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-core 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 application using 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 3.0 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.

Table of contents

Table of contents

1 Introduction

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

- Single-core microcontroller: Arm® Cortex®-M4 (CM4); or dual-core 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 (Board Support Packages) 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.

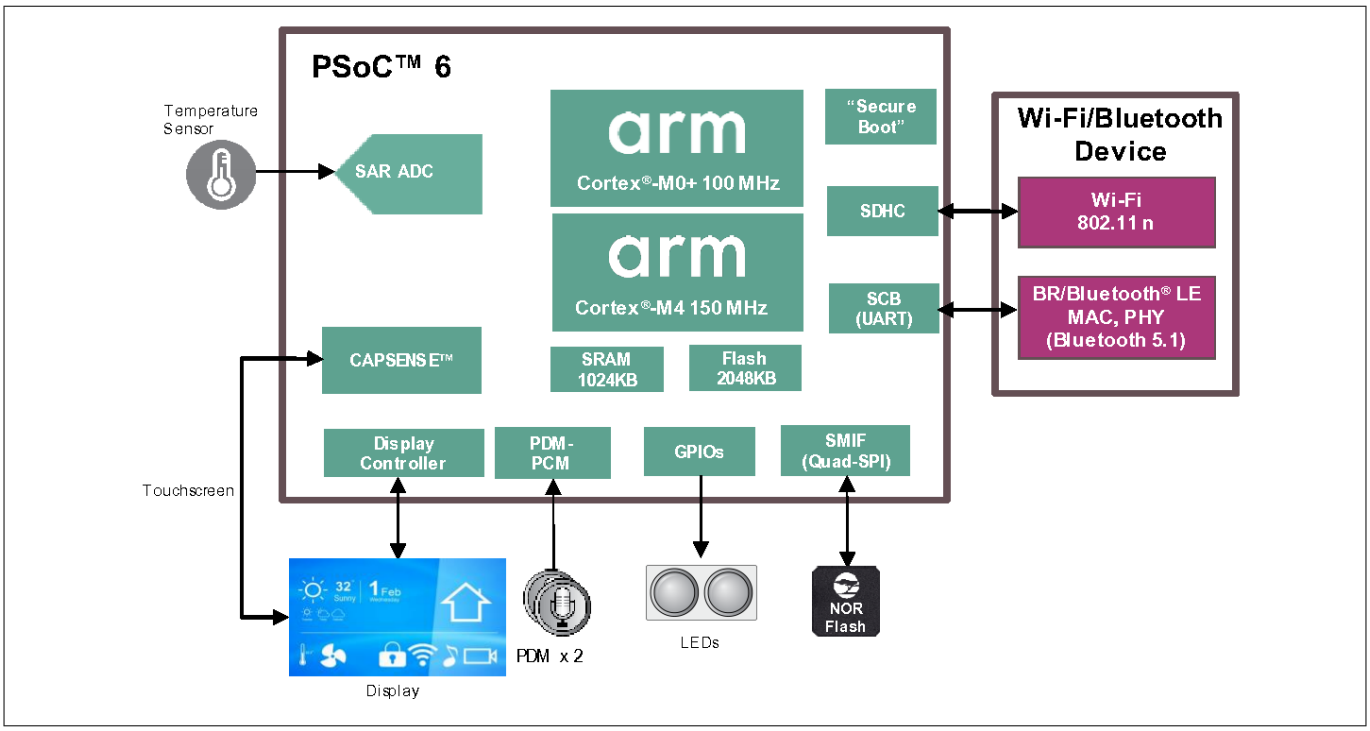

Figure 1 Application-level block diagram using PSoC™ 6 MCU

1 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+ core and application features executed by CM4 core

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.

1 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.

2 Development ecosystem

2.1 PSoC™ resources

A wealth of data available here helps you to select the right PSoC™ MCU 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™ 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.
- 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 MCUs.
- 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™:** This software includes configuration tools, low-level drivers, middleware libraries and other packages that enable you to create MCU and wireless applications. All tools run on Windows, macOS and Linux. ModusToolbox™ includes an Eclipse IDE, which provides an integrated flow with all the ModusToolbox™ tools. Other IDEs such as Visual Studio Code, IAR Embedded Workbench and Arm® MDK (µVision®) are also supported.

  ModusToolbox™ software supports stand-alone device and middleware configurators. Use the configurators to set the configuration of different blocks in the device and generate code that can be used in firmware development. The software supports all PSoC™ 6 MCUs. It is recommended that you use ModusToolbox™ software for all application development for PSoC™ 6 MCUs. See the ModusToolbox™ tools package 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.
  - Connectivity ecosystem – a set of libraries providing core functionality of Wi-Fi including connectivity, security, firmware upgrade support, and application layer protocols for applications.
  - Amazon FreeRTOS ecosystem – extends the FreeRTOS kernel with software libraries that make it easy to securely connect small, low-power Infineon devices to most cloud services.

ModusToolbox™ tools and resources can also be used on the command line. See the build system chapter in theModusToolbox™ tools package user guide for detailed documentation.

- **PSoC™ Creator:** A proprietary IDE that runs on Windows only. It supports a subset of PSoC™ 6 MCUs, as well as other PSoC™ 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 Installing the ModusToolbox™ tools package

Refer to the ModusToolbox™ tools package installation guide for details.

2.2.2 Choosing an IDE

ModusToolbox™ software, the latest-generation toolset, is supported across Windows, Linux, and macOS platforms. ModusToolbox™ software supports 3rd-party IDEs, including the Eclipse IDE, Visual Studio Code, Arm® MDK (µVision), and IAR Embedded Workbench. The tools package includes an implementation for the Eclipse IDE for your convenience. The tools support all PSoC™ 6 MCUs. The associated BSP and library configurators also work on all three host operating systems.

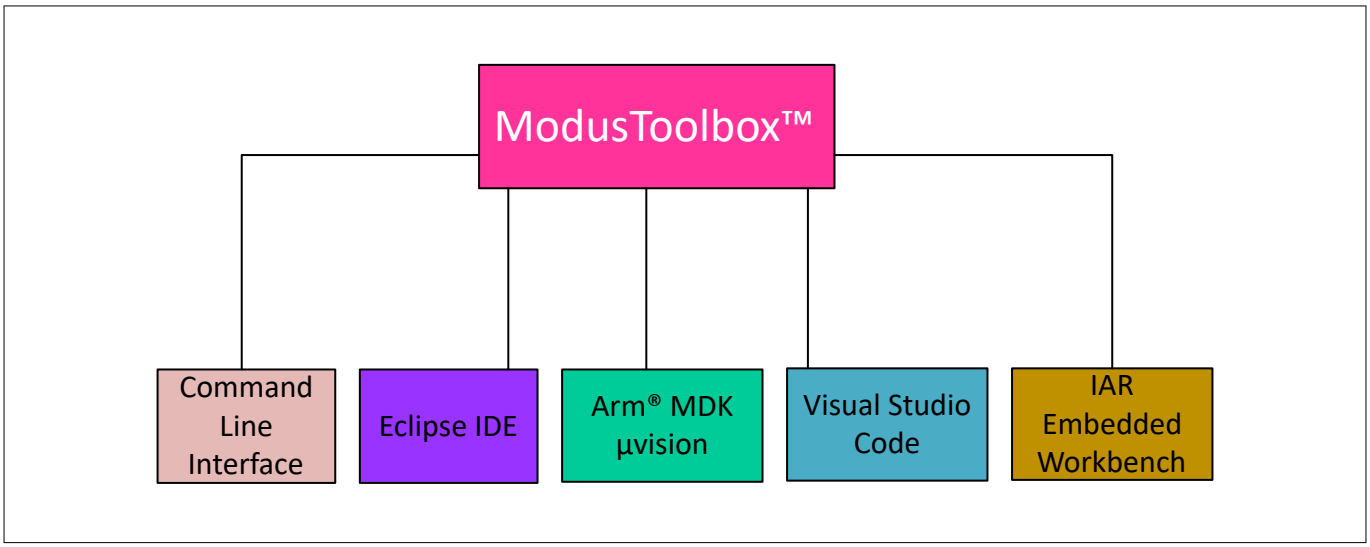

Figure 2 ModusToolbox™ environment

Certain features of the PSoC™ 6 MCU, such as UDBs (Universal Digital Blocks) and USB are not supported in ModusToolbox™ version 2.x and earlier. Newer versions of ModusToolbox™ support the USB host feature and improve the user experience with true multi-core debug support.

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 not supported in 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.

2.2.3 ModusToolbox™ software

ModusToolbox™ software 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. These include configuration tools, low-level drivers, libraries, and operating system support, most of which are compatible with Linux-, macOS-, and Windows-hosted environments.

Figure 3 shows a high-level view of what is available as part of ModusToolbox™ software. For a more in-depth overview of the ModusToolbox™ software, see ModusToolbox™ tools package user guide.

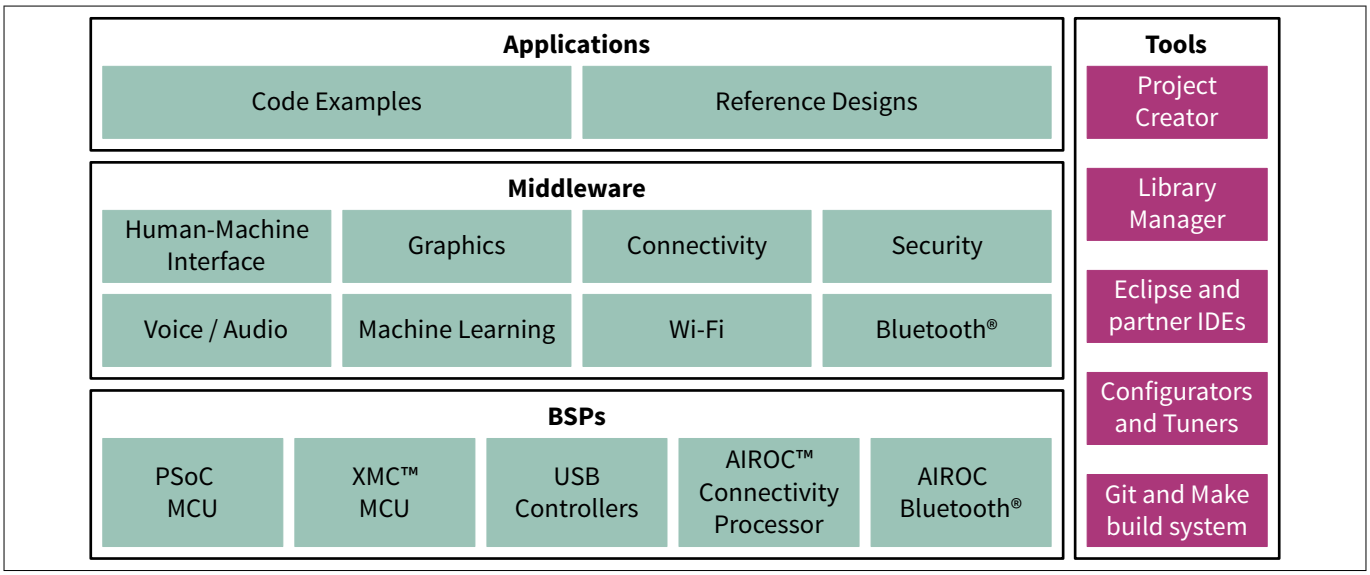

Figure 3 ModusToolbox™ software

The ModusToolbox™ tools package 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. This means you choose your compiler, IDE, RTOS, and ecosystem without compromising usability or access to our industry-leading CAPSENSE™ (Human-Machine Interface), AIROC™ Wi-Fi and Bluetooth®, security, and various other features.

