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

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

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

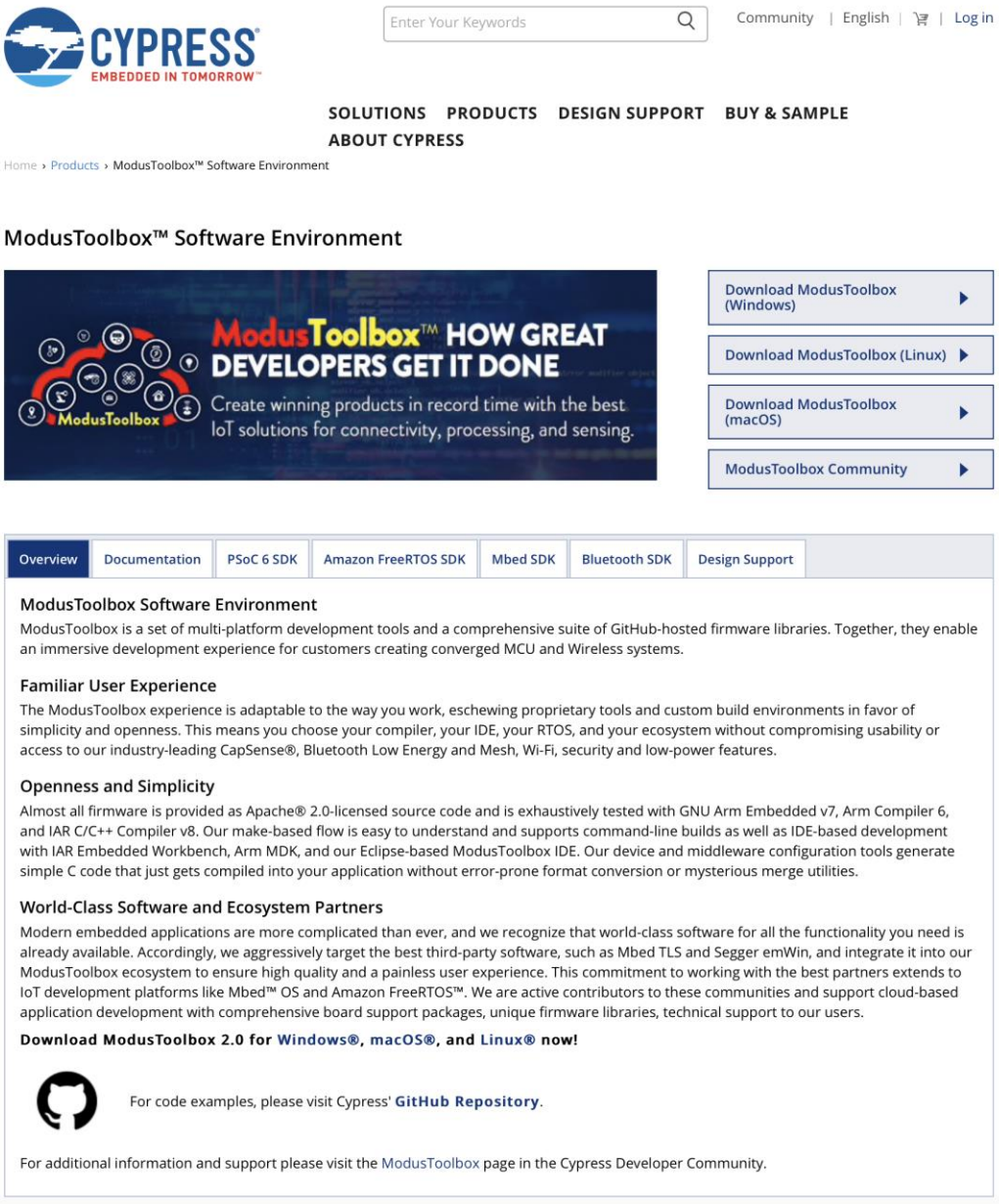
This section provides a brief tour of the ModusToolbox websites, IDE, and various tools.

5.1.1 Web

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

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

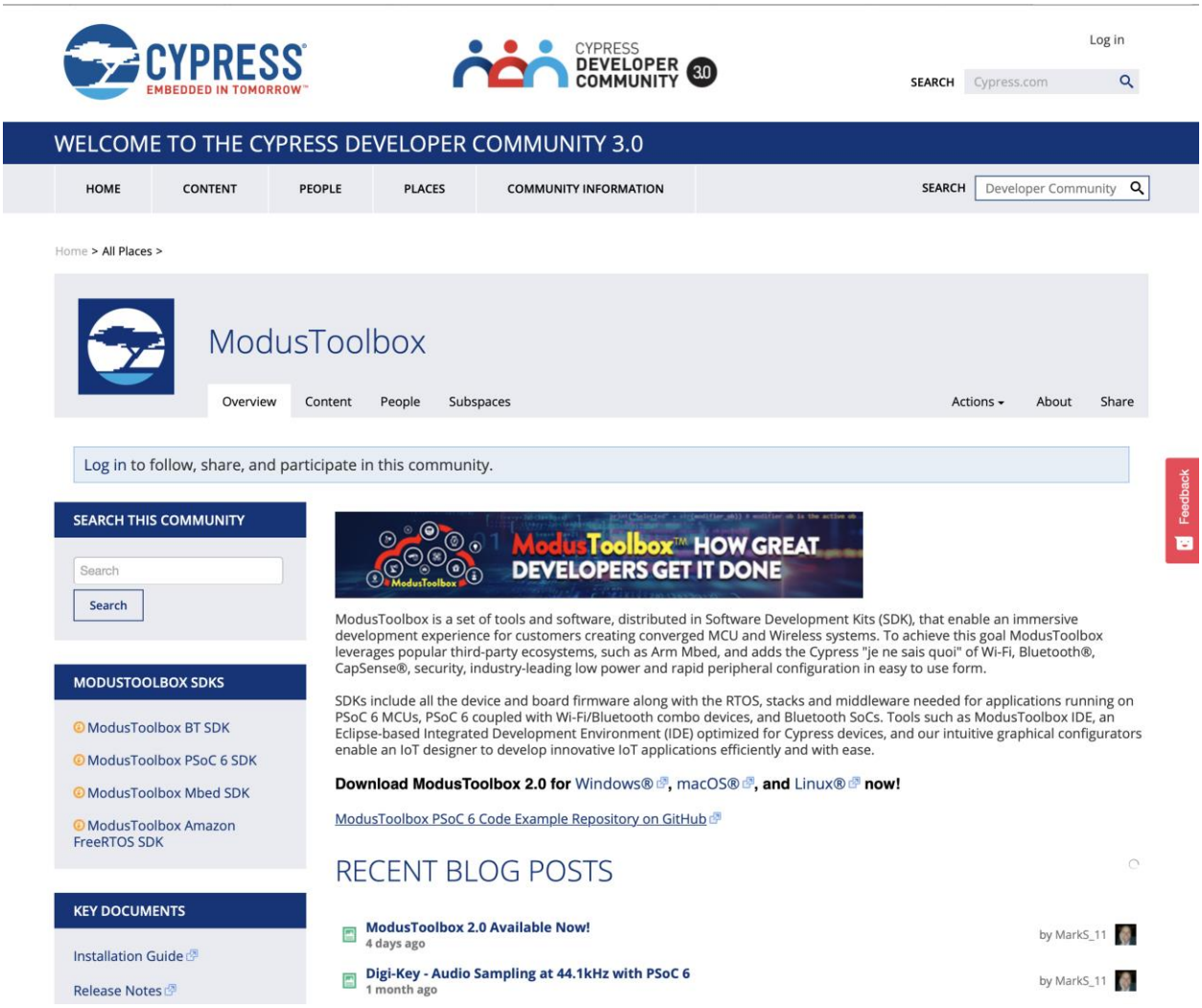
Cypress.com



Community

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

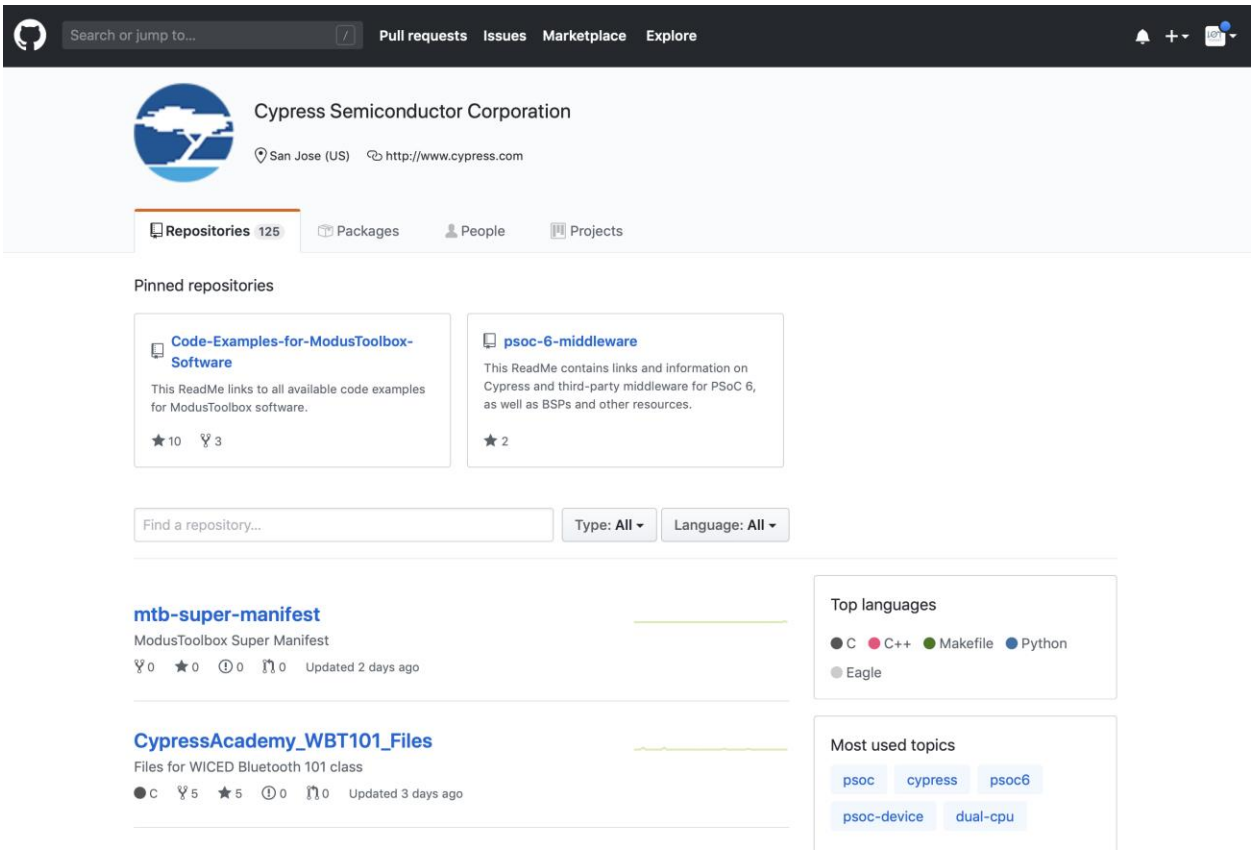
https://community.cypress.com/community/modustoolbox



Github.com

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

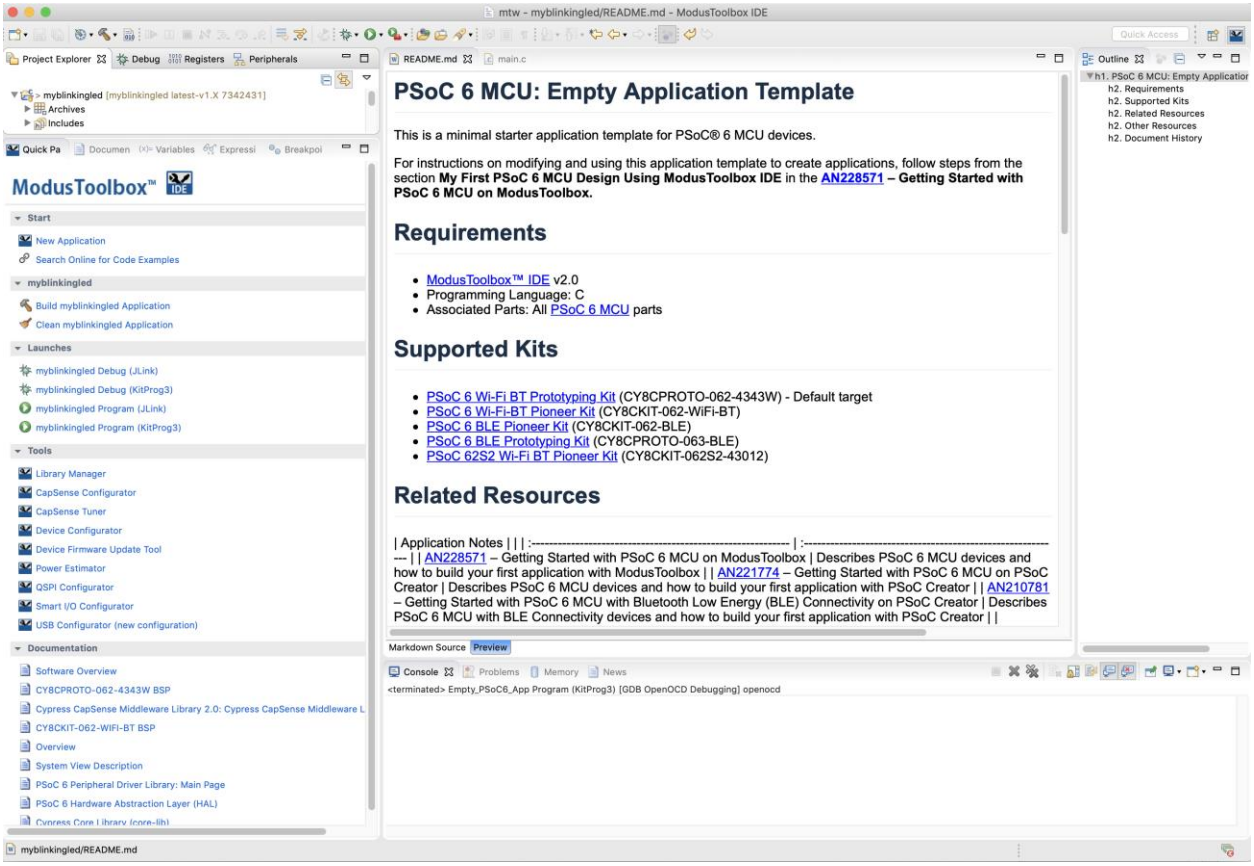
https://github.com/cypresssemiconductorco



5.1.2 Eclipse-Based IDE

The ModusToolbox IDE is based on the Eclipse IDE. It 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 ModusToolbox IDE User Guide:

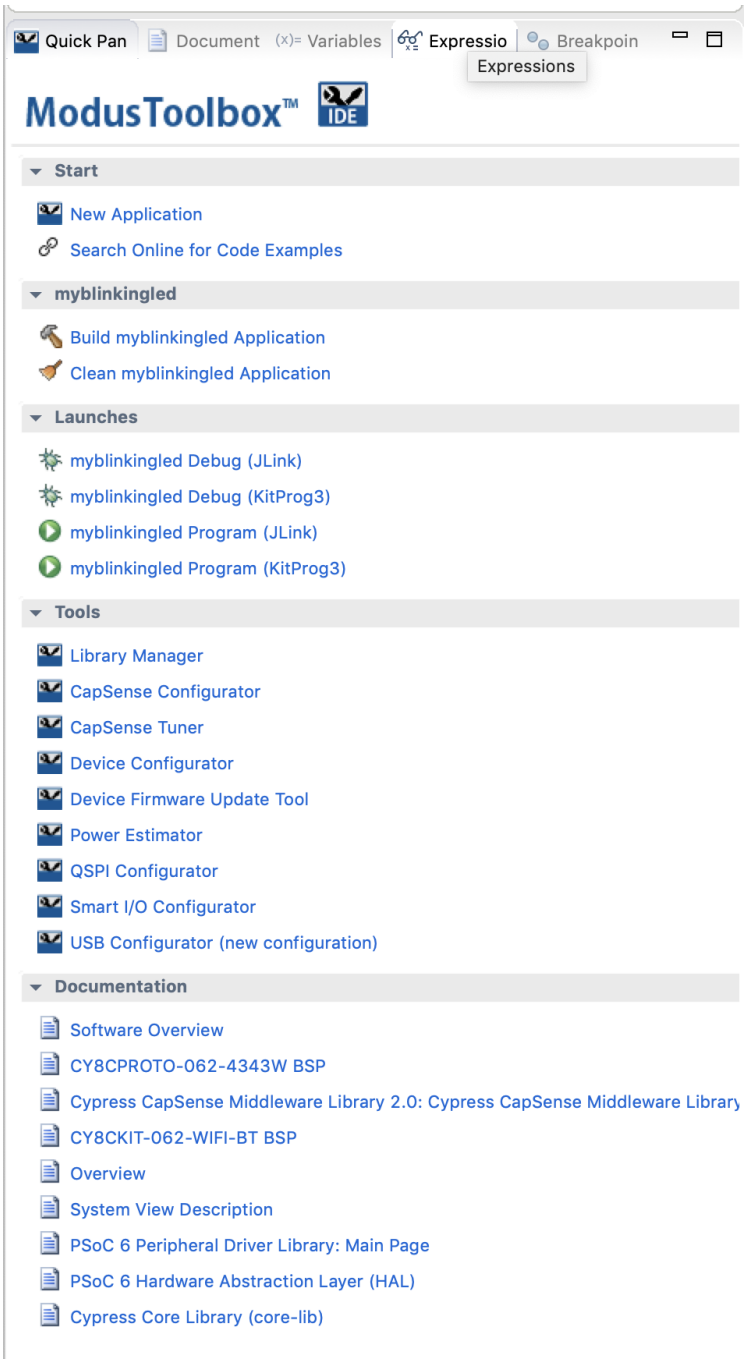
http://www.cypress.com/ModusToolboxUserGuide



Quick Panel

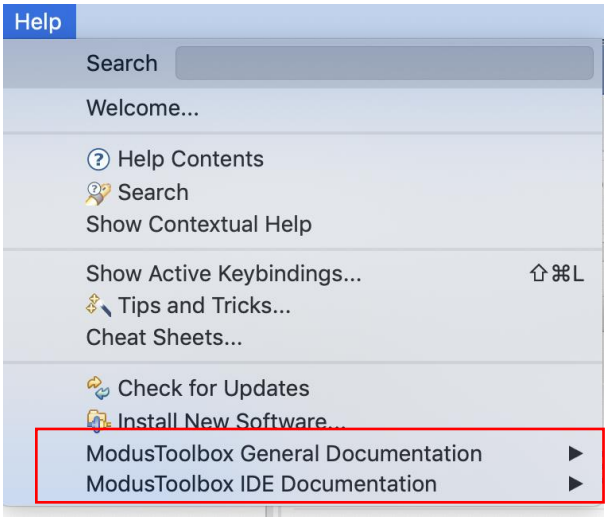
Cypress has extended the Eclipse functionality with a "ModusToolbox" Perspective. Among other things (such as adding a bunch of useful Debug windows), this perspective includes a panel with links to commonly used functions including:

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


Quick Pan
Document
Variables
Expressio
Breakpoin
Expressions
ModusToolbox™
IDE
Start
New Application
Search Online for Code Examples
myblinkingled
Build myblinkingled Application
Clean myblinkingled Application
Launches
myblinkingled Debug (JLink)
myblinkingled Debug (KitProg3)
myblinkingled Program (JLink)
myblinkingled Program (KitProg3)
Tools
Library Manager
CapSense Configurator
CapSense Tuner
Device Configurator
Device Firmware Update Tool
Power Estimator
QSPI Configurator
Smart I/O Configurator
USB Configurator (new configuration)
Documentation
Software Overview
CY8CPROTO-062-4343W BSP
Cypress CapSense Middleware Library 2.0: Cypress CapSense Middleware Library
CY8CKIT-062-WIFI-BT BSP
Overview
System View Description
PSoC 6 Peripheral Driver Library: Main Page
PSoC 6 Hardware Abstraction Layer (HAL)
Cypress Core Library (core-lib)

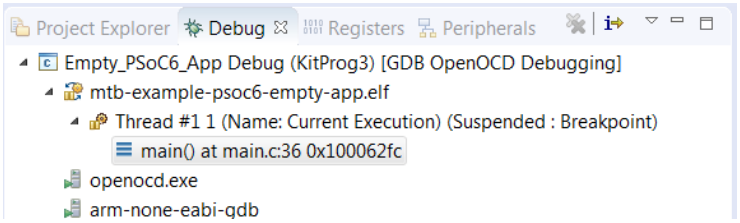
Help Menu

The IDE Help menu provides links to general documentation, such as the ModusToolbox Software Overview and ModusToolbox Release Notes, as well as IDE-specific documentation, such as the ModusToolbox IDE Quick Start Guide.



