Contents

1 Introduction ..............................1
  1.1 How to Use this Document .............2
2 General Dual-CPU Concepts ...............2
3 PSoC 6 MCU Dual-CPU Architecture ..4
4 PSoC 6 MCU Dual-CPU Development ....6
  4.1 ModusToolbox Instructions ..........6
  4.2 PSoC Creator Instructions ..........8
  4.3 Resource Assignment Considerations .11
  4.4 Interrupt Assignment Considerations .12
  4.5 Debug Considerations .................13
    4.5.1 ModusToolbox Instructions .....13
    4.5.2 PSoC Creator Instructions ......13
    4.5.3 Instructions for Other IDEs .......14
5 Summary ................................32
6 Related Documents .....................33
7 Worldwide Sales and Design Support .....35

1 Introduction

PSoC 6 MCU is Cypress’ 32-bit ultra-low-power PSoC, purpose-built for the Internet of Things (IoT). It integrates low-power flash and SRAM technology, programmable digital logic, programmable analog, high-performance analog-digital conversion, low-power comparators, and standard communication and timing peripherals.

Of particular interest in PSoC 6 MCU is the CPU subsystem. The architecture incorporates multiple bus masters – two CPUs, two DMA controllers, and a cryptography block (Crypto) – as Figure 1 shows:

Figure 1. PSoC 6 MCU Typical CPU Subsystem Architecture

AN215656 describes the dual-CPU architecture in PSoC 6 MCUs, which includes Arm® Cortex®-M4 and Cortex-M0+ CPUs, as well as an inter-processor communication (IPC) module. A dual-CPU architecture provides the flexibility to help improve system performance and efficiency, and reduce power consumption. The application note also shows how to build a simple dual-CPU design using Cypress' ModusToolbox™ and PSoC Creator Integrated Development Environments (IDEs), and how to debug the design using various IDEs.
Note: The contents of the block diagram in Figure 1 may vary depending on the device. Some PSoC 6 MCU parts have only one CPU. See the device datasheet for details. This application note does not apply to single-CPU PSoC 6 MCU devices.

Generally, all memory and peripherals are shared by all bus masters. Shared resources are accessed through standard Arm multi-layer bus arbitration. Exclusive accesses are supported by an inter-processor communication (IPC) block, which implements semaphores and mutual exclusion (mutexes) in hardware.

A dual-CPU architecture, along with the DMA and cryptography (Crypto) bus masters, presents unique opportunities for system-level design and performance optimization in a single MCU. With two CPUs you can:

- Allocate tasks to CPUs so that multiple tasks may be done at the same time
- Allocate resources to CPUs so that a CPU may be dedicated to managing those resources, thus improving efficiency
- Enable and disable CPUs to minimize power draw
- Send data between the CPUs using the IPC block. For more information, see code example CE216795, PSoC 6 MCU Dual-CPU Basics.

In one example application, the Cortex-M0+ CPU (CM0+) can “own” and manage all communication channels. The Cortex-M4 CPU (CM4) can send and receive messages from the channels via CM0+. This frees CM4 to do other tasks while CM0+ manages the communication details.

1.1 How to Use this Document

This document assumes that you are familiar with PSoC 6 MCU architecture, and application development for PSoC devices using either Cypress’ ModusToolbox IDE or PSoC Creator IDE. For an introduction to PSoC 6 MCU, see the following:

- A PSoC 6 MCU device datasheet
- AN221774, Getting Started with PSoC 6 MCU
- AN210781, Getting Started with PSoC 6 MCU with Bluetooth Low Energy (BLE) Connectivity

If you are new to ModusToolbox IDE, see the ModusToolbox IDE home page. Some PSoC 6 MCU devices are not supported by PSoC Creator; ModusToolbox IDE must be used.

If you are new to PSoC Creator, see the PSoC Creator home page. Use PSoC Creator version 4.2 or higher for PSoC 6 MCU-based designs.

Initial sections of this application note cover general concepts for dual-CPU MCUs and how they are implemented in PSoC 6 MCU. To skip to an overview of creating a ModusToolbox or PSoC Creator project for a PSoC 6 dual-CPU MCU, go to the PSoC 6 MCU Dual-CPU Development section.

2 General Dual-CPU Concepts

The process of developing firmware for a dual-CPU MCU is similar to that for a single-CPU MCU, except that you write code for two CPUs instead of one. You should also consider any need for inter-processor communication.

Performance: The main advantage of having two CPUs is that you essentially multiply your CPU power and bandwidth. With PSoC 6 MCUs, that increased bandwidth comes at a price that is frequently on par with single-CPU MCUs. How to use that increased bandwidth depends on the tasks that your application must perform:

- Single task: A single-task application may be less of a fit for a dual-CPU MCU unless the application is large and complex. In PSoC 6 MCU, you can execute the task on one of the CPUs and put the other CPU to sleep to reduce power.

- Dual task: This is the most obvious fit; assign each task to a CPU. Assign the task with larger computing requirements to the higher-performance CPU, i.e., Cortex-M4 in PSoC 6 MCU.

- Multiple tasks: Again, assign each task to a CPU. In each CPU, you must include a method for executing each task in a timely fashion.
RTOS: A complex multitasking system may be managed by a real-time operating system (RTOS). An RTOS basically allocates a number of CPU cycles to each task, depending on the task priority or whether a task is waiting for an event. You effectively do that yourself by assigning tasks to the CPUs. Some examples of dual-CPU RTOS architectures are:

- Each CPU has its own RTOS and its own set of tasks. Each RTOS should include a task to manage communications with the other CPU.
- Only one CPU has an RTOS and multiple tasks. The other CPU is idle until it is messaged to do a specified task. It then wakes up and does the task, then messages the result back to the first CPU. As an example, the lower-performance CPU, CM0+ in PSoC 6 MCU, can use the higher-performance CPU, CM4 in PSoC 6 MCU, to do computation-intensive tasks when needed.

Power: In a dual-CPU system, firmware can start and stop the CPUs to fine-tune power usage. In the previous example, to reduce power, the high-performance CPU is placed into a sleep state until needed for a computation-intensive task.

