Please note that Cypress is an Infineon Technologies Company. The document following this cover page is marked as “Cypress” document as this is the company that originally developed the product. Please note that Infineon will continue to offer the product to new and existing customers as part of the Infineon product portfolio.

**Continuity of document content**
The fact that Infineon offers the following product as part of the Infineon product portfolio does not lead to any changes to this document. Future revisions will occur when appropriate, and any changes will be set out on the document history page.

**Continuity of ordering part numbers**
Infineon continues to support existing part numbers. Please continue to use the ordering part numbers listed in the datasheet for ordering.
Getting Started with PSoC 6 MCU on ModusToolbox

Authors: Srinivas Nudurupati
Associated Part Family: All PSoC® 6 MCU devices
Software Version: ModusToolbox™ 2.0
Associated Application Notes and Code Examples: Click here.

More code examples? We heard you.

To access an ever-growing list of PSoC code examples using ModusToolbox IDE, please visit the Cypress GitHub site. You can also explore the PSoC video library here.

AN228571 introduces the PSoC 6 MCU, a dual-CPU programmable system-on-chip with Arm® Cortex®-M4 and Cortex-M0+ processors. This application note helps you explore the PSoC 6 MCU architecture and development tools and shows you how to create your first project using the ModusToolbox IDE. This application note also guides you to more resources available online to accelerate your learning about PSoC 6 MCU.

Contents

1 Introduction ..........................................................1
   1.1 Prerequisites ..................................................2
2 Development Ecosystem ..............................................3
   2.1 PSoC Resources .............................................3
   2.2 Firmware/Application Development .........................3
   2.3 Support for Other IDEs ..................................7
   2.4 FreeRTOS Support with ModusToolbox .................8
   2.5 Programming/Debugging .................................8
   2.6 PSoC 6 MCU Development Kits .......................8
3 Device Features .....................................................9
4 My First PSoC 6 MCU Design Using ModusToolbox IDE ..........10
   4.1 Using These Instructions .................................10
   4.2 About the Design ......................................10
   4.3 Part 1: Create a New Application ..................11
   4.4 Part 2: View and Modify the Design .............14
   4.5 Part 3: Write Firmware ..................................17
   4.6 Part 4: Build the Application .........................22
   4.7 Part 5: Program the Device .........................23
   4.8 Part 6: Test Your Design .............................24
5 Summary ............................................................25
6 Related Application Notes and Code Examples ..........26
   Appendix A. Glossary ....................................27
   Worldwide Sales and Design Support ..................29

1 Introduction

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

- Single-CPU microcontroller: Arm Cortex-M4 (CM4); or Dual-CPU microcontroller: Arm Cortex-M4 (CM4) and Cortex-M0+ (CM0+)
- Programmable analog and digital peripherals
- Up to 2 MB of flash and 1 MB of SRAM
- Fourth-generation CapSense® technology
- PSoC 6 MCU is suitable for a variety of power-sensitive applications such as:
  - 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

ModusToolbox™ IDE, the Eclipse-based IDE for developing PSoC 6 MCU applications, allows flexibility and dynamic fine-tuning of the design including the programmable analog and digital subsystems.

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

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

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

- A buck converter (not shown in the figure) 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 (Universal Digital Blocks or UDBs) and peripherals (Timer Counter PWM or TCPWM) to drive the display and LEDs respectively
- SDIO interface to a Cypress Wi-Fi/BT device to provide IoT cloud connectivity
- Product security features managed by CM0+ CPU and application features executed by CM4 CPU

See Device Features and the device datasheets for more details.

This application note introduces you to the capabilities of 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.

1.1 Prerequisites
Before you get started, make sure that you have the development kit and have installed the required software. You also need internet access to the Cypress GitHub repositories during project creation.

1.1.1 Hardware
- PSoC 6 Wi-Fi BT Prototyping Kit (CY8CPROTO-062-4343W)

1.1.2 Software
- ModusToolbox 2.0

After installing the software, refer to the Quick Start Guide and User Guide in ModusToolbox IDE to get an overview of the software.
2 Development Ecosystem

2.1 PSoC Resources

Cypress provides a wealth of data at www.cypress.com to help you to select the right PSoC device and quickly and effectively integrate it into your design. For a comprehensive list of PSoC 6 MCU resources, see How to Design with PSoC 6 MCU - KBA223067 in the Cypress community. The following is an abbreviated list of resources for PSoC 6 MCU.

