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 the Cypress PSoC Creator™ Integrated Design Environment (IDE), and how to debug the design using various IDEs.

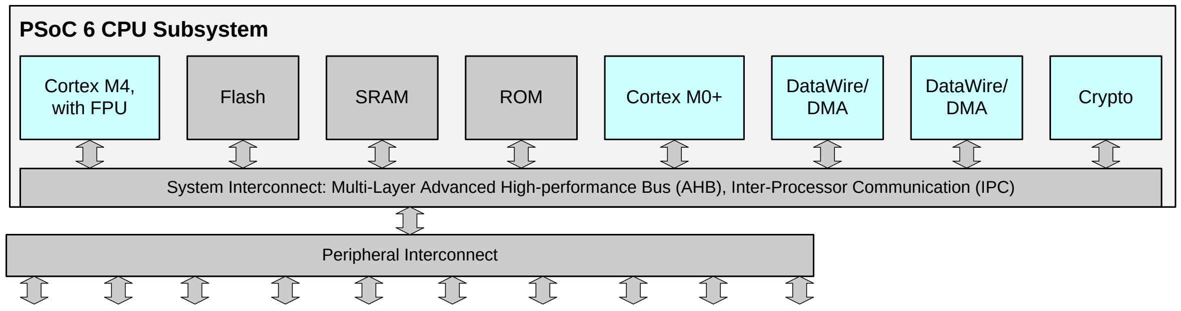
Contents

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

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

Figure 1. PSoC 6 MCU Typical CPU Subsystem Architecture



Note: The contents of the block diagram in Figure 1 may vary depending on the device. Some PSoC 6 MCU parts have only one CPU. See the device datasheet for details. This application note does not apply to single-CPU PSoC 6 MCU devices.

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

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

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

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

1.1 How to Use this Document

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

Note: Use PSoC Creator version 4.2 or higher for PSoC 6 MCU-based designs.

Initial sections of this application note cover general concepts for dual-CPU MCUs and how they are implemented in PSoC 6 MCU. To skip to an overview of creating a PSoC Creator project for a PSoC 6 dual-CPU MCU, go to the PSoC 6 MCU Dual-CPU Development 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 (CPU 1) has an RTOS and multiple tasks. The other CPU (CPU 2) is idle until CPU 1 messages it to do a specified task. CPU 2 wakes up and does the task, then messages the result back to CPU 1. As an example, CPU 1 can be a lower-performance CPU, and it uses CPU 2, the higher-performance CPU, to do computation-intensive tasks when needed.

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

Debug: 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.

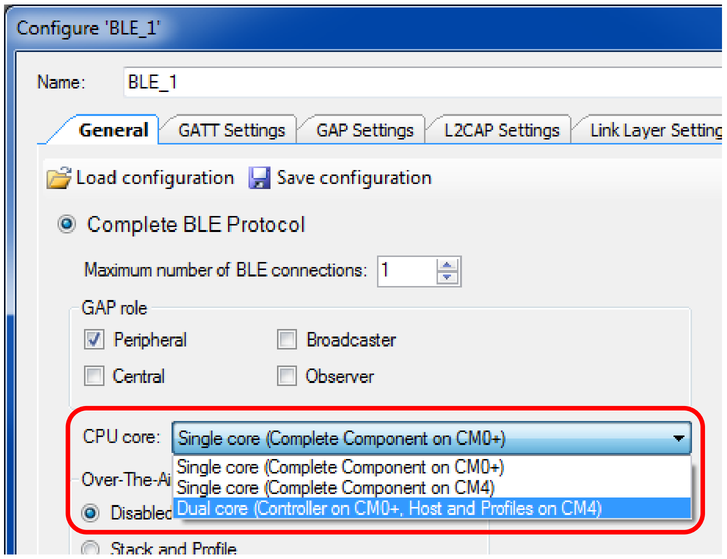
3 PSoC 6 MCU Dual-CPU Architecture

Figure 1 on page 2 shows the overall dual-CPU architecture in PSoC 6 MCU. (For detailed block diagrams of PSoC 6 MCU, see AN210781 and the device datasheet.) Specific features and other details related to dual CPUs are listed in this section. For more information, see the Arm documentation sets for Cortex-M4 and Cortex-M0+, and the PSoC device technical reference manual (TRM).

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

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

Figure 2. BLE Component Configuration for Execution on Multiple CPUs



Name: BLE_1

[✓ Complete BLE Protocol]

Maximum number of BLE connections: 1

GAP role
[✓ Peripheral] [ ] Broadcaster
[ ] Central [ ] Observer

CPU core: Single core (Complete Component on CM0+)
Single core (Complete Component on CM0+)
Single core (Complete Component on CM4)
Dual core (Controller on CM0+, Host and Profiles on CM4)

[✓ Disabled]

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

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

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

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

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

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

- **Interrupts:** Each CPU has its own set of interrupts. A peripheral can route its interrupt output to either or both CPUs. All peripheral interrupt lines are hard-wired to specific CM4 interrupt inputs. Peripheral interrupts are also multiplexed to CM0+'s limited set of 32 interrupt inputs. See Interrupt Assignment Considerations.
- **Power modes:** PSoC 6 MCU has several power modes that can affect either the whole 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 and provides 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 lower system current. This tradeoff lowers the core operating voltage which then requires 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 are in 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 system must be in LP or ULP mode.
  - In CPU sleep mode, the CPU clock is turned off and the CPU halts code execution. The system must be in 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 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. Through the DAP (and subject to device security restrictions), the host can access the device memory and peripherals as well as the registers in both CPUs.

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

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

  PSoC Creator supports debugging a single CPU (either CM4 or CM0+) at a time. For dual-CPU debugging, use a third-party IDE; see Debug Considerations for µVision and IAR. For more information on debugging PSoC devices with PSoC Creator, refer to the PSoC Creator Help.

4 PSoC 6 MCU Dual-CPU Development

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

PSoC Creator project development for a PSoC 6 MCU dual-CPU device is similar to that for any other device supported by PSoC Creator. To create a new project, select **File** > **New** > **Project**. A **Create Project** dialog is displayed, similar to Figure 3.

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

Figure 3. PSoC Creator Create Project Dialog

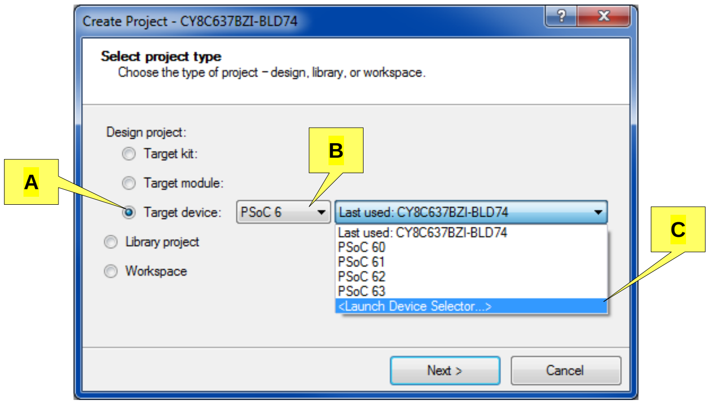
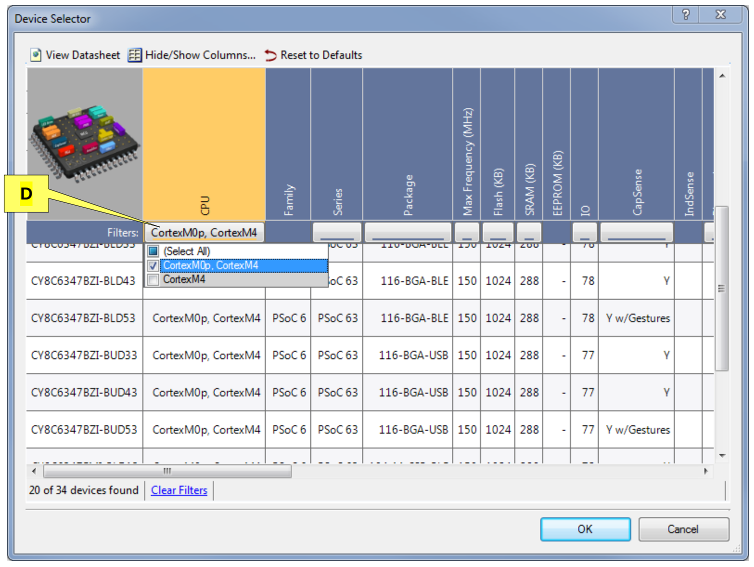

Create Project - CY8C637BZI-BLD74
Select project type
Choose the type of project – design, library, or workspace.
Design project:
Target kit:
Target module:
Target device:
PSoC 6
Last used: CY8C637BZI-BLD74
Last used: CY8C637BZI-BLD74
PSoC 60
PSoC 61
PSoC 62
PSoC 63
<Launch Device Selector...>
Library project
Workspace
Next >
Cancel
A
B
C

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

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

Figure 4. PSoC Creator Device Selector Dialog



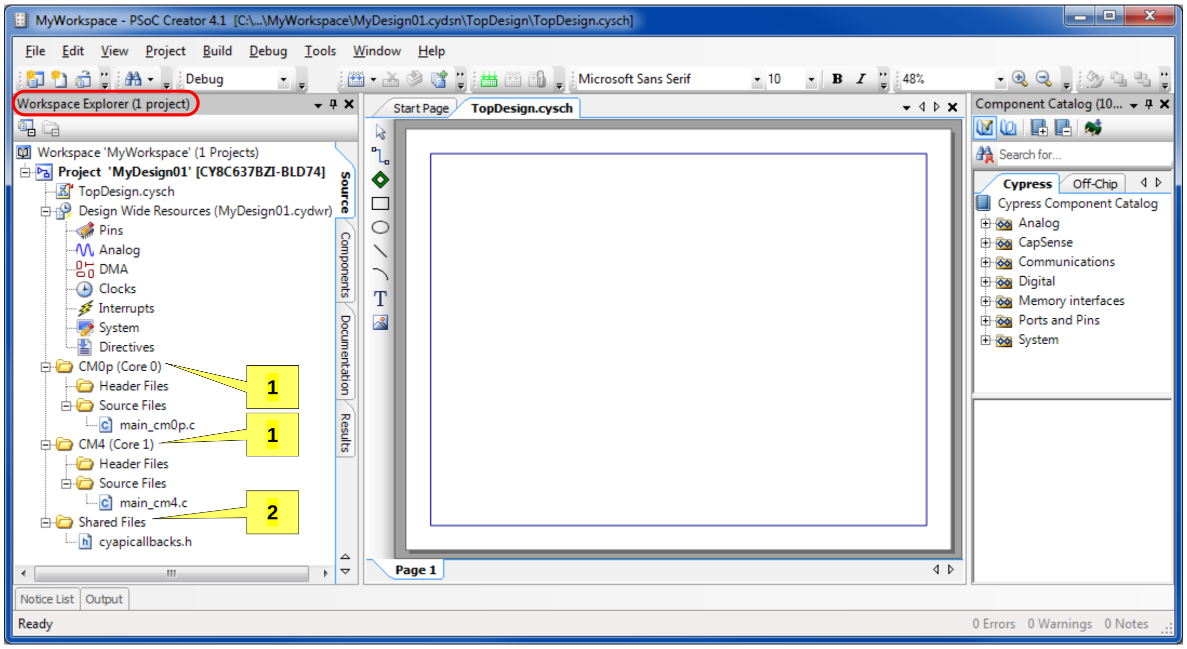
	CPU	Family	Series	Package	Max Frequency (MHz)	Flash (KB)	SRAM (KB)	EEPROM (KB)	IO	CapSense	IndSense
CY8C6347BZI-BLD43			oC 63	116-BGA-BLE	150	1024	288	-	78	Y	
CY8C6347BZI-BLD53	CortexM0p, CortexM4	PSoC 6	PSoC 63	116-BGA-BLE	150	1024	288	-	78	Y w/Gestures	
CY8C6347BZI-BUD33	CortexM0p, CortexM4	PSoC 6	PSoC 63	116-BGA-USB	150	1024	288	-	77	Y	
CY8C6347BZI-BUD43	CortexM0p, CortexM4	PSoC 6	PSoC 63	116-BGA-USB	150	1024	288	-	77	Y	
CY8C6347BZI-BUD53	CortexM0p, CortexM4	PSoC 6	PSoC 63	116-BGA-USB	150	1024	288	-	77	Y w/Gestures	

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

The initial project windows layout (Figure 5) includes a **Workspace Explorer** window with the following features for dual-CPU devices:

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

Figure 5. PSoC Creator Initial Project Layout for Dual-CPU Devices



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

After the project is created, implement your hardware design by dragging Components onto the schematic, and configuring and wiring them.

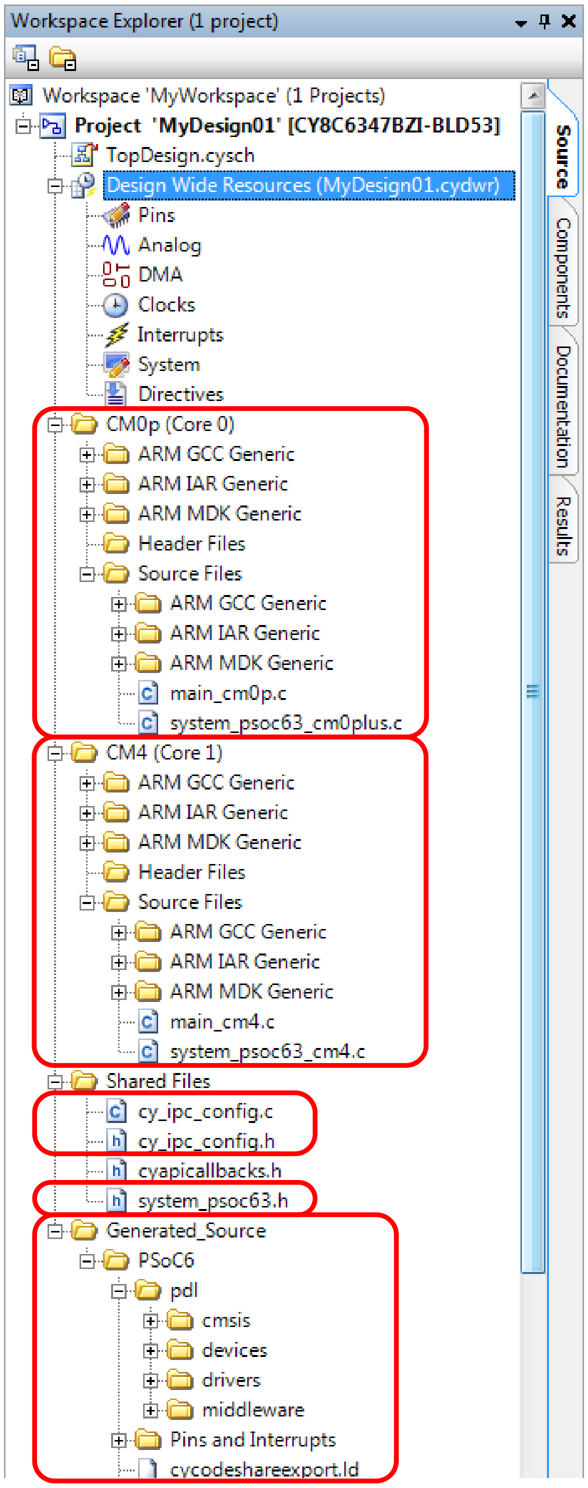
When schematic design entry is complete, select **Build** > **Generate Application**. This creates several system source code files and folders in the existing folders as well as in the new folder *Generated Source*, as Figure 6 shows.

