请注意赛普拉斯已正式并入英飞凌科技公司。
此封面页之后的文件标注有“赛普拉斯”的文件即该产品为此公司最初开发的。请注意作为英飞凌产品组合的部分，英飞凌将继续为新的及现有客户提供该产品。

文件内容的连续性
事实是英飞凌提供如下产品作为英飞凌产品组合的部分不会带来对于此文件的任何变更。未来的变更将在恰当的时候发生，且任何变更将在历史页面记录。

订购零件编号的连续性
英飞凌继续支持现有零件编号的使用。下单时请继续使用数据表中的订购零件编号。

AN215656

PSoC 6 MCU 双 CPU 系统设计

作者: Mark Ainsworth

相关部件系列:: 全部双 CPU 的 PSoC® 6 MCU 部件

相关代码示例: CE216795

相关应用笔记: 参阅相关文档

更多代码示例？我们倾听到了你的声音。

要查看持续增长的成百上千的 PSoC 代码示例，请访问代码示例网页。您也可以在这里浏览 PSoC 培训视频库 。

AN215656 描述了 PSoC 6 MCU 中的双 CPU 架构，其中包括 Arm® Cortex®-M4 和 Cortex-M0+ CPU，以及处理器间通信（IPC）模块。 双 CPU 架构提供了灵活性，有助于提高系统性能和效率，并降低功耗。 此应用笔记还介绍了如何使用赛普拉斯的 ModusToolbox™, 软件命令行接口（CLI）和 PSoC Creator 集成开发环境（IDE）构建简单的双 CPU 设计，以及如何使用各种 IDE 调试设计。

目录

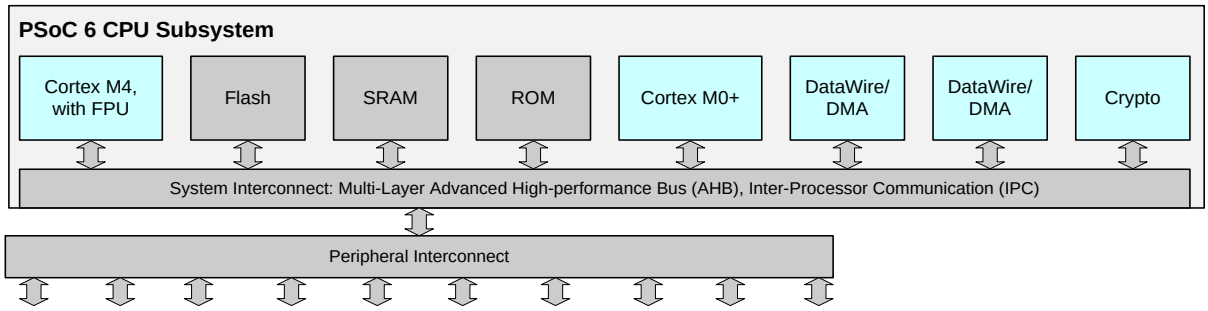
1 简介 1
1.1 如何使用本文档 2
2 通用双 CPU 概念 2
3 PSoC 6 MCU 双 CPU 架构 3
4 PSoC 6 MCU 双 CPU 开发 5
4.1 ModusToolbox 说明 5
4.2 PSoC Creator 说明 12
4.3 资源分配注意事项 15
4.4 中断分配注意事项 16
4.5 调试注意事项 17
5 总结 39
6 相关文档 40
文档修订记录 42
销售、解决方案以及法律信息 43

1 简介

PSoC 6 MCU 是赛普拉斯的 32 位超低功耗 PSoC，专为物联网（IoT）而设计。 它集成了低功耗闪存和 SRAM 技术、可编程数字逻辑，可编程模拟、高性能模数转换、低功耗比较器以及标准通信和定时外设。

PSoC 6 MCU 特别感兴趣的是 CPU 子系统。 该架构包含多个总线主控器 - 两个 CPU、两个 DMA 控制器和一个加密模块（Crypto）， 如 Figure 1 所示：