Debug: Debugging two bodies of code at the same time may be a complex process. Usually you debug code for one CPU, then debug code for the other CPU. In addition, a device such as an oscilloscope or a logic analyzer may be useful for monitoring communication between the CPUs.
3 PSoC 6 MCU Dual-CPU Architecture

Figure 1 on page 1 shows the overall dual-CPU architecture in PSoC 6 MCU. (For detailed block diagrams of PSoC 6 MCU, see the device datasheet or AN221774.) Specific features and other details related to dual CPUs are listed in this section. For more information, see the Arm documentation sets for Cortex-M4 and Cortex-M0+, and the PSoC device technical reference manual (TRM).

- **CPUs:** Both CPUs – Cortex M4 and Cortex M0+ – are 32-bit. CM4 runs at up to 150 MHz and has a floating-point unit (FPU). CM0+ runs at up to 100 MHz.

  CM4 is the main CPU. It is designed for a short interrupt response time, high code density, and high throughput. The CM0+ CPU is secondary; it is used in PSoC 6 MCU to implement system calls and device-level security, safety, and protection features. CM0+ is also recommended for functions such as BLE communications and CapSense. In one example, the PSoC Creator BLE Component can be configured to run on either or both CPUs, as Figure 2 shows:

  Figure 2. BLE Configuration for Execution on Multiple CPUs, Using ModusToolbox IDE

- **Performance:** CM0+ typically operates at a slower clock speed than CM4. The CM0+ instruction set is more limited than that of CM4. Therefore, it may require more cycles to implement a function on CM0+, and the cycle time is longer. Keep this in mind when deciding to which CPU to allocate tasks.

- **Security:** PSoC 6 MCU has several security features; see the TRM for details. To meet security requirements, CM0+ is used as a “secure CPU”. It is considered to be a trusted entity; it executes Cypress system code and application code. The use of CM0+ for system and security tasks may limit its availability for applications. For more information on secure systems, see AN221111, Creating a Secure System.

  Device system calls may be initiated by either CPU, but are always executed by CM0+.

- **Startup sequence:** After device reset, only CM0+ executes; CM4 is held in a reset state. CM0+ first executes Cypress system and security code, including SROM code, FlashBoot, and Secure Image. For more information on these code modules, see AN221111, Creating a Secure System.

  After CM0+ executes the system and security code, it executes the application code. In the application code, CM0+ may release the CM4 reset, causing CM4 to start executing its application code. ModusToolbox IDE and PSoC Creator both auto-generate code in CM0+ main() to release the CM4 reset.

- **Inter-processor communication (IPC):** IPC enables the CPUs to communicate and synchronize activities. The IPC hardware contains register structures for IPC channel functions and IPC interrupts. IPC channel registers implement mutual exclusion (mutex) lock/release mechanisms, and messaging between the CPUs. IPC interrupt registers generate interrupts to both CPUs for messaging events and lock/release events.

- **Interrupts:** Each CPU has its own set of interrupts. All peripheral interrupt lines are hard-wired to specific CM4 interrupt inputs. Peripheral interrupts are also multiplexed to CM0+’s limited set of 32 interrupt inputs (8 interrupt inputs in the PSoC 6A-2M device family). See Interrupt Assignment Considerations.
Power modes: PSoC 6 MCU has several power modes that can affect either the entire system or just a single CPU. CPU power modes are Active, Sleep, and Deep Sleep as defined by Arm. Device system power modes are LP, ULP, Deep Sleep, and Hibernate.

- System Low Power (LP) mode is the default operating mode of the device after reset. It provides the maximum performance. While in System LP mode, the CPUs may operate in any of the Arm-defined modes.
- System Ultra Low Power (ULP) mode is identical to LP mode with a performance tradeoff made to achieve a lower system current. This tradeoff lowers the core operating voltage, which then requires a reduced operating clock frequency, and limited high-frequency clock sources. While in system ULP mode, the CPUs may operate in any of the Arm-defined modes.
- In System Deep Sleep mode, all high-speed clock sources are OFF. This in turn stops both CPUs and makes high-speed peripherals unusable. However, low-speed clock sources and peripherals continue to operate, if configured and enabled by the firmware. Interrupts from these peripherals cause the device to return to System LP or ULP mode and one or more CPUs to wake up to Active mode. Each CPU has a Wakeup Interrupt Controller (WIC) to wake up the CPU.
- System Hibernate mode is the lowest power mode of the device. It is intended for applications that may go into a dormant state. The device goes through a reset on wakeup from Hibernate. See Startup Sequence.
- In CPU Active mode, the CPU executes code and all logic and memory is powered. The device must be in System LP or ULP mode.
- In CPU Sleep mode, the CPU clock is turned OFF and the CPU halts code execution. The device must be in System LP or ULP mode.
- In CPU Deep Sleep mode, the CPU requests the device to go into System Deep Sleep mode. When the device is ready, it enters System Deep Sleep mode. In PSoC 6 MCU, both CPUs must enter CPU Deep Sleep before the system transitions to Deep Sleep. If only one CPU has entered CPU Deep Sleep mode, the system remains in LP or ULP mode.

For more information on PSoC 6 MCU power modes, see AN219528, PSoC 6 MCU Low Power Modes and Power Reduction Techniques.

Debug: PSoC 6 MCU has a Debug Access Port (DAP) that acts as the interface for device programming and debug. An external programmer or debugger (the "host") communicates with the DAP through the device Serial Wire Debug (SWD) or Joint Test Action Group (JTAG) interface pins. Through the DAP (and subject to device security restrictions), the host can access the device memory and peripherals as well as the registers in both CPUs.

Each CPU offers several debug and trace features as follows:
- CM4 supports six hardware breakpoints and four watchpoints, 4-bit embedded trace macrocell (ETM), serial wire viewer (SWV), and printf()-style debugging through the single-wire output (SWO) pin.
- CM0+ supports four hardware breakpoints and two watchpoints, and a micro trace buffer (MTB) with 4 KB dedicated RAM.

PSoC 6 MCU also has an Embedded Cross Trigger for synchronized debugging and tracing of both CPUs.

