

## Getting Started with PSoC 6 MCU

**Authors:** Srinivas Nudurupati, Vaisakh K V

**Associated Part Family:** All **PSoc® 6 MCU** devices

**Software Version:** **ModusToolbox™ 1.0**, **PSoc Creator™ 4.2**

**Associated Application Notes and Code Examples:** [Click here.](#)

### More code examples? We heard you.

To access an ever-growing list of hundreds of PSoC code examples, please visit our [code examples web page](#). You can also explore the PSoC video library [here](#).

AN221774 introduces the PSoC 6 MCU, a dual-CPU programmable system-on-chip with Arm® Cortex®-M4 and Cortex-M0+ processors. This application note helps you explore PSoC 6 MCU architecture and development tools, and shows you how to create your first project using ModusToolbox and PSoC Creator. This application note also guides you to more resources available online to accelerate your learning about PSoC 6 MCU. To get started with the PSoC 6 MCU with BLE Connectivity device family, refer to [AN210781](#) – Getting Started with PSoC 6 MCU with BLE Connectivity.

## Contents

1	Introduction.....	2	5.6	Part 4: Build the Application.....	32
1.1	Prerequisites .....	3	5.7	Part 5: Program the Device.....	33
2	Development Ecosystem.....	4	5.8	Part 6: Test Your Design.....	34
2.1	PSoc Resources .....	4	6	My First PSoC 6 MCU Design	
2.2	Firmware/Application Development .....	5		Using PSoC Creator .....	36
2.3	Support for Other IDEs .....	9	6.1	Using These Instructions .....	36
2.4	RTOS Support .....	9	6.2	About the Design .....	37
2.5	Debugging.....	11	6.3	Part 1: Create a New Project from Scratch.....	38
2.6	PSoC 6 MCU Development Kits .....	11	6.4	Part 2: Implement the Design .....	42
3	Device Features .....	12	6.5	Part 3: Generate Source Code .....	50
4	Choosing an IDE .....	13	6.6	Part 4: Write the Firmware.....	52
5	My First PSoC 6 MCU Design Using		6.7	Part 5: Build the Project and	
	ModusToolbox IDE .....	14		Program the Device .....	56
5.1	Using These Instructions .....	14	6.8	Part 6: Test Your Design.....	58
5.2	About the Design .....	14	7	Summary .....	59
5.3	Part 1: Create a New Application .....	15	8	Related Application Notes and Code Examples .....	60
5.4	Part 2: Implement the Design and			Appendix A. Glossary.....	62
	Generate Source Code .....	19		Appendix B. PSoC 6 MCU Development Kits .....	63
5.5	Part 3: Write the Firmware .....	28		Worldwide Sales and Design Support.....	68

# 1 Introduction

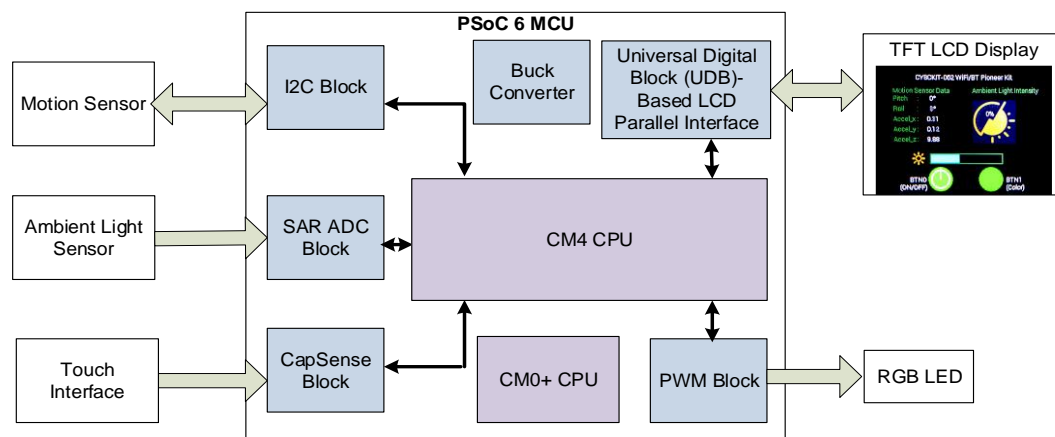
PSoC 6 MCU is Cypress' ultra-low-power PSoC device with a dual-CPU architecture tailored for smart homes, IoT gateways, etc. The PSoC 6 MCU device is a programmable embedded system-on-chip that integrates the following features on a single chip:

- Single-CPU microcontroller: Arm Cortex-M4 (CM4) or Dual-CPU microcontroller: Arm Cortex-M4 (CM4) and Cortex-M0+ (CM0+).
- Programmable analog and digital peripherals.
- Up to 2 MB of flash and 1 MB of SRAM.
- Fourth-generation CapSense® technology.
- PSoC 6 MCU is suitable for a variety of power-sensitive applications such as the following:
  - Smart home sensors and controllers.
  - Smart home appliances.
  - Gaming controllers.
  - Sports, smart phone, and virtual reality (VR) accessories.
  - Industrial sensor nodes.
  - Industrial logic controllers.
  - Advanced remote controllers.

The programmable analog and digital subsystems allow flexibility and dynamic fine-tuning of the design using [ModusToolbox™](#) IDE, the Eclipse-based IDE for developing PSoC 6 MCU applications, or [PSoC Creator](#), the schematic-based design tool.

[Figure 1](#) illustrates an application-level block diagram for a real-world use case using PSoC 6 MCU.

Figure 1. Application-Level Block Diagram Using PSoC 6 MCU



PSoC 6 MCU is a highly capable and flexible solution. For example, the real-world use case in [Figure 1](#) takes advantage of these features:

- A buck converter for ultra-low-power operation.
- An analog front end (AFE) within the device to condition and measure sensor outputs such as ambient light sensor.
- Serial Communication Blocks (SCBs) to interface with multiple digital sensors such as motion sensors.
- CapSense technology for reliable touch and proximity sensing.
- Digital logic (Universal Digital Blocks or UDBs) and peripherals (Timer Counter PWM or TCPWM) to drive the display and LEDs respectively.
- Product security features managed by CM0+ CPU and application features executed by CM4 CPU.

See [Device Features](#) and the device datasheets for more details.

This application note introduces you to the capabilities of PSoC 6 MCU, gives an overview of the development ecosystem, and gets you started with a simple design wherein you learn to use PSoC 6 MCU. This design is available as code example [CE221773](#) for ModusToolbox and PSoC Creator.

For hardware design considerations, see [AN218241 – PSoC 6 MCU Hardware Design Considerations](#).

## 1.1 Prerequisites

Before you get started, make sure that you have a development kit and have installed the required software. It is recommended that you download the code example for reference.

### 1.1.1 Hardware

- [CY8CKIT-062-WiFi-BT PSoC 6 Wi-Fi-BT Pioneer Kit](#)
- [CY8CKIT-062-BLE PSoC 6 BLE Pioneer Kit](#) or
- [CY8CPROTO-062-4343W PSoC 6 Wi-Fi BT Prototyping Kit](#) Note that kit is supported only on ModusToolbox; it is not supported in PSoC Creator.

### 1.1.2 Software

- [ModusToolbox 1.0](#) or
- [PSoC Creator 4.2](#) with [Peripheral Driver Library](#) (PDL v3.1.x or later)
- [CE221773](#) – PSoC 6 MCU Hello World Example Using ModusToolbox

## 2 Development Ecosystem

### 2.1 PSoC Resources

Cypress provides a wealth of data at [www.cypress.com](http://www.cypress.com) to help you to select the right PSoC device and quickly and effectively integrate it into your design. For a comprehensive list of PSoC 6 MCU resources, see [KBA223067](#) in the Cypress community. The following is an abbreviated list of resources for PSoC 6 MCU.

- **Overview:** [PSoC Portfolio](#), [PSoC Roadmap](#)
- **Product Selectors:** [PSoC 6 MCU](#)
- **Datasheets** describe and provide electrical specifications for each device family.
- **Application Notes and Code Examples** cover a broad range of topics, from basic to advanced level. You can also browse our collection of code examples. See [Code Examples](#).
- **Technical Reference Manuals (TRMs)** provide detailed descriptions of the architecture and registers in each device family.
- **PSoC 6 MCU Programming Specification** provides the information necessary to program the nonvolatile memory of PSoC 6 MCU devices.
- **CapSense Design Guides:** Learn how to design capacitive touch-sensing applications with PSoC devices.
- **Development Tools**
  - [CY8CKIT-062-WiFi-BT PSoC 6 Wi-Fi-BT Pioneer Kit](#) is a development kit that supports the PSoC 62 series MCU along with Wi-Fi and BT connectivity.
  - [CY8CKIT-062-BLE PSoC 6 BLE Pioneer Kit](#) is an easy-to-use and inexpensive development platform for PSoC 63 series MCU with BLE Connectivity.
  - [CY8CPROTO-062-4343W PSoC 6 Wi-Fi BT Prototyping Kit](#) is a development kit that supports the PSoC 62 series MCU along with CYW4343W module-based Wi-Fi and BT connectivity for development on ModusToolbox.
- **Training Videos:** Cypress provides [video training](#) on our products and tools, including a dedicated series on [PSoC 6 MCU](#).

## 2.2 Firmware/Application Development

Cypress provides two development platforms that you can use for application development with PSoC 6 MCU:

- **ModusToolbox:** An Eclipse-based development environment on Windows, macOS, and Linux platforms that includes the ModusToolbox IDE and the PSoC 6 SDK. ModusToolbox supports stand-alone device and middleware configurators that are fully integrated into the IDE. Use the configurators to set the configuration of different blocks in the device and generate code that can be used in firmware development. ModusToolbox supports all PSoC 6 MCU devices.

ModusToolbox relies on PSoC 6 SDK as the software development kit for all supported Cypress PSoC 6 devices. The SDK is provided as source code and in some cases, like BLE, as libraries. See [ModusToolbox IDE and the PSoC 6 SDK](#) for more information.

- **PSoC Creator:** A Cypress-proprietary IDE that runs on Windows only. It supports a subset of PSoC 6 MCU devices (up to 1 MB flash memory) as well as other PSoC device families such as PSoC 3, PSoC 4, and PSoC 5LP. See [PSoC Creator](#) for more information.

### 2.2.1 ModusToolbox IDE and the PSoC 6 SDK

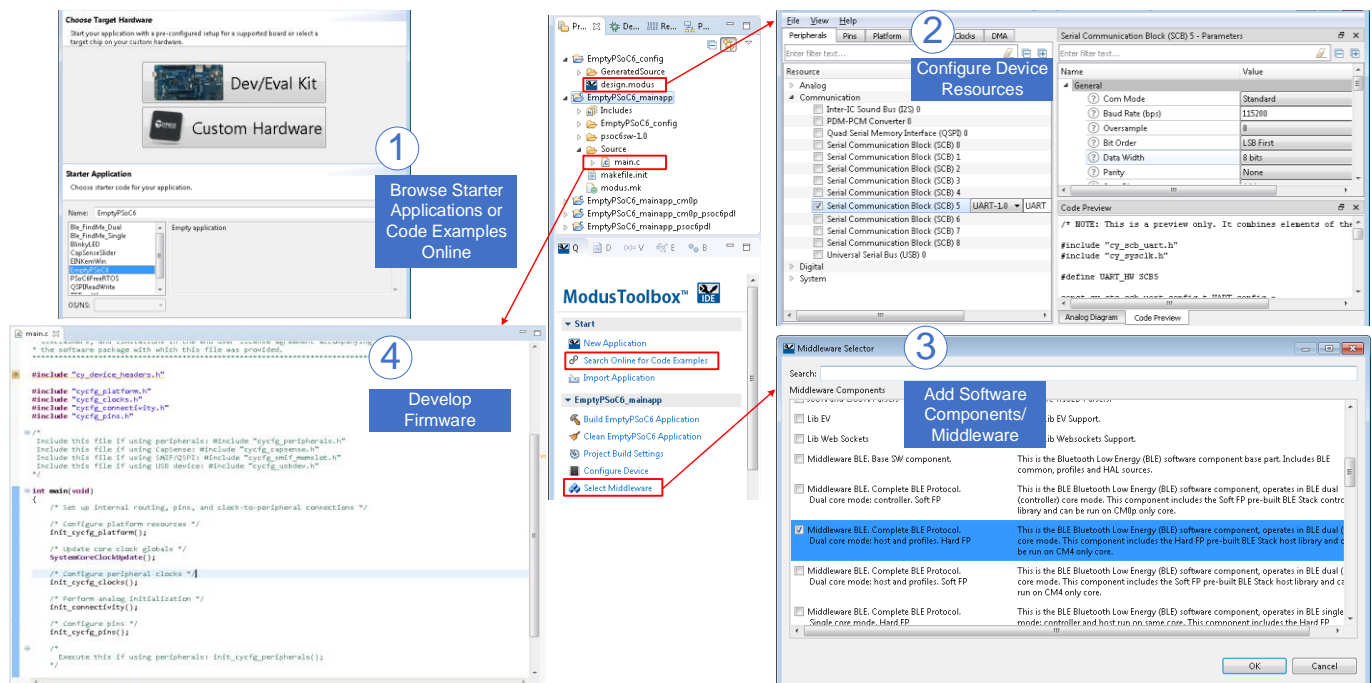
ModusToolbox is a free development ecosystem that includes the ModusToolbox IDE and the PSoC 6 SDK. The ModusToolbox IDE brings together several device resources, middleware, and firmware to build an application. Using ModusToolbox, you can enable and configure device resources and middleware libraries, write C/assembly source code, and program and debug the device.

The PSoC 6 SDK is the software development kit for the PSoC 6 MCU. The SDK makes it easier to develop firmware for supported devices. It helps you to build firmware without the need to understand the intricacies of the device resources.

As [Figure 2](#) shows, with the ModusToolbox IDE, you can:

1. Create a new application based on a list of starter applications, filtered by kit or device, or browse the collection of code examples online.
2. Configure device resources in *design.modus* to build your hardware system design in the workspace.
3. Add software components or middleware.
4. Develop your application firmware.

Figure 2. ModusToolbox IDE Resources and Middleware



### 2.2.1.1 ModusToolbox Help

Visit the [ModusToolbox](#) home page to download and install the latest version of ModusToolbox. Launch ModusToolbox and navigate to the following items:

- **Quick Start Guide:** Choose **Help > ModusToolbox IDE Documentation > Quick Start Guide**. This guide gives you the basics for using ModusToolbox.
- **PSoC 6 API Reference:** Choose **Help > ModusToolbox API Reference > PSoC PDL API Reference**. This guide gives you an insight into the PSoC 6 PDL APIs.

### 2.2.1.2 PSoC 6 SDK

The PSoC 6 SDK includes resource drivers and middleware configurators to get you started developing firmware with PSoC 6 MCU. The SDK includes the same driver code as found in the [Software Development Kits for PSoC 6 Devices](#).

The SDK provides the central core of the ModusToolbox software. It contains Configurators, drivers, libraries, middleware, as well as various utilities, makefiles, and scripts. It also includes relevant drivers, middleware, and examples for use with Cypress IoT devices and connectivity solutions. You may use any or all tools in any environment you prefer. The SDK is the one place where you can find all the development resources for PSoC 6 MCU devices.

#### 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 and set the baud rate, parity, stop bits. Upon saving the hardware configuration, the tool generates the "C" code to initialize the hardware with the desired configuration.

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 tools, or within a complete IDE. Everything is bundled together as part of the unified SDK for distribution purposes. Configurators are used for:

- Setting options and generating code to configure drivers
- Setting up connections such as pins and clocks for a peripheral
- Setting options and generating code to configure middleware

For PSoC 6 MCU applications, the available Configurators include:

- Device Configurator: Set up the system (platform) functions, as well as the basic peripherals (e.g., UART, Timer, PWM).
- CapSense Configurator and Tuner: Configure CapSense, and generate the required code.
- USB Configurator: Configure USB settings and generate the required code.
- QSPI Configurator: Configure external memory and generate the required code.
- Smart I/O Configurator: Configure the Smart I/O.
- BLE Configurator: Configure the Bluetooth Low Energy (BLE) settings.

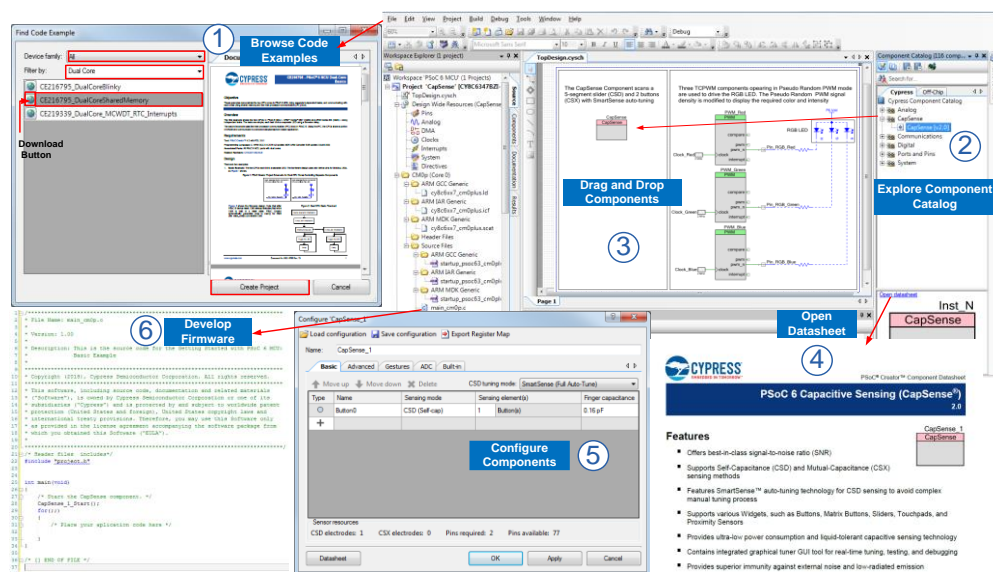
### 2.2.2 PSoC Creator

**PSoC Creator** is a free Windows-based Integrated Design Environment (IDE). It brings together several digital, analog, and system Components and firmware to build an application, and enables you to design hardware and firmware systems concurrently. Using PSoC Creator, you can select, place, and configure Components on a schematic; write C/assembly source code; and program and debug the device.

As **Figure 3** shows, with PSoC Creator, you can:

1. Browse the collection of code examples from the **File > Code Example...** menu.
  - a. Filter for examples based on device family.
  - b. Select from the menu of examples offered based on the **Filter by** options.
  - c. Download the code example using the download button.
  - d. Create a new project based on the selection.
2. Explore the library of more than 100 Components.
3. Drag and drop Components to build your hardware system design in the main design workspace.
4. Review the Component datasheets.
5. Configure the Components using configuration tools.
6. Co-design your application firmware with the PSoC hardware.

Figure 3. PSoC Creator Schematic Entry and Components



#### 2.2.2.1 PSoC Creator Help

Visit the [PSoC Creator](#) home page to download and install the latest version of PSoC Creator. Launch PSoC Creator and navigate to the following items:

- **Quick Start Guide:** Choose **Help > Documentation > Quick Start Guide**. This guide gives you the basics for developing PSoC Creator projects.
- **Code Examples:** Choose **File > Code Example** or click the **Find Code Example...** link on the **Start Page** tab. These code examples demonstrate how to configure and use PSoC resources.
- **Component Datasheets:** Right-click a Component and select **Open Datasheet**. Visit the [PSoC 6 MCU Component Datasheets](#) page for a list of all Component datasheets.



### 2.2.3 Software Development Kits for PSoC 6 Devices

Cypress provides significant source code and tools to enable software development for PSoC 6 MCU. You use tools to specify how you want to configure the hardware, generate code for that purpose which you use in your firmware, and include various middleware libraries for additional functionality, like BLE connectivity or FreeRTOS. This source code makes it easier to develop the firmware for supported devices. It helps you quickly customize and build firmware without the need to understand the register set.

For the PSoC Creator environment, Cypress provides the Peripheral Driver Library (PDL). The PDL supports both PSoC Creator and third-party IDEs. You use PSoC Creator Components to configure the hardware. PSoC Creator generates configuration code based on your choices. That code is based on the source code in the PDL drivers. The PDL also includes various middleware libraries. There may or may not be a Component to assist in configuring that code.

For ModusToolbox software, Cypress provides the PSoC 6 SDK. This SDK includes both the driver code and middleware libraries. In the ModusToolbox environment, you use Configurators to configure either the device, or a middleware library, like the BLE stack or CapSense functionality.

The driver code is delivered as the *psoc6pdl* library. Middleware is delivered as *psoc6mw*. The PDL source code is essentially identical, whether delivered with PSoC Creator or ModusToolbox IDE. There are implementation differences for the two IDEs.

There are differences in how the middleware is provided. For example, CapSense functionality is provided in PSoC Creator as a Component. For ModusToolbox software, there is a Configurator and a middleware library. See the respective documentation for the two IDEs for details of what's the same, and what's different.

Whether you use ModusToolbox IDE, PSoC Creator, or a third-party IDE, firmware developers who wish to work at the register level should also use the driver source code from the PDL. The PDL includes all the device-specific header files and startup code you need for your project. It also serves as a reference for each driver. Because the PDL is provided as source code, you can see how it accesses the hardware at the register level.

Some devices do not support particular peripherals. The PDL is a superset of all the drivers for any supported device. This superset design means:

- All API elements needed to initialize, configure, and use a peripheral are available.
- The PDL is useful across various PSoC 6 MCU devices, regardless of available peripherals.
- The PDL includes error checking to ensure that the targeted peripheral is present on the selected device.

This enables the code to maintain compatibility across some members of the PSoC 6 device family as long as the peripherals are available. A device header file specifies the peripherals that are available for a device. If you write code that attempts to use an unsupported peripheral, you will get an error at compile time. Before writing code to use a peripheral, consult the datasheet for the particular device to confirm support for the peripheral.

PSoC Creator provides Components that are based on the PDL. This retains the essence of PSoC Creator in utilizing Cypress or community-developed and pre-validated Components. However, the PDL is a source code library that you can use with any development environment.

The PDL includes the following key software resources:

- Header and source files for each peripheral driver.
- Header and source files for middleware libraries.
- Device-specific header, startup, and configuration files.
- Template projects for supported third-party IDEs.
- Full documentation, available in `<PDL install directory>\doc\`.



There are two key documents:

1. The PDL v3.x User Guide covers the fundamentals of working with the PDL, such as the following:
  - Creating a custom project using the PDL (including third-party IDEs).
  - Configuring a peripheral.
  - Managing pins in firmware.
  - Using the PDL as a learning tool for register-based programming.
  - Using the PDL API Reference documentation.
2. The *PDL 3.x API Reference Manual.html*. This reference has complete information on every driver in the PDL, including overview, configuration considerations, and details on every function, macro, data structure, and enumerated type.

## 2.3 Support for Other IDEs

You can also develop firmware for PSoC 6 MCU using your favorite IDE such as IAR Embedded Workbench. Cypress recommends that you generate resource configuration using a configuration tool. For ModusToolbox, the stand-alone device and middleware configurators generate the required code. For PSoC Creator, configuration is integral to the IDE.

You can use the PDL with another IDE by using PSoC Creator to design the system and generate configuration code and then export to a target IDE. See the [AN219434 – PSoC 6 MCU Importing Generated Code into an IDE](#) for details.

### 2.3.1 Using ModusToolbox to Target Another IDE

ModusToolbox configurators are standalone tools that can be used to set up and configure PSoC 6 MCU resources and other middleware components without using the ModusToolbox IDE. The device configurator and middleware configurators use the *design.modus* file within the application workspace. You can then point to the generated source code, and continue developing firmware in your IDE. If there is a change in the device configuration, edit the *design.modus* file using the configurators and regenerate the code for the target IDE.

### 2.3.2 Using PSoC Creator to Target Another IDE

PSoC Creator is used to set up and configure PSoC 6 MCU system resources and peripherals. You then export the project to your IDE, and continue developing firmware in your IDE. If there is a change in the device configuration, you edit the TopDesign schematic in PSoC Creator and regenerate the code for the target IDE.

