AN225588

Using ModusToolbox Software with a Third-Party IDE

Author: Jim Trudeau

Associated Part Family: PSoC 6 MCU

Associated Code Examples: CapSenseButtonsSliderFreeRTOS

Related Application Notes: see Related Documents

AN225588 shows how to integrate the ModusToolbox™2.0 software for PSoC 6 MCU into IAR Embedded Workbench or Keil µVision projects. You will learn how to find the software you need, generate the configuration code, and use that software in your project. When done, you will understand the resources available to you, and how to use them.

Note: This app note applies to ModusToolbox 2.0 only.

ModusToolbox 2.1 implements a much simpler mechanism for exporting a project to an IDE. Like ModusToolbox 2.0, you use the Project Creator tool to create a project folder that has all files required for the application. Then from the command line, you issue an IDE-specific `make` command to export your project to your selected IDE. The *ModusToolbox User Guide* chapter Exporting to IDEs tells you how for these supported IDEs:

- IAR Embedded Workbench
- Visual Studio Code
- Keil µVision
- Eclipse IDE

Contents

1 Introduction

ModusToolbox is the comprehensive package of tools and software for PSoC 6 MCU and IoT designers. It incorporates configuration tools, low-level drivers, middleware libraries, and operating system support that enable you to create MCU and wireless applications. It also includes the optional Eclipse-based ModusToolbox IDE. Unless specifically stated otherwise, ModusToolbox resources are compatible with Linux, macOS, and Windows-hosted environments.

ModusToolbox software is designed to adapt to your workflow. That's what this app note is about.

The ModusToolbox software libraries and generated code can be used in any IDE. Simply add the files and necessary paths to your project. Every IDE has idiosyncrasies. This app note discusses the steps needed to use either IAR Embedded Workbench (IAR EW) or Keil µVision tools.

The app note includes an example to show you how it's done. It uses several libraries such as the PSoC 6 driver library, the CapSense® middleware library, and FreeRTOS.

In this app note you will learn about:

- The overall architecture of ModusToolbox software at a high level
- Where to get the software you need for your application
- How to use that software in your preferred IDE

This app note covers the PSoC 6 MCU ecosystem for ModusToolbox v2.0. It does not use the Cypress Bluetooth SDK. A Bluetooth Low Energy library is available for PSoC 6 MCU devices with integrated BLE.

An earlier version of this app note discusses this same information for ModusToolbox v1.1, which differs significantly. If you are using ModusToolbox v1.1, make sure you use the original version of this app note.

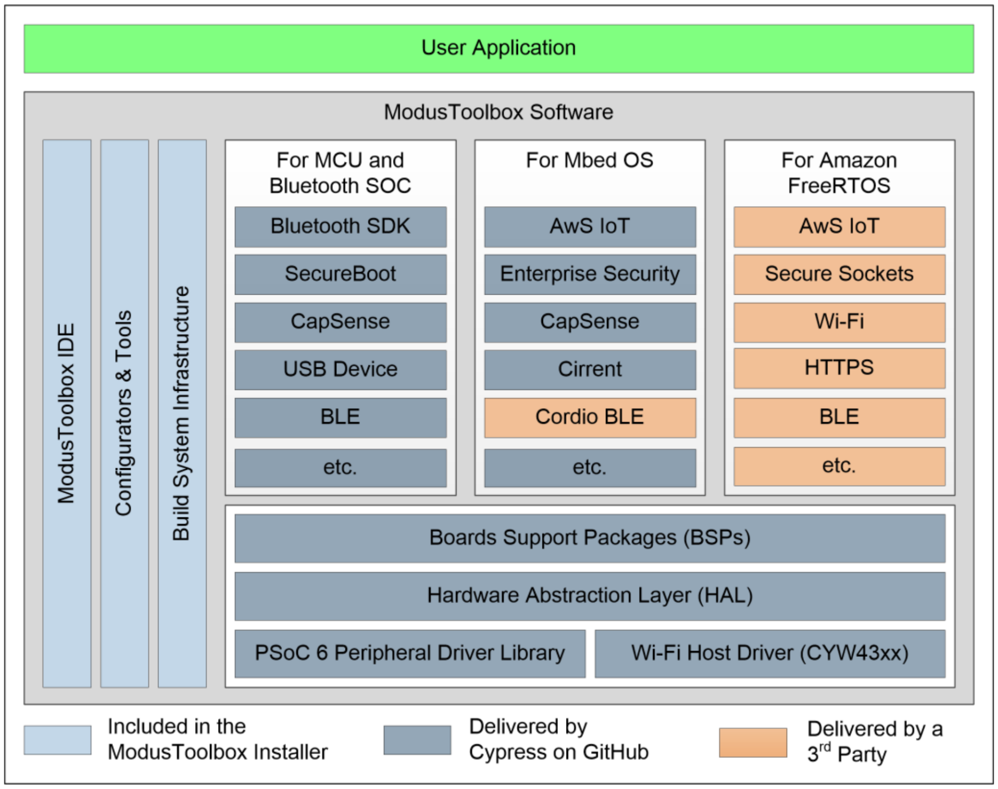
You must install the ModusToolbox Software environment to use ModusToolbox software effectively. It includes necessary configuration tools and a build system that makes project creation easy.

The ModusToolbox installer provides the core resources you need to get started, such as configurators that generate code based on your hardware design. The installer *does not include libraries*. Cypress provides libraries and other software at the Cypress GitHub site. Cypress software resources available at GitHub support one or more of these target ecosystems:

- PSoC 6 MCU and Connectivity ecosystem – a full-featured platform for PSoC 6 MCU, Wi-Fi, Bluetooth, and Bluetooth Low Energy application development
- Mbed OS ecosystem – provides an embedded operating system, transport security and cloud services to create connected embedded solutions
- Amazon FreeRTOS ecosystem – extends the FreeRTOS kernel with software libraries that make it easy to securely connect small, low-power devices to AWS cloud services

Some ModusToolbox resources support all ecosystems. Others are specific to an ecosystem. The block diagram in Figure 1 is not a comprehensive list. However, it conveys the idea that multiple resources are available to you. This app note discusses how to get the resources, and how to use them in your preferred IDE.

Figure 1. Some ModusToolbox Resources



2 Getting the Files and Libraries You Need

Most developers expect to use software provided by the silicon manufacturer or a third party. This section is not about what's available, but about how to get it.

You can use ModusToolbox software – that is, get the files and libraries into a project – in several ways:

- From the command line
- With the ModusToolbox IDE
- As a collection of modular assets that you use independently – discussed briefly here
- With the ModusToolbox Project Creator and a third-party IDE – that's what this app note is about

This app note does not discuss the first two options. To learn more about those paths, see Running ModusToolbox from the Command Line or the ModusToolbox IDE User Guide.

To use the libraries as modular assets, go to the Cypress GitHub site, find the repository that contains what you need, identify the release within the repository you intend to use, and get that code. You can download a zip file or clone the repository. You can then add the code to your project.

For a developer or company familiar with PSoC 6 MCU development and ModusToolbox software, this is a very effective approach. You collect a set of "known good" libraries that work together. You install them in your source control system on your company server. You use them as required in your project. You would typically point your projects at these files rather than replicating them locally. This is part of the flexibility built into the ModusToolbox approach. You get what you need, and only what you need.

However, there are a lot of libraries. Knowing which you need, and which version of each you need, is non-trivial. The ModusToolbox build infrastructure makes this process simple and transparent. It eliminates any uncertainty.

You use a tool called the "Project Creator" to identify the kit you want to use. You then identify a code example that has the same kind of functionality you want in your final application. The build infrastructure finds every file required automatically and puts them all in a single project folder. You can then use those files in your IDE. This is the fourth path listed above, and what this app note is about.

When you are comfortable with how the libraries are structured, there is no need to use the Project Creator. You can treat the available assets as stand-alone modules and use them as required for your project.

2.1 How the Build System Works

The build system is documented in Running ModusToolbox from the Command Line. Every application has a makefile. That makefile specifies everything required for the application. When you create a project, the build system creates a project folder that contains or references everything needed for the application.

Source files specific to the code example are in the code example's folder hierarchy. In the project creation process, all such files are discovered automatically. The makefile does not list each file. Files found and duplicated include all .c, .h, .cpp, .s, .a, and .o files.

Each library used in the project is identified by its own *.lib* file. This file is also somewhere in the code example's folder hierarchy, typically in a *libs* folder. The *.lib* file contains a single line of text that is the URL to a git repository, and a tag that identifies a specific version of the code within that repository.

For example, this is the content of a *.lib* file that includes the emWin graphics library:

```
https://github.com/cypresssemiconductorco/emwin/#release-v5.48.1
```

The build system clones the repository, checks out the specified commit, and copies all the files into the project folder. In the project folder, all such library code is placed in the *libs* folder.

One such *.lib* file specifies the Board Support Package (BSP). The BSP typically includes additional *.lib* files. The process of discovering and duplicating libraries proceeds automatically and recursively. The BSP also includes:

- startup code for each supported toolchain
- linker files for each supported toolchain
- the configuration design files (see Configurators and Device Configuration Files)

For the ModusToolbox IDE, the build system also creates an Eclipse project. The project folder is in the Eclipse workspace. For other IDEs, you specify where you want the project folder. No IDE project file is created.

The build system creates an output file (such as an .elf file) and supports multiple toolchains for that purpose:

Table 1. Supported Toolchains

Tools	Host OS
GNU Arm Embedded Compiler v7	macOS, Windows, Linux
Arm Compiler v6 (Keil tools)	Windows, Linux
Embedded Workbench v8.2 (IAR tools)	Windows

You control which is used by setting a variable in the makefile. The ModusToolbox installer installs the GNU Arm tools. You must provide the other toolchains if you wish to use them. Because this app note is about the IAR and Keil tools, it assumes you have those toolchains installed. Note that for the Keil tools, ModusToolbox software uses the v6 compiler. If you do not have that compiler, you must install it. It is available here.

This feature means that you can create a project and use either the IAR or Keil toolchain without using the corresponding IDE. However, this app note assumes that you use the IDE to manage your projects and builds.

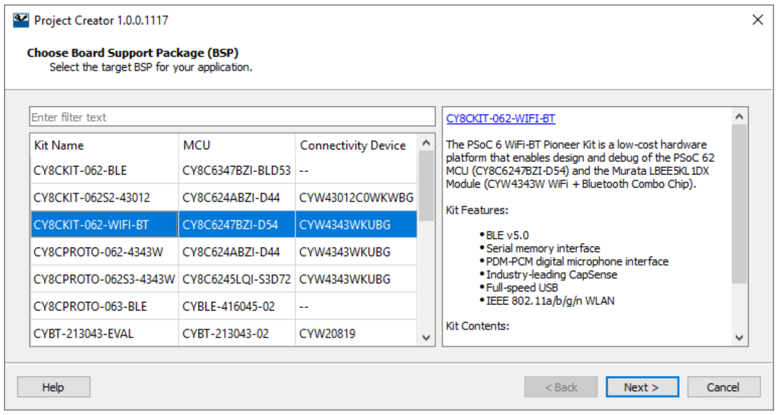
2.2 Using Project Creator

To take advantage of ModusToolbox automatic file discovery, you can use the Project Creator. It is a simple wizard you use to select a kit and a code example. See ModusToolbox Project Creator Guide for documentation.