Figure 1. PSoC 6 MCU 典型 CPU 子系统架构



注意：Figure 1 中的框图内容可能因设备而异。某些 PSoC 6 MCU 部件只有一个 CPU 可用于应用目的。有关详细信息，请参见器件数据手册。本应用笔记不适用于单 CPU PSoC 6 MCU 器件。

通常，所有存储器和外设都由所有总线主控器共享。通过标准 Arm 多层总线仲裁访问共享资源。独占访问由处理器间通信（IPC）模块支持，该模块在硬件中实现信号量和互斥（mutexes）。

双 CPU 架构以及 DMA 和加密（Crypto）总线主控器为单个 MCU 提供了独特的系统级设计和性能优化机会。使用两个 CPU，您可以：

- 将任务分配给 CPU，以便可以同时完成多个任务
- 为 CPU 分配资源，以便 CPU 可以专门用于管理这些资源，从而提高效率
- 启用和禁用 CPU 以最大限度地降低功耗
- 使用 IPC 模块在 CPU 之间发送数据。有关更多信息，请参见代码示例 CE216795，PSoC 6 MCU 双 CPU 基础知识。

在一个示例应用中，Cortex-M0 + CPU（CM0 +）可以“拥有”并管理所有通信信道。Cortex-M4 CPU（CM4）可以通过 CM0 +从通道发送和接收消息。这使 CM0 +管理通信详细信息时 CM4 可以执行其他任务。

1.1 工具和库

赛普拉斯提供了两个开发平台，您可以使用它们进行 PSoC 6 MCU 的应用开发。

- **ModusToolbox:** ModusToolbox 软件包括配置工具、低级驱动程序、中间件库和操作系统支持，以及其他使您能够创建 MCU 和无线应用的软件包。它还包括可选的 Eclipse IDE for ModusToolbox。
- **PSoC Creator:** 一个赛普拉斯专有的 IDE，只在 Windows 上运行。它支持 PSoC 6 MCU 器件的一个子集，以及其他 PSoC 器件系列，如 PSoC 3、PSoC 4 和 PSoC 5LP。

  这两个平台都提供了一种机制，可以将项目导出到第三方工具，如 Keil µVision、IAR Embedded Workbench 和 Visual Studio Code。

  赛普拉斯还提供了一套库，以促进 PSoC 6 MCU 的应用开发。

- **外设驱动库 (PDL):** PDL 驱动程序将硬件功能抽象为一组易于使用的 API。有关 PDL 的更多信息，请访问 https://github.com/cypresssemiconductorco/psoc6pdl 网页。该库可在 PSoC Creator 和 ModusToolbox 软件中获取。
- **硬件抽象层 (HAL):** HAL 是建立在 PDL 之上的。它为配置和使用赛普拉斯 MCU 上的硬件模块提供了一个高级接口。它是一个通用接口，可以在多个产品系列中使用。关于 HAL 的更多信息，请访问 https://github.com/cypresssemiconductorco/psoc6hal 网页。该 HAL 可在 ModusToolbox 软件中获取，并且仅适用于 CM4 CPU。PSoC Creator 不支持 HAL。
- **中间件:** 该类别包括为不同领域实现 API 的多个库，例如电容感应或 HTTP 服务器。中间件库可以由赛普拉斯创建，也可以来自第三方。在 PSoC Creator 中，中间件通常作为 PSoC Creator 组件提供。在 ModusToolbox 软件中，它可以作为一个库使用。

  板卡支持包（BSP）。BSP 为电路板的特性和功能提供了一个标准接口。该 API 在赛普拉斯套件中是一致的。其他软件（如中间件或应用程序）可以使用 BSP 来配置和控制硬件。BSP 仅在 ModusToolbox 软件中可用。

