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.

AN215656

PSoC 6 MCU Dual-CPU System Design

Author: Mark Ainsworth

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

Associated Code Example: 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 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® software, Command-line Interface (CLI), and PSoC Creator IDE, and how to debug the design using various IDEs.

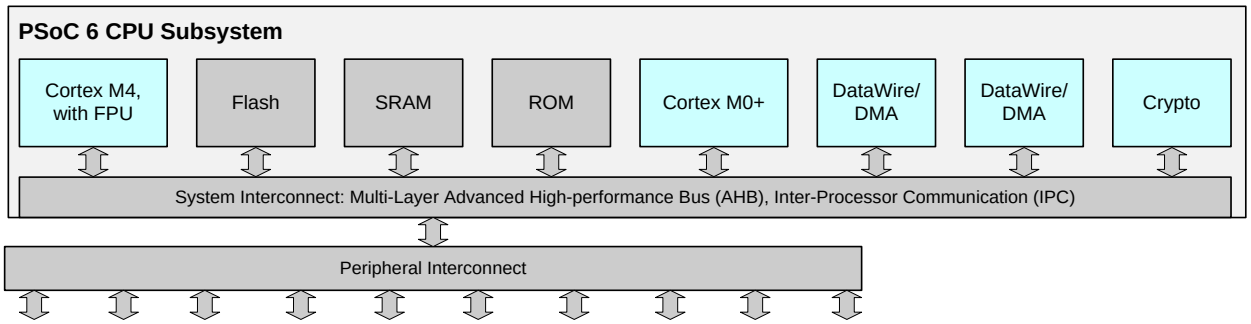
Contents

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, 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. Some PSoC 6 MCU parts have only one CPU available for application use. 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 the 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 Tools and Libraries

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

- **ModusToolbox:** ModusToolbox software includes configuration tools, low-level drivers, middleware libraries, and operating system support, as well as other packages that enable you to create MCU and wireless applications. It also includes the optional Eclipse IDE for ModusToolbox.
- **PSoC Creator:** A Cypress-proprietary IDE that runs on Windows only. It supports a subset of PSoC 6 MCU devices as well as other PSoC device families such as PSoC 3, PSoC 4, and PSoC 5LP.

Both platforms provide a mechanism to export projects to third-party tools, such as Keil µVision, IAR Embedded Workbench and Visual Studio Code.

Cypress also provides a set of libraries to facilitate application development with PSoC 6 MCU:

- **Peripheral Driver Library (PDL):** PDL drivers abstract hardware functions into a set of easy to use APIs. For more information on the PDL, go to the https://github.com/cypresssemiconductorco/psoc6pdl webpage. This library is available in PSoC Creator and ModusToolbox software.

- **Hardware Abstraction Layer (HAL):** HAL is built on top of the PDL. It provides a high-level interface to configure and use hardware blocks on Cypress MCUs. It is a generic interface that can be used across multiple product families. For more information on the HAL, go to the https://github.com/cypresssemiconductorco/psoc6hal webpage. The HAL is available in ModusToolbox software and works only with the CM4 CPU. PSoC Creator does not have support for HAL.
- **Middleware:** This category includes multiple libraries that implement APIs for various domains, for example capacitive sensing or an HTTP server. A middleware library may be created by Cypress or come from a third party. In PSoC Creator, the middleware is usually available as a PSoC Creator Component. In ModusToolbox software, it is available as a library.
- **Board Support Package (BSP):** BSP provides a standard interface to a board's features and capabilities. The API is consistent across Cypress kits. Other software (such as middleware or an application) can use the BSP to configure and control the hardware. The BSP is available in ModusToolbox software only.

1.2 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 software 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 software, see the ModusToolbox software home page. Some PSoC 6 MCU devices are not supported by PSoC Creator; in this case, ModusToolbox software must be used. If you are using ModusToolbox software, make sure that it's version 2.2 or higher for designs based on PSoC 6 MCU devices with dual CPUs.

If you are new to PSoC Creator, see the PSoC Creator home page. Use PSoC Creator version 4.3 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 software 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 2 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.

- **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 the Boot Code chapter of the Architecture TRM for details.

  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. PSoC Creator auto-generates code in CM0+ main() to release the CM4 reset. ModusToolbox software creates an empty project by default for CM0+.

- **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 8 interrupt inputs (32 interrupt inputs in the CY8C61x7, CY8C62x7, and CY8C63x7). 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 software 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 software, 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 software home page. Some PSoC 6 MCU devices are not supported by PSoC Creator; in this case, ModusToolbox software must be used. If you are using ModusToolbox software, then make sure it's version 2.2 or higher for designs based on PSoC 6 MCU devices with dual CPUs.
- The PSoC Creator home page. Use PSoC Creator version 4.3 or higher for PSoC 6 MCU-based designs.

4.1 ModusToolbox Software Instructions

ModusToolbox software application development for a PSoC 6 MCU dual-CPU device is similar to that for any other device supported by ModusToolbox software. By default, the Board Support Packages (BSP) that come with the ModusToolbox software provides several prebuilt CM0+ application images. This section explains how to create your own CM0+ application as a substitute for the prebuilt images.

To create a dual-CPU workspace, you must create two new applications, one for the CM0+ CPU and one for CM4 CPU. Alternatively, you can create a new project based on a dual-CPU code example, which already sets up both applications. You can use any code example with a repo name that includes "dual-cpu" as a start point. See the list here.

The next sections show how to create a dual-CPU workspace step-by-step.

Each section explains two methods using different tools with ModusToolbox software:

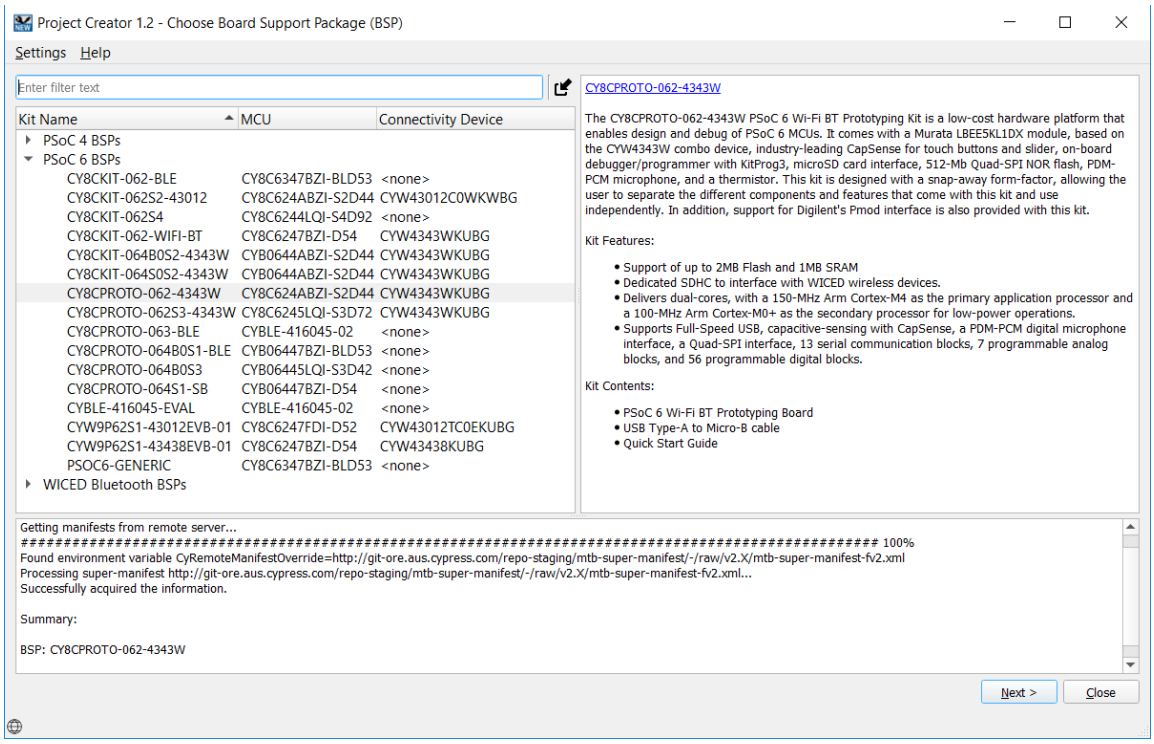
- Eclipse IDE for ModusToolbox
- Command-line Interface (CLI)

4.1.1 Creating the CM0+ CPU Application

4.1.1.1 In Eclipse IDE for ModusToolbox

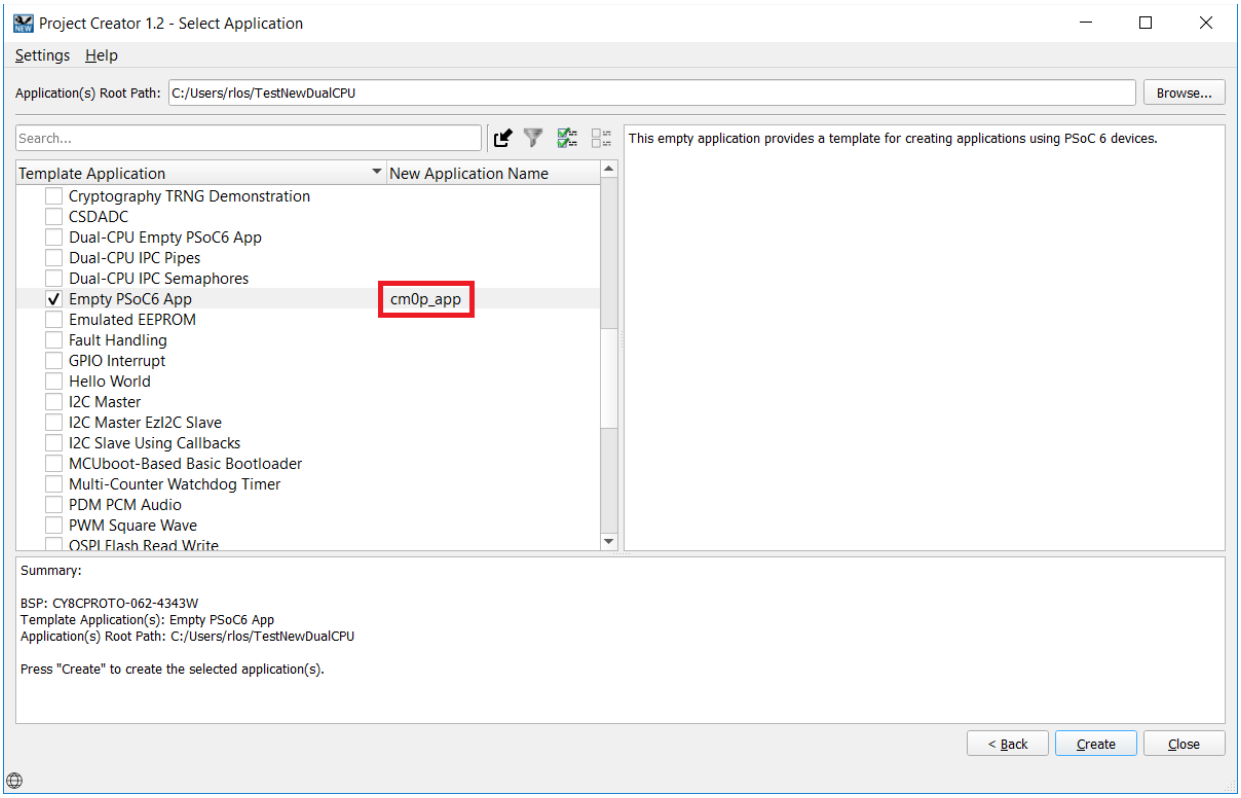
1. Click **New Application** in the Quick Panel. Then, choose one of the PSoC 6 BSPs, as Figure 2 shows.

Figure 2. Board Support Package Selection in ModusToolbox Software



2. Click **Next**, and then choose the **Empty PSoC6 App**. In the **New Application Name**, set it to “cm0p_app**”**, as shown in Figure 3. Complete the New Application dialogs, and a new application is created in the selected workspace.

Figure 3. CM0+ CPU Application Creation



4.1.1.2 In CLI

1. Open a CLI terminal and navigate to ModusToolbox software installation folder for project creation (*<install_path>ModusToolbox/tools_<version>/project-creator/*).
2. Create a new project by executing the `project-creator-cli` command.

   For example:

```
project-creator-cli \
       --board-id CY8CPROTO-062-4343W \
       --app-id mtb-example-psoc6-empty-app \
       --user-app-name cm0p_app \
       --target-dir "C:/workspace_directory"
```

Alternatively, you can execute the **project-creator** tool in the same folder and follow the same steps described In Eclipse IDE for ModusToolbox.

See the ModusToolbox User Guide (Section 2.3) for more details.

4.1.1.3 Final Changes

Once the application is created (independent of which method used), open the file *Makefile* and make the following changes:

- Create a new line to set the CORE to CM0+. By default, the make file assumes the application target is CM4. Adding this line forces the application target to be CM0+.

```
CORE=CM0P
```

- Optionally, change the `APPNAME` variable to **cy_m0p_image**, which is the default name used by the linker scripts.

```
APPNAME=cy_m0p_image
```

When writing code for the CM0+ CPU, you cannot make use of Cypress HAL, so use PDL APIs only. Modify the *main.c* file to remove any references to the HAL. Optionally, also remove any references to the BSP.