First, launch the Project Creator tool, located here: *<ModusToolbox Install>\tools_x.y\project-creator\*. The screenshots that follow show the Windows-hosted tool. The same functionality is available on macOS and Linux.

After you launch Project Creator, you select the target kit.

Figure 2. Selecting a Kit in Project Creator


Project Creator 1.0.0.1117

Choose Board Support Package (BSP)
Select the target BSP for your application.

Enter filter text

Kit Name	MCU	Connectivity Device
CY8CKIT-062-BLE	CY8C6347BZI-BLD53	--
CY8CKIT-062S2-43012	CY8C624ABZI-D44	CYW43012C0WKWBG
CY8CKIT-062-WIFI-BT	CY8C6247BZI-D54	CYW4343WKUBG
CY8CPROTO-062-4343W	CY8C624ABZI-D44	CYW4343WKUBG
CY8CPROTO-062S3-4343W	CY8C6245LQI-S3D72	CYW4343WKUBG
CY8CPROTO-063-BLE	CYBLE-416045-02	--
CYBT-213043-EVAL	CYBT-213043-02	CYW20819

CY8CKIT-062-WIFI-BT

The PSoC 6 WiFi-BT Pioneer Kit is a low-cost hardware platform that enables design and debug of the PSoC 62 MCU (CY8C6247BZI-D54) and the Murata LBEE5KL1DX Module (CYW4343W WiFi + Bluetooth Combo Chip).

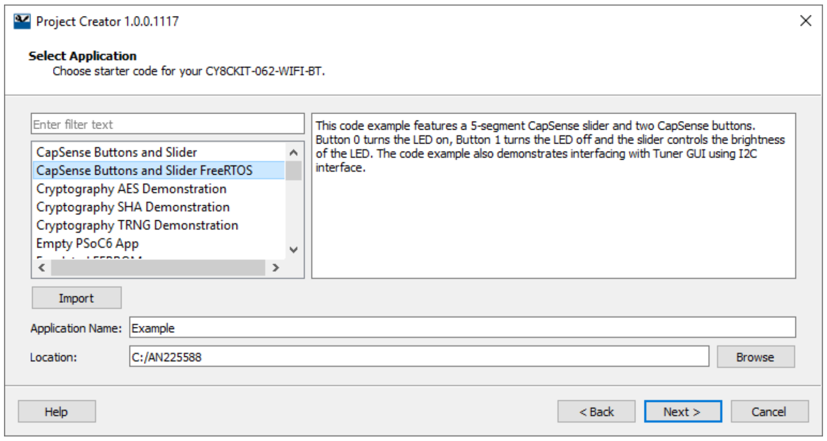
Kit Features:

- BLE v5.0
- Serial memory interface
- PDM-PCM digital microphone interface
- Industry-leading CapSense
- Full-speed USB
- IEEE 802.11a/b/g/n WLAN

Kit Contents:

Then click **Next** and choose the code example and a location for the project folder.

Figure 3. Select the Code Example


Project Creator 1.0.0.1117

Select Application
Choose starter code for your CY8CKIT-062-WIFI-BT.

Enter filter text

CapSense Buttons and Slider
CapSense Buttons and Slider FreeRTOS
Cryptography AES Demonstration
Cryptography SHA Demonstration
Cryptography TRNG Demonstration
Empty PSoC6 App

This code example features a 5-segment CapSense slider and two CapSense buttons. Button 0 turns the LED on, Button 1 turns the LED off and the slider controls the brightness of the LED. The code example also demonstrates interfacing with Tuner GUI using I2C interface.

Application Name: Example

Location: C:/AN225588

Click **Next** to see a summary of your choices. Then click the **Create** button. The ModusToolbox build system creates the project folder and populates it with every file required for the application.

The Project Creator provides a local copy of each file in the project folder, so the application is completely self-contained. As you develop your project, you can modify files as required, without affecting the originals. If at a later time you want to update a library (for example, a new version of a BSP), the Project Creator will not overwrite changed files. This gives you the opportunity to check in or preserve your changes. See Changing Default BSP Files for more information on modifying default files.

You can also create a project from the command line. If you clone or copy the original starter application, you can open a command window in that directory and issue a `make getlibs` command. That sets off the project creation process. See Running ModusToolbox from the Command Line for detailed documentation.

2.3 Using Library Manager

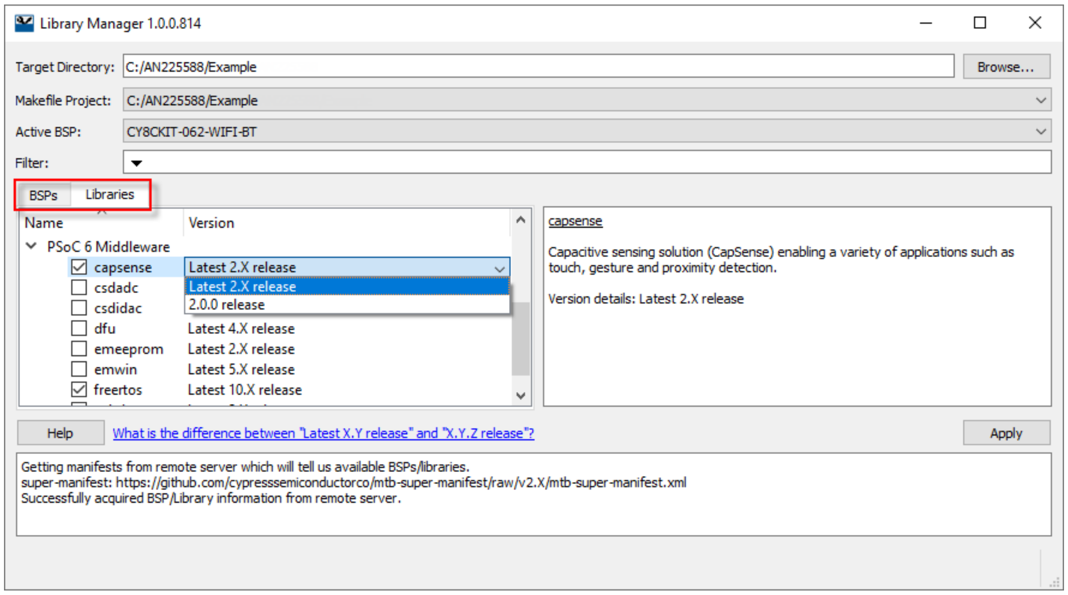
Use the Library Manager to change the BSP used by an application. You can also add, remove or update middleware libraries. This tool gives you the capability to modify the project to fit your use case more closely. See the ModusToolbox Library Manager Guide for documentation on this tool.

Launch the Library Manager, located here: *<ModusToolbox Install>\tools_x.y\library-manager\*

In the **Target Directory**, you select the root directory in which the tool discovers code example makefiles. All makefiles in this directory tree appear in the **Makefile Project** menu. Select the project you wish to modify. When you select the project, the **Active Board** for that project appears. The list of libraries includes only those that support the board.

You can update either the board or the libraries. Click the current **Version** to see available choices.

Figure 4. ModusToolbox Library Manager


Library Manager 1.0.0.814
Browse...
BSPs
Libraries
Help
What is the difference between "Latest X.Y release" and "X.Y.Z release"?
Apply

Target Directory: C:/AN225588/Example
Makefile Project: C:/AN225588/Example
Active BSP: CY8CKIT-062-WIFI-BT
Filter:

Name	Version
PSoC 6 Middleware	
☑ capsense	Latest 2.X release (Latest 2.X release / 2.0.0 release)
☐ csdadc	
☐ csdidac	
☐ dfu	Latest 4.X release
☐ emeeprom	Latest 2.X release
☐ emwin	Latest 5.X release
☑ freertos	Latest 10.X release

capsense

Capacitive sensing solution (CapSense) enabling a variety of applications such as touch, gesture and proximity detection.

Version details: Latest 2.X release

```
Getting manifests from remote server which will tell us available BSPs/libraries.
super-manifest: https://github.com/cypresssemiconductorco/mtb-super-manifest/raw/v2.X/mtb-super-manifest.xml
Successfully acquired BSP/Library information from remote server.
```

When you change the BSP (i.e. Active Board), the Library Manager modifies the makefile that describes the application. It sets the `TARGET` variable based on your choice. For example, if you change the BSP to the PSoC 6 Wi-Fi BT Prototyping Kit, the tool updates the makefile like this:

```
# Target board/hardware
TARGET=CY8CPROTO-062-4343W
```

When you add, remove, or change the version of a library, the Library Manager updates, duplicates, adds, or removes the corresponding .lib files as required.

Whenever you have completed your changes and click **Apply**, the tool runs `make getlibs` to recreate the application using the BSP and libraries you have selected.

2.4 Summary of Getting the Files You Need

As noted at the start of this section, a very common workflow is to have a “known good” set of libraries and enablement code on your company’s development server. When you build an application, you add files from the approved codebase. ModusToolbox fully supports this model.

The challenge is in knowing what’s available and what you need. The ModusToolbox build system makes it easy to find that answer. The process is extremely simple.

1. Launch the Project Creator.
2. Identify the kit and code example.
3. Click **Create.**

The ModusToolbox build system does the rest, and you get every file you need in one place.

You can use the Library Manager to change the BSP, libraries, or library versions. You can of course edit the makefile manually if you prefer.

After you have gained some familiarity with the available libraries, you can go to the Cypress GitHub site and get what you need when you need it. You are not required to use Project Creator or the ModusToolbox build system.

3 Understanding the ModusToolbox BSP

The BSP is a central feature of ModusToolbox software. The BSP specifies several critical items for the application, including:

- hardware configuration files for the device (e.g., *design.modus*)
- startup code and linker files for the device
- other libraries that are required to support a kit

For example, a BSP may include the psoc6hal, core-lib, and psoc6pdl (driver) libraries. The list of libraries included by the BSP varies based on the kit functionality. For example, if a kit supports CapSense®, the standard BSP for that kit will include the *capsense* library by default, even if your application does not use CapSense. If necessary, you can use the Library Manager to modify which libraries are added to an application.

3.1 Configurators and Device Configuration Files

ModusToolbox provides a series of configuration tools, such as the Device Configurator, the CapSense Configurator, and several others. Configurators are fully explained in the ModusToolbox documentation. Each is a stand-alone executable for macOS, Linux, or Windows. Each configurator provides a user-friendly interface to customize the settings for the associated driver or middleware library. Each configurator saves your design choices in a configuration file, such as *design.modus*.

The BSP includes configuration files. They define a default, generic configuration for each kit, including which clocks are used, clock frequencies, which peripherals are enabled, what pins are used, and so forth. The design files are here:

<ProjectFolder>\libs\TARGET_<KIT>\COMPONENT_BSP_DESIGN_MODUS

When you use Project Creator, the ModusToolbox build system automatically generates configuration code based on these design files. That code is placed beside the design files, in a folder called *GeneratedSource*. If you modify the design, the generated source is updated automatically when you save changes. However, see Changing Default BSP Files for the risks involved with modifying local copies of default files.