The generated source contains drivers for each Component on the schematic, as well as the Cypress Peripheral Driver Library (PDL). The PDL is a software development kit (SDK) that integrates device header files, startup code, 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 6. Add Generated Source to a Project



```
Workspace 'MyWorkspace' (1 Projects)
  Project 'MyDesign01' [CY8C6347BZI-BLD53]
    TopDesign.cysch
    Design Wide Resources (MyDesign01.cydwr)
      Pins
      Analog
      DMA
      Clocks
      Interrupts
      System
      Directives
    CM0p (Core 0)
      ARM GCC Generic
      ARM IAR Generic
      ARM MDK Generic
      Header Files
      Source Files
        ARM GCC Generic
        ARM IAR Generic
        ARM MDK Generic
        main_cm0p.c
        system_psoc63_cm0plus.c
    CM4 (Core 1)
      ARM GCC Generic
      ARM IAR Generic
      ARM MDK Generic
      Header Files
      Source Files
        ARM GCC Generic
        ARM IAR Generic
        ARM MDK Generic
        main_cm4.c
        system_psoc63_cm4.c
    Shared Files
      cy_ipc_config.c
      cy_ipc_config.h
      cyapicallbacks.h
      system_psoc63.h
    Generated_Source
      PSoC6
        pdl
          cmsis
          devices
          drivers
          middleware
        Pins and Interrupts
        cycodeshareexport.ld
```

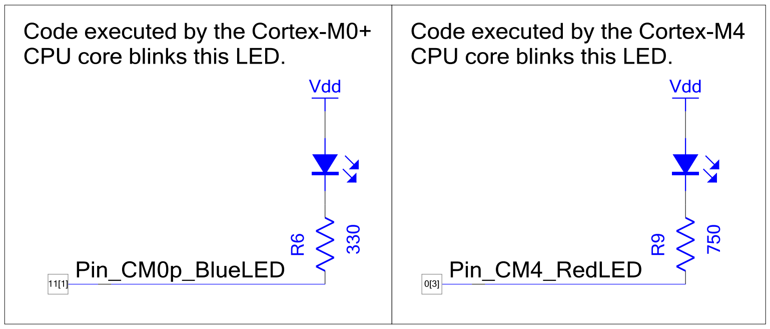
4.1 Resource Assignment Considerations

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

If the same source file is compiled into both binaries, then a function in that file may be executed simultaneously by both CPUs. It is also possible for a Component to be accessed by both CPUs; for example, both CPUs may send data through the same UART. Generally, Component API and PDL driver functions are “CPU-safe”, that is, can be executed simultaneously by both CPUs. However, you should make design decisions about assigning resources to each CPU. There are two ways to do this:

- **Dedicate a resource to one CPU.** A good practice is to indicate on the project schematic the CPU that “owns” the resource, as Figure 7 shows. Include code to use the resource only in the firmware for the desired CPU.

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


Code executed by the Cortex-M0+ CPU core blinks this LED.
Vdd
R6
330
Pin_CM0p_BlueLED
11[1]
Code executed by the Cortex-M4 CPU core blinks this LED.
Vdd
R9
750
Pin_CM4_RedLED
0[3]

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

Flash and SRAM that are allocated in a CPU’s binary 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 Output Pin Components that are mapped to physical pins on the same GPIO port, use only the following functions to change the pin outputs: `GPIO_Write()`, `GPIO_Set()`, `GPIO_Clr()`, and `GPIO_Inv()`. For more information, see the PSoC Creator Pins Component datasheet or the PDL documentation.

4.2 Interrupt Assignment Considerations

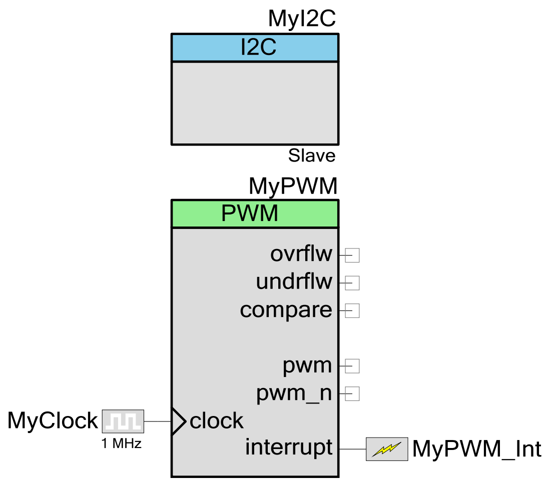
An important consideration for dual-CPU designs is assigning and handling interrupts. As noted previously, all device interrupts are available to CM4, and a subset of interrupts are routed through multiplexers to CM0+. You must decide which CPU will handle each interrupt.

Let us assign interrupts in an example design. Figure 8 shows a design with two interrupts; one from a PWM Component, connected to an Interrupt Component MyPWM_Int; and the other from an I2C Component.

In the **Design Wide Resources** window (file type *.cydwr*), select the **Interrupts** tab to see all of the interrupts in the design, as Figure 9 shows.

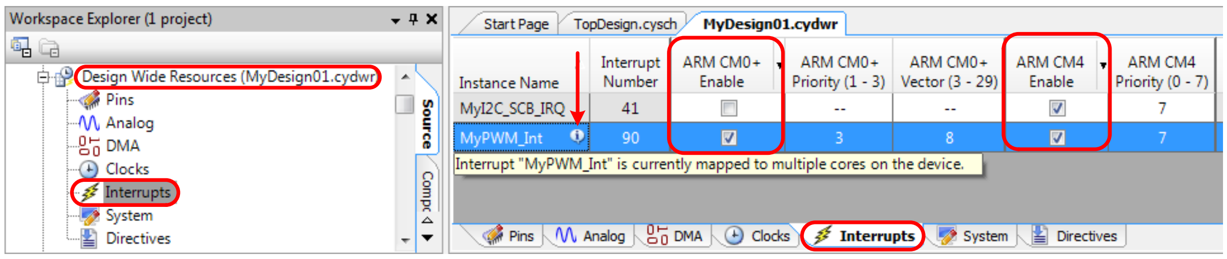
In this example, the I2C Component has an interrupt embedded in it. That interrupt is not shown on the schematic in Figure 8; it is shown in the Design-Wide Resources window as MyI2C_SCB_IRQ.

Figure 8. Example Schematic Design with Two Interrupts



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

Figure 9. Assign Interrupts to the CPUs



Instance Name	Interrupt Number	ARM CM0+ Enable	ARM CM0+ Priority (1 - 3)	ARM CM0+ Vector (3 - 29)	ARM CM4 Enable	ARM CM4 Priority (0 - 7)
MyI2C_SCB_IRQ	41	☐	--	--	☑	7
MyPWM_Int	90	☑	3	8	☑	7

Interrupt "MyPWM_Int" is currently mapped to multiple cores on the device.

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

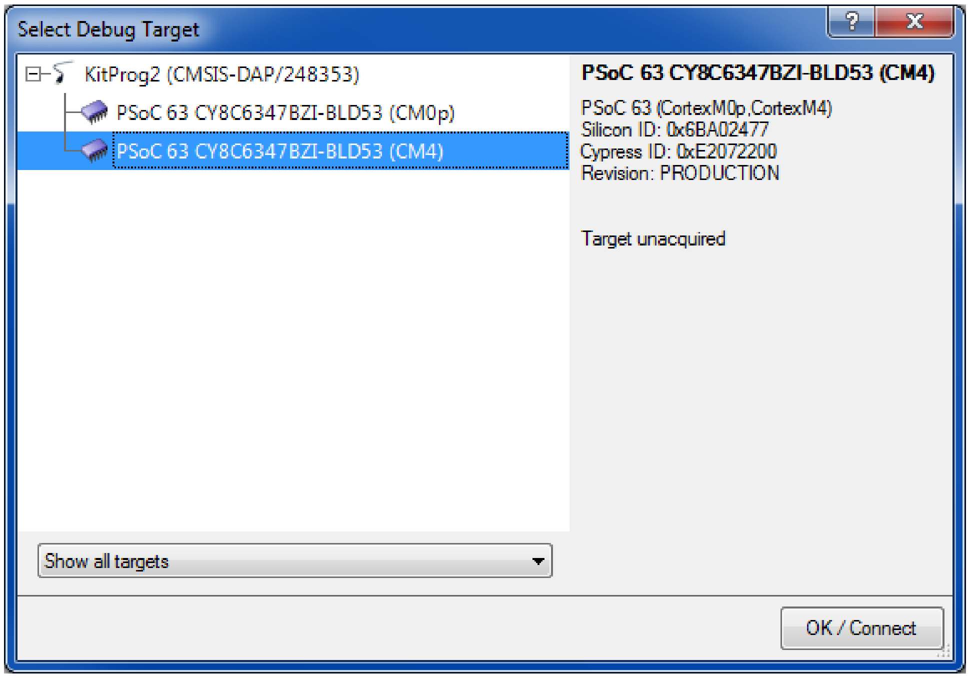
Note: A warning symbol and tooltip are displayed if an interrupt is assigned to both CPUs. This is generally not recommended; however an interrupt can be used to wake up one or both CPUs from their sleep modes.

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

4.3 Debug Considerations

As noted previously, 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 10 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 10. PSoC Creator Select CPU for Debug


Select Debug Target
KitProg2 (CMSIS-DAP/248353)
PSoC 63 CY8C6347BZI-BLD53 (CM0p)
PSoC 63 CY8C6347BZI-BLD53 (CM4)
Show all targets
OK / Connect

PSoC 63 CY8C6347BZI-BLD53 (CM4)
PSoC 63 (CortexM0p,CortexM4)
Silicon ID: 0x6BA02477
Cypress ID: 0xE2072200
Revision: PRODUCTION

Target unacquired

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.

4.3.1 Configure PSoC Creator Project

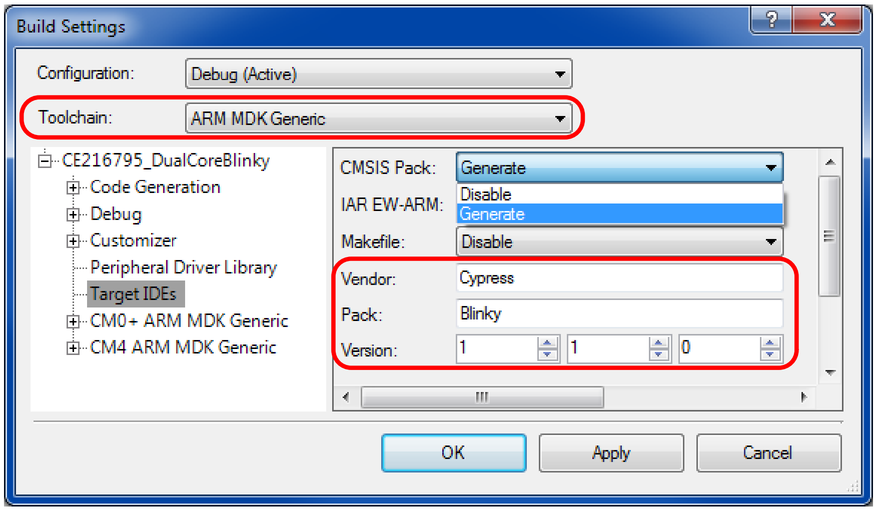
Update the **Target IDEs** settings in the project Build Settings, as Figure 11 shows.

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

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

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

Figure 11. Build Settings for Target IDEs



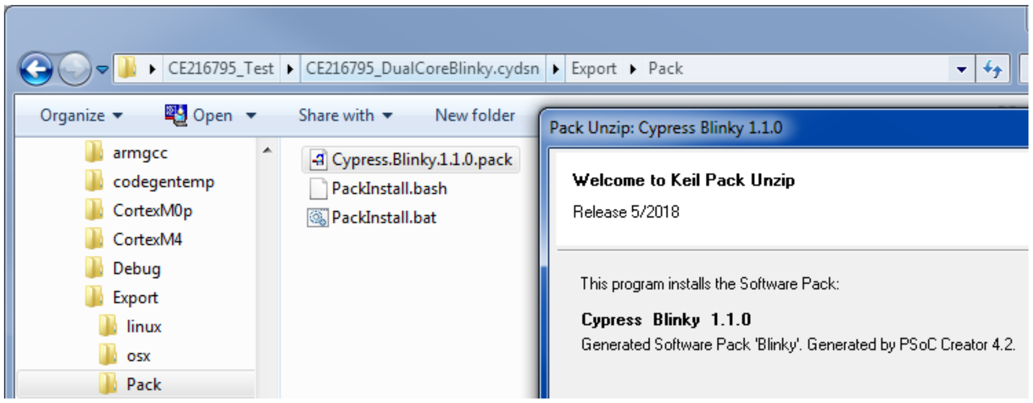
Configuration: Debug (Active)
Toolchain: ARM MDK Generic
CMSIS Pack: Generate
IAR EW-ARM:
Makefile: Disable
Vendor: Cypress
Pack: Blinky
Version: 1 1 0

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 12 shows.

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

Figure 12. Install µVision Pack from PSoC Creator Project



Pack Unzip: Cypress Blinky 1.1.0

Welcome to Keil Pack Unzip

Release 5/2018

This program installs the Software Pack:

Cypress Blinky 1.1.0

Generated Software Pack 'Blinky'. Generated by PSoC Creator 4.2.

