

Getting started with PSoC™ 6 MCU on ModusToolbox™ software

About this document

Scope and purpose

This application note 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 the PSoC™ 6 MCU architecture and development tools and shows you how to create your first project using the Eclipse IDE for ModusToolbox™ software. This application note also guides you to more resources available online to accelerate your learning about PSoC™ 6 MCU.

Intended audience

This document is intended for the users who are new to PSoC™ 6 MCU and ModusToolbox™ software.

Associated part family

All **PSoC™ 6 MCU** devices

Software version

ModusToolbox™ software 2.2 or above

More code examples? We heard you.

To access an ever-growing list of PSoC™ code examples using ModusToolbox™, please visit the [GitHub](#) site. You can also explore the PSoC™ video library [here](#).

Table of contents

About this document	1
Table of contents	1
1 Introduction	3
2 Development ecosystem	6
2.1 PSoC™ resources	6
2.2 Firmware/application development	6
2.2.1 Choosing an IDE	7
2.2.2 ModusToolbox™ software.....	7
2.2.3 PSoC™ 6 Software Resources.....	9
2.2.3.1 Configurators	9
2.2.3.2 Library management for PSoC™ 6 MCU	10
2.2.3.3 Software development for PSoC™ 6 MCU	10
2.2.4 ModusToolbox™ help	12
2.3 Support for other IDEs.....	13
2.4 FreeRTOS support with ModusToolbox™	13
2.5 Programming/debugging	13
2.6 PSoC™ 6 MCU development kits	14
3 Device features	15



Table of contents

4	My first PSoC™ 6 MCU design using Eclipse IDE for ModusToolbox™ software	17
4.1	Prerequisites.....	17
4.1.1	Hardware	17
4.1.2	Software	17
4.2	Using these instructions.....	17
4.3	About the design	18
4.4	Part 1: Create a new application.....	18
4.5	Part 2: View and modify the design	20
4.6	Part 3: Write firmware	24
4.7	Part 4: Build the application	30
4.8	Part 5: Program the device.....	30
4.9	Part 6: Test your design.....	32
5	Summary	34
	References.....	35
	Glossary	36
	Revision history.....	37

Introduction

1 Introduction

PSoC™ 6 MCU is an 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:
 - 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
 - Wearables

The **ModusToolbox™ software environment** supports PSoC™ 6 MCU application development with a set of tools for configuring the device, setting up peripherals, and complementing your projects with world-class middleware. See the **Infineon** GitHub repos for BSPs for all kits, libraries for popular functionality like CAPSENSE™ and emWin, and a comprehensive array of **example applications** to get you started.

Figure 1 illustrates an application-level block diagram for a real-world use case using PSoC™ 6 MCU.

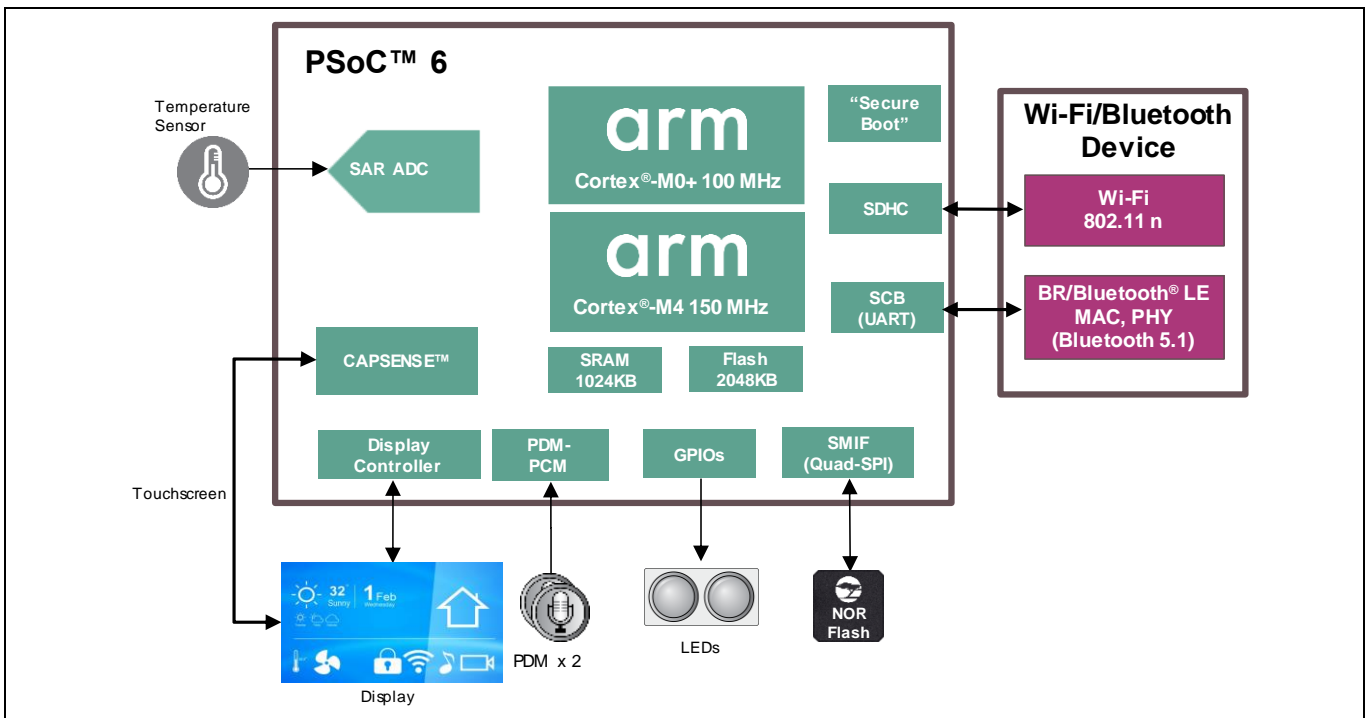


Figure 1 Application-level block diagram using PSoC™ 6 MCU

Introduction

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
- Programmable Digital logic (Smart I/O) and peripherals (Timer Counter PWM or TCPWM) to drive the display and LEDs respectively
- SDIO interface to a Wi-Fi/Bluetooth® device to provide IoT cloud connectivity
- Product security features managed by CM0+ CPU and application features executed by CM4 CPU

There are four product lines in PSoC™ 6 which cater to different application needs. [Table 1](#) provides overview of different product lines:

Table 1 PSoC™ 6 MCU product lines

Product Line	Security firmware	Device series	Details	Applications
Programmable	No	CY8C61x	Single core: 150-MHz Arm® Cortex®-M4	IoT gateways, smart home, home appliances, HMI, audio processing, and industrial concentrators
Performance	No	CY8C62x	Dual-core architecture: 150-MHz Arm® Cortex®-M4 and 100-MHz Cortex®-M0+	
Connectivity	No	CY8C63x	Dual-core architecture: 150-MHz Arm® Cortex®-M4 and 100-MHz Cortex®-M0+ Bluetooth® low energy (LE) 5.0 radio with 2-Mbps data throughput	Wearables, portable medical, industrial IoT, and smart home
Security	“Secure Boot”	CYB064x	150-MHz Arm® Cortex®-M4 for the user application Hardware isolated, 100-MHz Arm® Cortex®-M0+ with privileged access to memory and peripherals for security functions Bluetooth® LE 5.0 radio with 2-Mbps data throughput Arm® Platform Security Architecture Certifications- PSA L1, FIPS 140-2	
	AWS Standard “Secure”	CYS064x	150-MHz Arm® Cortex®-M4 for the user application Hardware isolated, 100-MHz Arm® Cortex®-M0+ with privileged access to memory and peripherals for security functions Arm® Platform Security Architecture Certifications- PSA L2	IoT gateways, smart home, home appliances, HMI, audio processing, and industrial concentrators

Note that not all the features available in all the devices in a product line. See the [device datasheets](#) for more details.

Introduction

This application note introduces you to the capabilities of the PSoC™ 6 MCU, gives an overview of the development ecosystem, and gets you started with a simple ‘Hello World’ application wherein you learn to use the PSoC™ 6 MCU. We will show you how to create the application from an empty starter application, but the completed design is available as a [code example for ModusToolbox™ on GitHub](#).

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

Development ecosystem

2 Development ecosystem

2.1 PSoC™ resources

A wealth of data available [here](#) helps 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 [How to design with PSoC™ 6 MCU - KBA223067](#). The following is an abbreviated list of resources for PSoC™ 6 MCU.

- **Overview:** [PSoC™ portfolio](#), [PSoC™ roadmap](#)
- [PSoC™ 6 MCU webpage](#)
- **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:** Many low-cost [kits and shield boards](#) are available for evaluation, design, and development of different applications using PSoC™ 6 MCUs.
- **Training videos:** [Video training](#) on our products and tools, including a dedicated series on [PSoC™ 6 MCU](#) is available.
- **Technical Support:** [PSoC™ 6 community forum](#), [Knowledge base articles](#)

2.2 Firmware/application development

There are two development platforms that you can use for application development with PSoC™ 6 MCU:

- **ModusToolbox™:** ModusToolbox™ software includes configuration tools, low-level drivers, middleware libraries, and operating system support, as well as other packages that enable you to create MCU and wireless applications. It also includes an Eclipse IDE, which provides an integrated flow with all the ModusToolbox™ tools.

ModusToolbox™ supports stand-alone device and middleware configurators that are fully integrated into the Eclipse 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. It is recommended that you use ModusToolbox™ for all application development for PSoC™ 6 MCUs. See the [ModusToolbox™ user guide](#) for more information.

Libraries and enablement software are available at the [GitHub](#) site.

Software resources available at GitHub support one or more of the target ecosystems:

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

ModusToolbox™ tools and resources can also be used in the command line. See the Using Command Line section in the [ModusToolbox™ user guide](#) for detailed documentation.

Development ecosystem

- **PSoC™ Creator:** A proprietary IDE that runs on Windows only. It supports a subset of PSoC™ 6 MCU devices as well as other PSoC™ device families such as PSoC™ 3, PSoC™ 4, and PSoC™ 5LP. See [AN221774 - Getting started with PSoC™ 6 on PSoC™ Creator](#) for more information.

2.2.1 Choosing an IDE

ModusToolbox™ software, the latest-generation toolset, includes the Eclipse IDE and therefore is supported across Windows, Linux, and macOS platforms. Eclipse IDE for ModusToolbox™ is integrated with quick launchers for tools and design configurators in the Quick Panel. ModusToolbox™ also supports 3rd-party IDEs, including Visual Studio Code, Arm® MDK (μVision), and IAR Embedded Workbench. 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. New versions of ModusToolbox™ will be released in the future to support these features and improve the user experience.

Use ModusToolbox™ to take advantage of the power and extensibility of an Eclipse-based IDE. ModusToolbox™ is supported on Windows, Linux, and macOS. It is recommended to use ModusToolbox™ if you want to build an IoT application using IoT devices, or if you are using a PSoC™ 6 MCU device not supported on PSoC™ Creator.

PSoC™ Creator is the long-standing 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, and PSoC™ 5LP devices, and a subset of PSoC™ 6 MCU devices.

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.

2.2.2 ModusToolbox™ software

ModusToolbox™ is a set of tools and software that enables an immersive development experience for creating converged MCU and wireless systems and enables you to integrate our devices into your existing development methodology. To achieve this goal, ModusToolbox™ leverages popular third-party ecosystems such as FreeRTOS and Arm® Mbed, and adds specific features for Wi-Fi, Bluetooth®, CAPSENSE™, and security.

Eclipse IDE for ModusToolbox™ is a multi-platform development environment that supports application configuration and development.

Figure 2 shows a high-level view of the tools/resources included in the ModusToolbox™ software. For a more in-depth overview of the ModusToolbox™ software, see [ModusToolbox™ user guide](#).