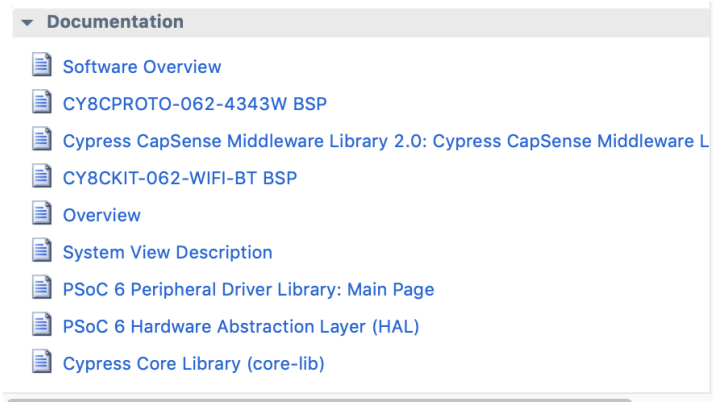
Integrated Debugger

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.



ModusToolbox 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:

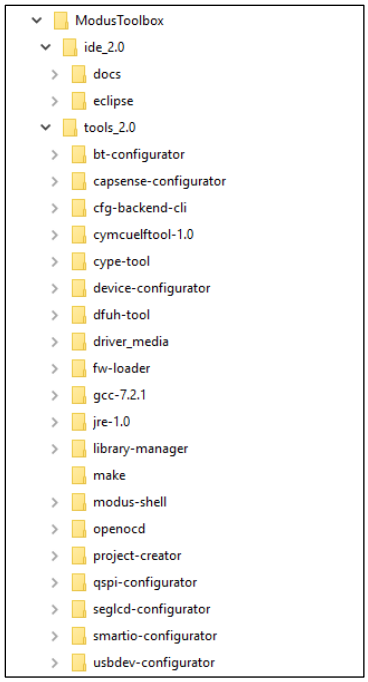
- If your code has IntelliSense issues, use **Program > Rebuild Index** and/or **Program > Build**.
- Sometimes when you import a project, you don't see all the files. Right-click on the project and select **Refresh**.
- Various menus and the Quick Panel show different things depending on what you select 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_2.0:



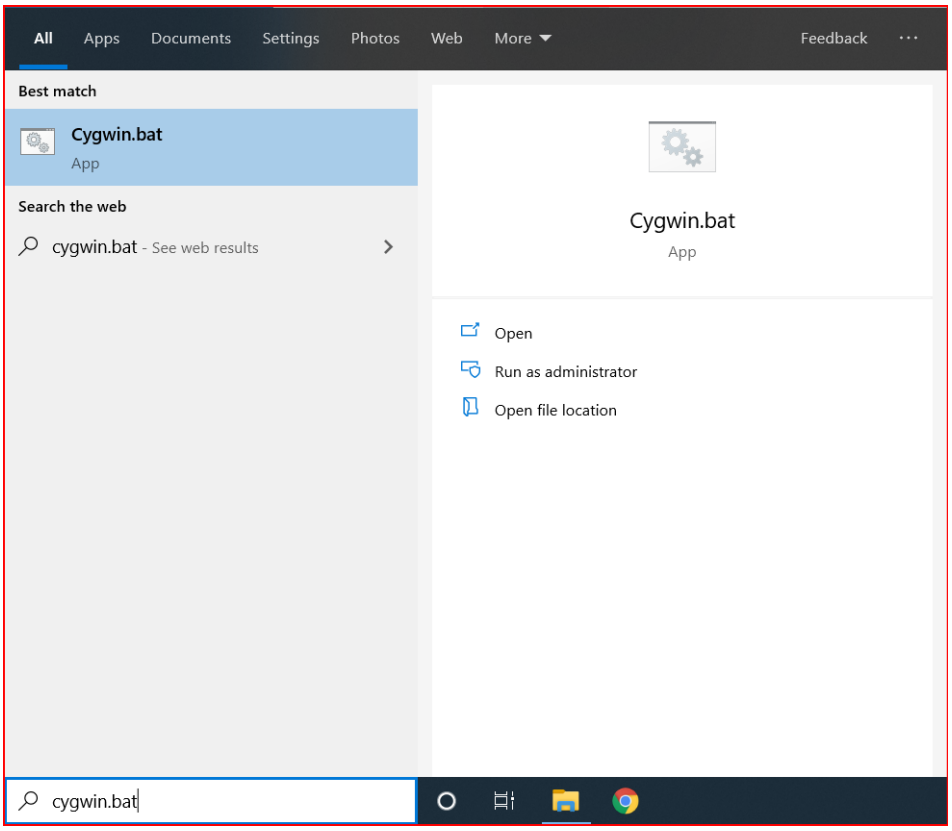
5.1.4 Command Line

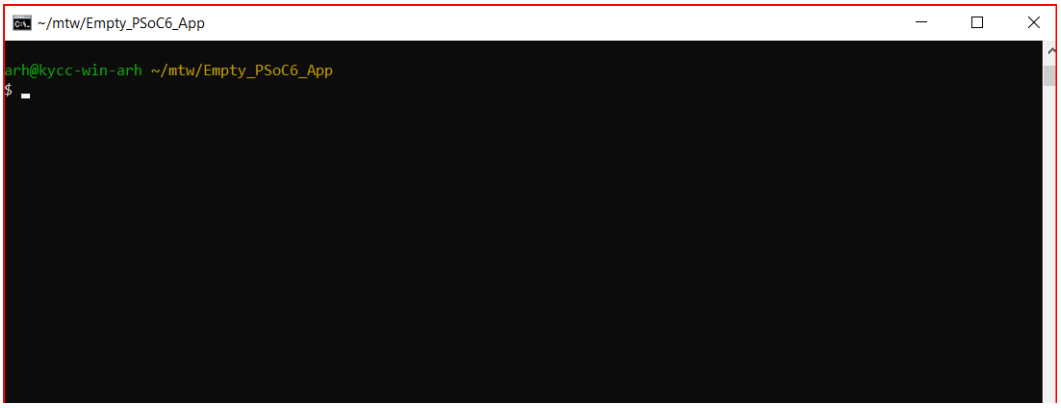
You can run all the same IDE and tool functions using the command line.

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_2.0/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 Running ModusToolbox from the Command Line document.

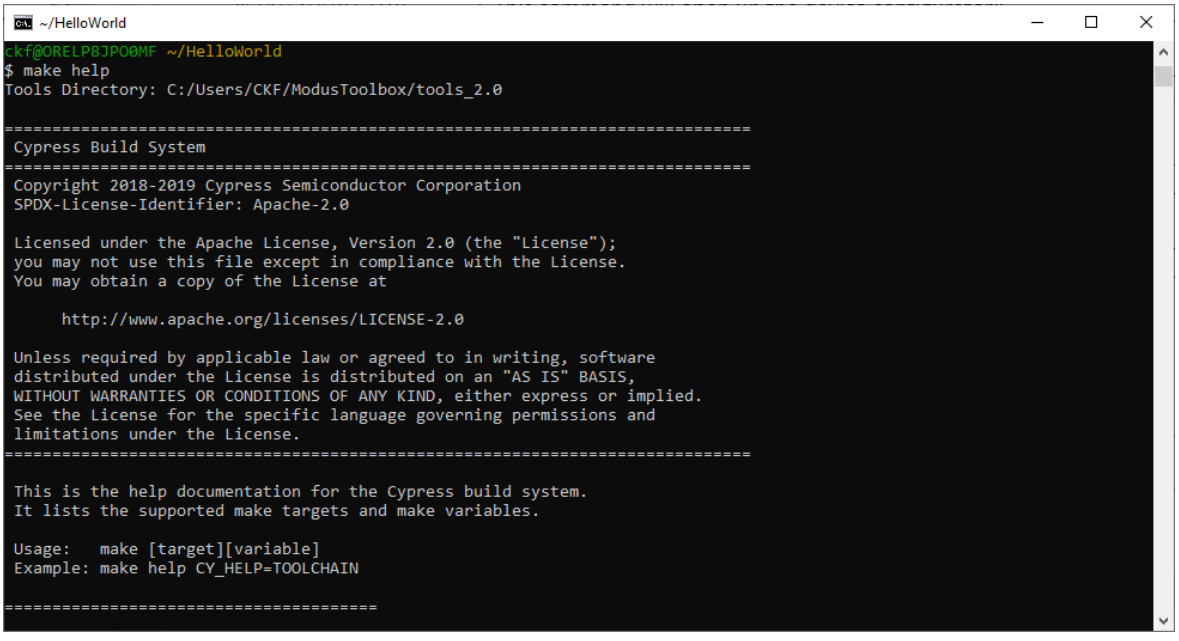
Make Command	Description
`make help`	This command will print out a list of all of the make targets. To get help on a specific target type "make help CY_TARGET=getlibs" (or whichever target you want)
`make getlibs`	Hierarchically process all of the ".lib" files and bring all of the libraries into your 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

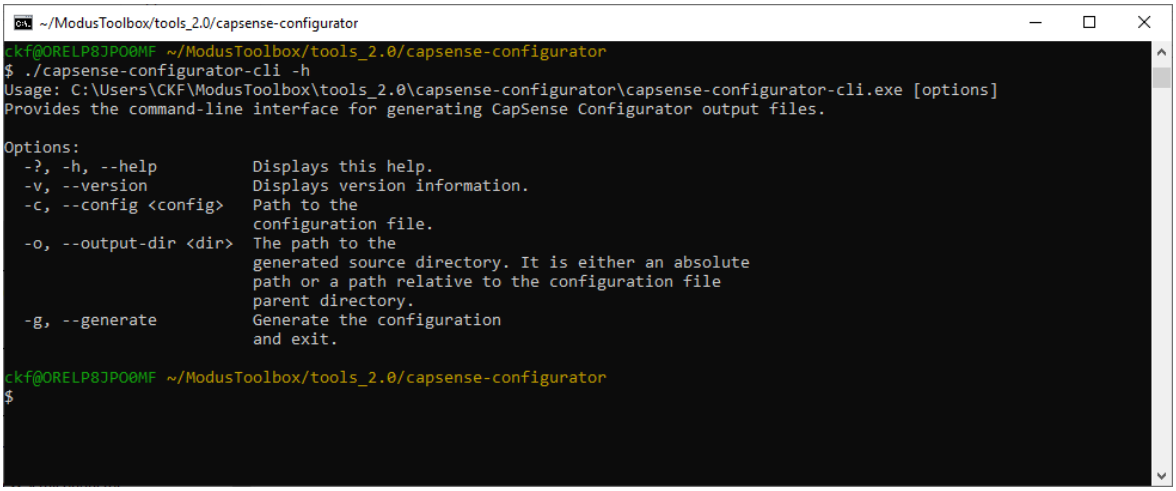
```
make help CY_HELP=TARGET
```



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_2.0. The `-h` option shows help. For example:

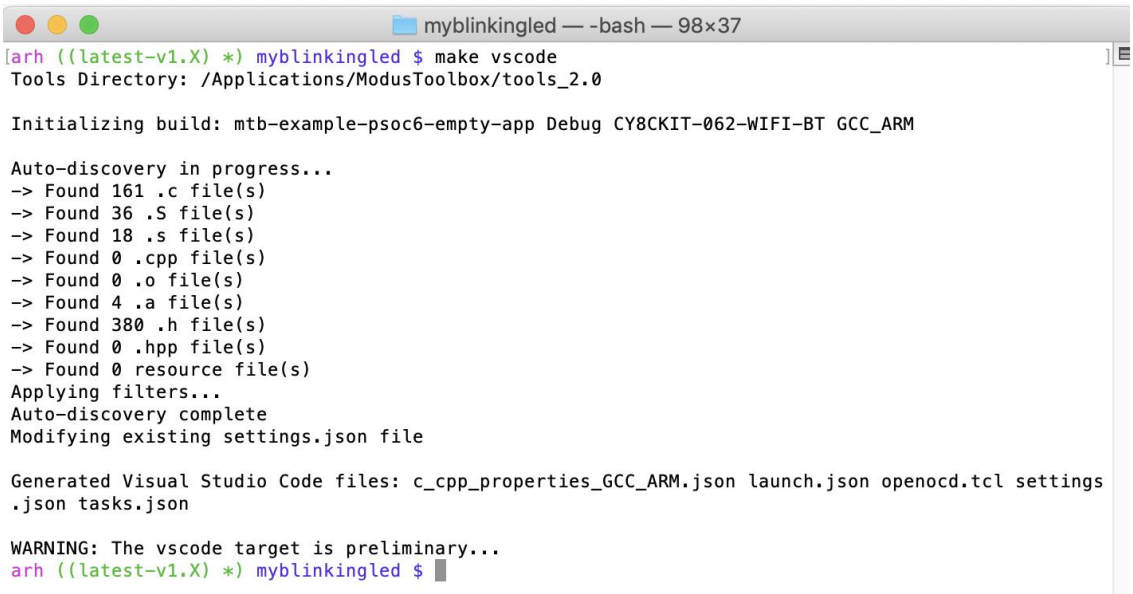
```
./capsense-configurator-cli -h
```



5.1.5 Visual Studio (VS) Code (Alpha)

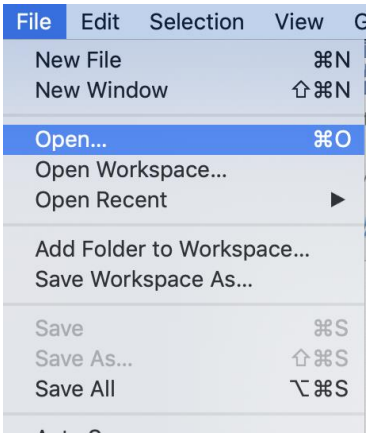
Another alternative to using the Eclipse-based IDE is a code editor program called VS Code. This tool is quickly becoming a favorite editor for developers. The ModusToolbox command line knows how to make all the files required for VS Code to edit, build, and program a ModusToolbox program. To create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

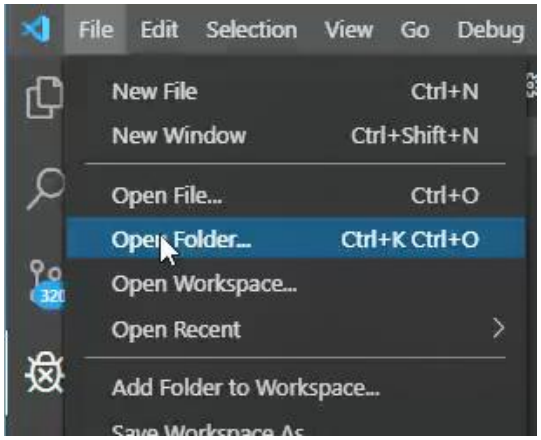
```
make vscode
```



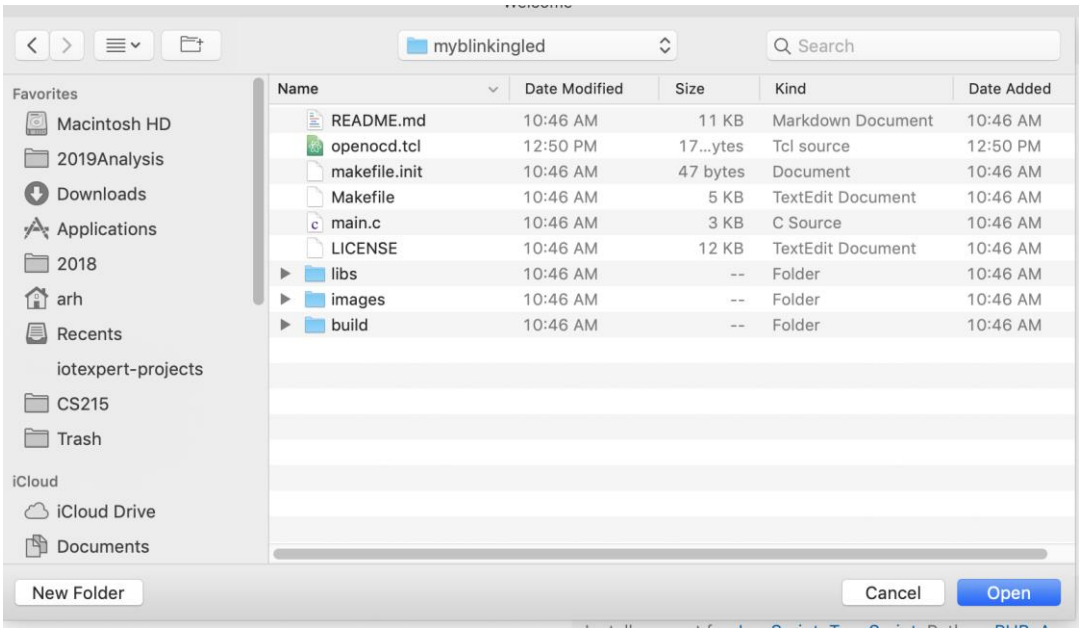
Note: By default, if you don't specify the target, this will create VS Code settings for the CY8CPROTO-062-4343W kit (since that is the target specified in the Makefile). If you want a different target, it is best to add a .lib for the desired target and update the TARGET variable in the Makefile before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

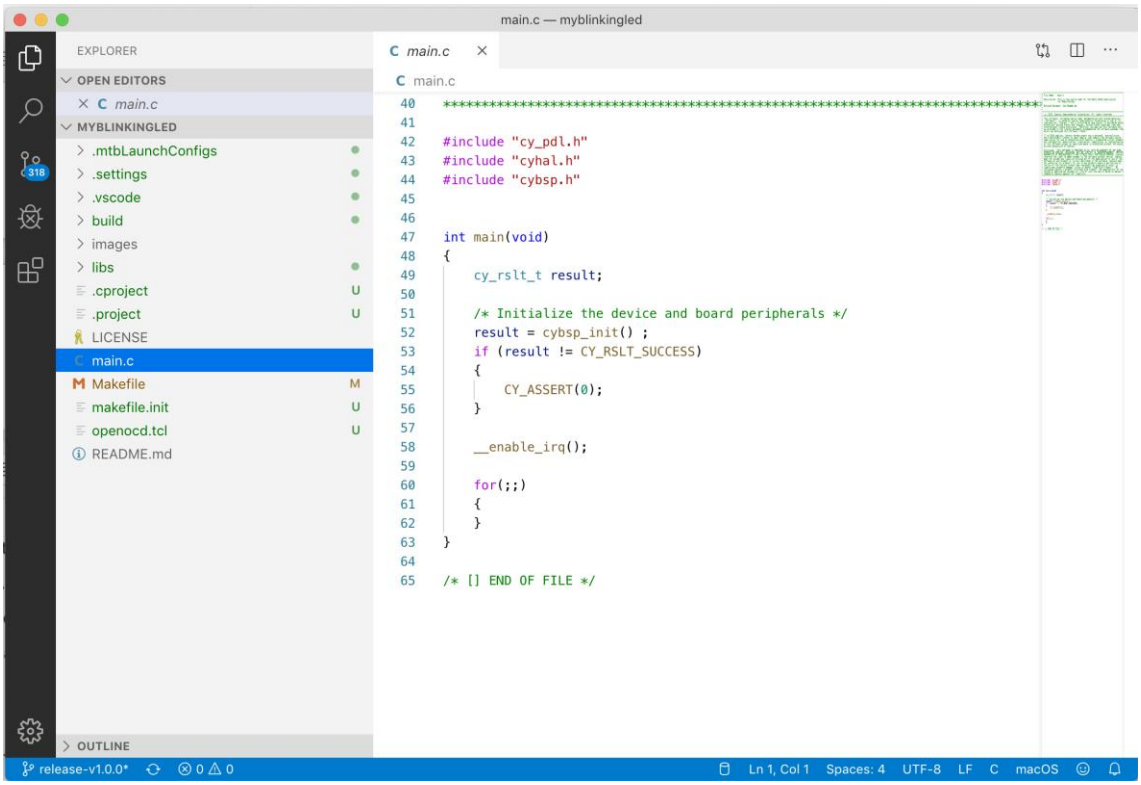




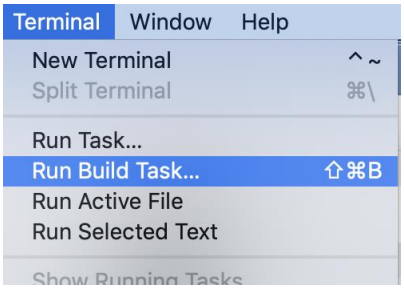
Then, navigate to the directory where your project resides and click **Open**.



