ModusToolbox™ IDE

User Guide

Document # 002-24375 Rev **

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709
http://www.cypress.com

Contents

1 Introduction

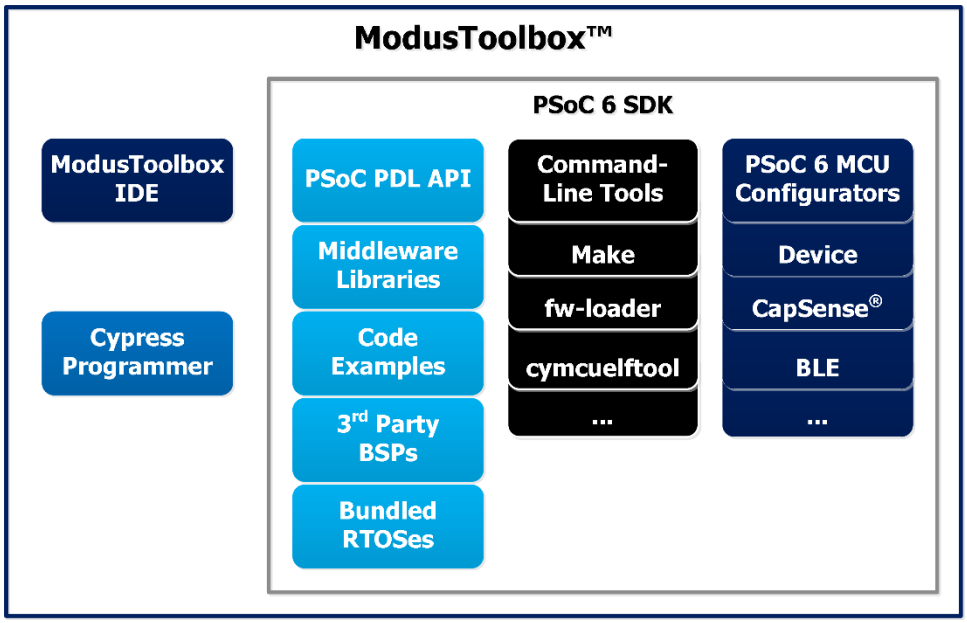
Overview

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

In the IDE, the top level entity that you ultimately program to a device is called an application. The application consists of one or more Eclipse projects. The IDE handles all of the dependencies between projects automatically. It also provides hooks for launching various tools provided by the software development kit (SDK).

The SDK provides the central core of the ModusToolbox software. It contains Configurators, drivers, libraries, middleware, as well as various utilities, Makefiles, and scripts. You may use any or all of these tools in any environment you prefer.

The following shows a high-level view of the tools included in the ModusToolbox software.



This document provides information about creating applications as well as building, programming, and debugging them. This guide primarily covers the ModusToolbox IDE aspects of these concepts. It also covers various aspects of the SDK.

References

The following documents should be referenced as needed for more information about a particular topic, For more information, see Documentation.

- ModusToolbox Installation Guide (available online)
- Configurator Documentation (open from a particular Configurator)
- Eclipse Documentation (available from Eclipse)
- Eclipse Survival Guide (available online)

Document Conventions

The following table lists the conventions used throughout this guide:

Convention	Usage
Courier New	Displays file locations and source code: `C:\ …cd\icc\, user entered text`
Italics	Displays file names and reference documentation: *sourcefile.hex*
[**bracketed, bold**]	Displays keyboard commands in procedures: [**Enter**] or [**Ctrl**] [**C**]
File > New Application	Represents menu paths: **File > New Application > Clone**
Bold	Displays commands, menu paths and selections, and icon names in procedures: Click the **Debugger** icon, and then click **Next**.
Text in gray boxes	Displays cautions or functionality unique to ModusToolbox software.

Revision History

Document Title: ModusToolbox™ IDE User Guide **Document Number: 002-24375**		
Revision	**Date**	**Description of Change**
**	11/21/18	New document.

2 Getting Started

This section provides a basic walkthrough for how to create a couple starter applications using the IDE, selecting either a kit or custom hardware. It also covers how to build and program them using the IDE and basic launch configurations supplied for the applications.

- Launch ModusToolbox IDE
- Create Starter Applications
- Download/Import Code Examples
- Build Starter Application
- Program Starter Application

Launch ModusToolbox IDE

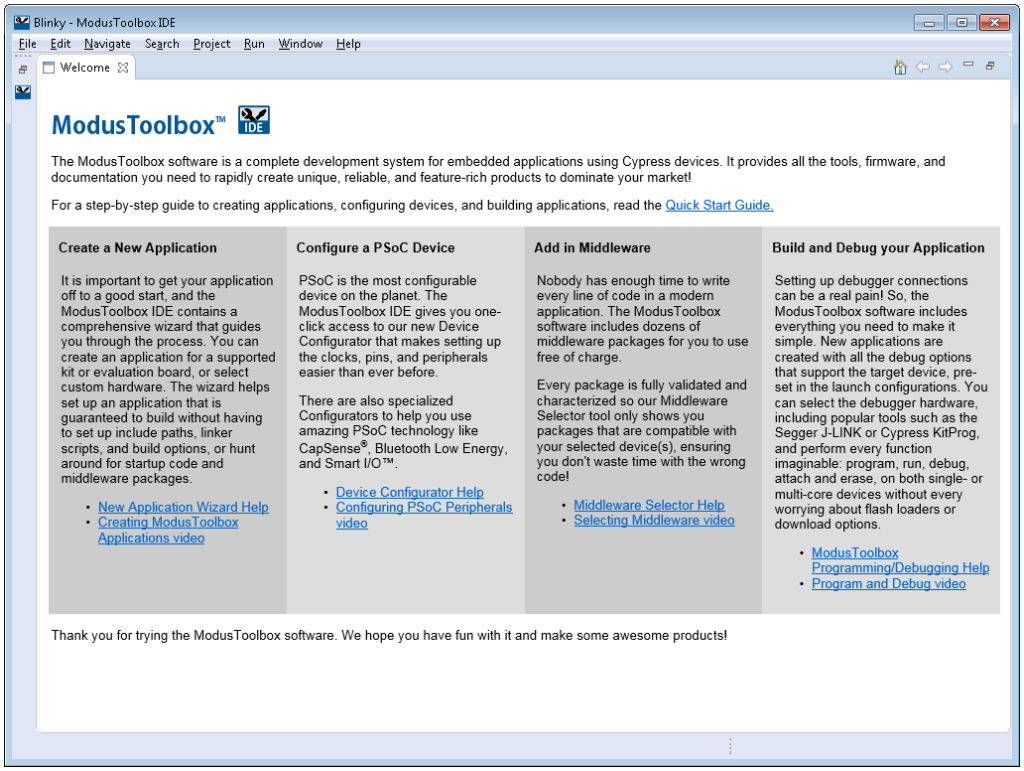
The ModusToolbox IDE is installed in the following directory in Windows, by default:

<user_home>\ModusToolbox_1.0\eclipse

For other operating systems, the installation directory will vary, based on where the software was installed.

Run the “ModusToolbox” executable file to launch the IDE as applicable for your operating system. Refer to the *Modus Toolbox Installation Guide* for more information.

When launching the ModusToolbox IDE, it provides an option to select the workspace location on your machine. This location is used by the IDE for creating and storing the files as part of application creation for a particular platform. When the IDE opens the first time (or for any new workspace), it shows the Welcome Page. See Welcome Page for more information.



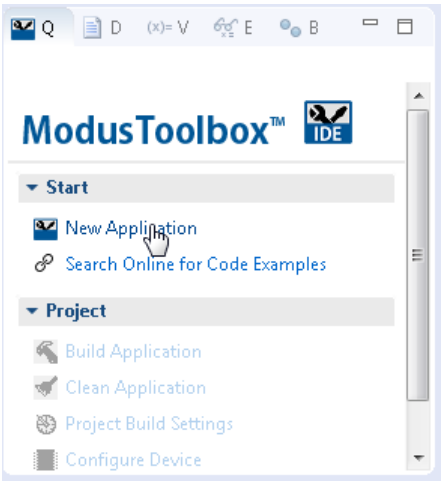
Click the **X** on the Welcome Page tab to close it, or click the **Restore** button to move it to the side.

Create Starter Applications

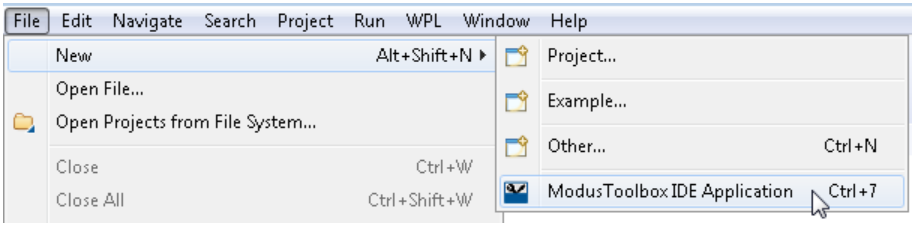
The ModusToolbox IDE provides several starter applications for use with different development kits. Refer to the *ModusToolbox IDE Release Notes* for the supported platforms. This section provides a walkthrough for creating an application by selecting a kit, and another application by selecting a chip.

Open New Application Dialog

Click the **New Application** link in the ModusToolbox IDE Quick Panel.

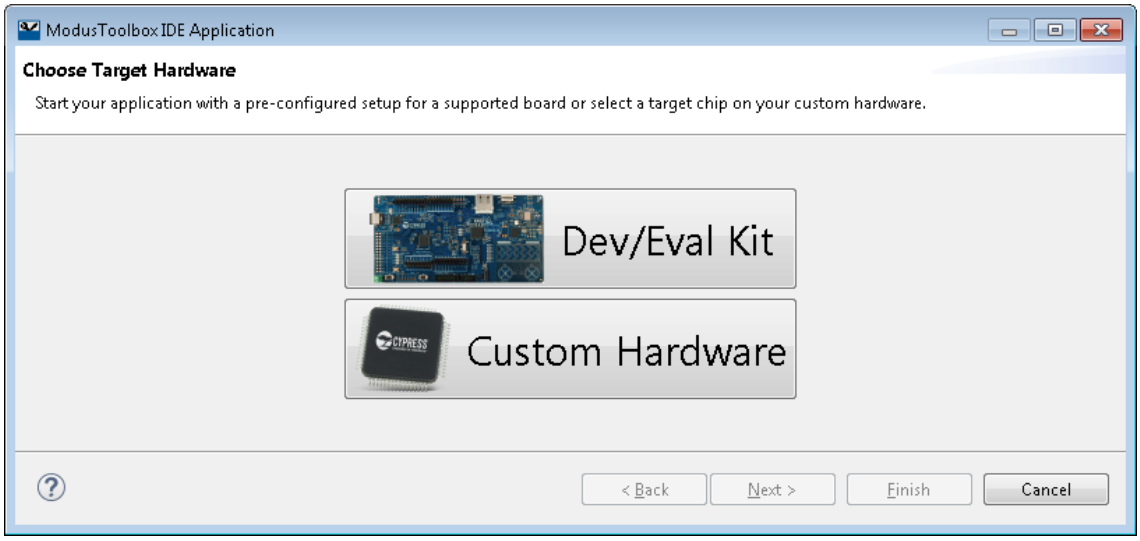


You can also select **File > New > ModusToolbox IDE Application**.



Select Kit or Hardware

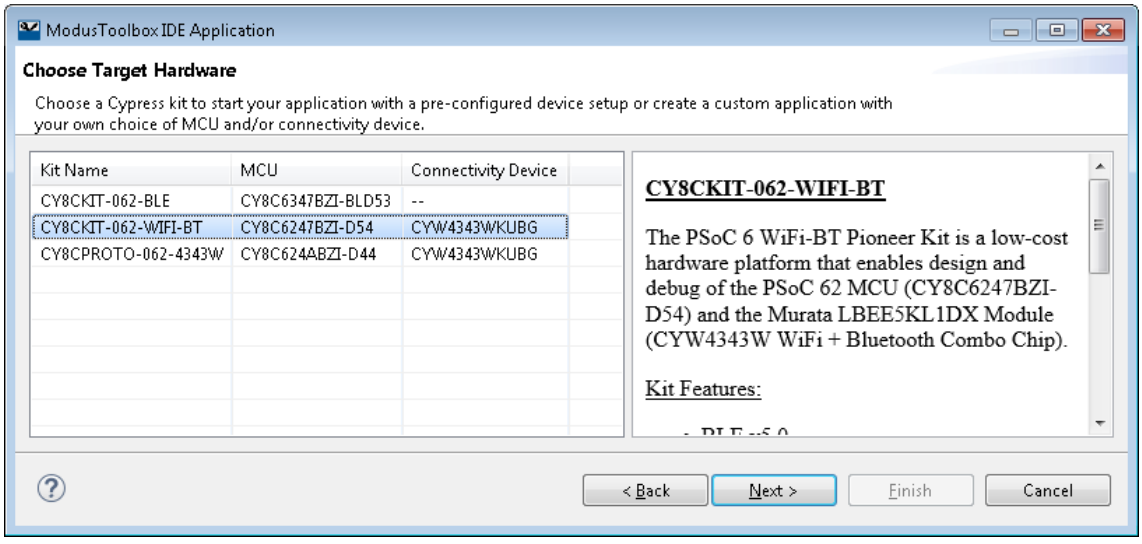
On the ModusToolbox IDE Application dialog, chose the option to start your application from a kit or custom hardware (a chip).



- See Dev/Eval Kit.
- See Custom Hardware.

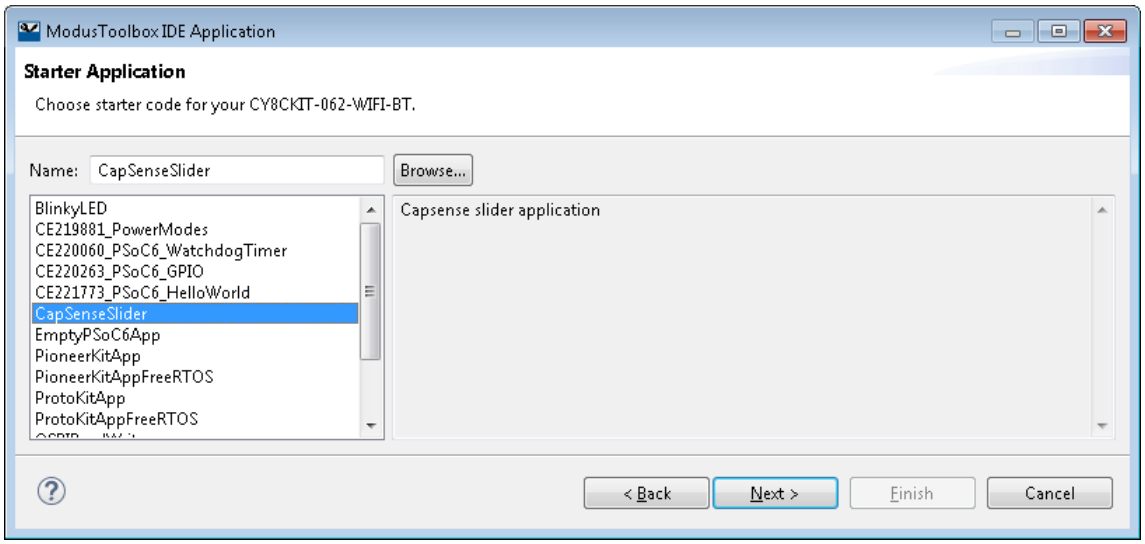
Dev/Eval Kit

If you choose the “Dev/Eval Kit” option in the first step, the dialog displays a list of kits, showing the Kit Name, MCU, and Connectivity Device (if applicable). As you select each of the kits shown, the description for that kit displays on the right.



For this example, select the CY8CKIT-062-WIFI-BT kit.

Click **Next >** to open the Starter Application page. This page lists various starter applications available for the selected kit. As you select a starter application, a description displays on the right.