3.2 Startup Code and Linker Files

The BSP contains default startup code and linker files for each supported toolchain. After creating a project folder, you find the startup code here:

<ProjectFolder>\libs\TARGET_<KIT>\startup\TOOLCHAIN_<NAME>

Linker files are similarly located here:

<ProjectFolder>\libs\TARGET_<KIT>\linker\TOOLCHAIN_<NAME>

3.3 Changing Default BSP Files

The Project Creator provides local copies of all files in the application, both middleware and BSP files. It is unlikely that you will modify middleware files.

However, it is likely that you will modify some BSP files for your application. You may need to customize a linker script, the startup code, or the design files that generate the source code. Modifying local copies of default files is not without risk because the Library Manager may overwrite these files.

For example, assume that you update the *design.modus* file to configure and enable a peripheral for your application, or you update a linker file or the startup code. Your local copies have changed from the default BSP. If you then use the Library Manager to change the version of your BSP, the Library Manager gets the default implementation for that version of the BSP, and your changes are lost without warning.

If you do not use the Library Manager to modify the BSP version or change to a different BSP, your local changes to BSP files are safe.

To eliminate this risk completely, you can override the *design.modus* file or provide your own BSP. Any changes are safe and will not be replaced. See a BSP User Guide for details. Each BSP has its own documentation, the link is to a representative example.

4 The Example

This app note includes an example to show you how to integrate ModusToolbox software into an IDE project. The example is based on the CapSense Buttons Slider FreeRTOS example. The example on GitHub is built for ModusToolbox IDE. All the work to move the example into the third-party IDE is complete for this app note.

To work with the example, download *AN225588.zip* and unzip the package to a location on your computer. The example projects were developed using IAR Embedded Workbench version 8.32 and Keil µVision version 5.25.

All work required to port the example into IAR and Keil IDEs has been done for you. The steps described in this app note are informational. You can do them if you wish to start from scratch, but the example is self-contained and complete.

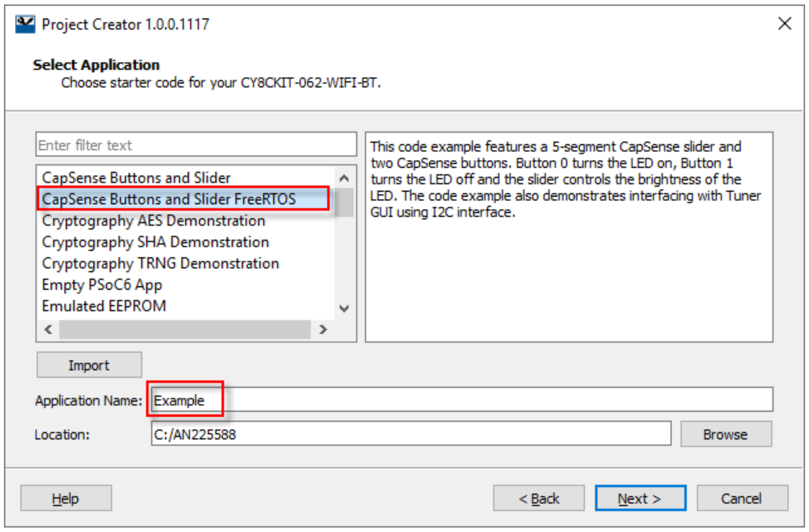
You also do not need to build and run the example. You can simply use it as a reference for one way to organize ModusToolbox software into a project in your IDE. If you do run the application, CapSense BTN1 turns a kit LED off. CapSense BTN0 turns it back on. The CapSense slider on the kit changes the brightness.

There are three folders in the example:

- *Example*
- *AN225588-IAR*
- *AN225588-Keil*

To build the project folder, we used the Project Creator and chose the WiFi kit BSP (CY8CKIT-062-WIFI-BT). Then we chose the starter application "CapSense Buttons and Slider FreeRTOS", entered a project name *Example* and specified a location for the project folder. See Using Project Creator.

Figure 5. Creating the Example Project Folder


Project Creator 1.0.0.1117

Select Application
Choose starter code for your CY8CKIT-062-WIFI-BT.

Enter filter text

CapSense Buttons and Slider
CapSense Buttons and Slider FreeRTOS
Cryptography AES Demonstration
Cryptography SHA Demonstration
Cryptography TRNG Demonstration
Empty PSoC6 App
Emulated EEPROM

This code example features a 5-segment CapSense slider and two CapSense buttons. Button 0 turns the LED on, Button 1 turns the LED off and the slider controls the brightness of the LED. The code example also demonstrates interfacing with Tuner GUI using I2C interface.

Application Name: Example
Location: C:/AN225588

We used the Library Manager (see Using Library Manager) to add the BSP for the BLE kit (TARGET_CY8CKIT-062-BLE) and removed the TARGET_CY8CPROTO-062-4343W BSP, which is not used in this App Note.

The IDE folders contain workspace and project files. For Keil tools, there are also some support files. Each IDE has two projects. One is for the CY8CKIT_062_WIFI_BT board, hereafter the WiFi kit. The other is for the CY8CKIT-062-BLE board, hereafter the BLE kit. We set up the project files in each project to use the correct BSP, as well as the middleware. How we did that is what the rest of this app note is about.

Each project for each IDE points to the files in *Example*. For a given IDE, the projects for each kit are essentially identical except that each uses the files from the corresponding BSP. All the middleware files are identical and reused in each project for each kit in each IDE.

The *Example* folder is a superset that contains all files for all libraries required for all supported IDEs, including the ModusToolbox IDE. Some of what is in the *Example* folder is unused and not necessary for a third-party IDE project. The steps for each IDE point out some instances where you do not need specific files.

Table 2. Contents of the Project Folder

Folder	Contents
Example	At the top level in this folder, application-specific source files, the example's readme file, and the application makefile (used by ModusToolbox)
images	Used in the example readme file
\libs\capsense	The CapSense middleware library
\libs\core-lib	Shared header files that provide device-independent basic types and utilities
\libs\freertos	The version of the FreeRTOS open source library tested with this example
\libs\psoc6cm0p	For dual CPU devices, a pre-built Arm Cortex-M0+ (CM0+) application provided as a C array to be compiled into the final application
\libs\psoc6hal	The PSoC 6 MCU hardware abstraction library, including pin package files.
\libs\psoc6make	Files required by the ModusToolbox build system. These are not used by the IAR or Keil IDEs.
\libs\psoc6pdl	The PSoC 6 MCU peripheral driver library
\libs\TARGET_CY8CKIT-062-BLE	The BSP for each kit supported by the example. Each BSP contains startup code and linker files for each supported toolchain for that kit. It also contains default design files, generated source code, and BSP-related source code..
\libs\TARGET_CY8CKIT-062-WIFI-BT	
\libs\TARGET_CY8CKIT-062-BLE.lib	The .lib file that instantiates each BSP library
\libs\TARGET_CY8CKIT-062-WIFI-BT.lib	

4.1 Adapting for the IDE Build System

ModusToolbox software implements a build system as described briefly in How the Build System Works. In addition to creating the project folder and supporting multiple toolchains, the ModusToolbox build system has some built-in defines and includes. The source code in a code example may take advantage of this and may not explicitly include certain headers or define certain terms.

As a result, when you use the IAR or Keil IDE, you may get compile-time errors when you build the project. The example used in the app note demonstrates this side effect of changing build systems from ModusToolbox to your IDE.

For example, a BSP uses the Cypress HAL. In the ModusToolbox build, this define is built into the makefile for the BSP:

```
DEFINES+=CY_USING_HAL
```

If you are not using the makefile (for example, adding files to an IDE project and building within the IDE), this term isn't defined.. If you do not define the symbol, an error occurs. For example, the µVision IDE build output says:

```
led_task.c(90): error: use of undeclared identifier 'CYBSP_USER_LED'
```

Even if you are unfamiliar with the subtleties of the ModusToolbox build system, figuring out what's going on is simple. A search for that term in the project folder quickly finds that `CYBSP_USER_LED` is conditionally compiled based on:

```
#if defined(CY_USING_HAL)
```

To fix the problem, we added this line to *led_task.h*:

```
#define CY_USING_HAL       (1u)
```

Similarly, the master header file for the peripheral driver library, *cy_pdl.h*, may not be included by a code example, even if it uses the PDL. In the example, we added this line to *capsense_task.c* to include the necessary header:

```
#include "cy_pdl.h"
```

If you encounter compile-time errors because the ModusToolbox build system does some things automatically, simply add the required `#define` or `#include` statement to the code. Conversely, if you make such changes to the source code and then try to build the example with the ModusToolbox command line, the build may fail because of duplicate defines.

4.2 Using the Example

The following sections discuss how the example works with IAR Embedded Workbench, and with Keil µVision tools. Each section covers the same topics and tasks, from the perspective of the individual toolchain. Each section discusses which files you need, where they are, and some IDE-specific issues.

The instructions are not prescriptive. With any good tool, there are often many ways to accomplish a task. The goal of the app note is to show you one way, so you understand what needs to be done.

5 Using ModusToolbox Software in IAR EW

This section outlines high-level tasks you need to perform to create a PSoC 6 MCU application using the IAR Embedded Workbench tools and ModusToolbox software. Each task has a description of how that task was implemented in the example. You do not need to perform these tasks; the example is complete.

If you were working from scratch, you would do three things to begin the process:

- Use the Project Creator to create a project folder with all the files you need
- Use the Library Manager (if necessary) to add/remove any library (including a BSP)
- Create an IAR workspace and an empty C project within that workspace.

The sections that follow describe what to do with that project, and include

- IAR Project Setup
- Add BSP Files to the Project
- Add Generated Code to the Project
- Add Code for a CM0+ Application (example-specific)
- Add FreeRTOS Files (example-specific)
- Add Other Libraries
- Manage Include Paths
- Write Your Software
- Build, Run, and Debug the Project

As noted, you do not need to do any of these tasks with the example. Open the IAR workspace to follow along: *AN225588.eww.*

5.1 IAR Project Setup

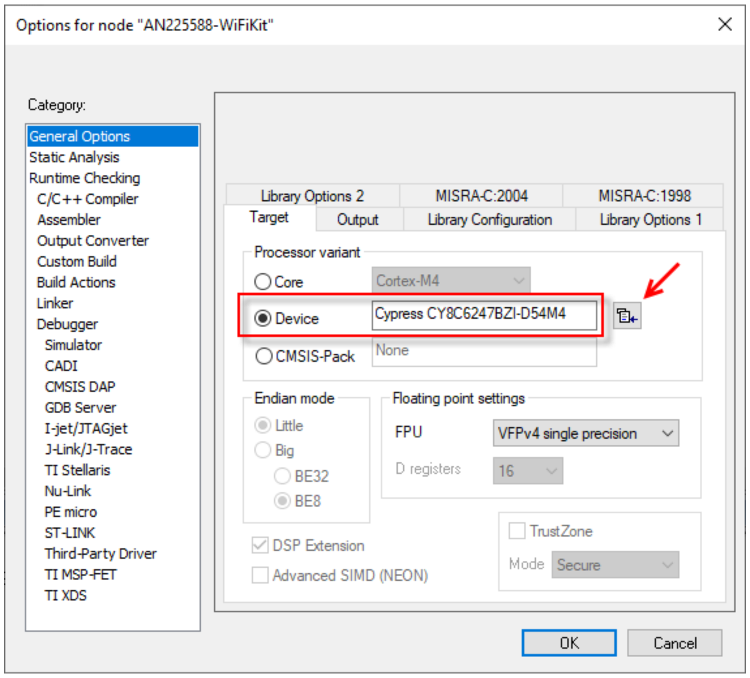
Note: Work is under way so that the ModusToolbox build system will produce an IAR project connection file. Inter-project connection is an IAR feature. The project connection sets most options and adds most if not all project files automatically, making project creation a very friendly process. We will update this app note with the particulars when that feature is available.