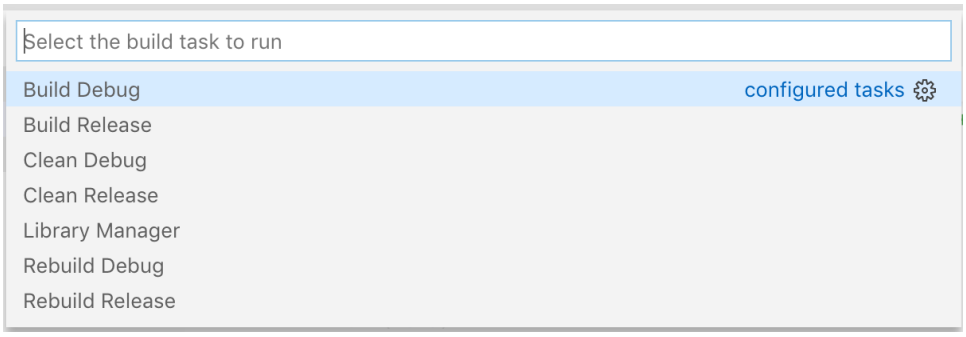
Now your windows should look something like this. You can open the files by clicking on them in the Explorer window.



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



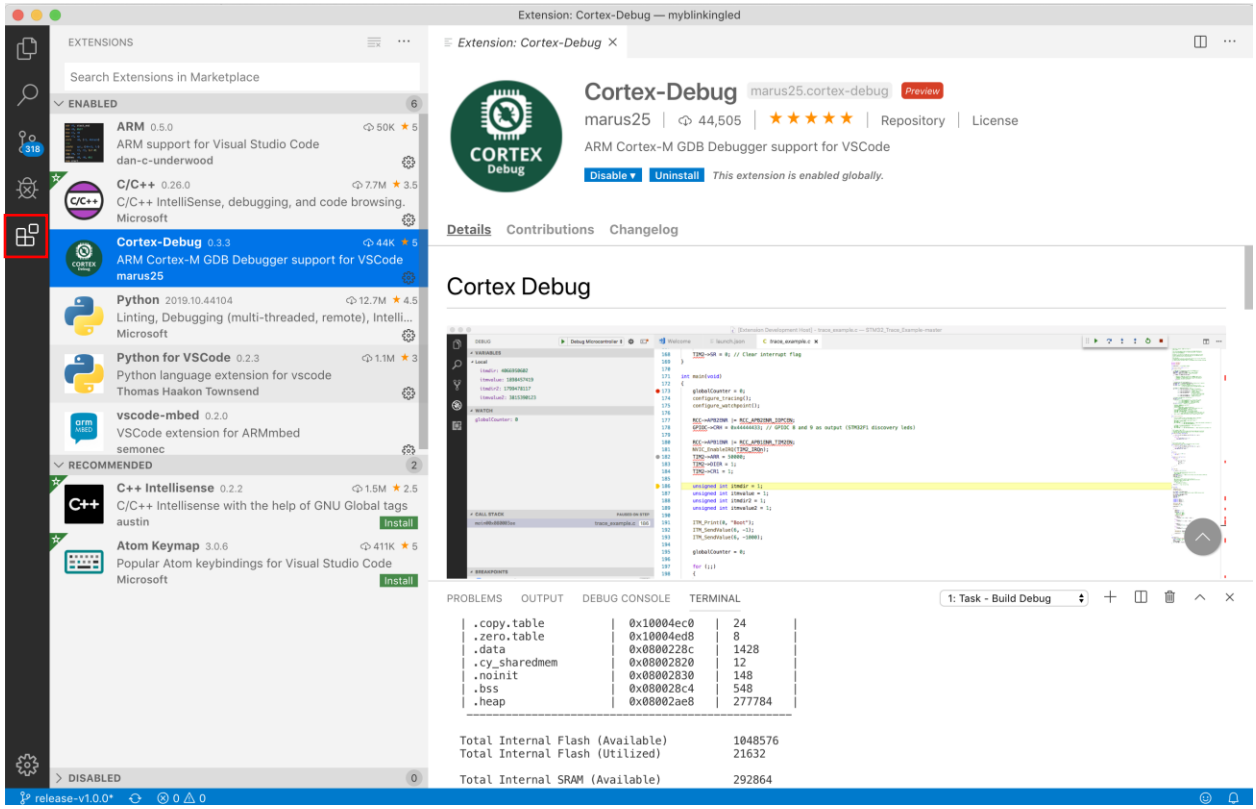
Then, select **Build Debug**. This will essentially run `make build` at the top level of your project.



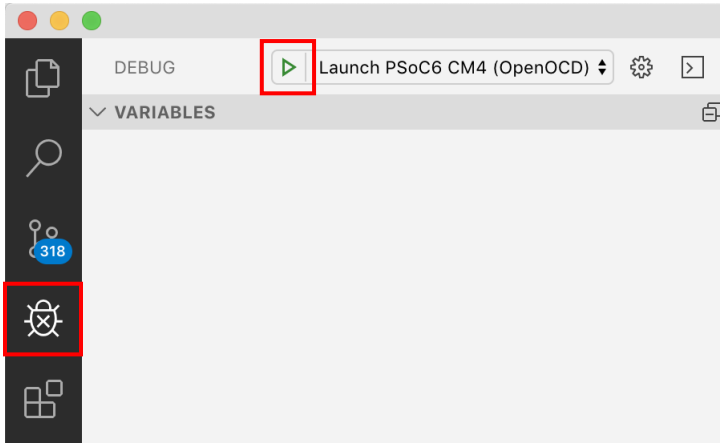
You should see the normal compiler output in the console at the bottom of VS Code:

```
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_sharedmem         |  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.
```

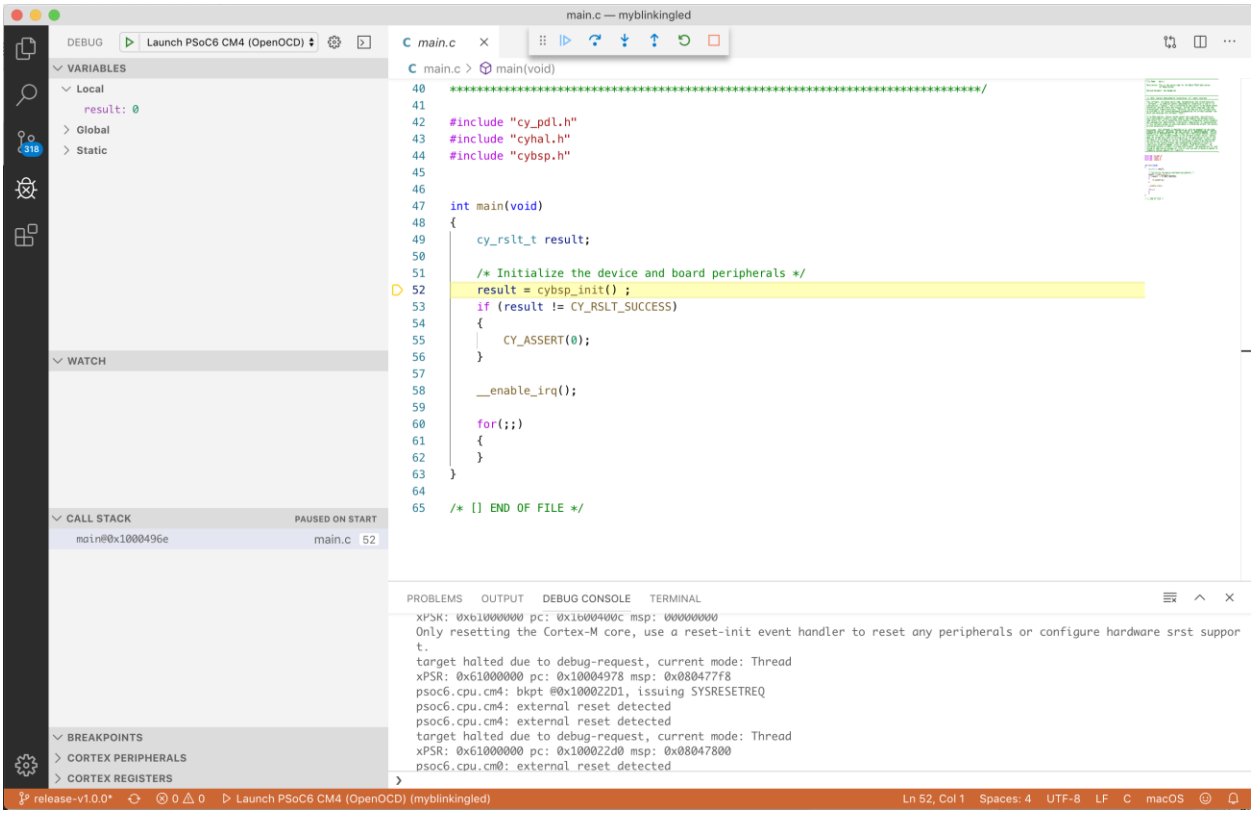
To program the kit or run the debugger, you need to install the “Cortex-Debug” VS Code plug in from the marketplace. Click the little boxes symbol. Then, search for Cortex-Debug. Then click **Install**.



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.



5.2 Demo Walkthrough

5.2.1 Blinking LED from IDE

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

5.2.2 Blinking LED from command line interface

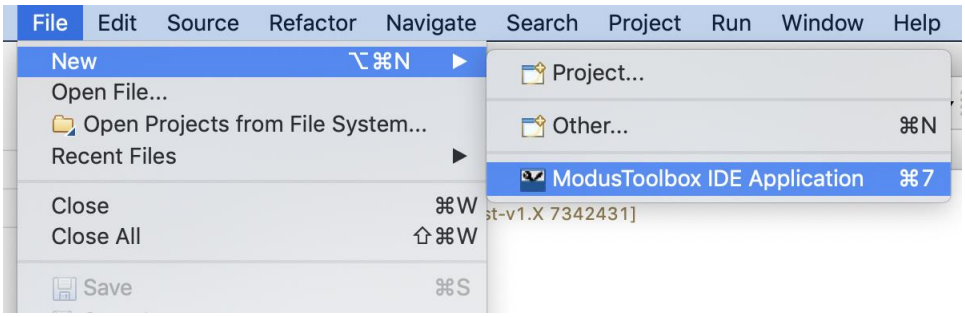
Repeat demonstration but using the command-line tools rather than an IDE.

5.3 Exercises (Part 1)

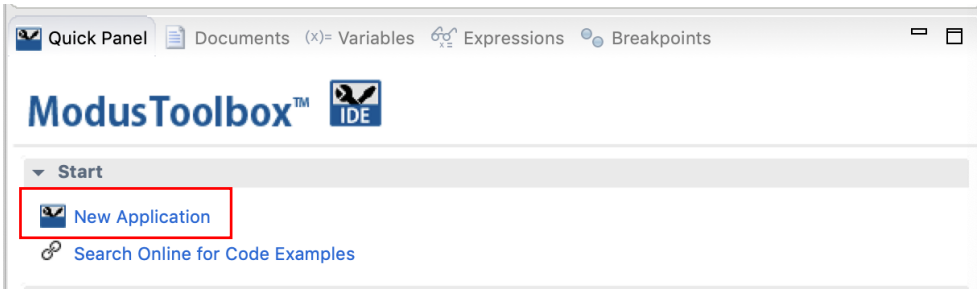
5.3.1 Blinking LED from IDE

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

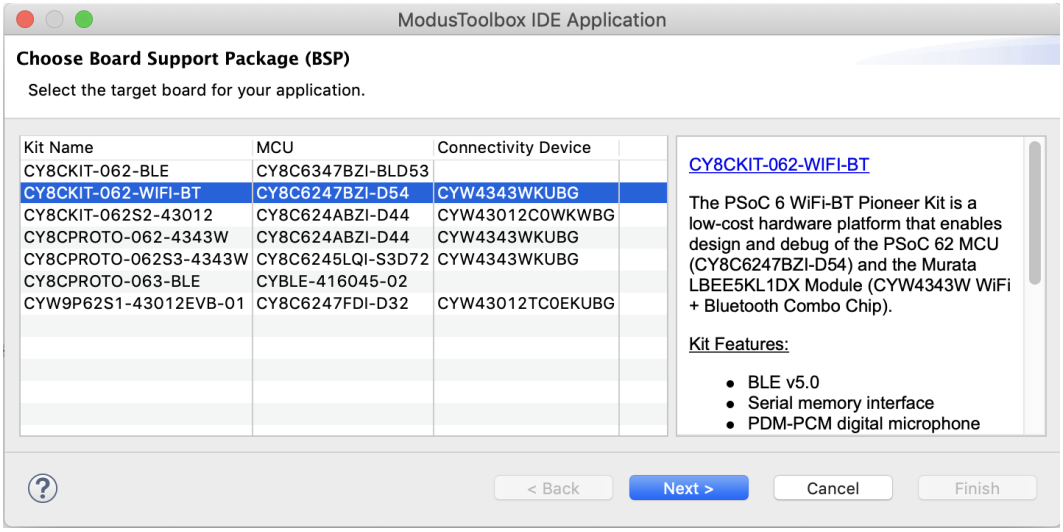
1. Select either **File > New > ModusToolbox IDE Application**.



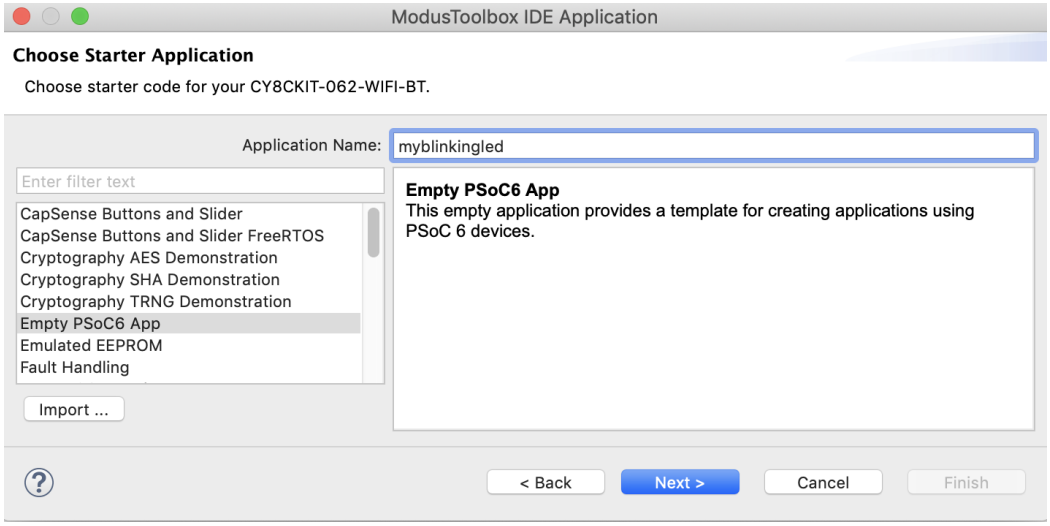
2. Or on the Quick Panel, click the **New Application** link.



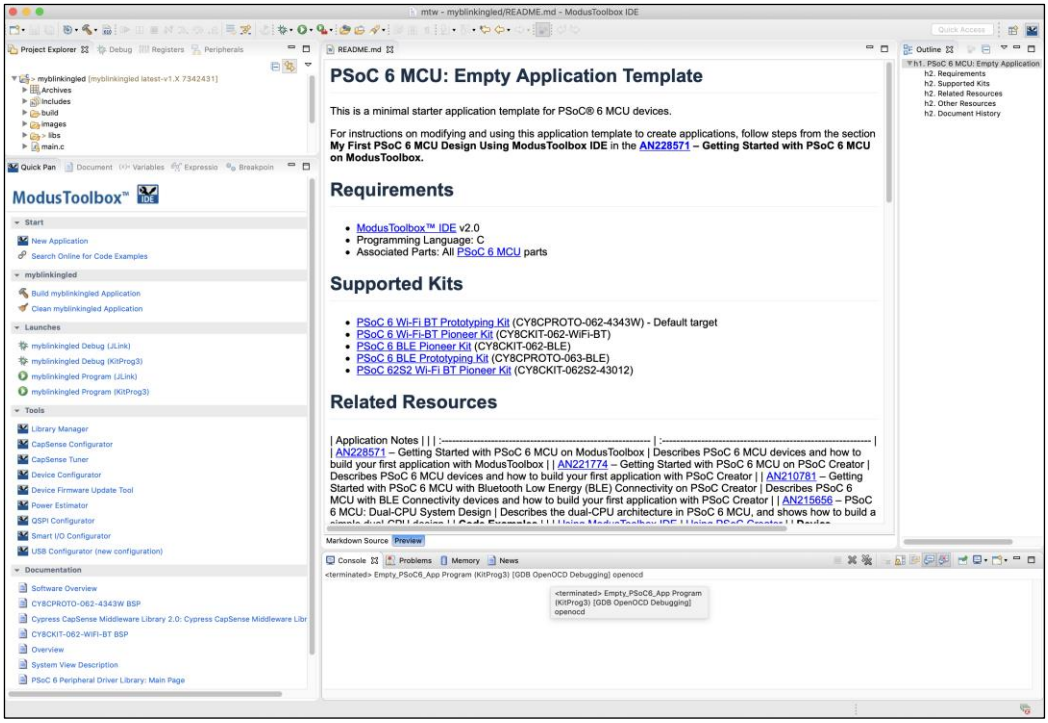
3. Choose the correct development kit.



4. Select "Empty PSoC 6 App" and give your project a name (in this case “myblinkingled”). Then, click **Next >** and **Finish**.



