

Getting started with PSOC™ Control C1 MCU on ModusToolbox™ software

About this document

Scope and purpose

This application note helps you to explore the PSOC™ Control C1 MCU architecture, development tools, and shows you how to create your first project using the Eclipse IDE for ModusToolbox™ software. Additionally, the application note also guides you to more online resources to accelerate your learning about the PSOC™ Control C1 MCU. The PSOC™ Control C1 MCU is a cost-effective, real-time control microcontroller designed for industrial drives, digital power conversions, and HMI applications.

Intended audience

This document is intended for users who are new to PSOC™ Control C1 MCU and ModusToolbox™ software.

Associated part family

All PSOC™ Control C1 MCU devices.

Software version

[ModusToolbox™ Software](#) 3.8 or above

More code examples? We heard you

To access an ever-growing list of PSOC™ Control C1 MCU code examples using ModusToolbox™, see the [GitHub](#) site.

Table of contents

	About this document	1
	Table of contents	2
1	Introduction	4
1.1	Architecture and product lines	4
1.2	PSOC™ Control C1 MCU features	6
1.3	Target applications	7
2	PSOC™ Control C1 MCU resources	9
3	PSOC™ Control C1 MCU development kits	10
4	PSOC™ Control C1 ecosystem for firmware/application development	11
4.1	Installing the ModusToolbox™ tools package	11
4.2	Choosing an IDE	11
4.3	ModusToolbox™ help	12
5	Getting started with PSOC™ Control C1 MCU design	13
5.1	Prerequisites	13
5.1.1	Hardware	13
5.1.2	Software	13
5.2	Application development instructions	13
5.3	About the design	14
5.4	Create a new application	14
5.4.1	Eclipse IDE for ModusToolbox™	14
5.4.1.1	View and modify the design	18
5.4.1.1.1	Open the Device Configurator	20
5.4.1.1.2	Add retarget-io-psc1 middleware	21
5.4.1.2	Write firmware	22
5.4.1.3	Build the application	27
5.4.1.4	Program the device	28
5.4.1.5	Test your design	30
5.4.1.6	Debugging the application using JLink	32
5.4.2	Visual Studio Code (VS Code) for ModusToolbox™	34
5.4.3	IAR Embedded Workbench for ModusToolbox™	34
5.4.4	Keil µVision for ModusToolbox™	34
6	Summary	35
	References	36
	Glossary	37
	Revision history	38
	Trademarks	39



Table of contents

Disclaimer 40

1 Introduction

1 Introduction

The PSOC™ Control C1 device is a microcontroller targeted at industrial applications. The PSOC™ Control C1 MCU integrates the following features on a single chip:

- Single core 48 MHz 32-bit Arm® Cortex®-M0 CPU with
 - Nested vectored interrupt controller (NVIC)
 - Math coprocessor (MATH) for coordinate rotation digital computer (CORDIC) and division operations
 - Up to 200 KB flash with ECC support
 - 16 KB SRAM with ECC support
- On-chip flash and SRAM
- Industrial control peripherals: Capture/Compare Units (CCU4 and CCU8) for complex pulse-width modulation (PWM) generation, Position Interface (POSIF) for motor sensors, and window watchdog timer (WDT)
- High-performance analog subsystem: 12-bit, up to 1.1 MS/s SAR ADC with up to 12 analog channels and four high-speed analog comparators
- ModusToolbox™ development environment with installable SDKs and libraries, industry standard Arm® tools, and RTOS support

This application note introduces you to the capabilities of the PSOC™ Control C1 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™ Control C1 MCU. Additionally, it shows you how to create an application from an empty starter application. The completed design of this application is available as a [code example for ModusToolbox™ on GitHub](#).

For hardware design considerations, see the Hardware design guide for the PSOC™ Control C1 MCU family.

1.1 Architecture and product lines

[Figure 1](#) shows a detailed block diagram of the MCU.

1 Introduction

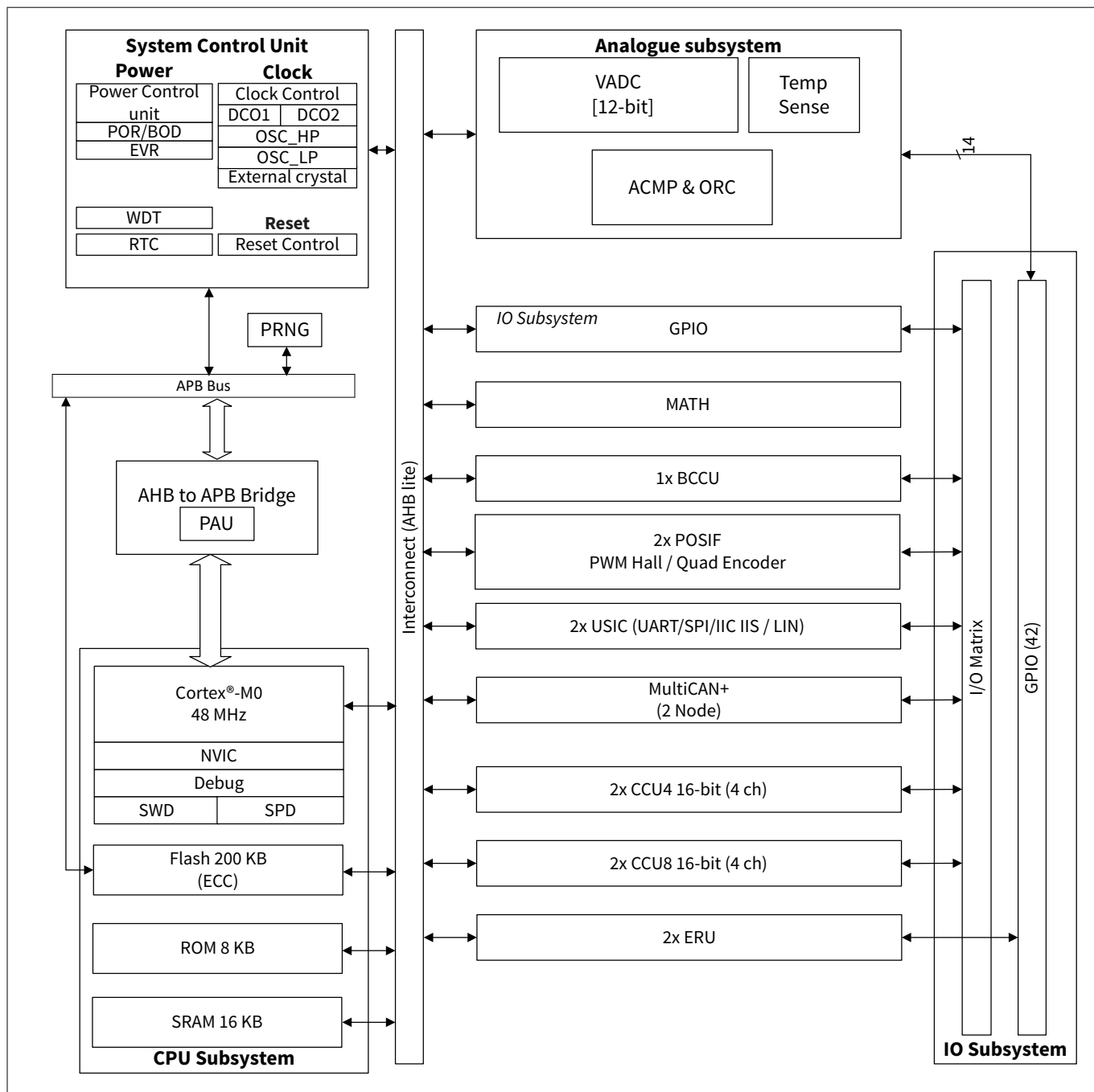


Figure 1 Block diagram

There are two variants in the PSOC™ Control C1 MCU. [Table 1](#) provides an overview of different product lines.

Table 1 PSOC™ Control C1 MCU product lines

Product line	Device series	Details
Entry line	PSC1M2x	Single-core 48 MHz Arm® Cortex®-M0, up to 200 KB flash, 16 KB SRAM, 12-bit ADC (up to 1.1 MS/s)
	PSC1M3x	Single-core 48 MHz Arm® Cortex®-M0, up to 200 KB flash, 16 KB SRAM, 12-bit ADC (up to 1.1 MS/s), Multi-CAN+

1 Introduction

1.2 PSOC™ Control C1 MCU features

PSOC™ Control C1 MCUs have extensive features as shown in [Figure 1](#). The following is a list of major features. For more information, see the device datasheet, reference manuals, and the [References](#) section.

- CPU subsystem
 - 32-bit Arm® Cortex®-M0 CPU core: 0.84 DMIPS/MHz (Dhrystone 2.1), running at up to 48 MHz
 - 3-stage pipeline von Neumann architecture with 32-bit hardware multiplier
 - Nested vectored interrupt controller (NVIC) with 64 interrupt nodes and 4 priority levels
 - MATH coprocessor:
 - Divider unit (DIV): Signed and unsigned 32-bit division
 - CORDIC coprocessor: Trigonometric, linear, and hyperbolic functions
 - 2×4 channels event request unit (ERU) for programmable event interconnections
 - On-chip debug: 4 breakpoints, 2 watchpoints, SWD (2-pin), and SPD (1-pin) debug interfaces
- On-chip memories
 - Up to 200 KB on-chip flash (ECC error correction): Word reads, block writes, page erase (256 B) or sector (4 KB), up to 50,000 erase cycles per page
 - 16 KB SRAM with parity protection
 - 8 KB Boot ROM (BROM): System initialization, bootstrap loaders (BSL), user routines
- Supply, reset, and clock
 - 1.8 V to 5.5 V operating supply range
 - On-chip power-on reset (POR) and brownout detector (BOD)
 - Clock sources:
 - DCO1: 48 MHz internal oscillator (range: 188 kHz to 96 MHz)
 - DCO2: 32 kHz standby and RTC clock
 - OSC_HP: External crystal/clock input (4 MHz to 48 MHz)
 - OSC_LP: External 32.768 kHz watch crystal for RTC
 - On-chip clock monitor; clock gating, and blanking for low-power operation
- System control
 - Window watchdog timer (WDT)
 - Real-time clock (RTC) module
 - Pseudo random number generator (PRNG)
 - System control unit (SCU): general system control, power management control, clock control, and reset control
- Communication peripherals
 - Two Universal Serial Interface (USIC) channels, each configurable as:
 - UART: Up to 12 Mb/s
 - Single-SPI: Up to 12 Mb/s
 - Double-SPI: Up to 2 x 12 Mb/s
 - Quad-SPI: Up to 4 x 12 Mb/s
 - I²C (IIC): Up to 400 kb/s
 - I²S (IIS): Up to 12 Mb/s
 - LIN: Up to 20 kb/s
 - MultiCAN+ (PSC1M3x only): Full-CAN/basic-CAN with 2 nodes, 32 shared message objects, up to 1 MBaud, ISO11898-1 compliant

1 Introduction

- Analog front-end peripherals
 - Versatile ADC (VADC): 12-bit SAR ADC, up to 12 analog inputs, 2 sample and hold stages, up to 1.1 MS/s, adjustable gain, 0 V to 5.5 V input range
 - Up to 8 out-of-range comparators (ORC) for overvoltage protection (OVP) on ADC inputs
 - Up to 4 fast analog comparators (ACMP)
 - On-chip temperature sensor (DTS)
- Industrial control peripherals
 - CCU4: 2×4 ch 16-bit timers at 96 MHz: General-purpose timers, signal monitoring, and PWM
 - CCU8: 2×4 ch 16-bit timers at 96 MHz: Complex PWM, complementary high/low-side switches, multi phase control
 - POSIF (×2): Position interface unit for Hall sensors and quadrature encoders
 - BCCU: 9-channel brightness and color control unit (sigma-delta) for LED lighting applications
- I/O and packages
 - Up to 43 GPIOs (1.8 V to 5.5 V capable); up to 8 high-current pads (50 mA sink)
 - Packages: TSSOP-38 (9.7 mm × 6.4 mm), VQFN-40/48 (5 mm × 5 mm/7 mm × 7 mm/8 mm × 8 mm), LQFP-48 (7 mm × 7 mm/12 mm × 12 mm)
- Programming and debug
 - Single-pin bootstrap loader (ASC BSL/CAN BSL/SSC BSL)
 - Secure bootstrap loader (SBSL, optional)
 - 2-pin SWD and 1-pin SPD debug interfaces
 - ModusToolbox™ toolchain: PDL drivers, configurators, and example applications

1.3 Target applications

The PSOC™ Control C1 MCU is ideally suited for cost-sensitive, real-time control applications that require compact packages, and a wide supply range. Typical applications include:

- **Motor control and automation:** Industrial drives, power tools, home appliances, light electric vehicles
- **Digital power conversion:** Switched-mode power supplies (SMPS), EV charging, renewable energy systems
- **Light electric vehicle solutions:** E-bikes, scooters, and other small electric vehicles
- **Power tools and appliance control:** Cordless tools, kitchen appliances, consumer devices
- **LED lighting:** Smart lighting systems, LED drivers, lighting control
- **Industrial automation and robotics:** Motor control, robotics, factory automation

The ModusToolbox™ software environment supports PSOC™ Control C1 MCU application development with a set of tools for configuring the device, setting up peripherals, and complementing your projects with middleware. See the Infineon GitHub repos for board support packages (BSPs) for all kits, libraries for popular functionality like motor control and power stage converter, and a comprehensive array of example applications to get you started.

