

Chapter 5: PSoC 6 SDK & Reference Flow

After completing this chapter, you will understand what the PSoC SDK is, what tools are included, and how to use those tools to compile and run programs on your device.

| | |
|--|-----------|
| 5.1 PSOC 6 SDK TOUR | 2 |
| 5.1.1 WEB | 3 |
| 5.1.2 ECLIPSE-BASED IDE | 6 |
| 5.1.3 TOOLS | 10 |
| 5.1.4 COMMAND LINE | 11 |
| 5.2 DEMO WALKTHROUGH..... | 13 |
| 5.2.1 BLINKING LED FROM THE ECLIPSE IDE..... | 13 |
| 5.2.2 BLINKING LED FROM COMMAND LINE INTERFACE..... | 13 |
| 5.3 EXERCISES (PART 1) | 14 |
| 5.3.1 BLINKING LED FROM THE ECLIPSE IDE..... | 14 |
| 5.3.2 BLINKING LED FROM THE COMMAND LINE | 17 |
| 5.4 CREATE MODUSTOOLBOX PROJECTS | 18 |
| 5.4.1 PROJECT CREATOR TOOL | 18 |
| 5.4.2 COMMAND LINE / MANUAL SETUP..... | 20 |
| 5.5 IMPORT PROJECTS | 22 |
| 5.5.1 IMPORT PROJECTS INTO ECLIPSE IDE..... | 22 |
| 5.5.2 IMPORT PROJECTS INTO PROJECT CREATOR TOOL | 23 |
| 5.6 EXPORTING MODUSTOOLBOX PROJECTS | 24 |
| 5.6.1 EXPORT FOR VISUAL STUDIO (VS) CODE..... | 24 |
| 5.6.2 EXPORT FOR OTHER 3 RD PARTY IDES | 29 |
| 5.7 PROJECT DIRECTORY ORGANIZATION | 29 |
| 5.7.1 ARCHIVES | 29 |
| 5.7.2 INCLUDES | 29 |
| 5.7.3 BUILD | 29 |
| 5.7.4 IMAGES..... | 29 |
| 5.7.5 LIBS | 29 |
| 5.7.6 MAIN.C | 29 |
| 5.7.7 LICENSE..... | 29 |
| 5.7.8 MAKEFILE | 30 |
| 5.7.9 MAKEFILE.INIT..... | 30 |
| 5.7.10 README.MD | 31 |
| 5.8 LIBRARIES & LIBRARY MANAGER..... | 31 |
| 5.8.1 LIBRARY MANAGER | 32 |
| 5.9 BOARD SUPPORT PACKAGES | 33 |
| 5.9.1 BSP DIRECTORY STRUCTURE..... | 33 |
| 5.9.2 BSP DOCUMENTATION..... | 35 |
| 5.9.3 CHANGING THE BSP WITH THE LIBRARY MANAGER..... | 36 |
| 5.9.4 SELECTING A BSP BY EDITING THE MAKEFILE | 36 |
| 5.9.5 MODIFYING THE BSP CONFIGURATION (E.G. DESIGN.MODUS) FOR A SINGLE APPLICATION | 37 |
| 5.9.6 CREATING YOUR OWN BSP | 38 |
| 5.10 HAL..... | 39 |
| 5.11 PDL | 40 |
| 5.12 MANIFESTS | 41 |
| 5.13 CYPRESS CONFIGURATORS..... | 42 |
| 5.13.1 BSP CONFIGURATORS | 43 |
| 5.13.2 LIBRARY CONFIGURATORS | 48 |

| | |
|--|-----------|
| 5.14 EXERCISES (PART 2) | 49 |
| 5.14.1 MODIFY THE BLINKING LED TO USE THE GREEN LED | 49 |
| 5.14.2 MODIFY THE PROJECT TO BLINK GREEN/RED USING THE HAL | 50 |
| 5.14.3 PRINT A MESSAGE USING THE RETARGET-IO LIBRARY..... | 50 |
| 5.14.4 CONTROL THE RGB LED WITH A LIBRARY..... | 52 |
| 5.14.5 CAPSENSE BUTTONS AND SLIDER | 53 |
| 5.14.6 WRITE A MESSAGE TO THE TFT SHIELD..... | 54 |
| 5.14.7 USE AN RTOS TO BLINK LEDs | 56 |
| 5.14.8 USE THE IoT EXPERT FREERTOS TEMPLATE | 57 |
| 5.14.9 CREATE AND USE A LIBRARY | 58 |
| 5.14.10 CREATE AND USE A NEW BSP..... | 63 |

5.1 PSoC 6 SDK Tour

The PSoC 6 SDK & Reference flow contains a “classic” MCU development flow and it includes:

- The Eclipse IDE for ModusToolbox
- Hardware Abstraction Layer (HAL)
- Board Support Packages (BSPs) for Cypress kits
- Peripheral Driver Library (PDL)
- Libraries (Retarget-IO, emWin etc.)
- Programming and debug tools

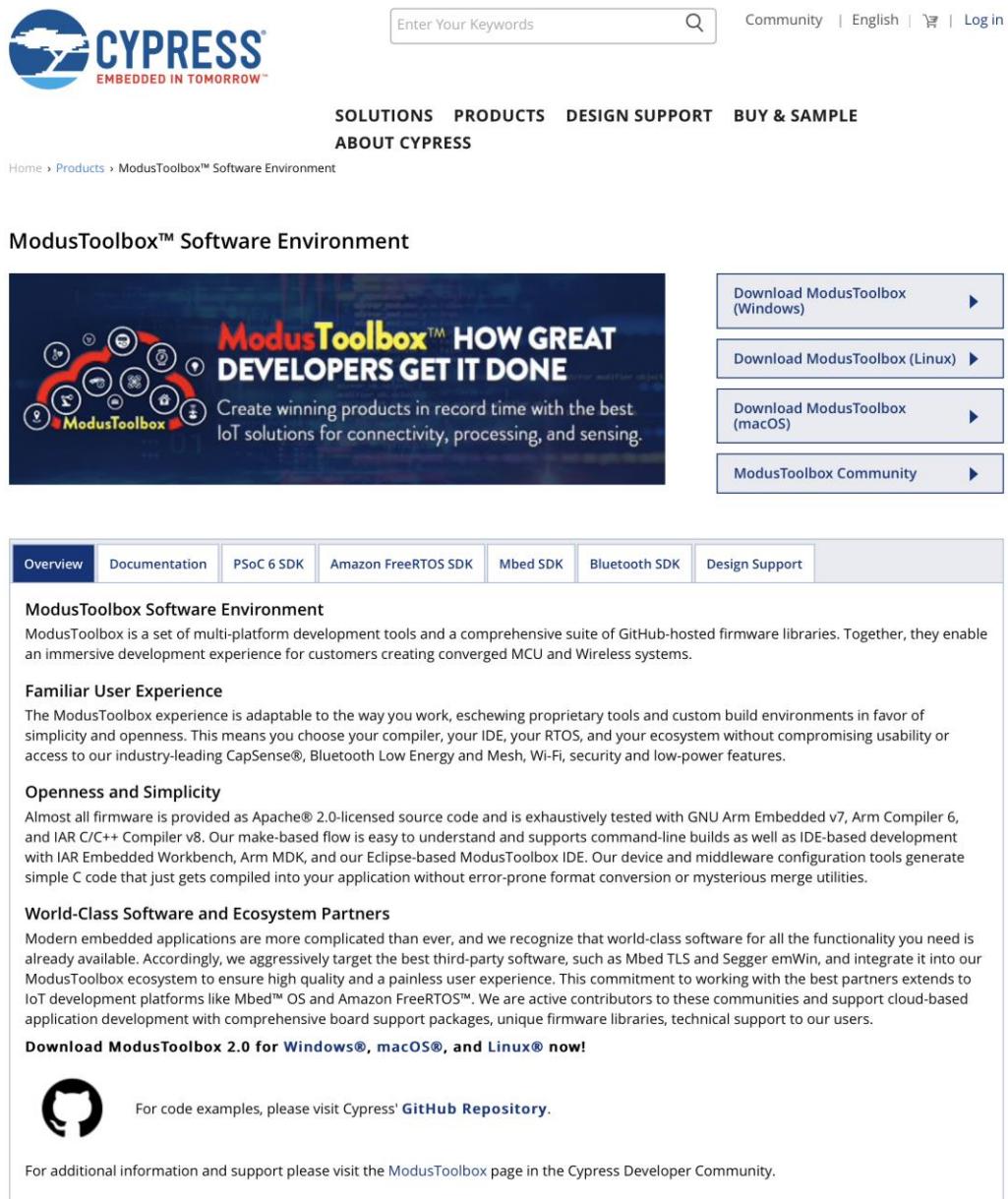
This section provides a brief tour of the ModusToolbox websites, Eclipse IDE, and various tools.

5.1.1 Web

Cypress.com

On the Cypress website, you can download the software, view the documentation, and access the SDKs:

<https://www.cypress.com/products/modustoolbox-software-environment>



The screenshot shows the Cypress website's navigation bar at the top, featuring the Cypress logo, a search bar, and links for Community, English, Log in, and various product categories like SOLUTIONS, PRODUCTS, DESIGN SUPPORT, BUY & SAMPLE, and ABOUT CYPRESS.

The main content area is titled "ModusToolbox™ Software Environment". It features a large banner with the text "ModusToolbox™ HOW GREAT DEVELOPERS GET IT DONE" and a subtext "Create winning products in record time with the best IoT solutions for connectivity, processing, and sensing." Below the banner are four download links: "Download ModusToolbox (Windows)", "Download ModusToolbox (Linux)", "Download ModusToolbox (macOS)", and "ModusToolbox Community".

A navigation menu below the banner includes "Overview", "Documentation", "PSoC 6 SDK", "Amazon FreeRTOS SDK", "Mbed SDK", "Bluetooth SDK", and "Design Support".

The "Overview" section contains several paragraphs about the ModusToolbox ecosystem, highlighting its multi-platform tools, GitHub-hosted libraries, and commitment to simplicity and openness. It also mentions the use of Apache 2.0-licensed source code and various compilers.

The "Documentation" section links to the PSoC 6 SDK, Amazon FreeRTOS SDK, Mbed SDK, Bluetooth SDK, and Design Support pages.

The "Design Support" section links to the ModusToolbox Community page.

At the bottom of the page, there is a GitHub icon and a link to the "GitHub Repository" for code examples.

Community

On the ModusToolbox Community website, you can interact with other developers and access various Knowledge Base Articles (KBAs):

<https://community.cypress.com/community/modustoolbox>



The screenshot shows the Cypress Developer Community 3.0 interface. At the top, there's a navigation bar with links for HOME, CONTENT, PEOPLE, PLACES, and COMMUNITY INFORMATION. On the right, there are Log in, SEARCH (with 'Cypress.com'), and a magnifying glass icon. Below the navigation is a dark blue header bar with the text 'WELCOME TO THE CYPRESS DEVELOPER COMMUNITY 3.0'. The main content area features a large image of the ModusToolbox logo with the text 'ModusToolbox HOW GREAT DEVELOPERS GET IT DONE'. To the left, there's a sidebar with sections for 'SEARCH THIS COMMUNITY' (with a search bar and 'Search' button), 'MODUSTOOLBOX SDKS' (listing BT, PSoC 6, Mbed, and Amazon FreeRTOS SDKs), and 'KEY DOCUMENTS' (links to Installation Guide and Release Notes). The main content area also includes a brief description of the ModusToolbox, a call-to-action for downloading the software, and two recent blog posts.

WELCOME TO THE CYPRESS DEVELOPER COMMUNITY 3.0

HOME CONTENT PEOPLE PLACES COMMUNITY INFORMATION

SEARCH Cypress.com

Log in

ModusToolbox

Overview Content People Subspaces Actions ▾ About Share

Log in to follow, share, and participate in this community.

SEARCH THIS COMMUNITY

Search

Search

MODUSTOOLBOX SDKS

- ModusToolbox BT SDK
- ModusToolbox PSoC 6 SDK
- ModusToolbox Mbed SDK
- ModusToolbox Amazon FreeRTOS SDK

KEY DOCUMENTS

- Installation Guide
- Release Notes

ModusToolbox™ HOW GREAT DEVELOPERS GET IT DONE

ModusToolbox is a set of tools and software, distributed in Software Development Kits (SDK), that enable an immersive development experience for customers creating converged MCU and Wireless systems. To achieve this goal ModusToolbox leverages popular third-party ecosystems, such as Arm Mbed, and adds the Cypress "je ne sais quoi" of Wi-Fi, Bluetooth®, CapSense®, security, industry-leading low power and rapid peripheral configuration in easy to use form.

SDks include all the device and board firmware along with the RTOS, stacks and middleware needed for applications running on PSoC 6 MCUs, PSoC 6 coupled with Wi-Fi/Bluetooth combo devices, and Bluetooth SoCs. Tools such as ModusToolbox IDE, an Eclipse-based Integrated Development Environment (IDE) optimized for Cypress devices, and our intuitive graphical configurators enable an IoT designer to develop innovative IoT applications efficiently and with ease.

Download ModusToolbox 2.0 for Windows®, macOS®, and Linux® now!

[ModusToolbox PSoC 6 Code Example Repository on GitHub](#)

RECENT BLOG POSTS

 **ModusToolbox 2.0 Available Now!**
4 days ago by MarkS_11

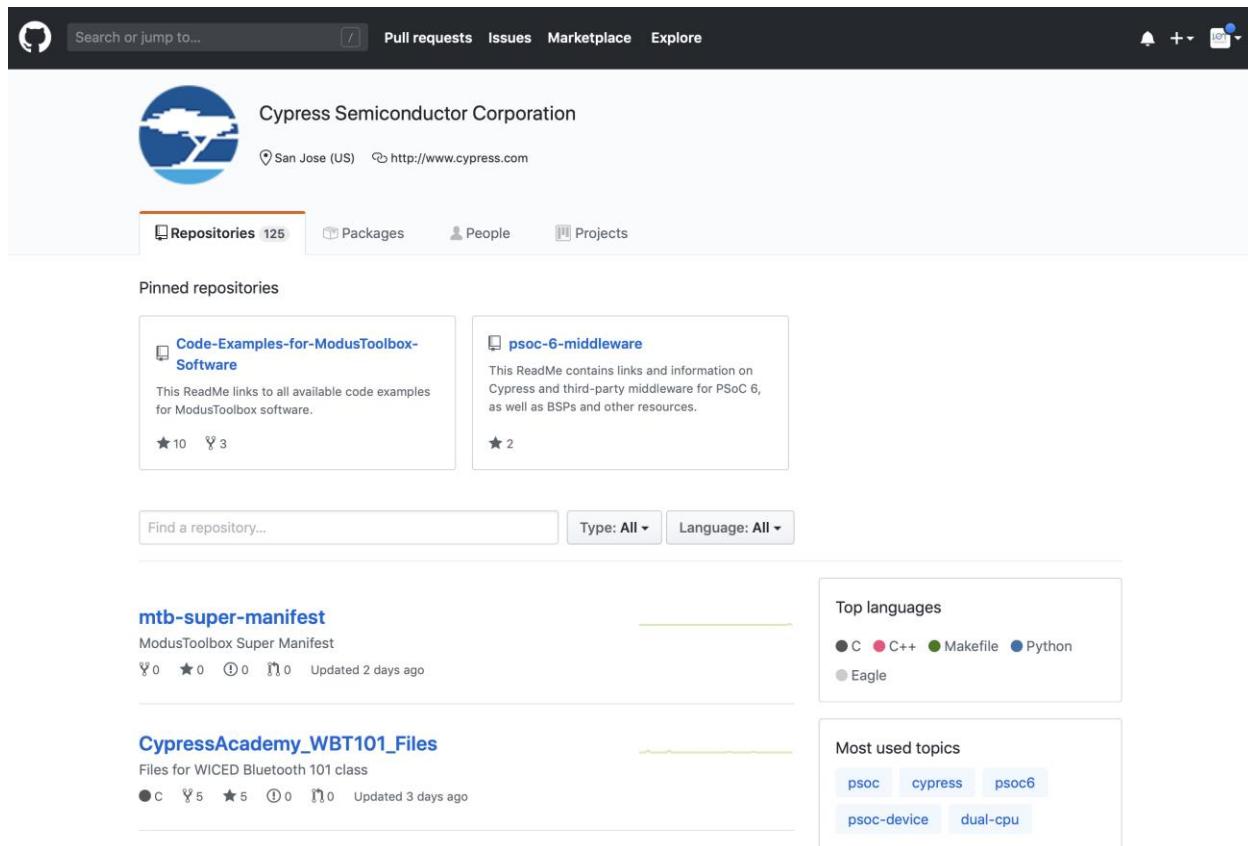
 **Digi-Key - Audio Sampling at 44.1kHz with PSoC 6**
1 month ago by MarkS_11

Feedback

Github.com

The Cypress GitHub website contains all the BSPs, code examples, and libraries for use with various ModusToolbox tools.

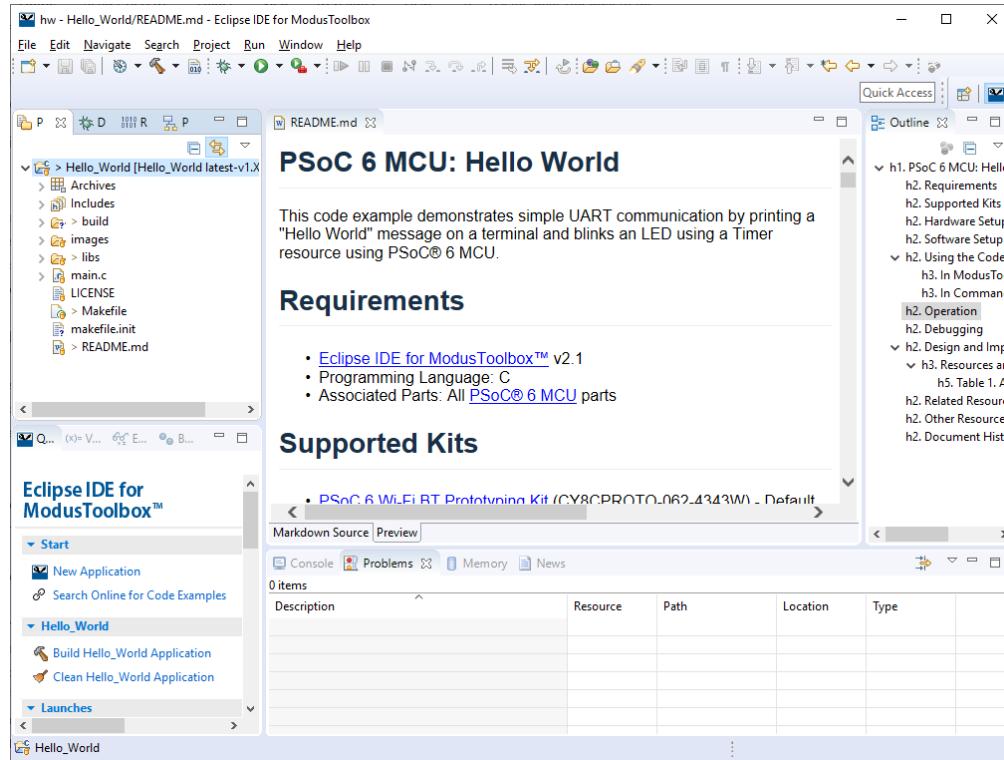
<https://github.com/cypresssemiconductorco>



The screenshot shows the GitHub organization page for Cypress Semiconductor Corporation. At the top, there's a navigation bar with a search bar, pull requests, issues, marketplace, and explore links. Below the header, the organization's logo and name are displayed, along with its location (San Jose, US) and website link. A navigation bar below the header includes repositories (125), packages, people, and projects. The main content area features a "Pinned repositories" section with two items: "Code-Examples-for-ModusToolbox-Software" and "psoc-6-middleware". Each pinned item has a thumbnail, title, description, and statistics (stars and forks). Below this, there's a search bar, type and language filters, and two repository cards: "mtb-super-manifest" and "CypressAcademy_WBT101_Files". To the right, there are two boxes: "Top languages" (C, C++, Makefile, Python, Eagle) and "Most used topics" (psoc, cypress, psoc6, psoc-device, dual-cpu).

5.1.2 Eclipse-Based IDE

The Eclipse IDE for ModusToolbox uses several plugins, including the Eclipse C/C++ Development Tools (CDT) plugin. The IDE contains Eclipse standard menus and toolbars, plus various views such as the Project Explorer, Code Editor, and Console.



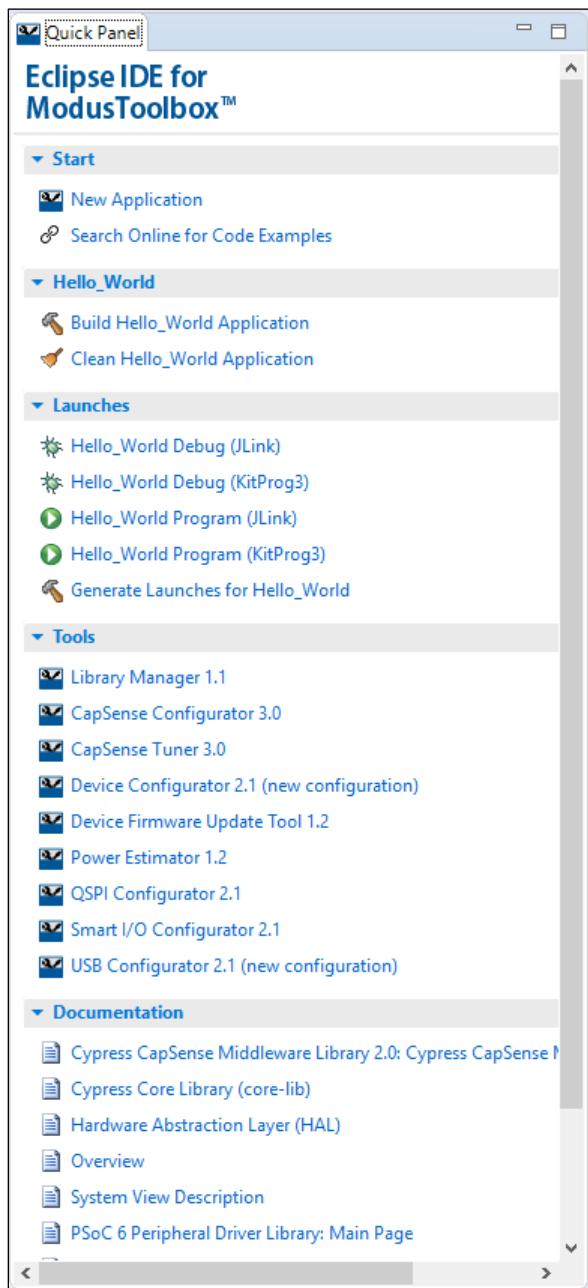
For more information about the IDE, refer to the “Eclipse IDE for ModusToolbox User Guide”:

<http://www.cypress.com/MTBEclipseIDEUserGuide>

Quick Panel

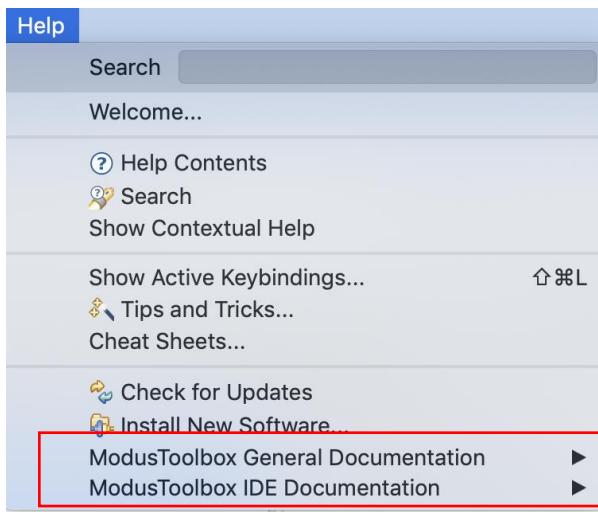
Cypress has extended the Eclipse functionality with a "ModusToolbox" Perspective. Among other things (such as adding a bunch of useful Debug windows), this perspective includes a panel with links to commonly used functions including:

- Create a New Application
- Clean & Build
- Program/Debug Launches
- Tools (Configurators)
- Documentation



Help Menu

The IDE Help menu provides links to general documentation, such as the [ModusToolbox User Guide](#) and [ModusToolbox Release Notes](#), as well as IDE-specific documentation, such as the [Eclipse IDE for ModusToolbox Quick Start Guide](#).



