Eclipse for ModusToolbox™ user guide

About this document

A newer version of this document may be available on the web here.

Scope and purpose

ModusToolbox™ software is a set of tools that enable you to integrate our devices into your existing development methodology. One of the tools is a multi-platform, Eclipse-based Integrated Development Environment (IDE) that supports application configuration and development.

This document provides information about creating applications as well as building, programming, and debugging them using Eclipse.

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
- ModusToolbox™ tools package user guide
- Eclipse survival guide

Table of contents

About this document . . . 1

Table of contents . . . 2

1 Getting started . . . 3
1.1 Installing Eclipse . . . 3
1.2 Launch Eclipse . . . 3
1.3 Configure tools & toolchains . . . 4
1.4 Open Project Creator tool . . . 5
1.5 Build application . . . 6
1.6 Program application . . . 7
1.7 Export/share application . . . 7
1.8 Import application . . . 8
1.9 Search online for code examples . . . 9
1.10 Search online for libraries/BSPs . . . 9
1.11 Access training material . . . 9

2 GUI description . . . 10
2.1 Project Explorer . . . 11
2.2 Quick Panel . . . 11

3 Configure applications . . . 12
3.1 Modify code . . . 12
3.2 How-to set configuration options . . . 13
3.3 Use configurators . . . 14
3.4 Use tools . . . 14
3.5 Use integrated terminal . . . 16
3.6 Refresh Quick Panel . . . 18
3.7 Rename application . . . 18
3.8 Restore shared directory . . . 19

4 Build applications . . . 21
4.1 Build with Make . . . 21
4.2 Build with Eclipse . . . 21

5 Program and debug . . . 22
5.1 PSOC™ MCU programming/debugging . . . 23
5.2 AIROC™ Bluetooth® programming/debugging . . . 38
5.3 XMC1xxx/4xxx devices programming/debugging . . . 39

Disclaimer . . . 42

1 Getting started

This section provides a basic walk-through for how to create a couple applications using Eclipse, selecting a BSP. It also covers how to build and program them using Eclipse and basic launch configurations supplied for the applications.

- Installing Eclipse
- Launch Eclipse
- Configure tools & toolchains
- Open Project Creator tool
- Build application
- Program application
- Export/share application
- Import application
- Search online for code examples
- Search online for libraries/BSPs
- Access training material

1.1 Installing Eclipse

Beginning with the ModusToolbox™ tools package version 3.4.0, Eclipse for ModusToolbox™ is no longer included by default. Instead, it is a separate package that you can install via the ModusToolbox™ Setup program.

The Setup program is located on our website here: https://softwaretools.infineon.com/tools/com.ifx.tb.tool.modustoolboxsetup. It is used to install base packages and additional packages on Windows, Linux, and macOS. Refer to the ModusToolbox™ software installation guide for specific instructions.

1.2 Launch Eclipse

Eclipse is installed in the following directory, by default:

C:\Users\<UserName>\Infineon\Tools\ModusToolboxEclipse

***Note**: If the software is not installed in the default location, you will need to set an environment variable. Refer to the ModusToolbox™ software installation guide for details.*

To launch Eclipse:

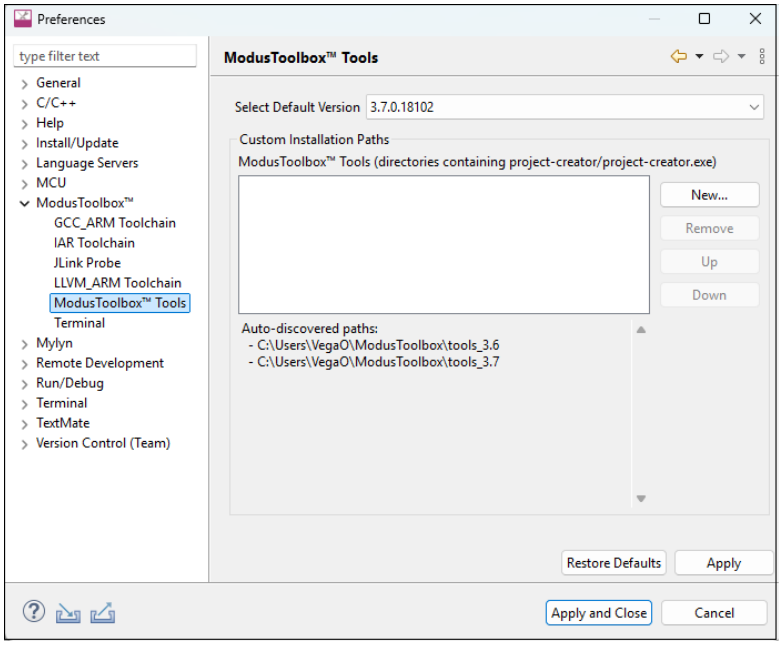
- On Windows, select Eclipse for **ModusToolbox™ <version> item** from the **Start** menu.
- For other operating systems, run the "modustoolbox" executable file.
- You can also launch Eclipse from the dashboard for all operating systems. For more information refer to the dashboard user guide.

When launching Eclipse, it provides an option to select the workspace location on your machine. This location is used by Eclipse for creating and storing the files as part of application creation for a particular platform. The default workspace location is a folder called "mtw" in your home directory. You may add additional folders under the "mtw" folder or to choose any other location for each workspace.

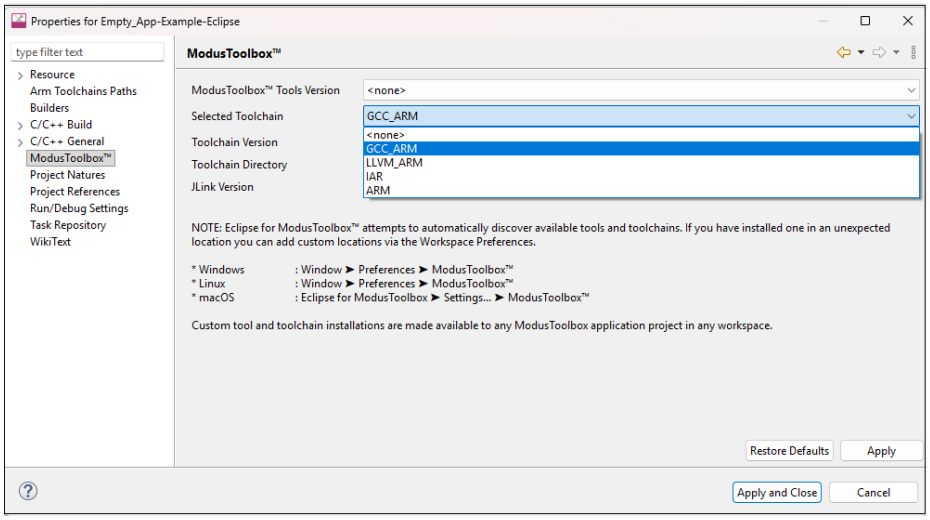
For more details about Eclipse, refer to Eclipse documentation, as well as the Eclipse survival guide.

1.3 Configure tools & toolchains

To configure Tools and Toolchains, open the **Preferences** menu and select **ModusToolbox™ Tools**.



To select your preferred toolchain, open the **Properties** menu and choose between the different toolchain options.



See the table below for more specific information on available tools and toolchains.

Sub-pane	Purpose
GCC_ARM Toolchain	View auto-discovered GCC_ARM toolchains and provide custom install paths when necessary. Auto-discovery will find installations installed by ModusToolbox™ Setup program.
IAR Toolchain	View auto-discovered IAR toolchains and provide custom install paths when necessary. Auto-discovery will look for IAR toolchain installations in the common search directories (defined below).

Sub-pane	Purpose
JLink Probe	View auto-discovered SEGGER J-Link installations provide custom install paths when necessary. Auto-discovery will look for SEGGER J-Link toolchain installations in the common search directories (defined below). ***Note**:* *J-Link installation directories must include the version number (e.g.,`C:\Program Files\SEGGER\JLink_V896`*
LLVM_ARM Toolchain	View auto-discovered LLVM_ARM toolchains and provide custom install paths when necessary. Auto-discovery will look for LLVM_ARM toolchain installations in the common search directories (defined below).
Terminal	Integrated terminal used to enter various commands for the selected project. ***Note**:* *For more information on this sub-pane, see* Use integrated terminal

Common auto-discovery search directories for each OS include:

Windows:

- `%ProgramFiles%` (typically `C:\Program Files`)
- `%ProgramFiles(x86)%` (typically `C:\Program Files (x86)`)
- `%ProgramFiles(Arm)%` (typically `C:\Program Files (Arm)`)
- `%USERPROFILE%` (user's home directory)
- `C:\` root of drive C:

Linux:

- `/usr/local/`
- `/opt/`
- `$HOME` (user's home directory)

macOS:

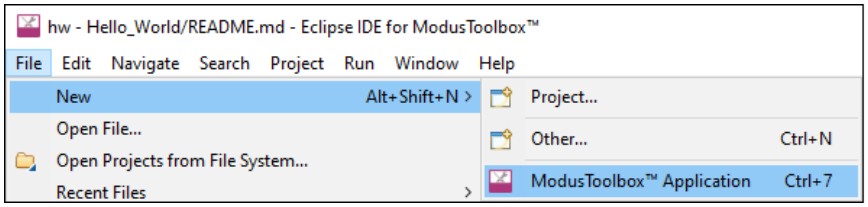
- `/Applications/`
- `/usr/local/`
- `/opt/`
- `$HOME` (user's home directory)

***Note**:* *The ARM toolchain is not supported by the Tools & Toolchain configuration preferences. It continues to use the `CY_COMPILER_ARM_DIR` environment variable.*

1.4 Open Project Creator tool

Click the **New Application** link in the Eclipse Quick Panel.

You can also select **File > New > ModusToolbox™ Application**.



These commands launch the Project Creator tool, which provides several applications for use with different development kits. The kits available may change over time.

For more details about using this tool, refer to the Project Creator user guide.

1.5 Build application

After loading an application, build it to generate the necessary files. Select a project. Then, in the Quick Panel, click the **Build Application** link.

Messages display in the Console, indicating whether the build was successful or not. For more details about building applications and the various Consoles available, see the Build applications chapter.

1.6 Program application

There are many more details about programming an application. This section only covers it briefly. For more detailed information, see the Program and debug chapter.

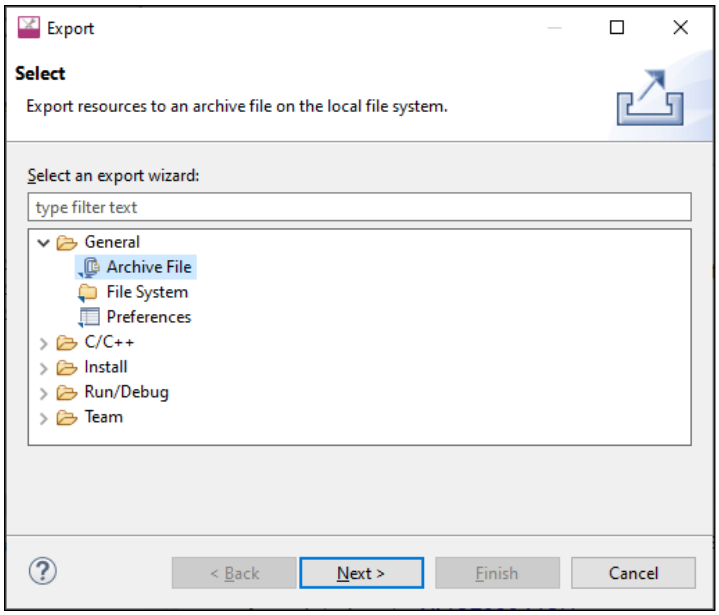
In the Project Explorer, select the desired project. Then, in the Quick Panel, click the **<app-name> Program (KitProg3_MiniProg4)** link for a PSOC™ MCU application, and **<app-name> Program** for an AIROC™ Bluetooth® application.

***Note**: The **Advanced KitProg3 Programming** item in the Quick Panel is used to launch the ModusToolbox™ Programmer with the device and a HEX file already selected and configured for you. You must build your application before using this item. For more details on using the Programmer tool, see the ModusToolbox™ Programmer user guide.*

1.7 Export/share application

There are several ways to share a ModusToolbox™ application. Refer to the ModusToolbox™ tools package user guide in the "Version Control and sharing applications" chapter for more general details.