5. Now you should have a screen that looks like this:



6. 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);
    }
}
```

7. 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.

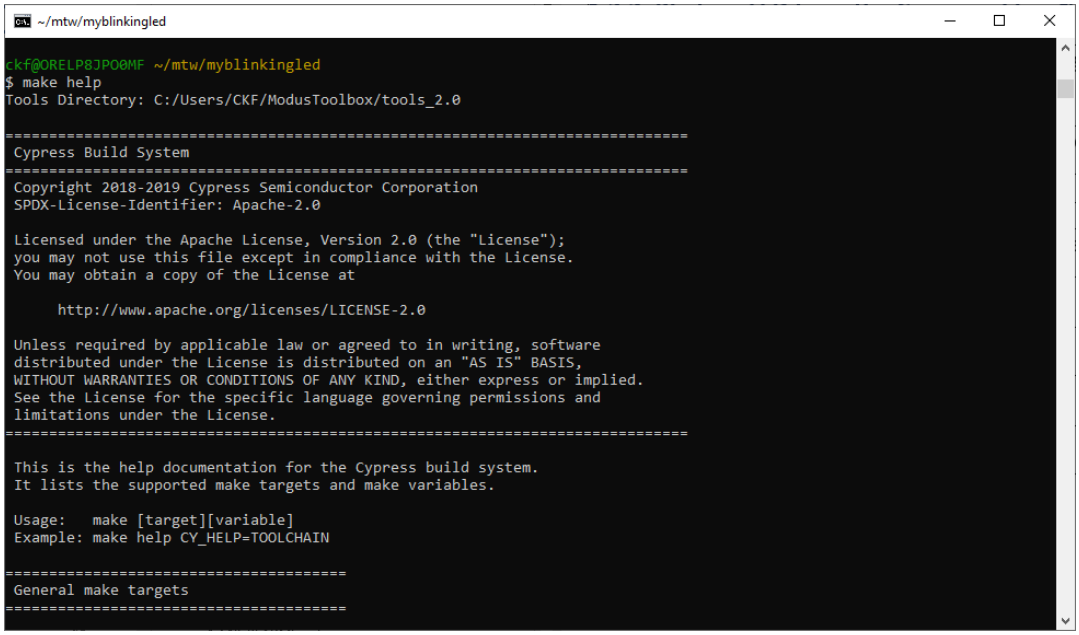
1. On Windows, navigate to the modus-shell directory and run Cygwin.bat.

   <install_dir>\ModusToolbox\tools_<version>\modus-shell\

2. On Linux or macOS, open a terminal window.
3. Navigate to the project directory that you created previously. For example:

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

4. Next run `make help` to see the list of commands.



5. Then you can run:

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

5.4 Create/Import ModusToolbox Projects

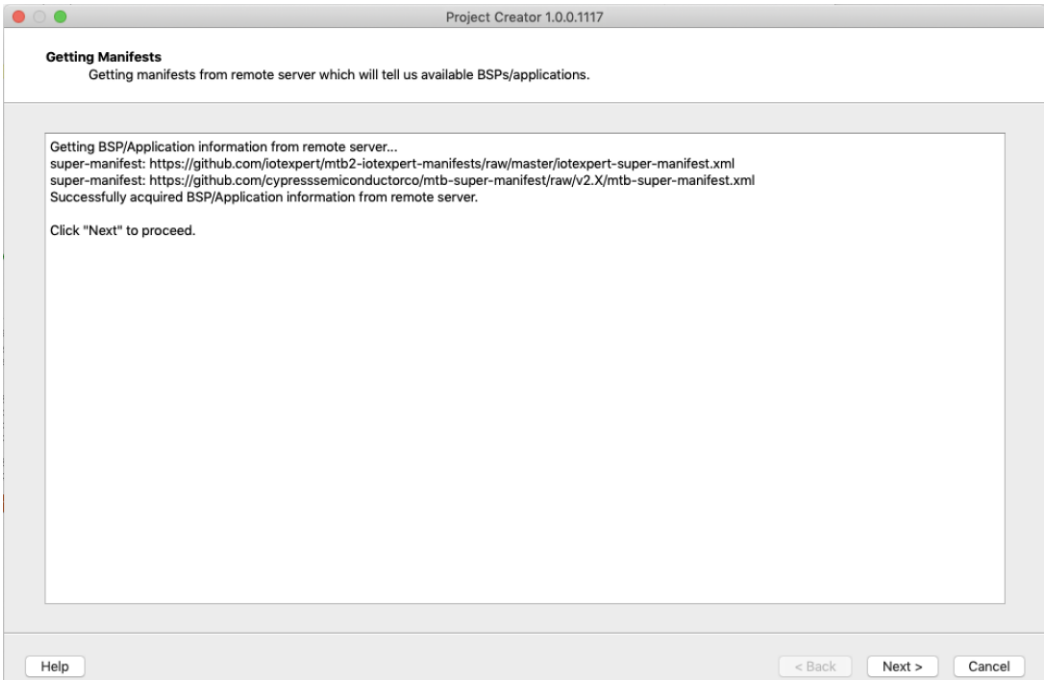
As we've discussed already, you can use the ModusToolbox Eclipse-based IDE to create projects. We understand many developers may not want to use the Eclipse IDE. So, there are several ways to create projects, and they are all based on the `git clone` mechanism. Here are some of the methods you can use:

- ModusToolbox IDE (discussed previously)
- Stand-alone Project Creator tool
- Command Line / Manual Setup
- Importing Projects

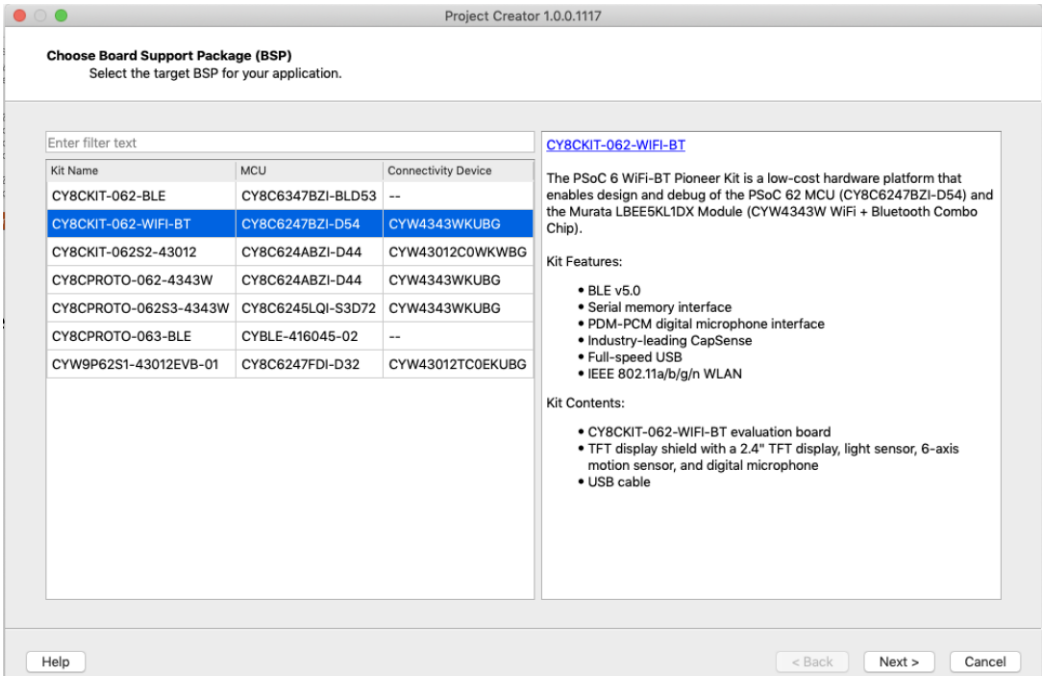
5.4.1 Project Creator Tool

Instead of using the New Application wizard inside of ModusToolbox IDE, you can create new projects with a stand-alone GUI. It is in the ModusToolbox/tools_2.0/project-creator directory. You can also just search for "project-creator".

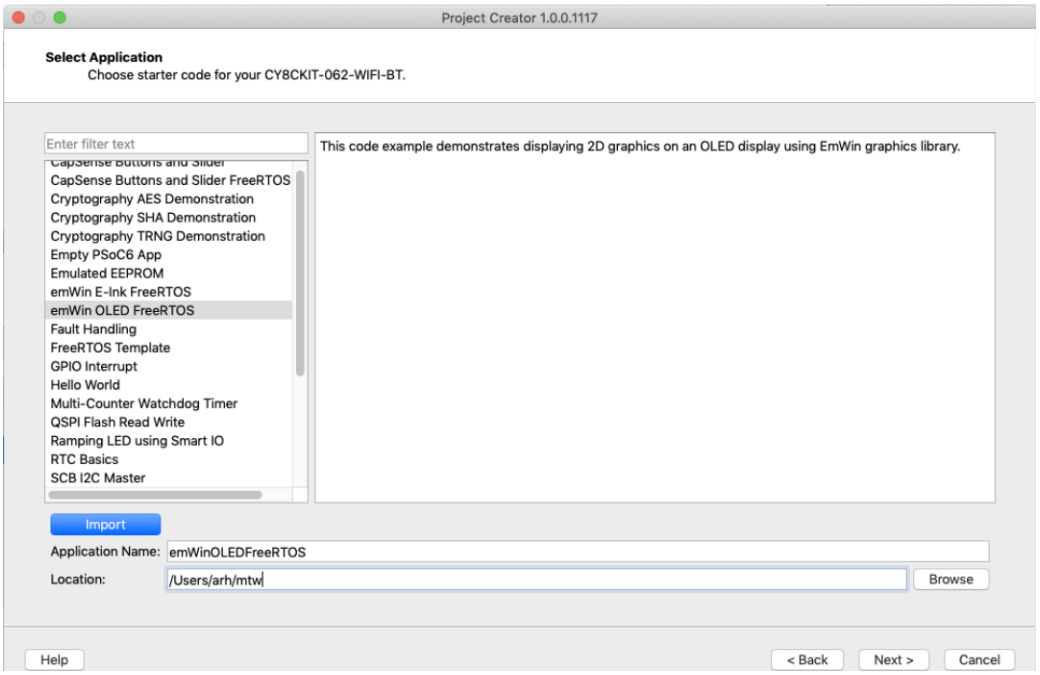
The first thing the project-creator tool does is read the Cypress configuration information from the Cypress GitHub site so that it knows all of the BSPs etc. that we support. Click **Next >**.



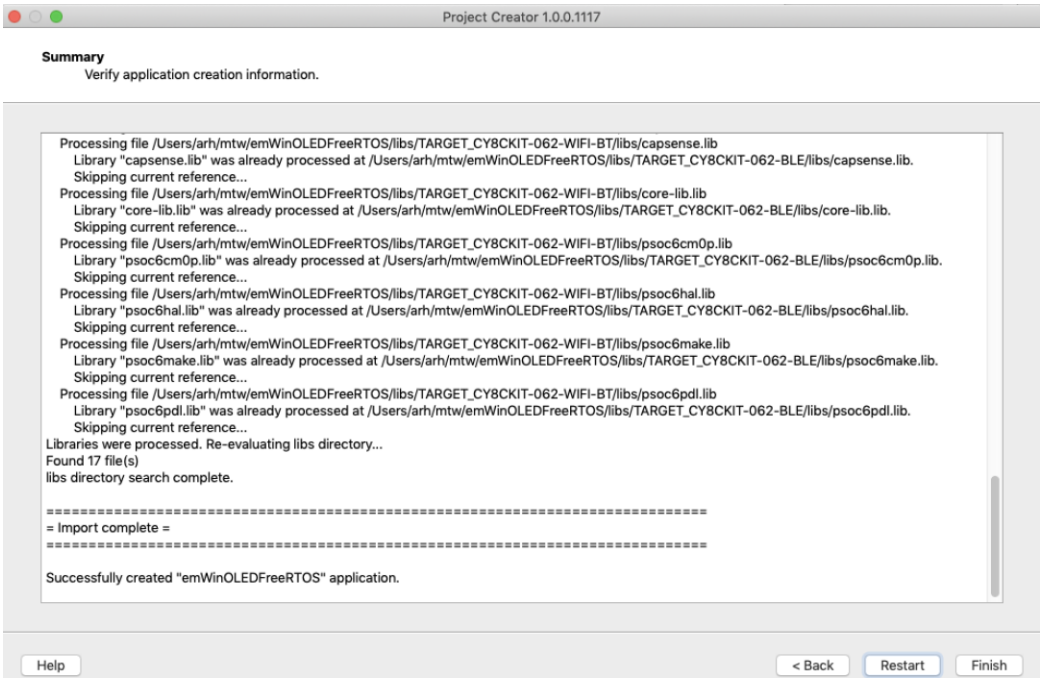
You will be given the option to choose a BSP to start from. Pick one and click **Next >**.



Now you will be presented with all the Cypress starter projects 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 **Next >** to advance to a summary, then 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.

5.4.2 Command Line / Manual Setup

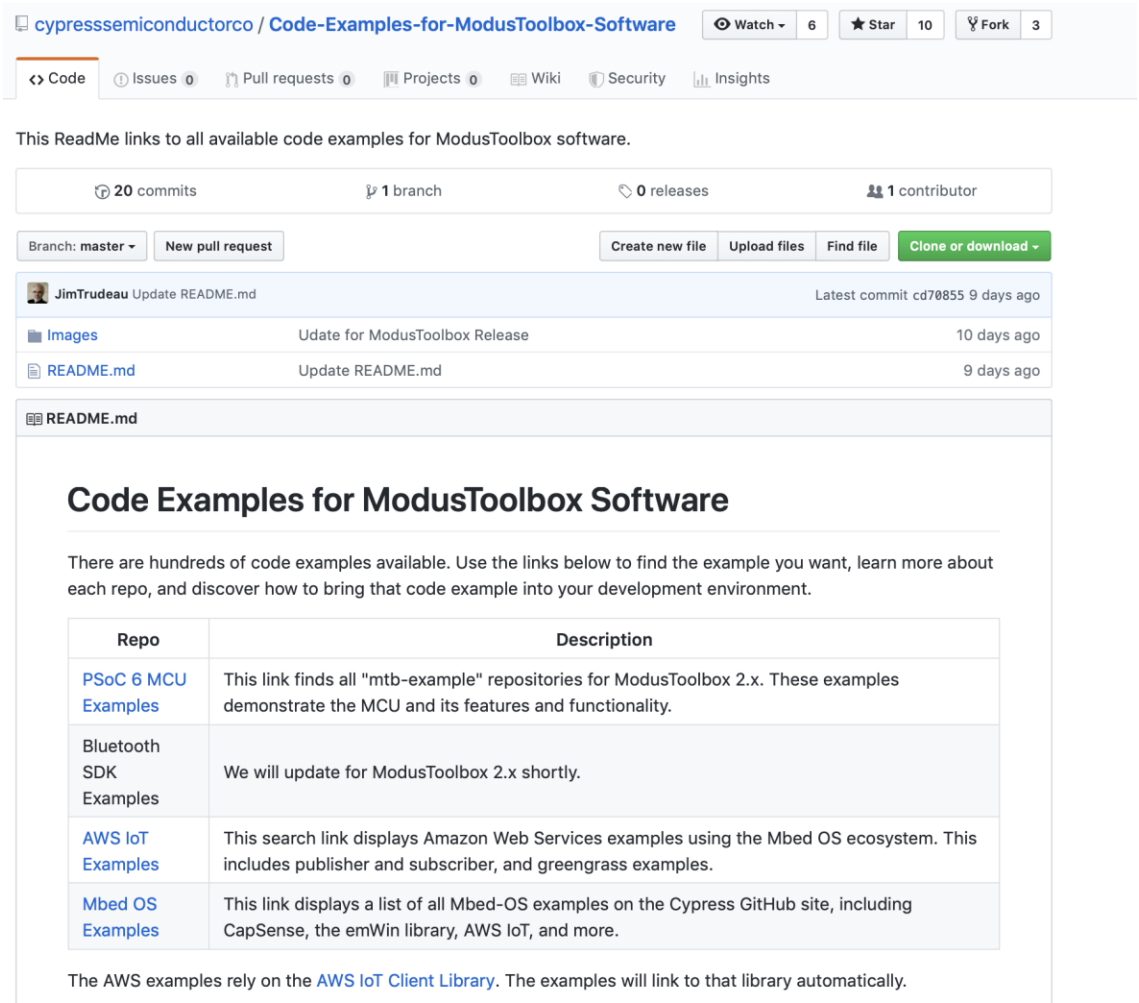
The Project Creator tool described above also has a command line version that you can run from a shell (e.g. Cygwin or modus-shell). It is in the same folder as the Project Creator GUI (ModusToolbox/tools_2.0/project-creator) but the executable is called project-creator-cli.exe. You can run it with no arguments to see the different options available including listing all the available BSPs (--list-boards) and all the available starter applications for a given BSP (--list-apps <BSP_Name>).

You can also use GitHub or use Git commands directly to clone a project. First, create a directory to hold your applications (e.g. mtb-examples), and change to that directory:

```
mkdir mtb-examples
cd mtb-examples
```

Using GitHub

You can get to the Cypress GitHub repo using the "Search Online for Code Examples" link in the ModusToolbox 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.



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:

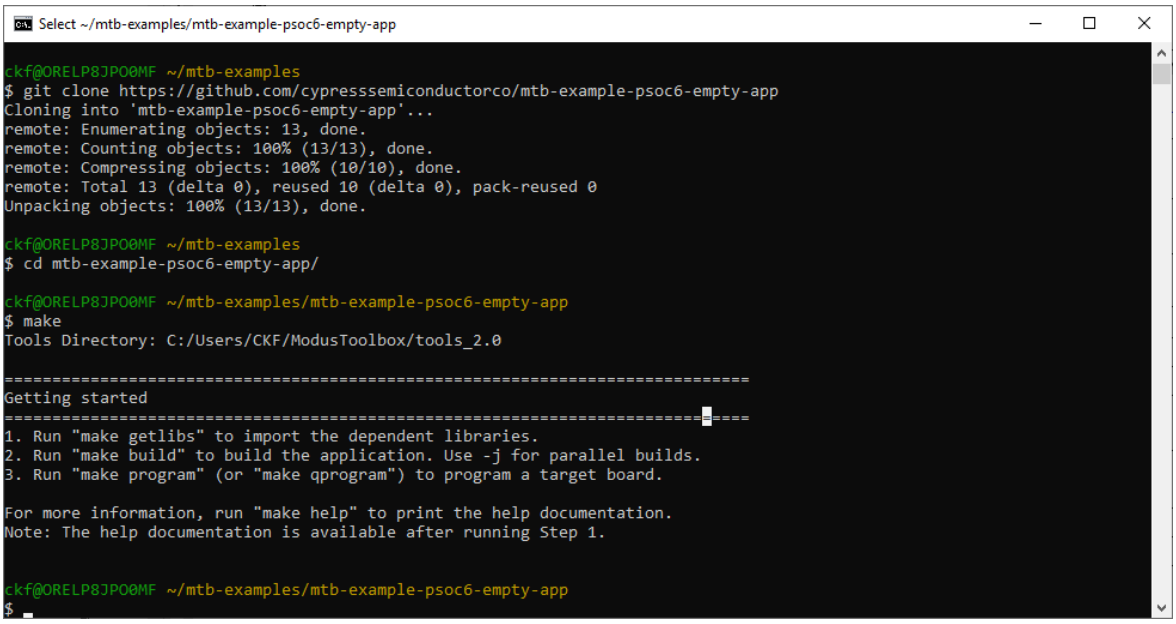
```
git clone https://github.com/cypresssemiconductorco/mtb-example-psoc6-empty-app
```

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:



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 5.6 Libraries & Library Manager for more details about libraries.

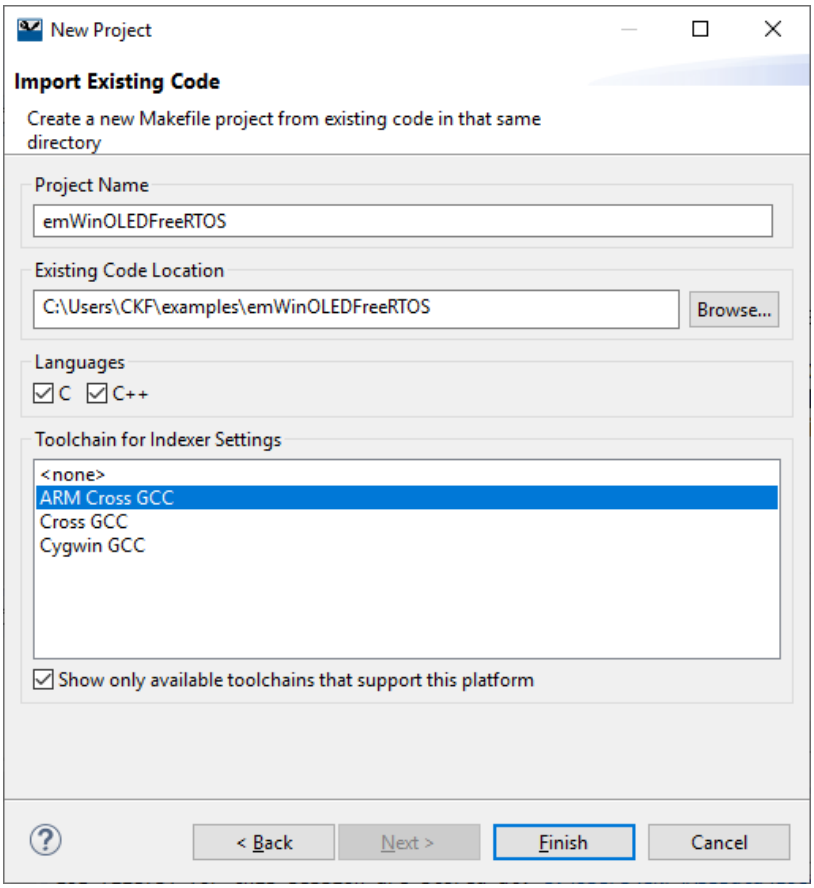
5.4.3 Importing Projects

Eclipse

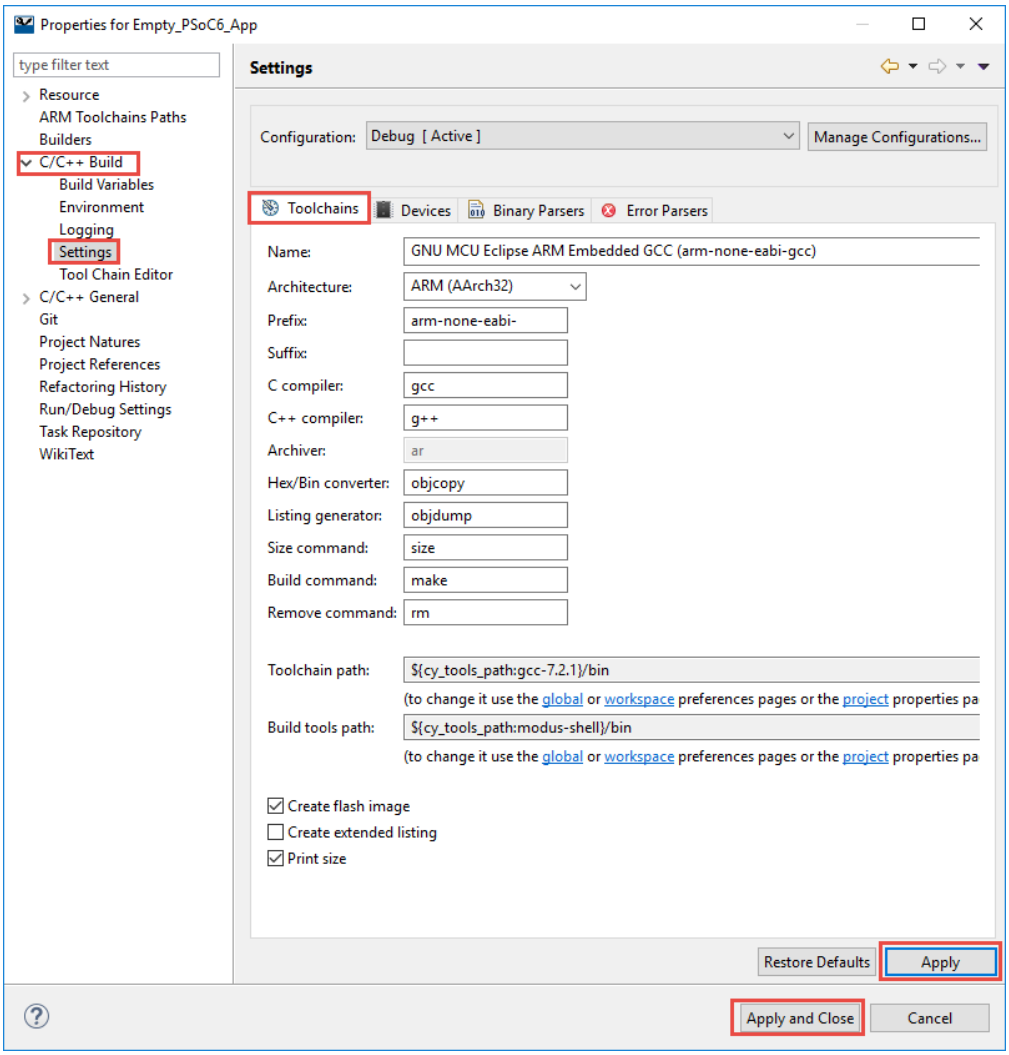
To import a project created with the Project Creator or command line into Eclipse, use the `make eclipse CY_IDE_PRJNAME=<project_name>` option where `<project_name>` is the name you will use for the project in Eclipse. It is typically the name of the directory containing the project.

Note: If you don't use the same name in both places, the Launches area of the Quick Panel will be blank after you import.

Then you'll need to use the Eclipse **File > Import > C/C++ > Existing Code as Makefile Project** function. Select ARM Cross GCC for the Toolchain as shown below.

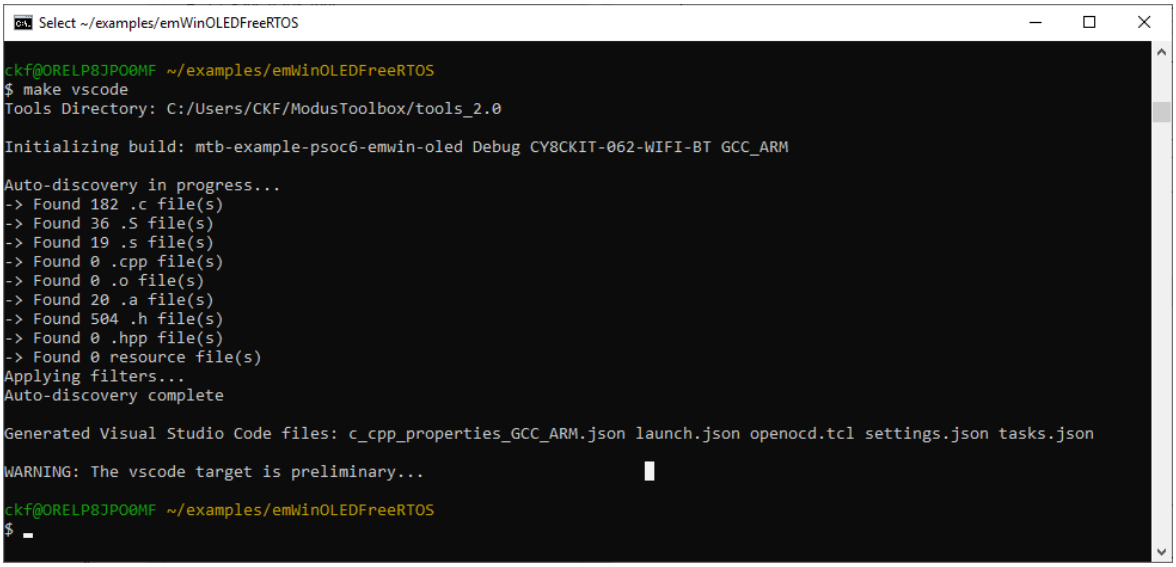


Once the project is created, right click on the project and select Properties. Got to *C/C++ Build > Settings > Toolchains*. Click *Apply* and then *Apply and Close* (you don't need to change anything but Apply is necessary to get the settings set up correctly in the project.



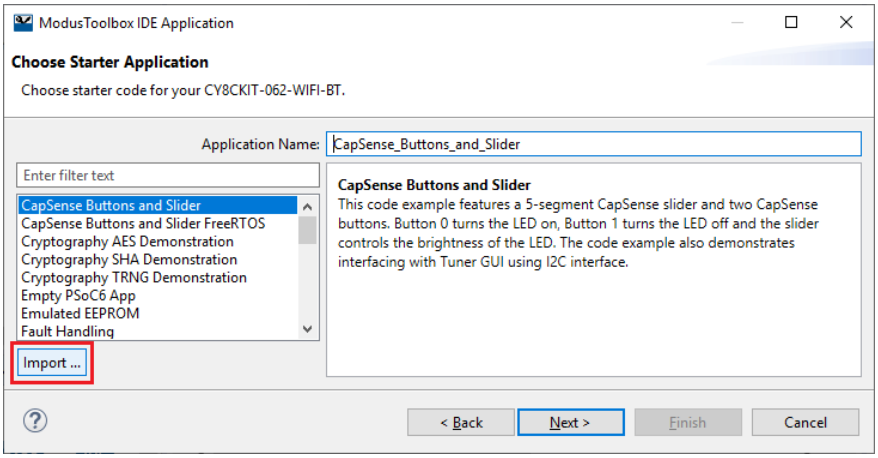
VS Code

To import a project into VS Code, use the `make vscode` option as discussed previously. See section 5.1.5 Visual Studio (VS) Code (Alpha) for more details.

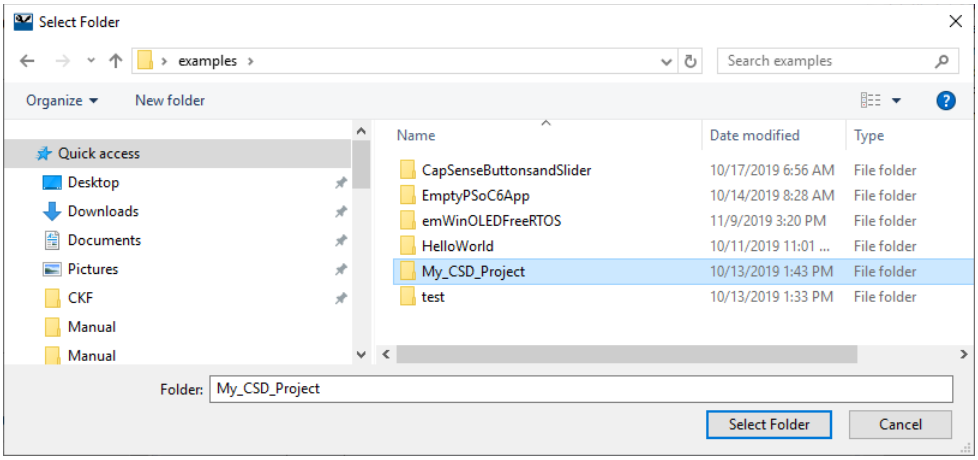


From an Existing Project

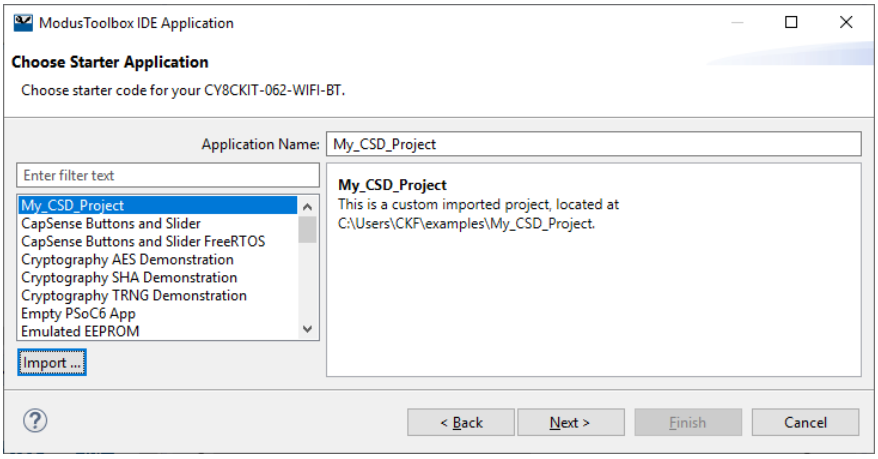
Another option is to base your project on an existing project that you have on your computer. To do this, instead of choosing a starter application, click the **Import** … button on the IDE New Application wizard or Project Creator tool.



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.



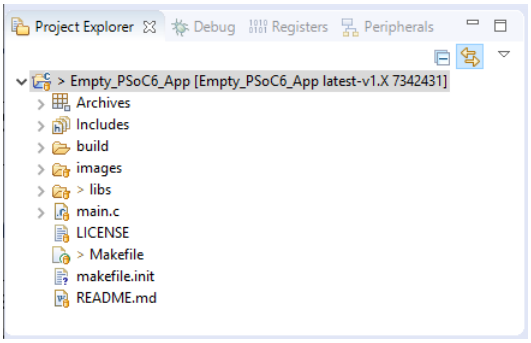
Note that the existing project can be one that is in your workspace or one that is located somewhere else. It does not need to be a full Eclipse project. At a minimum, it needs a Makefile and source code files, but it may contain other items such as configurator files and .lib files.



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.

5.5 Project Directory Organization

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



5.5.1 Archives

Virtual directory that shows static libraries.

5.5.2 Includes

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

5.5.3 build

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

5.5.4 images

This directory contains artwork images used by the documentation.

5.5.5 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.

5.5.6 main.c

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

5.5.7 LICENSE

This is the ModusToolbox license agreement file.

5.5.8 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 ModusToolbox 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
```

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=
```

5.5.9 Makefile.init

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

5.5.10 README.md

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

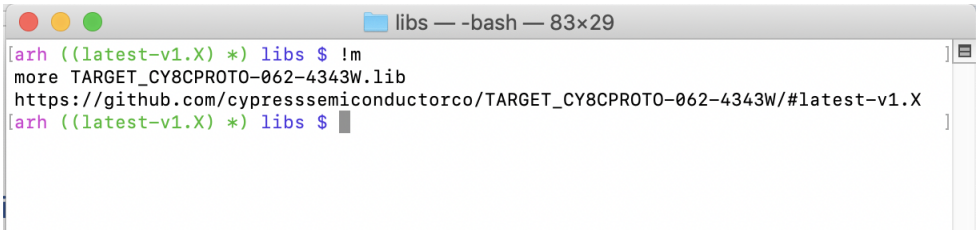
5.6 Libraries & Library Manager

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

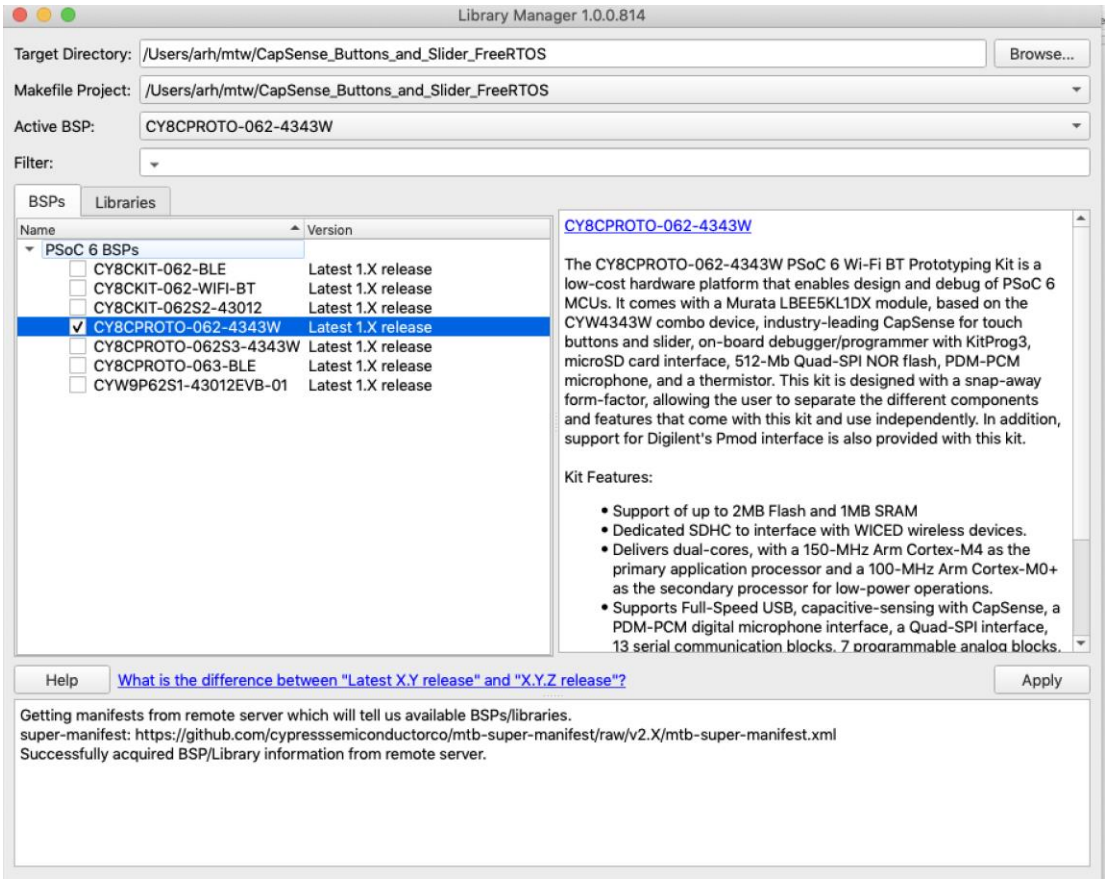
A typical library file will look something like this (notice this uses the tag name for the version).



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.

5.6.1 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_2.0/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.

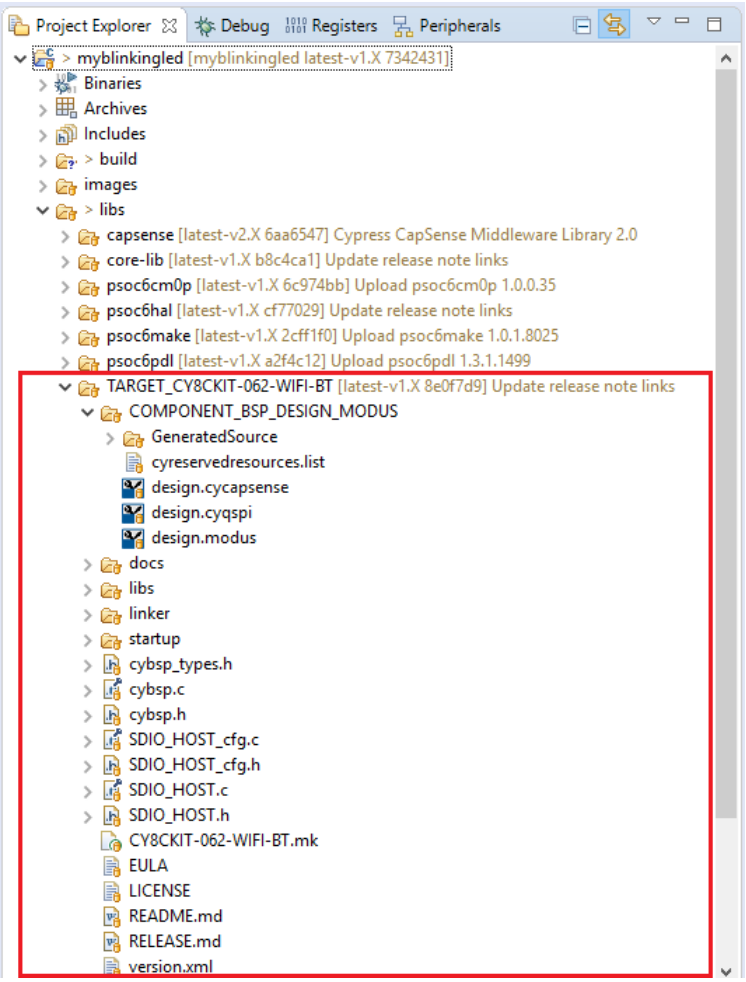
Libraries can be hierarchical, meaning that a library can have “.libs” inside of it and the `make getlibs` command will bring in all the libraries through the hierarchy.

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

5.7 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:

5.7.1 BSP Directory Structure



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.

libs

This directory contains the .lib files that specify the required libraries for the underlying device family. You can use the `make getlibs` command to fetch these libraries. The IDE's New Application wizard, or the stand-alone Project Creator tool, invoke `make getlibs` automatically.

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.

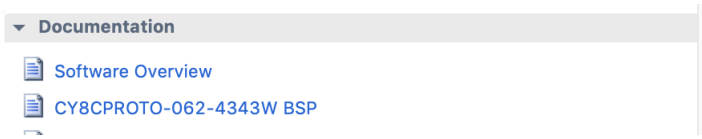
You need to include only cybsp.h in your application to use all the features of a BSP. Call the cybsp_init() function from your code to initialize the board (that function is defined in cybsp.c).

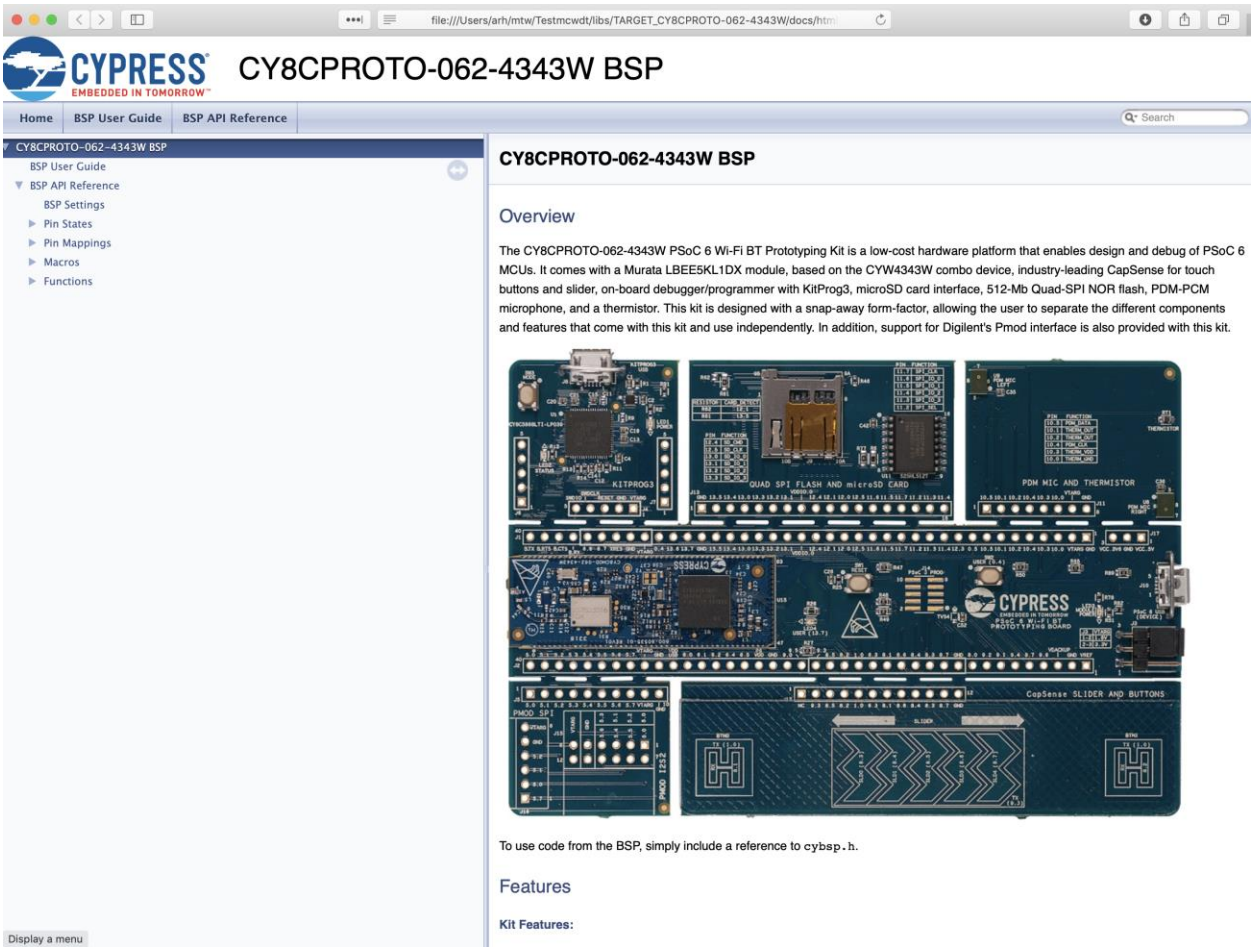
<targetname>.mk

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

5.7.2 BSP Documentation

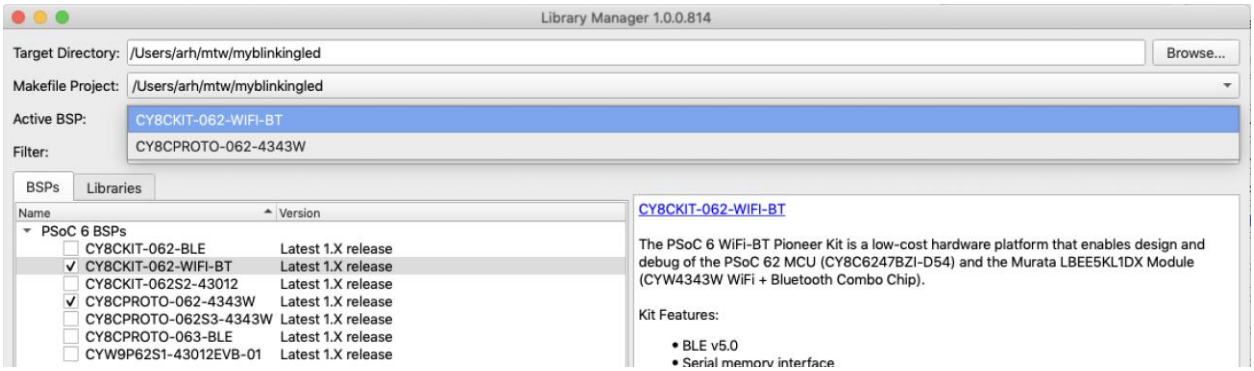
Each BSP provides HTML documentation that is specific to the selected board. It also includes an API Reference and the BSP User Guide. After creating a project, there is a link to the BSP documentation in the IDE Quick Panel. As mentioned previously, this documentation is located in the BSP's "docs"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



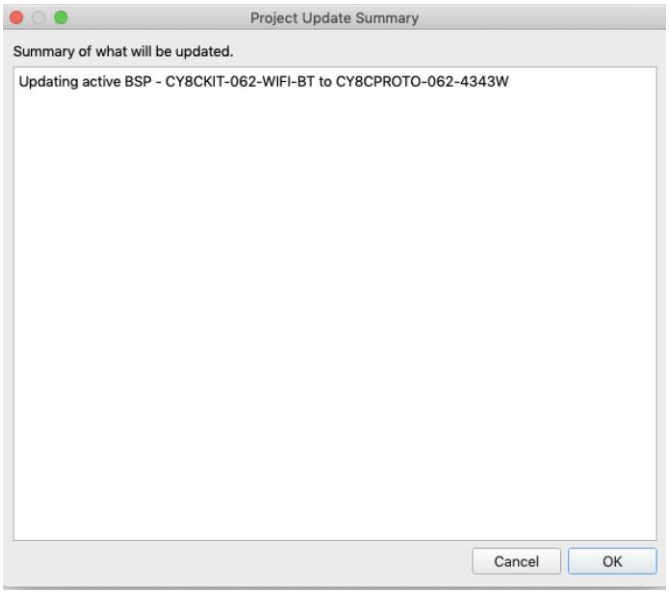


5.7.3 Changing the BSP with the Library Manager

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.



Click **Apply** to see a Summary of the changes to be made. Then click **OK** to update your application.



5.7.4 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
```