In the meantime, there are some matters you should be aware of when setting up a project in the IAR tools.

Processor Selection

The example project is set up for you. If you begin with an empty C project, you need to specify the target device. Figure 6 shows the setup for the device on the WiFi kit.

Figure 6. Device Setup



Options for node "AN225588-WiFiKit"

Category:

- General Options
- Static Analysis
- Runtime Checking
  - C/C++ Compiler
  - Assembler
  - Output Converter
  - Custom Build
  - Build Actions
  - Linker
  - Debugger
    - Simulator
    - CADI
    - CMSIS DAP
    - GDB Server
    - I-jet/JTAGjet
    - J-Link/J-Trace
    - TI Stellaris
    - Nu-Link
    - PE micro
    - ST-LINK
    - Third-Party Driver
    - TI MSP-FET
    - TI XDS

Tabs: Library Options 2 | MISRA-C:2004 | MISRA-C:1998 | Target | Output | Library Configuration | Library Options 1

Processor variant
- ○ Core: Cortex-M4
- [✓ ● Device]: Cypress CY8C6247BZI-D54M4
- ○ CMSIS-Pack: None

Endian mode
- [✓ Little]
- Big
  - BE32
  - BE8

Floating point settings
- FPU: VFPv4 single precision
- D registers: 16

[✓] DSP Extension
[ ] Advanced SIMD (NEON)
[ ] TrustZone
Mode: Secure

Include Paths

Being self-contained, all paths in the example are project-relative. This works well when you use the Project Creator, because all the files are in one location relative to your project file. In your workflow, you may prefer to use source files from a development server rather than a local snapshot. If so, you need to manage include paths.

Preprocessor Symbols

In the IAR IDE, preprocessor symbols are set in the **Project** > **Options** > **C\C++ Compiler** > **Preprocessor** panel. These are already set correctly in the example. A symbol identifies the target device. You can specify the preprocessor symbol manually. The available symbols are listed in the *cy_device_headers.h* file. In the Project Creator folder used in the example, it is here: *\Example\libs\psoc6pdl\devices\include*\.

The example also defines DEBUG so that debug information is created.

Linker File Selection

While an IDE may include a database of configuration and setup files, like a linker file, these are provided directly by ModusToolbox software for supported kits and toolchains. It is likely that this linker file is more current than what comes with an IDE.

In the **Project** > **Options** > **Linker** panel, the project overrides the default and points to the correct linker file in the BSP folder. In the example, the linker file is here:
\Example\libs\TARGET_<KIT> \linker\TOOLCHAIN_IAR\

5.2 Add BSP Files to the Project

Remember that all these steps are done in the example. The description tells you about the task and how it is done.

The BSP includes some kit-specific files such as startup code. In the IDE project, the files are in a group called **BSP**. The table lists the files, and their origin in the BSP folder:

Table 3. BSP Files for IAR EW

File	Note	Folder
cybsp.c	Code to start up the hardware on the kit	*libs\TARGET_<KIT>*
system_psoc6_cm4.c	Clock and system setup	*libs\TARGET_<KIT>\startup*
startup_psoc6_01_cm4.s	IDE-specific startup code	*libs\TARGET_<KIT>\startup\TOOCHAIN_IAR*

The BSP has additional source files to support retargeting standard I/O for the kit. The example application does not use standard I/O, so those files are not added to the IDE project. This highlights a key feature of how ModusToolbox software is structured and illustrates how you can adapt ModusToolbox software to your workflow.

A ModusToolbox BSP has everything required for every common use case. Using standard I/O is a common use case. As a result, the required files are present in the BSP. This gives you a choice.

You can add every source file in the library repository to your project, whether your application needs it or not. You rely on the linker to eliminate any code not used. This approach means that you don't need to know what is implemented where or identify what you do and do not need. Just add everything.

However, your workflow may require that all code has a purpose. In that case, you can remove anything you don't need. In the example we intentionally did not add the standard I/O-related files to illustrate the point.

5.3 Add Generated Code to the Project

The Project Creator automatically executes the design files in the BSP and generates configuration code. The files appear in the *GeneratedSource* folder beside the *design.modus* file. Those files are found here:

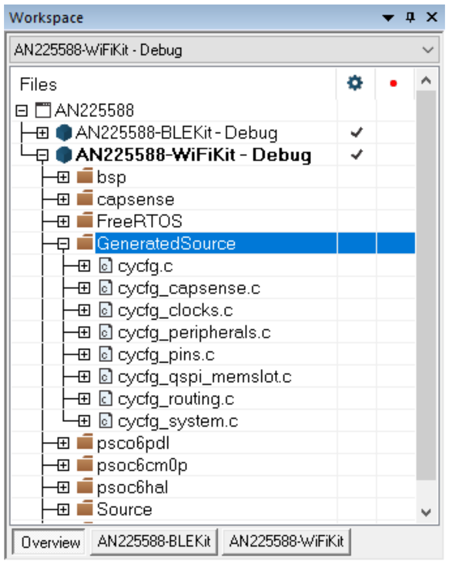
Example\libs\TARGET_<KIT>\COMPONENT_BSP_DESIGN_MODUS\GeneratedSource\

If the *GeneratedSource* folder does not exist (for example, you didn't use the Project Creator), simply use the Device Configurator to open the *design.modus* file and save it again. This generates the code.

The example has a **GeneratedSource** group (highlighted in the figure). To add files, right-click the group, or simply drag and drop the files into the group. Add the files directly from the *GeneratedSource* folder in the Project Creator's project folder. If you modify the design, changed files will be noted in the project automatically. If your design generates new files, add them to the project.

The actual design configuration files (for example, *design.modus*) are not source files, so have not been added to the IDE project in the example. You can do so if you wish and configure the IDE to open the Device Configurator.

Figure 7. Add Generated Source Files



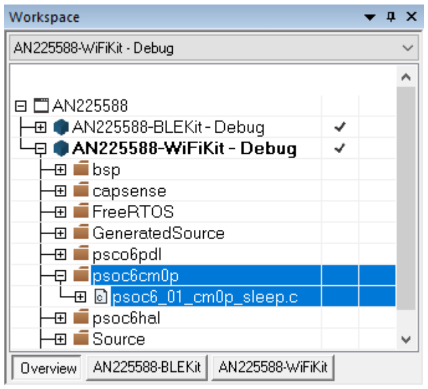
The example application makes no changes to the default design files. Your application might. See Understanding the ModusToolbox BSP and in particular Changing Default BSP Files to understand the risks involved in changing default BSP files like *design.modus*.

5.4 Add Code for a CM0+ Application

This step is necessary for PSoC 6 MCU devices with dual CPUs. Each kit supported by this app note has a PSoC 6 MCU with a dual CPU. The example application enables the Cortex-M4 (CM4) CPU and puts the Cortex-M0+ CPU (CM0+) to sleep. The CM0+ application for this purpose is found here: *\<ProjectFolder \libs\psoc6cm0p\COMPONENT_CM0P_SLEEP\*

Each application is provided as a source file in the form of a C array that is compiled into the final application.

Figure 8. Add a CM0+ Application



Different CM0+ applications have different memory footprints, so linker sections must be adjusted if you change the CM0+ application. See Dual-CPU Devices for a more detailed discussion of this topic.

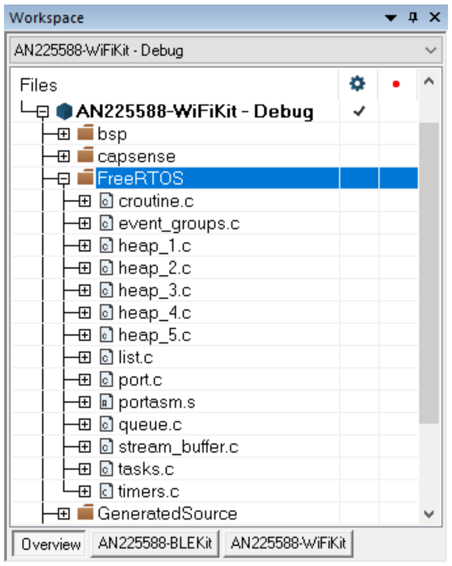
5.5 Add the FreeRTOS Files

This step is example-specific, because it uses FreeRTOS.

The Project Creator puts the files here: *\<ProjectFolder>\libs\freertos\*. The source files exist in several subfolders within this hierarchy, based on the open source project for FreeRTOS. For simplicity, the example project has all these files in a single (flat) group.

FreeRTOS is not a Cypress product. What Cypress provides on GitHub may not be the latest version. What is on the Cypress GitHub site, and what code examples refer to, is the most recent version tested with Cypress devices and firmware. You may substitute any version of the RTOS that fits your development requirements, but keep in mind that it may not have been tested with ModusToolbox software.

Figure 9. Add FreeRTOS Files



5.6 Add Other Libraries

If you use the Project Creator, every library needed by the application is here: *\<ProjectFolder>\libs\<name>*. The additional libraries used include:

- capsense
- core-lib

- psoc6cm0p
- psoc6hal
- psoc6pdl
- TARGET_<KITNAME> (in the BSP group in the IDE project file). You can organize them however you wish.

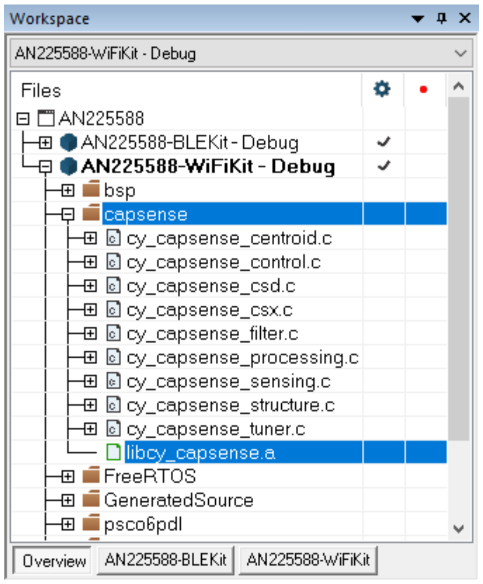
Add source files from each library to the project. The example sets up a group for each library and adds the files to that group. The typical workflow is to use the entire library.

Middleware may provide a static (precompiled) library for some functionality. If there is a static library, it is in a folder that identifies the toolchain for which it was compiled. The CapSense library is a good example. It has static libraries for hardware and software floating point support. The project uses software floating point. For the IAR tools the static library is located here.