For example:

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

int main(void)
{
    cy_rslt_t result;

    /* Initialize the device and board peripherals */
    result = cybsp_init() ;
    if (result != CY_RSLT_SUCCESS)
    {
        CY_ASSERT(0);
    }

    enable_irq();

    Cy_SysEnableCM4(CY_CORTEX_M4_APPL_ADDR);

    for (;;)
    {
        Cy_SysPm_DeepSleep(CY_SYSPM_WAIT_FOR_INTERRUPT);
    }
}
```

The Cypress HAL was not designed to run on CM0+, therefore it will not compile correctly. It is also recommended to initialize the BSP in one CPU only, preferentially the CM4 CPU.

The CM0+ application also does not refer to the *design.modus* generated files by default. If you want to include these files as part of the CM0+ application, add to the COMPONENTS variable in CM0+ Makefile the following:

```
COMPONENTS=BSP_DESIGN_MODUS
```

Or if you are using a custom *design.modus*:

```
COMPONENTS=CUSTOM_DESIGN_MODUS
```

4.1.2 Creating the CM4 CPU Application

4.1.2.1 In Eclipse IDE for ModusToolbox

Create a new application based on the same BSP chosen for the CM0+ CPU. You can choose any application template. Name the application as “cm4_app” and complete the application creation.

4.1.2.2 In CLI

Run the project-creator-cli again for creating the “cm4_app” application. For example:

```
project-creator-cli \
       --board-id CY8CPROTO-062-4343W \
       --app-id mtb-example-psoc6-hello-world \
       --user-app-name cm4_app \
       --target-dir "C:/workspace_directory"
```

Note you can use any template application in *–app-id*. To see the complete list, go to Cypress GitHub repository.

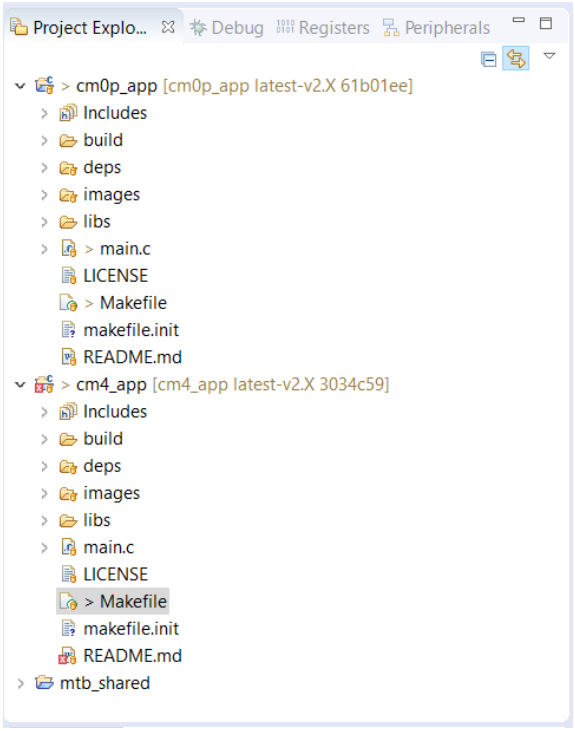
4.1.2.3 Final Changes

Once the application is created, open *Makefile* and make the following changes:

- Add "`CM0P_SLEEP`" to the `DISABLE_COMPONENT` list. If you are using another pre-built library, change this accordingly.
  ```
  DISABLE_COMPONENTS=CM0P_SLEEP
  ```
- Create a new line to add a relative path to the *cm0p_app* as a dependency:
  ```
  DEPENDENT_APP_PATHS=../cm0p_app
  ```

After following the steps, you can view the applications in the Project Explorer window (In Eclipse IDE for ModusToolbox only), as Figure 4 shows. Both applications have their own set of libraries, configuration and make files.

Figure 4. Project Explorer



The CM4 application automatically builds the dependent CM0+ application. It links the application twice, once without the CM0+ image and once with it. The default image includes the CM0+ image just as it would when building a CM4 application using one of the pre-built images.

In Eclipse IDE for ModusToolbox, when building or programming the application, you need to select the application. You can click on any file or folder that belongs to the application in the Project Explorer. The Quick Panel updates its links, including the launch scripts based on the selected application.

It is recommended to initialize the hardware in one of the CPUs only. Therefore, only one of the *design.modus* files should be edited.

The *mtb_shared* folder contains the libraries and BSP files that might be shared by both applications. Each application has its own *deps* folder. Add any dependency library that the application might use in this folder. Note that the same library might be presented in both applications.

4.1.3 Customizing Linker Scripts

The CM0+ and CM4 applications each have their own linker scripts. By default, the CM0+ CPU consumes only 8192 [0x2000] bytes of flash and 8192 [0x2000] bytes of SRAM. If your CM0+ application requires more memory, both linker scripts require changes. The following steps list the changes:

1. Create a shared folder to store the custom linker scripts in the root directory of the project. For example:

   shared / linker_script / COMPONENT_CM0P

   shared / linker_script / COMPONENT_CM4

2. Copy the CM0+ linker script file from the folder below to the "*shared / linker_script / COMPONENT_CM0P"* folder:

   mtb_shared / TARGET_<BSP_NAME> / COMPONENT_CM0P / TOOLCHAIN_<TOOL_NAME> /

   By default, the BSP supports the ARM (*.sct), GCC_ARM (*.ld), and IAR (*.icf) toolchains. Each toolchain has its own linker script.

3. Edit the FLASH and SRAM size as desired:

Toolchain:	Where to change:
ARM (*.sct)	`#define RAM_SIZE 0x00002000` `#define FLASH_SIZE 0x00002000`
GCC_ARM (*.ld)	`ram (rwx) : ORIGIN = 0x08000000, LENGTH = 0x2000` `flash (rx) : ORIGIN = 0x10000000, LENGTH = 0x2000`
IAR (*.icf)	`define symbol __ICFEDIT_region_IRAM1_end__ = 0x08001FFF;` `define symbol __ICFEDIT_region_IROM1_end__ = 0x10001FFF;`

4. Copy the CM4 linker script file from the folder below to the *"shared / linker_script / COMPONENT_CM4"* folder:

   mtb_shared / TARGET_<BSP_NAME> / COMPONENT_CM4 / TOOLCHAIN_<TOOL_NAME> /

5. Edit the values based on the CM0+ memory size:

Toolchain:	Where to change:

ARM (*.sct)	`#define RAM_START 0x08002000`
	`#define RAM_SIZE 0x000FD800`
	`#define FLASH_CM0P_SIZE 0x2000`

GCC_ARM (*.ld)	`FLASH_CM0P_SIZE = 0x2000;` `ram (rwx) : ORIGIN = 0x08002000, LENGTH = 0xFD800`
IAR (*.icf)	`define symbol __ICFEDIT_region_IRAM1_start__ = 0x08002000;` `define symbol __ICFEDIT_region_IRAM1_end__ = 0x080FF7FF;`

6. If you named the APPNAME in the CM0+ application make file different from "cy_m0p_image", any reference in the linker scripts need to be substituted with the new name.

7. In the CM0+ and CM4 Makefile, add the line below to refer to the custom linker script:

   LINKER_SCRIPT=../shared/linker_script/COMPONENT_$(CORE)/<FILENAME>.<EXTENSION>

   You need to substitute the FILENAME and EXTENSION based on the file copied to the linker script folder you create in step (1) and (2).

8. In the CM0+ Makefile, add to DEFINES the CM4 application address **(CY_CORTEX_M4_APPL_ADDR)**. For example, if the size of CM0+ image is 0x8000, set it to:

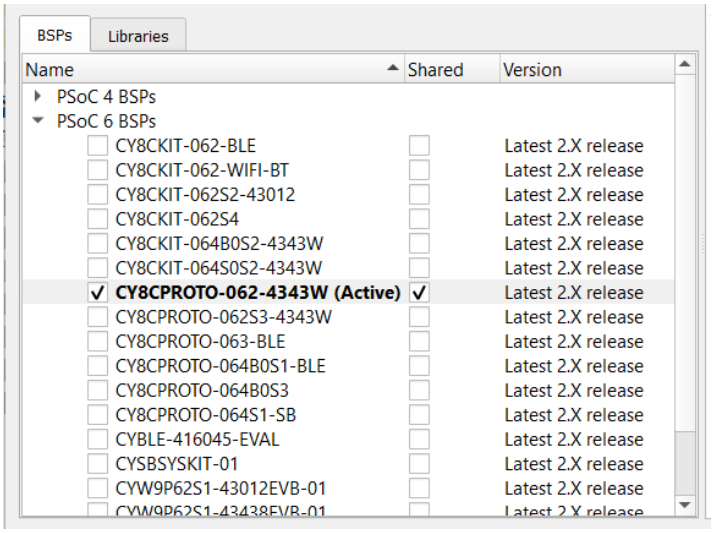
   DEFINES=CY_CORTEX_M4_APPL_ADDR=0x10008000