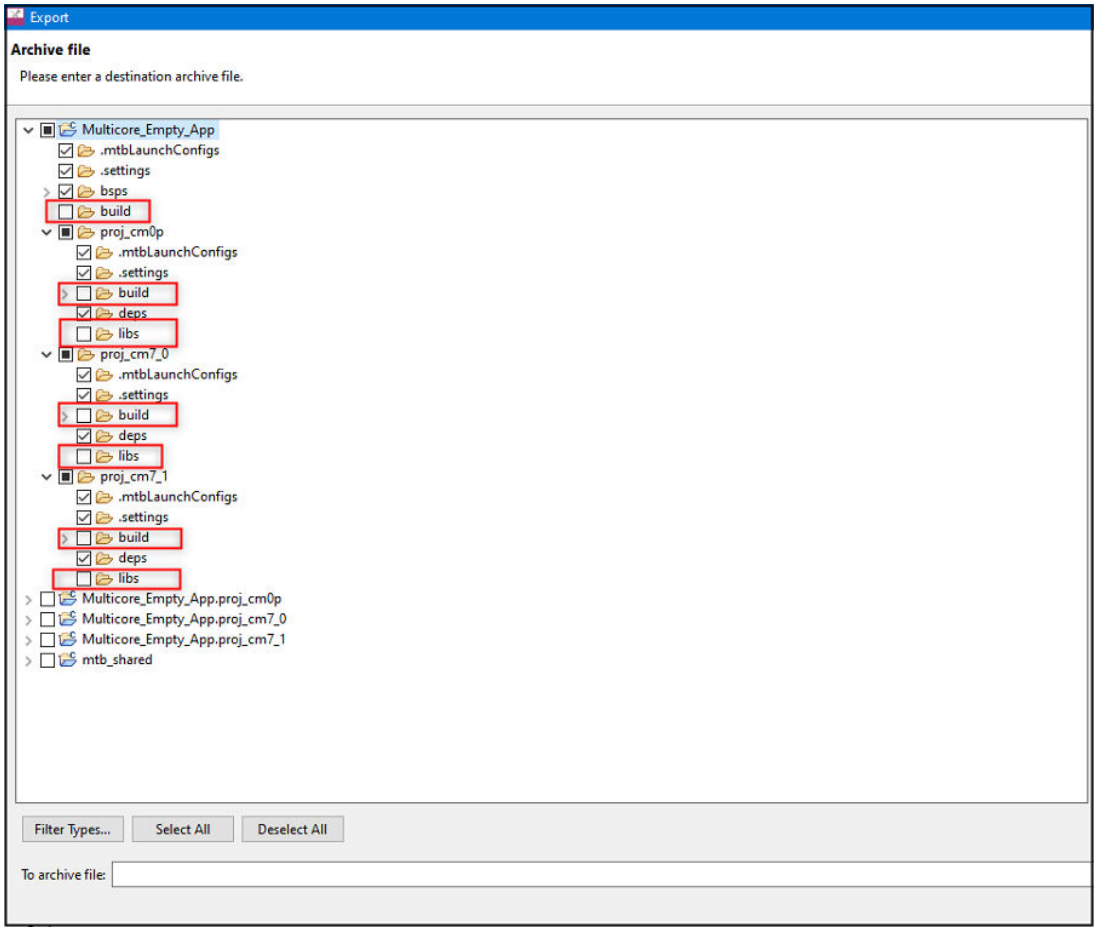
Using Eclipse, you can use the export function to essentially make a copy of your application that you can share with a colleague. There are several ways to export using Eclipse. The option we recommend is to select the application and then use **File > Export > General > Archive File**, which creates a ZIP file.



Keep in mind that a typical ModusToolbox™ application includes many libraries that are available on GitHub. These get regenerated during the make getlibs command. Therefore, you can substantially reduce the size of your exported application by excluding the libraries. Plus, the exported *mtb_shared* folder might retain hard-coded paths that would cause issues after importing the application.

For multi-core applications, Eclipse export function shows the core project folders along side the main application folder. These project folders are included in that main application folder, so you should exclude them from the export process as well.

On the Export dialog, select only the main application folder(s) to export. You may need to review your application in Eclipse to determine which folders you should exclude. By default, subproject folders include an extension after the application name.



***Note**: Ignore the extra project folders shown at the bottom of the window under the destination archive files. They are a byproduct of Eclipse and unnecessary to the export process.*

Click **Browse…** to navigate to the location to save the archive and provide a file name.

Click **Finish** to generate the archive file.

***Note**: Make sure build, libs, and mtb_shared are not selected to exclude them from the archive.*

1.8 Import application

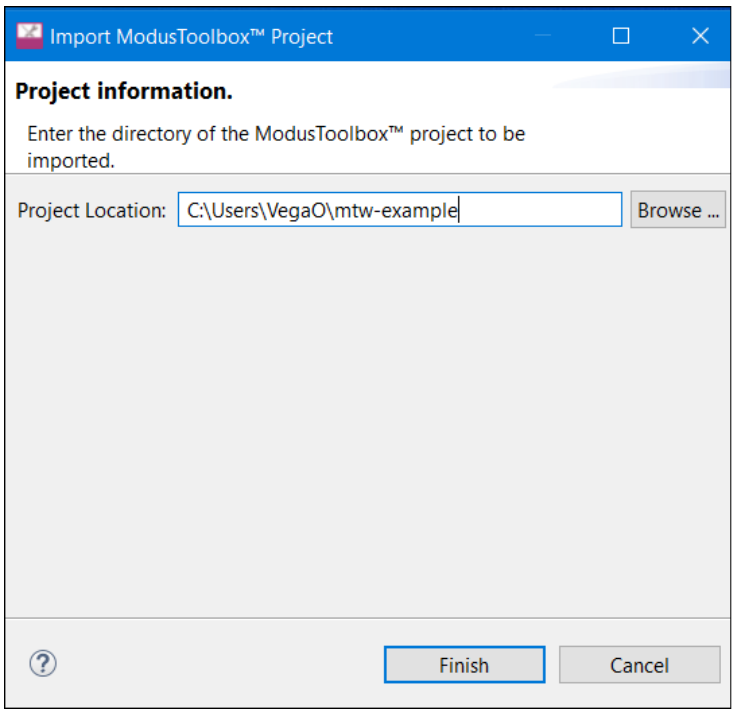
If you have a ModusToolbox™ application that you want to import into Eclipse, run the make getlibs command before importing it into Eclipse.

***Note**: If the application is included in a ZIP file, you will need to extract it before running various commands.*

Then, either use the **Import Existing Application In-Place** link on the Quick panel or open through the **File > Import ModusToolbox™ > Import Existing Application In-Place** option.

***Note**: This option does not copy or move the application location on disk; it only makes a reference for Eclipse to the current application location.*

On the next page, click the **Browse…** button, navigate to the application directory, and click **Select Folder**.



Click **Finish** to begin the import process. This will take a few moments, and then the application will display in Eclipse Project Explorer.

If the Console displays a message, such as "Error creating Eclipse configurations," open the Library Manager and click **Update**. This runs the `make getlibs` operation to generate the necessary files and libraries.

***Note**: There are various ways to import examples into Eclipse. If you prefer a different method, make sure that all of the project files are copied into the workspace directory.*

1.9 Search online for code examples

Infineon provides many code examples. These examples allow you to explore the capabilities provided by the SDK, create applications based on them, examine the source code demonstrated in them, and read their associated documentation. The Quick Panel provides a link to access online code examples. Click the **Search Online for Code Examples** link. This opens a web browser to the GitHub repository to select and download the appropriate examples.

1.10 Search online for libraries/BSPs

Infineon also provides all the libraries and BSPs online at GitHub. The Quick Panel provides a link to access these. Click the **Search Online for Libraries and BSPs** link. This opens a web browser to the GitHub repository that shows the ModusToolbox™ software page.

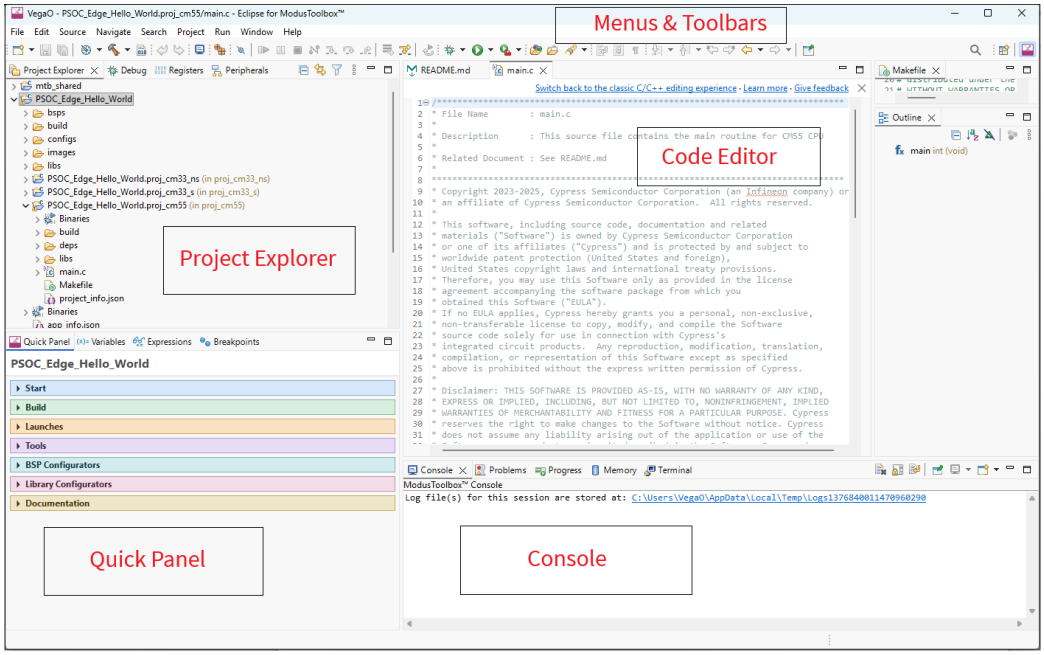
1.11 Access training material

Infineon also provides training material at GitHub. The Quick Panel provides a link to access these. Click the **Training Material** link. This opens a web browser to the GitHub repository that shows the ModusToolbox™ training page.

2 GUI description

The GUI is based on Eclipse. It uses several plugins, including the Eclipse C/C++ Development Tools (CDT) plugin. For more information about Eclipse, refer to the Eclipse Workbench User Guide. We also provide a document called the Eclipse survival guide, which provides tips and hints for how to use Eclipse.

The GUI contains Eclipse menus and toolbars, plus various panes such as the Project Explorer, Code Editor, and Console. One difference from Eclipse is the "ModusToolbox™ Perspective." This perspective provides the "Quick Panel," and adds tabs to the Project Explorer. "Perspective" is an Eclipse term for the initial set and layout of views.



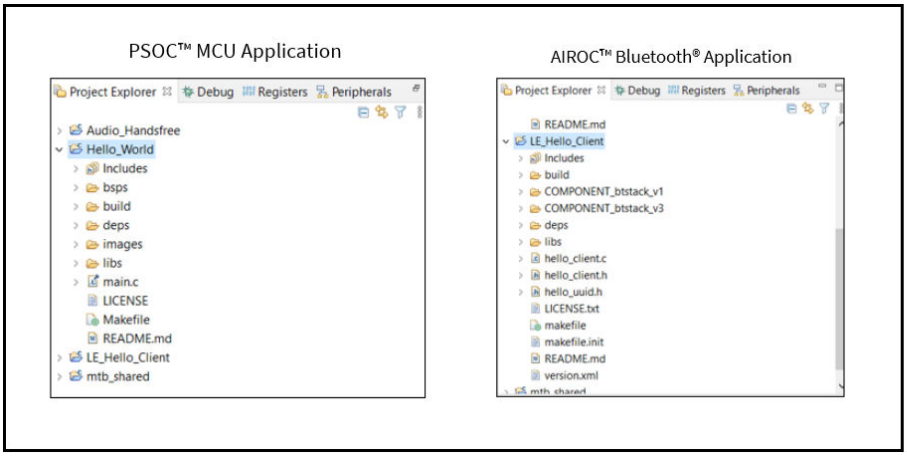
***Note**: If you switch to a different perspective, you can restore the ModusToolbox™ Perspective by clicking the ModusToolbox™ icon button in the upper-right corner. You can also select Perspective > Open Perspective from the Window menu. To restore the ModusToolbox™ Perspective to the original layout, select Perspective > Reset Perspective from the Window menu.*

The following describe different parts of the GUI:

- **Menus and toolbars** – Use the various menus and toolbars to access build/program/debug commands for your application. Many of these are covered in the Eclipse Workbench User Guide. Note that some menu and toolbar items have been removed from standard Eclipse. We recommend using the Quick Panel with ModusToolbox™ version 3.2.
- **Project Explorer** – Use the Project Explorer to find and open files in your application. See Project Explorer for more information.
- Quick Panel – Use this tab to access appropriate commands, based on what you select in the Project Explorer.
- **Code Editor** – Use the Code Editor to edit various source files in your application.
- **Console** – Use these tools to review messages and access the integrated terminal.

2.1 Project Explorer

In Eclipse, after creating an application, the Project Explorer contains one or more related project folders. The following images show a PSOC™ MCU application and an AIROC™ Bluetooth® application.



Both types of applications contain a similar project structure. Each contains the main application source code, and a Makefile. Note that PSOC™ MCU applications contain a libs directory, while AIROC™ Bluetooth® applications have a wiced_btsdk project with shared SDK, BSPs, and libraries for all applications in a workspace.

2.2 Quick Panel

As stated previously, the Quick Panel is part of the "ModusToolbox™ Perspective." It provides quick access to commands and documentation based on what you have selected in the Project Explorer.

The Quick Panel contains links to various commands and documentation, organized as follows:

- **Start** – This contains the New Application link to create new applications, and links to find Code Examples, Libraries, BSPs, and training material.
- **Build** – This contains different project-related links based on the project that is selected in the Project Explorer, as well as the type of application. Links here include: Build and Clean the application.
- **Launches** – This contains various Launch Configurations, based on the selected application project and device, which can be used to program the device and launch the debugger. For more information, see Launch configurations.
- **Tools** – This contains links to the various tools available for the selected project. For more information, see Use configurators and Use tools.
- **BSP Configurators** – This contains links to BSP and application configurators available for the selected project.
- **Library Configurators** – This contains links to library configurators available for the selected project.
- **Documentation** – This may contain several documents in HTML format, which are included as part of the chosen BSP. This also includes the most recent version of the Eclipse for ModusToolbox™ release notes and User Guide .