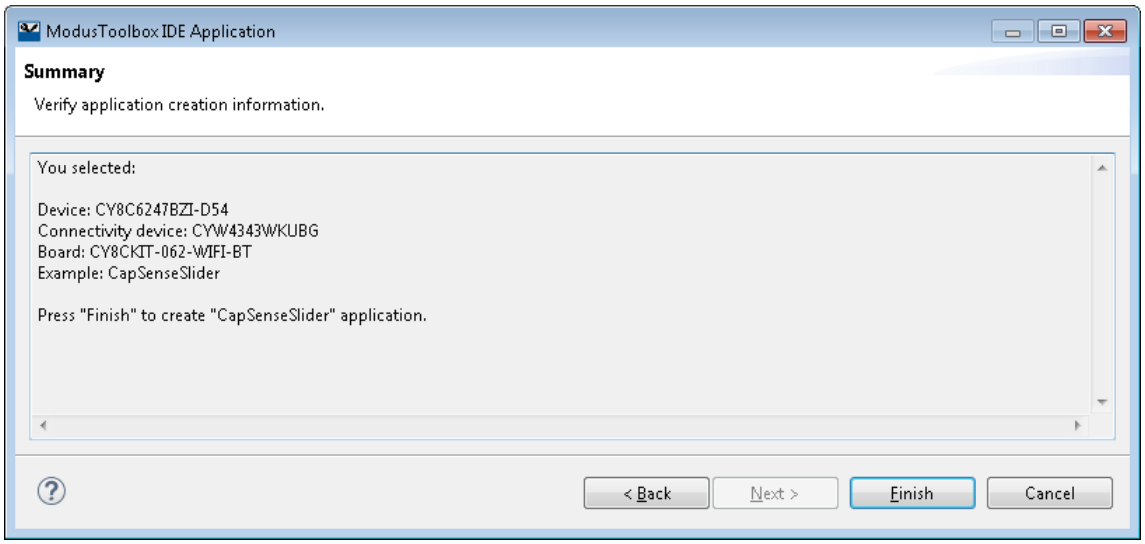


For this example:

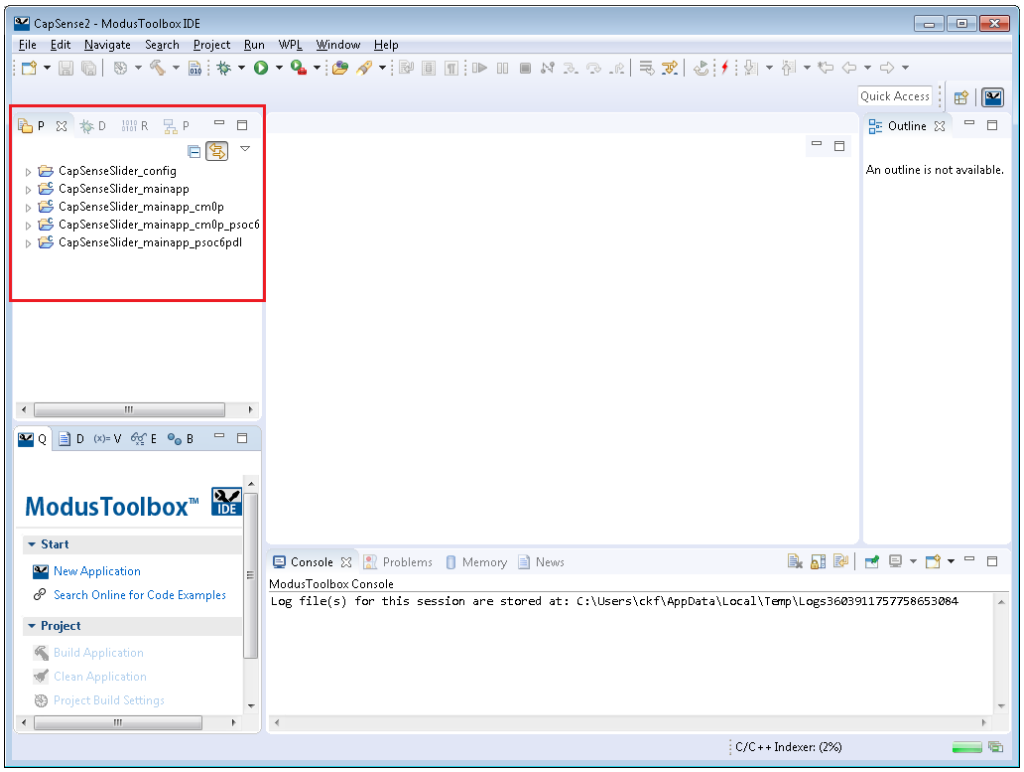
- Select the ”CapSenseSlider” application from the list.
- Type a name for your application. Do not use spaces in the application name. In this case, “CapSenseSlider” is the default name.

Note You can use the **Browse…** button to select other examples. Many examples are available for download from the web, or you may have received an example from a colleague. In the Open dialog, select only examples that are supported by the hardware you selected for this application. Then, the example will be shown in the New Application dialog with all the other Starter Applications. See also Import Code Examples for additional details.

Click **Next >** to open the Summary page. This page shows the various options chosen for this application. Review them to ensure they are correct.



Click **Finish** to create the application. After a few moments, the ModusToolbox IDE displays progress information. When complete, the application's project folders display in the Project Explorer.

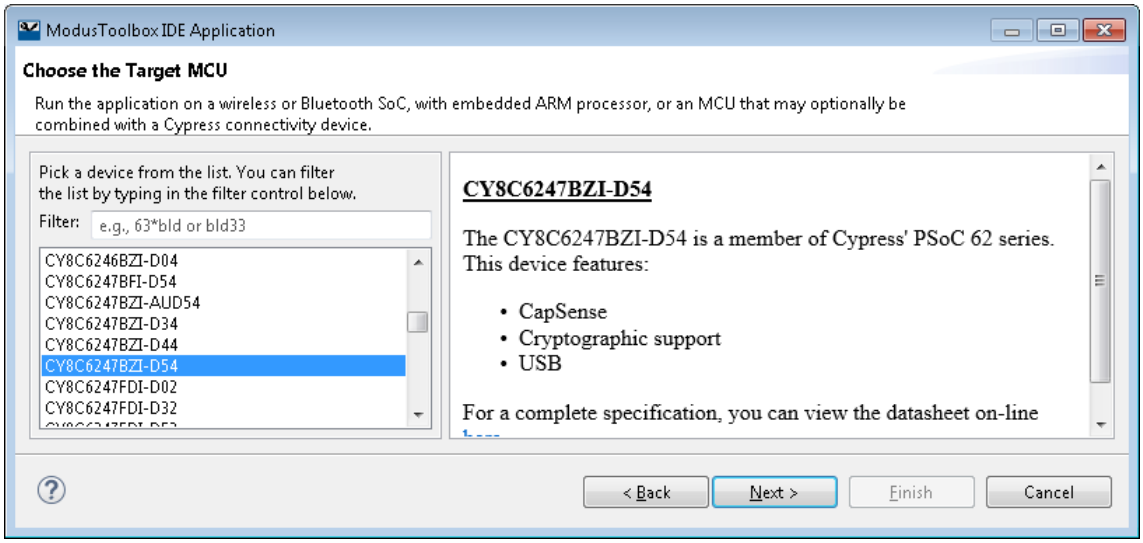


To learn how to build the application, go to the Build Starter Application section.

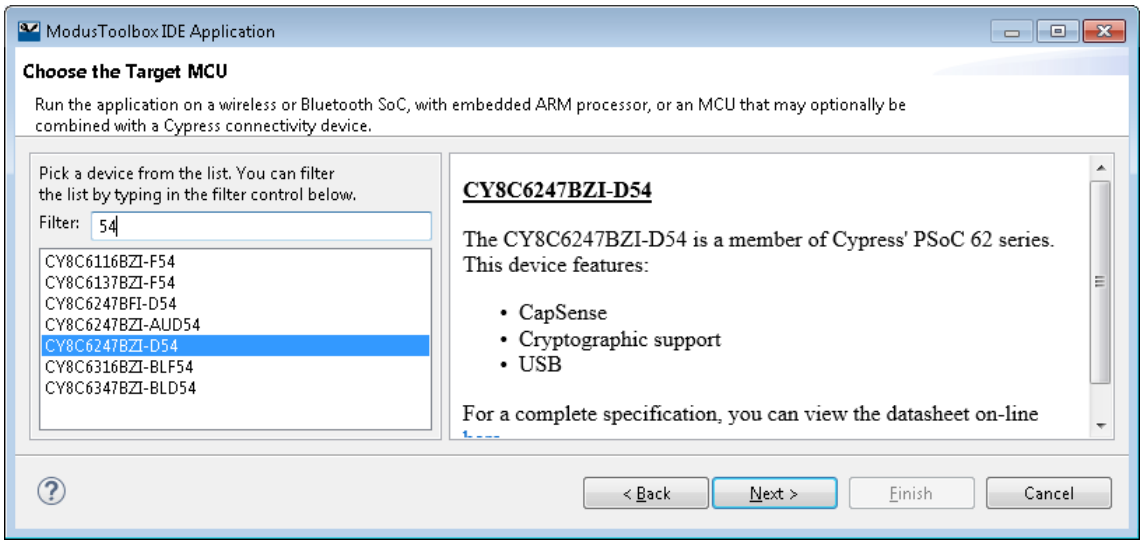
Custom Hardware

If you choose the “Custom Hardware” option in the Select Kit or Hardware step, the dialog displays a list of devices by part numbers. When you select a part, the description displays on the right.

Note At this time, this option only applies to PSoC devices.

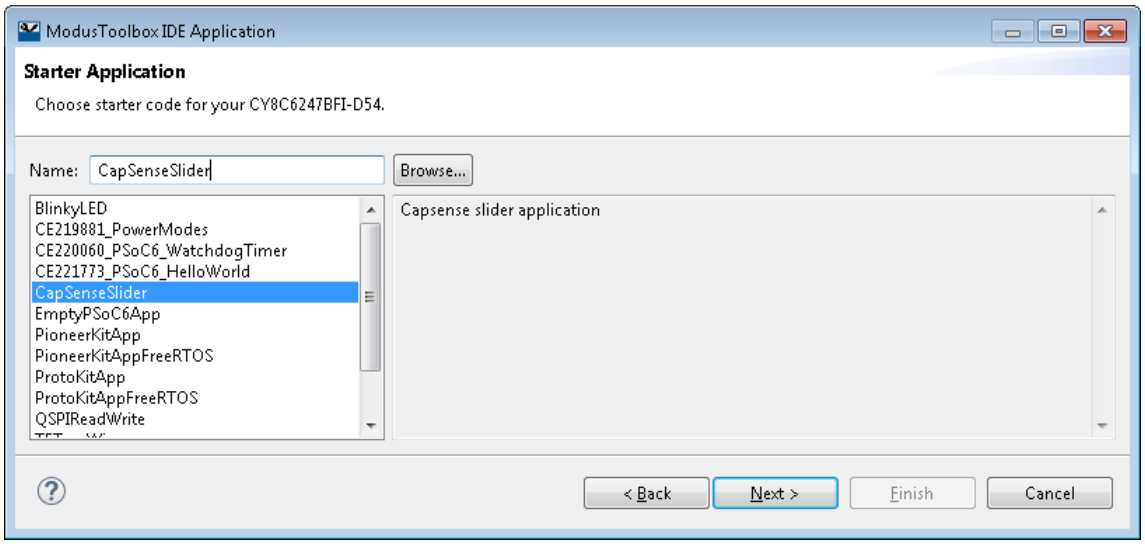


You can type in the **Filter** box to limit the number of devices displayed.



For this example, select the “CY8C6247BZI-D54” device.

Click **Next >** to open the Starter Application page. This page lists various starter applications available for the selected device. As you select a starter applications, a description displays on the right.

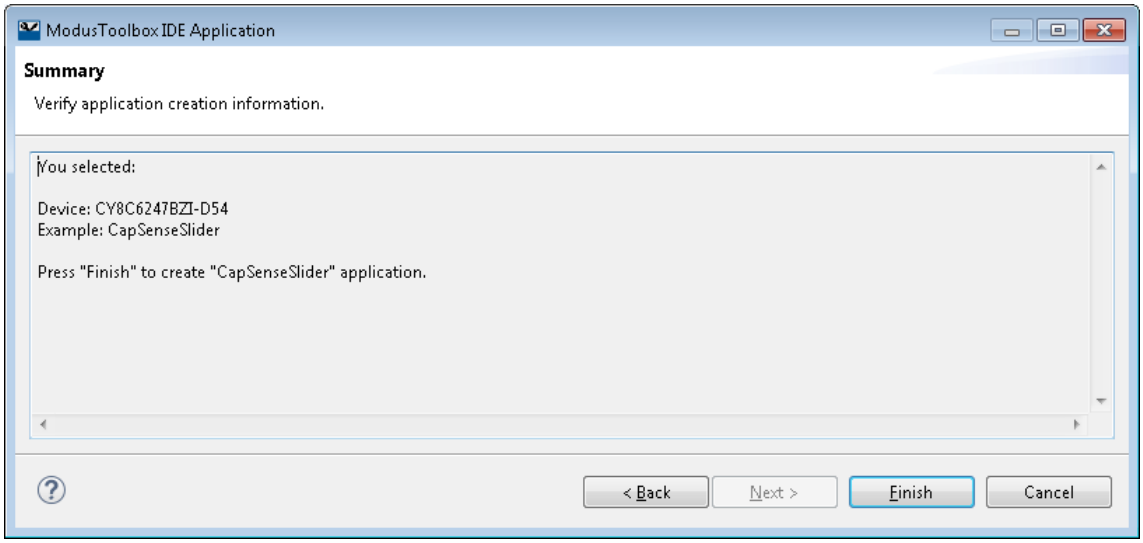


For this example:

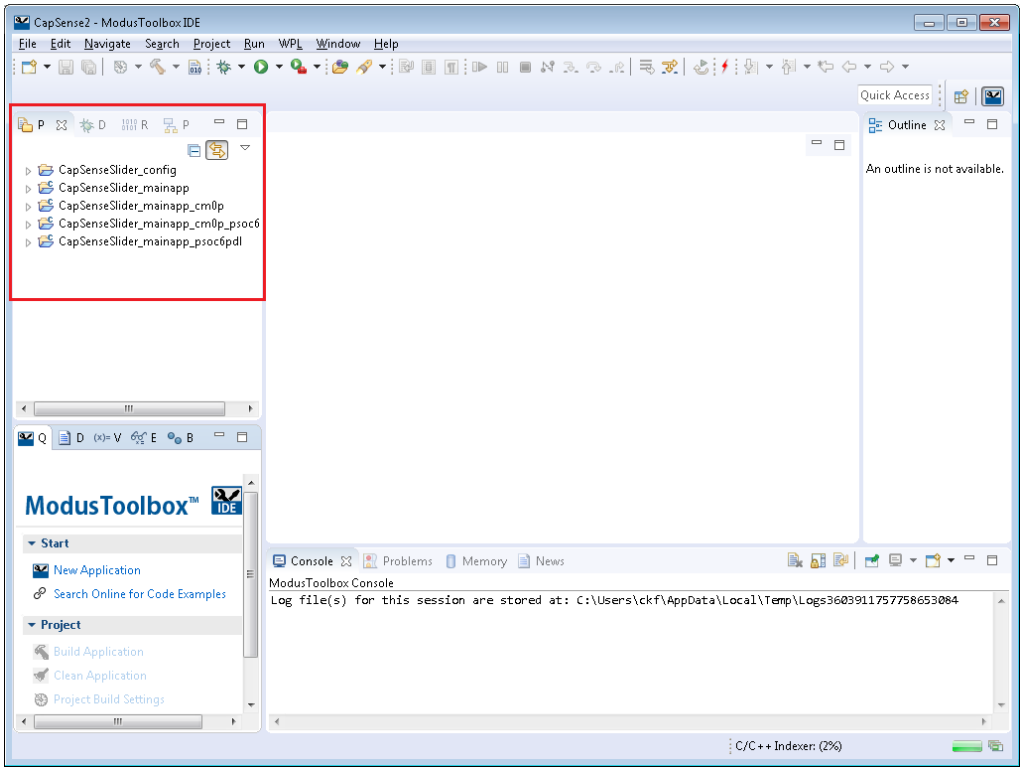
- Select the CapSenseSlider application from the list.
- Type a name for your application. Do not use spaces in the application name. In this case, “CapSenseSlider” is the default name.

Note You can use the **Browse…** button to select other examples. Many examples are available for download from the web, or you may have received an example from a colleague. In the Open dialog, select only examples that are supported by the hardware you selected for this application. Then, the example will be shown in the New Application dialog with all the other Starter Applications. See also Import Code Examples for additional details.

