistant extension, the tool will close automatically upon successful completion.*

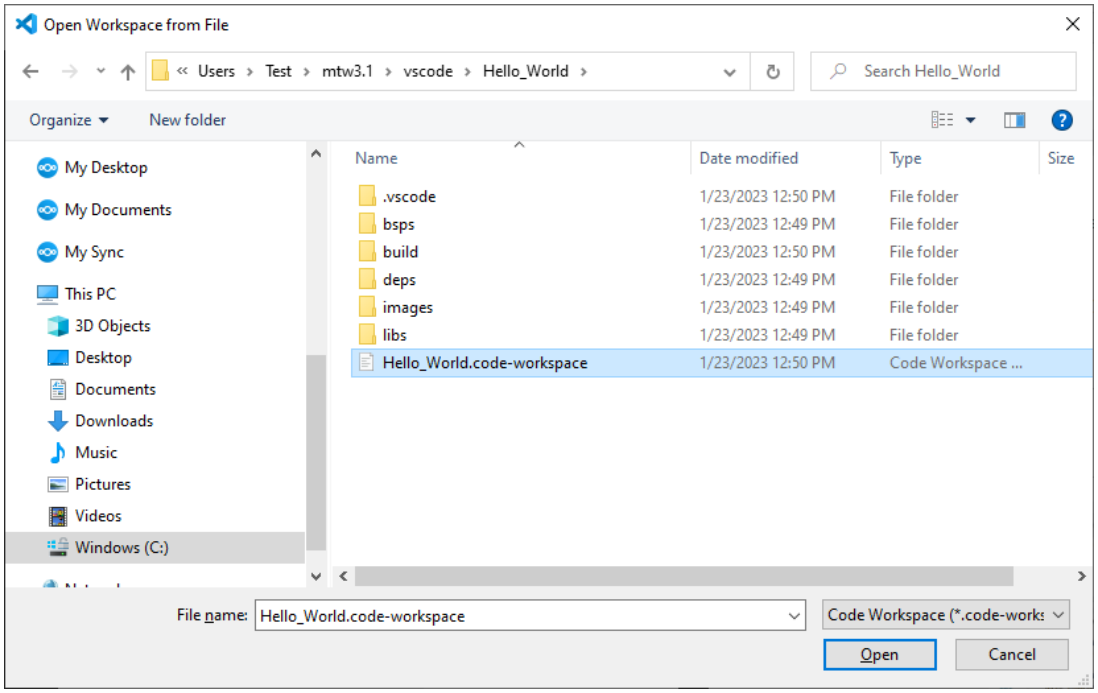
2.2 Export existing application

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. Open a terminal window in the application directory and run the command `make vscode`.

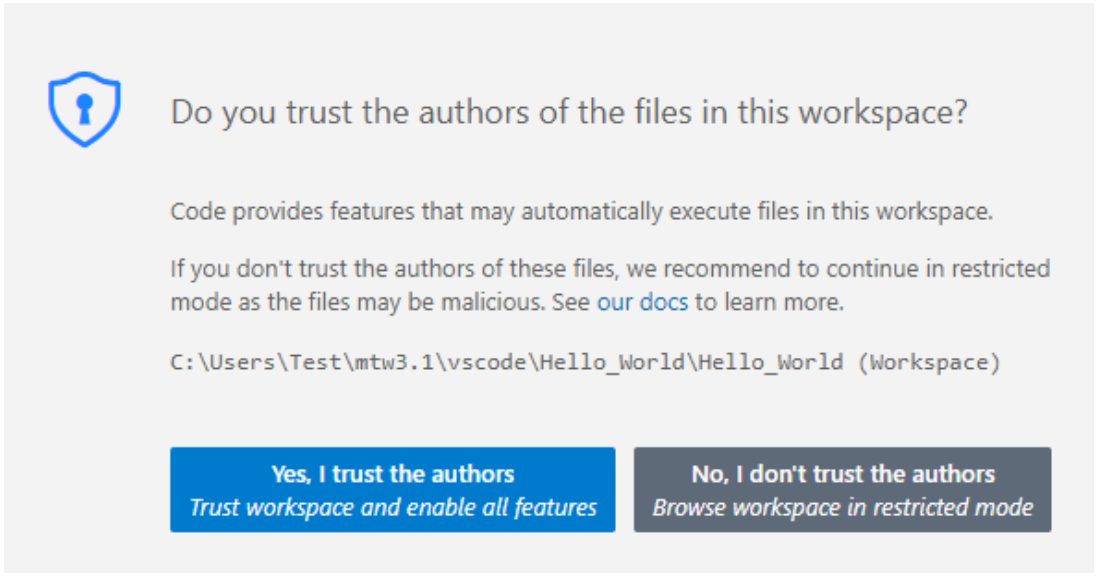
2.3 Open workspace in VS Code

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**.