4.1.4 Sharing Libraries and Peripherals

As described before, the CM4 and CM0+ can share or not some of the libraries used by the application. The *deps* folder of each application contains the list of libraries used by the application. ModusToolbox provides a tool to manage the libraries and BSPs, called Library Manager. You can launch this tool from the Eclipse IDE for ModusToolbox, **Quick Panel** > **Tools** > **Library Manager**. Alternatively, you can execute the **library-manager** tool located at this folder: (*<install_path>ModusToolbox/tools_<version>/library-manager/*).

When selecting the BSP or library, the tool provides an option to share it (column *Shared*), as shown in Figure 5.

Figure 5. Library Manager



In dual-CPU applications, it is recommended to shared all the libraries and the BSPs.

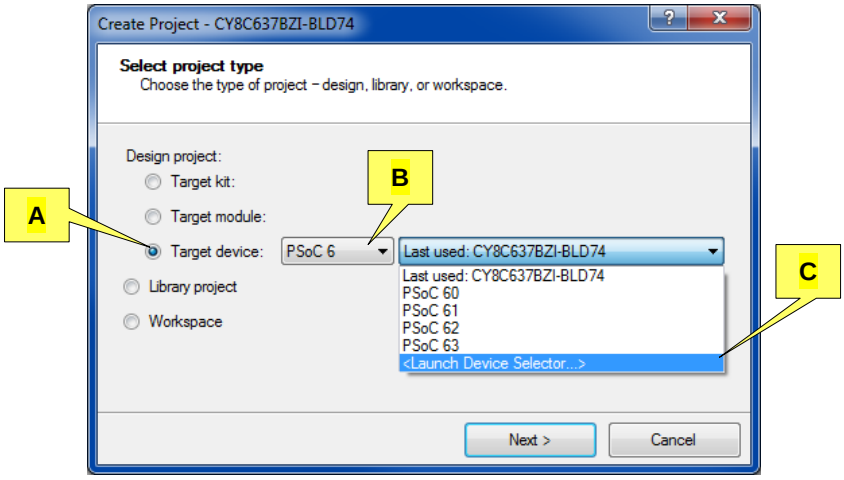
When it comes to configure the peripherals, it is recommended to have only one *design.modus* in dual-CPU applications and have only one of the CPUs to call `init_cycfg_all()` or `cybsp_init()`, preferentially the CM4. If a peripheral is only used by one of the CPUs, you can use as you would use in a single CPU application.

If a peripheral is shared, you should only initialize it in one of the CPUs and use some sort of mutex or synchronization mechanism to avoid both CPUs using it at the same time. It is not recommended to use the HAL for shared peripherals, since the CM0+ would not have access to the HAL object created by the CM4.

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, as shown in Figure 6.

Figure 6. PSoC Creator Create Project Dialog

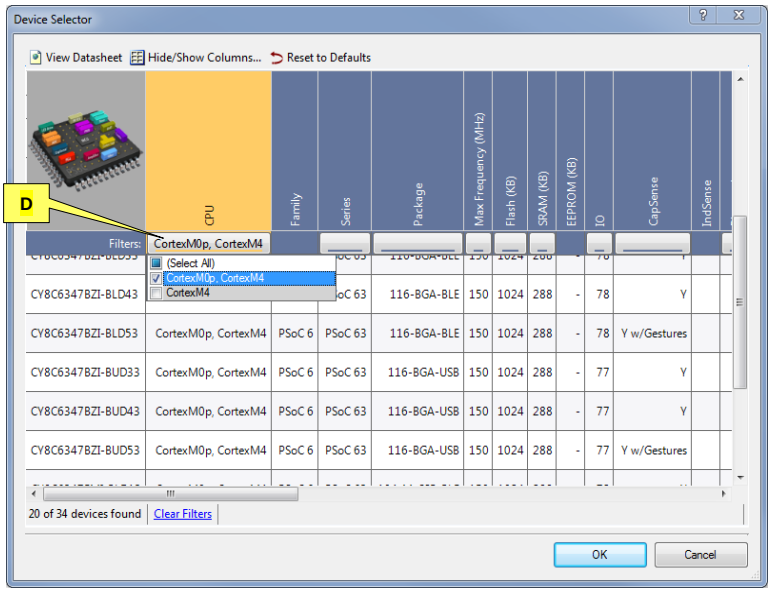


Select **Target Device** (A), and **PSoC 6** (B). On the pull-down list (C), select **<Launch Device Selector...>** to see a list of PSoC 6 devices.

Figure 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

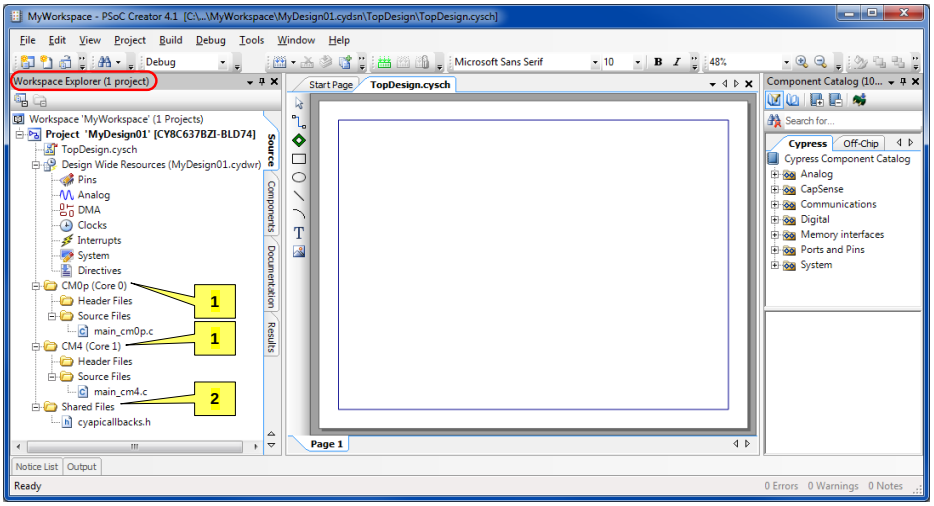


After selecting a PSoC 6 MCU part, the rest of the project creation process is the same as for other devices. Click through the rest of the Create Project dialogs; PSoC Creator creates the project.

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

- 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.
- 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.

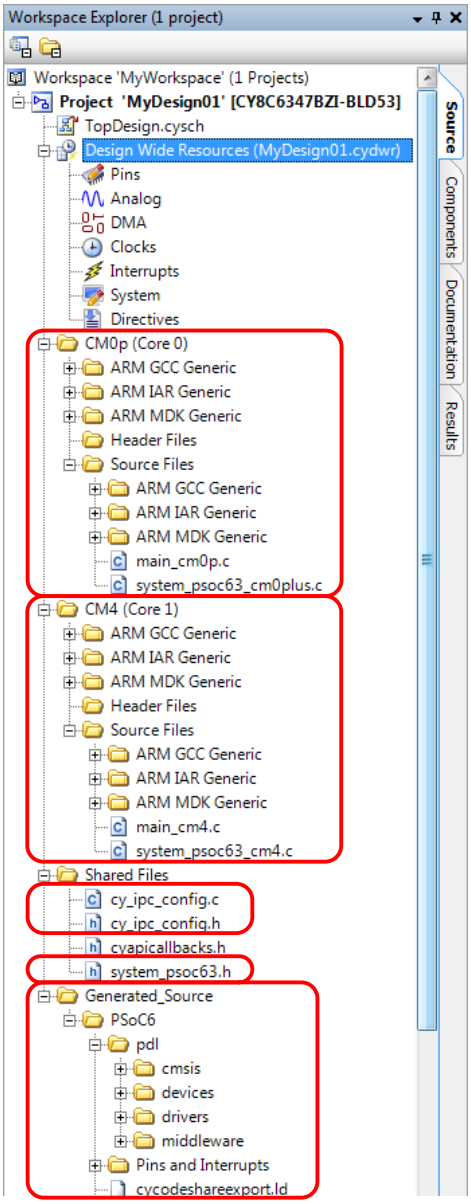
When schematic design entry is complete, select **Build** > **Generate Application**. This creates several system source code files and folders in the existing folders as well as in the new folder *Generated Source*, as Figure 9 shows.

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)*.

Figure 9. Add Generated Source to a Project



4.3 Resource Assignment Considerations

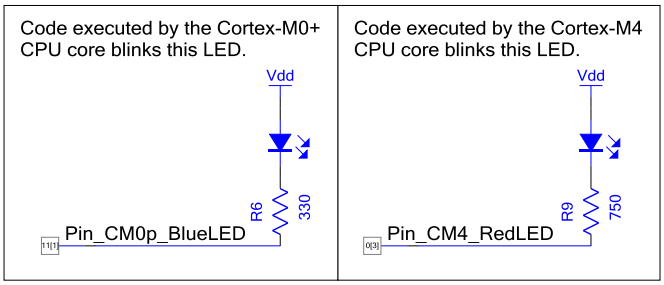
All PDL driver source and other API files are available to both CPUs. If code for a CPU references any API element in a generated source file, that file is compiled into the binary for that CPU. The same file can be compiled into both binaries – see code example CE216795, *PSoC 6 MCU Dual-CPU Basics*.

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



- **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 PDL 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*.

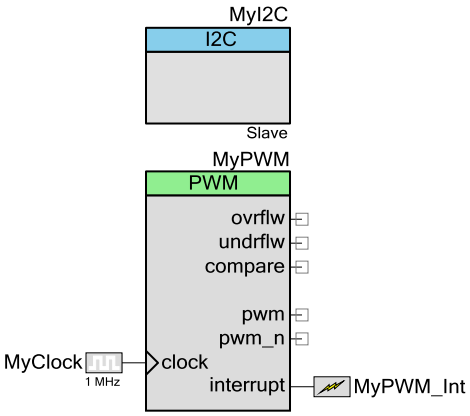
ModusToolbox software: In ModusToolbox software, interrupts are assigned programmatically; at this time, there is no GUI support. Examples of the required code are in AN217666; and in the PDL documentation, Cypress BLE Middleware Library section, Configure BLESS Interrupt subsection. The code in this 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. Cypress HAL is only supported in CM4, therefore all interrupts required by a HAL driver are assigned to CM4.

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.

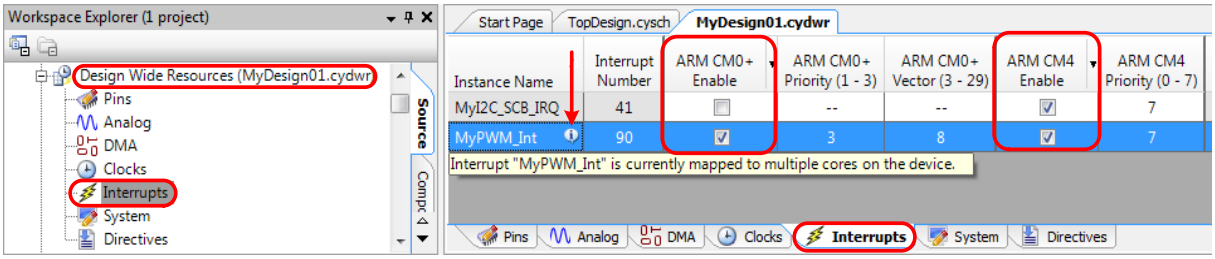
In this example, the I2C Component has an interrupt embedded in it. That interrupt is not shown on the schematic in Figure 11; it is shown in the Design-Wide Resources window as MyI2C_SCB_IRQ.

Figure 11. 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 12. 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.

4.5 Debug Considerations

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

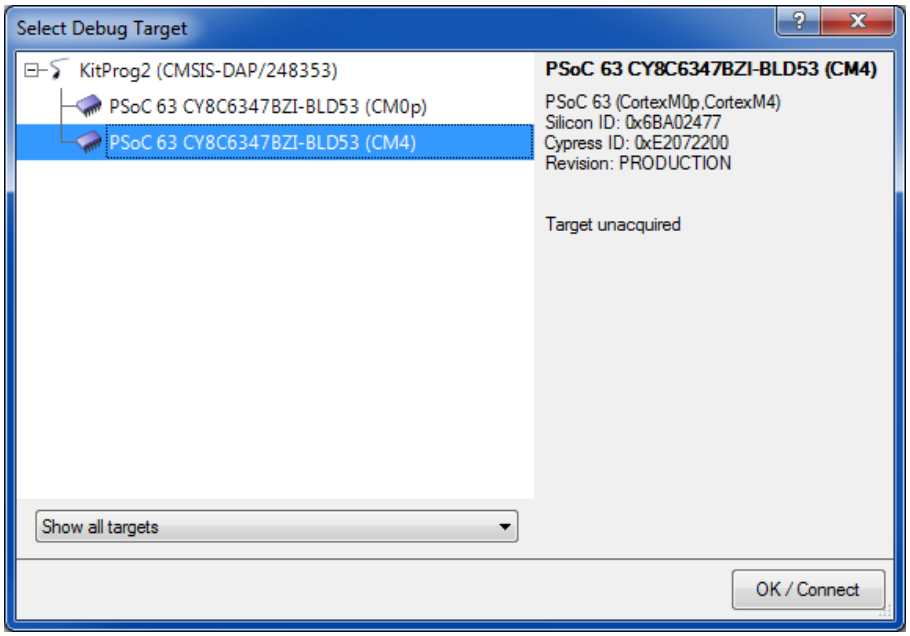
4.5.1 ModusToolbox Instructions

Eclipse IDE for ModusToolbox software supports debugging a single CPU (either CM4 or CM0+) at a time. For detailed instructions, open the ModusToolbox User Guide, and see the subsection **PSoC 6 MCU Programming/Debugging**.