Figure 2 shows an application-level block diagram for a real-world use case using PSOC™ Control C1 MCU.

1 Introduction

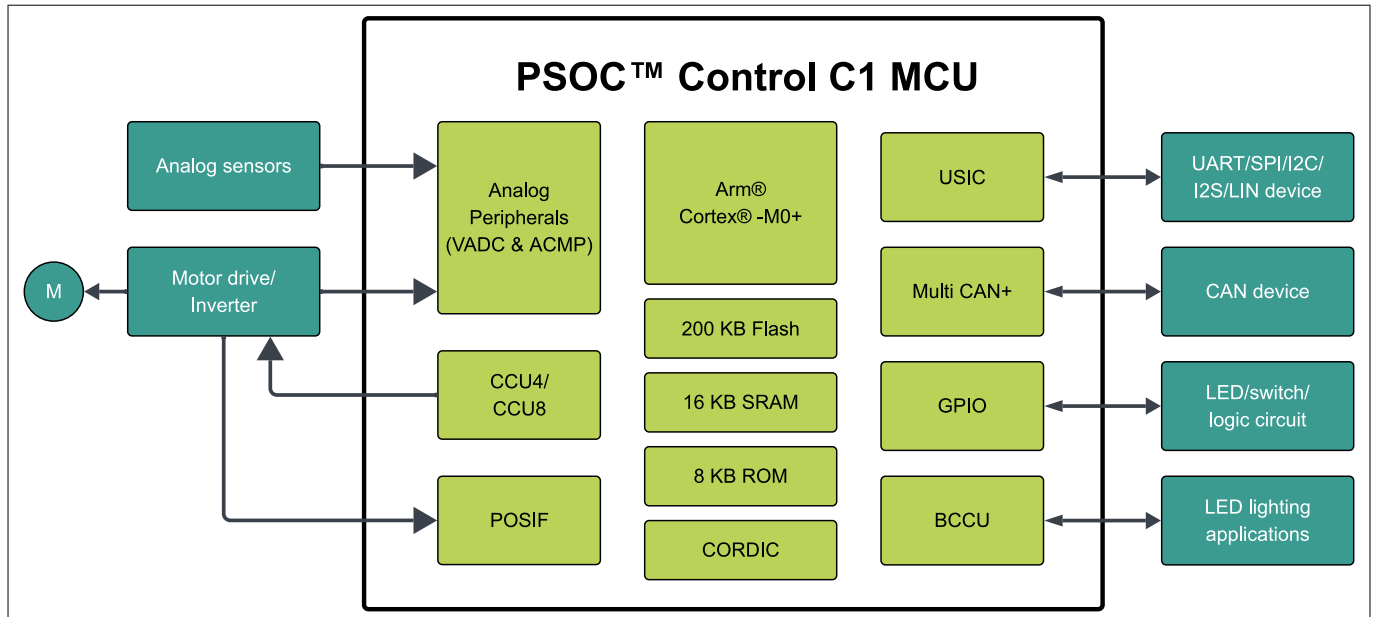


Figure 2 Application-level block diagram using PSOC™ Control C1 MCU

PSOC™ Control C1 MCU is a highly capable and flexible solution for industrial applications. For example, the real-world use case in the [Figure 2](#) takes advantage of the following features:

- CCU8 to drive the gate signals of a 3-phase motor inverter
- VADC for reading analog current and voltage feedback from the inverter
- POSIF to decode rotor position from Hall sensors or quadrature encoder
- CORDIC coprocessor for fast trigonometric calculations in FOC algorithms
- USIC (UART/SPI/I²C) for interfacing with HMI, sensors, or communication peripherals
- Multi-CAN+ (PSC1M3x) for CAN-based communication in industrial networks
- BCCU for LED indicator lighting on the control board

2 PSOC™ Control C1 MCU resources

A wealth of technical resources is available to develop applications with the PSOC™ Control C1 MCU. These resources are as follows:

- **Overview:** PSOC™ Control C1 MCU webpage
- **Product selectors:** PSOC™ Control C1 MCU
- **Datasheet:** Describes the device family and provides electrical specifications
- **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
- **Reference manuals:** Architecture and registers manuals provide detailed descriptions of the architecture and registers in each device family
- **Development tools:** Many low-cost kits are available for evaluation, design, and development of different applications using PSOC™ Control C1 MCUs
- **Technical support:** PSOC™ Control C1 MCU community forum

3 PSOC™ Control C1 MCU development kits

3 PSOC™ Control C1 MCU development kits

Infineon provides a wide variety of hardware development kits in various form factors and features to enable easy and rapid evaluation and prototyping of PSOC™ Control C1 MCU-based applications.

The following table lists the PSOC™ Control C1 MCU kits with various features.

ModusToolbox™ software is the software development platform for creating embedded applications using the development kits.

Table 2 Development kits

Kit	Product	Kit name	BSP GitHub repo
KIT_PSC1M3_EVK	PSC1M3x	PSOC™ Control C1 Evaluation Kit	BSP
KIT_PSC1M2_M1	PSC1M2x	PSOC™ Control C1 MADK Control Card	BSP

For the complete list of kits for the PSOC™ Control C1 MCU along with the shield modules, see the [PSOC™ Control C1 MCU](#) webpage.

4 PSOC™ Control C1 ecosystem for firmware/application development

Infineon provides the ModusToolbox™ software for firmware/application development on PSOC™ Control C1 MCUs. ModusToolbox™ software is a modern, extensible development ecosystem supporting a wide range of Infineon microcontroller devices, including PSOC™ Arm® Cortex® microcontrollers, TRAVEO™ T2G Arm® Cortex® microcontrollers, XMC™ industrial microcontrollers, AIROC™ Wi-Fi devices, AIROC™ Bluetooth® devices, and USB-C Power Delivery microcontrollers. 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.

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

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

4.1 Installing the ModusToolbox™ tools package

The ModusToolbox™ tools package can be installed using any of the below methods:

- **ModusToolbox™ Setup program:** It is highly recommended to use the ModusToolbox™ Setup program for a simplified installation experience. You can download and use the ModusToolbox™ Setup program from [Infineon Development Center](#) to install the tools package, as well as many other packages, such as early access packs, programming tools, etc. See the [ModusToolbox™ software installation guide](#) for more details
- **Standalone installation:** You can download just the ModusToolbox™ tools package from the [Infineon Development Center](#) for offline installation. For more details, see the [ModusToolbox™ tools package installation guide](#)

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

4.2 Choosing an IDE

The ModusToolbox™ software is supported across Windows, Linux, and macOS platforms. The ModusToolbox™ software supports third-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 convenience. The tools support all PSOC™ Control C1 MCUs. The associated BSP and library configurators also work on all three host operating systems.

Beginning with the ModusToolbox™ tools package version 3.8.0, the Eclipse IDE for ModusToolbox™ is no longer included by default. Instead, it is a separate package that you can install via the ModusToolbox™ Setup program.

4 PSOC™ Control C1 ecosystem for firmware/application development

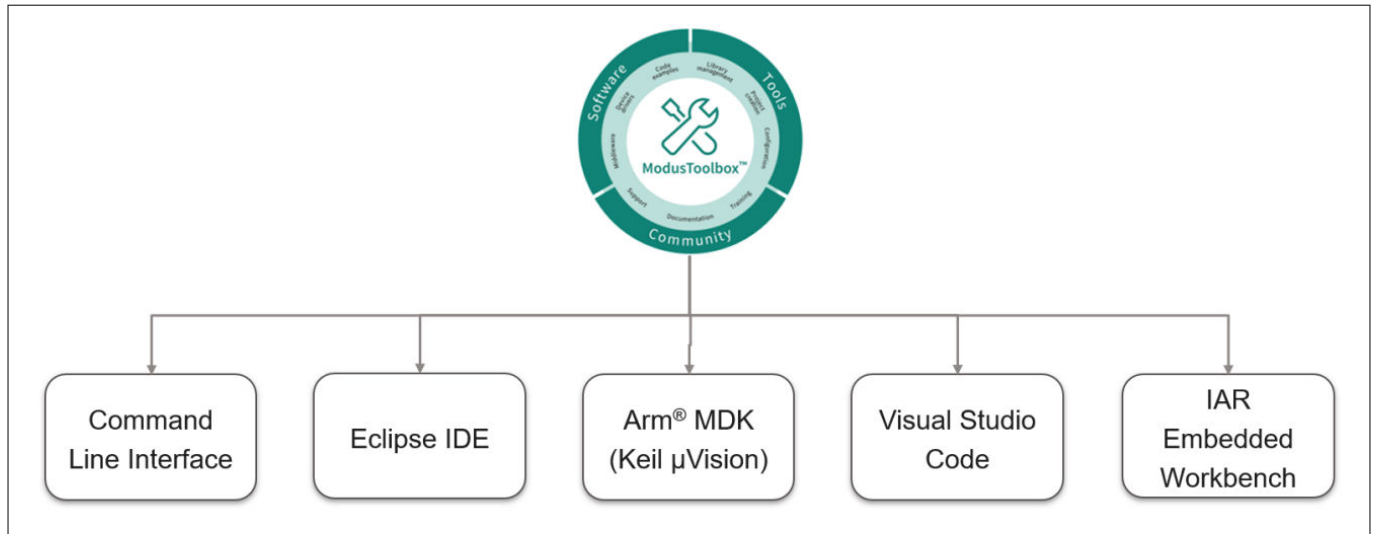


Figure 3 ModusToolbox™ environment

4.3 ModusToolbox™ help

The ModusToolbox™ ecosystem provides documentation and training. Launch the Eclipse IDE for ModusToolbox™ software and navigate 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 of 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. Additionally, it covers various aspects of the tools installed along with the IDE
- **ModusToolbox™ Training Material:** Links to the training material are available at <https://github.com/Infineon/training-modustoolbox>
- **Release Notes:** Describes the features and known limitations for the ModusToolbox™ software, provided as part of the ModusToolbox™ tools package included with the installer

For documentation on Eclipse IDE for ModusToolbox™, choose **Help > Eclipse IDE for ModusToolbox™ documentation**.

- **User Guide:** Provides descriptions about creating applications as well as building, programming, and debugging them using the Eclipse IDE
- **Survival Guide:** This is a link to a forum with answers for questions about how to get common tasks done

5 Getting started with PSOC™ Control C1 MCU design

This section provides the following:

- Demonstrates how to build a simple PSOC™ Control C1 MCU-based design and program into the development kit
- Makes it easy to learn PSOC™ Control C1 MCU design techniques and how to use the ModusToolbox™ software and Eclipse IDE

5.1 Prerequisites

Before you get started with the application development instructions, ensure that you have the appropriate development kit for your PSOC™ Control C1 MCU product line and have installed the required software. Additionally, you need the Internet to access the GitHub repositories during project creation.