Development ecosystem

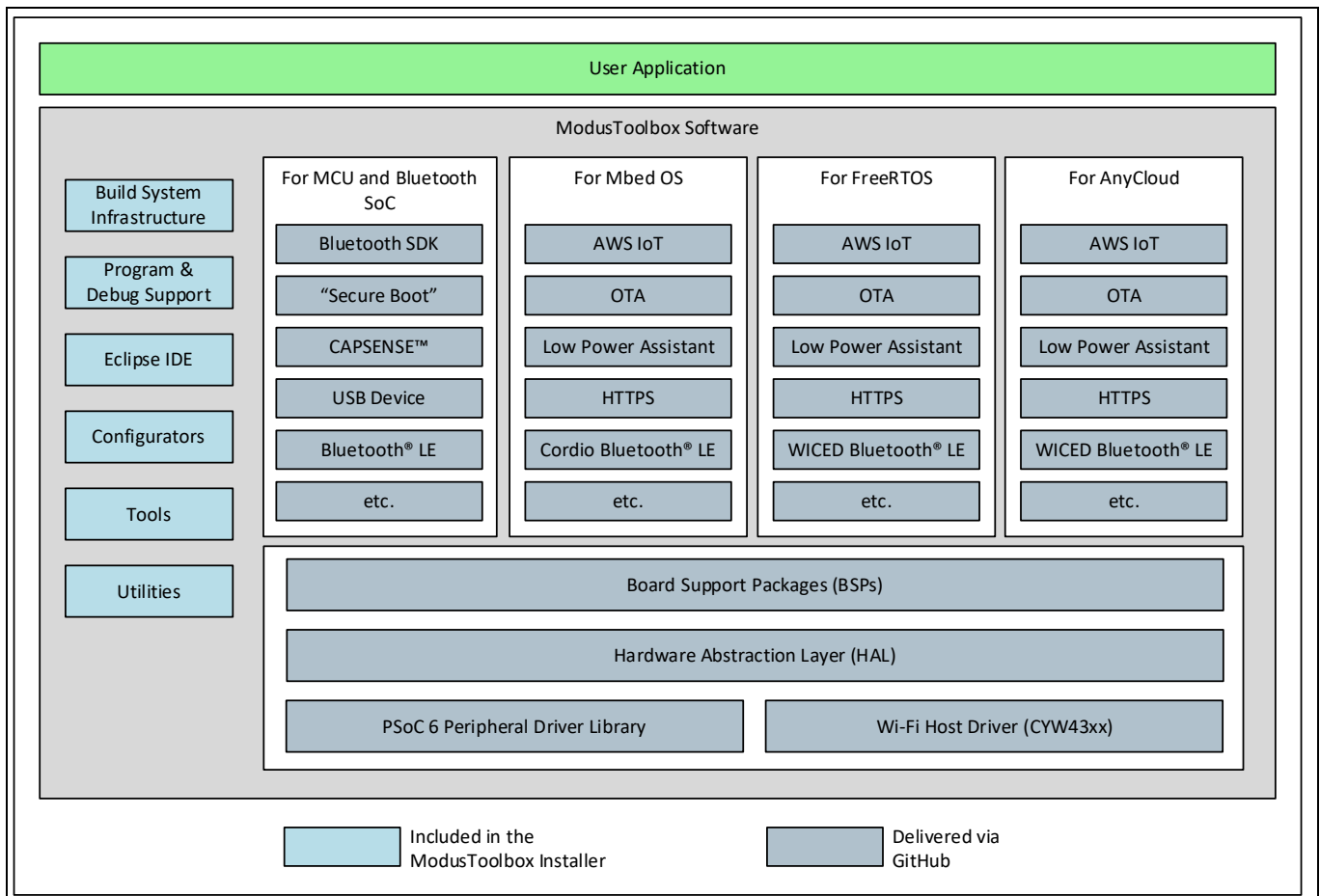


Figure 2 ModusToolbox™ software

ModusToolbox™ installer includes the design configurators and tools, and the build system infrastructure.

The build system infrastructure includes the new project creation wizard that can be run independent of the Eclipse IDE, the make infrastructure, and other tools.

ModusToolbox™ provides four reference flows for firmware development using PSoC™ 6 MCU and other Wi-Fi/Bluetooth® devices.

1. **FreeRTOS** development flow – FreeRTOS is an open source operating system for microcontrollers that makes small, low-power edge devices easy to program, deploy, secure, connect, and manage. FreeRTOS extends the FreeRTOS kernel, a popular open source operating system for microcontrollers, with software libraries that make it easy to securely connect small, low-power devices to AWS cloud services like **AWS IoT Core** or to more powerful edge devices running **AWS IoT Greengrass**.

To get started with firmware development on PSoC™ 6 MCU using Amazon FreeRTOS, visit [Getting started with FreeRTOS](#).

2. **Arm® Mbed OS** development flow – Arm® Mbed OS is a free, open-source embedded operating system designed specifically for the "things" in the Internet of Things. It includes all the features you need to develop a connected product based on an Arm® Cortex®-M microcontroller, including security, connectivity, an RTOS, and drivers for sensors and I/O devices.

To get started with firmware development on PSoC™ 6 using Mbed OS, refer to [Getting started with PSoC™ 6 MCU and CYW43xxx in Mbed OS](#).

3. **MCU and Bluetooth® SoC ecosystem** – This is the native development flow supported by the Eclipse IDE for ModusToolbox™ for PSoC™ 6 MCU. You can use it on the bare metal or with an RTOS of your choosing.

Development ecosystem

4. **AnyCloud** development flow – AnyCloud is a collection of libraries that is based on industry-standard lwIP TCP/IP stack and Mbed TLS network security. It enables rapid development of Wi-Fi and Bluetooth® applications on PSoC™ 6 MCU devices. AnyCloud enables development with custom or alternative third-party cloud management approaches with a fully open, customizable, and extensible source code distribution. AnyCloud includes Wi-Fi Connection Manager, Wi-Fi Middleware Core, Wi-Fi Host Driver, FreeRTOS kernel, RTOS Abstraction library, MQTT, OTA, LPA, Secure Sockets, and Bluetooth® Stack. To get started with AnyCloud on PSoC™ 6 MCU, refer to the [AnyCloud code examples](#) on the [Infineon GitHub](#) repo.

All the above development flows depend on the provided low level resources. These include:

- **Board support packages (BSP)** – A BSP is the layer of firmware containing board-specific drivers and other functions. The board support package is a set of libraries that provide APIs to initialize the board and provide access to board level peripherals. It includes low-level resources such as peripheral driver library (PDL) for PSoC™ 6 MCU and has macros for board peripherals. It uses the HAL to configure the board. Custom BSPs can be created to enable support for end-application boards. Refer to the Board support Packages section in [ModusToolbox™ user guide](#) for more information.
- **Hardware abstraction layer (HAL)** – The hardware abstraction layer (HAL) provides a high-level interface to configure and use hardware blocks on MCUs. It is a generic interface that can be used across multiple product families. The focus on ease-of-use and portability means the HAL does not expose all the low-level peripheral functionality. The HAL wraps the lower level drivers (like PSoC™ 6 PDL) and provides a high-level interface to the MCU. The interface is abstracted to work on any MCU. This helps you write application firmware independent of the target MCU.

The HAL can be combined with platform-specific libraries (such as PSoC™ 6 PDL) within a single application. You can leverage the HAL's simpler and more generic interface for most of an application, even if one portion requires finer-grained control.

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

The PDL reduces the need to understand register usage and bit structures, thus easing software development for the extensive set of peripherals in the PSoC™ 6 MCU series. You configure the driver for your application, and then use API calls to initialize and use the peripheral.

- **Extensive middleware libraries** that provides specific capabilities to an application. The [available middleware](#) spans across connectivity (Bluetooth®, AWS IoT, Bluetooth® LE, Secure Sockets) to PSoC™ 6-specific functionality (CAPSENSE™, USB, device firmware upgrade (DFU), emWin). All the middleware is delivered as libraries and via GitHub repositories.

2.2.3 PSoC™ 6 Software Resources

The PSoC™ 6 software includes driver and middleware configurators to get you started developing firmware with PSoC™ 6 MCU. 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 IoT devices and connectivity solutions. You may use any or all tools in any environment you prefer.

2.2.3.1 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

Development ecosystem

baud rate, parity, and 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. 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.
- **Bluetooth® LE configurator:** Configure the Bluetooth® LE settings.
- **EZ-PD™ configurator:** Configure parameters and select the features of the PDStack middleware.
- **Secure policy configurator:** Open, create or change policy configuration files for the “Secure” MCU devices.
- **SegLCD configurator:** Configure and generate the required structures for SegLCD driver.

Each of the above configurators create their own files (e.g.: *design.cycapsense* for CAPSENSE™). The configurator files (*design.modus* or *design.cycapsense*) are usually provided with the BSP. When an application is created based on a BSP, the files are copied into the application. You can also create custom device configurator files for an application and override the ones provided by the BSP. See [ModusToolbox™ help](#) for more details.

2.2.3.2 Library management for PSoC™ 6 MCU

With the release of ModusToolbox™ v2.2, there are now two flows for developing applications: MTB Flow and LIB Flow. The difference between the two flows is the way BSPs and libraries are structured.

MTB flow: This is the new flow for all new applications using MTB 2.2; it is recommended to use this flow for developing applications. Using this flow, applications can optionally share board support packages (BSPs) and libraries. If needed, different applications can use different versions of the same BSP/library. The file types associated with libraries using this flow have a *.mtb* extension.

Going further, Section 4 of this document describes creating a new application using the MTB Flow.

LIB flow: Here, the board support packages (BSPs) and libraries are structured such that each project has its own version of BSP and Libraries which cannot be shared with other applications. The file types associated with libraries using this flow have a *.lib* extension.

For more information on MTB and LIB Flow, see the [Library manager user guide](#) located at `<install_dir>/ModusToolbox/tools_<version>/library-manager/docs/library-manager.pdf`.

2.2.3.3 Software development for PSoC™ 6 MCU

Significant source code and tools are provided to enable software development for PSoC™ 6 MCUs. You use tools to specify how you want to configure the hardware, generate code for that purpose which you use in your

Development ecosystem

firmware, and include various middleware libraries for additional functionality, like Bluetooth® LE 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.

In the ModusToolbox™ environment, you use configurators to configure either the device, or a middleware library, like the Bluetooth® LE stack or CAPSENSE™ functionality.

The PSoC™ 6 peripheral driver library code is delivered as the **mtb-pdl-cat1** library. Middleware is delivered as separate libraries for each feature/function.

Whether you use Eclipse IDE, a third-party IDE, or the command line, firmware developers who wish to work at the register level should refer to 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 members of the PSoC™ 6 MCU 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 that peripheral.

As **Figure 3** shows, with the Eclipse IDE for ModusToolbox™ software, you can:

1. Choose a board support package (BSP).
2. Create a new application based on a list of starter applications, filtered by kit.
3. Add middleware.
4. Develop your application firmware using the HAL or PDL for PSoC™ 6 MCU.

Development ecosystem

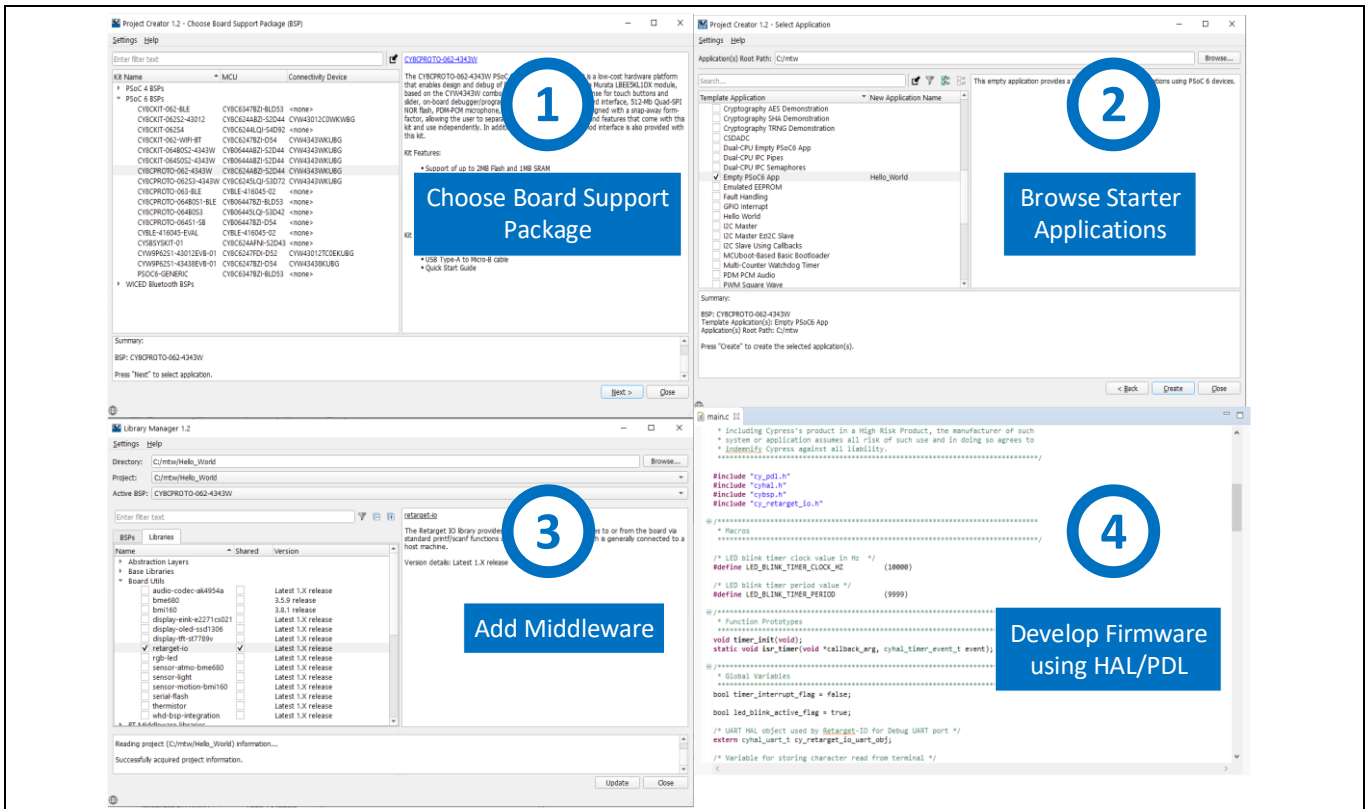


Figure 3 Eclipse IDE for ModusToolbox™ resources and middleware

2.2.4 ModusToolbox™ help

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

Choose **Help > ModusToolbox General Documentation:**

- **User Guide:** This guide primarily covers the ModusToolbox™ aspects of building, programming and debugging applications. It also covers various aspects of the tools installed along with the IDE
- **ModusToolbox™ Documentation Index:** Provides brief descriptions and links to various types of documentation included as part the ModusToolbox™ software.
- **ModusToolbox™ Installation Guide:** Provides instructions for installing the ModusToolbox™ software.
- **Release Notes**