ModusToolbox IDE and some third-party IDEs support dual-CPU debugging, use; see Debug Considerations.
PSoC Creator supports debugging a single CPU (either CM4 or CM0+) at a time.
4 PSoC 6 MCU Dual-CPU Development

This section shows only those development aspects that are unique to PSoC 6 MCU dual-CPU devices. To learn more about PSoC 6 MCU, ModusToolbox IDE, or PSoC Creator, see one or more of the following:

- AN221774, Getting Started with PSoC 6 MCU
- AN210781, Getting Started with PSoC 6 MCU with Bluetooth Low Energy (BLE) Connectivity
- The ModusToolbox IDE home page. Some PSoC 6 MCU devices are not supported by PSoC Creator; ModusToolbox IDE must be used.
- The PSoC Creator home page. Use PSoC Creator version 4.2 or higher for PSoC 6 MCU-based designs.

4.1 ModusToolbox Instructions

ModusToolbox application development for a PSoC 6 MCU dual-CPU device is similar to that for any other device supported by ModusToolbox IDE. Click New Application in the Quick Panel. Then, in the Choose Target Hardware dialog, click Custom Hardware for a list of PSoC 6 MCU devices, as Figure 3 shows. Review the device family datasheets to find devices with dual CPUs.

Figure 3. Device Selection in ModusToolbox IDE

Instead of a device, you can choose a kit in the Choose Target Hardware dialog. In the PSoC 6 BLE Pioneer Kit CY8CKIT-062-BLE, the PSoC 6 MCU dual-CPU device part number is CY8C6347BZI-BLD53. In the PSoC 6 WiFi-BT Pioneer Kit CY8CKIT-062-WiFi-BT, the PSoC 6 MCU dual-CPU device part number is CY8C6247BZI-D54.
Complete the New Application dialogs, and a new application is created in the selected workspace. For dual-CPU devices, five projects are created for the application. You can view the projects in the Project Explorer window, as Figure 4 shows. Two of the projects support the CM4 and two others support the CM0+ (note the ".cm0p" in the project folder name).

![Figure 4. ModusToolbox Projects for a Dual-CPU Application](image)

Two projects support Cypress’ Peripheral Driver Library (PDL). PDL drivers abstract hardware functions into a set of easy-to-use APIs. For more information on the PDL, select Help > ModusToolbox API Reference > PSoC PDL API Reference.

When the application is created, source code files and folders are automatically created, as Figure 5 shows. Among other files, each CPU project has a main.c, a linker script file (.ld), and a make file modus.mk.

The _config project shown in Figure 5 includes a design.modus file. Implement your hardware design by opening and editing this file. When design entry is complete, save the file (and, optionally, close the Device Configurator window). This updates several system source code files in multiple projects.
4.2 PSoC Creator Instructions

PSoC Creator project development for a PSoC 6 MCU dual-CPU device is similar to that for any other device supported by PSoC Creator. To create a new project, select File > New > Project. A Create Project dialog is displayed, similar to Figure 6.

Select Target Device (A), and PSoC 6 (B). On the pull-down list (C), select <Launch Device Selector...> to see a list of PSoC 6 devices.

Figure 6. PSoC Creator Create Project Dialog

Figure 7 shows the Device Selector dialog. To see a list of dual-CPU devices, click the CPU category (D) and select only CortexM0p, CortexM4.

In the PSoC 6 BLE Pioneer Kit CY8CKIT-062-BLE, the PSoC 6 MCU dual-CPU device part number is CY8C6347BZI-BLD53. In the PSoC 6 WiFi-BT Pioneer Kit CY8CKIT-062-WIFI-BT, the PSoC 6 MCU dual-CPU device part number is CY8C6247BZI-D54.

Figure 7. PSoC Creator Device Selector Dialog
After selecting a PSoC 6 MCU part, the rest of the project creation process is the same as for other devices. Click through the rest of the Create Project dialogs; PSoC Creator creates the project.

The initial project windows layout (Figure 8) includes a Workspace Explorer window with the following features for dual-CPU devices:

1. Separate main.c files – main_cm0p.c and main_cm4.c – for each CPU. Sources in the folders CM0p (Core 0) and CM4 (Core 1) are compiled into separate binaries for the respective CPUs.
2. A Shared Files folder. Source files in this folder are compiled into both binaries.

Figure 8. PSoC Creator Initial Project Layout for Dual-CPU Devices

The initial project layout also includes a TopDesign hardware schematic, along with an associated Component Catalog window.

After the project is created, implement your hardware design by dragging Components onto the schematic, and configuring and wiring them.
When schematic design entry is complete, select **Build > Generate Application**. This creates several system source code files and folders in the existing folders as well as in the new folder **Generated Source**, as **Figure 9** shows.

The generated source contains drivers for each Component on the schematic, as well as Cypress’ Peripheral Driver Library (PDL). The PDL is a software development kit (SDK) that integrates device header files, startup code, and peripheral drivers. The peripheral drivers abstract the hardware functions into a set of easy-to-use APIs.

For more information on the PDL, select PSoC Creator **Help > Documentation > Peripheral Driver Library**. Also, each Component has a datasheet that documents the driver API for that Component. Right-click the Component and select **Open Datasheet**....

PSoC Creator creates several other files and folders, and places them in existing folders **CM0p (Core 0)**, **CM4 (Core 1)**, and **Shared Files**. These files generally support configuration, startup, and linking options for PSoC Creator as well as other IDEs. For more information on these files, see PSoC Creator Help article, **Generated Files (PSoC 6)**.
4.3 Resource Assignment Considerations

All PDL driver source and other API files are available to both CPUs. If code for a CPU references any API element in a generated source file, that file is compiled into the binary for that CPU. The same file can be compiled into both binaries – see code example CE216795, PSoC 6 MCU Dual-CPU Basics.

If the same source file is compiled into both binaries, a function in that file is duplicated in both binaries. Even though the copies are in separate builds and binaries, in some cases it is convenient to consider a function to be executed simultaneously by both CPUs.

As noted previously, it is possible for a peripheral to be accessed by both CPUs; for example, both CPUs may send data through the same UART. Generally, PDL driver functions are “CPU-safe”, that is, these functions can effectively be executed simultaneously by both CPUs. However, you should make design decisions about assigning resources to each CPU. There are two ways to do this:

- **Dedicate a resource to one CPU.** Include code to use the resource only in the firmware for the desired CPU. Also, if you are using PSoC Creator, a good practice is to indicate on the project schematic the CPU that “owns” the resource, as Figure 10 shows.

