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

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

Continuity of ordering part numbers
Infineon continues to support existing part numbers. Please continue to use the ordering part numbers listed in the datasheet for ordering.

CE216795 - PSoC 6 MCU Dual-CPU Basics

Objective

These examples demonstrate 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; using ModusToolbox™ IDE.

Requirements

Tool: ModusToolbox™ IDE 1.0

Programming Language: C

Associated Parts: All PSoC® 6 MCU parts with dual CPUs

Related Hardware: PSoC 6 BLE Pioneer Kit

Note: The PSoC 6 BLE Pioneer kit is shipped with KitProg2, and ModusToolbox IDE only works with KitProg3. Before testing this code example, make sure that the kit is upgraded to KtiProg3. See ModusToolbox IDE **Help** > **ModusToolbox IDE Documentation** > **User Guide,** Section **PSoC 6 MCU KitProg Firmware Loader**.

Overview

The first example shows the two CPUs in PSoC 6 MCU – Arm® Cortex®-M0+ (CM0+) and Arm Cortex-M4 (CM4) – doing independent tasks. The tasks are simple; each task blinks a separate LED using a firmware delay.

The second example uses the inter-processor communication (IPC) block in PSoC 6 MCU. Using the IPC block, the CPUs share a portion of the SRAM and communicate in a simple mutex/semaphore-based application.

Hardware Setup

This example uses the kit's default configuration. Refer to the kit guide to ensure that the kit is configured correctly.

Software Setup

None.

Operation

1. Connect the kit to your PC using the provided USB cable.
2. Add the code example to the IDE, in a new workspace. See KBA225201.
3. Program the PSoC 6 MCU device. In the project explorer, select the **mainapp** project. In the Quick Panel, scroll to the **Launches** section and click the **Program (KitProg3)** configuration.

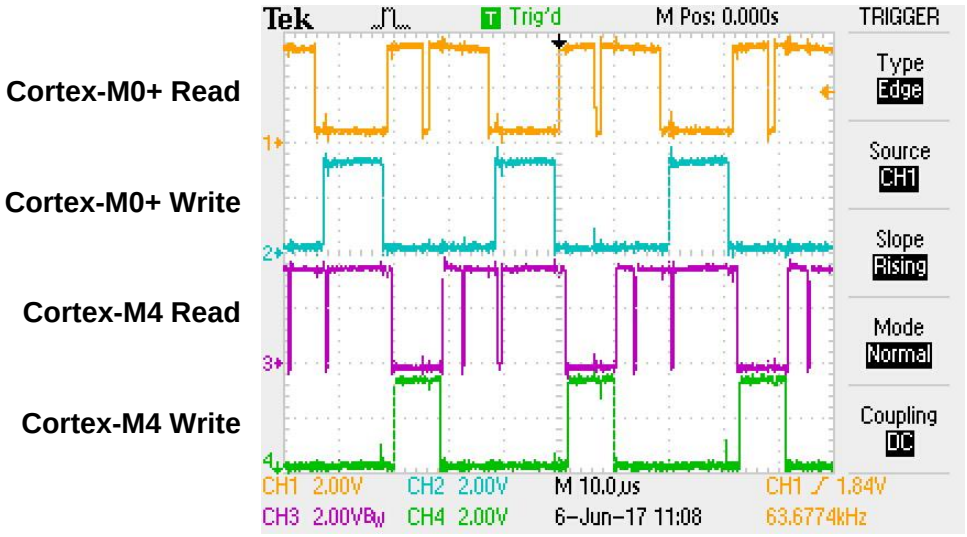
LED Blink Example

4. Confirm that the kit's blue and red LEDs blink, at slightly different rates.

IPC Shared Memory Example

5. Connect oscilloscope probes to four kit pins P5[3:0] on the kit connector J4. Confirm that the oscilloscope display is similar to Figure 1, when all four pins are monitored.

Figure 1. Screenshot Showing Dual CPUs Reading and Writing Shared Memory



Debugging

You can debug the example to step through the code. Use the **Debug (KitProg3)** configuration. See KBA224621 to learn how to start a debug session with ModusToolbox IDE.

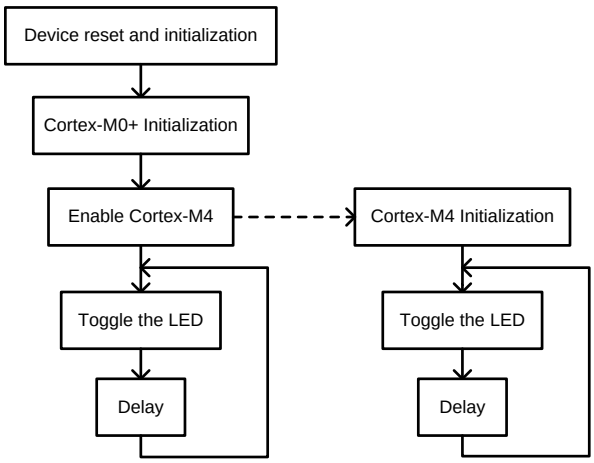
Design and Implementation

There are two examples:

1. **Basic Example:** The two CPUs each blink a separate LED. The hardware design uses only device pins for blinking LEDs.