2 Getting Started

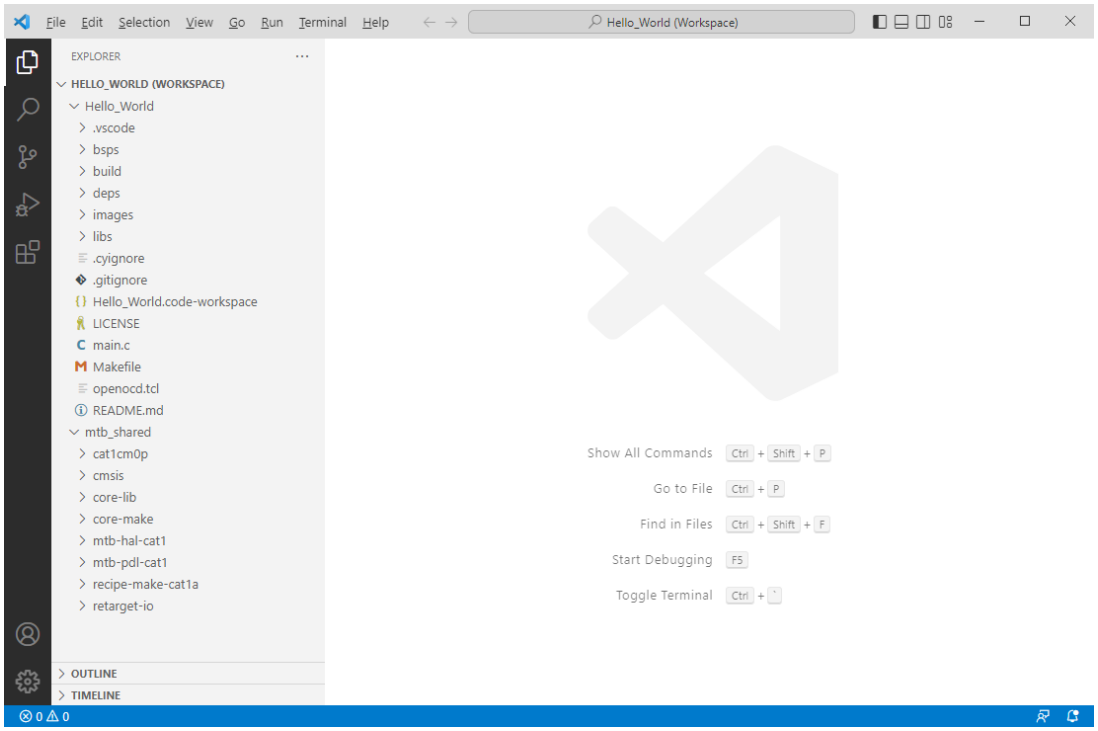


Depending on your settings in VS Code, you may see a message about trusting the authors. If so, click **Yes, I trust the authors**.



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

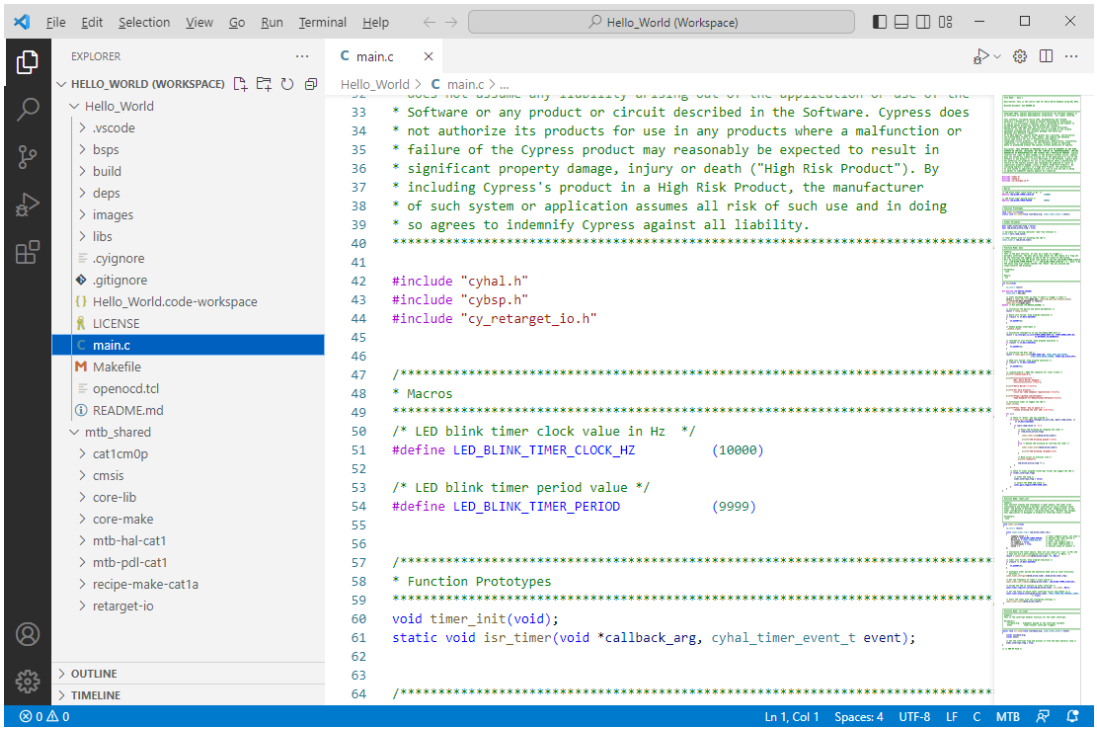
2 Getting Started



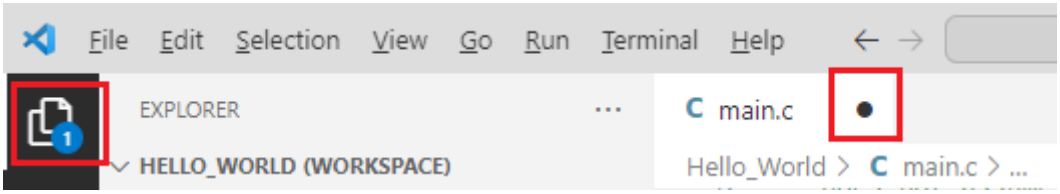
3 Add/modify application code

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.



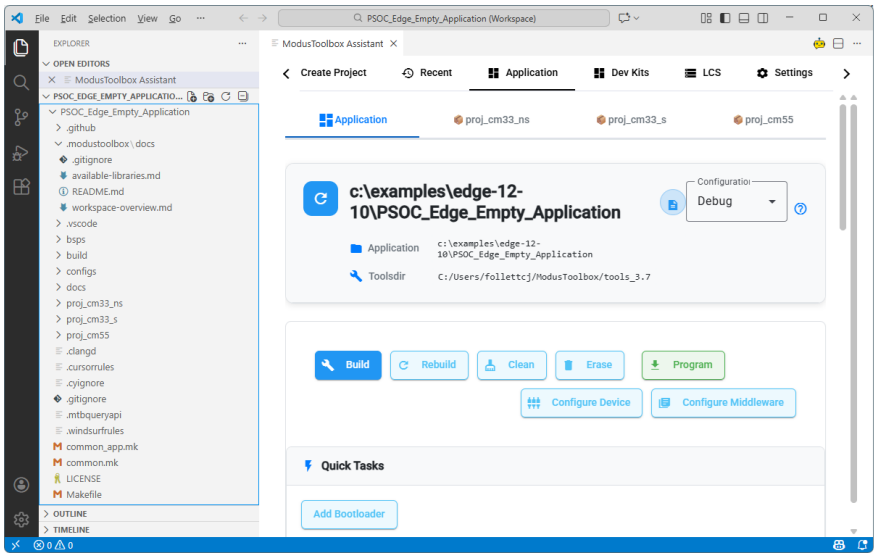
As you type into the file, a dot will appear in the file's tab to indicate changes were made. The file icon will also indicate that there are unsaved changes.