One part of the ModusToolbox™ ecosystem is run-time software that helps you rapidly develop Wi-Fi and Bluetooth® applications using connectivity combo devices, such as AIROC™ CYW43012 and CYW43439 (among others), with the PSoC™ 6 MCU. See the ModusToolbox™ run-time software reference guide for details.

Design configurators are the tools that help you create the configurable code for your BSP/Middleware. Jump to Configurators to know more about it.

Figure 4 shows a run-time software diagram to showcase some of the application capabilities of Infineon devices using ModusToolbox™ software.

2 Development ecosystem

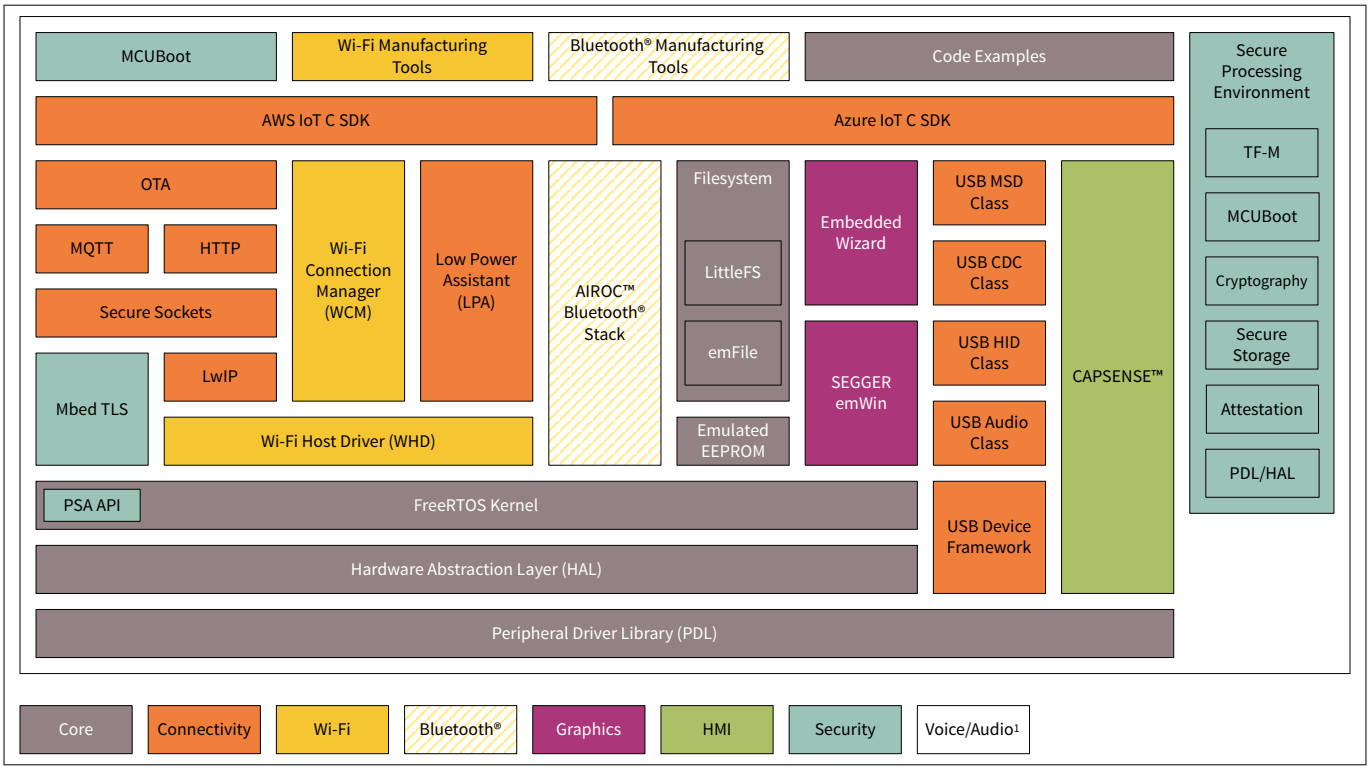

Figure 4 ModusToolbox™ run-time software diagram

All the application-level 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 BSP 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. See BSP Assistant to create your BSP.
- 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 lower-level control.
- PSoC™ 6 peripheral driver library (PDL) – The PDL integrates device header files, start-up code, and peripheral drivers into a single package. The PDL supports the PSoC™ 6 MCU 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.
- Middleware (MW) – Extensive middleware libraries that provides specific capabilities to an application. The available middleware spans across connectivity (OTA, Bluetooth®, AWS IoT, Bluetooth® LE, Secure Sockets) to PSoC™ 6 MCU-specific functionality (CAPSENSE™, USB, device firmware upgrade (DFU), emWin). All the middleware is delivered as libraries via GitHub repositories.

2.2.4 ModusToolbox™ applications

With the release of ModusToolbox™ v3.x, multi-core support is introduced, which has altered the folder structure slightly from the previous version of ModusToolbox™.

ModusToolbox™ has two types of applications:

- Single-core application
- Multi-core application

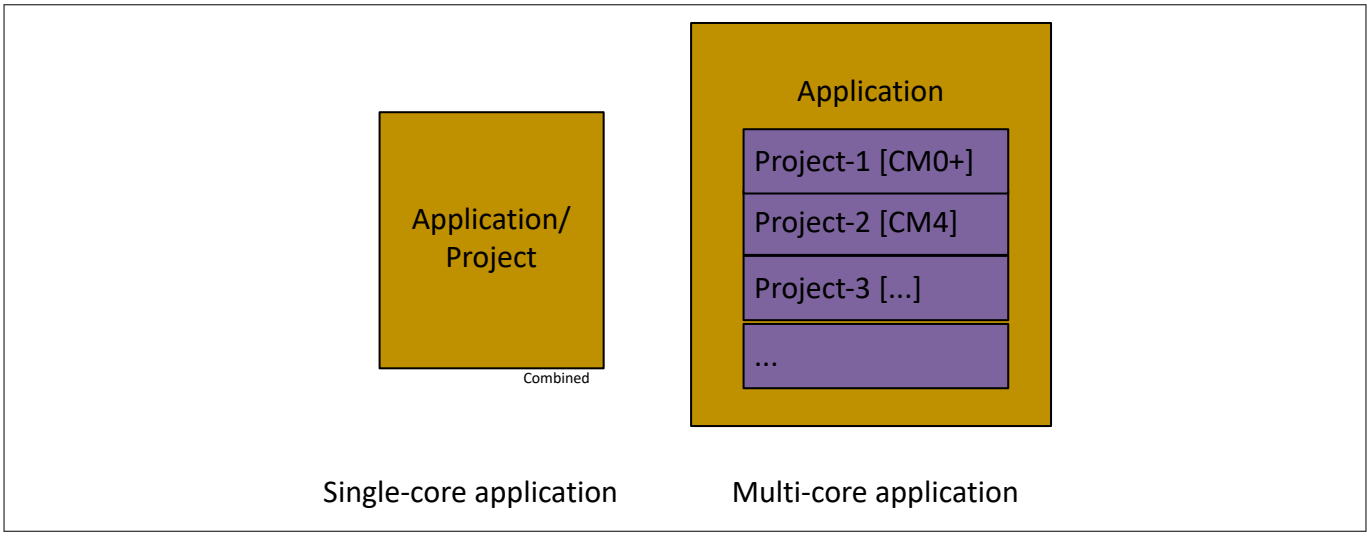

Figure 5 Application types

2 Development ecosystem

The following shows the new folder structure for an example single-core application:

```
<root>
  ApplicationName
  ->Makefile (MTB_TYPE=COMBINED)
  ->deps
      lib1.mtb (local)
      lib2.mtb (shared)
  ->libs
      lib1 (Infineon Git repo)
  ->bsps
      TARGET_BSP1 (not an Infineon Git repo; completely app-owned)
  ->templates
      TARGET_BSP1
        design.modus
        design.capsense
  ->main.c
  ->helper.h
  ->helper.c
  mtb_shared
    lib2/... (Infineon Git repo)
```

Figure 6 Folder structure for single-core applications

The following shows the new folder structure for an example multi-core application:

```
<root>
  ApplicationName
  ->Makefile (MTB_TYPE=APPLICATION)
  ->common.mk
  ->common_app.mk
  ->bsps
      TARGET_BSP1 (not an Infineon Git repo; completely app-owned)
  ->templates
      TARGET_BSP1
        design.modus
        design.capsense
  ->project1
      Makefile (MTB_TYPE=PROJECT)
      deps
        lib3.mtb (local)
        lib4.mtb (shared)
      libs
        lib3 (Infineon Git repo)
      main.c
      project1_helper.h
      project1_helper.c
  ->project2
      Makefile (MTB_TYPE=PROJECT)
      deps
        lib5.mtb (local)
        lib6.mtb (shared)
      libs
        lib5 (Infineon Git repo)
      main.c
      project2_helper.h
      project2_helper.c
  mtb_shared
    lib4/... (Infineon Git repo)
    lib6/... (Infineon Git repo)
```

Figure 7 Folder Structure for multi-core applications

The new flow using ModusToolbox™ versions 3.x can support multiple projects in an application. For multi-core applications, there are multiple projects, but only one project per core. The applications have app-owned BSPs, meaning the BSP will be common to all projects inside a multi-core application.

Going further, section 4 of this document describes creating a new single-core application using ModusToolbox™ software.

2.2.5 PSoC™ 6 software resources

The software for PSoC™ 6 MCUs includes 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.

2.2.5.1 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 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.

There are two types of configurators: BSP configurators that configure items that are specific to the MCU hardware and library configurators that configure options for middleware libraries.

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, pins, and the basic peripherals (e.g., UART, Timer, PWM).
- **CAPSENSE™ configurator and tuner:** Configure CAPSENSE™ and generate the required code and tune CAPSENSE™ applications.
- **LIN configurator:** Configure LIN middleware and generate the required configuration.
- **ML configurator:** To fit the pre-trained model of choice to the target device with a set of optimization parameters (Only available as a part of separate pack)
- **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 Smart I/O pins.
- **Bluetooth® configurator:** Configure the Bluetooth® settings.
- **EZ-PD™ configurator:** Configure parameters and select the features of the PD Stack 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™). BSP configurator files (e.g. *design.modus* or *design.cycapsense*) are provided as part of the BSP with default configurations while library configurators (e.g. design.cybt) are provided by the application. When an application is created based on Infineon BSP, the application makes use of BSP configurator files from the Infineon BSP repo. You can customize/create all the configurator files as per your application requirement using ModusToolbox™ software. See BSP Assistant to create your custom BSP. See ModusToolbox™ help for more details..

2.2.5.2 Library management for PSoC™ 6 MCU

The application can have shared/local libraries for the projects. If needed, different projects can use different versions of the same library. The shared libraries are downloaded under the `mtb_shared` directory. The application should use the `deps` folder to add library dependencies. The `deps` folder contains files with the `.mtb` file extension, which is used by ModusToolbox™ to download its git repository. These libraries are direct dependencies of the ModusToolbox™ project.

The Library Manager helps to add/remove/update the libraries of your projects. It also identifies whether the particular library has a direct dependency on any other library using the manifest repository available on GitHub and fetches all the dependencies of that particular library. These dependency libraries are indirect dependencies of the ModusToolbox™ project. These dependencies can be seen under the `libs` folder. For more information, see the Library Manager user guide located at `<install_dir> /ModusToolbox/tools_<version>/library-manager/docs/library-manager.pdf`.

2.2.5.3 Software development for PSoC™ 6 MCU

The ModusToolbox™ ecosystem provides significant source code and tools 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 firmware
- 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™. The BSP configurator files are used to configure device peripherals, pins, and memory using peripheral driver library code. The middleware is delivered as separate libraries for each feature/function such that it can be used across multiple platforms. For example, abstraction-rtos, lwip, usb, etc.