   Figure 2 shows the firmware design. Note that after the PSoC 6 MCU device reset, CM0+ always executes first while CM4 is held in a reset state.

Figure 2. Dual CPU Basic Flowchart

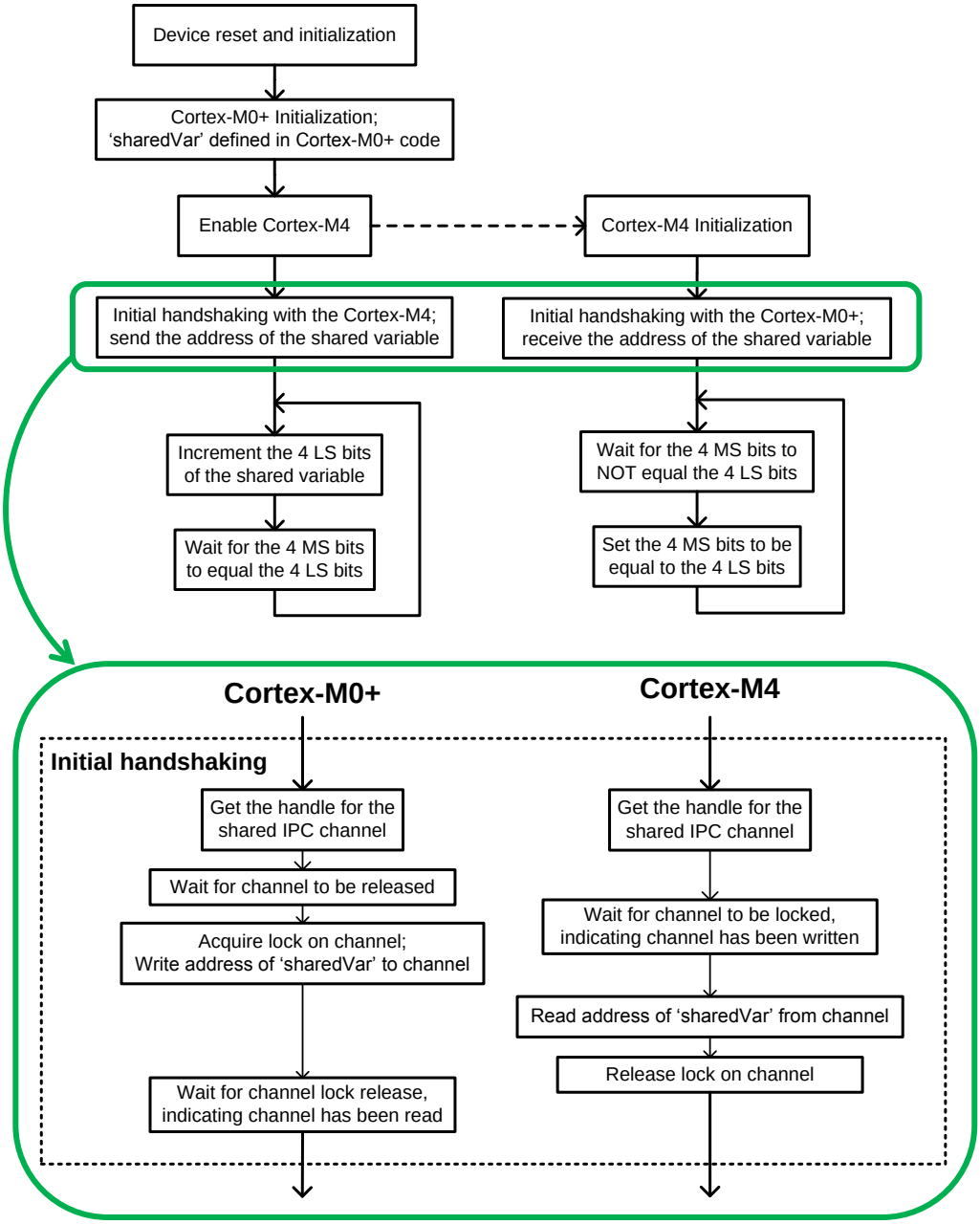


2. **Inter-Processor Communication (IPC) Example:** The CPUs communicate with each other using a mutex to control the access to a shared variable. Each CPU performs actions based on the value in the variable. The firmware uses the IPC application programming interface (API) in the PSoC 6 SDK.

   Figure 3 shows the firmware design. The CPUs update a one-byte shared variable in SRAM as follows:

   - CM0+ increments the least-significant (LS) 4-bit nibble, then waits for the most-significant (MS) nibble to equal the LS nibble.
   - CM4 waits for the LS nibble to not equal the MS nibble, then copies the LS nibble into the MS nibble to make the two halves of the byte equal.