4 Using ModusToolbox™ tools

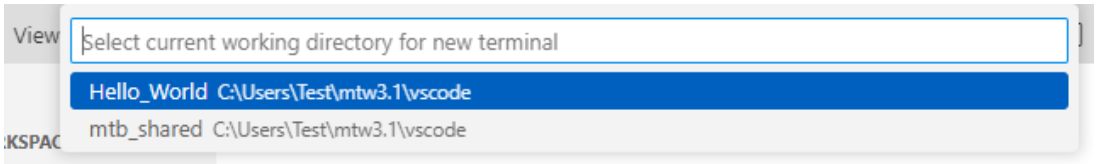
4.1 ModusToolbox™ Assistant extension

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 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 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
```

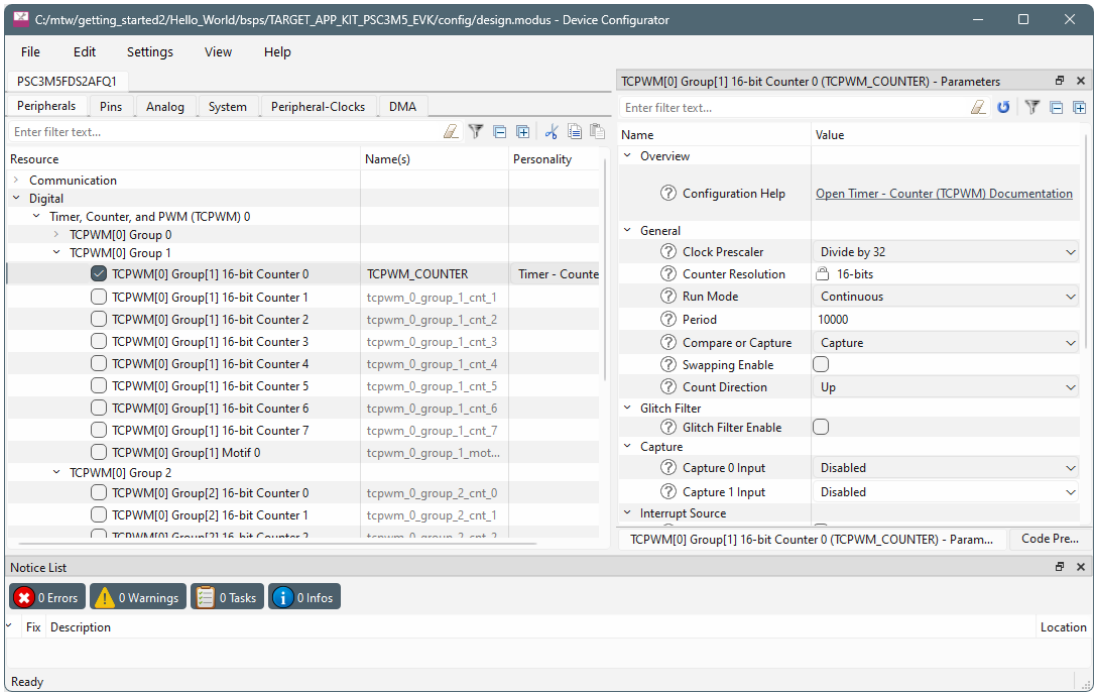
Refer to the BSP Assistant user guide for details about that tool.

4.2.3 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.

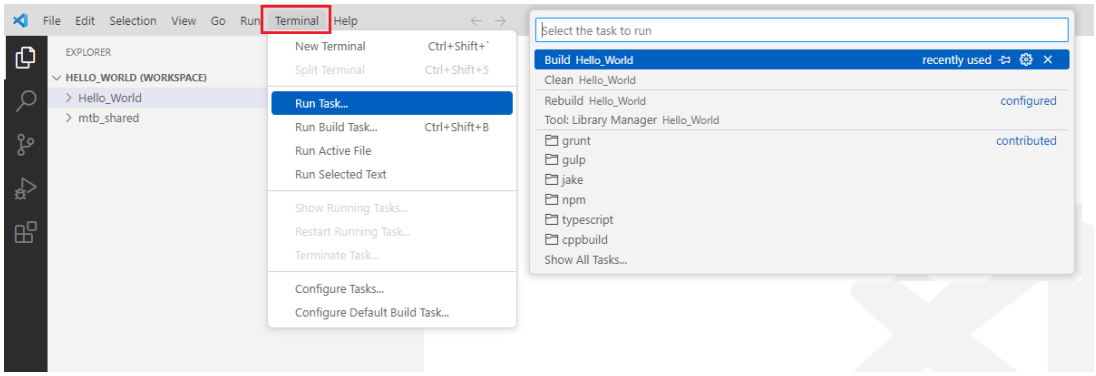


***Note**: The Device Configurator cannot be used to open Library Configurators, such as Bluetooth®.*

5 Build the Application

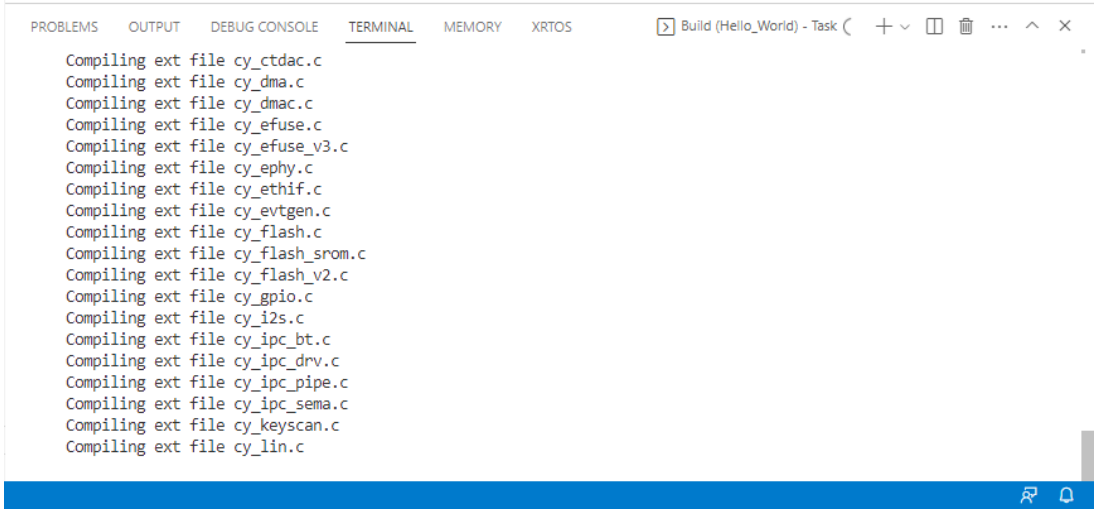
Building the application is not specifically required, because building will be performed as part of the programming and debugging process. However, if you are running VS Code without any hardware attached you may wish to build your application to ensure all the code is correct.

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



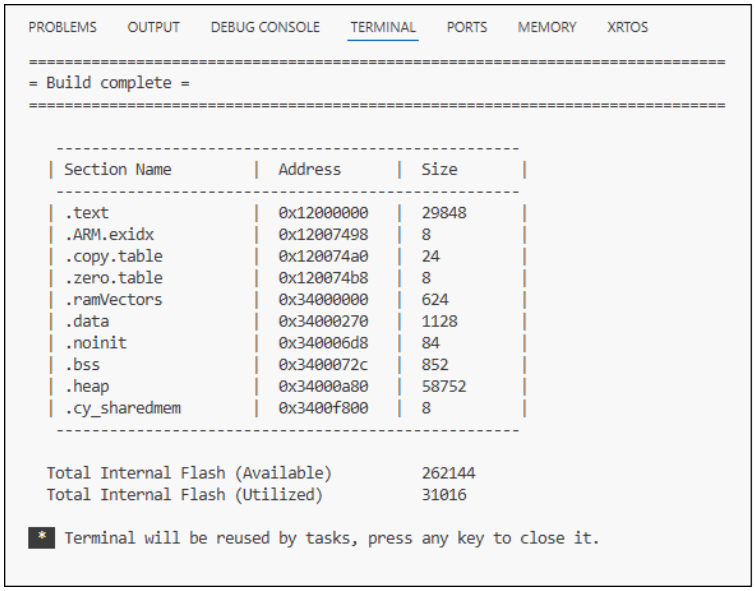
***Note**: If you have the ModusToolbox™ Assistant extension installed, you can build using the option under **Quick Links**.*

Build information will display in the Terminal.