1.2 如何使用本文档

本文档假定您熟悉 PSoC 6 MCU 架构、以及使用赛普拉斯 ModusToolbox 软件或 PSoC Creator IDE 的 PSoC 器件的应用开发。有关 PSoC 6 MCU 的介绍，请参阅以下内容：

- PSoC6 MCU 器件数据表
- AN221774，PSoC 6 MCU 入门
- AN210781，带蓝牙低功耗（BLE）连接的 PSoC 6 MCU 入门

如果您不熟悉 ModusToolbox 软件，请参阅 ModusToolbox 软件主页。PSoC Creator 不支持某些 PSoC 6 MCU 器件;在此情况下必须使用 ModusToolbox 软件。如果您正使用 ModusToolbox 软件，确保版本为 2.2 或更高，以进行双 CPU 的 PSoC 6 MCU 器件设计。

如果您不熟悉 PSoC Creator，请参见 PSoC Creator 主页。将 PSoC Creator 4.3 或更高版本用于基于 PSoC 6 MCU 的设计。

本应用笔记的初始部分介绍了双 CPU MCU 的一般概念以及它们在 PSoC 6 MCU 中的实现方式。要跳过有关为 PSoC 6 双 CPU MCU 创建 ModusToolbox 软件或 PSoC Creator 项目的概述，请转至 PSoC 6 MCU 双 CPU 开发部分。

2 通用双 CPU 概念

为双 CPU MCU 开发固件的过程类似于单 CPU MCU 的过程，除了您为两个 CPU 而不是一个 CPU 编写代码。您还应该考虑对处理器间通信的任何需求。

- **性能：**拥有两个 CPU 的主要优点是您肯定能增加 CPU 功率和带宽。使用 PSoC 6 MCU，增加带宽的代价通常与单 CPU MCU 相当。如何使用增加的带宽取决于应用程序必须执行的任务：

- **单个任务**：单任务应用程序可能不太适合双 CPU MCU，除非应用程序庞大且复杂。在 PSoC 6 MCU 中，您可以在其中一个 CPU 上执行任务，并将另一个 CPU 置于休眠状态以降低功耗。
- **双重任务**：这是最明显的合适选择;将每个任务分配给 CPU。将具有更大计算要求的任务分配给更高性能的 CPU，即 PSoC 6 MCU 中的 Cortex M4。
- **多重任务**：再次将每个任务分配给 CPU。在每个 CPU 中，您必须包含一个及时执行每个任务的方法。
- **RTOS**：复杂的多任务系统可以由实时操作系统（RTOS）管理。 RTOS 基本上为每个任务分配了多个 CPU 周期，具体取决于任务优先级或任务是否在等待事件。您可以通过将任务分配给 CPU 来有效地完成此操作。双 CPU RTOS 架构的一些示例如下：
  - 每个 CPU 都有自己的 RTOS 和自己的一组任务。每个 RTOS 都应包含管理与其他 CPU 通信的任务。
  - 只有一个 CPU 具有 RTOS 和多个任务。另一个 CPU 处于空闲状态，直到它被告知执行指定的任务。 然后它会唤醒并完成任务，然后将结果发送回第一个 CPU。 例如，PSoC 6 MCU 中性能较低的 CPU（CM0 +）可以使用 PSoC 6 MCU 中性能更高的 CPU（CM4）在需要时执行计算密集型任务。

  功耗：在双 CPU 系统中，固件可以启动和停止 CPU 以微调电源使用。 在前面的示例中，为了降低功耗，高性能 CPU 被置于睡眠状态，直到需要执行计算密集型任务为止。

  调试：同时调试两个代码体可能是一个复杂的过程。 通常，您调试一个 CPU 的代码，然后调试另一个 CPU 的代码。 另外，诸如示波器或逻辑分析器的设备可用于监视 CPU 之间的通信。