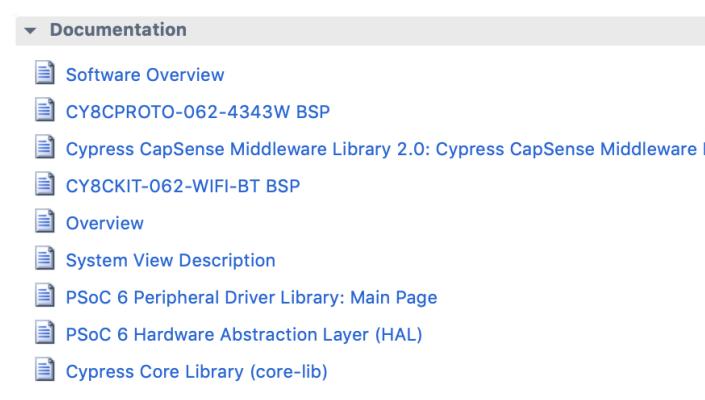
Integrated Debugger

The Eclipse IDE provides an integrated debugger using either the KitProg3 (OpenOCD) on the PSoC 6 development kit, or through a MinProg4 or Segger J-Link debug probe.



Documentation

After creating a project, assorted links to documentation are available directly from the Quick Panel, under the "Documentation" section.



Eclipse IDE Tips & Tricks

Eclipse has several quirks that new users may find hard to understand at first. Here are a few tips to make the experience less difficult:

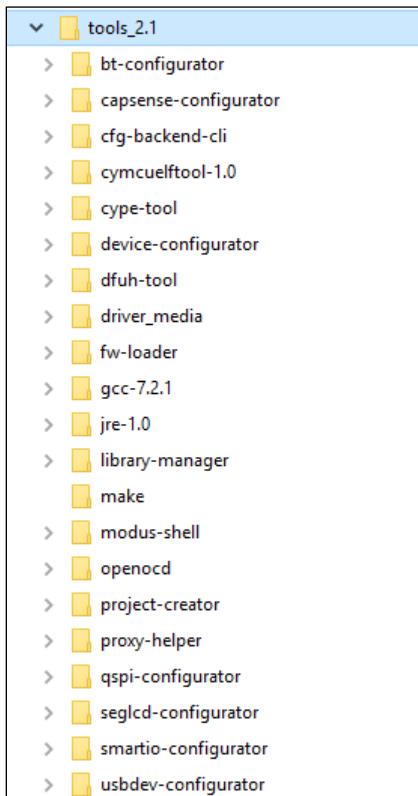
- If your code has IntelliSense issues, use **Program > Rebuild Index** and/or **Program > Build**.
- Sometimes when you import a project, you don't see all the files. Right-click on the project and select **Refresh**.
- Various menus and the Quick Panel show different things depending on what you select in the Project Explorer. Make sure you click on the project you want when trying to use various commands.
- Right-click in the column on the left- side of the text editor view to pop up a menu to:
 - Show/hide line numbers
 - Set/clear breakpoints

Refer also to the Cypress [Eclipse Survival Guide](#) for more tips and tricks.

5.1.3 Tools

The ModusToolbox installer includes several tools that can be used along with the IDE, or as stand-alone tools. These tools include Configurators that are used to configure hardware blocks, as well as utilities to create projects without the IDE or to manage BSPs and libraries. All the tools can be found in the installation directory. The default path (Windows) is:

C:/Users/<user-name>/ModusToolbox/tools_2.1:



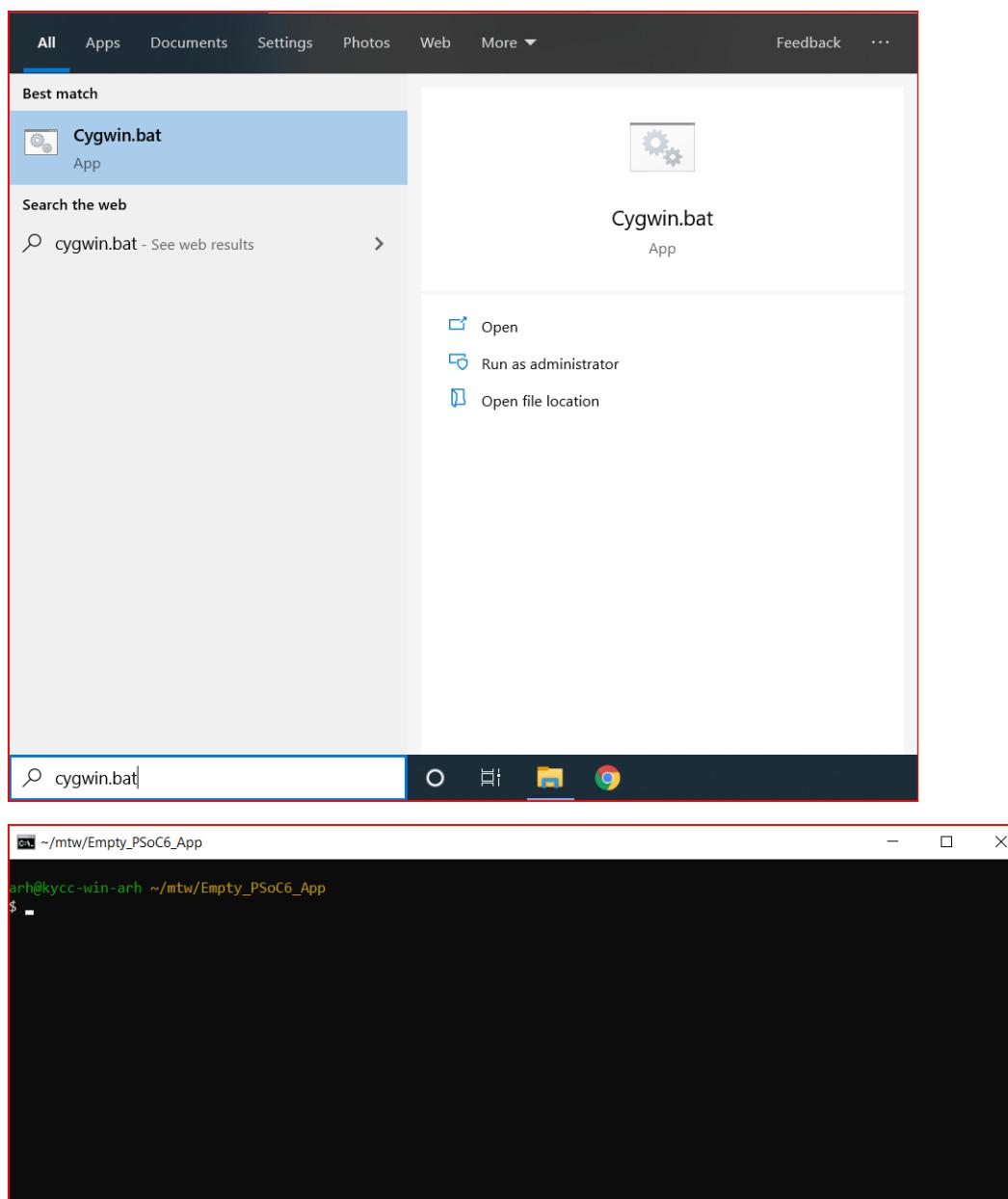
5.1.4 Command Line

You can run all the tool functions using the command line.

Windows

To run the command line, you need a shell that has the correct tools (such as git, make, etc.). This could be your own Cygwin installation or Git Bash. ModusToolbox includes the “modus shell”, which is based on Cygwin. To run this, search for Cygwin.bat (it is in the ModusToolbox installation under ModusToolbox/tools_2.1/modus-shell).

Note: modus-shell is only expected to be required by developers who don't already have something like Cygwin installed.



macOS / Linux

To run the command line on macOS or Linux, just open a terminal window.

Make Targets (Commands)

In order to have the command line commands work, you need to be at the top level of a ModusToolbox project where the Makefile is located.

The following table lists the most commonly used make targets (i.e. commands). Refer also to the [ModusToolbox User Guide](#) document.

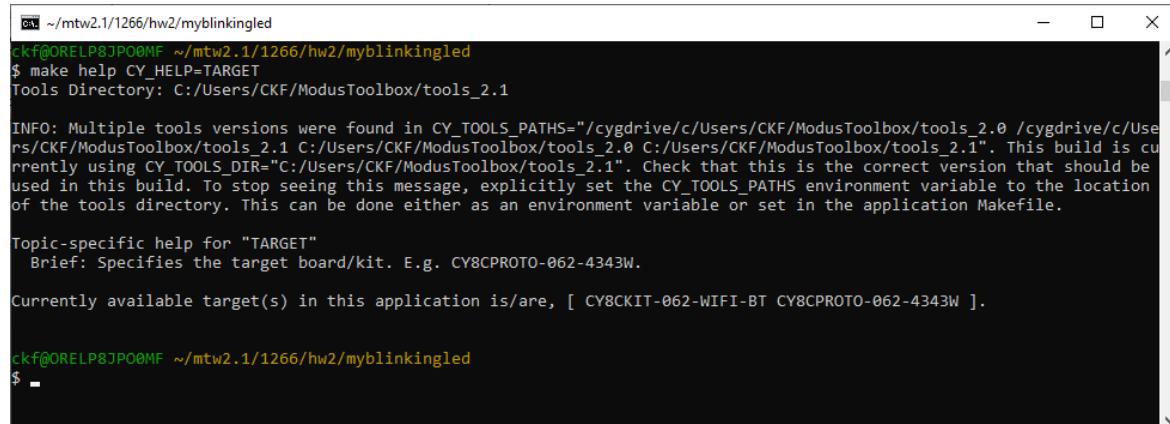
| Make Command | Description |
|---------------|---|
| make help | This command will print out a list of all of the make targets. To get help on a specific target type “make help CY_TARGET=getlibs” (or whichever target you want) |
| make getlibs | Hierarchically process all of the “.lib” files and bring all of the libraries into your project |
| make debug | Build your project, program it to the device, and then launch a GDB server for debugging |
| make program | Build and program your project |
| make qprogram | Program without building. |
| make config | This command will open up the device configurator |

The help make target by itself will print out top level help information. For help on a specific variable or target use `make help CY_HELP=<variable or target>`. For example:

```
make help CY_HELP=build
```

or

```
make help CY_HELP=TARGET
```



```
ckf@ORELP8JP00MF ~/mtw2.1/1266/hw2/myblinkingled
$ make help CY_HELP=TARGET
Tools Directory: C:/Users/CKF/ModusToolbox/tools_2.1

INFO: Multiple tools versions were found in CY_TOOLS_PATHS="/cygdrive/c/Users/CKF/ModusToolbox/tools_2.0 /cygdrive/c/Users/CKF/ModusToolbox/tools_2.1 C:/Users/CKF/ModusToolbox/tools_2.0 C:/Users/CKF/ModusToolbox/tools_2.1". This build is currently using CY_TOOLS_DIR="C:/Users/CKF/ModusToolbox/tools_2.1". Check that this is the correct version that should be used in this build. To stop seeing this message, explicitly set the CY_TOOLS_PATHS environment variable to the location of the tools directory. This can be done either as an environment variable or set in the application Makefile.

Topic-specific help for "TARGET"
  Brief: Specifies the target board/kit. E.g. CY8CPROTO-062-4343W.

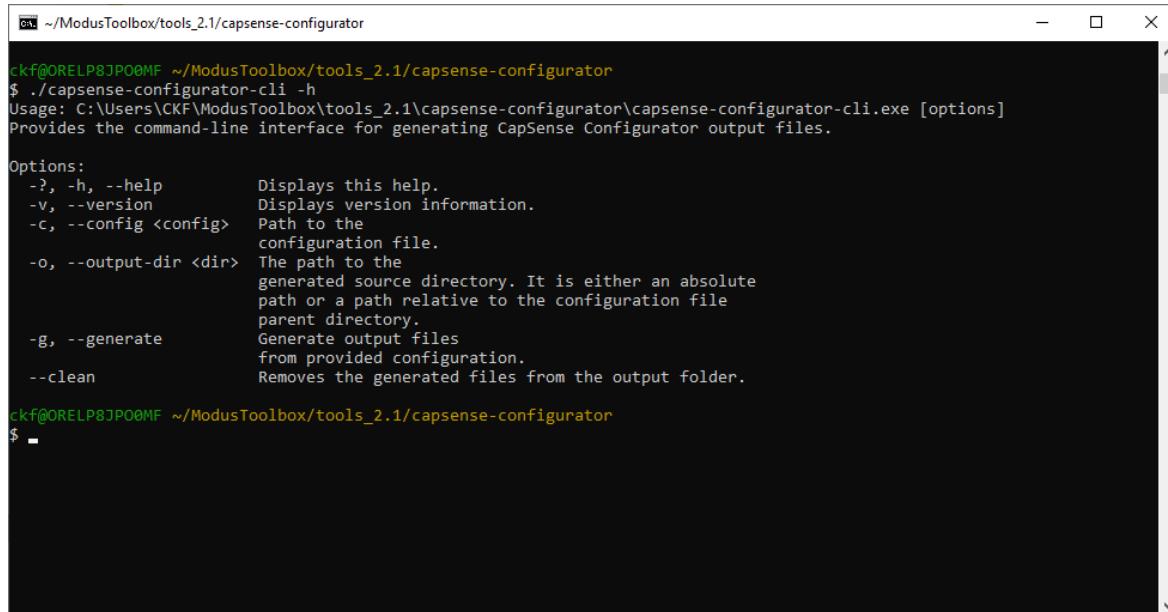
Currently available target(s) in this application is/are, [ CY8CKIT-062-WIFI-BT CY8CPROTO-062-4343W ].

ckf@ORELP8JP00MF ~/mtw2.1/1266/hw2/myblinkingled
$ -
```

Accessing Tools from the Command Line

All the tools can be run from the command line as well. Launch each tool from its respective directory inside ModusToolbox/tools_2.1. The `-h` option shows help. For example:

```
./capsense-configurator-cli -h
```



The screenshot shows a terminal window with the following text:

```
ckf@ORELP8JP0MF ~/ModusToolbox/tools_2.1/capsense-configurator
$ ./capsense-configurator-cli -h
Usage: C:\Users\CKF\ModusToolbox\tools_2.1\capsense-configurator\capsense-configurator-cli.exe [options]
Provides the command-line interface for generating CapSense Configurator output files.

Options:
 -?, -h, --help           Displays this help.
 -v, --version            Displays version information.
 -c, --config <config>   Path to the
                         configuration file.
 -o, --output-dir <dir>  The path to the
                         generated source directory. It is either an absolute
                         path or a path relative to the configuration file
                         parent directory.
 -g, --generate           Generate output files
                         from provided configuration.
 --clean                  Removes the generated files from the output folder.

ckf@ORELP8JP0MF ~/ModusToolbox/tools_2.1/capsense-configurator
$ -
```

5.2 Demo Walkthrough

5.2.1 Blinking LED from the Eclipse IDE

Demonstration of how to create a project, build it, and program the kit.

5.2.2 Blinking LED from command line interface

Repeat demonstration but using the command-line tools rather than an IDE.

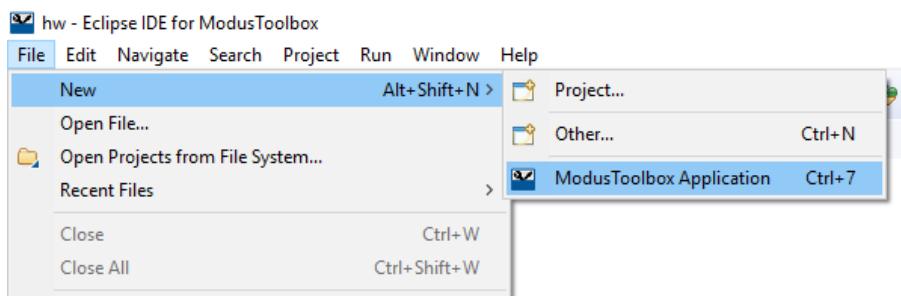
5.3 Exercises (Part 1)

5.3.1 Blinking LED from the Eclipse IDE

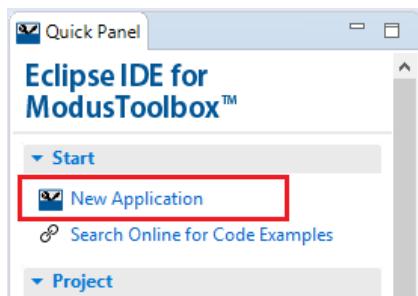
For the first example project, we will use the Eclipse IDE to create, build, and program the classic blinking LED project.



1. Select either **File > New > ModusToolbox Application**.



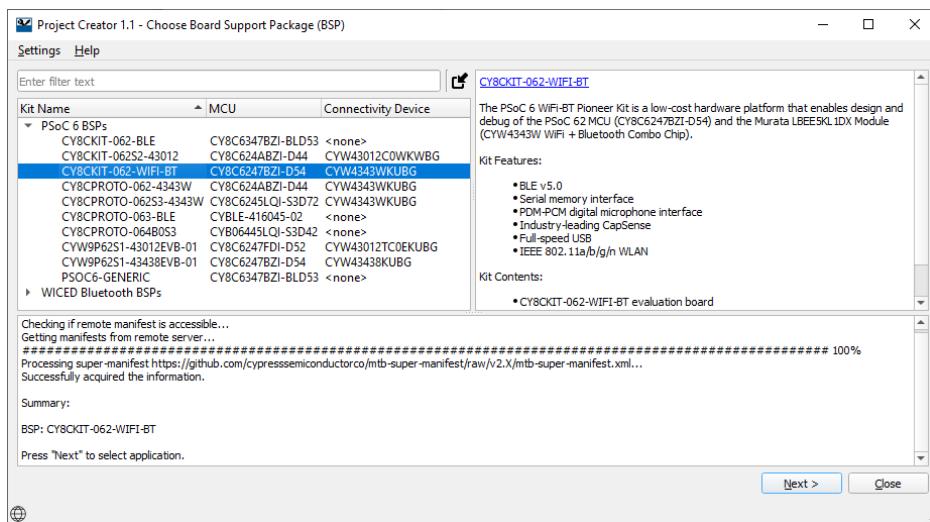
Or on the Quick Panel, click the **New Application** link.



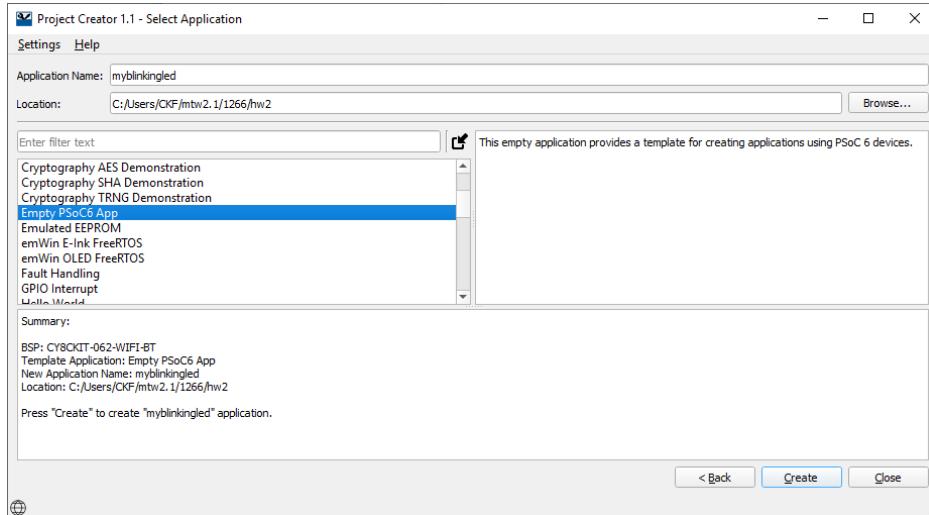
Either option launches the ModusToolbox Project Creator tool.



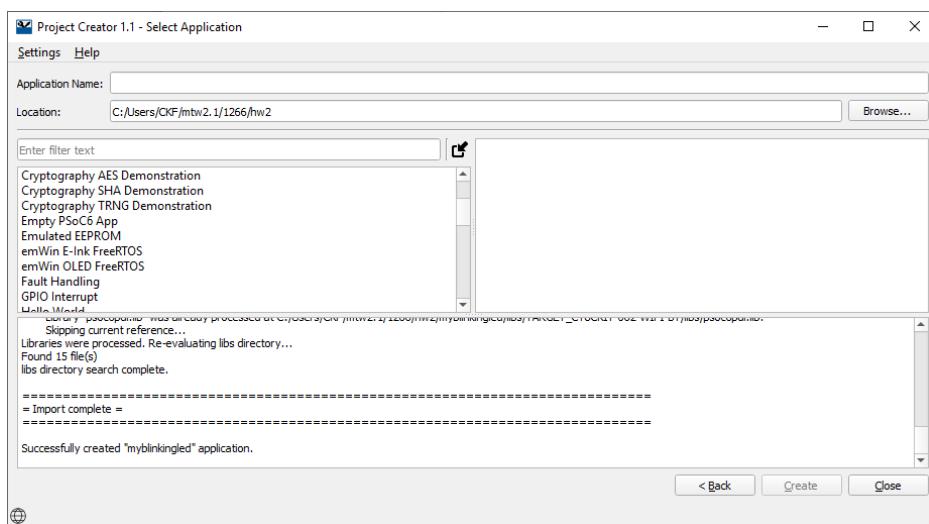
2. Choose the correct development kit.



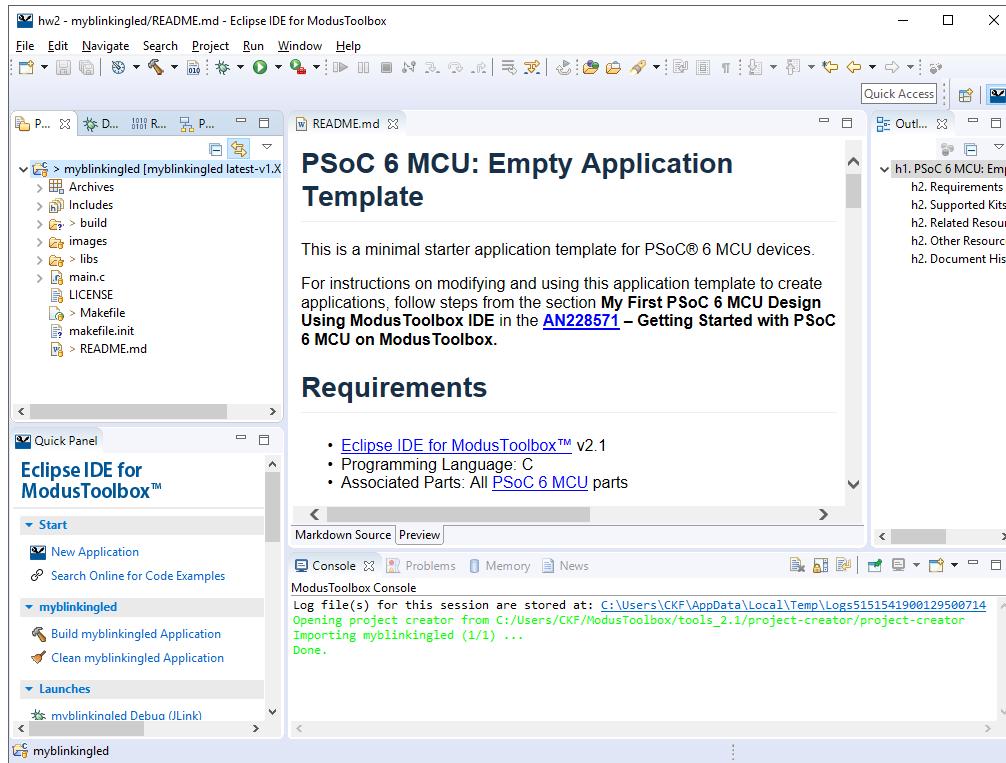
3. Select "Empty PSoC 6 App" and give your project a name (in this case "myblinkingled"). Then, click **Create**.