\Example\libs\capsense\COMPONENT_SOFTFP\TOOLCHAIN_IAR\

There is a similar folder for each supported toolchain. When adding the files for any given library, look for *TOOLCHAIN* folders and add any required static library.

Figure 10. Add Source and Static Library Files



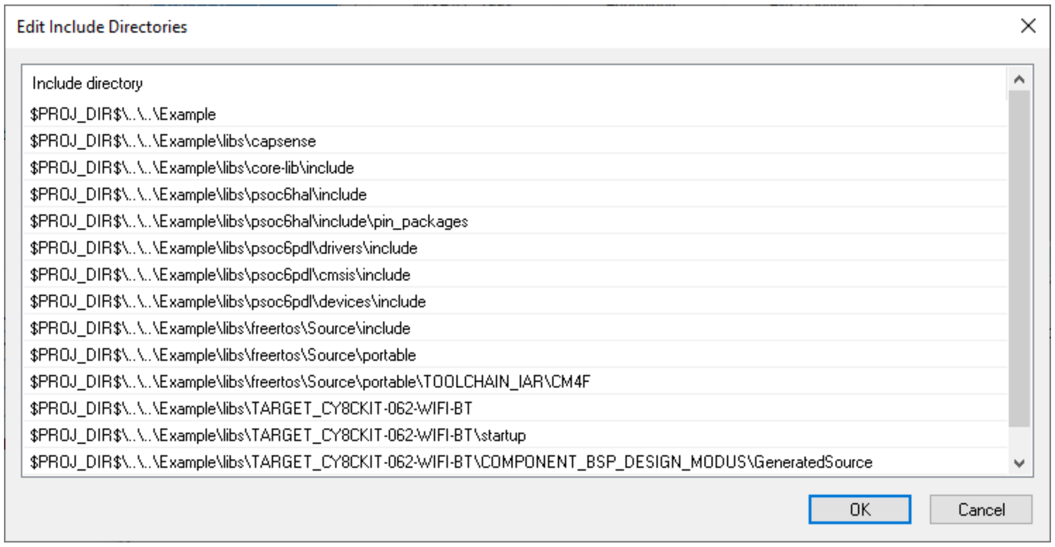
```
AN225588
├ AN225588-BLEKit - Debug ✓
└ AN225588-WiFiKit - Debug ✓
  ├ bsp
  ├ capsense
  │ ├ cy_capsense_centroid.c
  │ ├ cy_capsense_control.c
  │ ├ cy_capsense_csd.c
  │ ├ cy_capsense_csx.c
  │ ├ cy_capsense_filter.c
  │ ├ cy_capsense_processing.c
  │ ├ cy_capsense_sensing.c
  │ ├ cy_capsense_structure.c
  │ ├ cy_capsense_tuner.c
  │ └ libcy_capsense.a
  ├ FreeRTOS
  ├ GeneratedSource
  ├ psco6pdl
```

5.7 Manage Include Paths

As a reminder, this has already been done in the example. All paths are project relative and point to the header files in the project folder This includes paths to the header files for each library, the BSP, the startup code, and the generated source, among others. The *core-lib* headers are used by the *psoc6hal* library.

In the IAR tools, you can add or edit paths in **Project** > **Options** > **C/C++ Compiler**, on the Preprocessor tab.

Figure 11. Application Include Paths



Edit Include Directories

Include directory

```
$PROJ_DIR$\..\..\Example
$PROJ_DIR$\..\..\Example\libs\capsense
$PROJ_DIR$\..\..\Example\libs\core-lib\include
$PROJ_DIR$\..\..\Example\libs\psoc6hal\include
$PROJ_DIR$\..\..\Example\libs\psoc6hal\include\pin_packages
$PROJ_DIR$\..\..\Example\libs\psoc6pdl\drivers\include
$PROJ_DIR$\..\..\Example\libs\psoc6pdl\cmsis\include
$PROJ_DIR$\..\..\Example\libs\psoc6pdl\devices\include
$PROJ_DIR$\..\..\Example\libs\freertos\Source\include
$PROJ_DIR$\..\..\Example\libs\freertos\Source\portable
$PROJ_DIR$\..\..\Example\libs\freertos\Source\portable\TOOLCHAIN_IAR\CM4F
$PROJ_DIR$\..\..\Example\libs\TARGET_CY8CKIT-062-WIFI-BT
$PROJ_DIR$\..\..\Example\libs\TARGET_CY8CKIT-062-WIFI-BT\startup
$PROJ_DIR$\..\..\Example\libs\TARGET_CY8CKIT-062-WIFI-BT\COMPONENT_BSP_DESIGN_MODUS\GeneratedSource
```

5.8 Write Your Software

This of course is the most challenging part of a project. The process of setting up the project to point to and use ModusToolbox software is trivial by comparison. Use the configuration structures and symbols from the generated code, and the APIs for the libraries you use in your project. The example project code is in the *main.c* file. LED behavior is set up in *led_task.c*.

For general information about any given library, or to learn details of its API, refer to the library documentation. In most cases, you can see this directly on GitHub. For example, here is the CapSense API documentation.

Because the Project Creator replicates the entire code repository, the documentation is also available locally in the project folder. Again using the CapSense library as an example, the path to that in the project folder is:

\Example\libs\capsense\docs\ capsense_api_reference_manual.html

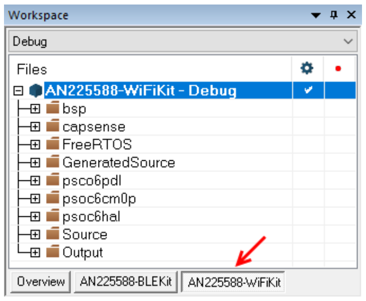
Each library also has a *readme.md* and a *release.md* file, available on GitHub or locally in the project folder. There may be additional documentation available for a library.

5.9 Build, Run, and Debug the Project

This step is optional. You can debug the project if you wish, although that isn't necessary. The goal of this app note is to show you one way to structure a project and how to get files into the IDE.

The example supports two kits. The workspace has a project for each. Pick the project for the kit you use. Each project has a single build target called **Debug**. To build a project, use **Project** > **Make**.

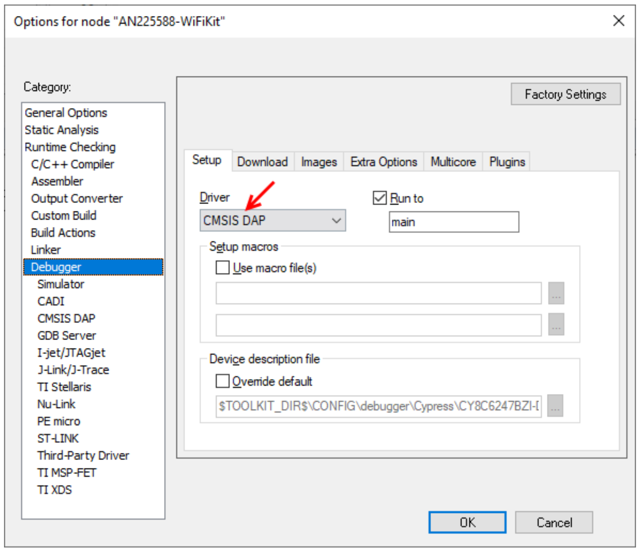
Figure 12. Select the Project for a Kit



CMSIS-DAP support is built into Cypress kits. The project has a debug configuration set up for CMSIS-DAP. If you wish to use another probe, such as I-jet or J-Link, you may need to configure the debug connection to work correctly.

Use **Project** > **Options** > **Debugger**. Confirm that the driver is CMSIS DAP: The CMSIS-DAP settings are already configured correctly.

Figure 13. Use the CMSIS DAP Debug Probe



Driver: CMSIS DAP

☑ Run to: main

☐ Use macro file(s)

☐ Override default

$TOOLKIT_DIR$\CONFIG\debugger\Cypress\CY8C6247BZI-[

Plug in the kit using the USB cable that came with the kit. Observe the amber LED2 and confirm that the kit is in CMSIS-DAP mode. LED behavior varies with different versions of the communication firmware. See CMSIS-DAP Debug Issues in the Troubleshooting appendix for LED behavior. If necessary, press the Mode Select switch on the kit to change mode.

After confirming that the debug connection is in the right mode, use **Project**> **Download and Debug** to begin a debug session.

6 Using ModusToolbox Software in Keil µVision

This section outlines high-level tasks you need to perform to create a PSoC 6 MCU application using the Keil µVision tools and ModusToolbox software. Each task has a description of how that task was implemented in the example. You do not need to perform these tasks; the example is complete.

If you were working from scratch, you would do three things to begin the process:

- Use the Project Creator to create a project folder with all the files you need
- Use the Library Manager (if necessary) to add/remove any library (including a BSP)
- Create a Keil workspace and a C project within that workspace.

The sections that follow describe what to do with that project, and include

- Keil Project Setup
- Add BSP Files to the Project
- Add Generated Code to the Project
- Add Code for a CM0+ Application (example-specific)
- Add the FreeRTOS Fils (example-specific)
- Add Other Libraries
- Manage Include Paths
- Write Your Software
- Build, Run, and Debug the Project

As noted, you do not need to do any of these tasks with the example. Open the Keil workspace to follow along: *AN225588.uvmpw*

6.1 Keil Project Setup

Note: Work is under way so that the ModusToolbox build system will produce a CPDSC (CMSIS Project Description) file you can use to create the Keil project, set most options, and add all project files automatically, making project creation a very friendly process. We will update this app note with the particulars when that feature is available.

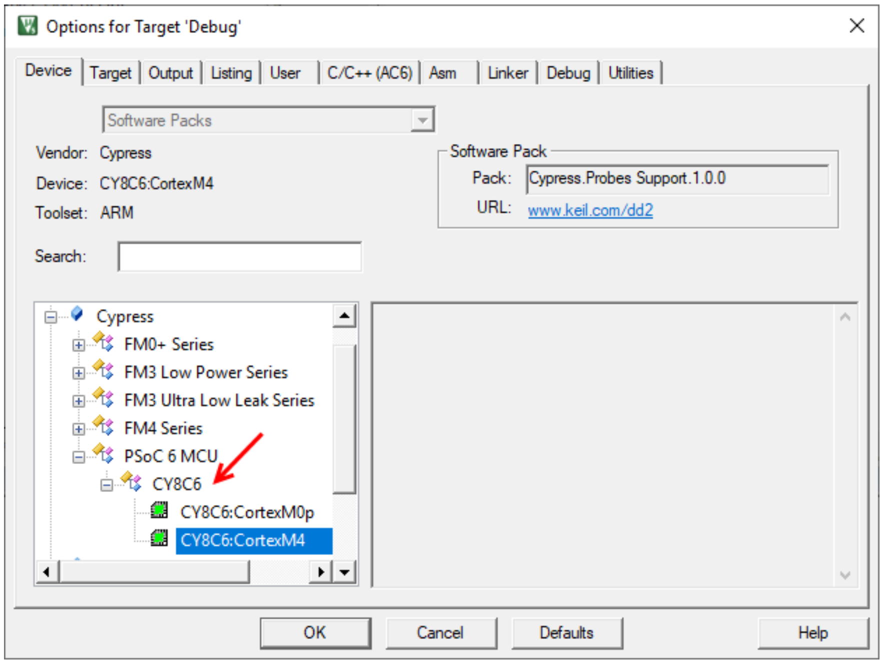
There are some matters you should be aware of when setting up a project in the Keil tools.