Click **Next >** to open the Summary page. This page shows the various options chosen for this application. Review them to ensure they are correct.



Click **Finish** to create the application. After a few moments, the ModusToolbox IDE will display progress information. When complete, the application's project folders will display in the Project Explorer.



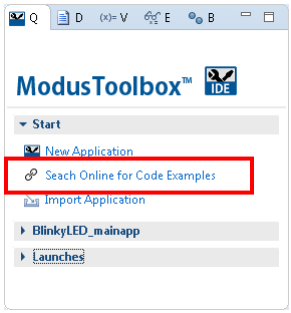
To learn how to build the application, go to the Build Starter Application section.

Download/Import Code Examples

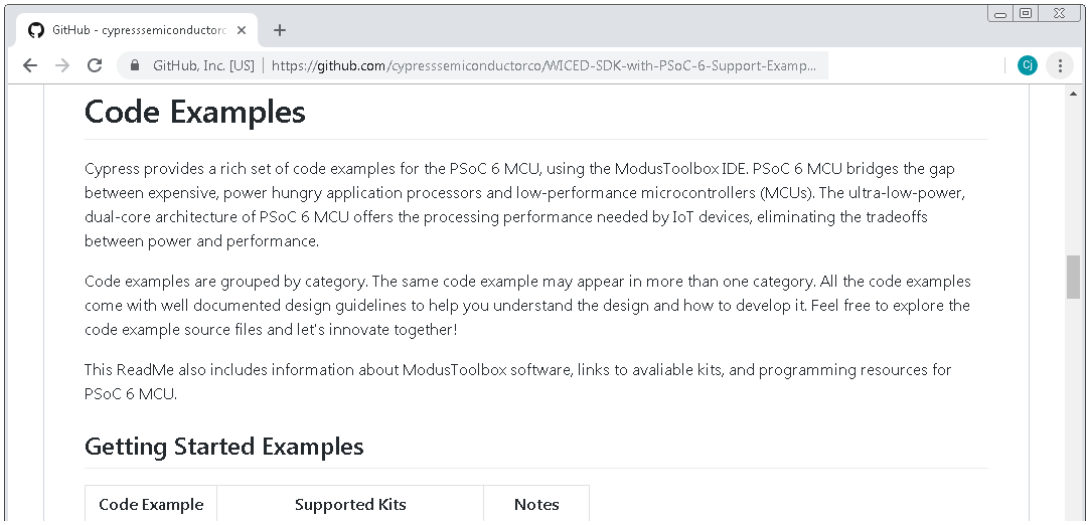
Cypress provides many code examples. These examples allow you to explore the capabilities provided by the SDK, create applications based on them, examine the source code demonstrated in them, and read their associated documentation.

Download from GitHub

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.



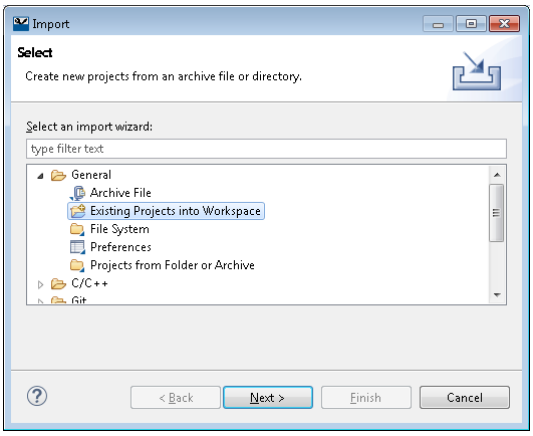
Examples are collected into repositories on the Cypress GitHub site. You can clone or download a repository to your computer using GitHub mechanisms. You can also import a GitHub repository directly to your workspace. Use **File > Import > Git > Projects from Git** and follow the instructions in the wizard. This is standard Eclipse functionality.

Importing the repository in this way does not create an application you can build. It does bring in all the source code for all the examples in the repository for you to examine. If you instead want an application that you can build, follow the instructions in the next section to import a code example as a ModusToolbox application.

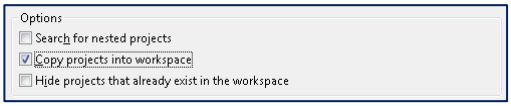
Import Code Examples

Whether you've downloaded an example, or received one from a colleague, Cypress recommends the following methods to import the example into the Modus Toolbox IDE:

- If you have an Eclipse-ready code example that you want to import into the ModusToolbox IDE, use the standard **File > Import** process. Note the following:
  - On the Import dialog, select **General > Existing Projects into Workspace**.



  - On the next page, enable the **Copy projects into Workspace** check box.

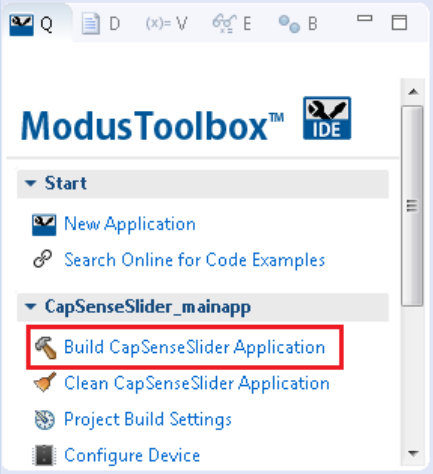


Note There are various ways to import examples into Eclipse. If you prefer a different method, make sure that all of the project files are copied into the workspace directory.

- If you've downloaded an example that includes a Makefile (*modus.mk*), use the New Application wizard to create a new application, and select the **Browse…** button to open the Makefile to use as the starter application.

Build Starter Application

After loading an application, build the entire application to generate the necessary files. Select any project in the application. Then, in the **Quick Panel**, click the “Build <app-name> Application” link.

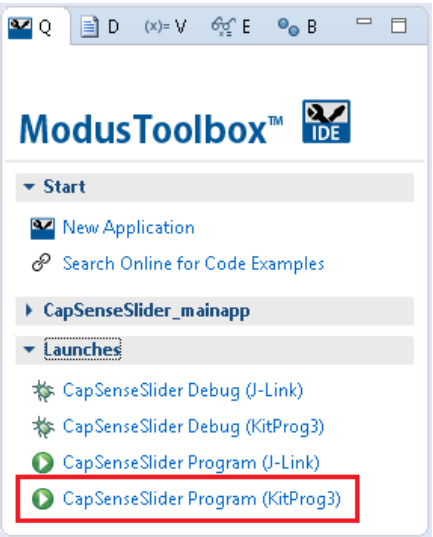


Messages display in the Console, indicating whether the build was successful or not. For more details about building applications and the various Consoles available, see the Build Applications chapter.

Program Starter Application

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

In the Project Explorer, select the <*app-name*>*_mainapp* project. Then, in the **Quick Panel**, click the “<app-name> Program KitProg3” link.

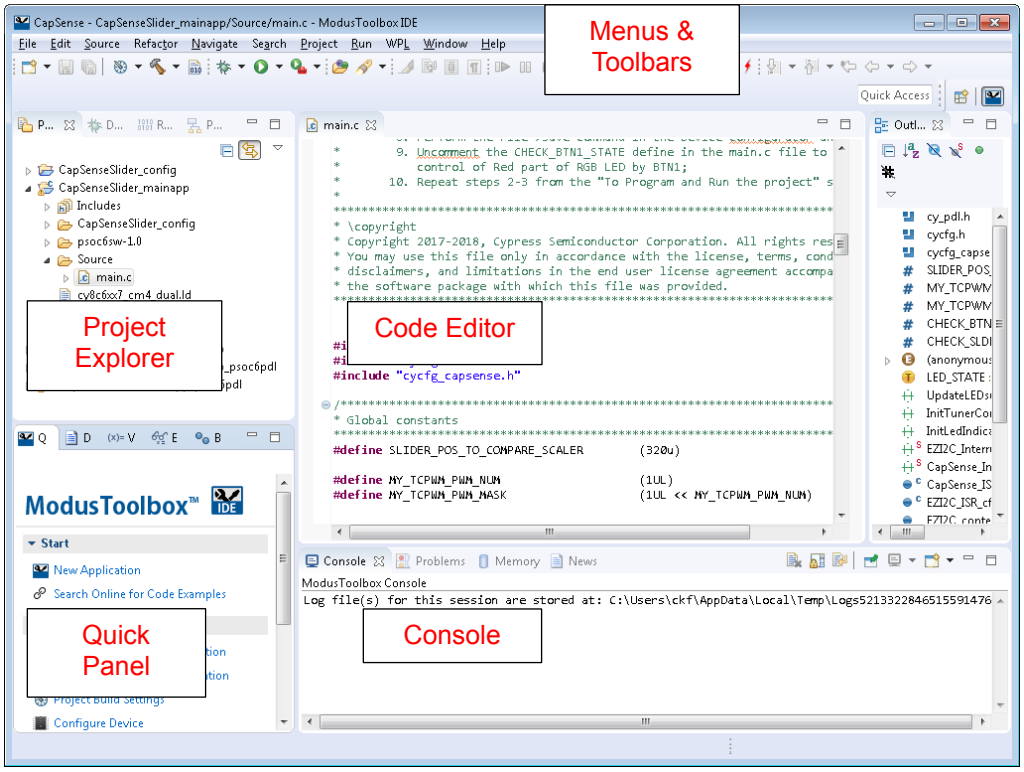


3 IDE Description

Overview

The ModusToolbox IDE is based on the Eclipse IDE “Oxygen” version. 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*. Cypress also provides a document called the *Eclipse Survival Guide*, which provides tips and hints for how to use the ModusToolbox IDE.

The IDE contains Eclipse standard menus and toolbars, plus various panes such as the Project Explorer, Code Editor, and Console. One difference from the standard Eclipse IDE is the “ModusToolbox Perspective.” This perspective provides the “Quick Panel,” a “News View,” and adds tabs to the Project Explorer. “Perspective” is an Eclipse term for the initial set and layout of views in the IDE. The ModusToolbox IDE also provides a Welcome Page, which displays on first launch of the IDE for a given workspace.



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 > ModusToolbox** from the **Window** menu. To restore the ModusToolbox perspective to the original layout, select **Perspective > Reset Perspective** from the **Window** menu.

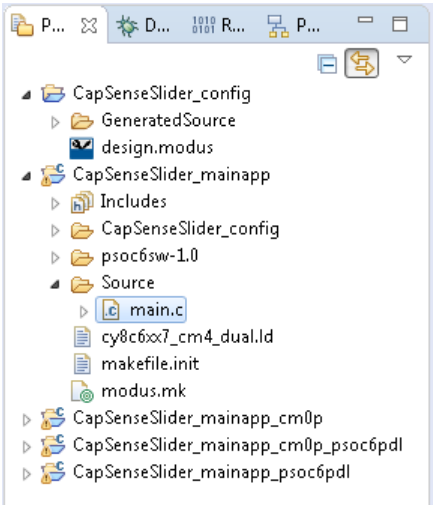
The following describe different parts of the IDE:

- Menus and Toolbars – Use the various menus and toolbars to access build/program/debug commands for your application. These are covered in the *Eclipse Workbench User Guide*.
- Project Explorer – Use the Project Explorer to find and open files in your application. See Project Explorer for more information.
- Quick Panel Tab – Use this tab to access appropriate commands, based on what you select in the Project Explorer.
- Documents Tab – Use this tab to access various documents. You can also open documentation from the **Help** menu.
- News View – This is an Eclipse view in the ModusToolbox Perspective that displays a page of blog articles from cypress.com. This is located in the same panel as the Console and Problems views.

- Code Editor – Use the Code Editor to edit various source files in your application.
- Welcome Page – This is an HTML document. It contains links to Help topics and videos. You can open it it any time from the **Help** menu.

Project Explorer

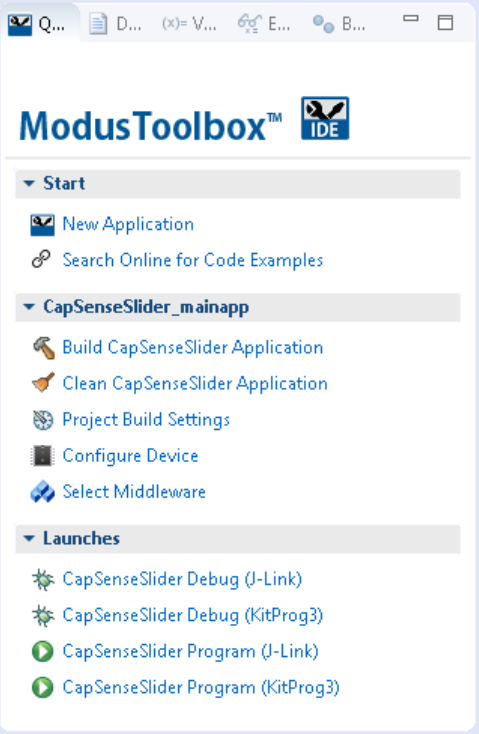
In the ModusToolbox IDE, after creating an application, the Project Explorer contains a collection of related projects. The following image shows an application with multiple project folders.



All projects that are part of the application are prefixed with the same base name used when creating the application. In most cases, focus on the *<app-name>_mainapp* project. It contains the main application source code and access to the Makefile.

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.



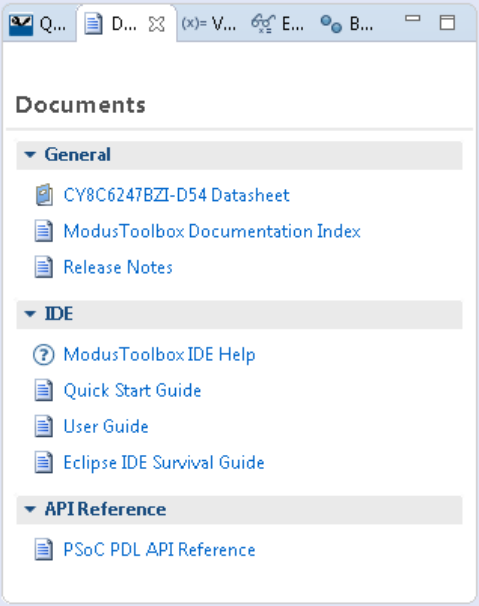
Quick Panel Tab

The first tab of the Quick Panel contains links to various commands, organized as follows:

- **Start** – This contains the New Application link to create new applications, and a link to find Code Examples.
- **Selected <app-name>_project** – 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, access Build Settings, Configure Device, and Select Middleware.
- **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. This area is only populated if you have the top project in your application selected (either *<app-name>_mainapp* or *<app-name>_lut* depending on the device). For more information, see Launch Configurations.

Documents Tab

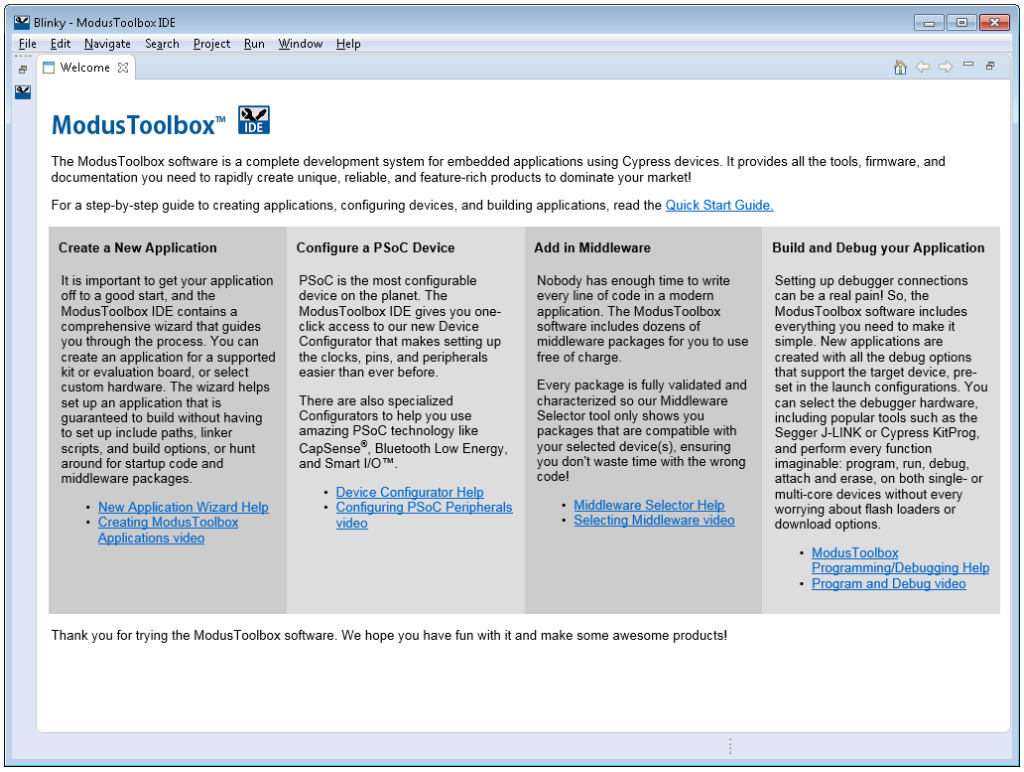
This tab contains links to various documents, such as the applicable device datasheet, IDE documentation, and API documentation.