4. After several moments, the application is created. Click **Close**.



Then, the application is imported into the Eclipse IDE.



5. Double click on main.c and change the code to look like this:

```
#include "cy_pdl.h"
#include "cyhal.h"
#include "cybsp.h"
int main(void)
{
    cy_rslt_t result;

    /* Initialize the device and board peripherals */
    result = cybsp_init() ;
    CY_ASSERT(result == CY_RSLT_SUCCESS);

    __enable_irq();

    cyhal_gpio_init(CYBSP_USER_LED1,CYHAL_GPIO_DIR_OUTPUT,
CYHAL_GPIO_DRIVE_STRONG, CYBSP_LED_STATE_OFF);

    for(;;)
    {
        cyhal_gpio_toggle(CYBSP_USER_LED1);
        Cy_SysLib_Delay(500);
    }
}
```



6. Finally, click on "myblinkingled Program (KitProg3)" from the Quick Panel Launches.



You should now have a blinking LED. If you don't, then you probably need to switch the mode of the KitProg to be CMSIS-DAP Bulk or HID. Refer to the "Firmware Loader" section in Chapter 1 for details.

5.3.2 Blinking LED from the Command Line

You can build and program the same project using the command line. For more details, refer to section [5.1.4 Command Line](#).



1. On Windows, navigate to the modus-shell directory and run Cygwin.bat.

```
<install_dir>\ModusToolbox\tools_2.1\modus-shell\
```

On Linux or macOS, open a terminal window.



2. Navigate to the project directory that you created previously. For example:

```
cd <user_home>/mtw/myblinkingled
```



3. Next, run make help to see the list of commands.

```

~/mtw2.1/1266/hw2/myblinkingled
CKFQRELPOOMF ~/mtw2.1/1266/hw2/myblinkingled
$ make help
Tools Directory: C:/Users/CKF/ModusToolbox/tools_2.1

INFO: Multiple tools versions were found in CY_TOOLS_PATHS="/cygdrive/c/Users/CKF/ModusToolbox/tools_2.0 /cygdrive/c/Users/CKF/ModusToolbox/tools_2.1 C:/Users/CKF/ModusToolbox/tools_2.0 C:/Users/CKF/ModusToolbox/tools_2.1". This build is currently using CY_TOOLS_DIR="C:/Users/CKF/ModusToolbox/tools_2.1". Check that this is the correct version that should be used in this build. To stop seeing this message, explicitly set the CY_TOOLS_PATHS environment variable to the location of the tools directory. This can be done either as an environment variable or set in the application Makefile.

=====
Cypress Build System
=====
Copyright 2018-2020 Cypress Semiconductor Corporation
SPDX-License-Identifier: Apache-2.0

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.

=====

This is the help documentation for the Cypress build system.
It lists the supported make targets and make variables.

Usage: make [target][variable]
Example: make help CY_HELP=TOOLCHAIN

=====
General make targets
=====
all           Same as build. i.e. Builds the application.
getlibs       Clones the repositories, and checks out the identified commit.
build         Builds the application.
qbuild        Builds the application using the previous build's source list.

```



4. Then you can run:

```
make clean  
make build  
make program
```

5.4 Create ModusToolbox Projects

As we've discussed already, you can use the ModusToolbox Eclipse-based IDE to develop projects. We understand many developers may not want to use the Eclipse IDE. So, there are several ways to create projects (all based on the `git clone` mechanism). Here are some of the methods you can use:

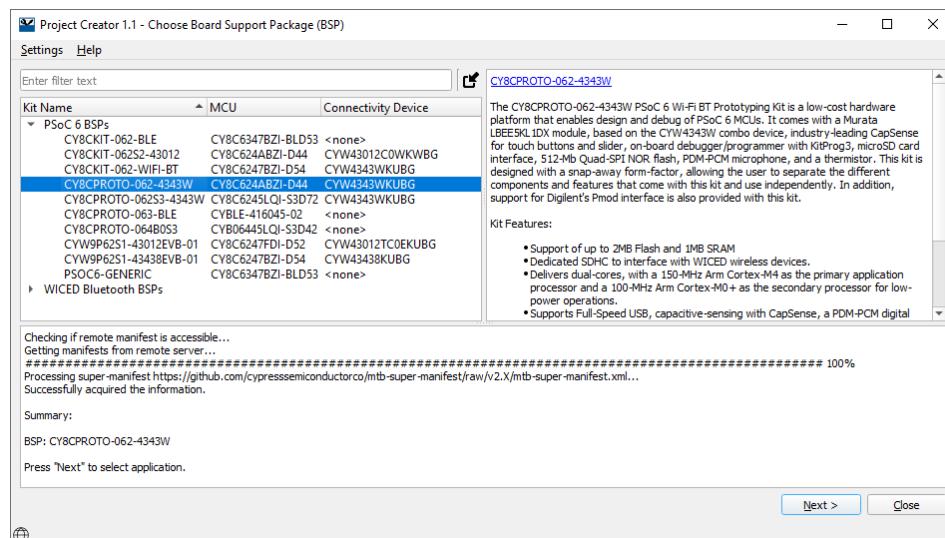
- Project Creator tool through Eclipse IDE (discussed previously)
- Stand-alone Project Creator tool
- Command Line / Manual Setup

5.4.1 Project Creator Tool

Instead of using the Project Creator tool through the Eclipse IDE, you can create new projects with the tool in stand-alone mode. It is in the ModusToolbox/tools_2.1/project-creator directory. You can also just search for "project-creator".

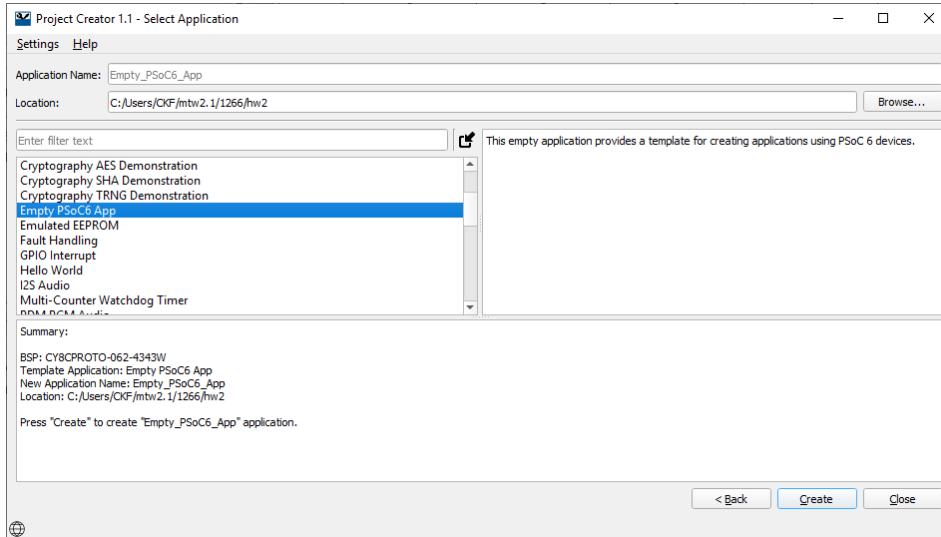


The first thing the project-creator tool does is read the Cypress configuration information from the Cypress GitHub site so that it knows all of the BSPs etc. that we support.

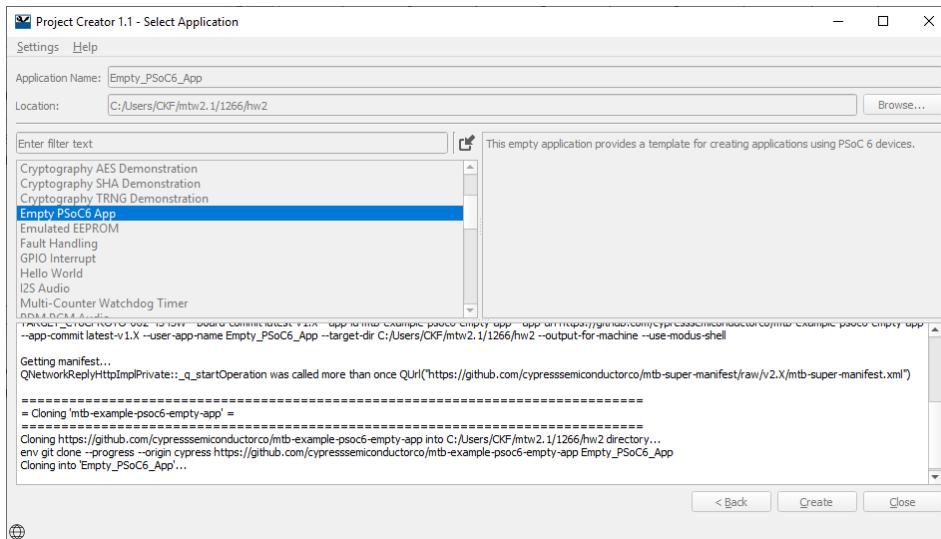


Select a kit and click **Next >**

Now you will be presented with all the Cypress applications supported by the BSP you chose. You can either pick one or click **Import** to start from your own existing project. Pick a location for the project and give it a name.



Click **Create** and it will create the project for you.



The tool runs the `git clone` operation and then the `make getlibs` operation. The project is saved in the directory you specified previously.

When finished, click **Close**.

5.4.2 Command Line / Manual Setup

Project Creator CLI

The Project Creator tool described above also has a command line version that you can run from a shell (e.g. Cygwin or modus-shell). It is in the same folder as the Project Creator GUI (ModusToolbox/tools_2.1/project-creator) but the executable is called project-creator-cli.exe. You can run it with no arguments to see the different options available. You can run it to see a listing of all the available BSPs:

```
./project-creator-cli --list-boards
```

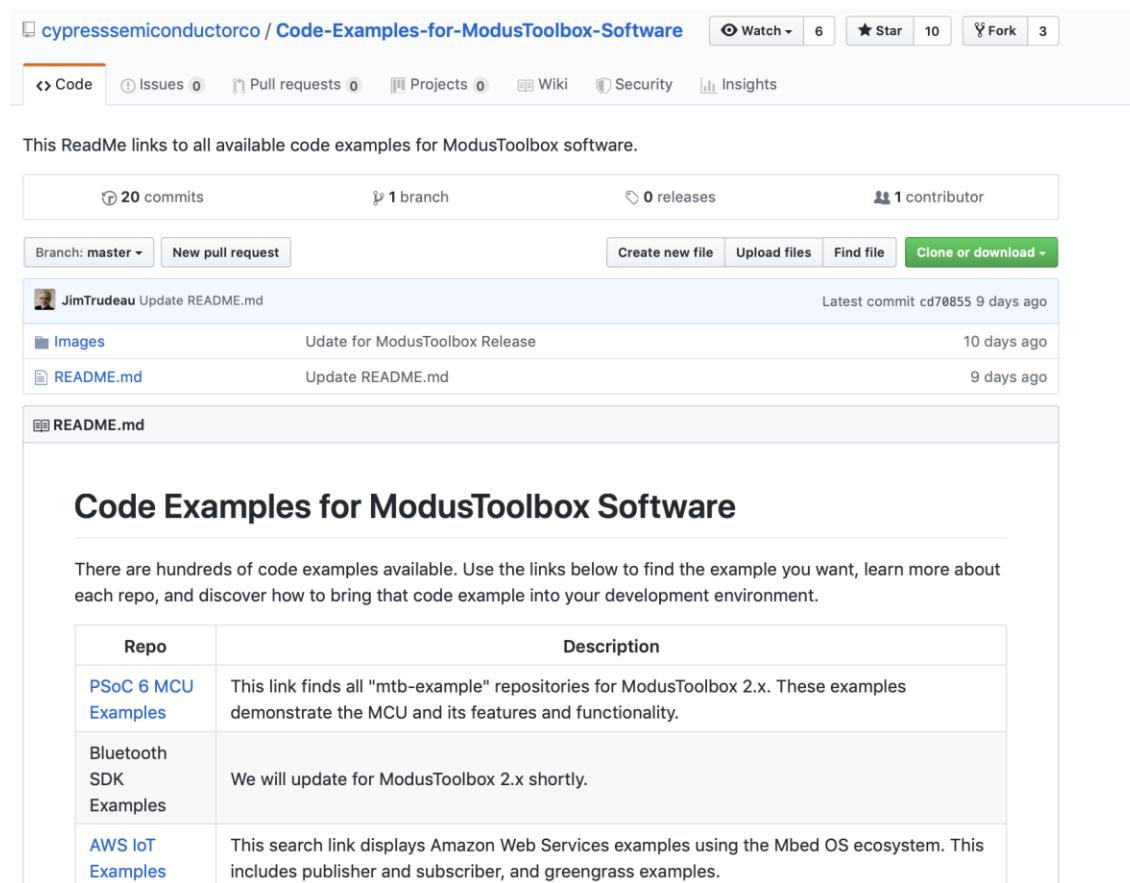
You can also see all the available applications for a given BSP:

```
./project-creator-cli --list-apps <BSP_Name>
```

Instead of using the Project Creator CLI tool, you can use git commands directly to clone a project.

Using GitHub

You can get to the Cypress GitHub repo using the "Search Online for Code Examples" link in the Eclipse IDE Quick Panel. That will take you the following website. You can use the GitHub website tools to clone projects. You can also use the command line to clone as described in the next section.



This screenshot shows the GitHub repository page for 'cypresssemiconductorco / Code-Examples-for-ModusToolbox-Software'. The repository has 6 releases, 10 forks, and 3 issues. The code tab is selected. The repository contains 20 commits, 1 branch, and 1 contributor. The latest commit was made 9 days ago by JimTrudeau. There are three files listed: Images, README.md, and README.md. A table titled 'Code Examples for ModusToolbox Software' provides links to various examples.

| Repo | Description |
|------------------------|--|
| PSoC 6 MCU Examples | This link finds all "mtb-example" repositories for ModusToolbox 2.x. These examples demonstrate the MCU and its features and functionality. |
| Bluetooth SDK Examples | We will update for ModusToolbox 2.x shortly. |
| AWS IoT Examples | This search link displays Amazon Web Services examples using the Mbed OS ecosystem. This includes publisher and subscriber, and greengrass examples. |

Using Git from the Command Line

If you know the URL and don't want to use GitHub, from a shell you can use the `git clone` command to clone the project:

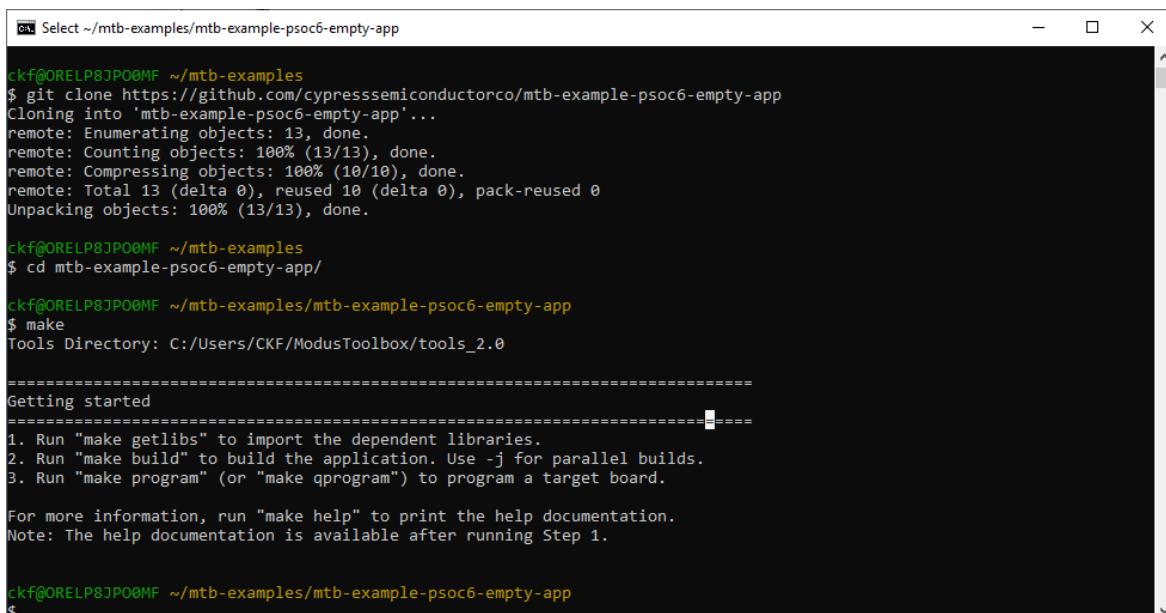
```
git clone https://github.com/cypresssemiconductorco/mtb-example-psoc6-empty-app
```

Make

Once you have cloned the project from GitHub or the command line, change to the project's directory:

```
cd mtb-example-psoc6-empty-app
```

Run make to see the list of available commands:



```
ckf@ORELP8JP00MF ~/mtb-examples
$ git clone https://github.com/cypresssemiconductorco/mtb-example-psoc6-empty-app
Cloning into 'mtb-example-psoc6-empty-app'...
remote: Enumerating objects: 13, done.
remote: Counting objects: 100% (13/13), done.
remote: Compressing objects: 100% (10/10), done.
remote: Total 13 (delta 0), reused 10 (delta 0), pack-reused 0
Unpacking objects: 100% (13/13), done.

ckf@ORELP8JP00MF ~/mtb-examples
$ cd mtb-example-psoc6-empty-app/
ckf@ORELP8JP00MF ~/mtb-examples/mtb-example-psoc6-empty-app
$ make
Tools Directory: C:/Users/CKF/ModusToolbox/tools_2.0

=====
Getting started
=====
1. Run "make getlibs" to import the dependent libraries.
2. Run "make build" to build the application. Use -j for parallel builds.
3. Run "make program" (or "make qprogram") to program a target board.

For more information, run "make help" to print the help documentation.
Note: The help documentation is available after running Step 1.

ckf@ORELP8JP00MF ~/mtb-examples/mtb-example-psoc6-empty-app
$
```

Then you can run:

```
make getlibs
make build
make program
```

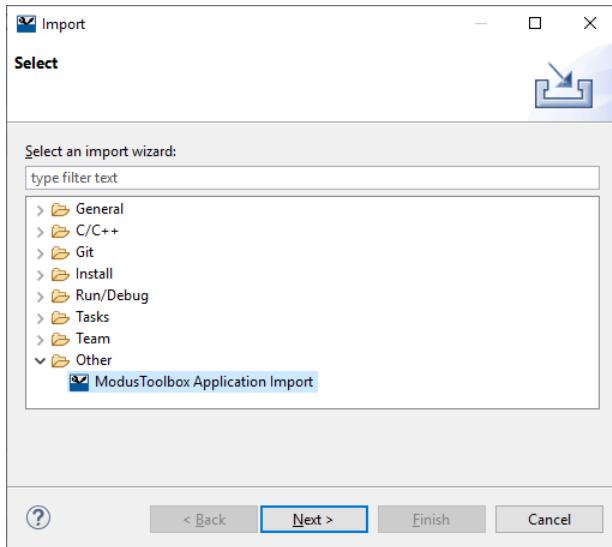
Note: If you used the CY8CKIT-062-WIFI-BT kit, programming will not succeed since the default library and TARGET are set for the CY8CPROTO-062-4343W kit. You will need to add the CY8CKIT-062-WIFI-BT library to the project (or use the Library Manager) and change the TARGET in the Makefile (or specify it on the command line). See section [5.8 Libraries & Library Manager](#) for more details about libraries.

5.5 Import Projects

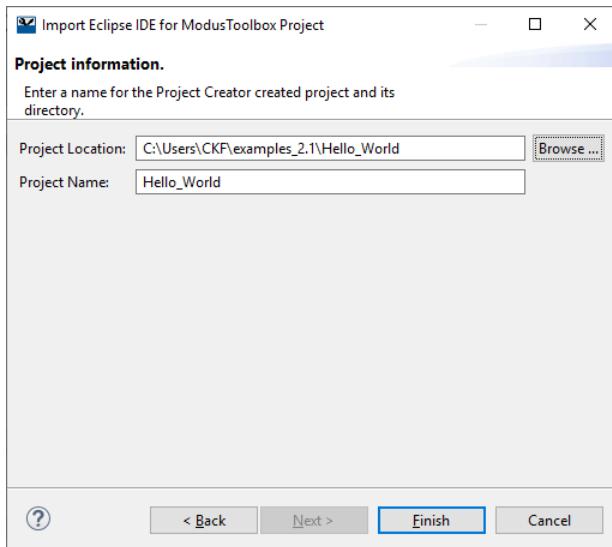
In addition to creating new projects with the Project Creator tool, you can import projects for use in the IDE, as well as to create projects based on existing projects.

5.5.1 Import Projects into Eclipse IDE

To import a project created with the Project Creator tool or command line into the Eclipse IDE, use the Import feature: **Import > Other > ModusToolbox Application Import**.



After clicking **Next >**, browse to the location of the application's directory, select it, and click **Finish**.



This import mechanism runs various commands to make the application run smoothly in the Eclipse IDE for ModusToolbox.

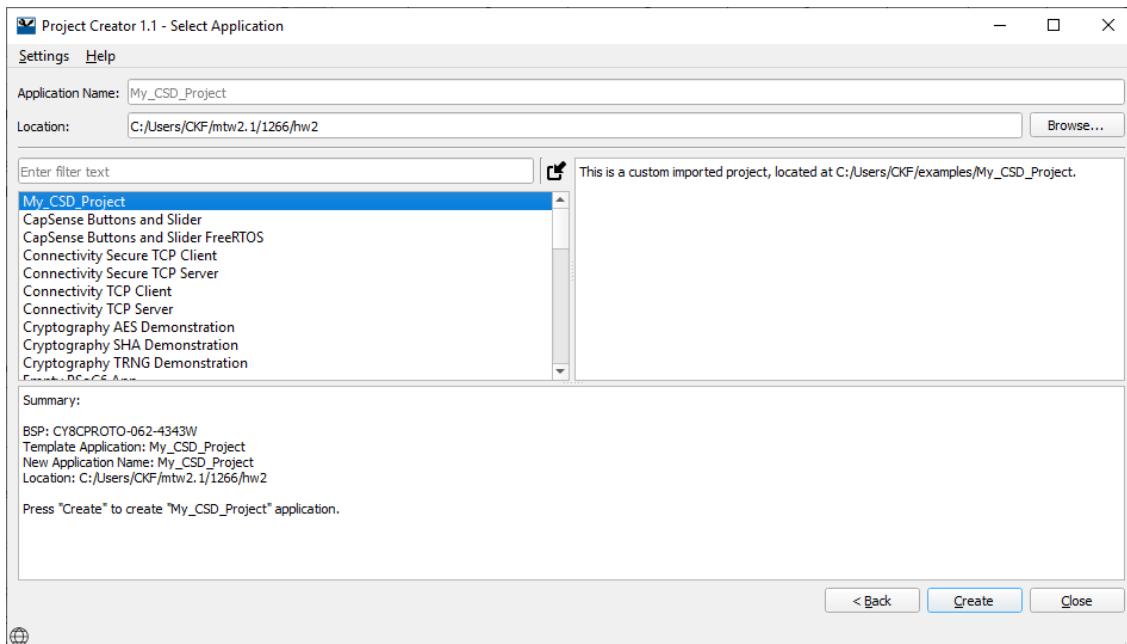
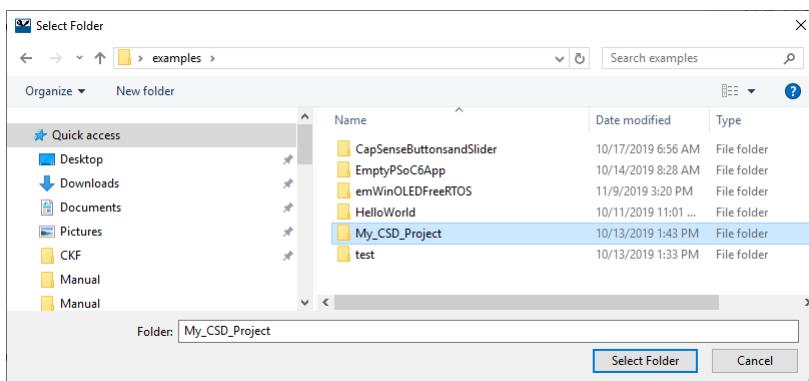
5.5.2 Import Projects into Project Creator Tool

Another option is to create a new project based on an existing project that you have on your computer. To do this, open the Project Creator tool, and select the board for the project.