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 MCUs, 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 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 8 shows, with the ModusToolbox™ software, you can:

1. Choose a BSP (Project Creator).
2. Create a new application based on a list of starter applications, filtered by the BSPs that each application supports (Project Creator).
3. Add BSP or middleware libraries (Library Manager).
4. Develop your application firmware using the HAL or PDL for PSoC™ 6 MCU (IDE of choice or command line).

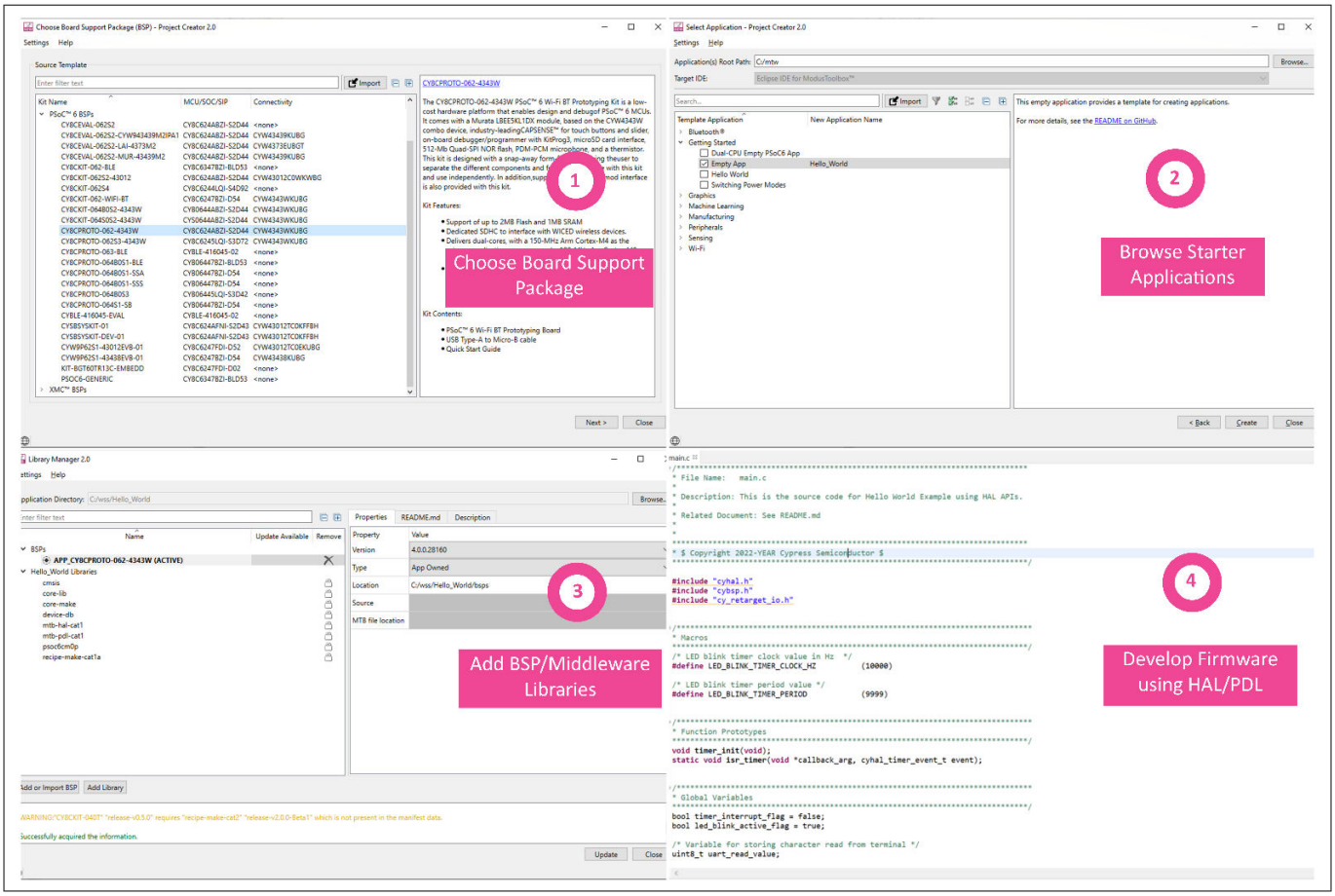

Figure 8 ModusToolbox™ resources and middleware

2.2.6 ModusToolbox™ help

The ModusToolbox™ ecosystem provides documentation and training. One way to access it is launching the Eclipse IDE for ModusToolbox™ software and navigating to the following **Help** menu items:

Choose **Help** > **ModusToolbox™ General Documentation**:

- **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.
- **ModusToolbox™ 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™ Training Class Material:** Links to the training material available at https://github.com/Infineon/training-modustoolbox.
- **Release Notes**

For documentation on Eclipse IDE for ModusToolbox™, choose **Help** > **Eclipse IDE for ModusToolbox™ General 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 Survival Guide**

2.3 Support for other IDEs

You can develop firmware for PSoC™ 6 MCUs using your favorite IDE such as IAR Embedded Workbench, Keil µVision 5 or Visual Studio Code.

ModusToolbox™ configurators are stand-alone 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. It is recommended that you generate resource configurations using the configuration tools provided with ModusToolbox™ software.

See ModusToolbox™ tools package user guide for details.

2.4 FreeRTOS support with ModusToolbox™

Adding native FreeRTOS support to a ModusToolbox™ application project is like adding any middleware library. You can include the FreeRTOS middleware in your application by using the Library Manager. If using the Eclipse IDE, select the application project and click the **Library Manager** link in the **Quick Panel**. Click **Add Library** and select **freertos** from the **Core** dialog, as Figure 9 shows.

The `.mtb` file pointing to the FreeRTOS middleware is added to the application project's `deps` directory. The middleware content is also downloaded and placed inside the corresponding folder called **freertos**. The default location is in the shared asset repo named mtb_shared. To continue working with FreeRTOS follow the steps in the Quick Start section of FreeRTOS documentation.

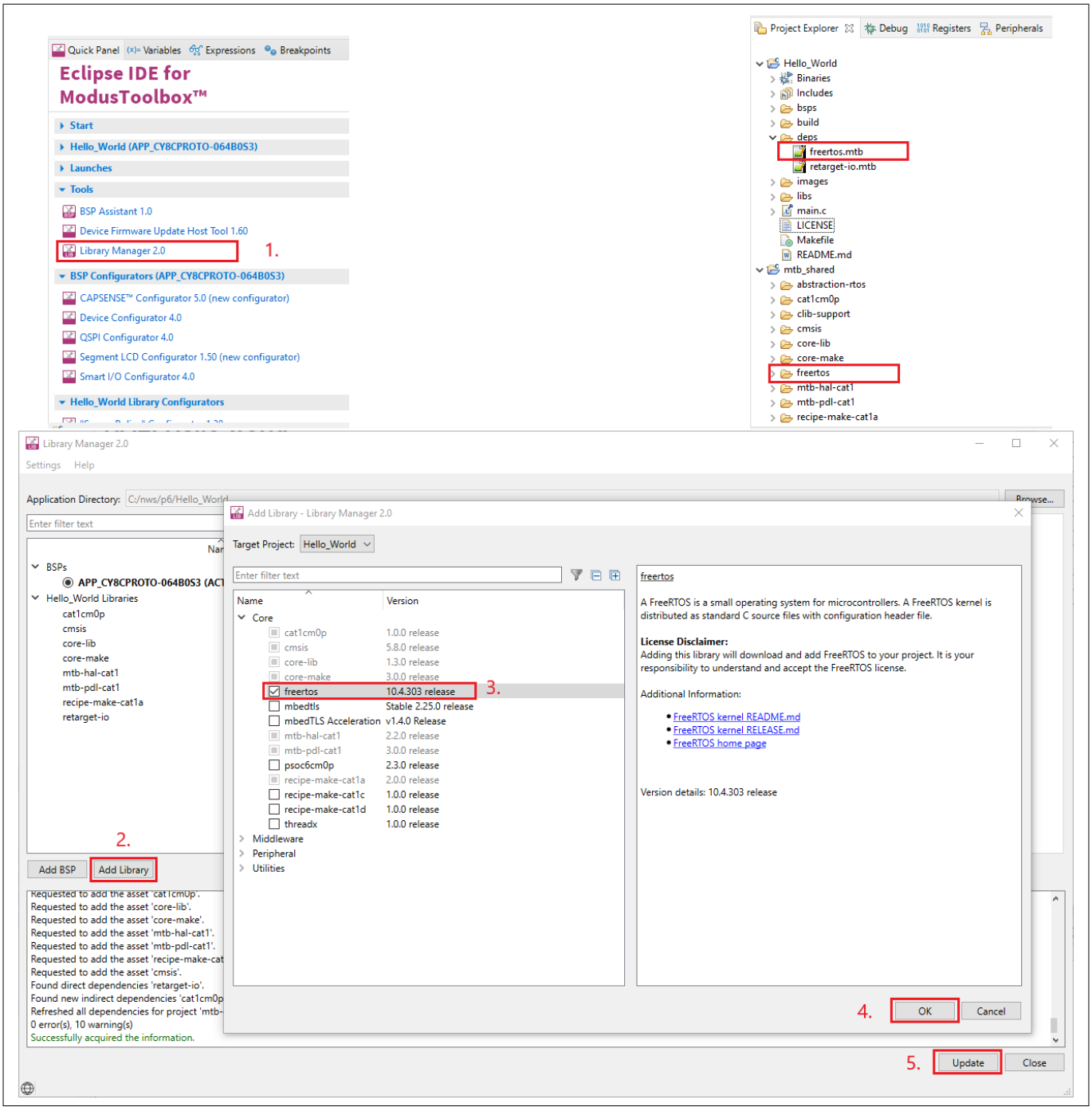

Figure 9 Import FreeRTOS middleware in ModusToolbox™ application

2.5 Programming/debugging using Eclipse IDE

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.