- **Overview**: PSoC Portfolio, PSoC Roadmap
- **Product Selectors**: PSoC 6 MCU
- **Datasheets** describe and provide electrical specifications for each device family.
- **Application Notes and Code Examples** cover a broad range of topics, from basic to advanced level. You can also browse our collection of code examples. See Code Examples.
- **Technical Reference Manuals (TRMs)** provide detailed descriptions of the architecture and registers in each device family.
- **PSoC 6 MCU Programming Specification** provides the information necessary to program the nonvolatile memory of PSoC 6 MCU devices.
- **CapSense Design Guides**: Learn how to design capacitive touch-sensing applications with PSoC devices.

- **Development Tools**
  - PSoC 6 Wi-Fi-BT Pioneer Kit (CY8CKIT-062-WiFi-BT) is a development kit that supports the PSoC 62 series MCU along with Wi-Fi and BT connectivity.
  - PSoC 6 BLE Pioneer Kit (CY8CKIT-062-BLE) is an easy-to-use and development platform for PSoC 63 series MCU with BLE Connectivity.
  - PSoC 6 Wi-Fi BT Prototyping Kit (CY8CPROTO-062-4343W) is a kit with a snap-away form-factor that supports the PSoC 62 series MCU along with CYW4343W module-based Wi-Fi and BT connectivity.
  - PSoC 6 BLE Prototyping Kit (CY8CPROTO-063-BLE) is a kit with a snap-away form-factor that supports the PSoC 63 series MCU with BLE Connectivity.

- **Training Videos**: Cypress provides video training on our products and tools, including a dedicated series on PSoC 6 MCU.

2.2 Firmware/Application Development

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

- **ModusToolbox**: ModusToolbox software includes configuration tools, low-level drivers, middleware libraries, and operating system support, as well as other packages that enable you to create MCU and wireless applications. It also includes the optional ModusToolbox IDE.
  
  ModusToolbox IDE is an Eclipse-based development environment that runs on Windows, macOS, and Linux platforms and includes various tools.
  
  ModusToolbox supports stand-alone device and middleware configurators that are fully integrated into the IDE. Use the configurators to set the configuration of different blocks in the device and generate code that can be used in firmware development. ModusToolbox supports all PSoC 6 MCU devices. It is recommended that you use ModusToolbox for all application development for PSoC 6 MCUs. See the ModusToolbox Software Overview for more information.
  
  Cypress provides libraries and enablement software at the Cypress GitHub site. Some resources will be used by all developers. Others will be used by developers in particular ecosystems.
  
  Cypress 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, Wi-Fi, Bluetooth, and Bluetooth Low Energy application development
  - Mbed OS ecosystem – provides an embedded operating system, transport security and cloud services to create connected embedded solutions
  - Amazon FreeRTOS ecosystem – extends the FreeRTOS kernel with software libraries that make it easy to securely connect small, low-power devices to AWS cloud services

  ModusToolbox tools and resources can also be used in the command line. See Running ModusToolbox from the Command Line for detailed documentation.

- **PSoC Creator**: A Cypress-proprietary IDE that runs on Windows only. It supports a subset of PSoC 6 MCU devices as well as other PSoC device families such as PSoC 3, PSoC 4, and PSoC 5LP. See AN221774 Getting Started with PSoC 6 on PSoC Creator for more information.
2.2.1 Choosing an IDE

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

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

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

PSoc Creator is the long-standing Cypress-proprietary tool that runs on Windows only. This mature IDE includes a graphical editor that supports schematic based design entry with the help of Components. PSoC Creator supports all PSoC 3, PSoC 4, PSoC 5LP devices and a subset of PSoC 6 MCU devices. The subset of PSoC 6 MCU devices includes devices up to 1 MB of flash.

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

2.2.2 ModusToolbox Software

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

One of the tools is a multi-platform, Eclipse-based Integrated Development Environment (IDE) called the ModusToolbox IDE that supports application configuration and development. Figure 2 shows a high-level view of the tools/resources included in the ModusToolbox software. For a more in-depth overview of the ModusToolbox software, see ModusToolbox Software Overview.

![Figure 2. ModusToolbox Software](image)

ModusToolbox installer includes the ModusToolbox IDE, 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 ModusToolbox IDE, the make infrastructure, and other tools.