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 13 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 13. PSoC Creator Select CPU for Debug



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

4.5.3.1 Exporting from ModusToolbox Software

ModusToolbox software has an export mechanism for the following IDEs through the make file:

1. IAR Embedded Workbench: `make ewarm8`
2. Keil µVision: `make uvision5`
3. Visual Studio Code: `make vscode`

For more details on how to export and set up debugging with these tools, see the “Exporting to IDEs” chapter in the ModusToolbox User Guide.

All these IDEs support dual-CPU debugging; however only Keil µVision dual-CPU debugging works with projects exported through ModusToolbox.

The instructions on the ModusToolbox User Guide are designed to program/debug CM4. The same instructions apply for CM0+, with a few changes. Depending on which IDE you are using, do the following extra steps:

1. **Visual Studio Code with CM0+**

After executing the `make vscode` command for the CM0+ project, a new folder called ".vscode" is created. In the folder, open the *launch.json* file and make the following changes:

a. Rename all instances of CM4 to CM0+

b. Set all `targetProcessor` instances to 0 (instead of 1).

2. **Keil µVision with CM0+**

Follow the same steps in creating projects for CM4. To perform dual-CPU debugging, open two instances of Keil µVision, one with the CM0+ project, and the other with the CM4 project. In both instances, go to **Debug > Start/Stop Debug Session.**

3. **IAR Embedded Workbench**

The IAR export mechanism in ModusToolbox does not support CM0+ projects. You can export CM4 projects only.

4.5.3.2 Exporting from PSoC Creator

If you are using PSoC Creator, you can export your project to Keil µVision or IAR Embedded Workbench for dual-CPU debugging. Following are the steps to do so:

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

1. Configure the PSoC Creator Project

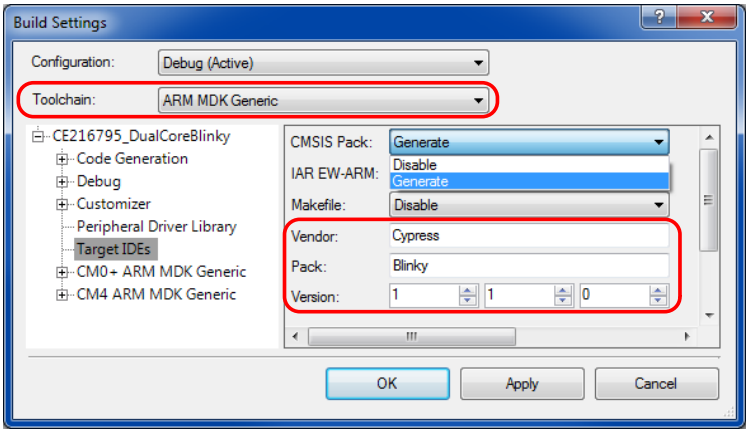
Update the **Target IDEs** settings in the project Build Settings, as Figure 14 shows.

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.) IAR has its own compiler (not supported by PSoC Creator), so the Toolchain selection is irrelevant.

Figure 14. Build Settings for Target IDEs

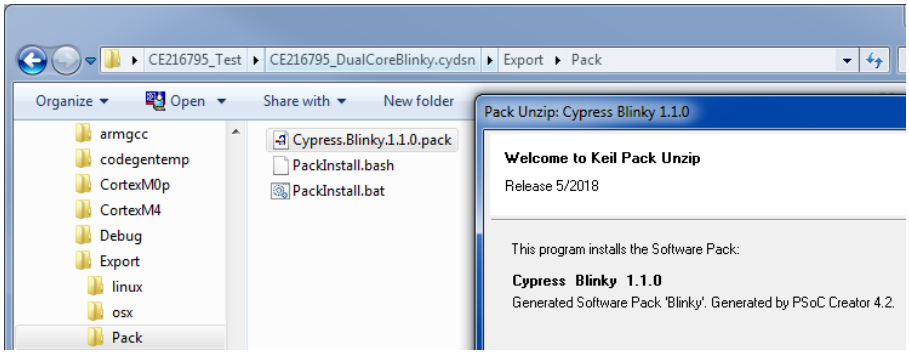


Then build your PSoC Creator project in the usual manner. A folder *Export* is created in your *<project>.cydsn* folder, which contains relevant files for exporting to the selected IDE or IDEs.

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 15 shows.

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

Figure 15. Install µVision Pack from PSoC Creator Project



Note: if you update the PSoC Creator project, consider changing the µVision pack version number (see Figure 14) and installing the new pack.

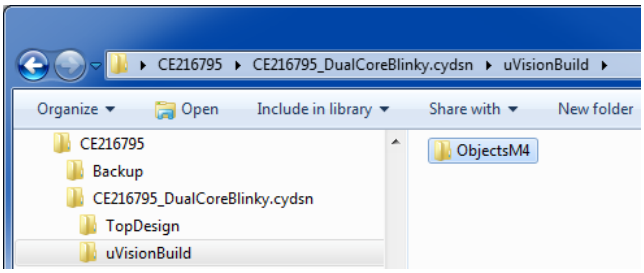
For more information, see AN219434 - Importing PSoC Creator Code into an IDE for a PSoC 6 Project.

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 16 shows:

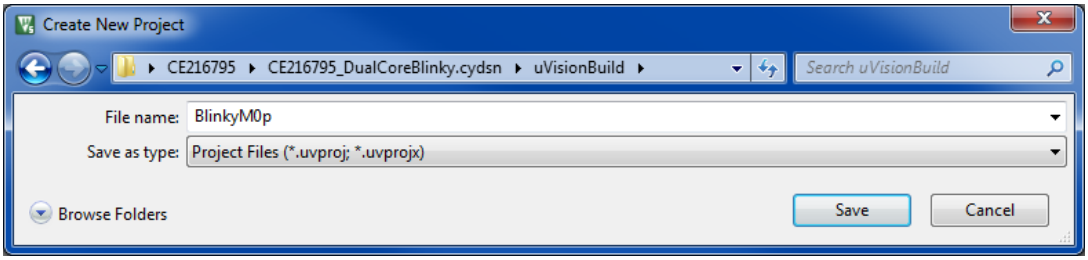
Figure 16. New Folders for µVision Projects



Open µVision 5.25 or later, and create a new project (**Project** > **New µVision Project...**) in the *uVisionBuild* folder.

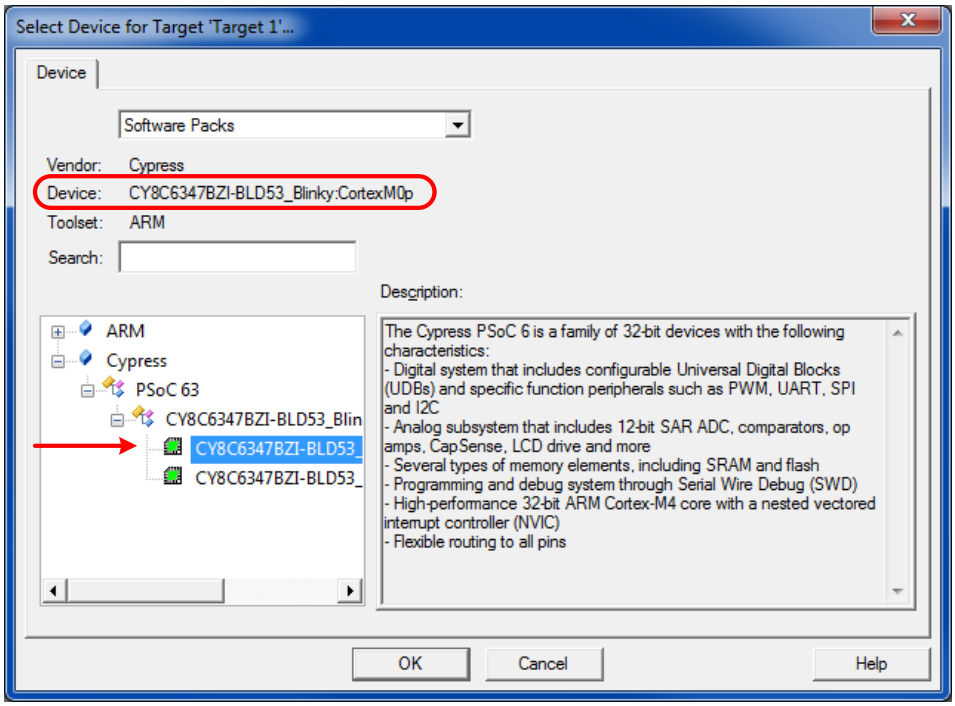
Recommended: name the project based on the original PSoC Creator project name and the target CPU. For example, for the CE216795 dual-CPU blinky project, create a µVision project *BlinkyM0p* for the CM0+ CPU, as Figure 17 shows:

Figure 17. Create a µVision Project for CM0+



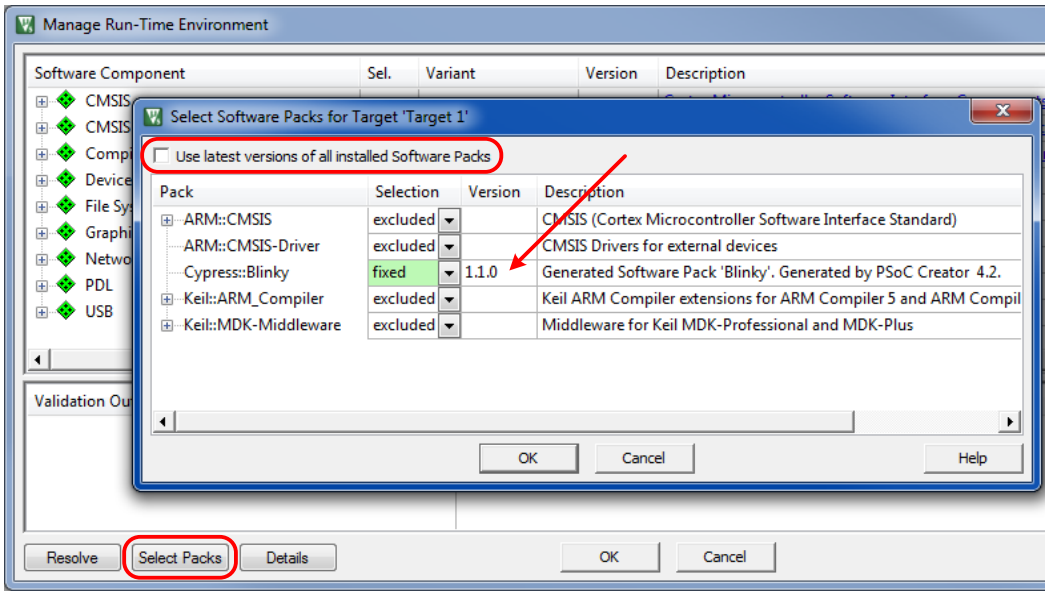
After you click **Save**, a **Select Device for Target 'Target 1'...** dialog box is displayed. The two PSoC 6 MCU CPUs that were defined in the previously installed pack (Figure 15) are displayed. Select the CM0+ CPU, as Figure 18 shows. Click **OK**.

Figure 18. 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 19 shows:

Figure 19. Select the PSoC Creator Project Pack



Click **OK**; the Manage Run-Time Environment dialog changes as Figure 20 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 21 shows.

Figure 20. Select Pack Startup and PDL Driver Files

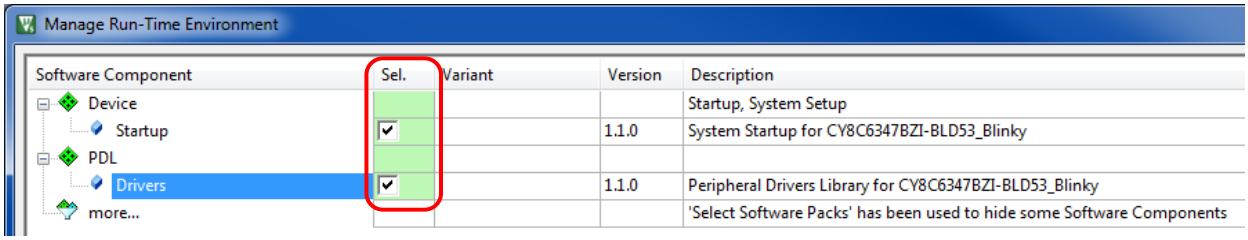
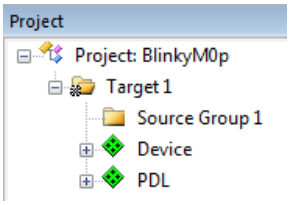
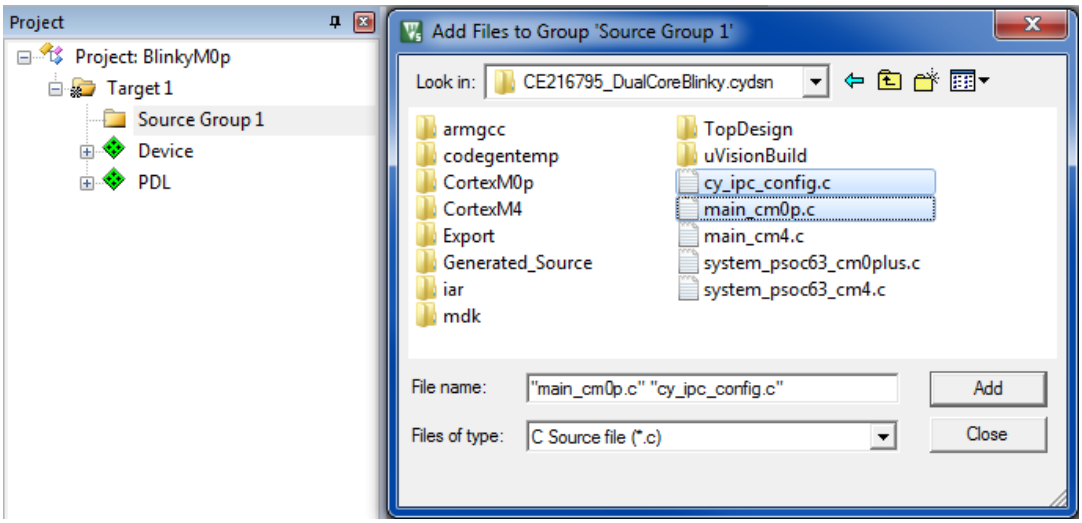


Figure 21. Initial Project Creation



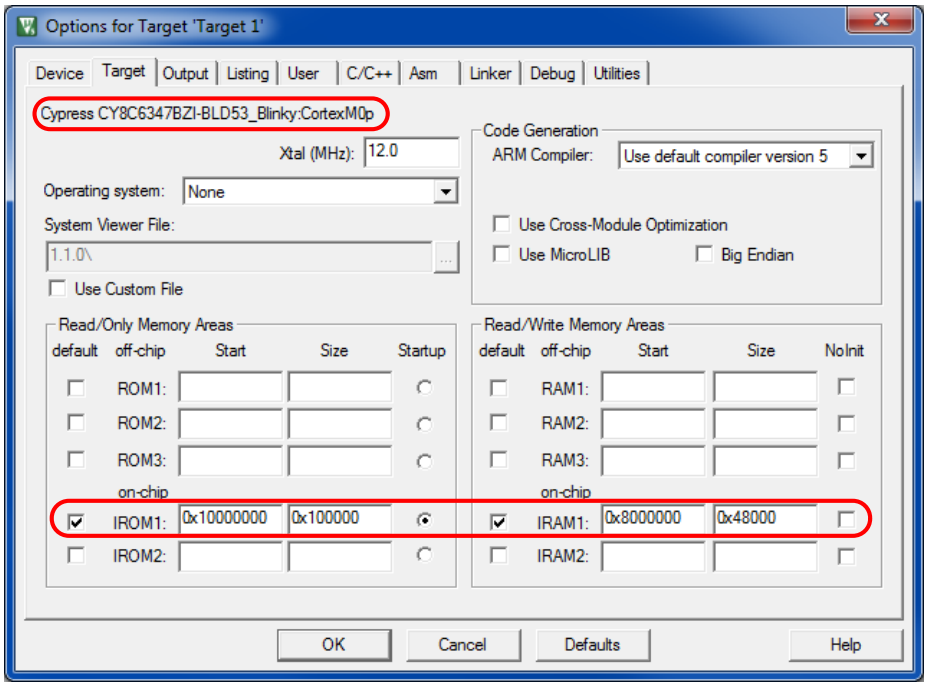
Right-click **Source Group 1**, and select **Add Existing Files to Group 'Source Group 1'...**. Navigate to your PSoC Creator project folder and select *main_cm0p.c*, *cy_ipc_config.c*, and all other non-system *.c* and assembler files needed for your project, as Figure 22 shows. You do not have to add 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 22. Add PSoC Creator Project C Source Files to the Source Group