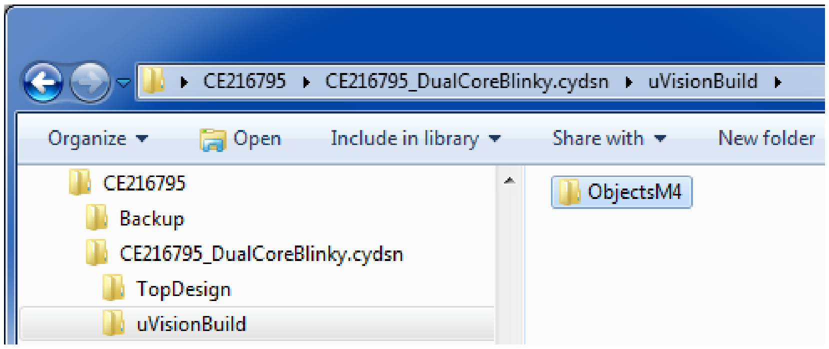
Note: if you update the PSoC Creator project, consider changing the µVision pack version number (see Figure 11) and installing the new pack.

4.3.2 Create µVision Projects

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

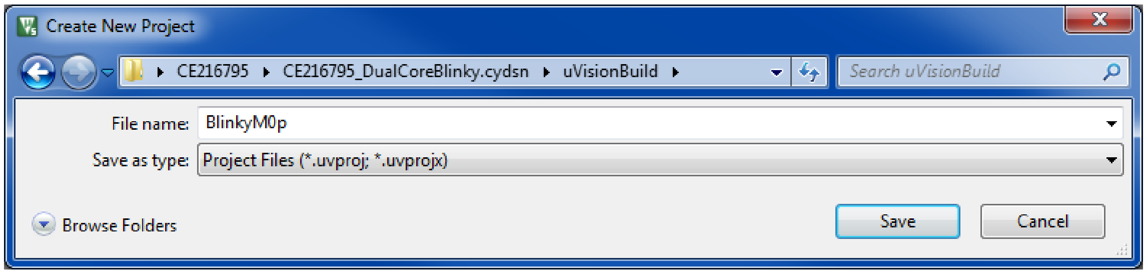
Recommended: create a new folder (e.g. *uVisionBuild*) within your PSoC Creator *<project>.cydsn* folder to store all µVision project files separately from the PSoC Creator files (this is different from the IAR instructions). Within that folder, create another new folder for CM4 object files (e.g., *ObjectsM4*), as Figure 13 shows:

Figure 13. 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 14 shows:

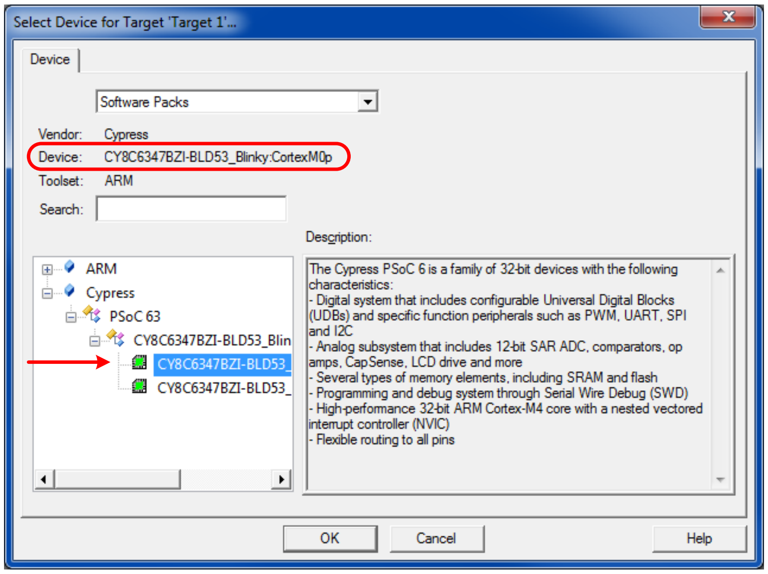
Figure 14. Create a µVision Project for CM0+



File name: BlinkyM0p
Save as type: Project Files (*.uvproj; *.uvprojx)

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 12) are displayed. Select the CM0+ CPU, as Figure 15 shows. Click **OK**.

Figure 15. Select CM0+ as the Project Device

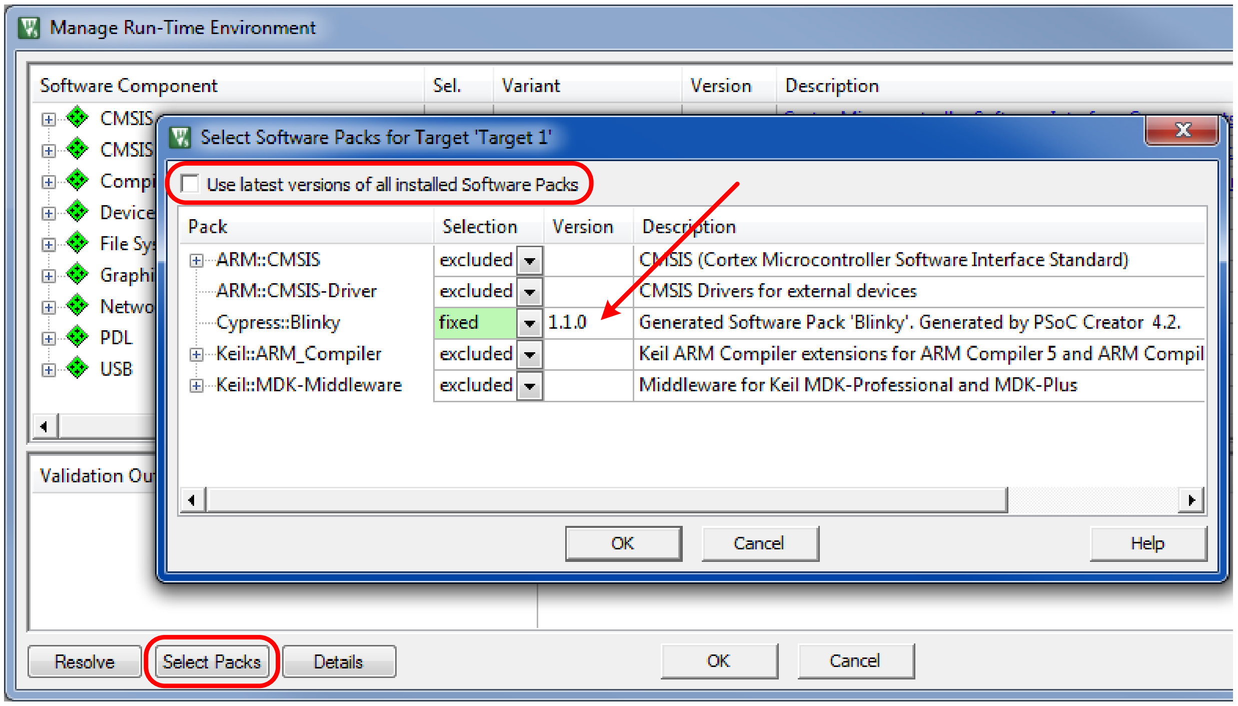


Vendor: Cypress
Device: CY8C6347BZI-BLD53_Blinky:CortexM0p
Toolset: ARM
Search:

Description:
The Cypress PSoC 6 is a family of 32-bit devices with the following characteristics:
- Digital system that includes configurable Universal Digital Blocks (UDBs) and specific function peripherals such as PWM, UART, SPI and I2C
- Analog subsystem that includes 12-bit SAR ADC, comparators, op amps, CapSense, LCD drive and more
- Several types of memory elements, including SRAM and flash
- Programming and debug system through Serial Wire Debug (SWD)
- High-performance 32-bit ARM Cortex-M4 core with a nested vectored interrupt controller (NVIC)
- Flexible routing to all pins

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 16 shows:

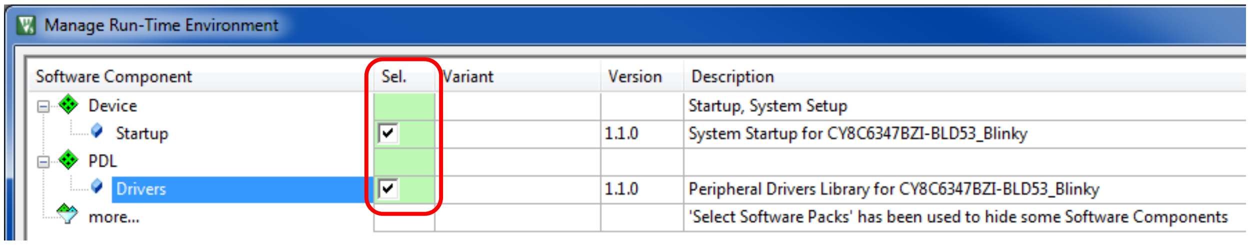
Figure 16. Select the PSoC Creator Project Pack



Pack	Selection	Version	Description
ARM::CMSIS	excluded		CMSIS (Cortex Microcontroller Software Interface Standard)
ARM::CMSIS-Driver	excluded		CMSIS Drivers for external devices
Cypress::Blinky	fixed	1.1.0	Generated Software Pack 'Blinky'. Generated by PSoC Creator 4.2.
Keil::ARM_Compiler	excluded		Keil ARM Compiler extensions for ARM Compiler 5 and ARM Compil
Keil::MDK-Middleware	excluded		Middleware for Keil MDK-Professional and MDK-Plus

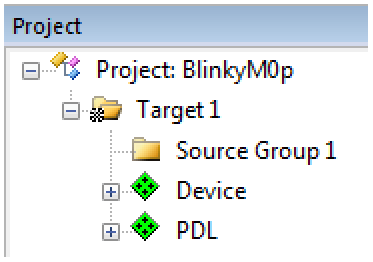
Click **OK**; the Manage Run-Time Environment dialog changes as Figure 17 shows. Select the Device Startup and PDL Drivers, and click **OK**. The project is created, with a Target 1, a Source Group 1, and Device startup and PDL files, as Figure 18 shows.

Figure 17. Select Pack Startup and PDL Driver Files



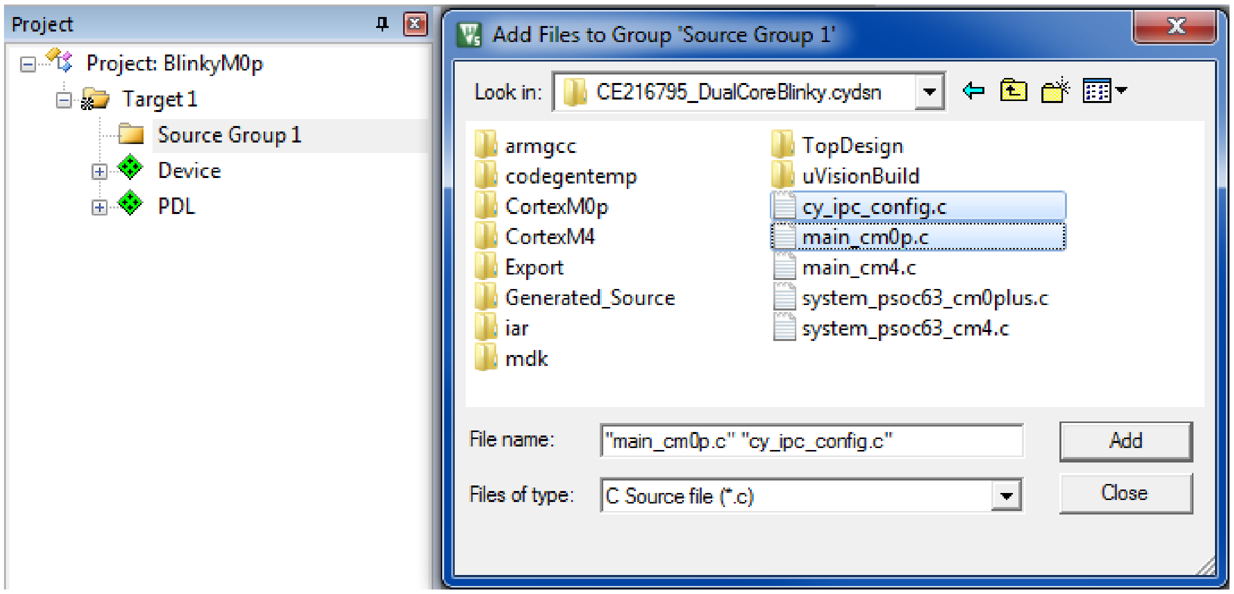
Software Component	Sel.	Variant	Version	Description
Device				Startup, System Setup
Startup	✓		1.1.0	System Startup for CY8C6347BZI-BLD53_Blinky
PDL				
Drivers	✓		1.1.0	Peripheral Drivers Library for CY8C6347BZI-BLD53_Blinky
more...				'Select Software Packs' has been used to hide some Software Components

Figure 18. 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 19 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 19. Add PSoC Creator Project C Source Files to the Source Group

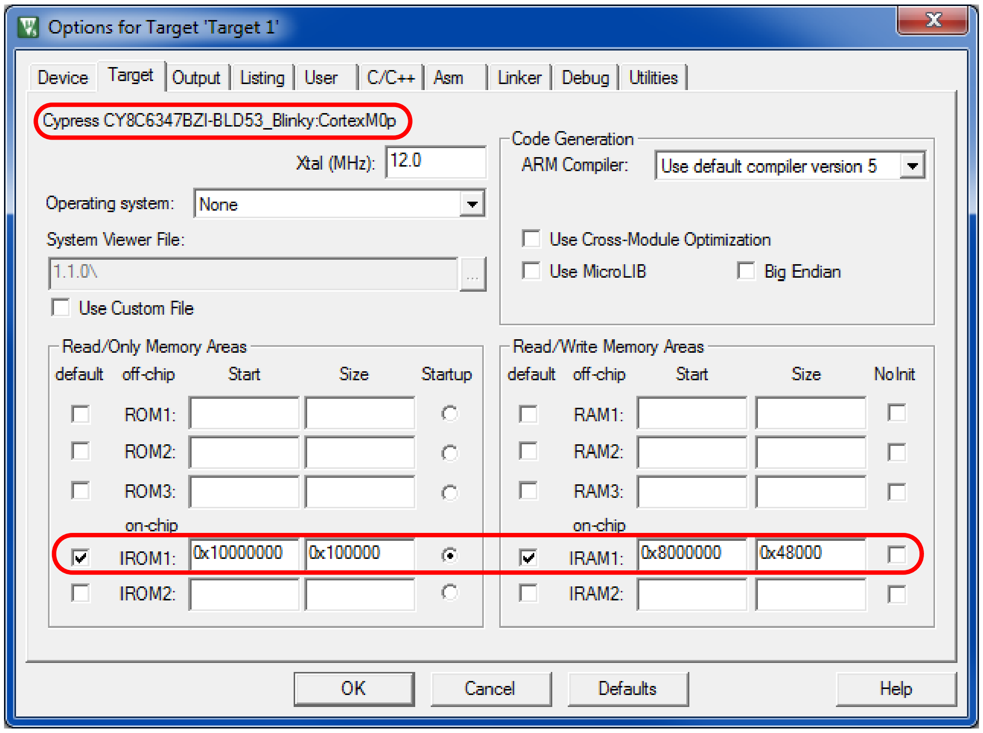


File name: "main_cm0p.c" "cy_ipc_config.c"

Files of type: C Source file (*.c)

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 20 shows. Updating other fields such as Xtal (MHz) and Operating system is optional.