3 Configure applications

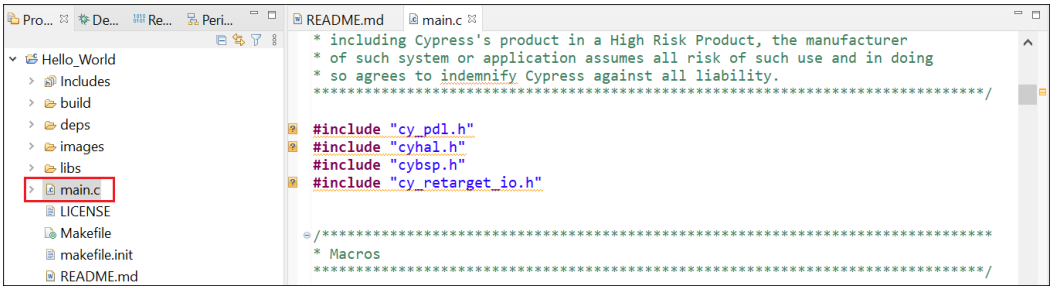
This chapter covers how to make various changes to your application. It includes:

- Modify code
- How-to set configuration options
- Use configurators
- Use tools
- Use integrated terminal
- Refresh Quick Panel
- Rename application
- Restore shared directory

3.1 Modify code

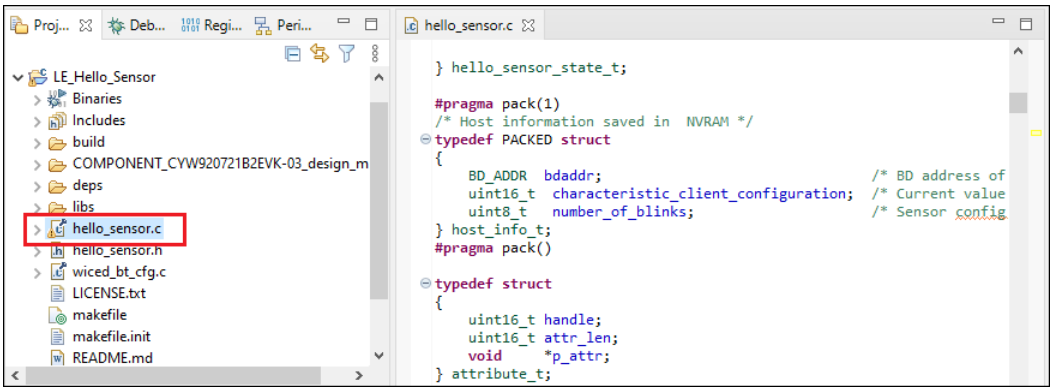
Most code examples work as they are, and there is no need to add or modify code in order to build or program them. However, if you want to update and change an application to do something else, or if you are developing your own application, open the appropriate file in the code editor.

- **PSOC™ MCU**: In the Project Explorer, double-click the *main.c* file.



***Note**: The "includes" in this example show "unresolved inclusions" because the files are located in the mtb_shared folder. This and other issues with IntelliSense are resolved after a build.*

- **AIROC™ Bluetooth®**: In the Project Explorer, expand the <*app-name*> project folder and double-click the application <*app-name*>.*c* file.



As you type into the file, an asterisk (*) will appear in the file's tab to indicate changes were made. The **Save/ Save As** commands will also become available to select.

***Note**: C/C++ Search is no longer supported by ModusToolbox™ Eclipse due to programmatic disabling of indexing. You should disable C/C++ Search via the **Search > Search > Customize** menu. Other search types are not affected.*

3.2 How-to set configuration options

Since a ModusToolbox™ application is based on the GNU-make system, most of the project configuration is controlled by editing the Makefile. This section contains common configuration options that Eclipse users need to be aware of. For more information about the ModusToolbox™ build system, see chapter five in the tools package user guide.

3.2.1 Specify debug and release configurations

Edit the project/application *Makefile* `CONFIG` parameter. Valid options are: Debug, Release, and Custom. If you set the value to Custom, then also set the optimization flag in the `CFLAGS` variable.

***Note**: Do not use the global configuration **Debug/Release** menu items, and do not use the **Project > Build Configurations** option. These are both ignored.*

3.2.2 Exclude files/folders from build

By default, a ModusToolbox™ application includes a *.cyignore* file that specifies files and folders to exclude from a build. You can add entries to this file. You can also edit the *Makefile* to add `CYIGNORE` make variables.

Do not use **Resource Configurations > Exclude From Build...** This will be ignored.

3.2.3 Specify linker settings (including per-file linker settings)

Edit the project/application *Makefile* for these variables: `CFLAGS`, `CXXFLAGS`, `ASFLAGS`, `LDFLAGS`

There is no other way to specify linker settings in Eclipse.

3.2.4 Specify probe and generate launch configs

Most PSOC™-based kits use KitProg3/MiniProg4 as the default programmer/debugger, and they show those configurations in the Quick Panel.

Switch to J-Link programmer/debugger

If you want to use SEGGER J-Link as the programmer/debugger, do the following to change configurations:

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

```
BSP_PROGRAM_INTERFACE=JLink
```

2. Also enter the following variable to specify the location of where J-Link will be installed:

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

3. Save the bsp.mk file.
4. Click **Generate Launches for <project_name>**.

   When Eclipse refreshes, J-Link configurations will be shown.

   ***Note**: For more information on specifying the correct location of J-Link installation, refer to the Configure tools & toolchains section.*

Switch back to KitProg3

If you want to switch back to KitProg3, delete the variables you entered above and regenerate the launch configurations.

3.2.5 Configure search paths

Source files under the project root directory are automatically included into a build. Additional files/directories can be added to the search path using the SEARCH make variable.

Do not add a source file to the project as a link (for example, by dragging and dropping a file into a project and selecting **Link to files**). This will be ignored.

3.3 Use configurators

ModusToolbox™ software provides graphical applications called configurators that make it easier to configure a hardware block. However, before you make changes to settings in configurators, you should first copy the configuration information to the application and override the BSP configuration or create a custom BSP. If you make changes to a standard BSP library, it will cause the repo to become dirty. Additionally, if the BSP is in the shared asset repository, changes will impact all applications that use the shared BSP. For more information, refer to the ModusToolbox™ tools package user guide.

Each configurator provides a separate guide, available from the configurator's **Help** menu.

3.3.1 Launching configurators from Eclipse

To launch a configurator from Eclipse, right-click on the <app-name> project in the Project Explorer, select **ModusToolbox™**, and then select the appropriate configurator.

***Note**: You can also launch available configurators from links in the Quick Panel.*

Depending on the enabled resources in your application, there may be several configurators available to launch.

- If you launch the Device Configurator from Eclipse, you are opening the project's design.modus file, which is responsible for holding all of the BSP configuration information. It contains the following:
  - Selected device
  - Resource parameters
  - Constraints
- If you launch any of the other configurators from Eclipse, they will open using that configurator's configuration file (*design.cycapsense*, *design.cyseglcd*, etc.). These files are specific to the given resource, and they may rely on configuration data from the design.modus file.

3.3.2 Launching configurators without Eclipse

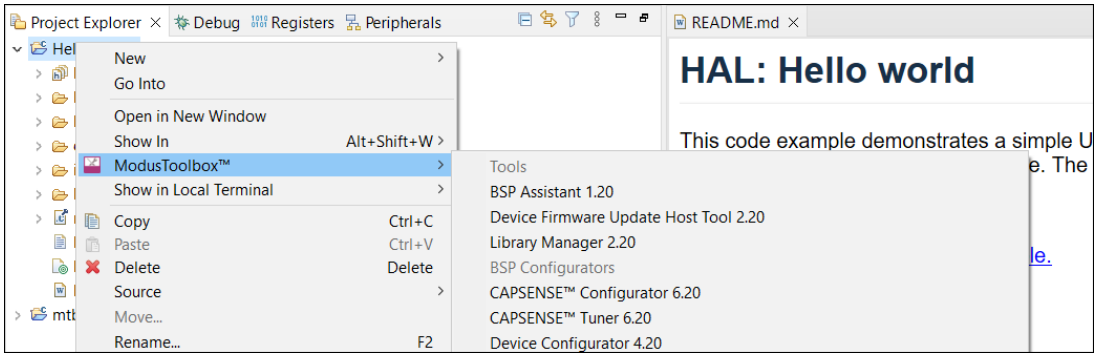
To launch any Configurator without using Eclipse, refer to the applicable configurator guide for details about configuration information. You may need to open a configuration file or create a new one.

3.4 Use tools

In addition to the configurators, there are several tools, including Library Manager, BTSpy, ClientControl, and Device Firmware Update (DFU) Host Tool. Many tools do not apply to all types of projects. So, the available tools depend on the project/application you have selected in the Project Explorer.

3.4.1 Launching tools from Eclipse

To launch a tool from Eclipse, right-click on the <app-name> project in the Project Explorer, select **ModusToolbox™**, and then select the appropriate tool.



***Note**: You can also launch available tools from links in the Quick Panel.*

3.4.2 Library Manager

The Library Manager allows you to select which Board Support Package (BSP) and version should be used by default when building a ModusToolbox™ application. It also allows you to add and remove libraries, as well as change their versions.

For more information about how to use this tool, refer to the Library Manager user guide.

3.4.3 BTSpy and ClientControl

BTSpy is a trace utility that can be used in the AIROC™ Bluetooth® applications to view protocol and generic trace messages from the embedded device. The tool listens on the UDP port 9876 and can receive specially formatted message from another application on the same or different system.

BTSpy can be used in conjunction with ClientControl to receive, decode and display protocol application and stack trace messages. ClientControl communicates with the embedded app to perform various functionalities, tests, exercising features, etc.

3.4.4 DFU Host tool

The Device Firmware Update (DFU) Host tool is a stand-alone program used to communicate with a PSOC™ MCU that has already been programmed with an application that includes device firmware update capability. For more information, refer to the Device Firmware Update Host tool user guide.

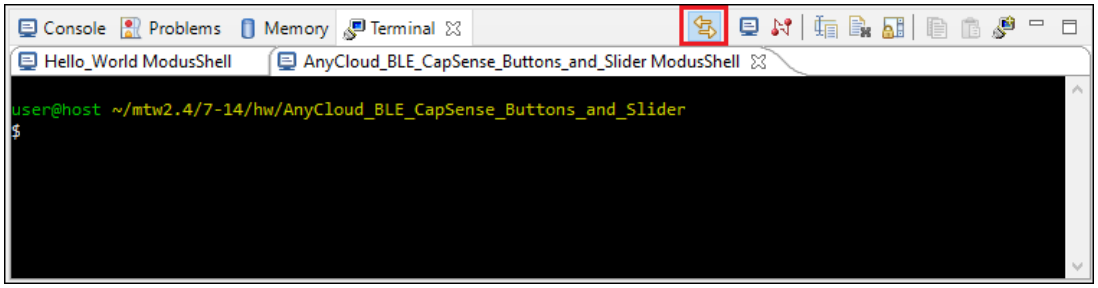
3.5 Use integrated terminal

Eclipse for ModusToolbox™ software includes an integrated terminal where you can enter various commands for the selected project. To view the terminal, click the **Terminal** tab in the bottom pane. Then, select a project in the Project Explorer to open a shell in the project directory.

***Note**: You can configure the terminal colors for the Workspace under **Window > Preferences >Terminal**.*

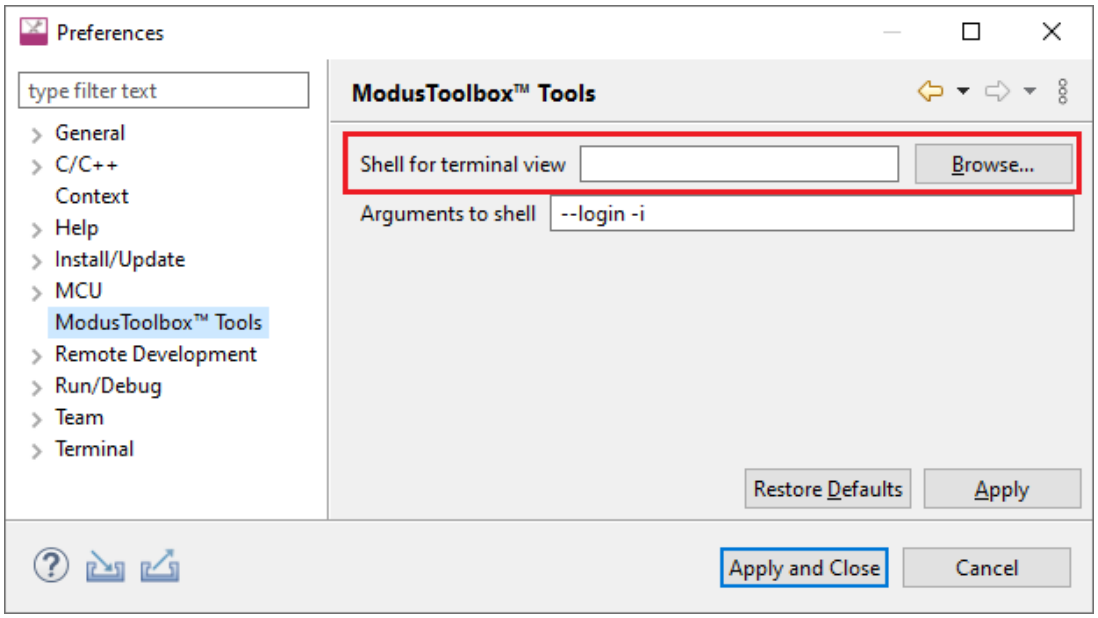
3.5.1 Switch projects

When you switch projects, Eclipse creates additional tabs and automatically switches between them, so that a shell in the current project directory is always in focus. To disable this automatic switching, deselect the **Track Current Project** button in the terminal toolbar.