Then, instead of choosing an application from the list, click the **Import** button .



This allows you to search your computer and then create a new project based on an old project. Make sure the path you select is to the directory that contains the Makefile (or one above it – more on that in a minute) – otherwise the import will fail. The import will import a new copy of the project into your workspace with the name you provide.



Note that the existing project can be one that is in your workspace or one that is located somewhere else. It does not need to be a full Eclipse project. At a minimum, it needs a Makefile and source code files, but it may contain other items such as configurator files and .lib files.

If you specify a path to a directory that is above the Makefile directory, the hierarchy will be maintained, and the path will be added to each individual application name. This can be useful if you have a directory containing multiple applications in sub-directories and you want to import them all at once. For example, if you have a directory called "myapps" containing the 2 subdirectories "myapp1" and "myapp2", when you import from the "myapps" level you will get a project called myapps.myapp1 and myapps.myapp2.

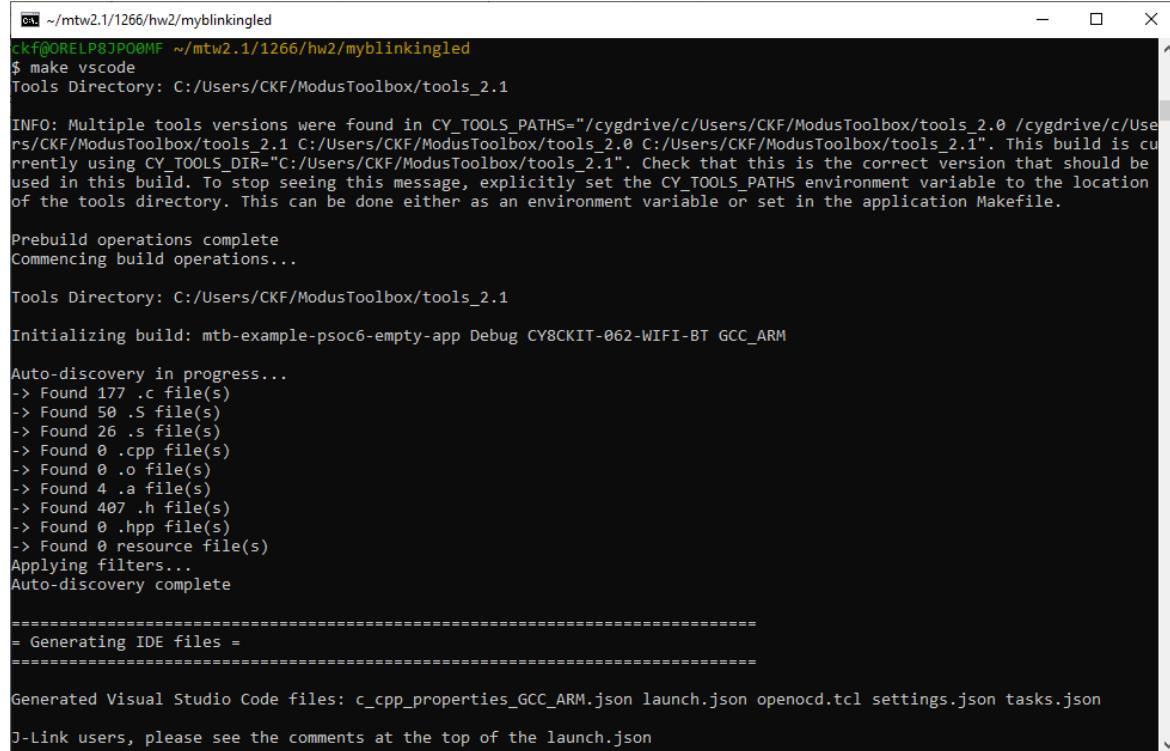
5.6 Exporting ModusToolbox Projects

For users who prefer not to use the Eclipse IDE, the ModusToolbox build system provides make targets to generate files for various 3rd party IDEs.

5.6.1 Export for Visual Studio (VS) Code

One alternative to using the Eclipse-based IDE is a code editor program called VS Code. This tool is quickly becoming a favorite editor for developers. The ModusToolbox command line knows how to make all the files required for VS Code to edit, build, and program a ModusToolbox program. To create these files, go to an existing project directory and type the following from the command line:

```
make vscode
```



```
ckf@ORELP8JPO0MF ~/mtw2.1/1266/hw2/myblinkingled
$ make vscode
Tools Directory: C:/Users/CKF/ModusToolbox/tools_2.1

INFO: Multiple tools versions were found in CY_TOOLS_PATHS="/cygdrive/c/Users/CKF/ModusToolbox/tools_2.0 /cygdrive/c/Users/CKF/ModusToolbox/tools_2.1 C:/Users/CKF/ModusToolbox/tools_2.0 C:/Users/CKF/ModusToolbox/tools_2.1". This build is currently using CY_TOOLS_DIR="C:/Users/CKF/ModusToolbox/tools_2.1". Check that this is the correct version that should be used in this build. To stop seeing this message, explicitly set the CY_TOOLS_PATHS environment variable to the location of the tools directory. This can be done either as an environment variable or set in the application Makefile.

Prebuild operations complete
Commencing build operations...

Tools Directory: C:/Users/CKF/ModusToolbox/tools_2.1

Initializing build: mtb-example-psoc6-empty-app Debug CY8CKIT-062-WIFI-BT GCC_ARM

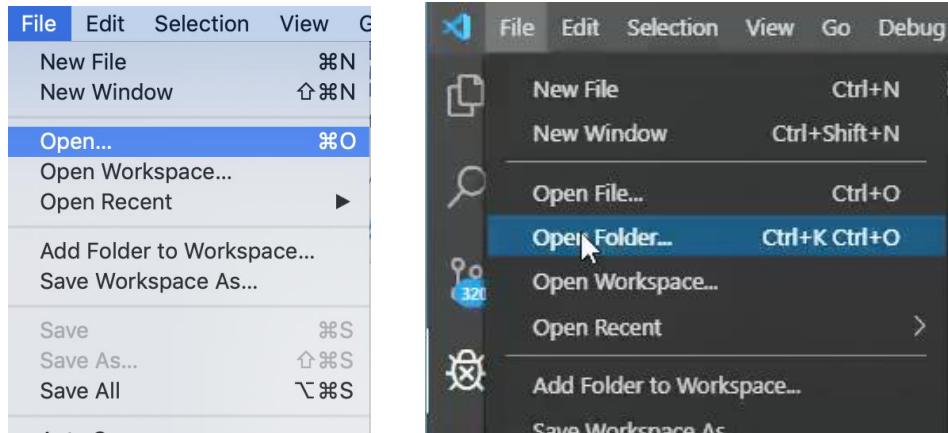
Auto-discovery in progress...
-> Found 177 .c file(s)
-> Found 50 .S file(s)
-> Found 26 .s file(s)
-> Found 0 .cpp file(s)
-> Found 0 .o file(s)
-> Found 4 .a file(s)
-> Found 407 .h file(s)
-> Found 0 .hpp file(s)
-> Found 0 resource file(s)
Applying filters...
Auto-discovery complete
=====
= Generating IDE files =
=====

Generated Visual Studio Code files: c_cpp_properties_GCC_ARM.json launch.json openocd.tcl settings.json tasks.json

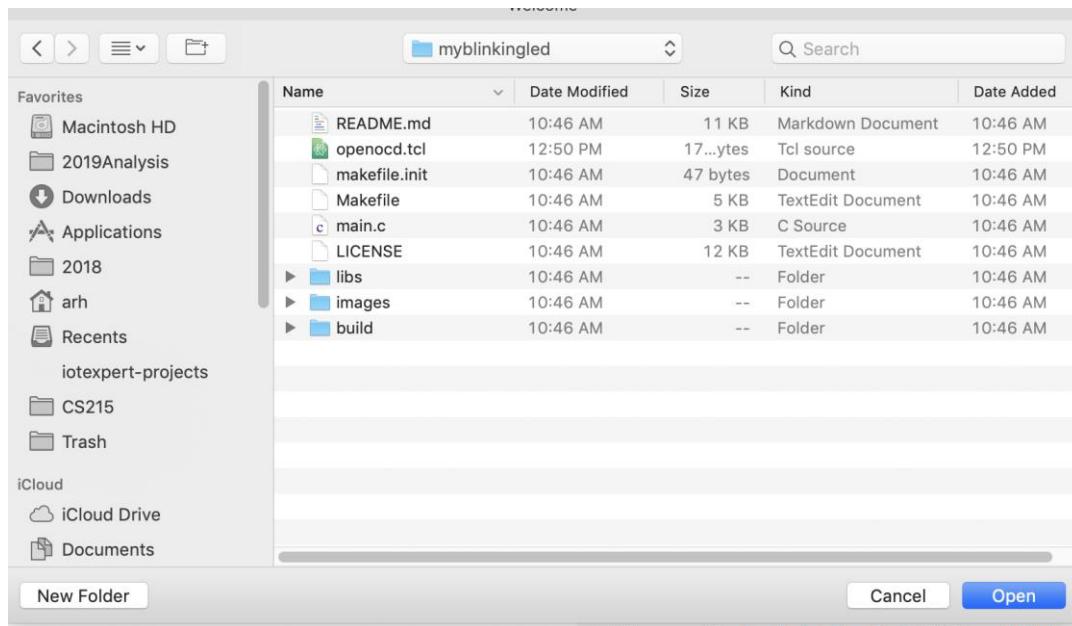
J-Link users, please see the comments at the top of the launch.json
```

Note: By default, if you don't specify the target, this will create VS Code settings for the CY8CPROTO-062-4343W kit (since that is the target specified in the Makefile). If you want a different target, it is best to add a .lib for the desired target and update the TARGET variable in the Makefile before running make vscode.

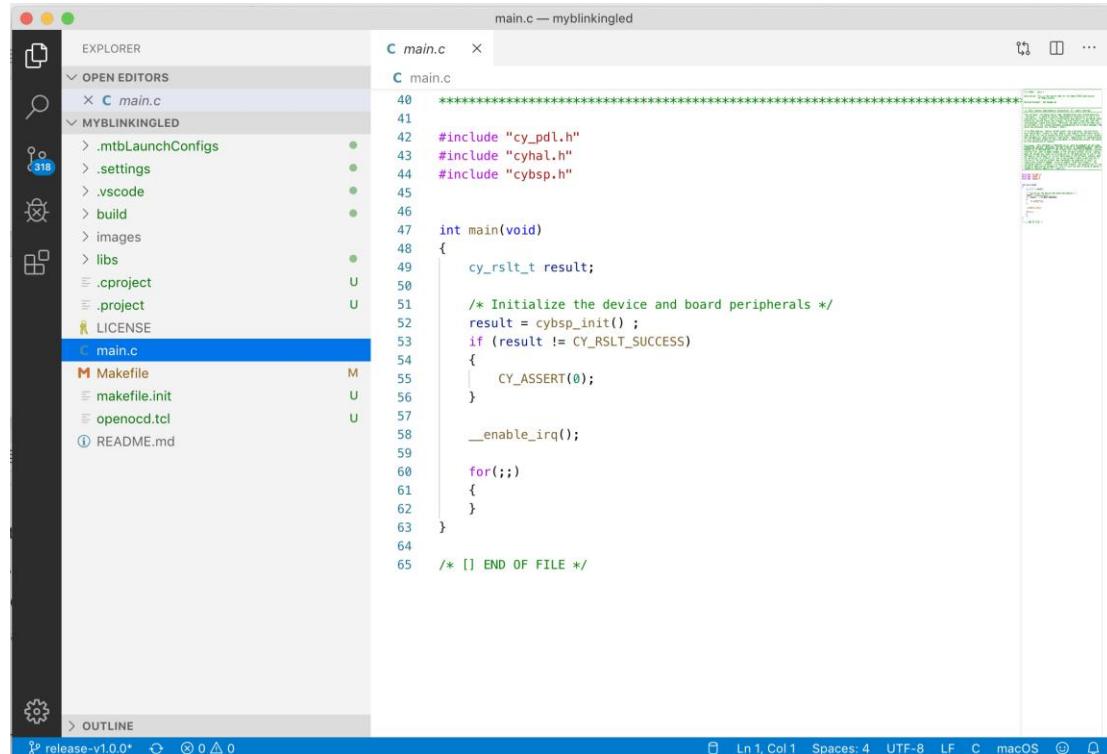
Once you have generated the necessary files using `make vscode`, open the project in Visual Studio Code using **File > Open** (on macOS) or **File > Open Folder** (on Windows).



Then, navigate to the directory where your project resides and click **Open**.



Now your windows should look something like this. You can open the files by clicking on them in the Explorer window.



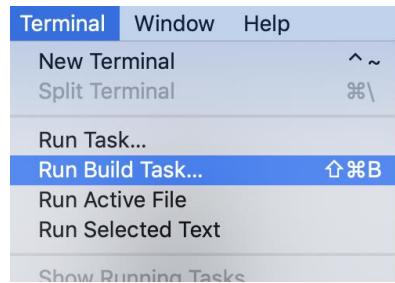
The screenshot shows the ModusToolbox IDE interface. On the left is the Explorer window displaying the project structure:

- OPEN EDITORS: main.c
- MYBLINKINGLED:
 - .mtbLaunchConfigs
 - .settings
 - .vscode
 - build
 - images
 - libs
 - .cproject
 - .project
 - LICENSE
 - main.c
- Makefile
- makefile.init
- openocd.tcl
- README.md

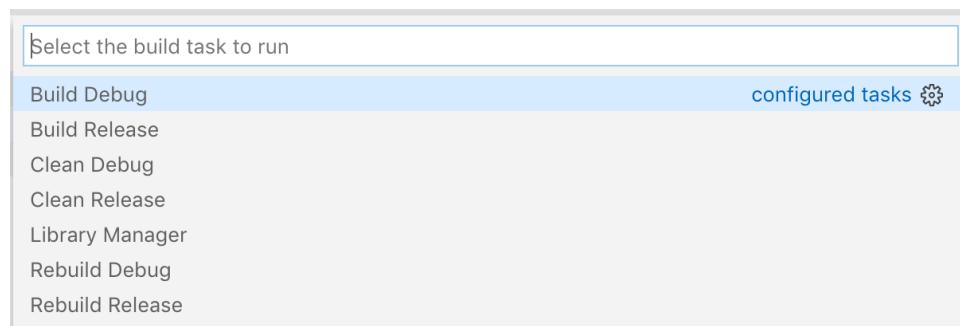
The main editor window shows the content of main.c:main.c — myblinkingled
C main.c
40 ****
41
42 #include "cy_pdl.h"
43 #include "cyhal.h"
44 #include "cybsp.h"
45
46 int main(void)
47 {
48 cy_rslt_t result;
49
50 /* Initialize the device and board peripherals */
51 result = cybsp_init();
52 if (result != CY_RSLT_SUCCESS)
53 {
54 CY_ASSERT(0);
55 }
56
57 __enable_irq();
58
59 for(;;)
60 {
61 }
62 }
63
64 /* [] END OF FILE */
65

At the bottom, the status bar shows: release-v1.0.0* Ln 1, Col 1 Spaces: 4 UTF-8 LF C macOS

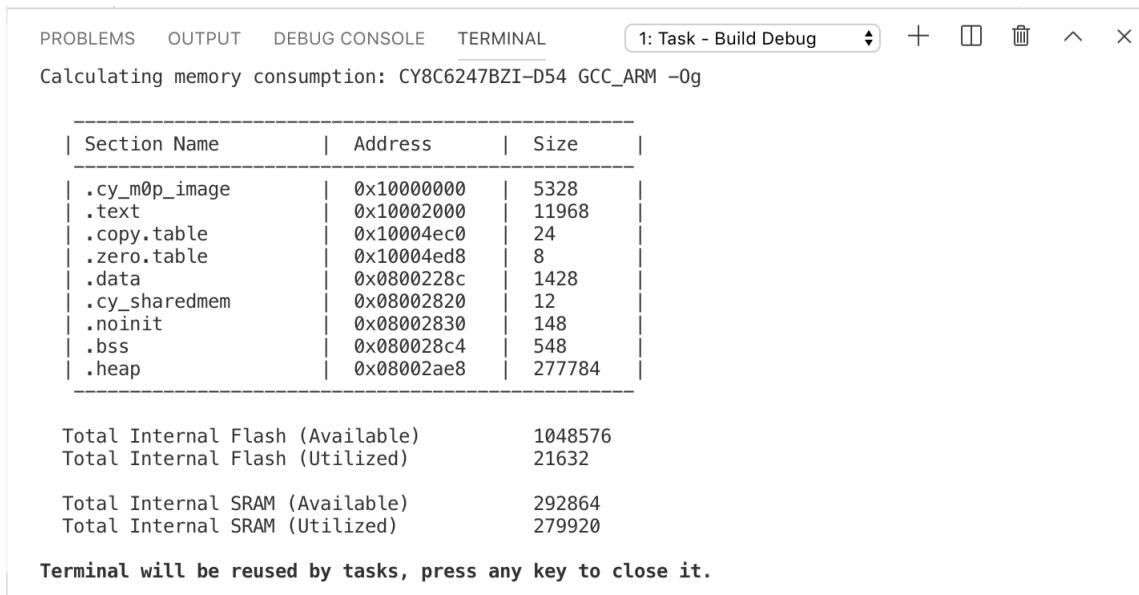
In order to build your project, select **Terminal > Run Build Task...**



Then, select **Build Debug**. This will essentially run make build at the top level of your project.



You should see the normal compiler output in the console at the bottom of VS Code:



PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL 1: Task - Build Debug + ⌂ ⌂ ⌂ ⌂ ⌂ ⌂ ⌂ ⌂ ⌂

Calculating memory consumption: CY8C6247BZI-D54 GCC_ARM -Og

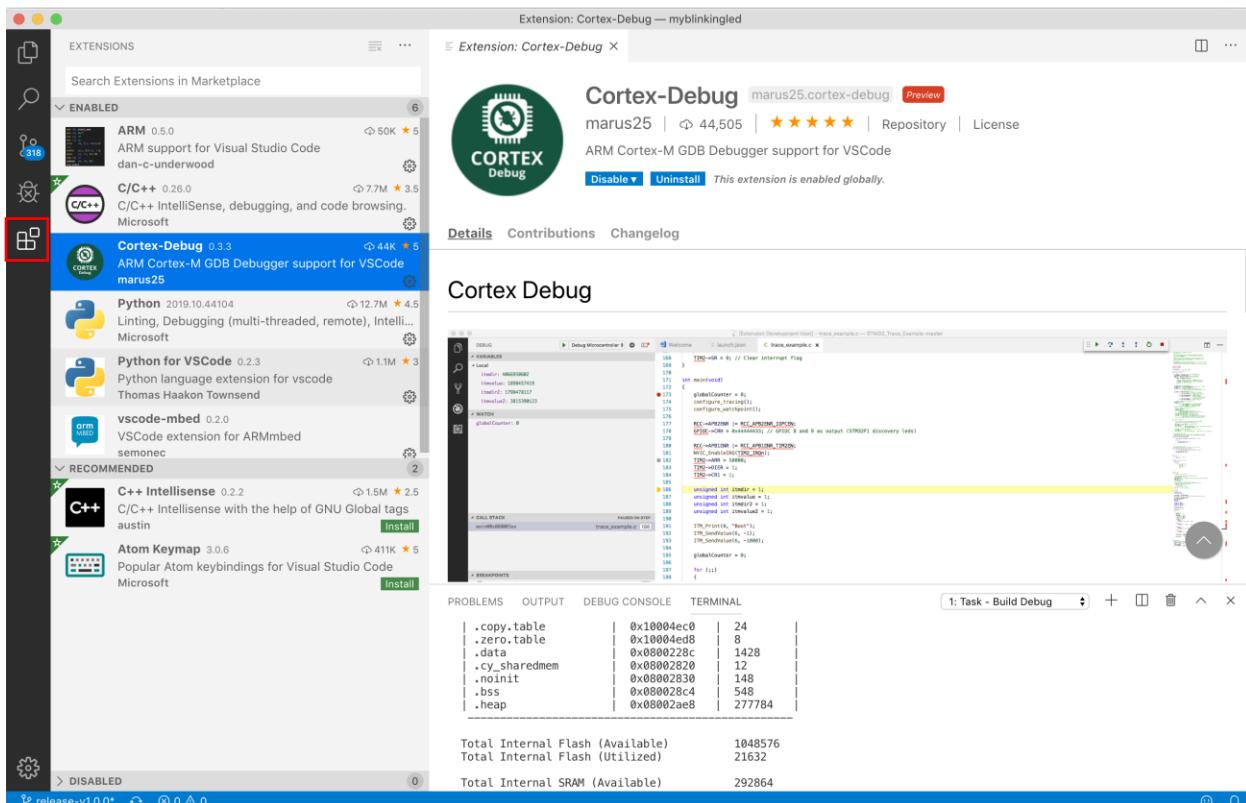
| Section Name | Address | Size |
|---------------|------------|--------|
| .cy_m0p_image | 0x10000000 | 5328 |
| .text | 0x10002000 | 11968 |
| .copy.table | 0x10004ec0 | 24 |
| .zero.table | 0x10004ed8 | 8 |
| .data | 0x0800228c | 1428 |
| .cy_shmem | 0x08002820 | 12 |
| .noinit | 0x08002830 | 148 |
| .bss | 0x080028c4 | 548 |
| .heap | 0x08002ae8 | 277784 |

Total Internal Flash (Available) 1048576
Total Internal Flash (Utilized) 21632

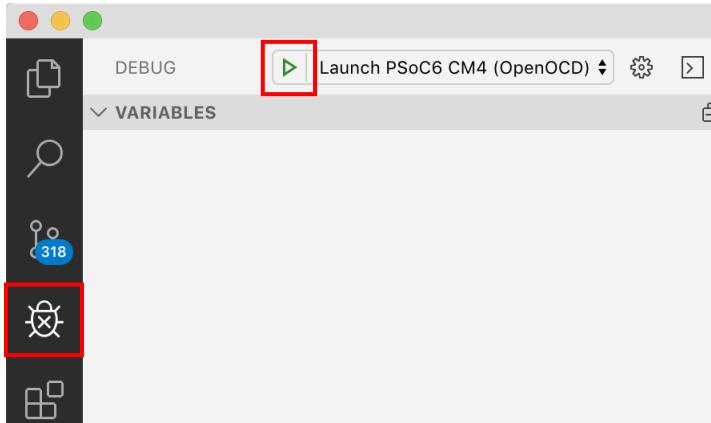
Total Internal SRAM (Available) 292864
Total Internal SRAM (Utilized) 279920

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

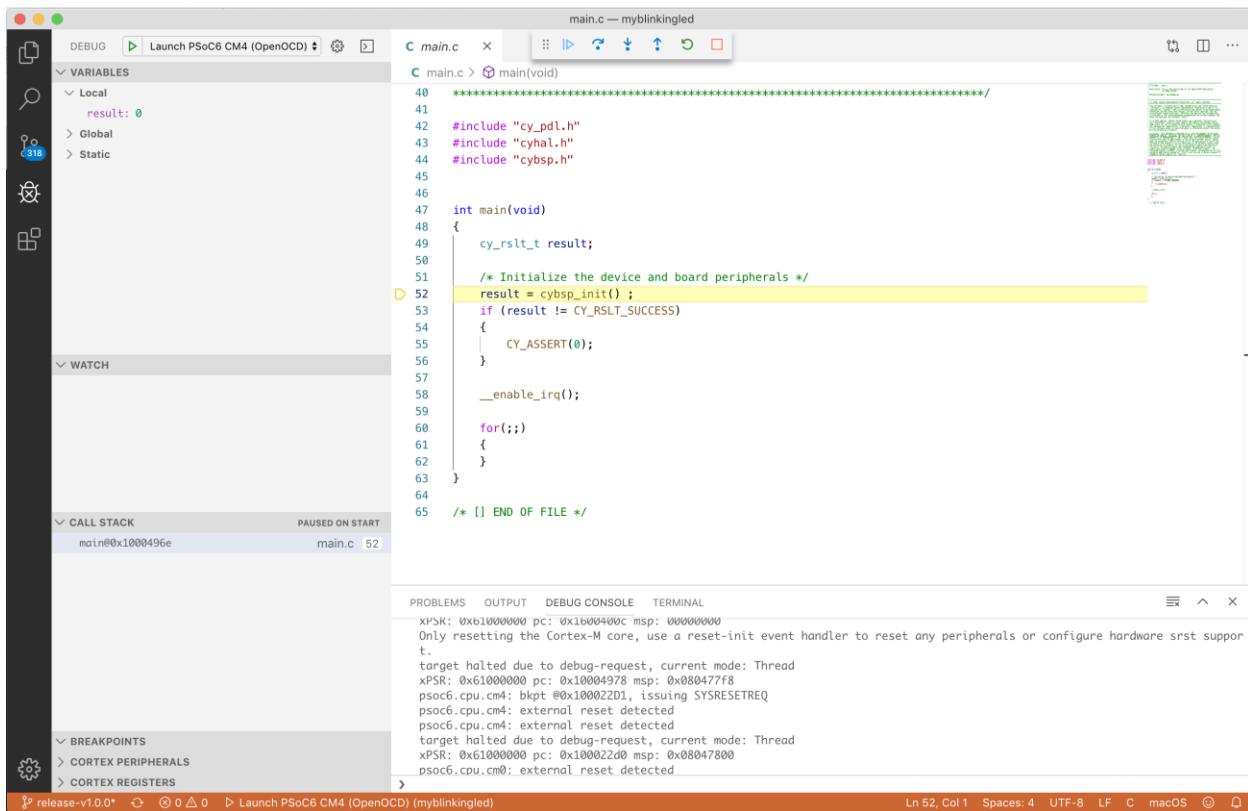
To program the kit or run the debugger, you need to install the “Cortex-Debug” VS Code plug in from the marketplace. Click the little boxes symbol. Then, search for Cortex-Debug. Then click **Install**.