3 PSoC 6 MCU 双 CPU 架构

第 1 页的 Figure 1 显示了 PSoC 6 MCU 中的总体双 CPU 架构。（有关 PSoC 6 MCU 的详细框图，请参见器件数据手册或 AN221774）。本节列出了与双 CPU 相关的特定功能和其他详细信息。有关更多信息，请参阅 Cortex-M4 和 Cortex-M0 +的 Arm 文档集以及 PSoC 器件技术参考手册（TRM）(Only English)。

- **CPU**：两个 CPU - Cortex M4 和 Cortex M0 + - 都是 32 位。 CM4 运行速度高达 150 MHz，并具有浮点单元（FPU）。 CM0 +运行速度高达 100 MHz。

  CM4 是主 CPU。它专为短中断响应时间、高代码密度和高吞吐量而设计。 CM0 + CPU 是副 CPU； 它在 PSoC 6 MCU 中用于实现系统调用和设备级加密、安全和保护功能。 CM0 +也推荐用于 BLE 通信和 CapSense 等功能。

- **性能**：CM0 +通常以比 CM4 更慢的时钟速度运行。 CM0 +指令集比 CM4 更有限。因此，在 CM0 +上实现功能可能需要更多周期，并且周期时间更长。在决定分配任务的 CPU 时，请记住这一点。
- **安全**：PSoC 6 MCU 具有多种安全功能;有关详细信息，请参阅 TRM。为了满足安全要求，CM0 +用作“安全 CPU”。它被认为是一个可信任的实体;它执行赛普拉斯系统代码和应用程序代码。将 CM0 +用于系统和安全任务可能会限制其应用程序的可用性。有关安全系统的更多信息，请参阅 AN221111，创建安全系统 (Only English)。

  设备系统调用可以由任一 CPU 启动，但始终由 CM0 +执行。

- **启动顺序**：器件复位后，只执行 CM0 +；CM4 保持在复位状态。 CM0 +首先执行赛普拉斯系统和安全代码，包括 SROM 代码、FlashBoot 和安全映像。有关这些代码模块的更多信息，请参阅架构 TRM 的 Boot Code 章节了解更多信息。

  在 CM0 +执行系统和安全代码后，它执行应用程序代码。在应用程序代码中，CM0 +可能会释放 CM4 重置，导致 CM4 开始执行其应用程序代码。 ModusToolbox IDE 和 PSoC Creator 都在 CM0 + main（）中自动生成代码以释放 CM4 复位。

- **处理器间通信（IPC）**：IPC 使 CPU 能够进行通信和同步活动。 IPC 硬件包含 IPC 通道功能和 IPC 中断的寄存器结构。 IPC 通道寄存器实现互斥（mutex）锁定/释放机制以及 CPU 之间的消息传递。 IPC 中断寄存器为两个 CPU 产生中断，用于发送消息事件和锁定/释放事件。
- **中断**：每个 CPU 都有自己的一组中断。所有外设中断线都硬连线到特定的 CM4 中断输入。外设中断也与 CM0 +有限的 8 个中断输入（CY8C61x7, CY8C62x7, and CY8C63x7 中的 32 中断输入）复用。请参阅中断分配注意事项。
- **功耗模式**：PSoC 6 MCU 具有多种功耗模式，可以影响整个系统或仅影响单个 CPU。 CPU 功耗模式为 Arm 定义的活动、睡眠和深度睡眠。设备系统功耗模式为 LP，ULP，深度睡眠和休眠。
  - 系统低功耗（LP）模式是复位后器件的默认工作模式。它提供了最大的性能。在系统 LP 模式下，CPU 可以在任何 Arm 定义的模式下运行。
  - 系统超低功耗（ULP）模式与 LP 模式相同，性能平衡以实现更低的系统电流。这