These documents are also available from the **Help** menu, along with other useful documentation.

Welcome Page

The Welcome page is a static HTML file that provides an overview of the ModusToolbox software, as well as links to various Help topics and videos. This opens by default for a new installation and any new workspace.



Click the **X** on the Welcome Page tab to close it, or click the **Restore**  button to move it to the side.

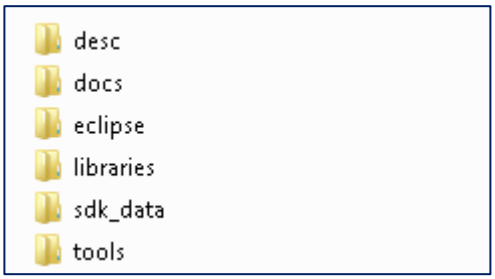
You can access this page at any time from **Help > Welcome**.

4 SDK Description

The SDK provides the central core of ModusToolbox software. It contains configuration tools, drivers, libraries, middleware, as well as various utilities, Makefiles, and scripts. You may use one or a few of these tools in any environment you prefer.

Directory Structure

Refer to the *ModusToolbox Installation Guide* for information about installing ModusToolbox. Once it is installed, the various ModusToolbox top-level directories are organized as follows:



These directories contain the following files and folders:

- **desc –** This contains files used to track installed SDKs. You do **not** need to use anything in this folder.
- **docs –** This is the top-level documentation directory. It contains various top-level documents and an html file with links to documents provided as part of ModusToolbox. See Documentation for more information.
- **eclipse (or ModusToolbox.app on macOS) –** This contains the IDE. See IDE Description.
- **libraries –** This contains firmware and resources files, including:
  - platforms-1.0 – This contains internal / helper Makefiles used with the IDE and command-line builds.
  - psoc6sw-1.0 – This contains PSoC 6 support drivers, middleware, and documentation.
  - udd-1.0 – This contains internal device data files. You do **not** need to use anything in this folder.
- **sdk_data –** This contains files used to track installed SDKs. You do **not** need to use anything in this folder.
- **tools –** This contains all the various tools and scripts provided as part of ModusToolbox. See Tools for more information.

Documentation

The */docs* directory contains top-level documents and an html document with links to all the documents included in the installation and on the web.

Release Notes

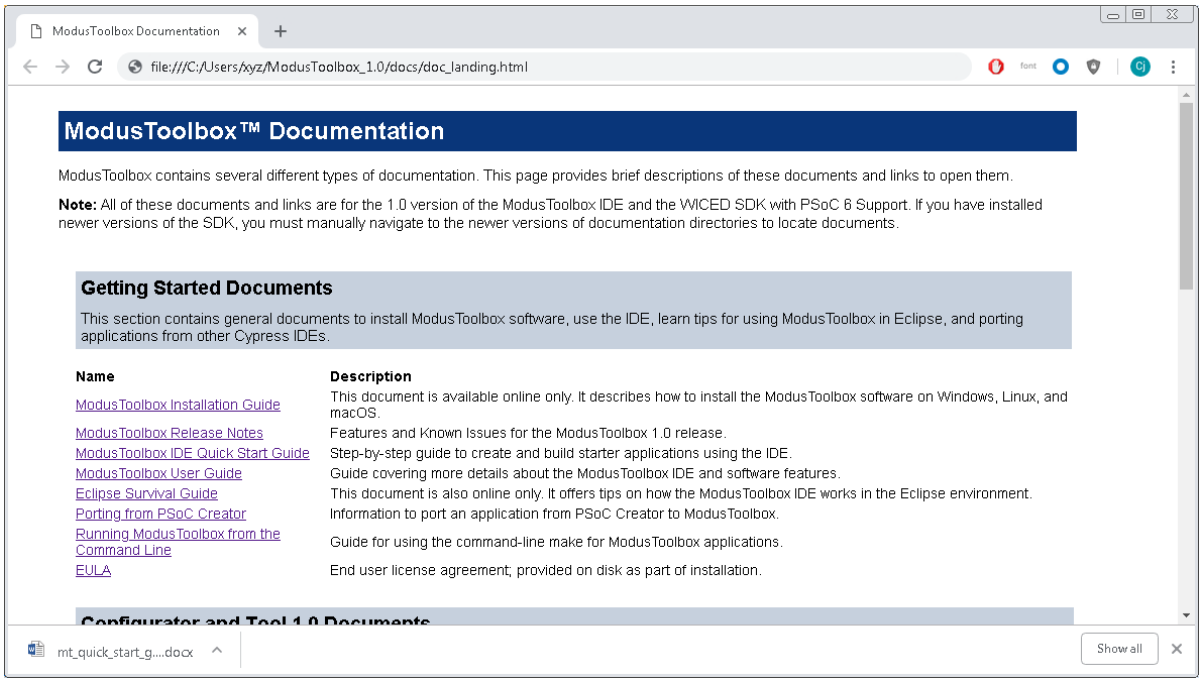
For the 1.0 release, the release notes document is for all of the ModusToolbox software included in the installation. In the future, there may be separate SDK-only and IDE-only versions of the release notes, depending on what updates you install.

Top-Level Documents

This folder contains the ModusToolbox IDE Quick Start Guide and this user guide. These guides cover different aspects of using the IDE.

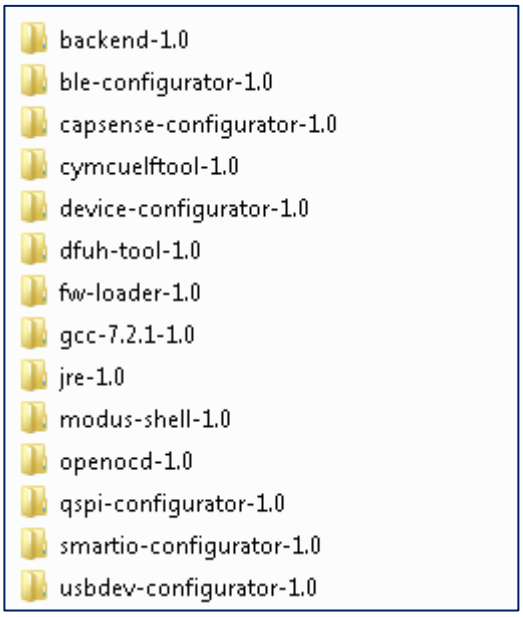
Document Index Page

The *doc_landing.html* file provides links to all the documents included in the installation and on the web. This file is available from the IDE **Help** menu and **Quick Panel**.



Tools

The */tools* folder contains the following:



- backend – This contains backend support files used by the system. You do **not** need to interact with this folder.
- PSoC Configurators – There are several Configurators used to update various settings for different peripherals. See Use PSoC Configurators.
- cymcuelftool – This tool is used to manipulate Elf files. Refer to the *CyMCUElfTool User Guide* located in the tool's doc folder.
- fw-loader – This is the Firmware Loader tool used to update firmware on PSoC 6 kits. See KitProg Firmware Loader.
- GCC – ModusToolbox software includes GCC version 7.2.1 as the preferred toolchain. See https://www.gnu.org/software/gcc/ for information.
- JRE – This folder contains the Java Runtime Environment version provided as part of the tool. This is used by the IDE and the backend. See https://www.java.com for more information.
- Modus-Shell – This folder contains various helper utilities used by the system. You do **not** need to interact with this folder.
- Open OCD – This contains the version of the Open On-Chip Debugger used by ModusToolbox to program various boards. For more information, refer to the *Cypress Programmer 2.0 User Guide*.

5 Configure Applications

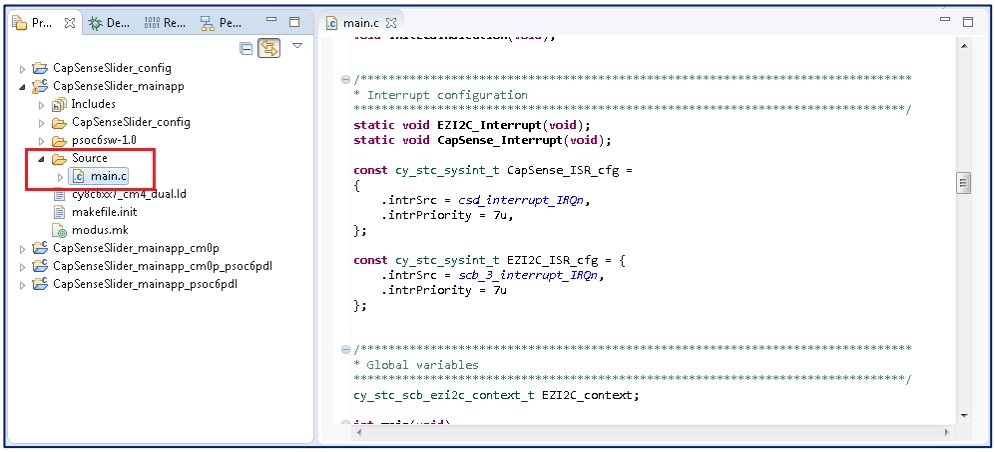
This chapter covers how to make various changes to your application. It includes:

- Modify Code
- Use PSoC Configurators
- Application Transitions

Modify Code

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

- In the Project Explorer, expand the *<app-name>_mainapp\Source* project folder and double click the *main.c* file.



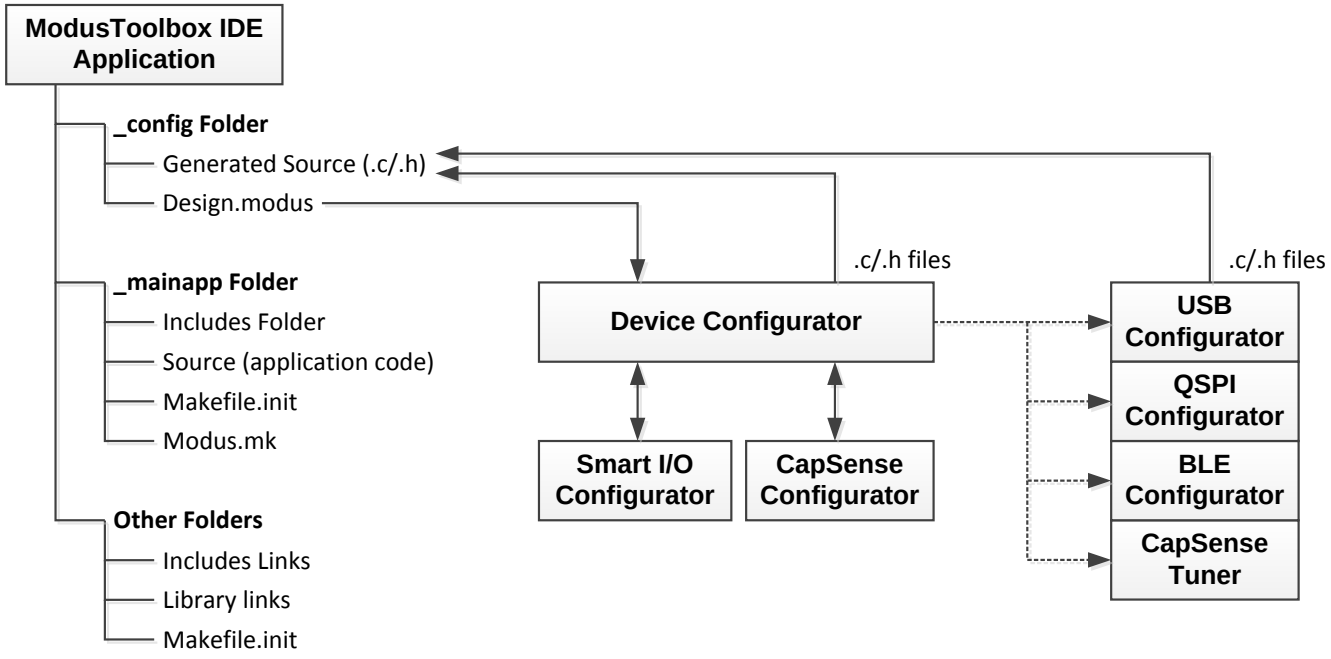
As you type into the file, an asterisk (*) will appear in the file's tab to indicate changes were made. The **Save/Save As** commands will also become available to select.

Use PSoC Configurators

ModusToolbox software provides graphical applications that make it easier to configure a hardware block. For example, instead of having to search through all the documentation to configure a serial communication block as a UART with a desired configuration, open the appropriate tool and set the baud rate, parity, stop bits, etc. Upon saving the hardware configuration, the tool generates the C code to initialize the hardware with the desired configuration.

Overview

The ModusToolbox IDE manages configuration of resources in a target application. The IDE stores configuration settings in the *design.modus* file. This file is used by the graphical configurators, which generate firmware. This firmware is stored in the application's "GeneratedSource" folder. The following diagram provides a high-level view of this interaction between the IDE and the configurators.



The *design.modus* file is responsible for holding all of the hardware configuration information. It contains the following:

- Selected device
- Resource parameters
- Configurator parameters
- Constraints
- Referenced kits

Configurators are independent of each other, but they can be used together to provide flexible configuration options. If intended to be used together, the generated source needs to be saved in the same location. They can be used stand alone, in conjunction with other tools, or within a complete IDE. Everything is bundled together as part of the unified SDK for distribution purposes. Configurators are used for:

- Displaying a user interface for editing parameters
- Setting up connections such as pins and clocks for a peripheral
- Generating code to configure middleware

Available Configurators

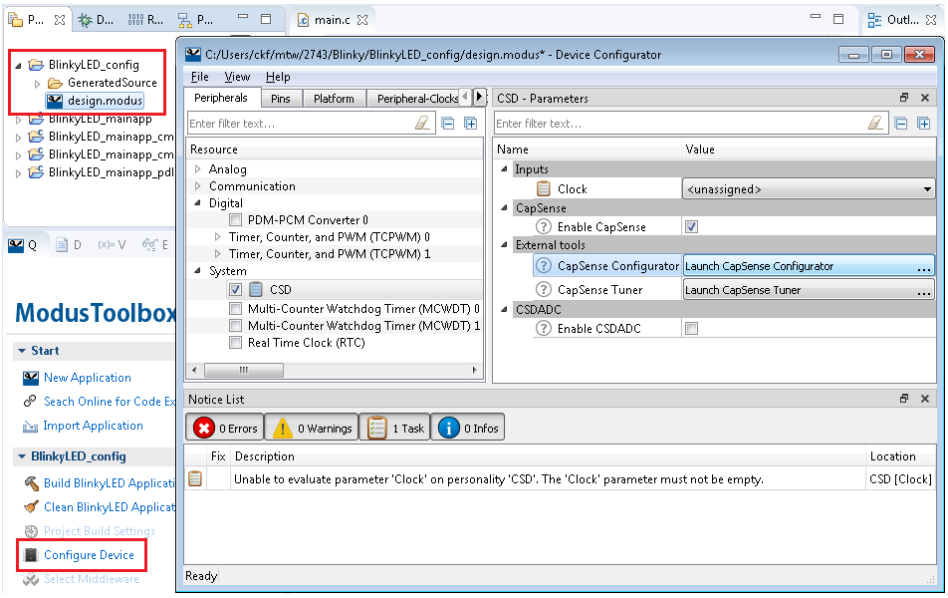
For PSoC 6 MCU applications, the available Configurators include:

- Device Configurator: Set up the system (platform) functions such as pins, interrupts, clocks, and DMA, as well as the basic peripherals, including UART, Timer, etc.
- CapSense Configurator and Tuner: Configure CapSense, test it, and generate the required firmware.
- USB Configurator: Configure USB settings and generate the required firmware.
- QSPI Configurator: Configure external memory and generate the required firmware.
- Smart I/O™ Configurator: Configure the Smart I/O.
- BLE Configurator: Configure the Bluetooth Low Energy (BLE) settings.