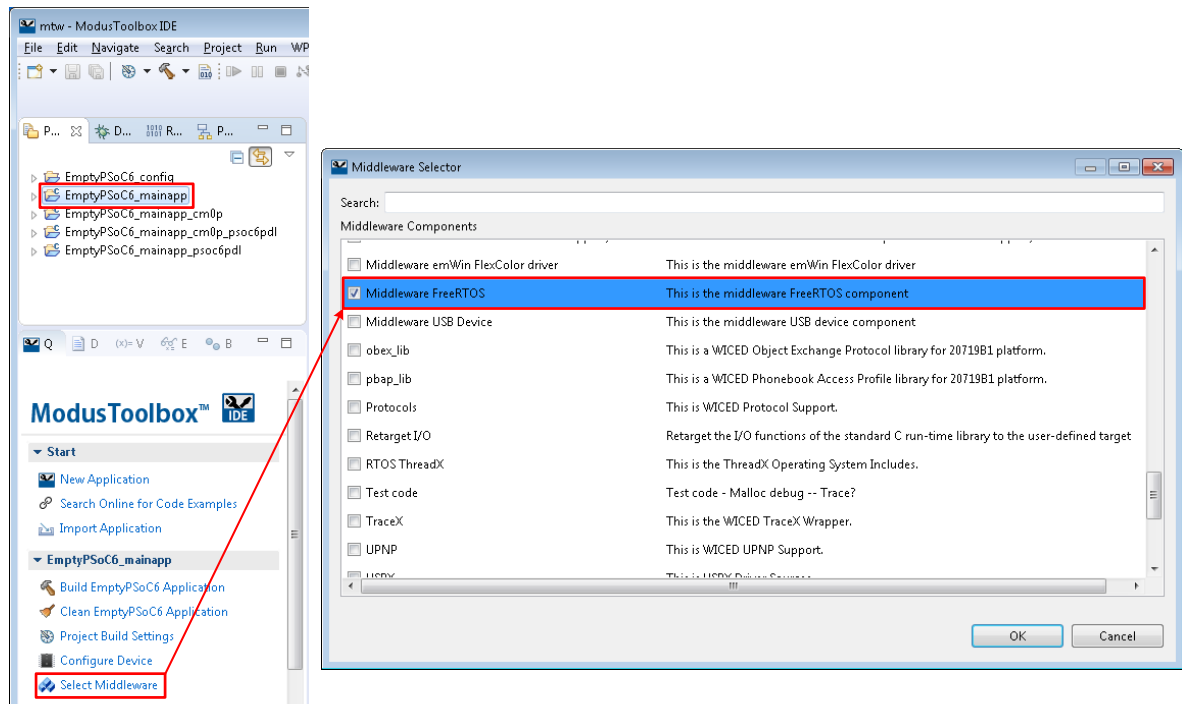
You can work effectively in most if not all IDEs. If your IDE is not supported in the Target IDEs panel, you can still use PSoC Creator. After you generate code, add the necessary files directly to your IDE's project. [AN219434 – PSoC 6 MCU Importing Generated Code into an IDE](#) provide detailed steps for manually importing the generated code into another IDE.

## 2.4 RTOS Support

### 2.4.1 RTOS Support with ModusToolbox

The PSoC 6 SDK includes RTOS for PSoC 6 MCU development. The FreeRTOS source code is fully integrated with the SDK as part of software components/middleware. You can import the FreeRTOS middleware into your application by using the Select Middleware option. Select the mainapp project, and then click the **Select Middleware** link in the **Quick Panel**. Then select **FreeRTOS** from the **Middleware Selector** dialog, as [Figure 4](#) shows.

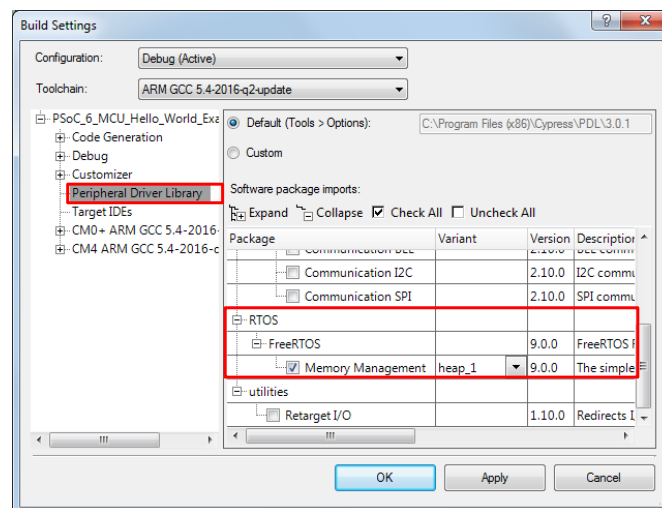
Figure 4. Import FreeRTOS in ModusToolbox IDE Application



## 2.4.2 RTOS Support with PSoC Creator

The PDL includes RTOS support for PSoC 6 MCU development: FreeRTOS source code is fully integrated and included with the PDL. You can import the FreeRTOS software package into your project by using the PSoC Creator RTOS import option. Navigate to the **Project > Build Settings** menu and select **FreeRTOS** from the **Software package imports** option under **Peripheral Driver Library > FreeRTOS** as shown in Figure 5.

Figure 5. Import FreeRTOS in PSoC Creator Project



If you have a preferred RTOS, use the resources provided as examples on how to integrate such code with the PDL.

## 2.5 Debugging

The [CY8CKIT-062-WiFi-BT PSoC 6 Wi-Fi-BT Pioneer Kit](#) and [CY8CKIT-062-BLE PSoC 6 BLE Pioneer Kit](#) have the KitProg2 onboard programmer/debugger. It supports Cortex Microcontroller Software Interface Standard - Debug Access Port (CMSIS-DAP) and custom modes of operations, as well as the KitProg2 connection. This makes debugging the PSoC 6 MCU Pioneer Kit extremely flexible. See the [KitProg2 User Guide](#) for details.

The [CY8CPROTO-062-4343W PSoC 6 Wi-Fi BT Prototyping Kit](#) has the KitProg3 onboard programmer/debugger. It supports Cortex Microcontroller Software Interface Standard - Debug Access Port (CMSIS-DAP). See the [KitProg3 User Guide](#) for details.

### 2.5.1 Debugging with ModusToolbox

The ModusToolbox IDE requires KitProg3 for debugging PSoC 6 MCU applications. It also supports GDB debugging using industry standard probes like the Segger J-Link.

ModusToolbox supports simultaneous debugging both Cortex-M4 and Cortex-M0+ CPUs. For more information on debugging firmware on PSoC devices with ModusToolbox, refer to the ModusToolbox Help.

ModusToolbox includes the **fw-loader** command-line tool to update CY8CIT-062-WiFi-BT and CY8CKIT-062-BLE kits, and switch the KitProg firmware from KitProg2 to KitProg3. Refer to the **PSoC 6 MCU KitProg Firmware Loader** section in the ModusToolbox IDE User Guide for more details.

### 2.5.2 Debugging with PSoC Creator

PSoC Creator supports debugging a single CPU (either Cortex-M4 or Cortex-M0+) at a time. Some third-party IDEs support multi-CPU debugging. For more information on debugging firmware on PSoC devices with PSoC Creator, refer to the PSoC Creator Help.

## 2.6 PSoC 6 MCU Development Kits

[CY8CKIT-062-WiFi-BT PSoC 6 Wi-Fi-BT Pioneer Kit](#) and [CY8CPROTO-062-4343W PSoC 6 Wi-Fi BT Prototyping Kit](#) are development kits that supports the PSoC 62 series MCU along with Wi-Fi and BT connectivity. The [CY8CKIT-062-BLE PSoC 6 BLE Pioneer Kit](#) and [CY8CPROTO-063-BLE PSoC 6 BLE Prototyping Kit](#) support the PSoC 6 MCU with Bluetooth Low Energy (BLE) Connectivity. Refer to [Appendix B](#) for more information. Note that the [CY8CPROTO-062-4343W PSoC 6 Wi-Fi BT Prototyping Kit](#) is not supported on PSoC Creator and is only supported on the ModusToolbox IDE.

### 3 Device Features

The PSoC 6 MCU device has an extensive feature set as shown in Figure 6. The following is a list of its major features. For more information, see the device [datasheet](#), the [Technical Reference Manual \(TRM\)](#), and the section on [Related Application Notes and Code Examples](#).

#### ■ 32-bit dual-CPU subsystem

- 150-MHz Arm Cortex-M4 and 100-MHz Arm Cortex-M0+
- Up to 2 MB of flash with additional 32 KB for EEPROM emulation and 32-KB supervisory flash
- Up to 1 MB of SRAM with selectable Deep Sleep retention granularity at 32-KB retention boundaries
- Inter-processor communication supported in hardware
- Cryptography accelerators and true random number generator function
- Up to three DMA controllers
- eFUSE: one-time programmable bits
- Secure boot with hardware hash-based authentication

#### ■ I/O subsystem

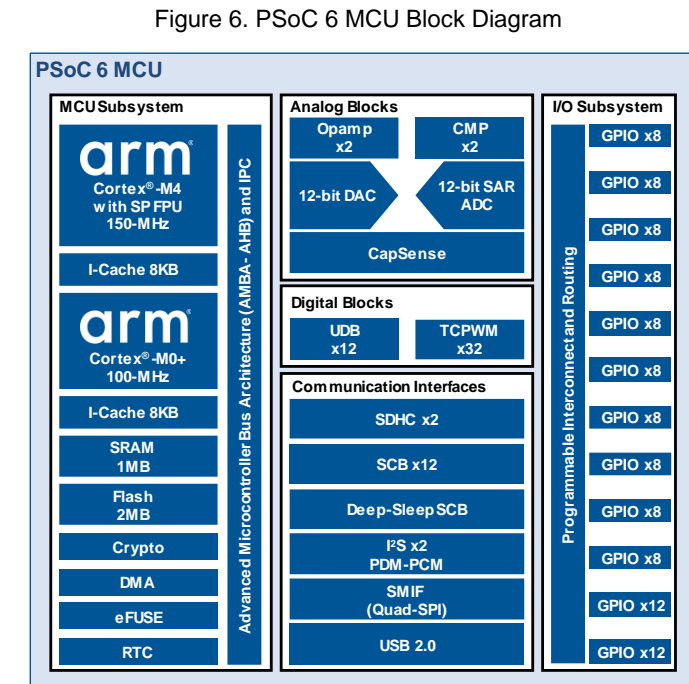
- Up to 104 GPIOs with programmable drive modes, drive strength, slew rates
- Two ports with Smart I/O that can implement Boolean operations

#### ■ Programmable analog blocks

- Two opamps of 6-MHz gain bandwidth (GBW) and two low-power comparators
- Up to One 12-bit, 1-Msps SAR ADC and one 12-bit voltage-mode DAC

#### ■ Programmable digital blocks, communication interfaces

- Up to 12 UDBs for custom digital peripherals
- Up to 32 TCPWM blocks configurable as 16-bit/ 32-bit timer, counter, PWM, or quadrature decoder
- Up to 13 SCBs configurable as I2C Master or Slave, SPI Master or Slave, or UART
- Audio subsystem with up to two I2S interface and two PDM channels
- SMIF interface with support for execute-in-place from external quad SPI flash memory and on-the-fly encryption and decryption
- Secure Digital Host Controller with support for SD, SDIO, and eMMC interfaces
- Full-Speed, dual-role USB with device and host capability



#### ■ CapSense with SmartSense™ auto-tuning

- Supports both CapSense Sigma-Delta (CSD) and CapSense Transmit/Receive (CSX) controllers
- Provides best-in-class SNR, liquid tolerance, and proximity sensing

#### ■ Operating voltage range, power domains, and low-power modes

- Device operating voltage: 1.71 V to 3.6 V with user-selectable core logic operation at either 1.1 V or 0.9 V
- Multiple on-chip regulators: low-drop out (LDO for Active, Deep Sleep modes), single-input multiple-output (SIMO) buck converter
- Active, Low-Power Active, Sleep, Low-Power Sleep, Deep Sleep, and Hibernate modes for fine power management
- An "Always ON" backup power domain with built-in RTC, power management integrated circuit (PMIC) control, and limited SRAM backup

## 4 Choosing an IDE

ModusToolbox, the latest-generation toolset, includes the ModusToolbox IDE.. The IDE is Eclipse-based and therefore is supported across Windows, Linux, and MacOS platforms. The tool supports all PSoC 6 MCU devices. The associated hardware and middleware configurators also work on all three host operating systems.

Certain features of PSoC 6 MCU such as UDBs and USB host are not currently supported in ModusToolbox IDE. Cypress will release new versions of ModusToolbox to support these features and improve the user experience.

Choose ModusToolbox if you have prior experience with Eclipse-based tools and want to take advantage of the power and extensibility of an Eclipse-based IDE, or if you want your development environment on Linux or MacOS. You should also choose ModusToolbox if you want to build an IoT application using Cypress IoT devices, or if you are using a PSoC 6 MCU device not supported on PSoC Creator.

PSoC Creator is the long-standing Cypress-proprietary tool that runs on Windows only. This mature IDE includes a graphical editor that supports schematic based design entry with the help of Components. PSoC Creator supports all PSoC 3, PSoC 4, PSoC 5LP devices and a subset of PSoC 6 MCU devices. The subset of PSoC 6 MCU devices include devices up to 1 MB of flash. It does not support PSoC 6 MCU devices with a USB interface.

Choose PSoC Creator if you are inclined towards using a graphical editor for design entry and code generation, and if the PSoC MCU that you are planning to use is supported by the IDE or if you are intending to use the UDBs on the PSoC MCU.

The sections that follow provide detailed steps to create an application for PSoC 6 MCU. Navigate to section

[My First PSoC 6 MCU Design Using ModusToolbox](#) if you would like to use ModusToolbox. Navigate to section [My First PSoC 6 MCU Design Using PSoC Creator](#) if you would like to use PSoC Creator development environment.

## 5 My First PSoC 6 MCU Design Using ModusToolbox IDE

This section does the following:

- Demonstrates how to build a simple PSoC 6 MCU-based design and program it on to the development kit.
- Provides detailed steps that make it easy to learn PSoC 6 MCU design techniques and how to use the ModusToolbox IDE.

### 5.1 Using These Instructions

These instructions are grouped into several sections. Each section is devoted to a phase of the application development workflow. The major sections are:

- [Part 1: Create a New Application](#)
- [Part 2: Implement the Design](#)
- [Part 3: Write the Firmware](#)
- [Part 4: Build the Application](#)
- [Part 5: Program the Device](#)
- [Part 6: Test Your Design](#)

If you are familiar with developing projects with ModusToolbox, download and use ModusToolbox version of the code example [CE221773 – PSoC 6 MCU Hello World Example Using ModusToolbox](#) directly. It is a complete design, with all the firmware written for the CY8CKIT-062-WiFi-BT kit. You can walk through the instructions and observe how the steps are implemented in the code example.

If you start from scratch and follow all the instructions in this application note, you use the code example as a reference while following the instructions.

This design is developed for the [CY8CKIT-062-WiFi-BT PSoC 6 Wi-Fi-BT Pioneer Kit](#). You can also use [CY8CKIT-062-BLE PSoC 6 BLE Pioneer Kit](#) or [CY8CPROTO-062-4343W PSoC 6 Wi-Fi BT Prototyping Kit](#) to test this example by selecting the appropriate kit or device while creating the application.

### 5.2 About the Design

This design uses the CM4 CPU of PSoC 6 MCU to execute two tasks: UART communication and LED control.

At device reset, the CM0+ CPU enables the CM4 CPU. The CM4 CPU uses the UART personality to print a “Hello World” message to the serial port stream and when the user presses the enter key, the LED on the PSoC 6 MCU Wi-Fi-BT Pioneer Kit starts blinking.

### 5.3 Part 1: Create a New Application

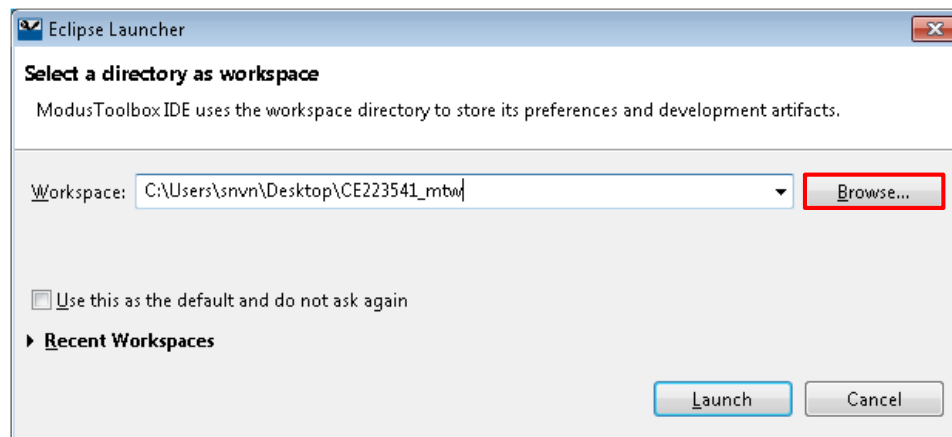
This section takes you on a step-by-step guided tour of the design process. It starts with creating an empty application and guides you through hardware and firmware design development stages.

Launch ModusToolbox and get started.

#### 1. Select a new workspace.

At launch, ModusToolbox presents a dialog to choose a directory for use as the workspace directory. The workspace directory is used to store workspace preferences and development artifacts. You can choose an existing empty directory by clicking the **Browse** button, as [Figure 7](#) shows. Alternatively, you can type in a directory name to be used as the workspace directory along with the complete path and ModusToolbox will create the directory for you.

Figure 7. Select a Directory as Workspace



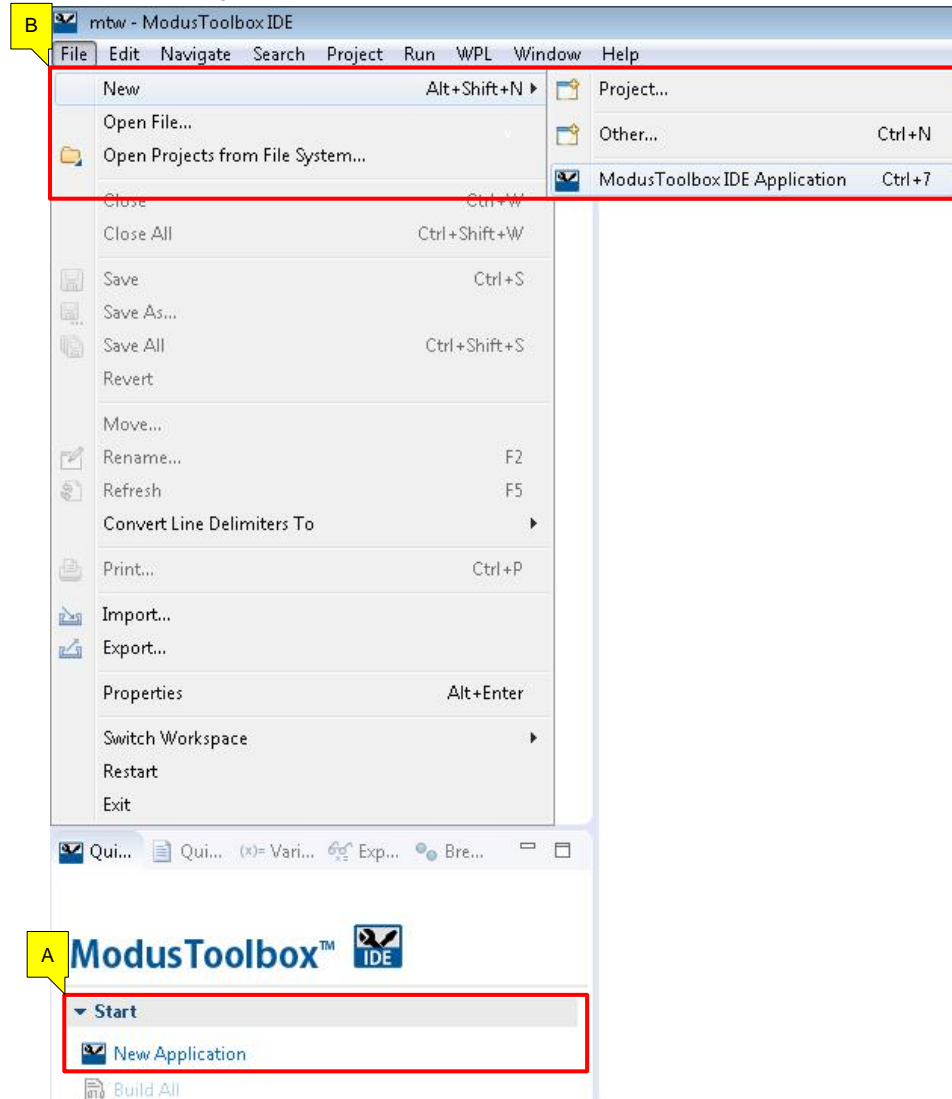
#### 2. Create a new ModusToolbox Application.

- A. Click **New Application** in the **Start** group of the **Quick Panel**.
- B. Alternatively, you can choose **File > New > ModusToolbox IDE Application**, as [Figure 8](#) shows.

The ModusToolbox IDE Application window appears.



Figure 8. Create a New ModusToolbox IDE Application



### 3. Select PSoC 6 MCU as the target device.

ModusToolbox presents two options to get you started with the application creation process: **Dev/Eval Kit** or **Custom Hardware**. In both the cases, ModusToolbox speeds up the development process by automatically setting various workspace/project options for specified development kits or target devices.

Use the **Dev/Eval Kit** option for creating this example and follow the steps below. See [Figure 9](#) for help with this step.

You start from **Dev/Eval Kit** in case you have a Cypress PSoC 6 MCU development kit on hand. In this case, ModusToolbox provides you with an application template, with all kit resources pre-configured. You must configure and enable additional resources that you intend to use in your application.

**Custom Hardware** provides you an option to choose the target device directly in case you have a PSoC 6 MCU-based custom board on hand. In this case, you must configure all resources like UART, pins, and peripheral clocks that you intend to use in your application.

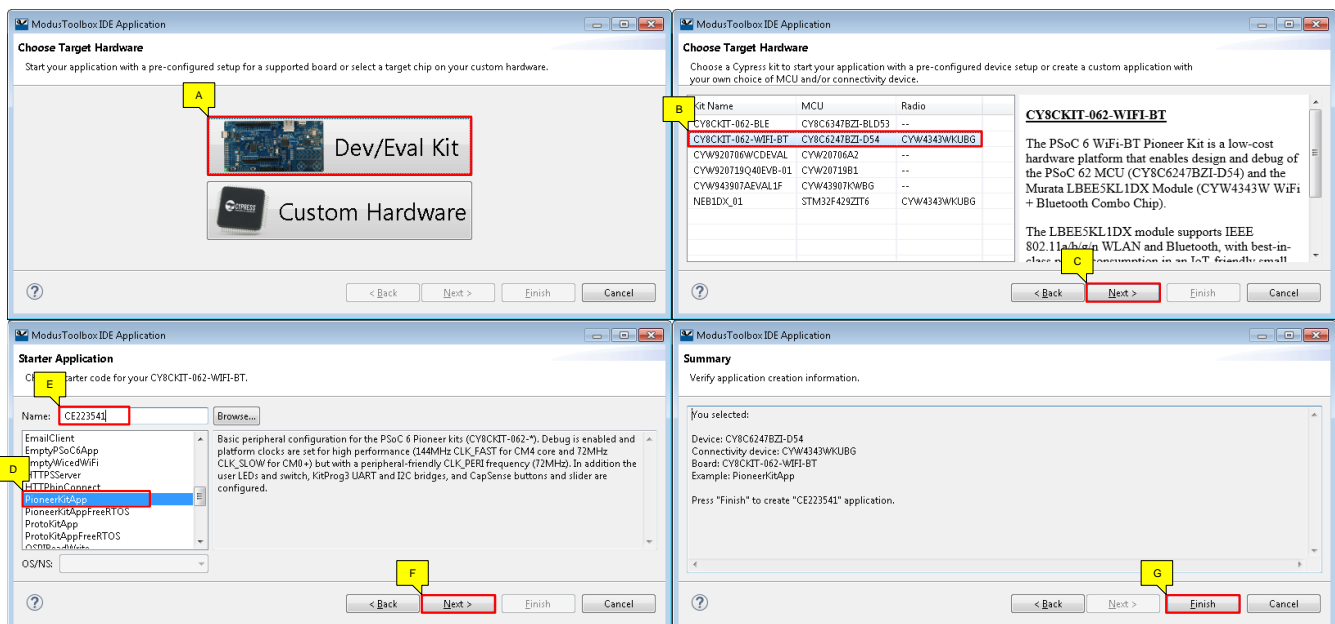
- A. Click **Dev/Eval Kit**.
- B. In the **Choose Target Hardware** dialog, choose the **Kit Name** that you have. The steps that follow assume **CY8CKIT-062-WIFI-BT**.

- C. Click **Next**.
- D. In the **Starter Application** dialog, select **PioneerKitApp** application.
- E. In the **Name** field, type in a name for the application.
- F. Click **Next**. The application summary dialog appears.
- G. Click **Finish** to let ModusToolbox create the application projects for you.

ModusToolbox uses CY8C6247BZI-D54 as the default device that is mounted on the [CY8CKIT-062-WiFi-BT PSoC 6 Wi-Fi-BT Pioneer Kit](#) along with the CYW4343WKUBG Wi-Fi/BT radio.

The **PioneerKitApp** application template has the all the resources available on the Pioneer kit pre-configured and ready for use. These resources include user LEDs, user switches, I2C bridge and UART bridge. Additionally, the template includes the system clock configuration.