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

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

   To get started with firmware development on PSoC 6 using Amazon FreeRTOS, visit the Getting Started with Amazon FreeRTOS.

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

   To get started with firmware development on PSoC 6 using Mbed OS, refer to Getting Started with PSoC 6 MCU and CYW43xxx in Mbed OS.

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

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

- Board Support Packages (BSP) - A BSP is the layer of firmware containing board-specific drivers and other functions. The board support package is a set of libraries that provide firmware APIs to initialize the board and provide access to board level peripherals. It includes any low-level resources (like PDL library for PSoC 6 MCU), and has definitions for macros for board peripherals. It uses the HAL to configure the board. Custom BSPs can be created to enable support for end-application boards. Refer to the ModusToolbox BSP User Guide for more information.

- Hardware Abstraction Layer (HAL) - The Cypress Hardware Abstraction Layer (HAL) provides a high-level interface to configure and use hardware blocks on Cypress MCUs. It is a generic interface that can be used across multiple product families. The focus on ease-of-use and portability means the HAL does not expose all the low-level peripheral functionality. The HAL wraps the lower level drivers (like PSoC 6 PDL) and provides a high-level interface to the MCU. The interface is abstracted to work on any MCU. This helps you write application firmware independent of the target MCU.

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

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

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

- Extensive middleware libraries that provides specific capabilities to an application. The available middleware spans across connectivity (Bluetooth, AWS IoT, BLE) to PSoC-6-specific functionality (CapSense, USB, device firmware upgrade (DFU), emWin). All the middleware is delivered as libraries and via Cypress GitHub repositories.

2.2.3 PSoC 6 Software Resources

The PSoC 6 software includes driver and middleware configurators to get you started developing firmware with PSoC 6 MCU. It contains Configurators, drivers, libraries, middleware, as well as various utilities, makefiles, and scripts. It also includes relevant drivers, middleware, and examples for use with Cypress IoT devices and connectivity solutions. You may use any or all tools in any environment you prefer.
2.2.3.1 Configurators

ModusToolbox software provides graphical applications called Configurators that make it easier to configure a hardware block. For example, instead of having to search through all the documentation to configure a serial communication block as a UART with a desired configuration, open the appropriate Configurator and set the baud rate, parity, stop bits. Upon saving the hardware configuration, the tool generates the "C" code to initialize the hardware with the desired configuration.

Configurators are independent of each other, but they can be used together to provide flexible configuration options. They can be used stand alone, in conjunction with other tools, or within a complete IDE. Everything is bundled together as part of the unified SDK for distribution purposes. Configurators are used for:

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

For PSoC 6 MCU applications, the available Configurators include:

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

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

2.2.3.2 Software Development for PSoC 6 MCU

Cypress 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, and include various middleware libraries for additional functionality, like BLE connectivity or FreeRTOS. This source code makes it easier to develop the firmware for supported devices. It helps you quickly customize and build firmware without the need to understand the register set.

In the ModusToolbox environment, you use Configurators to configure either the device, or a middleware library, like the BLE stack or CapSense functionality.

The driver code is delivered as the psoc6pdl library. Middleware is delivered as separate libraries for each feature/function.

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

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

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

This enables the code to maintain compatibility across members of the PSoC 6 device family as long as the peripherals are available. A device header file specifies the peripherals that are available for a device. If you write code that attempts to use an unsupported peripheral, you will get an error at compile time. Before writing code to use a peripheral, consult the datasheet for the particular device to confirm support for that peripheral.
As Figure 3 shows, with the ModusToolbox IDE, you can:

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

Figure 3. ModusToolbox IDE Resources and Middleware

2.2.4 ModusToolbox Help

ModusToolbox Software Overview provides a high-level overview of the ModusToolbox software. Visit the ModusToolbox home page to download and install the latest version of ModusToolbox. Launch ModusToolbox IDE and navigate to the following items:

Choose Help > ModusToolbox IDE Documentation >:

- Quick Start Guide: This guide gives you the basics for using ModusToolbox.
- User Guide: This guide primarily covers the ModusToolbox IDE aspects of building, programming and debugging applications. It also covers various aspects of the tools installed along with the IDE.
- Eclipse IDE Survival Guide

Additional documentation is available in the ModusToolbox Documentation landing page under Help > ModusToolbox General Documentation.

