AN215656

PSoC 6 MCU Dual-CPU System Design

Author: Mark Ainsworth

Associated Part Family: All PSoC® 6 MCU devices with dual CPUs

Associated Code Examples: CE216795

Related Application Notes: see Related Documents

More code examples? We heard you.

To access an ever-growing list of hundreds of PSoC code examples, please visit our code examples web page. You can also explore the Cypress video training library here.

AN215656 describes the dual-CPU architecture in PSoC 6 MCU, 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. The application note also shows how to build a simple dual-CPU design using the Cypress PSoC Creator™ Integrated Design Environment (IDE).

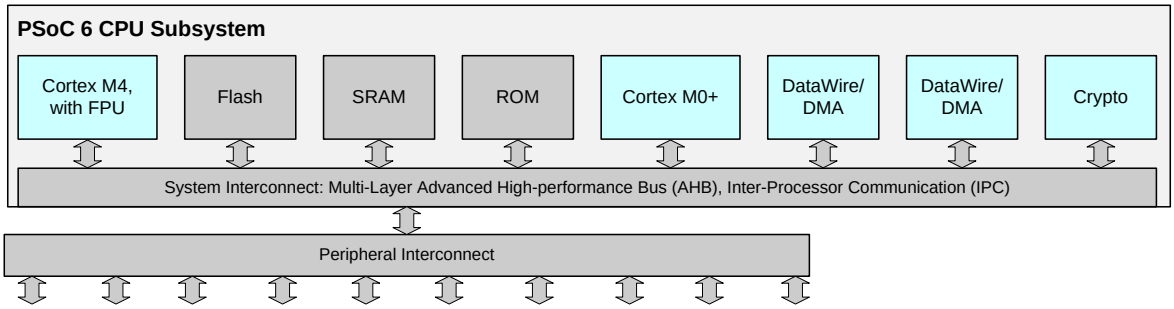
Contents

1 Introduction

PSoC 6 MCU is Cypress' new, 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. For more information, see a PSoC 6 MCU device datasheet or AN210781, *Getting Started with PSoC 6 MCU with Bluetooth Low Energy (BLE) Connectivity*.

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



Note: The contents of the block diagram in Figure 1 may vary depending on the device part number selected.

Generally, all memory and peripherals are shared by all of the 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 hardware semaphores and mutual exclusion (mutexes).

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*.

For example, 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 the PSoC 6 MCU architecture, and application development for PSoC using the Cypress PSoC Creator IDE. For an introduction to PSoC 6 MCU, see a PSoC 6 MCU device datasheet or AN210781, *Getting Started with PSoC 6 MCU with Bluetooth Low Energy (BLE) Connectivity*. If you are new to PSoC Creator, see the PSoC Creator home page.

Note: 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 PSoC Creator project for a PSoC 6 dual-CPU MCU, go to the PSoC 6 MCU Dual-CPU Development Process 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 a 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; simply assign each task to a CPU. Assign the task with larger computing requirements to the higher-performance CPU, e.g., the Cortex-M4 (CM4) CPU 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 just 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 (CPU 1) has an RTOS and multiple tasks. The other CPU (CPU 2) is idle until CPU 1 messages it to do a specified task. CPU 2 wakes up and does the task, then messages the result back to CPU 1. As an example, CPU 1 can be a lower-performance CPU, and it uses CPU 2, the higher-performance CPU, 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: It is difficult to debug two bodies of code at the same time. 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

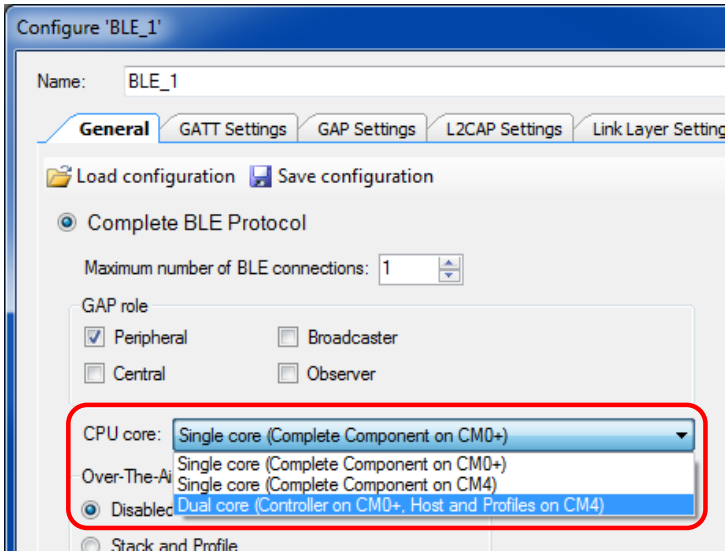
Figure 1 shows the overall dual-CPU architecture in PSoC 6 MCU. Specific features and other details are listed in this section. For more information, see the Arm documentation sets for Cortex-M4 and Cortex-M0+, and the PSoC device technical reference manual (TRM).