Refer to the applicable Configurator guide, available from that tool's **Help** menu.

Launching Device Configurator

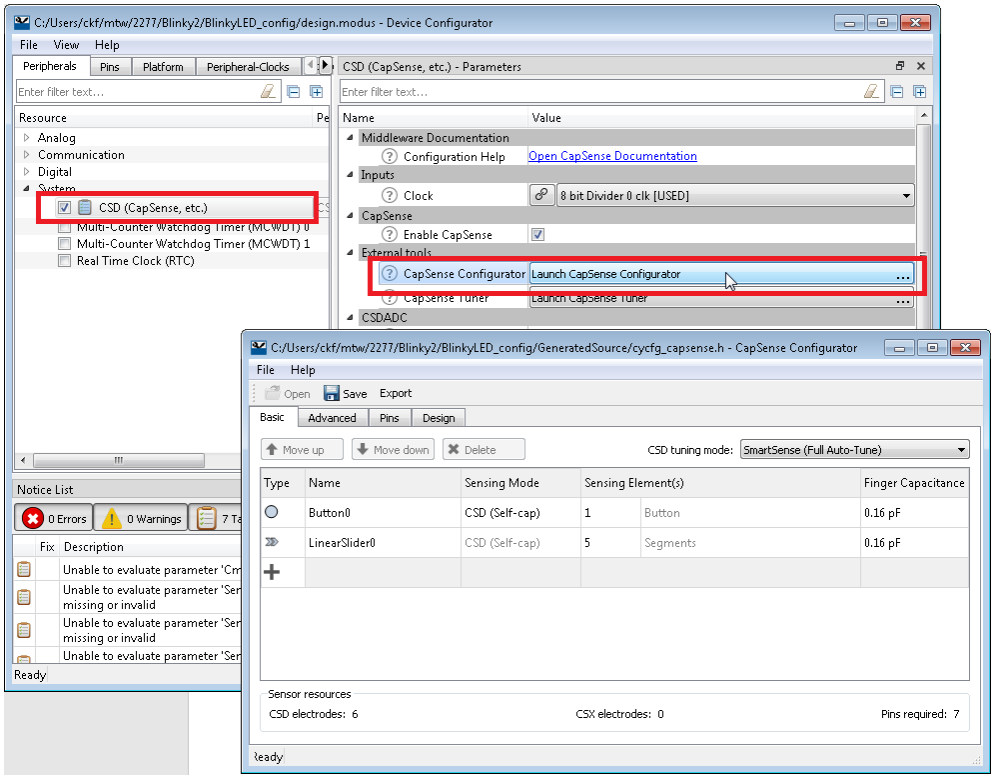
If your application includes a *design.modus* file, double-click on it in the Project Explorer to launch the Device Configurator from the ModusToolbox IDE. You can also click the “Configure Device” link in the **Quick Panel**.



As you select and enable different resources in the Device Configurator, the **Parameters** pane displays various configuration settings that can be updated. For more details, refer to the *ModusToolbox Device Configurator Guide* available from the **Help** menu.

Launching Separate Configurators

To launch a separate configurator from the Device Configurator, enable the desired resource under **Peripherals > Resource**, then click the **[Launch . . .]** button under **Parameters**.



In this example, the CapSense Configurator opens in a separate window. Refer to the individual configurator guides for more information. These are available from the configurators' **Help** menu.

Application Transitions

The IDE provides several "transition" context menu items that allow you to transform the application in some way. These menu items are available when you right-click on an application project. They do not always apply in every situation. See each section for various limitations. The items include:

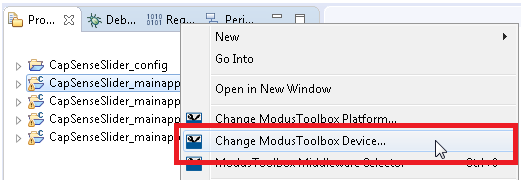
- Change Device
- Select Middleware
- Rename Application

Change Device

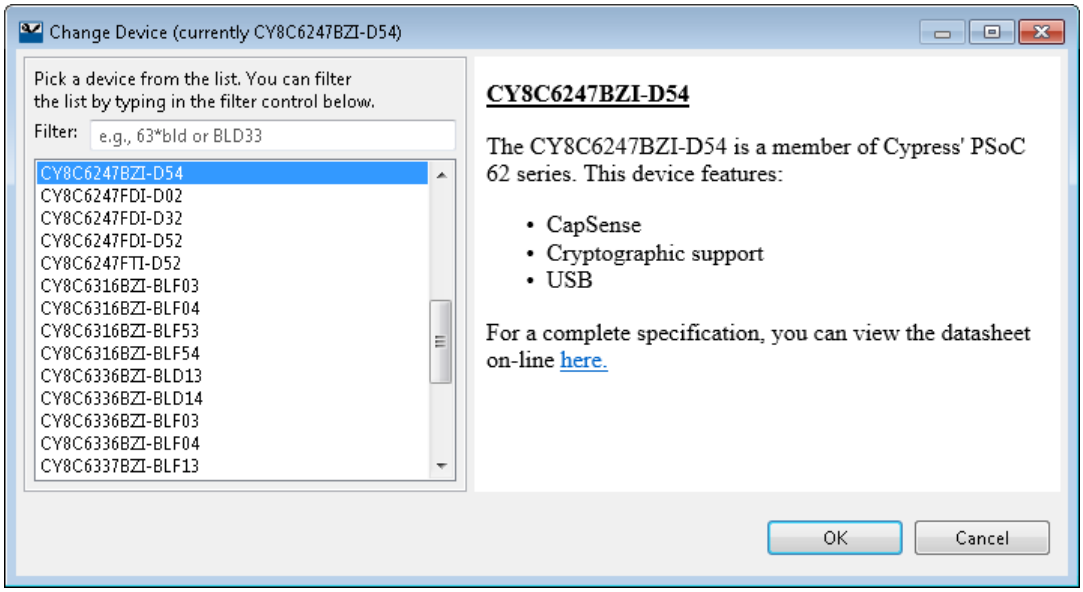
The Change Device dialog allows you to choose another device for you application.

Launch the Dialog

Right-click on the *<app-name>_mainapp* project and select **Change ModusToolbox Device …**

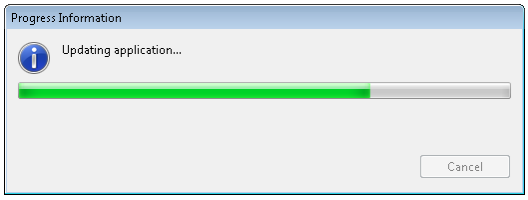


On the Change Device dialog, select the appropriate device from the list and click **OK**.



Note You can type in the **Filter** box to limit the number of devices displayed.

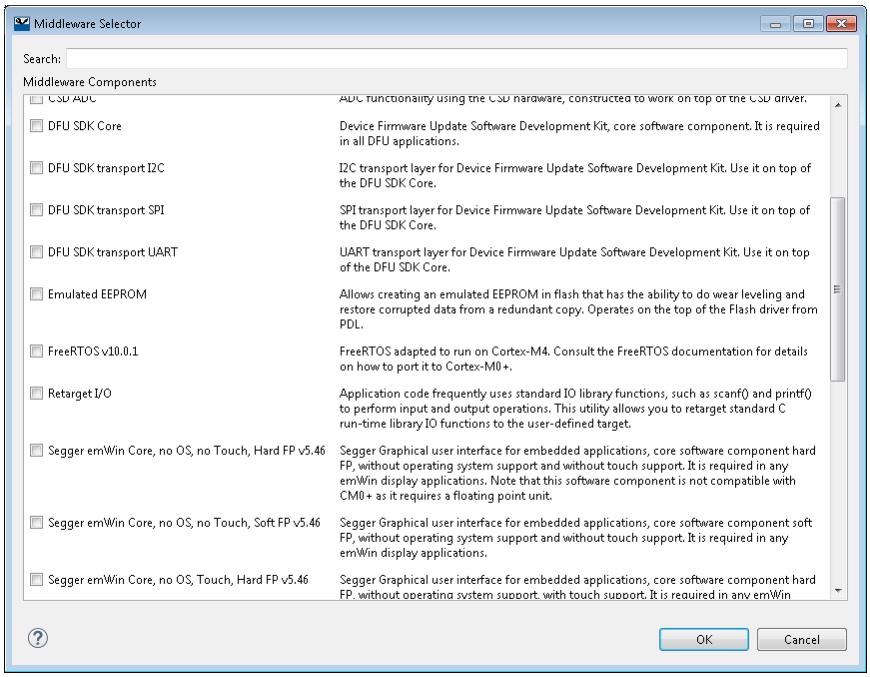
Depending on the chosen device, this operation can take several moments. A progress bar indicates the status of the operation.



When complete, the application will apply to the new device.

Select Middleware

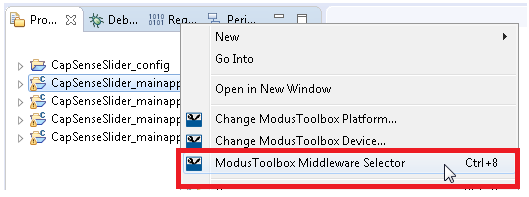
The Middleware Selector allows you to select what firmware packages to add or remove from the application. The tool displays only middleware that is valid for the current platform. After adding or removing middleware from an application, the necessary compiler options, such as include paths, will be added or removed.



A middleware component can be distributed as a pre-compiled library. In this case, there are library variants compiled using soft-float and hard-float application binary interfaces (ABIs). The ABI used to compile a library is specified in the middleware component name. You must compile your entire application with the same ABI and link with a compatible set of libraries. Therefore, use the same floating-point ABI for middleware libraries and the toolchain settings for your application. To change the Float ABI toolchain settings, open **Project > Properties**. Then select the **C/C++ Build > Settings > Target processor** and specify the Float ABI option. You must also specify the same Float ABI for all dependent projects.

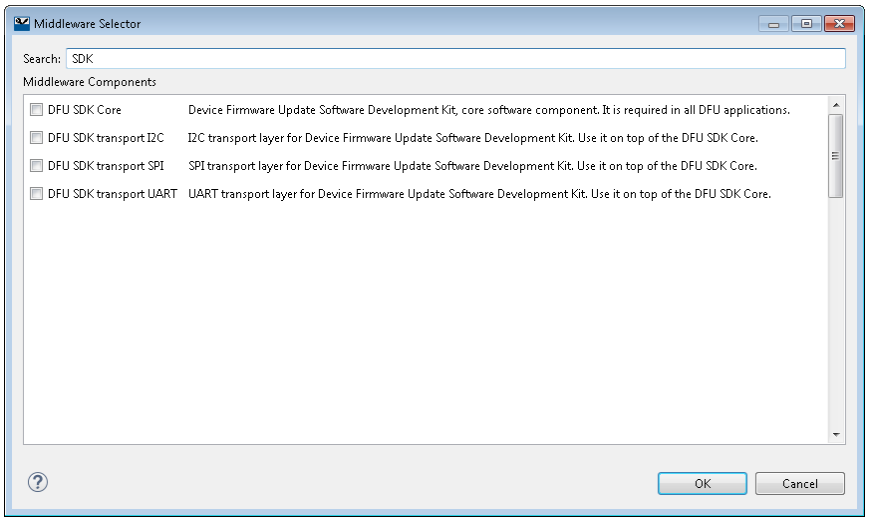
Launch the Dialog

Right-click on the main project and select **ModusToobox Middleware Selector**.



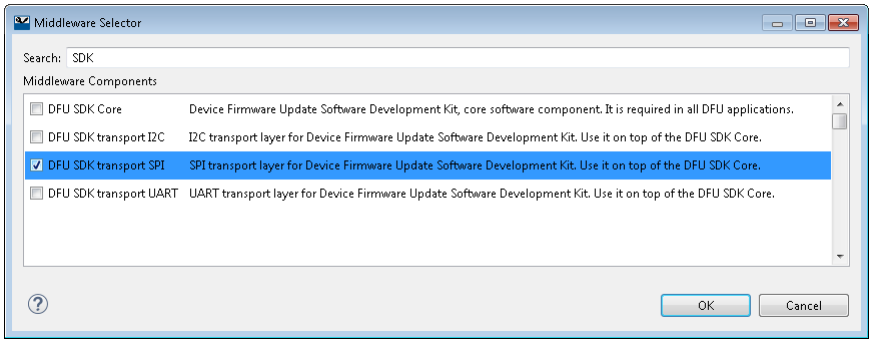
Search

The Search box helps you find various components in the dialog by filtering the matching string. This follows the Java pattern matching syntax (https://docs.oracle.com/javase/7/docs/api/java/util/regex/Pattern.html). For example, a period (.) will show everything.



Enable/Disable Middleware Component

Click the check box next to the desired middleware component to enable it or disable it as applicable.



When you click on a middleware component, it is highlighted.

- If there are any dependencies, they will be highlighted in a lighter color.
- If an enabled component has dependencies, all dependencies will also be enabled.
- Disabling a component will **not** disable dependent components. You must disable dependent components one at a time.

Click **OK** to close the dialog and update your application.

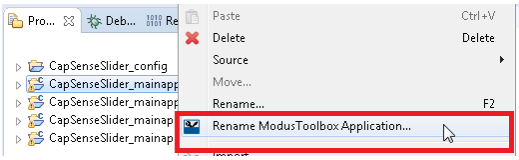
Rename Application

Due to the tight relationship between ModusToolbox projects that make up an application, you must use the **Rename ModusToolbox Application** feature to rename all the projects in an entire ModusToolbox IDE application.

Note If you try to use the native Eclipse **Rename** function for a ModusToolbox application, an error message will display.

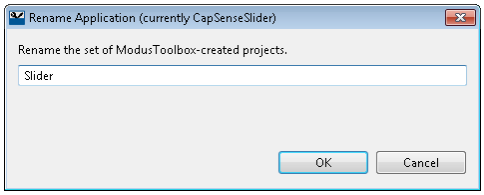
Launch the Dialog

In the Project Explorer, right-click on any of the project folders and select **Rename ModusToolbox Application …**

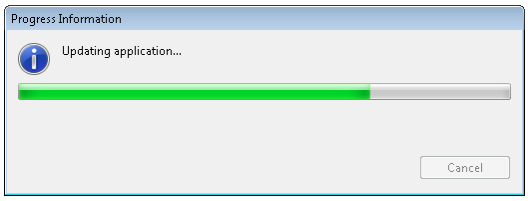


Note This menu item is disabled while the indexer is running.

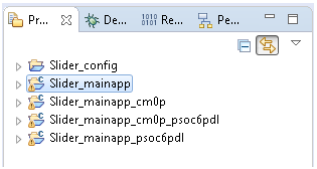
On the Rename Application dialog, type a new name for the application and click **OK**.



Depending on how many projects are in the application, this operation can take a while. A progress bar indicates the status of the operation.



When complete, all associated projects have the new application name and all dependencies will have been updated.



6 Build Applications

This chapter covers various aspects of building applications. Building applications is not specifically required, because building is performed as part of the programming and debugging process. However, if you are running the ModusToolbox IDE without any hardware attached, you may wish to build your application to ensure all the code is correct. If you changed code in one of your projects, you may wish to build just that project.