Figure 20. Project Target Options



Cypress CY8C6347BZI-BLD53_Blinky:CortexM0p

Xtal (MHz): 12.0

Operating system: None

System Viewer File: 1.1.0\

Use Custom File: ☐

Code Generation

ARM Compiler: Use default compiler version 5

☐ Use Cross-Module Optimization

☐ Use MicroLIB ☐ Big Endian

Read/Only Memory Areas

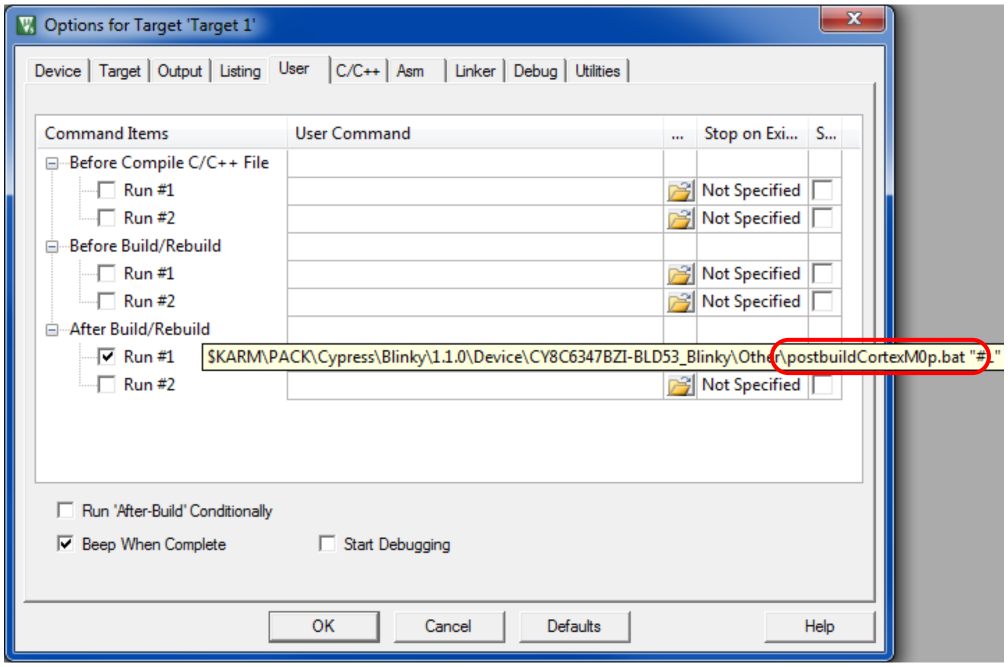
default	off-chip	Start	Size	Startup
☐	ROM1:			○
☐	ROM2:			○
☐	ROM3:			○
	on-chip			
✓	IROM1:	0x10000000	0x100000	●
☐	IROM2:			○

Read/Write Memory Areas

default	off-chip	Start	Size	NoInit
☐	RAM1:			☐
☐	RAM2:			☐
☐	RAM3:			☐
	on-chip			
✓	IRAM1:	0x8000000	0x48000	☐
☐	IRAM2:			☐

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 21 shows. Add other pre- and post-build batch files, and select other options, as needed.

Figure 21. Project User Options



Options for Target 'Target 1'

Device | Target | Output | Listing | User | C/C++ | Asm | Linker | Debug | Utilities

Command Items	User Command	...	Stop on Exi...	S...
Before Compile C/C++ File				
☐ Run #1			Not Specified	☐
☐ Run #2			Not Specified	☐
Before Build/Rebuild				
☐ Run #1			Not Specified	☐
☐ Run #2			Not Specified	☐
After Build/Rebuild				
☑ Run #1	$KARM\PACK\Cypress\Blinky\1.1.0\Device\CY8C6347BZI-BLD53_Blinky\Other\postbuildCortexM0p.bat "#L"			
☐ Run #2			Not Specified	☐

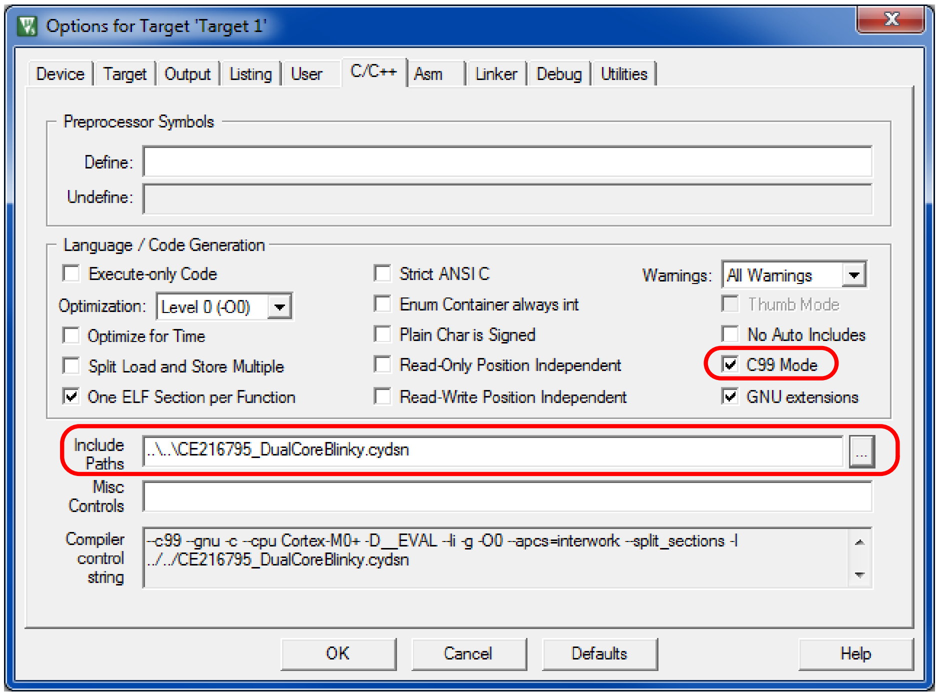
☐ Run 'After-Build' Conditionally

☑ Beep When Complete ☐ Start Debugging

OK | Cancel | Defaults | Help

Confirm in the **C/C++** tab that the **C99 mode** option is checked, as Figure 22 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 22. Project C/C++ Options



Options for Target 'Target 1'

Device | Target | Output | Listing | User | C/C++ | Asm | Linker | Debug | Utilities

Preprocessor Symbols

Define:

Undefine:

Language / Code Generation

☐ Execute-only Code ☐ Strict ANSI C Warnings: All Warnings

Optimization: Level 0 (-O0) ☐ Enum Container always int ☐ Thumb Mode

☐ Optimize for Time ☐ Plain Char is Signed ☐ No Auto Includes

☐ Split Load and Store Multiple ☐ Read-Only Position Independent ☑ C99 Mode

☑ One ELF Section per Function ☐ Read-Write Position Independent ☑ GNU extensions

Include Paths: ..\..\CE216795_DualCoreBlinky.cydsn

Misc Controls:

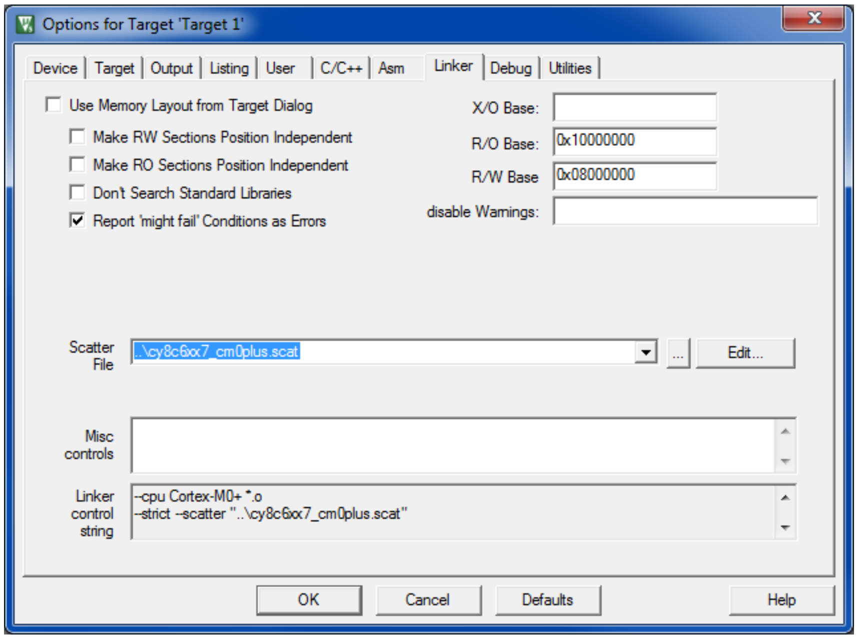
Compiler control string:

```
--c99 --gnu -c --cpu Cortex-M0+ -D__EVAL -li -g -O0 --apcs=interwork --split_sections -I
../../CE216795_DualCoreBlinky.cydsn
```

OK | Cancel | Defaults | Help

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

Figure 23. Project Linker Options


Options for Target 'Target 1'
Device | Target | Output | Listing | User | C/C++ | Asm | Linker | Debug | Utilities
OK | Cancel | Defaults | Help
Edit...

☐ Use Memory Layout from Target Dialog
☐ Make RW Sections Position Independent
☐ Make RO Sections Position Independent
☐ Don't Search Standard Libraries
☑ Report 'might fail' Conditions as Errors

X/O Base:
R/O Base: 0x10000000
R/W Base 0x08000000
disable Warnings:

Scatter File: ..\cy8c6xx7_cm0plus.scat

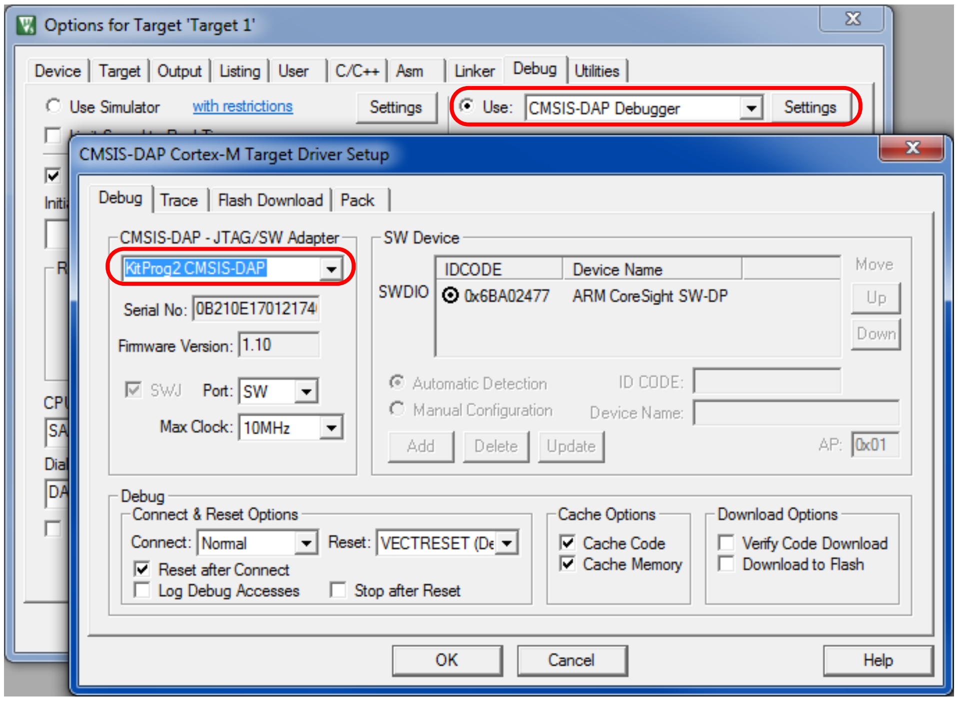
Misc controls:

Linker control string:
```
--cpu Cortex-M0+ *.o
--strict --scatter "..\cy8c6xx7_cm0plus.scat"
```

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 24 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 24. Project Debug Options


Options for Target 'Target 1'
Device | Target | Output | Listing | User | C/C++ | Asm | Linker | Debug | Utilities
Use Simulator
with restrictions
Settings
CMSIS-DAP Cortex-M Target Driver Setup
Debug | Trace | Flash Download | Pack
Move
Up
Down
Add
Delete
Update
OK
Cancel
Help

Use: CMSIS-DAP Debugger

CMSIS-DAP - JTAG/SW Adapter: KitProg2 CMSIS-DAP
Serial No: 0B210E17012174
Firmware Version: 1.10
☑ SWJ Port: SW
Max Clock: 10MHz