Typically, you will need to build first, then program or program and start the debugger. The program and debug can be done in one step but not the build. That is, you don't need to explicitly program before debugging, but you do need to build first to see any changes. To do programming or debug, click the little "bug" icon on the left side of the screen. Then click the drop-down next to the green **Play** button to select Program (to program), Launch PSoC 6 CM4 (to program and debug), or Attach PSoC 6 CM4 (to debug without programming).



If you chose one of the debugging options, after clicking **Play**, your screen should look something like this. Your application has been programmed into the chip (if you chose the Launch option). The reset of the chip has happened, and the project has run until "main". Notice the yellow highlighted line. It is waiting on you. You can now add breakpoints or get the program running or single step.



5.6.2 Export for Other 3rd Party IDEs

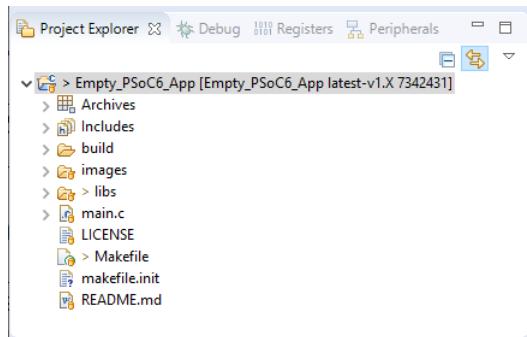
The ModusToolbox build system also provides make targets for IAR Embedded Workbench and Keil µVision. Similar to VS Code, you run the appropriate `make <IDE>` command:

```
make ewarm8
make uvision5
```

For details about using these other 3rd party IDEs, refer to the [ModusToolbox User Guide](#).

5.7 Project Directory Organization

Once created, a typical ModusToolbox PSoC 6 project contains various files and directories.



5.7.1 Archives

Virtual directory that shows static libraries.

5.7.2 Includes

Virtual directory that shows the include paths used to find header files.

5.7.3 build

This directory contains build files, such as the .elf and .hex files.

5.7.4 images

This directory contains artwork images used by the documentation.

5.7.5 libs

The libs directory contains all the code libraries and supporting files, including the targets, BSP, HAL, PDL, firmware, components, and documentation. These are covered in later sections.

5.7.6 main.c

The main.c file is the primary application file that contains the project's application code.

5.7.7 LICENSE

This is the ModusToolbox license agreement file.

5.7.8 Makefile

The Makefile is used in the application creation process – it defines everything that ModusToolbox needs to know to create/build/program the application. This file is interchangeable between Eclipse IDE and the Command Line Interface so once you create an application, you can go back and forth between the IDE and CLI at will.

Various build settings can be set in the Makefile to change the build system behavior. These can be "make" settings or they can be settings that are passed to the compiler. Some examples are:

Target Device (`TARGET=`)

```
27 #####  
28 # Basic Configuration  
29 #####  
30  
31 # Target board/hardware  
32 TARGET=CY8CKIT-062-WIFI-BT  
33 #####
```

Build Configuration (`CONFIG=`)

```
47 "# Debug -- build with minimal optimizations, focus on debugging.  
48 "# Release -- build with full optimizations  
49 CONFIG=Debug
```

Adding Components (`COMPONENTS=`)

```
#####  
# Advanced Configuration  
#####  
  
# Enable optional code that is ordinarily disabled by default.  
#  
# Available components depend on the specific targeted hardware and firmware  
# in use. In general, if you have  
#  
#     COMPONENTS=foo bar  
#  
# ... then code in directories named COMPONENT_foo and COMPONENT_bar will be  
# added to the build  
#  
COMPONENTS=  
  
# Like COMPONENTS, but disable optional code that was enabled by default.  
DISABLE_COMPONENTS=
```

5.7.9 Makefile.init

This file contains variables used by the make process. This is a legacy file not needed any more.

5.7.10 README.md

Almost every code example / starter application includes a README.md (mark down) file that provides a high-level description of the project.

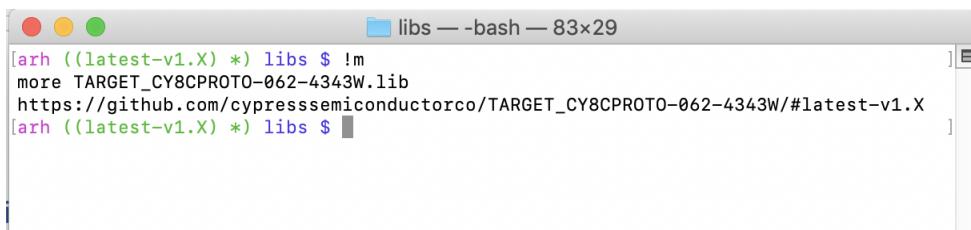
5.8 Libraries & Library Manager

A ModusToolbox project is made up of your application files plus libraries. A library is a related set of code, either in the form of C-source or compiled into archive files. These libraries contain code which is used by your application to get things done. These libraries range from low-level drivers required to boot the chip, to the configuration of your development kit (called Board Support Package) to Graphics or RTOS or CapSense, etc.

ModusToolbox knows about a library in your project by having a “dot lib” file somewhere in the directories of your project. A “dot lib” file is simply a text file that has two parts:

- A URL to a Git repository somewhere that is accessible by your computer such as GitHub
- A Git Commit Hash or Tag that tells which version of the library that you want

A typical library file will look something like this (notice this uses the tag name for the version).

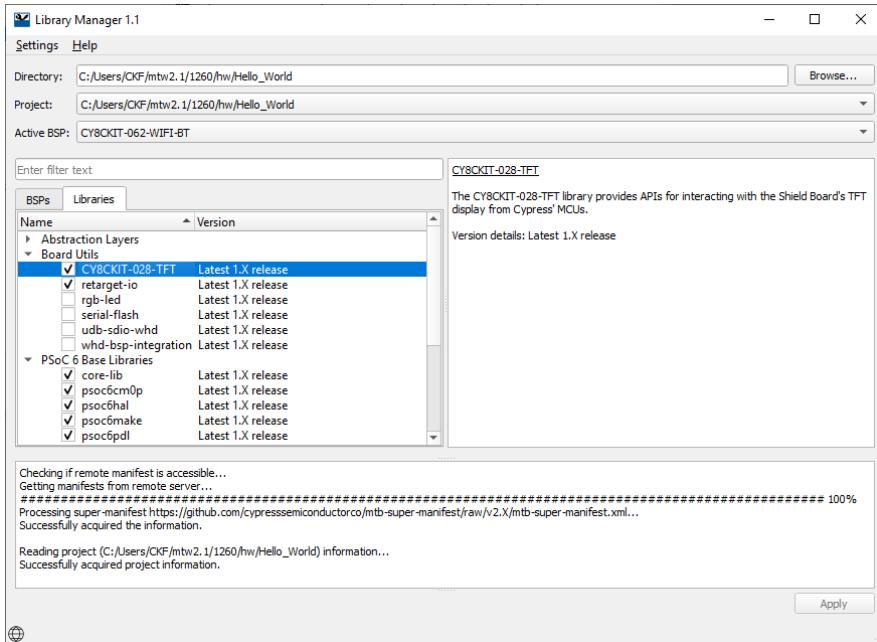


```
libs — -bash — 83x29
[arh ((latest-v1.X) *) libs $ !m
more TARGET_CY8CPROTO-062-4343W.lib
https://github.com/cypresssemiconductorco/TARGET_CY8CPROTO-062-4343W/#latest-v1.X
[arh ((latest-v1.X) *) libs $ ]
```

When you create a new project, ModusToolbox will create a directory in your project called “libs” which will contain all the libraries in your project. In order to actually get the library files into your project, you need to run the command line function `make getlibs` (or run the library manager). This will search your project, find all the “dot lib” files, and bring in the libraries to your project.

5.8.1 Library Manager

ModusToolbox provides a GUI for helping you manage library files. You can run this from the Quick panel link "Library Manager". It is also available outside the IDE from ModusToolbox/tools_2.1/library-manager.



The Library Manager provides a GUI to select which Board Support Package (BSP) and version should be used by default when building a ModusToolbox application. An application can contain multiple BSPs, but a build from the IDE will build and program for the "Active BSP" selected in the library manager. The tool also allows you to add and remove libraries, as well as change their versions.

When you run the Library Manager GUI, it will create ".lib" files. Then it will run `make getlibs`. You can always do this for yourself. You can also remove directories in the libs directory, and they can be recreated by running `make getlibs`. This means that you can check-in just the ".libs" into your code repository and the `make getlibs` command will do the rest of the work.

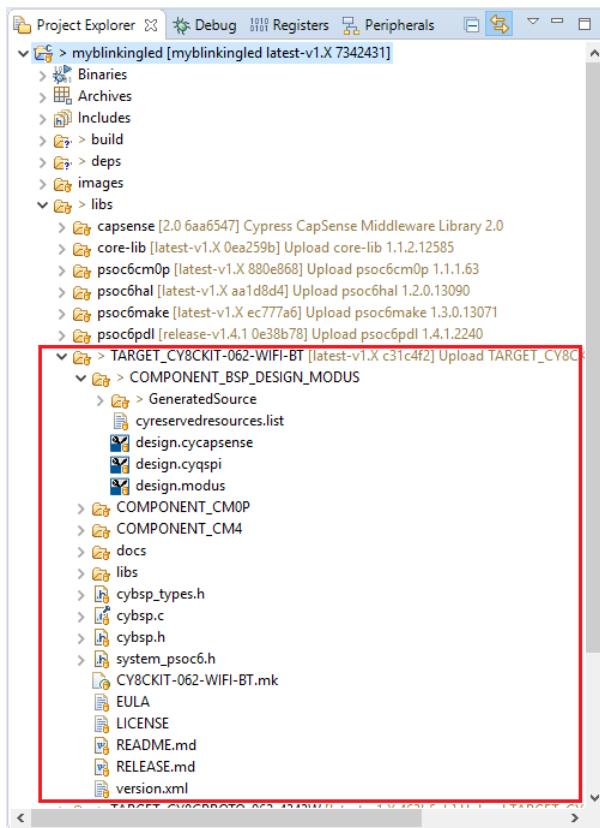
Libraries can be hierarchical, meaning that a library can have ".libs" inside of it and the `make getlibs` command will bring in all the libraries through the hierarchy.

The Library Manager knows where to find libraries by using Manifest files, which will be discussed in the [Manifests](#) section.

5.9 Board Support Packages

Each project is based on a target set of hardware. This target is called a "Board Support Package" (BSP). It contains information about the chip(s) on the board, how they are programmed, how they are connected, what peripherals are on the board, how the device pins are connected, etc. A BSP directory starts with the keyword "TARGET" and can be seen in the project directory as shown here:

5.9.1 BSP Directory Structure



deps

For newer ModusToolbox applications, this directory contains.lib files that specify the required libraries for the underlying device family. These are provided by Cypress and always point to a Cypress owned repo or a repo that is sanctioned by Cypress. You can use the `make getlibs` command to fetch these libraries. The Project Creator and Library Manager tools invoke `make getlibs` automatically.

libs

This directory contains all the libraries after the `make getlibs` command has been invoked.

This directory stores the imported library files after running `make getlibs` to process the .lib files. For older versions of ModusToolbox applications, this directory also contains .lib files. The build system supports both.

COMPONENT_BSP DESIGN MODUS

This directory contains the configuration files (such as design.modus) for use with various BSP configurator tools, including the Device Configurator, QSPI Configurator, and CapSense Configurator. At the start of a build, the build system invokes these tools to generate the source files in the GeneratedSource directory.

docs

The docs directory contains the HTML based documentation for the BSP.

linker

This directory contains linker scripts for all supported toolchains: GCC ARM, IAR, and ARM.

startup

This directory contains the startup code with the reset handler for the target device, written in assembly language for all supported toolchains: GCC ARM, IAR, and ARM.

cybsp_types.h

The cybsp_types.h file contains the aliases (macro definitions) for all the board resources.

cybsp.h / cybsp.c

These files contain the API interface to the board's resources.

You need to include only cybsp.h in your application to use all the features of a BSP. Call the cybsp_init() function from your code to initialize the board (that function is defined in cybsp.c).

<targetname>.mk

This file defines the DEVICE and other BSP-specific make variables such as COMPONENTS.

5.9.2 BSP Documentation

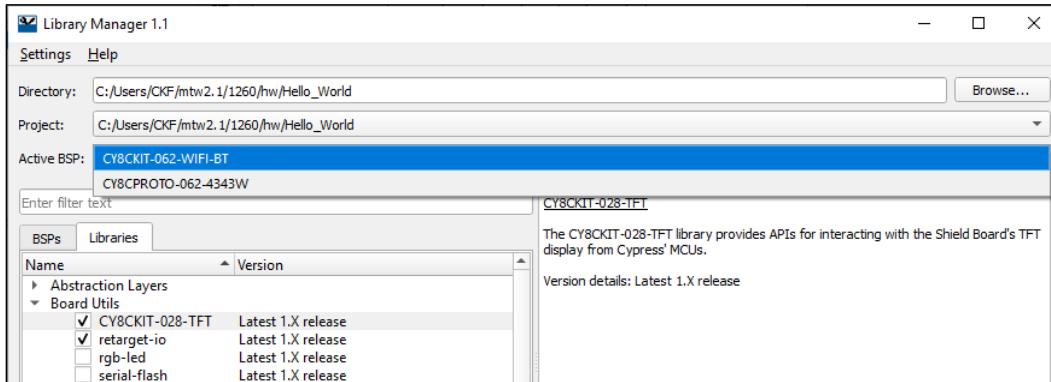
Each BSP provides HTML documentation that is specific to the selected board. It also includes an API Reference and the BSP Overview. After creating a project, there is a link to the BSP documentation in the IDE Quick Panel. As mentioned previously, this documentation is located in the BSP's "docs" directory.

Screenshot of the ModusToolbox 101 IDE Quick Panel showing the "Documentation" section expanded. The "CY8CPROTO-062-4343W BSP" link is highlighted with a red box.

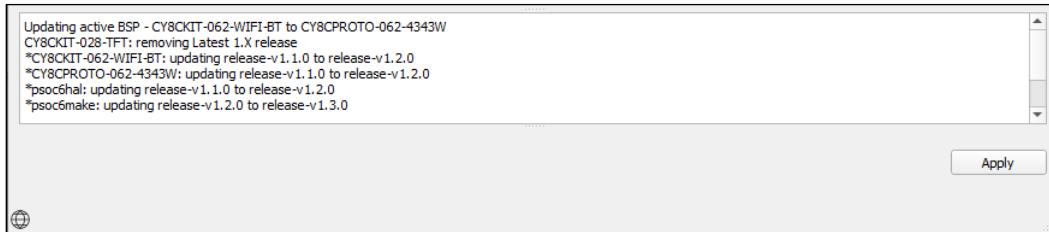
Screenshot of the CY8CPROTO-062-4343W BSP documentation page. The page title is "CY8CPROTO-062-4343W BSP". The left sidebar shows navigation links for the BSP User Guide and API Reference. The main content area displays the "CY8CPROTO-062-4343W BSP" overview, which includes a photograph of the PSoC 6 Wi-Fi BT Prototyping Kit and a note about including a reference to `cybsp.h`. Below the overview, there are sections for "Features" and "Kit Features".

5.9.3 Changing the BSP with the Library Manager

You can use the Library Manager to select different BSPs and different versions of BSPs to include in your application. Select the **Active BSP** to specify which one to use currently.



The console displays the changes to be made. Click **Apply** to update your application.



5.9.4 Selecting a BSP by editing the Makefile

To specify a different BSP by editing the Makefile, update the TARGET value to the desired board.

```
~^
27 #####
28 # Basic Configuration
29 #####
30
31 # Target board/hardware
32 TARGET=CY8CPROTO-062-4343W
```

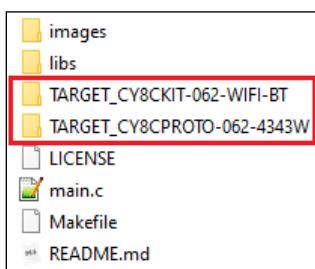
IMPORTANT: This will change the project being built by the IDE, but it will **NOT** update the Launches in the Quick Panel in the Eclipse IDE, so it does not affect which hex file is copied to the kit. Therefore, it is recommended to use the Library Manager's Active BSP selection from inside the Eclipse IDE to change the target board if you are using the IDE to program/debug.

5.9.5 Modifying the BSP Configuration (e.g. design.modus) for a Single Application

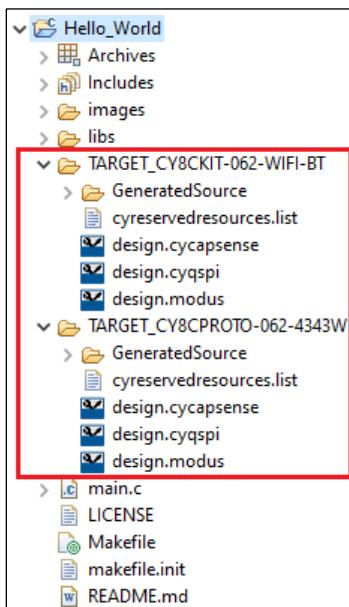
If you want to modify the BSP configuration for a single application (such as different pin or peripheral settings), it is preferable to NOT modify the BSP directly since that results in changes to the BSP library which would prevent you from updating the repository in the future. Instead, there is a mechanism to use a custom set of configuration files in an application. The steps are:

1. Create a directory for each target that you want to support at the top-level directory in your application. The directory name must be TARGET_(board_name). For example, TARGET_CY8CPROTO-062-4343W.

Remember that the build system automatically includes all the source files inside a directory that begins with TARGET_ followed by the target name for compilation when that target is specified in the application make file. The file structure appears as follows. Here, the BSP_DESIGN_MODUS component is overridden for two targets: CY8CPROTO-062-4343W and CY8CKIT-062-WIFI-BT.



2. Copy the design.modus file and other configuration files (i.e. everything from inside the BSP's COMPONENT_BSP_DESIGN_MODUS folder) inside this new directory and customize as required. When you save the changes in the design.modus file, the source files are generated and placed under the GeneratedSource directory. The file structure appears as follows:



3. In the makefile, add the following lines:

```
DISABLE_COMPONENTS += BSP_DESIGN_MODUS
COMPONENTS += CUSTOM_DESIGN_MODUS
```

The first line disables the configuration files from the BSP. The second line is required so that the init_cycfg_all function is still called from the cybsp_init function. The init_cycfg_all function is used to initialize the hardware that was setup in the configuration files.

Note that the makefile already contains blank COMPONENTS and DISABLE_COMPONENTS lines where you can add the appropriate names.

5.9.6 Creating your Own BSP

If you want to change more than just the configuration from the COMPONENT_BSP_DESIGN_MODUS folder (such as for your own custom hardware or for different linker options), you can create a full BSP based on an existing one. To create your own board, do the following:

1. Locate the closest-matching BSP to your custom BSP and set that as the default TARGET for the project in the makefile.
2. Run the `make bsp` target, specifying the new board name by passing the value to the `TARGET_GEN` variable. Optionally, specify the new device (`DEVICE_GEN`) and additional devices (`ADDITIONAL_DEVICES_GEN`). For example:

```
make bsp TARGET_GEN=MyBSP DEVICE_GEN=CY8C624ABZI-D44
ADDITIONAL_DEVICES_GEN=CYW4343WKUBG
```

This will create a new BSP with the provided name at the top of the application project. It will also automatically copy the relevant startup and linker scripts based on the MPN specified by `DEVICE_GEN` into the newly created BSP.