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

5 Build the Application



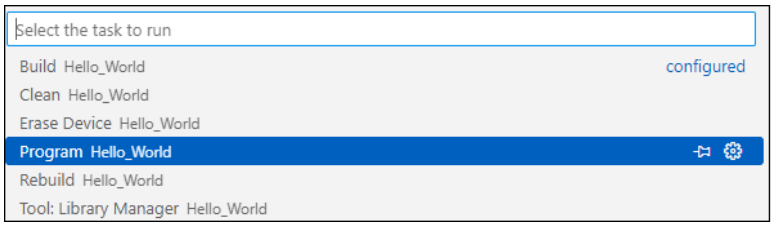
6 Program/debug common

The VS Code GUI shows these launch configurations by default:

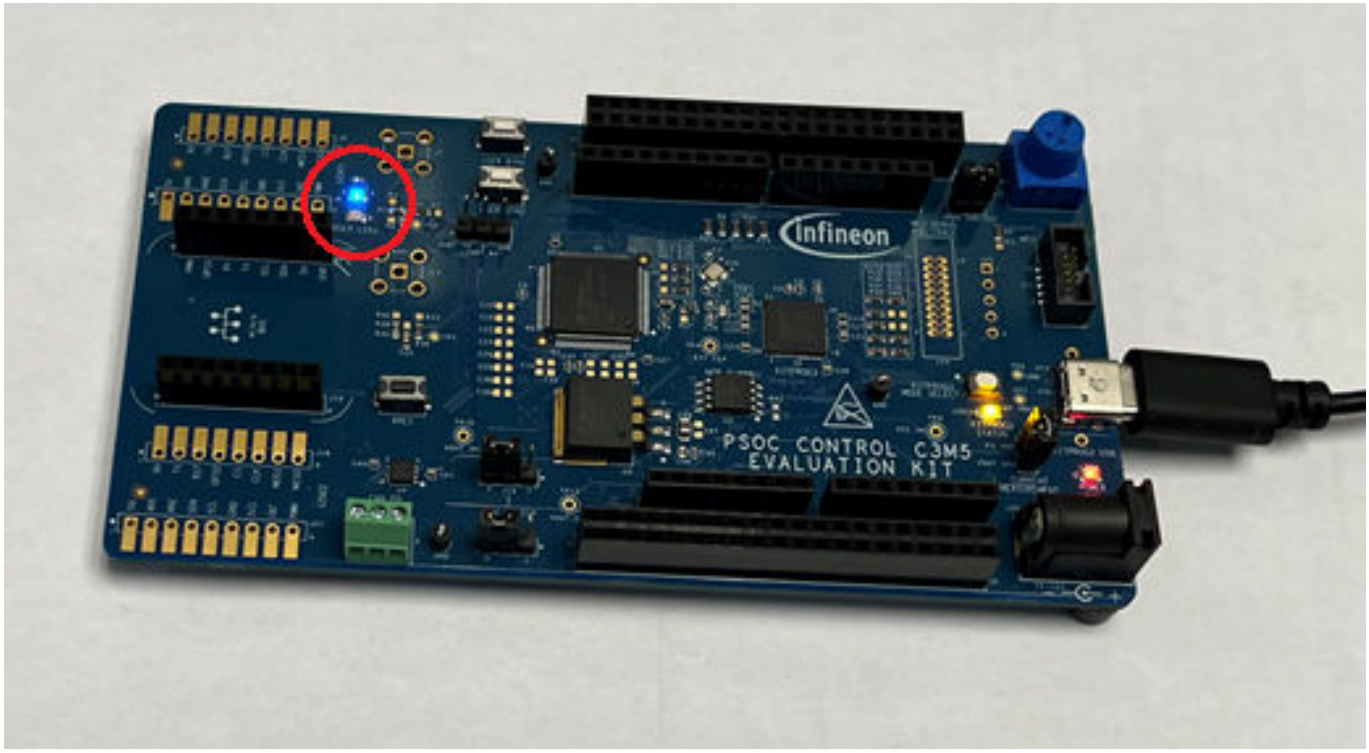
- **Launch**: This builds the associated project, programs project-specific output file, and then starts a debugging session.
- **Attach**: This starts a debugging session attaching to a running target without programming or resetting.
- **Erase Device**: This erases all internal memories.
- **Erase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

6.1 Program

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



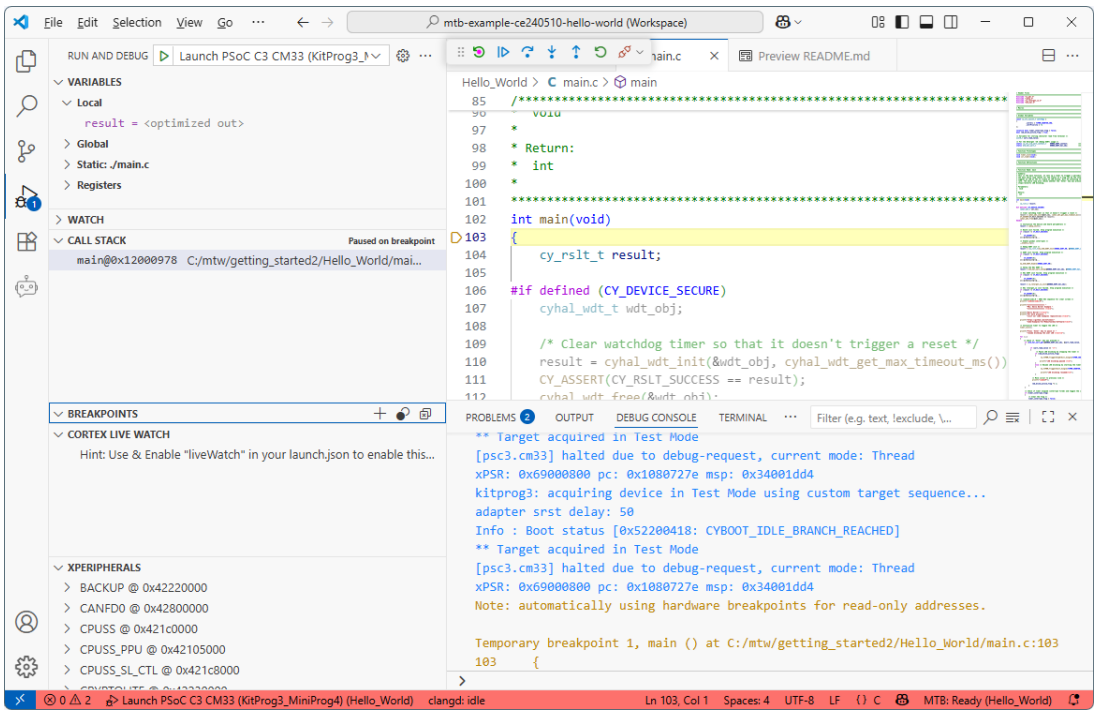
If needed, VS Code builds the application and messages display in the Terminal. If the build is successful, device programming starts immediately. If there are build errors, then error messages will indicate as such. When programming completes successfully, the LED will start blinking.



6.2 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.



6.3 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:

6.4 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
```