![Figure 10. PSoC Creator Project Schematic for Dual CPUs Controlling Separate Pin Components](image)

- **Share resources between the CPUs.** Code example CE216795 shows how the PSoC 6 MCU’s IPC block may be used to implement a mutex to share memory between the CPUs. Use the same technique to share a peripheral resource such as a UART.

Flash and SRAM that are allocated in a CPU’s binary are generally separate from that for the other CPU. If custom sections and section placement are defined in the CPUs’ linker scripts, you must ensure that the sections do not overlap. Conversely, another way to share memory is to define for each CPU custom sections with the same address.

**Note:** If you have both CPUs controlling pins on the same GPIO port, use only the following API functions to change the pin outputs: Cy_GPIO_Write(), Cy_GPIO_Set(), Cy_GPIO_Clr(), and Cy_GPIO_Inv(). For more information, see the GPIO API reference in the PDL documentation.
4.4 Interrupt Assignment Considerations

An important consideration for dual-CPU designs is assigning and handling interrupts. As noted previously, all device interrupts are available to CM4, and a subset of interrupts are routed through multiplexers to CM0+. You must decide which CPU will handle each interrupt.

For more information, see application note AN217666, *PSoC 6 MCU Interrupts*.

**ModusToolbox IDE:** In ModusToolbox IDE, interrupts are assigned programmatically; at this time, there is no GUI support. Examples of the required code are in AN217666; and in the PDL documentation, Cypress BLE Middleware Library section, Configure BLESS Interrupt subsection. The code in this subsection shows how to assign PSoC 6 MCU BLE subsystem (BLESS) interrupts to either CM0+ or CM4. The code can be easily modified to support other interrupt sources.

**PSoC Creator:** For PSoC Creator, let us assign interrupts in an example design. Figure 11 shows a design with two interrupts; one from a PWM Component, connected to an Interrupt Component MyPWM_Int; and the other from an I2C Component.

In the Design Wide Resources window (file type .cydwr), select the Interrupts tab to see all of the interrupts in the design, as Figure 12 shows.

In this example, the I2C Component has an interrupt embedded in it. That interrupt is not shown on the schematic in Figure 11; it is shown in the Design-Wide Resources window as MyI2C_SCB_IRQ.

Check or uncheck the boxes in the ARM CM0+ Enable and ARM CM4 Enable columns to assign interrupts to the respective CPUs.

Each peripheral interrupt is hard-wired to CM4, so the Interrupt Number is automatically assigned by PSoC Creator when you build the project. Because interrupts are routed through multiplexers to CM0+, you can select an ARM CM0+ Vector for each interrupt.

**Note:** A warning symbol and tooltip are displayed if an interrupt is assigned to both CPUs. This is generally not recommended; however an interrupt can be used to wake up one or both CPUs from their Sleep modes.
4.5 Debug Considerations

ModusToolbox IDE and third-party IDEs such as Keil µVision and IAR Embedded Workbench support dual-CPU debugging. PSoC Creator supports debugging a single CPU (either CM4 or CM0+) at a time.

4.5.1 ModusToolbox Instructions

ModusToolbox IDE supports dual-CPU debugging. For detailed instructions, click Help > ModusToolbox IDE Documentation > User Guide, and see the subsection PSoC 6 MCU Dual-Core Debugging. To switch between the CPUs for debug operations, in the Debug tab, click within the Thread corresponding to the desired CPU, as Figure 13 shows:

![Figure 13. ModusToolbox IDE Dual-CPU Debug Windows](image)

Click the Registers tab to see the registers for the currently selected CPU. You can observe and modify memory from either CPU.

4.5.2 PSoC Creator Instructions

PSoC Creator supports debugging just one CPU at a time. Before starting a debug session with PSoC 6 MCU, select the desired debug target (Debug > Select Debug Target...), as Figure 14 shows. Select the desired CPU and click OK / Connect. To debug the other CPU, you must exit the debugger and then re-enter it with a connection to that CPU.

![Figure 14. PSoC Creator Select CPU for Debug](image)
**Recommended**: develop and debug first the portions of code where the CPUs communicate with each other. After that, code executed by an individual CPU can be debugged separately. For example, when the shared memory project in CE216795 was developed, the portion where CM0+ sends an initial message to CM4 was developed and debugged before subsequent portions of code were developed.

You can debug both CPUs simultaneously by using other IDEs such as µVision or IAR. To do so, you must export your PSoC Creator project to the other IDE. PSoC Creator documents this topic in the help articles *Integrating into 3rd Party IDEs, PSoC 6 Designs*. Review the instructions in the help articles; the general steps are summarized in the following sections.

### 4.5.3 Instructions for Other IDEs

IDEs such as Keil µVision and IAR Embedded Workbench support dual-CPU debugging. If you are using PSoC Creator, you can export your project to one of these IDEs for dual-CPU debugging. Following are the steps to do so:

1. **Configure the PSoC Creator Project**
2. **Create µVision Projects**
3. **Debug µVision Projects**
4. **Create IAR-EW Projects**
5. **Debug IAR-EW Projects**

#### 1. Configure the PSoC Creator Project

Update the *Target IDEs* settings in the project Build Settings, as Figure 15 shows.

For µVision, select **CMSIS Pack**: > Generate. Enter appropriate identifying text for the CMSIS pack in the *Vendor*, *Pack*, and *Version* fields.

**Recommended**: select **Toolchain**: > ARM MDK Generic.

For IAR, you only need to select **IAR EW-ARM**: > Generate. (An advanced option, Generate without copying PDL files, is also available.) Because IAR has its own compiler (not supported by PSoC Creator), the Toolchain selection is not relevant.

![Figure 15. Build Settings for Target IDEs](image)

Then build your PSoC Creator project in the usual manner. A folder *Export* is created in your `<project>-cydsn` folder, which contains relevant files for exporting to the selected IDE or IDEs.
For µVision, after the PSoC Creator project is built, find the corresponding .pack file in the folder Export \ Pack. Double-click the file to install it as a µVision pack, as Figure 16 shows.

**Note:** Do not use the µVision Pack Installer Wizard File Import function to install this pack.

![Figure 16. Install µVision Pack from PSoC Creator Project](image)

**Note:** if you update the PSoC Creator project, consider changing the µVision pack version number (see Figure 15) and installing the new pack.