5.1.1 Hardware

The following design example is developed for the [PSOC™ Control C1M3 Evaluation Kit \(KIT_PSC1M3_EVK\)](#). However, you can build the application for other development kits also. For more details, see the section [Application development instructions](#).

5.1.2 Software

Following are the software prerequisites:

- [ModusToolbox™ software](#) v3.8 or above.
See [Installing the ModusToolbox™ tools package](#) for detailed installation instructions
- [SEGGER J-Link software](#)
- [Arm® GNU Toolchain \(GCC\) v14.2.1](#)

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

Note: *If you install the ModusToolbox™ tools package using the ModusToolbox™ Setup program, the GCC Toolchain is installed automatically as required dependencies. If you instead get the tools package from the Infineon Developer Center, you must download and install these dependencies manually.*

5.2 Application development instructions

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

1. [Create a new application](#)
2. [View and modify the design](#)
3. [Write firmware](#)
4. [Build the application](#)
5. [Program the device](#)
6. [Test your design](#)

This design is developed for the [PSOC™ Control C1M3 Evaluation Kit](#). You can use other supported kits to test this example by selecting the appropriate kit while creating the application.

5.3 About the design

This design uses the PSOC™ Control C1 MCU to execute two tasks: UART communication and LED control. After device reset/power cycle, the CPU uses the USIC UART peripheral to print a "Hello World" message to the serial port and starts blinking the user LED on the kit. When you press the 'Enter' key on the serial console, the LED blinking is paused or resumed.

5.4 Create a new application

This section takes you on a step-by-step guided tour of the new application creation process. It uses the **Empty App** starter application and manually adds the functionality from the **Hello World** starter application.

As mentioned in section [Choosing an IDE](#), ModusToolbox™ software supports the following third-party IDEs:

- [Eclipse IDE for ModusToolbox™](#)
- [Visual Studio Code \(VS Code\) for ModusToolbox™](#)
- [IAR Embedded Workbench for ModusToolbox™](#)
- [Keil µVision for ModusToolbox™](#)

The following sections provide details on how to create a new application on different IDEs.

5.4.1 Eclipse IDE for ModusToolbox™

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 Dashboard 3.8 application to get started.

Note: *Dashboard 3.8 application needs access to the Internet to successfully clone the starter application onto your machine.*

The Dashboard 3.8 application helps you to get started using the various tools with easy access to documentation and training material, a simple path for creating applications and creating and editing BSPs.

1. Open the Dashboard 3.8 application
To open the Dashboard 3.8 application, do one of these:
 - **Windows:** Navigate to [ModusToolbox installation path]/tools_3.8/dashboard/dashboard.exe or you can also select the "ModusToolbox™ Dashboard 3.8" item from the Windows Start menu
 - **Linux:** [ModusToolbox installation path]/tools_3.8/dashboard and run the executable
 - **macOS:** Run the "dashboard" app
2. On the Dashboard 3.8 window, in the right pane, in the **Target IDE** drop down list, select **Eclipse IDE for ModusToolbox™**, and click **Launch Eclipse IDE for ModusToolbox™**.

5 Getting started with PSOC™ Control C1 MCU design

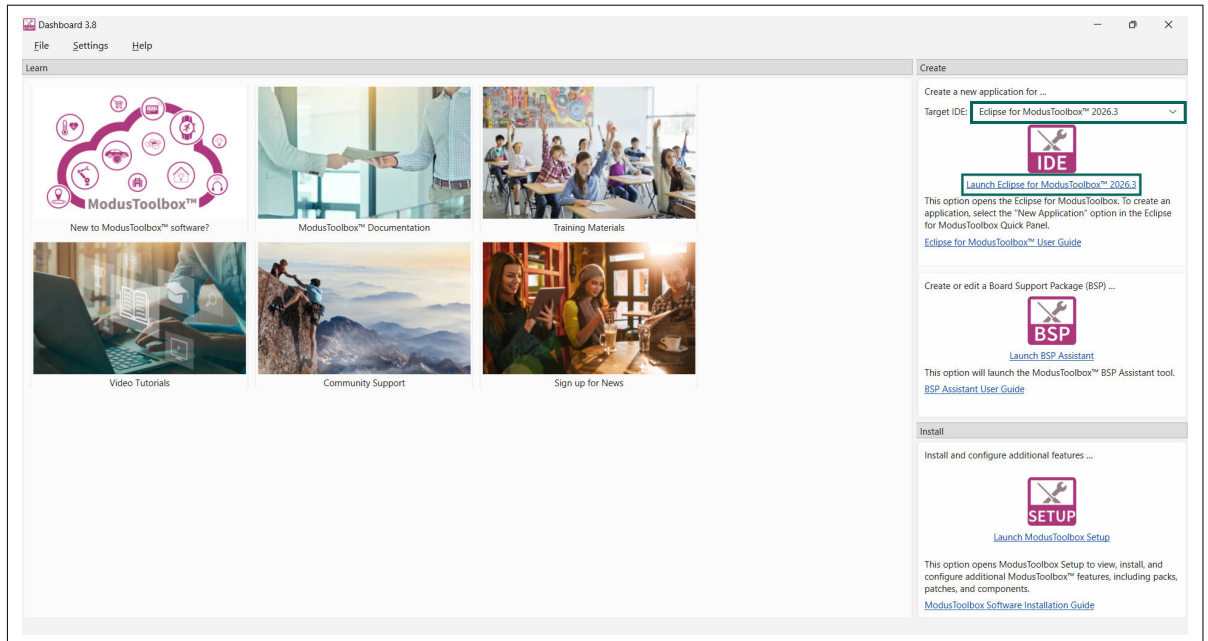


Figure 4 Dashboard 3.8 application

3. Select a new workspace

At launch, Eclipse IDE for ModusToolbox™ displays 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 shown in [Figure 5](#). Alternatively, you can type in a directory name to be used as the workspace directory along with the complete path, and the IDE will create the directory for you.

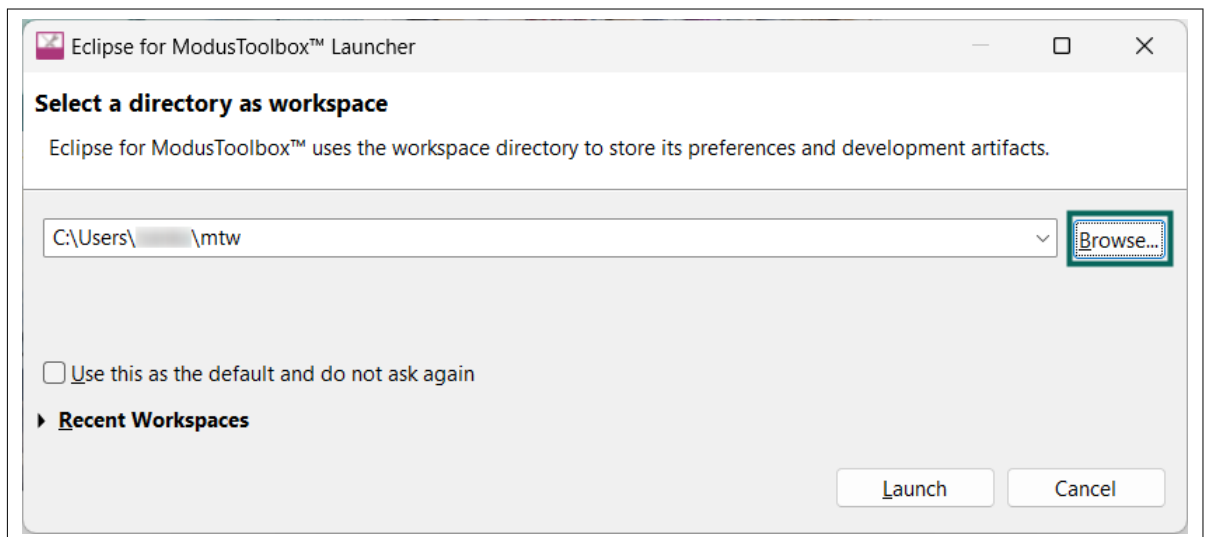


Figure 5 Select a directory as the workspace

4. 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 6](#) shows. Displays the Eclipse IDE for ModusToolbox™ Application window

5 Getting started with PSOC™ Control C1 MCU design

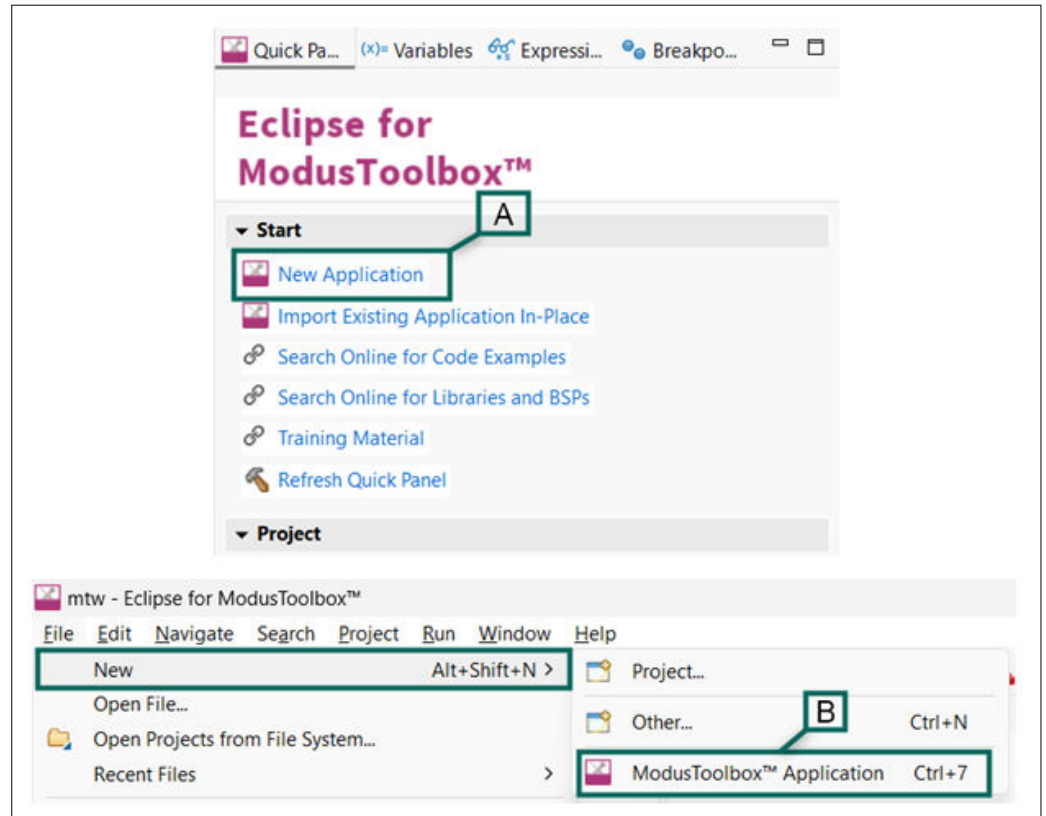


Figure 6 Create a New ModusToolbox™ Application

5. Select a target PSOC™ Control C1M3 Evaluation 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 are followed use **KIT_PSC1M3_EVK**, as shown in [Figure 7](#)
- b. Click **Next**