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 firmware pre-installed. To work with ModusToolbox™, upgrade the firmware to KitProg3 using the fw-loader

command-line tool included in the ModusToolbox™ software. 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 the *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	PSoC™ 6 WiFi-Bluetooth® pioneer kit (CY8CKIT-062-WiFi-BT) PSoC™ 6 Wi-Fi Bluetooth® prototyping kit (CY8CPROTO-062-4343W) PSoC™ 62S2 Wi-Fi Bluetooth® pioneer kit (CY8CKIT-062S2-43012) PSoC™ 62S3 Wi-Fi Bluetooth® prototyping kit (CY8CPROTO-062S3-4343W) PSoC™ 62S1 Wi-Fi Bluetooth® pioneer kit (CYW9P62S1-43438EVB-01) PSoC™ 62S1 Wi-Fi Bluetooth® pioneer kit (CYW9P62S1-43012EVB-01) PSoC™ 62S4 pioneer kit (CY8CKIT-062S4)
Connectivity	PSoC™ 6 Bluetooth® LE pioneer kit (CY8CKIT-062-BLE) PSoC™ 6 Bluetooth® LE prototyping kit (CY8CPROTO-063-BLE)
Security	PSoC™ 64 “Secure Boot” Wi-Fi Bluetooth® pioneer kit (CY8CKIT-064B0S2-4343W) PSoC™ 64 Standard “Secure” - AWS Wi-Fi Bluetooth® pioneer kit (CY8CKIT-064S0S2-4343W)

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

3 Device features

PSoC™ 6 MCUs have extensive features as shown in Figure 10. The following is a list of 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

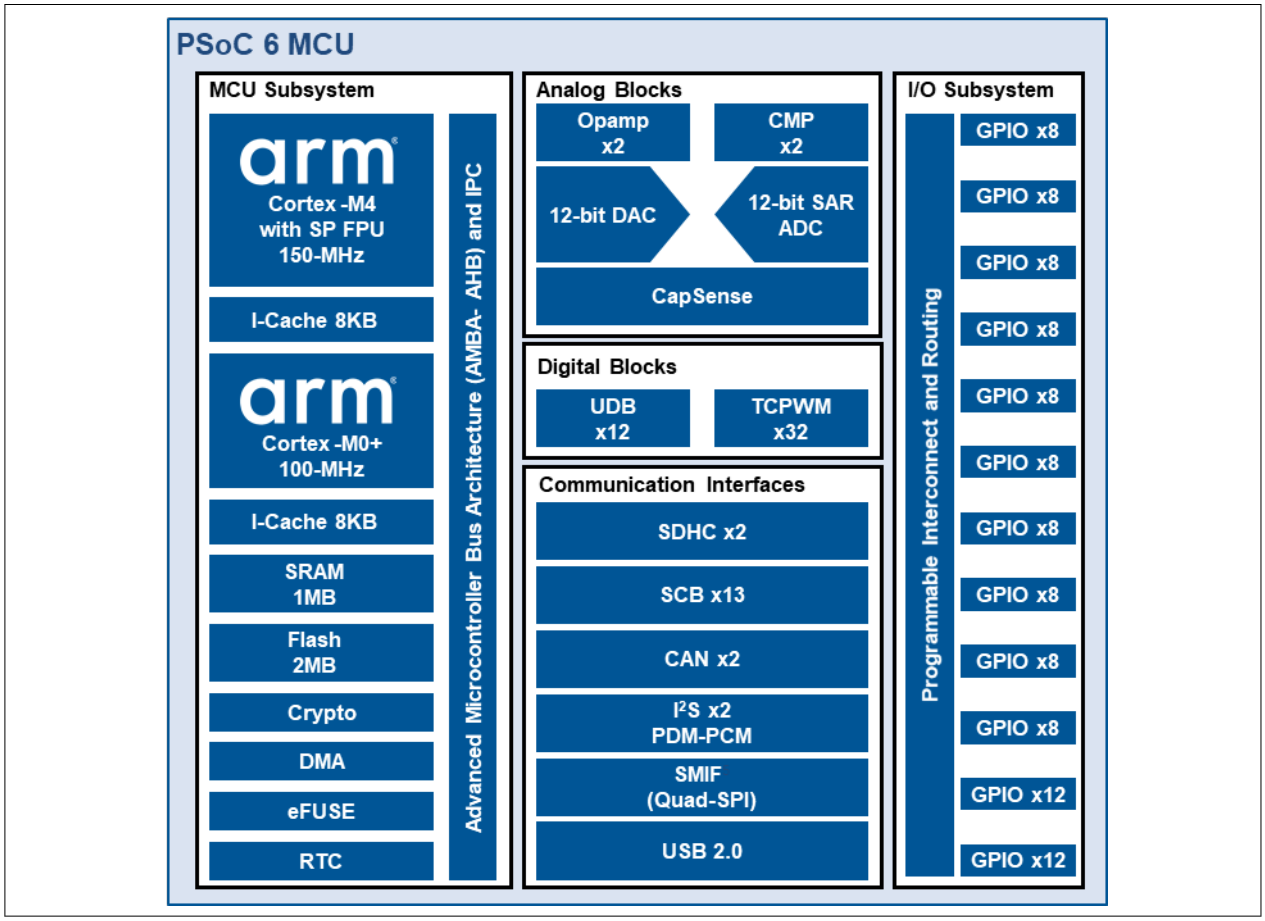

Figure 10 PSoC™ 6 MCU block diagram

- **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

3 Device features

- 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
- 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

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
- Makes 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 example design shown below is developed for the PSoC™ 6 Wi-Fi Bluetooth® prototyping kit (CY8CPROTO-062-4343W). However, you can build the application for other development kits. See the Using these instructions section.

4.1.2 Software

- ModusToolbox™ 3.0 or above

After installing the software, refer to the ModusToolbox™ tools package 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 configuration
- 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)

4.3 About the design

This design uses the CM4 core 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 core and configures the CM0+ core to go to sleep. The CM4 core 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 App** starter application and manually adds the functionality from the **Hello World** starter application. The Eclipse IDE for ModusToolbox™ is used in the instructions, but you can use any IDE or the command line if you prefer.

If you are familiar with developing projects with ModusToolbox™ software, 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 can use the **Hello World** code example as a reference while following the instructions.

Launch the Eclipse IDE for ModusToolbox™ to get started. Please note that 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 11 shows. Alternatively, you can type in a directory name to be used as the workspace directory along with the complete path, and the Eclipse IDE will create the directory for you.

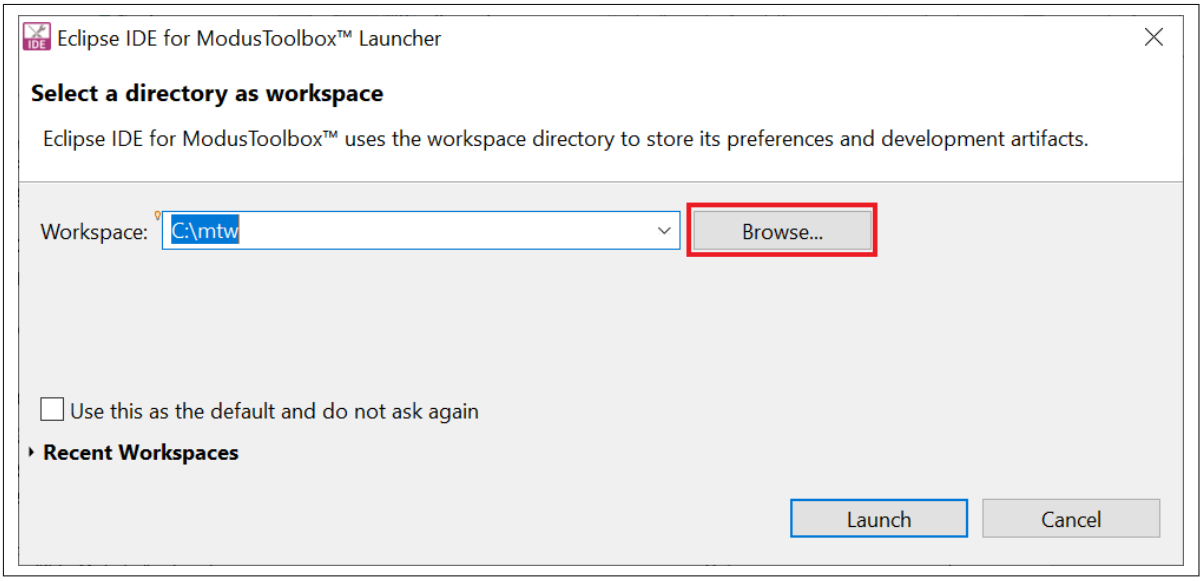

Figure 11 Select a directory as the workspace

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™ Application**, as Figure 12 shows.

   The Project Creator opens.

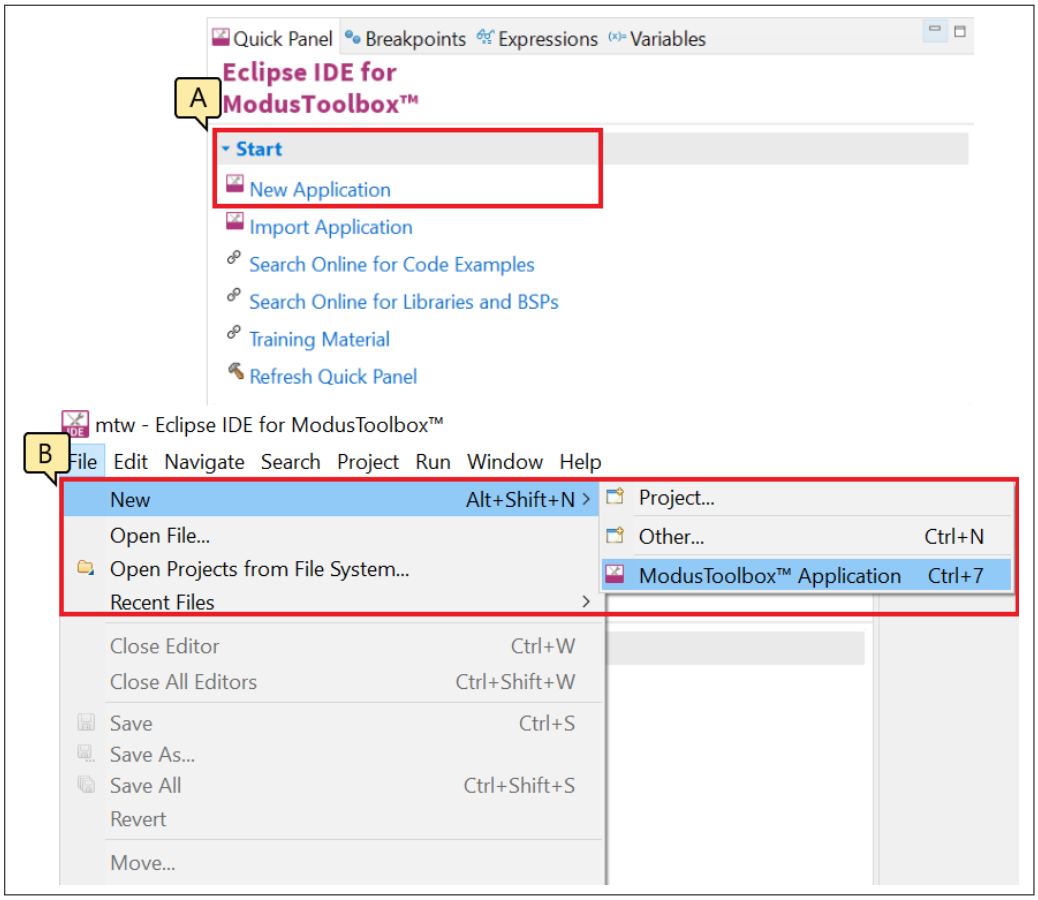

Figure 12 Create a new ModusToolbox™ application

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.

a. 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 13 for help with this step

b. Click **Next**

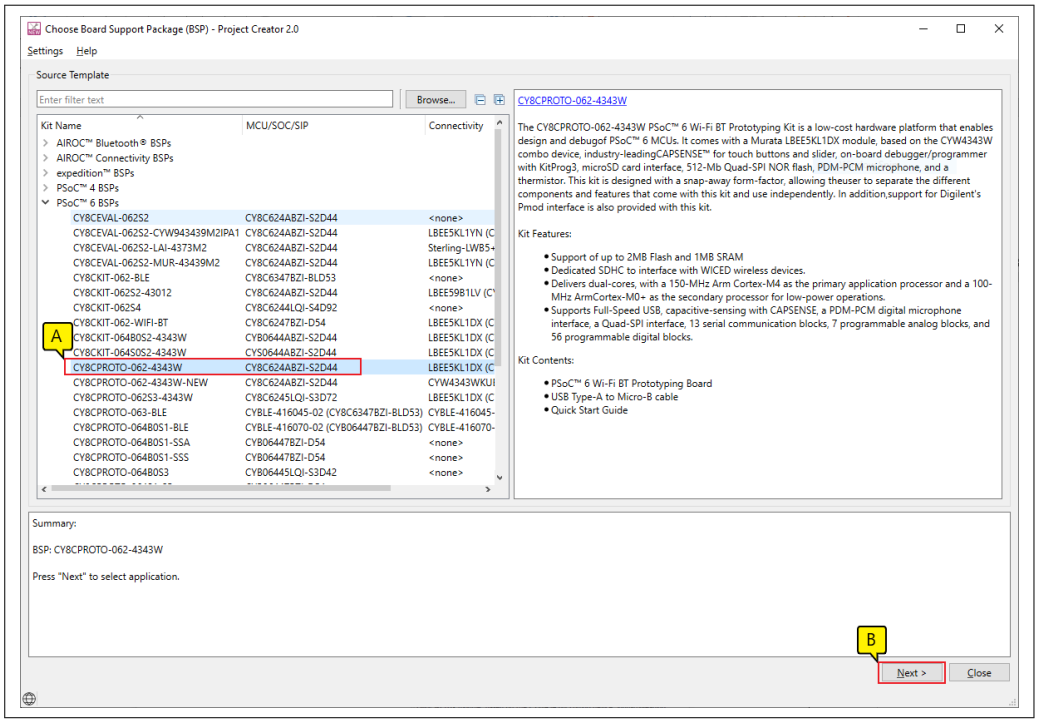

Figure 13 Choose target hardware

c. In the **Select Application** dialog, select **Empty App** starter application, as Figure 14 shows.

d. 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.

e. Click **Create** to create the application, as Figure 14 shows, wait for the Project Creator to automatically close once the project is successfully created.