Device Support

For Keil tools, a one-time setup step is required to enable support for PSoC 6 MCU devices and debug probes. If you have never used Keil tools for PSoC 6 MCU development, you should install the support pack. It is included in the *AN225588-Keil\SupportFiles\mdk support pack* folder. Just double-click the pack, and the required files are installed in your Keil installation.

After installing, you can then specify the device. Figure 14 shows the Device set up for the WiFi Kit.

Figure 14. Device Setup



Vendor: Cypress
Device: CY8C6:CortexM4
Toolset: ARM
Pack: Cypress.Probes Support.1.0.0
URL: www.keil.com/dd2

Include Paths

Being self-contained, all paths in the example are project-relative. This works well when you use the Project Creator, because all the files are in one location relative to your project file. In your workflow you may prefer to use source files from a development server rather than a local snapshot. If so, you need to manage include paths.

Preprocessor Symbols

In the µVision IDE, preprocessor symbols are set in the **Project** > **Options** > **C/C++** panel. These are already set correctly in the example. A symbol identifies the target device. You can specify the preprocessor symbol manually. The available symbols are listed in the *cy_device_headers.h* file. In the Project Creator folder used in the example, it is here: *\Example\libs\psoc6pdl\devices\include\*

The example also defines DEBUG so that debug information is created.

Linker File Selection

While an IDE may include a database of configuration and setup files, like a linker file, these are provided directly by ModusToolbox software for supported toolchains. It is likely that this linker file is more current than what comes with an IDE.

The linker (scatter) file is specified in the **Project** > **Options** > **Linker** panel. The project points to the correct scatter file in the BSP folder. In the example, the linker file is here:

\Example\libs\TARGET_<KIT> \linker\TOOLCHAIN_ARM\

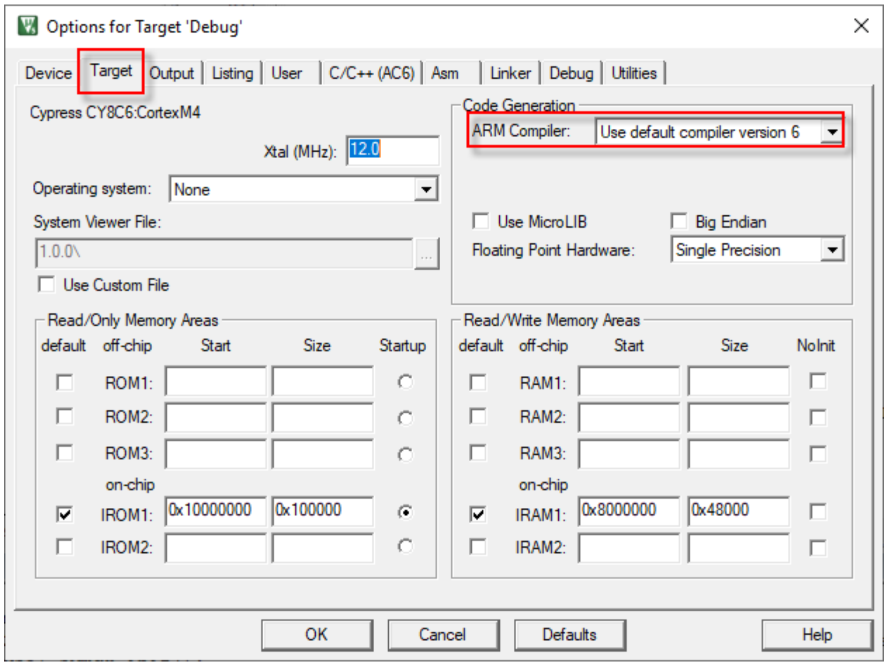
Linker Warnings Suppressed

The linker settings for µVision IDE include this command: `--diag_suppress=L6314W,L6329W`. The scatter file provided with ModusToolbox is generic and handles all common use cases. This project does not use every section defined in the generic linker file. These warnings can be safely ignored and are suppressed in this project.

Compiler Settings

Some libraries include precompiled static libraries for certain functionality. The CapSense library is one example. The µVision IDE supports both V5 and V6 Arm compilers. ModusToolbox software is built using the V6 compiler. Libraries may not link correctly if you use the V5 compiler for your application. The compiler is set in **Project** > **Options** > **Target**. It is already set correctly in the example.

Figure 15. Use the Arm V6 Compiler



Options for Target 'Debug'

Cypress CY8C6:CortexM4

Xtal (MHz): 12.0

Operating system: None

System Viewer File: 1.0.0\

☐ Use Custom File

Code Generation

ARM Compiler: Use default compiler version 6

☐ Use MicroLIB ☐ Big Endian

Floating Point Hardware: Single Precision

Read/Only Memory Areas

default	off-chip	Start	Size	Startup
☐	ROM1:			○
☐	ROM2:			○
☐	ROM3:			○
	on-chip			
✓	IROM1:	0x10000000	0x100000	●
☐	IROM2:			○

Read/Write Memory Areas

default	off-chip	Start	Size	NoInit
☐	RAM1:			☐
☐	RAM2:			☐
☐	RAM3:			☐
	on-chip			
✓	IRAM1:	0x8000000	0x48000	☐
☐	IRAM2:			☐

The default compiler options on the C/C++ (AC6) panel should work. The static libraries are compiled with **short enums/wchar** enabled.

6.2 Add BSP Files to the Project

Remember that all these steps are done in the example. The description tells you about the task and how it is done.

The BSP includes some kit-specific files such as startup code. In the IDE project, the files are in a group called **BSP**. The table lists the files, and their origin in the BSP folder:

Table 4. BSP Files for Keil µVision

File	Note	Folder
cybsp.c	Code to start up the hardware on the kit	*libs\TARGET_<KIT>*
system_psoc6_cm4.c	Clock and system setup	*libs\TARGET_<KIT>\startup*
startup_psoc6_01_cm4.s	IDE-specific startup code	*libs\TARGET_<KIT>\startup\TOOCHAIN_ARM*

The BSP has additional source files to support retargeting standard I/O for the kit. The example application does not use standard I/O, so those files are not added to the IDE project. This highlights a key feature of how ModusToolbox software is structured and illustrates how you can adapt ModusToolbox software to your workflow.

A ModusToolbox BSP has everything required for every common use case. Using standard I/O is a common use case. As a result, the required files are present in the BSP. This gives you a choice.

You can add every source file in the library repository to your project, whether your application needs it or not. You rely on the linker to eliminate any code not used. This approach means that you don't need to know what is implemented where or identify what you do and do not need. Just add everything.

However, your workflow may require that all code has a purpose. In that case, you can remove anything you don't need. In the example we intentionally did not add the standard I/O-related files to illustrate the point.

6.3 Add Generated Code to the Project

The Project Creator automatically executes the design files in the BSP and generates configuration code. The files appear in the *GeneratedSource* folder beside the *design.modus* file. Those files are found here:

Example\libs\TARGET_<KIT>\COMPONENT_BSP_DESIGN_MODUS\GeneratedSource\

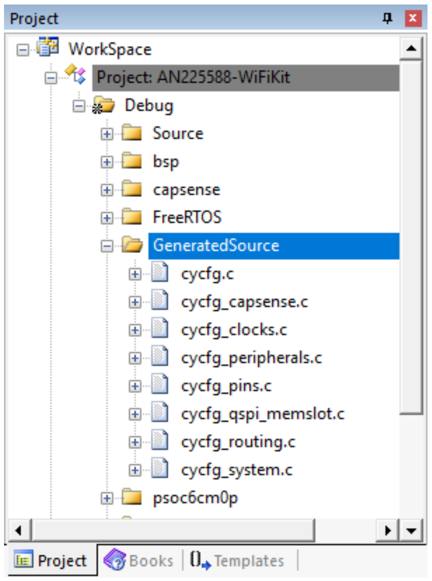
If the *GeneratedSource* folder does not exist (for example, you didn't use the Project Creator), simply use the Device Configurator to open the *design.modus* file and save it again. This generates the code.

To add files, right-click the group, or simply drag and drop the files into the group. Add the files directly from the *GeneratedSource* folder in the Project Creator's project folder. If you modify the design, changed files will be noted in the project automatically. If your design generates new files, add them to the project.

The example has a **GeneratedSource** group (highlighted in the figure). To add files, with the project active right-click the group and **Add Existing Files to Group.** Add the files directly from the *GeneratedSource* folder in the Project Creator's project folder. If you modify the design, changed files will be noted in the project automatically. If your design generates new files, add them to the project.

The actual design configuration files (for example, *design.modus*) are not source files, so have not been added to the IDE project in the example. You can do so if you wish and configure the IDE to open the Device Configurator.

Figure 16. Add Generated Source Files



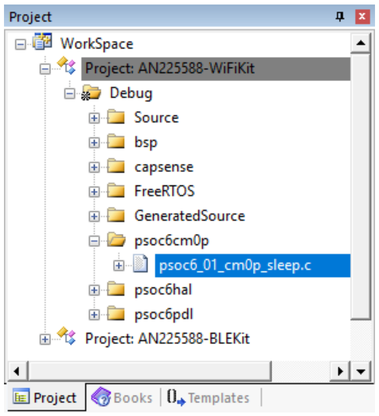
The example application makes no changes to the default design files. Your application might. See Understanding the ModusToolbox BSP and in particular Changing Default BSP Files to understand the risks involved in changing default BSP files like *design.modus*.

6.4 Add Code for a CM0+ Application

This step is necessary for PSoC 6 MCU devices with dual CPUs. Each kit supported by this app note has a PSoC 6 MCU with a dual CPU. The example application enables the Cortex-M4 CPU and puts the Cortex-M0+ CPU (CM0+) to sleep. The CM0+ application for this purpose is found here: *\<ProjectFolder \libs\psoc6cm0p\COMPONENT_CM0P_SLEEP\*

Each application is provided as a source file in the form of a C array that is compiled into the final application.

Figure 17. Add a CM0+ Application



Default CM0+ applications have different memory footprints, so linker sections must be adjusted if you change the CM0+ application. See Dual-CPU Devices for a more detailed discussion of this topic.

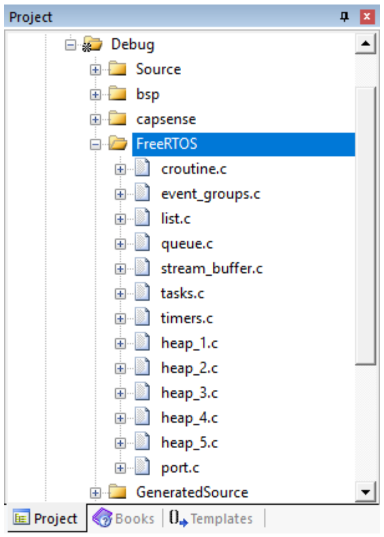
6.5 Add the FreeRTOS Files

This step is example-specific, because it uses FreeRTOS.