SW Device

	IDCODE	Device Name
SWDIO	0x6BA02477	ARM CoreSight SW-DP

Automatic Detection ID CODE:
Manual Configuration Device Name:
AP: 0x01

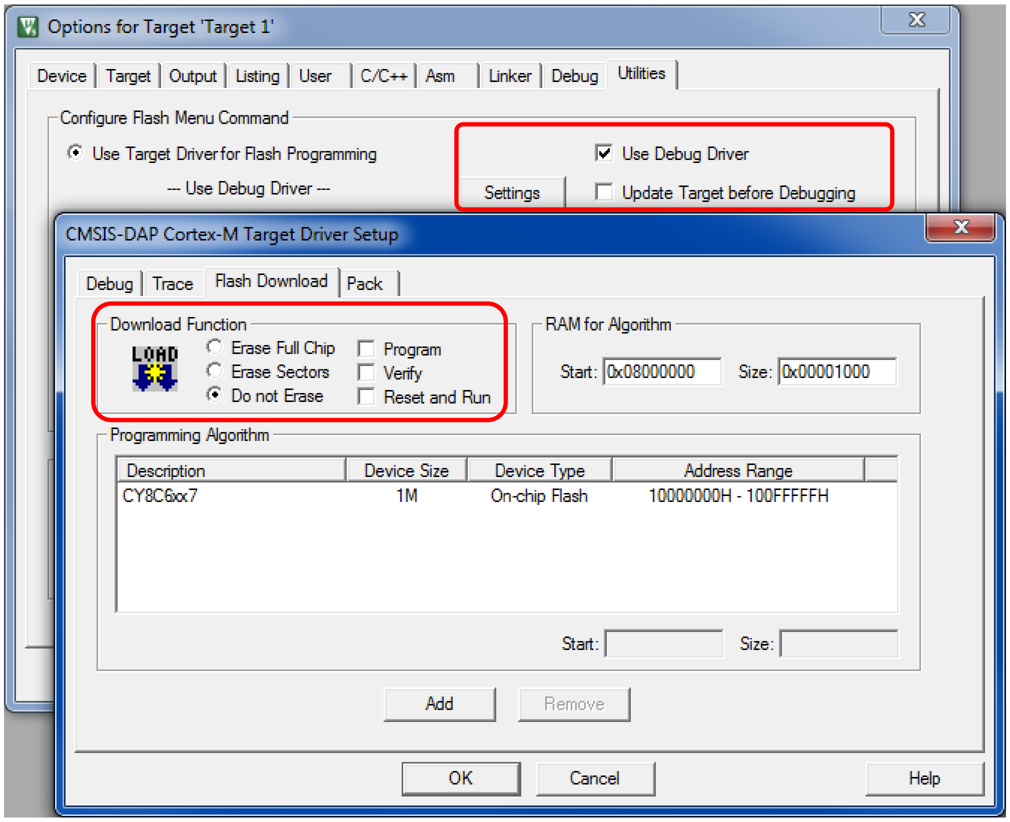
Debug
Connect & Reset Options
Connect: Normal Reset: VECTRESET (De
☑ Reset after Connect
☐ Log Debug Accesses ☐ Stop after Reset

Cache Options
☑ Cache Code
☑ Cache Memory

Download Options
☐ Verify Code Download
☐ Download to Flash

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 25 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 25. Project Utilities Options


Options for Target 'Target 1'

Device | Target | Output | Listing | User | C/C++ | Asm | Linker | Debug | Utilities

Configure Flash Menu Command

(●) Use Target Driver for Flash Programming

--- Use Debug Driver ---

Settings

☑ Use Debug Driver

☐ Update Target before Debugging

CMSIS-DAP Cortex-M Target Driver Setup

Debug | Trace | Flash Download | Pack

Download Function

○ Erase Full Chip ☐ Program

○ Erase Sectors ☐ Verify

● Do not Erase ☐ Reset and Run

RAM for Algorithm

Start: 0x08000000 Size: 0x00001000

Programming Algorithm

Description	Device Size	Device Type	Address Range
CY8C6xx7	1M	On-chip Flash	10000000H - 100FFFFFH

Start: Size:

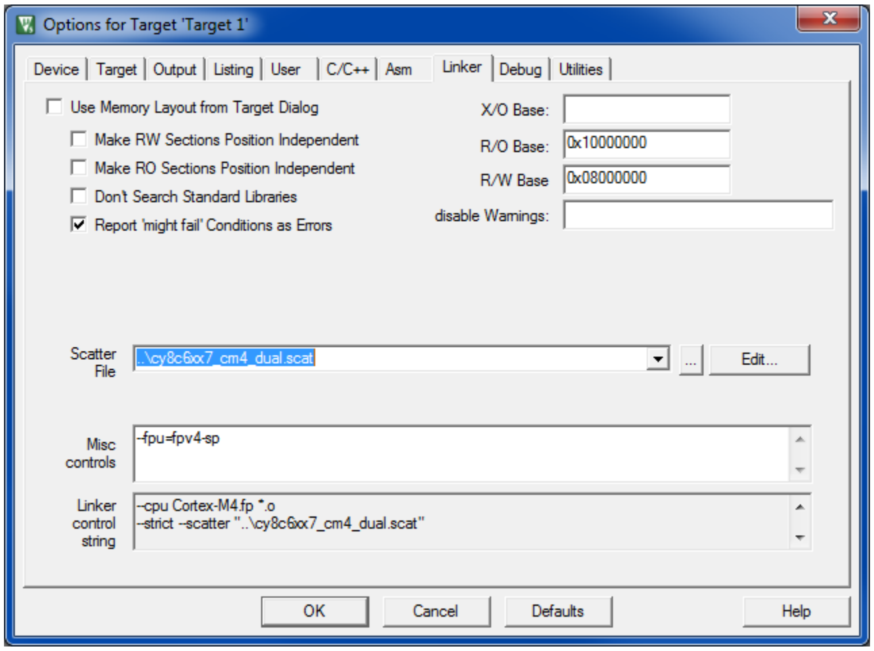
Repeat the previous steps and create a second project for CM4.

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 14. 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 14.
- Select the CM4 CPU from the previously installed pack; see Figure 15.
- 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 19 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 13.
- In the **Options** dialog, **C/C++** tab, add `--fpu=fpv4-sp` to **Misc Controls**; see Figure 22.

- In the **Options** dialog, **Linker** tab, select the "cm4_dual" scatter file, as Figure 26 shows. The CM4 project will contain code for both CPUs. Add `--fpu=fpv4-sp` to **Misc Controls**.

Figure 26. Linker Options for CM4 Project


Options for Target 'Target 1'
Device | Target | Output | Listing | User | C/C++ | Asm | Linker | Debug | Utilities
OK
Cancel
Defaults
Help
Edit...

- [ ] Use Memory Layout from Target Dialog
- [ ] Make RW Sections Position Independent
- [ ] Make RO Sections Position Independent
- [ ] Don't Search Standard Libraries
- [✓] Report 'might fail' Conditions as Errors

X/O Base:
R/O Base: 0x10000000
R/W Base: 0x08000000
disable Warnings:

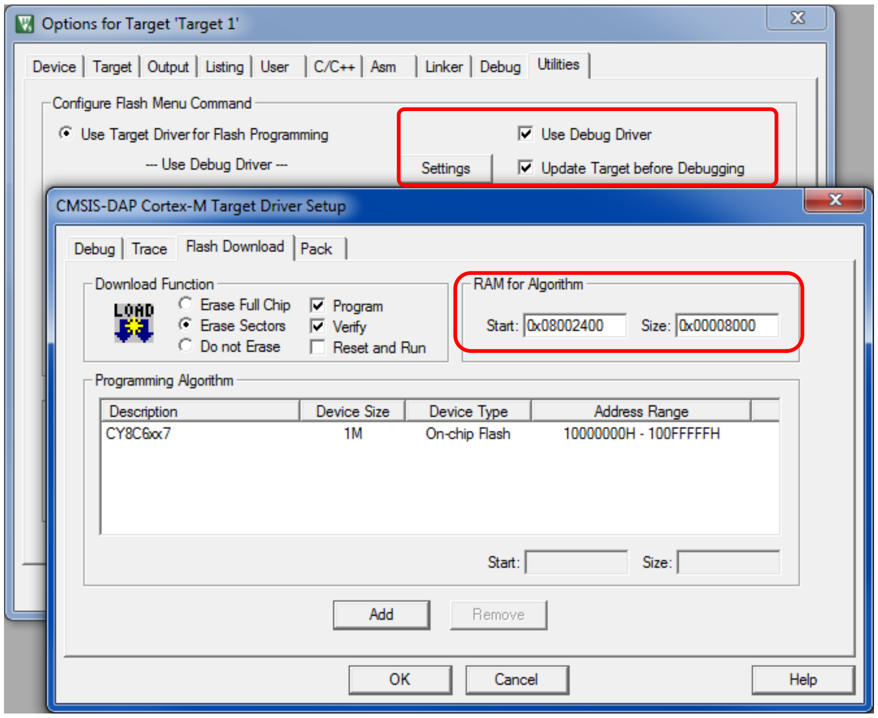
Scatter File: ..\cy8c6xx7_cm4_dual.scat

Misc controls: --fpu=fpv4-sp

Linker control string:
```
--cpu Cortex-M4.fp *.o
--strict --scatter "..\cy8c6xx7_cm4_dual.scat"
```

- In the **Options** dialog, **Debug** tab, Target Driver Setup, select VECTRESET for the **Reset** option; see Figure 24.
- In the **Options** dialog, **Utilities** tab, confirm that **Update Target before Debugging** is checked, as Figure 27 shows. Set the RAM for Algorithm values as indicated. Checking **Reset and Run** is optional but convenient.

Figure 27. Utilities Options for CM4 Project


Options for Target 'Target 1'
Device | Target | Output | Listing | User | C/C++ | Asm | Linker | Debug | Utilities
Settings
CMSIS-DAP Cortex-M Target Driver Setup
Debug | Trace | Flash Download | Pack
LOAD
Add
Remove
OK
Cancel
Help

Configure Flash Menu Command
- (●) Use Target Driver for Flash Programming
- --- Use Debug Driver ---
- [✓] Use Debug Driver
- [✓] Update Target before Debugging

Download Function
- ( ) Erase Full Chip
- (●) Erase Sectors
- ( ) Do not Erase
- [✓] Program
- [✓] Verify
- [ ] Reset and Run

RAM for Algorithm
Start: 0x08002400 Size: 0x00008000

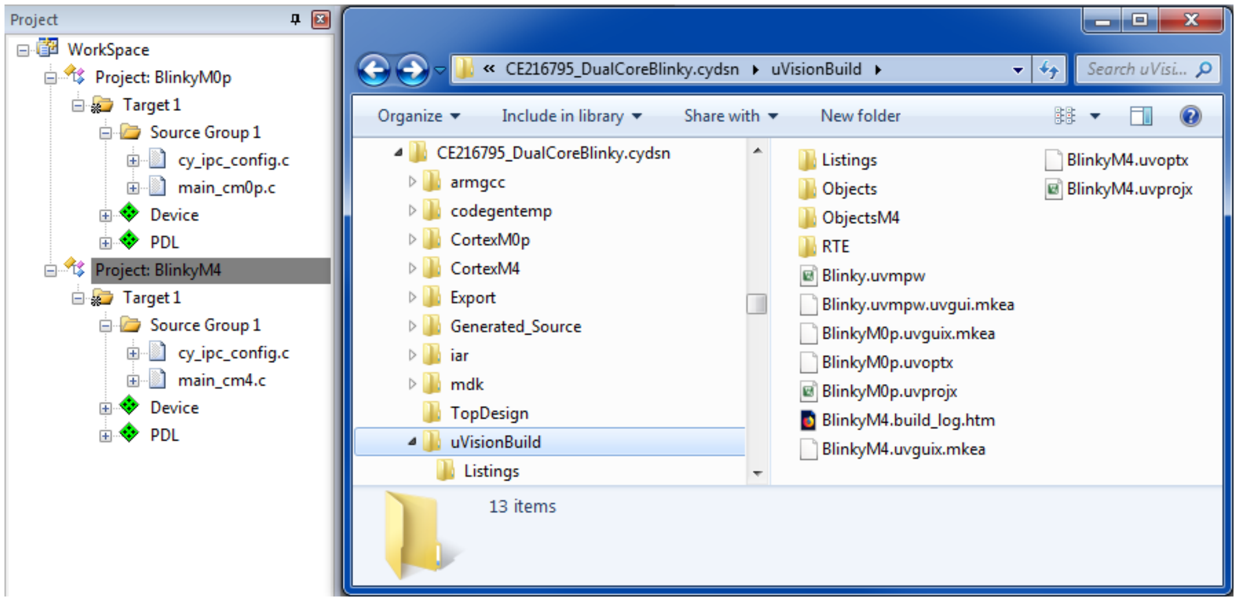
Programming Algorithm

Description	Device Size	Device Type	Address Range
CY8C6xx7	1M	On-chip Flash	10000000H - 100FFFFFH

Start: Size:

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 28.

Figure 28. Resultant µVision Project Window and Project Files



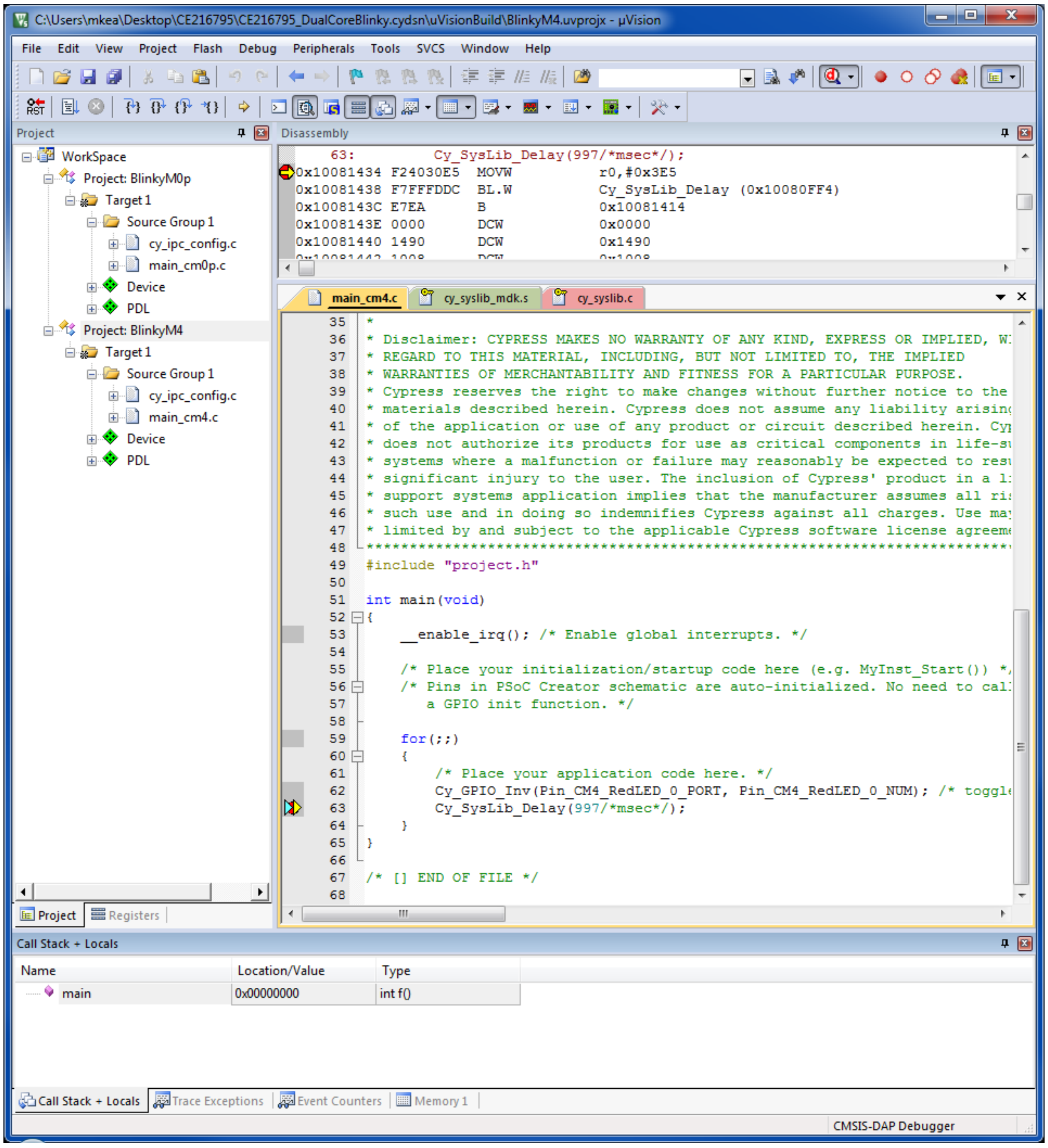
Build the projects in sequence; build the CM0+ project first. Note that µVision has a batch build feature to automate the process. After building is successfully completed, right-click the BlinkyM4 project and set it as the active project. Then test your build options by (1) erasing flash (**Flash** > **Erase**), and (2) downloading the project (**Flash** > **Download**) and confirming correct operation. If you did not select Reset and Run (see Figure 27), 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.