6.5 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:

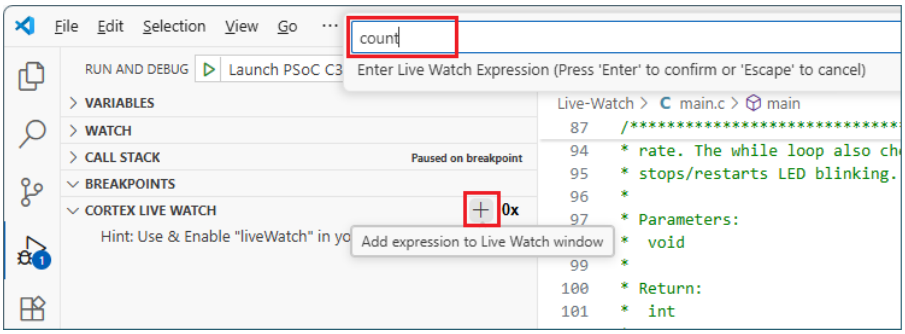
```
count++;
```

```
        /* 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
```

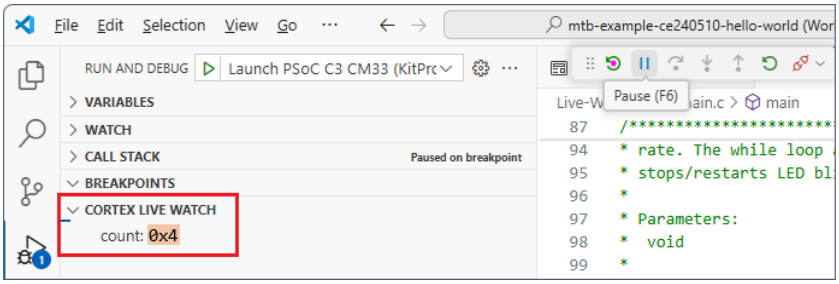
5. Launch a serial terminal such as PuTTY to monitor the output.

   ***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.



6.6 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#master#$$ASSET_REPO$$/RTT/master
```

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

```
# Segger RTT
$(SEARCH_RTT)/Examples
$(SEARCH_RTT)/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.