- If there were any issues with the new device's configuration, open the Device Configurator to address.
 - The BSP used as your starting point may have library references (for example, capsense.lib or udb-sdio-whd.lib) that are not needed by your custom BSP. These can be deleted from the BSP.
3. Define or update the alias for pins in the `cybsp_types.h` file.
 4. Customize the `design.modus` file and other configuration files with new settings for clocks, power supplies, and peripherals as required.
 5. Update the `make TARGET` variable to point to your new BSP. If you're starting from a Cypress provided example project, you will find this in the `makefile` file in the root of your project.
 6. If using an IDE, regenerate the configuration settings to reflect the new BSP. Pick the appropriate command(s) for the IDE(s) that are being used. For example: `make vscode`

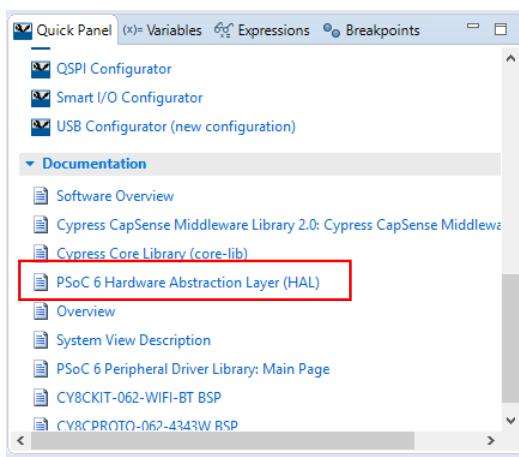
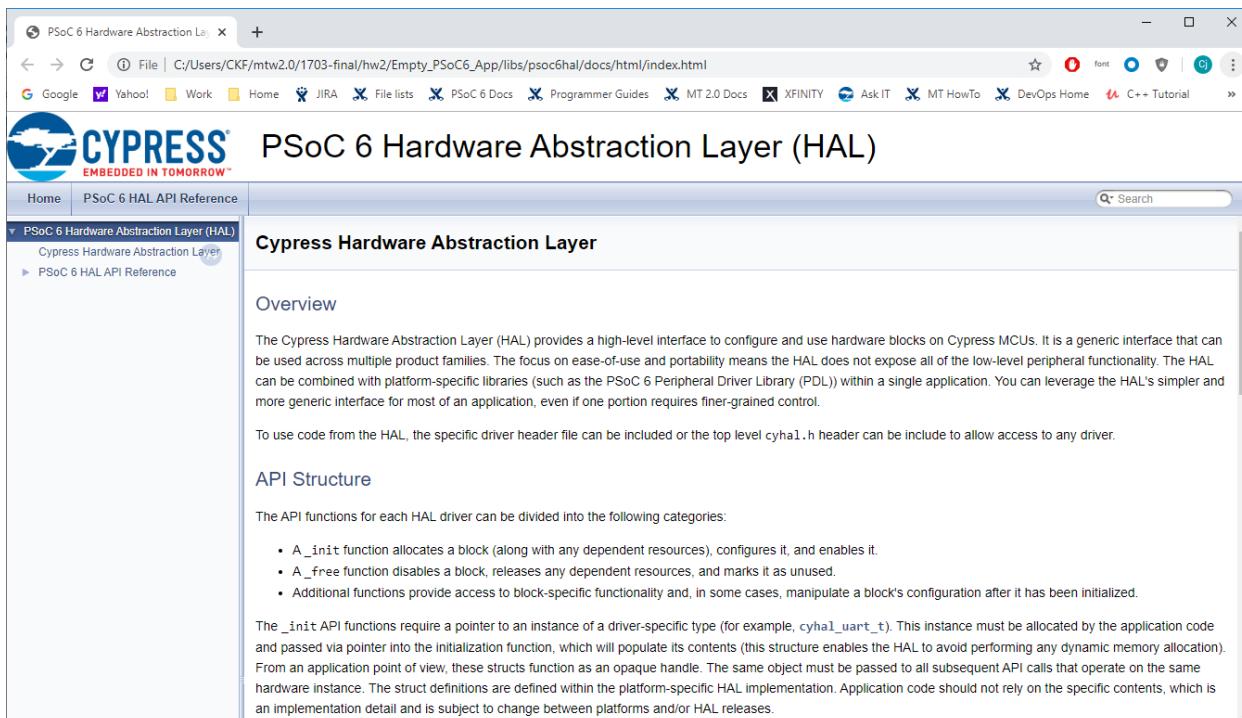
Note: The full list of IDEs is dependent on the make build system being used. Use `make help` to see all supported IDE make targets.

If you want to re-use a custom BSP on multiple applications, you can copy it into each application, or you can put it into a git repo so that you can use it just like any other BSP during application creation.

5.10 HAL

The Cypress Hardware Abstraction Layer (HAL) provides a high-level interface to configure and use hardware blocks on Cypress MCUs. It is a generic interface that can be used across multiple product families. The focus on ease-of-use and portability means the HAL does not expose all of the low-level peripheral functionality. The HAL can be combined with platform-specific libraries (such as the PSoC 6 Peripheral Driver Library (PDL)) within a single application. You can leverage the HAL's simpler and more generic interface for most of an application, even if one portion requires finer-grained control.

After creating a PSoC 6 project, there is a link to the HAL documentation in the Quick Panel under "Documentation."

PSoC 6 Hardware Abstraction Layer (HAL)

Cypress Hardware Abstraction Layer

Overview

The Cypress Hardware Abstraction Layer (HAL) provides a high-level interface to configure and use hardware blocks on Cypress MCUs. It is a generic interface that can be used across multiple product families. The focus on ease-of-use and portability means the HAL does not expose all of the low-level peripheral functionality. The HAL can be combined with platform-specific libraries (such as the PSoC 6 Peripheral Driver Library (PDL)) within a single application. You can leverage the HAL's simpler and more generic interface for most of an application, even if one portion requires finer-grained control.

To use code from the HAL, the specific driver header file can be included or the top level cyhal.h header can be include to allow access to any driver.

API Structure

The API functions for each HAL driver can be divided into the following categories:

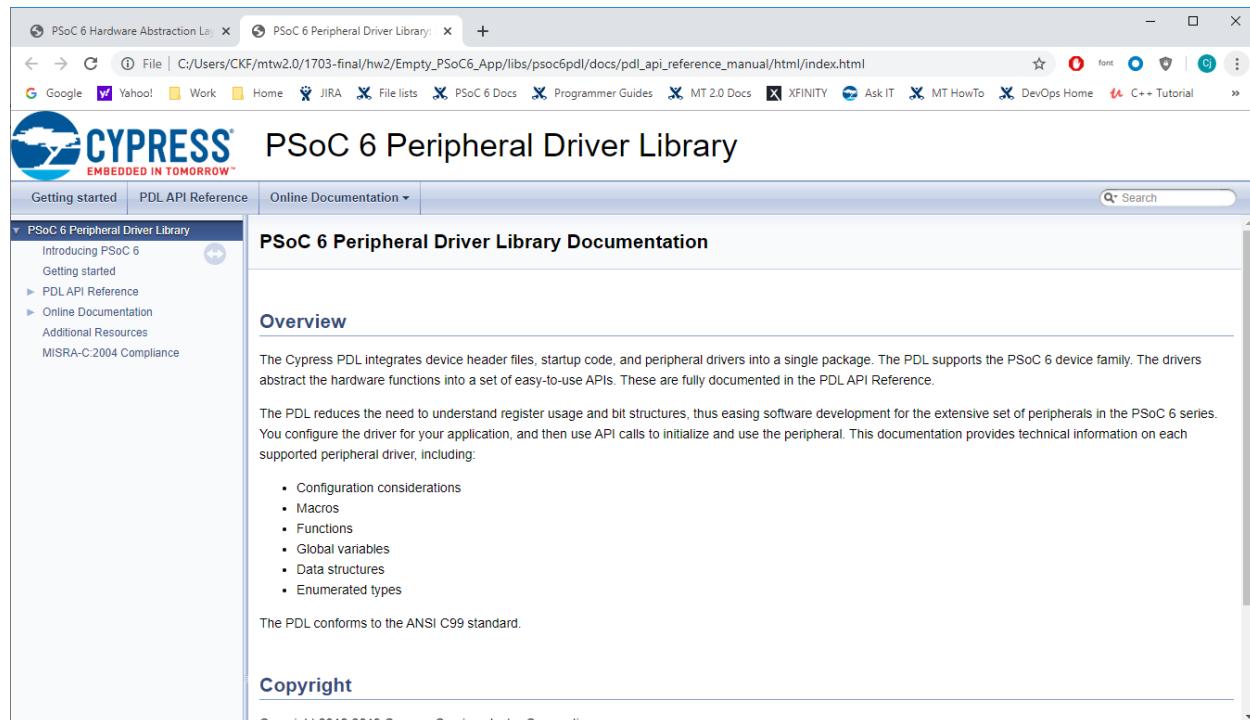
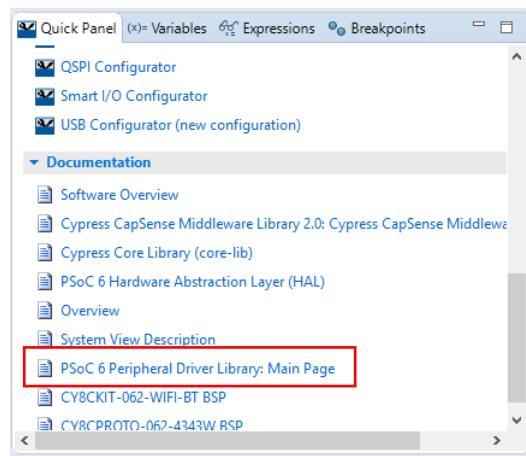
- A _init function allocates a block (along with any dependent resources), configures it, and enables it.
- A _free function disables a block, releases any dependent resources, and marks it as unused.
- Additional functions provide access to block-specific functionality and, in some cases, manipulate a block's configuration after it has been initialized.

The _init API functions require a pointer to an instance of a driver-specific type (for example, cyhal_uart_t). This instance must be allocated by the application code and passed via pointer into the initialization function, which will populate its contents (this structure enables the HAL to avoid performing any dynamic memory allocation). From an application point of view, these structs function as an opaque handle. The same object must be passed to all subsequent API calls that operate on the same hardware instance. The struct definitions are defined within the platform-specific HAL implementation. Application code should not rely on the specific contents, which is an implementation detail and is subject to change between platforms and/or HAL releases.

5.11 PDL

The Peripheral Driver Library (PDL) integrates device header files, startup code, and peripheral drivers into a single package. The PDL reduces the need to understand register usage and bit structures, thus easing software development for the extensive set of peripherals in the PSoC 6 series.

After creating a PSoC 6 project, there is a link to the PDL documentation in the Quick Panel under "Documentation."



The Cypress PDL integrates device header files, startup code, and peripheral drivers into a single package. The PDL supports the PSoC 6 device family. The drivers abstract the hardware functions into a set of easy-to-use APIs. These are fully documented in the PDL API Reference.

The PDL reduces the need to understand register usage and bit structures, thus easing software development for the extensive set of peripherals in the PSoC 6 series. You configure the driver for your application, and then use API calls to initialize and use the peripheral. This documentation provides technical information on each supported peripheral driver, including:

- Configuration considerations
- Macros
- Functions
- Global variables
- Data structures
- Enumerated types

The PDL conforms to the ANSI C99 standard.

Copyright

Copyright 2016-2019 Cypress Semiconductor Corporation

5.12 Manifests

Manifests are XML files that tell the IDE New Application wizard, Project Creator and Library Manager how to discover the list of available boards, example projects, and libraries. There are several manifest files.

- The "super manifest" file contains a list of URLs that software uses to find the board, code example, and middleware manifest files.
- The "app-manifest" file contains a list of all code examples that should be made available to the user.
- The "board-manifest" file contains a list of the boards that should be presented to the user in the new project creation wizards as well as the list of BSP packages that are presented in the Library Manager tool.
- The "middleware-manifest" file contains a list of the available middleware (libraries). Each middleware item can have one or more versions of that middleware available.

To use your own examples, BSPs, and middleware, you need to create manifest files for your content and a super-manifest that points to your manifest files.

Once you have the files created an in a Git repo, place a manifest.loc file in your <user_home>/.modustoolbox directory that specifies the location of your custom super-manifest file (which in turn points to your custom manifest files). For example, a manifest.loc file may have:

```
# This points to the IOT Expert template set
https://github.com/iotexpert/mtb2-iotexpert-manifests/raw/master/iotexpert-super-
manifest.xml
```

To see examples of the syntax of super-manifest and manifest files, you can look at the Cypress provided files on GitHub:

Super Manifest: <https://github.com/cypresssemiconductorco/mtb-super-manifest>

App Manifest: <https://github.com/cypresssemiconductorco/mtb-ce-manifest>

Board Manifest: <https://github.com/cypresssemiconductorco/mtb-bsp-manifest>

Middleware Manifest: <https://github.com/cypresssemiconductorco/mtb-mw-manifest>

5.13 Cypress Configurators

ModusToolbox software provides graphical applications called configurators 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 configurator to 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.

Many configurators do not apply to all types of projects. So, the available configurators depend on the project/application you have selected in the Project Explorer. Configurators are independent of each other, but they can be used together to provide flexible configuration options. They can be used stand alone, in conjunction with other configurators, or within a complete IDE. Everything is bundled together as part of the installation. Each configurator provides a separate guide, available from the configurator's Help menu. Configurators perform tasks such as:

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

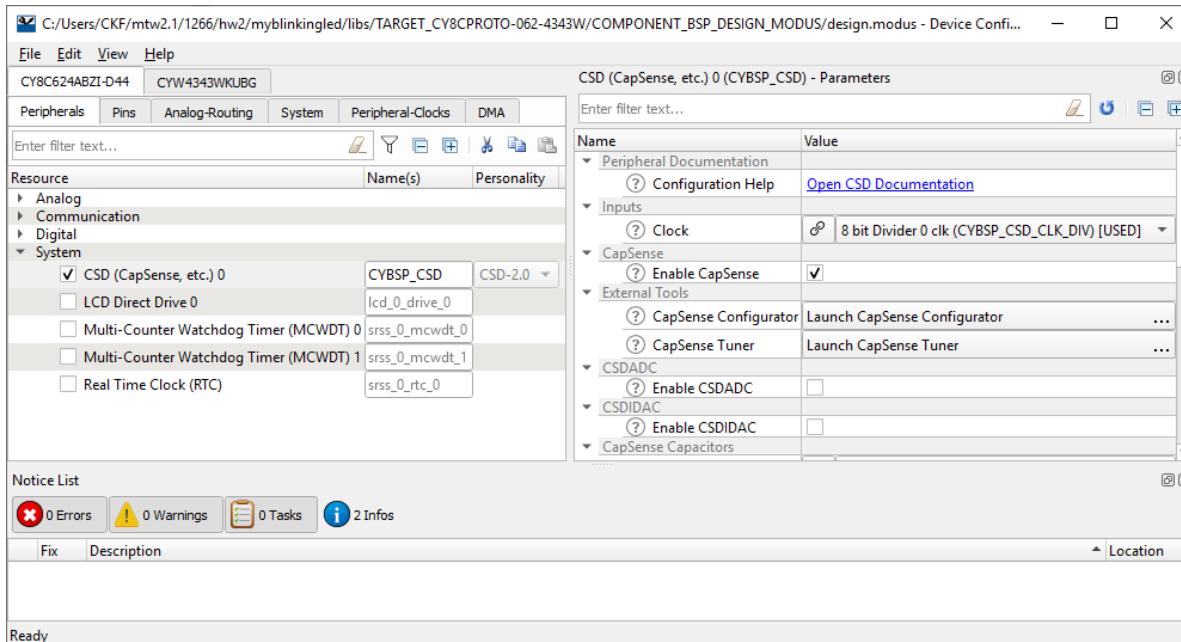
Configurators are divided into two types: Board Support Package (BSP) Configurators and Library Configurators.

5.13.1 BSP Configurators

BSP Configurators are closely related to the hardware resources on the board such as CapSense sensors, external Flash memory, etc. These are typically provided inside the BSP hierarchy, although they can be customized for a given application as was described earlier.

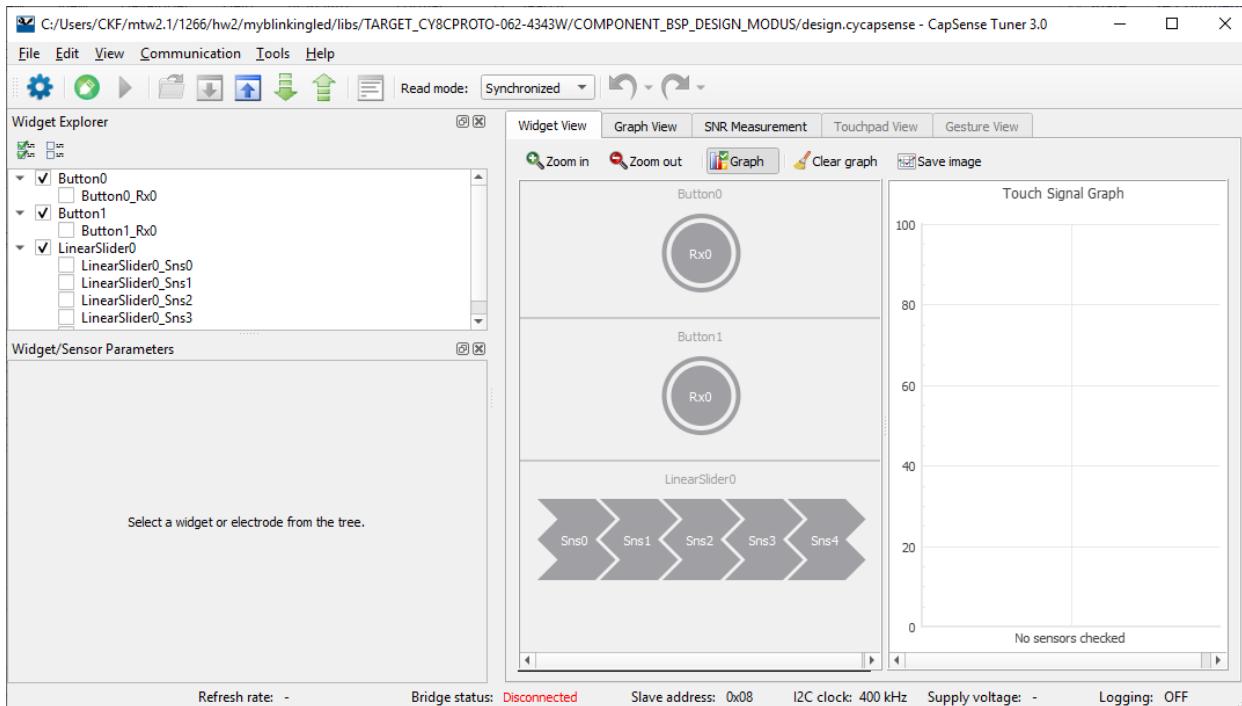
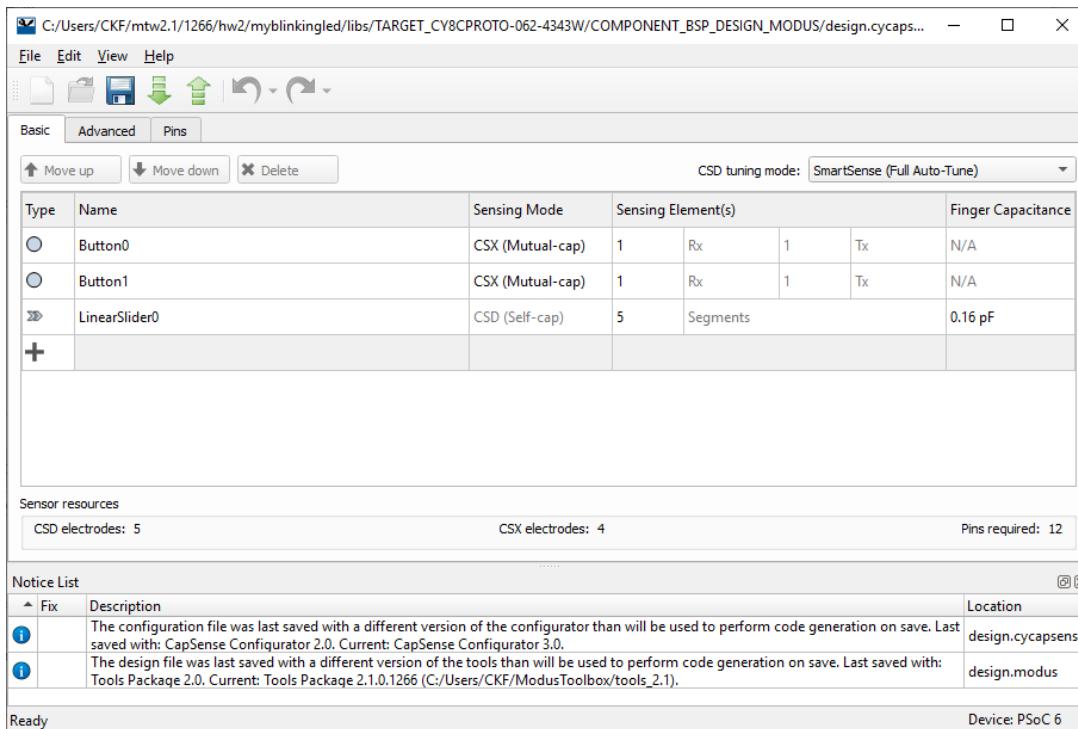
Device Configurator

Set up the system (platform) functions such as pins, interrupts, clocks, and DMA, as well as the basic peripherals, including UART, Timer, etc.



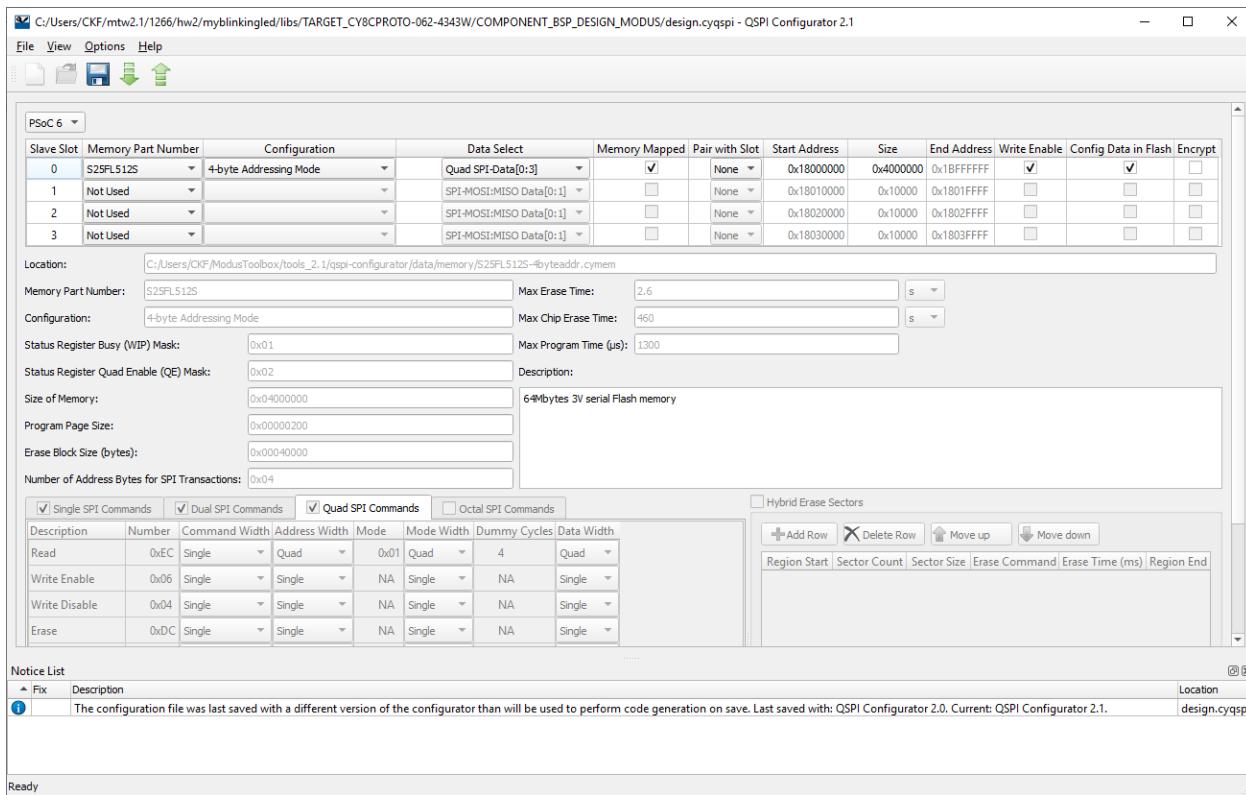
CapSense Configurator/Tuner

Configure CapSense hardware, and generate the required firmware. This includes tasks such as mapping pins to sensors and how the sensors are scanned.



QSPI Configurator

Configure external memory and generate the required firmware. This includes defining and configuring what external memories are being communicated with.