5 Getting started with PSOC™ Control C1 MCU design

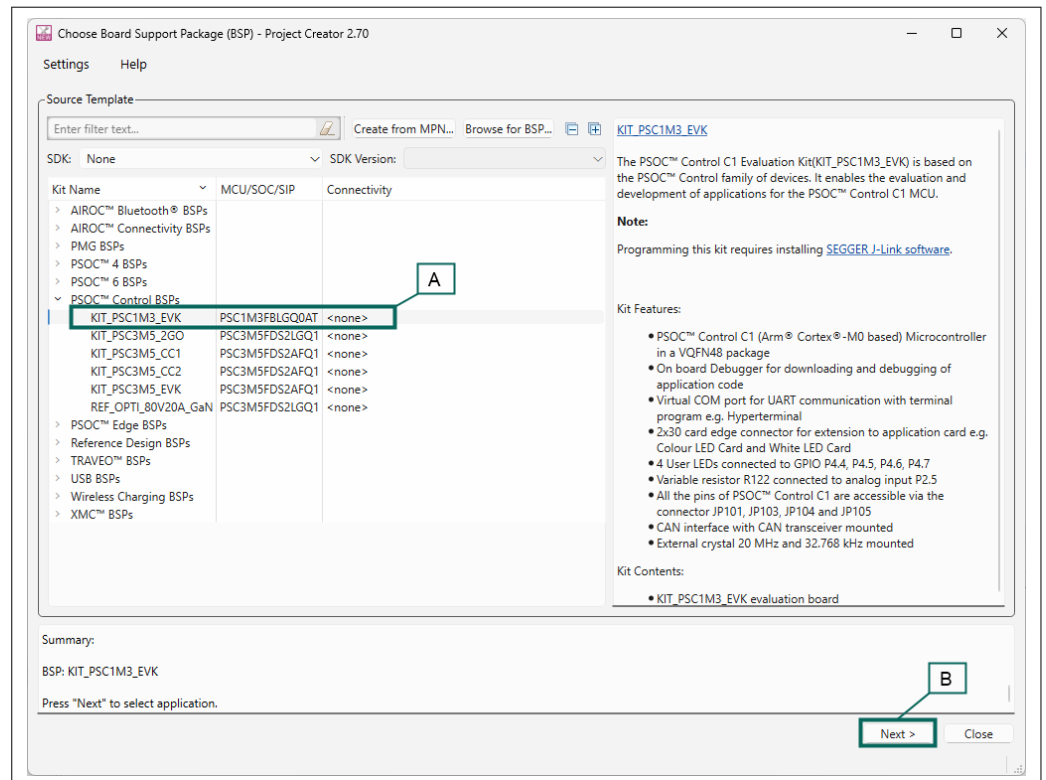


Figure 7 Choose target hardware

- In the **Select Application** dialog, select **Empty App** starter application, as shown in the following [Figure 8](#)
- 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

Note: Try to use a short name without spaces in between

- Click **Create** to create the application, as shown in the following [Figure 8](#), wait for the Project Creator to automatically close once the project is successfully created

5 Getting started with PSOC™ Control C1 MCU design

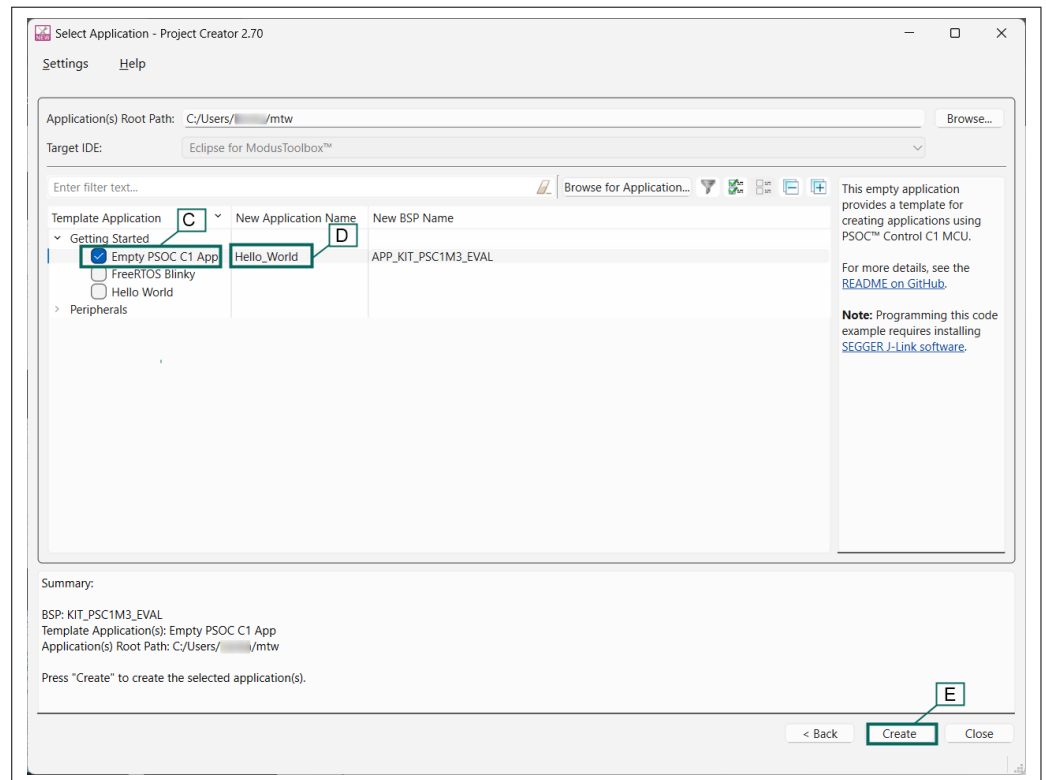


Figure 8 Choose starter application

You have successfully created a new ModusToolbox™ application for a PSOC™ Control C1 MCU.

The BSP uses PSC1M3FBLGQ0AT as the default device that is mounted on the PSOC™ Control C1M3 Evaluation Kit.

If you are using custom hardware based on the PSOC™ Control C1 MCU or a different PSOC™ Control C1 MCU part number, see the "Creating your Own BSP" section in the [ModusToolbox™ user guide](#).

5.4.1.1 View and modify the design

[Figure 9](#) shows the ModusToolbox™ project explorer interface displaying the structure of the application project. PSOC™ Control C1 MCU has a single Cortex® -M0 core. This application note shows firmware development for the Cortex® -M0 core with ModusToolbox™.

5 Getting started with PSOC™ Control C1 MCU design

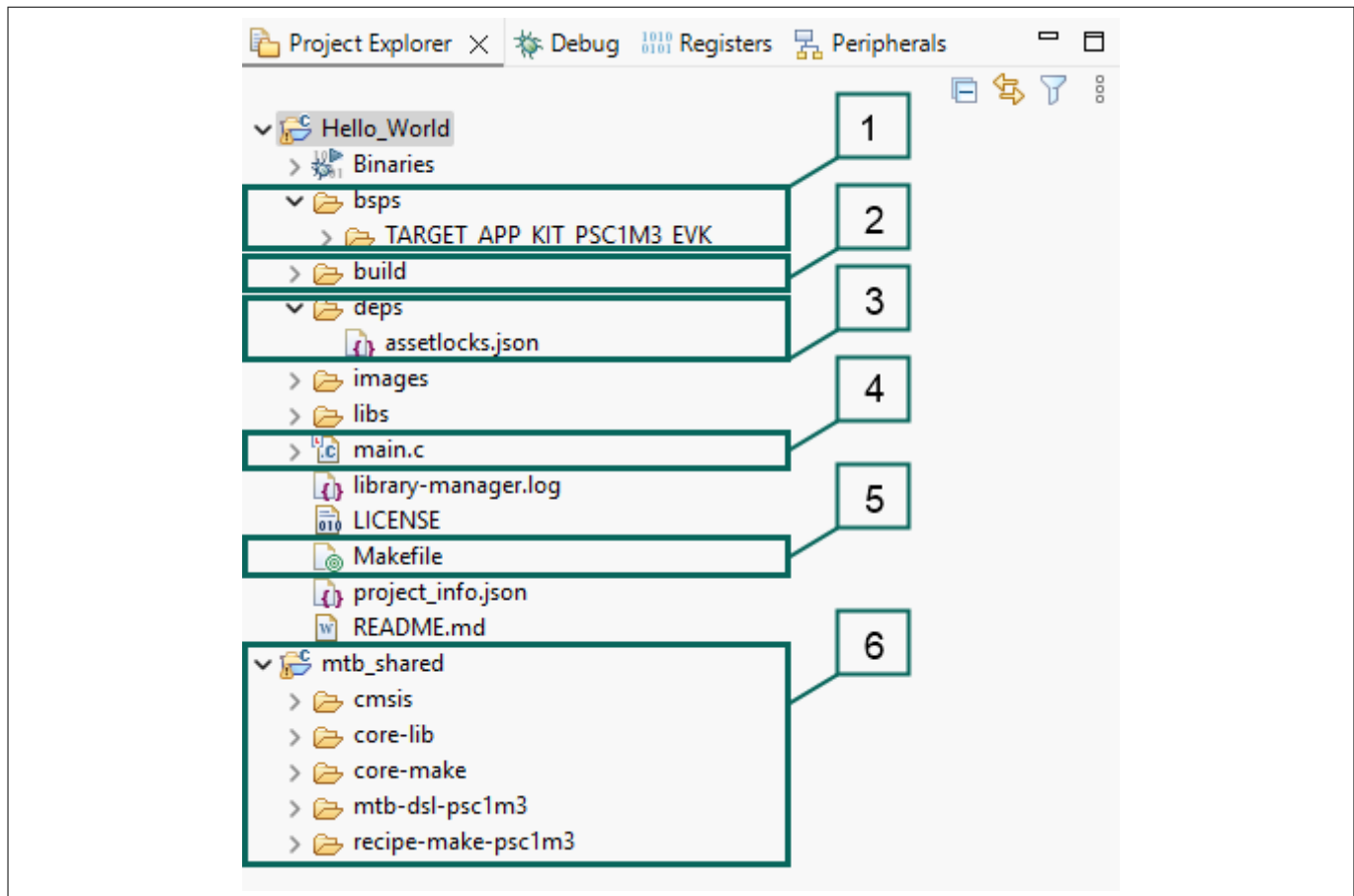


Figure 9 Project Explorer view

A project folder consists of various subfolders, each denoting a specific aspect of the project.

1. The files provided by the BSP are in the `bsps` folder and are listed under `TARGET_` subfolders. 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 `bsps` folder also contains the linker scripts and the start-up code for the PSOC™ Control C1 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-psc1.mtb` points to `mtb://retarget-io-psc1#latest-v1.X#$$ASSET_REPO$$/retarget-io-psc1/latest-v1.X`. The variable `$$ASSET_REPO$$` points to the root of the shared location, which defaults to the `mtb_shared` folder. 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 a `.mtb` file in this folder. These files have been populated based on the targets available in `deps` folder

5 Getting started with PSOC™ Control C1 MCU design

For example, using BSP KIT_PSC1M3_EVK populates the `libs` folder with the following .mtb files: [cmsis.mtb](#), [core-lib.mtb](#), [core-make.mtb](#), [device-db.mtb](#), [mtb-dsl-psc1m3.mtb](#), [recipe-make-psc1m3.mtb](#)

The `libs` folder contains the file `mtb.mk`, which stores the relative paths of 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
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.

5.4.1.1.1 Open the Device Configurator

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

The **Device Configurator** 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 **Universal Serial Interface Channel (USIC)** resource can have **I2C**, **SPI**, or **UART** personalities. The **Name(s)** 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 can 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**. For more details, see the [Device Configurator user guide](#).

[Figure 10](#) also shows that the **USIC0_CH1** is enabled and configured. As the **USIC0_CH1** is used for communicating with the user through the debug UART terminal, replicate the same configuration in your setup for the Hello World application.