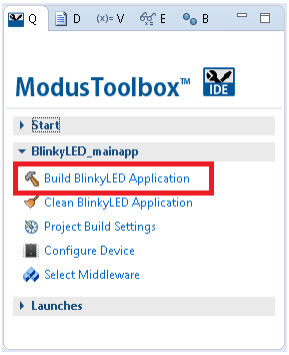
- Build with Make
- Build with IDE
- Generated ELF Files
- Enable HEX File Generation

Build with Make

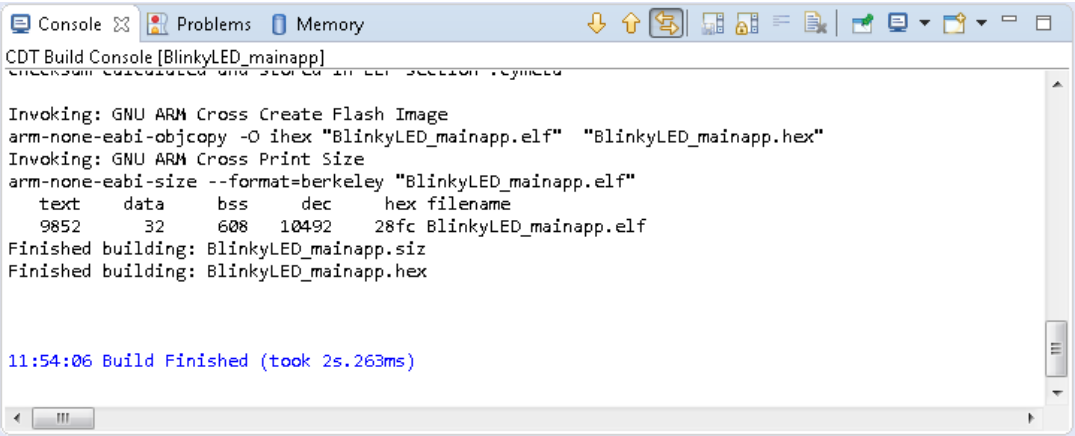
You can build applications from the command line using your preferred version of Make. Refer to the *Running ModusToolbox from the Command Line* document for more information.

Build with IDE

After loading an application, it is best to first build everything to generate the necessary files. Click on an application project in the Project Explorer. Then in the **Quick Panel**, click the “Build <app-name> Application” link.



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 application starts building. You won't see any errors if they are not on the final project that gets built.

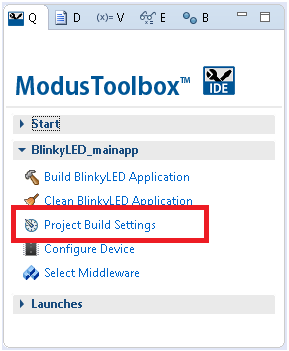
For subsequent updates, you can build one or more projects using the right-click menu options. Any projects that are dependent on the project being built will also be built. The ModusToolbox IDE supports all the usual application and build options available for the native Eclipse IDE.

Build Settings

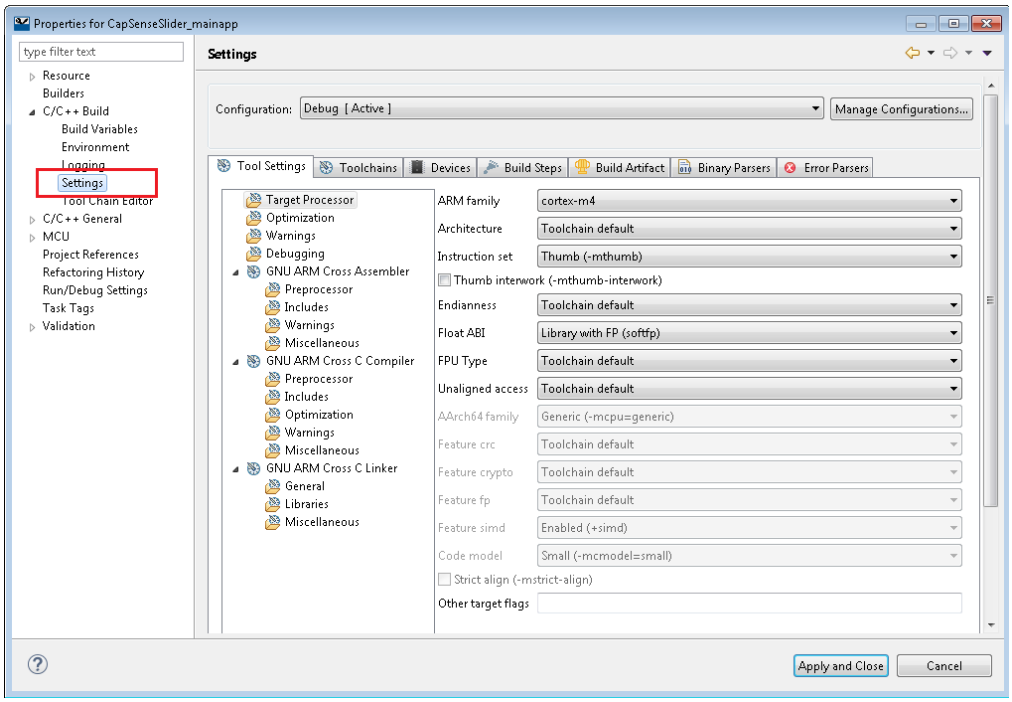
The ModusToolbox IDE uses standard eclipse CDT build settings for things such as:

- Custom defines
- Include paths
- General options to the compiler/linker

To change build settings, first select the appropriate project and then click the "Project Build Settings" link in the **Quick Panel**. Note that build settings are project specific, NOT application specific. So, they must be set in the project or projects in which they are to be used.



This link opens the Eclipse Properties dialog on **C/C++ Build > Settings** page. This dialog has many pages with a lot of settings. The following sections show a few that you may wish to use for your application and environment.



Note Any changes made to C/C++ build settings in the IDE are not written back to external Makefiles, such as *modus.mk*.

This page contains the following settings:

- Target Processor – This section allows for configuring settings related to the target device and what optional hardware features it provides.
- Optimization – This section allows for configuring optimization options that apply to both the assembler and compiler.
- Warnings – Provides controls for configuring common warning options for the assembler and compiler.
- Debugging – Provides controls for configuring common debug related information for the assembler and compiler.
- GNU ARM Cross Assembler – Provides a summary of the assembler command line options that were configured in any of the other pages.
  - Preprocessor – Provides options for setting up additional command line specified preprocessor defines and undefines for the assembler.
  - Includes – Provides options for setting up include paths for the assembler.
  - Warnings – Provides general input for specifying command line arguments for controlling how different warnings are handled for the assembler.
  - Miscellaneous – Provides general input for any additional assembler command line options to specify.

- GNU ARM Cross C Compiler – Provides a summary of the compiler command line options that were configured in any of the other pages.
  - Preprocessor - Provides options for setting up additional command line specified preprocessor defines and undefines for the compiler.
  - Includes – Provides options for setting up include paths for the compiler.
  - Optimization – This allows for compiler specific optimization settings which where not specified in the top level Optimization section.
  - Warnings – Provides general input for specifying command line arguments for controlling how different warnings are handled for the compiler
  - Miscellaneous – Provides general input for any additional assembler command line options.
- GNU ARM Cross C Linker – Provides a summary of the linker command line options that were configured in any of the other pages.
  - General – Provides options for specifying basic linker command line options.
  - Libraries – Provides options for specifying what libraries to link against and where those libraries are located.
  - Miscellaneous – Provides options for specifying additional linker command line arguments.

For details about what options are available, and what they do, refer to the GCC option summary:

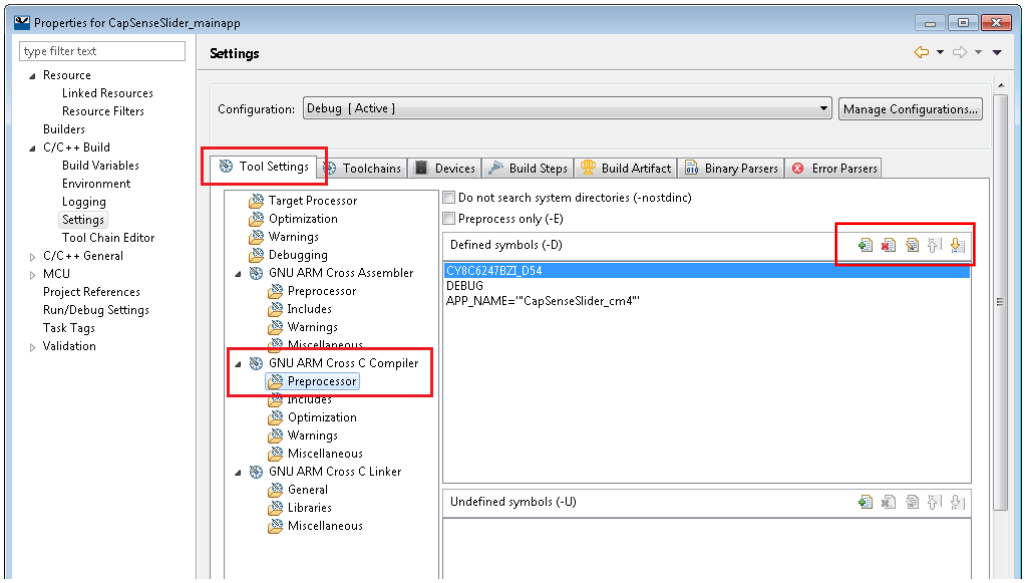
https://gcc.gnu.org/onlinedocs/gcc-7.2.0/gcc/Option-Summary.html

More details about the Eclipse build settings can be found here:

https://help.eclipse.org/oxygen/index.jsp?topic=%2Forg.eclipse.cdt.doc.user%2Freference%2Fcdt_u_properties.htm

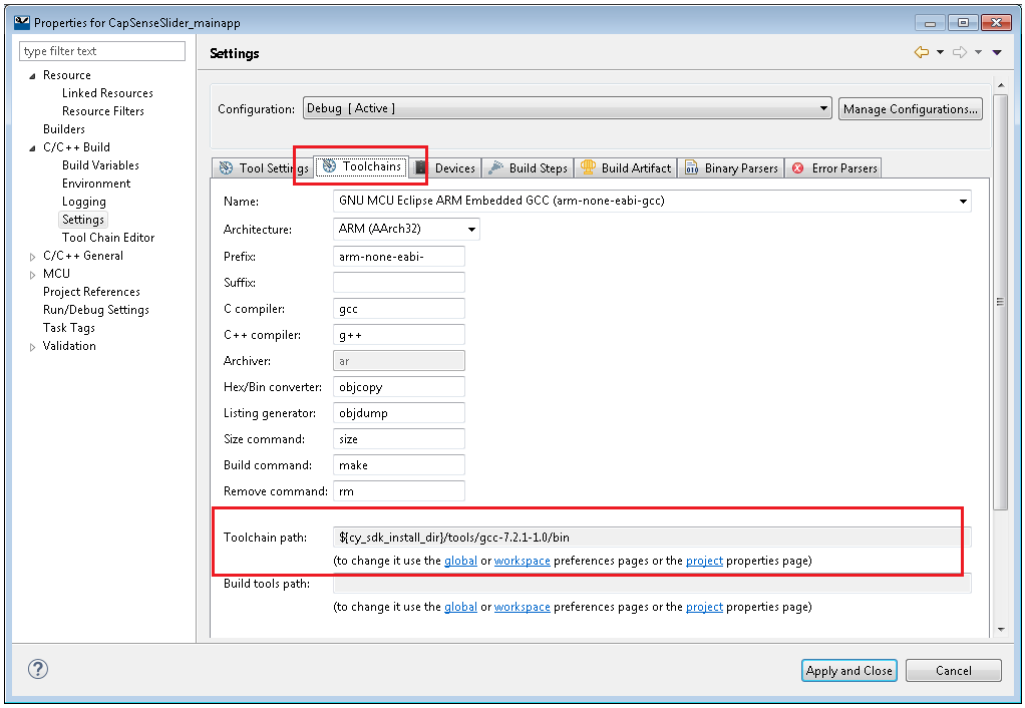
Macros

To define a macro, click the **Tool Settings** tab, and select **GNU ARM Cross C Compiler > Preprocessor**. Use the icons to add, delete, edit, or reorder macros.



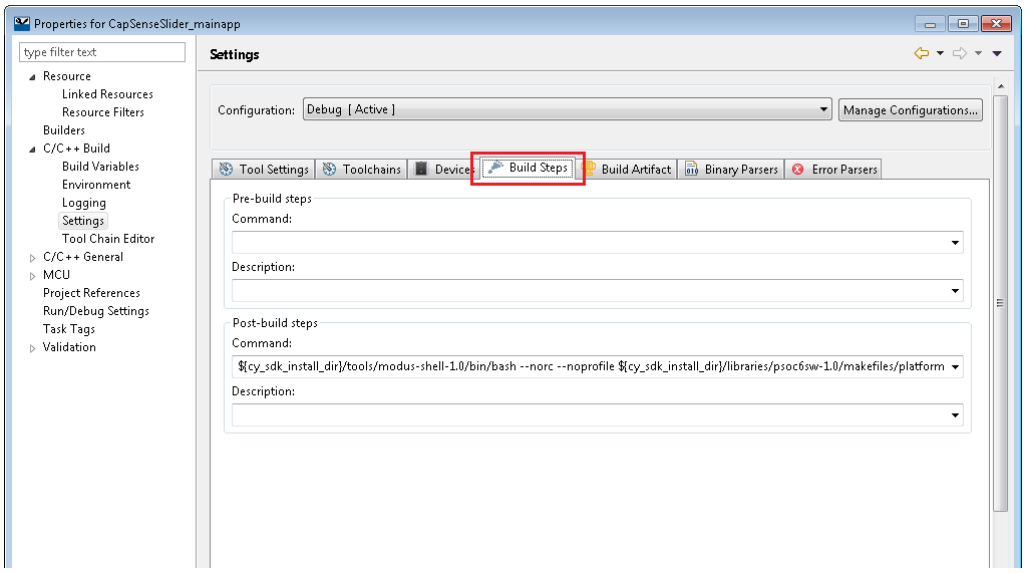
GCC Version

ModusToolbox software includes GCC version 7.2.1 as the preferred toolchain to use with the ModusToolbox IDE. If you have a different version of GCC you prefer, update the project properties to point to the appropriate toolchain folder. You must do this for all projects in an application. Click the **Toolchains** tab, and then click the appropriate link to update global or workspace preferences, or project properties.



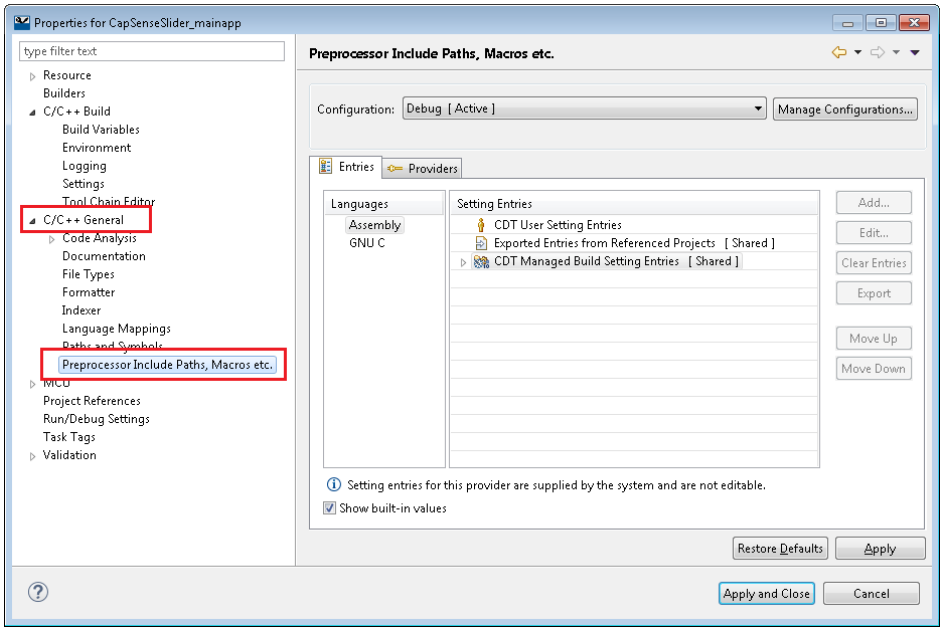
Pre-Build/Post-Build Commands

Click the **Build Steps** tab to access pre-build and post-build commands.



Include Paths

To add/update include paths, go to **C/C++ General > Processor Include Paths …**

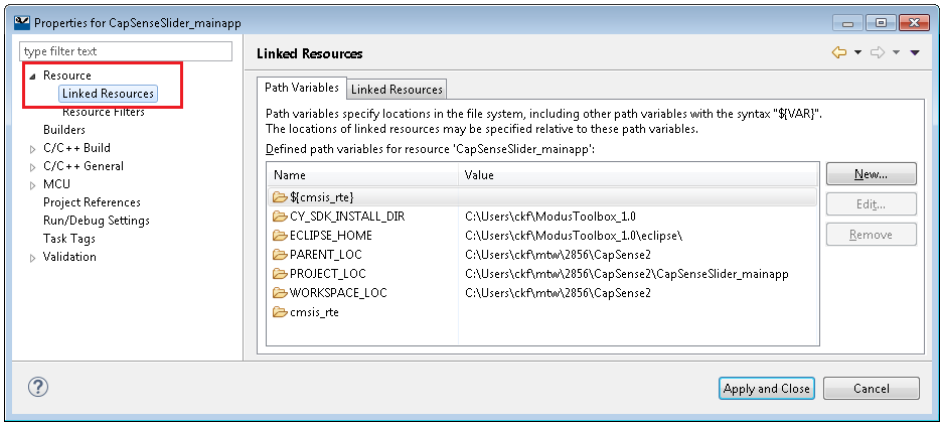


Path/Build Variables

As shown in previous screen captures, there are path variables used in various settings. Some are defined by the underlying Eclipse framework, others are added for use with the ModusToolbox IDE. The following sections show where these are defined.