种平衡降低了核心工作电压，然后也需要降低工作时钟频率和有限的高频时钟源。在系统 ULP 模式下，CPU 可以在任何 Arm 定义的模式下操作。

- 在系统深度睡眠模式下，所有高速时钟源均为 OFF。这反过来会阻止两个 CPU 并使高速外设无法使用。但是，如果由固件配置并启用，则低速时钟源和外设将继续运行。来自这些外设的中断导致器件返回到系统 LP 或 ULP 模式，并使一个或多个 CPU 唤醒到活动模式。每个 CPU 都有一个唤醒中断控制器（WIC）来唤醒 CPU。
- 系统休眠模式是设备的最低功耗模式。它适用于可能进入休眠状态的应用程序。从 Hibernate 唤醒时，设备会重置。请参阅启动顺序。
- 在 CPU 活动模式下，CPU 执行代码并且所有逻辑和存储器都通电。设备必须处于 System LP 或 ULP 模式。
- 在 CPU 睡眠模式下，CPU 时钟关闭，CPU 暂停代码执行。设备必须处于 System LP 或 ULP 模式。
- 在 CPU 深度睡眠模式下，CPU 请求设备进入系统深度睡眠模式。设备准备就绪后，进入系统深度睡眠模式。在 PSoC 6 MCU 中，两个 CPU 必须在系统转换为深度睡眠之前进入 CPU 深度睡眠模式。如果只有一个 CPU 进入 CPU 深度睡眠模式，则系统仍处于 LP 或 ULP 模式。

有关 PSoC 6 MCU 功耗模式的更多信息，请参见 AN219528，PSoC 6 MCU 低功耗模式和功耗降低技术(Only English)。

- **调试：**PSoC 6 MCU 具有调试访问端口（DAP），可用作器件编程和调试的接口。外部编程器或调试器（“主机”）通过器件串行线调试（SWD）或联合测试操作组（JTAG）接口引脚与 DAP 通信。通过 DAP（并受设备安全限制），主机可以访问设备存储器和外设以及两个 CPU 中的寄存器。

  每个 CPU 提供以下几种调试和跟踪功能：

  - CM4 通过单线输出（SWO）引脚支持六个硬件断点和四个观察点、4 位嵌入式跟踪宏单元（ETM）、串行线查看器（SWV）和 printf（）样式调试。
  - CM0+支持四个硬件断点和两个观察点，以及一个带 4 KB 专用 RAM 的微跟踪缓冲区（MTB）。

  PSoC 6 MCU 还具有嵌入式交叉触发器，用于同步调试和跟踪两个 CPU。

  ModusToolbox 软件和一些第三方 IDE 支持双 CPU 调试和使用；请参阅调试注意事项。PSoC Creator 支持一次调试单个 CPU（CM4 或 CM0 +）。

4 PSoC 6 MCU 双 CPU 开发

本节仅显示 PSoC 6 MCU 双 CPU 器件独有的开发方面。要了解有关 PSoC 6 MCU，ModusToolbox 软件或 PSoC Creator 的更多信息，请参阅以下一项或多项：

- AN221774，PSoC 6 MCU 入门
- AN210781，带蓝牙低功耗（BLE）连接的 PSoC 6 MCU 入门
- ModusToolbox 软件主页。 PSoC Creator 不支持某些 PSoC 6 MCU 器件;此情况下必须使用 ModusToolbox 。如果您使用的是 ModusToolbox 软件，请确保版本为 2.2 或更高，以进行基于双 CPU PSoC 6 MCU 器件的设计。
- PSoC Creator 主页。将 PSoC Creator 4.2 或更高版本用于基于 PSoC 6 MCU 的设计。

4.1 ModusToolbox 软件说明

针对 PSoC 6 MCU 双 CPU 器件的 ModusToolbox 软件应用开发与 ModusToolbox 软件支持的任何其他器件类似。默认情况下，ModusToolbox 软件附带的板卡支持包（BSP）提供了几个预置的 CM0+应用程序镜像。本节解释了如何创建你自己的 CM0+应用程序，以替代预置的镜像。