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

  Note: Some PSoC 6 MCU parts have only one CPU. See the device datasheet for details.

  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 Component Configuration for Execution on Multiple CPUs



- **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 both Cypress system code and user code. The use of CM0+ for system and security tasks may limit its availability for user 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 system and security code, it executes user code. In the user code, CM0+ may release the CM4 reset, causing CM4 to start executing its user code. PSoC Creator auto-generates 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 for IPC interrupts. The IPC channel registers implement mutual exclusion (mutex) lock and release mechanisms, and messaging between the CPUs. The IPC interrupt registers generate interrupts to both CPUs for messaging events and lock and release events.

- **Interrupts:** Each CPU has its own set of interrupts. A peripheral can route its interrupt output to either or both CPUs. 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. See Interrupt Assignment Considerations.

- **Power modes:** PSoC 6 MCU has several power modes: Active, Low-power Active (LPACTIVE), Sleep, Low-power Sleep (LPSLEEP), Deep Sleep, and Hibernate. CPU cores operate in each of the power modes as follows:

  - In Active and LPACTIVE modes, CPU cores execute code; all memory blocks and peripherals are available. Firmware may enable or disable specific peripherals and power domains. The device enters Active mode upon any device reset.

    LPACTIVE is similar to Active mode, with performance reductions for lower power; including reduced operating clock frequency, limited high-frequency clock sources, and lower core operating voltage.

  - In all other power modes, CPU clocks are turned OFF and the CPUs are in Sleep or Deep Sleep mode. Each CPU core supports its own sleep modes, independent of the state of the other CPU. The device is considered to be in Sleep or Deep Sleep mode when both cores are in Sleep or Deep Sleep mode, respectively.

    All peripherals available in Active modes are also available in the Sleep modes. Any peripheral interrupt, masked to the CPU, wakes up the CPU to Active mode.

    Only a subset of low-speed peripherals operate in Deep Sleep mode. Interrupts from these peripherals cause a CPU to wake up to Active mode. Each CPU has a Wakeup Interrupt Controller (WIC) to wake up the CPU from its Deep Sleep mode.

  The device wakes up from Hibernate mode through a device reset. CPU cores go through the startup sequence described previously.

  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. 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.

  PSoC Creator supports debugging a single CPU (either CM4 or CM0+) at a time. For dual-CPU debugging, use a third-party IDE and debugger support. For more information on debugging PSoC devices with PSoC Creator, refer to the PSoC Creator Help.

4 PSoC 6 MCU Dual-CPU Development Process

Note: This section shows only those aspects of the PSoC Creator development process that are unique to PSoC 6 MCU dual-CPU devices. If you are not familiar with PSoC 6 MCU or PSoC Creator, see AN210781, Getting *Started with PSoC 6 MCU with Bluetooth Low Energy (BLE) Connectivity*, or the PSoC Creator home page. Use PSoC Creator version 4.2 or higher for PSoC 6 MCU-based designs.

The PSoC Creator development process 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 3.

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 3. PSoC Creator Create Project Dialog