For more information, see AN219434 - Importing PSoC Creator Code into an IDE for a PSoC 6 Project

### 2. Create µVision Projects

For µVision, you must create two projects: one for each PSoC 6 MCU CPU: CM0+ and CM4. Do the following:

**Recommended:** create a new folder (e.g. uVisionBuild) within your PSoC Creator <project>.cydsn folder to store all µVision project files separately from the PSoC Creator files (this is different from the IAR instructions). Within that folder, create another new folder for CM4 object files (e.g., ObjectsM4), as Figure 17 shows:

![Figure 17. New Folders for µVision Projects](image)
Open µVision 5.25 or later, and create a new project (Project > New µVision Project...) in the uVisionBuild folder. **Recommended:** name the project based on the original PSoC Creator project name and the target CPU. For example, for the CE216795 dual-CPU blinky project, create a µVision project BlinkyM0p for the CM0+ CPU, as Figure 18 shows:

![Create New Project](image1)

Figure 18. Create a µVision Project for CM0+

After you click **Save**, a Select Device for Target ‘Target 1’... dialog box is displayed. The two PSoC 6 MCU CPUs that were defined in the previously installed pack (Figure 16) are displayed. Select the CM0+ CPU, as Figure 19 shows. Click **OK**.

![Select Device for Target](image2)

Figure 19. Select CM0+ as the Project Device
Next, a Manage Run-Time Environment dialog box is displayed. Click **Select Packs**, and uncheck **Use latest versions of all installed Software Packs**. Select the pack from the PSoC Creator project, as **Figure 20** shows:

**Figure 20. Select the PSoC Creator Project Pack**

Click **OK**; the Manage Run-Time Environment dialog changes as **Figure 21** shows. Select the Device Startup and PDL Drivers, and click **OK**. The project is created, with a Target 1, a Source Group 1, and Device startup and PDL files, as **Figure 22** shows.

**Figure 21. Select Pack Startup and PDL Driver Files**

**Figure 22. Initial Project Creation**
Right-click **Source Group 1**, and select **Add Existing Files to Group ‘Source Group 1’**…. Navigate to your PSoC Creator project folder and select `main_cm0p.c`, `cy_ipc_config.c`, and all other non-system `.c` and assembler files needed for your project, as Figure 23 shows. You do not have to add any `.h` files, startup, or system `.c`, or assembler files. Click **Add**; the files are added to the source group in the µVision project. Click **Close**.

Figure 23. Add PSoC Creator Project C Source Files to the Source Group

Now that the project is created, you must set its options. Right-click **Target 1**, and select **Options for Target ‘Target 1’**…. Confirm in the **Target** tab that the device, CPU, IROM1, and IRAM1 are correct for your PSoC 6 MCU device, as Figure 24 shows. Updating other fields such as Xtal (MHz) and Operating system is optional.

Figure 24. Project Target Options
In the **User** tab, verify that the correct post-build batch file from the pack is being called. Hover the cursor over the **User Command** field and confirm that `postbuildCortexm0p.bat` is called, as Figure 25 shows. Add other pre- and post-build batch files, and select other options, as needed.

**Figure 25. Project User Options**

![Project User Options](image)

Confirm in the **C/C++** tab that the **C99 mode** option is checked, as Figure 26 shows. (PDL is developed based on C99.) Add the PSoC Creator `<project>.cydsn` folder to the **Include Paths**; this provides a link to the `.h` files in the PSoC Creator project. Update other options and fields as needed.

**Figure 26. Project C/C++ Options**

![Project C/C++ Options](image)
Confirm in the **Linker** tab that the **R/O Base** and **R/W Base** fields are correct for your PSoC 6 MCU device, as Figure 27 shows. Select the appropriate **Scatter File** from your PSoC Creator project folder.

Figure 27. Project Linker Options

Connect the CY8CKIT-062-BLE USB port to your computer. Press the kit button SW3 to put KitProg2 into CMSIS-DAP mode; see the kit guide for details. This allows debugging without using any external probes.

In the **Debug** tab, select **Use CMSIS-DAP Debugger**, as Figure 28 shows. Click **Settings**, select **KitProg2 CMSIS-DAP**, and confirm that all other settings are at the defaults shown. Click **OK** and go back to the Options dialog.

Figure 28. Project Debug Options
In the **Utilities** tab, confirm that Use Debug Driver is checked, and then uncheck **Update Target before Debugging**. Click **Settings**, and uncheck all **Download Function** boxes, as Figure 29 shows. Click **Do not Erase**. Click **OK** and go back to the **Options** dialog. A warning "Nothing to do ... " is displayed; click **OK**. The application will be loaded by the CM4 project. Click **OK** to save and close the options settings.

![Figure 29. Project Utilities Options](image)

Repeat the previous steps and create a second project for CM4.

**Recommended:** name the project based on the original PSoC Creator project name and the target CPU. For example, for the CE216795 dual-CPU blinky project, create µVision project *BlinkyM4p*; see Figure 18. Configure the project in the same manner as the CM0+ project, with the following differences:

- The CM4 project must be in the same folder as the CM0+ project; in this case, *uVisionBuild*. See Figure 18.
- Select the CM4 CPU from the previously installed pack; see Figure 19.
- Navigate to your PSoC Creator project folder and select *main_cm4.c*, *cy_ipc_config.c*, and all other non-system *.c* and assembler files needed for your project, as Figure 23 shows. You do not have to add any *.h* files, startup, system *.c*, or assembler files.
- In the **Options** dialog, **Output** tab, click **Select Folder for Objects...**, and select the *ObjectsM4* folder that you created; see Figure 17.
- In the **Options** dialog, **C/C++** tab, add **--fpu=fpv4-sp** to **Misc Controls**; see Figure 26.
- In the **Options** dialog, **Linker** tab, select the "cm4_dual" scatter file, as Figure 30 shows. The CM4 project will contain code for both CPUs. Add `-fpv4 -sp` to Misc Controls.

Figure 30. Linker Options for CM4 Project

- In the **Options** dialog, **Debug** tab, Target Driver Setup, select VECTRESET for the Reset option; see Figure 28.

- In the **Options** dialog, **Utilities** tab, confirm that **Update Target before Debugging** is checked, as Figure 31 shows. Set the RAM for Algorithm values as indicated. Checking **Reset and Run** is optional but convenient.