5 Getting started with PSOC™ Control C1 MCU design

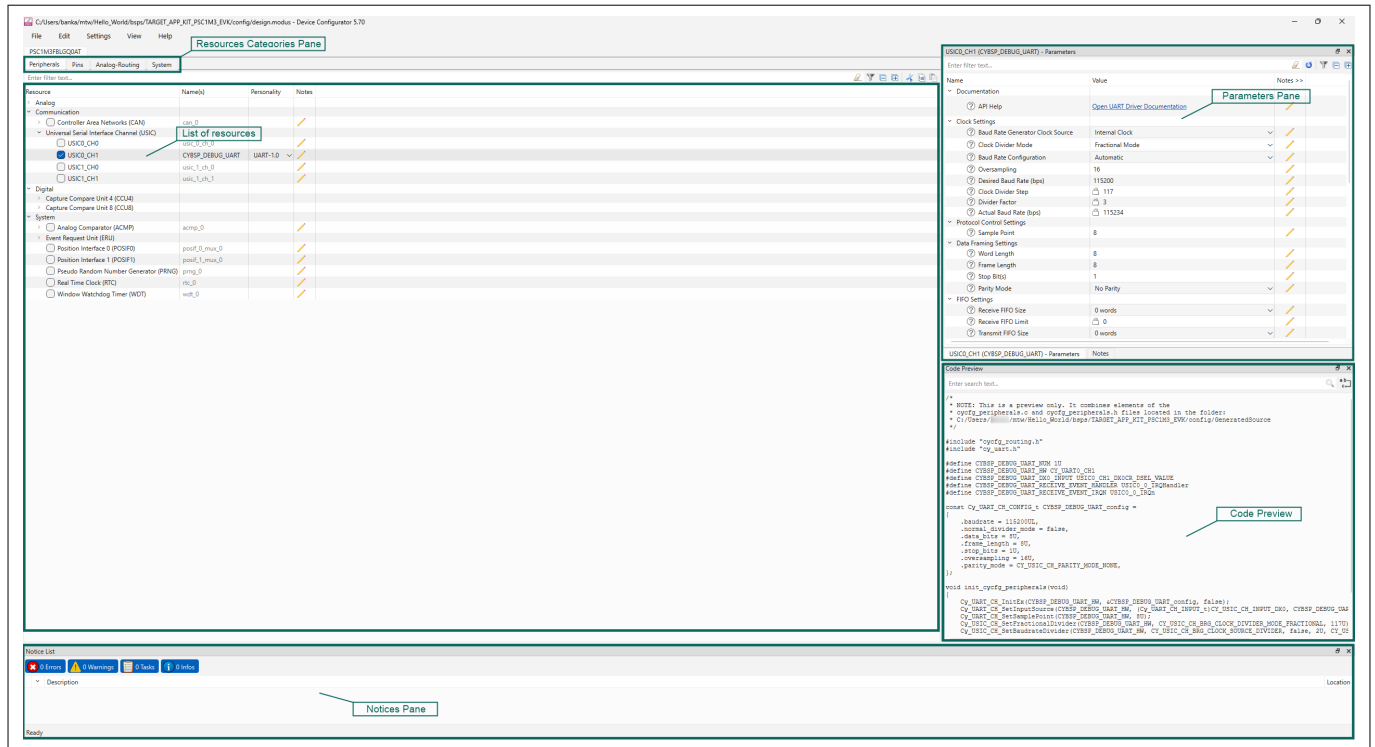


Figure 10 Device Configurator: UART configuration

The application project contains source files that help you create an application for the Cortex®-M0 core (for example, `main.c`). This C file is compiled and linked with the Cortex®-M0 image as part of the normal build process.

At this point in the development process, the required middleware is ready to be added to the design. The only middleware required for the Hello World application is the [retarget-io-psc1](#) library.

5.4.1.1.2 Add retarget-io-psc1 middleware

In this section, you will add the [retarget-io-psc1](#) middleware to redirect standard input and output streams to the UART configured by the BSP. The initialization of the middleware will be done in the `main.c` file.

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-psc1](#)
4. Click **OK** and then **Update**

The files necessary to use the [retarget-io-psc1](#) middleware are added in the `mtb_shared > retarget-io-psc1` folder, and the `.mtb` file is added to the `deps` folder as shown in [Figure 11](#).

5 Getting started with PSoC™ Control C1 MCU design

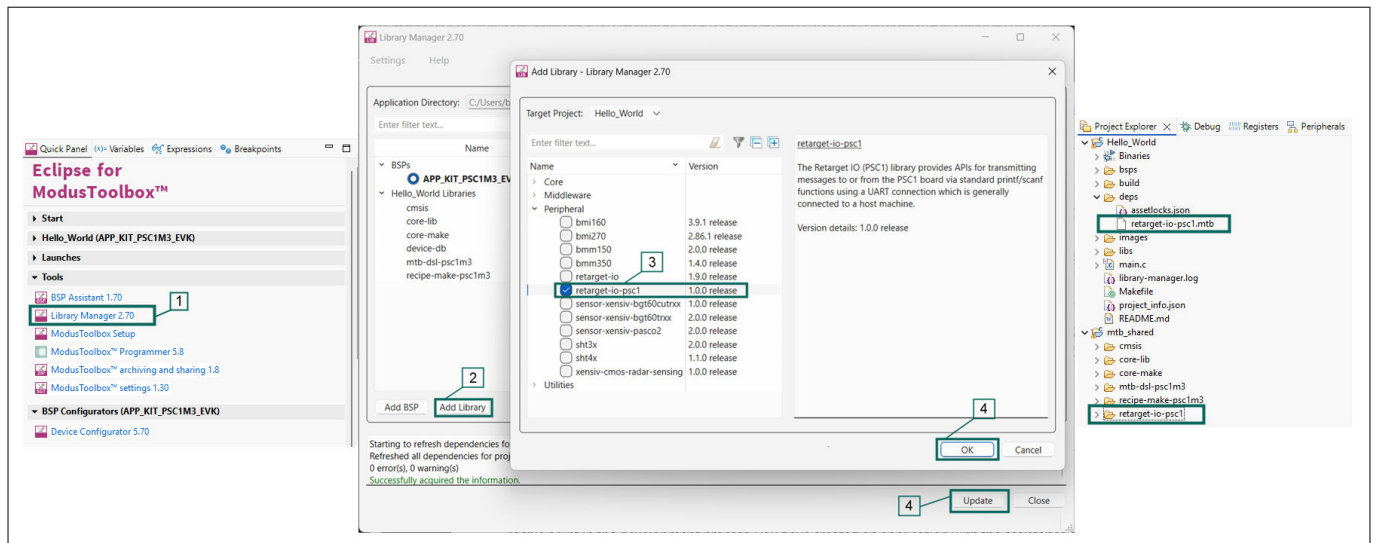


Figure 11 Add the retarget-io-psc1 middleware

5.4.1.2 Write firmware

At this point in the development process, you have created an application with the assistance of an application template, configured the peripherals using the Device Configurator, and modified it to add the [retarget-io-psc1](#) middleware. In this section, you will write the firmware that implements the design functionality.

If you are working from scratch using the Empty PSoC™ Control C1 MCU starter application, you can copy the code snippet provided in this section to the `main.c` file of the application project.

Firmware flow

Examine the code in the `main.c` file of the application. [Figure 12](#) shows the firmware flowchart.

After reset, resource initialization for this example is performed by the Cortex®-M0 CPU. It configures the system clocks, pins, clock to peripheral connections, and other platform resources.

Then the clocks and system resources are initialized by the BSP initialization function. The `retarget-io-psc1` 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 onboard J-Link acts as the USB-to-UART bridge to create the virtual COM port. The SysTick timer is then configured to generate an interrupt every 1 millisecond.

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/PDL functions to execute the intended functionality.

- **cybsp_init():** This BSP function initializes all the system resources of the device, including but not limited to the system clocks and power regulators
- **cy_retargt_io_init():** These functions set up the UART and redirect the input/output stream to the debug UART
- **SysTick_Config():** Configures the Arm® SysTick timer to generate an interrupt every 1 ms
- **SysTick_Handler():** This is the SysTick interrupt service routine (ISR) executed every 1 ms, which increments a global tick counter used for time tracking
- **sys_now():** This inline helper function returns the current tick count in milliseconds
- **Cy_USIC_CH_GetReceiveBufferStatus():** This function is polled in the main loop to check for newly received UART data without blocking execution
- **Cy_UART_CH_GetReceivedData():** This function reads the received character from the UART buffer when data is available

5 Getting started with PSOC™ Control C1 MCU design

- **Cy_GPIO_ToggleOutput():** This function toggles the USER LED pin when LED blinking is enabled and the defined toggle interval has elapsed
- **led_blink_active_flag:** This global volatile flag is controlled by the Enter key input and is evaluated in each main loop iteration to enable or disable LED blinking

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

5 Getting started with PSOC™ Control C1 MCU design

main.c file

```
#include "cybsp.h"
#include "cy_utils.h"
#include "cy_retarget_io.h"

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

/* SysTick timer frequency in Hz */
#define TICKS_PER_SECOND      1000

/* USER LED toggle period in milliseconds */
#define USER_LED_TOGGLE_PERIOD_MS  500

/*****
 * Global Variables
 *****/

/* Variable for keeping track of time */
static volatile uint32_t ticks = 0;

/* Flag to control LED blinking: true = blinking active, false = paused */
static volatile bool led_blink_active_flag = true;

/*****
 * Function Name: SysTick_Handler
 *****/
* Summary:
* This is the interrupt handler function for the SysTick timer interrupt.
* It counts the time elapsed in milliseconds since the timer started.
*
* Parameters:
* none
*
* Return:
* none
*
*****/
void SysTick_Handler(void)
{
    ticks++;
}

/*****
 * Function Name: sys_now
 *****/
* Summary:
* Returns the current time in milliseconds.
*
* Parameters:
* none
```

5 Getting started with PSOC™ Control C1 MCU design

```
*
* Return:
*   the current time in milliseconds
*
*****/
__STATIC_INLINE uint32_t sys_now(void)
{
    return ticks;
}

/*****
* Function Name: main
*****
* Summary:
* This is the main function. It sets up a timer to trigger a
* periodic interrupt. The main while loop checks for the elapsed time
* and toggles an LED at 1 Hz to create an LED blinky.
*
* Parameters:
*   none
*
* Return:
*   int
*
*****/
int main(void)
{
    cy_rslt_t result;

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

    cy_retarget_io_init(CYBSP_DEBUG_UART_HW);

    /* System timer configuration */
    SysTick_Config(SystemCoreClock/TICKS_PER_SECOND);

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

    printf("Hello World!!!\r\n\n");
    printf("For more projects, "
    "visit our code examples repositories:\r\n\n");
    printf("https://github.com/Infineon/"
    "Code-Examples-for-ModusToolbox-Software\r\n\n");

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

    uint32_t last_toggle = sys_now();
```