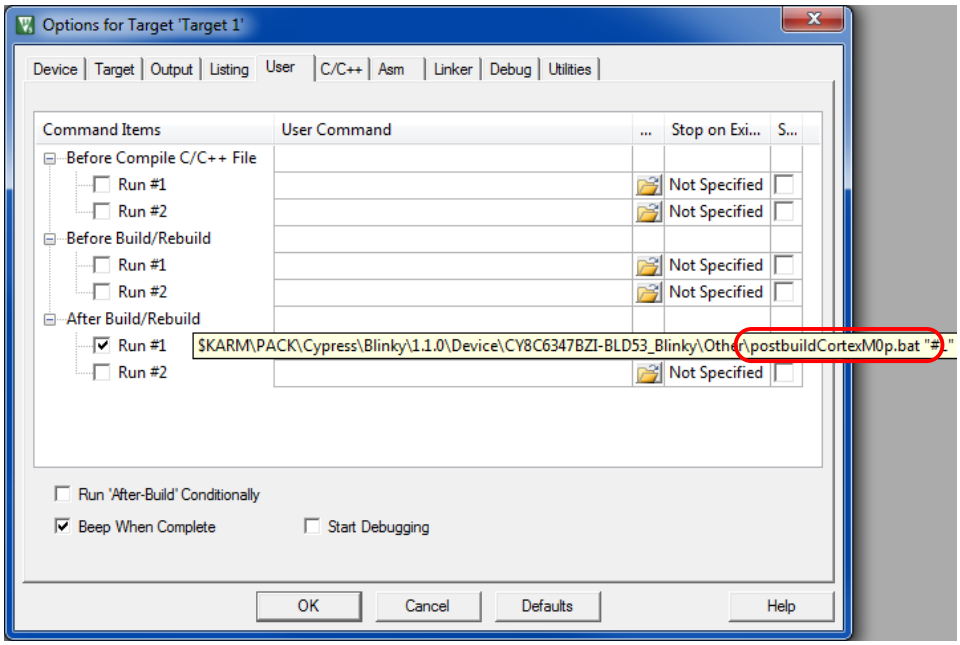
Now that the project is created, you must set its options. Right-click **Target 1**, and select **Options for Target 'Target 1'...**. Confirm in the **Target** tab that the device, CPU, IROM1, and IRAM1 are correct for your PSoC 6 MCU device, as Figure 23 shows. Updating other fields such as Xtal (MHz) and Operating system is optional.

Figure 23. Project Target Options



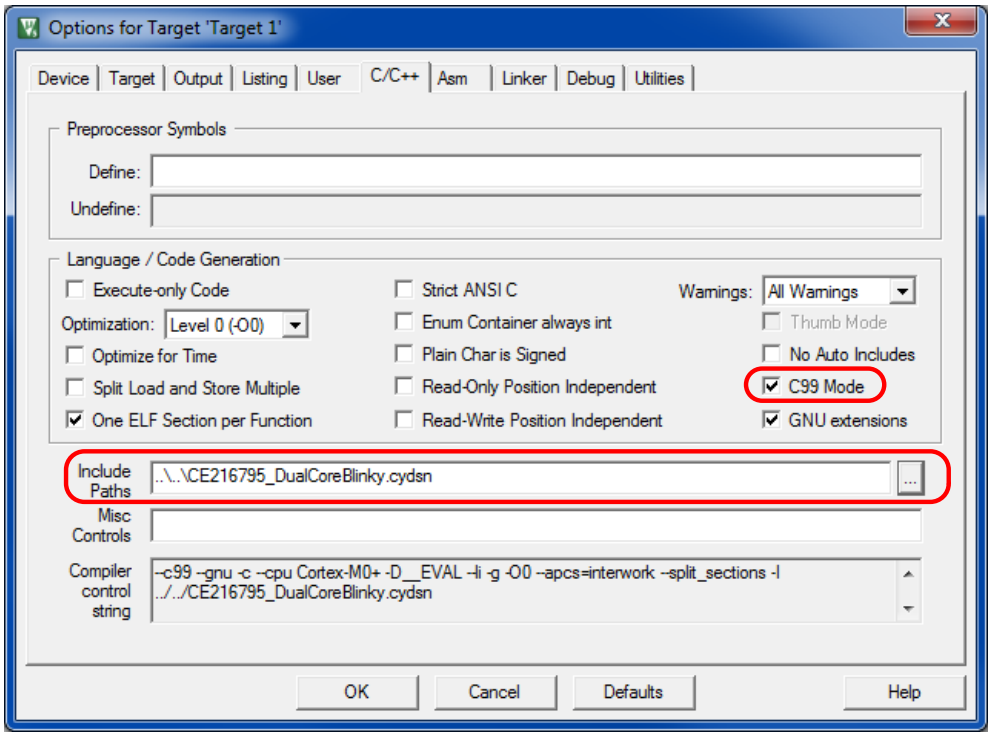
In the **User** tab, verify that the correct post-build batch file from the pack is being called. Hover the cursor over the **User Command** field and confirm that *postbuildCortexm0p.bat* is called, as Figure 24 shows. Add other pre- and post-build batch files, and select other options, as needed.

Figure 24. Project User Options



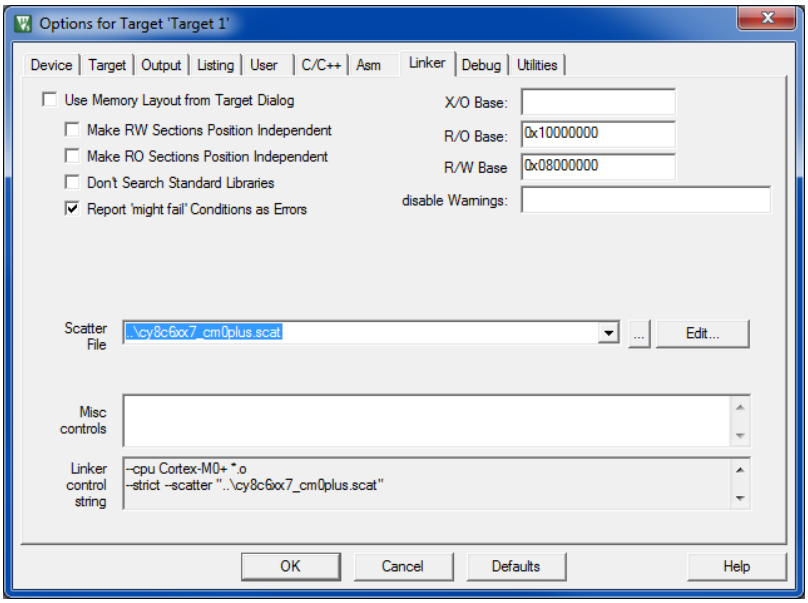
Confirm in the **C/C++** tab that the **C99 mode** option is checked, as Figure 25 shows. (PDL is developed based on C99.) Add the PSoC Creator *<project>.cydsn* folder to the **Include Paths**; this provides a link to the *.h* files in the PSoC Creator project. Update other options and fields as needed.

Figure 25. Project C/C++ Options



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 26 shows. Select the appropriate **Scatter File** from your PSoC Creator project folder.

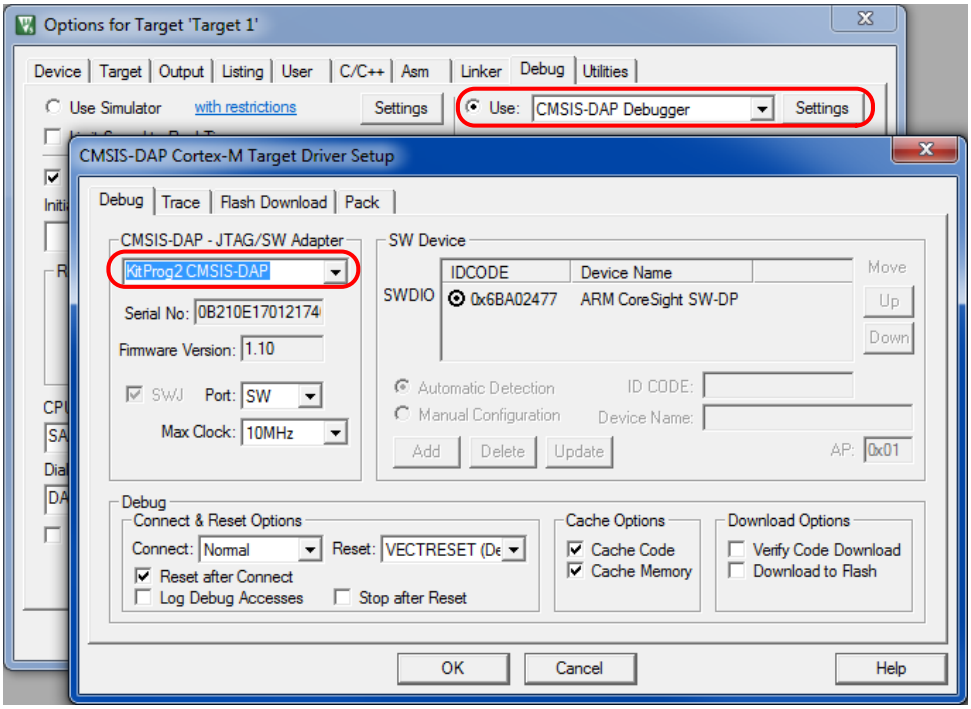
Figure 26. Project Linker Options



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.

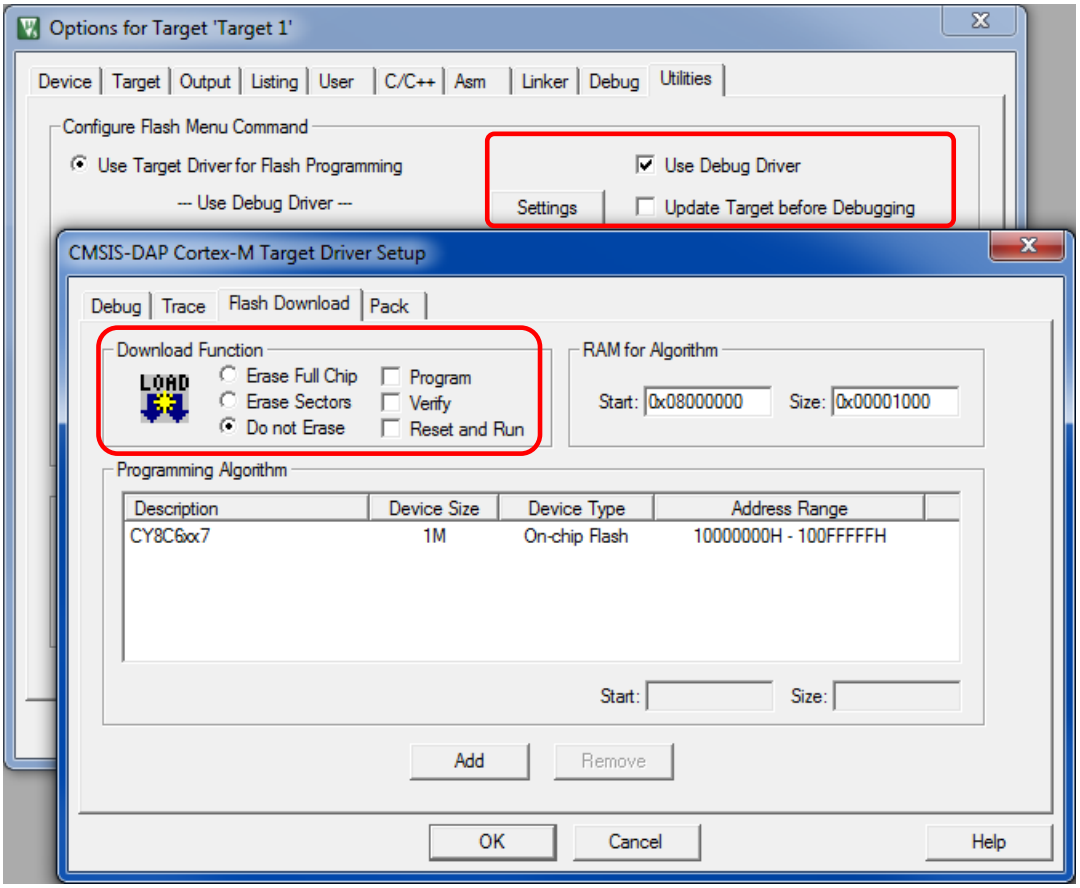
In the **Debug** tab, select **Use CMSIS-DAP Debugger**, as Figure 27 shows. Click **Settings**, select **KitProg2 CMSIS-DAP**, and confirm that all other settings are at the defaults shown. Click **OK** and go back to the Options dialog.

Figure 27. 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 28 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 28. Project Utilities Options

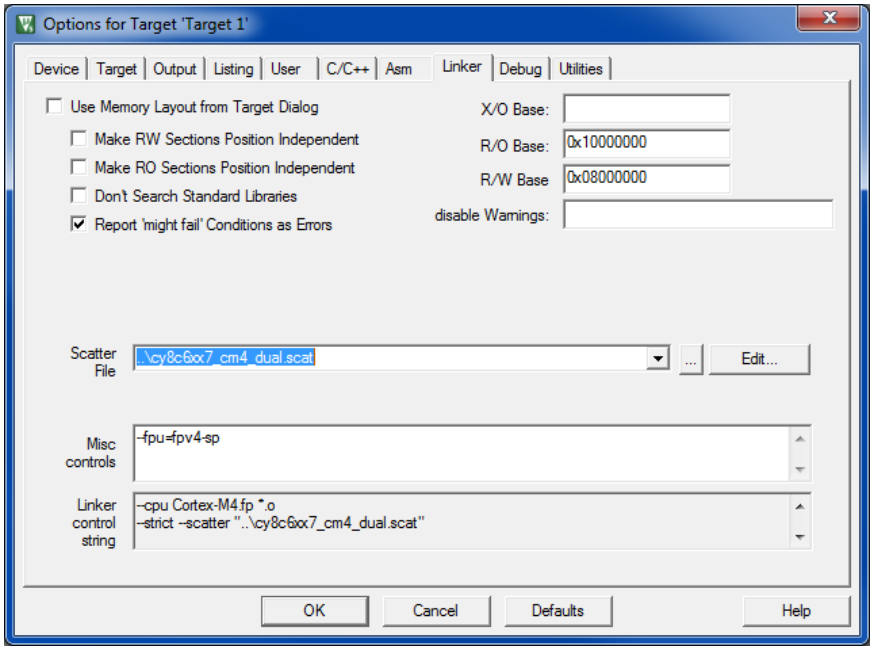


Repeat the previous steps and create a second project for CM4.

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 17. Configure the project in the same manner as the CM0+ project, with the following differences:

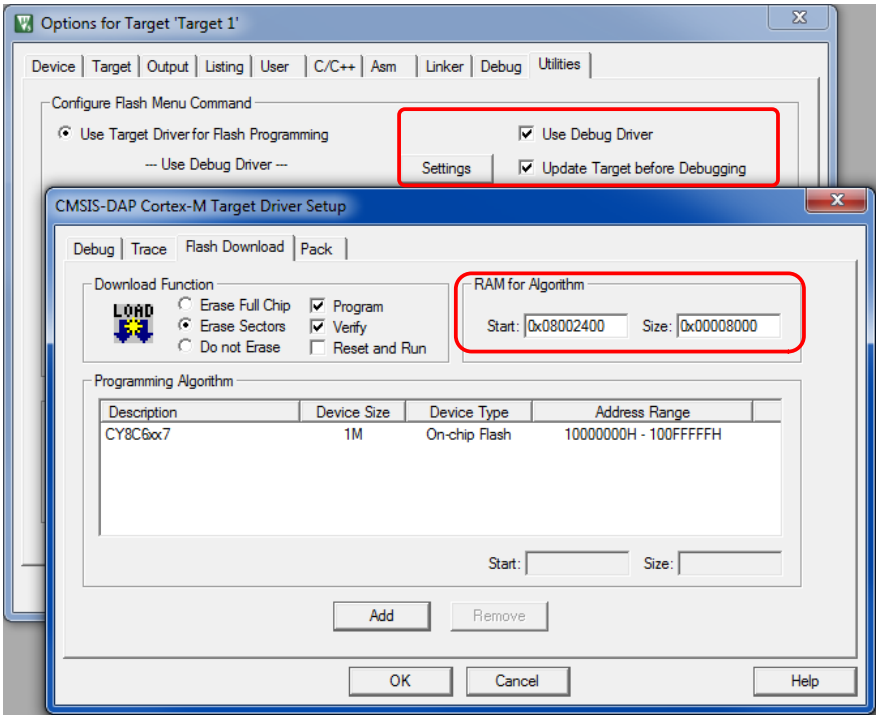
- The CM4 project must be in the same folder as the CM0+ project; in this case, *uVisionBuild*. See Figure 17.
- Select the CM4 CPU from the previously installed pack; see Figure 18.
- Navigate to your PSoC Creator project folder and select *main_cm4.c*, *cy_ipc_config.c*, and all other non-system *.c* and assembler files needed for your project, as Figure 22 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 16.
- In the **Options** dialog, **C/C++** tab, add `--fpu=fpv4-sp` to **Misc Controls**; see Figure 25.
- In the **Options** dialog, **Linker** tab, select the "cm4_dual" scatter file, as Figure 29 shows. The CM4 project will contain code for both CPUs. Add `--fpu=fpv4-sp` to **Misc Controls**.

Figure 29. Linker Options for CM4 Project



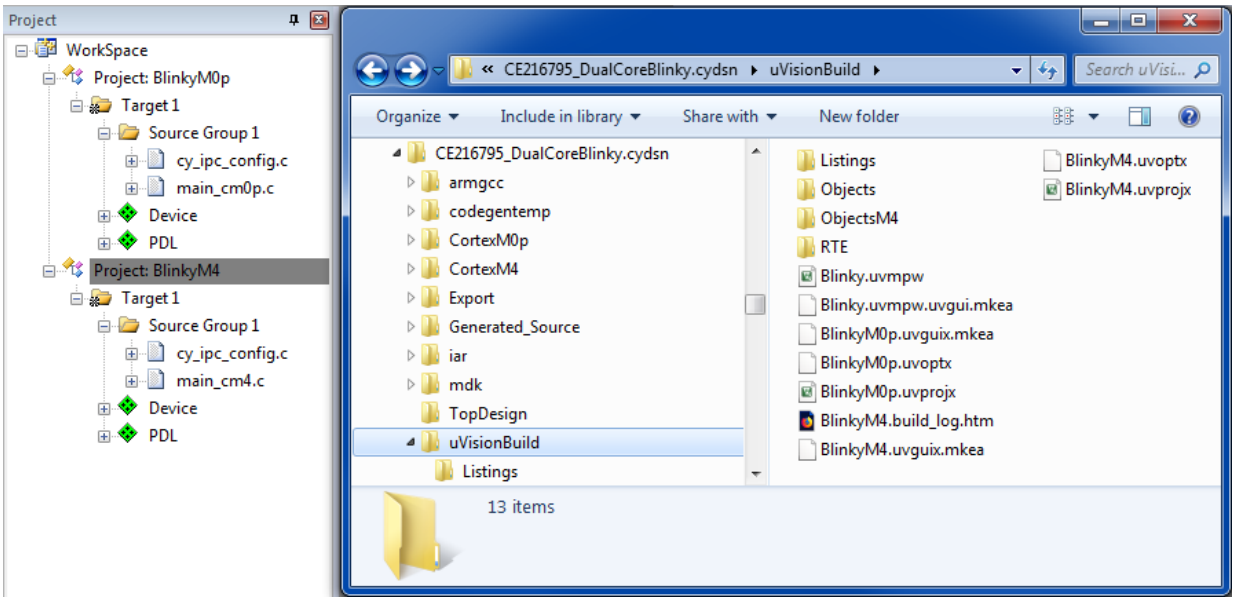
- In the **Options** dialog, **Debug** tab, Target Driver Setup, select VECTRESET for the **Reset** option; see Figure 27.
- In the **Options** dialog, **Utilities** tab, confirm that **Update Target before Debugging** is checked, as Figure 30 shows. Set the RAM for Algorithm values as indicated. Checking **Reset and Run** is optional but convenient.

Figure 30. 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 31.

Figure 31. Resultant µVision Project Window and Project Files



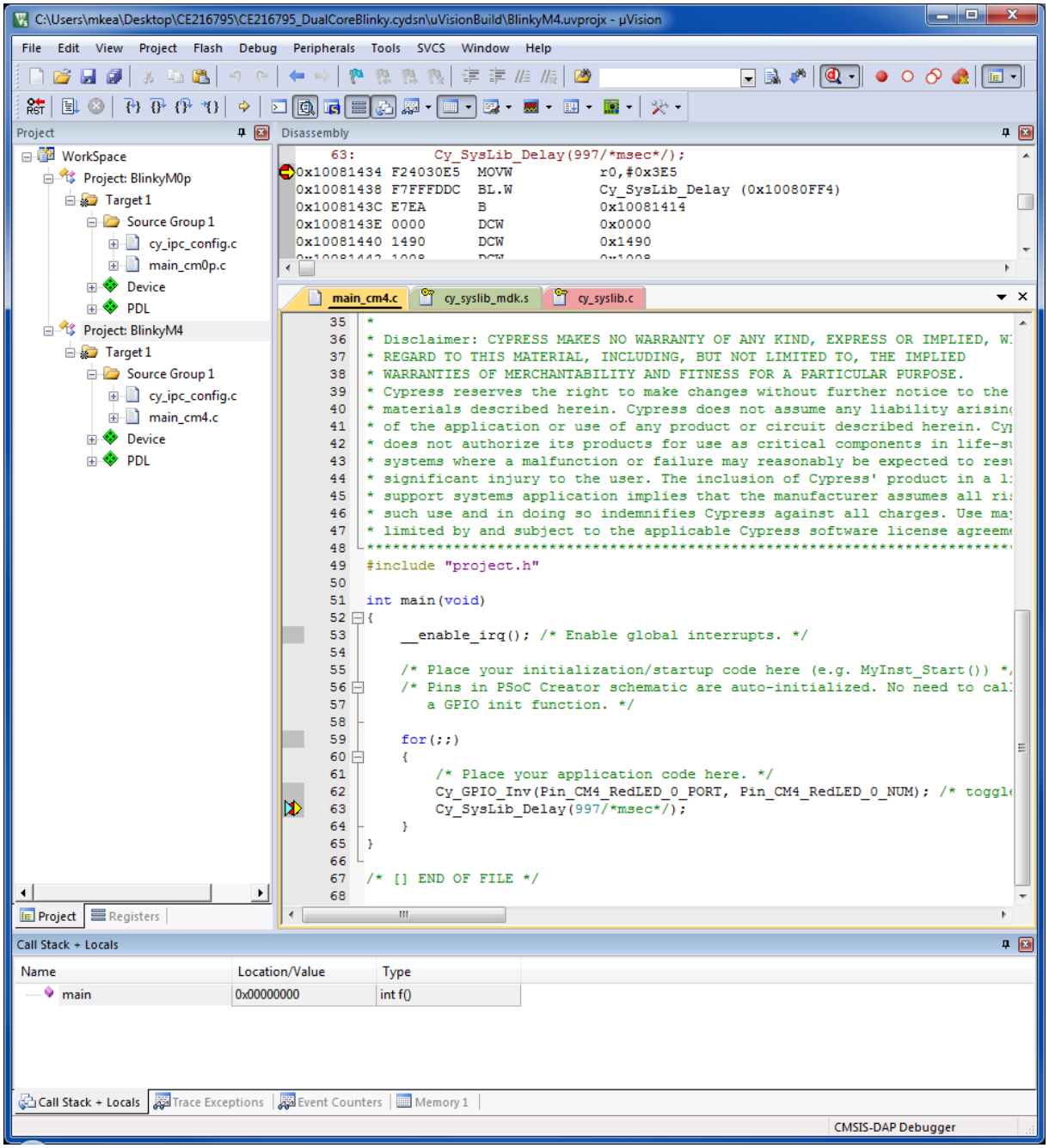
Build the projects in sequence; build the CM0+ project first. Note that µVision has a batch build feature to automate the process. After building is successfully completed, right-click the BlinkyM4 project and set it as the active project. Then test your build options by (1) erasing flash (**Flash** > **Erase**), and (2) downloading the project (**Flash** > **Download**) and confirming correct operation. If you did not select Reset and Run (see Figure 30), 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 32:

Figure 32. CM4 Debug Window



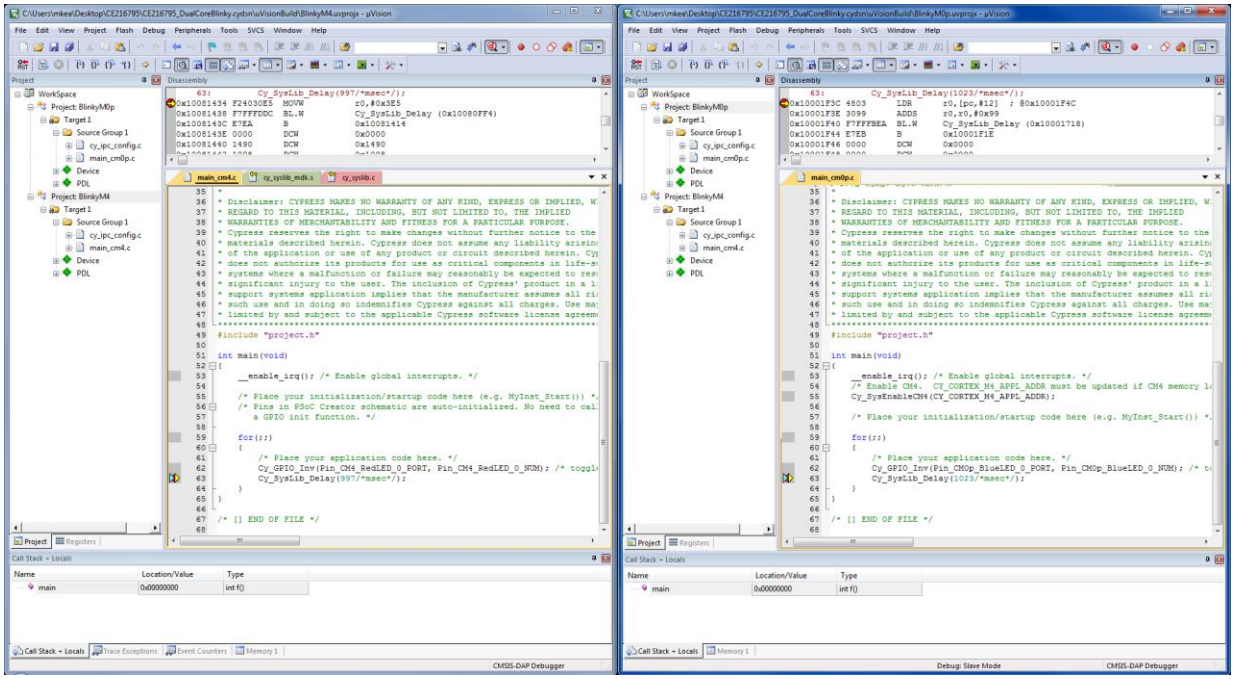
If you are running the CE216795 dual-CPU blinky project, set a breakpoint at line 63, `Cy_Syslib_Delay()`. Then repeatedly click **Debug** > **Run**, and the red LED toggles on each stop at the breakpoint.

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 33. 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 33. µVision Dual-CPU Debugging