For documentation on Eclipse IDE for ModusToolbox, choose **Help > Eclipse IDE for ModusToolbox Documentation:**

- **Quick Start Guide:** Provides you the basics for using Eclipse IDE for ModusToolbox
- **User Guide:** Provides descriptions about creating applications as well as building, programming, and debugging them using Eclipse IDE
- **Eclipse IDE for ModusToolbox Help:** Provides description on how to create new applications, update application code, change middleware settings, and program/debug applications
- **Eclipse IDE Tutorial Guide**

Development ecosystem

2.3 Support for other IDEs

You can develop firmware for PSoC™ 6 MCUs using your favorite IDE such as [IAR Embedded Workbench](#) or [Visual Studio Code](#).

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 Eclipse IDE. The device configurator and middleware configurators use the *design.x* files 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.x* files using the configurators and regenerate the code for the target IDE. It is recommended that you generate resource configurations using the configuration tools provided with ModusToolbox™ software.

See [AN225588 – Using ModusToolbox™ software with a third-party IDE](#) for details.

2.4 FreeRTOS support with ModusToolbox™

Adding native FreeRTOS support to a ModusToolbox™ application project is like adding any library/middleware. You can import the FreeRTOS middleware into your application by using the Library Manager. Select the application project and click the **Library Manager** link in the **Quick Panel**. Select **freertos** from the **Libraries > PSoC 6 Middleware** dialog, as [Figure 4](#) shows.

The *.mtb* file pointing to the FreeRTOS middleware is added to the application project. The middleware content is also downloaded and placed inside the corresponding folder called **freertos**. To continue working with FreeRTOS follow the steps in the Quick Start section of [FreeRTOS documentation](#).

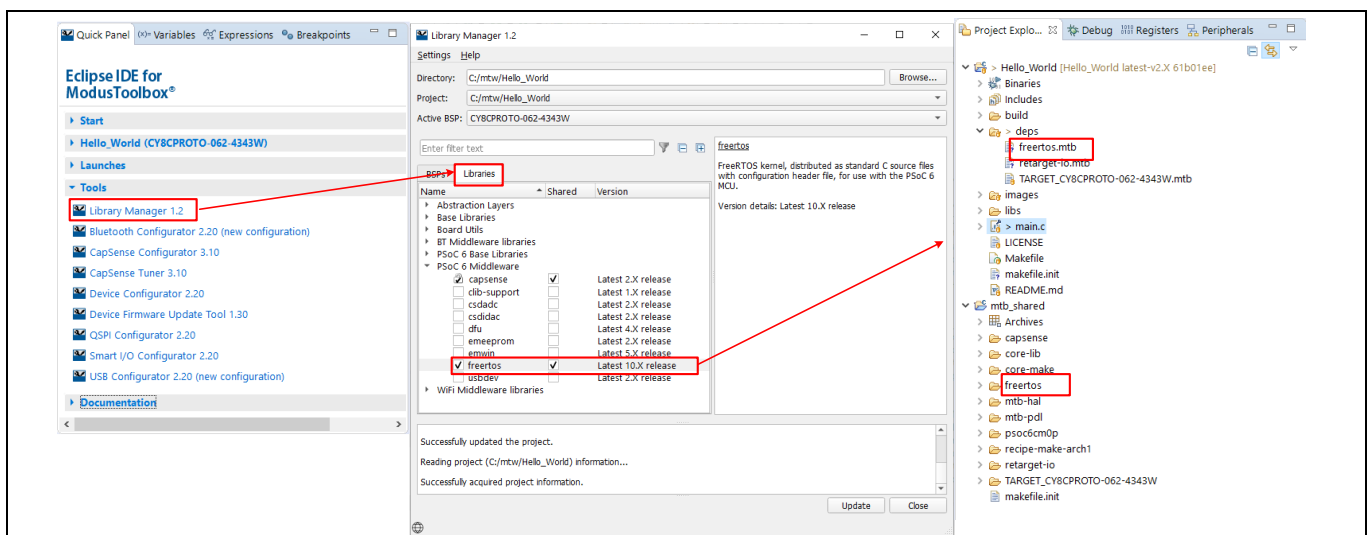


Figure 4 Import FreeRTOS middleware in ModusToolbox™ application

2.5 Programming/debugging

All PSoC™ 6 Kits have a KitProg3 onboard programmer/debugger. It supports Cortex® Microcontroller Software Interface Standard - Debug Access Port (CMSIS-DAP). See the [KitProg3 user guide](#) for details.

The Eclipse IDE requires KitProg3 and uses the [OpenOCD](#) protocol for debugging PSoC™ 6 MCU applications. It also supports GDB debugging using industry standard probes like the [Segger J-Link](#).

Development ecosystem

Note: The **PSoC™ 6 Wi-Fi-Bluetooth® pioneer kit (CY8CKIT-062-WiFi-BT)** and **PSoC™ 6 Bluetooth® LE pioneer kit (CY8CKIT-062-BLE)** have the KitProg2 onboard programmer/debugger. To work with Eclipse IDE, upgrade the kit to KitProg3.

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 Programming/Debugging - KitProg Firmware Loader** section in the Eclipse IDE for ModusToolbox™ user guide for more details.

For more information on debugging firmware on PSoC™ devices with ModusToolbox™, refer to *Program and Debug* section in the Eclipse IDE for ModusToolbox™ user guide.

2.6 PSoC™ 6 MCU development kits

Table 2 Development kits

Product line	Development kits
Performance	<p>PSoC™ 6 Wi-Fi-Bluetooth® pioneer kit (CY8CKIT-062-WiFi-BT)</p> <p>PSoC™ 6 Wi-Fi Bluetooth® prototyping kit (CY8CPROTO-062-4343W)</p> <p>PSoC™ 62S2 Wi-Fi Bluetooth® pioneer kit (CY8CKIT-062S2-43012)</p> <p>PSoC™ 62S3 Wi-Fi Bluetooth® prototyping kit (CY8CPROTO-062S3-4343W)</p> <p>PSoC™ 62S1 Wi-Fi Bluetooth® pioneer kit (CYW9P62S1-43438EVB-01)</p> <p>PSoC™ 62S1 Wi-Fi Bluetooth® pioneer kit (CYW9P62S1-43012EVB-01)</p> <p>PSoC™ 62S4 pioneer kit (CY8CKIT-062S4)</p>
Connectivity	<p>PSoC™ 6 Bluetooth® LE pioneer kit (CY8CKIT-062-BLE)</p> <p>PSoC™ 6 Bluetooth® LE prototyping kit (CY8CPROTO-063-BLE)</p>
Security	<p>PSoC™ 64 “Secure Boot” Wi-Fi Bluetooth® pioneer kit (CY8CKIT-064B0S2-4343W)</p> <p>PSoC™ 64 Standard “Secure” - AWS Wi-Fi Bluetooth® pioneer kit (CY8CKIT-064S0S2-4343W)</p>

For the complete list of kits for the PSoC™ 6 MCU along with the shield modules, see the [Microcontroller \(MCUs\) kits](#) page.

Device features

3 Device features

PSoC™ 6 MCU product lines have extensive feature sets as shown in [Figure 5](#). 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 [References](#).

- **MCU 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
 - DMA controllers
- **Security features**
 - Cryptography accelerators and true random number generator function
 - One-time programmable eFUSE for secure key storage
 - “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

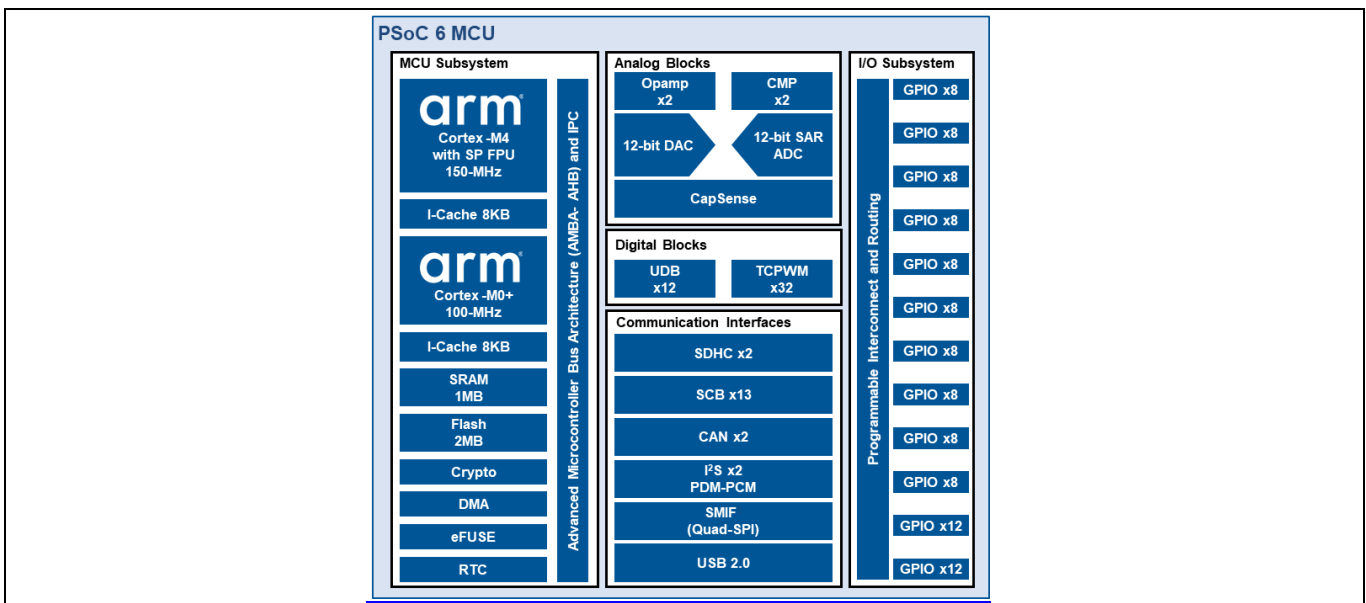


Figure 5 PSoC™ 6 MCU block diagram

- **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
 - Controller Area Network interface with Flexible Data-Rate
 - Up to two “Secure” Digital Host Controllers with support for SD, SDIO, and eMMC interfaces
 - Audio subsystem with up to two I2S interface and two PDM channels

Device features

- SMIF interface with support for execute-in-place from external quad SPI flash memory and on-the-fly encryption and decryption
- USB Full-Speed Device Interface
- **Programmable analog blocks**
 - Up to two opamps that can operate in system deep sleep mode
 - Up to two 12-bit SAR ADCs with maximum of 2-Msps sample rate and capability to function in system deep sleep mode in some of the PSoC™ 6 MCUs
 - One 12-bit, 500 ksps voltage-mode DAC
 - Up to two low-power comparators which can be used to wake up the device from all the low-power modes
 - 1.2-V bandgap reference with 1% tolerance for use with SAR ADCs and the DAC.
- **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), buck converter
 - Six power modes for fine-grained power management
 - An “Always ON” backup power domain with built-in RTC, power management integrated circuit (PMIC) control, and limited SRAM backup

My first PSoC™ 6 MCU design using Eclipse IDE for ModusToolbox™ software

4 My first PSoC™ 6 MCU design using Eclipse IDE for ModusToolbox™ software

This section does the following:

- Demonstrate how to build a simple PSoC™ 6 MCU-based design and program it on to the development kit.
- Make it easy to learn PSoC™ 6 MCU design techniques and how to use the Eclipse IDE for ModusToolbox™ software.