The screenshot shows the QSPI Configurator 2.1 interface. At the top, there's a menu bar with File, View, Options, Help, and a toolbar with icons for Open, Save, Print, and Export. The main window displays a table for Slave Slot Configuration:

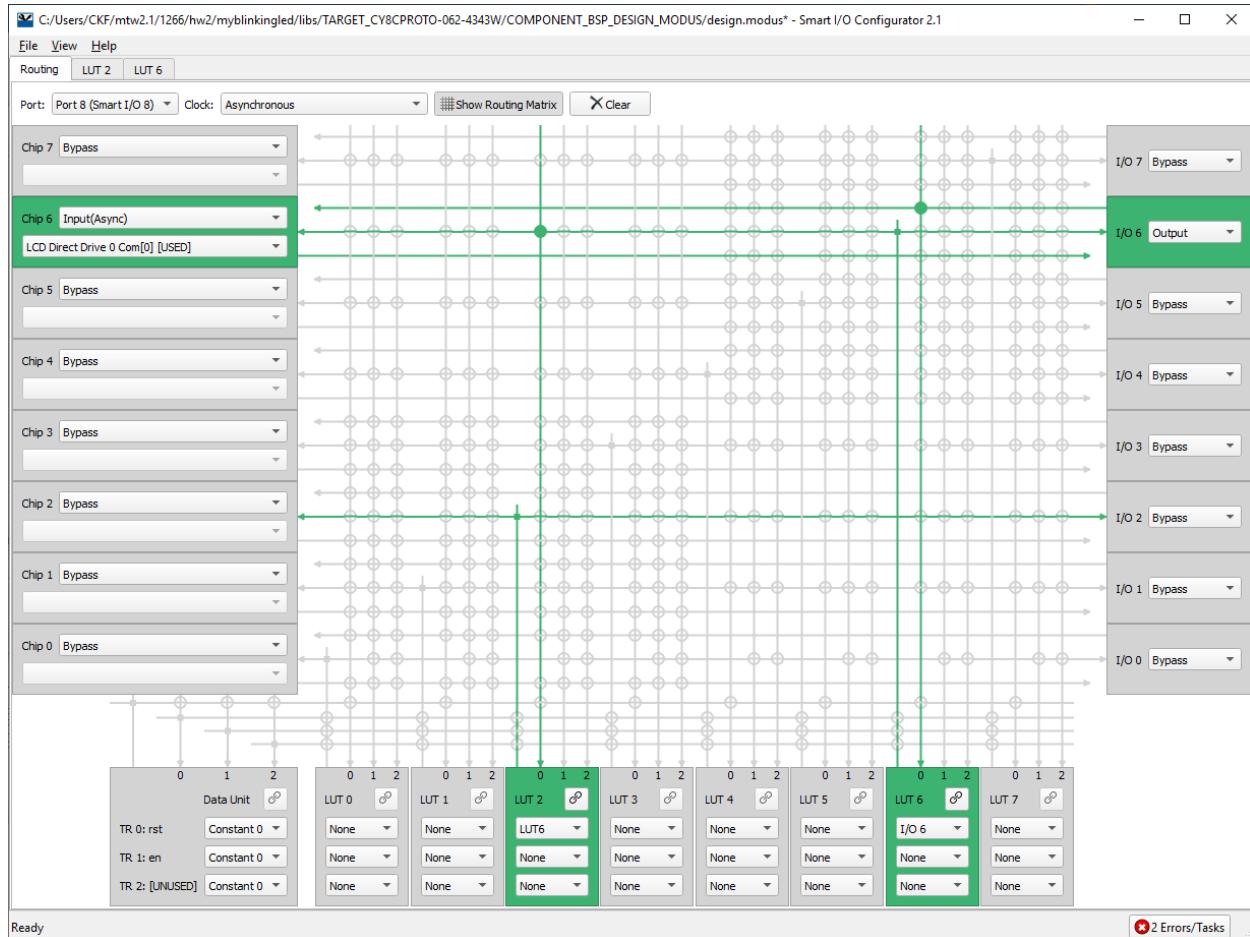
| Slave Slot | Memory Part Number | Configuration | Data Select | Memory Mapped | Pair with Slot | Start Address | Size | End Address | Write Enable | Config Data in Flash | Encrypt |
|------------|--------------------|------------------------|-------------------------|-------------------------------------|----------------|---------------|-----------|-------------|-------------------------------------|-------------------------------------|--------------------------|
| 0 | S25FL512S | 4-byte Addressing Mode | Quad SPI-Data[0:3] | <input checked="" type="checkbox"/> | None | 0x18000000 | 0x4000000 | 0x1BFFFFF | <input checked="" type="checkbox"/> | <input checked="" type="checkbox"/> | <input type="checkbox"/> |
| 1 | Not Used | | SPI-MOSI:MISO Data[0:1] | <input type="checkbox"/> | None | 0x18010000 | 0x10000 | 0x1801FFFF | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |
| 2 | Not Used | | SPI-MOSI:MISO Data[0:1] | <input type="checkbox"/> | None | 0x18020000 | 0x10000 | 0x1802FFFF | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |
| 3 | Not Used | | SPI-MOSI:MISO Data[0:1] | <input type="checkbox"/> | None | 0x18030000 | 0x10000 | 0x1803FFFF | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |

Below the table, there are fields for Location (C:\Users\CKF\ModusToolbox\tools_2.1\qspi-configurator\data\memory\S25FL512S-4byteaddr.cymem), Memory Part Number (S25FL512S), Configuration (4-byte Addressing Mode), and various timing parameters (Max Erase Time: 2.6s, Max Chip Erase Time: 460s, Max Program Time (μs): 1300). There are also fields for Status Register Busy (WIP) Mask (0x01), Status Register Quad Enable (QE) Mask (0x02), Size of Memory (0x04000000), Program Page Size (0x00000200), Erase Block Size (bytes) (0x000040000), and Number of Address Bytes for SPI Transactions (0x04).

At the bottom left, there's a "Notice List" section with a fix for a configuration file saved with a different version. The fix description is: "The configuration file was last saved with a different version of the configurator than will be used to perform code generation on save. Last saved with: QSPI Configurator 2.0. Current: QSPI Configurator 2.1." The location is listed as design.qspi. The status is "Ready".

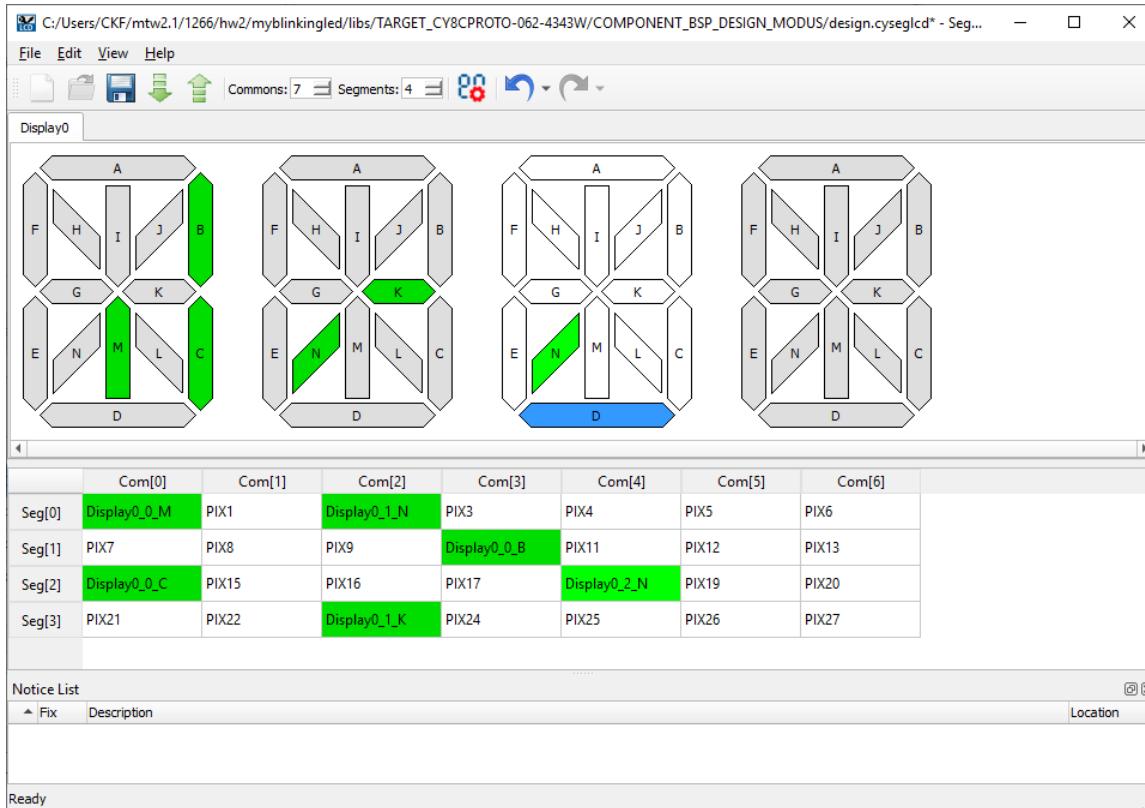
Smart I/O™ Configurator

Configure the Smart I/O block, which adds programmable logic to an I/O port.



SegLCD Configurator

Configure LCD displays. This configuration defines a matrix Seg LCD connection and allows you to setup the connections and easily write to the display.

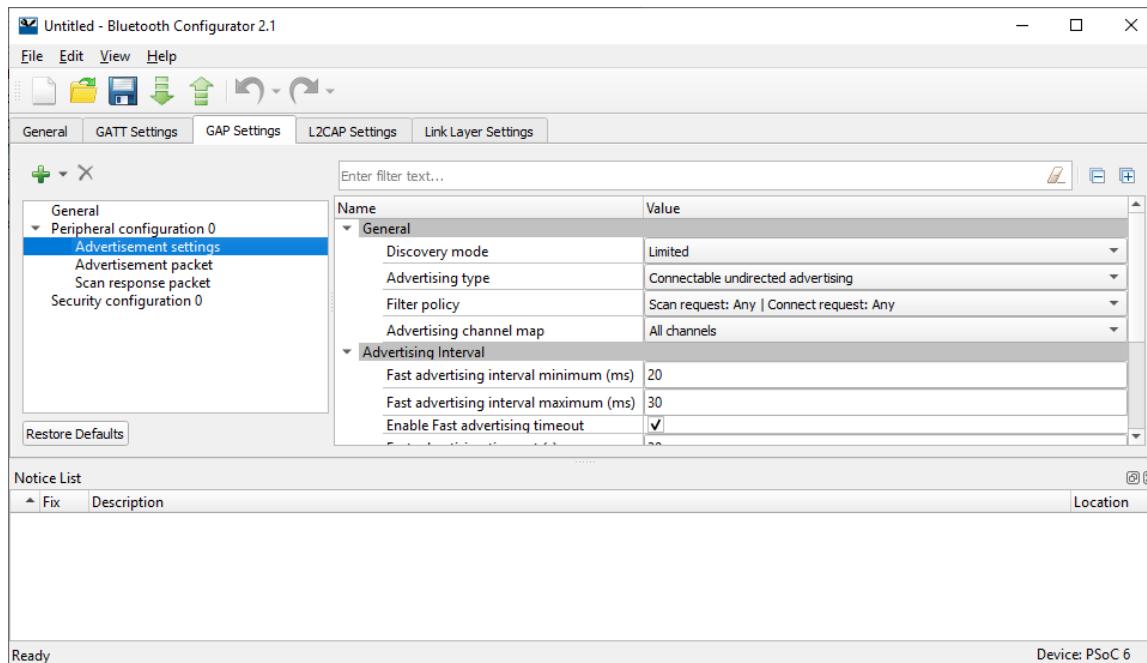


5.13.2 Library configurators

Library Configurators are used to setup and configure middleware libraries. The files for these are contained in the project hierarchy rather than inside the BSP.

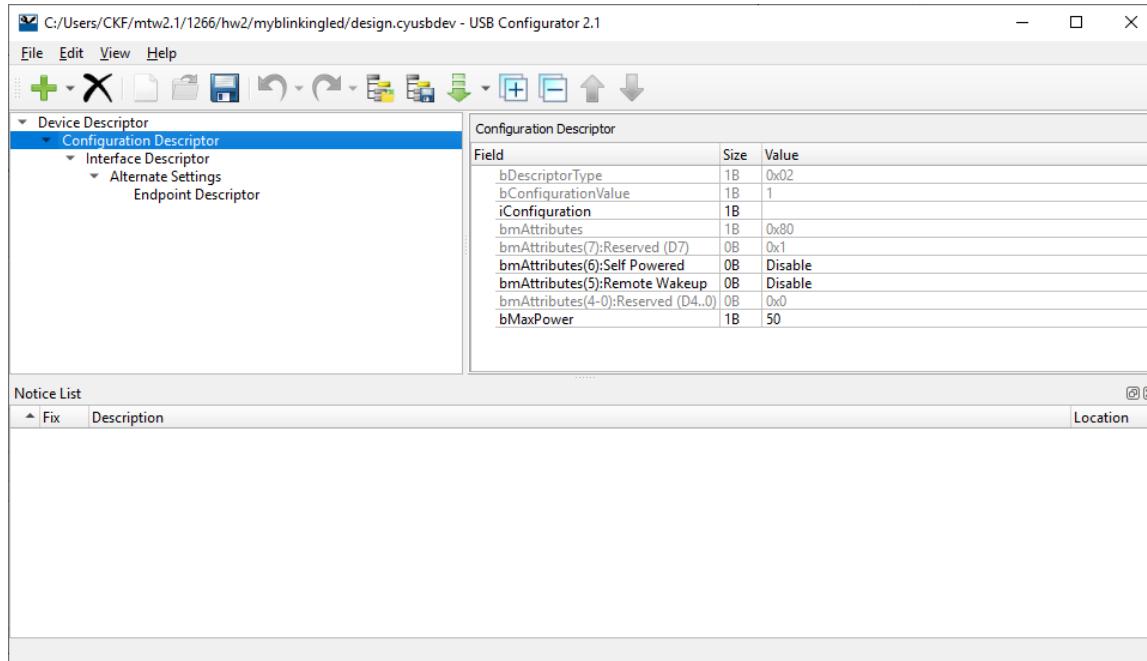
Bluetooth Configurator

Configure Bluetooth settings. This includes options for specifying what services and profiles to use and what features to offer by creating SDP and/or GATT databases in generated code. This configurator supports both PSoC MCU and WICED Bluetooth applications.



USB Configurator

Configure USB settings and generate the required firmware. This includes options for defining the 'Device' Descriptor and Settings.



5.14 Exercises (Part 2)

All of the following exercises can be completed from the command line or within the IDE. Feel free to use the method that feels most natural, interesting, educational, challenging (or easy) – it's your choice! You can also change your mind half-way through if you wish to practice using both approaches.

5.14.1 Modify the blinking LED to use the Green LED



1. Open the main.c file from your previous project in an editor.

2. Locate this line of code:

```
cyhal_gpio_init(CYBSP_USER_LED1, CYHAL_GPIO_DIR_OUTPUT, CYHAL_GPIO_DRIVE_STRONG,  
G, CYBSP_LED_STATE_OFF);
```



3. Change the LED that you want to blink from CYBSP_USER_LED1 to CYBSP_USER_LED4, which is the green LED in the RGB LED.



4. Locate this line of code and make the same change so that the LED will blink:

```
cyhal_gpio_toggle(CYBSP_USER_LED1);
```



5. Build the project and program the board.

5.14.2 Modify the project to Blink Green/Red using the HAL



1. Make a copy of the line of code that initializes the CYBSP_USER_LED4.
2. Change the LED to CYBSP_USER_LED3 so you are initializing the red LED from the RGB LED as well as the green LED.
3. Change the final argument to cyhal_gpio_init() so that CYBSP_USER_LED3 is initialized in the ON state while CYBSP_USER_LED4 is OFF.
4. Make a copy of the line of code that toggles the green LED and edit it to toggle the red LED.
5. Build the project program the board.

5.14.3 Print a message using the retarget-io library



1. Create a new project for the CY8CKIT-062-WIFI-BT kit with the “Empty PSoC6 App” template.

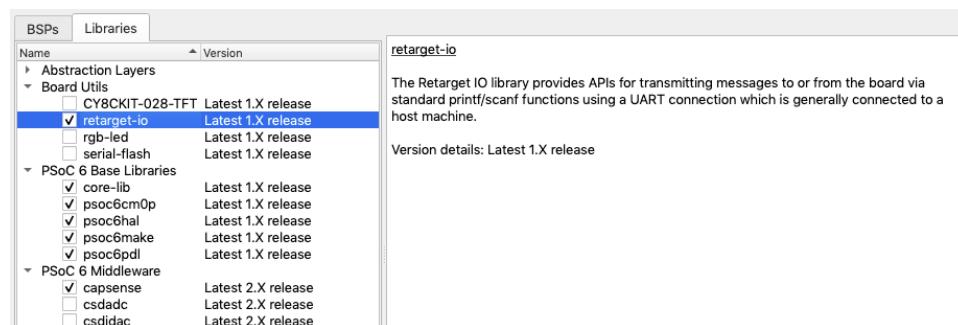
Note: If you are working from the command line, you will need to add the BSP for your kit. The following commands create the project, add the BSP, build and program the kit.

```
$ git clone https://github.com/cypresssemiconductorco/mtb-example-psoc6-empty-app
$ cd mtb-example-psoc6-empty-app
$ git clone https://github.com/cypresssemiconductorco/TARGET_CY8CKIT-062-WIFI-BT
$ make getlibs
$ make program TARGET=CY8CKIT-062-WIFI-BT
```

Note: if you prefer, instead of specifying the BSP in the last command, you can edit the Makefile to change the kit with TARGET=CY8CKIT-062-WIFI-BT and use the shorter form of the command: make program.



2. Launch the Library Manager.
 - In the IDE you can do this from the Quick Panel (Tools) or use the right-mouse menu on the project root node and go to ModusToolbox > Library Manager.
 - From the command line launch library-manager.exe from ModusToolbox\tools_2.1\library-manager and use the Browse... button to point to your project directory.
3. Observe the available BSPs, then switch to the **Libraries** tab.
4. Find the retarget-io library, check the box, and click **Apply** to add the library to your application.



5. Back in the IDE, open main.c and add the following include file to tell the compiler you wish to use the retarget-io functions:

```
#include "cy_retarget_io.h"
```

6. Add the following include file to tell the compiler you wish to use the printf() function.

```
#include <stdio.h>
```

7. In the main() function, add the following code to initialize the UART and associate with the retarget-io library.

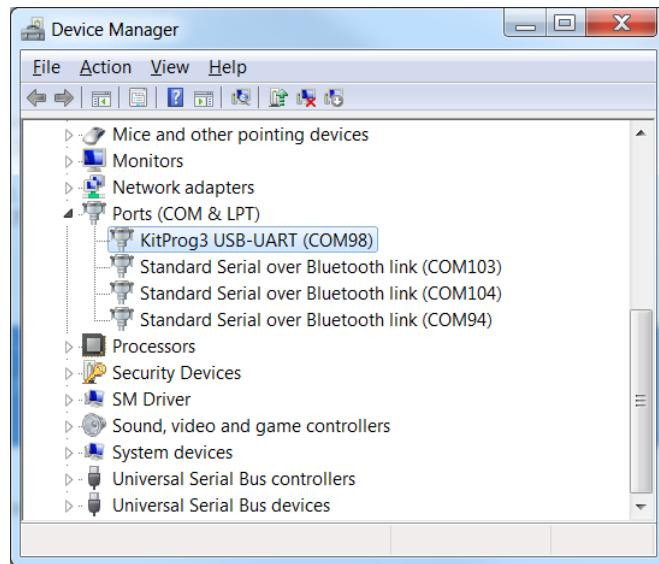
```
result = cy_retarget_io_init(CYBSP_DEBUG_UART_TX, CYBSP_DEBUG_UART_RX, \
CY_RETARGET_IO_BAUDRATE);
CY_ASSERT(result == CY_RSLT_SUCCESS);
```

8. Before the forever loop, use printf() to write “PSoC Rocks!\r\n” to the UART.

9. Build the project and program the board.

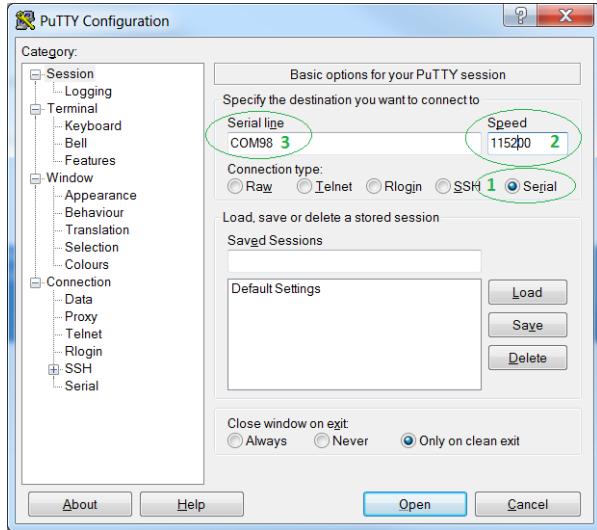
10. From the Windows Search, type “device manager” and launch that utility.

11. Look in the “Ports (COM&LPT)” directory for your board’s KitProg3. Note the COM port number.



12. From Windows, launch the PuTTY application (or your favorite terminal emulator).

- 13. Specify a Serial session, at a Speed of 115200 (baud rate), using the Serial Line from the device Manager (COM PORT).



- 14. Observe the output from your kit when you reset the kit.

5.14.4 Control the RGB LED with a Library

- 1. Return to the Library Manager and follow the same process to add the `rgb-led` library to your project.
□ 2. In the IDE, at the top of `main.c`, and add the following include file to tell the compiler you wish to use the `rgb_led` functions:

```
#include "cy_rgb_led.h"
```

- 3. Add this macro – it translates the name of a `#define` into a string:

```
#define DEFINE_TO_STRING(macro) (#macro)
```

- 4. In the main function, add this code to initialize the RGB LED. It defines the LED pins (defined for you in the BSP) and sets the polarity:

```
result = cy_rgb_led_init(CYBSP_LED_RGB_RED, CYBSP_LED_RGB_GREEN,  
CYBSP_LED_RGB_BLUE, CY_RGB_LED_ACTIVE_LOW);  
  
if (result != CY_RSLT_SUCCESS)  
{  
CY_ASSERT(0);  
}
```

- 5. In the forever loop add this code to make the LED yellow for a second.

```
printf("Color is %s\r\n", DEFINE_TO_STRING(CY_RGB_LED_COLOR_YELLOW));  
cy_rgb_led_on(CY_RGB_LED_COLOR_YELLOW, CY_RGB_LED_MAX_BRIGHTNESS);  
Cy_SysLib_Delay(1000);
```

- 6. Make copies of those three lines and turn the LED PURPLE and CYAN.
□ 7. Build the project program the board. Observe the printed output in the terminal and the LED behavior.

5.14.5 CapSense Buttons and Slider

Build the example code

1. Create the CapSense Starter application using the IDE or command line.

Note: If you are working from the command line you will need to add the BSP for your kit.

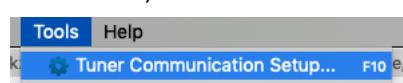
2. Build the project and program the board.

Run the CapSense Tuner

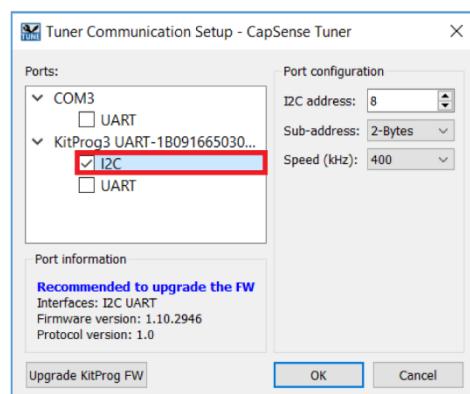
1. Launch the CapSense Tuner through the IDE or in stand-alone mode. Note that, in the latter case, you will need to manually open the CapSense configuration file:

`libs/TARGET_CY8CKIT-062-WIFI-BT/COMPONENT_BSP_DESIGN_MODUS/design.cycapsense`

2. In the Tuner, click **Tools > Tuner Communication Setup** or press [F10].



3. In the dialog, select I2C under KitProg and configure as follows:



4. Check the parameters and close the dialog.

- I2C Address: 8
- Sub-address: 2-Bytes
- Speed (kHz): 400

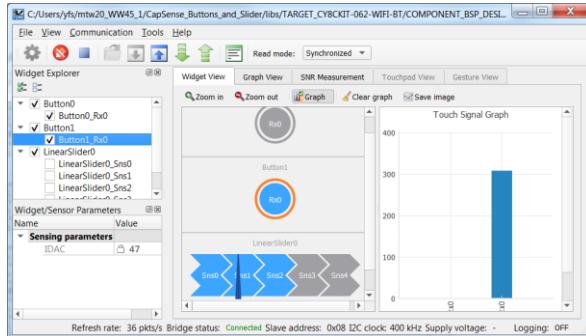
5. In the Tuner, click the **Connect** button to establish communication with the target (F4).