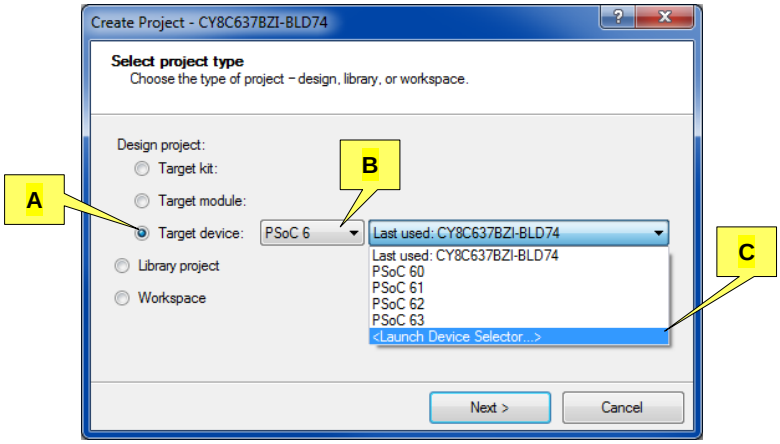
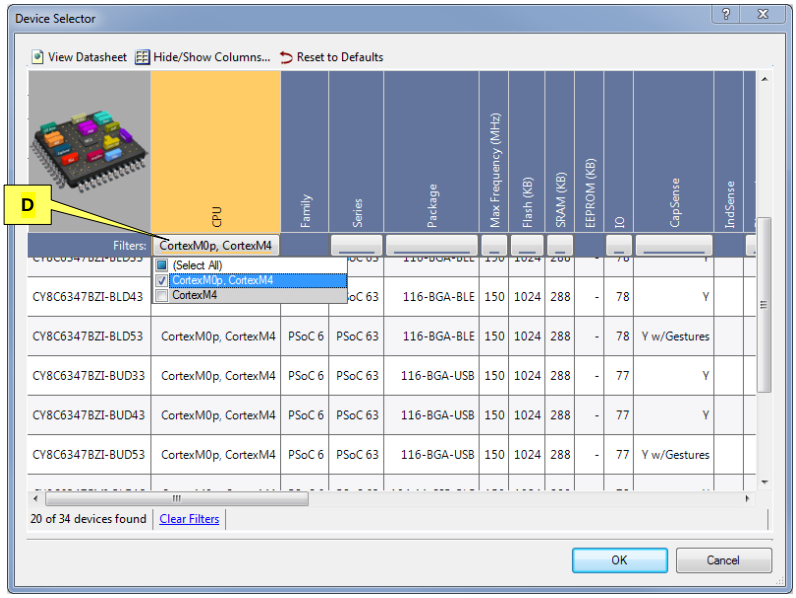


Figure 4 shows the Device Selector dialog. To see a list of dual-CPU devices, click the **CPU** category (D) and select only **CortexM0p, CortexM4**.