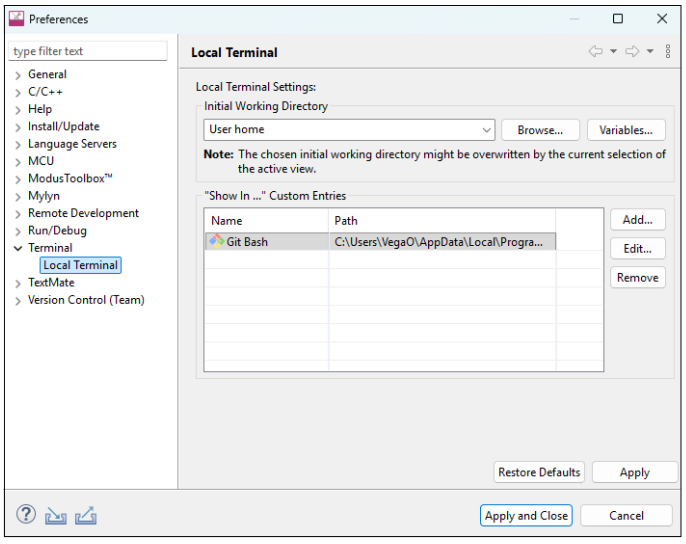
3.5.2 Specify alternate shell

By default, the terminal uses your login shell on Linux and macOS, or the "modus-shell" Cygwin bash shell provided with your ModusToolbox™ tools installation on Windows. You can specify a different shell from **Window > Preferences > ModusToolbox™ Tools**.



3.5.3 Update local terminal settings

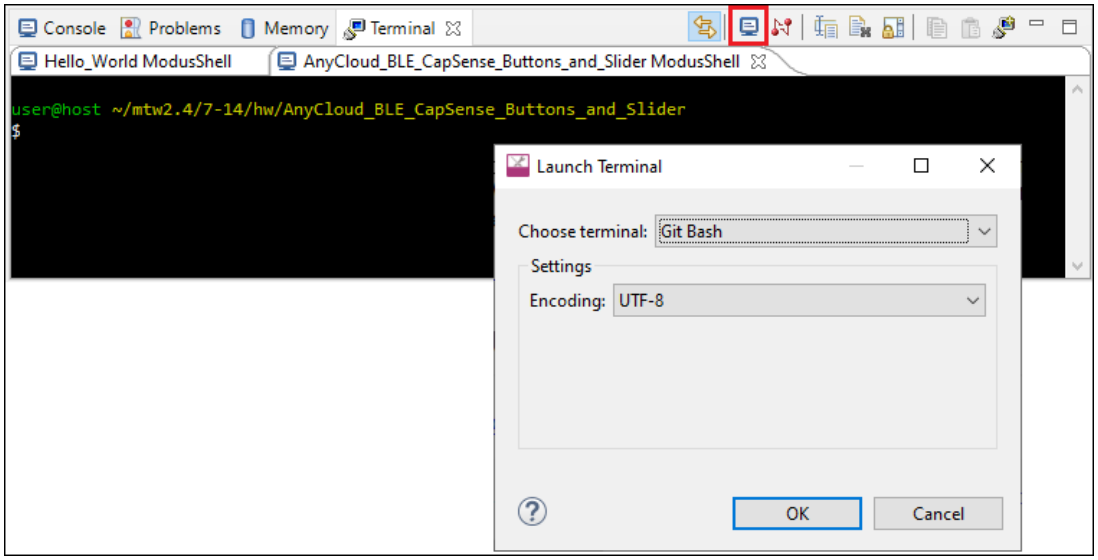
By default, the local terminal uses your User home directory as the Initial Working Directory. You can specify different options from **Window > Preferences > Local Terminal**.



***Note**: Linux and macOS include additional settings for Shell Command and Arguments.*

3.5.4 Connect to remote machine/board

You can also connect to a remote machine or board via serial, telnet, or SSH by clicking the **Open a Terminal** button in the terminal toolbar.



Use the **Choose terminal** pull-down menu to select the appropriate terminal.

3.6 Refresh Quick Panel

When you use tools external to Eclipse, such as the Library Manager or Device Configurator, it is likely that Eclipse will not refresh to detect changes made in those other tools. Use this link to refresh Eclipse. You may notice new documentation links or a new Active BSP in the Quick Panel as an indication that your application has new libraries and/or components.

3.7 Rename application

3.7.1 Single-core application

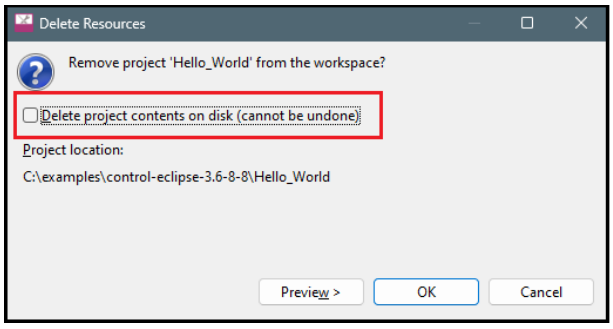
Eclipse for ModusToolbox™ software uses the standard Eclipse rename functionality. That is, right-click on the application and select **Rename**. If you use the rename feature, you will need to update your application's launch configurations. The easiest way to do this is to use the "Generate Launches…" link in the Quick Panel.

After renaming the application, select it in the Project Explorer and notice that there is only one item under Launches. Click on the **Generate Launches for…** link. After a few moments, the generate process completes. Click on the application again in the Project Explorer and notice that all the items are shown under Launches in the Quick Panel.

3.7.2 Multi-core application

To rename a multi-core application, you cannot use the standard Eclipse rename functionality. Follow these steps instead:

1. Right-click on the affected project(s) and select **Delete** to remove from the Eclipse workspace.
2. On the dialog do **not** select "Delete project contents on disk". Click **OK**.



3. Go to the workspace location on disk and manually rename the relevant project directories.
4. When renaming a child project (e.g., proj_cm33_s) for a multi-core application, edit the application *Makefile*. Look for `MTB_PROJECTS=...`, and update the list with the new child project names.
5. Open a Terminal (modus-shell in Windows) and run `make eclipse`.
6. Switch back to Eclipse and select **Import Existing Application In-Place** using the Quick Panel.

3.8 Restore shared directory

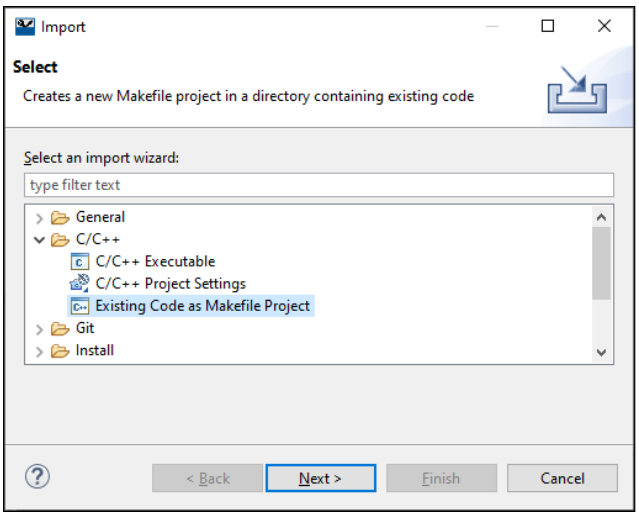
Various shared libraries imported from GitHub are located in a shared directory named mtb_shared (by default) adjacent to your application directories. You can delete the mtb_shared directory at any time because it can be recreated. You might do this when sharing the application, for example. The shared directory only contains files that are already controlled and versioned, so you should **not** check it into a revision control system.

When using Eclipse, if you delete the shared library directory from disk and then regenerate it, the directory will **not** be restored properly. This is because several files required by Eclipse are not restored as they were when the application was created. To resolve this issue:

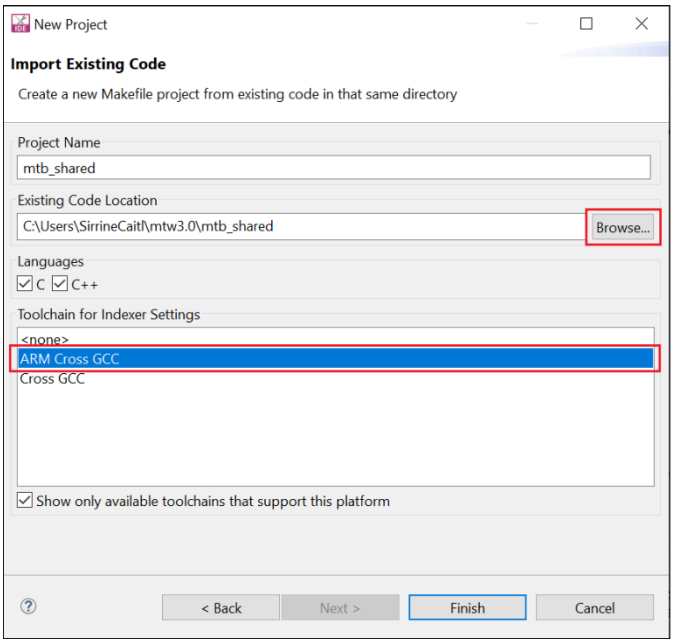
1. Regenerate the mtb_shared directory and assorted libraries on disk using make getlibs or the Library Manager.
2. In Eclipse, delete the "mtb_shared" folder shown in the Project Explorer.

   ***Note**: Do **NOT** select the check box "Delete project contents on disk" (if you do, you will have to regenerate it again).*

3. Then, select **File > Import > C/C++ > Existing Code as Makefile Project** and click **Next >**.



4. On the Import Existing Code page:



- Under **Existing Code Location**, click **Browse…**, navigate to the application's root directory, select the "mtb_shared" folder, and click **Select Folder**.
- Under **Toolchain for Indexer Settings**, select **ARM Cross GCC**.
- Click **Finish**.

5. After the import completes, build the application.

4 Build applications

This chapter covers various aspects of building applications. Building applications is not specifically required, because building is performed as part of the Program and debug. However, if you are running Eclipse without any hardware attached, you may wish to build your application to ensure all the code is correct.

- Build with Make
- Build with Eclipse

4.1 Build with Make

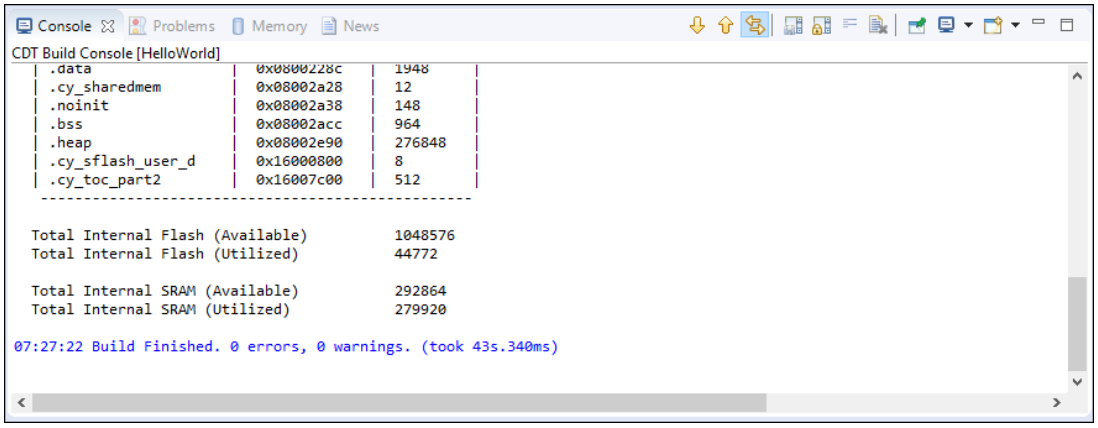
You can build applications from the command line using Make. Refer to the ModusToolbox™ tools package user guide (also located in the ~*ModusToolbox**docs_<version>* directory) for more information.

4.2 Build with Eclipse

After loading an application, it is best to first build everything to generate the necessary files. Click on a project in the Project Explorer. Then in the Quick Panel, click the "Build Application" link.

Building an application will typically allow IntelliSense to work better. It may also be useful to right-click on a project and select **Index > Rebuild** to allow IntelliSense to find references.

Messages display in the Console, indicating whether the build was successful or not.



***Note**: Be aware that there are several Console views available.*

You can access the different views using the **Display Selected Console** button (or the pull-down arrow). The Global Build Console is the only way to see all of the results in one place. If you just have the standard Console open, it resets every time a new project in the application starts building. You won't see any errors if they are not on the final project that gets built.

For subsequent updates, we recommend that you use the **Build Application** option from the Quick Panel.

***Note**: When the active build configuration is changed it affects only the selected project. The active build configuration for any dependent projects is specified separately for each project. Therefore, if you want to use the same configuration for all projects (for example, Debug or Release), it must be set for each project. It is possible to select multiple projects at once from the Project Explorer and then select the active configuration.*

5 Program and debug

Programming and debugging are native to your chosen development environment. Infineon devices are supported in the major program and development solutions. Primarily, this means J-Link and OpenOCD. These solutions provide for programming flash within a device and provide a GDB server for debugging.