2.3 Support for Other IDEs

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

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

See AN225588 – Using ModusToolbox Software with a Third-Party IDE for details.

2.4 FreeRTOS Support with ModusToolbox

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

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

Figure 4. Import FreeRTOS middleware in ModusToolbox IDE Application

2.5 Programming/Debugging

The PSoC 6 Wi-Fi BT Prototyping Kit (CY8CPROTO-062-4343W) has 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 ModusToolbox 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-BT Pioneer Kit (CY8CKIT-062-WiFi-BT) and PSoC 6 BLE Pioneer Kit (CY8CKIT-062-BLE) have the KitProg2 onboard programmer/debugger. To work with ModusToolbox IDE, upgrade the kit to KitProg3.

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

For more information on debugging firmware on PSoC devices with ModusToolbox, refer to ModusToolbox Help.

2.6 PSoC 6 MCU Development Kits

PSoC 6 Wi-Fi-BT Pioneer Kit (CY8CKIT-062-WiFi-BT) and PSoC 6 Wi-Fi BT Prototyping Kit (CY8CPROTO-062-4343W) are development kits that supports the PSoC 62 series MCU along with Wi-Fi and BT connectivity.

The PSoC 6 BLE Pioneer Kit (CY8CKIT-062-BLE) and PSoC 6 BLE Prototyping Kit (CY8CPROTO-063-BLE) support the PSoC 6 MCU with Bluetooth Low Energy (BLE) Connectivity. Refer to the PSoC 6 product page for more information.
3 Device Features

The PSoC 6 MCU device has an extensive feature set as shown in Figure 5. The following is a list of its major features. For more information, see the device datasheet, the Technical Reference Manual (TRM), and the section on Related Application Notes and Code Examples.

- **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

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

- **Programmable digital blocks, communication interfaces**
  - Up to 12 UDBs for custom digital peripherals
  - Up to 32 TCPWM blocks configurable as 16-bit/32-bit timer, counter, PWM, or quadrature decoder
  - Up to 13 SCBs configurable as I2C Master or Slave, SPI Master or Slave, or UART
  - 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
  - Full-Speed, dual-role USB with device and host capability

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

- **Operating voltage range, power domains, and low-power modes**
  - Device operating voltage: 1.71 V to 3.6 V with user-selectable core logic operation at either 1.1 V or 0.9 V
  - Multiple on-chip regulators: low-drop out (LDO for Active, Deep Sleep modes), 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 ModusToolbox IDE

This section does the following:

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

4.1 Using These Instructions

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

- Part 1: Create a New Application
- Part 2: View and Modify the Design
- Part 3: Write Firmware
- Part 4: Build the Application
- Part 5: Program the Device
- Part 6: Test Your Design

This design is developed for the PSoC 6 Wi-Fi BT 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-BT Pioneer Kit (CY8CKIT-062-WiFi-BT)
- PSoC 6 BLE Pioneer Kit (CY8CKIT-062-BLE)
- PSoC 6 BLE Prototyping Kit (CY8CPROTO-063-BLE)
- PSoC 62S2 Wi-Fi BT Pioneer Kit (CY8CKIT-062S2-43012)

4.2 About the Design

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

At device reset, the Cypress-supplied pre-built CM0+ application image enables the CM4 CPU and configures the CM0+ CPU to go to sleep. The CM4 CPU uses the UART to print a “Hello World” message to the serial port stream, and starts blinking the user LED on the kit. When the user presses the enter key on the serial console, the blinking is paused or resumed.
Part 1: Create a New Application

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

If you are familiar with developing projects with ModusToolbox, you can use the ‘Hello World’ starter application directly. It is a complete design, with all the firmware written for the CY8CPROTO-062-4343W kit. You can walk through the instructions and observe how the steps are implemented in the code example.

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

Launch ModusToolbox to get started. Please note that ModusToolbox will need access to the internet to successfully clone the starter application onto your machine.

1. Select a new workspace.

   At launch, ModusToolbox presents a dialog to choose a directory for use as the workspace directory. The workspace directory is used to store workspace preferences and development artifacts. You can choose an existing empty directory by clicking the Browse button, as Figure 6 shows. Alternatively, you can type in a directory name to be used as the workspace directory along with the complete path and ModusToolbox will create the directory for you.