5 Getting started with PSOC™ Control C1 MCU design

```
while (1)
{
    /* Non-blocking check for received UART character */
    if (Cy_USIC_CH_GetReceiveBufferStatus(CYBSP_DEBUG_UART_HW) != 0U)
    {
        uint16_t ch = Cy_UART_CH_GetReceivedData(CYBSP_DEBUG_UART_HW);
        if (ch == '\r')
        {
            if (led_blink_active_flag)
            {
                printf("LED blinking resumed\r\n");
            }
            else
            {
                printf("LED blinking paused \r\n");
            }
            /* Move cursor to previous line */
            printf("\x1b[1F");
            led_blink_active_flag ^= 1;
        }
    }

    /* Non-blocking LED toggle at USER_LED_TOGGLE_PERIOD_MS rate */
    if (led_blink_active_flag)
    {
        uint32_t now = sys_now();
        if ((now - last_toggle) >= USER_LED_TOGGLE_PERIOD_MS)
        {
            last_toggle = now;
            Cy_GPIO_ToggleOutput(CYBSP_USER_LED_PORT, CYBSP_USER_LED_PIN);
        }
    }
    else
    {
        /* Keep last_toggle current so blinking resumes without a skipped beat */
        last_toggle = sys_now();
    }
}

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

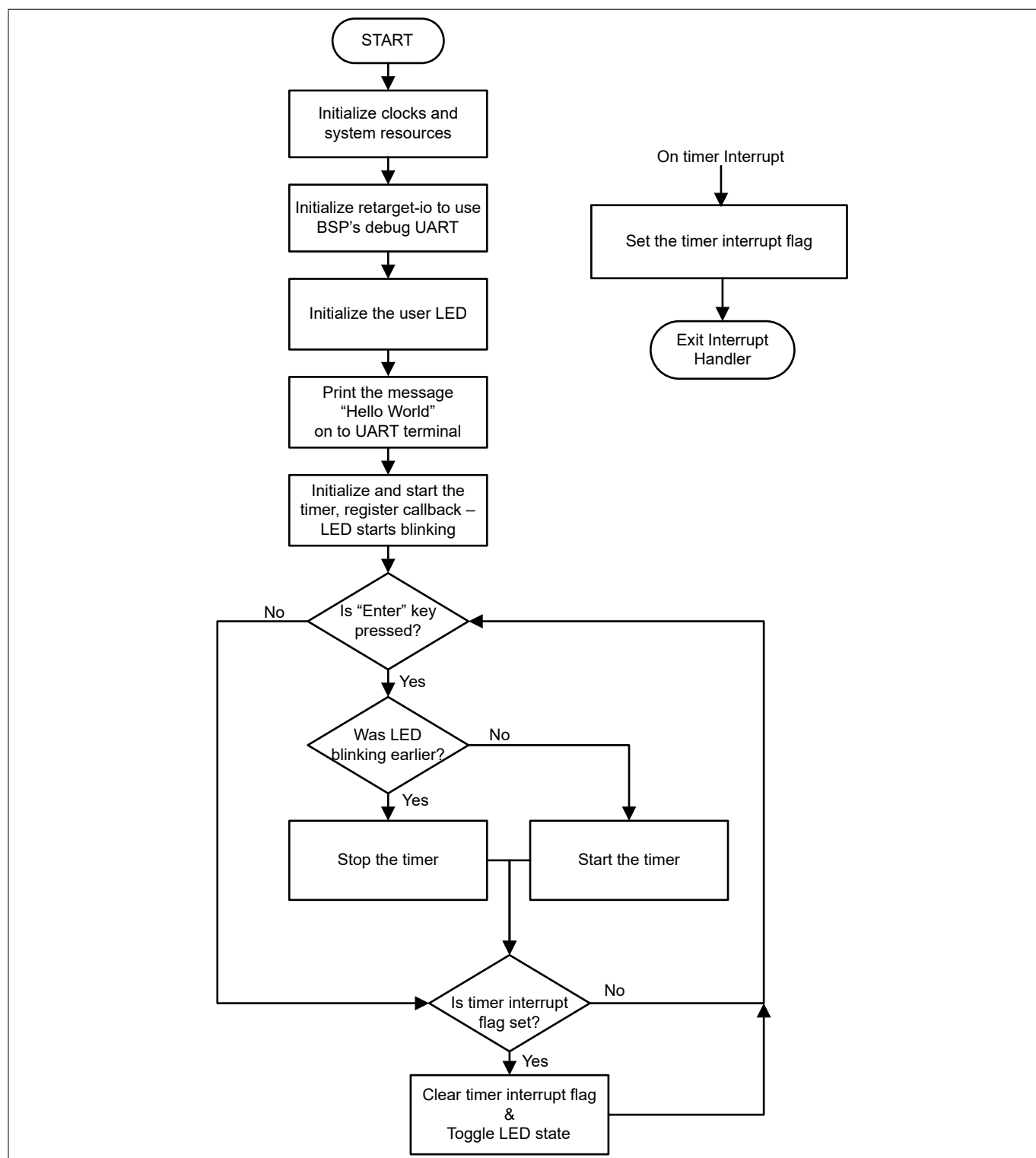


Figure 12 Firmware flowchart

This completes the summary of how the firmware works in the code example. Explore the source files for a deeper understanding.

5.4.1.3 Build the application

This section shows how to build the application.

1. Select the application project in the **Project Explorer** view

5 Getting started with PSoC™ Control C1 MCU design

2. Click **Build Project** under the **Hello_World** group in the **Quick Panel**

It selects the build configuration from the 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 13](#) shows

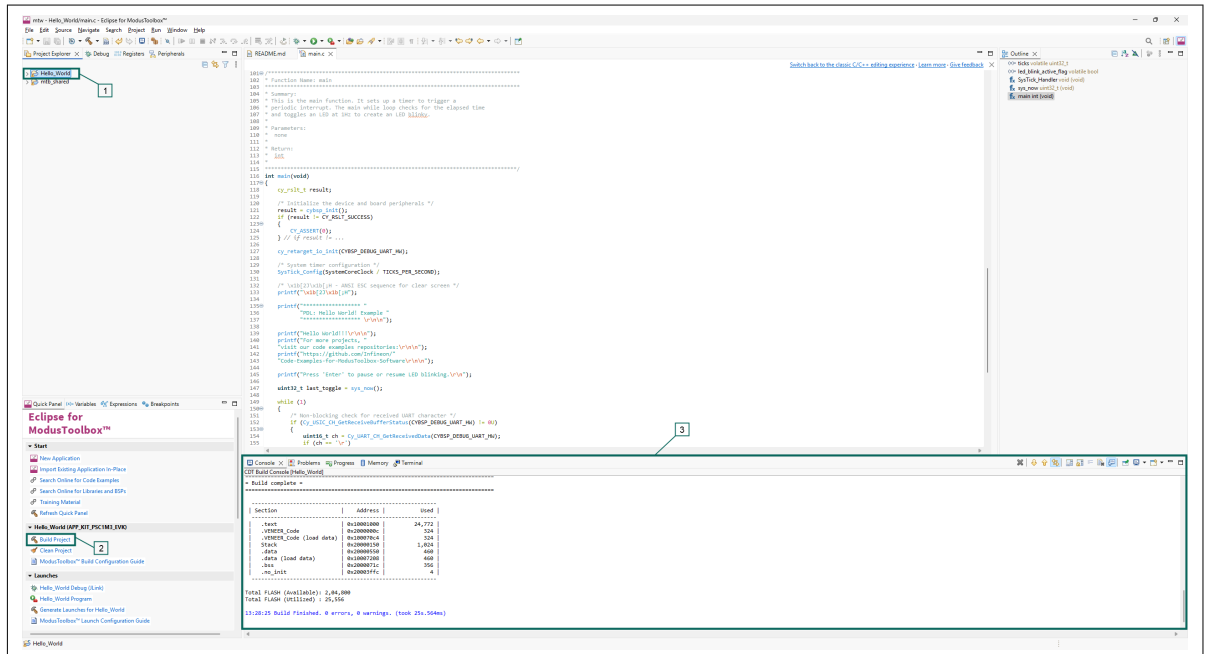


Figure 13 Build the application

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

Note: You can also use the command-line interface (CLI) to build the application. See the **Build system** section in the [ModusToolbox™ tools package user guide](#). This document is located in the /docs_/_ folder in the ModusToolbox™ installation.

5.4.1.4 Program the device

This section shows how to program the PSoC™ Control C1 MCU.

ModusToolbox™ software uses the [Segger J-Link](#) protocol to program and debug applications on PSoC™ Control C1 MCUs on the evaluation kit.

As the evaluation kit is with a built-in programmer, connect the board to your computer using the provided USB cable.

If you are developing on your own hardware, you may need a hardware programmer/debugger, for example, a J-Link, or ULinkpro, or MiniProg.

1. Program the application

- a. Select the application project and click on the **Hello_World Program (JLink)** under the **Launches** group in the Quick Panel, as [Figure 14](#) 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

5 Getting started with PSOC™ Control C1 MCU design

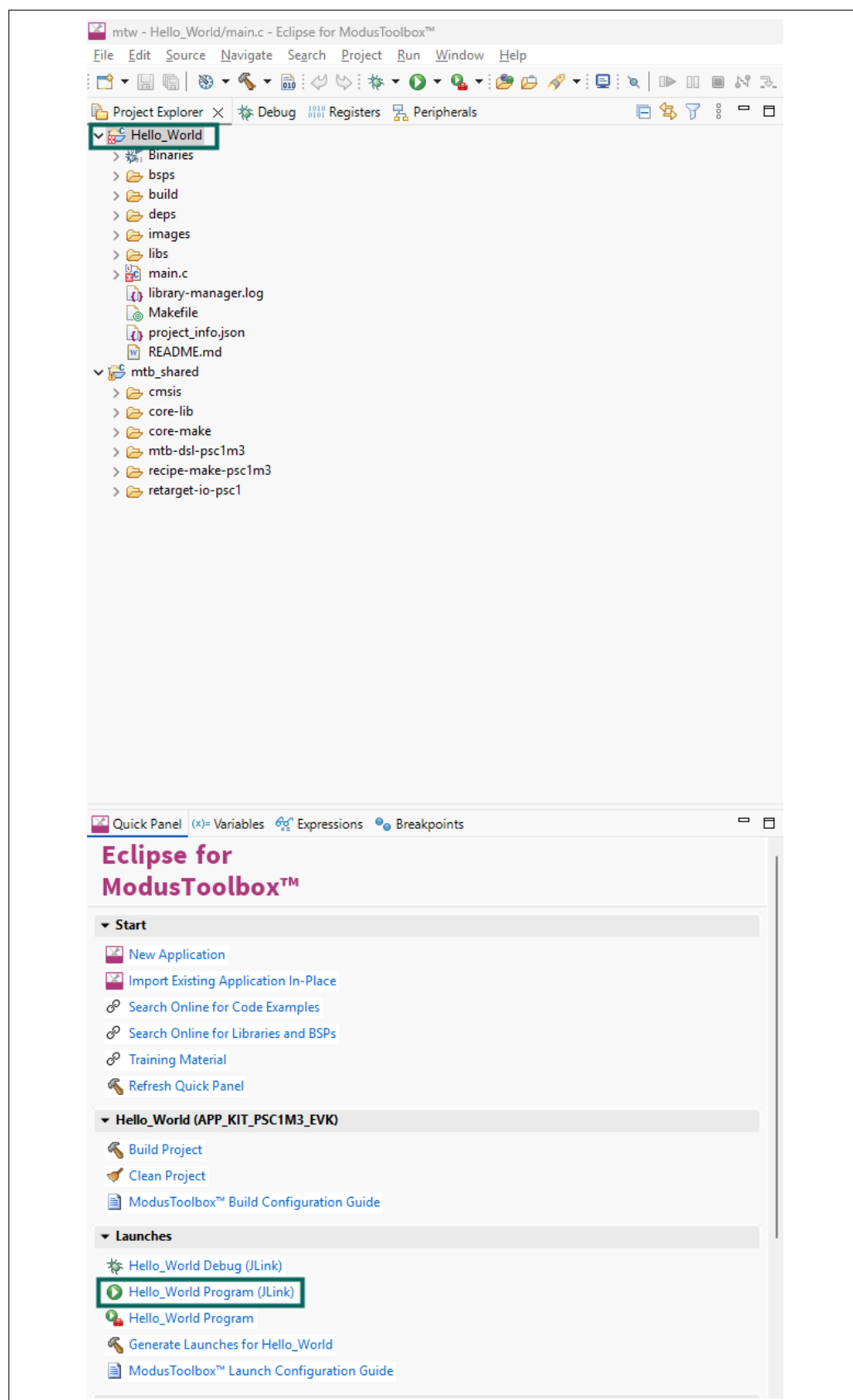


Figure 14 Programming an application to a device

The **Console** view lists the results of the programming operation, as [Figure 15](#) shows.

5 Getting started with PSoC™ Control C1 MCU design



Figure 15 Console: Programming results

5.4.1.5 Test your design

This section describes how to test your design.

Follow these steps to observe the output of your design. This application uses Tera Term as the UART terminal emulator to view the results, but 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 16](#) shows. Note that your COM port number may be different.

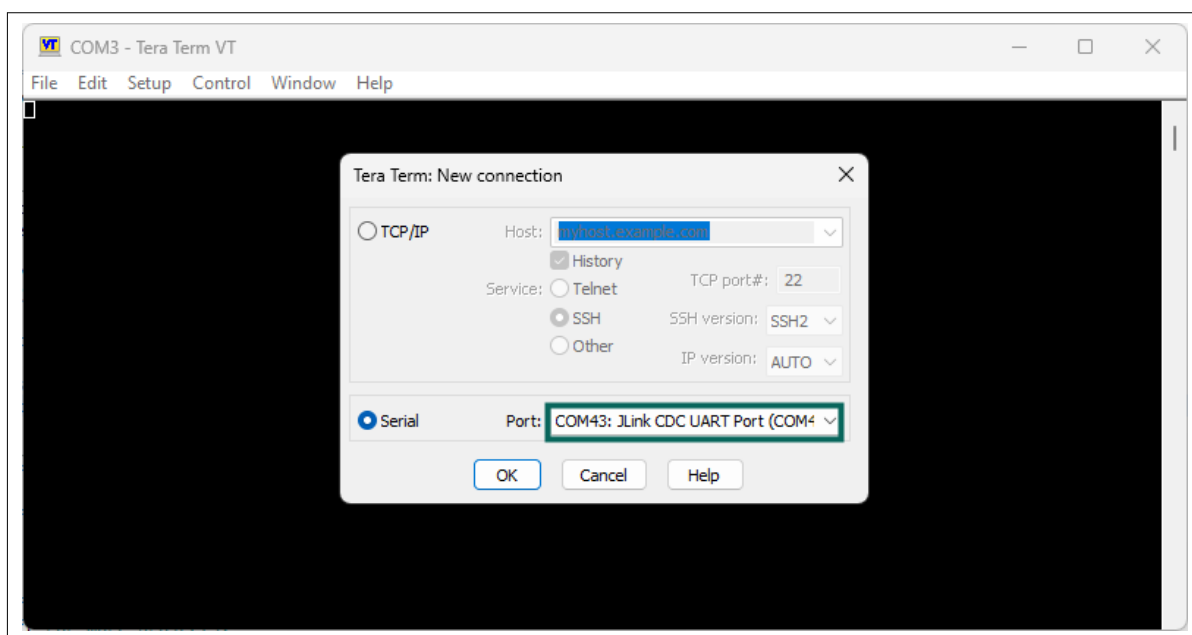


Figure 16 Selecting the JLink COM port in Tera Term

2. Set the baud rate

Set the baud rate to 115200 under **Setup > Serial Port** as [Figure 17](#) shows.

5 Getting started with PSOC™ Control C1 MCU design

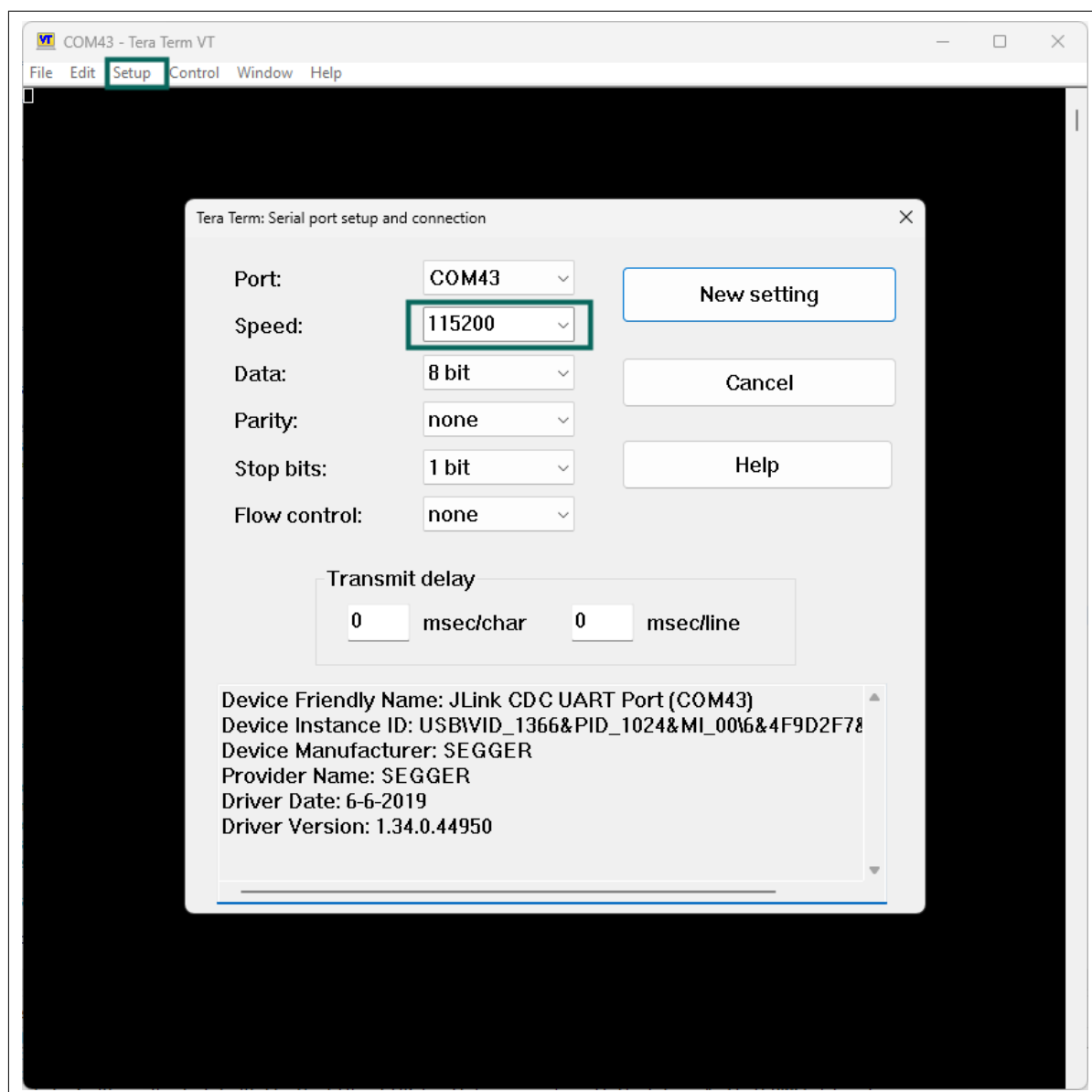


Figure 17 Configuring the baud rate in Tera Term

3. Reset the device

Power cycle the kit to reset the device. The message shown in [Figure 18](#) displays on the terminal. The user LED on the kit will start blinking.

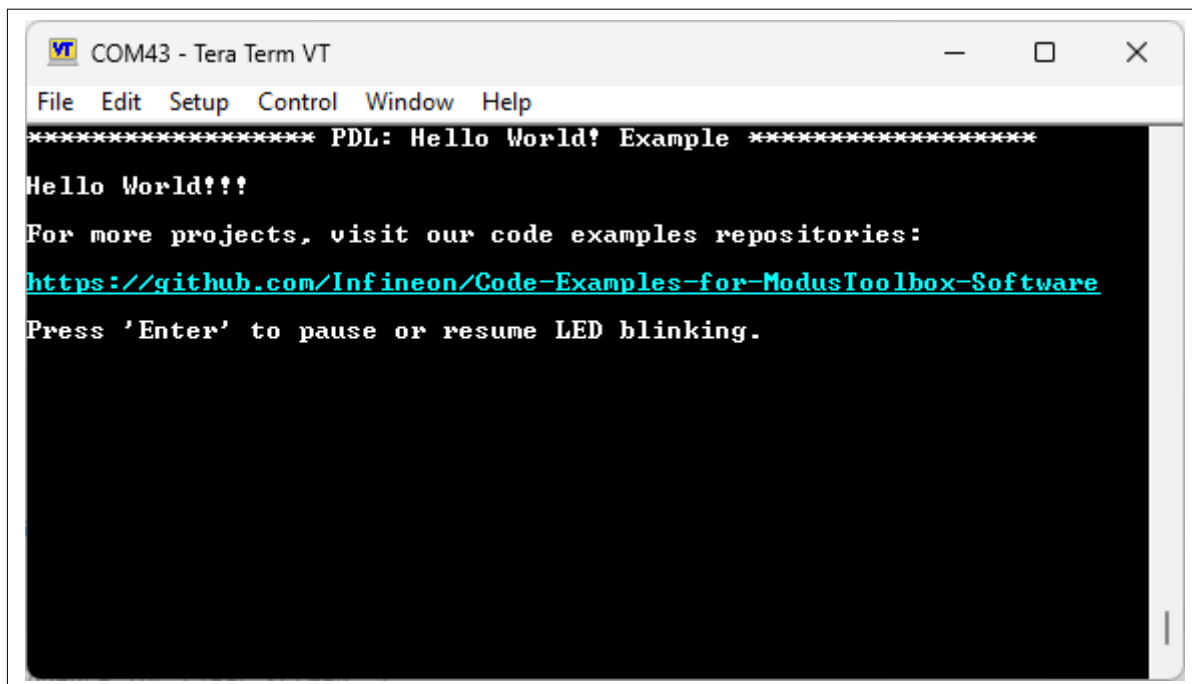


Figure 18 Printed UART message

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 shown in [Figure 19](#).

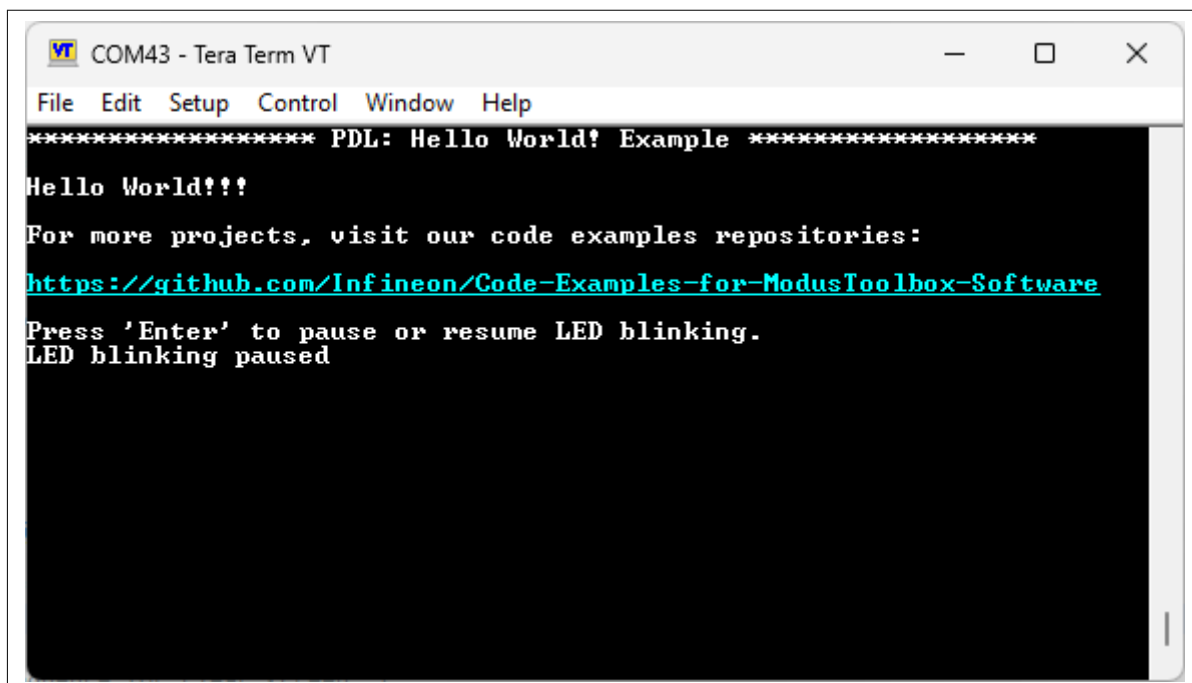


Figure 19 Printed UART message