The Project Creator puts the files here: *\<ProjectFolder>\libs\freertos\*. The source files exist in several subfolders within this hierarchy, based on the open source project for FreeRTOS. For simplicity, the example project has all these files in a single (flat) group.

FreeRTOS is not a Cypress product. What Cypress provides on GitHub may not be the latest version. What is on the Cypress GitHub site, and what code examples refer to, is the most recent version tested with Cypress devices and firmware. You may substitute any version of the RTOS that fits your development requirements, but keep in mind that it may not have been tested with ModusToolbox software.

Figure 18. Add FreeRTOS Files



6.6 Add Other Libraries

If you use the Project Creator, every library needed by the application is here: *\<ProjectFolder>\libs\<name>*. The libraries used include:

- capsense
- core-lib
- psoc6hal

- psoc6pdl
- TARGET_<KITNAME> (in the BSP group in the IDE project file) You can organize them however you wish.

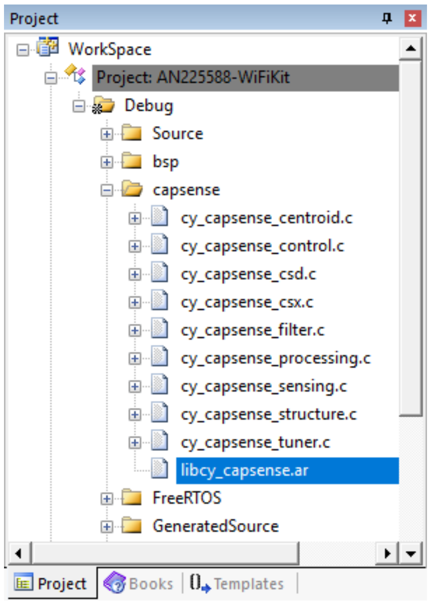
Add source files from each library to the project. You can organize them however you wish. The example sets up a group for each library and adds the files to that group. The typical workflow is to use the entire library.

Middleware may provide a static (precompiled) library for some functionality. If there is a static library, it is in a folder that identifies the toolchain for which it was compiled. The CapSense library is a good example. It has static libraries for hardware and software floating point support. The project uses software floating point. For the Keil tools the static library is located here.

\Example\libs\capsense\COMPONENT_SOFTFP\TOOLCHAIN_ARM\

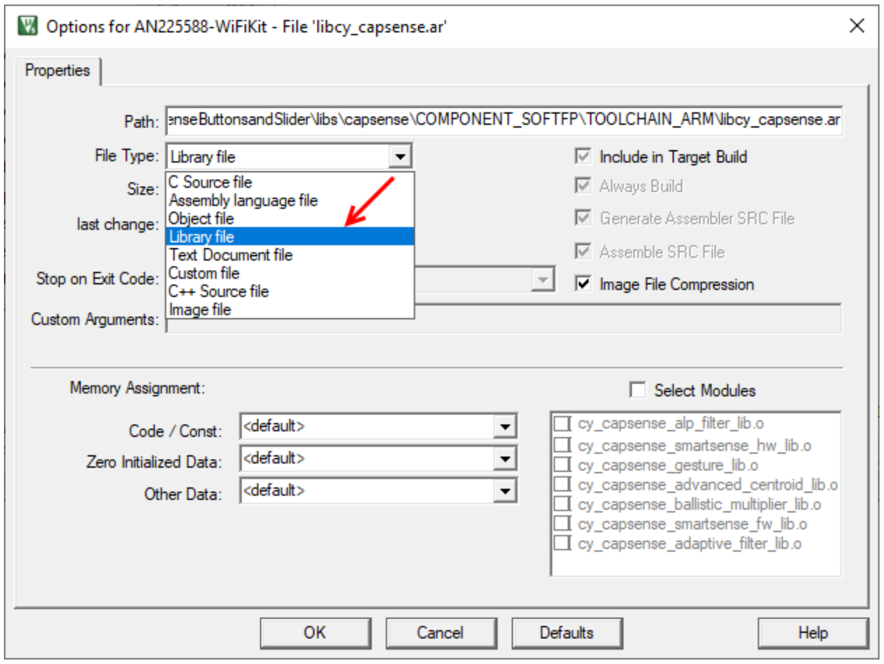
There is a similar folder for each supported toolchain. When adding the files for any given library, look for *TOOLCHAIN* folders and add any required static library.

Figure 19. Add Source and Static Library Files



The Keil tools assume that the .ar file is an assembly file. It is not. In the project explorer, right-click the file, choose **Options**, and identify the file as a library file.

Figure 20. Specifying a Library File


Options for AN225588-WiFiKit - File 'libcy_capsense.ar'

Properties

Path: ənseButtonsandSlider\libs\capsense\COMPONENT_SOFTFP\TOOLCHAIN_ARM\libcy_capsense.ar

File Type: Library file

Size:

last change:

Stop on Exit Code:

Custom Arguments:

C Source file
Assembly language file
Object file
Library file
Text Document file
Custom file
C++ Source file
Image file

☑ Include in Target Build
☑ Always Build
☑ Generate Assembler SRC File
☑ Assemble SRC File
☑ Image File Compression

Memory Assignment:

Code / Const: <default>
Zero Initialized Data: <default>
Other Data: <default>

☐ Select Modules

☐ cy_capsense_alp_filter_lib.o
☐ cy_capsense_smartsense_hw_lib.o
☐ cy_capsense_gesture_lib.o
☐ cy_capsense_advanced_centroid_lib.o
☐ cy_capsense_ballistic_multiplier_lib.o
☐ cy_capsense_smartsense_fw_lib.o
☐ cy_capsense_adaptive_filter_lib.o

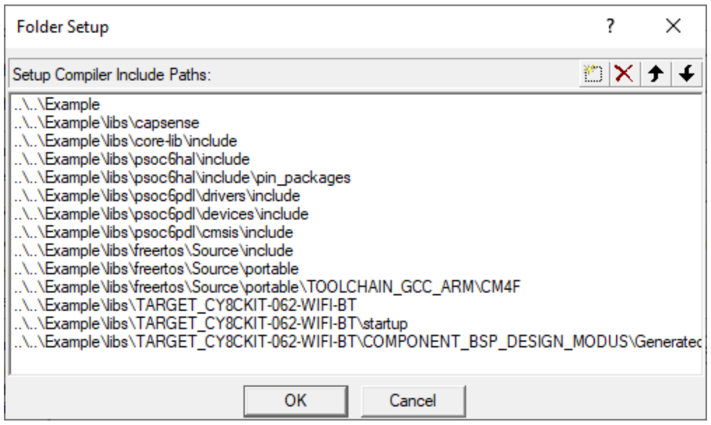
OK | Cancel | Defaults | Help

6.7 Manage Include Paths

As a reminder, this has already been done in the example. All paths are project relative and point to the header files in the *Example* folder. This includes paths to the header files for each library, the BSP, the startup code, and the generated source, among others. The *core-lib* headers are used by the *psoc6hal* library.

In the Keil tools, you can add or edit paths in **Project** > **Options for <name>** > **C/C++ (AC6)** panel.

Figure 21. Application Include Paths


Folder Setup

Setup Compiler Include Paths:

```
..\..\Example
..\..\Example\libs\capsense
..\..\Example\libs\core-lib\include
..\..\Example\libs\psoc6hal\include
..\..\Example\libs\psoc6hal\include\pin_packages
..\..\Example\libs\psoc6pdl\drivers\include
..\..\Example\libs\psoc6pdl\devices\include
..\..\Example\libs\psoc6pdl\cmsis\include
..\..\Example\libs\freertos\Source\include
..\..\Example\libs\freertos\Source\portable
..\..\Example\libs\freertos\Source\portable\TOOLCHAIN_GCC_ARM\CM4F
..\..\Example\libs\TARGET_CY8CKIT-062-WIFI-BT
..\..\Example\libs\TARGET_CY8CKIT-062-WIFI-BT\startup
..\..\Example\libs\TARGET_CY8CKIT-062-WIFI-BT\COMPONENT_BSP_DESIGN_MODUS\Generatec
```

OK | Cancel

6.8 Write Your Software

This of course is the most challenging part of a project. The process of setting up the project to point to and use ModusToolbox software is trivial by comparison. Use the configuration structures and symbols from the generated code, and the APIs for the libraries you use in your project. The example project code is in the *main.c* file. LED behavior is set up in *led_task.c*.

For general information about any given library, or to learn details of its API, refer to the library documentation. In most cases you can see this directly on GitHub. For example, here is the CapSense API documentation.

Because the Project Creator replicates the entire code repository, the documentation is also available locally in the project folder. Again using the CapSense library as an example, the path to that in the project folder is:

\Example\libs\capsense\docs\ capsense_api_reference_manual.html

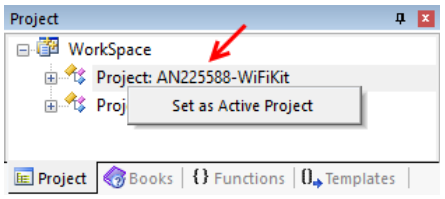
Each library also has a *readme.md* and a *release.md* file, available on GitHub or locally in the project folder. There may be additional documentation available for a library.

6.9 Build, Run, and Debug the Project

This step is optional. You can debug the project if you wish, although that isn't necessary. The goal of this app note is to show you one way to structure a project and how to get files into the IDE.

The example supports two kits. The workspace has a project for each. Right-click the project for the kit you want to use, and then choose **Set as Active Project**.

Figure 22. Select the Project for a Kit

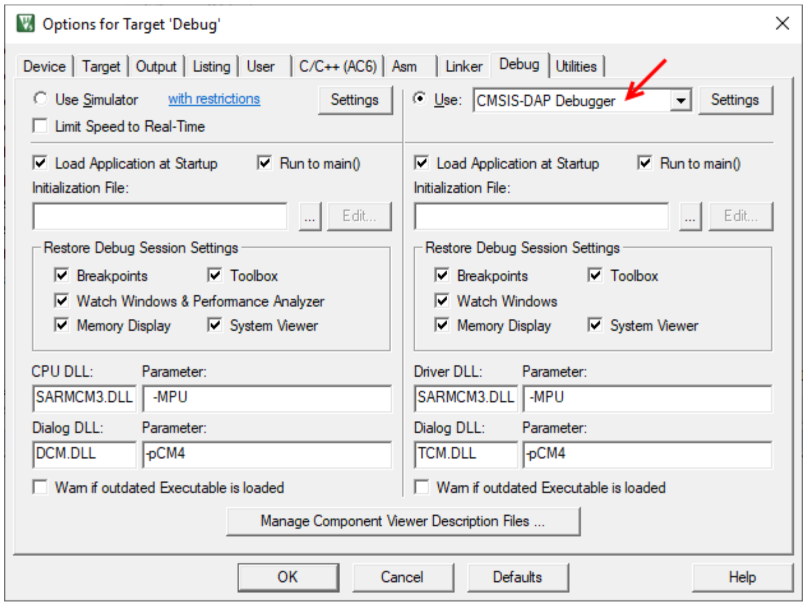


Each project has a single build target called **Debug**. To build a project, use **Project** > **Build**.