Figure 31. Utilities Options for CM4 Project
Finally, create a µVision workspace (Project > New Multi-Project Workspace...), named for example Blinky, in the uVisionBuild folder. Add the two created projects to that workspace. The created workspace and projects, and the corresponding files, should be similar to Figure 32.

Figure 32. Resultant µVision Project Window and Project Files

Build the projects in sequence; build the CM0+ project first. Note that µVision has a batch build feature to automate the process. After building is successfully completed, right-click the BlinkyM4 project and set it as the active project. Then test your build options by (1) erasing flash (Flash > Erase), and (2) downloading the project (Flash > Download) and confirming correct operation. If you did not select Reset and Run (see Figure 31), you must press the kit reset button (RST / SW1) to start operation.

Note: If you change any code in the CM0+ project, you must rebuild both projects. Note that µVision has a batch build feature to automate the process.
3. Debug µVision Projects

Start debugging with the CM4 project – downloading the CM4 project installs code for both CPUs. Set the CM4 project as the active project, download it if needed, and click Debug > Start/Stop Debug Session to start debugging. The µVision window appears similar to Figure 33:

Figure 33. CM4 Debug Window

If you are running the CE216795 dual-CPU blinky project, set a breakpoint at line 63, Cy_Syslib_Delay(). Then repeatedly click Debug > Run, and the red LED toggles on each stop at the breakpoint.
Now open a second instance of µVision and load the same workspace. Both instances share the kit connection and the PSoC 6 MCU debug access port (DAP). Make the CM0+ project active, and start a debug session. Set a breakpoint at line 63, `Cy_Syslib_Delay()`. Then repeatedly click **Debug > Run**, and the blue LED toggles on each stop at the breakpoint.

**Note:** Executing the `Cy.SysEnableCM4()` function call at line 55 causes CM4 to start running again. Go to the CM4 window, click **Debug > Stop**, then **Debug > Run**. CM4 runs to the breakpoint again.

It helps to place the instance windows side by side on your desktop. The windows appear similar to Figure 34. Click in the appropriate window to perform a debug operation on the desired CPU. Note that breakpoints can be set separately for each CPU. You can read and update the same memory addresses from either window.

**Figure 34. µVision Dual-CPU Debugging**
4. Create IAR-EW Projects

For IAR Embedded Workbench (IAR-EW), you must create two projects: one for each PSoC 6 MCU CPU: CM0+ and CM4. Do the following:

**Note:** The IAR-EW project files should be created in your PSoC Creator `<project>.cydsn` folder. Do not create a separate folder within your PSoC Creator `<project>.cydsn` folder (this is different from the µVision instructions). **Recommended:** add a tag such as “IAR_” to each project and workspace file name, to distinguish the IAR-EW files from the PSoC Creator files in the same folder.

Open IAR Embedded Workbench for ARM 8.22 or later, and create a new project (**Project > Create New Project...**). In the Create New Project dialog (**Figure 35**), confirm that the Tool chain is **Arm**, select the **Empty project** template, then click **OK**.

**Figure 35. IAR Embedded Workbench Create New Project Dialog**

**Recommended:** in the Save As dialog (**Figure 36**), name the project based on the original PSoC Creator project name and the target CPU. For example, for the **CE216795** dual-CPU blinky project, create a µVision project **IAR_BlinkyM0p** for the CM0+ CPU.

**Figure 36. Create an IAR Embedded Workbench Project for CM0+**
Select Tools > Options and make sure that Enable project connections is checked. Click OK. Then select Project > Add Project Connection.... In the next dialog, select Connect using IAR Project Connection, and click OK. Then select the ...

Figure 37. Select IAR Project Connection File from PSoC Creator Project Export Folder

Now that the project is created, you must set its options. Right-click the project, and select Options.... Confirm in the Options dialog, Build Actions section that postbuildCortexM0p.bat is called, as Figure 38 shows.

Figure 38. Select PSoC Creator Post-Build Batch File
In the Debugger section, Setup tab, select the CMSIS DAP driver. In the Download tab, check Suppress download. In the CMSIS DAP section, Setup tab, set Reset to Disabled (no reset). The application will be loaded by the CM4 project. In the Interface tab, select SWD. Click OK.

Repeat the previous steps and create a second project for CM4. Recommended: name the project based on the original PSoC Creator project name and the target CPU. For example, for the CE216795 dual-CPU blinky project, create IAR-EW project IAR_BlinkyM4; see Figure 36. Configure the project similar to the CM0+ project, with the following differences:

- The CM4 project must be in the same folder as the CM0+ project; in this case, your PSoC Creator <project>.cydsn folder. See Figure 36.
- Select the ...CortexM4.ipcf file; see Figure 37.
- In the Options dialog, General Options section, Output tab, change the output directories for object and list files, as Figure 39 shows. Do not change the executables/libraries output folder.

In the Build Actions section, confirm that postbuildCortexM4.bat is called, see Figure 38.

In the Debugger section, Setup tab, select the CMSIS DAP driver. In the CMSIS DAP section, Setup tab, confirm that Reset is set to System (default). In the Interface tab, select SWD. Click OK.
Select File > Save All. All files for both projects are saved, and a workspace file is automatically generated. In the Save Workspace As dialog, create an IAR-EW workspace, named for example IAR_Blinky, in your PSoC Creator <project>.cydsn folder. The created workspace and projects, and the corresponding files, should be similar to Figure 40. The files and folders generated by IAR-EW are highlighted.

Figure 40. Resultant IAR Embedded Workbench Project Window and Project Files

Connect the CY8CKIT-062-BLE USB port to your computer. Press kit button SW3 to put KitProg2 into CMSIS-DAP mode; see the kit guide for details. This allows debugging without using any external debug probes.

Build the projects in sequence; build the CM0+ project first. Note that IAR-EW has a batch build feature to automate the process. After building is successfully completed, right-click the BlinkyM4 project and set it as the active project. Then confirm that your build options are correct, by (1) erasing flash (Project > Download > Erase memory), and (2) downloading the project (Project > Download > Download active application) and confirming correct operation. After downloading, press the kit reset button (RST / SW1) to start operation.