   Figure 6. 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 IDE Application, as Figure 7 shows.

   The ModusToolbox IDE Application window appears.

   Figure 7. Create a New ModusToolbox IDE 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 assume **CY8CPROTO-062-4343W**. See Figure 8 for help with this step.

B. Click **Next**.

Figure 8. Choose Target Hardware

C. In the **Starter Application** dialog, select **Empty PSoC6 App** starter application.

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 **Next**. The application summary dialog appears.

Figure 9. Choose Starter Application
F. Click **Finish** to let ModusToolbox create the application project for you.

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-BT Prototyping Kit (CY8CPROTO-062-4343W) along with the CYW4343WKUBG Wi-Fi/BT radio.

If you are using custom hardware based on PSoC 6 MCU, or a different PSoC 6 MCU part number, please refer to the **ModusToolbox BSP User Guide**. The guide is also available under *ide_2.0>docs* folder of the ModusToolbox installation directory.
4.4 Part 2: View and Modify the Design

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

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

a. The build folder contains all the artifacts resulting from the make build of the project. The output files are organized by target BSPs.

b. The libs folder has folders belonging to different middleware, BSP, and PSoC 6 PDL. These are individual libraries that Cypress provides and are downloaded based on the .lib files provided within the project.

c. The files provided by the BSP are listed under the TARGET_x folder. All the configuration files generated by the device and peripheral configurators are included in the GeneratedSource folder of the BSP and are prefixed with cycfg_. These files contain the design configuration as defined by the BSP. You can view and modify the design configuration by double-clicking the design.modus file in the project. However, note that if you upgrade the BSP library to a newer version, the manual edits done to the design.x files are lost. You can also create custom device configurator files for an application and override the ones provided by the BSP. See ModusToolbox Help for more details.

The BSP folder also contains the .lib files that specify the middleware and other libraries used in the project, the linker scripts and the startup code for the PSoC 6 MCU device used on the board.
d. The .lib files provide the location from which ModusToolbox pulls the content from. These files typically contain the GitHub location of the entire library. A .lib file can point to content that contains another .lib file. ModusToolbox processes this nested .lib file recursively and downloads all libraries.

For example, the BSP lib file `TARGET_CY8CPROTO-062-4343W.lib` points to
https://github.com/cypresssemiconductorco/TARGET_CY8CPROTO-062-4343W/#latest-v1.X. The `latest-v1.X` tag in the link denotes the specific release of the BSP.

As a second example, the `retarget-io.lib` points to the library hosted at
https://github.com/cypresssemiconductorco/retarget-io/#latest-v1.X.

e. Note that application project contains a `makefile` file. It has instructions on how to recreate the project. This file also contains the set of directives that the make tool uses to compile and link the application project.

Of interest are the configuration files that are in the `COMPONENT_BSP.x` folder. Double-click the `design.modus` file in the config project in the Project Explorer, or click the Device Configurator link in the Quick Panel. Figure 12 shows the resulting window called the Device Configurator window. You can also double-click open the other `design.x` files to open them in their respective configurators or click the corresponding links in the Quick Panel.

![Figure 12. design.modus Overview](image)

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

You can choose how a resource behaves by choosing a Personality for the resource. For example, a Serial Communication Block (SCB) resource can have a 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. Any errors, warnings, and information messages arising out of the configuration are displayed in the Notices pane.

The application project contains relevant files that help you create an application for the CM4 CPU (`main.c`), while the CM0+ application is supplied as a C file (`psoc6_02_cm0p_sleep.c` for CY8C624ABZI-D44 device) by Cypress. See the `psoc6cm0p` library. This C file is compiled and linked with the CM4 image as part of the normal build process.
At this point in the development process, we are ready to add the required middleware to the design. The only middleware required for the Hello World application is the retarget-io library.

**Figure 13. Add the retarget-io Middleware**

1. **Add retarget-io middleware.**

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

   A. In the **Quick Panel**, click on the **Library Manager** link.

   B. In the subsequent dialog, select the **Libraries** tab.

   C. Under **Board Utils**, select and enable **retarget-io**.

   D. Click **Apply**.

   The files necessary to use the retarget-io middleware are added in the `libs > retarget_io` folder, and the `.lib` file is also added, as **Figure 13** shows.