CMSIS-DAP support is built into Cypress kits. The project has a debug configuration set up for CMSIS-DAP. If you wish to use another probe, such as a uLink or J-Link, you may need to configure the debug connection to work correctly.

Use **Project** > **Options** > **Debug**. Confirm that the project is using the CMSIS-DAP debugger. The CMSIS-DAP settings are already configured correctly.

Figure 23. Use the CMSIS DAP Debug Probe


Options for Target 'Debug'
Device | Target | Output | Listing | User | C/C++ (AC6) | Asm | Linker | Debug | Utilities
Settings
Edit...
Manage Component Viewer Description Files ...
OK
Cancel
Defaults
Help

Use Simulator with restrictions [ ] — [✓ Use: CMSIS-DAP Debugger]
[ ] Limit Speed to Real-Time

	Simulator	CMSIS-DAP Debugger
Load Application at Startup	✓	✓
Run to main()	✓	✓
Initialization File		
Restore Debug Session Settings: Breakpoints	✓	✓
Toolbox	✓	✓
Watch Windows & Performance Analyzer / Watch Windows	✓	✓
Memory Display	✓	✓
System Viewer	✓	✓
CPU DLL / Driver DLL	SARMCM3.DLL	SARMCM3.DLL
Parameter	-MPU	-MPU
Dialog DLL	DCM.DLL	TCM.DLL
Parameter	-pCM4	-pCM4
Warn if outdated Executable is loaded		

Plug in the kit using the USB cable that came with the kit. Observe the amber LED2 and confirm that the kit is in CMSIS-DAP mode. LED behavior varies with different versions of the communication firmware. See CMSIS-DAP Debug Issues in the Troubleshooting appendix for LED behavior. If necessary, press the Mode Select switch on the kit to change mode.

After confirming that the debug connection is in the right mode, **Debug** > **Start/Stop Debug Session** downloads the executable and begins a debug session.

7 Dual-CPU Devices

Many PSoC 6 devices have both an Arm Cortex-M4 (CM4) and a Cortex-M0+ (CM0+) CPU. Cypress provides default CM0+ applications that implement certain functionality on the CM0+ CPU. When you use the Project Creator, you can find all the applications here:

\Example \libs\psoc6cm0p\

The three available applications and a brief description are in the table. Each application enables the CM4 CPU, where the user application runs. Each has a *readme.md* file that describes the application in more detail.

Table 5. Default Applications for Cortex-M0+

CM0+ Application	Description
COMPONENT_CM0P_BLESS	This image is used only in BLE dual-CPU mode. In this mode, BLE functionality is split between CM0+ (controller) and CM4 (host). The application sets up the BLE Controller. It uses inter-processor communication (IPC) between two CPUs where the controller and host run.
COMPONENT_CM0P_CRYPTO	This application sets up the crypto server on the CM0+ core so an application can use the cryptographic capabilities of the Peripheral Driver Library. The application configures an IPC channel for data exchange between client and server applications. There is a different version of this application for different PSoC 6 MCU devices.
COMPONENT_CM0P_SLEEP	After enabling the CM4 CPU, this application simply puts the CM0+ CPU to deep sleep. There is a different version of this application for different PSoC 6 MCU devices.

For a dual-CPU PSoC 6 MCU device, you must provide one of these CM0+ applications.

To use any of these applications, add the associated .c file to your project. It is provided as a C array to be compiled into the final application. Each has a different memory footprint. Information about how to manage the linker file to ensure that space is provided for the CM0+ application is in each *readme.md*.

8 Summary

ModusToolbox software is a comprehensive set of modular libraries available at the Cypress GitHub site. The challenge is in knowing not only which library you need, but which version of each library to use.

The ModusToolbox build system and code examples solve that problem. Use the Project Creator to specify your kit, and an application like what you need. It does all the work of gathering every necessary file from every library into a single project folder. This includes configuration design files that generate required configuration code. You can use the Library Manager to change the BSP, or add, remove, or change the version of any library.

Every file is provided locally in your project folder, so you can modify any file you want. This includes the default BSP configuration, the startup code, linker command files, and so on. To eliminate any risk of losing such changes, you can provide a custom BSP to override the default BSP.

To use this software in your IDE, just add the files to your project. In many cases there are IDE-specific files (such as precompiled libraries, or startup code). Every required file for every supported toolchain is available in the Project Creator's project folder.

There will be IDE-specific issues unrelated to ModusToolbox software. Some are obvious, like configuring the project for your preferred debug probe. Using the example project as a model, you can successfully develop firmware for PSoC 6 MCU using ModusToolbox software.

9 Related Documents

For a comprehensive list of PSoC 6 MCU resources, see KBA223067 in the Cypress community.

Application Notes	
AN228571 – Getting Started with PSoC 6 MCU on ModusToolbox	Explores the PSoC 6 MCU architecture and development tools and shows you how to create your first project using the ModusToolbox IDE.
AN219528 – PSoC 6 MCU Low-Power Modes and Power Reduction Techniques	Describes how to use the PSoC 6 MCU power modes to optimize power consumption.
AN225270 – CYW208xx BLE Low-Power Guidelines	Describes how to use CYW208xx power modes to optimize power consumption. Major topics include the low-power modes in these devices, and power management techniques using those modes.
AN227069 – Getting Started with Bluetooth Mesh	Introduces you to Cypress' Bluetooth SIG-compliant Mesh solution. This application note covers Cypress' BLE Mesh architecture, code structure and how to make changes based on your application needs.
AN218241 – PSoC 6 MCU Hardware Design Considerations	How to design a hardware system around a PSoC 6 MCU device, including package selection, power, clocking, reset, I/O usage, and more
Getting Started with PSoC 6 MCU and CYW43xxx in Mbed OS	This article helps you get started on using PSoC 6 MCU and the CYW43xxx WLAN/Bluetooth combo device for developing applications in Mbed OS.

Code Examples
Visit the Cypress GitHub site for a comprehensive collection of code examples using ModusToolbox IDE

Other Resources
Datasheets
Technical Reference Manuals
Development Kits

Tools and Documentation	
ModusToolbox	The tools and software you need for converged MCU and wireless systems
ModusToolbox Software Overview	A high-level look at the tools and software available in ModusToolbox
ModusToolbox Project Creator Guide	Details on the Project Creator and how to use it
ModusToolbox Library Manager Guide	Details on the Library Manager and how to use it

About the Author

Name:	Jim Trudeau
Title:	Senior Principal Applications Engineer

Appendix A. Troubleshooting

This appendix covers issues you may encounter if you debug the example application.

A.1 CMSIS-DAP Debug Issues

When I start a CMSIS-DAP debug session an error occurs. IAR: **Probe not found**; *Keil:* **No Debug Unit Found**.

Each tool requires a CMSIS-DAP connection. You need to switch the kit to the mode required by the tool.

CMSIS-DAP debugging uses the on-board debug communication firmware built into every Cypress kit. This firmware is called KitProg. There are two different implementations of KitProg, KitProg2 and KitProg3. Each has different modes of operation. The **SW3 Mode Select** button, near the amber LED2 changes modes. LED2 tells you what mode you are in. The LED behaves differently as well.

If your kit has KitProg2 installed, LED2 must be OFF.

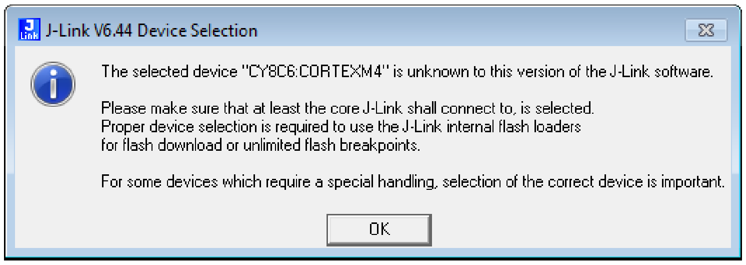
If your kit has KitProg3 installed, LED2 must be ON, or breathing at 1HZ.

The kits supported in this app note come from the factory with KitProg2 installed. You can get the latest version of KitProg on GitHub. The Firmware Loader tool connects to a kit and allows the firmware to be switched between KitProg2 and KitProg3.

A.2 Segger J-LINK Debug Issues

How do I configure J-LINK for debugging with the Keil tools?

The first time you select the probe you may see an alert like this. This is normal J-LINK behavior. Click **OK** to specify the device. For CM4, select the Cypress CY8C6xx7_CM4.


J-Link V6.44 Device Selection

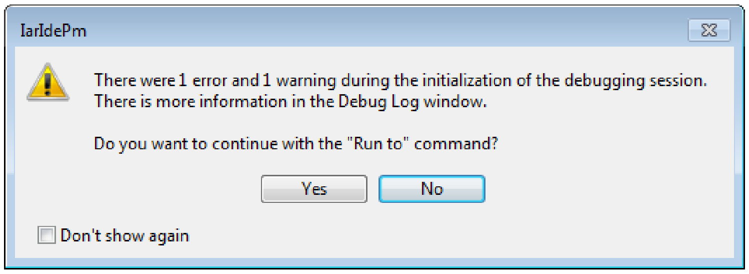
The selected device "CY8C6:CORTEXM4" is unknown to this version of the J-Link software.

Please make sure that at least the core J-Link shall connect to, is selected.
Proper device selection is required to use the J-Link internal flash loaders
for flash download or unlimited flash breakpoints.

For some devices which require a special handling, selection of the correct device is important.

With IAR tools, when I start a debug session using J-LINK I get an error.

Click **Yes** and ignore the error. Debugging works despite this error. The issue has been reported.


IarIdePm

There were 1 error and 1 warning during the initialization of the debugging session.
There is more information in the Debug Log window.

Do you want to continue with the "Run to" command?

Document History

Document Title: AN225588 - Using ModusToolbox Software with a Third-Party IDE

Document Number: 002-25588

Revision	ECN	Date	Description of Change
**	6532139	04/03/2019	New Application Note, applies to ModusToolbox 1.1 only
*A	6725872	02/13/2020	Major update for ModusToolbox 2.0
*B	6841131	03/27/2020	Add note, for ModusToolbox 2.0 only, provide pointer to the 2.1 documentation

Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturer's representatives, and distributors. To find the office closest to you, visit us at Cypress Locations.

Products

Arm® Cortex® Microcontrollers	cypress.com/arm
Automotive	cypress.com/automotive
Clocks & Buffers	cypress.com/clocks
Interface	cypress.com/interface
Internet of Things	cypress.com/iot
Memory	cypress.com/memory
Microcontrollers	cypress.com/mcu
PSoC	cypress.com/psoc
Power Management ICs	cypress.com/pmic
Touch Sensing	cypress.com/touch
USB Controllers	cypress.com/usb
Wireless Connectivity	cypress.com/wireless

PSoC® Solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP | PSoC 6 MCU

Cypress Developer Community

Community | Code Examples | Projects | Videos | Blogs | Training | Components

Technical Support

cypress.com/support

All other trademarks or registered trademarks referenced herein are the property of their respective owners.

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709