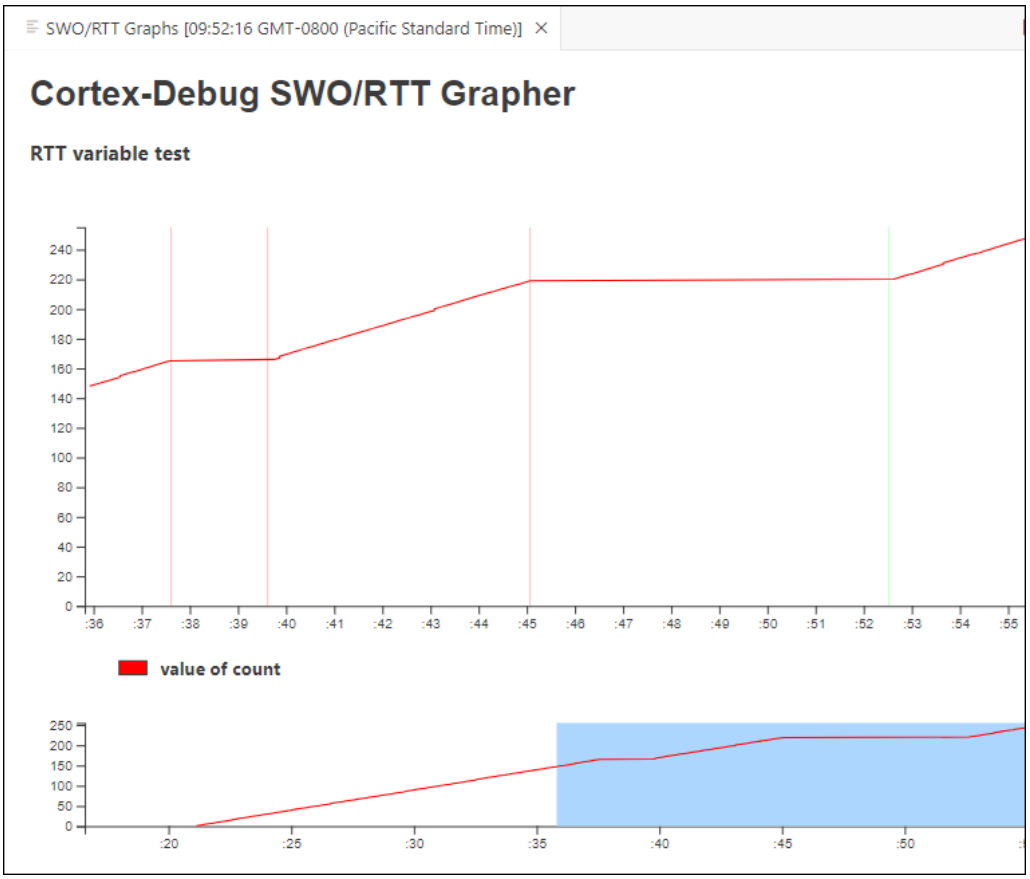
9. Open the *launch.json* file and, in the same "Launch" configuration where you added the live watch, and add the `rttConfig` configuration, as follows:

```
"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

6.7 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

6 Program/debug common

- 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.

```
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.

6 Program/debug common

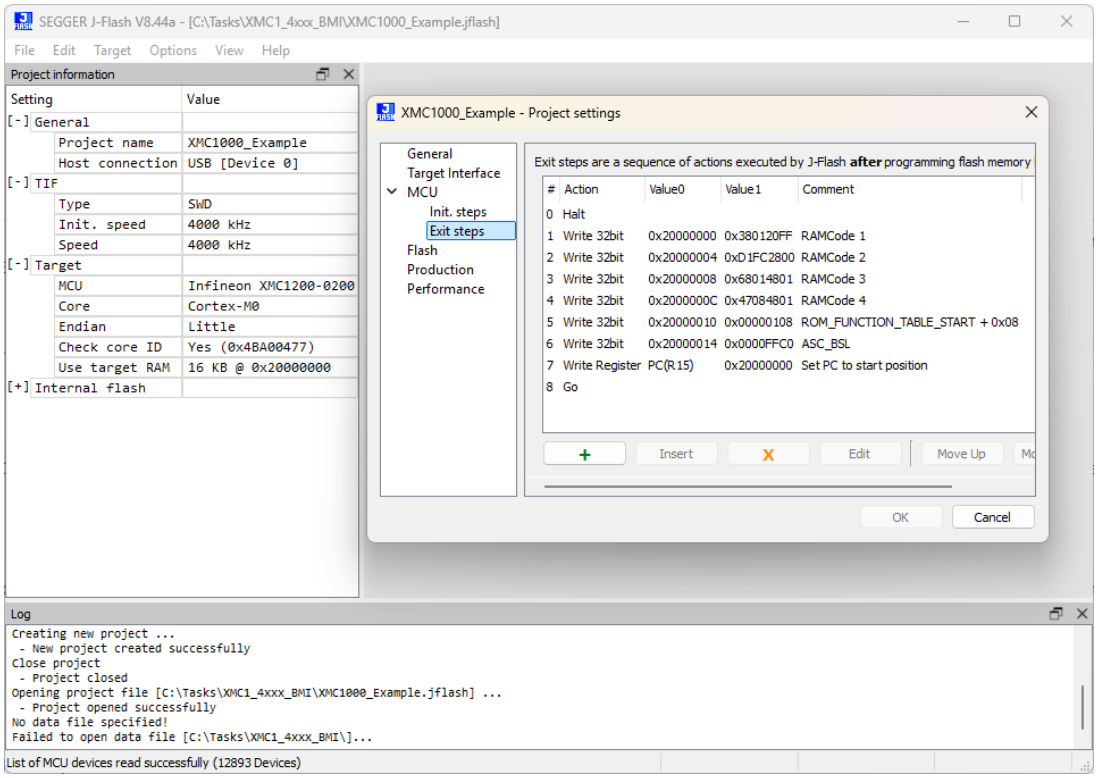
```
Connecting to target via SWD
Trying to identify target via SPD
Could not identify target via SPD. Trying again via SWD.
Found SW-DP with ID 0x0BB11477
DPIDR: 0x0BB11477
CoreSight SoC-400 or earlier
Scanning AP map to find all available APs
AP[1]: Stopped AP scan as end of AP map has been reached
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.



7 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.

7.1 Connect the Kit

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

7.2 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.*

7.2.1 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.

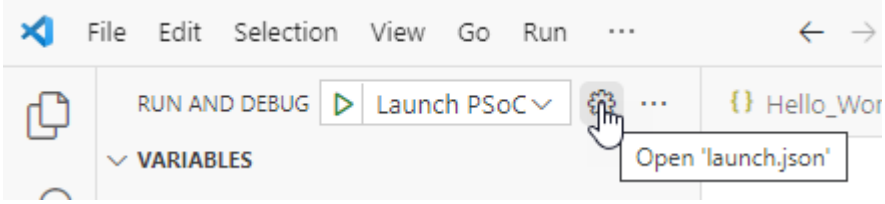
***Note**: Verify the voltage range supported by the target MCU, since it can be damaged by supplying unsupported voltage. Make sure that your MCU is not powered externally before supplying power via the KitProg3_MiniProg4 launch configuration. This supply is limited to approximately 200 mA and is protected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

7.2.1.1 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>"
```