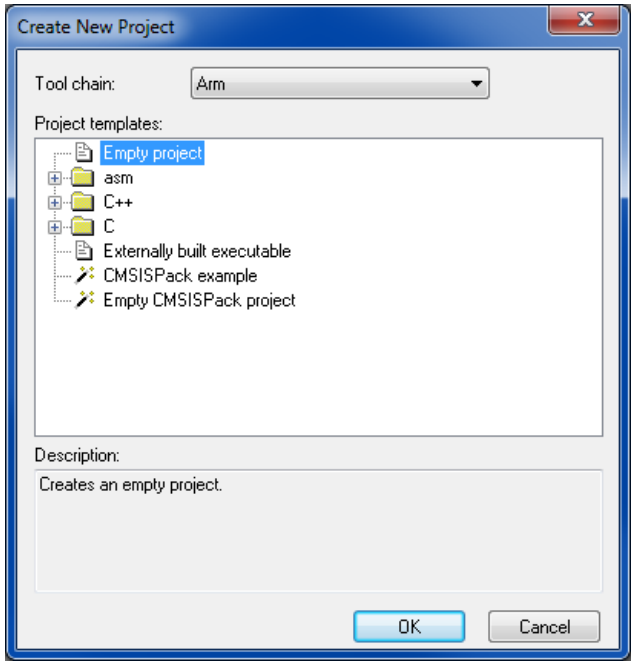
4. Create IAR-EW Projects

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.

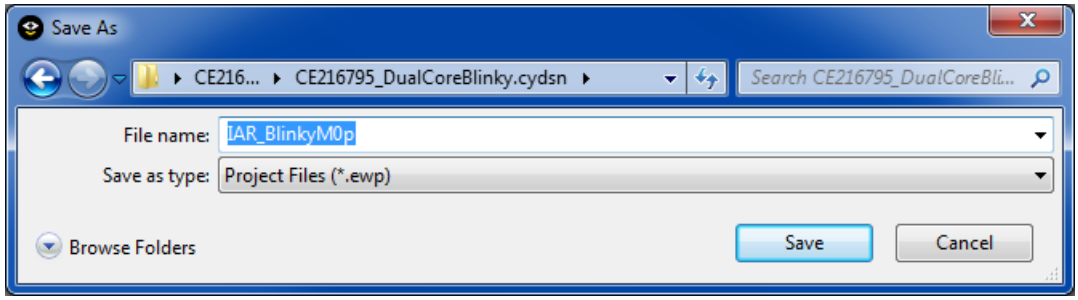
Open IAR Embedded Workbench for ARM 8.22 or later, and create a new project (**Project** > **Create New Project...**). In the Create New Project dialog (Figure 34), confirm that the Tool chain is **Arm**, select the **Empty project** template, then click **OK**.

Figure 34. IAR Embedded Workbench Create New Project Dialog



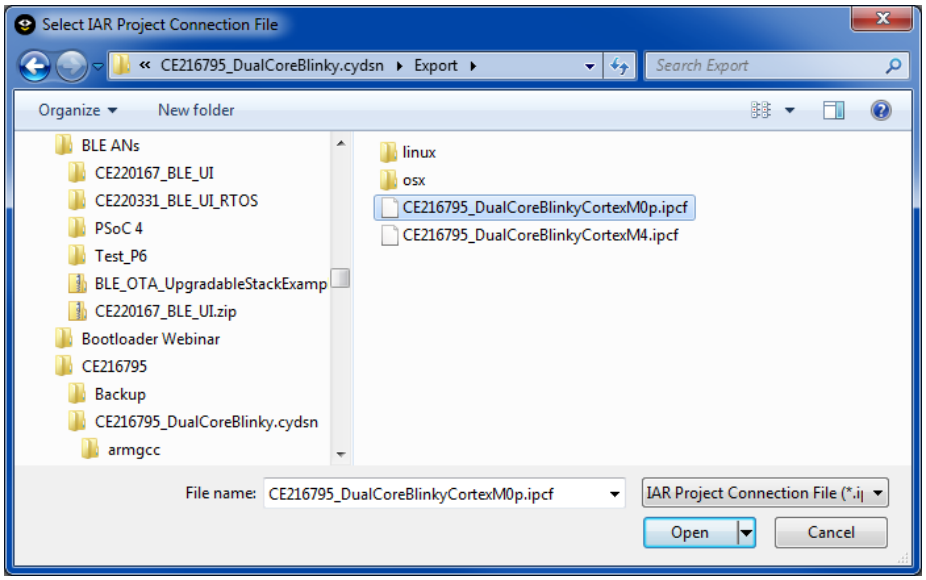
Recommended: in the Save As dialog (Figure 35), name the project based on the original PSoC Creator project name and the target CPU. For example, for the CE216795 dual-CPU blinky project, create a µVision project *IAR_BlinkyM0p* for the CM0+ CPU.

Figure 35. Create an IAR Embedded Workbench Project for CM0+



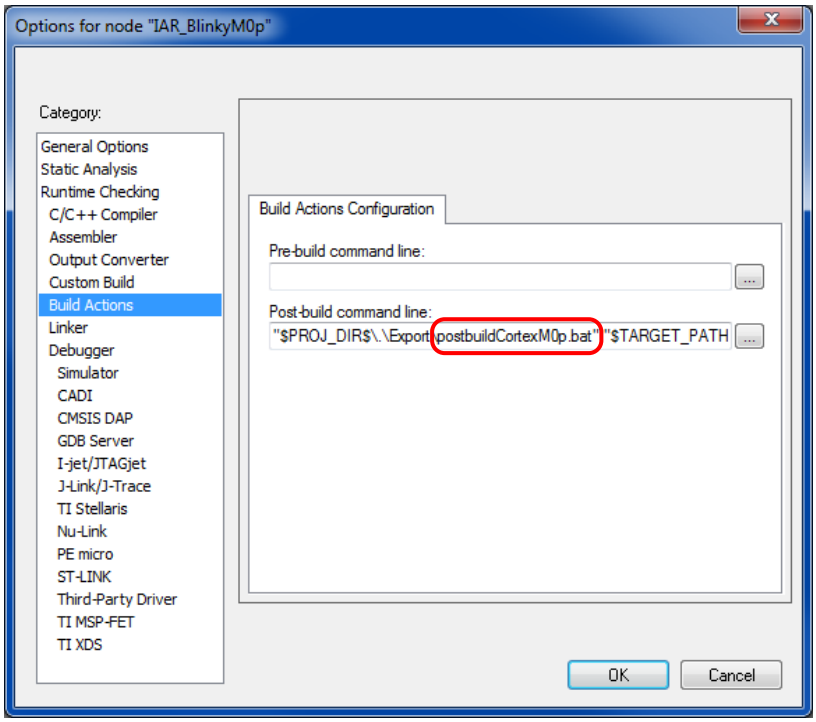
Select **Tools** > **Options** and make sure that **Enable project connections** is checked. Click **OK**. Then select **Project** > **Add Project Connection...**. In the next dialog, select Connect using **IAR Project Connection**, and click **OK**. Then select the *...CortexM0p.ipcf* file, as Figure 36 shows. Click **OK**, and several folders and files are added to the project in the Workspace window.

Figure 36. 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 37 shows.

Figure 37. Select PSoC Creator Post-Build Batch File

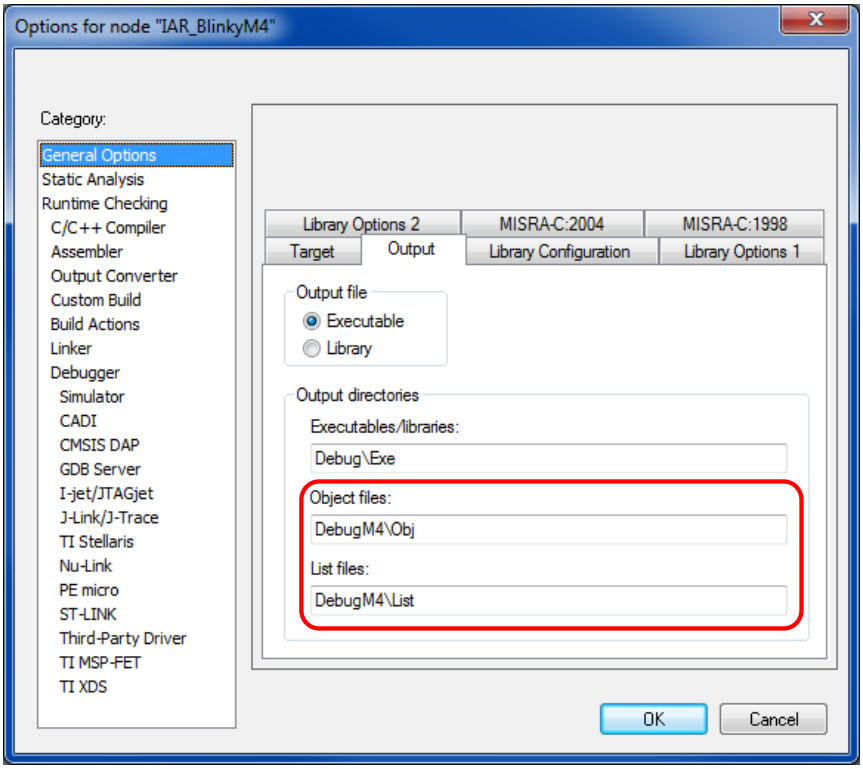


In the **Debugger** section, **Setup** tab, select the **CMSIS DAP** driver. In the **Download** tab, check **Suppress download**. In the **CMSIS DAP** section, **Setup** tab, set **Reset** to **Disabled (no reset)**. The application will be loaded by the CM4 project. In the **Interface** tab, select **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 35. 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 35.
- Select the *...CortexM4.ipcf* file; see Figure 36.
- In the **Options** dialog, **General Options** section, **Output** tab, change the output directories for object and list files, as Figure 38 shows. Do not change the executables/libraries output folder.

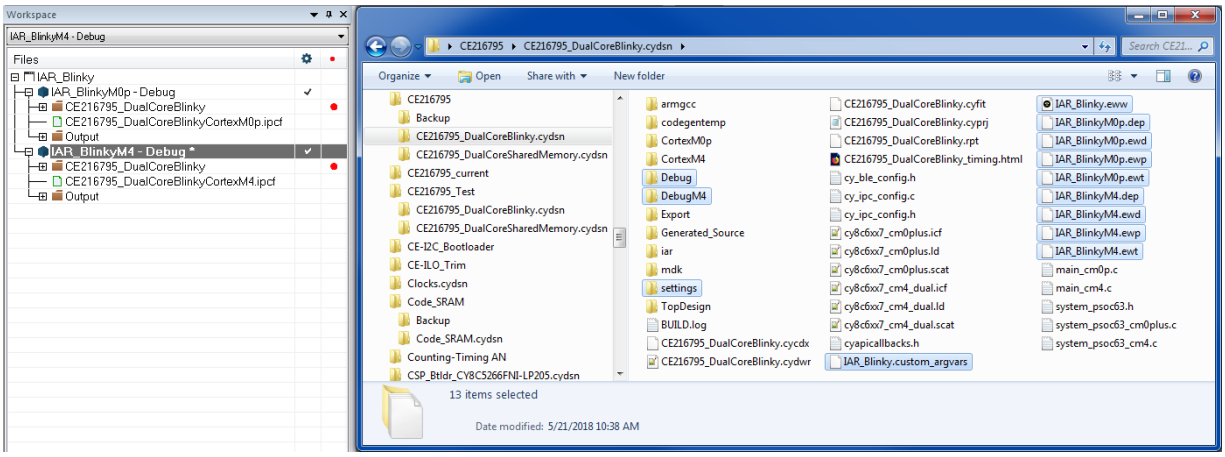
Figure 38. Unique Output Folders for CM4 Project



- In the **Build Actions** section, confirm that *postbuildCortexM4.bat* is called, see Figure 37.
- 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 39. The files and folders generated by IAR-EW are highlighted.

Figure 39. Resultant IAR Embedded Workbench Project Window and Project Files

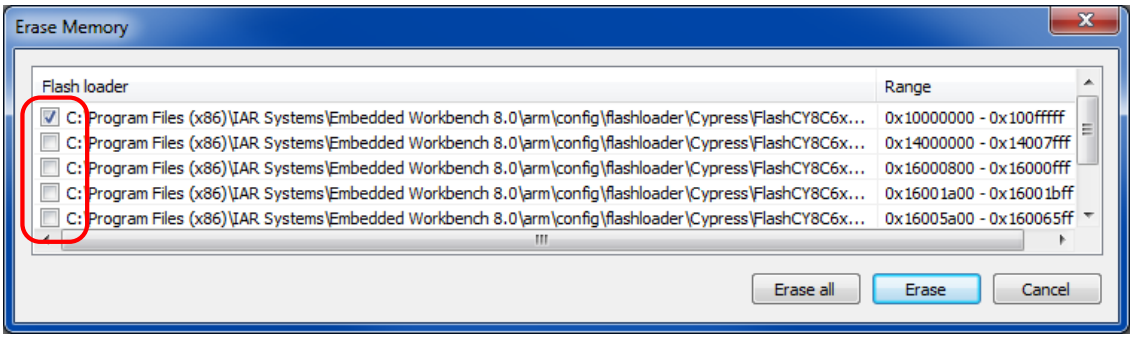


Connect the CY8CKIT-062-BLE USB port to your computer. Press kit button SW3 to put KitProg2 into CMSIS-DAP mode; see the kit guide for details. This allows debugging without using any external debug probes.

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 40 shows:

Figure 40. IAR Embedded Workbench Erase Memory Dialog for PSoC 6 MCU

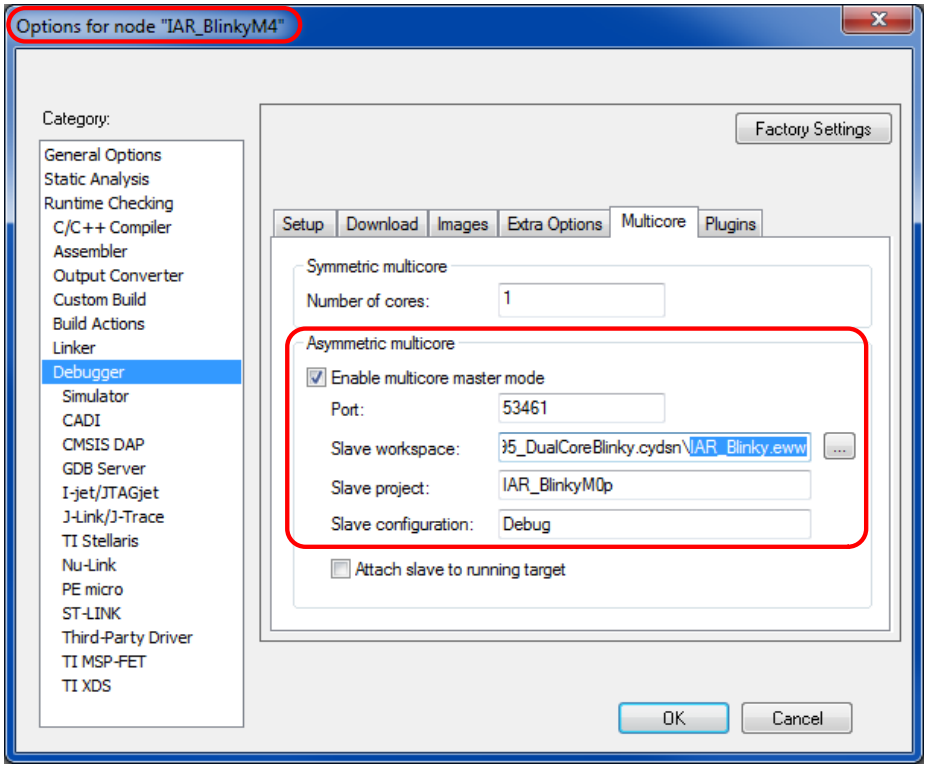


Note: If you change any code in the CM0+ project, you must rebuild both projects. Note that IAR-EW has a batch build feature to automate the process.

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 41 shows. Checking **Enable multicore master mode** makes the CM4 CPU the master for download and debugging purposes. Do not change the **Port**.

Figure 41. Set Up Multicore Debugging



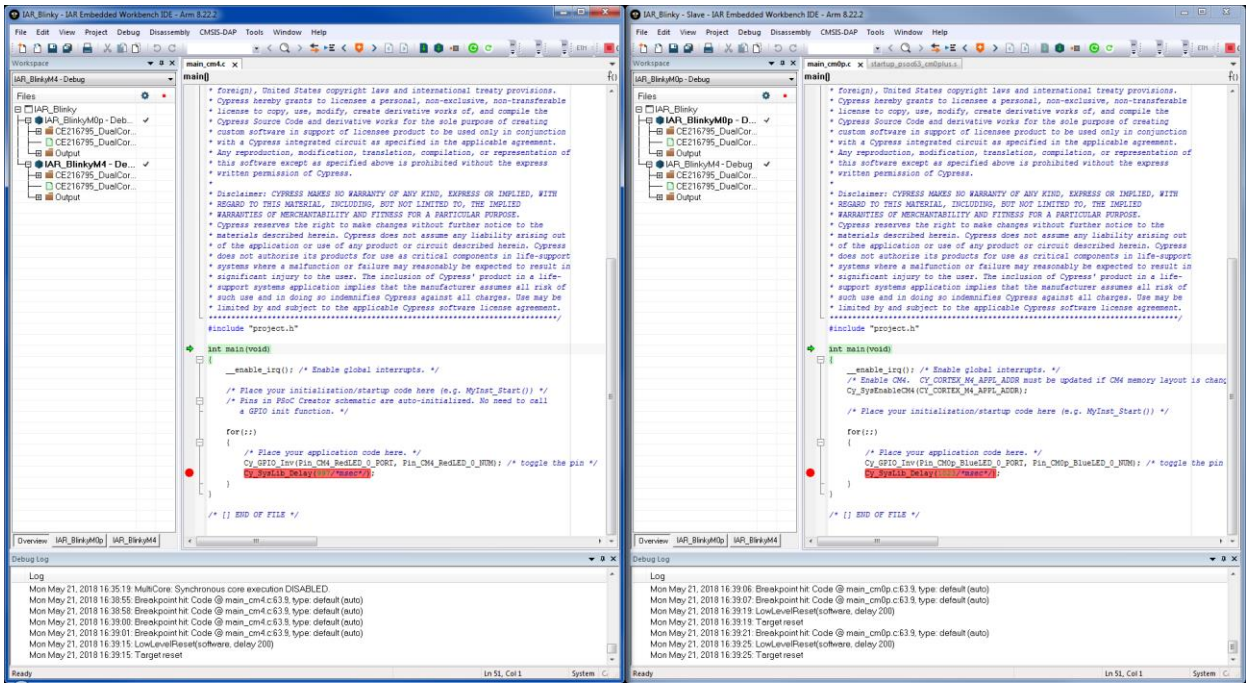
Select **File** > **Save All** to save the project options changes. Then start debugging by selecting either **Project** > **Download and Debug** or **Project** > **Debug without Downloading**. A second (slave) instance of IAR Embedded Workbench is automatically opened for the CM0+ project. Both instances 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 42. 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 42. 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 PSoC 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 software 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. The Cypress HAL only supports the CM4 CPU, therefore the DMA HAL driver can only work for CM4 applications.

- **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. The Cypress HAL does not have a crypto driver at the moment.

- **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. At the moment, only PSoC Creator supports UDBs. ModusToolbox software does not have any support for UDBs.

- 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 I/O™, which can be used to perform Boolean operations directly on signals going to, and coming from, GPIO pins. The Cypress HAL does not have support for Smart I/O.