4.1 Prerequisites

Before you get started, make sure that you have the appropriate development kit for your PSoC™ 6 MCU product line, and have installed the required software. You also need internet access to the GitHub repositories during project creation.

4.1.1 Hardware

- The design is developed for [PSoC™ 6 Wi-Fi Bluetooth® prototyping kit \(CY8CPROTO-062-4343W\)](#). However, you can build the projects for other development kits. See the [Using these instructions](#) section.

4.1.2 Software

- [ModusToolbox™](#) 2.2 or above

After installing the software, refer to the [ModusToolbox™ user guide](#) to get an overview of the software.

4.2 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: View and modify the design](#)
- [Part 3: Write firmware](#)
- [Part 4: Build the application](#)
- [Part 5: Program the device](#)
- [Part 6: Test your design](#)

This design is developed for the [PSoC™ 6 Wi-Fi Bluetooth® prototyping kit \(CY8CPROTO-062-4343W\)](#). You can use other supported kits to test this example by selecting the appropriate kit while creating the application. The code described in the sections that follow has been tested on the following additional kits.

- [PSoC™ 6 Wi-Fi Bluetooth® pioneer kit \(CY8CKIT-062-WiFi-BT\)](#)
- [PSoC™ 6 Bluetooth® LE pioneer kit \(CY8CKIT-062-BLE\)](#)
- [PSoC™ 6 Bluetooth® LE prototyping kit \(CY8CPROTO-063-BLE\)](#)
- [PSoC™ 62S2 Wi-Fi Bluetooth® pioneer kit \(CY8CKIT-062S2-43012\)](#)
- [PSoC™ 62S1 Wi-Fi Bluetooth® pioneer kit \(CYW9P62S1-43438EVB-01\)](#)
- [PSoC™ 62S1 Wi-Fi Bluetooth® pioneer kit \(CYW9P62S1-43012EVB-01\)](#)
- [PSoC™ 62S3 Wi-Fi Bluetooth® prototyping kit \(CY8CPROTO-062S3-4343W\)](#)
- [PSoC™ 64 “Secure Boot” Wi-Fi Bluetooth® pioneer kit \(CY8CKIT-064B0S2-4343W\)](#)
- [PSoC™ 62S4 pioneer kit \(CY8CKIT-062S4\)](#)

My first PSoC™ 6 MCU design using Eclipse IDE for ModusToolbox™ software

4.3 About the design

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

At device reset, the Infineon-supplied pre-built CM0+ application image enables the CM4 CPU and configures the CM0+ CPU to go to sleep. The CM4 CPU uses the UART to print a “Hello World” message to the serial port stream, and starts blinking the user LED on the kit. When the user presses the enter key on the serial console, the blinking is paused or resumed.

4.4 Part 1: Create a new application

This section takes you on a step-by-step guided tour of the new application process. It uses the ‘**Empty PSoC6 App**’ starter application and guides you through the design development stages, and programming.

If you are familiar with developing projects with ModusToolbox™, you can use the ‘**Hello World**’ starter application directly. It is a complete design, with all the firmware written for the supported kits. 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.

Launch Eclipse IDE for ModusToolbox™ to get started. Please note that Eclipse IDE for ModusToolbox™ software needs access to the internet to successfully clone the starter application onto your machine.

1. Select a new workspace.

At launch, Eclipse IDE for 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 6** shows. Alternatively, you can type in a directory name to be used as the workspace directory along with the complete path, and Eclipse IDE will create the directory for you.

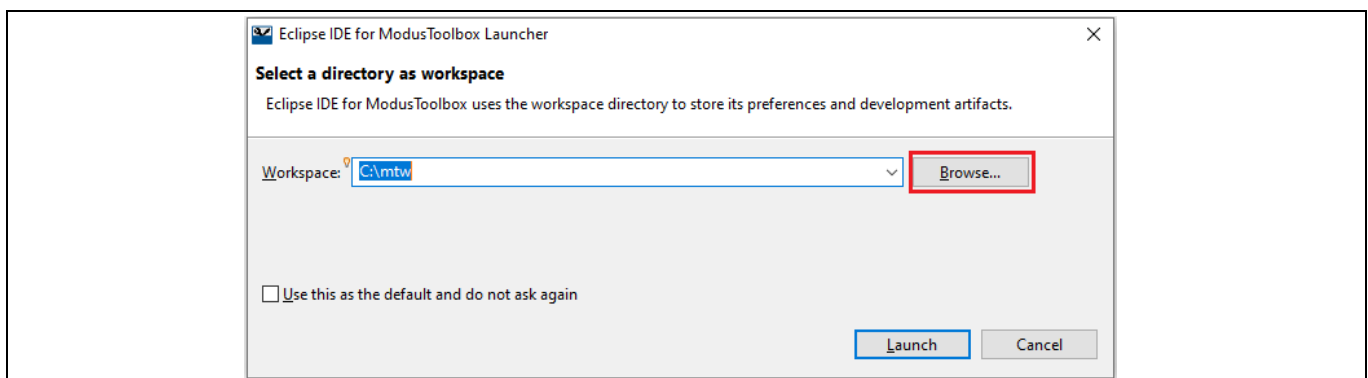


Figure 6 Select a directory as the workspace

2. Create a new ModusToolbox™ application.

- Click New Application in the Start group of the Quick Panel.
- Alternatively, you can choose File > New > ModusToolbox Application, as **Figure 7** shows. The Eclipse IDE for ModusToolbox Application window appears.

My first PSoC™ 6 MCU design using Eclipse IDE for ModusToolbox™ software

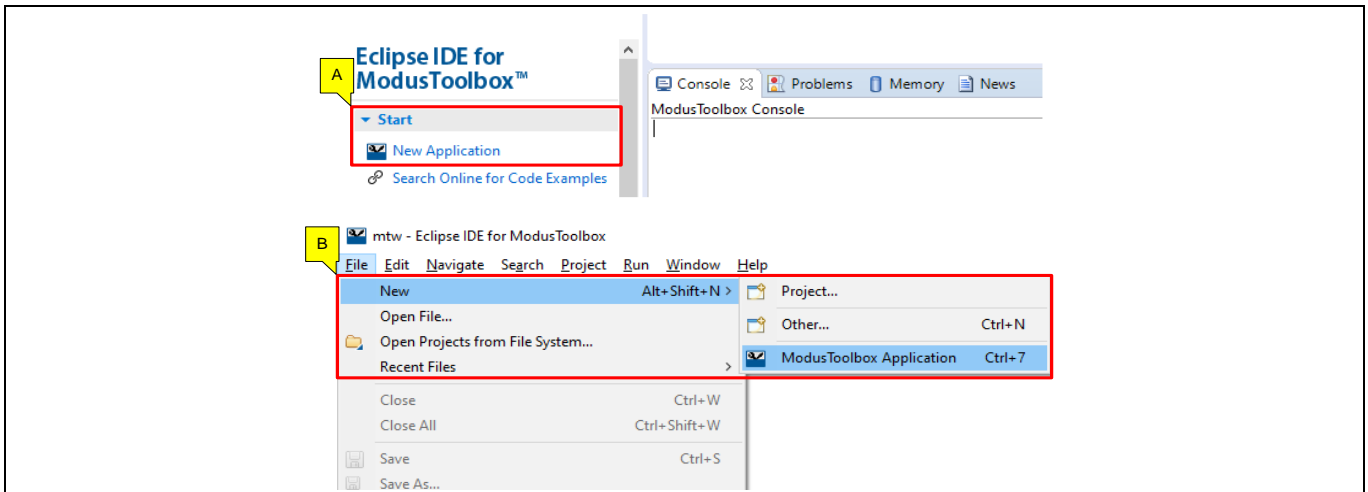


Figure 7 Create a New ModusToolbox Application

3. Select a target PSoC™ 6 development kit.

ModusToolbox™ speeds up the development process by providing BSPs that set various workspace/project options for the specified development kit in the new application dialog.

- In the **Choose Board Support Package (BSP)** dialog, choose the **Kit Name** that you have. The steps that follow use **CY8CPROTO-062-4343W**. See **Figure 8** for help with this step.
- Click **Next**.

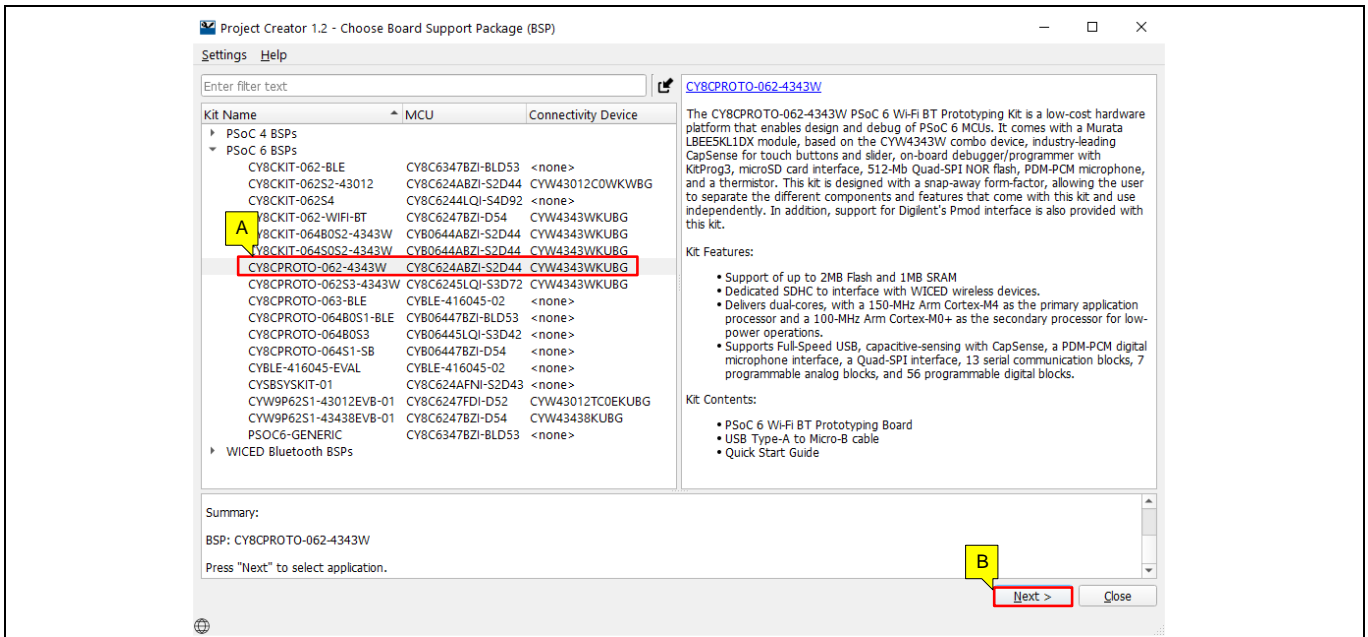


Figure 8 Choose target hardware

- In the **Starter Application** dialog, select **Empty PSoC6 App** starter application, as **Figure 9** shows.
- In the **Name** field, type in a name for the application, such as **Hello_World**. You can choose to leave the default name if you prefer.
- Click **Create** to create the application, as **Figure 9** shows, wait for the Project Creator to automatically close once the project is successfully created.