4.3.3 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 29:

Figure 29. CM4 Debug Window



```
    63:         Cy_SysLib_Delay(997/*msec*/);
0x10081434 F24030E5  MOVW          r0,#0x3E5
0x10081438 F7FFFDDC  BL.W          Cy_SysLib_Delay (0x10080FF4)
0x1008143C E7EA      B             0x10081414
0x1008143E 0000      DCW           0x0000
0x10081440 1490      DCW           0x1490
```

```
35  *
36  * Disclaimer: CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, W
37  * REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
38  * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.
39  * Cypress reserves the right to make changes without further notice to the
40  * materials described herein. Cypress does not assume any liability arisin
41  * of the application or use of any product or circuit described herein. Cy
42  * does not authorize its products for use as critical components in life-s
43  * systems where a malfunction or failure may reasonably be expected to res
44  * significant injury to the user. The inclusion of Cypress' product in a l
45  * support systems application implies that the manufacturer assumes all ri
46  * such use and in doing so indemnifies Cypress against all charges. Use ma
47  * limited by and subject to the applicable Cypress software license agreem
48  ***********************************************************************
49  #include "project.h"
50
51  int main(void)
52  {
53      __enable_irq(); /* Enable global interrupts. */
54
55      /* Place your initialization/startup code here (e.g. MyInst_Start()) *
56      /* Pins in PSoC Creator schematic are auto-initialized. No need to cal
57         a GPIO init function. */
58
59      for(;;)
60      {
61          /* Place your application code here. */
62          Cy_GPIO_Inv(Pin_CM4_RedLED_0_PORT, Pin_CM4_RedLED_0_NUM); /* toggl
63          Cy_SysLib_Delay(997/*msec*/);
64      }
65  }
66
67  /* [] END OF FILE */
68
```

Name	Location/Value	Type
main	0x00000000	int f()

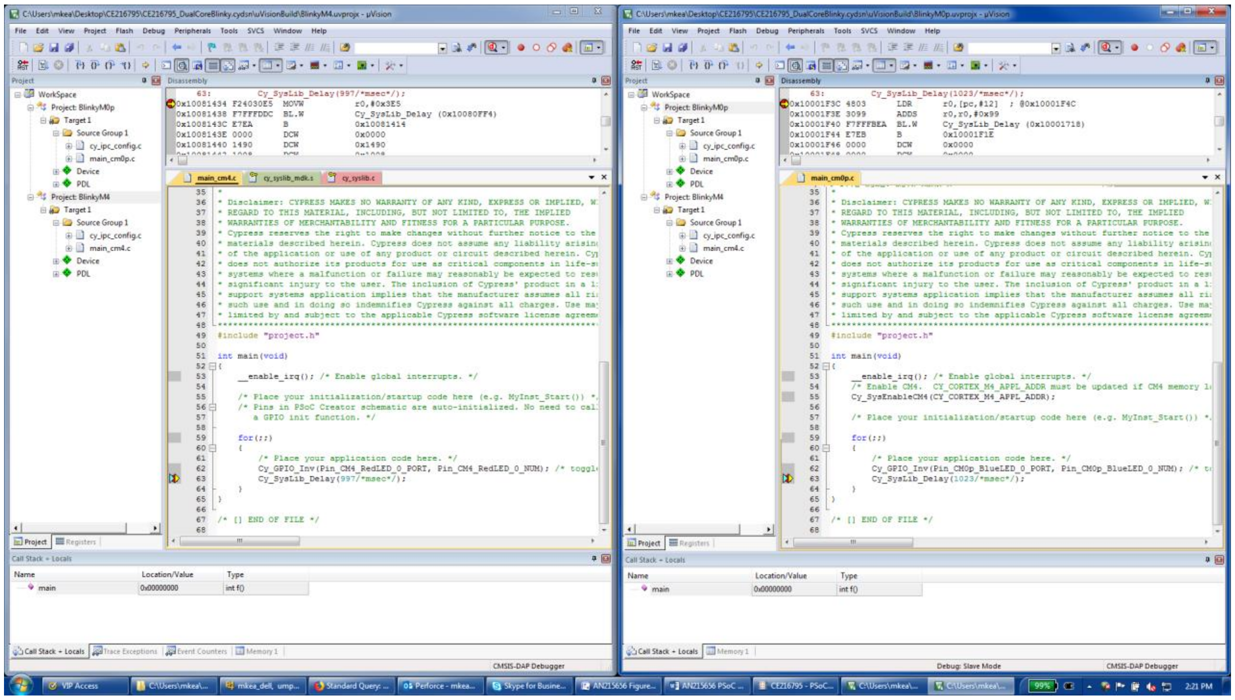
If you are running the CE216795 dual-CPU blinky project, set a breakpoint at line 63, `Cy_Syslib_Delay()`. Then repeatedly click **Debug** > **Run**, and the red LED toggles on each stop at the breakpoint.

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 30. 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 30. µVision Dual-CPU Debugging



C:\Users\mkea\Desktop\CE216795\CE216795_DualCoreBlinky.cydsn\uVisionBuild\BlinkyM4.uvprojx - µVision

```
    63:         Cy_SysLib_Delay(997/*msec*/);
0x10081434 F24030E5  MOVW         r0,#0x3E5
0x10081438 F7FFFDDC  BL.W         Cy_SysLib_Delay (0x10080FF4)
0x1008143C E7EA      B            0x10081414
0x1008143E 0000      DCW          0x0000
0x10081440 1490      DCW          0x1490
```

main_cm4.c

```
35  *
36  * Disclaimer: CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, W
37  * REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
38  * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.
39  * Cypress reserves the right to make changes without further notice to the
40  * materials described herein. Cypress does not assume any liability arisin
41  * of the application or use of any product or circuit described herein. Cy
42  * does not authorize its products for use as critical components in life-s
43  * systems where a malfunction or failure may reasonably be expected to res
44  * significant injury to the user. The inclusion of Cypress' product in a l
45  * support systems application implies that the manufacturer assumes all ri
46  * such use and in doing so indemnifies Cypress against all charges. Use ma
47  * limited by and subject to the applicable Cypress software license agreem
48  *************************************************************************
49  #include "project.h"
50
51  int main(void)
52  {
53      __enable_irq(); /* Enable global interrupts. */
54
55      /* Place your initialization/startup code here (e.g. MyInst_Start()) *
56      /* Pins in PSoC Creator schematic are auto-initialized. No need to cal
57         a GPIO init function. */
58
59      for(;;)
60      {
61          /* Place your application code here. */
62          Cy_GPIO_Inv(Pin_CM4_RedLED_0_PORT, Pin_CM4_RedLED_0_NUM); /* toggl
63          Cy_SysLib_Delay(997/*msec*/);
64      }
65  }
66
67  /* [] END OF FILE */
68
```

Name	Location/Value	Type
main	0x00000000	int f()

C:\Users\mkea\Desktop\CE216795\CE216795_DualCoreBlinky.cydsn\uVisionBuild\BlinkyM0p.uvprojx - µVision

```
    63:         Cy_SysLib_Delay(1023/*msec*/);
0x10001F3C 4803      LDR      r0,[pc,#12]  ; @0x10001F4C
0x10001F3E 3099      ADDS     r0,r0,#0x99
0x10001F40 F7FFFBEA  BL.W     Cy_SysLib_Delay (0x10001718)
0x10001F44 E7EB      B        0x10001F1E
0x10001F46 0000      DCW      0x0000
```

main_cm0p.c

```
35  *
36  * Disclaimer: CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, W
37  * REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
38  * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.
39  * Cypress reserves the right to make changes without further notice to the
40  * materials described herein. Cypress does not assume any liability arisin
41  * of the application or use of any product or circuit described herein. Cy
42  * does not authorize its products for use as critical components in life-s
43  * systems where a malfunction or failure may reasonably be expected to res
44  * significant injury to the user. The inclusion of Cypress' product in a l
45  * support systems application implies that the manufacturer assumes all ri
46  * such use and in doing so indemnifies Cypress against all charges. Use ma
47  * limited by and subject to the applicable Cypress software license agreem
48  *************************************************************************
49  #include "project.h"
50
51  int main(void)
52  {
53      __enable_irq(); /* Enable global interrupts. */
54      /* Enable CM4.  CY_CORTEX_M4_APPL_ADDR must be updated if CM4 memory l
55      Cy_SysEnableCM4(CY_CORTEX_M4_APPL_ADDR);
56
57      /* Place your initialization/startup code here (e.g. MyInst_Start()) *
58
59      for(;;)
60      {
61          /* Place your application code here. */
62          Cy_GPIO_Inv(Pin_CM0p_BlueLED_0_PORT, Pin_CM0p_BlueLED_0_NUM); /* t
63          Cy_SysLib_Delay(1023/*msec*/);
64      }
65  }
66
67  /* [] END OF FILE */
68
```

Name	Location/Value	Type
main	0x00000000	int f()

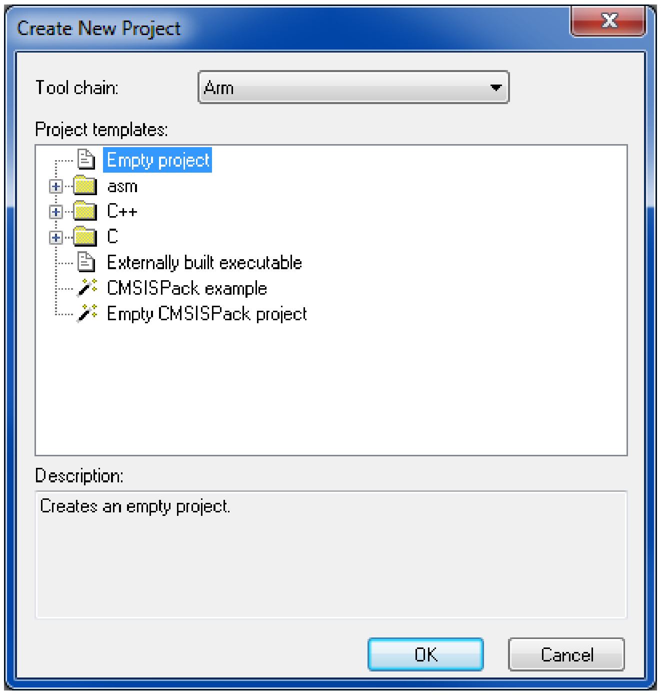
4.3.4 Create IAR-EW Projects

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.

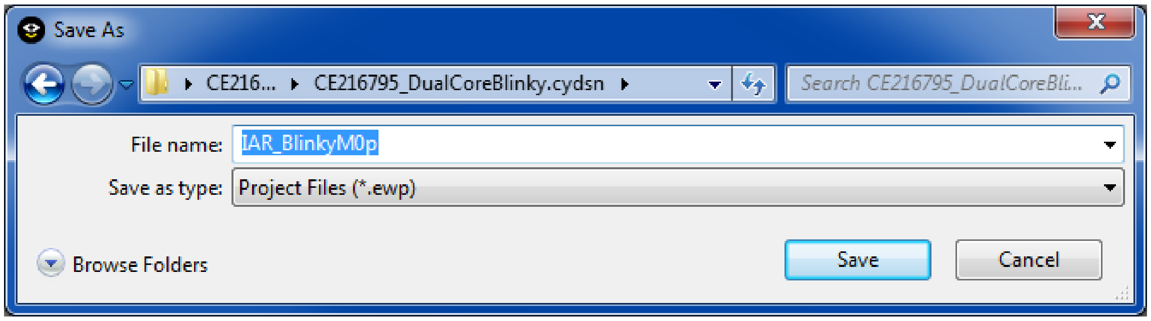
Open IAR Embedded Workbench for ARM 8.22 or later, and create a new project (**Project** > **Create New Project...**). In the Create New Project dialog (Figure 31), confirm that the Tool chain is **Arm**, select the **Empty project** template, then click **OK**.

Figure 31. IAR Embedded Workbench Create New Project Dialog



Recommended: in the Save As dialog (Figure 32), 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.