Figure 9. Start from a Development/Evaluation Kit

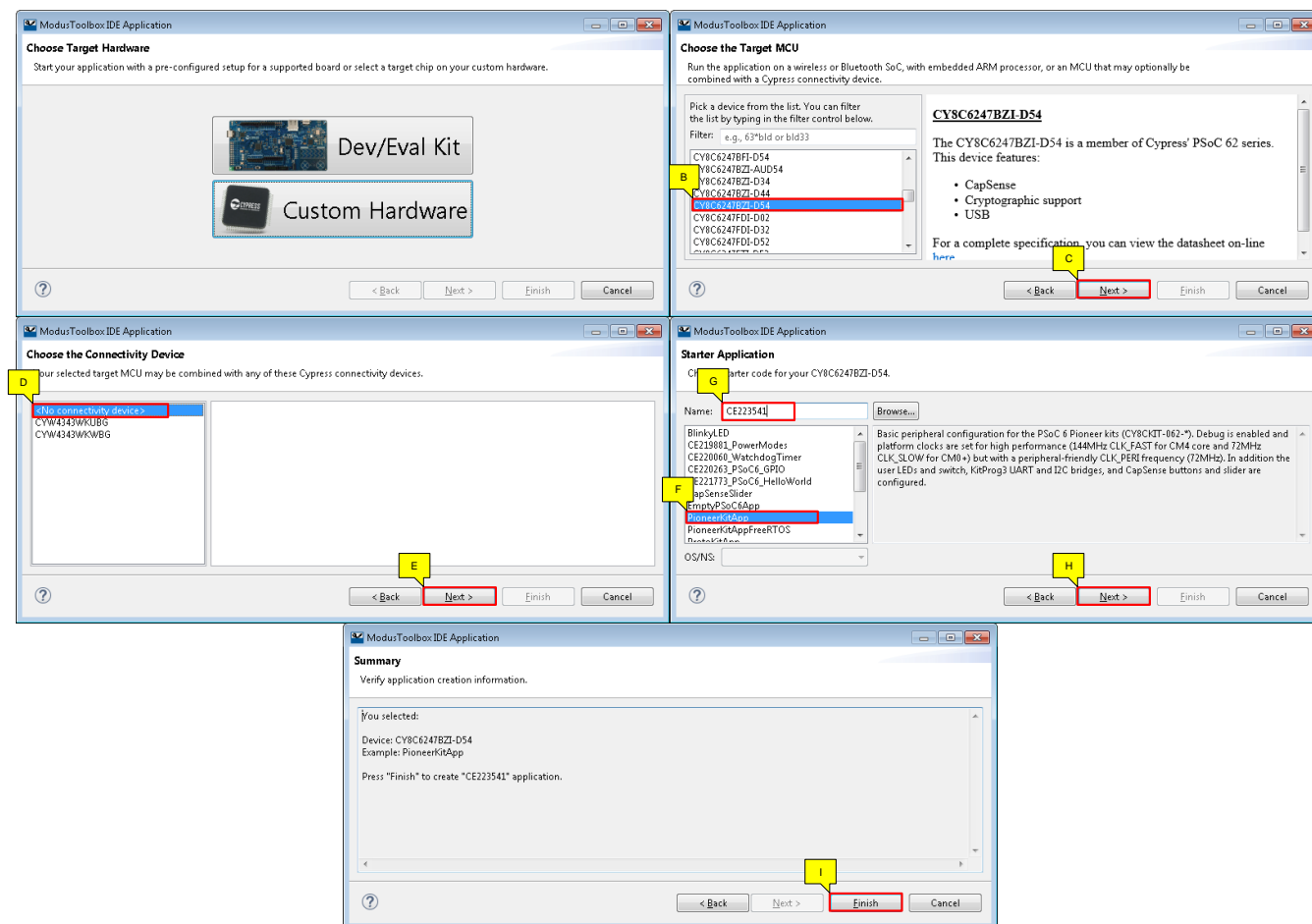


If you are using a custom hardware based on PSoC 6 MCU, or a different PSoC 6 MCU part number, this is the place you choose the **Custom Hardware** option and select the appropriate part number.

If you are using the **Custom Hardware** option, follow the steps below. See [Figure 10](#) for help with this step.

- A. Click **Custom Hardware**.
- B. In the **Choose the Target MCU** dialog, pick a device from the list. You can filter the device list by typing in the filter control just above the list. Pick **CY8C6247BZI-D54** for the steps that follow or if you have a custom board with a different PSoC 6 MCU part number, choose the appropriate device here.
- C. Click **Next**.
- D. In the **Choose the Connectivity Device** dialog, select **No Connectivity Device**. The application that you will develop will target only the PSoC 6 MCU device.
- E. Click **Next**.
- F. In the **Starter Application** dialog, select **PioneerKitApp** application.
- G. In the **Name** field, type in a name for the application.
- H. Click **Next**. The application summary dialog appears.
- I. Click **Finish** to let ModusToolbox create the application projects for you.

Figure 10. Start from a Custom Hardware



You have successfully created a new ModusToolbox application for PSoC 6 MCU.

## 5.4 Part 2: Implement the Design and Generate Source Code

Now that you have an application set up from an existing Pioneer kit template, it is time to add additional resources in hardware design required for this application. The template includes configuration for pins, UART and clocks required for this example. If you are using the code example directly, you already have a complete design.

Before you add additional resources to the design, a quick tour of the ModusToolbox project explorer is in order.

Figure 11 shows the ModusToolbox project explorer interface displaying the structure of the application projects.

In the ModusToolbox IDE, a PSoC 6 MCU application consists of several projects to develop code for each CPU, and the drivers for each. The config project includes a project folder with a base set of *cycfg\_* files that are already created for you. These files contain the design configuration. You can modify the design configuration by double-clicking the *design.modus* file in the config project.

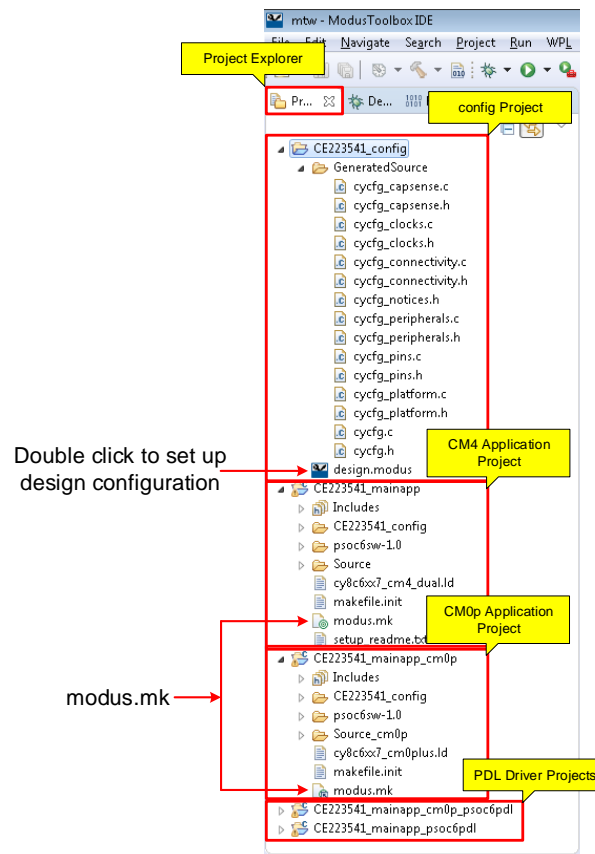
The CM4 application project (*\_mainapp*) contains relevant files that help you create an application for the CM4 CPU, while the CM0+ application project (*\_mainapp\_cm0p*) contains those for the CM0+ CPU. Notice that both the projects refer to the config project. This ensures that the design configuration files are available for the source files of both CPUs. Additionally, there are two PDL driver projects, one for each CPU that bring in the peripheral driver library files to be compiled into the application.

The projects have dependencies. The Quick Panel provides a one-click link to build the application, so you don't have to consider which project depends on which.

Note that each CPU application project points to a *modus.mk* file. It contains instructions on how to recreate the project.

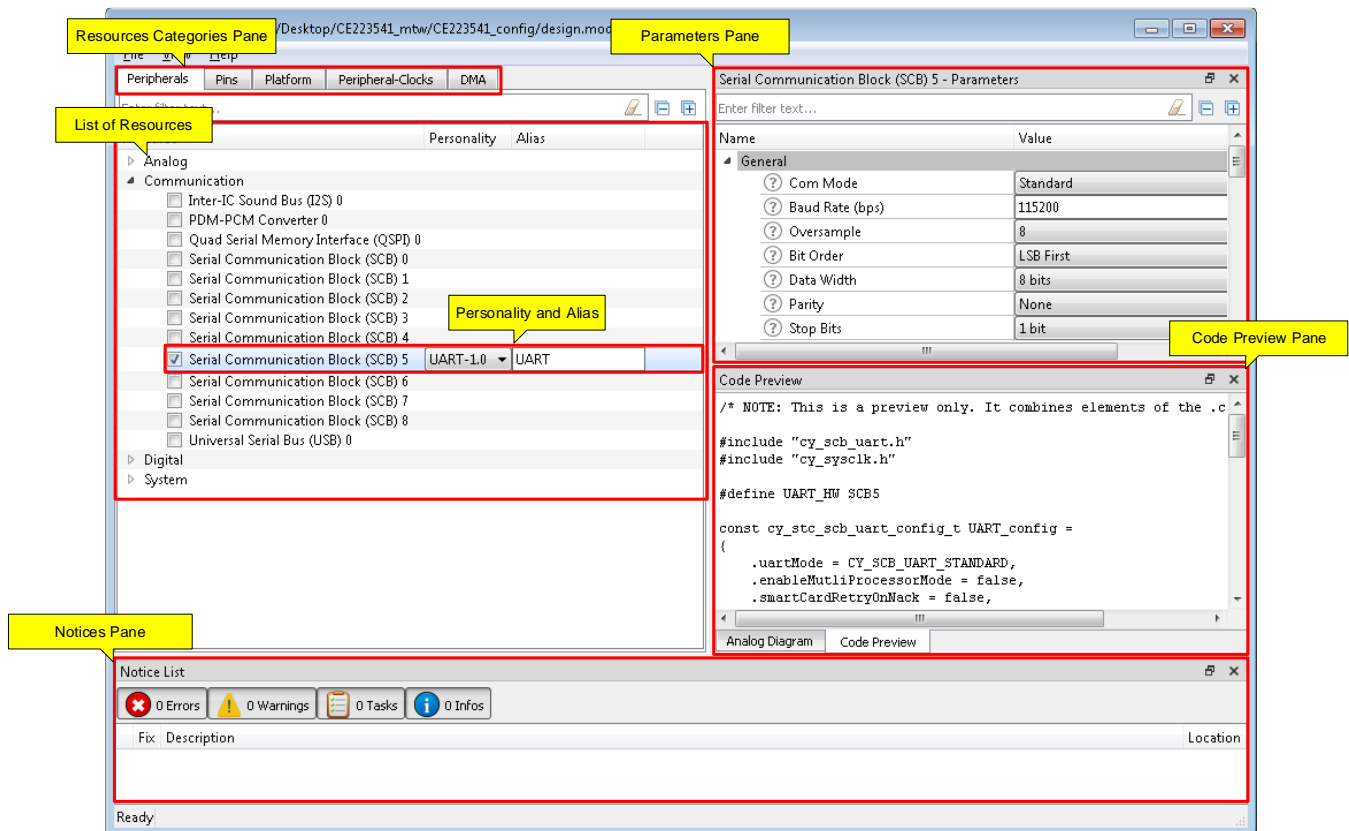
The makefile required to compile and link the projects is created automatically by the IDE, and is in *\_mainapp/Debug* folder after the application is built. This file contains the set of directives that the inbuilt make tool uses to compile and link the corresponding application project.

Figure 11. Project Explorer View



Double-click the *design.modus* file in the config project in the Project Explorer view or click on the **Configure Device** link in the **Quick Panel**. Figure 12 shows the resulting window called the **Device Configurator** window.

Figure 12. *design.modus* Overview



The **Device Configurator** window provides a **Resources Categories** pane. Here you can choose between different resources available in the device such as peripherals, pins, and clocks from the **List of Resources**.

You can choose how a resource behaves by choosing a **Personality** for the resource. For example, a **Serial Communication Block (SCB)** resource can have an **EZIO2C**, **I2C**, **SPI** or **UART** personalities. The **Alias** is your name for the resource, which is used in firmware development.

The **Parameters** pane is where you enter the configuration parameters for each enabled resource and the selected personality. The **Code Preview** pane shows the configuration code generated per the configuration parameters selected. This code is populated in the *cycfg\_* files in the config project. Any errors, warnings, and information messages arising out of the configuration are displayed in the **Notices** pane.

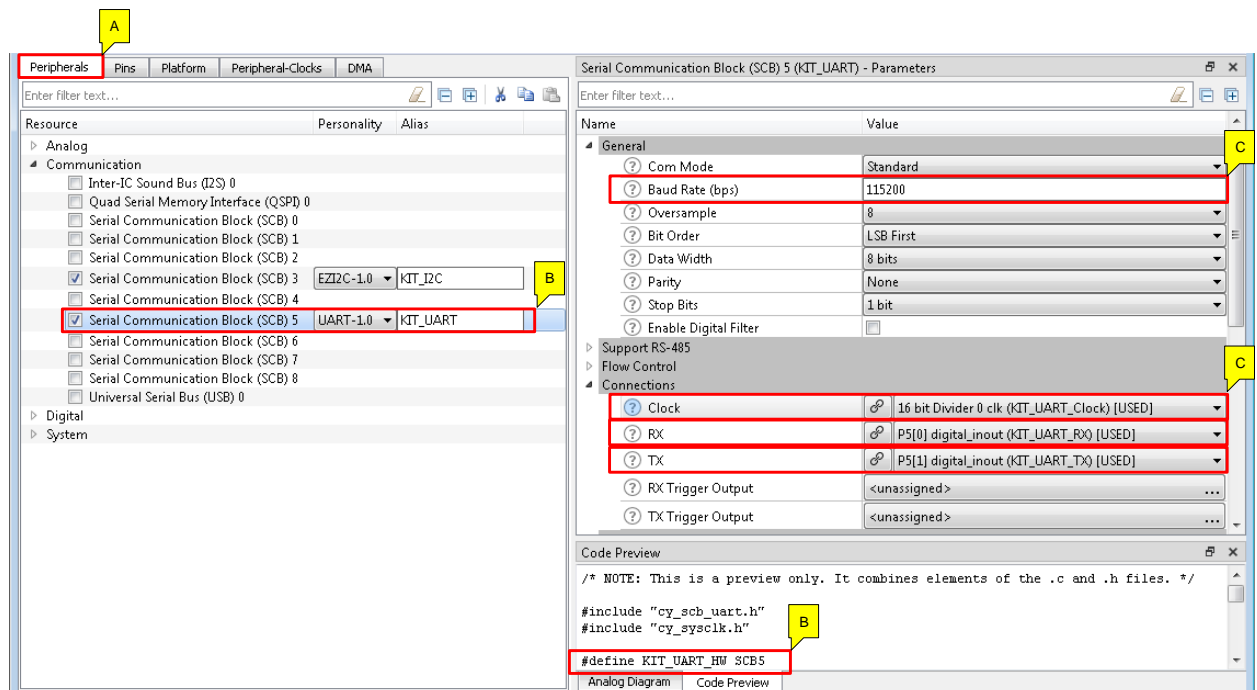
The design uses several resources: three digital pins, a UART, a Timer, and an Interrupt. In the following steps, you will note the settings of the resources already enabled, and you will add the Timer resource to the design and configure it.

## 1. UART Configuration.

Serial Communication Block (SCB) 5 is already configured as **KIT\_UART** by the application template. To view the pre-configured settings, follow the steps below. See [Figure 13](#) for help with this step.

- Expand the **Peripherals** resources tab and navigate to **Communication** group.
- Notice that **Serial Communication Block (SCB) 5** is configured as a UART personality with **KIT\_UART** as the alias/name. Notice that the hardware resource **SCB5** is given an alias **KIT\_UART\_HW** by the configurator. See the **Code Preview** window in [Figure 13](#). This alias will be used later in the firmware.
- Also notice that the baud rate is set to **115200 bps** and the SCB is configured to **16 bit Divider 0 clk** as the clock source. The RX and TX signals are also routed to port pins **P5[0]** and **P5[1]** respectively, as [Figure 13](#) shows.

Figure 13. SCB 5 Resource as UART

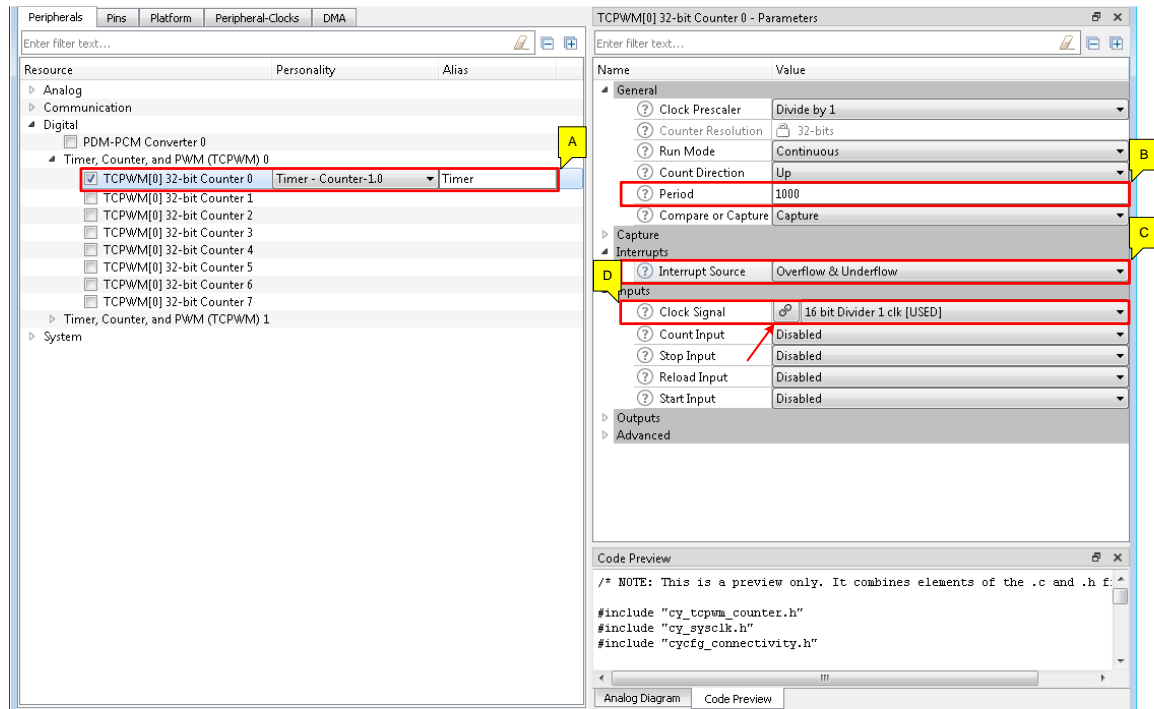


The screenshot displays the PSoC 6 MCU configuration tool interface. On the left, the 'Peripherals' tab is active, showing a list of resources. 'Serial Communication Block (SCB) 5' is selected, with its personality set to 'UART-1.0' and alias to 'KIT\_UART'. On the right, the 'Parameters' window for 'Serial Communication Block (SCB) 5 (KIT\_UART)' is open. It shows various configuration options: 'Baud Rate (bps)' is 115200, 'Data Width' is 8 bits, 'Parity' is None, and 'Stop Bits' is 1 bit. The 'Clock' is set to '16 bit Divider 0 clk (KIT\_UART\_Clock) [USED]'. The 'RX' pin is 'P5[0] digital\_inout (KIT\_UART\_RX) [USED]' and the 'TX' pin is 'P5[1] digital\_inout (KIT\_UART\_TX) [USED]'. At the bottom, the 'Code Preview' window shows the definition: `#define KIT_UART_HW SCB5`.

## 2. Configure the Timer.

The TCPWM resource is configured as a timer to generated interrupts every 500 milliseconds. Note that the application template does not pre-configure the timer for you.

Figure 14. Enable and Configure TCPWM[0] Counter 0 Resource



Follow the steps below to configure the TCPWM 32-bit Counter 0 to be used as a software timer.

- In the **Peripherals** resources tab, expand the **Digital > Timer, Counter and PWM (TCPWM) 0** group, and select the checkbox next to **TCPWM (0) 32-bit Counter 0** to configure it. Select **Timer-Counter-1.0** as the personality. Set the alias as **Timer** here for use in the firmware.
- Enter the desired period in the **Period** field of **General** parameters group. For this example, you will use **1000**.
- Set **Overflow & Underflow** as the **Interrupt Source**. Whenever the timer overflows, this setting will cause it to generate an interrupt.

Now, you will set the clock connections to the TCPWM resource.

- Select **16 bit Divider 1 clk** as the **Clock Signal**. When you select the peripheral clock, ModusToolbox automatically enables the corresponding peripheral clock, but does not set the divider. Click the link icon next to the selected clock signal. This will take you to the **16 bit Divider 1** settings. You will set the divider value in the next step.



### 3. Peripheral-Clocks.

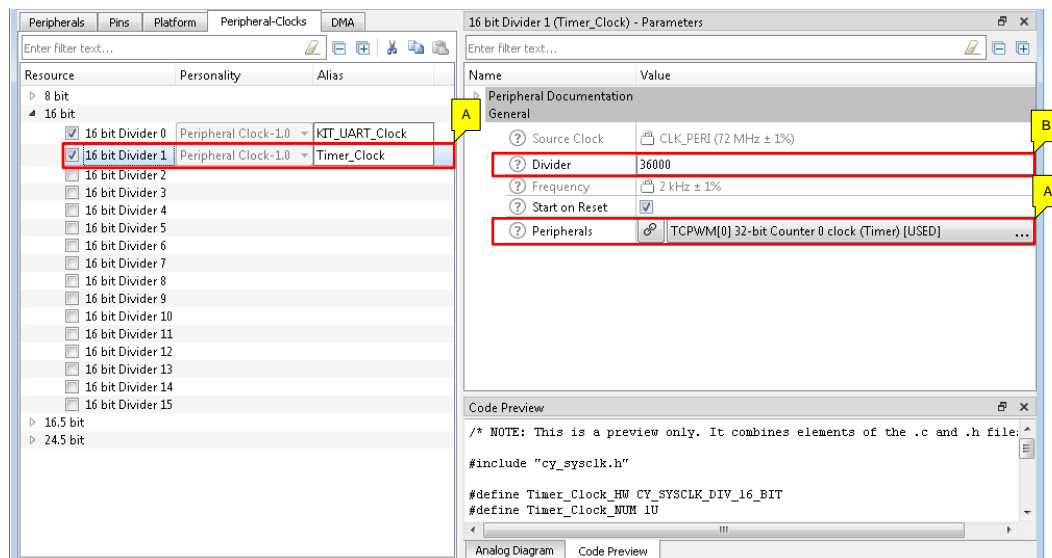
In this step, you configure the peripheral clock divider that is required to source the Timer resource. You will also note the settings of the peripheral clock divider sourcing the UART resource.

- A. At this point, the **Peripheral-Clocks** tab must be in focus. Notice that the **16 bit Divider 1** is already enabled with the **Divider** value set to **1** and connected to **TCPWM(0) 32-bit Counter 0 clock**, as [Figure 15](#) shows.

Set the alias to **Timer\_Clock**. This is an optional step to identify the clock divider and is not necessary for firmware.

- B. Set the **Divider** value to **36000**. This would generate a clock of 2 kHz because the **CLK\_PERI** is configured to be 72 MHz in the application template. You will see this in a later step.

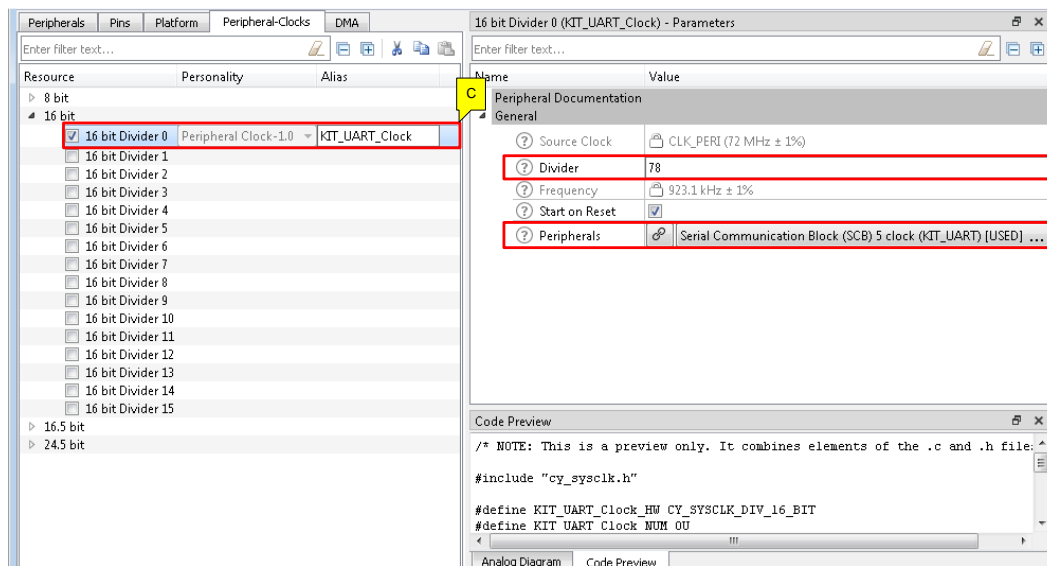
Figure 15. 16 bit Divider 1 Configuration



- C. Also notice that **16 bit Divider 0** is already selected with the **Divider** value set to **78** and connected to **Serial Communication Block (SCB) 5 clock**, as [Figure 16](#) shows.