My first PSoC™ 6 MCU design using Eclipse IDE for ModusToolbox™ software

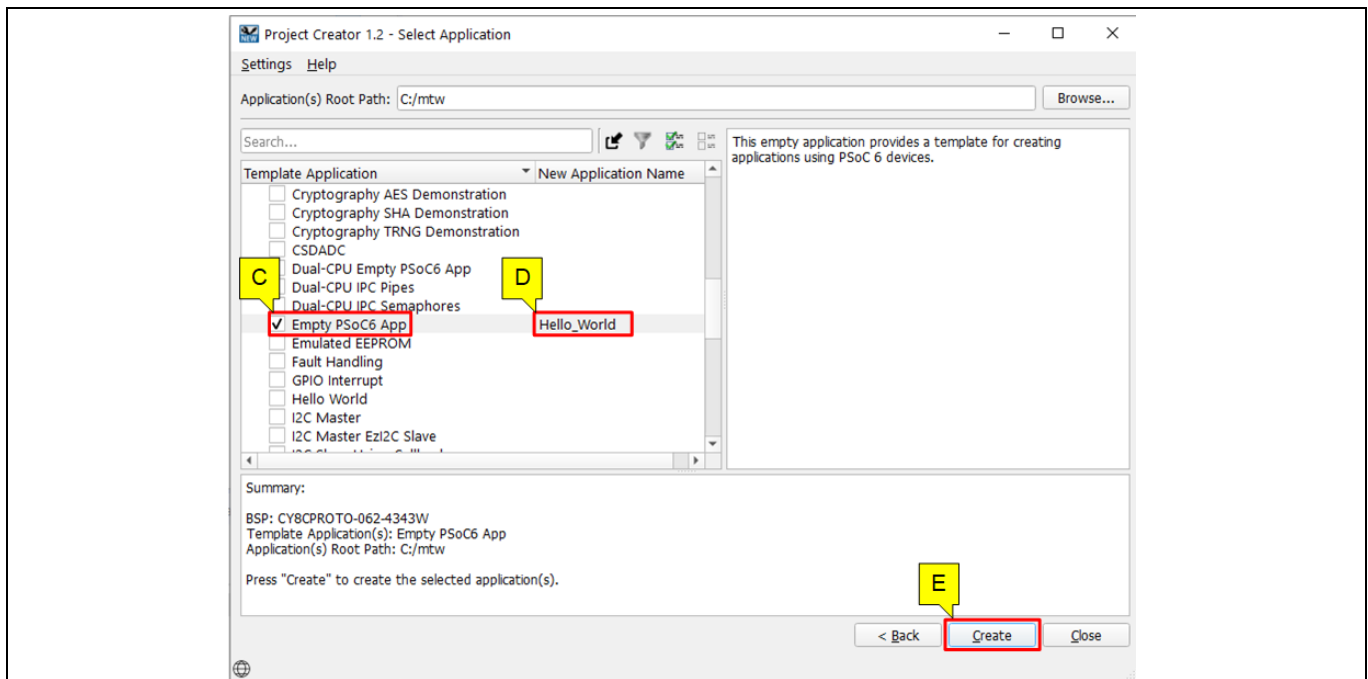


Figure 9 Choose Starter Application

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

The BSP uses CY8C624ABZI-D54 as the default device that is mounted on the **PSoC™ 6 Wi-Fi-Bluetooth® prototyping kit (CY8CPROTO-062-4343W)** along with the CYW4343WKUBG Wi-Fi/Bluetooth® radio.

If you are using custom hardware based on PSoC™ 6 MCU, or a different PSoC™ 6 MCU part number, please refer to the **Creating your Own BSP** section in the **ModusToolbox™ user guide**. The guide is also available under *ide_2.2>docs* folder of the ModusToolbox™ installation directory.

4.5 Part 2: View and modify the design

Figure 10 shows the ModusToolbox™ project explorer interface displaying the structure of the application project.

In the Eclipse IDE for ModusToolbox™ software, a PSoC™ 6 MCU application™ consists of a project to develop code for the CM4 CPU. A project folder consists of various subfolders – each denoting a specific aspect of the project.

- An application project contains a Makefile which is typically at the root folder. It has instructions on how to recreate the project. This file also contains the set of directives that the make tool uses to compile and link the application project. There can be more than one project in an application, and each dependent project usually resides within its own folder within the application folder and contains its own Makefile.
- The *build* folder contains all the artifacts resulting from the make build of the project. The output files are organized by target BSPs.

My first PSoC™ 6 MCU design using Eclipse IDE for ModusToolbox™ software

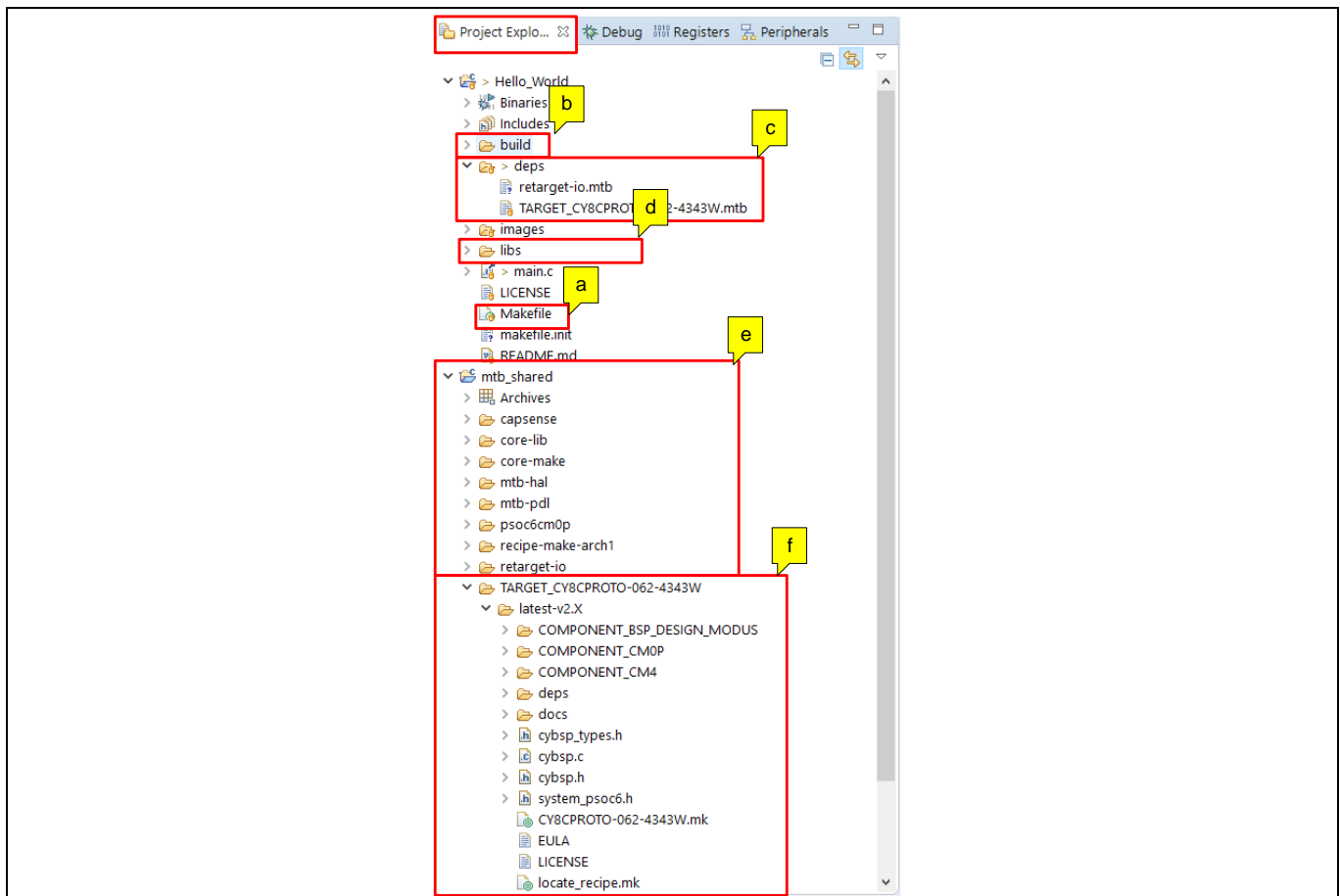


Figure 10 Project Explorer view

- c) The *deps* folder contains *.mtb* files, which provide the location from which ModusToolbox™ pulls the BSP/library that is directly referenced by the application. These files typically contain the GitHub location of the entire library. The *.mtb* files also contains a git Commit Hash or Tag that tells which version of the library is to be fetched and a path as to where the library should be stored.

For example, the *TARGET_CY8CPROTO-062-4343W.mtb* file points to

[https://github.com/cypresssemiconductorco/TARGET_CY8CKIT-062S2-43012/#latestv1.X#\\$\\$ASSET_REPO\\$\\$/TARGET_CY8CKIT-062S2-43012/latest-v1.X](https://github.com/cypresssemiconductorco/TARGET_CY8CKIT-062S2-43012/#latestv1.X#$$ASSET_REPO$$/TARGET_CY8CKIT-062S2-43012/latest-v1.X). The *latest-v1.X* tag in the link denotes the specific release of the BSP. The variable *\$\$ASSET_REPO\$\$* points to the root of the shared location. If the library has to be local to the application instead of shared, use *\$\$LOCAL\$\$* instead of *\$\$ASSET_REPO\$\$*.

Similarly, *retarget-io.mtb* points to [https://github.com/cypresssemiconductorco/retarget-io#latest-v1.X#\\$\\$ASSET_REPO\\$\\$/retarget-io/latest-v1.X](https://github.com/cypresssemiconductorco/retarget-io#latest-v1.X#$$ASSET_REPO$$/retarget-io/latest-v1.X).

- d) The *libs* folder also contains *.mtb* files, these point to libraries that are included indirectly as a dependency of a BSP or another library. For each indirect dependency, the Library Manager places an *.mtb* file in this folder. These files have been populated based on the targets available in *deps* folder.

For example, using BSP lib file *TARGET_CY8CPROTO-062-4343W.mtb* populates *libs* folder with the following *.mtb* files: [capsense.mtb](#), [core-lib.mtb](#), [core-make.mtb](#), [mtb-hal-cat1.mtb](#), [mtb-pdl-cat1.mtb](#), [psoc6cm0p.mtb](#), [recipe-make-cat1a.mtb](#).

The *libs* folder contains *mtb.mk* file, which stores the relative paths of the all the libraries required by the application. The build system uses this file to find all the libraries required by the application.

My first PSoC™ 6 MCU design using Eclipse IDE for ModusToolbox™ software

e) By default, when creating a new application or adding a BSP/library to an existing application and specifying it as shared, all BSPs/libraries are placed in an *mtb_shared* directory adjacent to the application directory(ies).

The *mtb_shared* folder is shared between different applications that use the same versions of BSP/library.

f) The files provided by the BSP are listed under the *TARGET_x* folder. All the configuration files generated by the device and peripheral configurators are included in the *GeneratedSource* folder of the BSP and are prefixed with *cycfg_*. These files contain the design configuration as defined by the BSP. You can view and modify the design configuration by clicking the Device Configurator link in the Quick Panel. However, note that if you upgrade the BSP library to a newer version, the manual edits done to the *design.x* files are lost. You can also create custom device configurator files for an application and override the ones provided by the BSP. See **Modifying the BSP Configuration for a Single Application** section in the **ModusToolbox™ user guide** for more details.

The BSP folder also contains the linker scripts and the startup code for the PSoC™ 6 MCU device used on the board.

Because the *mtb_shared* folder is shared between different applications that use the same versions of BSP/library, any changes made to the BSP for one application will be reflected in the other application. If multiple applications use the same BSP, you should create a custom BSP for each application. For more information on creating custom BSP for an application, see the “Modifying the BSP Configuration for a Single Application” section in the **ModusToolbox™ user guide**.

Of interest are the configuration files that are in the *COMPONENT_BSP_x* folder. Click on the **Device Configurator** link in the **Quick Panel**. **Figure 11** shows the resulting window called the **Device Configurator** window. You can also double-click open the other *design.x* files to open them in their respective configurators or click the corresponding links in the **Quick Panel**.

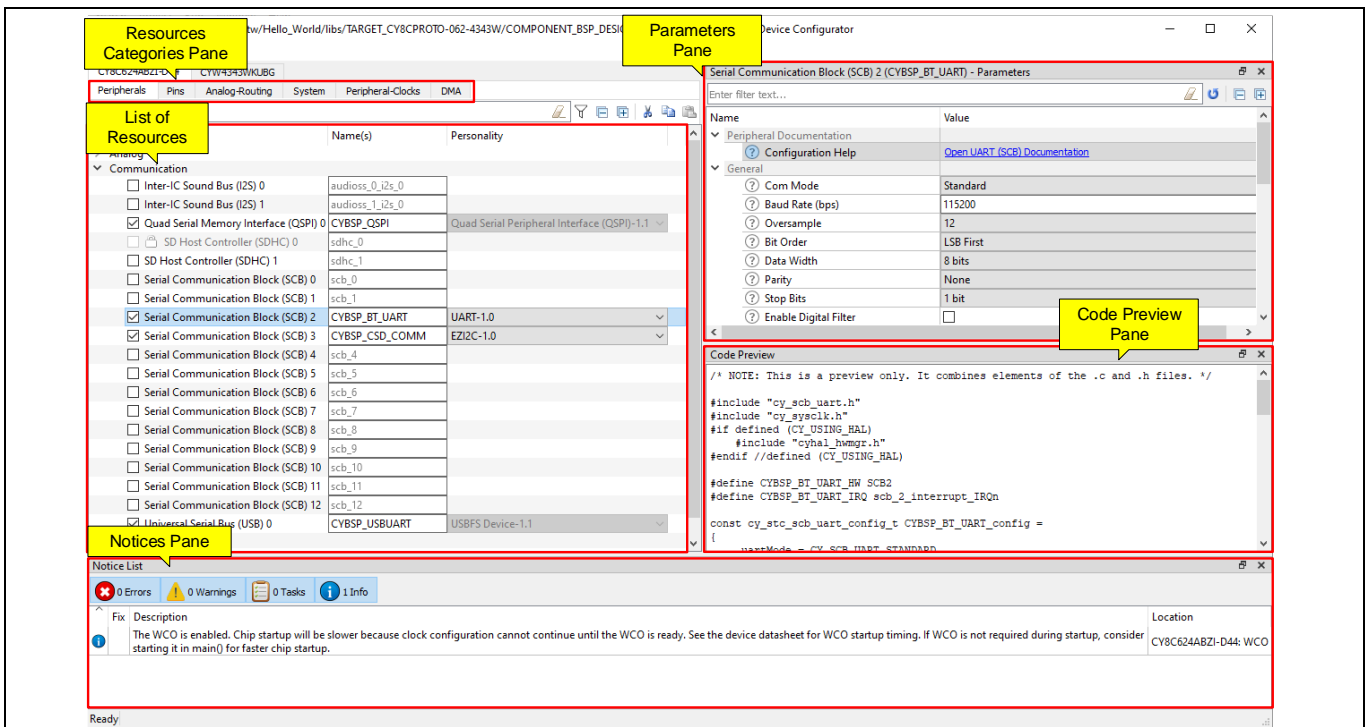


Figure 11 design.modus overview

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

My first PSoC™ 6 MCU design using Eclipse IDE for ModusToolbox™ software

You can choose how a resource behaves by choosing a **Personality** for the resource. For example, a **Serial Communication Block (SCB)** resource can have a **EZ12C, I2C, SPI** or **UART** personalities. The **Alias** is your name for the resource, which is used in firmware development. One or more aliases can be specified by using a comma to separate them (with no spaces)

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 GeneratedSource folder. Any errors, warnings, and information messages arising out of the configuration are displayed in the **Notices** pane.