Note: If you have a terminal emulator connected to the serial port this can interfere with the tuner connection so close that application if you have connection difficulties.

6. Click the **Start** button to begin collecting tuning data (F5).

-  7. Verify that tuning is operational by touching widgets and observing the data in the Widget and Graph view.

Note: You must enable each sensor in the Widget Explorer window to use the graph view.



-  8. Stop collecting data (Shift+F5).
9. Close the connection (Shift+F4).

5.14.6 Write a message to the TFT shield

This project displays a message on the CY8CKIT-028-TFT shield, which is attached to the CY8CKIT-062-WIFI-BT board.

-  1. Create an Empty_PSoC6_App project for the CY8CKIT-062-WIFI-BT kit.
Note: If you are working from the command line, you will need to add the BSP for your kit.
-  2. Using the Library Manager, add the CY8CKIT-028-TFT library from **Board Utils** and the emwin library from **PSoC 6 Middleware**.
3. Click **Apply** and close the Library Manager.
4. Open the project Makefile to add emwin configuration for bare metal without touchscreen:

```
COMPONENTS+=EMWIN_NOSNTS
```



5. In the main.c file, change the code to match the following:

```
#include "cy_pdl.h"
#include "cyhal.h"
#include "cybsp.h"
#include "GUI.h"

int main(void)
{
    cy_rslt_t result;

    /* Initialize the device and board peripherals */
    result = cybsp_init();
    if (result != CY_RSLT_SUCCESS)
    {
        CY_ASSERT(0);
    }

    __enable_irq();

    GUI_Init();
    GUI_SetColor(GUI_WHITE);
    GUI_SetBkColor(GUI_BLACK);
    GUI_SetFont(GUI_FONT_32B_1);
    GUI_SetTextAlign(GUI_TA_CENTER);
    /* Change this text as appropriate */
    GUI_DispStringAt("I feel good!",
        GUI_GetScreenSizeX()/2, GUI_GetScreenSizeY()/2 - GUI_GetFontSizeY()/2);

    for(;;)
    {
    }
}
```



6. Build and program your project.

Observe the message displayed on TFT.



5.14.7 Use an RTOS to Blink LEDs

1. Create another new project for the CY8CKIT-062-WIFI-BT kit with the “Empty PSoC6 App” template.

Note: If you are working from the command line you will need to add the BSP for your kit.

2. Use the Library Manager to add the freertos library.
 3. Close the Library Manager.
 4. Back in the IDE, open main.c add the FreeRTOS headers.

```
#include "FreeRTOS.h"  
#include "task.h"
```

Note: FreeRTOS.h is needed in all files that make RTOS calls and tasks.h, mutex.h, semphr.h, and so on are used when code accesses resources of that type.

5. Write a function to blink an LED that will be called by the RTOS when the task starts.

Note: do not use Cy_SysLib_Delay() in a task because it will not do what you want - use vTaskDelay() instead.

```
void LED1_Task(void *arg)  
{  
    cyhal_gpio_init(CYBSP_USER_LED1, CYHAL_GPIO_DIR_OUTPUT,  
                    CYHAL_GPIO_DRIVE_STRONG, CYBSP_LED_STATE_OFF);  
    for(;;)  
    {  
        cyhal_gpio_toggle(CYBSP_USER_LED1);  
        vTaskDelay(50);  
    }  
}
```

6. In main() create the task and start the task scheduler.

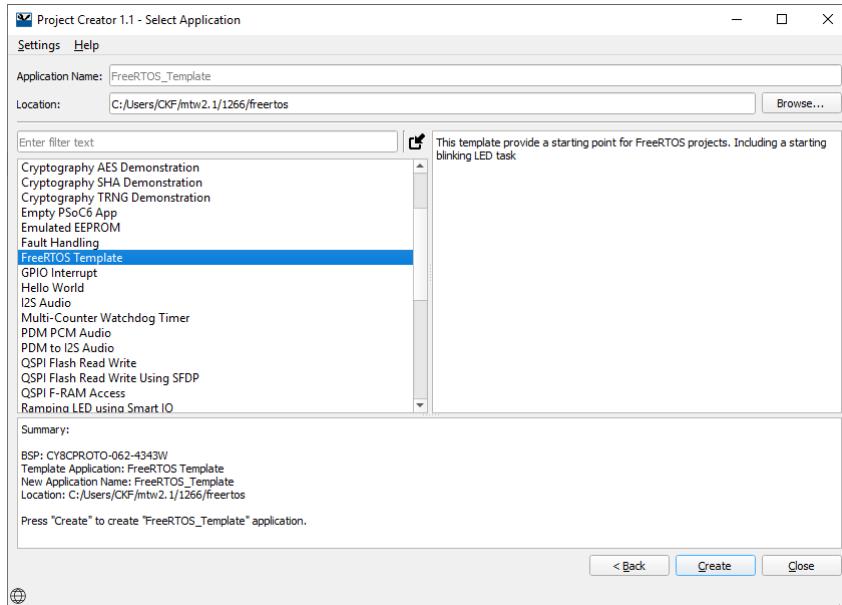
```
xTaskCreate(LED1_Task, "LED1", configMINIMAL_STACK_SIZE, 0 /* args */ , 0 /*  
priority */, NULL /* handle */);  
vTaskStartScheduler();
```

7. Build the project and program the kit. Observe the LED blinking.
 8. Make a copy of the LED1_Task for LED2 and change the delay value so the LEDs blink asynchronously.
 9. In main() create this task with the same priority and stack size.
 10. Build the program and program the kit. Observe the LEDs blinking at different rates.

5.14.8 Use the IoT Expert FreeRTOS Template

There is a set of manifest files located at: <https://github.com/iotexpert/mtb2-iotexpert-manifests>. Use these manifest files to create the IoT Expert FreeRTOS Template project.

- 1. Copy just the manifest.loc file to your home directory in `~/.modustoolbox`.
- 2. Create a new project using the “FreeRTOS Template” that gets picked up by the Project Creator from the iotexpert manifests.



- 3. Build and Program.
- 4. What does the project do?
- 5. How many .lib files are in the project? What are they?
- 6. Make the LED blink faster.
- 7. Add the ntshell library (it is in the iotexpert directory).
- 8. Follow the instructions in the libs/middleware-ntshell/README.md file to create the ntshell task.
Look in the section entitled ‘How to add to the PSoC6 SDK’.
Note: There is a template directory in the library with default usrcmd.* files.
- 9. Build and Program.
- 10. Open a terminal emulator to test the shell.

5.14.9 Create and Use a Library

For this project, create a library and add a .lib that uses the HAL to instantiate the PWM and change the brightness of an LED.



1. In a browser, go to <https://www.github.com/>
2. Follow the instructions to create an account.
3. Click on the Repositories tab and click the green New button to create a repo.



Overview **Repositories 0** Projects 0 Packages 0 Stars 0 Followers 0 Following 0

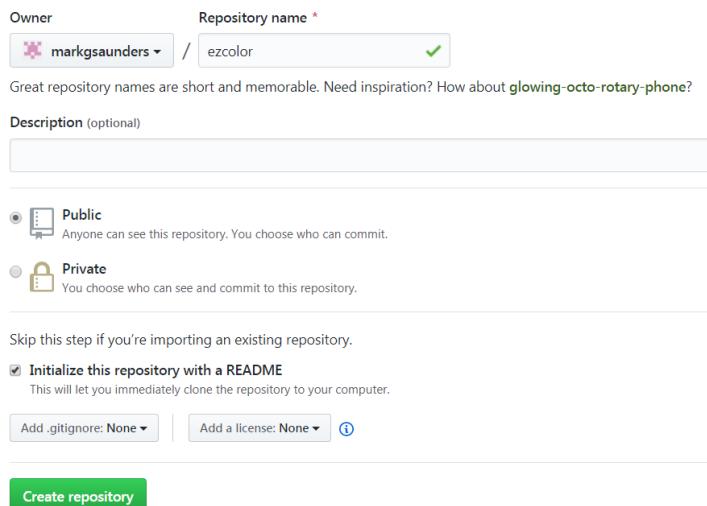
Find a repository... Type: All ▾ Language: All ▾ **New**



4. Give your repo a name and description. Make it public and let it create a README.md file. Press the Create Repository button.

Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere?
[Import a repository](#).



Owner Repository name *

markgsaunders / ezcolor ✓

Great repository names are short and memorable. Need inspiration? How about [glowing-octo-rotary-phone](#)?

Description (optional)

Public Anyone can see this repository. You choose who can commit.
Private You choose who can see and commit to this repository.

Skip this step if you're importing an existing repository.

Initialize this repository with a README
This will let you immediately clone the repository to your computer.

Add .gitignore: None Add a license: None

Create repository



5. Press the Create new file button
6. Name it "ezcolor.h"



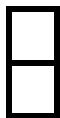
7. Add these lines into the editor.

```
#include "cybsp.h"
#include "cy_rgb_led.h"
typedef enum
{
    RED      = CY_RGB_LED_COLOR_RED,
    GREEN   = CY_RGB_LED_COLOR_GREEN,
    BLUE    = CY_RGB_LED_COLOR_BLUE,
    YELLOW  = CY_RGB_LED_COLOR_RED|CY_RGB_LED_COLOR_GREEN,
    CYAN    = CY_RGB_LED_COLOR_BLUE|CY_RGB_LED_COLOR_GREEN,
    MAGENTA = CY_RGB_LED_COLOR_RED|CY_RGB_LED_COLOR_BLUE,
    WHITE   =
    CY_RGB_LED_COLOR_RED|CY_RGB_LED_COLOR_GREEN|CY_RGB_LED_COLOR_BLUE,
    BLACK   = (~WHITE)&WHITE
} color_t;
cy_rslt_t init_colors( void );
void color( color_t color );
```

ezcolor / ezcolor.h Cancel



```
1 #include "cybsp.h"
2 #include "cy_rgb_led.h"
3
4
5 typedef enum
6 {
7     RED      = CY_RGB_LED_COLOR_RED,
8     GREEN   = CY_RGB_LED_COLOR_GREEN,
9     BLUE    = CY_RGB_LED_COLOR_BLUE,
10    YELLOW  = CY_RGB_LED_COLOR_RED|CY_RGB_LED_COLOR_GREEN,
11    CYAN    = CY_RGB_LED_COLOR_BLUE|CY_RGB_LED_COLOR_GREEN,
12    MAGENTA = CY_RGB_LED_COLOR_RED|CY_RGB_LED_COLOR_BLUE,
13    WHITE   = CY_RGB_LED_COLOR_RED|CY_RGB_LED_COLOR_GREEN|CY_RGB_LED_COLOR_BLUE,
14    BLACK   = (~WHITE)&WHITE
15 } color_t;
16
17 cy_rslt_t init_colors( void );
18 void color( color_t color );
```



8. Press the Commit new file button.

9. Repeat the process to create “ezcolor.c”. Paste this code into the editor.

```
#include "ezcolor.h"
cy_rslt_t init_colors( void )
{
    return cy_rgb_led_init( CYBSP_LED_RGB_RED, CYBSP_LED_RGB_GREEN,
CYBSP_LED_RGB_BLUE, CY_RGB_LED_ACTIVE_LOW );
}
void color( color_t color )
{
    cy_rgb_led_on( color, CY_RGB_LED_MAX_BRIGHTNESS );
}
```



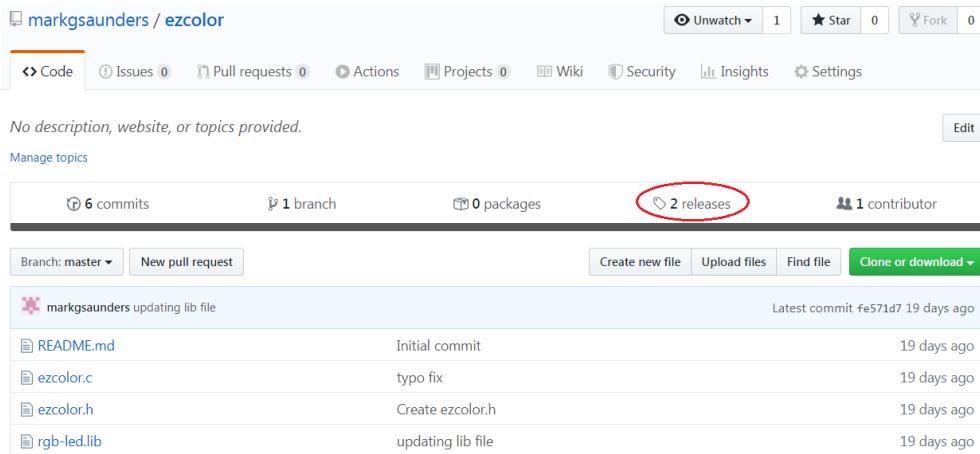
10. Repeat the process a final time to create “rgb-led.lib” with the following content:

<https://github.com/cypresssemiconductorco/rgb-led/#latest-v1.X>

This lib file will cause the `make getlibs` command to automatically pull in the required `rgb-led` library along with `ez-color`.



11. Your repo should now contain four files; the readme, a header, a source file, and the lib file.

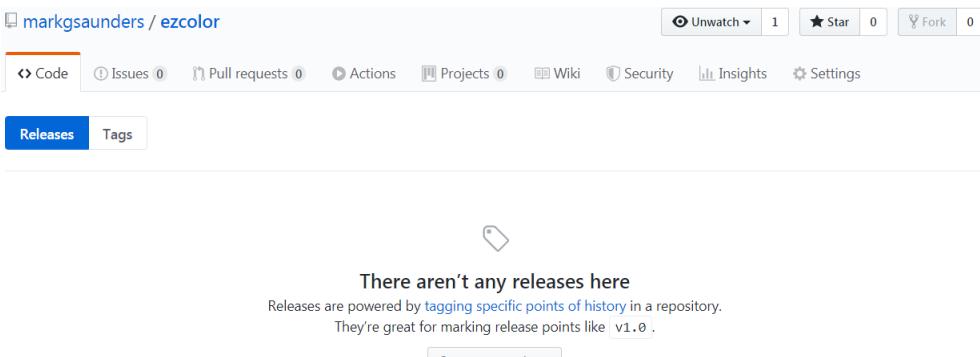


The screenshot shows a GitHub repository page for 'markgsaunders / ezcolor'. At the top, there are buttons for Unwatch (1), Star (0), Fork (0), and Edit. Below the header are tabs for Code, Issues (0), Pull requests (0), Actions, Projects (0), Wiki, Security, Insights, and Settings. A note says 'No description, website, or topics provided.' and there is a 'Manage topics' link. Below this, a summary bar shows 6 commits, 1 branch, 0 packages, and 2 releases (which is circled in red). It also shows 1 contributor. Underneath is a table of commits:

| File | Description | Time |
|-------------|-------------------|-------------|
| README.md | Initial commit | 19 days ago |
| ezcolor.c | typo fix | 19 days ago |
| ezcolor.h | Create ezcolor.h | 19 days ago |
| rgb-led.lib | updating lib file | 19 days ago |



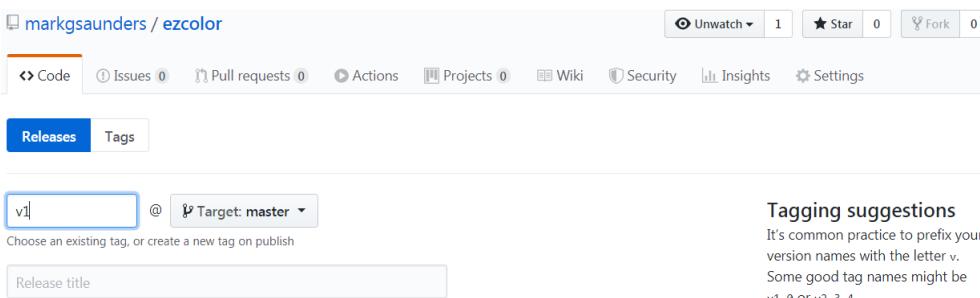
12. Next you need to tag your repo in a release so click on the Releases tab as shown.



The screenshot shows the same GitHub repository page for 'markgsaunders / ezcolor'. The 'Releases' tab is now active and highlighted in blue. Below it is a 'Tags' tab. The main content area displays a message: 'There aren't any releases here'. It includes a note that releases are powered by tagging specific points of history and provides an example of 'v1.0'. There is a 'Create a new release' button.



13. Press the Create a new release button. Give your release a version string and Publish it.



The screenshot shows the GitHub repository page for 'markgsaunders / ezcolor' with the 'Releases' tab active. A search bar at the top has 'v1' typed into it. To its right is a dropdown menu set to 'Target: master'. Below the search bar is a note: 'Choose an existing tag, or create a new tag on publish.' A 'Release title' input field is empty. On the right side, there is a 'Tagging suggestions' section with the following text: 'It's common practice to prefix your version names with the letter v. Some good tag names might be v1.0 or v2.3.4.'



14. Your library is now ready to use. Open a Modus Shell by running this batch file.

~/ModusToolbox/tools_2.1/modus-shell/Cygwin.bat



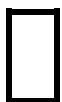
15. In the shell create and cd into a suitable directory for your project.

**16. Create a new project by cloning the Cypress empty app example.**

```
$ git clone https://github.com/cypresssemiconductorco/mtb-example-psoc6-empty-app
Cloning into 'mtb-example-psoc6-empty-app'...
remote: Enumerating objects: 13, done.
remote: Counting objects: 100% (13/13), done.
remote: Compressing objects: 100% (10/10), done.
remote: Total 13 (delta 0), reused 10 (delta 0), pack-reused 0
Unpacking objects: 100% (13/13), done.
```

**17. Move into your project directory.**

```
$ cd mtb-example-psoc6-empty-app
```

**18. Create a .lib file to tell ModusToolbox to include your new library in this project. The file will contain a single line:**

```
https://github.com/YOUR_GITHUB_NAME/YOUR_REPO_NAME/#YOUR_RELEASE_TAG
```

Be very careful with the punctuation, particularly the “#” between the repo and tag names.
You can create this file with an editor or a command like (but not the same as) this:

```
$ echo "https://github.com/markgsaunders/ezcolor/#v2" > ezcolor.lib
$ cat ezcolor.lib
https://github.com/markgsaunders/ezcolor/#v2
```

**19. Now run make getlibs to pull in all the library firmware into your project.**

```
$ make getlibs
Tools Directory: C:/Users/yfs/ModusToolbox/tools_2.1
Initializing import: mtb-example-psoc6-empty-app
=====
= Importing libraries =
=====
Git is git version 2.17.0, found at /usr/bin/git
Searching application directories...
Found 1 file(s)
    Processing file C:/Users/yfs/colortest/mtb-example-psoc6-empty-app/ezcolor.lib
Application directories search complete.
Searching libs directory...
Found 8 file(s)
    Processing file C:/Users/yfs/colortest/mtb-example-psoc6-empty-app/libs/TARGET_CY8CPROTO-062-4343W.lib
        Processing file C:/Users/yfs/colortest/mtb-example-psoc6-empty-app/libs/ezcolor/rgb-led.lib
            Processing file C:/Users/yfs/colortest/mtb-example-psoc6-empty-app/libs/TARGET_CY8CPROTO-062-4343W/libs/capsense.lib
                Processing file C:/Users/yfs/colortest/mtb-example-psoc6-empty-app/libs/TARGET_CY8CPROTO-062-4343W/libs/core-lib.lib
                    Processing file C:/Users/yfs/colortest/mtb-example-psoc6-empty-app/libs/TARGET_CY8CPROTO-062-4343W/libs/psoc6cm0p.lib
                        Processing file C:/Users/yfs/colortest/mtb-example-psoc6-empty-app/libs/TARGET_CY8CPROTO-062-4343W/libs/psoc6hal.lib
                            Processing file C:/Users/yfs/colortest/mtb-example-psoc6-empty-app/libs/TARGET_CY8CPROTO-062-4343W/libs/psoc6make.lib
                                Processing file C:/Users/yfs/colortest/mtb-example-psoc6-empty-app/libs/TARGET_CY8CPROTO-062-4343W/libs/psoc6pd1.lib
                                    Libraries were processed. Re-evaluating libs directory...
Found 8 file(s)
libs directory search complete.
=====
= Import complete =
=====
```

- 20. Unfortunately, this example does not include the BSP for your kit, so we need to add it. It is a simple git command to fetch the required firmware.

```
$ git clone https://github.com/cypresssemiconductorco/TARGET_CY8CKIT-062-WIFI-BT
Cloning into 'TARGET_CY8CKIT-062-WIFI-BT'...
remote: Enumerating objects: 254, done.
remote: Counting objects: 100% (254/254), done.
remote: Compressing objects: 100% (155/155), done.
remote: Total 254 (delta 98), reused 244 (delta 91), pack-reused 0
Receiving objects: 100% (254/254), 1.08 MiB | 1.99 MiB/s, done.
Resolving deltas: 100% (98/98), done.
Checking out files: 100% (221/221), done.
```

- 21. Edit your Makefile to change the target BSP.

```
#####
# Basic Configuration
#####
# Target board/hardware
# TARGET=CY8CPROTO-062-4343W
TARGET=CY8CKIT-062-WIFI-BT
```

- 22. Overwrite your main.c with the following code.

```
#include "cy_pdl.h"
#include "cyhal.h"
#include "cybsp.h"
#include "ezcolor.h"
int main(void)
{
    cy_rslt_t result;
    /* Initialize the device and board peripherals */
    result = cybsp_init() ;
    if (result != CY_RSLT_SUCCESS)
    {
        CY_ASSERT(0);
    }
    __enable_irq();
    init_colors();
    for(;;)
    {
        color( RED );
        Cy_SysLib_Delay( 1000 );
        color( GREEN );
        Cy_SysLib_Delay( 1000 );
        color( BLUE );
        Cy_SysLib_Delay( 1000 );
        color( YELLOW );
        Cy_SysLib_Delay( 1000 );
        color( CYAN );
        Cy_SysLib_Delay( 1000 );
        color( MAGENTA );
        Cy_SysLib_Delay( 1000 );
        color( WHITE );
        Cy_SysLib_Delay( 1000 );
        color( BLACK );
        Cy_SysLib_Delay( 1000 );
    }
}
```

- 23. Run make program and observe the LED colors.

5.14.10 Create and Use a New BSP



1. From the command line, create a new project: Hello World.

```
$ git clone https://github.com/cypresssemiconductorco/mtb-example-psoc6-Hello-World
```



2. Pull in the BSP for your kit (this is done for you by the Project Creator tool when you use the IDE) and change the TARGET rule in the Makefile (e.g. TARGET_CY8CKIT-062-WIFI-BT).

```
$ git clone https://github.com/cypresssemiconductorco/TARGET_CY8CKIT-062-WIFI-BT
```



3. Run make getlibs to pull in all the library firmware into your project.

```
$ make getlibs
```



4. Build and program the kit and observe the LED that pauses/resumes blinking when you press the return key in your terminal emulator.

```
$ make program
```



5. Copy the TARGET_CY8CKIT-062-WIFI-BT BSP directory to TARGET_MYKIT

```
cp -r TARGET_CY8CKIT-062-WIFI-BT MYKIT
```



6. In the new directory, edit the cybsp_types.h file to swap the LEDs assigned to CYBSP_USER_LED1 and CYBSP_USER_LED2.

```
/** LED 9; User LED1 */  
#define CYBSP_USER_LED1 (CYBSP_LED9)  
/** LED 8; User LED2 */  
#define CYBSP_USER_LED2 (CYBSP_LED8)
```



7. Rename the CY8CKIT-062-WIFI-BT.mk to match your kit name.

```
$ mv CY8CKIT-062-WIFI-BT.mk MYKIT.mk
```



8. Build and program the kit with your new BSP and observe a different LED blinking.

```
$ make program TARGET=MYKIT
```