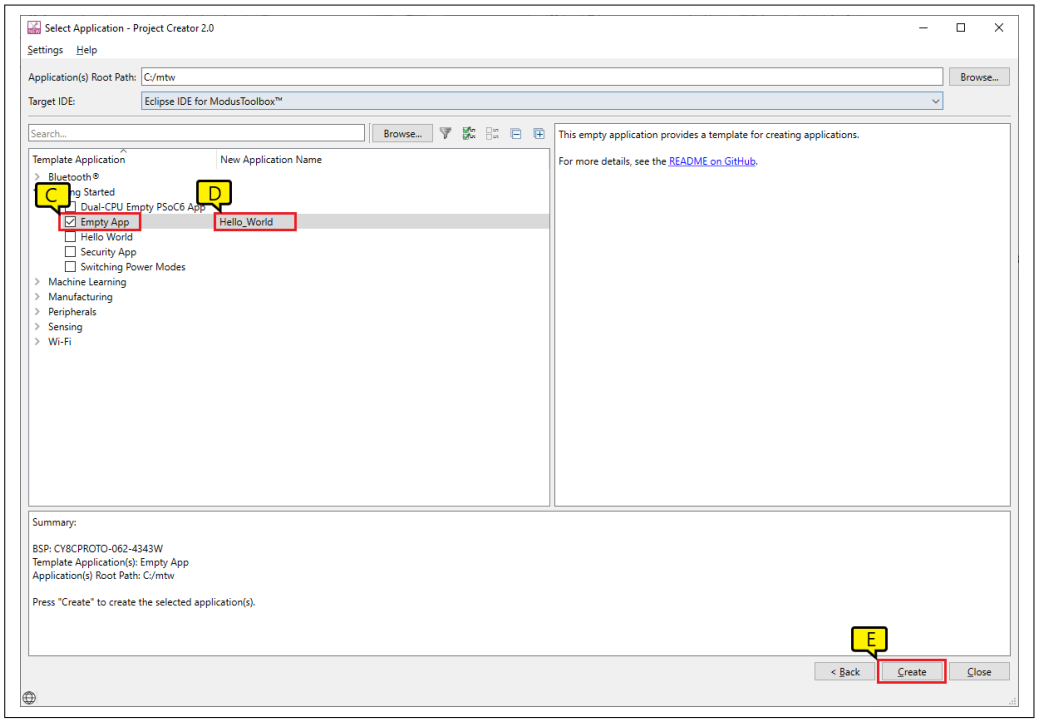

Figure 14 Choose starter application

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 Custom BSP App Note or the BSP Assistant user guide.

4.5 Part 2: View and modify the design configuration

Figure 15 shows the Eclipse IDE Project Explorer interface displaying the structure of the application project. A PSoC™ 6 MCU application consists of a project to develop code for the CM4 core. A project folder consists of various subfolders – each denoting a specific aspect of the project.

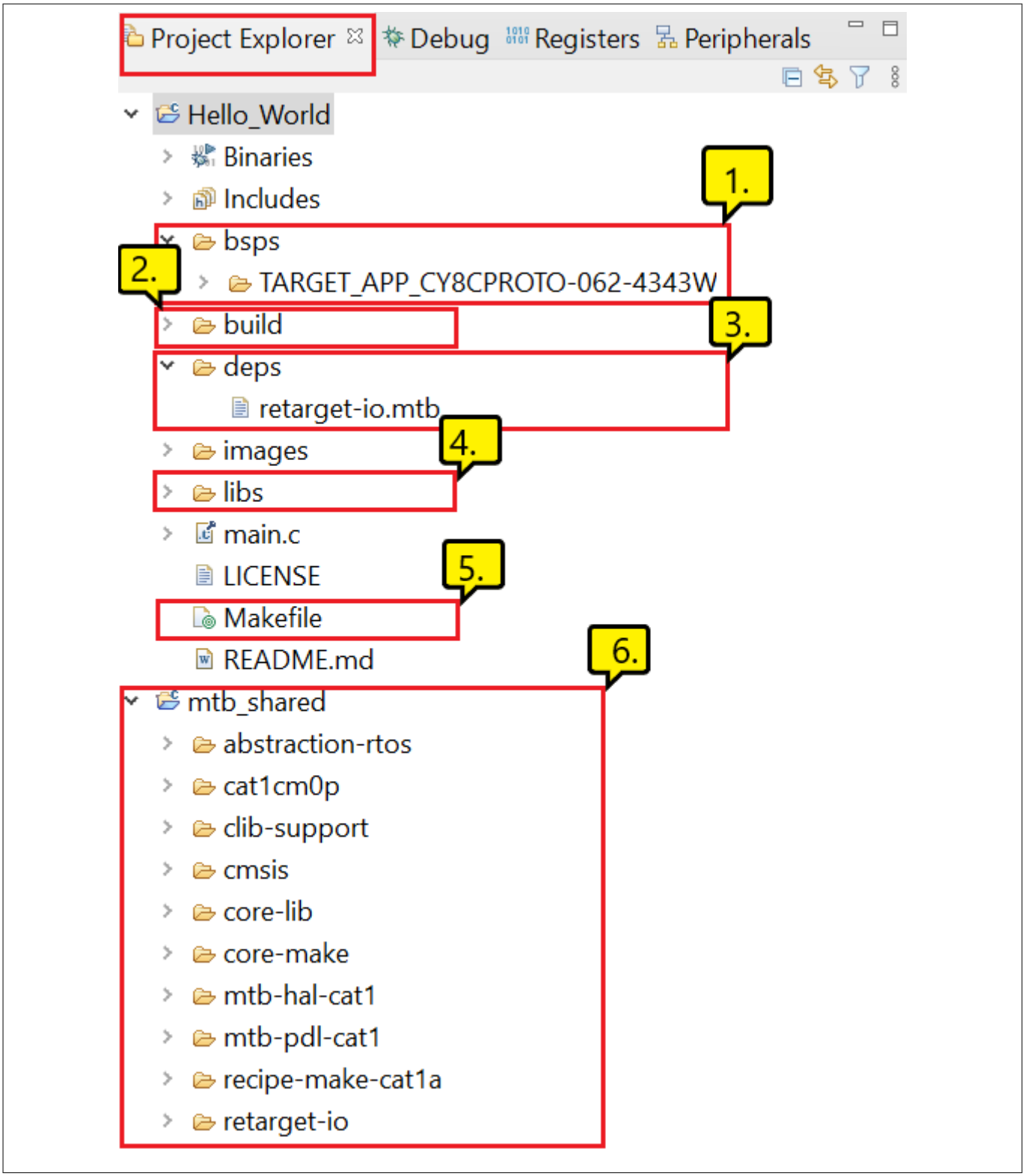

Figure 15 Project Explorer view

1. The files provided by the BSP are in the `bsps` folder and are listed under `TARGET_<bsp name>` sub-folders. All the input files for the device and peripheral configurators are in the config folder inside the BSP. The `GeneratedSource` folder in the BSP contains the files that are generated by the configurators and are prefixed with `cycfg_`. These files contain the design configuration as defined by the BSP. From ModusToolbox™ 3.x or later, you can directly customize configurator files of BSP for your application

rather than overriding the default design configurator files with custom design configurator files since BSPs are completely owned by the application.

The BSP folder also contains the linker scripts and the start-up code for the PSoC™ 6 MCU used on the board.

2. The `build` folder contains all the artifacts resulting from a build of the project. The output files are organized by target BSPs.
3. The `deps` folder contains `.mtb` files, which provide the locations from which ModusToolbox™ pulls the libraries that are directly referenced by the application. These files typically each contain the GitHub location of a library. The `.mtb` files also contain 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 locally.

   For example, Here, `retarget-io.mtb` points to `mtb://retarget-io#latest-v1.X#$$ASSET_REPO$$/retarget-io/latest-v1.X`. The variable `$$ASSET_REPO$$` points to the root of the shared location which defaults to `mtb_shared`. If the library must be local to the application instead of shared, use `$$LOCAL$$` instead of `$$ASSET_REPO$$`.
4. The `libs` folder also contains `.mtb` files. In this case, they 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 `CY8CPROTO-062-4343W` populates `libs` folder with the following `.mtb` files: *cmsis.mtb*, ***core-lib.mtb***, *core-make.mtb*, *mtb-hal-cat1.mtb*, *mtb-pdl-cat1.mtb*, *cat1cm0p.mtb*, *recipe-make-cat1a.mtb*.

   The `libs` folder contains the file `mtb.mk`, 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.

   Everything in the `libs` folder is generated by the Library Manager so you should not manually edit anything in that folder.
5. An application contains a Makefile which is at the application's root folder. This file 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. In that case there is a Makefile at the application level and one inside each project. See AN215656 - PSoC™ 6 MCU dual-core system design for details related to multi-project applications.
6. By default, when creating a new application or adding a library to an existing application and specifying it as shared, all libraries are placed in an `mtb_shared`directory adjacent to the application directories.

   The `mtb_shared` folder is shared between different applications within a workspace. Different applications may use different versions of shared libraries if necessary.

4.5.1 Opening the Device Configurator

BSP configurator files are in the `bsps/TARGET_<BSP-name>/config` folder. For example, click **<Application-name>** from **Project Explorer** then click **Device Configurator** link in the **Quick Panel** to open the file `design.modus` in the **Device Configurator** as shown in Figure 16. You can also open other configuration files in their respective configurators or click the corresponding links in the **Quick Panel**.