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

   The configuration of the debug UART peripheral, timer peripheral, pins and system clocks can be done directly in the code using the function APIs provided by BSP and HAL. See **Part 3: Write Firmware**.
4.5 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.

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

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

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

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

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

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

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

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

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

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

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

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

/* 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;

/* UART HAL object used by Retarget-IO for Debug UART port */
extern cyhal_uart_t cy_retarget_io_uart_obj;

/* 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
*******************************************************************************

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, CYBSP_DEBUG_UART_RX);

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

/* Initialize the User LED */
result = cyhal_gpio_init({cyhal_gpio_t 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("1. ModusToolbox Examples:\r\nhttps://github.com/
"cyressemiconductorco/Code-Examples-for-ModusToolbox"
"Software\r\n\n");

printf("2. Mbed OS Examples:\r\nhttps://os.mbed.com/teams/Cypress/\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)\n== 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((cyhal_gpio_t) CYBSP_USER_LED);
}

/******************************************************************************
* Function Name: timer_init
********************************************************************
***********/
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 pin output ('pin' is NC) and */
    /* does not use a pre-configured clock source ('clk' is NULL). */
    result = cyhal_timer_init(&led_blink_timer, (cyhal_gpio_t) 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
*******************************************************************************/
static void isr_timer(void *callback_arg, cyhal_timer_event_t event)
{
    (void) callback_arg;
    (void) event;
    timer_interrupt_flag = true;
}
Figure 14. Firmware Flowchart

START

Enable CM4 CPU & go to Sleep

Initialize clocks and system resources

Initialize retarget-io to use BSP’s debug UART

Initialize the user LED

Print the message “Hello World” on to UART terminal

Initialize and start the timer, register callback – LED starts blinking

Is “Enter” key pressed?

Yes

Was LED blinking earlier?

No

Yes

Stop the timer

Start the timer

Is timer interrupt flag set?

No

Yes

Clear timer interrupt flag & Toggle LED state

This completes the summary of how the firmware works in the code example. Feel free to explore the source files for a deeper understanding.
4.6 Part 4: Build the Application

This section shows how to build the application.

1. Build the Application.

   A. Select the application project in the Project Explorer window and click on the Build <name> Application shortcut under the <name> group in the Quick Panel. It selects the Debug build configuration and compiles/links all projects that constitute the application.

   B. The Console view lists the results of the build operation, as Figure 15 shows.

   Figure 15. Build the Application

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

Note: You can also use the command line interface (CLI) to build the application. Please refer to Running ModusToolbox from the Command Line guide. This document is located in the /ide_2.0/docs/ folder in the ModusToolbox installation.
4.7 Part 5: Program the Device

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

ModusToolbox uses the OpenOCD protocol to program and debug applications on PSoC 6 MCU devices. For ModusToolbox to identify the device on the kit, the kit must be running KitProg3. Some Cypress kits are shipped with KitProg2 firmware instead of KitProg3. See Programming/Debugging for details. ModusToolbox includes the `fw-loader` command-line tool to switch the KitProg firmware from KitProg2 to KitProg3. Refer to the PSoC 6 MCU KitProg Firmware Loader section in the ModusToolbox IDE 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 may need a hardware programmer/debugger; for example, a Cypress CY8CKIT-005 MiniProg4.

1. Program the Application.

   A. Connect to the board and perform the following step.

   B. Select the application project and click on the `<application name> Program (KitProg3)` shortcut under the Launches group in the Quick Panel, as Figure 16 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.

The Console view lists the results of the programming operation, as Figure 17 shows.
4.8 Part 6: Test Your Design

This section describes how to test your design. Follow the steps below to observe the output of your design. 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 shown in Figure 18. Note that your COM port number may be different.

2. Set the baud rate.

   Set the baud rate to 115200 under Setup > Serial port as Figure 19 shows.
3. Reset the device.
   Press the reset switch (SW1) on the kit. A message appears on the terminal as Figure 20 shows. The user LED on the kit will start blinking.

   Figure 20. UART Message Printed from CM4 CPU

4. Pause/resume LED blinking functionality.
   Press the Enter Key to pause/resume blinking the LED. When the LED blinking is paused, a corresponding message will be displayed on the terminal as shown in Figure 21.