要创建一个双 CPU 工作区，你必须创建两个新的应用程序，一个用于 CM0+ CPU，一个用于 CM4 CPU。或者，你可以根据一个双 CPU 的代码范例创建一个新的项目，该范例已经设置了两个应用程序。你可以使用任何代码示例，其 repo 名称包括 "dual-cpu"，作为起始点。请看这里的列表。

接下来的章节展示了如何逐步创建一个双 CPU 工作区。

每一节都解释了使用 ModusToolbox 软件的不同工具的两种方法：

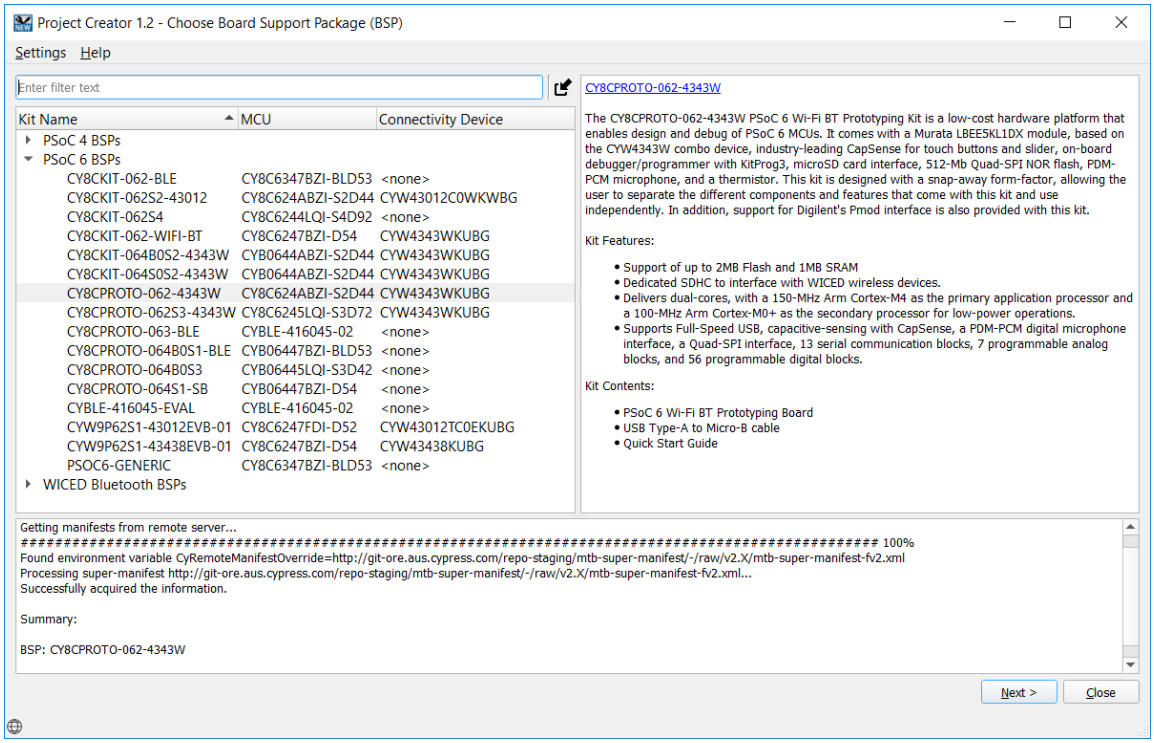
- Eclipse IDE for ModusToolbox
- Command-line Interface (CLI)