5.4.1.6 Debugging the application using JLink

PSOC™ Control C1 MCU kits come with J-Link onboard programmer/debugger. See the [J-Link user guide](#) for the details of J-Link.

5 Getting started with PSoC™ Control C1 MCU design

The Eclipse IDE contains several launch configurations that control various settings for programming the devices and launching the debugger. Depending on the kit and the type of applications you are using, there are various launch configurations available. One such configuration is the JLink launch configuration. See the "PSoC™ MCU programming/debugging" section in the [Eclipse IDE for ModusToolbox™ user guide](#) for more details on the launch configurations.

When an application is created for the PSoC™ Control C1 MCU Evaluation Kit, the tool generates the launch configurations for J-link under **Launches** in the **Quick Panel**, as shown in the following figure.

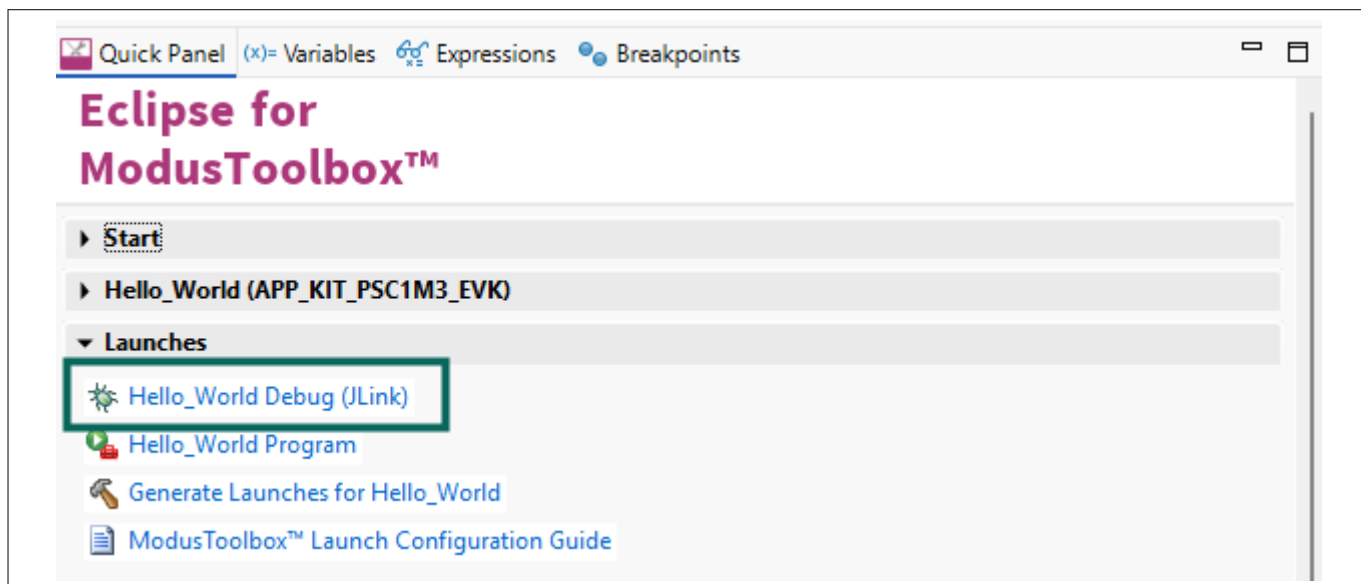


Figure 20 JLink launch configuration

Connect the device to the host machine and click on the **Hello_World Debug (JLink)** launch to start debugging, as shown in [Figure 20](#). Once the debugging starts, the execution halts at the `main()` function, and you can start debugging from the start of the `main()` function, as shown in the following figure.

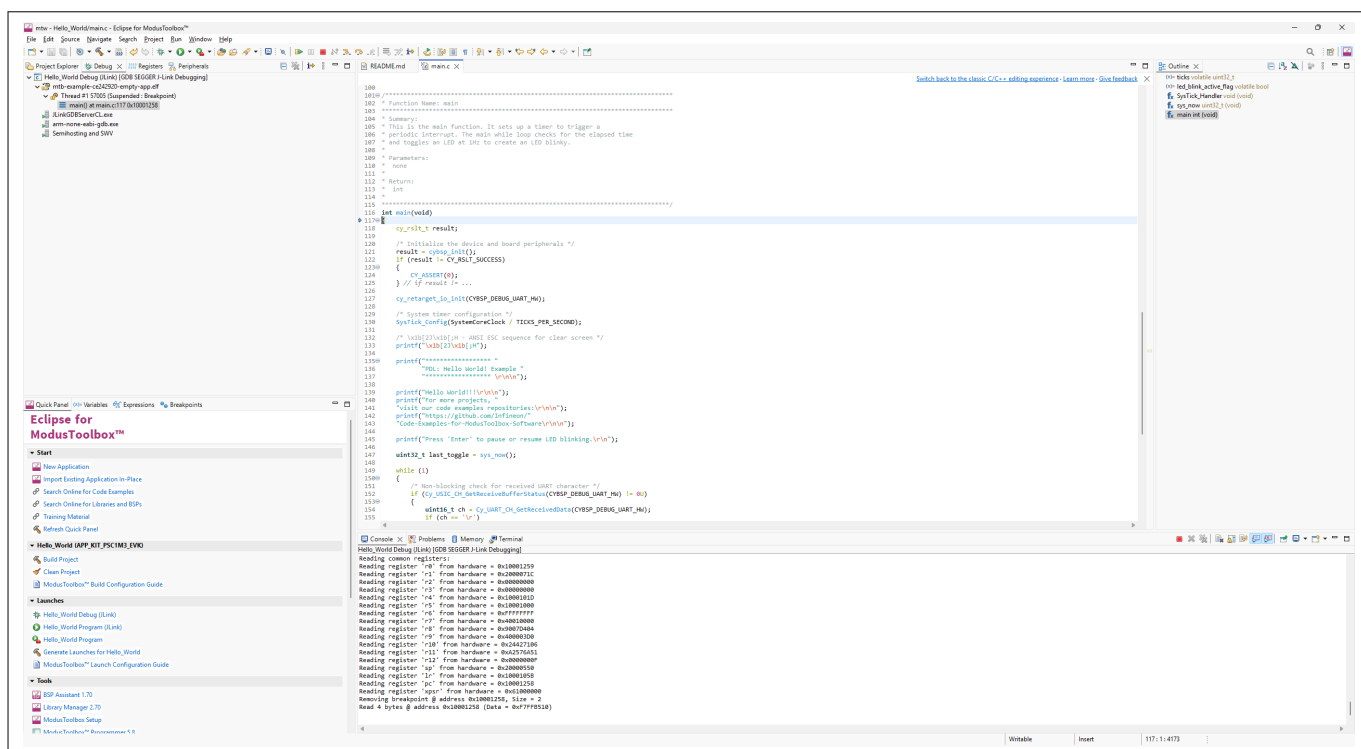


Figure 21 Debug main()

5.4.2 Visual Studio Code (VS Code) for ModusToolbox™

See the [Visual Studio Code for ModusToolbox™ user guide](#) for creating a new application on VS Code.

5.4.3 IAR Embedded Workbench for ModusToolbox™

See the [IAR Embedded Workbench for ModusToolbox™ user guide](#) for creating a new application on IAR.

5.4.4 Keil µVision for ModusToolbox™

See the [Keil µVision for ModusToolbox™ user guide](#) for creating a new application on Keil µVision.

6 Summary

This application note explored the PSOC™ Control C1 MCU device architecture and the associated development tools. PSOC™ Control C1 MCU is a cost-effective, programmable embedded system-on-chip featuring the Arm® Cortex®-M0 processor with configurable analog and digital peripheral functions, memory, and a powerful processor on a chip. The integrated features, low-power modes, compact packages, and ModusToolbox™ toolchain make the PSOC™ Control C1 an ideal choice for industrial drives, motor control, digital power conversion, and other-related applications.

References

Sample code

[1] Infineon Technologies AG: *PSOC™ Control C1 MCU code examples*; [Available online](#)

Documentation

[2] Infineon Technologies AG: *PSOC™ Control C1 MCU-related documents*; [Available online](#)

Glossary

This section lists the most commonly used terms that you might encounter while working with the PSOC™ MCU 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
- **JLink:** The JLink is an onboard programmer/debugger with USB-to-UART bridge functionality. The JLink is integrated onto most PSOC™ Control C1 MCU development kits
- **Personality:** A personality expresses the configurability of a resource for a functionality. For example, the USIC resource can be configured to be an UART, SPI, or I2C personalities
- **Middleware:** Middleware is a set of firmware modules that provide specific capabilities to an application. Some middleware can provide network protocols, and some can provide high-level software interfaces to device features (e.g., EEPROM)
- **ModusToolbox™:** An Eclipse-based embedded design platform for embedded systems 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 PDL simplifies software development for the PSOC™ Control C1 MCU architecture. The PDL reduces the need to understand register usage and bit structures, so easing software development for the extensive set of peripherals available

Revision history

Document revision	Date	Description of changes
**	2026-07-23	Initial release

Trademarks

The Bluetooth® word mark and logos are registered trademarks owned by Bluetooth SIG, Inc., and any use of such marks by Infineon is under license.

PSOC™, formerly known as PSoC™, is a trademark of Infineon Technologies. Any references to PSoC™ in this document or others shall be deemed to refer to PSOC™.

Trademarks

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

Edition 2026-07-23

Published by

Infineon Technologies AG
81726 Munich, Germany

© 2026 Infineon Technologies AG