The alias has also been set to **KIT\_UART\_Clock**. This is optional and is used to identify the clock divider.

Figure 16. 16-bit Divider 0 Configuration

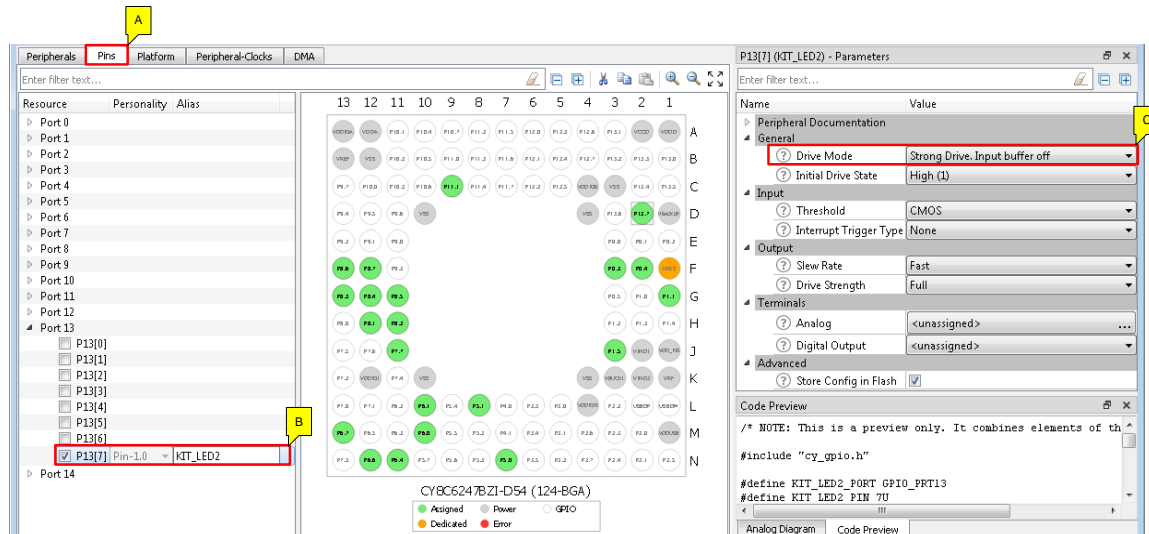


#### 4. LED Pin Configuration.

The user LED on the PSoC 6 Wi-Fi BT Pioneer Kit is active LOW; that is, the logic HIGH pin-drive state turns OFF the LED, and the logic LOW pin-drive state turns it ON. Figure 17 shows the configuration. The configuration is already set by the application template.

- In the **Device Configurator** window, navigate to the **Pins** resources tab.
- Expand the **Port 13** group and notice that the checkbox next to **P13[7]** is checked to enable the pin. Notice that the alias is set to **KIT\_LED2** for use in the firmware.
- Notice that the **Drive Mode** has been set to **Strong Drive, Input buffer off**.

Figure 17. Output Pin Resource



Navigate to the Port 5 group and notice that P5[0] and P5[1] pins are pre-configured for your application use as UART pins.

## 5. System Clocks.

The design uses default values for the high-frequency system clock settings as set up in the template. Although you do not modify high-frequency clocks for this design, you should know how ModusToolbox manages them. If you are working with your own board, you may need to modify these clocks.

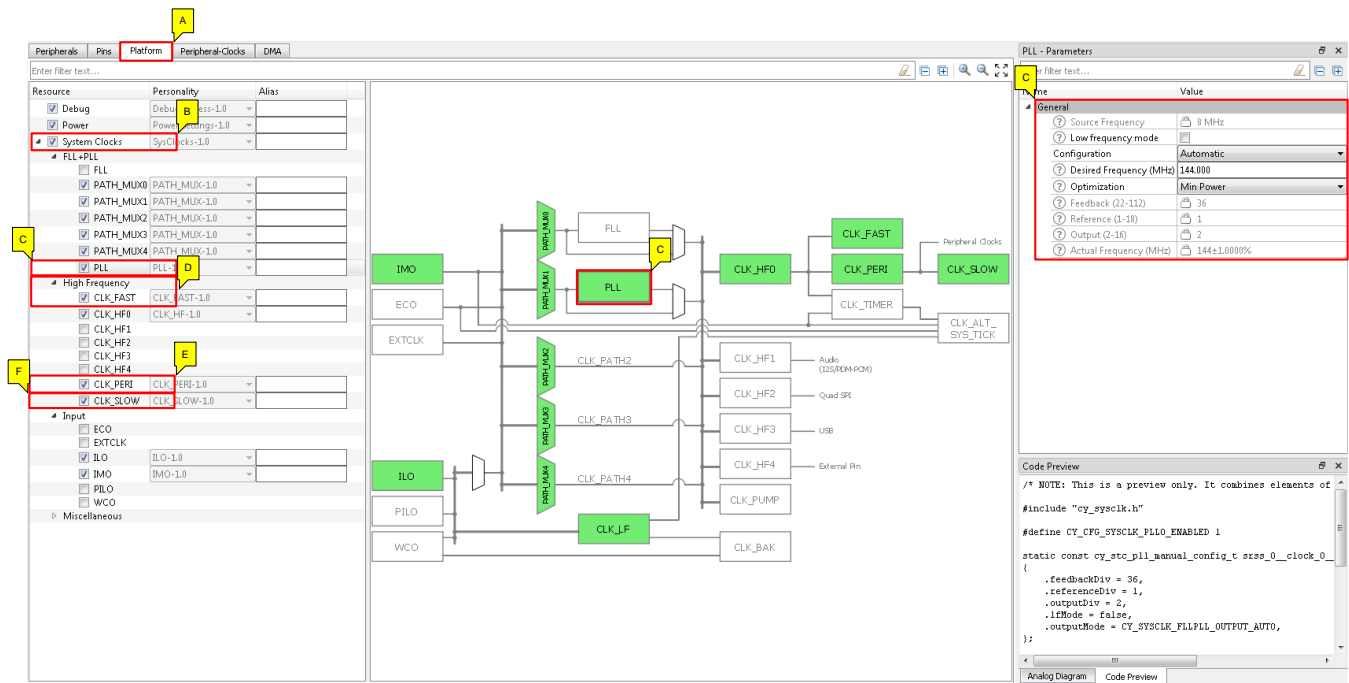
- A. In the **Device Configurator** window, navigate to the **Platform** resources tab.
- B. Expand the **System Clocks** Group.

Here, you can see the clock tree, and modify the clock/clock dividers as required. Note that there are checkboxes for different types of clocks under **FLL+PLL**, **High Frequency** Clocks, **Input** Clocks, and **Miscellaneous** Clocks groups.

- C. Click **PLL** clock under the **FLL+PLL** group. By default, the starter application template enables **PLL** and sets the frequency to 144 MHz.

Alternatively, you can click on the **PLL** clock block in the clock diagram and the parameters show up in the **Parameters** tab as [Figure 18](#) shows.

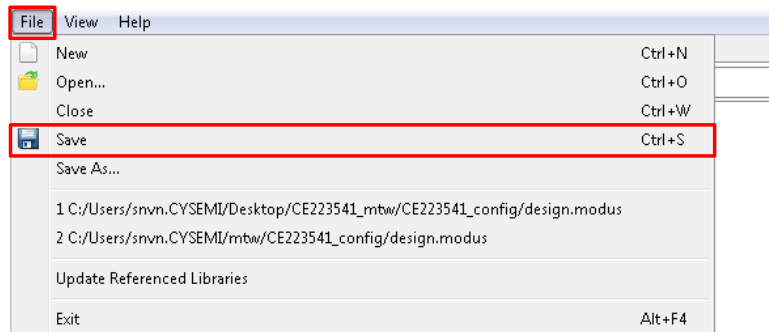
Figure 18. System Clock Configuration



- D. Expand the **High Frequency** Clocks group and select the already enabled **CLK\_FAST** clock. Note that the **Divider** for this clock is set to 1. This sets **CLK\_FAST** to **144 MHz**. This clock path sources the CM4 CPU in PSoC 6 MCU.
- E. In the **High Frequency** Clocks group, select the already enabled **CLK\_PERI** clock. Note that the **Divider** for this clock is set to 2. This sets the **CLK\_PERI** to **72 MHz**. This clock path sources the peripheral clock dividers in PSoC 6 MCU.
- F. In the **High Frequency** Clocks group, select the already enabled **CLK\_SLOW** clock. Note that the **Divider** for this clock is set to 1. This sets the **CLK\_SLOW** to **72 MHz**. This clock path sources the CM0+ CPU in PSoC 6 MCU.

Click **File > Save** in the ModusToolbox Configurator window, as [Figure 19](#) shows. This step causes the configurator to generate the design configuration and save it to the *design.modus* file. This step also completes the code generation process for the configuration. The configuration structures and associated code are saved to the *cycfg\_* files.

Figure 19. Save Design Configuration



## 6. Add Retarget I/O Software Component.

In this step, you will add the Retarget I/O software component to redirect standard input and output streams to the UART configured in Step 1.

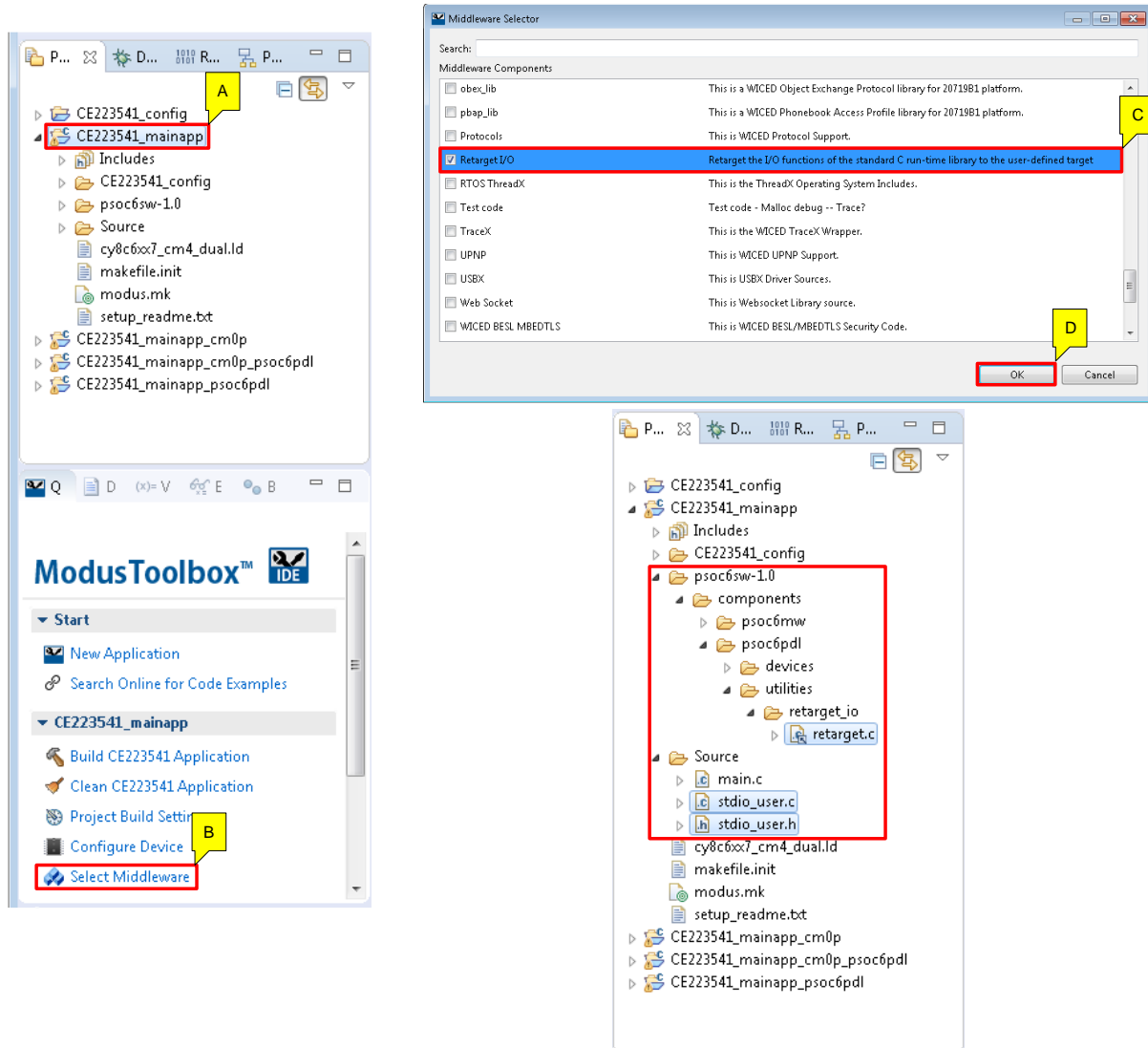
- A. In the **Project Explorer** panel, navigate to the `_mainapp` project.
- B. In the **Quick Panel**, click on the **Select Middleware** link.
- C. In the **Middleware Selector** dialog, select the **Retarget I/O** software component.
- D. Click **OK**.

The files necessary to use the Retarget I/O component are added in the **psoc6sw-1.0 > components > psoc6pdl > utilities > retarget\_io** folder, and the **Source** folder under the `_mainapp` project as [Figure 20](#) shows.

**psoc6sw-1.0 > components > psoc6pdl > utilities > retarget\_io:** `retarget_io.c`

**Source:** `stdio_user.c` and `stdio_user.h`

Figure 20. Add Retarget I/O Software Component



## 5.5 Part 3: Write the Firmware

At this point in the development process, you have created an application, completed the hardware design with the assistance of an application template/Configurators, and generated the code. In this part, you write the firmware that implements the design functionality.

The steps in this part discuss the firmware for the design that you configured in [Part 2: Implement the Design](#).

The code example has all the required code. If you are working from scratch, you can copy the respective source code to the *main.c* of the CM4 application project from the code snippet provided in this section. If you are using the code example, files are already in your application.

### Firmware Flow

In the remaining steps, we examine the code in the *main.c* files of CM0+ and CM4 applications.

When the PSoC 6 MCU device is reset, the firmware first performs system initialization, which includes setting up the CPUs for execution, enabling global interrupts, and enabling other peripherals used in the design.

The CM0+ CPU comes out of reset and enables the CM4 CPU. The resource initialization for this example is performed by CM4 CPU. It configures the system clocks, pins, clock to peripheral connections, and other platform resources. The CM0+ CPU code snippet is given below.

```
#include "cy_device_headers.h"
#include "cy_syslib.h"
#include "cy_sypm.h"

int main(void)
{
    __enable_irq();

    /* Allow the CM4 core to run */
    Cy_SysEnableCM4( CY_CORTEX_M4_APPL_ADDR );

    for(;;)
    {
        Cy_SysPm_DeepSleep( CY_SYSPM_WAIT_FOR_INTERRUPT );
    }
}
```

The **retarget\_io** middleware must be configured to use the UART resource to output the serial data. Navigate to **Source** folder of your CM4 application project, open *stdio\_user.h*.

Note that **IO\_STDx\_UART** is defined as **KIT\_UART\_HW**. The **KIT\_UART\_HW** points to the SCB resource used by the UART. The redirection is highlighted in yellow as shown below.

```
#include "cy_device_headers.h"
#include "cy_cfg.h"

/* Must remain uncommented to use this utility */
#define IO_STDOUT_ENABLE
#define IO_STDIN_ENABLE
#define IO_STDOUT_UART    KIT_UART_HW
#define IO_STDIN_UART     KIT_UART_HW
```

When the CM4 CPU is enabled, the UART peripheral is initialized and started. It prints a "Hello World!" message on the terminal emulator. A Timer Counter PWM (TCPWM) peripheral is configured to generate an interrupt every 500 milliseconds. At each Timer interrupt, the CM4 CPU toggles the LED state on the kit.

Copy the following code snippet to *main.c* of your CM4 application project.

```
#include "cy_device_headers.h"
#include "cycfg.h"
#include "cy_sysint.h"
#include "stdio.h"
#include "stdio_user.h"

/*****
 * Macros
```

```

*****/
#define LED_ON          (0u)
#define LED_OFF         (1u)

/*****
 * Function Prototypes
 *****/
void UartInit(void);
void TimerInit(void);
void Isr_Timer(void);

/*****
 * Global Variables
 *****/
bool LEDUpdateFlag = false;

/* The instance-specific context structure.
 * The driver uses this as a scratch pad for its operations.
 * Do not modify this structure.
 */
cy_stc_scb_uart_context_t KIT_UART_context;

/* Isr_Timer configuration structure*/
cy_stc_sysint_t Isr_Timer_config = {
    .intrSrc = (IRQn_Type) Timer_IRQ,
    .intrPriority = 7u
};

/*****
 * Function Name: main
 *****/
int main(void)
{
    /* Set up the device based on configurator selections */
    init_cycfg_all();

    /* Start the UART peripheral */
    UartInit();

    /* Enable global interrupts */
    __enable_irq();

    /* \x1b[2J\x1b[H - ANSI ESC sequence for clear screen */
    printf("\x1b[2J\x1b[H");

    printf("*****CE221773 - PSoC 6 MCU:"\
        " Hello World! Example*****\r\n\n");

    printf("Hello World!!!\r\n\n");

    printf("Press Enter key to start blinking the LED\r\n\n");

    /* Wait for the user to press Enter key */
    while(getchar() != '\r');

    /* Start the TCPWM peripheral. TCPWM is configured as a Timer */
    TimerInit();
    printf("Observe the LED blinking on the kit!!!\r\n");

    for(;;)
    {
        if(LEDUpdateFlag)
        {
            /* Clear the flag */
            LEDUpdateFlag = false;

            /* Invert the LED state*/
            Cy_GPIO_Inv(KIT_LED2_PORT, KIT_LED2_PIN);
        }
    }
}

```



```

    }

    return(0);
}

/*****
* Function Name: UartInit
*****/
void UartInit(void)
{
    /* Configure the UART peripheral.
       UART_config structure is defined by the UART personality based on
       parameters entered in the Component configuration*/
    Cy_SCB_UART_Init(KIT_UART_HW, &KIT_UART_config, &KIT_UART_context);

    /* Enable the UART peripheral */
    Cy_SCB_UART_Enable(KIT_UART_HW);
}

/*****
* Function Name: TimerInit
*****/
void TimerInit(void)
{
    /* Configure the TCPWM peripheral.
       Counter_config structure is defined based on the parameters entered
       in the Component configuration */
    Cy_TCPWM_Counter_Init(Timer_HW, Timer_NUM, &Timer_config);

    /* Enable the initialized counter */
    Cy_TCPWM_Counter_Enable(Timer_HW, Timer_NUM);

    /* Start the enabled counter */
    Cy_TCPWM_TriggerStart(Timer_HW, Timer_MASK);

    /* Configure the ISR for the TCPWM peripheral*/
    Cy_SysInt_Init(&Isr_Timer_config, Isr_Timer);

    /* Enable interrupt in NVIC */
    NVIC_EnableIRQ((IRQn_Type)Isr_Timer_config.intrSrc);
}

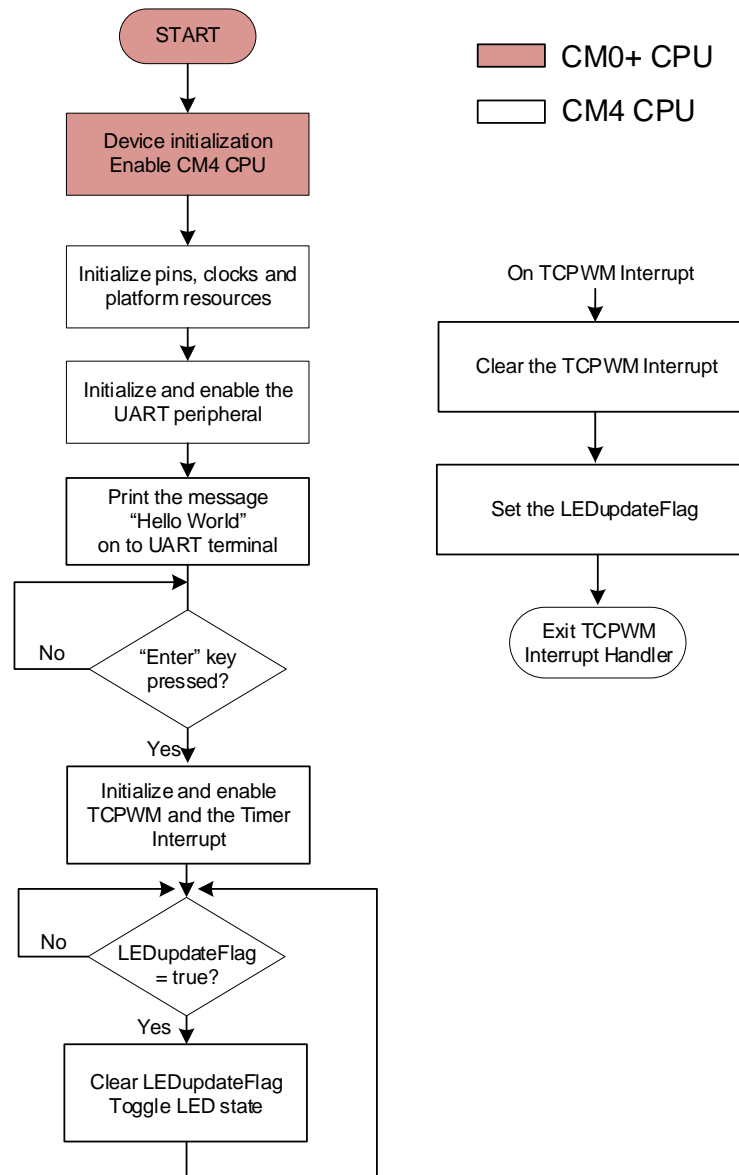
/*****
* Function Name: Isr_Timer
*****/
void Isr_Timer(void)
{
    /* Clear the TCPWM peripheral interrupt */
    Cy_TCPWM_ClearInterrupt(Timer_HW, Timer_NUM, CY_TCPWM_INT_ON_TC);

    /* Clear the CM4 NVIC pending interrupt for TCPWM */
    NVIC_ClearPendingIRQ(Isr_Timer_config.intrSrc);

    LEDUpdateFlag = true;
}

```

Figure 21. Firmware Flowchart



This completes the summary of how the firmware works in the code example. Feel free to explore the source files for a deeper understanding.

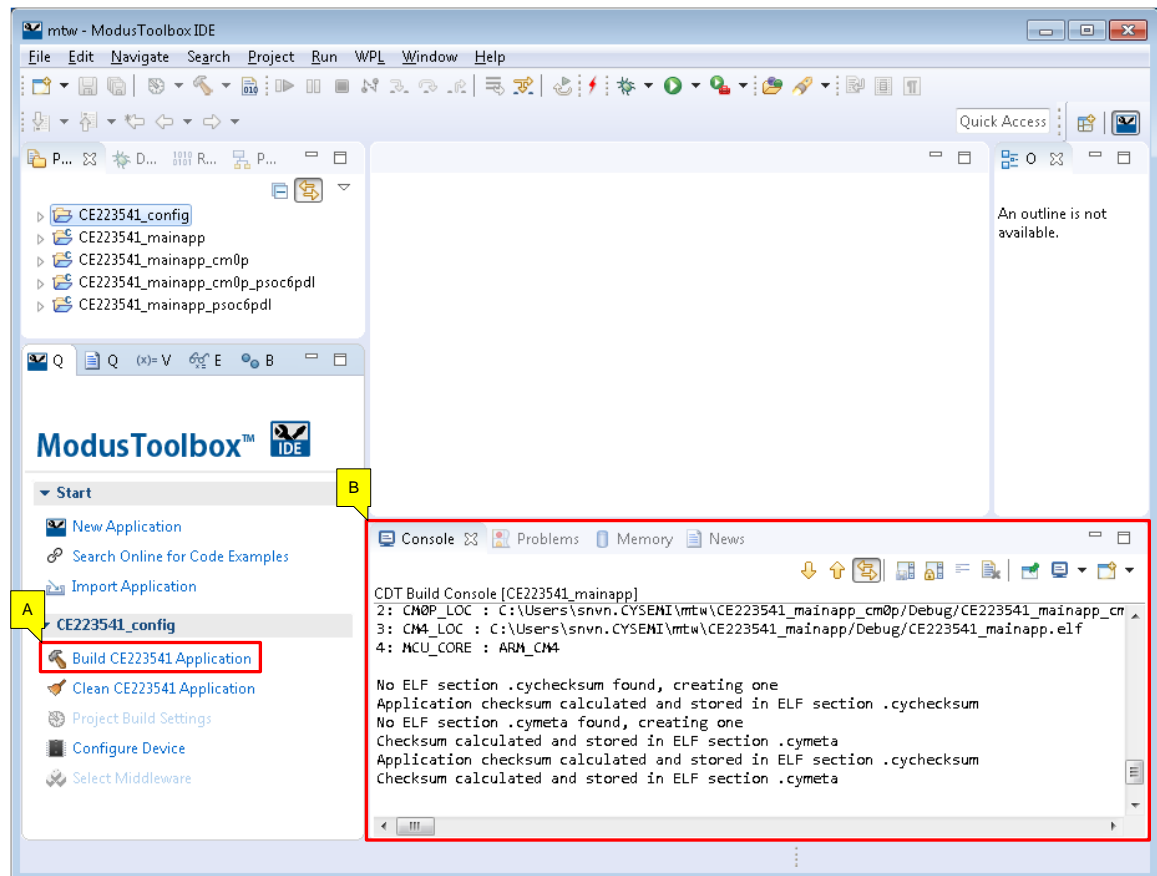
## 5.6 Part 4: Build the Application

This section shows how to build the application.