4.1.1 创建 CM0+ CPU 应用

4.1.1.1 在 Eclipse IDE for ModusToolbox 中

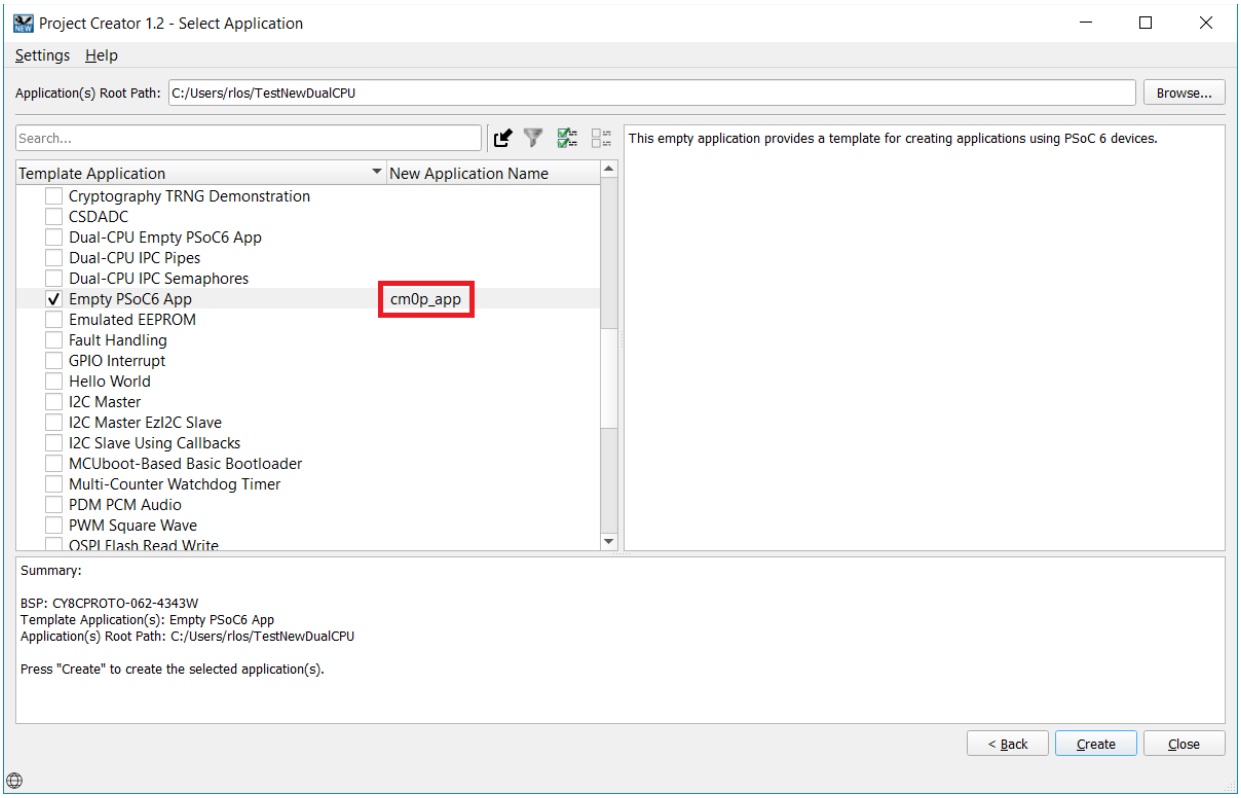
1. 点击快速面板中的**New Application**。然后，选择一项PSoC 6 BSP，如**Error! Reference source not found.**所示。

Figure 2.软件中的板卡支持包选择



2. 点击 **Next**，然后选择 **Empty PSoC6 App**。在 **New Application Name** 中，将其设置为 "cm0p_app"，如 Figure 3.所示。完成 "新应用程序 "对话，一个新的应用程序就在选定的工作区中创建了。

Figure 3. CM0+ CPU 应用创建



4.1.1.2 在 CLI 中创建

1. 打开一个 CLI 终端并导航到 ModusToolbox 软件安装文件夹进行项目创建 (*<install_path>ModusToolbox/tools_<version>/project-creator/*).
2. 通过执行 `project-creator-cli` 命令创建一个新项目：

   例如：

```
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"
```

或者，你可以在同一文件夹中执行 **project-creator** 工具，并遵循在 Eclipse IDE for ModusToolbox 中描述的相同步骤。