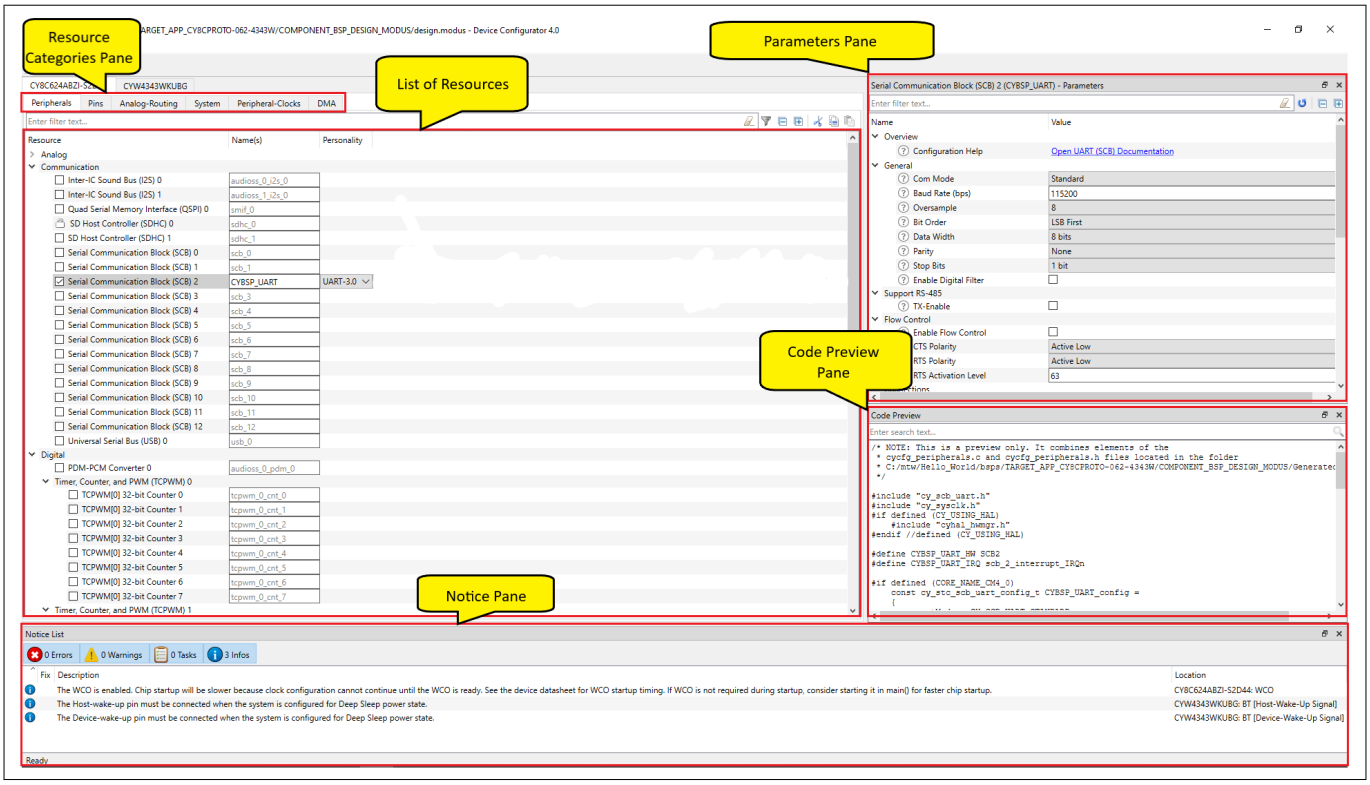

Figure 16 Device Configurator

The **DeviceConfigurator** provides a set of **Resources Categories** tabs. 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 **EZI2C**, **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. The Parameters pane and Code Preview pane may be displayed as tabs instead of separate windows but the contents will be the same.

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

Currently, the device configurator supports configurations using PDL source. If you choose to use HAL libraries in your application then you do not need to do any device configurations changes in here. The application project contains source files that help you create an application for the CM4 core (for example, `main.c`), while the CM0+ application is supplied as a default C file (`psoc6_02_cm0p_sleep.c` for the `CY8C624ABZI-D44` device). See the cat1cm0p 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.

4.5.2 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`.

1. In the **Quick Panel**, click the **Library Manager** link.

2. In the subsequent dialog, click **Add Libraries**.
3. Under **Peripherals**, select and enable retarget-io.
4. Click **OK** and then **Update**.

The files necessary to use the retarget-io middleware are added in the `mtb_shared > retarget_io` folder, and the `.mtb` file is added to the `deps` folder, as Figure 17 shows.

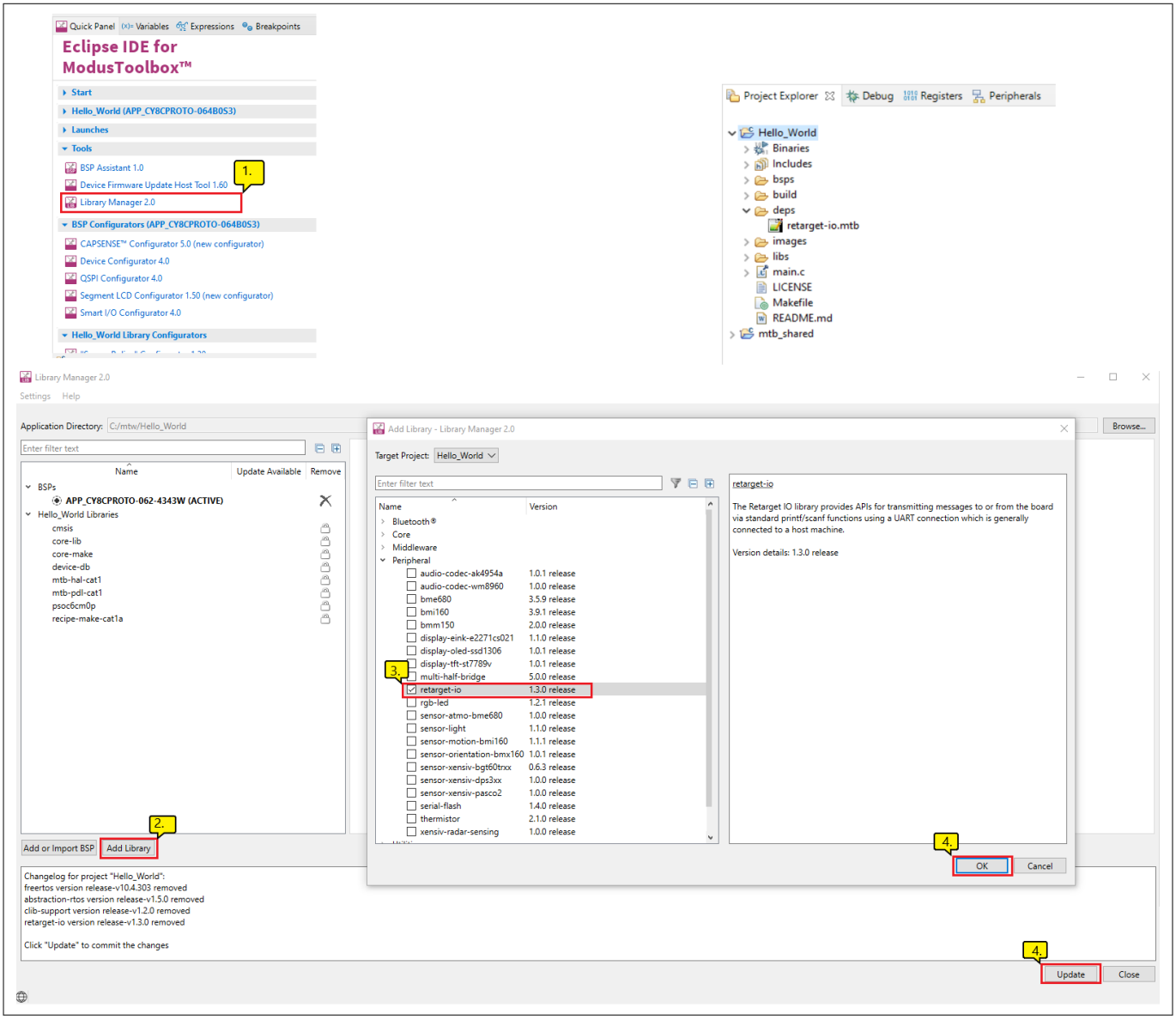

Figure 17 Add the retarget-io middleware

4.5.3 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 the BSP and HAL. Therefore, it is not necessary to configure them with the Device Configurator. See Part 3: Write firmware.

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 18 shows the firmware flowchart.

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

When the CM4 core 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 core toggles the LED state on the kit.

The firmware is designed to accept the '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 the retarget-io middleware uses the aliases set up in the BSP for the 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.

Note: You can open the Device Configurator to view the aliases that are set up in the BSP.

`cyhal_gpio_init()`- This function from the GPIO HAL initializes the physical pin to drive the LED. The LED used is derived from the alias for the pin set up in the BSP.

`timer_init()`- This function wraps a set of timer HAL function calls to instantiate and configure a hardware 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: `main.c` file

```
#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)

/* 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 core. 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 blinky. 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 */
    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);

                    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);
    }

    /* 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;
}
```

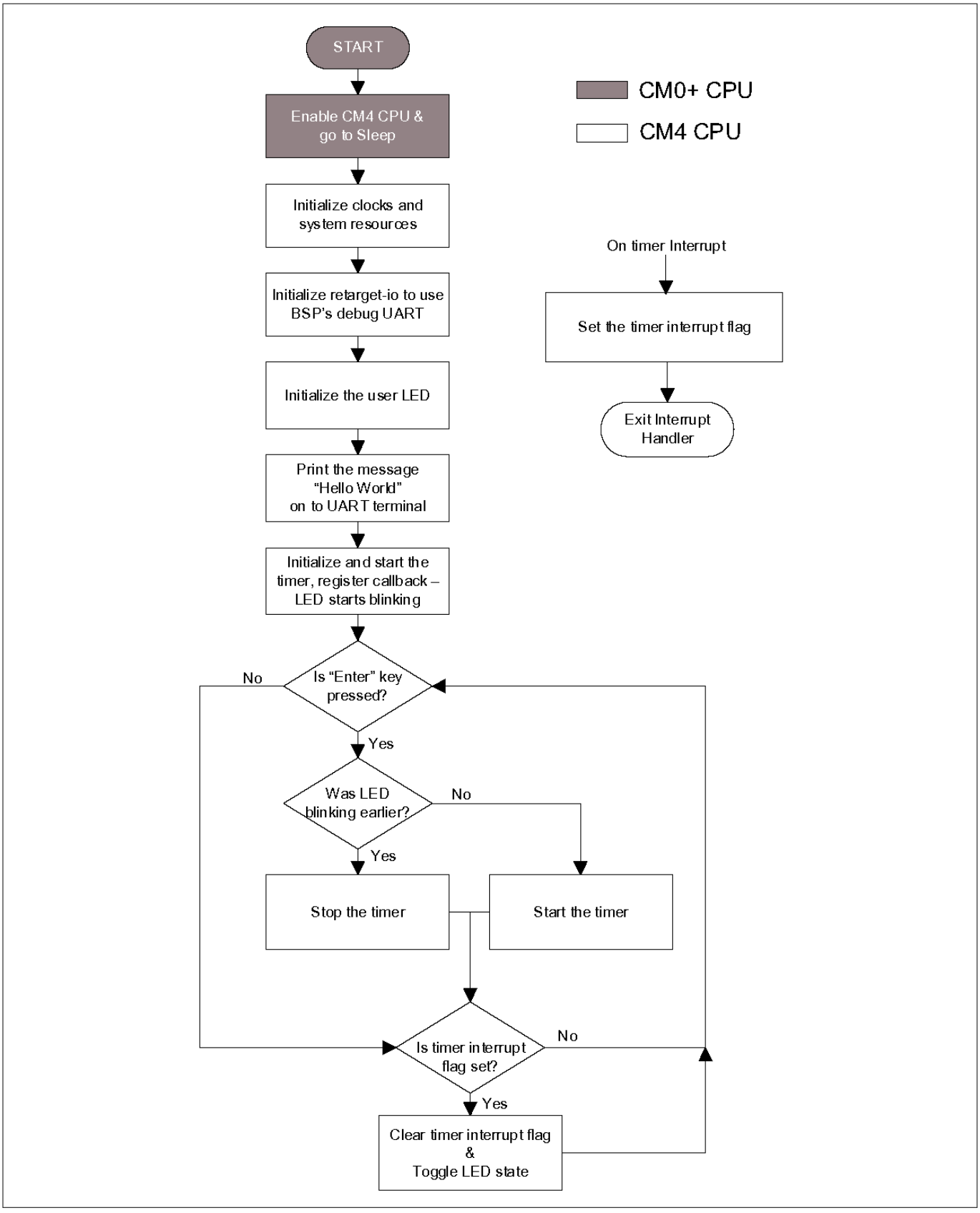

Figure 18 Firmware flowchart

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

4.7 Part 4: Build the application

This section shows how to build the application.