   Figure 21. UART Message from CM4 CPU

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

For a complete and updated list of PSoC 6 MCU code examples, please visit our Cypress GitHub. For more PSoC 6 MCU-related documents, please visit our PSoC 6 MCU product web page.

Table 1 lists the system-level and general application notes that are recommended for the next steps in learning about PSoC 6 MCU and ModusToolbox.

Table 1. General and System-Level Application Notes

<table>
<thead>
<tr>
<th>Document</th>
<th>Document Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>AN221774</td>
<td>Getting Started with PSoC 6 MCU on PSoC Creator</td>
</tr>
<tr>
<td>AN210781</td>
<td>Getting Started with PSoC 6 MCU with Bluetooth Low Energy (BLE) Connectivity on PSoC Creator</td>
</tr>
<tr>
<td>AN218241</td>
<td>PSoC 6 MCU Hardware Design Considerations</td>
</tr>
<tr>
<td>AN225588</td>
<td>Using ModusToolbox Software with a Third-Party IDE</td>
</tr>
<tr>
<td>AN219528</td>
<td>PSoC 6 MCU Low-Power Modes and Power Reduction Techniques</td>
</tr>
</tbody>
</table>

Table 2 lists the application notes (AN) for specific peripherals and applications.

Table 2. Documents Related to PSoC 6 MCU Features

<table>
<thead>
<tr>
<th>Document</th>
<th>Document Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>System Resources, CPU, and Interrupts</td>
<td></td>
</tr>
<tr>
<td>AN215656</td>
<td>PSoC 6 MCU Dual-CPU System Design</td>
</tr>
<tr>
<td>AN217666</td>
<td>PSoC 6 MCU Interrupts</td>
</tr>
<tr>
<td>CapSense</td>
<td></td>
</tr>
<tr>
<td>AN92239</td>
<td>Proximity Sensing with CapSense</td>
</tr>
<tr>
<td>AN85951</td>
<td>PSoC 4 and PSoC 6 MCU CapSense Design Guide</td>
</tr>
<tr>
<td>Device Firmware Update</td>
<td></td>
</tr>
<tr>
<td>AN213924</td>
<td>PSoC 6 MCU Device Firmware Update Software Development Kit Guide</td>
</tr>
</tbody>
</table>
Appendix A. Glossary

This section lists the most commonly used terms that you might encounter while working with Cypress’s 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 PSoC 6 PDL) and provides a high-level interface to the MCU. The interface is abstracted to work on any MCU.

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

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

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

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

**Middleware:** Middleware is a set of firmware modules that provide specific capabilities to an application. Some middleware may provide network protocols (e.g. MQTT), and some may provide high level software interfaces to device features (e.g. USB, audio).

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

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

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

Document Title: AN228571 – Getting Started with PSoC 6 MCU on ModusToolbox
Document Number: 002-28571

<table>
<thead>
<tr>
<th>Revision</th>
<th>ECN</th>
<th>Submission Date</th>
<th>Description of Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>**</td>
<td>6697981</td>
<td>10/11/2019</td>
<td>New application note</td>
</tr>
</tbody>
</table>
Worldwide Sales and Design Support

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

Products

<table>
<thead>
<tr>
<th>Category</th>
<th>Cypress Website</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arm® Cortex® Microcontrollers</td>
<td>cypress.com/arm</td>
</tr>
<tr>
<td>Automotive</td>
<td>cypress.com/automotive</td>
</tr>
<tr>
<td>Clocks &amp; Buffers</td>
<td>cypress.com/clocks</td>
</tr>
<tr>
<td>Interface</td>
<td>cypress.com/interface</td>
</tr>
<tr>
<td>Internet of Things</td>
<td>cypress.com/iot</td>
</tr>
<tr>
<td>Memory</td>
<td>cypress.com/memory</td>
</tr>
<tr>
<td>Microcontrollers</td>
<td>cypress.com/mcu</td>
</tr>
<tr>
<td>PSoC</td>
<td>cypress.com/psoc</td>
</tr>
<tr>
<td>Power Management ICs</td>
<td>cypress.com/pmic</td>
</tr>
<tr>
<td>Touch Sensing</td>
<td>cypress.com/touch</td>
</tr>
<tr>
<td>USB Controllers</td>
<td>cypress.com/usb</td>
</tr>
<tr>
<td>Wireless Connectivity</td>
<td>cypress.com/wireless</td>
</tr>
</tbody>
</table>