All Rights Reserved.

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

Email: erratum@infineon.com

Document reference

IFX-vdk1774259808941

Important notice

Products which may also include samples and may be comprised of hardware or software or both ("Product(s)") are sold or provided and delivered by Infineon Technologies AG and its affiliates ("Infineon") subject to the terms and conditions of the frame supply contract or other written agreement(s) executed by a customer and Infineon or, in the absence of the foregoing, the applicable Sales Conditions of Infineon. General terms and conditions of a customer or deviations from applicable Sales Conditions of Infineon shall only be binding for Infineon if and to the extent Infineon has given its express written consent.

For the avoidance of doubt, Infineon disclaims all warranties of non-infringement of third-party rights and implied warranties such as warranties of fitness for a specific use/purpose or merchantability. Infineon shall not be responsible for any information with respect to samples, the application or customer's specific use of any Product or for any examples or typical values given in this document.

If this document is marked as 'Preliminary', 'Target', 'Draft' or 'Proposal', Infineon reserves the right to change all information given in this document at any time without notice. Subject to the development and release of a Product for series supply by Infineon, the technical specifications of the Product are set forth in the relevant datasheet provided by Infineon without such marking.

The data contained in this document is exclusively intended for technically qualified and skilled customer representatives. It is the responsibility of the customer to evaluate the suitability of the Product for the intended application and the customer's specific use and to verify all relevant technical data contained in this document in the intended application and the customer's specific use. The customer is responsible for properly designing, programming, and testing the functionality and safety of the intended application, as well as complying with any legal requirements related to its use.

Unless otherwise explicitly approved by Infineon, Products may not be used in any application where a failure of the Products or any consequences of the use thereof can reasonably be expected to result in personal injury.

However, the foregoing shall not prevent the customer from using any Product in such fields of use that Infineon has explicitly designed and sold it for, provided that the overall responsibility for the application lies with the customer. Infineon expressly reserves the right to use its content for commercial text and data mining (TDM) according to applicable laws, e.g. Section 44b of the German Copyright Act (UrhG). If the Product includes security features:

Because no computing device can be absolutely secure, and despite security measures implemented in the Product, Infineon does not guarantee that the Product will be free from intrusion, data theft or loss, or other breaches ("Security Breaches"), and Infineon shall have no liability arising out of any Security Breaches.

If this document includes or references software:

The software is owned by Infineon under the intellectual property laws and treaties of the United States, Germany, and other countries worldwide. All rights reserved. Therefore, you may use the software only as provided in the software license agreement accompanying the software.

If no software license agreement applies, Infineon hereby grants you a personal, non-exclusive, non-transferable license (without the right to sublicense) under its intellectual property rights in the software (a) for software provided in source code form, to modify and reproduce the software solely for use with Infineon hardware products, only internally within your organization, and (b) to distribute the software in binary code form externally to end users, solely for use on Infineon hardware products. Any other use, reproduction, modification, translation, or compilation of the software is prohibited.

For further information on the Product, technology, delivery terms and conditions, and prices, please contact your nearest Infineon office or visit www.infineon.com.