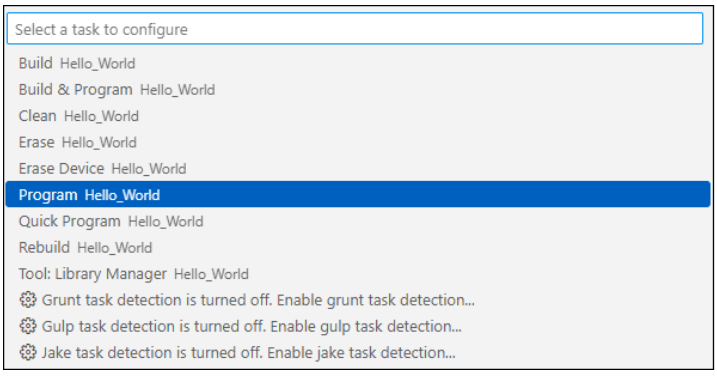
Where `<mV>` defines target voltage in millivolts. For example:

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.

7.2.2 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.

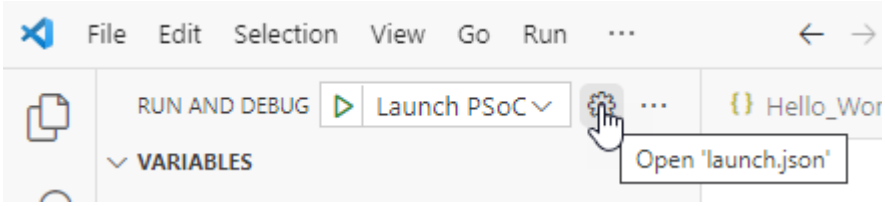
***Note**: Verify the voltage range supported by the target MCU, since it can be damaged by supplying unsupported voltage. Make sure that your MCU is not powered externally before supplying power via the KitProg3_MiniProg4.*

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.

7 Program/debug using KitProg3/MiniProg4



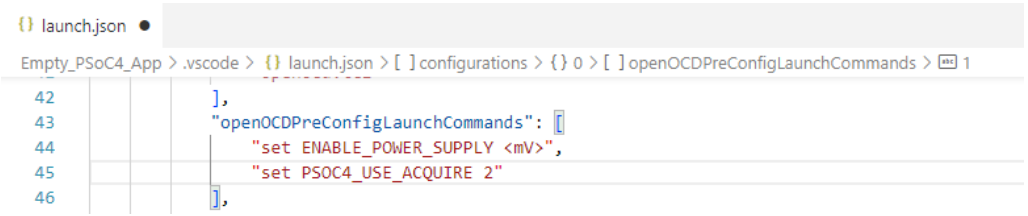
2. Look for the `openOCDPreConfigLaunchCommands` property. If it is not present, add it. Update the property to include the following value:
   - For PSOC™ 6:

```
"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:

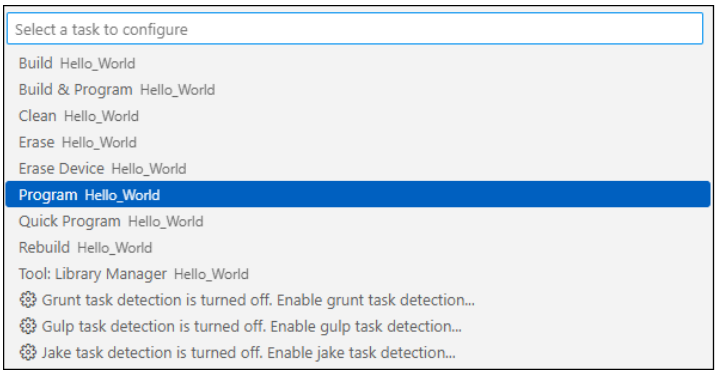


3. Save the changes to the file.

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.

8 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.

8.1 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:

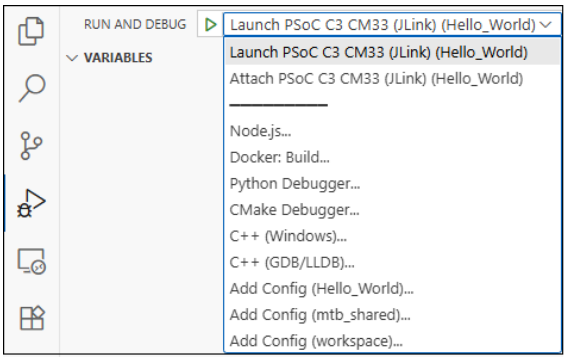
```
MTB_JLINK_DIR=<path to J-Link>
```

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"
```