- Other smart peripherals include serial communication blocks (SCB), counter/timer/PWM blocks (TCPWM), Bluetooth Low Energy (BLE), I2S/PDM audio, programmable analog, and CapSense®. 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.

ModusToolbox software provides a set of tools for setting up peripherals, pre-defined BSPs for all Cypress kits, libraries for popular functionality like CapSense and emWin, and a comprehensive array of example applications to get you started.

Cypress offers extensive application note and code example support for PSoC peripherals, as well as detailed data in the device datasheets, PDL documentation, HAL 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.

Application Notes	
AN221774 – Getting Started with PSoC 6 MCU	Describes PSoC 6 MCU devices and how to build your first ModusToolbox 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
AN217666 – PSoC 6 MCU Interrupts	Describes PSoC 6 MCU interrupt architecture and how to configure interrupts
AN219434 – Importing PSoC Creator Code into an IDE for a PSoC 6 MCU Project	Describes how to import the code generated by PSoC Creator into your preferred IDE
Code Examples (PSoC Creator)	
CE216795 – PSoC 6 MCU Dual-CPU 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.
CE223820 – PSoC 6 MCU IPC Pipes	This example demonstrates how to use the inter-processor communication (IPC) driver to implement a message pipe in PSoC 6 MCU. The pipe is used to send messages between CPUs.
CE223549 – PSoC 6 MCU IPC Semaphore	This example demonstrates how to use the inter processor communication (IPC) driver to implement a semaphore in PSoC 6 MCU. The semaphore is used as a lock to control access to a resource shared by the CPUs.
CE226306 – PSoC 6 MCU Power Measurements	This example shows how to achieve the power measurements listed in the PSoC 6 MCU datasheets.
Code Examples (ModusToolbox)	
mtb-example-psoc6-dual-cpu-empty-app	This is a minimal starter Dual-CPU application template for PSoC 6 MCU devices.
mtb-example-psoc6-dual-cpu-ipc-sema	This example demonstrates how to use the inter-processor communication (IPC) driver to implement a semaphore in PSoC 6 MCU. The semaphore is used to lock to control access to a resource shared by the CPUs and synchronize the initialization instructions.
mtb-example-psoc6-dual-cpu-ipc-pipes	This example demonstrates how to use the inter-processor communication (IPC) driver to implement a message pipe in PSoC 6 MCU. The pipe is used to send messages between CPUs
PSoC Creator Component Datasheets	
Interrupt	Supports generating interrupts from hardware signals
Device Documentation	
PSoC 6 MCU Datasheets	PSoC 6 Technical Reference Manuals
Development Kit Documentation	
CY8CKIT-062-BLE	PSoC 6 BLE Pioneer Kit
CY8CKIT-062-WiFi-BT	PSoC 6 WiFi-BT Pioneer Kit
CY8CKIT-062S2-43012	PSoC 62S2 Wi-Fi BT Pioneer Kit
CY8CPROTO-063-BLE	PSoC 6 BLE Prototyping Kit
CY8CPROTO-062-4343W	PSoC 6 Wi-Fi BT Prototyping Kit
CY8CPROTO-062S3-4343W	PSoC 62S3 Wi-Fi BT Prototyping Kit
CYW9P62S1-43438EVB-01	PSoC 62S1 Wi-Fi BT Pioneer Kit
CYW9P62S1-43012EVB-01	PSoC 62S1 Wi-Fi BT Pioneer Kit

Tool Documentation	
ModusToolbox Software	ModusToolbox software 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.
PSoC Creator	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.
Peripheral Driver Library (PDL)	Installed by PSoC Creator 4.3 and available through GitHub. Visit https://github.com/cypresssemiconductorco/psoc6pdl; for PSoC Creator, look in <*PDL install folder*>*doc* for the User Guide and API Reference
Hardware Abstraction Layer (HAL)	Available through GitHub only. Visit https://github.com/cypresssemiconductorco/psoc6hal

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

Revision	ECN	Date	Description of Change
**	5634375	02/16/2017	New application note
*A	5653730	03/08/2017	Updated template
*B	5777874	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	08/23/2017	Minor edits Ported to new application note document template Confidential tag removed
*D	6065641	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
*E	6201597	06/11/2018	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
*F	6403371	12/06/2018	Added support for ModusToolbox.
*G	6878749	05/25/2020	Updated instructions based on ModusToolbox 2.1 Added HAL support
*H	6954363	08/26/2020	Updated instructions based on ModusToolbox 2.2

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® Solutio

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

Cypress Semiconductor
An Infineon Technologies Company
198 Champion Court
San Jose, CA 95134-1709