AN215656

PSoC 6 MCU Dual-Core CPU System Design

Author: Mark Ainsworth

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

Associated Code Example: CE216795

Related Application Note: AN210781

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 PSoC video library here.

AN215656 describes the dual-core 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-core architecture enables high-performance and high-efficiency systems, and facilitates low-power designs. The application note also teaches how to build a simple dual-core design using the Cypress PSoC Creator™ Integrated Design Environment (IDE).

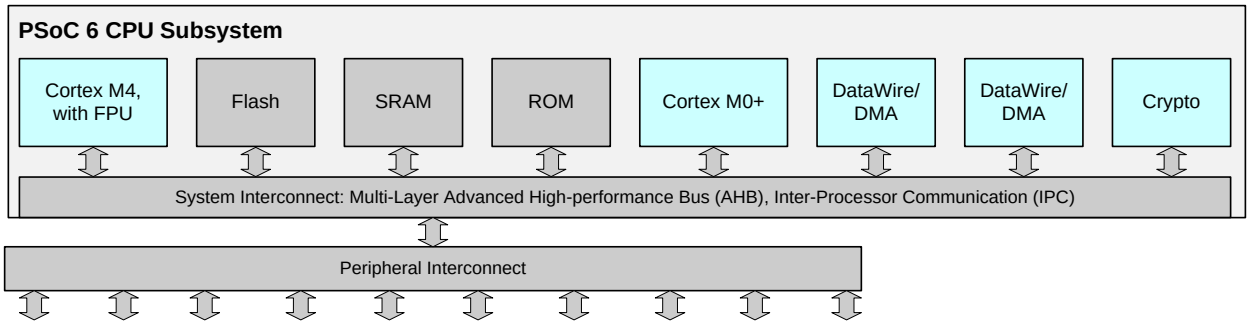
Contents

1 Introduction

PSoC 6 MCU is Cypress' new, 32-bit ultra-low-power PSoC, specifically designed for wearables and Internet of Things (IoT) products. 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 CPU cores, two DMA controllers, and a cryptography block (Crypto)—as Figure 1 shows:

Figure 1. PSoC 6 MCU CPU Subsystem Architecture



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 CPU cores you can:

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

For example, the Cortex-M0+ core (CM0) can "own" and manage all communication channels. The Cortex-M4 core (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 using the Cypress PSoC Creator. For an introduction to PSoC 6 MCU, see 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-core 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-core MCU, go to the section PSoC 6 MCU Dual-Core Development Process.

2 General Dual-Core Concepts

The process of developing firmware for a dual-core MCU is similar to that for a single-core 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 MCU that increased bandwidth comes at a price that is frequently on a par with single-core 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 dual-core, unless the application is large and complex. In that case, consider breaking the task up into subtasks, and assigning the subtasks to the CPU cores.
- **Dual task:** This is the most obvious fit; simply assign each task to a CPU core. Assign the task with larger computing requirements to the higher-performance core, e.g., the Cortex-M4 (CM4) core in PSoC 6 MCU.
- **Multiple tasks:** Again, assign each task to a CPU core. In each core, you must work out 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 cores. Some examples of dual-core RTOS architectures are:
  - Each core has its own RTOS and its own set of tasks. Each RTOS should have a task to manage communications with the other core.
  - Only one core (core 1) has an RTOS and multiple tasks. The other core (core 2) is idle until core 1 messages it to do a specified task. Core 2 messages back to core 1 when that task is done. For example, core 1 is the lower-performance CPU, and it uses core 2, the higher-performance CPU, to do computation-intensive tasks when needed.

Power: In a dual-core system, firmware can start and stop the cores to fine-tune power usage. In the previous example, to reduce power, the high-performance core is turned off 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 core, then debug code for the other core. In addition, a device such as an oscilloscope or a logic analyzer may be useful for monitoring communication between the cores.

3 PSoC 6 MCU Dual-Core Architecture

Figure 1 shows the overall dual-core 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 CPU cores 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.

  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 device-level security, safety, and protection features.