### 1. Build the Application.

- A. Click on the **Build <name> Application** shortcut under the **Start** group in the Quick Panel. It selects the **Debug** build configuration and compiles/links all projects that constitute the application.
- B. The **Console** view lists the results of the build operation, as [Figure 22](#) shows.

Figure 22. Build the Application



If you are working from scratch and encounter errors, revisit prior steps to ensure that you accomplished all the required tasks. You can work to resolve errors or switch to the code example for these final steps.

## 5.7 Part 5: Program the Device

This section shows how to program the PSoC 6 MCU device.

ModusToolbox uses the OpenOCD protocol to program and debug applications on PSoC 6 MCU devices. For ModusToolbox to identify the device on the kit, the kit must be running KitProg3. ModusToolbox includes a command-line tool **fw-loader** to update CY8CIT-062-WiFi-BT and CY8CKIT-062-BLE kits, and switch the KitProg firmware from KitProg2 to KitProg3. Refer to the **PSoC 6 MCU KitProg Firmware Loader** section in the ModusToolbox IDE User Guide for more details.

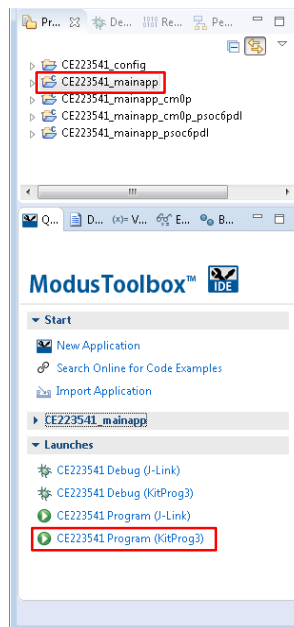
If you are using a development kit with a built-in programmer (the CY8CKIT-062-WiFi-BT Pioneer Kit, for example), connect the board to your computer using the USB cable.

If you are developing on your own hardware, you may need a hardware programmer/debugger; for example, a Cypress [CY8CKIT-005 MiniProg4](#).

### 1. Program the Application.

- A. Connect to the board and perform the following step.
- B. Select the CM4 `_mainapp` application project and click on the **<application name> Program (KitProg3)** shortcut under the **Launches** group in the Quick Panel, as [Figure 23](#) shows. The IDE will select and run the appropriate run configuration.

Figure 23. Programming an Application to a Device



The **Console** view lists the results of the programming operation, as [Figure 24](#) shows.

Figure 24. Console – Programming Results



## 5.8 Part 6: Test Your Design

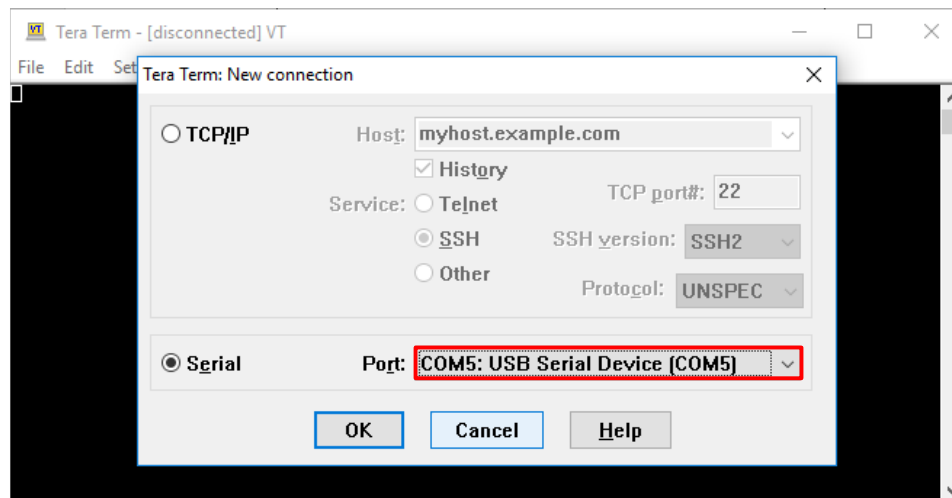
This section describes how to test your design.

Follow the steps below to observe the output of your design. Note that the below steps use Tera Term as the UART terminal emulator to view the results. You can use any terminal of your choice to view the output.

### 1. Select the serial port.

Launch Tera Term and select the USB-UART COM port as shown in [Figure 25](#).

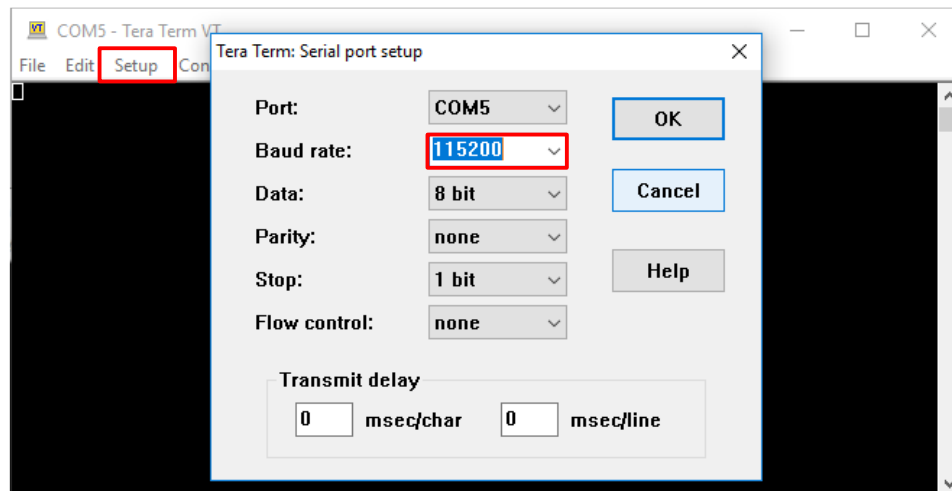
Figure 25. Selecting the USB-UART COM Port in Tera Term



### 2. Set the baud rate.

Set the baud rate to 115200 under **Setup > Serial port** as [Figure 26](#) shows.

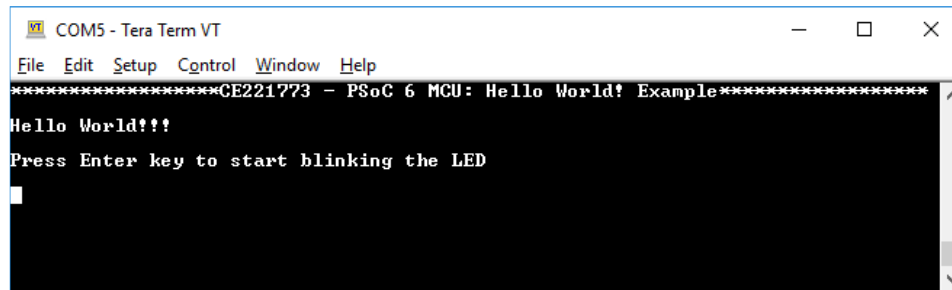
Figure 26. Configuring the Baud Rate in Tera Term



### 3. Reset the device.

Press the reset switch (**SW1**) on the Pioneer Kit. The following message appears on the terminal as [Figure 27](#) shows.

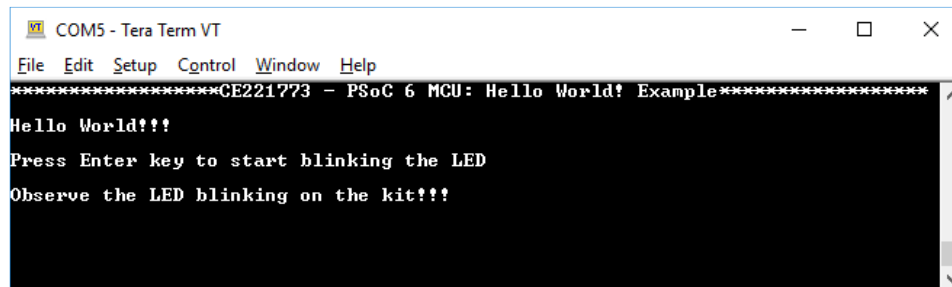
Figure 27. UART Message Printed from CM4 CPU



#### 4. Enable the LED Blinking functionality.

Press the **Enter** Key to start blinking the LED. When the LED starts blinking, the following message will be displayed on the UART terminal as shown in [Figure 28](#).

Figure 28. UART Message from CM4 CPU



## 6 My First PSoC 6 MCU Design Using PSoC Creator

This section does the following:

- Demonstrates how to build a simple PSoC 6 MCU-based design and program it on to the development kit.
- Provides detailed steps that make it easy to learn PSoC 6 MCU design techniques and how to use the PSoC Creator IDE.

### 6.1 Using These Instructions

These instructions are grouped into several sections. Each section is devoted to a particular phase of the application development workflow. The major sections are:

- [Part 1: Create a New Project from Scratch](#)
- [Part 2: Implement the Design](#)
- [Part 3: Generate Source Code](#)
- [Part 4: Write the Firmware](#)
- [Part 5: Build the Project and Program the Device](#)
- [Part 6: Test Your Design](#)

If you are familiar with developing projects with PSoC Creator, you can use the PSoC Creator version of the code example [CE221773 – PSoC 6 MCU Hello World Example](#) directly. It is a complete design, with all the firmware written. You can walk through the instructions and observe how the steps are implemented in the code example.

If you start from scratch and follow all the instructions in this application note, you use the code example as a reference while following the instructions.

You can download the code example from the Cypress website by clicking the link above. You can also use the PSoC Creator **File > Code Example** command. Set the **Device family** to PSoC 62. Select the PSoC MCU Hello World Example. Download the code example by clicking on the download icon adjacent to the example and then click on **Create Project**, and follow the on-screen instructions.

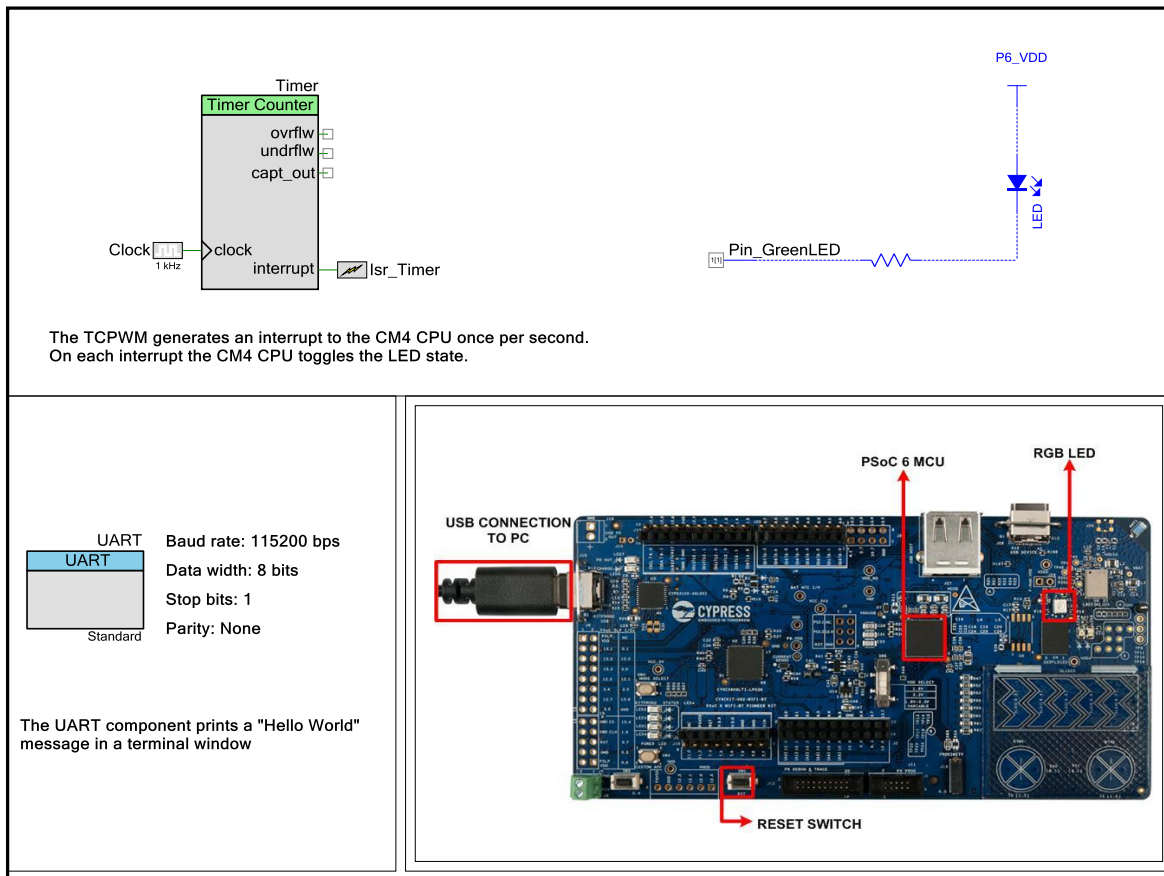
This design is developed for the [CY8CKIT-062-WiFi-BT PSoC 6 Wi-Fi-BT Pioneer Kit](#). You can also use [CY8CKIT-062-BLE PSoC 6 BLE Pioneer Kit](#) to test this example by selecting the appropriate device from the **Device Selector**.



## 6.2 About the Design

This design uses the CM4 CPU of PSoC 6 MCU to execute two tasks: UART communication and LED control. At device reset, the CM0+ CPU enables the CM4 CPU. The CM4 CPU uses the UART Component to print a “Hello World” message to the serial port stream and when the Enter Key is pressed by the user, the LED on the PSoC 6 MCU Wi-Fi-BT Pioneer Kit starts blinking.

Figure 29. My First PSoC 6 MCU Design



## 6.3 Part 1: Create a New Project from Scratch

This section takes you on a step-by-step guided tour of the design process. It starts with creating an empty project and guides you through hardware and firmware design development stages.

**Note:** These instructions assume that you are using PSoC Creator 4.2. The overall development process is the same for subsequent versions of PSoC Creator; but the user interface may change over time.

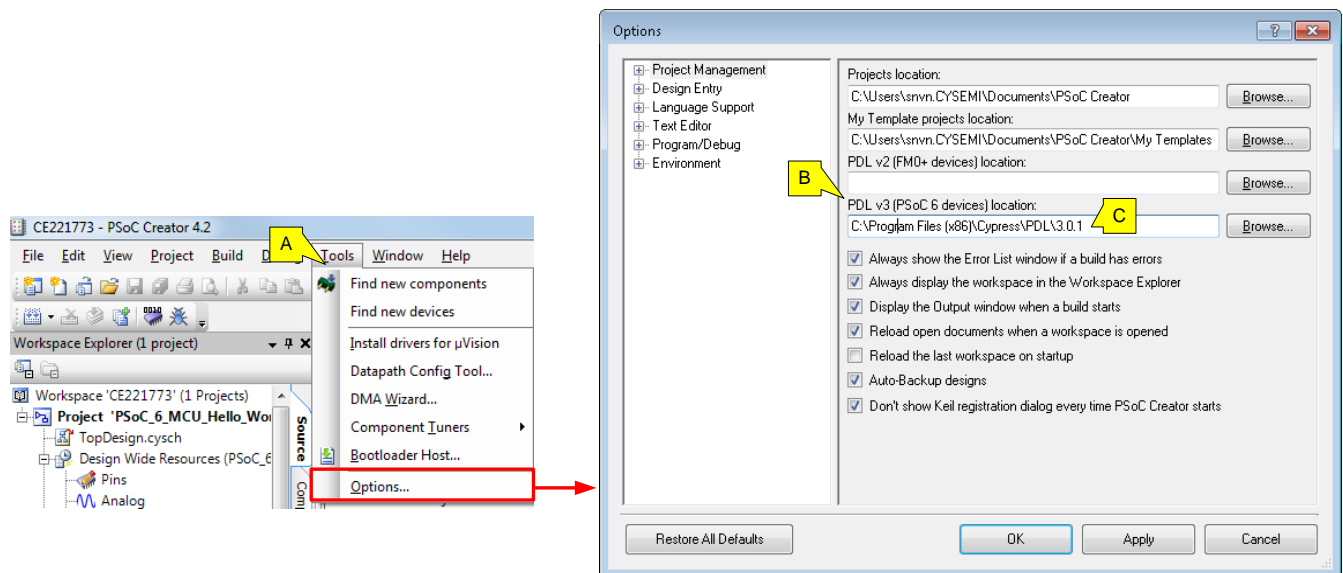
Launch PSoC Creator and get started.

### 2. Ensure that PSoC Creator can find the PDL.

This should be set correctly automatically during installation, but nothing works if this isn't set up right. Refer to [Figure 30](#) for help with this step.

- Choose **Tools > Options**.
- On the **Project Management** panel, check the path in the **PDL v3 (PSoC 6 Devices) location** field.
- Ensure that it is correct. If it is not, click the **Browse** button and locate the installed directory of the PDL. The default location is `C:\Program Files (x86)\Cypress\PDL\3.0.1`.

Figure 30. Peripheral Driver Library (PDL) Location

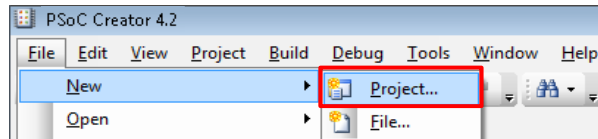


**Optional:** Jump to [Part 2: Implement the Design](#).

### 3. Create a new PSoC Creator project.

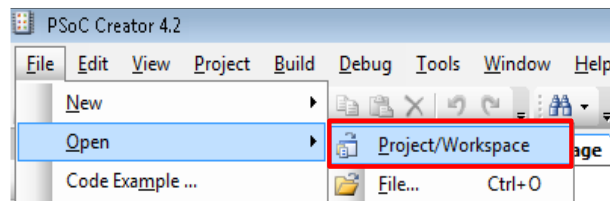
Choose **File > New > Project**, as Figure 31 shows. The **Create Project** window appears.

Figure 31. Create a New PSoC Creator Project



**Note:** If you are using the code example, choose **File > Open > Project/Workspace**, as Figure 32 shows. The **Open** window appears. Point to the location of the code example workspace and open the workspace.

Figure 32. Open Existing Code Example Workspace



### 4. Select PSoC 6 MCU as the target device.

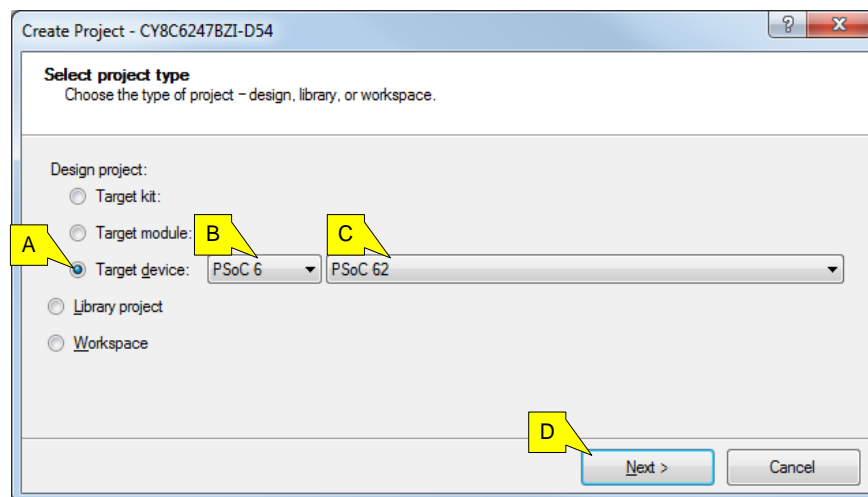
PSoC Creator speeds up the development process by automatically setting various project options for specified development kits or target devices. See Figure 33 for help with this step.

- A. Click **Target device**.
- B. In the family drop-down menu, select **PSoC 6**.
- C. In the device drop-down menu, select **PSoC 62**.
- D. Click **Next**. The Select project template panel appears.

PSoC Creator uses CY8C6247BZI-D54 as the default device in the PSoC 6 MCU family. This device is mounted on the [CY8CKIT-062-WiFi-BT PSoC 6 Wi-Fi-BT Pioneer Kit](#).

If you are using custom hardware based on PSoC 6 MCU, or a different PSoC 6 MCU part number, this is the place you choose to **Launch Device Selector** option in **Target device** and select the appropriate part number.

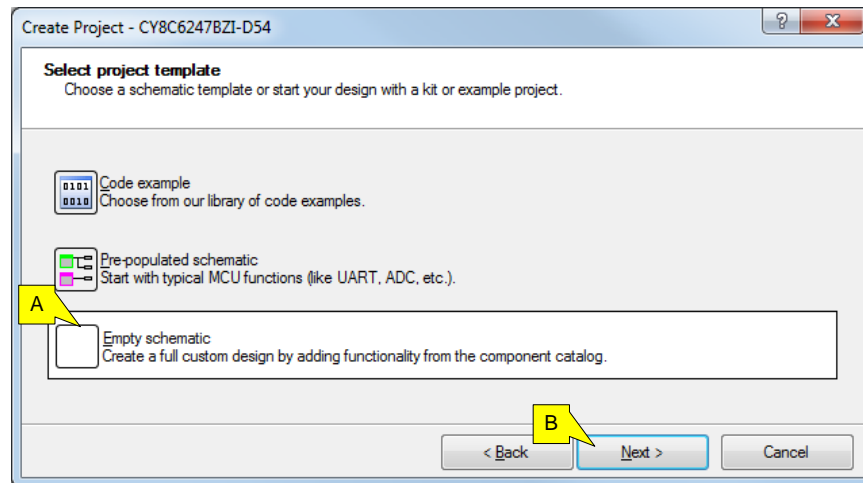
Figure 33. Selecting Target Device



## 5. Pick a project template.

- A. Choose **Empty Schematic**.
- B. Click **Next**.

Figure 34. Pick a Project Template

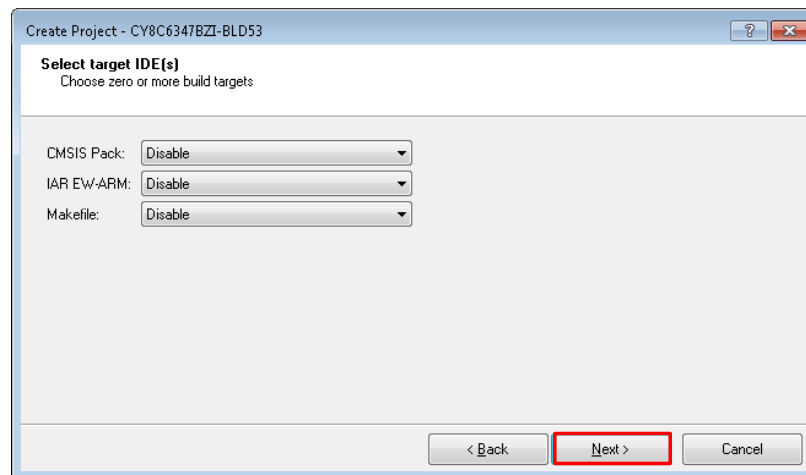


## 6. Select target IDE(s).

If you expect to export the code from the project, specify the target IDE. By default, all export options are disabled. You can modify this setting later if circumstances change.

Click **Next** to accept the default options.

Figure 35. Select Target IDEs (All Disabled)

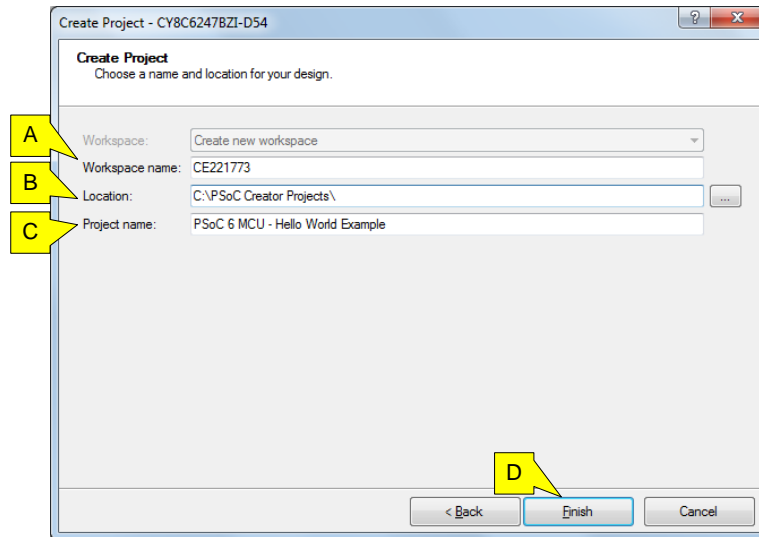


## 7. Create the project.

In this step, you set the name and location for your workplace, and a name for the project. See [Figure 36](#) for help with this step. A workspace is a container for one or more projects.