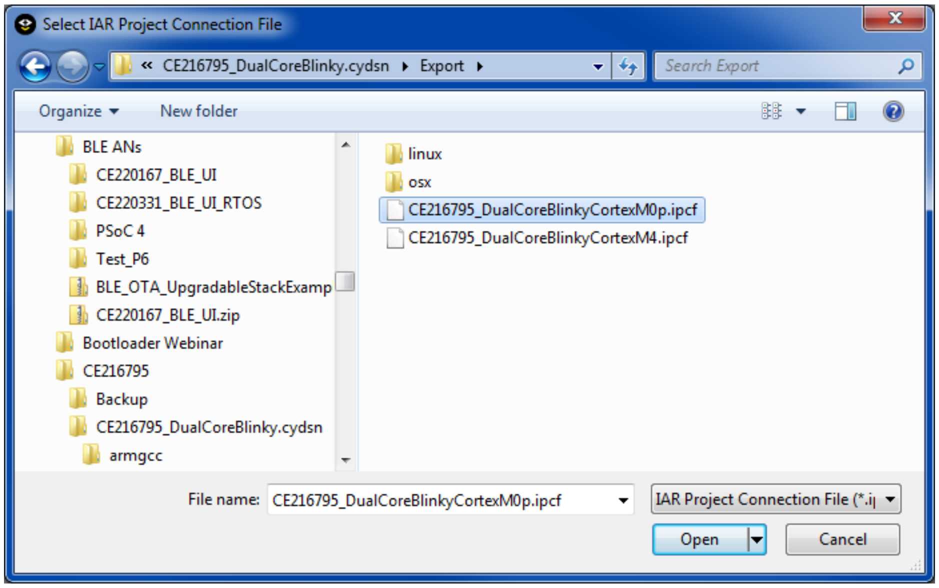
Figure 32. Create an IAR Embedded Workbench Project for CM0+


Save As
CE216... ▸ CE216795_DualCoreBlinky.cydsn ▸
Search CE216795_DualCoreBli...
Browse Folders
Save
Cancel

File name: IAR_BlinkyM0p
Save as type: Project Files (*.ewp)

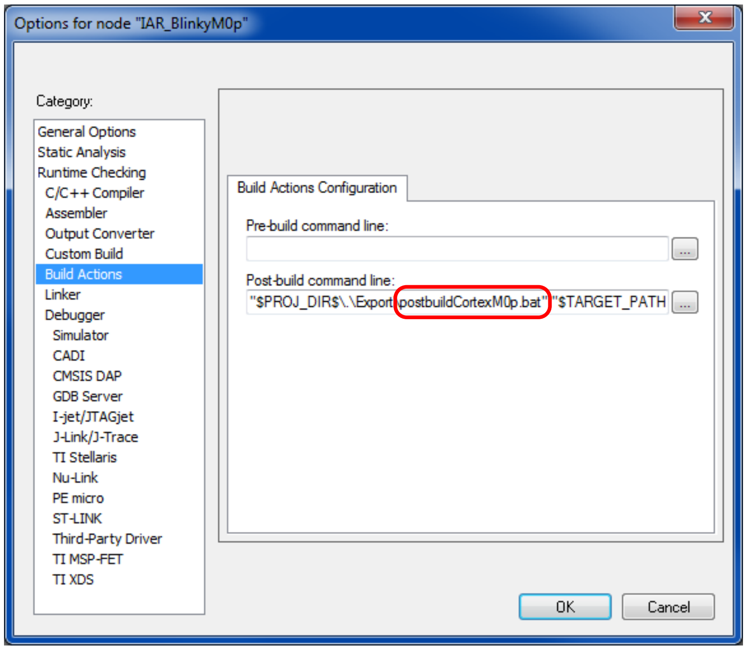
Select **Tools** > **Options** and make sure that **Enable project connections** is checked. Click **OK**. Then select **Project** > **Add Project Connection...**. In the next dialog, select Connect using **IAR Project Connection**, and click **OK**. Then select the *...CortexM0p.ipcf* file, as Figure 33 shows. Click **OK**, and several folders and files are added to the project in the Workspace window.

Figure 33. Select IAR Project Connection File from PSoC Creator Project Export Folder


Select IAR Project Connection File
« CE216795_DualCoreBlinky.cydsn ▸ Export ▸
Search Export
Organize
New folder
BLE ANs
CE220167_BLE_UI
CE220331_BLE_UI_RTOS
PSoC 4
Test_P6
BLE_OTA_UpgradableStackExamp
CE220167_BLE_UI.zip
Bootloader Webinar
CE216795
Backup
CE216795_DualCoreBlinky.cydsn
armgcc
linux
osx
CE216795_DualCoreBlinkyCortexM0p.ipcf
CE216795_DualCoreBlinkyCortexM4.ipcf
File name: CE216795_DualCoreBlinkyCortexM0p.ipcf
IAR Project Connection File (*.ip
Open
Cancel

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 34 shows.

Figure 34. Select PSoC Creator Post-Build Batch File



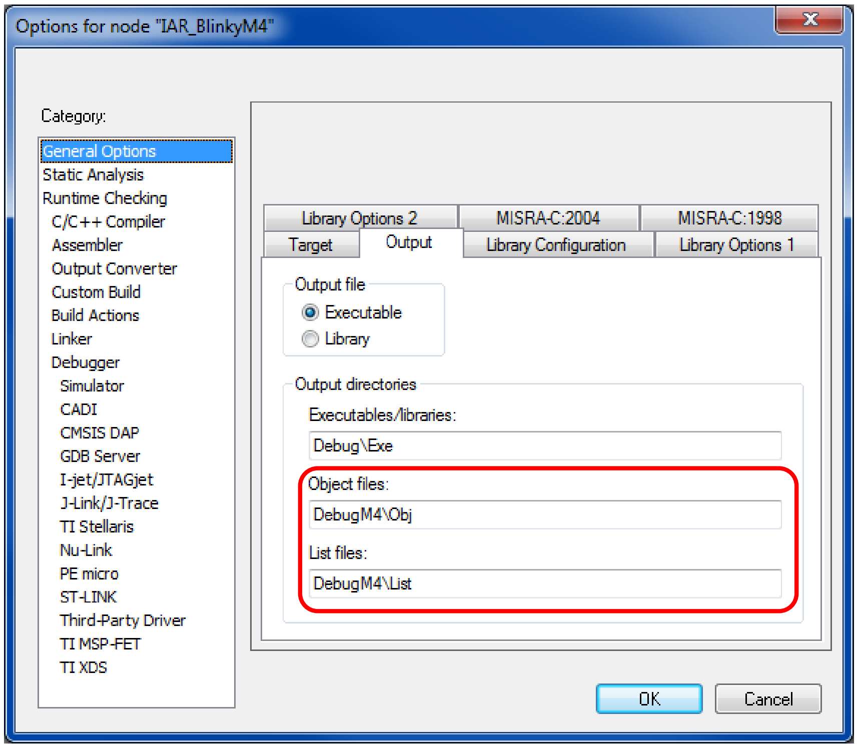
Post-build command line: "$PROJ_DIR$\..\Export\postbuildCortexM0p.bat" "$TARGET_PATH

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 32. 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 32.
- Select the *...CortexM4.ipcf* file; see Figure 33.
- In the **Options** dialog, **General Options** section, **Output** tab, change the output directories for object and list files, as Figure 35 shows. Do not change the executables/libraries output folder.

Figure 35. Unique Output Folders for CM4 Project


Options for node "IAR_BlinkyM4"

Category:

General Options
Static Analysis
Runtime Checking
C/C++ Compiler
Assembler
Output Converter
Custom Build
Build Actions
Linker
Debugger
Simulator
CADI
CMSIS DAP
GDB Server
I-jet/JTAGjet
J-Link/J-Trace
TI Stellaris
Nu-Link
PE micro
ST-LINK
Third-Party Driver
TI MSP-FET
TI XDS

Library Options 2 | MISRA-C:2004 | MISRA-C:1998
Target | Output | Library Configuration | Library Options 1

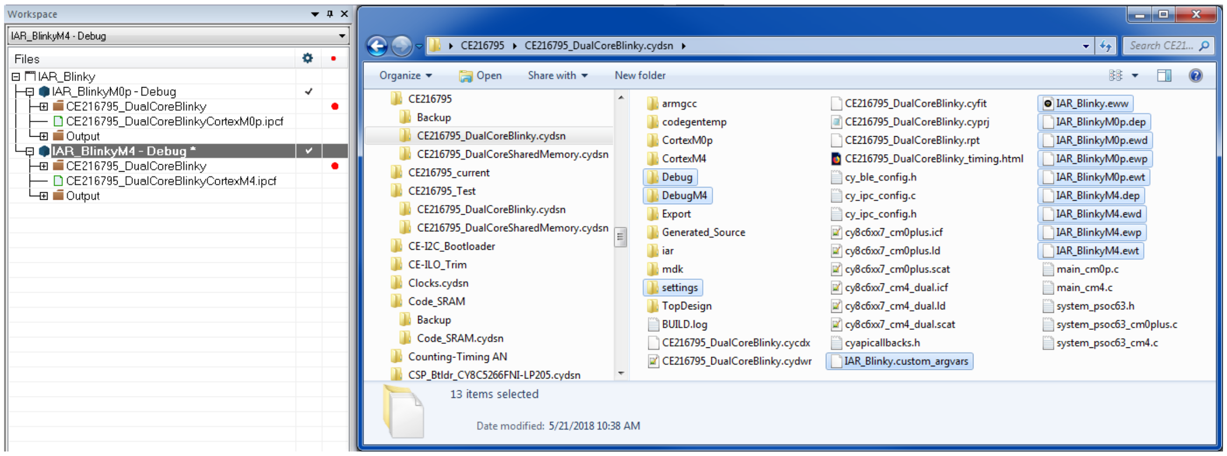
Output file
(● Executable) ○ Library

Output directories
Executables/libraries: Debug\Exe
Object files: DebugM4\Obj
List files: DebugM4\List

- In the **Build Actions** section, confirm that *postbuildCortexM4.bat* is called, see Figure 34.
- 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 36. The files and folders generated by IAR-EW are highlighted.

Figure 36. Resultant IAR Embedded Workbench Project Window and Project Files



```
Workspace
IAR_BlinkyM4 - Debug
Files
⊟ IAR_Blinky
├⊟ IAR_BlinkyM0p - Debug                         ✓
│ ├⊞ CE216795_DualCoreBlinky                      ●
│ ├── CE216795_DualCoreBlinkyCortexM0p.ipcf
│ └⊞ Output
└⊟ IAR_BlinkyM4 - Debug *                        ✓
  ├⊞ CE216795_DualCoreBlinky                      ●
  ├── CE216795_DualCoreBlinkyCortexM4.ipcf
  └⊞ Output
```

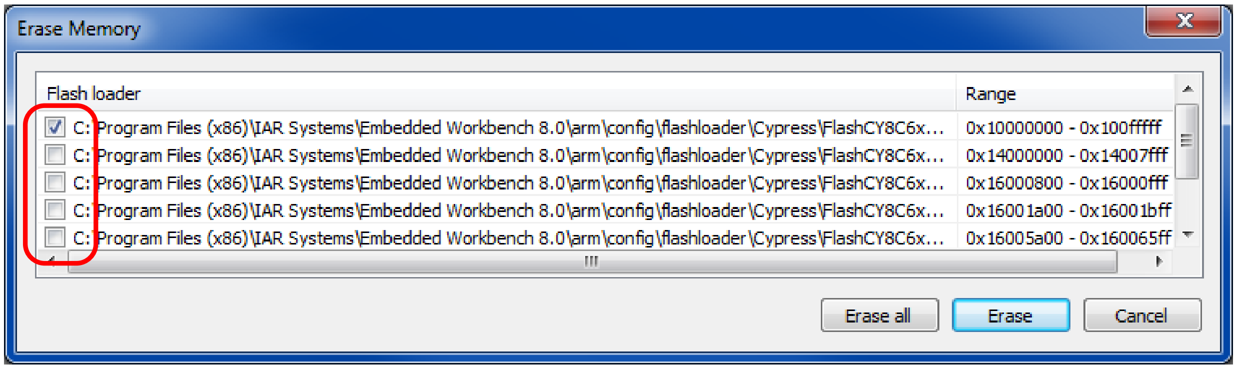
```
CE216795 ▸ CE216795_DualCoreBlinky.cydsn ▸
Organize ▾   Open   Share with ▾   New folder

CE216795
  Backup
  CE216795_DualCoreBlinky.cydsn
  CE216795_DualCoreSharedMemory.cydsn
CE216795_current
CE216795_Test
  CE216795_DualCoreBlinky.cydsn
  CE216795_DualCoreSharedMemory.cydsn
CE-I2C_Bootloader
CE-ILO_Trim
Clocks.cydsn
Code_SRAM
  Backup
  Code_SRAM.cydsn
Counting-Timing AN
CSP_Btldr_CY8C5266FNI-LP205.cydsn

armgcc                              CE216795_DualCoreBlinky.cyfit         IAR_Blinky.eww
codegentemp                         CE216795_DualCoreBlinky.cyprj         IAR_BlinkyM0p.dep
CortexM0p                           CE216795_DualCoreBlinky.rpt           IAR_BlinkyM0p.ewd
CortexM4                            CE216795_DualCoreBlinky_timing.html   IAR_BlinkyM0p.ewp
Debug                               cy_ble_config.h                       IAR_BlinkyM0p.ewt
DebugM4                             cy_ipc_config.c                       IAR_BlinkyM4.dep
Export                              cy_ipc_config.h                       IAR_BlinkyM4.ewd
Generated_Source                    cy8c6xx7_cm0plus.icf                  IAR_BlinkyM4.ewp
iar                                 cy8c6xx7_cm0plus.ld                   IAR_BlinkyM4.ewt
mdk                                 cy8c6xx7_cm0plus.scat                 main_cm0p.c
settings                            cy8c6xx7_cm4_dual.icf                 main_cm4.c
TopDesign                           cy8c6xx7_cm4_dual.ld                  system_psoc63.h
BUILD.log                           cy8c6xx7_cm4_dual.scat                system_psoc63_cm0plus.c
CE216795_DualCoreBlinky.cycdx       cyapicallbacks.h                      system_psoc63_cm4.c
CE216795_DualCoreBlinky.cydwr       IAR_Blinky.custom_argvars

13 items selected
Date modified: 5/21/2018 10:38 AM
```

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 37 shows:

Figure 37. IAR Embedded Workbench Erase Memory Dialog for PSoC 6 MCU



Erase Memory

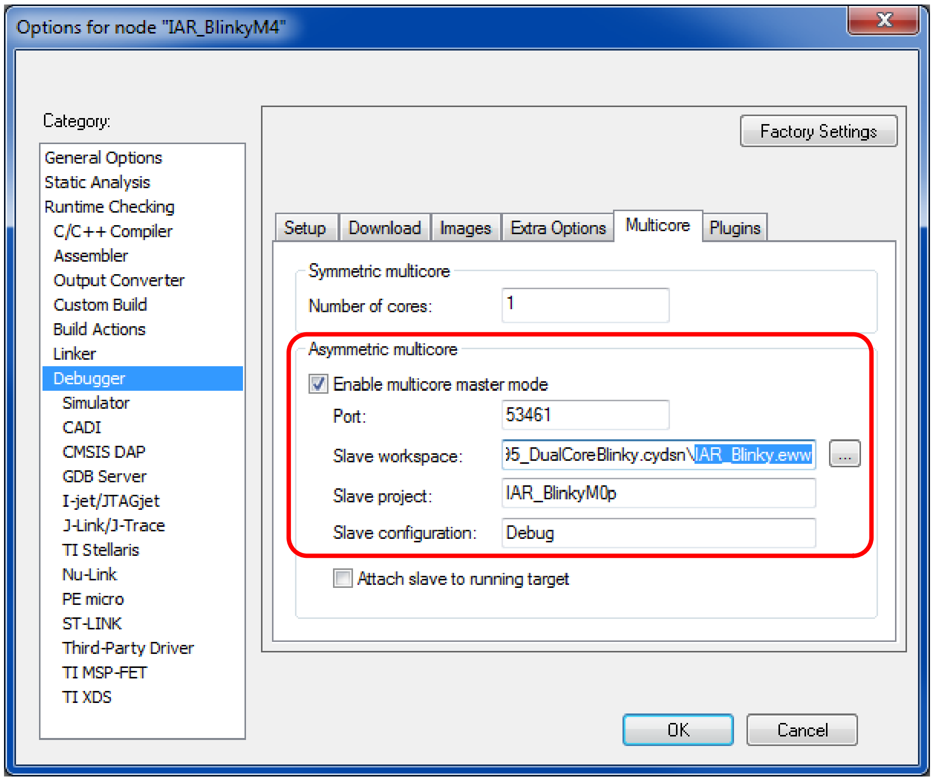
Flash loader	Range
✓ C:\Program Files (x86)\IAR Systems\Embedded Workbench 8.0\arm\config\flashloader\Cypress\FlashCY8C6x…	0x10000000 - 0x100fffff
☐ C:\Program Files (x86)\IAR Systems\Embedded Workbench 8.0\arm\config\flashloader\Cypress\FlashCY8C6x…	0x14000000 - 0x14007fff
☐ C:\Program Files (x86)\IAR Systems\Embedded Workbench 8.0\arm\config\flashloader\Cypress\FlashCY8C6x…	0x16000800 - 0x16000fff
☐ C:\Program Files (x86)\IAR Systems\Embedded Workbench 8.0\arm\config\flashloader\Cypress\FlashCY8C6x…	0x16001a00 - 0x16001bff
☐ C:\Program Files (x86)\IAR Systems\Embedded Workbench 8.0\arm\config\flashloader\Cypress\FlashCY8C6x…	0x16005a00 - 0x160065ff

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.

4.3.5 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 38 shows. Checking **Enable multicore master mode** makes the CM4 the master for download and debugging purposes. Do not change the **Port**.

Figure 38. Set Up Multicore Debugging



Symmetric multicore
Number of cores: 1

Asymmetric multicore
[✓] Enable multicore master mode
Port: 53461
Slave workspace:)5_DualCoreBlinky.cydsn\IAR_Blinky.eww
Slave project: IAR_BlinkyM0p
Slave configuration: Debug

[ ] Attach slave to running target

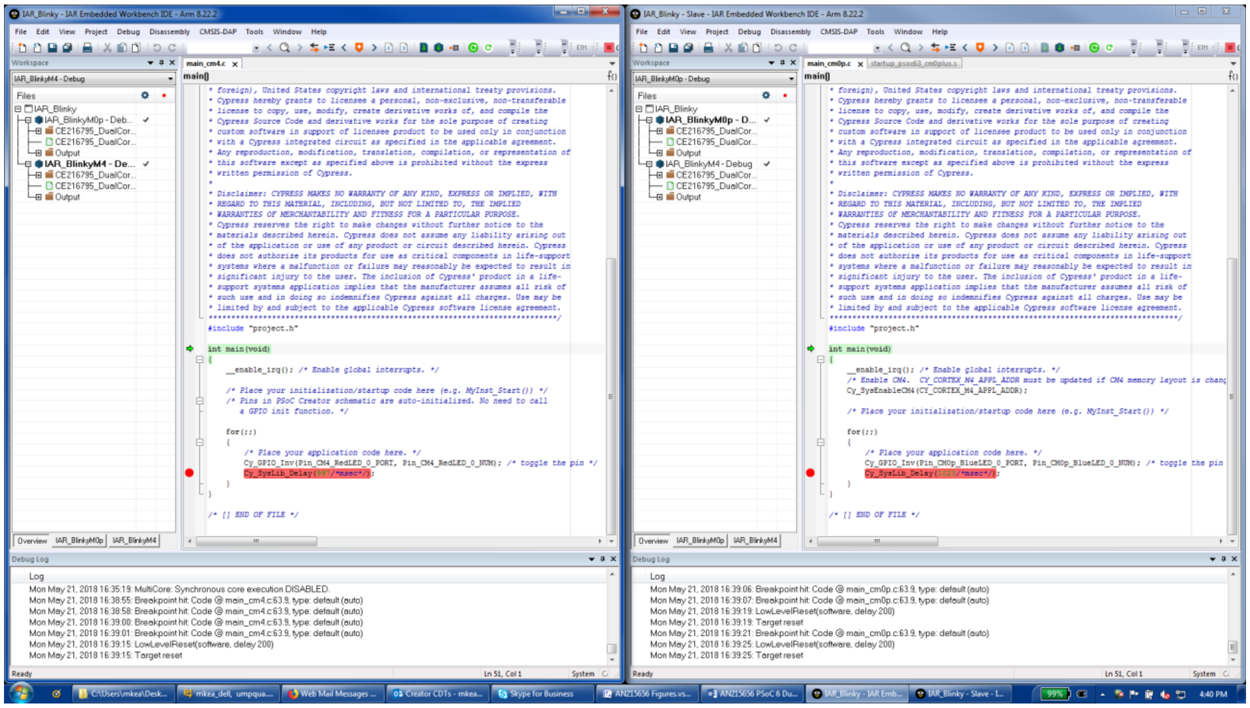
Select **File** > **Save All** to save the project options changes. Then start debugging by selecting either **Project** > **Download and Debug** or **Project** > **Debug without Downloading**. A second (slave) instance of IAR Embedded Workbench is automatically opened for the CM0+ project. Both instances 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 39. 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 39. IAR Embedded Workbench Dual-CPU Debugging



```
 * foreign), United States copyright laws and international treaty provisions.
 * Cypress hereby grants to licensee a personal, non-exclusive, non-transferable
 * license to copy, use, modify, create derivative works of, and compile the
 * Cypress Source Code and derivative works for the sole purpose of creating
 * custom software in support of licensee product to be used only in conjunction
 * with a Cypress integrated circuit as specified in the applicable agreement.
 * Any reproduction, modification, translation, compilation, or representation of
 * this software except as specified above is prohibited without the express
 * written permission of Cypress.
 *
 * Disclaimer: CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH
 * REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.
 * Cypress reserves the right to make changes without further notice to the
 * materials described herein. Cypress does not assume any liability arising out
 * of the application or use of any product or circuit described herein. Cypress
 * does not authorize its products for use as critical components in life-support
 * systems where a malfunction or failure may reasonably be expected to result in
 * significant injury to the user. The inclusion of Cypress' product in a life-
 * support systems application implies that the manufacturer assumes all risk of
 * such use and in doing so indemnifies Cypress against all charges. Use may be
 * limited by and subject to the applicable Cypress software license agreement.
 *******************************************************************************/
#include "project.h"

int main(void)
{
    __enable_irq(); /* Enable global interrupts. */

    /* Place your initialization/startup code here (e.g. MyInst_Start()) */
    /* Pins in PSoC Creator schematic are auto-initialized. No need to call
       a GPIO init function. */

    for(;;)
    {
        /* Place your application code here. */
        Cy_GPIO_Inv(Pin_CM4_RedLED_0_PORT, Pin_CM4_RedLED_0_NUM); /* toggle the pin */
        Cy_SysLib_Delay(997/*msec*/);
    }
}

/* [] END OF FILE */
```

```
Log
Mon May 21, 2018 16:35:19: MultiCore: Synchronous core execution DISABLED.
Mon May 21, 2018 16:38:55: Breakpoint hit: Code @ main_cm4.c:63.9, type: default (auto)
Mon May 21, 2018 16:38:58: Breakpoint hit: Code @ main_cm4.c:63.9, type: default (auto)
Mon May 21, 2018 16:39:00: Breakpoint hit: Code @ main_cm4.c:63.9, type: default (auto)
Mon May 21, 2018 16:39:01: Breakpoint hit: Code @ main_cm4.c:63.9, type: default (auto)
Mon May 21, 2018 16:39:15: LowLevelReset(software, delay 200)
Mon May 21, 2018 16:39:15: Target reset
```

```
 * foreign), United States copyright laws and international treaty provisions.
 * Cypress hereby grants to licensee a personal, non-exclusive, non-transferable
 * license to copy, use, modify, create derivative works of, and compile the
 * Cypress Source Code and derivative works for the sole purpose of creating
 * custom software in support of licensee product to be used only in conjunction
 * with a Cypress integrated circuit as specified in the applicable agreement.
 * Any reproduction, modification, translation, compilation, or representation of
 * this software except as specified above is prohibited without the express
 * written permission of Cypress.
 *
 * Disclaimer: CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH
 * REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED
 * WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.
 * Cypress reserves the right to make changes without further notice to the
 * materials described herein. Cypress does not assume any liability arising out
 * of the application or use of any product or circuit described herein. Cypress
 * does not authorize its products for use as critical components in life-support
 * systems where a malfunction or failure may reasonably be expected to result in
 * significant injury to the user. The inclusion of Cypress' product in a life-
 * support systems application implies that the manufacturer assumes all risk of
 * such use and in doing so indemnifies Cypress against all charges. Use may be
 * limited by and subject to the applicable Cypress software license agreement.
 *******************************************************************************/
#include "project.h"

int main(void)
{
    __enable_irq(); /* Enable global interrupts. */
    /* Enable CM4.  CY_CORTEX_M4_APPL_ADDR must be updated if CM4 memory layout is chang
    Cy_SysEnableCM4(CY_CORTEX_M4_APPL_ADDR);

    /* Place your initialization/startup code here (e.g. MyInst_Start()) */

    for(;;)
    {
        /* Place your application code here. */
        Cy_GPIO_Inv(Pin_CM0p_BlueLED_0_PORT, Pin_CM0p_BlueLED_0_NUM); /* toggle the pin
        Cy_SysLib_Delay(1023/*msec*/);
    }
}

/* [] END OF FILE */
```

```
Log
Mon May 21, 2018 16:39:06: Breakpoint hit: Code @ main_cm0p.c:63.9, type: default (auto)
Mon May 21, 2018 16:39:07: Breakpoint hit: Code @ main_cm0p.c:63.9, type: default (auto)
Mon May 21, 2018 16:39:19: LowLevelReset(software, delay 200)
Mon May 21, 2018 16:39:19: Target reset
Mon May 21, 2018 16:39:21: Breakpoint hit: Code @ main_cm0p.c:63.9, type: default (auto)
Mon May 21, 2018 16:39:25: LowLevelReset(software, delay 200)
Mon May 21, 2018 16:39:25: Target reset
```

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

5 Summary

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

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

- DMA Controllers. Note that the most common CPU assembler instructions output by C compilers are MOV, LDR, and STR, which implies that the CPU spends a lot of cycles just moving bytes around. Let the DMA controllers do that instead.

  Note: The PSoC 6 MCU DMA controllers have an extensive set of features that enable you to construct complex data transfer and control systems that are independent of the CPUs. Software support of these features is provided by both a PSoC Creator DMA Component and an API in the PDL. For more information, see the DMA Component datasheet and the PDL documentation.

- Crypto Block. This block offers hardware acceleration for symmetric and asymmetric cryptographic methods (AES, 3DES, RSA, and ECC) and hash functions (SHA-512, SHA-256). It also has a true random number generator (TRNG) function. Software support for these features is provided by an API in the PDL; see the PDL documentation.
- Universal Digital Blocks (UDBs). There are as many as 12 UDBs, and each UDB has an 8-bit datapath that can add, subtract, and do bitwise operations, shifts, and cyclic redundancy check (CRC). Datapaths can be chained for word-wide calculations. Consider offloading CPU calculations to the datapaths.
- UDBs also have programmable logic devices (PLDs) which can be used to build state machines; see for example the Lookup Table (LUT) Component datasheet. LUTs can be an effective hardware-based alternative to programming state machines in the CPU, for example by using C switch / case statements.

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

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

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

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

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

6 Related Documents

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

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

About the Author

Name:	Mark Ainsworth
Title:	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	Orig. of Change	Submission Date	Description of Change
**	5634375	MKEA	02/16/2017	New application note
*A	5653730	MKEA	03/08/2017	Updated template
*B	5777874	MKEA	06/09/2017	Updated text and screen shots for release versions of PSoC Creator 4.1 and PDL 3.0.0 Other miscellaneous edits
*C	5861685	MKEA	08/23/2017	Minor edits Ported to new application note document template Confidential tag removed
*D	6065641	MKEA	03/07/2018	Added a new Figure 2 Updated Figure 4 and associated kit device part number Updated Figures 6 and 9 for PSoC Creator 4.2 beta 2 Emphasized using CM0+ as a support CPU for tasks such as BLE and CapSense Added references to AN221111, Creating a Secure System; AN217666, PSoC 6 Interrupts; AN219528, PSoC 6 Low Power Modes; and CE216795, PSoC 6 Dual-CPU Updated power modes description Miscellaneous minor edits Ported to new application note template Changed the document title to PSoC 6 MCU Dual-CPU System Design
*E	6201597	MKEA	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

Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturer's representatives, and distributors. To find the office closest to you, visit us at Cypress Locations.

Products

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

PSoC® Solutions

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

Cypress Developer Community

Community | Projects | Videos | Blogs | Training | Components

Technical Support

cypress.com/support

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

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