The application project contains relevant files that help you create an application for the CM4 CPU (*main.c*), while the CM0+ application is supplied as a C file (*psoc6_02_cm0p_sleep.c* for CY8C624ABZI-D44 device). See the [psoc6cm0p](#) library. This C file is compiled and linked with the CM4 image as part of the normal build process.

At this point in the development process, we are ready to add the required middleware to the design. The only middleware required for the Hello World application is the [retarget-io](#) library.

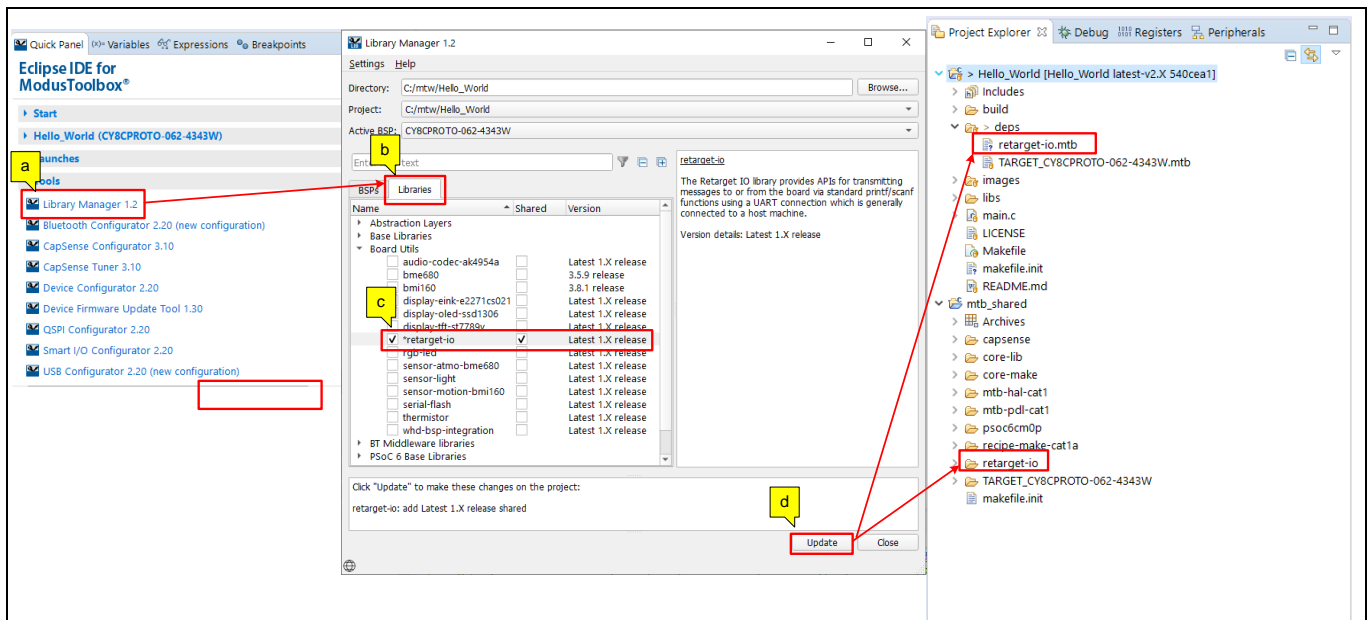


Figure 12 Add the [retarget-io](#) middleware

1. Add [retarget-io](#) middleware.

In this step, you will add the [retarget-io](#) middleware to redirect standard input and output streams to the UART configured by the BSP. The initialization of the middleware will be done in *main.c* code.

- In the **Quick Panel**, click on the **Library Manager** link.
- In the subsequent dialog, select the **Libraries** tab.
- Under **Board Utils**, select and enable [retarget-io](#).
- Click **Update**.

The files necessary to use the [retarget-io](#) middleware are added in the `mtb_shared > retarget_io` folder, and the *.mtb* file is also added to the *deps* folder, as [Figure 12](#) shows.

2. Configuration of UART, timer peripherals, pins and system clocks

The configuration of the debug UART peripheral, timer peripheral, pins and system clocks can be done directly in the code using the function APIs provided by BSP and HAL. See [Part 3: Write firmware](#).

My first PSoC™ 6 MCU design using Eclipse IDE for ModusToolbox™ software

4.6 Part 3: Write firmware

At this point in the development process, you have created an application, with the assistance of an application template and modified it to add the **retarget-io** middleware. In this part, you write the firmware that implements the design functionality.

If you are working from scratch using the Empty PSoC™ 6 starter application, you can copy the respective source code to the *main.c* of the application project from the code snippet provided in this section. If you are using the Hello World code example, all the required files are already in the application.

Firmware flow

We now examine the code in the *main.c* file of the application. **Figure 13** shows the firmware flowchart.

The CM0+ CPU comes out of reset and enables the CM4 CPU. The CM0+ CPU is then configured to go to sleep by the provided CM0+ application. Resource initialization for this example is performed by the CM4 CPU. It configures the system clocks, pins, clock to peripheral connections, and other platform resources.

When the CM4 CPU is enabled, the clocks and system resources are initialized by the BSP initialization function. The **retarget-io** middleware is configured to use the debug UART, and the user LED is initialized. The debug UART prints a “Hello World!” message on the terminal emulator – the on-board KitProg3 acts the USB-UART bridge to create the virtual COM port. A timer object is configured to generate an interrupt every 1000 milliseconds. At each Timer interrupt, the CM4 CPU toggles the LED state on the kit.

The firmware is designed to accept ‘Enter’ key as an input and on every press of the ‘Enter’ key the firmware starts or stops the blinking of the LED.

Note that the application code uses BSP/HAL/middleware functions to execute the intended functionality.

`cybsp_init()` - This BSP function sets up the HAL hardware manager and initializes all the system resources of the device including but not limited to the system clocks and power regulators.

`cy_retarget_io_init()` - This function from **retarget-io** middleware uses the aliases set up for debug UART pins to configure the debug UART with a standard baud rate of 115200 and also redirects the input/output stream to the debug UART.

`cyhal_gpio_init()` - This function from the gpio HAL initializes the physical pin to drive the LED. The LED used is derived from the BSP definition.

`timer_init()` - This function wraps a set of timer HAL function calls to instantiate and configure a timer. It also sets up a callback for the timer interrupt.

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

Code Listing 1

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

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

/* LED blink timer clock value in Hz */
#define LED_BLINK_TIMER_CLOCK_HZ (10000)
```


My first PSoC™ 6 MCU design using Eclipse IDE for ModusToolbox™ software

Code Listing 1

```

/* LED blink timer period value */
#define LED_BLINK_TIMER_PERIOD                (9999)

/*****
 * Function Prototypes
 *****/
void timer_init(void);
static void isr_timer(void *callback_arg, cyhal_timer_event_t event);

/*****
 * Global Variables
 *****/
bool timer_interrupt_flag = false;
bool led_blink_active_flag = true;

/* Variable for storing character read from terminal */
uint8_t uart_read_value;

/* Timer object used for blinking the LED */
cyhal_timer_t led_blink_timer;

/*****
 * Function Name: main
 *****/
* Summary:
* This is the main function for CM4 CPU. It sets up a timer to trigger a
* periodic interrupt. The main while loop checks for the status of a flag set
* by the interrupt and toggles an LED at 1Hz to create an LED blink. The
* while loop also checks whether the 'Enter' key was pressed and
* stops/restarts LED blinking.
*
* Parameters:
* none
*
* Return:
* int
*****/
int main(void)
{
    cy_rslt_t result;

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

    /* Board init failed. Stop program execution */
    if (result != CY_RSLT_SUCCESS)
    {
        CY_ASSERT(0);
    }

    /* Enable global interrupts */
    __enable_irq();

    /* Initialize retarget-io to use the debug UART port */

```

My first PSoC™ 6 MCU design using Eclipse IDE for ModusToolbox™ software

Code Listing 1

```

result = cy_retarget_io_init(CYBSP_DEBUG_UART_TX, CYBSP_DEBUG_UART_RX,
                            CY_RETARGET_IO_BAUDRATE);

/* retarget-io init failed. Stop program execution */
if (result != CY_RSLT_SUCCESS)
{
    CY_ASSERT(0);
}

/* Initialize the User LED */
result = cyhal_gpio_init(CYBSP_USER_LED, CYHAL_GPIO_DIR_OUTPUT,
                        CYHAL_GPIO_DRIVE_STRONG, CYBSP_LED_STATE_OFF);

/* GPIO init failed. Stop program execution */
if (result != CY_RSLT_SUCCESS)
{
    CY_ASSERT(0);
}

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

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

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

printf("For more PSoC 6 MCU projects, "
       "visit our code examples repositories:\r\n\n");

printf("https://github.com/Infineon/"
       "Code-Examples-for-ModusToolbox-Software\r\n\n");

/* Initialize timer to toggle the LED */
timer_init();

printf("Press 'Enter' key to pause or "
       "resume blinking the user LED \r\n\r\n");

for (;;)
{
    /* Check if 'Enter' key was pressed */
    if (cyhal_uart_getc(&cy_retarget_io_uart_obj, &uart_read_value, 1)
        == CY_RSLT_SUCCESS)
    {
        if (uart_read_value == '\r')
        {
            /* Pause LED blinking by stopping the timer */
            if (led_blink_active_flag)
            {
                cyhal_timer_stop(&led_blink_timer);

                printf("LED blinking paused \r\n");
            }
            else /* Resume LED blinking by starting the timer */
            {
                cyhal_timer_start(&led_blink_timer);
            }
        }
    }
}

```

My first PSoC™ 6 MCU design using Eclipse IDE for ModusToolbox™ software

Code Listing 1

```

        printf("LED blinking resumed\r\n");
    }

    /* Move cursor to previous line */
    printf("\x1b[1F");

    led_blink_active_flag ^= 1;
}

/* Check if timer elapsed (interrupt fired) and toggle the LED */
if (timer_interrupt_flag)
{
    /* Clear the flag */
    timer_interrupt_flag = false;

    /* Invert the USER LED state */
    cyhal_gpio_toggle(CYBSP_USER_LED);
}
}

/*****
* Function Name: timer_init
*****/
* Summary:
* This function creates and configures a Timer object. The timer ticks
* continuously and produces a periodic interrupt on every terminal count
* event. The period is defined by the 'period' and 'compare_value' of the
* timer configuration structure 'led_blink_timer_cfg'. Without any changes,
* this application is designed to produce an interrupt every 1 second.
*
* Parameters:
* none
*
*****/
void timer_init(void)
{
    cy_rslt_t result;

    const cyhal_timer_cfg_t led_blink_timer_cfg =
    {
        .compare_value = 0,           /* Timer compare value, not used */
        .period = LED_BLINK_TIMER_PERIOD, /* Defines the timer period */
        .direction = CYHAL_TIMER_DIR_UP, /* Timer counts up */
        .is_compare = false,         /* Don't use compare mode */
        .is_continuous = true,       /* Run timer indefinitely */
        .value = 0                   /* Initial value of counter */
    };

    /* Initialize the timer object. Does not use input pin ('pin' is NC) and
    * does not use a pre-configured clock source ('clk' is NULL). */
    result = cyhal_timer_init(&led_blink_timer, NC, NULL);