1. Select the application project in the **Project Explorer** view.
2. Click **Build Application** shortcut under the **<name>** group in the **Quick Panel**.

   It selects the build configuration from Makefile and compiles/links all projects that constitute the application. By default, Debug configurations are selected.
3. The **Console view** lists the results of the build operation, as Figure 19 shows.

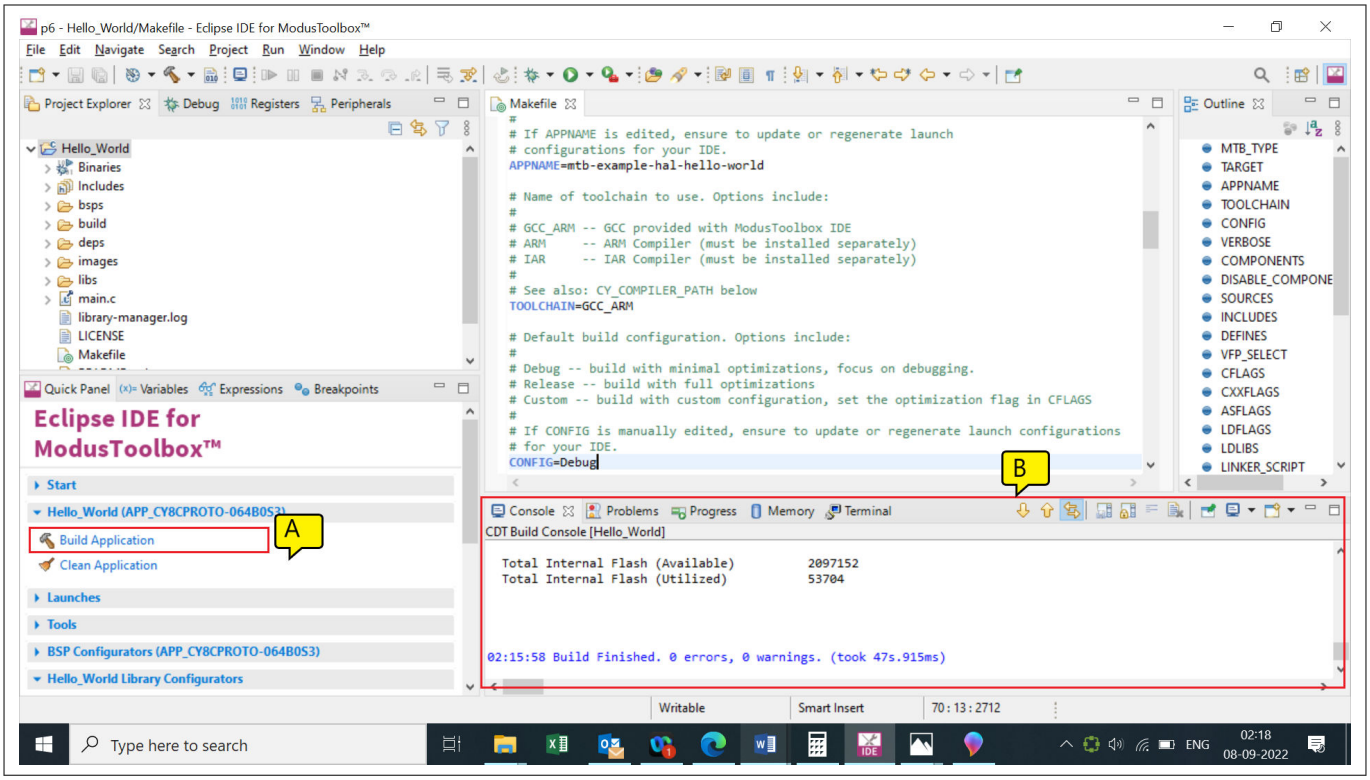

Figure 19 Build the application

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

*Note: You can also use the command line interface (CLI) to build the application. Please refer to the **Build system chapter** in the ModusToolbox™ tools package user guide. This document is located in the `/docs_<version>/` folder in the ModusToolbox™ installation.*

4.8 Part 5: Program the device

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

ModusToolbox™ software uses the OpenOCD protocol to program and debug applications on PSoC™ 6 MCUs. The kit must be running KitProg3. Some kits are shipped with KitProg2 firmware instead of KitProg3. See Programming/debugging using Eclipse IDE for details. The ModusToolbox™ tools package includes the `fw-loader` command-line tool to switch the KitProg firmware from KitProg2 to KitProg3. See 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.

If you are developing on your own hardware, you can use a hardware programmer/debugger; for example, a CY8CKIT-005 MiniProg4, J-Link, or ULINKpro.

Select the application project and click the **<application name> Program (KitProg3_MiniProg4)** shortcut under the **Launches** group in the **Quick Panel**, as Figure 20 shows. The IDE will select and run the appropriate run configuration.

Note: This step also performs a build if any files have been modified since the last build.

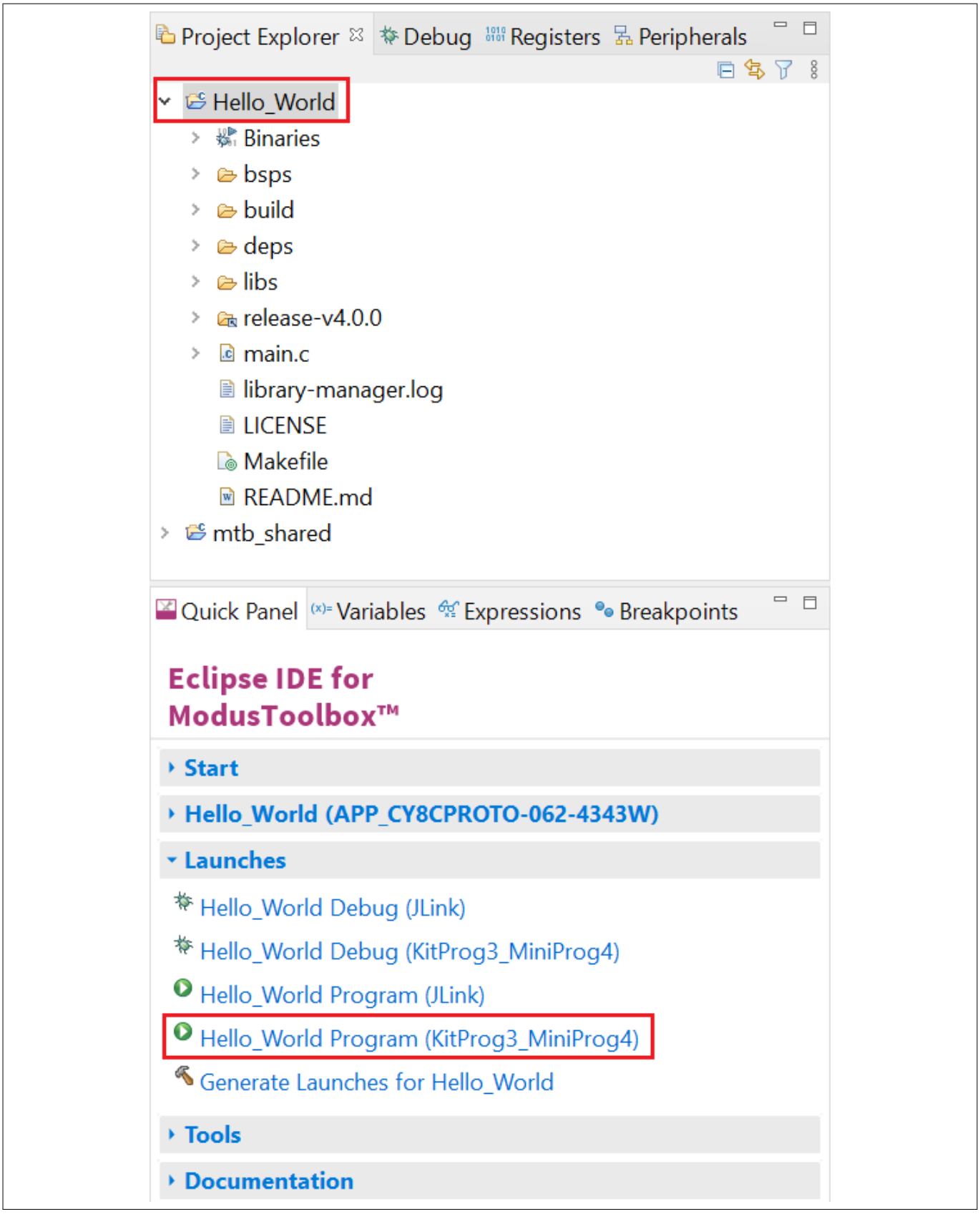

Figure 20 Programming an application to a device

The Console view lists the results of the programming operation, as Figure 21 shows.

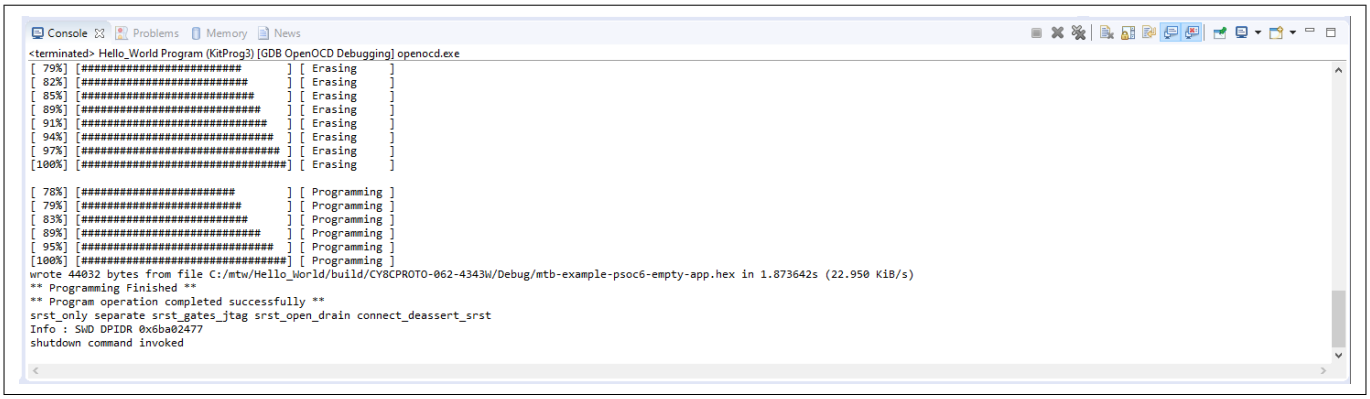

Figure 21 Programming an application to a device

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 22 shows. Note that your COM port number may be different.