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

- **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 slower. 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.

  Device system calls may be initiated by either core, 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, followed by 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 CPU cores 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 cores. The IPC interrupt registers generate interrupts to both cores for lock and release events.

- **Interrupts:** Each core has its own set of interrupts. A peripheral can route its interrupt output to either or both cores. All peripheral interrupt lines are hard-wired to specific CM4 interrupt inputs. The 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.

- **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 CPU cores.

  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 supports Embedded Cross Trigger for synchronized debugging and tracing of both CPUs.

  PSoC Creator supports debugging a single core (either CM4 or CM0) at a time. For dual-core 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-Core Development Process

Note: This section shows only those aspects of the PSoC Creator development process that are unique to PSoC 6 MCU dual-core 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-core 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 2.

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 2. PSoC Creator Create Project Dialog

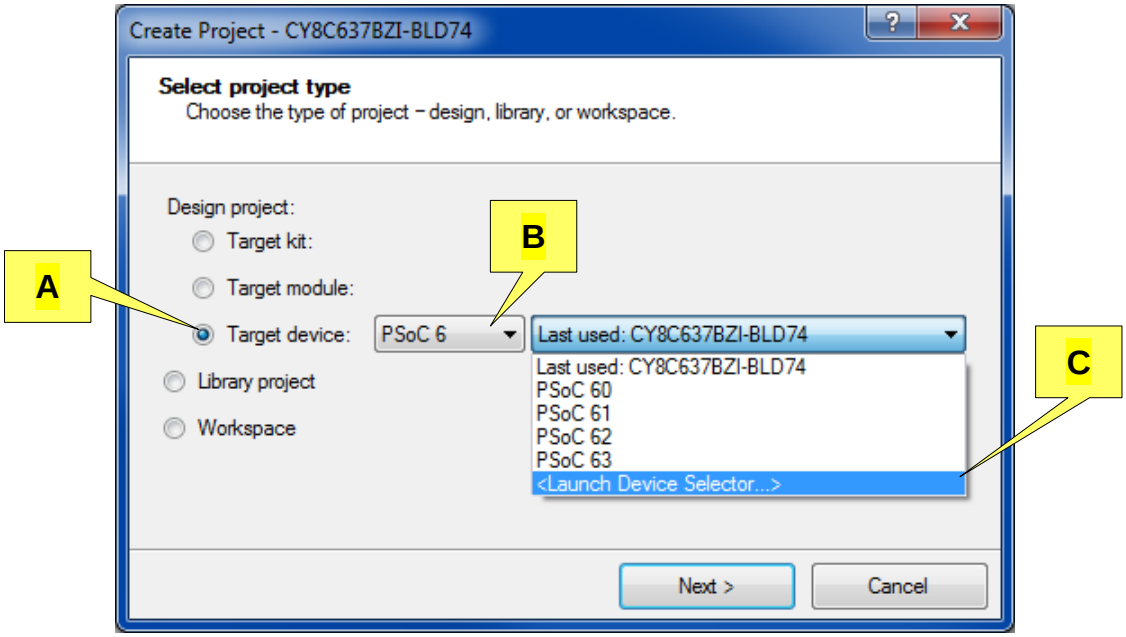
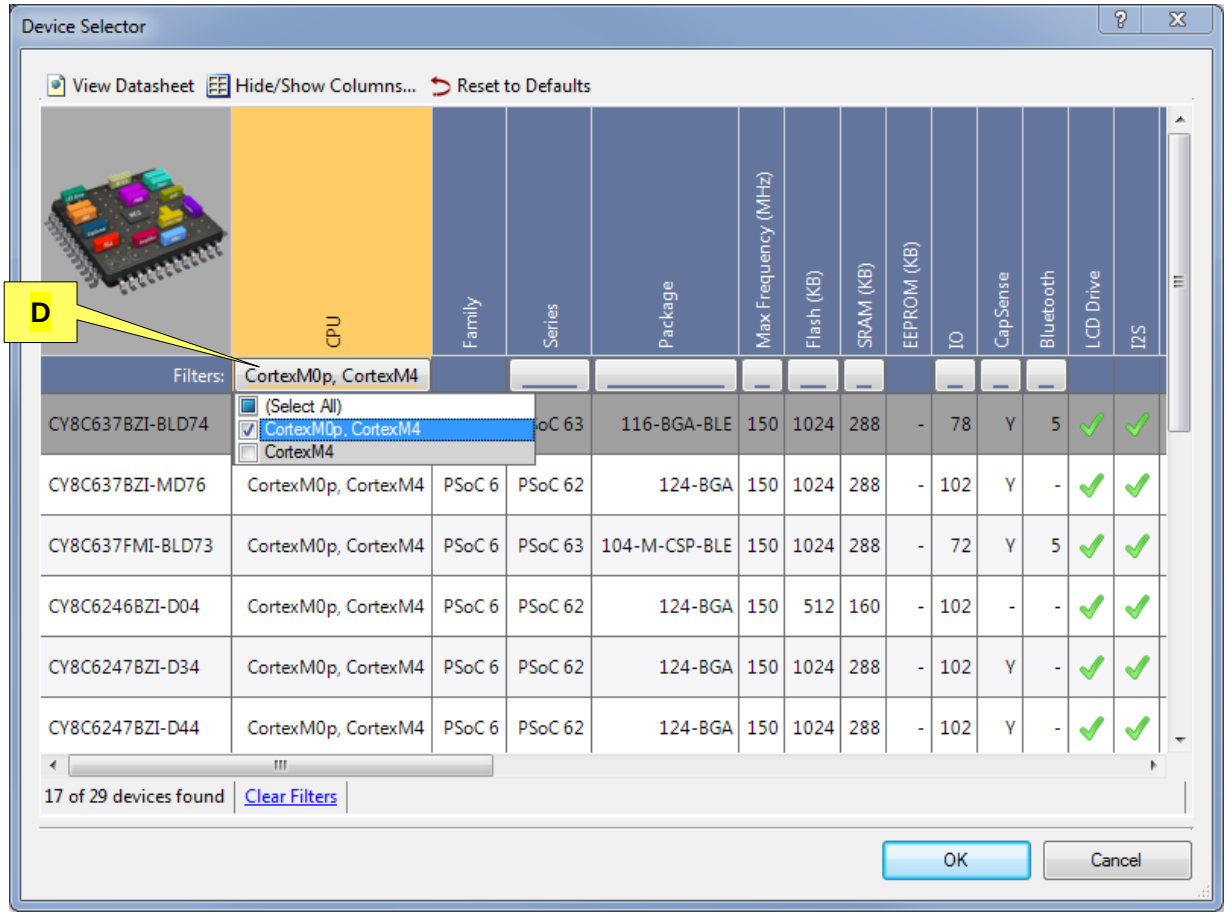