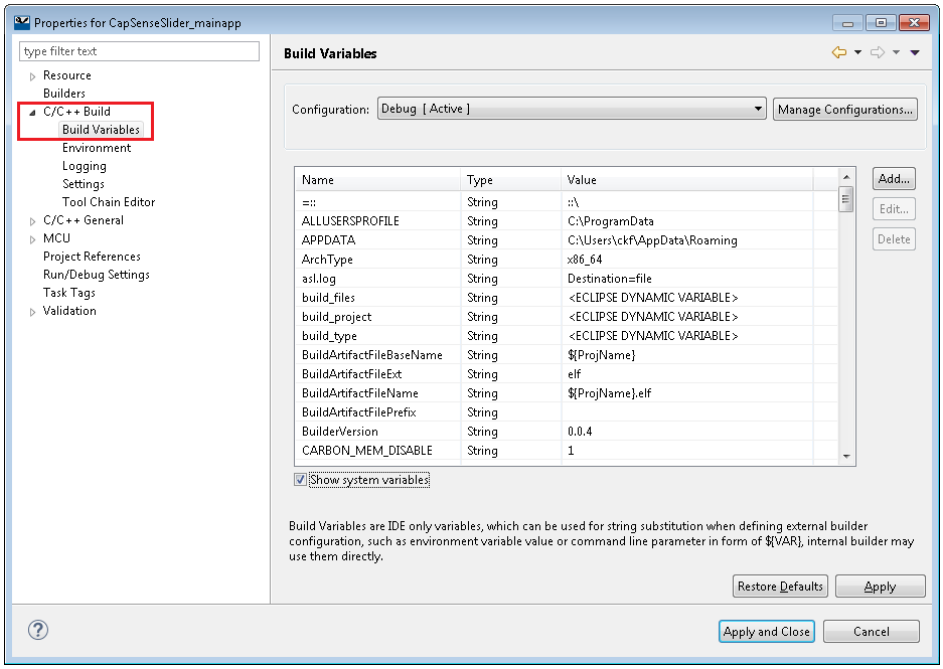
Eclipse Path Variables

These are defined on the **Resource > Linked Resources** Properties page.



ModusToolbox IDE Build Variables

These are defined on the **C/C++ Build > Build Variables** page.



Generated ELF Files

By default, the IDE only generates ELF files as part of the build. To enable generating HEX files, see Enable HEX File Generation. Each type of application generates ELF files as follows:

Dual-Core Devices

For PSoC 6 MCU dual-core devices, the following ELF files are generated:

CM4:

After a build, the ELF files are located in <app-name_mainapp>/Debug/

- <app-name>_mainapp_final.elf
- <app-name>_mainapp_signed.elf
- <app-name>_mainapp.elf

CM0+:

After a build, the ELF files are located in <app-name_mainapp_cm0p>/Debug/

- <app-name>_mainapp_cm0p_signed.elf
- <app-name>_mainapp_cm0p.elf

The <app-name>_mainapp_cm0p.elf and <app-name>_mainapp.elf files are standard ELF files created by the linkers in the PSoC 6 MCU build process.

The <app-name>_mainapp_cm0p_signed.elf, <app-name>_mainapp_signed.elf, and <app-name>_mainapp_final.elf files are used by the command-line interface (CLI) to program CM0+, CM4, and the merged image via OpenOCD.

The <app-name>_mainapp_cm0p_signed.elf and <app-name>_mainapp_final.elf files are used by the Run / Debug Configurations in the IDE.

Single Core Devices

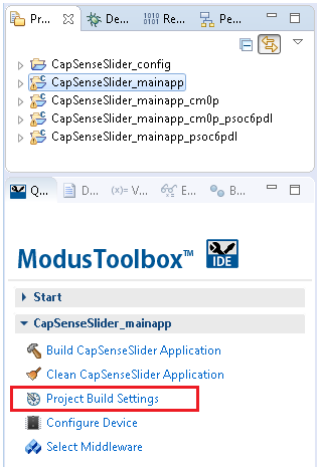
For PSoC 6 MCU single-core devices, generated files include:

- <app-name>_mainapp.elf
- <app-name>_mainapp_final.elf

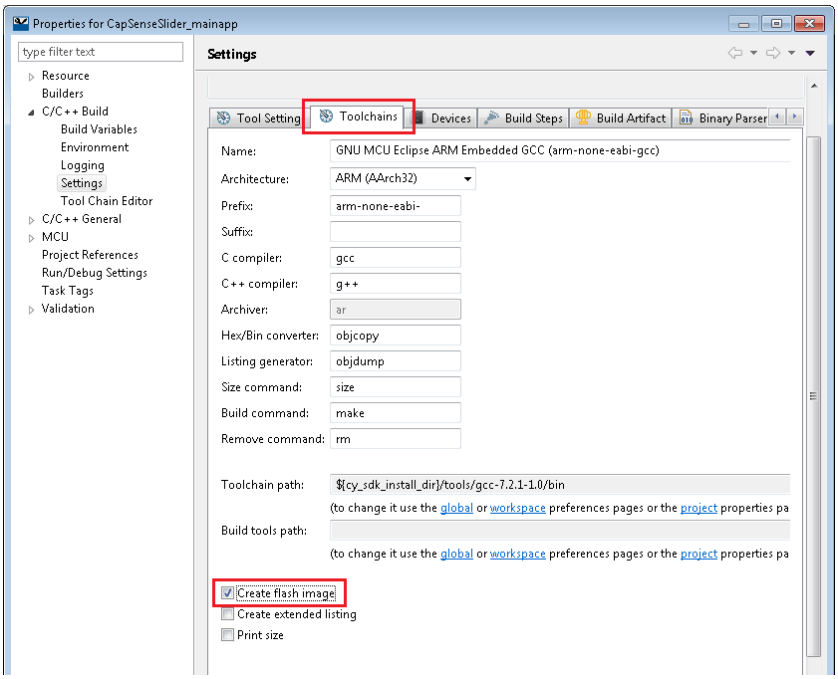
Enable HEX File Generation

Follow these instructions to enable HEX file generation in the ModusToolbox IDE. These steps include using the final ELF file as an input.

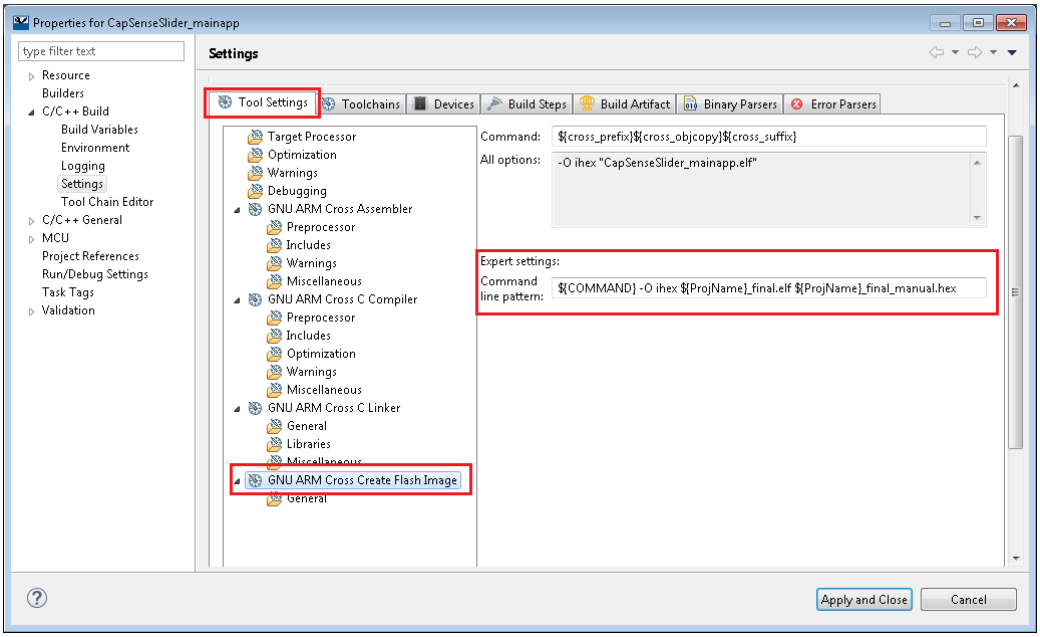
1. Click on the <app-name>_mainapp project and select the “Build Settings: link in the the **Quick Panel**.



2. On the Properties dialog, select the **Toolchains** tab on the **C/C++ Build > Settings** page, select the **Create flash image** check box.



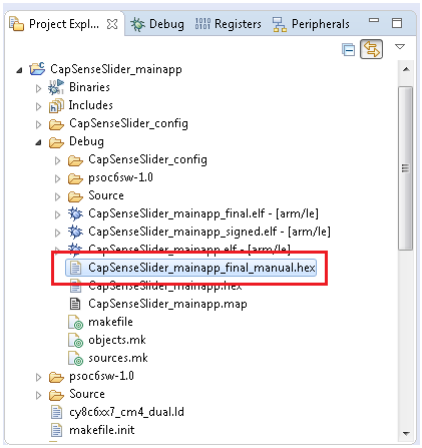
3. Select the **Tool Settings** tab Then, select **GNU ARM Cross Create Flash Image** in the tree and edit the **Expert Settings, Command line pattern** field.



   a. Change the existing pattern to the following:
   ```
   ${COMMAND} -O ihex ${ProjName}_final.elf ${ProjName}_final_manual.hex
   ```

   b. Click **Apply and Close**.

4. Build the application. When complete, expand the **Debug** folder and observe that the specified HEX file was generated.



You can use the HEX file with the Cypress Programmer tool to program the device. Make sure you set the **Reset Chip** flag. Refer to the *Cypress Programmer 2.0 GUI User Guide* for more information.

7 Programming and Debugging

Programming and debugging is native to your chosen development environment. Cypress 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.

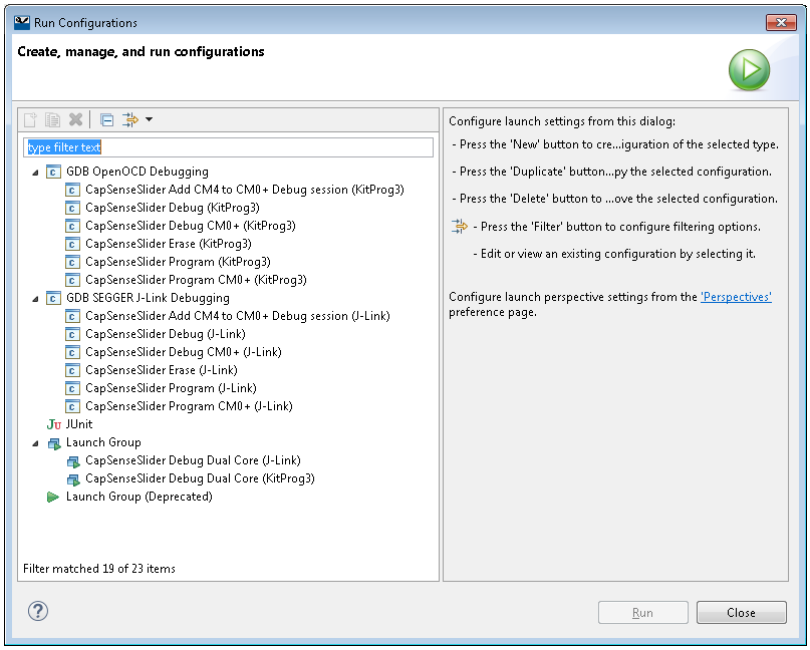
This chapter covers various topics related to building and debugging using the ModusToolbox IDE.

- Program/Debug Launch Configurations
- Debug Connection Options
- KitProg Firmware Loader
- Dual-Core Debugging

Program/Debug Launch Configurations

The flow for programming and debugging is similar for all devices. The ModusToolbox IDE 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 (included on-board on most Cypress PSoC 6 based kits) and another for J-Link. These are shown in the Run/Debug Configurations dialog, similar to the following.



You can open these dialogs 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:

CapSenseSlider Program (KitProg3)

CapSenseSlider Debug (J-Link)

Note KitProg3 configurations may not work if the J-Link probe is attached to the kit.

When an application is created, the tool generates the following launch configurations for both KitProg3 and J-Link. Some items display in the **Quick Panel** for the CM4, some for the CM0+, and some are in the Run/Debug Configurations dialog only.

- **Add CM4 to CM0+ Debug session:** This launch configuration is used for the dual core debug Launch Group items.
- **Debug:** This launch configuration builds the entire application on both cores, programs all the device's memories, and then starts a Cortex-M4 debugging session.
- **Debug CM0+:** This launch configuration builds the CM0+ core only, programs the core's memories, and then starts a Cortex-M0+ debugging session (CM4 application is undefined).

- **Erase:** This launch configuration erases all internal memories.
- **Program:** This launch configuration builds the entire application on both cores, programs all the device's memories, and then runs the program.
- **Program CM0+:** This launch configuration builds the CM0+ core only, programs the core's memories, and then runs the program (CM4 application is undefined).

There are also configurations under **Launch Group** used for dual-core debugging for each protocol:

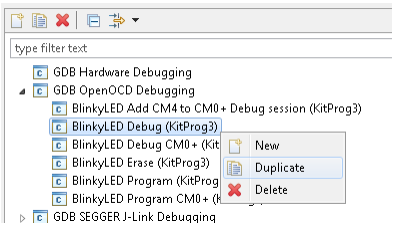
- **Debug Dual Core:** This launch group builds the entire application on both cores, programs the Cortex-M0+ core only, and attaches the Cortex-M4. Use the Program Launch Configuration before using this one. See Dual-Core Debugging.

Attach to Running PSoC 6 Target

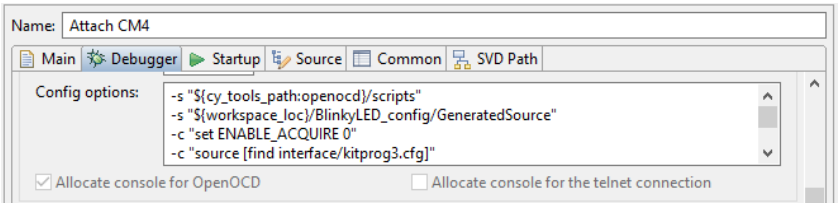
There are some cases where you may wish to attach to a running PSoC 6 target as part of a debug session. This is accomplished by copying and modifying existing debug configurations.

KitProg3

1. Duplicate the appropriate debug configuration for the target core; for example, "Debug (KitProg3)" or "Debug CM0+ (KitProg3)."

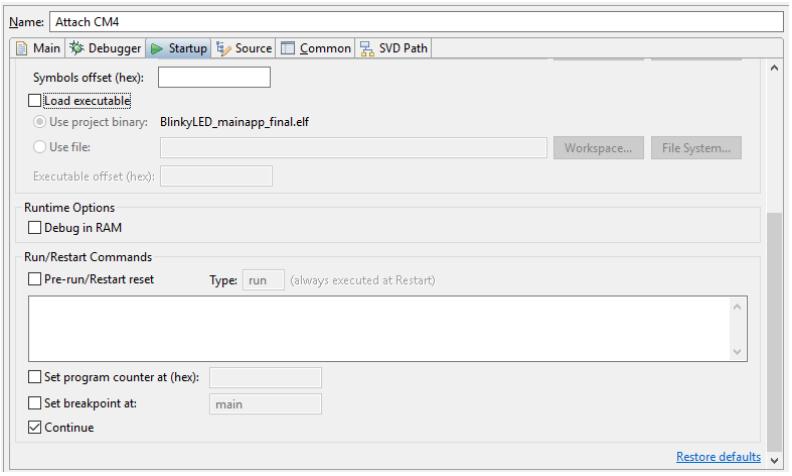


2. Rename the new Debug configuration to something meaningful, such as "Attach CM4" or "Attach CM0+."
3. Go to the **Debugger** tab.



   a. In the **Config options** field, insert the following as the first command:
   ```
   -c "set ENABLE_ACQUIRE 0"
   ```
   b. If setting up "Attach CM4," delete the following in the **Config options**:
   ```
   -c "init; reset init"
   ```

4. Go to the **Startup** tab.



   a. Unselect **Initial Reset**.

   b. Unselect **Load executable**.

   c. Unselect **Pre-run/Restart reset**.

   d. Delete everything in the text field under the **Pre-run/Restart reset** check box.