Figure 3. Dual CPU Shared Memory Communications Flowchart



In the main() functions for both CPUs, the shared variable is never accessed directly. Instead, the main function calls utility functions that provide mutex lock and release access for reading and writing the variable. The mutex access utility functions in turn call IPC driver functions that provide an atomic IPC channel lock and release capability. This ensures that only one CPU at a time accesses the shared variable.

The mutex access utility functions are in the code example project files *ce216795_common.h/.c*. Copies of the same file, with the same utility functions, exist in both the *_mainapp* and the *_mainapp_cm0p* projects. Even though the files and functions are in separate builds and binaries, for good design practice, the functions should be considered to be executed simultaneously by both CPUs. This is similar to mutex techniques in RTOS designs except you have multiple CPUs instead of multiple tasks.

Four pins are used for debug purposes; see Figure 1.

Design Considerations

Basic Example

CY8CKIT-062-BLE has one RGB LED module. Therefore, in the LED blink example, one of four colors may be displayed at any time: black (both LEDs OFF), blue, red, and purple (both LEDs ON). To see the transitions, keep the two blink rates low, and different from each other.

If the pins are on the same GPIO port, only the following GPIO API functions should be used to update the pins: `GPIO_Write()`, `GPIO_Set()`, `GPIO_Clr()`, and `GPIO_Inv()`. For more information, see the ModusToolbox IDE documentation.

IPC Example

The example may be switched such that the shared variable is defined in the CM4 code and its address sent to CM0+.

The example includes error handling in the form of a `HandleError()` function, which is called if an IPC error or a mutex lock/release timeout is detected. The function just drops into a placeholder loop; it can be modified for application-specific error handling.

A number of other IPC-based code examples are available; they demonstrate more complex features of the IPC block and PDL driver. For more information, see Related Documents.

Dedicated and Shared Resources

This code example shows two general ways to allocate resources (e.g., pins, UARTs) to two CPUs:

- **Dedicate a resource to a CPU.** A good practice is to document the CPU that “owns” the resource. Include code to use the resource only in the firmware for the desired CPU – place it in either the *_mainapp* or the *_mainapp_cm0p* project in your application.
- **Share memory or other resources between the CPUs.** The IPC shared memory example shows how a mutex may be implemented to share memory between the CPUs. Use the same technique to share a resource such as a UART.

Flash and SRAM memory that are allocated in a CPU’s executable 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 that have the same address.

Resources and Settings

This example uses only GPIO pins, all configured for strong drive, input buffer OFF. The *design.modus* file contains all the configuration settings. For pin usage and configuration, open the **Pins** tab of the design file.

Reusing This Example

This example is designed for the kit indicated in Related Hardware. It is easily portable to the PSoC 6 WiFi-BT Pioneer Kit, which has the same pin assignments for the LEDs and button as CY8CKIT-062-BLE. Change the device to CY8C6247BZI-D54.

To port this code example to a different platform or device, right-click the *..._mainapp* project and click **Change ModusToolbox Platform...** or **Change ModusToolbox Device...**. If changing to a different platform, you may need to reassign pins. Note that the basic example uses the red and blue LED in an RGB LED module. Other kits have different LED configurations; adapt the application to those LED configurations.

In some cases, a resource used by a code example is not supported on another device. In that case, the example will not work. If you build the code targeted at such a device, you will get errors. See the device datasheet for information on which resources a device supports.

Related Documents

Application Notes	
AN215656 – PSoC 6 MCU: Dual-CPU System Design	Describes the dual-CPU architecture in PSoC 6 MCU, and shows how to build a simple dual-CPU design
AN221774 – Getting Started with PSoC 6 MCU	Describes PSoC 6 MCU devices and how to build your first ModusToolbox IDE or PSoC Creator project
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
Code Examples	
Visit the Cypress GitHub site for a comprehensive collection of code examples using ModusToolbox IDE	
Device Documentation	
PSoC 6 MCU: PSoC 63 with BLE Datasheet	PSoC 6 MCU: PSoC 63 with BLE Architecture Technical Reference Manual
Development Kits	
CY8CKIT-062-BLE	PSoC 6 BLE Pioneer Kit
CY8CKIT-062-WiFi-BT	PSoC 6 WiFi-BT Pioneer Kit
CY8CPROTO-063-BLE	PSoC 6 BLE Prototyping Kit
CY8CPROTO-062-4343W	PSoC 6 Wi-Fi Prototyping Kit
Tool Documentation	
ModusToolbox IDE	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.

Cypress Resources

Cypress provides a wealth of data at www.cypress.com to help you to select the right device, and quickly and effectively integrate the device into your design.

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

Document History

Document Title: CE216795 – PSoC 6 MCU Dual-CPU Basics

Document Number: 002-25617

Revision	ECN	Orig. of Change	Submission Date	Description of Change
**	6397585	MKEA	12/03/2018	New version of CE216795, updated for ModusToolbox IDE.

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

Technical Support

cypress.com/support

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

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

© Cypress Semiconductor Corporation, 2018. 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. 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, 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.