Figure 3 shows the Device Selector dialog. To see a list of dual-core devices, click the **CPU** category (D) and select only **CortexM0p, CortexM4**.

Note: In the CY8CKIT-062-BLE PSoC 63 with BLE Connectivity Pioneer Kit (CY8CKIT-062-BLE), the PSoC 6 MCU dual-core device part number is CY8C637BZI-BLD74.

Figure 3. PSoC Creator Device Selector Dialog

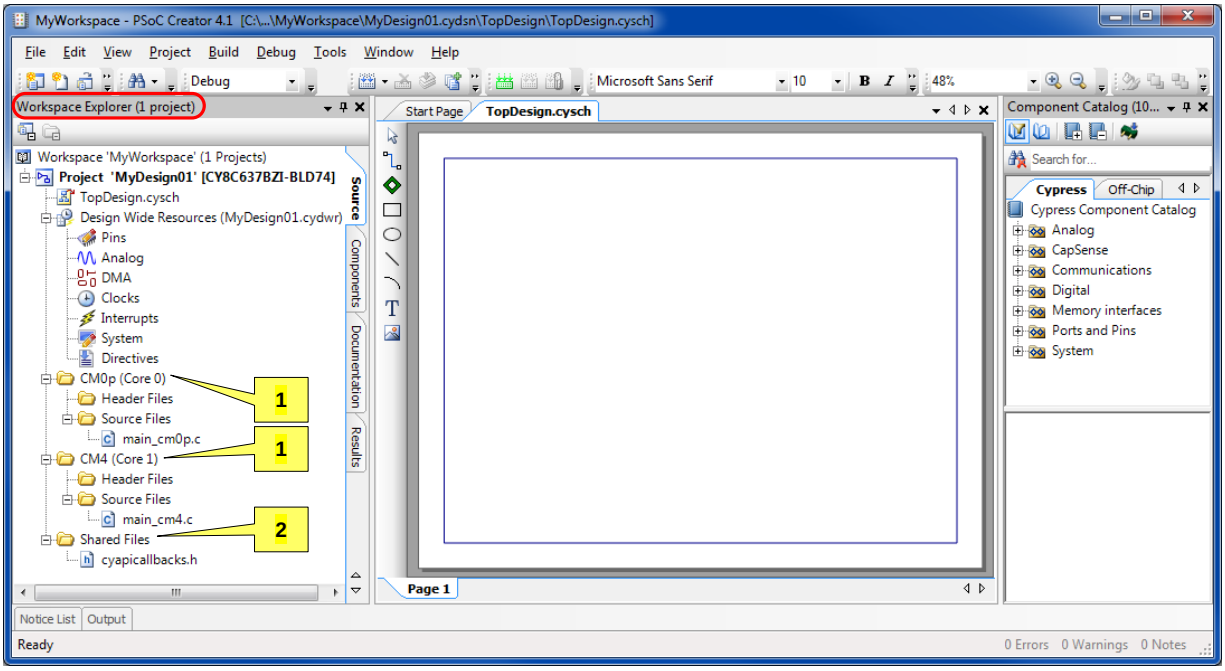


After selecting a PSoC 6 MCU part, the rest of the project creation process is the same as for other devices. Click through the rest of the **Create Project** dialogs; PSoC Creator creates the project.

The initial project windows layout (Figure 4) includes a **Workspace Explorer** window with the following features for dual-core devices:

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

Figure 4. PSoC Creator Initial Project Layout



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.