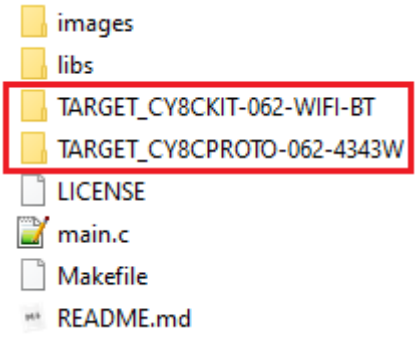
IMPORTANT: This will change the project being built by the IDE, but it will **NOT** update the Launches in the Quick Panel in the Eclipse IDE, so it does not affect which hex file is copied to the kit. Therefore, it is recommended to use the Library Manger's Active BSP selection from inside the Eclipse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

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

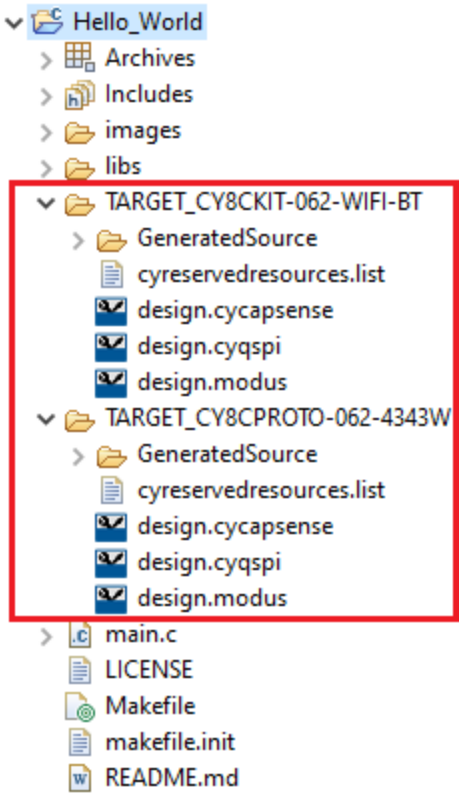
If you want to modify the BSP configuration for a single application (such as different pin or peripheral settings), it is preferable to NOT modify the BSP directly since that results in changes to the BSP library which would prevent you from updating the repository in the future. Instead, there is a 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.

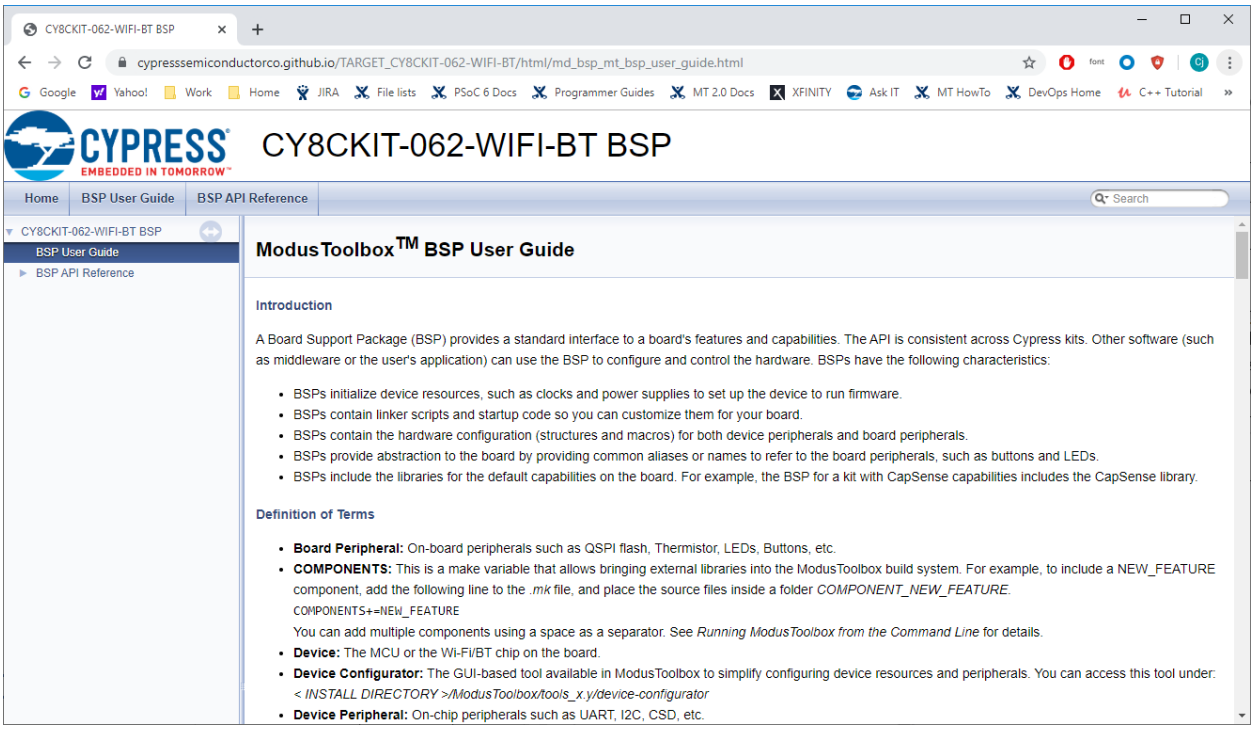
Note that the makefile already contains blank COMPONENTS and DISABLE_COMPONENTS lines where you can add the appropriate names.

5.7.6 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.

Each BSP document includes a link to the BSP User Guide, which contains a section for creating your own BSP:

https://cypresssemiconductorco.github.io/TARGET_CY8CKIT-062-WIFI-BT/html/md_bsp_mt_bsp_user_guide.html



The basic steps are:

1. Create a project using the closest-matching BSP and make a copy of the TARGET_xxx directory with a name to match your board. For example, TARGET_MYBOARDNAME.
2. Rename the .mk file to match the board name. For example, MYBOARDNAME.mk.
3. Update the DEVICE variable in the .mk file with the correct part number of the device/MCU if it is different than the BSP that you started from.
4. Update the linker script and startup code if required.
5. If you changed DEVICE:
    a. In your BSP's COMPONENT_BSP_DESIGN_MODUS directory, delete the entire GeneratedSource directory.
    b. Using the command line in your project space, run `make build TARGET=MYBOARDNAME`. The build system will automatically update the DEVICE in the

design.modus file, build the generated source, and attempt to build the project for the new target.

c. The BSP used as your starting point may have source that is not needed by your custom BSP. In particular, if you encounter issues with SDIO_HOST or SDIO_HOST_cfg source files, delete these.

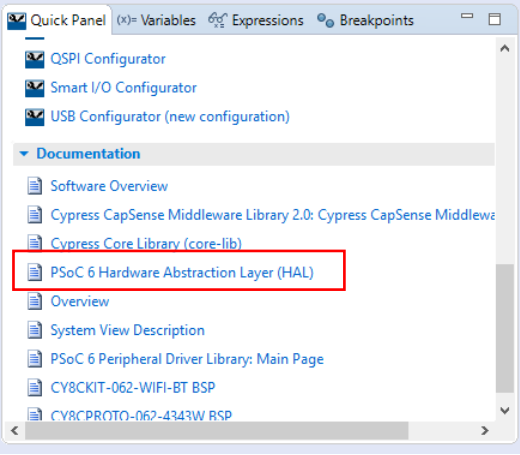
6. Define or update the aliases for pins in the cybsp_types.h file.
7. Customize the design.modus file and other configuration files with new settings for clocks, power supplies, and peripherals as required.
8. Update the cybsp_init() function in the cybsp.c file to correctly initialize your board.
9. Update the comment blocks in the cybsp.h file with any custom configuration used by your board and update the contents of README.md file to match your board.
10. In the project Makefile, change the BSP with TARGET=MYBOARDNAME.
11. In order to use your board to create new projects, you will also need to create appropriate manifest files and upload them to GitHub or some other server. See the Manifests section.

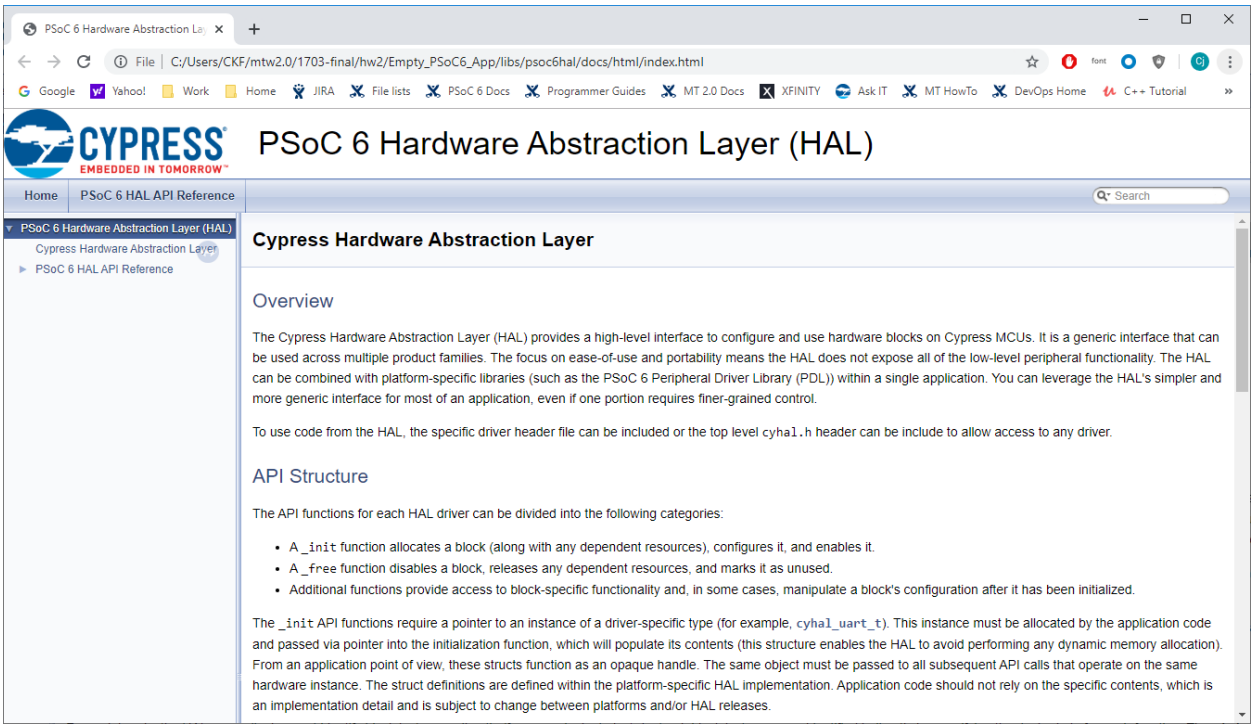
There is a command line make target called "bsp" that can be used to automate many of the steps above. Run `make help CY_HELP=bsp` for additional information.

5.8 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."

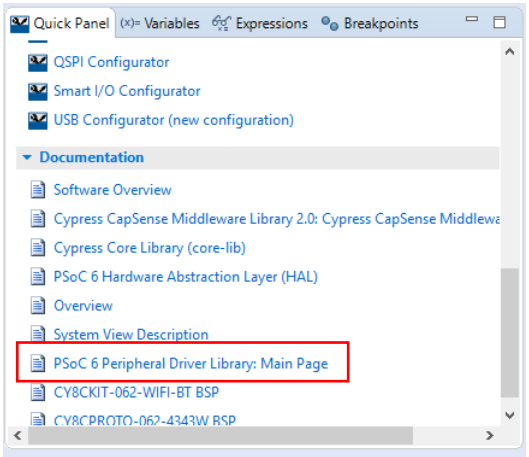


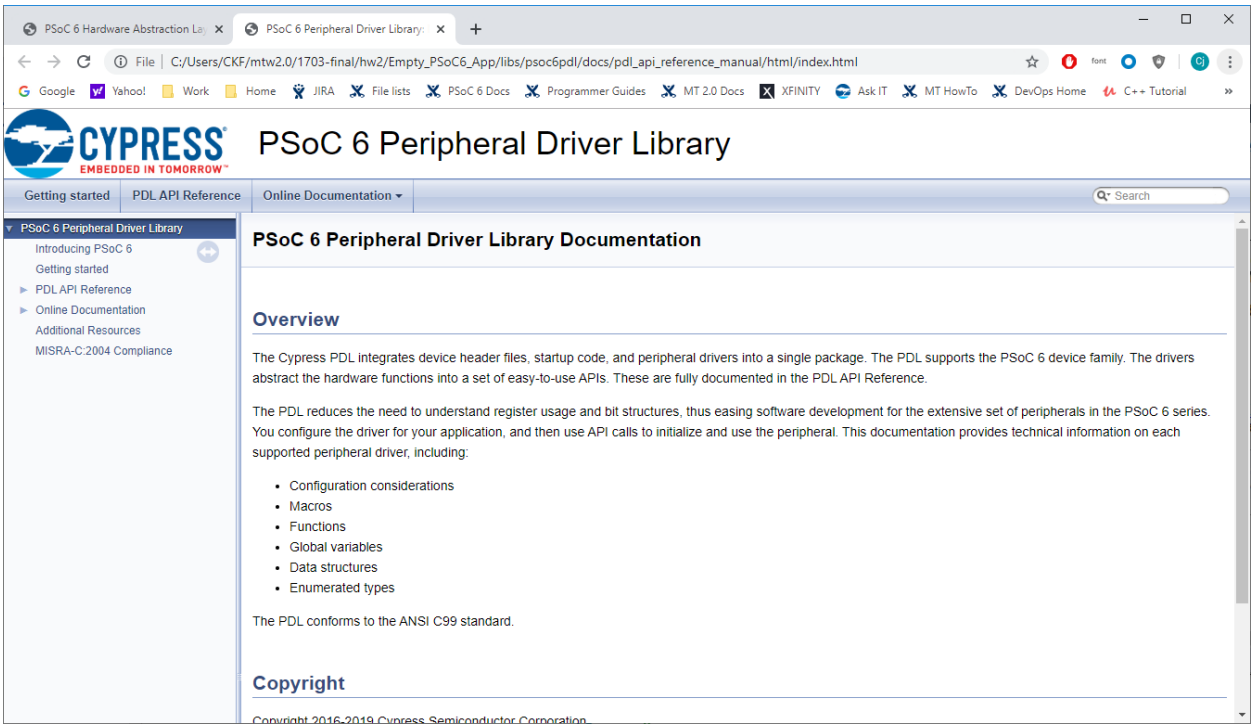


5.9 PDL

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."