Note: When erasing flash, you typically only need to erase PSoC 6 MCU application flash (0x1000 0000 – 0x100F FFFF), as Figure 41 shows:

Figure 41. IAR Embedded Workbench Erase Memory Dialog for PSoC 6 MCU

Note: If you change any code in the CM0+ project, you must rebuild both projects. Note that IAR-EW has a batch build feature to automate the process.
5. Debug IAR-EW Projects

Reopen the options for the CM4 project, and go to the Debugger section, Multicore folder. PSoC 6 MCU has different cores, i.e., CM0+ and CM4, which is referred to as "asymmetric multicore". Therefore, fill in the fields in the Asymmetric multicore section as Figure 42 shows. Checking Enable multicore master mode makes the CM4 the master for download and debugging purposes. Do not change the Port.

![Figure 42. Set Up Multicore Debugging](image)

Select File > Save All to save the project options changes. Then start debugging by selecting either Project > Download and Debug or Project > Debug without Downloading. A second (slave) instance of IAR Embedded Workbench is automatically opened for the CM0+ project. Both instances share the kit connection and the PSoC 6 MCU debug access port (DAP).

In the slave instance, set a breakpoint at line 63, Cy_Syslib_Delay(). Then repeatedly click Debug > Go, and the blue LED toggles on each stop at the breakpoint.

Click anywhere in the CM4 instance window and repeat the process. The red LED toggles on each stop at the breakpoint.
It helps to place the instance windows side by side on your desktop. The windows appear similar to Figure 43. Click in the appropriate window to perform a debug operation on the desired CPU. Note that breakpoints can be set separately for each CPU. You can read and update the same memory addresses from either window.

Figure 43. IAR Embedded Workbench Dual-CPU Debugging

You can stop debugging in either window; debugging is ended for both CPUs. Press the kit reset button (RST / SW1) to restart kit operation.
5 Summary

This application note has shown how to use and optimize your firmware and hardware designs for the dual-CPU feature in PSoC 6 MCUs.

Another way to optimize your PSoC 6 MCU design is based on the fact that the PSoC family devices are designed to be flexible, and enable you to build custom functions in programmable analog and digital blocks. For example, PSoC 6 MCU has the following peripherals that can act as “co-processors”:

- **DMA Controllers.** Note that the most common CPU assembler instructions output by C compilers are MOV, LDR, and STR, which implies that the CPU spends a lot of cycles just moving bytes around. Let the DMA controllers do that instead.
  
  **Note:** PSoC 6 MCU DMA controllers have an extensive set of features that enable you to construct complex data transfer and control systems that are independent of the CPUs. Software support of these features is provided in the ModusToolbox IDE Device Configurator, a PSoC Creator DMA Component, and an API in the PDL. For more information, see the Device Configurator Help Guide, the DMA Component datasheet, or the PDL documentation.

- **Crypto Block.** This block offers hardware acceleration for symmetric and asymmetric cryptographic methods (AES, 3DES, RSA, and ECC) and hash functions (SHA-512, SHA-256). It also has a true random number generator (TRNG) function. Software support for these features is provided by an API in the PDL; see the PDL documentation.

- **Universal Digital Blocks (UDBs).** There are as many as 12 UDBs, and each UDB has an 8-bit datapath that can add, subtract, and do bitwise operations, shifts, and cyclic redundancy check (CRC). Datapaths can be chained for word-wide calculations. Consider offloading CPU calculations to the datapaths.

- **UDBs** also have programmable logic devices (PLDs) which can be used to build state machines; see for example the Lookup Table (LUT) Component datasheet. LUTs can be an effective hardware-based alternative to programming state machines in the CPU, for example by using C switch / case statements.

  In addition, two GPIO ports include Smart IO, which can be used to perform Boolean operations directly on signals going to, and coming from, GPIO pins.

- Other smart peripherals include serial communication blocks (SCB), counter/timer/PWM blocks (TCPWM), Bluetooth Low Energy (BLE), I2S/PDM audio, programmable analog, CapSense®, and energy profiler. Use these peripherals to further offload processing from the CPUs.

PSoC Creator offers many Components and extensive APIs in the PDL for support of the peripherals’ functions. This allows you to develop an effective multiprocessing system in a single chip, offloading a lot of functionality from the CPUs. This in turn can not only reduce code size, but by reducing the number of tasks that the CPUs must perform, presents an opportunity to reduce CPU speed and power consumption.

For example, you can implement a digital system to control multiplexed ADC inputs, and interface with DMA to save the data in the SRAM to create an advanced analog data collection system with zero usage of the CPUs.

Cypress offers extensive application note and code example support for PSoC peripherals, as well as detailed data in the device datasheets, PDL documentation, and technical reference manuals (TRMs). For more information, see Related Documents.
6  Related Documents

For a comprehensive list of PSoC 6 MCU resources, see KBA223067 in the Cypress community.

<table>
<thead>
<tr>
<th>Application Notes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AN221774 – Getting Started with PSoC 6 MCU</td>
<td>Describes PSoC 6 MCU devices and how to build your first ModusToolbox or PSoC Creator project</td>
</tr>
<tr>
<td>AN210781  – Getting Started with PSoC 6 MCU with Bluetooth Low Energy (BLE) Connectivity</td>
<td>Describes PSoC 6 MCU with BLE Connectivity devices and how to build your first PSoC Creator project</td>
</tr>
<tr>
<td>AN217666, PSoC 6 MCU Interrupts</td>
<td>Describes PSoC 6 MCU interrupt architecture and how to configure interrupts</td>
</tr>
<tr>
<td>AN219434 – Importing PSoC Creator Code into an IDE for a PSoC 6 MCU Project</td>
<td>Describes how to import the code generated by PSoC Creator into your preferred IDE</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Code Examples</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CE216795 – PSoC 6 MCU Dual-CPU Basics</td>
<td>Demonstrates the two CPU cores in PSoC 6 MCU doing separate independent tasks, and communicating with each other using shared memory and the inter-processor communication (IPC) block.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PSoC Creator Component Datasheets</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interrupt</td>
<td>Supports generating interrupts from hardware signals</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Device Documentation</th>
<th>Description</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Development Kit Documentation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CY8CKIT-062-BLE</td>
<td>PSoC 6 BLE Pioneer Kit</td>
</tr>
<tr>
<td>CY8CKIT-062-WIFI-BT</td>
<td>PSoC 6 WiFi-BT Pioneer Kit</td>
</tr>
<tr>
<td>CY8CPROTO-063-BLE</td>
<td>PSoC 6 BLE Prototyping Kit</td>
</tr>
<tr>
<td>CY8CPROTO-062-4343W</td>
<td>PSoC 6 Wi-Fi Prototyping Kit</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Tool Documentation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ModusToolbox IDE</td>
<td>ModusToolbox IDE simplifies development for IoT designers. It delivers easy-to-use tools and a familiar microcontroller (MCU) integrated development environment (IDE) for Windows, macOS, and Linux.</td>
</tr>
<tr>
<td>PSoC Creator</td>
<td>PSoC Creator enables concurrent hardware and firmware editing, compiling and debugging of PSoC devices. Applications are created using schematic capture and over 150 pre-verified, production-ready peripheral Components. Look in the downloads tab for Quick Start and User Guides.</td>
</tr>
<tr>
<td>Peripheral Driver Library (PDL)</td>
<td>Installed by ModusToolbox IDE or PSoC Creator 4.2. Look in &lt;ModusToolbox install folder&gt;libraries/docs or, for PSoC Creator in &lt;PDL install folder&gt;doc, for the User Guide and the API Reference</td>
</tr>
</tbody>
</table>