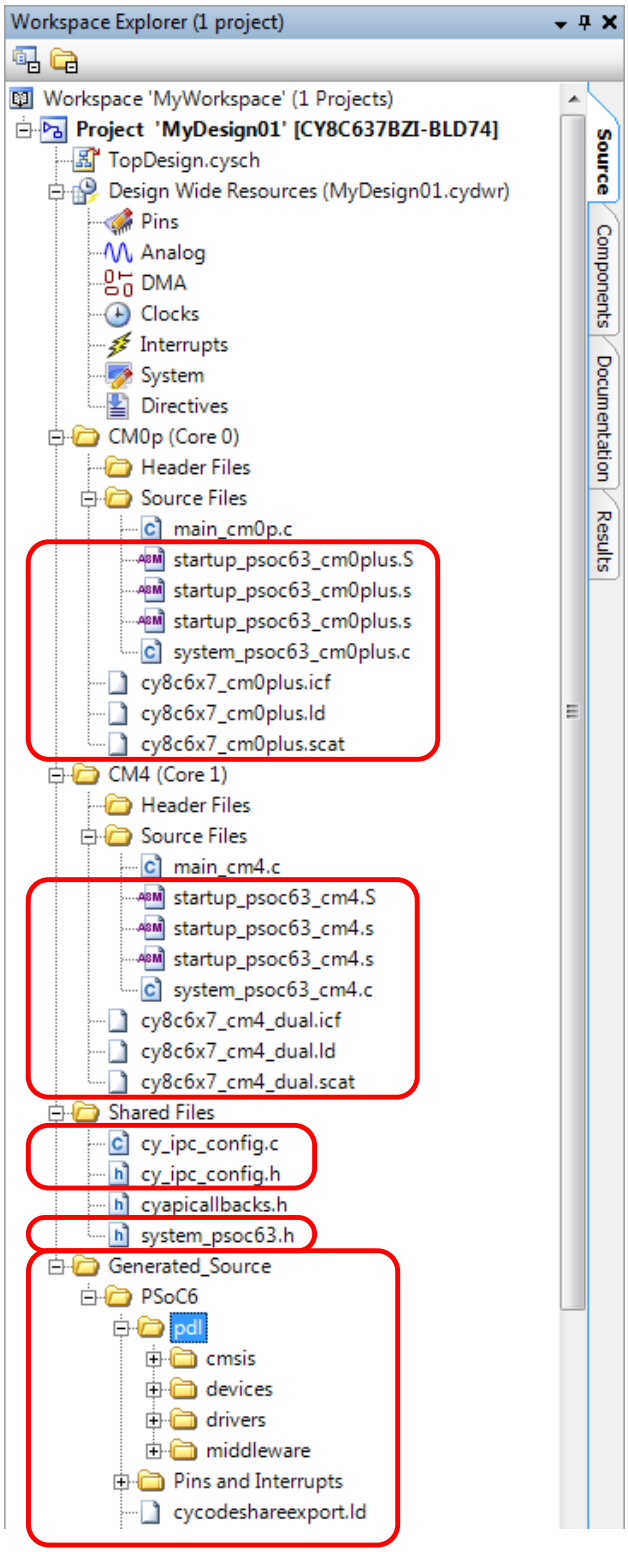
When schematic design entry is complete, select **Build** > **Generate Application**. This creates several system source code files in the existing folders as well as in the new folder *Generated Source*, as Figure 5 shows.

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 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 5. Add Generated Source to a Project



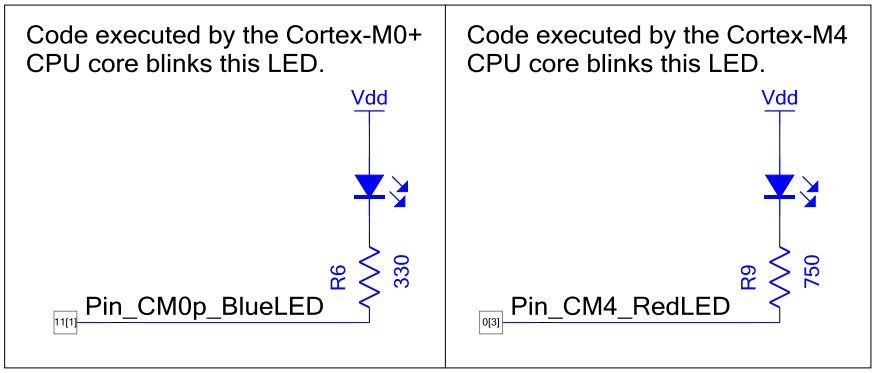
4.1 Resource Assignment Considerations

All generated Component API and PDL driver source files are available to both CPU cores, same as the files in the *Shared Files* folder. If code in a core references any API element in a generated source file, that file is compiled into the binary for that core. The same file can be compiled into both binaries—see code example CE216795, *PSoC 6 MCU Dual-Core Basics*.

If the same source file is compiled into both binaries, then a function in that file may be executed simultaneously by both CPU cores. It is also possible for a Component to be accessed by both cores; for example, both cores may send data through the same UART. Generally, Component API and PDL driver functions are "core-safe", that is, can be executed simultaneously by both cores. However, you should make design decisions about assigning resources to each core. 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 core that "owns" the resource, as Figure 6 shows. Include code to use the resource only in the firmware for the desired CPU.

Figure 6. PSoC Creator Project Schematic for Dual CPU Cores 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 CPU cores. 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 custom sections for each core with the same address.