5.10 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. Note that, currently, the manifest files must reside in a Git repo. Future releases of ModusToolbox will support local 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
```

5.11 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

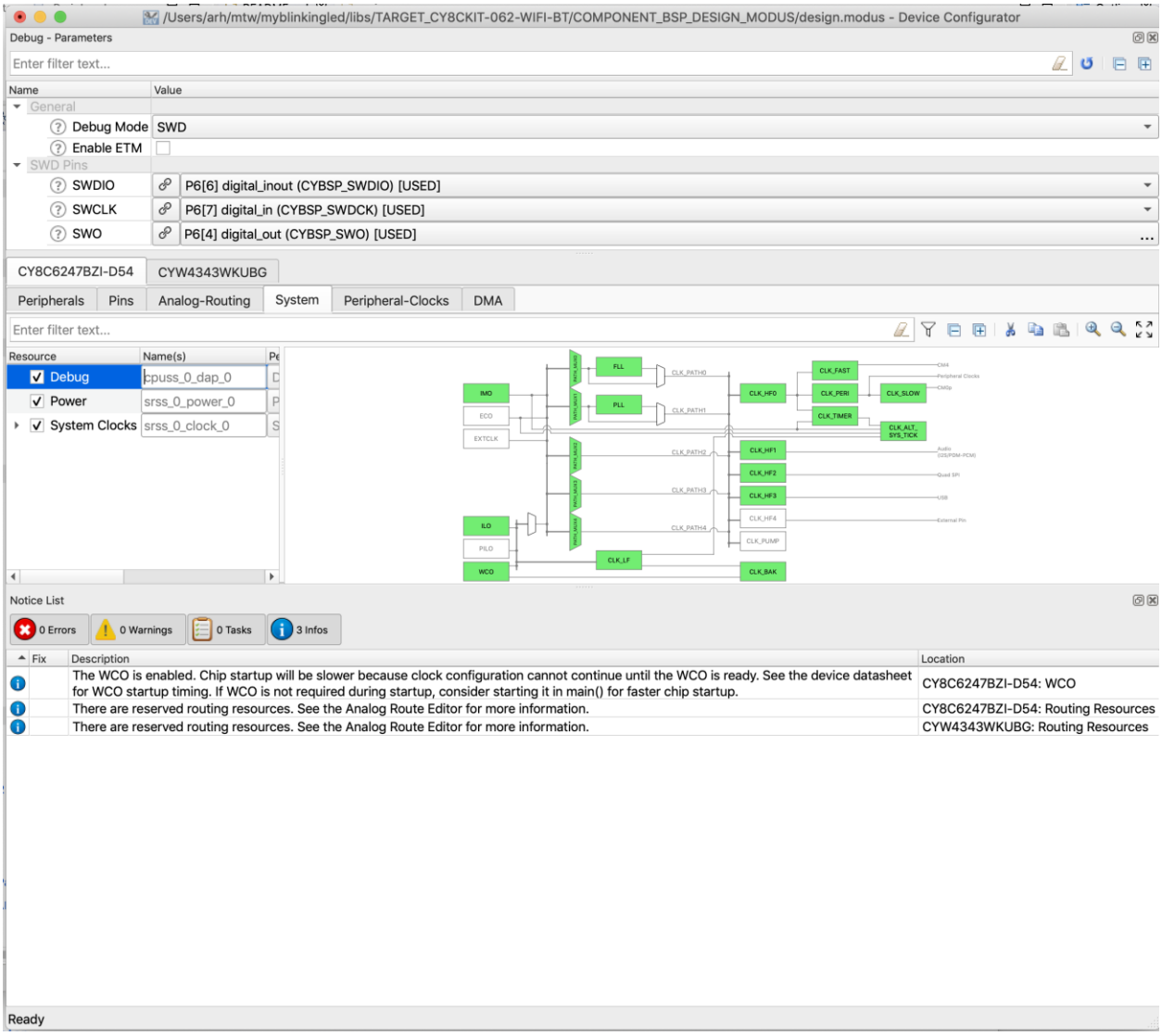
Configurators are divided into two types: Board Support Package (BSP) Configurators and Library Configurators.

5.11.1 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.

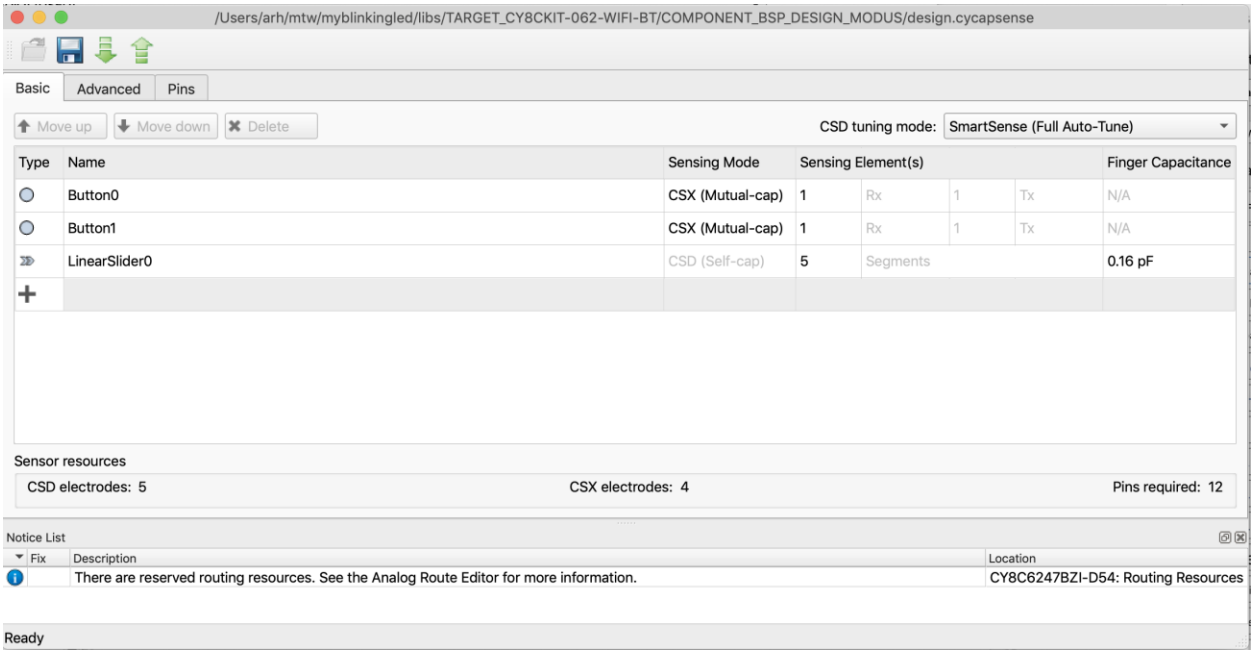
Device Configurator

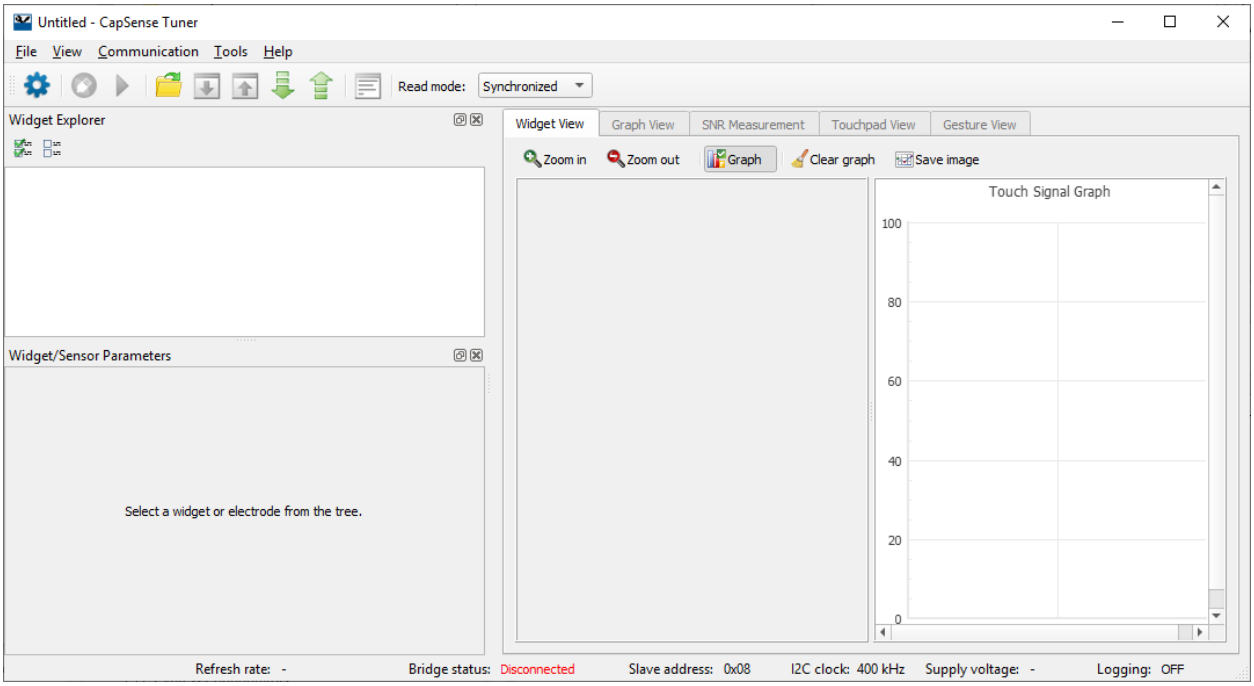
Set up the system (platform) functions such as pins, interrupts, clocks, and DMA, as well as the basic peripherals, including UART, Timer, etc.



CapSense Configurator/Tuner

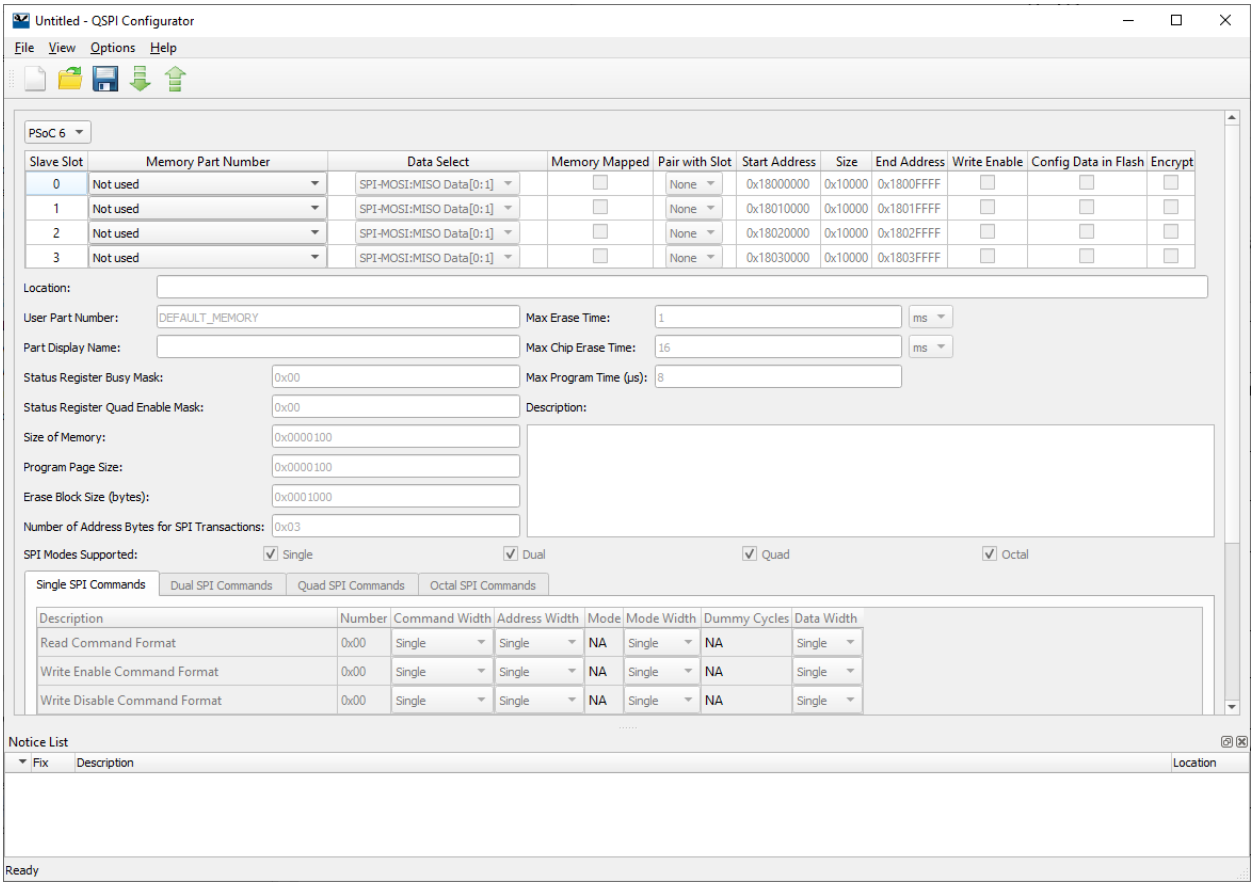
Configure CapSense hardware, and generate the required firmware. This includes tasks such as mapping pins to sensors and how the sensors are scanned.





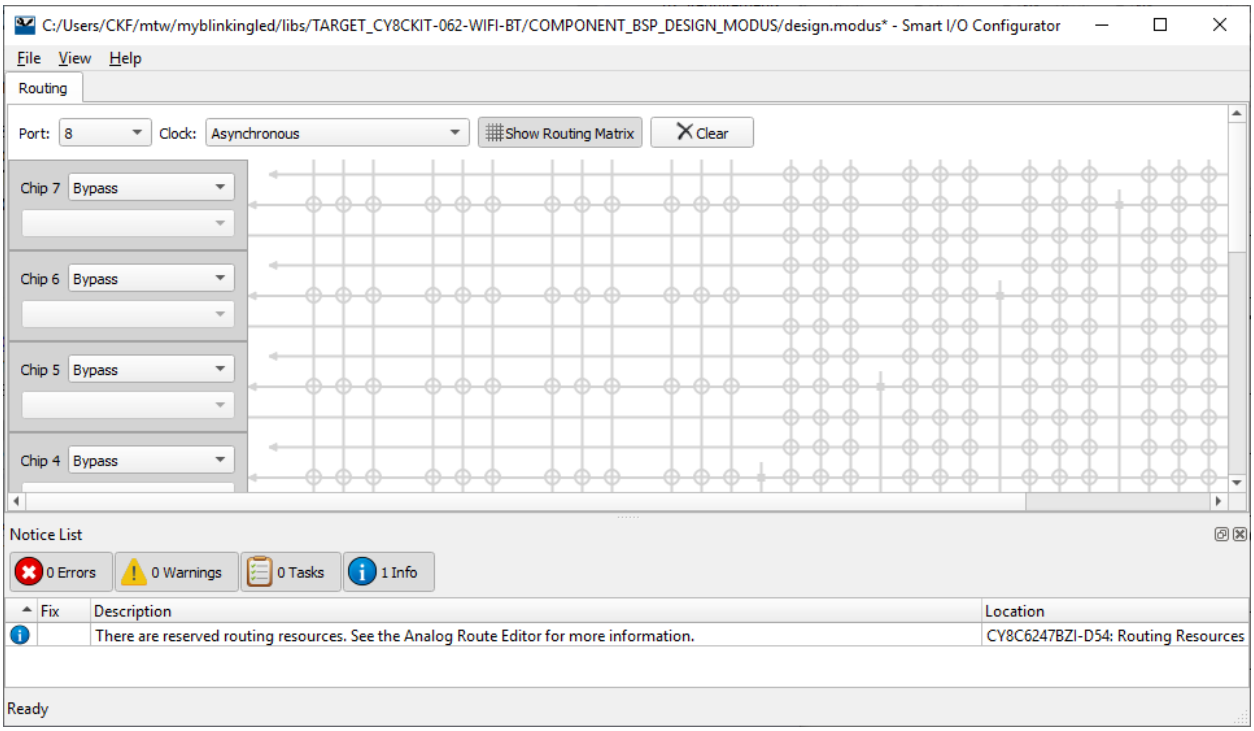
QSPI Configurator

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



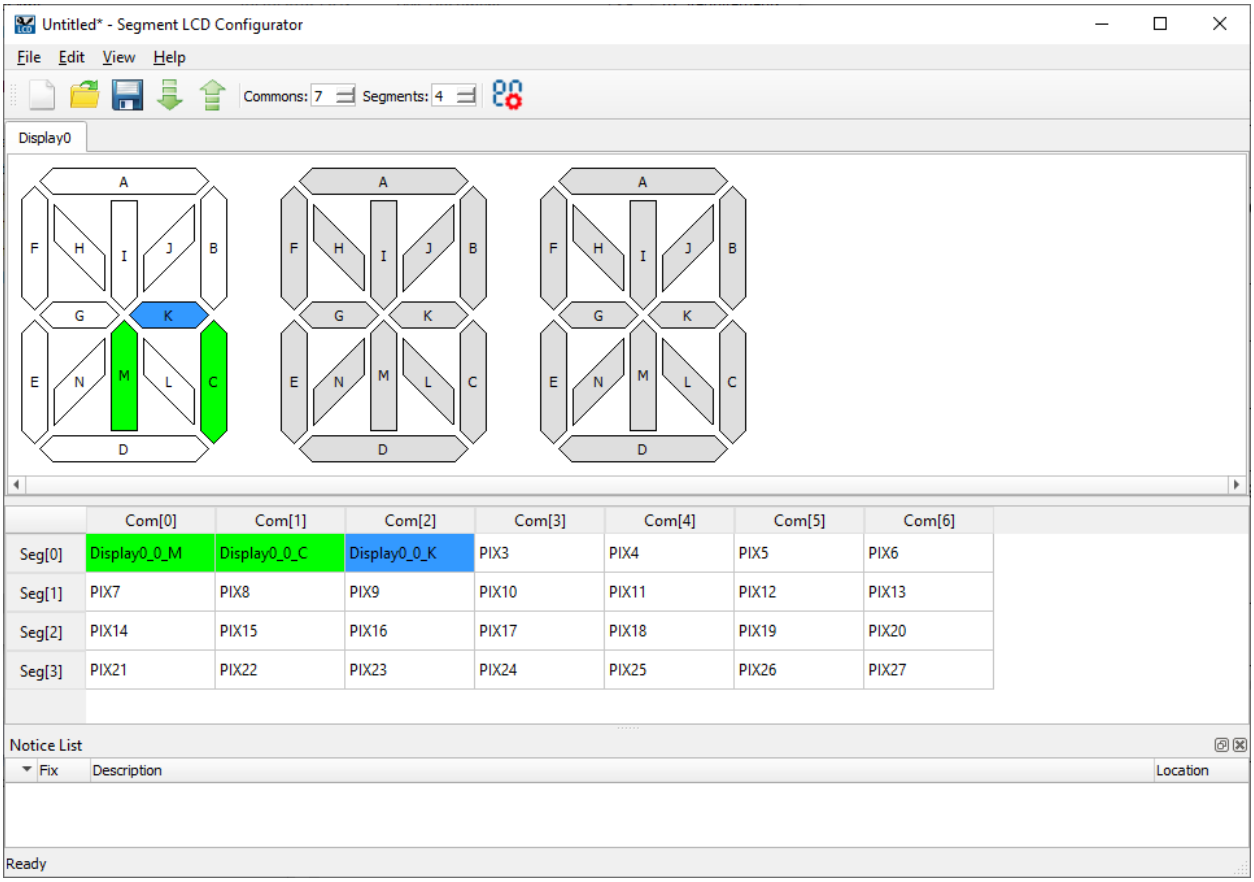
Smart I/O™ Configurator

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.

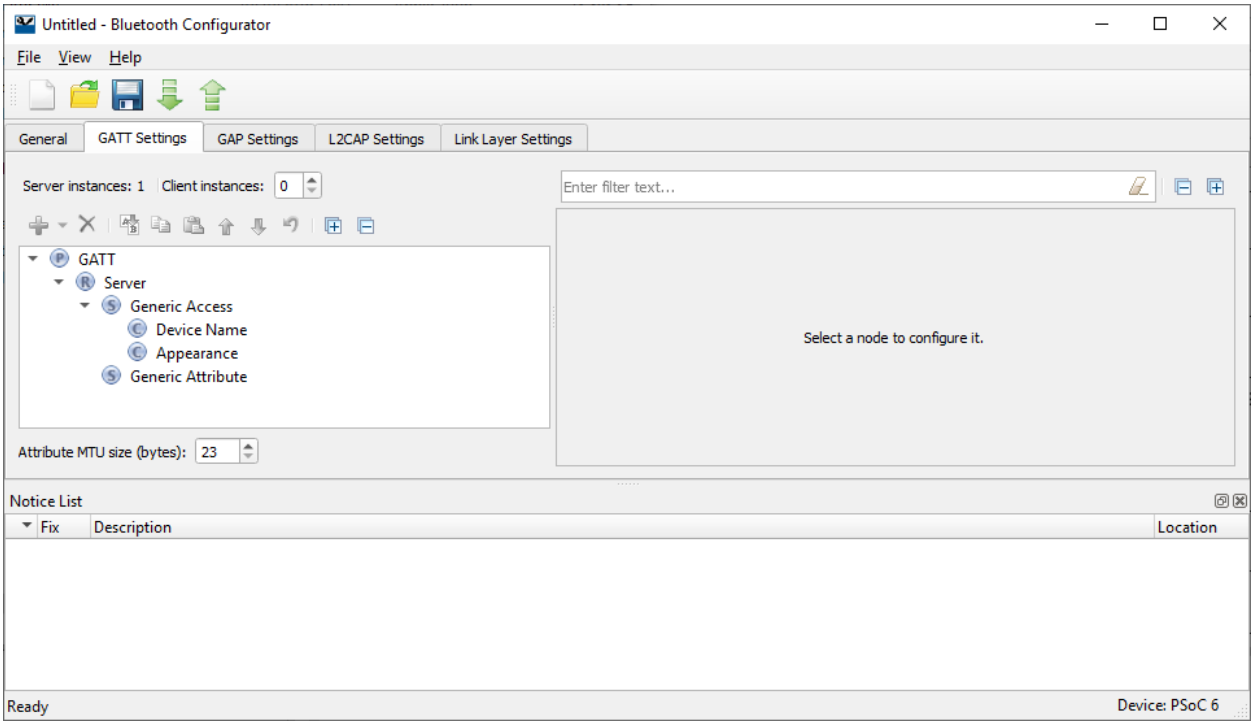


5.11.2 Library configurators

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.

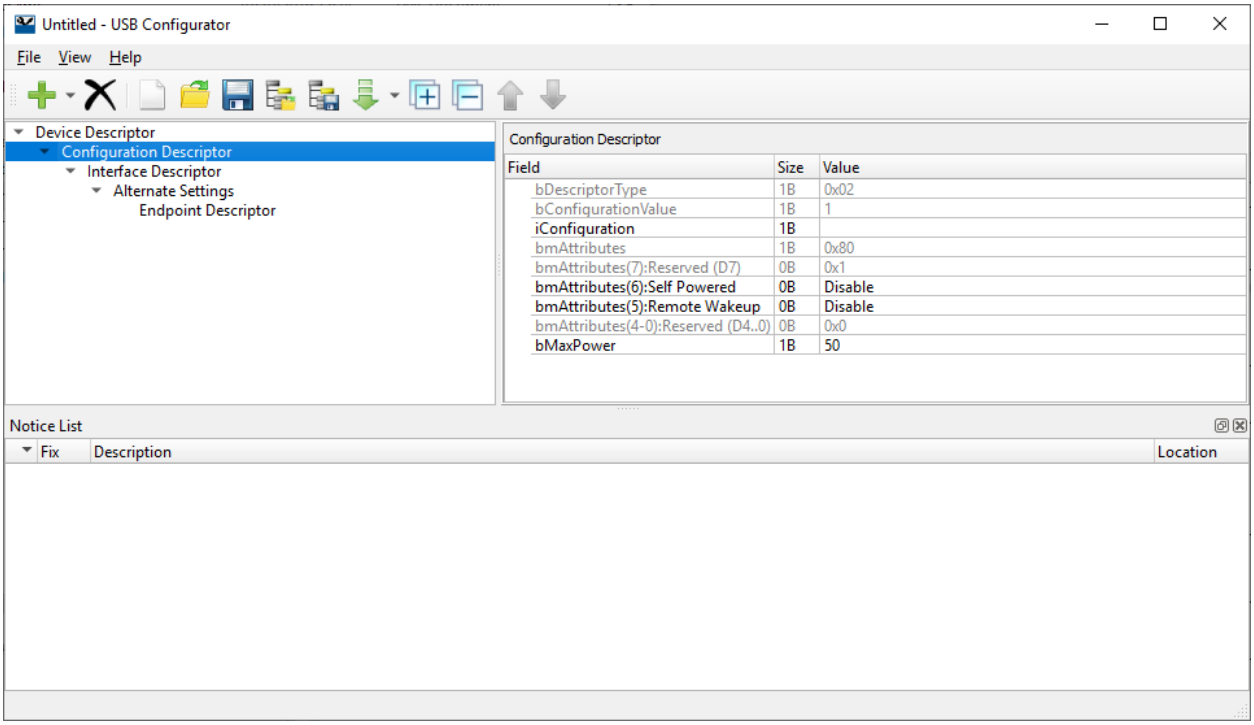
Bluetooth Configurator

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.



5.12 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.

5.12.1 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, 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.

5.12.2 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.

5.12.3 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.

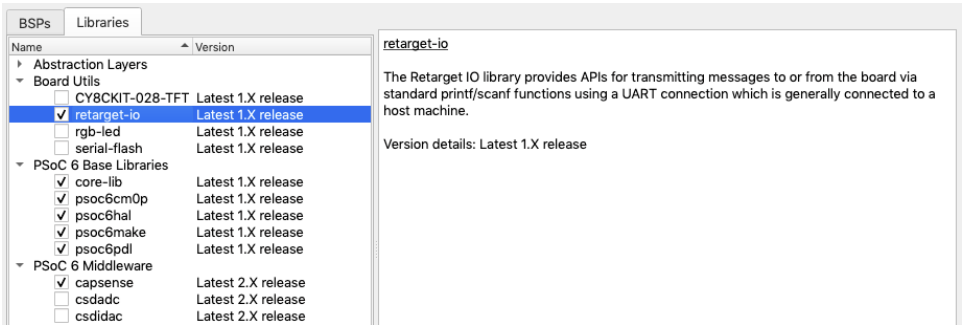
   Note: If you are working from the command line, you will need to add the BSP for your kit. The following commands create the project, add the BSP, build and program the kit.

   ```
   $ 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_2.0\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.