- A. Set the Workspace name.
- B. Specify the **Location** of your workspace.
- C. Set a **Project name**. The project and workspace names can be the same or different.
- D. Click **Finish**.

Figure 36. Project Naming and Location



You have successfully created a new PSoC Creator project.

## 6.4 Part 2: Implement the Design

Now that you have a project file, it is time to implement the hardware design using PSoC Creator Components. If you are using the code example directly, you already have a complete design.

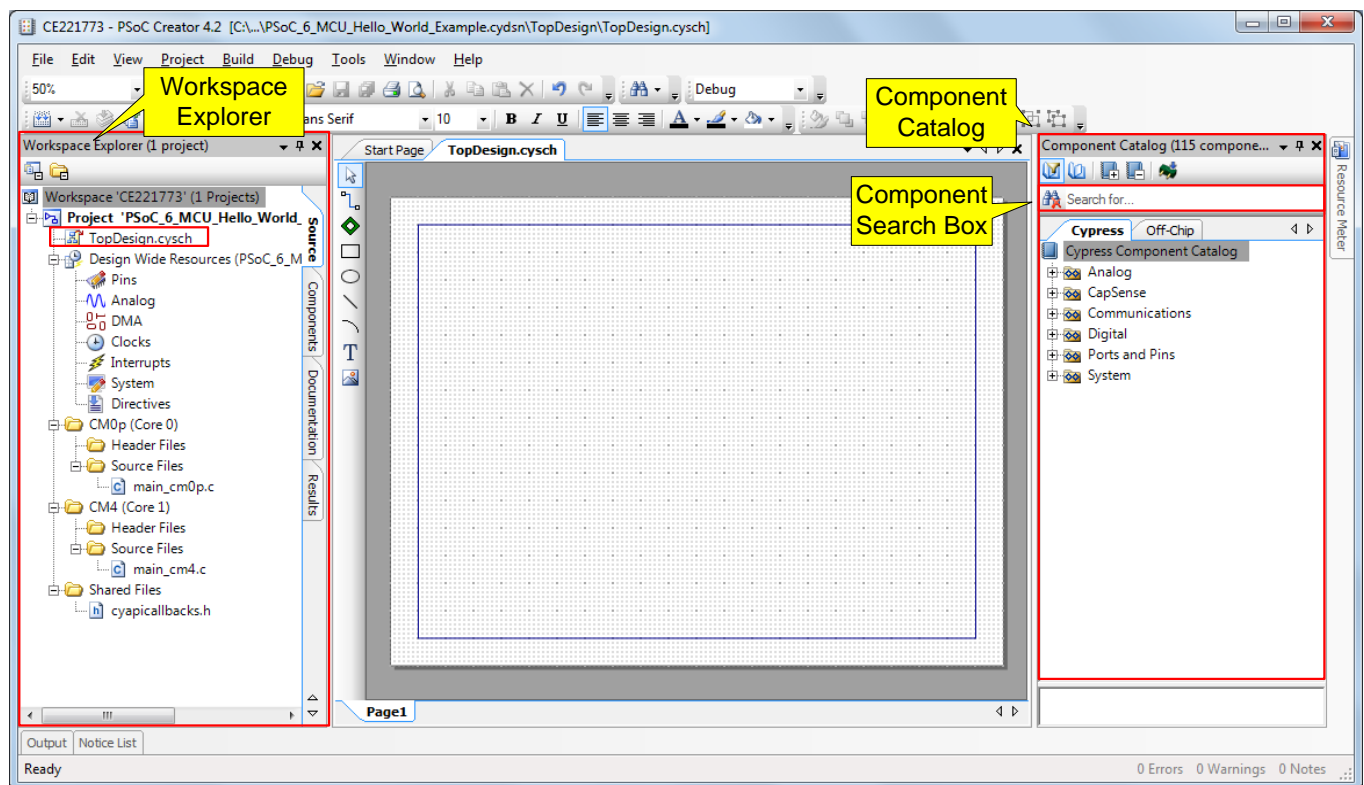
Before you implement the design, a quick tour of the PSoC Creator interface is in order.

Figure 37 shows the PSoC Creator application displaying an empty design schematic.

The project includes a project folder with a base set of files. You view these files in the **Workspace Explorer** pane to the left. The project schematic opens by default. This is the *TopDesign.cysch* file. Double-click the file name in the explorer pane to open the schematic at any time. In a new project, the schematic is empty. If you are using the code example, this is the schematic for the design.

The Component catalog is on the right side of the window. You can open it with the **View > Component Catalog** menu item. You can search for a particular Component by typing the name of the Component in the **Search for...** text box and then pressing the enter key. See Figure 37.

Figure 37. Schematic and Component Catalog

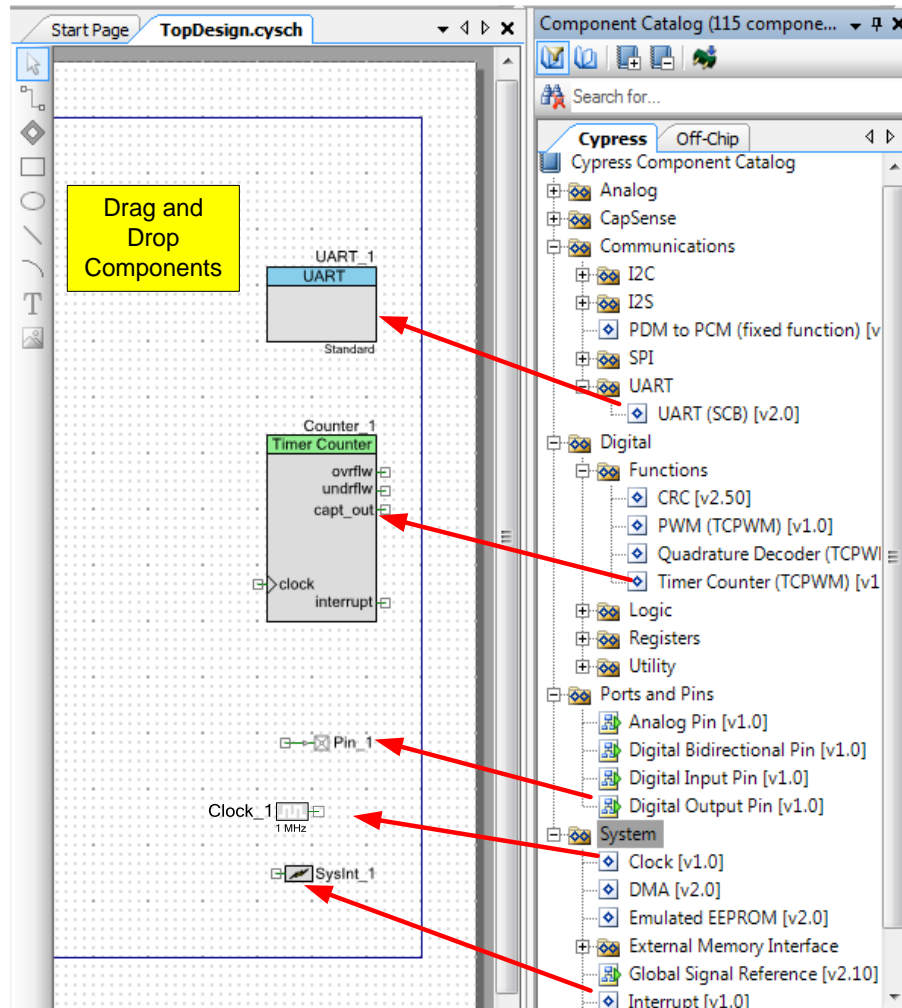


### 8. Place Components in the design.

This design uses several Components: three digital output pins, a UART, a Watchdog Timer, and an Interrupt. In this step, you add them to the design. You configure them in subsequent steps. Figure 38 shows the result.

- In the **Component Catalog**, expand the **Communications** group, drag a **UART (SCB)** Component into the schematic, and drop it. It doesn't matter where you put a Component.
- Expand the **Ports and Pins** group, and drag a **Digital Output Pin** into the design.
- Expand the **Digital** group, and drag a **Timer Counter (TCPWM)** Component into the design.
- Expand the **System** group, and drag an **Interrupt** Component and a **Clock** Component into the design.

Figure 38. Place Components in the Design



PSoC Creator gives each Component a default name and properties. Default values may or may not be suitable for any given design. In subsequent steps, you modify the name and some of the properties.

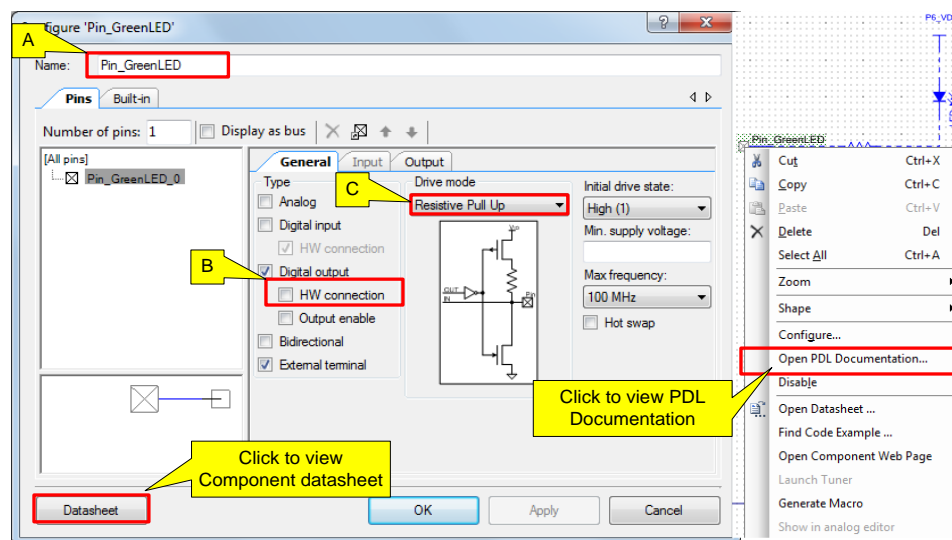
## 9. Configure the LED pin.

The output pin drives the LED. The LED on the PSoC 6 Wi-Fi-BT Pioneer Kit is active LOW; that is, the logic HIGH pin-drive state turns OFF the LED, and the logic LOW pin-drive state turns it ON. Figure 39 shows the configuration.

Double-click the Component placed on the schematic to open the configuration dialog. Then perform the following steps.

- D. Change the name of the Component instance to **Pin\_GreenLED**.
- E. Deselect **HW connection**. The firmware will drive the pin.
- F. Set the **Drive Mode** to Resistive Pull Up.

Figure 39. Configuring an Output Pin Component

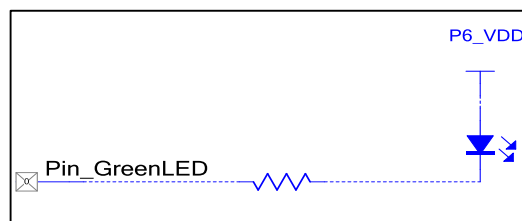


**Tip:** Each Component has an associated datasheet that can be accessed from the configuration window. The Component datasheet provides more information on the Component configuration, the application programming interface (API), and the electrical specifications.

**Tip:** You can open the API reference document of the associated PDL driver of a Component by right-clicking the Component and clicking on **Open PDL Documentation...** link. See Figure 39.

**Tip:** For a pin, if you enable **External terminal**, you can add external “off-chip” Components to a design. External Components on the schematic are included for descriptive purposes only; they have no effect on the generated code. Off-chip Components are optional, but can assist the hardware design team understanding how the design works. You can also add text boxes to a design with descriptions. Figure 40 shows how you could enhance the design for the LED. In this case, the off-chip components were configured with the **Instance\_Name\_Visible** option unchecked. The resistor was configured with the **Value** field left blank. The power terminal was configured with the **Supply\_Name** set to P6\_VDD.

Figure 40. An Output Pin with Off-Chip Components





## 10. Configure the UART Component.

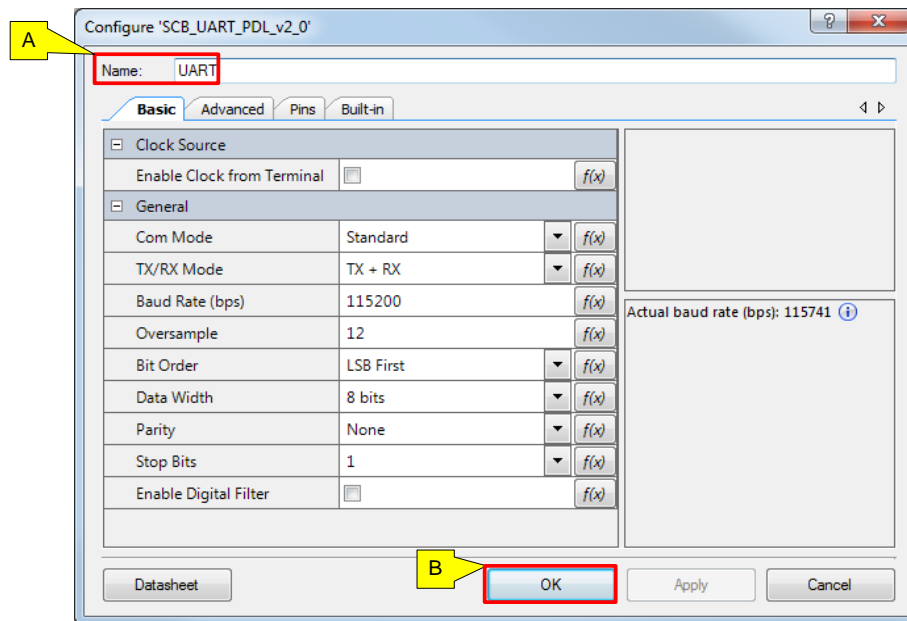
Double-click the Component to open the configuration window. The design uses this Component to display messages in a terminal window at a baud rate of 115200 bps.

A. Change the **Name** of the Component instance to **UART**.

B. Click **OK**.

The design uses default values for all other settings.

Figure 41. Configuring the SCB-Based UART Component



## 11. Configure the Timer Counter (TCPWM) Component to trigger an interrupt.

In this step, you configure the Timer Counter (TCPWM) Component to trigger an interrupt every second (1 Hz). The clock source of the TCPWM is the peripheral clock (Clk\_Per). The design will use this interrupt to toggle the LED state. Open the Component customizer and follow the steps illustrated in [Figure 42](#).

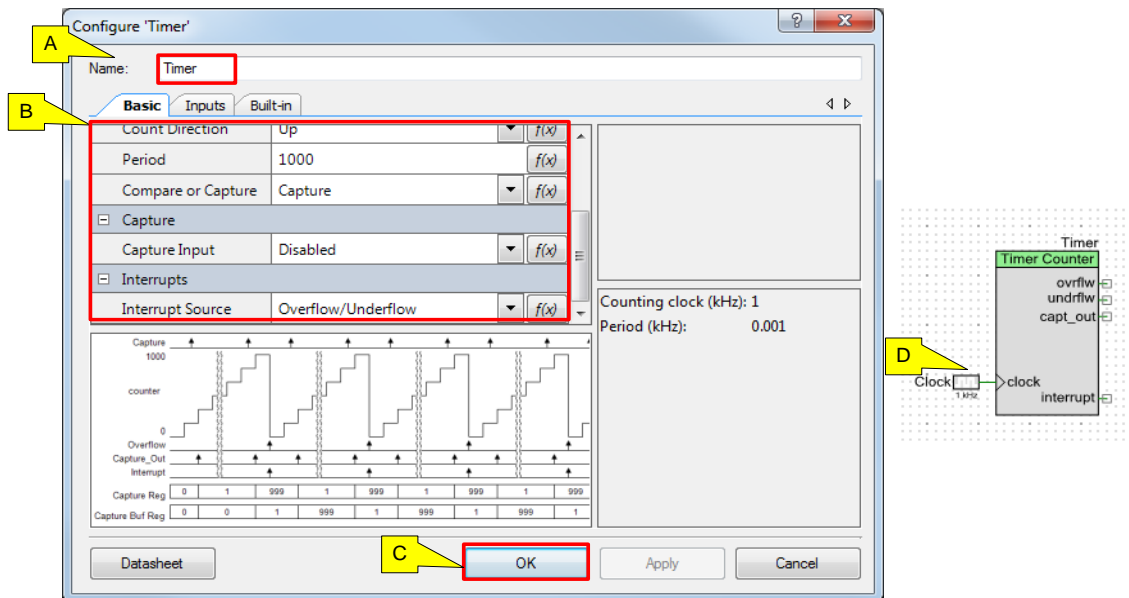
A. Change the **Name** to **Timer**.

B. Set the **Period** to 1000 and **Interrupt Source** as Overflow/Underflow.

C. Click **OK** to complete the configuration of the TCPWM Component.

D. Connect the clock terminal of the TCPWM to the 1-kHz clock source. In the schematic, use the wire tool button or press the 'W' key to start wiring the Clock Component to the clock terminal of the TCPWM Component.

Figure 42. Configuring the TCPWM Component

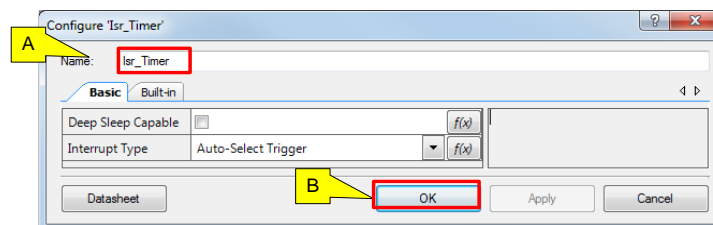


## 12. Configure the interrupt Component.

In this step, you configure the SysInt Component to map the TCPWM interrupt to the CM4 CPU. Open the Component customizer and follow the steps illustrated in Figure 43.

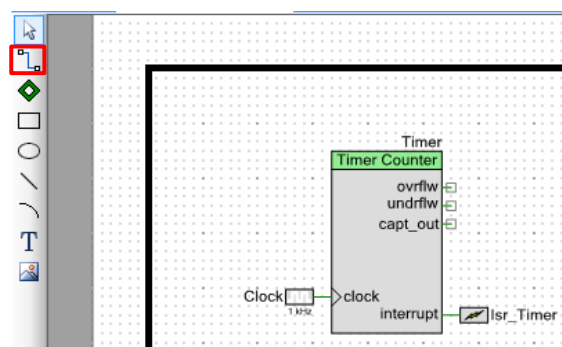
- A. Change the **Name** to **Isr\_Timer**.
- B. Click **OK** to complete the configuration of the SysInt Component.

Figure 43. SysInt\_PDL Settings



As the final step, connect the interrupt output of the TCPWM Component to the Isr\_TCPWM Component input. This routes the TCPWM interrupt to the CM4 CPU (the selection of the CM4 CPU for this interrupt will be set in the system interrupt configuration in a later step). In the schematic, use the wire tool button or press the 'W' key to start wiring the Components.

Figure 44. Connect TCPWM Peripheral Interrupt to CM4 CPU



### 13. Set the physical pins for each Pin Component.

One task remains to complete the design. You must associate each Component with the required physical pins on the device. The choice of which pin to use is driven by the board design. You can find this information in the kit schematic. Figure 45 shows the result of this step. You can connect external LEDs to the selected pins.

To set a pin, type either the port number or pin number in the corresponding field, or use the drop-down menu to pick the port or pin. Typically, the port number is used instead of the pin number since these names are independent of the specific package being used.

- A. Open the pin selector.

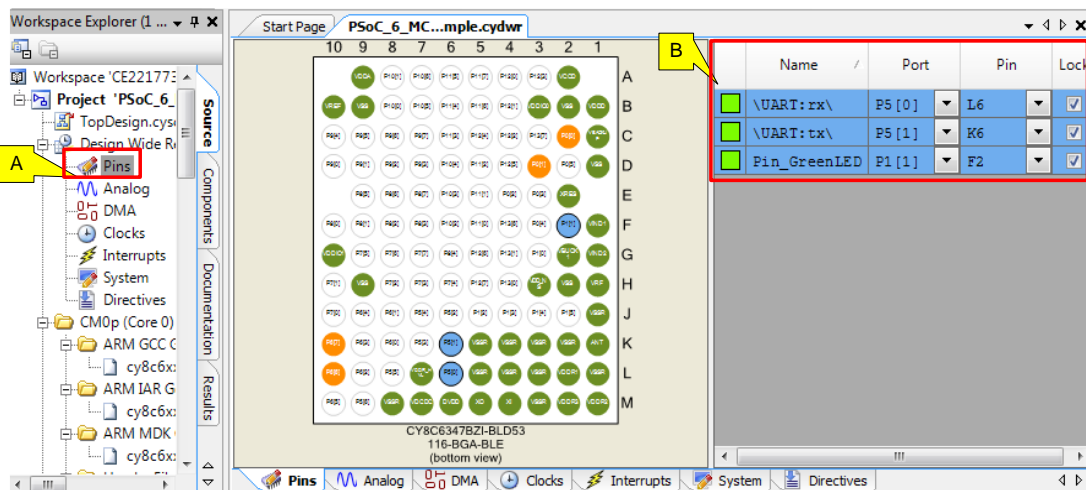
In the **Workspace Explorer** pane, double-click the **Pins** item under the Design Wide Resources. The pin selector for this device appears.

- B. Set each pin as shown in Table 1.

Table 1. Physical Pin Assignments for CY8CKIT-062-WiFi-BT Pioneer Kit

Pin Component Name	Port Name
UART: rx	P5[0]
UART: tx	P5[1]
Pin_GreenLED	P1[1]

Figure 45. Pin Assignment



### 14. Configure System Clock.

The design uses default values for the high-frequency system clock settings. Although you do not modify high frequency clocks for this design, you should know how PSoC Creator manages them. If you are working with your own board, you may need to modify these clocks.

- A. In the **Workspace Explorer** pane, double-click the **Clocks** item under **Design Wide Resources**. The list of clocks appears.

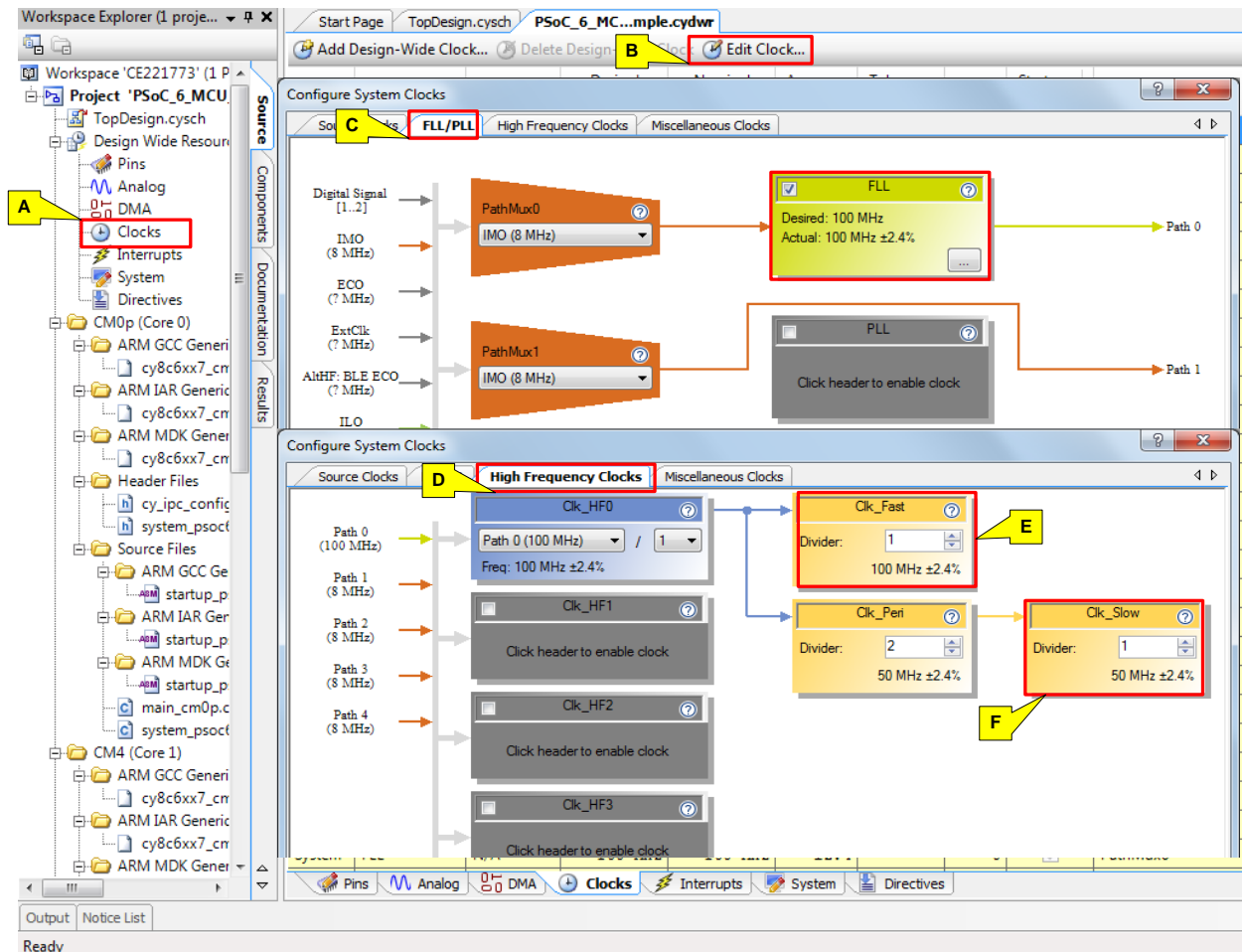
- B. Click **Edit Clock**. The **Configure System Clocks** dialog appears.