Note: For 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

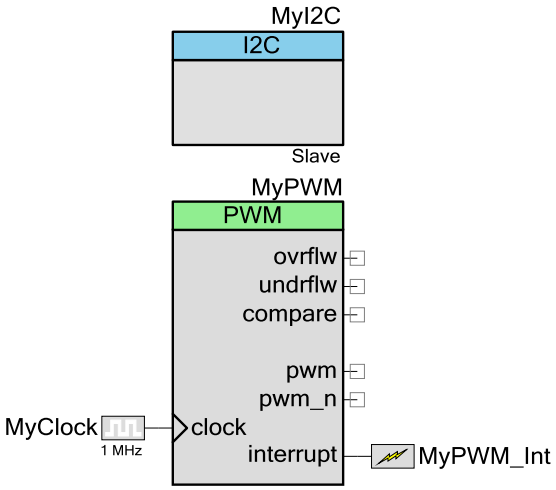
An important consideration for dual-core designs is assigning and handling interrupts. As noted previously, all device interrupts are available to the CM4 core, and a subset of interrupts are routed through a multiplexer to the CM0+ core. You must decide which core will handle each interrupt.

Let us assign interrupts in an example design. Figure 7 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 8 shows.

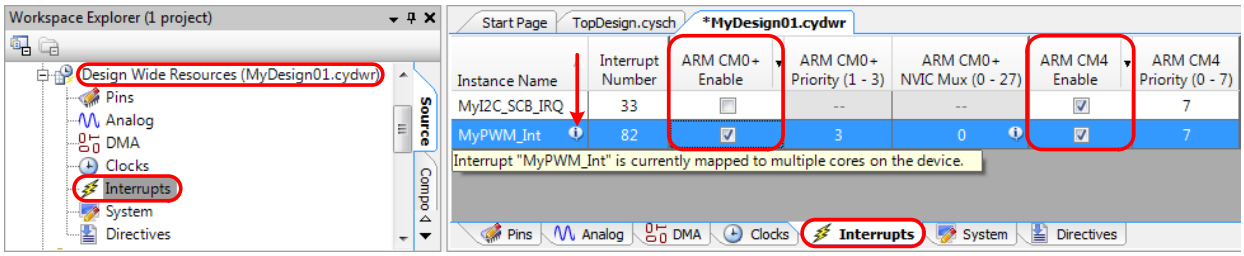
Note: In this example, the I2C Component has an I^2C interrupt embedded in it. That interrupt is not shown on the schematic in Figure 7; it is shown in the Design-Wide Resources window as MyI2C_SCB_IRQ.

Figure 7. 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 cores.

Figure 8. Assign Interrupts to the CM4 Core



Each interrupt is hard-wired to the CM4 core, so the **Interrupt Number** is automatically assigned by PSoC Creator when you build the project. Because interrupts are routed through a multiplexer to the CM0 core, you can select an **ARM CM0+ NVIC Mux** number for each interrupt.

Note: A warning symbol and tooltip are displayed if an interrupt is assigned to both cores. It is possible to assign an interrupt to both cores, but it is not recommended for most designs.

5 Summary

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

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, in PSoC 6 MCU, you have 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.
- 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.

- 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 using C switch / case statements.

  In addition, two GPIO ports include Smart I/O, which can be used to perform Boolean operations 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 a large number of Components to implement various functions in these peripherals. 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 support for PSoC peripherals, as well as detailed data in the device datasheets 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 the PSoC 6 MCU with BLE connectivity, and shows how to build a basic code example.
Code Examples	
CE216795 – PSoC 6 MCU Dual-Core Basics	Demonstrates PSoC 6 MCU dual-CPU features
Device Documentation	
PSoC 6 MCU Datasheets	
PSoC 6 MCU Technical Reference Manuals	
Development Kit (DVK) Documentation	
PSoC 6 MCU Kits	

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-Core 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.

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

Cypress Developer Community

Forums | WICED IOT Forums | 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
198 Champion Court
San Jose, CA 95134-1709

© Cypress Semiconductor Corporation, 2017. This document is the property of Cypress Semiconductor Corporation and its subsidiaries, including Spansion 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, nontransferable 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. 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, Spansion, the Spansion 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.