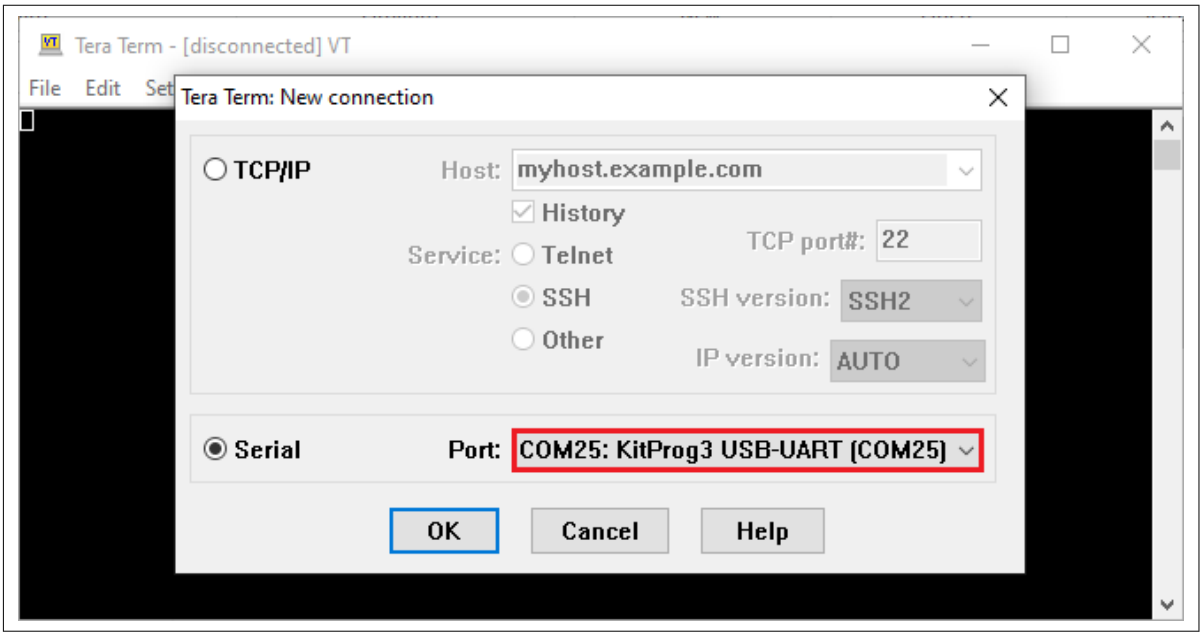

Figure 22 Selecting the KitProg3 COM port in Tera Term

2. Set the baud rate

Set the baud rate to 115200 under **Setup > Serial port** as Figure 23 shows.

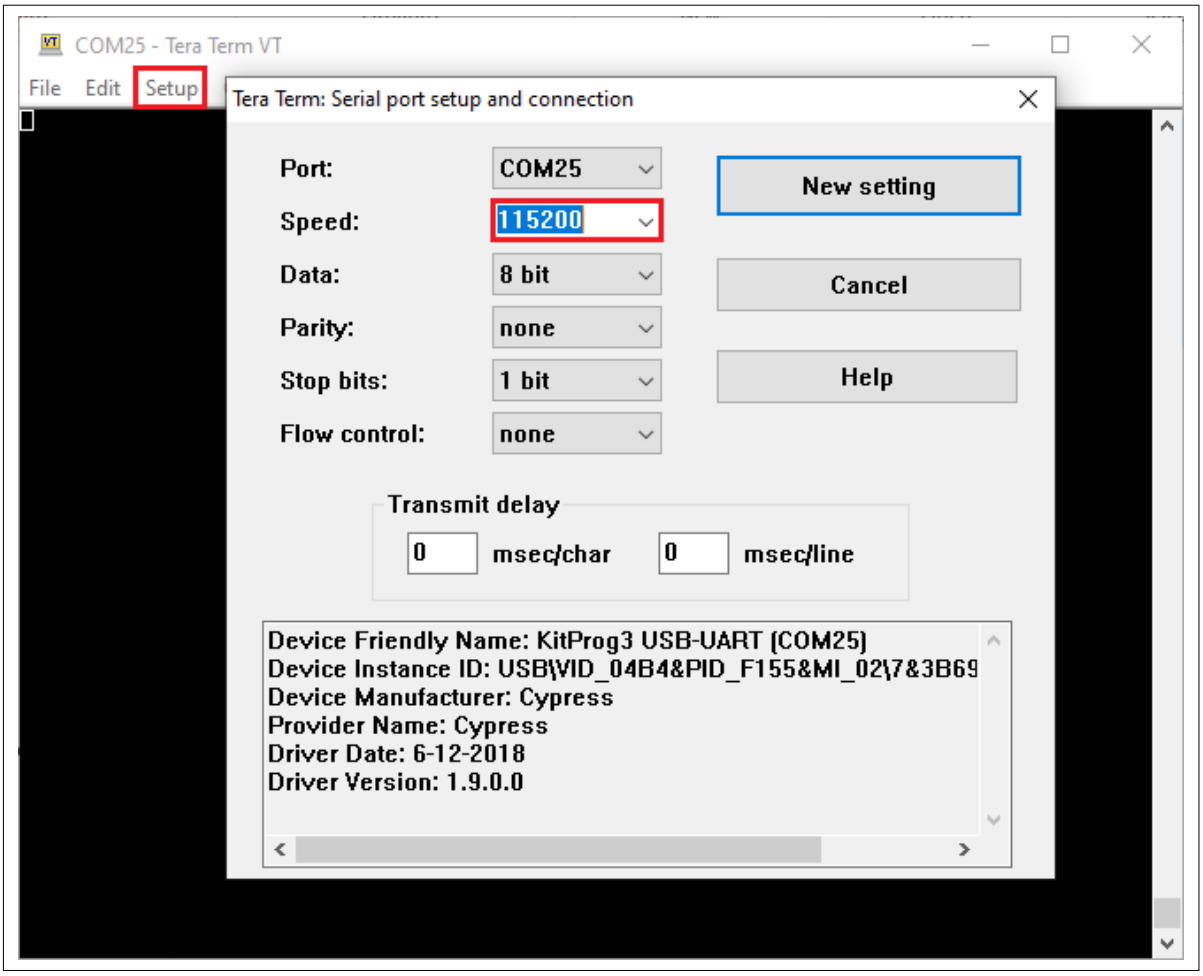

Figure 23 Configuring the baud rate in Tera Term

3. Reset the device

Press the reset switch (**SW1**) on the kit. A message appears on the terminal as Figure 24 shows. The user LED on the kit will start blinking.

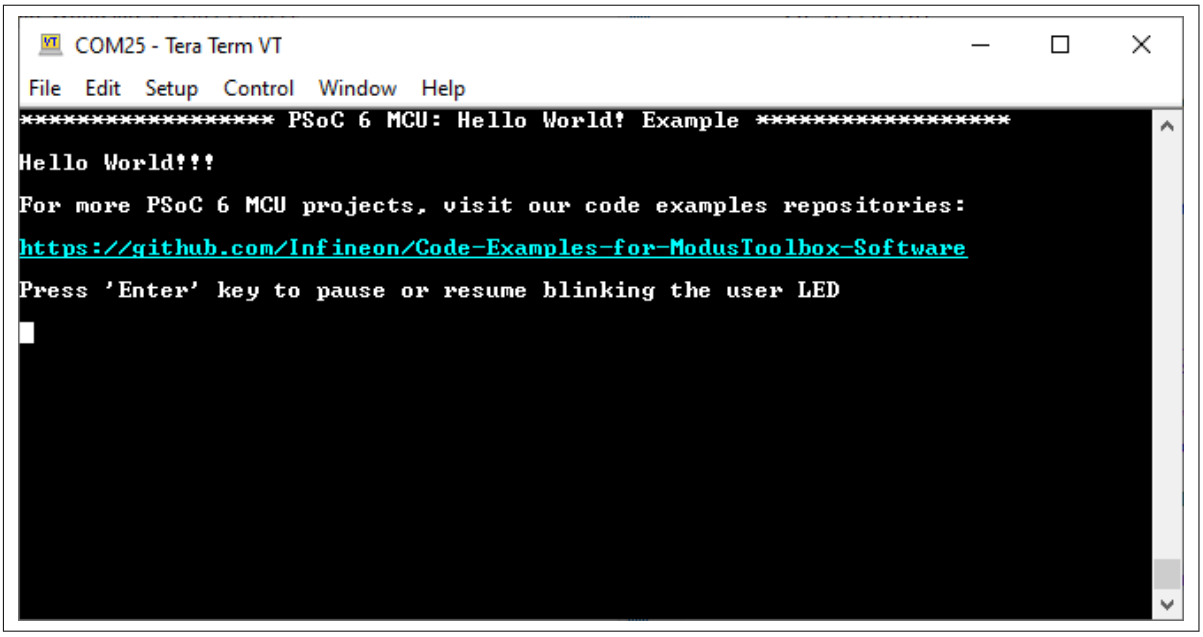

Figure 24 UART message printed from CM4 core

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 25 shows.

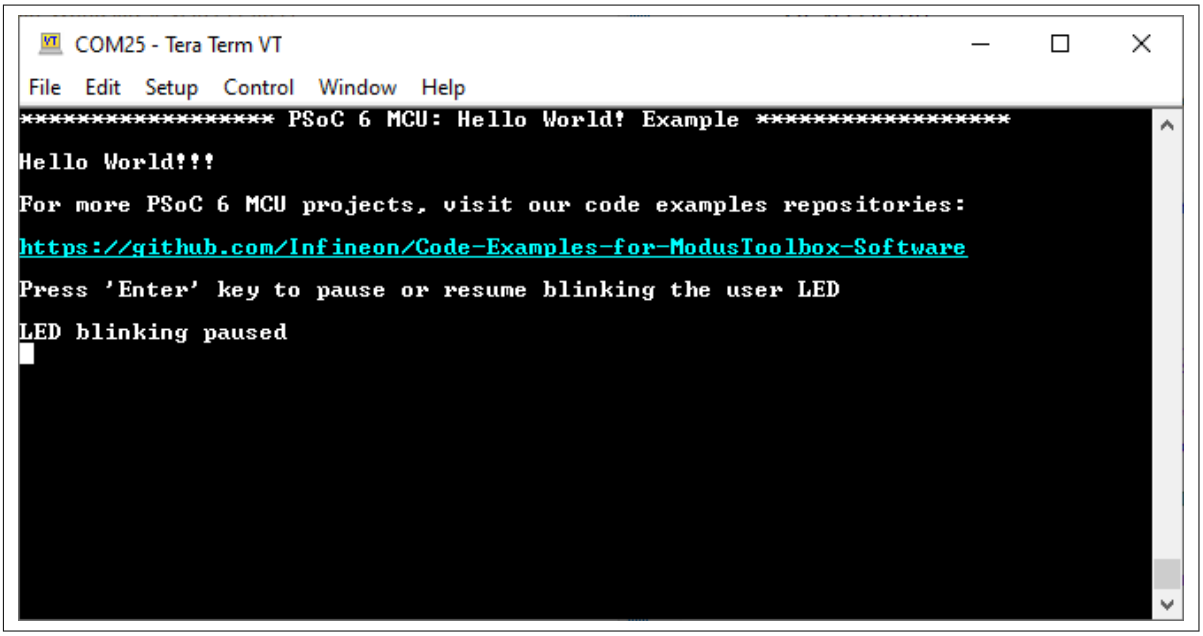

Figure 25 UART message from CM4 core

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-core 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

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
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-core system design
AN217666	PSoC™ 6 MCU interrupts
AN235279	Performing ETM and ITM Trace on PSoC 6 MCU
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

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 (PDL): 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

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 22 to Figure 25 Firmware updated to the latest version
*F	2022-07-21	Template update
*G	2022-09-12	Updated the development flow as per MTB v3.0 software Updated link references Updated configurator info Added new Figure 2 and Figure 7 Updated Figure 8 and Figure 14 Added reference to new AN235279 Added a new section for ModusToolbox™ applications Removed reference to deprecated AN225588 Updated Figure 17 to Figure 19

Trademarks

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

Edition 2022-09-12

Published by

Infineon Technologies AG

81726 Munich, Germany

Do you have a question about any aspect of this document?

Email: erratum@infineon.com

Document reference
IFX-ofg1649405242329

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.

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.