    /* timer_init failed. Stop program execution */
    if (result != CY_RSLT_SUCCESS)
    {
        CY_ASSERT(0);
    }
}

```

My first PSoC™ 6 MCU design using Eclipse IDE for ModusToolbox™ software

Code Listing 1

```

}

/* Configure timer period and operation mode such as count direction,
duration */
cyhal_timer_configure(&led_blink_timer, &led_blink_timer_cfg);

/* Set the frequency of timer's clock source */
cyhal_timer_set_frequency(&led_blink_timer, LED_BLINK_TIMER_CLOCK_HZ);

/* Assign the ISR to execute on timer interrupt */
cyhal_timer_register_callback(&led_blink_timer, isr_timer, NULL);

/* Set the event on which timer interrupt occurs and enable it */
cyhal_timer_enable_event(&led_blink_timer, CYHAL_TIMER_IRQ_TERMINAL_COUNT,
                        7, true);

/* Start the timer with the configured settings */
cyhal_timer_start(&led_blink_timer);
}

/*****
* Function Name: isr_timer
*****/
* Summary:
* This is the interrupt handler function for the timer interrupt.
*
* Parameters:
*   callback_arg   Arguments passed to the interrupt callback
*   event          Timer/counter interrupt triggers
*
*****/
static void isr_timer(void *callback_arg, cyhal_timer_event_t event)
{
    (void) callback_arg;
    (void) event;