This chapter covers various topics related to building and debugging, including:

- PSOC™ MCU programming/debugging
  - Launch configurations
  - PSOC™ Control: Setting the OpenOCD probe frequency
  - Debug connection options
  - KitProg Firmware Loader
  - Supplying power with KitProg3_MiniProg4
  - Power cycle programming mode with KitProg3_MiniProg4
  - Switch to J-Link programmer/debugger
  - Using JTAG interface in MiniProg4
  - Changing programming interface SWD/JTAG
  - Erasing external memory
  - Programming eFuse
  - Select specific CMSIS-DAP device
  - Select Specific J-Link Device
  - PSOC™ 4 flash security programming
  - Debugging/programming for secure configuration devices
- AIROC™ Bluetooth® programming/debugging
  - Program configuration
  - Attach configurations
  - Debug settings
- XMC1xxx/4xxx devices programming/debugging

5.1 PSOC™ MCU programming/debugging

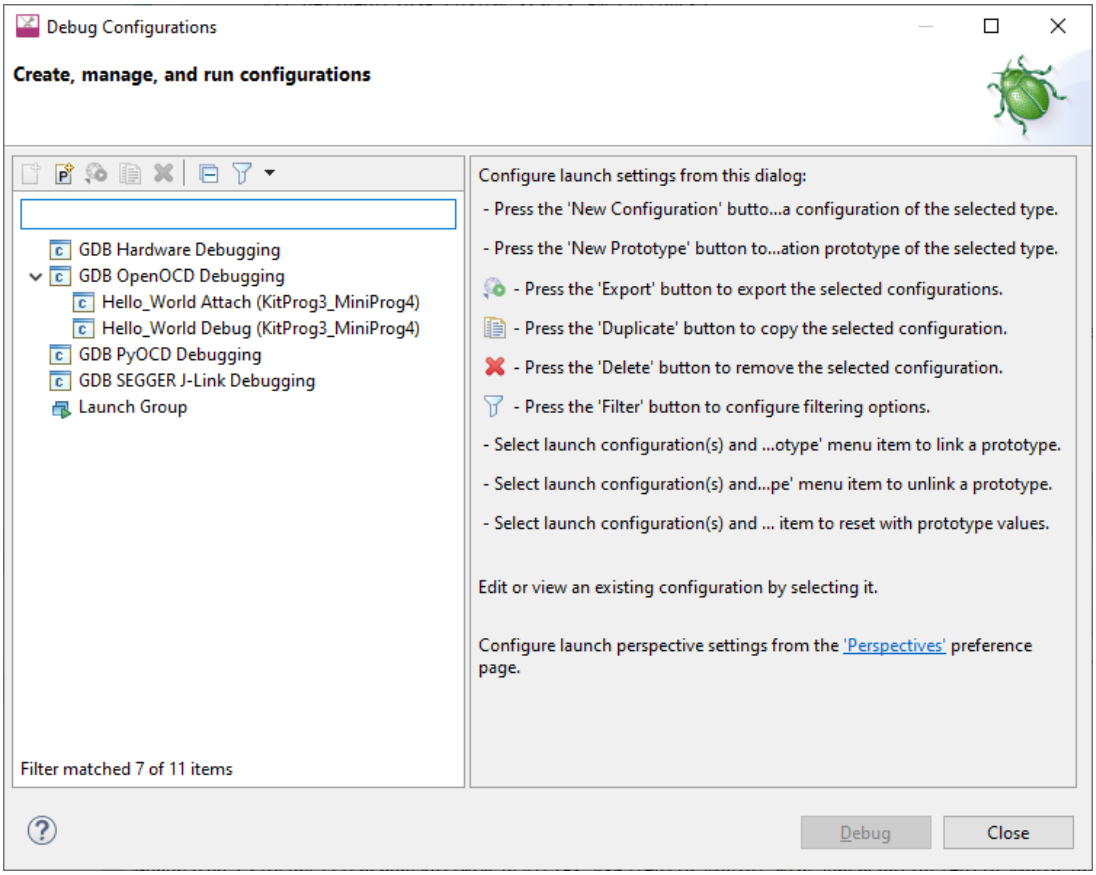
5.1.1 Launch configurations

The flow for programming and debugging is similar for all devices. Eclipse contains several Launch Configurations that control various settings for programming the devices and launching the debugger. Depending on the kit and type of applications you are using, there are various Launch Configurations available.

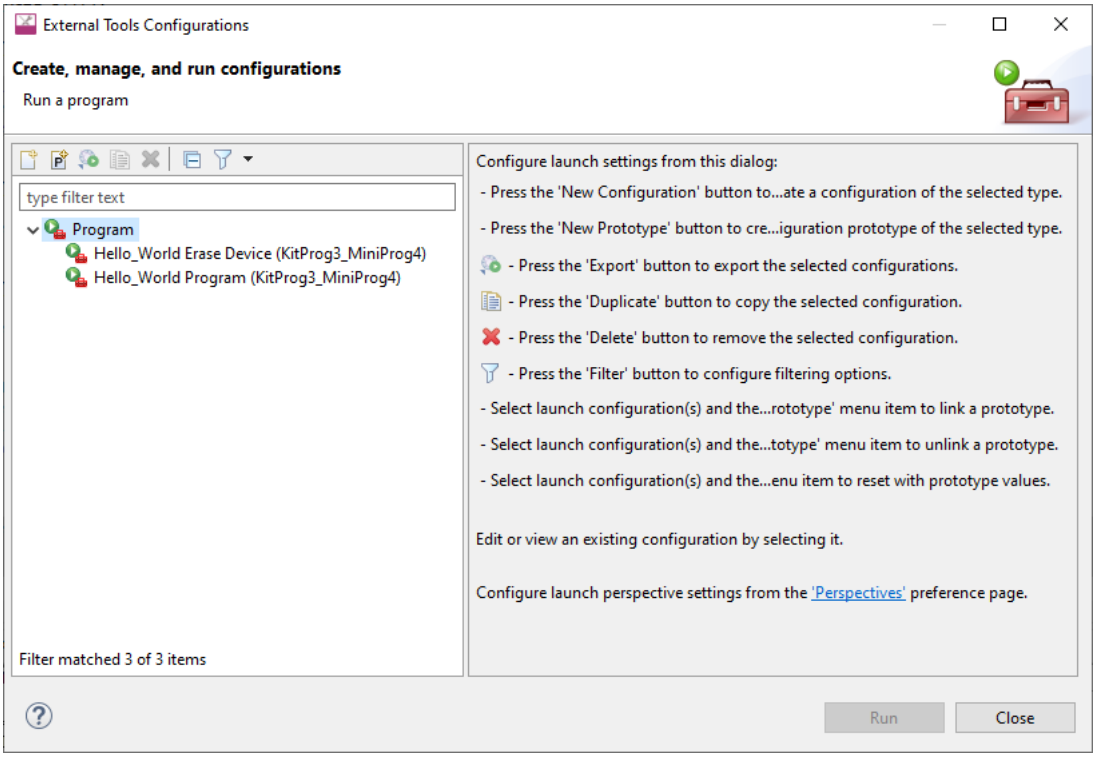
There are two sets of configurations: one for KitProg3_MiniProg4 (included on-board on most Infineon PSOC™ 6 and PSOC™ 4 based kits) and another for SEGGER J-Link.

***Note**: The KitProg3_MiniProg4 launch configuration also supports the MiniProg4, which you attach to the kit via a 10-pin debug connection. See the MiniProg4 product page for details. The MiniProg3 is not supported. See also Supplying power with KitProg3_MiniProg4 later in this document for configuration requirements.*

The Debug/Attach Launch Configurations are shown in the Run/Debug Configurations dialog, similar to the following.



In addition to the Debug and Attach Launch Configurations, Program/Erase configurations are available in the External Tools Configurations dialog. You can open this dialog from the **Run** menu or by selecting the down arrow next to the **Run** and **Debug** commands.



These configurations include the application name and protocol, for example:

- Hello_World Program (KitProg3_MiniProg4)
- Hello_World Debug (JLink) For more information, see Debug connection options.

***Note**: KitProg3_MiniProg4 configurations may not work if a J-Link probe is attached to the kit.*

When an application is created, the tool generates the following launch configurations for KitProg3 by default. You can edit the Makefile to create configurations for J-Link instead. See Specify probe and generate launch configs for further details. Some items display in the Quick Panel, and some are in the Run/Debug Configurations or External Tools Configurations dialogs only.

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

5.1.2 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 clicking the option in the Quick Panel.

5.1.3 Debug connection options

A typical PSOC™ 6 application created for Eclipse may have launch configurations set up for one of two probes: Infineon KitProg3 (built into Infineon kits) or Segger J-Link.

Communication Firmware	Debug Connection	More Information
Infineon-provided KitProg3	OpenOCD via CMSIS-DAP	https://www.infineon.com/cms/en/design-support/tools/programming-testing/psoc-programming-solutions/
SEGGER-provided J-Link DLL	SEGGER J-Link	SEGGER J-Link

5.1.4 KitProg Firmware Loader

The PSOC™ MCU kits include onboard programmer/debug firmware, called KitProg. The CY8CPROTO-062-4343W kit has KitProg3 by default. However, the CY8CKIT-062-BLE and CY8CKIT-062-WIFI-BT kits come with KitProg2 firmware installed, which does not work with the ModusToolbox™ software. **You must update 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.*

5.1.5 Supplying power with KitProg3_MiniProg4

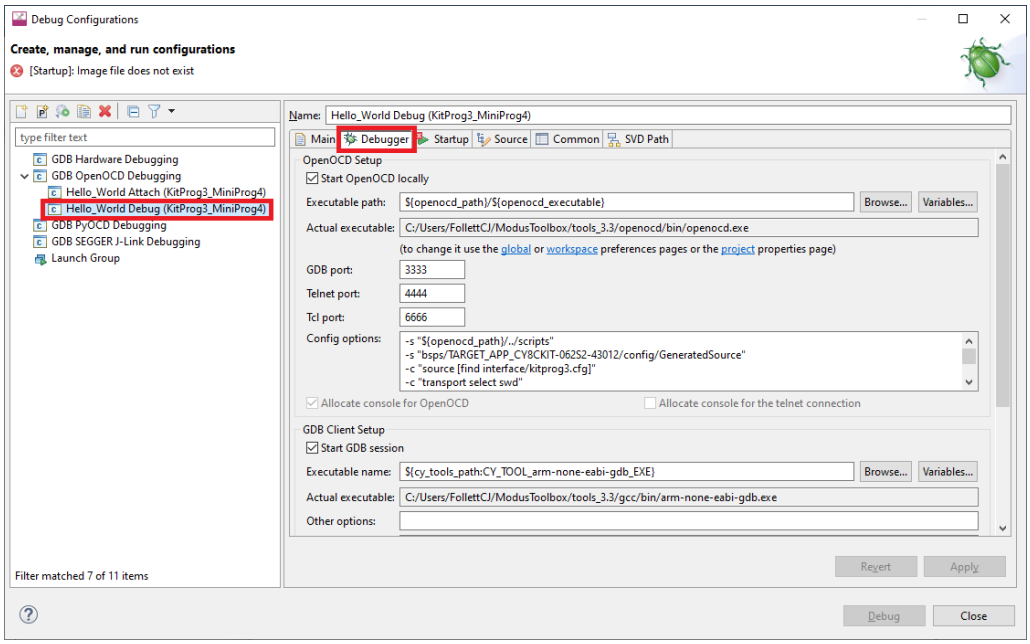
If using the KitProg3 connector on a kit, power is generally supplied by the host PC. When using a MiniProg4, power is not supplied via the MiniProg4 by default. It is expected that the target MCU will be powered externally. However, the MiniProg4 does provide the ability to supply power to the target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

5.1.5.1 Turning power supply on

To turn power supply on, edit the Debug/Attach Launch configurations:

1. Open the Launch Configuration to modify, and select the **Debugger** tab.



2. In the **Config options** field, insert the following line (after -c "source [find interface/kitprog3.cfg]"):

```
-c "set ENABLE_POWER_SUPPLY <mV>"
```

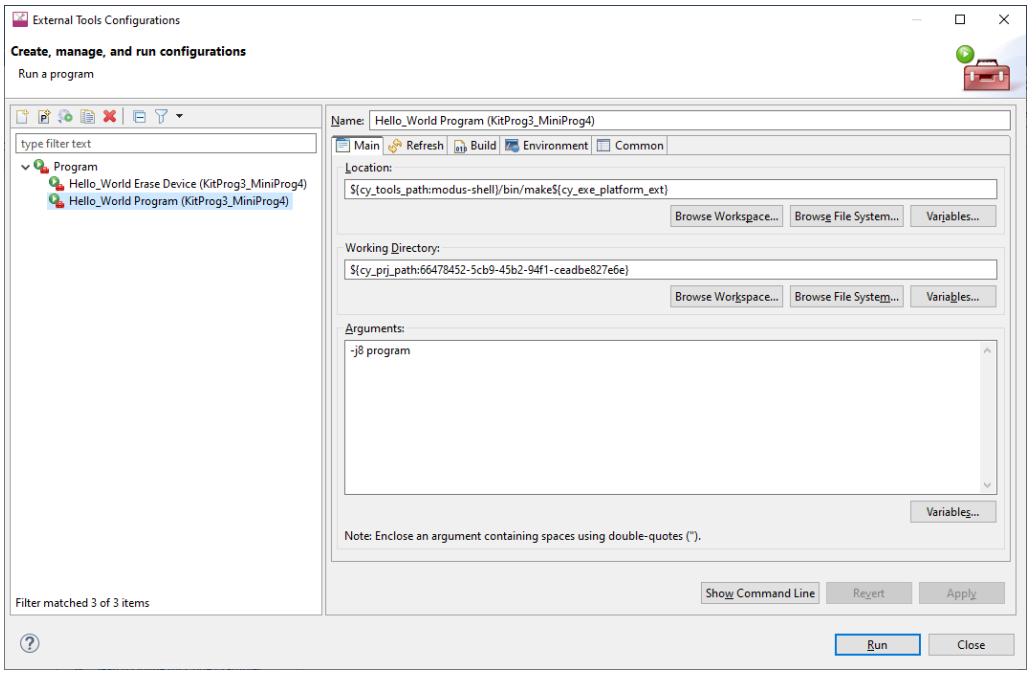
Where `<mV>` defines target voltage in millivolts. For example:



3. Click **Apply** to save the change.

And for the Program/Erase Launch configurations:

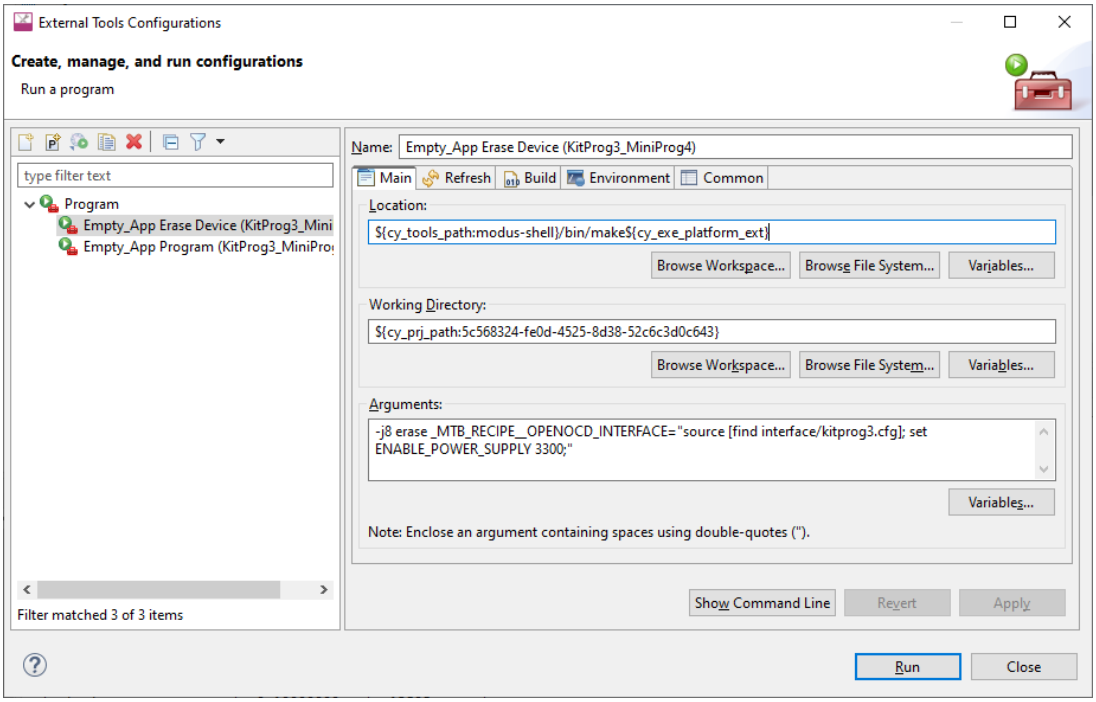
1. Open the External Tools Configuration to modify in External Tools Configurations dialog.



2. In the Arguments field, add the following:

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

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



5.1.6 Power cycle programming mode with KitProg3_MiniProg4

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

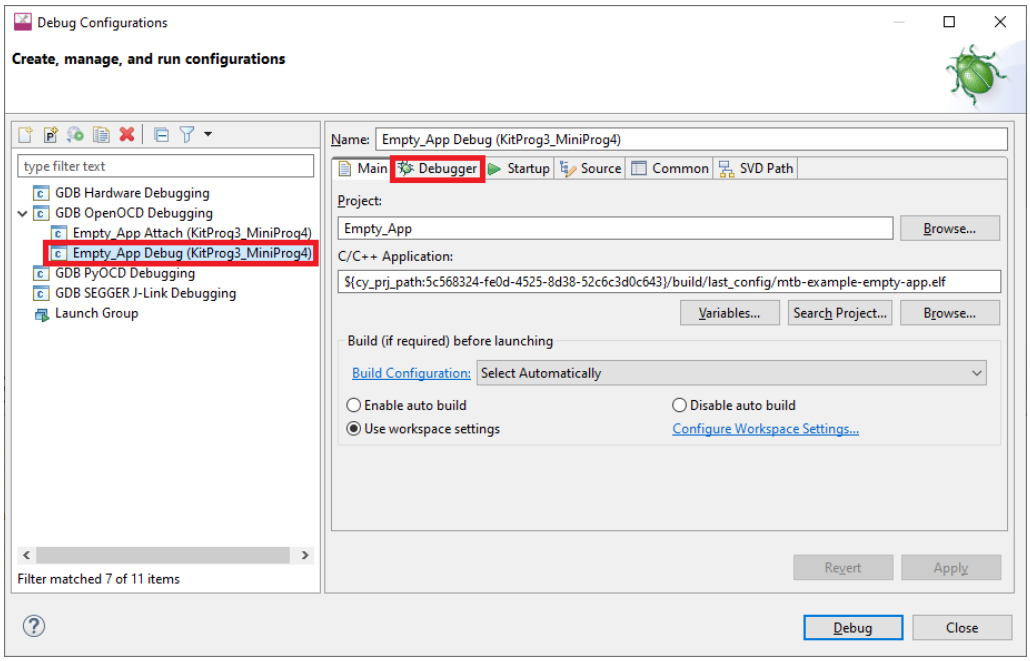
By default, Launch Configurations use Reset mode to program the device. However, Reset mode is not available in all situations (for example, if the XRES pin is not available on the part's package). In these cases, Launch Configurations use an alternative reset using software. However, using the software reset type is not sufficient in some cases when 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.*

To update Debug Launch configurations:

1. Open the Launch Configuration to modify and select the **Debugger** tab.



2. In the **Config options** field, insert the following lines(after `-c "source [find interface/kitprog3.cfg]"`) :

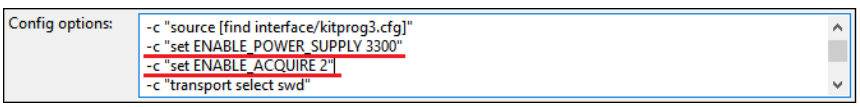
- For PSOC™ 6:

```
-c "set ENABLE_POWER_SUPPLY <mV>"
-c "set ENABLE_ACQUIRE 2"
```

- For PSOC™ 4:

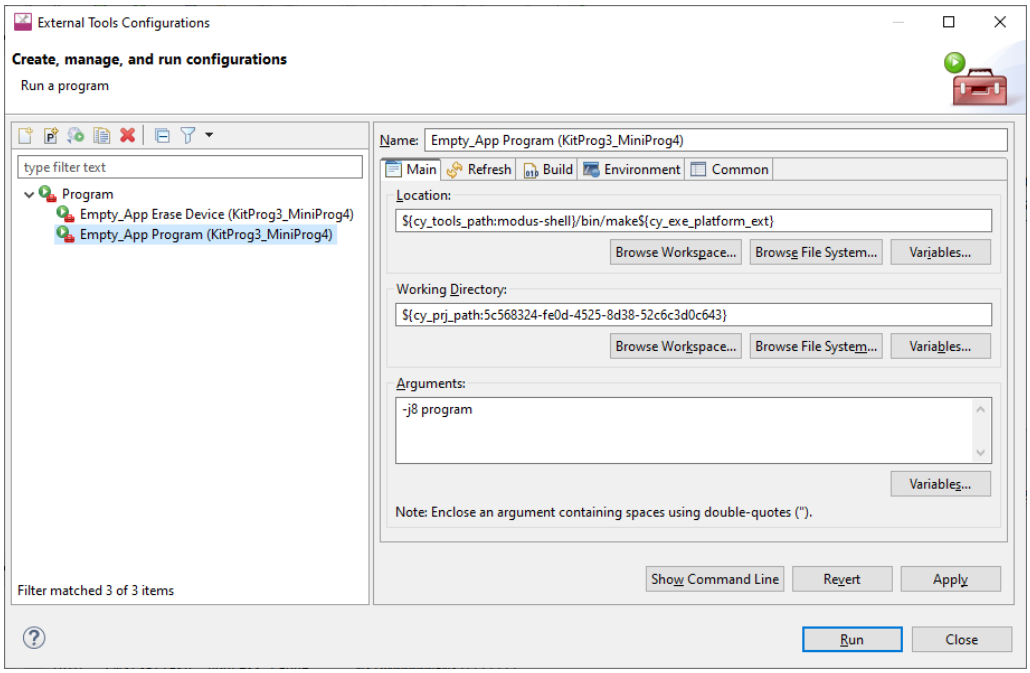
```
-c "set ENABLE_POWER_SUPPLY <mV>"
-c "set PSOC4_USE_ACQUIRE 2"
```

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



And for Program/Erase Launch configurations:

1. Open the External Tools Configuration to modify, then select the **Main** tab.



2. In the **Arguments** field add the following:

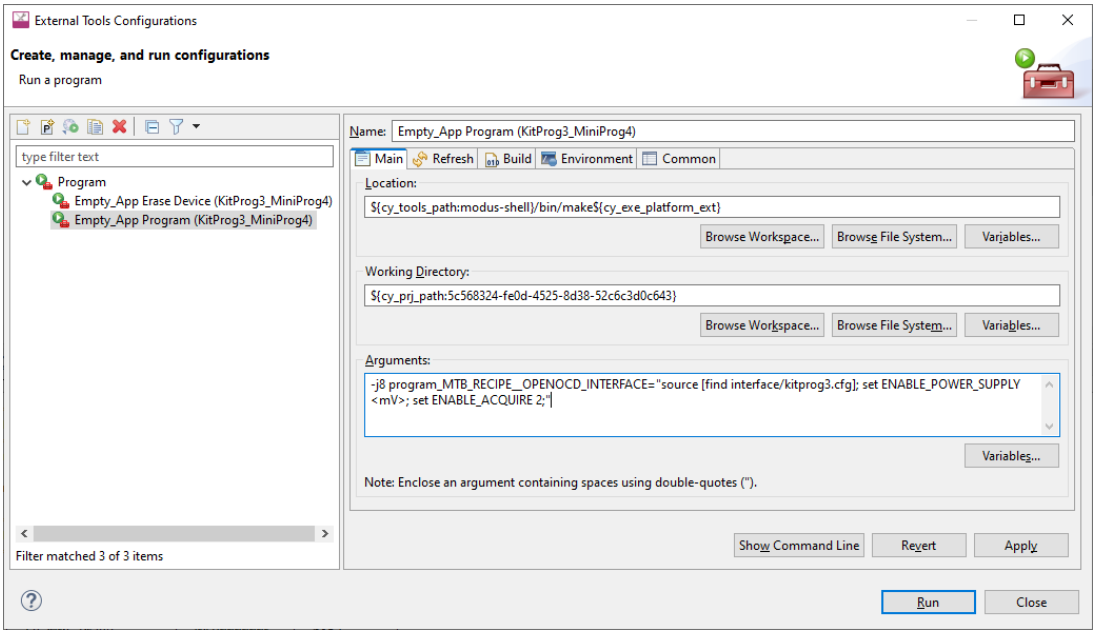
- For PSOC™ 6:

```
_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:

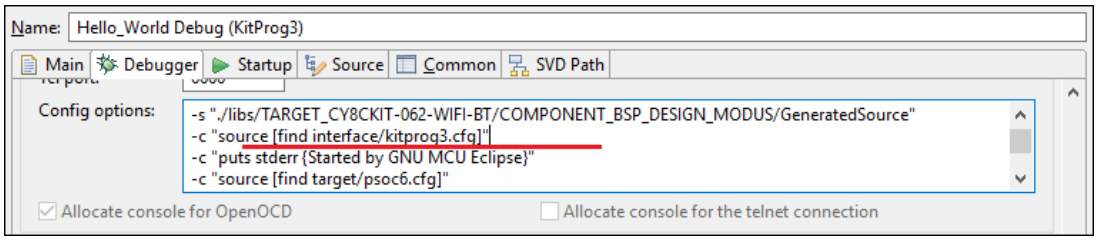


5.1.7 Using JTAG interface in MiniProg4

The MiniProg4 can interact with the target via the SWD or JTAG protocol. By default, the SWD protocol will be used.

In order to use JTAG, you need to add specific commands for the appropriate Debug Configuration. Under the **Debugger** tab in the **Config options** field (after `-c "source [find interface/kitprog3.cfg]"`), insert the following lines:

- `-c "cmsis_dap_vid_pid 0x04B4 0xF151 0x04B4 0xF152"`
- `-c "transport select jtag"`



5.1.8 Changing programming interface SWD/JTAG

Usually MCUs support programming/debugging via SWD and JTAG interfaces. The SWD interface is the default, but you can switch to JTAG, and vice-versa.

To change target interface, update application's bsp.mk by adding make variable as shown below (possible values are 'swd' and 'jtag') and regenerate launch configurations:

```
MTB_PROBE_INTERFACE=swd
```

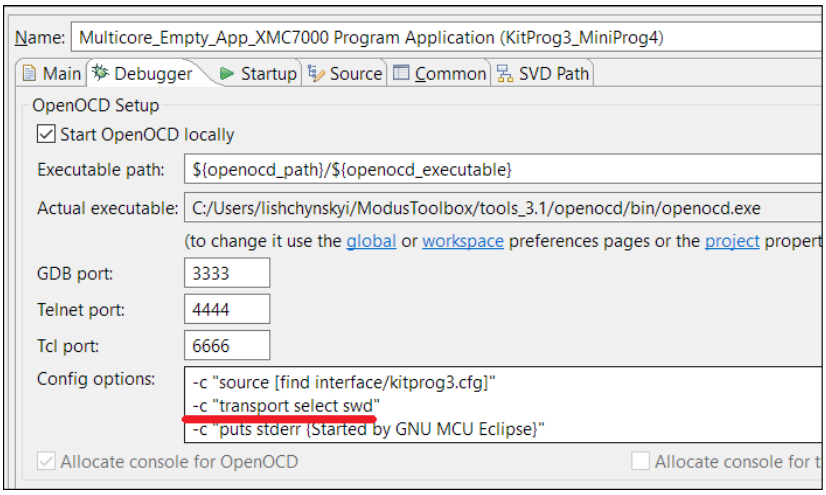
***Note**: There might be a difference in how the MCU functions depending on the selected debugging interface. All known issues are documented in Release Notes.*

5.1.8.1 Single-core projects

Kitprog3/MiniProg4

To change the target interface for KitProg3_MiniProg4, do the following:

1. Open the Debug Configurations dialog and select the required configuration.
2. Switch to the **Debugger** tab and find the "transport select ..." entry in **Config options**.
3. Set to the value to either 'swd' or 'jtag' as applicable.



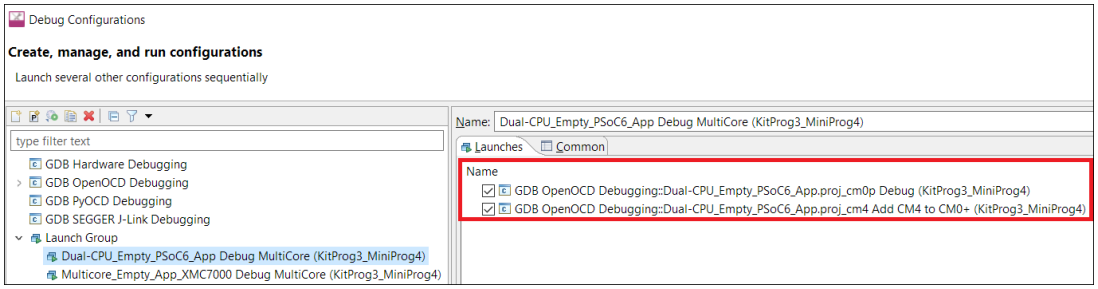
J-Link

For J-Link, there is a dedicated radio button in the Debug Configurations dialog, under the **Debugger** tab. Select the applicable option:



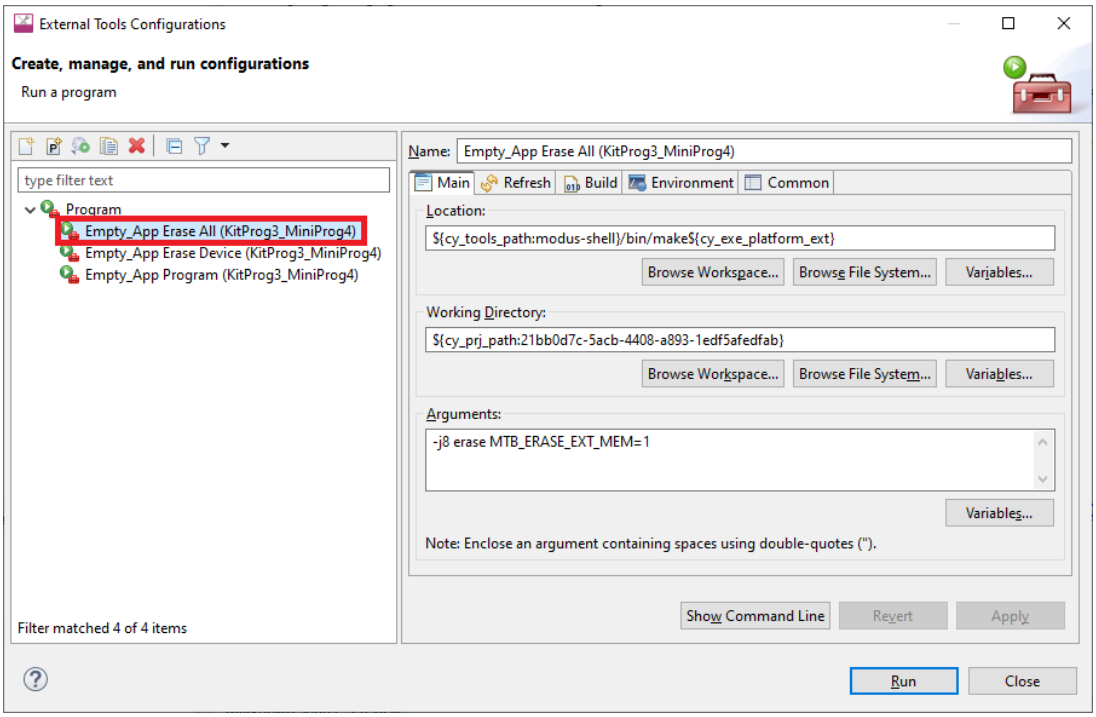
5.1.8.2 Multi-core projects

There is a special case for multi-core debugging. This is usually launched via the 'Debug MultiCore' configuration. To switch from one interface to another, the same interface must be set across all involved configurations. The Debug Configurations dialog shows the list of configurations that are launched as a part of multi-core debugging.



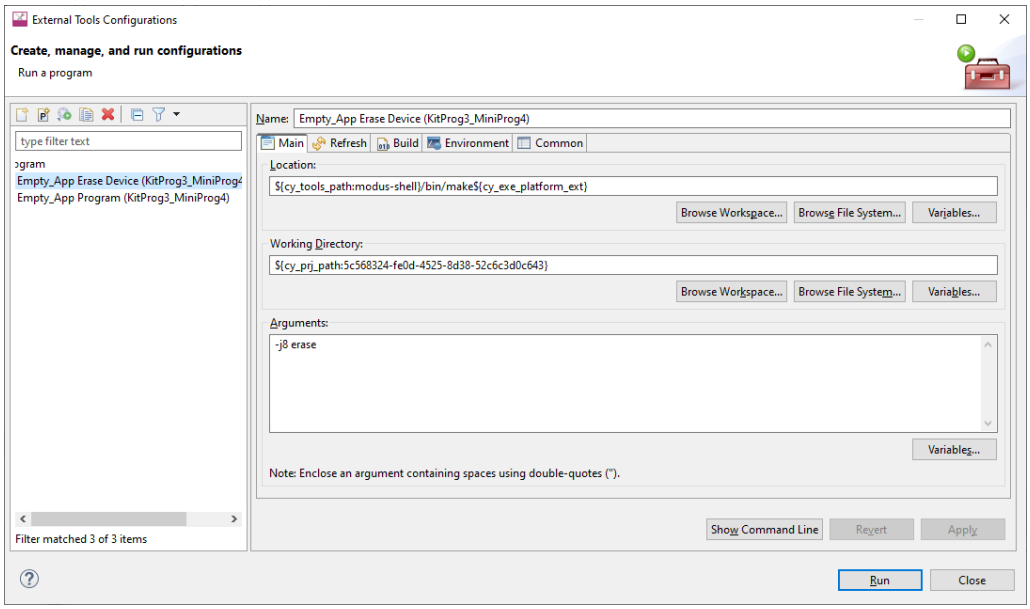
5.1.9 Erasing external memory

To erase external memory use the Erase All configuration:



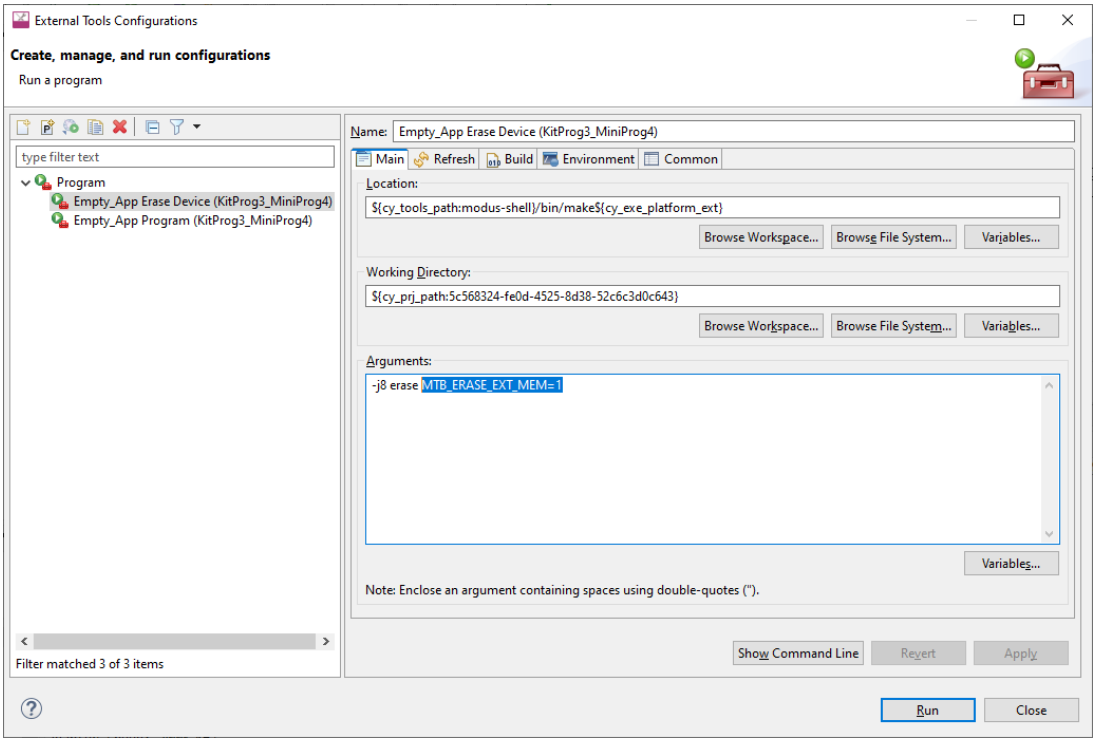
However, for PSOC™ 6 and PSOC™ 64 Secure Boot kits by default, erasing of external memory is turned off. To turn it on, you must modify the "Erase" launch configuration options in the External Tools Configuration as follows:

1. Open the External Tools Configuration to modify, and select the **Main** tab.



2. In the Arguments field add the following:

```
MTB_ERASE_EXT_MEM=1
```



5.1.10 Programming eFuse

PSOC™ 6 MCUs contain electronic fuses (eFuses), which are one-time programmable. They store device specific information, protection settings and customer data.

By default, eFuses are not programmed even if they are present in the programming file. To enable eFuse programming, add the following command for the Program Configuration, under the **Main** tab in the **Arguments** field (after `-j8 program`):

```
"_MTB_RECIPE__OPENOCD_TARGET=source [find target/$(_MTB_RECIPE__OPENOCD_DEVICE_CFG)]; psoc6
allow_efuse_program on;"
```

***Attention**: Since blowing an eFuse is an irreversible process, Infineon recommends programming only in mass production programming under controlled factory conditions and not prototyping stages. Programming fuses requires the associated I/O supply to be at a specific level: the device VDDIO0 (or VDDIO if only one VDDIO is present in the package) supply should be set to 2.5 V (±5%).*

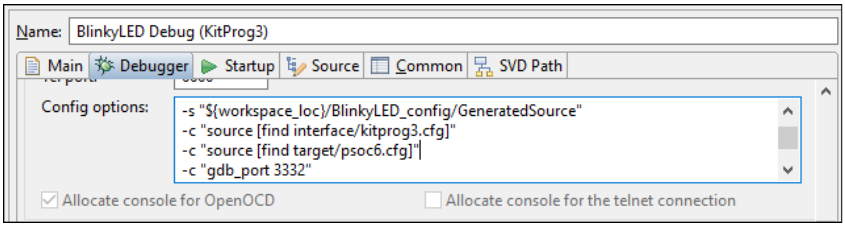
5.1.11 Select specific CMSIS-DAP device

If there are two or more CMSIS-DAP devices connected to your computer, the first detected device will be used by default. KitProg3 supports both CMSIS-DAP modes: HID and BULK. BULK devices are selected first, then HID devices. You can specify a CMSIS-DAP device by:

- VID and PID of the USB device
- Serial Number of the USB device
- VID, PID, and Serial Number of the USB device

There are two ways of specifying a CMSIS-DAP device:

- Add a specific command for the appropriate Debug Configuration, under the **Debugger** tab in the **Config options** field (after `-c "source [find interface/kitprog3.cfg]"`



- Modify ModusToolbox™ application's bsp.mk by adding a `MTB_PROBE_SERIAL` variable, and regenerate launch configurations

5.1.11.1 Selecting by VID and PID

Use OS-specific tools to determine the VID and PID of connected devices. For example, on Windows, use the Device Manager. Use the "cmsis_dap_vid_pid" command to select a CMSIS-DAP device with a specific VID and PID. If there are two or more devices with the same specified VID/PID pair, OpenOCD uses the first detected device from the passed list.

- To specify KitProg3 in CMSIS-DAP BULK mode with VID = 0x04B4 and PID = 0xF155:

```
cmsis_dap_vid_pid 0x04B4 0xF155
```

- To specify KitProg3 in CMSIS-DAP HID mode with VID = 0x04B4 and PID = 0xF154:

```
cmsis_dap_vid_pid 0x04B4 0xF154
```

- To specify any (HID or BULK) connected KitProg3 device:

```
cmsis_dap_vid_pid 0x04B4 0xF154 0x04B4 0xF155
```

5.1.11.2 Selecting by serial number

There should not be more than one device 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 devices. You can also use the fw-loader utility with `--device-list` option. See PSOC™ 4 flash security programming.

Use the "cmsis_dap_serial" command to select a CMSIS-DAP device with a specific Serial Number.

- To specify a CMSIS-DAP device with Serial Number = 0B0B0F9701047400 via Debug Configuration the following command is used:

```
-c "adapter serial 0B0B0F9701047400"
```

- Alternatively, update the application's bsp.mk by adding the variable below, and regenerate launch configurations:

```
MTB_PROBE_SERIAL=0B0B0F9701047400
```

5.1.11.3 Selecting by both VID/PID and serial number

You can use both commands together in any order directly in Debug Configuration. For example:

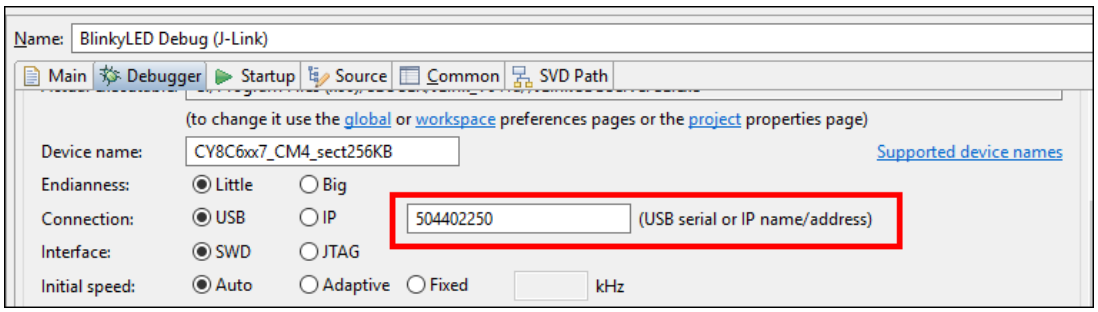
```
cmsis_dap_vid_pid 04B4 F155 cmsis_dap_serial 0B0B0F9701047400
```

When updating through an application's bsp.mk, add the variable below, and regenerate launch configurations:

```
MTB_PROBE_SERIAL=0B0B0F9701047400; cmsis_dap_vid_pid 04B4 F155;
```

5.1.12 Select Specific J-Link Device

If there are two or more J-Link devices connected to your computer, the first detected device is used by default. You can select a specific J-link device by setting the serial number for the appropriate Debug Configuration, under the **Debugger** tab:



Alternatively, update the application's bsp.mk by adding the variable below, and regenerate launch configurations:

```
MTB_PROBE_SERIAL=0B0B0F9701047400
```

5.1.13 PSOC™ 4 flash security programming

PSOC™ 4 devices include a flexible flash-protection system that controls access to flash memory. This feature secures proprietary code, but it can also be used to protect against inadvertent writes to the bootloader portion of flash. Refer to the device's Architecture Technical Reference Manual for details.

Flash consists of rows (64/128/256 bytes, depending on the PSOC™ 4 device). Each row of flash can have its protection level independently set. You can assign one of two protection levels to each row, as shown in the following table:

Protection Setting	Allowed	Not Allowed
Unprotected	External read and write, Internal read and write	–
Full Protection	External read, Internal read	External write, Internal write

To apply flash security, include the appropriate data in the application by creating an array of num_rows / 8 size, where num_rows is the actual number of flash rows on the device. Each byte of this array is responsible for protection of 8 flash rows, and it sets one of two protection levels:

- 0 – unprotected
- 1 – full protection

For example, to protect the first 25 rows and the last 5 out of 256 rows, include the following array in your application:

```
CY_SECTION(".cyflashprotect") __USED
static const unsigned char flash_protect[0x20] =
{
  0xFF, 0xFF, 0xFF, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00,
  0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
};
```

To enable flash security, program the device with the updated code. If you wish to reprogram a protected device, you must first erase it.

5.1.14 Debugging/programming for 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 further details about PSOC™ Control C3 and PSOC™ Edge E8 MCU security, refer to the following 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.

5.1.14.1 OpenOCD

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

Make Variable (Program and Debug)

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 via the Quick Panel.*

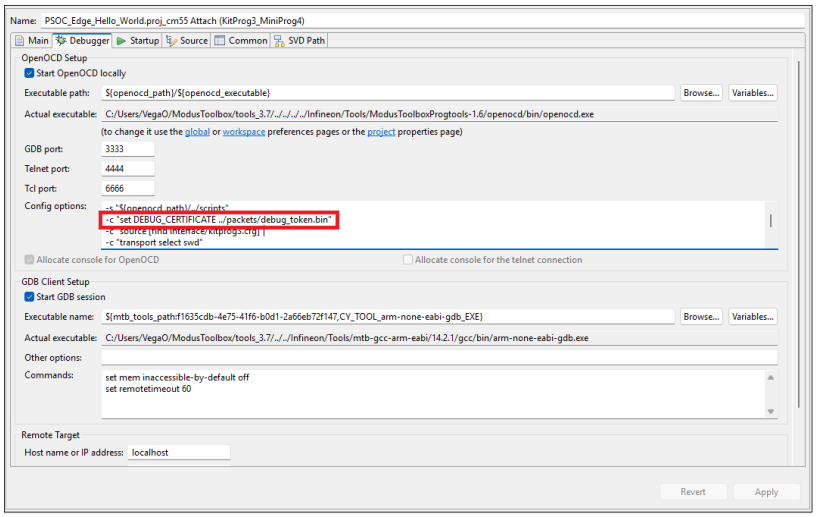
You can also run the make variable in a terminal, and it will only apply to the specific make command. For example:

```
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:

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



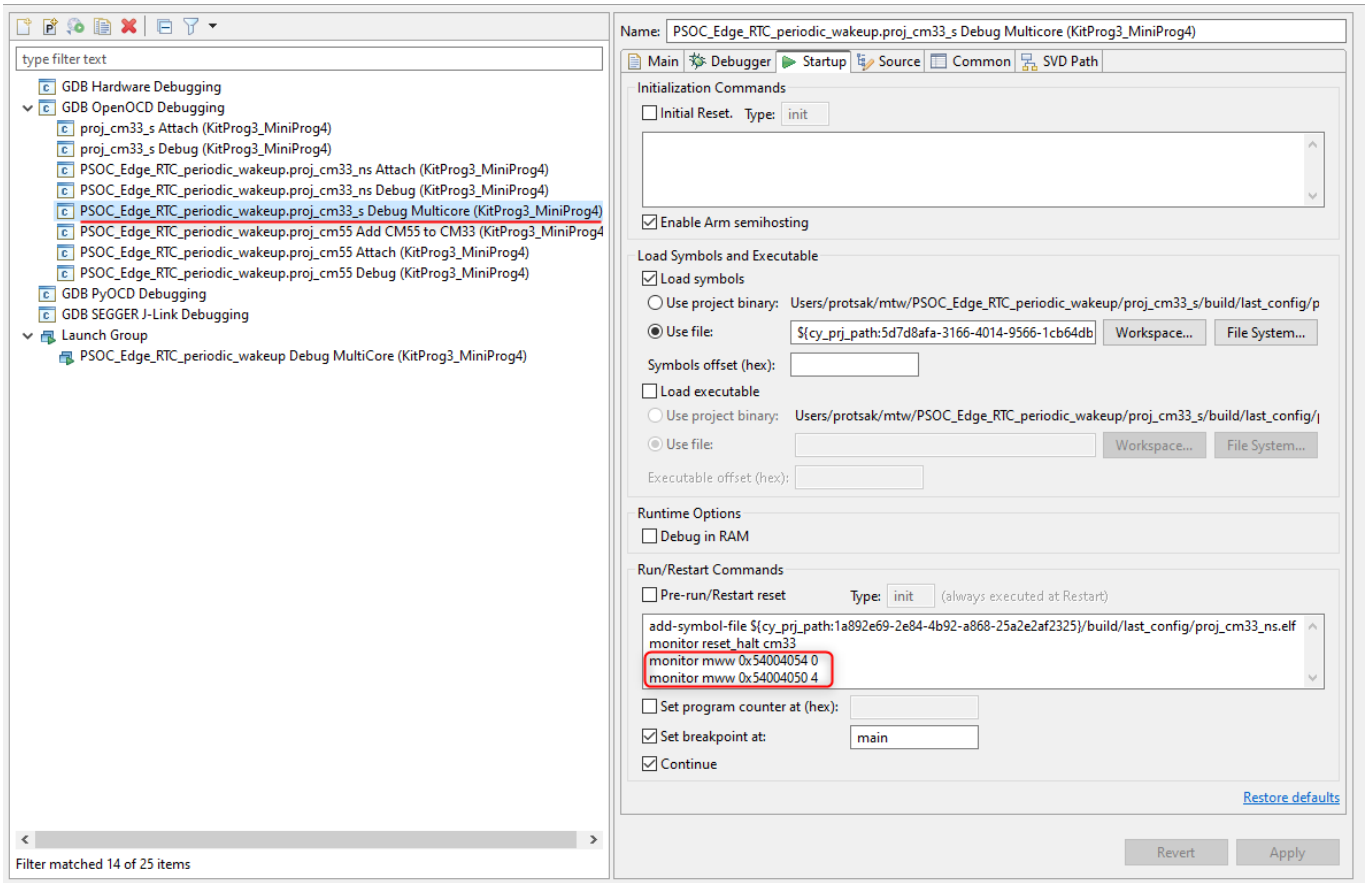
5.1.14.2 J-Link

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.

5.1.14.3 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.



Add the following under the **Startup** tab under **Run/Restart Commands**:

For KitProg3/MiniProg4

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

For J-Link

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

5.2 AIROC™ Bluetooth® programming/debugging

AIROC™ Bluetooth® applications have different launch configurations than PSOC™ MCU applications. Bluetooth® applications use External Tools configurations for programming the device, and Attach configurations for attaching to the debugger. The Program launch configurations are available from the Quick Panel. Each configuration is preceded by the application name.

5.2.1 Program configuration

This launch configuration runs a script and programs the device's memories.

- **<app-name> Program**: Program is used to build and program embedded applications onto the target device.

5.2.2 Attach configurations

The Attach configurations are used to attach to the debugger without first programming the device. They include:

- **<app-name> Attach_JLink**
- **<app-name> Attach_KitProg3_MiniProg4**

5.2.3 Debug settings

AIROC™ Bluetooth® devices use the JTAG SWD interface on a kit for debug via OpenOCD or J-Link probe. Typically, GPIOs need to be reprogrammed (via firmware) to enable SWD, so debug can't be performed directly after a device reset. The debugger is usually attached to a running device and symbols only are loaded.

The CYW920819EVB-02 and CYBT-213043-MESH kits have additional requirements in order to launch Eclipse debugger. In general, most debugging for these kits is done with logs and sniffers, since real time execution is usually needed for the protocols to run properly.

Refer to the AIROC™ Hardware Debugging document for more details.

5.3 XMC1xxx/4xxx devices programming/debugging

5.3.1 BMI for XMC1xxx/4xxx and PSOC™ Control C1xxx 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.

```
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.

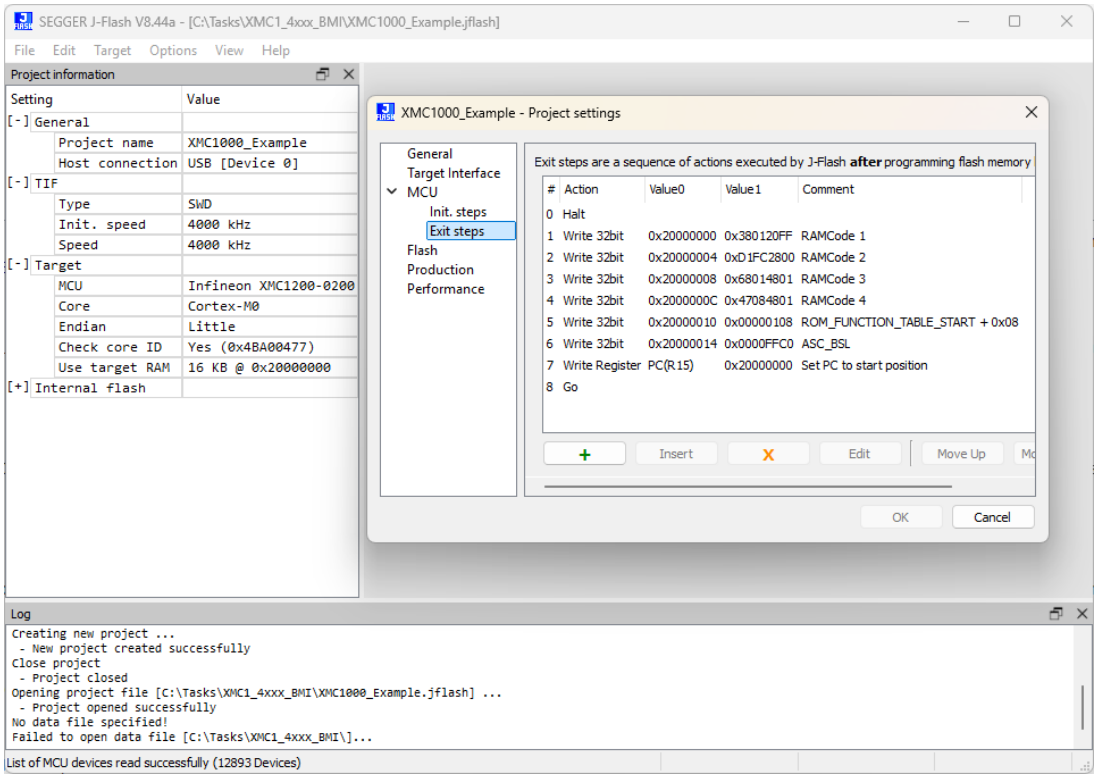
```
Connecting to target via SWD
Trying to identify target via SPD
Could not identify target via SPD. Trying again via SWD.
Found SW-DP with ID 0x0BB11477
DPIDR: 0x0BB11477
CoreSight SoC-400 or earlier
Scanning AP map to find all available APs
AP[1]: Stopped AP scan as end of AP map has been reached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



Trademarks

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

Edition 2026-09-11

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-txj1712264844100

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.