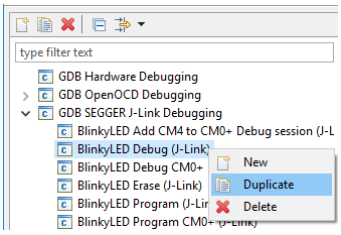
   e. Unselect **Set breakpoint at**.

   f. Select **Continue**.

5. Program your device, and attempt to attach to the running target using the new launch configuration.

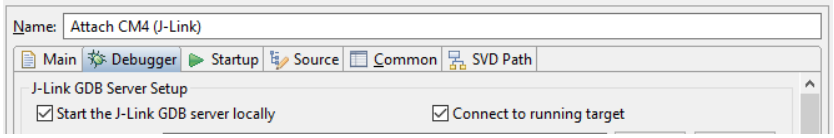
J-Link

1. Duplicate the appropriate debug configuration for the target core; for example, "Debug (J-Link)" or "Debug CM0+ (J-Link)."



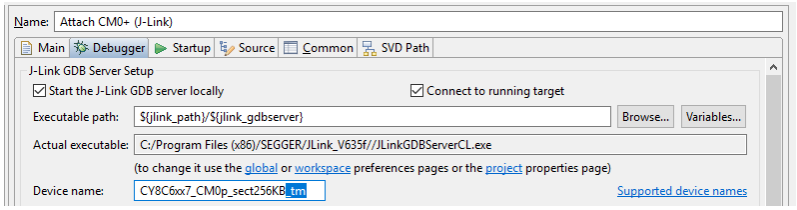
2. Rename the new Debug configuration to something meaningful, such as "Attach CM4 (J-Link)" or "Attach CM0+ (J-Link)"

3. Go to the **Debugger** tab.

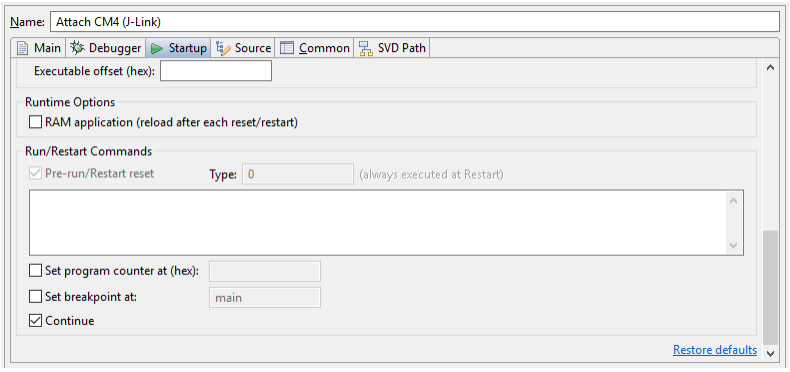


a. Select **Connect to running target**.

b. If setting up “Attach CM0+”, delete “_tm” part of the device name, if present:



4. Go to the **Startup** tab.



a. Delete everything in the text field under the **Pre-run/Restart reset** check box.

b. Unselect **Set breakpoint at**.

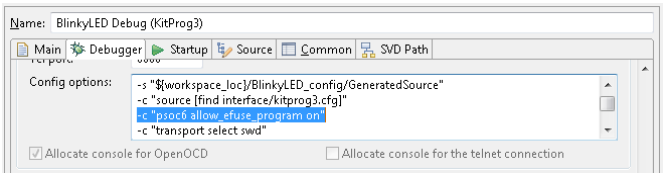
c. Select **Continue**.

5. Program your device, and attempt to attach to the running target using the new launch configuration.

Programming eFuse

To program eFuse, add the following command for the appropriate Debug Configuration, under the **Debugger** tab in the **Config options** field (after `-c "source [find target/psoc6.cfg]"`):

```
-c "psoc6 allow_efuse_program on"
```



Note This feature is not supported using the J-Link Debug Configurations.

Debug Connection Options

For the CY8CKIT-062-BLE and CY8CKIT-062-WIFI-BT kits, there are two main PSoC 6 MCU debug flows:

- GDB SEGGER J-Link Debugging, which supports J-Link and J-Trace debug probes with the SEGGER J-Link drivers. Refer to the *Modus Toolbox Installation Guide* for information about supported SEGGER J-Link driver versions.
- GDB OpenOCD Debugging, which supports CMSIS debug probes like ULINK2, MiniProg4, and KitProg3.

The following table shows these possible connections:

Debug Connection	Communication Firmware	More Information
SEGGER J-Link	SEGGER-provided J-Link DLL	SEGGER J-Link
OpenOCD via CMSIS-DAP	Cypress-provided KitProg3	
OpenOCD via SEGGER J-Link	OpenOCD-provided firmware	• SEGGER OpenOCD • OpenOCD.org

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. BULK devices are selected first, then HID devices. You can specify a CMSIS-DAP device by using the following methods:

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

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
```

Example

```
openocd -s scripts -c "source [find interface/cmsis-dap.cfg]; cmsis_dap_vid_pid 0x04B4
0xF154; source [find target/psoc6.cfg]; init;reset init; dump_image ram.hex 0x8000000
0x100;reset run; exit"
```

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 KitProg Firmware Loader.

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 the following command is used:

  ```
  cmsis_dap_serial 0B0B0F9701047400
  ```

Example

```
openocd -s scripts -c "source [find interface/cmsis-dap.cfg]; cmsis_dap_serial
0B0B0F9701047400; source [find target/psoc6.cfg]; init; reset init; dump_image ram.hex
0x8000000 0x100; reset run; exit"
```

Selecting by both VID/PID and Serial Number

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

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

Example

```
openocd -s scripts -c "source [find interface/cmsis-dap.cfg]; cmsis_dap_vid_pid 0x04B4
0xF154; cmsis_dap_serial 0B0B0F9701047400; source [find target/psoc6.cfg]; init; reset
init;dump_image ram.hex 0x8000000 0x100;reset run; exit"
```

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 example

```
jlink serial <serial number>
```

If you use this option as part of a configuration command, it can be used only before `init`.

KitProg Firmware Loader

The PSoC 6 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 (HID) protocol and the CMSIS-DAP (Bulk) protocol, which is up to ~2.5 times faster. Both modes can be used via OpenOCD.

ModusToolbox software includes a command-line tool “fw-loader” to update Cypress kits and switch the KitProg firmware from KitProg2 to KitProg3, and back. The following is the default installation directory of the tool on Windows:

<user_home>\ModusToolbox_1.0\tools\fw-loader-1.0\bin

For other operating systems, the installation directory will vary, based on where the software was installed.

Use the fw-loader tool to update the KitProg firmware as required for your needs. KitProg2 does not work with the ModusToolbox software. Likewise, if you update to KitProg3, PSoC Creator won’t work with Cypress kits until you restore KitProg2.

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

Command-Line Options

Run the tool from the command line:

```
[install_path]>fw-loader
```

Use the following options, as needed:

- `--help` or `/?` (or no arguments) – Display a list of supported commands with their descriptions.
- `--device-list` – Display a list of connected devices.
- `--update-kp3 [device-name]` – Update the FW of the specified device name to KitProg3.
- `--update-kp2 [device-name]` – Updates the FW of the specified device name to KitProg2.

Display List of Devices

Use the following command to see if the device is recognized. For KitProg2 devices, use the **SW3 Mode Switch** button on the kit to enable Proprietary mode. Otherwise, the fw-loader tool will not recognize the device. See Mode Switch. If you have more than one KitProg attached, use the following command to display devices by name and identifier:

```
[install_path]>fw-loader --device-list

Cypress Firmware Updater, Version: 1.0.0.214
(C) Copyright 2018 by Cypress Semiconductorfw-loader.exe
All Rights Reserved

Info: Start API initialization
Info: Connected - KitProg2-1718126C03227400
Info: Connected - KitProg3-181B025303147400
Info: Hardware initialization complete (701 ms)
Connected devices:
        1: KitProg2-1718126C03227400
        2: KitProg3-181B025303147400
```

Update to KitProg3

Use the following command to update to KitProg3. If you have more than one KitProg device attached, you must use the `[device-name]` option to specify the KitProg to update.

```
[install_path]>tools>fw-loader-1.0>bin>fw-loader --update-kp3 KitProg2-1718126C03227400

Cypress Firmware Updater, Version: 1.0.0.214
(C) Copyright 2018 by Cypress Semiconductor
All Rights Reserved

Info: Start API initialization
Info: Connected - KitProg3-1718126C03227400
Info: Connected - KitProg3-181B025303147400
Info: Hardware initialization complete (894 ms)
Device 'KitProg3-1718126C03227400' opened successfully
Info: Kit FW is 'KitProg3' ver. 256.0. Upgrade file is 'KitProg2' ver. 1.5
Info: Disconnected - KitProg3-1718126C03227400
Info: FW Upgrade to version: 1.5
Info: Bootloader Version: Major 1, Minor 1, Build 40
Info: Bootloading ...
Info: Verifying ...
FW update completed successfully
```

Switch Back to KitProg2

Use the following command to switch back to KitProg2. If you have more than one device attached, you must use the `[device-name]` option to specify the KitProg to update.

```
[install_path]>fw-loader --update-kp2 KitProg2-1718126C03227400

Cypress Firmware Updater, Version: 1.0.0.214
(C) Copyright 2018 by Cypress Semiconductor
All Rights Reserved

Info: Start API initialization
Info: Connected - KitProg3-1718126C03227400
Info: Connected - KitProg3-181B025303147400
Info: Hardware initialization complete (894 ms)
Device 'KitProg3-1718126C03227400' opened successfully
Info: Kit FW is 'KitProg3' ver. 256.0. Upgrade file is 'KitProg2' ver. 1.5
Info: Disconnected - KitProg3-1718126C03227400
Info: FW Upgrade to version: 1.5
Info: Bootloader Version: Major 1, Minor 1, Build 40
Info: Bootloading ...
Info: Verifying ...
FW update completed successfully
```

Mode Switch

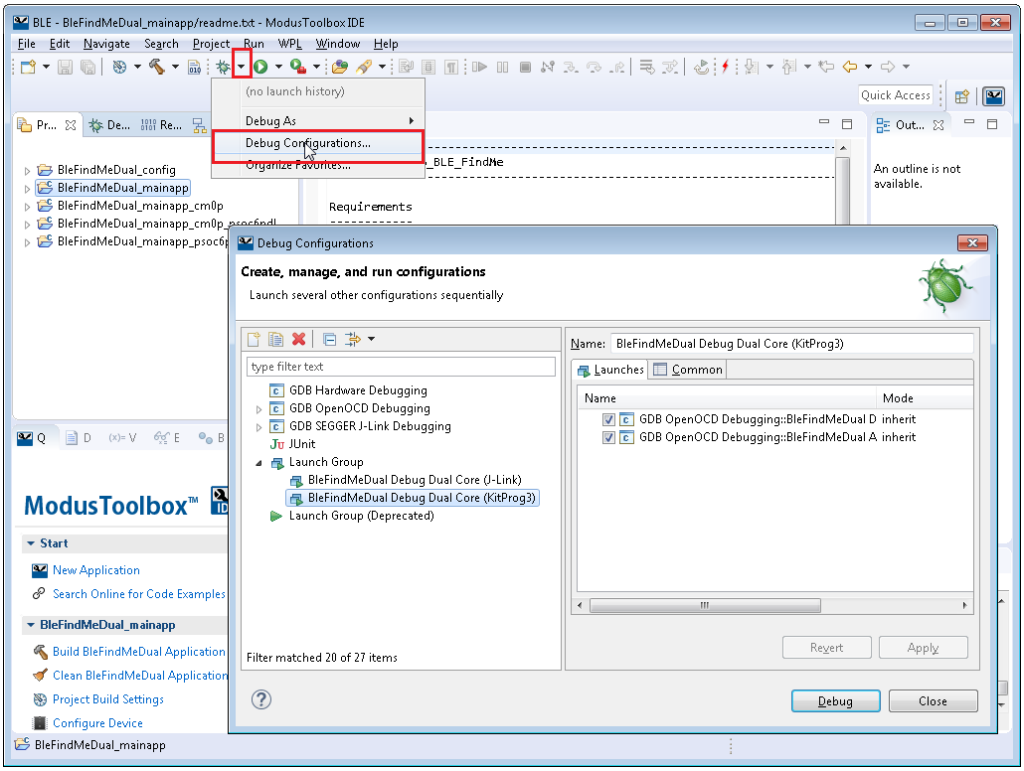
The kits have a set of LEDs and a **SW3 Mode Switch** button. To switch KitProg modes, push and release the **SW3 Mode Select** button on the kit. LED2 indicates the status, as follows:

Firmware	Switches Between	Status of LED2
KitProg2	Proprietary and CMSIS-DAP HID	On = Proprietary Off = CMSIS-DAP HID
KitProg3	CMSIS-DAP HID and Bulk	On = CMSIS-DAP Bulk Breathing (1 Hz) = CMSIS-DAP HID

Dual-Core Debugging

Applications that include the PSoC 6 MCU may exercise code on two cores: CM4 and CM0p. The CM4 is used for application firmware, and the integrated Cortex M0+ is used to offload tasks like communication and security. For these applications, it is possible to debug both cores of the PSoC 6 MCU using the Launch Group. The following provides an example:

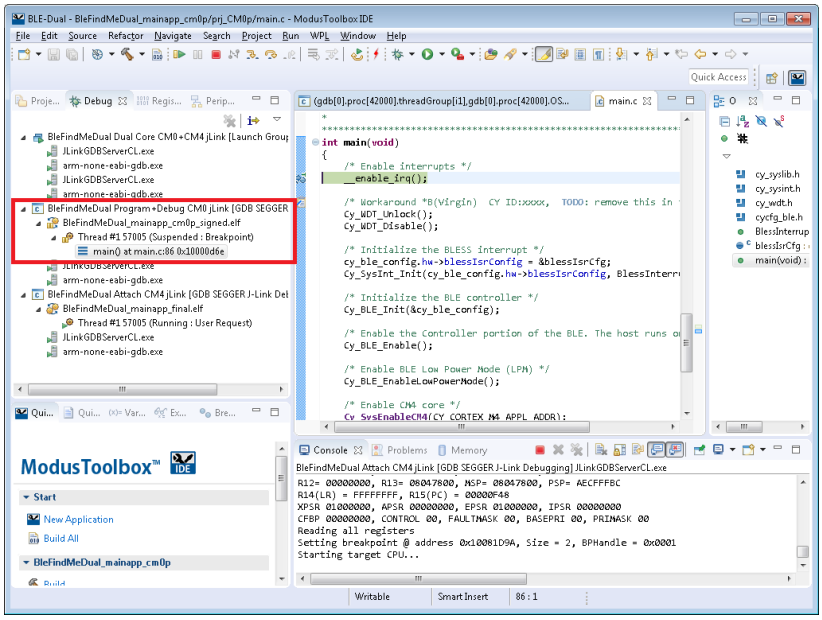
1. Create a “CE212736_PSoC6_BLE_FindMe” starter application for the CY8CKIT-062-BLE kit, and build it. See Build with IDE.
2. After a successful build, open the Debug Configurations dialog. See Program/Debug Launch Configurations.
   a. Click the down arrow next to the **Debug** command, and select **Debug Configurations…**
   b. Then Expand the **Launch Group** node.



3. Select the appropriate **Launch Group**:
   - □ <app-name> Debug Dual Core (J-Link)
   - □ <app-name> Debug Dual Core (KitProg3)

4. Click **Debug**.

The debugger will start and suspend at main for the CM0p core, ready for debugging.



With the CM0p core selected, click **Resume** and observe that the CM4 core is suspended at main, ready for debugging.