About the Author

Name: Mark Ainsworth  
Title: Sr. Principal Applications Engineer  
Background: Mark Ainsworth has a BS in Computer Engineering from Syracuse University and an MSEE from the University of Washington, as well as many years of experience designing and building embedded systems.
Document History

Document Title: AN215656 - PSoC 6 MCU Dual-CPU System Design

Document Number: 002-15656

<table>
<thead>
<tr>
<th>Revision</th>
<th>ECN</th>
<th>Orig. of Change</th>
<th>Submission Date</th>
<th>Description of Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>**</td>
<td>5634375</td>
<td>MKEA</td>
<td>02/16/2017</td>
<td>New application note</td>
</tr>
<tr>
<td>*A</td>
<td>5653730</td>
<td>MKEA</td>
<td>03/08/2017</td>
<td>Updated template</td>
</tr>
<tr>
<td>*B</td>
<td>5777874</td>
<td>MKEA</td>
<td>06/09/2017</td>
<td>Updated text and screen shots for release versions of PSoC Creator 4.1 and PDL 3.0.0 Other miscellaneous edits</td>
</tr>
<tr>
<td>*C</td>
<td>5861685</td>
<td>MKEA</td>
<td>08/23/2017</td>
<td>Minor edits Ported to new application note document template Confidential tag removed</td>
</tr>
<tr>
<td>*D</td>
<td>6065641</td>
<td>MKEA</td>
<td>03/07/2018</td>
<td>Added a new Figure 2 Updated Figure 4 and associated kit device part number Updated Figures 6 and 9 for PSoC Creator 4.2 beta 2 Emphasized using CM0+ as a support CPU for tasks such as BLE and CapSense Added references to AN221111, Creating a Secure System; AN217666, PSoC 6 Interrupts; AN219528, PSoC 6 Low Power Modes; and CE216795, PSoC 6 Dual-CPU Updated power modes description Miscellaneous minor edits Ported to new application note template Changed the document title to PSoC 6 MCU Dual-CPU System Design</td>
</tr>
<tr>
<td>*E</td>
<td>6201597</td>
<td>MKEA</td>
<td>06/11/2018</td>
<td>Expanded section 4.3 to include dual-CPU debugging with µVision and IAR Embedded Workbench IDEs Updated power mode descriptions in section 3 Miscellaneous minor edits Ported to *Y application note template</td>
</tr>
<tr>
<td>*F</td>
<td>6403371</td>
<td>MKEA</td>
<td>12/06/2018</td>
<td>Added support for ModusToolbox.</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

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

PSoC® Solutions

| PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP | PSoC 6 MCU |

Cypress Developer Community

Community | Projects | Videos | Blogs | Training | Components

Technical Support

cypress.com/support

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

© Cypress Semiconductor Corporation, 2017-2018. This document is the property of Cypress Semiconductor Corporation and its subsidiaries, including Span silicon LLC (“Cypress”). This document, including any software or firmware included or referenced in this document (“Software”), is owned by Cypress under the intellectual property laws and treaties of the United States and other countries worldwide. Cypress reserves all rights under such laws and treaties and does not, except as specifically stated in this paragraph, grant any license under its patents, copyrights, trademarks, or other intellectual property rights. If the Software is not accompanied by a license agreement and you do not otherwise have a written agreement with Cypress governing the use of the Software, then Cypress hereby grants you a personal, non-exclusive, 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 ACOMPANYING HARDWARE, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. No computing device can be absolutely secure. Therefore, despite security measures implemented in Cypress hardware or software products, Cypress does not assume any liability arising out of any security breach, such as unauthorized access to or use of a Cypress product. In addition, the products described in these materials may contain design defects or errors known as errata which may cause the product to deviate from published specifications. To the extent permitted by applicable law, Cypress reserves the right to make changes to this document without further notice. Cypress does not assume any liability arising out of the application or use of any product or circuit described in this document. Any information provided in this document, including any sample design information or programming code, is provided only for reference purposes. It is the responsibility of the user of this document to properly design, program, and test the functionality and safety of any application made of this information and any resulting product. Cypress products are not designed, intended, or authorized for use as critical components in systems designed or intended for the operation of weapons, weapons systems, nuclear installations, life-support devices or systems, other medical devices or systems (including resuscitation equipment and surgical implants), pollution control or hazardous substances management, or other uses where the failure of the device or system could cause personal injury, death, or property damage (“Unintended Uses”). A critical component is any component of a device or system whose failure to perform can be reasonably expected to cause the failure of the device or system, or to affect its safety or effectiveness. Cypress is not liable, in whole or in part, and you shall and hereby do release Cypress from any claim, damage, or other liability arising from or related to all Unintended Uses of Cypress products. You shall indemnify and hold Cypress harmless from and against all claims, costs, damages, and other liabilities, including claims for personal injury or death, arising from or related to any Unintended Uses of Cypress products.

Cypress, the Cypress logo, Span silicon, the Span silicon 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.