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

Figure 4. PSoC Creator Device Selector Dialog

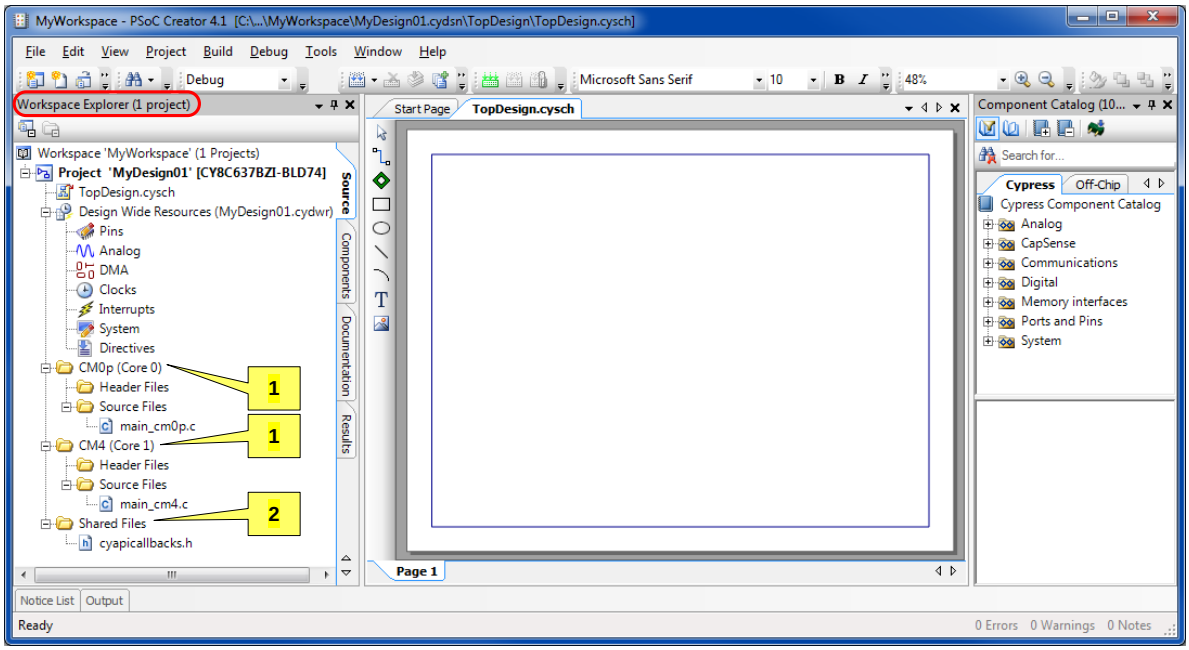


After selecting a PSoC 6 MCU part, the rest of the project creation process is the same as for other devices. Click through the rest of the **Create Project** dialogs; PSoC Creator creates the project.

The initial project windows layout (Figure 5) 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 5. 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.

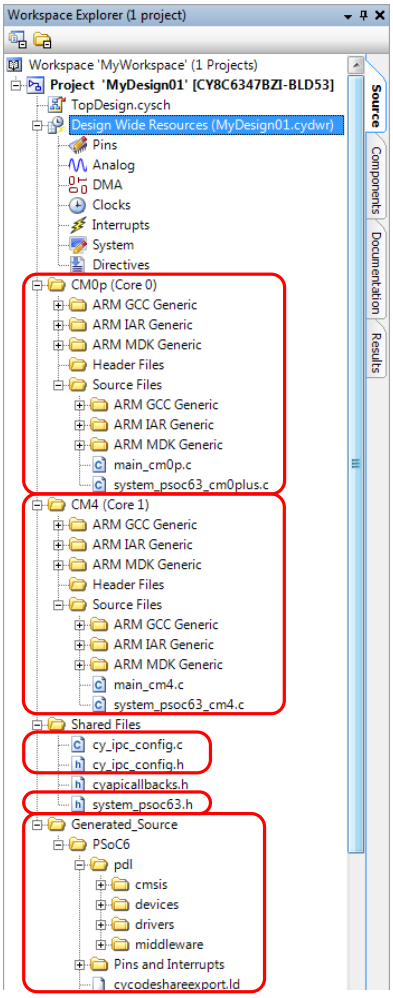
When schematic design entry is complete, select **Build** > **Generate Application**. This creates several system source code files and folders in the existing folders as well as in the new folder *Generated Source*, as Figure 6 shows.

The generated source contains drivers for each Component on the schematic, as well as the Cypress Peripheral Driver Library (PDL). The PDL is a software development kit (SDK) that integrates device header files, startup code, platform drivers, and peripheral drivers. The platform and 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)*.

Figure 6. Add Generated Source to a Project



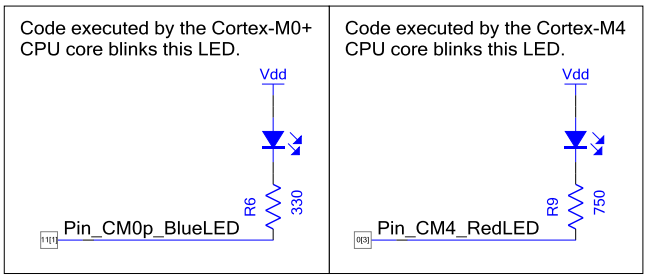
4.1 Resource Assignment Considerations

All generated Component API and PDL driver source files are available to both CPUs, same as the files in the *Shared Files* folder. If code in 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, then a function in that file may be executed simultaneously by both CPUs. It is also possible for a Component to be accessed by both CPUs; for example, both CPUs may send data through the same UART. Generally, Component API and PDL driver functions are "CPU-safe", that is, can 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.** A good practice is to indicate on the project schematic the CPU that "owns" the resource, as Figure 7 shows. Include code to use the resource only in the firmware for the desired CPU.

Figure 7. PSoC Creator Project Schematic for Dual CPUs Controlling Separate Pin Components



- **Share resources between the CPUs.** Code example CE216795 shows how the PSoC 6 MCU 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 is 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 Output Pin Components that are mapped to physical pins on the same GPIO port, use only the following functions to change the pin outputs: `GPIO_Write()`, `GPIO_Set()`, `GPIO_Clr()`, and `GPIO_Inv()`. For more information, see the PSoC Creator Pins Component datasheet or the PDL documentation.