Here, you can see the clock tree, and modify the clocks as required. Note that there are tabs for different types of clocks such as **Source Clocks**, **FLL/PLL**, **High Frequency Clocks**, and **Miscellaneous Clocks**.

- C. Click on the **FLL/PLL** tab. By default, **PSoC Creator** enables **FLL** and sets the frequency to 100 MHz.
- D. Click on the **High Frequency Clocks** tab.
- E. You can set the CM4 CPU clock by setting the divider in **Clk\_Fast**. By default, the divider is set to 1.

- F. You can set the CM0+ CPU clock by setting the divider in **Clk\_Slow**. By default, the divider is set to 1. See Figure 46.

Figure 46. Clock Configuration



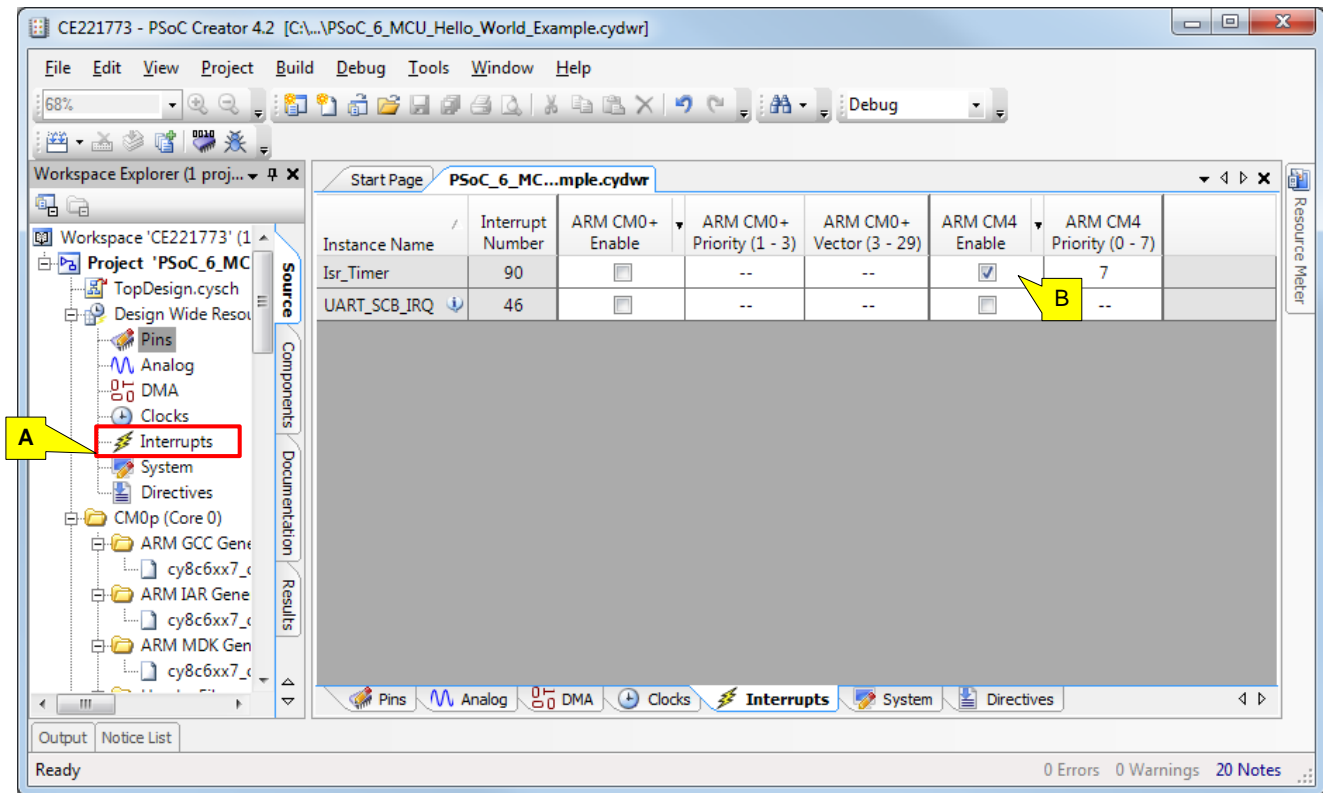
## 15. Configure System Interrupts.

In this step, you configure the system interrupts. See Figure 47.

- In the **Workspace Explorer** pane, double-click the **Interrupts** item under **Design Wide Resources**. The list of interrupts appears.
- Enable **Isr\_Timer** for the CM4 CPU.

The interrupt numbers are generated automatically by PSoC Creator when you generate the code in [Part 3: Generate Source Code](#).

Figure 47. Interrupt Configuration



The next part in the development process is to generate code.

**Note:** This exercise does not detail how to export your work to a target IDE. However, if you wish to use a target IDE, this is the point in the workflow where you would ensure that the correct target IDE is selected before you generate the source code.

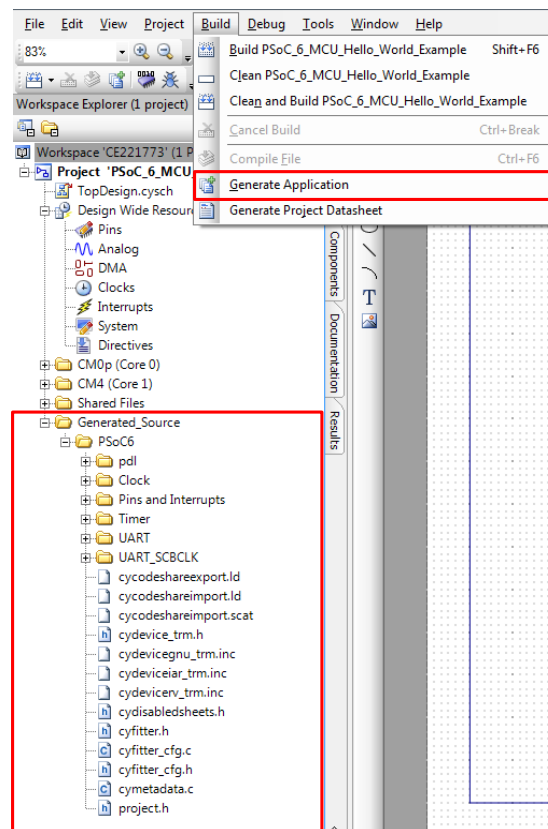
## 6.5 Part 3: Generate Source Code

PSoC Creator generates the source code based upon the design. The recommended workflow is to generate code before writing firmware. PSoC Creator will automatically create macros, constants, and API calls that you may then use in your firmware.

### 1. Generate the application.

Choose **Build > Generate Application**. PSoC Creator generates the source code based on the design and puts the files in the *Generated\_Source* folder. See [Figure 48](#). PSoC Creator will alert you to errors or problems that may occur. If you are working from scratch and encounter errors, revisit the configuration steps in [Part 2: Implement the Design](#) to ensure you have performed them correctly.

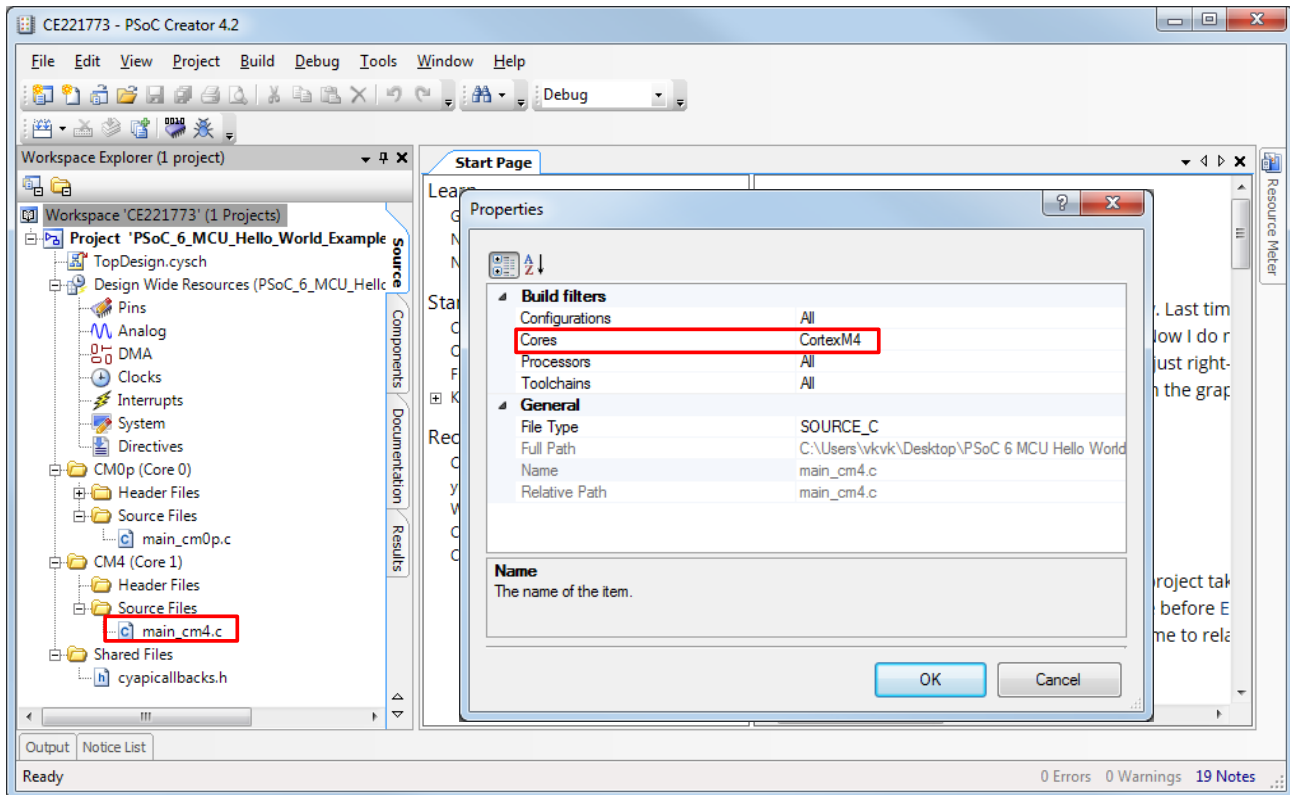
Figure 48. Generate Application



**Background:** PSoC 6 MCU is a dual-CPU platform. You can target firmware to run either on Cortex-M4 or Cortex-M0+. You set this at the source file level by accessing the file properties. Right-click on a source file, and select **Properties**. [Figure 49](#) shows the **Properties** dialog window. By default, the *main\_cm0p.c* file is targeted to the Cortex-M0+ and the *main\_cm4.c* file is targeted to the Cortex-M4. You do not need to modify the properties for any other file. They are already set in the code example.

By convention, files targeted to run on the CM0+ CPU are located in the *CM0p* folder and files targeted to run on the CM4 CPU are located in the *CM4* folder.

Figure 49. Setting Target Processor for a Source C File



## 6.6 Part 4: Write the Firmware

At this point in the development process, you have created a project, implemented a hardware design, and generated the code. In this part, you write the firmware that implements the design functionality.

The steps in this part discuss the firmware for the design that you configured in [Part 2: Implement the Design](#).

The code example has all the required code. If you are working from scratch, you can copy the respective source codes to *main\_cm0p.c* and *main\_cm4.c* from the code snippet provided in this section. If you are using the code example, files are already in your project.

### Firmware Flow

In the remaining steps, we examine code in the *main\_cm0p.c* and *main\_cm4.c* file.

When the PSoC 6 MCU device is reset, the firmware first performs system initialization, which includes setting up the CPUs for execution, enabling global interrupts, and enabling other Components used in the design.

The initialization is split across the CPUs. The CM0+ CPU comes out of reset and enables the CM4 CPU. The CM0+ CPU code snippet is given below. Copy the following code snippet to the *main\_cm0p.c* file of your project.

```
/* Header files includes*/
#include "project.h"
int main(void)
{
    __enable_irq(); /* Enable global interrupts. */

    /* Enable CM4. CY_CORTEX_M4_APPL_ADDR must be updated
       if CM4 memory layout is changed. */
    Cy_SysEnableCM4(CY_CORTEX_M4_APPL_ADDR);

    for(;;)
    {

    }
}

/* [] END OF FILE */
```

When the CM4 CPU is enabled, the UART Component is started and prints a "Hello World!" message on the terminal emulator. A Timer Counter PWM (TCPWM) Component is configured to generate an interrupt every second. At each interrupt, the CM4 CPU toggles the LED (**LED5**) state on the kit. Copy the following code snippet to *main\_cm4.c* of your project.

```
/* Header files includes*/
#include "project.h"

/*****
 * Macros
 *****/

#define LED_ON    (0)
#define LED_OFF   (!LED_ON)

/*****
 * Function Prototypes
 *****/

void UartInit(void);
void TimerInit(void);
void Isr_Timer(void);

/*****
```



```

* Global Variables
*****/
bool LEDUpdateFlag = false;

/*****

* Function Name: main
*****/

int main(void)
{
    /* Start the UART peripheral */
    UartInit();

    /* Enable global interrupts. */
    __enable_irq();

    /* \x1b[2J\x1b[H - ANSI ESC sequence for clear screen */
    Cy_SCB_UART_PutString(UART_HW, "\x1b[2J\x1b[H");

    Cy_SCB_UART_PutString(UART_HW, "*****CE221773 - PSoC 6 MCU:"\
        " Hello World! Example*****\r\n\n");

    Cy_SCB_UART_PutString(UART_HW, "Hello World!!!\r\n\n");
    Cy_SCB_UART_PutString(UART_HW, "Press Enter key to start blinking the LED\r\n\n");

    /* Wait for the user to Press Enter key */
    while(Cy_SCB_UART_Get(UART_HW) != '\r');

    /* Start the TCPWM peripheral. TCPWM is configured as a Timer */
    TimerInit();

    Cy_SCB_UART_PutString(UART_HW, "Observe the LED blinking on the kit!!!\r\n");

    for(;;)
    {
        if(LEDUpdateFlag)
        {
            /* Clear the flag */
            LEDUpdateFlag = false;

            /* Invert the LED state*/
            Cy_GPIO_Inv(Pin_GreenLED_0_PORT, Pin_GreenLED_0_NUM);
        }
    }
}

/*****

* Function Name: UartInit
*****/

void UartInit(void)
{
    /* Configure the UART peripheral.
    UART_config structure is defined by the UART_PDL component based on
    parameters entered in the Component configuration*/
    Cy_SCB_UART_Init(UART_HW, &UART_config, &UART_context);

```

```

    /* Enable the UART peripheral */
    Cy_SCB_UART_Enable(UART_HW);
}

/*****
* Function Name: TimerInit
*****/
void TimerInit(void)
{
    /* Configure the TCPWM peripheral.
       Counter_config structure is defined based on the parameters entered
       in the Component configuration */
    Cy_TCPWM_Counter_Init(Timer_HW, Timer_CNT_NUM, &Timer_config);

    /* Enable the initialized counter */
    Cy_TCPWM_Counter_Enable(Timer_HW, Timer_CNT_NUM);

    /* Start the enabled counter */
    Cy_TCPWM_TriggerStart(Timer_HW, Timer_CNT_MASK);

    /* Configure the ISR for the TCPWM peripheral*/
    Cy_SysInt_Init(&Isr_Timer_cfg, Isr_Timer);

    /* Enable interrupt in NVIC */
    NVIC_EnableIRQ((IRQn_Type)Isr_Timer_cfg.intrSrc);
}

/*****
* Function Name: Isr_Timer
*****/
void Isr_Timer(void)
{
    /* Clear the TCPWM peripheral interrupt */
    Cy_TCPWM_ClearInterrupt(Timer_HW, Timer_CNT_NUM, CY_TCPWM_INT_ON_TC );

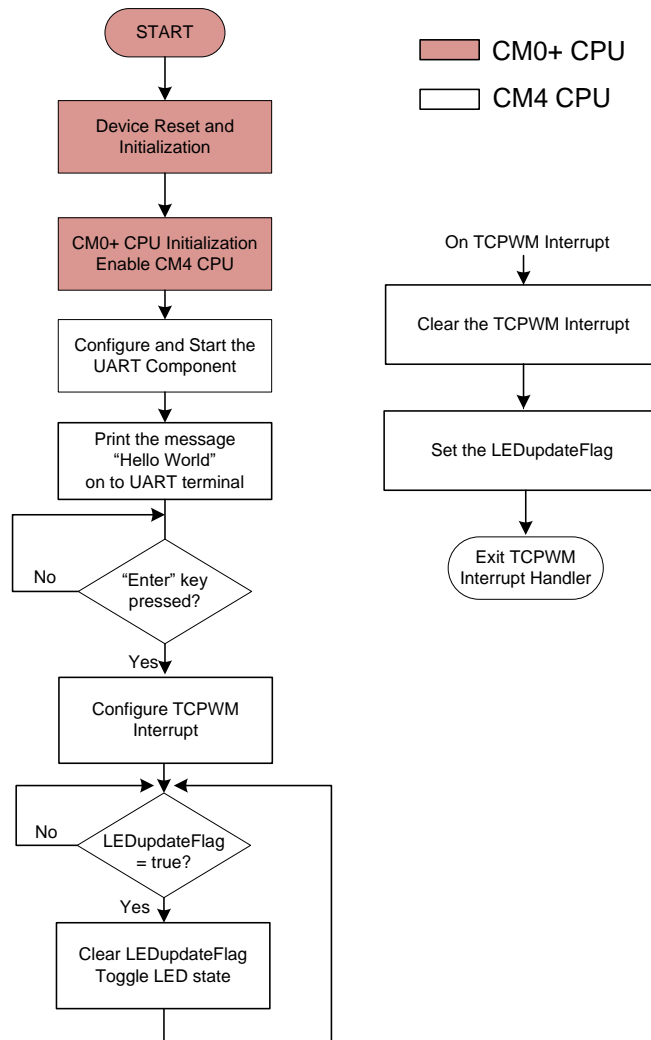
    /* Clear the CM4 NVIC pending interrupt for TCPWM */
    NVIC_ClearPendingIRQ(Isr_Timer_cfg.intrSrc);

    LEDupdateFlag = true;
}

/* [] END OF FILE */

```

Figure 50. Firmware Flowchart



This completes the summary of how the firmware works in the code example. Feel free to explore the source files for a deeper understanding.

## 6.7 Part 5: Build the Project and Program the Device

This section shows how to program the PSoC 6 MCU device. If you are using a development kit with a built-in programmer (the CY8CKIT-062-WiFi-BT Pioneer Kit, for example), connect the board to your computer using the USB cable. If you are developing on your own hardware, you may need a hardware programmer/debugger; for example, a Cypress [CY8CKIT-002 MiniProg3](#).

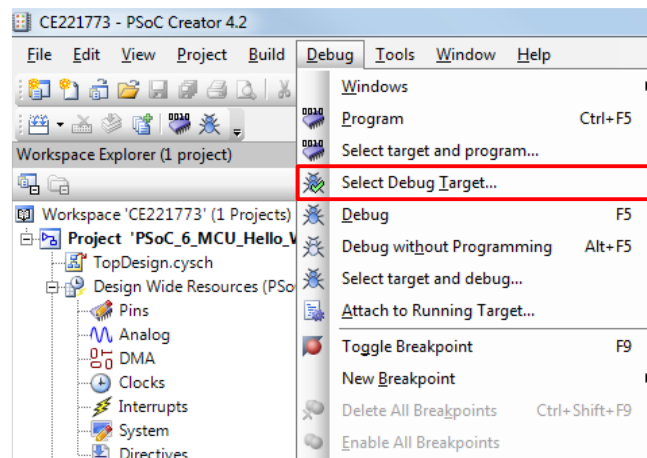
If you are working from scratch and encounter errors, revisit prior steps to ensure that you accomplished all the required tasks. You can work to resolve errors or switch to the code example for these final steps.

### 1. Select the debug target.

PSoC Creator can debug one CPU at a time.

- A. In PSoC Creator, choose **Debug > Select Debug Target**, as [Figure 51](#) shows.

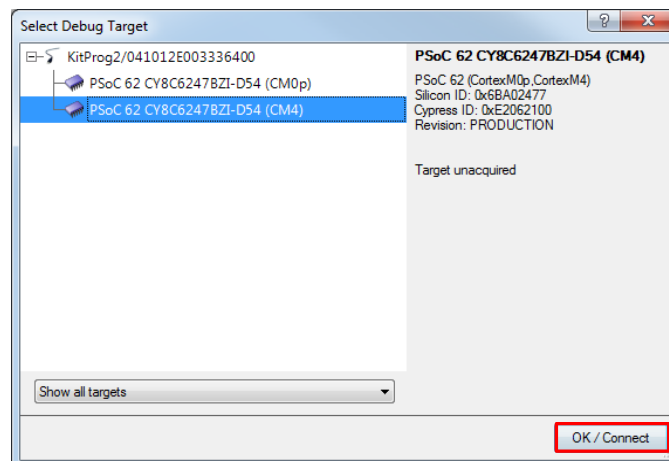
Figure 51. Selecting Debug Target



- B. Connect to the board.

In the **Select Debug Target** dialog box, select the CM4 target, then click **OK/Connect**, as [Figure 52](#) shows.

Figure 52. Connecting to a Device

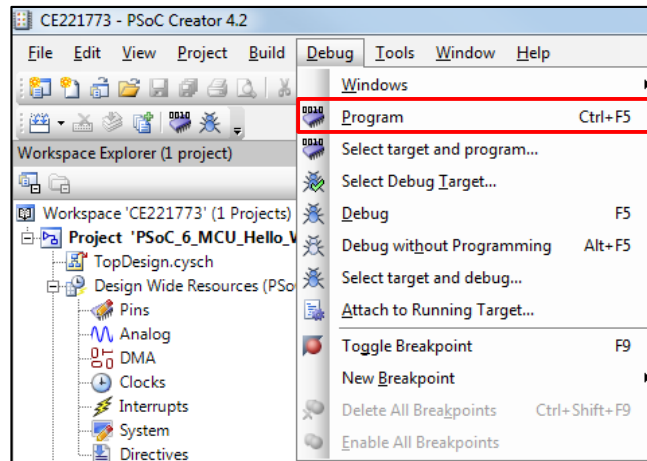


**TIP:** For programming the board, you can pick either target. The CPUs share the same memory space. Programming either CPU programs both CPUs. However, if you are debugging, this choice matters. The debugger will see only the CPU you connect to. These instructions do not use the debugger.

## 2. Program the board.

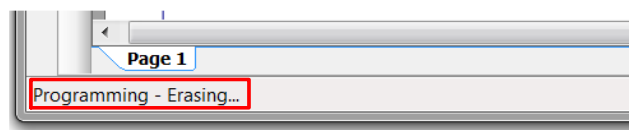
Choose **Debug > Program** to program the device with the project, as [Figure 53](#) shows.

Figure 53. Programming the Device



You can view the programming status in the lower left corner of the PSoC Creator window, as [Figure 54](#) shows.

Figure 54. Programming Status



**TIP:** The **Debug > Debug** command also programs the board. If any code needs to be generated or rebuilt, that happens automatically when you issue a **Program** or **Debug** command. You can also debug without programming the board. However, these instructions do not use the debugger.

**NOTE:** The KitProg2 firmware on the kit might require an update. See the respective kit user guide for step-wise instructions on updating the firmware.

## 6.8 Part 6: Test Your Design

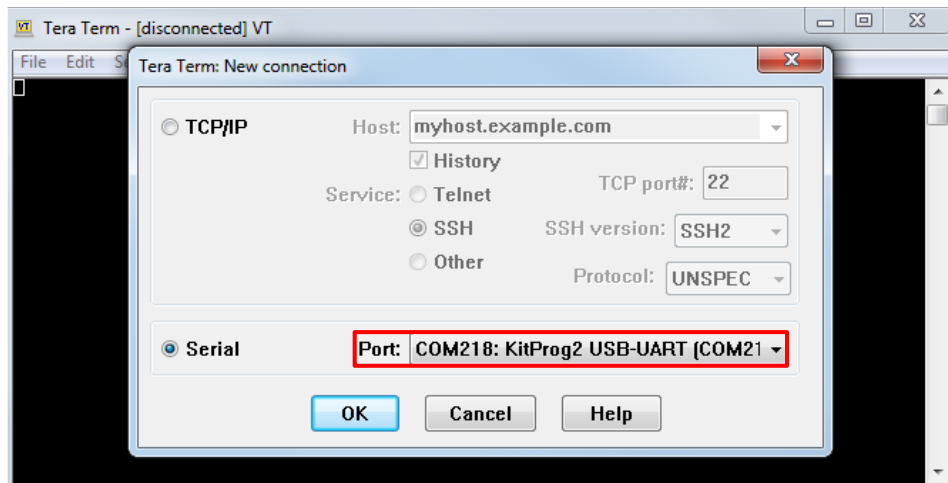
This section describes how to test your design.