8 Program/debug using J-Link

settings.json

Hello_World > .vscode > {} settings.json > ...

```
13    "modustoolbox.toolsPath": "C:/Users/follettcj/ModusToolbox/tools_3.1",
14    "cortex-debug.armToolchainPath": "${config:modustoolbox.toolsPath}/gcc/bin",
15    "cortex-debug.openocdPath": "${config:modustoolbox.toolsPath}/openocd/bin/openocd.exe",
16    "cortex-debug.JLinkGDBServerPath.windows": "C:/Program Files/SEGGER/JLink/JLinkGDBServerCL.exe",
17    "cortex-debug.JLinkGDBServerPath.osx": "/Applications/SEGGER/JLink/JLinkGDBServerCLExe",
18    "cortex-debug.JLinkGDBServerPath.linux": "JLinkGDBServerCLExe"
19  }
```

mtb-example-hal-hello-world.code-workspace

Hello_World > {} mtb-example-hal-hello-world.code-workspace > ...

```
21      "modustoolbox.toolsPath": "C:/Users/follettcj/ModusToolbox/tools_3.1",
22      "cortex-debug.armToolchainPath": "${config:modustoolbox.toolsPath}/gcc/bin",
23      "cortex-debug.openocdPath": "${config:modustoolbox.toolsPath}/openocd/bin/openocd.exe",
24      "cortex-debug.JLinkGDBServerPath.windows": "C:/Program Files/SEGGER/JLink/JLinkGDBServerCL.exe",
25      "cortex-debug.JLinkGDBServerPath.osx": "/Applications/SEGGER/JLink/JLinkGDBServerCLExe",
26      "cortex-debug.JLinkGDBServerPath.linux": "JLinkGDBServerCLExe"
27    },
```

8.2 Connect the Kit

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

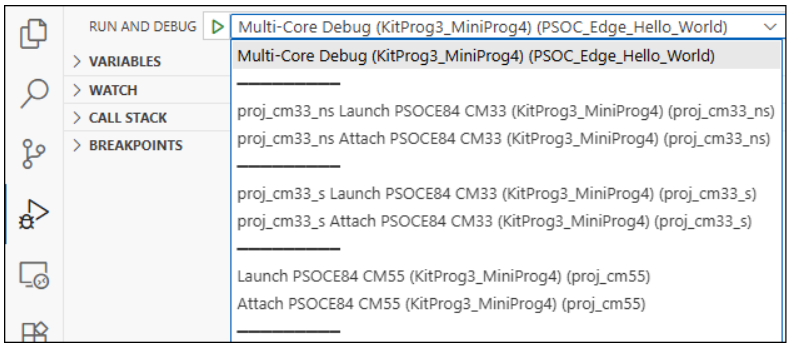
9 Multi-core debugging

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)

9.1 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

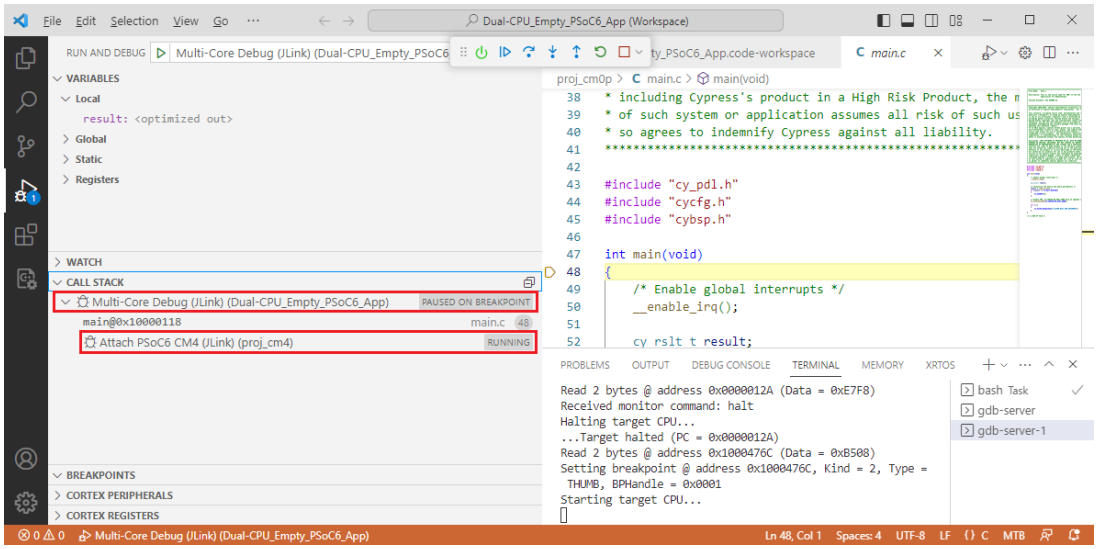
In addition to these configurations, there is associated VS Code tasks available through the main menu **Terminal > Run Task**.

- **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.

9.2 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.

9 Multi-core debugging



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.*

9.3 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.

9 Multi-core debugging

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
```

9 Multi-core debugging

For J-Link

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

Revision history

Revision	Date	Description
**	2023-05-16	New document.
*A	2023-07-18	Added instructions for using MiniProg4 and powering the MCU.
*B	2024-01-25	Updates for version 3.2 tools package.
*C	2024-10-02	Updates for version 3.3 tools package.
*D	2024-10-11	Updated information about tasks and power control.
*E	2024-12-06	Updates for version 3.4 tools package.
*F	2025-03-24	Updates for version 3.5 tools package.
*G	2025-05-20	Added note that Live Watch is not supported for multi-core applications. Updated path to fw-loader; located only in Programming tools.
*H	2025-09-03	Updates for version 3.6 tools package. Added section for multi-core debug CM33 secure application booting from RRAM.
*I	2025-12-12	Updates for version 3.7 tools package. Added BMI topic for XMC1xxx/XMC4xxx devices. Updated information on ModusToolbox™ Assistant extension.

Trademarks

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

Edition 2025-12-12

Published by

Infineon Technologies AG

81726 Munich, Germany

© 2025 Infineon Technologies AG

All Rights Reserved.

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

Email: erratum@infineon.com

Document reference
IFX-kpt1712774185041

Important notice

The information contained in this application note is given as a hint for the implementation of the product only and shall in no event be regarded as a description or warranty of a certain functionality, condition or quality of the product. Before implementation of the product, the recipient of this application note must verify any function and other technical information given herein in the real application. 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) with respect to any and all information given in this application note.

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.