4.2 Interrupt Assignment Considerations

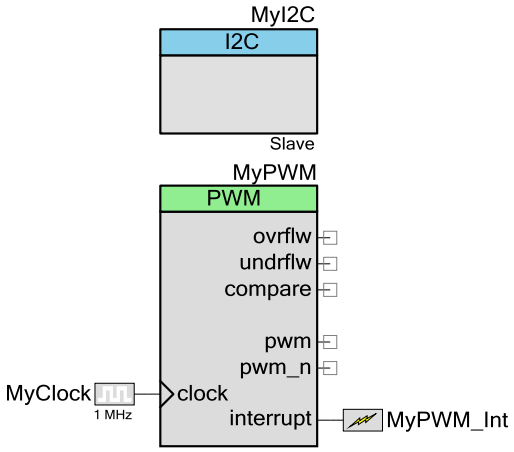
An important consideration for dual-CPU designs is assigning and handling interrupts. As noted previously, all device interrupts are available to CM4, and a subset of interrupts are routed through multiplexers to CM0+. You must decide which CPU will handle each interrupt.

Let us assign interrupts in an example design. Figure 8 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 9 shows.

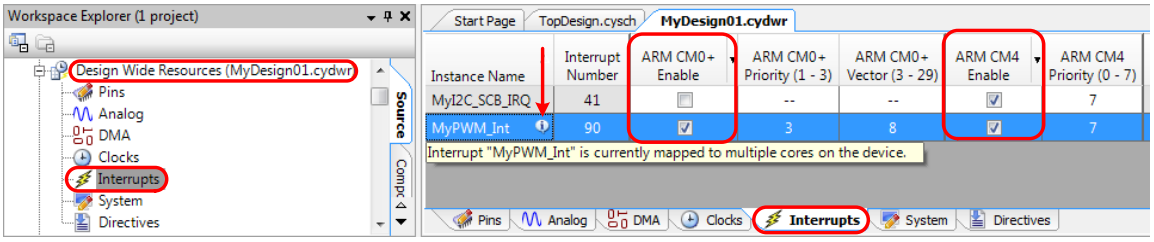
Note: In this example, the I2C Component has an interrupt embedded in it. That interrupt is not shown on the schematic in Figure 8; it is shown in the Design-Wide Resources window as MyI2C_SCB_IRQ.

Figure 8. Example Schematic Design with Two Interrupts



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

Figure 9. Assign Interrupts to the 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.

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

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 is designed to be a flexible device that enables 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 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: The 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 by both a PSoC Creator DMA Component and an API in the PDL. For more information, see the DMA Component datasheet and 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 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

Application Notes	
AN210781 – Getting Started with PSoC 6 MCU with Bluetooth Low Energy (BLE) Connectivity	Describes PSoC 6 MCU with BLE Connectivity devices and how to build your first PSoC Creator project
AN221111 – Creating a Secure System	Describes what is required to create a secure system, including the boot process from reset to application execution
AN217666 – PSoC 6 MCU Interrupts	A guide in developing projects that use interrupts. Includes advanced interrupt concepts such as interrupt latency, code optimization, and debug techniques.
AN219528 – PSoC 6 MCU Low Power Modes and Power Reduction Techniques	Describes how to use the PSoC 6 MCU power modes to optimize power consumption.
Code Examples	
CE216795 – PSoC 6 MCU Dual-Core Basics	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.
PSoC Creator Component Datasheets	
Interrupt	Supports generating CPU interrupts from hardware signals
Device Documentation	
PSoC 6 MCU: PSoC 63 with BLE Datasheet	PSoC 6 MCU: PSoC 63 with BLE Architecture Technical Reference Manual
Development Kit Documentation	
CY8CKIT-062-BLE PSoC 6 BLE Pioneer Kit	

About the Author

Name: Mark Ainsworth

Title: Applications Engineer Principal

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

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	5634375	MKEA	02/16/2017	New application note
*A	5653730	MKEA	03/08/2017	Updated template
*B	5777874	MKEA	06/09/2017	Updated text and screen shots for release versions of PSoC Creator 4.1 and PDL 3.0.0 Other miscellaneous edits
*C	5861685	MKEA	08/23/2017	Minor edits Ported to new application note document template Confidential tag removed
*D	6065641	MKEA	03/07/2018	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

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

Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709