Follow the steps below to observe the output of your design. Note that the below steps use Tera Term as the UART terminal emulator to view the results. You can use any terminal of your choice to view the output.

### 2. Select the serial port.

Launch Tera Term and select the KitProg2 USB-UART COM port as shown in [Figure 55](#).

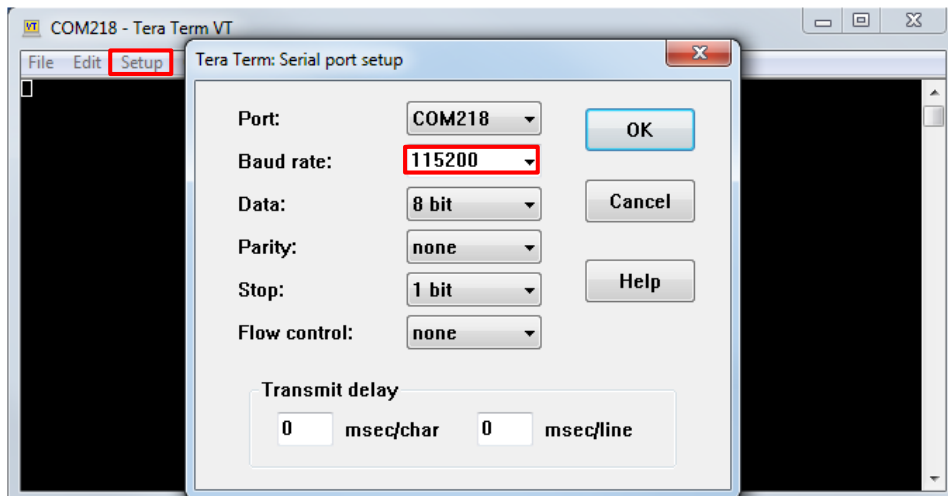
Figure 55. Selecting the KitProg2 USB-UART COM Port in Tera Term



### 3. Set the baud rate.

Set the baud rate to 115200 under **Setup > Serial port** as [Figure 56](#) shows.

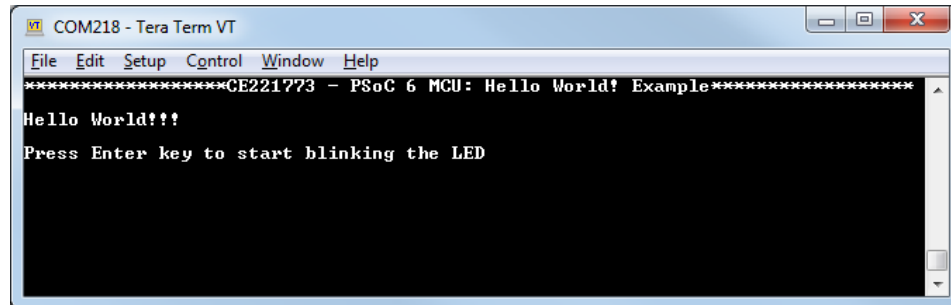
Figure 56. Configuring the Baud Rate in Tera Term



#### 4. Reset the device.

Press the reset switch (**SW1**) on the Pioneer Kit. The following message appears on the terminal as [Figure 57](#) shows.

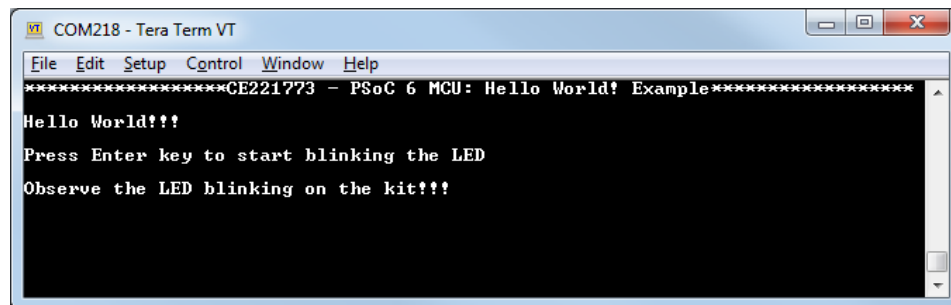
Figure 57. UART Message Printed from CM4 CPU



#### 5. Enable the LED Blinking functionality.

Press the **Enter** Key to start blinking the LED. When the LED starts blinking, the following message will be displayed on the UART terminal as shown in [Figure 58](#).

Figure 58. UART Message from CM4 CPU



## 7 Summary

This application note explored the PSoC 6 MCU device architecture and the associated development tools. PSoC 6 MCU is a truly programmable embedded system-on-chip with configurable analog and digital peripheral functions, memory, and a dual-CPU system on a single chip. The integrated features and low-power modes make PSoC 6 MCU an ideal choice for smart home, IoT gateways, and other related applications.

## Related Application Notes and Code Examples

For a complete and updated list of PSoC 6 MCU code examples, please visit our [code examples web page](#). For more PSoC 6 MCU-related documents, please visit our [PSoC 6 MCU](#) product web page.

[Table 2](#) lists the system-level and general application notes that are recommended for the next steps in learning about PSoC 6 MCU and PSoC Creator.

Table 2. General and System-Level Application Notes

Document	Document Name
<a href="#">AN210781</a>	Getting Started with PSoC 6 MCU with Bluetooth Low Energy (BLE) Connectivity
<a href="#">AN218241</a>	PSoC 6 MCU Hardware Design Considerations
<a href="#">AN219434</a>	PSoC 6 MCU Importing Generated Code into an IDE
<a href="#">AN219528</a>	PSoC 6 MCU Low-Power Modes and Power Reduction Techniques

[Table 3](#) lists the application notes (AN) and code examples (CE) for specific peripherals and applications.

Table 3. Documents Related to PSoC 6 MCU Features

Document	Document Name
<b>System Resources, CPU, and Interrupts</b>	
<a href="#">AN215656</a>	PSoC 6 MCU Dual-CPU System Design
<a href="#">AN217666</a>	PSoC 6 MCU Interrupts
<a href="#">CE221773</a>	PSoC 6 MCU Hello World Example
<a href="#">CE216795</a>	PSoC 6 MCU Dual-Core Basics
<a href="#">CE216825</a>	PSoC 6 MCU Real-Time Clock Basics
<a href="#">CE218129</a>	PSoC 6 MCU Wake up from Hibernate Using Low-Power Comparator
<a href="#">CE218541</a>	PSoC 6 MCU Fault-Handling Basics
<a href="#">CE218542</a>	PSoC 6 Custom Tick Timer Using RTC Alarm Interrupt
<a href="#">CE218552</a>	PSoC 6 MCU UART to Memory Buffer Using DMA
<a href="#">CE218964</a>	PSoC 6 MCU RTC Daily Alarm
<a href="#">CE219339</a>	PSoC 6 MCU MCWDT and RTC Interrupts (Dual Core)
<a href="#">CE219521</a>	PSoC 6 MCU GPIO Interrupt
<a href="#">CE219881</a>	PSoC 6 MCU Switching Power Modes
<a href="#">CE220060</a>	PSoC 6 MCU Watchdog Timer
<a href="#">CE220061</a>	PSoC 6 MCU Multi-Counter Watchdog Interrupts
<a href="#">CE220120</a>	PSoC 6 MCU Blocking Mode Flash Write
<a href="#">CE220169</a>	PSoC 6 MCU Periodic Interrupt Using TCPWM
<b>GPIO</b>	
<a href="#">CE219490</a>	PSoC 6 Breathing LED Using SMART IO
<a href="#">CE219506</a>	PSoC 6 Clock Buffer Using SMART IO
<a href="#">CE220263</a>	PSoC 6 MCU GPIO Pins Example
<b>CapSense</b>	
<a href="#">AN92239</a>	Proximity Sensing with CapSense
<a href="#">AN85951</a>	PSoC 4 and PSoC 6 MCU CapSense Design Guide
<b>Bootloader</b>	
<a href="#">AN213924</a>	PSoC 6 MCU Bootloader Software Development Kit (SDK) Guide
<a href="#">CE213903</a>	PSoC 6 MCU Basic Bootloaders



Document	Document Name
<b>Communications</b>	
<a href="#">CE220541</a>	PSoC 6 MCU SCB EzI2C
<b>Audio</b>	
<a href="#">CE218636</a>	PSoC 6 MCU Inter-IC Sound (I2S) Example
<a href="#">CE219431</a>	PSoC 6 MCU PDM-to-PCM Example
<b>RTOS</b>	
<a href="#">CE217911</a>	PSoC 6 MCU FreeRTOS™ Example Project
<b>Security</b>	
<a href="#">CE220465</a>	PSoC 6 MCU Cryptography – AES Demonstration
<a href="#">CE220511</a>	PSoC 6 MCU Cryptography – SHA Demonstration

## Appendix A. Glossary

This section lists the most commonly used terms that you might encounter while working with Cypress's PSoC family of devices.

**Component Customizer:** Simple GUI in PSoC Creator that is embedded in each Component. It is used to customize the Component parameters and is accessed by right-clicking a Component.

**Components:** Components are used to integrate multiple ICs and system interfaces into one PSoC Component that is inherently connected to the MCU via the main system bus. For example, the BLE Component creates Bluetooth Smart products in minutes. Similarly, you can use the Programmable Analog Components for sensors.

**KitProg:** The KitProg is an onboard programmer/debugger with USB-I2C and USB-UART bridge functionality. The KitProg is integrated onto most PSoC development kits.

**MiniProg3 / MiniProg4:** Programming hardware for development that is used to program PSoC devices on your custom board or PSoC development kits that do not support a built-in programmer.

**Personality:** A personality expresses the configurability of a resource for a functionality. For example, the SCB resource can be configured to be an UART, SPI or I2C personalities.

**PSoC:** A programmable, embedded design platform that includes a CPU, such as the 32-bit Arm Cortex-M0, with both analog and digital programmable blocks. It accelerates embedded system design with reliable, easy-to-use solutions, such as touch sensing, and enables low-power designs.

**ModusToolbox:** An Eclipse based embedded design platform for IoT designers that provides a single, coherent, and familiar design experience combining the industry's most deployed Wi-Fi and Bluetooth technologies, and the lowest power, most flexible MCUs with best-in-class sensing.

**PSoC Creator:** PSoC 3, PSoC 4, PSoC 5LP, PSoC 6 MCU and PSoC 6 BLE Integrated Design Environment (IDE) software that installs on your PC and allows concurrent hardware and firmware design of PSoC systems, or hardware design followed by export to other popular IDEs.

**Peripheral Driver Library:** The Peripheral Driver Library (PDL) simplifies software development for the PSoC 6 MCU architecture. The PDL reduces the need to understand register usage and bit structures, thus easing software development for the extensive set of peripherals available.

**PSoC Programmer:** A flexible, integrated programming application for programming PSoC devices. PSoC Programmer is integrated with PSoC Creator to program PSoC 3, PSoC 4, PSoC 6, PSoC 5LP, PSoC 6 MCU, and PSoC 6 BLE designs.

**WICED:** Cypress's WICED (Wireless Internet Connectivity for Embedded Devices) is a full-featured platform with proven Software Development Kits (SDKs) and turnkey hardware solutions from partners to readily enable Wi-Fi and Bluetooth connectivity in system design.

## Appendix B. PSoC 6 MCU Development Kits

### B.1 CY8CKIT-062-WiFi-BT PSoC 6 Wi-Fi-BT Pioneer Kit

The PSoC 6 Wi-Fi-BT Pioneer Kit shown in [Figure 59](#) is a development kit from Cypress that supports the PSoC 6 MCU family of devices. The following are the features of the PSoC 6 Wi-Fi-BT Pioneer kit baseboard:

- Expansion headers that are compatible with Arduino Uno 3.3-V shields and Digilent Pmod modules
- Type 1DX ultra-small 2.4-GHz WLAN and Bluetooth functionality module
- 512-Mbit external quad-SPI NOR flash that provides a fast, expandable memory for data and code
- KitProg2 onboard programmer/debugger with mass storage programming, USB to UART/I2C/SPI bridge functionality, and custom applications support
- EZ-PD CCG3 USB Type-C power delivery (PD) system with rechargeable lithium-ion polymer (Li-Po) battery support
- CapSense touch-sensing slider (five elements) and two buttons, all of which are capable of both self-capacitance (CSD) and mutual-capacitance (CSX) operation, and a CSD proximity sensor that allows you to evaluate Cypress' fourth-generation CapSense technology
- 1.8-V to 3.3-V operation of PSoC 6 MCU is supported. An additional 330-mF super-capacitor is provided for backup domain supply (Vbackup)
- Two user LEDs, an RGB LED, a user button, and a reset button for PSoC 6 MCU. Two buttons and three LEDs for KitProg2.

Figure 59. CY8CKIT-062-WiFi-BT PSoC 6 Wi-Fi-BT Pioneer Kit



The kit includes a TFT display shield with the following features:

- A 2.4-inch TFT LCD display with 240x320 pixel resolution.
- A three-axis acceleration and three-axis gyroscopic motion sensor.
- A PDM microphone for voice input.
- A 32-bit stereo codec with microphone, headphone, and speaker amplifier capability.
- An audio jack with a provision of connecting both AHJ and OMTP headphones. The headset standard can be set by an onboard switch.
- An ambient light sensor IC made of an NPN phototransistor.
- An LDO that converts 3.3 V to 1.8 V for the digital supply of the audio codec.

For more details, refer to [CY8CKIT-062-WiFi-BT PSoC 6 Wi-Fi-BT Pioneer kit user guide](#).

## B.2 CY8CKIT-062-BLE

The PSoC 6 BLE Pioneer Kit shown in [Figure 60](#) is a BLE development kit from Cypress that supports the PSoC 6 BLE family of devices.

Following are the features of the PSoC 6 BLE Pioneer kit baseboard:

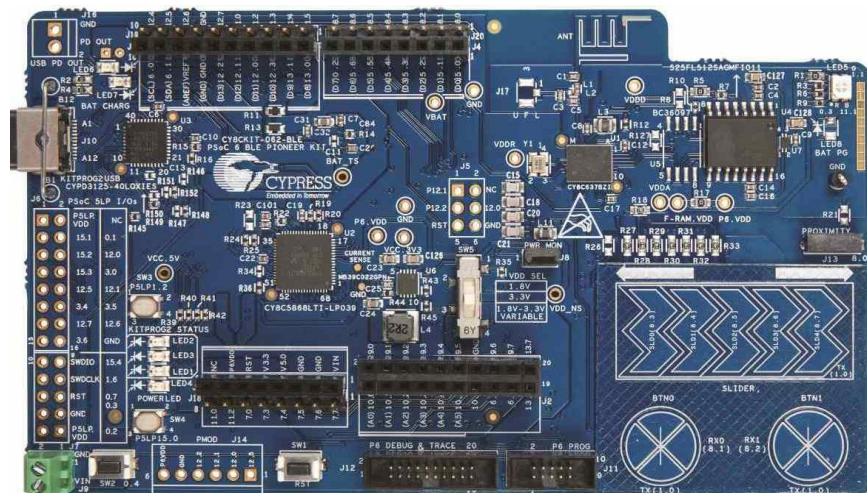
- Can be powered by a coin-cell battery or through the Type-C USB interface. The Type-C USB interface also supports up to 12 V, 3 A power delivery (PD) consumer and provider profiles.
- Enables development of battery-operated low-power BLE designs that work in conjunction with standard, Arduino Uno connector-compliant shields or the onboard PSoC 6 BLE device capabilities, such as the CapSense user interface and serial memory interface.
- Supports third-party programming, debugging, and tracing with the Cortex Debug/ETM connector.
- Includes an additional header that supports interfacing with Pmod™ daughter cards from third-party vendors such as Digilent.
- Supports PDM-PCM microphone for voice-over-BLE functionality.
- Includes QSPI NOR flash and F-RAM™.

The kit includes the following:

- A USB-BLE dongle that acts as a BLE link master and works with the CySmart Host Emulation Tool to provide a BLE host emulation platform on non-BLE Windows PCs.
- An E-INK display.

The kit consists of a set of BLE example projects and documentation that help you get started on developing your own BLE applications. Visit the [CY8CKIT-062-BLE PSoC 6 BLE Pioneer Kit](#) webpage to get the latest updates on the kit and download the kit design, example projects, and documentation files.

Figure 60. CY8CKIT-062-BLE PSoC 6 BLE Pioneer Kit



### B.3 CY8CPROTO-063-BLE

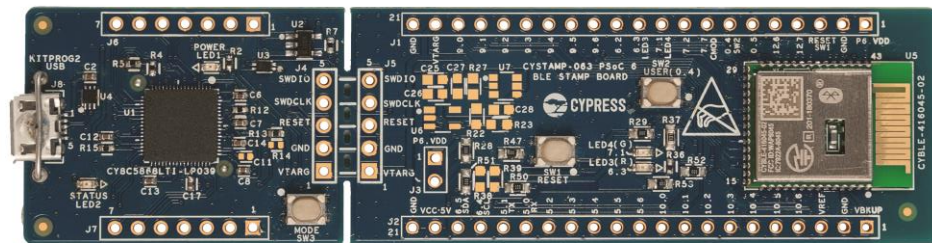
The PSoC 6 BLE Prototyping Kit shown in [Figure 61](#) is a development kit from Cypress that supports the PSoC 6 BLE family of devices. It offers an open footprint breakout board to maximize the end utility of the PSoC 6 MCU with Bluetooth Low Energy Connectivity (PSoC 6 BLE) device. This kit provides a low-cost alternative to device samples while providing a platform to easily develop and integrate the PSoC 6 BLE device into your end-system.

Following are the features of the PSoC 6 BLE Prototyping kit:

- CYBLE-416045-02 PSoC 6 BLE module.
- 3.3-V operation.
- Two user LEDs, a user button, and a reset button for PSoC 6. One mode switch and two LEDs for KitProg2.

The kit consists of a set of IoT example projects and documentation that help you get started on developing your own IoT applications. Visit the [CY8CPROTO-063-BLE PSoC 6 BLE Prototyping Kit](#) webpage to get the latest updates on the kit and download the kit design, example projects, and documentation files.

Figure 61. CY8CPROTO-063-BLE PSoC 6 BLE Prototyping Kit





## B.4 CY8CPROTO-062-4343W

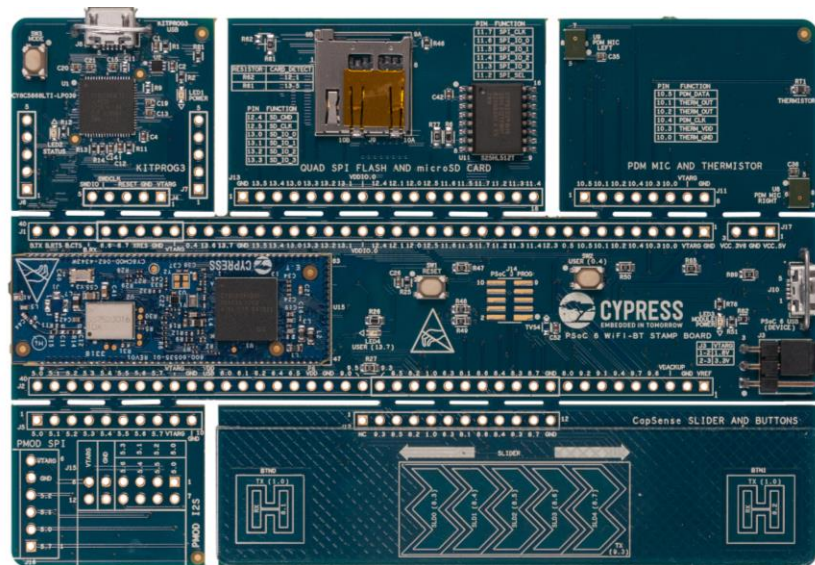
The PSoC 6 Wi-Fi-BT Prototyping Kit shown in [Figure 60](#) is a development kit from Cypress that supports the PSoC 6 MCU family of devices with 2 MB flash memory.

Following are the features of the PSoC 6 Wi-Fi-BT Prototyping kit:

- PSoC 6 MCU with SDHC.
- Type 1DX ultra-small 2.4-GHz WLAN and Bluetooth functionality module based on CYW4343W.
- microSD card slot.
- 512-Mbit external quad-SPI NOR Flash that provides a fast, expandable memory for data and code.
- A thermistor to measure ambient temperature and two PDM microphones for voice input.
- KitProg3 onboard programmer/debugger with CMSIS-DAP mode, USB to UART/I2C bridge functionality.
- CapSense touch-sensing slider (5 elements), two buttons, all of which are capable of both self-capacitance (CSD) and mutual-capacitance (CSX) operation.
- A micro-B connector for USB device interface.
- Expansion headers that are compatible with Digilent Pmod modules.
- 1.8-V to 3.3-V operation of PSoC 6 MCU is supported.
- One user LED, a user button, and a reset button for PSoC 6 MCU. One mode switch and two LEDs for KitProg3.

The kit consists of a set of IoT example projects and documentation that help you get started on developing your own IoT applications. Visit the [CY8CPROTO-062-4343W PSoC 6 Wi-Fi BT Prototyping Kit](#) webpage to get the latest updates on the kit and download the kit design, example projects, and documentation files.

Figure 62. CY8CPROTO-062-4343W PSoC 6 Wi-Fi BT Prototyping Kit



## Document History

Document Title: AN221774 – Getting Started with PSoC 6 MCU

Document Number: 002-21774

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	6049560	SNVN, VKVK	03/28/2018	New application note
*A	6332672	SNVN	10/04/2018	Updated for ModusToolbox
*B	6372351	SNVN	10/31/2018	Updated images
*C	6385422	SNVN	11/15/2018	Updated for public release

## Worldwide Sales and Design Support

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

### Products

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

### PSoC® Solutions

[PSoC 1](#) | [PSoC 3](#) | [PSoC 4](#) | [PSoC 5LP](#) | [PSoC 6 MCU](#)

### Cypress Developer Community

[Community](#) | [Projects](#) | [Videos](#) | [Blogs](#) | [Training](#) | [Components](#)

### Technical Support

[cypress.com/support](http://cypress.com/support)

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



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

© Cypress Semiconductor Corporation, 2018. This document is the property of Cypress Semiconductor Corporation and its subsidiaries, including Spanion LLC ("Cypress"). This document, including any software or firmware included or referenced in this document ("Software"), is owned by Cypress under the intellectual property laws and treaties of the United States and other countries worldwide. Cypress reserves all rights under such laws and treaties and does not, except as specifically stated in this paragraph, grant any license under its patents, copyrights, trademarks, or other intellectual property rights. If the Software is not accompanied by a license agreement and you do not otherwise have a written agreement with Cypress governing the use of the Software, then Cypress hereby grants you a personal, non-exclusive, nontransferable license (without the right to sublicense) (1) under its copyright rights in the Software (a) for Software provided in source code form, to modify and reproduce the Software solely for use with Cypress hardware products, only internally within your organization, and (b) to distribute the Software in binary code form externally to end users (either directly or indirectly through resellers and distributors), solely for use on Cypress hardware product units, and (2) under those claims of Cypress's patents that are infringed by the Software (as provided by Cypress, unmodified) to make, use, distribute, and import the Software solely for use with Cypress hardware products. Any other use, reproduction, modification, translation, or compilation of the Software is prohibited.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS DOCUMENT OR ANY SOFTWARE OR ACCOMPANYING HARDWARE, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. No computing device can be absolutely secure. Therefore, despite security measures implemented in Cypress hardware or software products, Cypress does not assume any liability arising out of any security breach, such as unauthorized access to or use of a Cypress product. In addition, the products described in these materials may contain design defects or errors known as errata which may cause the product to deviate from published specifications. To the extent permitted by applicable law, Cypress reserves the right to make changes to this document without further notice. Cypress does not assume any liability arising out of the application or use of any product or circuit described in this document. Any information provided in this document, including any sample design information or programming code, is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. Cypress products are not designed, intended, or authorized for use as critical components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or system could cause personal injury, death, or property damage ("Unintended Uses"). A critical component is any component of a device or system whose failure to perform can be reasonably expected to cause the failure of the device or system, or to affect its safety or effectiveness. Cypress is not liable, in whole or in part, and you shall and hereby do release Cypress from any claim, damage, or other liability arising from or related to all Unintended Uses of Cypress products. You shall indemnify and hold Cypress harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of Cypress products.

Cypress, the Cypress logo, Spanion, the Spanion logo, and combinations thereof, WICED, PSoC, CapSense, EZ-USB, F-RAM, and Traveo are trademarks or registered trademarks of Cypress in the United States and other countries. For a more complete list of Cypress trademarks, visit [cypress.com](http://cypress.com). Other names and brands may be claimed as property of their respective owners.