PSoC® Solutions

<table>
<thead>
<tr>
<th>Version</th>
</tr>
</thead>
<tbody>
<tr>
<td>PSoC 1</td>
</tr>
<tr>
<td>PSoC 3</td>
</tr>
<tr>
<td>PSoC 4</td>
</tr>
<tr>
<td>PSoC 5LP</td>
</tr>
<tr>
<td>PSoC 6 MCU</td>
</tr>
</tbody>
</table>

Cypress Developer Community

Community | Code Examples | Projects | Videos | Blogs | Training | Components

Technical Support

cypress.com/support

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

© Cypress Semiconductor Corporation, 2019. This document is the property of Cypress Semiconductor Corporation and its subsidiaries (“Cypress”). This document, including any software or firmware included or referenced in this document (“Software”), is owned by Cypress under the intellectual property laws of the United States and other countries worldwide. Cypress reserves all rights under such laws and treaties and does not, except as specifically stated in this paragraph, grant any license under its patents, copyrights, trademarks, or other intellectual property rights. If the Software is not accompanied by a license agreement and you do not otherwise have a written agreement with Cypress governing the use of the Software, then Cypress hereby grants you a personal, non-exclusive, non-transferable license (without the right to sublicense) (1) under its copyright rights in the Software (a) for Software provided in source code form, to modify and reproduce the Software solely for use with Cypress hardware products, only internally within your organization, and (b) to distribute the Software in binary code form externally to end users, (either directly or indirectly through resellers and distributors), solely for use on Cypress hardware product units, and (2) under those claims of Cypress’s patents that are infringed by the Software (as provided by Cypress, unmodified) to make, use, distribute, and import the Software solely for use with Cypress hardware products. Any other use, reproduction, modification, translation, or compilation of the Software is prohibited.

TO THE EXTENT PERMITTED BY APPLICABLE LAW, CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS DOCUMENT OR ANY SOFTWARE OR ACCOMPANYING HARDWARE, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. No computing device can be absolutely secure. Therefore, despite security measures implemented in Cypress hardware or software products, Cypress shall have no liability arising out of any security breach, such as unauthorized access to or use of a Cypress product. CYPRESS DOES NOT REPRESENT, WARRANT, OR GUARANTEE THAT CYPRESS PRODUCTS, OR SYSTEMS CREATED USING CYPRESS PRODUCTS, WILL BE FREE FROM CORRUPTION, ATTACK, VIRUSES, INTERFERENCE, HACKING, DATA LOSS OR THEFT, OR OTHER SECURITY INTRUSION (collectively, “Security Breach”). Cypress disclaims any liability relating to any Security Breach, and you shall and hereby do release Cypress from any claim, damage, or other liability arising from any Security Breach. In addition, the products described in these materials may contain design defects or errors known as errata which may cause the product to deviate from published specifications. To the extent permitted by applicable law, Cypress reserves the right to make changes to this document without further notice. Cypress does not assume any liability arising out of the application or use of any product or circuit described in this document. Any information provided in this document, including any sample design information or programming code, is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. “High-Risk Device” means any device or system whose failure could cause personal injury, death, or property damage. Examples of High-Risk Devices are weapons, nuclear installations, surgical implants, and other medical devices. “Critical Component” means any component of a High-Risk Device whose failure to perform can be reasonably expected to cause, directly or indirectly, the failure of the High-Risk Device, or to affect its safety or effectiveness. Cypress is not liable, in whole or in part, and you and shall and hereby do release Cypress from any claim, damage, or other liability arising from any use of a Cypress product as a Critical Component in a High-Risk Device. You shall indemnify and hold Cypress, its directors, officers, employees, agents, affiliates, distributors, and assigns harmless from and against all claims, costs, damages, and expenses, arising out of any claim, including claims for product liability, personal injury or death, or property damage arising from any use of a Cypress product as a Critical Component in a High-Risk Device. Cypress products are not intended or authorized for use as a Critical Component in any High-Risk Device except to the limited extent that (i) Cypress’s published data sheet for the product explicitly states Cypress has qualified the product for use in a specific High-Risk Device, or (ii) Cypress has given you advance written authorization to use the product as a Critical Component in the specific High-Risk Device and you have signed a separate indemnification agreement.

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

© Cypress Semiconductor Corporation, 1988-2019. All rights reserved.