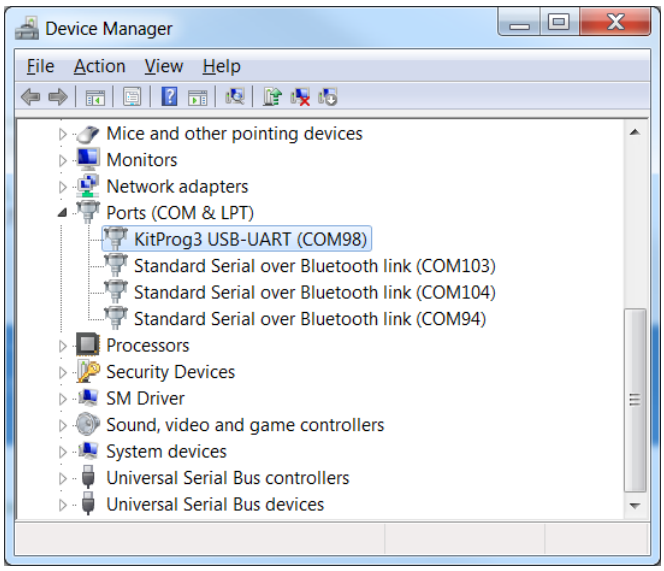
```
#include <stdio.h>
```

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

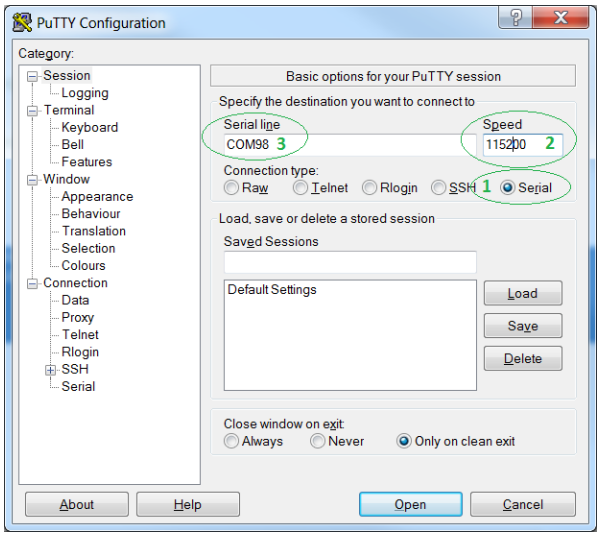
```
result = cy_retarget_io_init(CYBSP_DEBUG_UART_TX, CYBSP_DEBUG_UART_RX, \
CY_RETARGET_IO_BAUDRATE);
CY_ASSERT(result == CY_RSLT_SUCCESS);
```

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.

5.12.4 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.

5.12.5 CapSense Buttons and Slider

Build the example code

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

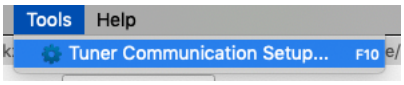
   Note: If you are working from the command line you will need to add the BSP for your kit.
2. Build the project and program the board.

Run the CapSense Tuner

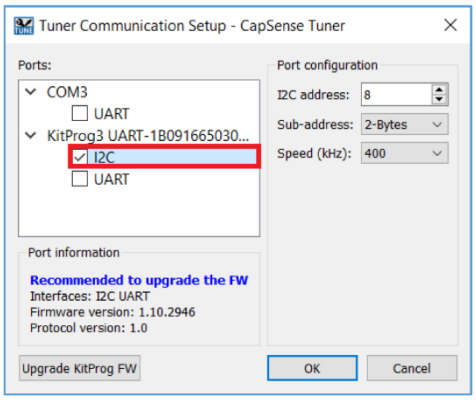
1. Launch the CapSense Tuner through the IDE or in stand-alone mode. Note that, in the latter case, you will need to manually open the CapSense configuration file:

   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

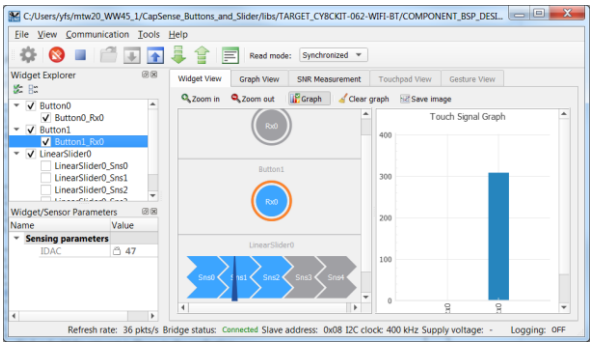
5. In the Tuner, click the **Connect** button to establish communication with the target (F4).

   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).

5.12.6 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.

5.12.7 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.

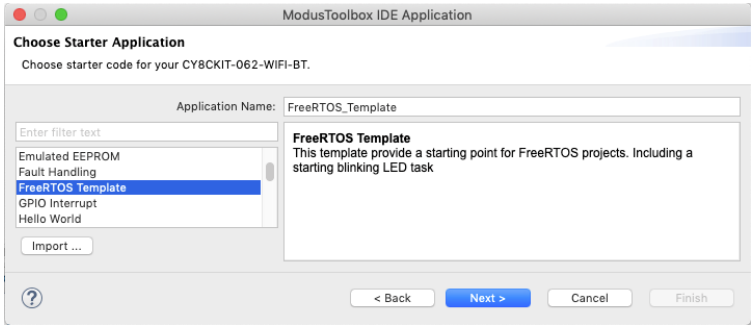
```
xTaskCreate(LED1_Task, "LED1", configMINIMAL_STACK_SIZE,0 /* args */ ,0 /*
priority */, NULL /* handle */);
vTaskStartScheduler();
```

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.

5.12.8 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”.

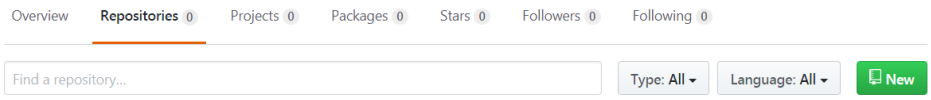
Note: There is a template directory in the library with default usrcmd.* files.

9. Build and Program.
10. Open a terminal emulator to test the shell.

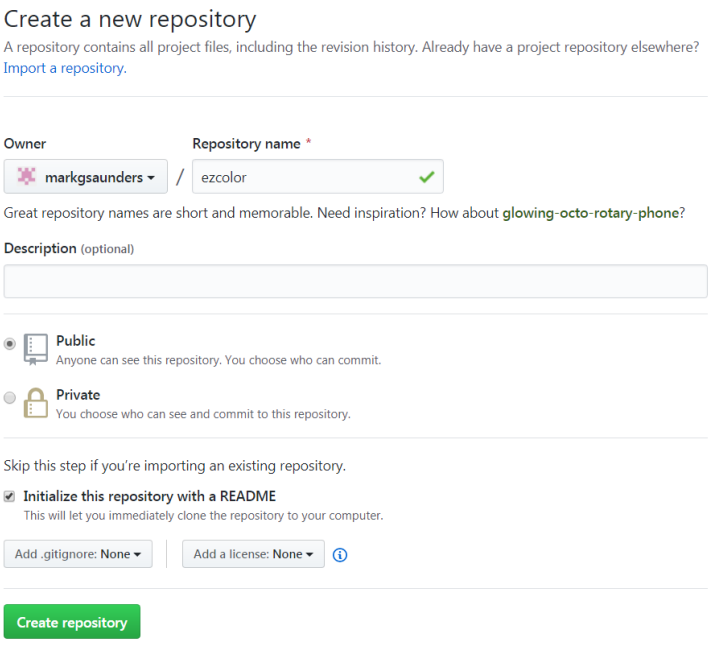
5.12.9 Create and Use a Library

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.



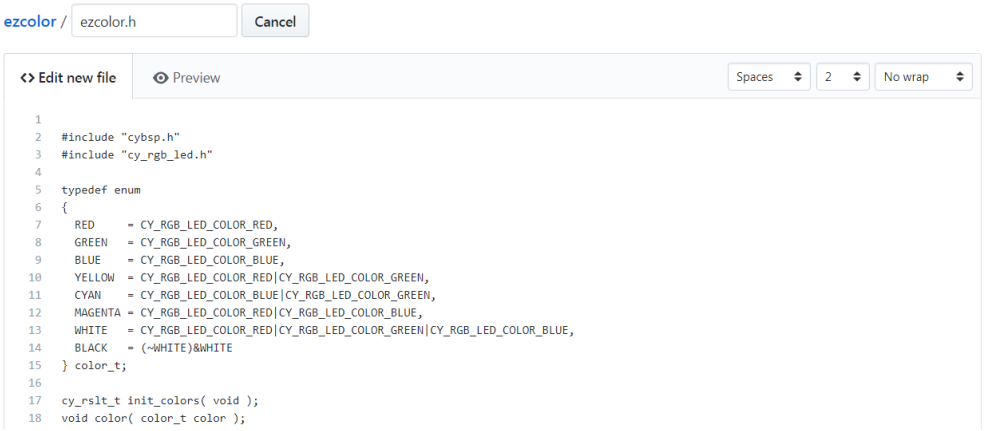
4. Give your repo a name and description. Make it public and let it create a README.md file. Press the Create Repository button.



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 );
```



8. Press the Commit new file button.

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

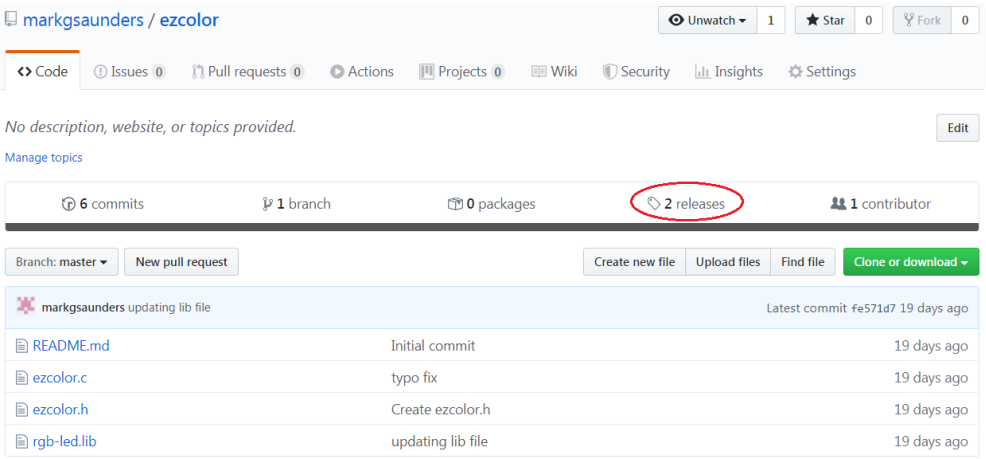
```
#include "ezcolor.h"
cy_rslt_t init_colors( void )
{
  return cy_rgb_led_init( CYBSP_LED_RGB_RED, CYBSP_LED_RGB_GREEN,
CYBSP_LED_RGB_BLUE, CY_RGB_LED_ACTIVE_LOW );
}
void color( color_t color )
{
  cy_rgb_led_on( color, CY_RGB_LED_MAX_BRIGHTNESS );
}
```

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

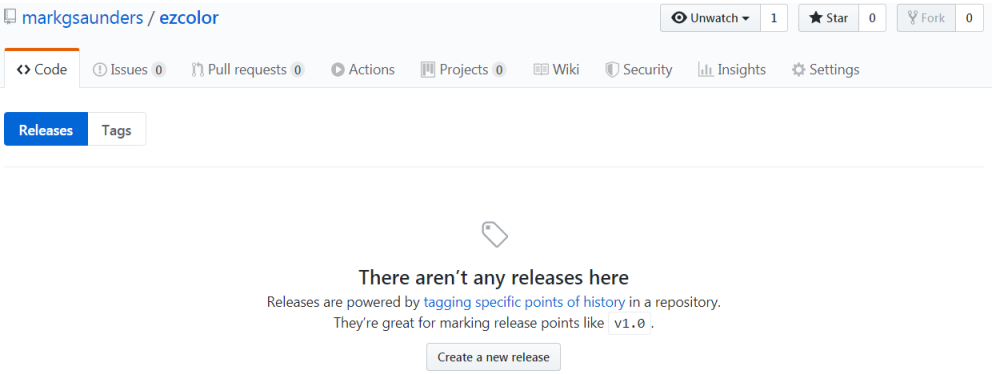
    `https://github.com/cypresssemiconductorco/rgb-led/#latest-v1.X`

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

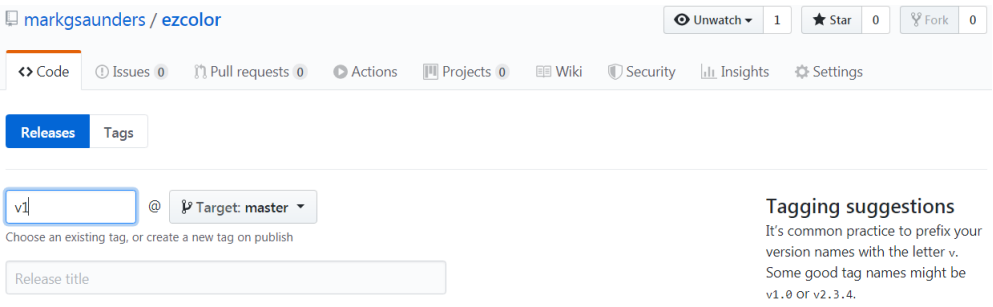
11. Your repo should now contain four files; the readme, a header, a source file, and the lib file.



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



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



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

```
~/ModusToolbox/tools_2.0/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_2.0
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/psoc6pdl.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.

5.12.10 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
   ```