    /* Set the interrupt flag and process it from the main while(1) loop */
    timer_interrupt_flag = true;
}

```

My first PSoC™ 6 MCU design using Eclipse IDE for ModusToolbox™ software

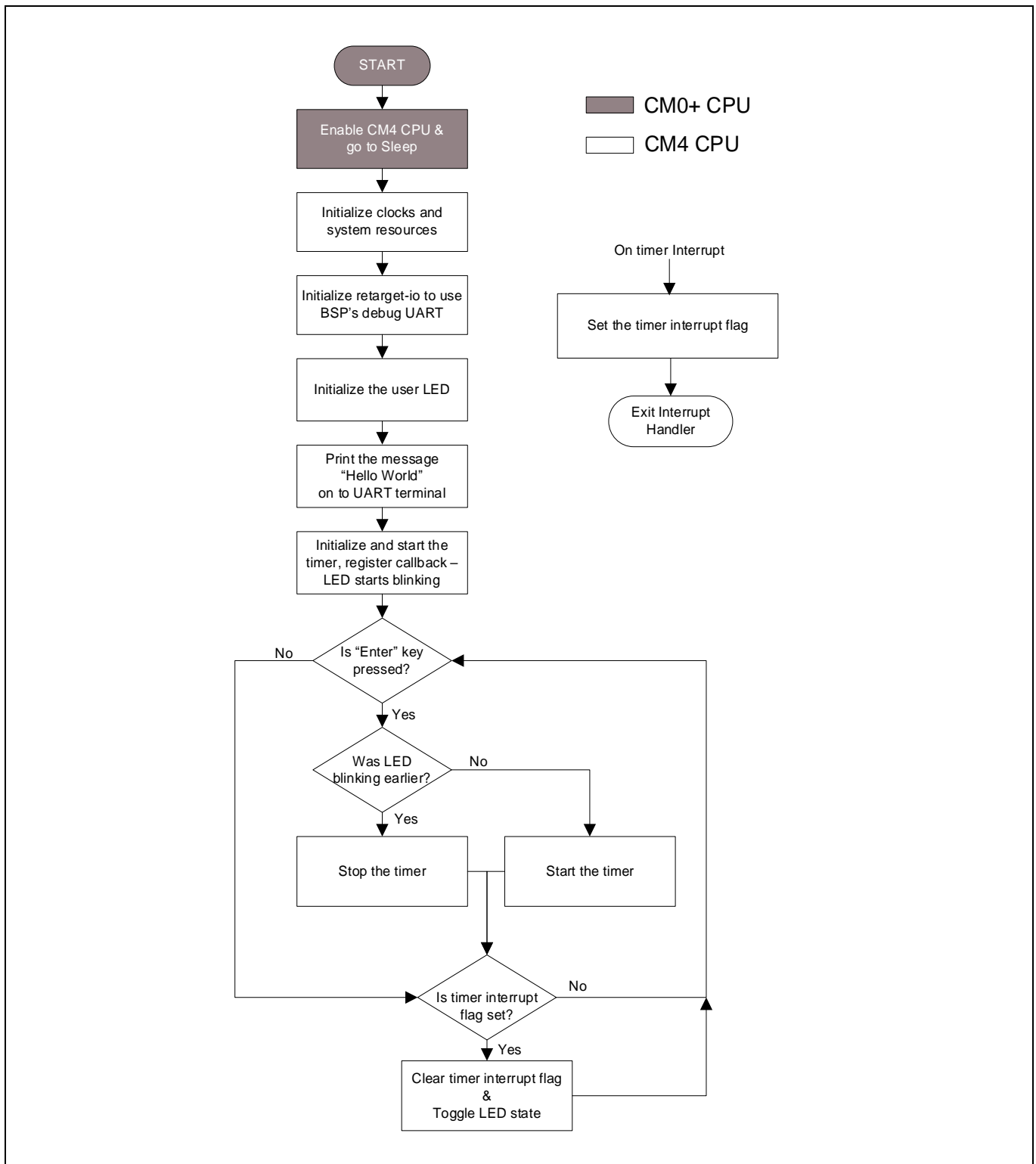


Figure 13 Firmware flowchart

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

My first PSoC™ 6 MCU design using Eclipse IDE for ModusToolbox™ software

4.7 Part 4: Build the application

This section shows how to build the application.

1. Build the application.

- Select the application project in the Project Explorer window and click on the **Build <name> Application** shortcut under the <name> group in the Quick Panel. It selects the **Debug** build configuration and compiles/links all projects that constitute the application.
- The **Console** view lists the results of the build operation, as **Figure 14** shows.

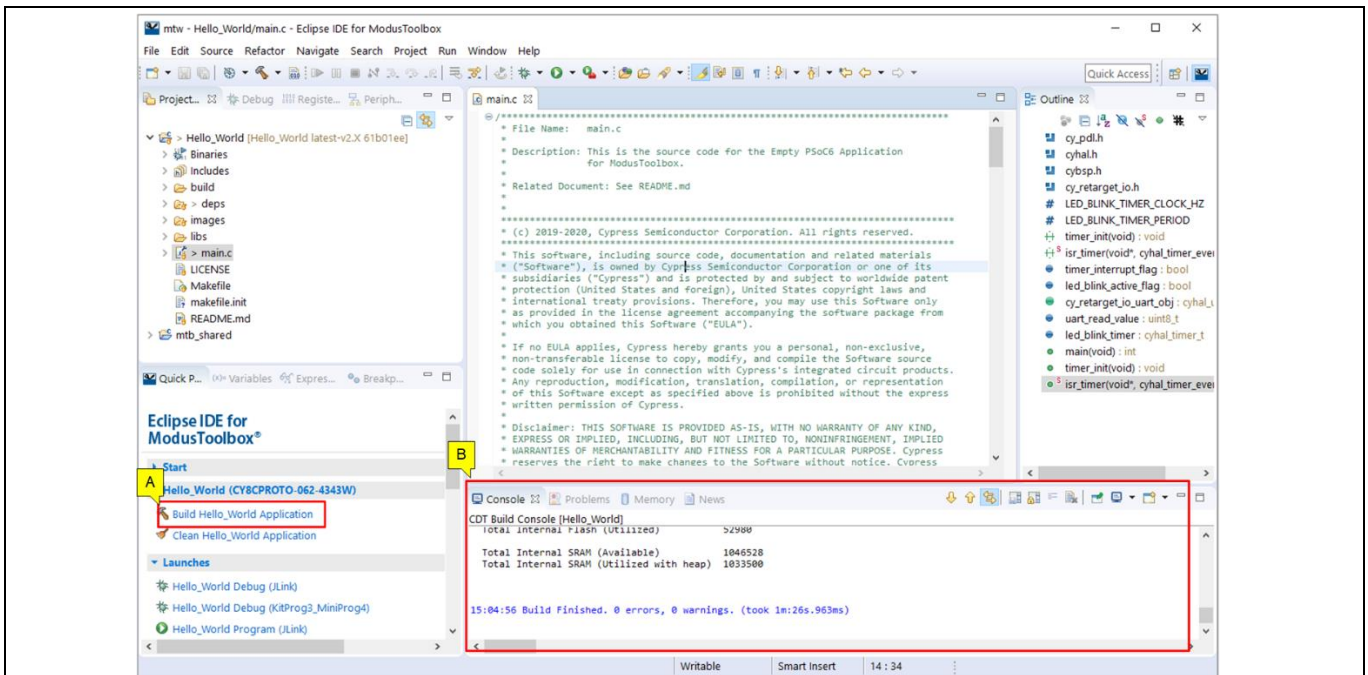


Figure 14 Build the application

If you encounter errors, revisit prior steps to ensure that you accomplished all the required tasks.

Note: You can also use the command line interface (CLI) to build the application. Please refer to **Using the Command Line** section in the **ModusToolbox™ user guide**. This document is located in the `/ide_<version>/docs/` folder in the ModusToolbox™ installation.

4.8 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. Some kits are shipped with KitProg2 firmware instead of KitProg3. See **Programming/debugging** for details. ModusToolbox™ includes the **fw-loader** command-line tool to switch the KitProg firmware from KitProg2 to KitProg3. Refer to the PSoC™ 6 MCU **KitProg Firmware Loader** section in the **Eclipse IDE for ModusToolbox™ user guide** for more details.

If you are using a development kit with a built-in programmer (the CY8CPROTO-062-4343W, for example), connect the board to your computer using the USB cable.

My first PSoC™ 6 MCU design using Eclipse IDE for ModusToolbox™ software

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

1. Program the application.

- a) Connect to the board and perform the following step.
- b) Select the application project and click on the **<application name> Program (KitProg3_MiniProg4)** shortcut under the **Launches** group in the Quick Panel, as **Figure 15** shows. The IDE will select and run the appropriate run configuration. Note that this step will also perform a build if any files have been modified since the last build.



Figure 15 Programming an application to a device

The **Console** view lists the results of the programming operation, as **Figure 16** shows.



Figure 16 Console – programming results

My first PSoC™ 6 MCU design using Eclipse IDE for ModusToolbox™ software

4.9 Part 6: Test your design

This section describes how to test your design.

Follow the steps below to observe the output of your design. This note uses 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 **Figure 17** shows. Note that your COM port number may be different.

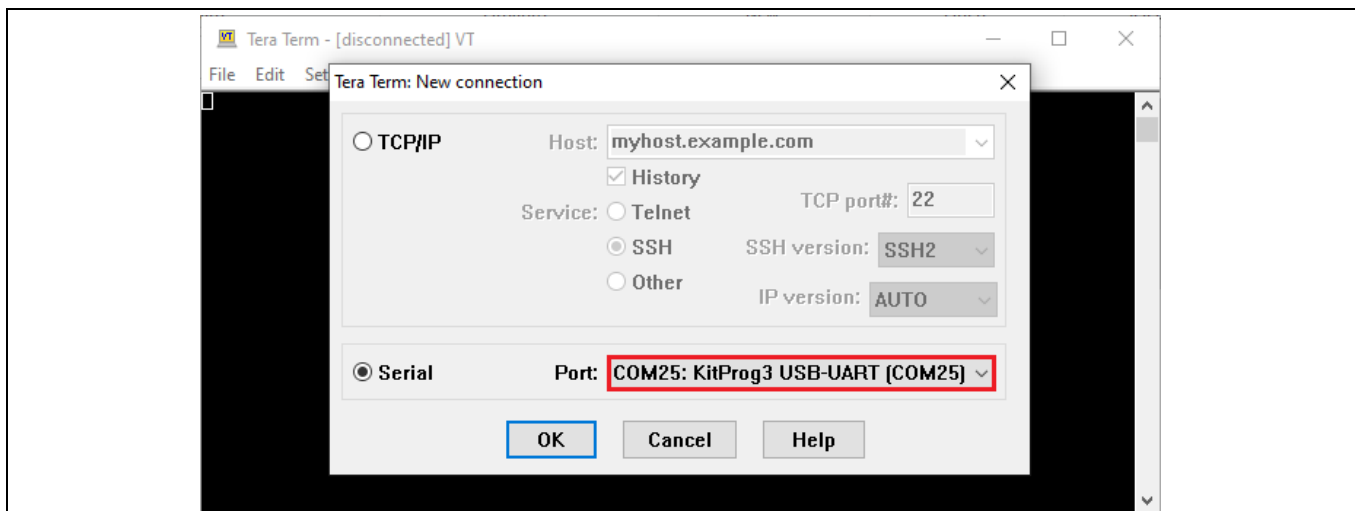


Figure 17 Selecting the KitProg3 COM port in Tera Term

2. Set the baud rate.

Set the baud rate to 115200 under **Setup > Serial port** as **Figure 18** shows.

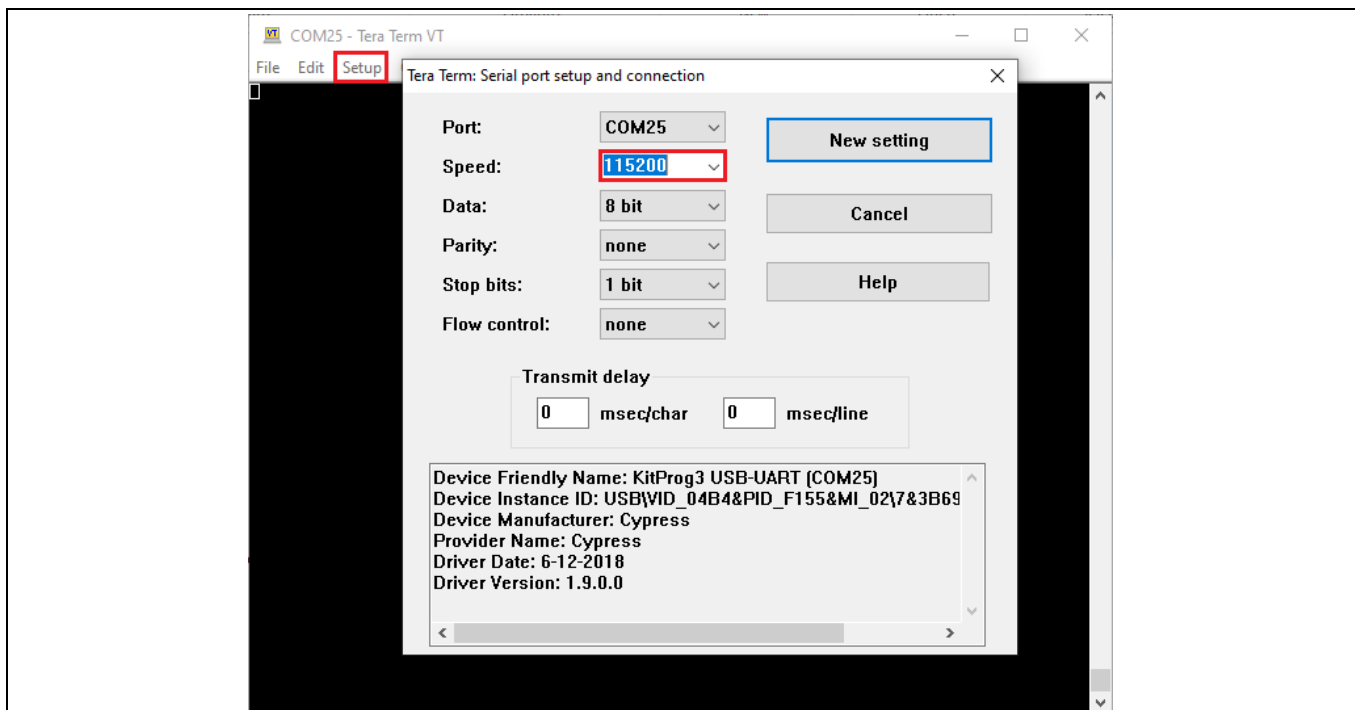


Figure 18 Configuring the baud rate in Tera Term

My first PSoC™ 6 MCU design using Eclipse IDE for ModusToolbox™ software

3. Reset the device.

Press the reset switch (**SW1**) on the kit. A message appears on the terminal as **Figure 19** shows. The user LED on the kit will start blinking.

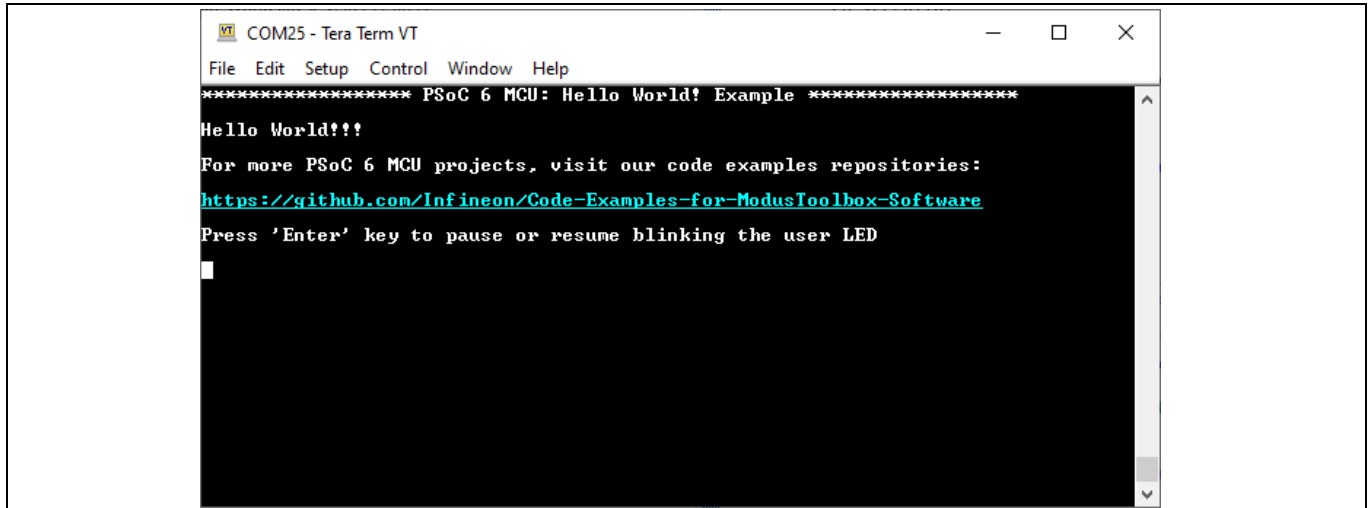


Figure 19 UART message printed from CM4 CPU

4. Pause/resume LED blinking functionality.

Press the **Enter** Key to pause/resume blinking the LED. When the LED blinking is paused, a corresponding message will be displayed on the terminal as **Figure 20** shows.

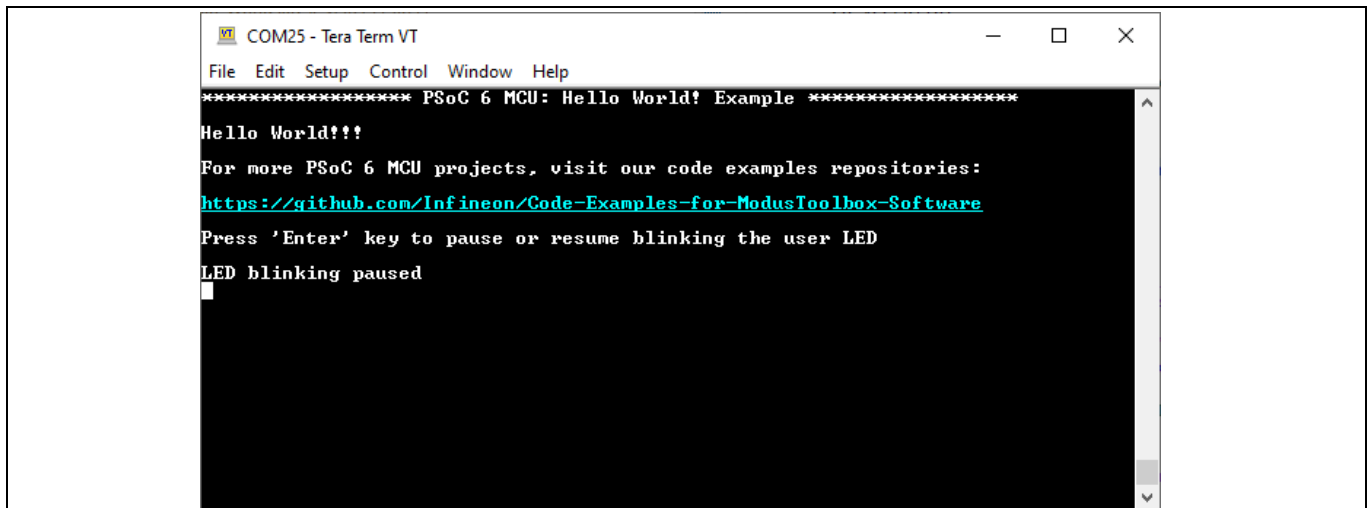


Figure 20 UART message from CM4 CPU

Summary

5 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.

References
References

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

Table 3 lists the system-level and general application notes that are recommended for the next steps in learning about PSoC™ 6 MCU and ModusToolbox™.

Table 3 General and system-level application notes

Document	Document name
AN221774	Getting started with PSoC™ 6 MCU on PSoC™ Creator
AN210781	Getting started with PSoC™ 6 MCU with Bluetooth® Low Energy (BLE) connectivity on PSoC™ Creator
AN218241	PSoC™ 6 MCU hardware design considerations
AN225588	Using ModusToolbox™ software with a third-party IDE
AN219528	PSoC™ 6 MCU low-power modes and power reduction techniques

Table 4 lists the application notes (AN) for specific peripherals and applications.

Table 4 Documents related to PSoC™ 6 MCU features

Document	Document name
System resources, CPU, and interrupts	
AN215656	PSoC™ 6 MCU dual-CPU system design
AN217666	PSoC™ 6 MCU interrupts
CAPSENSE™	
AN92239	Proximity sensing with CAPSENSE™
AN85951	PSoC™ 4 and PSoC™ 6 MCU CAPSENSE™ design guide
Device Firmware Update	
AN213924	PSoC™ 6 MCU device firmware update software development kit guide
Low-power analog	
AN230938	PSoC™ 6 MCU low-power analog

Glossary

Glossary

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

- **Board support package (BSP):** A BSP is the layer of firmware containing board-specific drivers and other functions. The board support package is a set of libraries that provide firmware APIs to initialize the board and provide access to board level peripherals.
- **Cypress Programmer:** Cypress Programmer is a flexible, cross-platform application for programming Cypress devices. It can Program, Erase, Verify, and Read the flash of the target device.
- **Hardware abstraction layer (HAL):** The HAL wraps the lower level drivers (like **MTB-PDL-CAT1**) and provides a high-level interface to the MCU. The interface is abstracted to work on any MCU.
- **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.
- **Middleware:** Middleware is a set of firmware modules that provide specific capabilities to an application. Some middleware may provide network protocols (e.g. MQTT), and some may provide high level software interfaces to device features (e.g. USB, audio).
- **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.
- **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.
- **WICED:** 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.

Revision history
Revision history

Document version	Date of release	Description of changes
**	2017-07-26	New application note
*A	2018-01-09	Updated screenshots with latest release of ModusToolbox™ Added new supported PSoC™ 6 MCU devices Added AnyCloud description under ModusToolbox™ software
*B	2019-04-16	Added new supported PSoC™ 6 MCU device – PSoC™ 62S4 Added information on PSoC™ 6 product lines and development kits available for each product line
*C	2020-05-06	Updated Figure 1 . Updated Screenshots with MTB v2.2. Added mtb_shared folder description, updated application creation process with MTB flow.
*D	2021-03-11	Updated to Infineon template
*E	2021-07-09	Updated the GitHub links Added reference to new PSoC™ 6 MCU low-power analog Updated Figure 17 to Figure 20 Firmware updated to the latest version

Trademarks

All referenced product or service names and trademarks are the property of their respective owners.

Edition 2021-07-09

Published by

Infineon Technologies AG

81726 Munich, Germany

© 2021 Infineon Technologies AG.

All Rights Reserved.

Do you have a question about this document?

Go to: www.cypress.com/support

Document reference

002-28571 Rev. *E

IMPORTANT NOTICE

The information contained in this application note is given as a hint for the implementation of the product only and shall in no event be regarded as a description or warranty of a certain functionality, condition or quality of the product. Before implementation of the product, the recipient of this application note must verify any function and other technical information given herein in the real application. Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind (including without limitation warranties of non-infringement of intellectual property rights of any third party) with respect to any and all information given in this application note.

The data contained in this document is exclusively intended for technically trained staff. It is the responsibility of customer's technical departments to evaluate the suitability of the product for the intended application and the completeness of the product information given in this document with respect to such application.

For further information on the product, technology, delivery terms and conditions and prices please contact your nearest Infineon Technologies office (www.infineon.com).

WARNINGS

Due to technical requirements products may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies office.

Except as otherwise explicitly approved by Infineon Technologies in a written document signed by authorized representatives of Infineon Technologies, Infineon Technologies' products may not be used in any applications where a failure of the product or any consequences of the use thereof can reasonably be expected to result in personal injury.