更多细节见 ModusToolbox User Guide（第 2.3 节）。

4.1.1.3 最终修改

一旦应用程序被创建（与所使用的方法无关），打开 Makefile 文件并做如下修改。

- 创建新行，将 CORE 设置为 CM0+。默认情况下，make 文件假定应用程序的目标是 CM4。添加这一行可以强制应用目标为 CM0+。

  ```
  CORE=CM0P
  ```

- 可以选择将 APPNAME 变量改为 **cy_m0p_image**，这是链接器脚本使用的默认名称。

  ```
  APPNAME=cy_m0p_image
  ```

当为 CM0+ CPU 编写代码时，你不能利用 Cypress HAL，所以只能使用 PDL APIs。修改 *main.c* 文件，删除对 HAL 的任何引用。也可以选择删除对 BSP 的任何引用。

例如:

```
#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);
    }
}
```

赛普拉斯的 HAL 不是为在 CM0+上运行而设计的，因此它将不能正确编译。另外，建议只在一个 CPU 中初始化 BSP，最好是 CM4 CPU。

CM0+应用程序在默认情况下也不参考 *design.modus* 生成的文件。如果你想把这些文件作为 CM0+应用程序的一部分，在 CM0+ Makefile 的 COMPONENTS 变量中加入以下内容:

```
COMPONENTS=BSP_DESIGN_MODUS
```

或者如果你使用的是自定义的 design.modus。

```
COMPONENTS=CUSTOM_DESIGN_MODUS
```

4.1.2 创建 CM4 CPU 应用

4.1.2.1 在 Eclipse IDE for ModusToolbox 中

在为 CM0+ CPU 选择的相同 BSP 基础上创建一个新的应用程序。你可以选择任何应用程序模板。将应用程序命名为 "cm4_app "并完成应用程序的创建。

4.1.2.2 在 CLI 中

Run the project-creator-cli again for creating the “cm4_app” application. For example: 再次运行 project-creator-cli 来创建 "cm4_app" 应用程序。例如

```
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”
```

注意你可以使用任何在-app-id 中的模板应用程序。要查看完整的列表，请跳转到 Cypress GitHub 仓库。

4.1.2.3 最终修改

一旦应用程序被创建，打开 Makefile 并做如下修改。

- 在 DISABLE_COMPONENT 列表中加入 "CM0P_SLEEP"。如果你使用的是另一个预置库，请相应地进行修改。

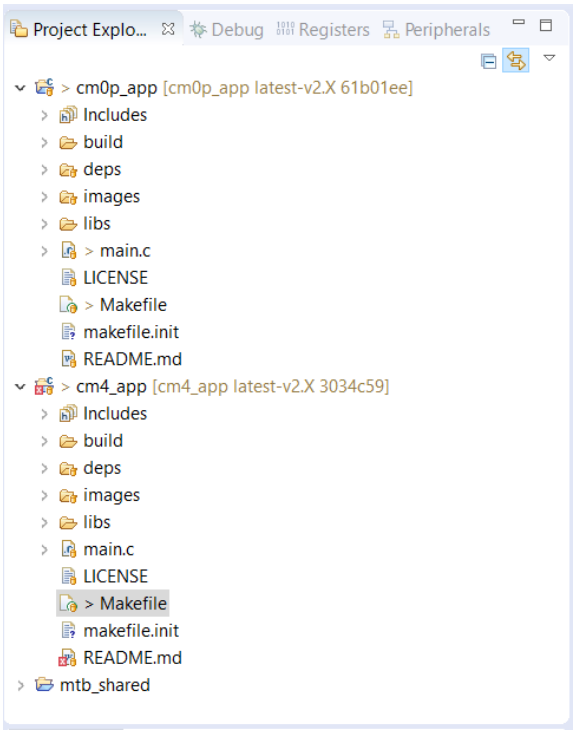
```
DISABLE_COMPONENTS=CM0P_SLEEP
```

- 创建一个新行来添加 cm0p_app 的相对路径作为依赖文件。

```
DEPENDENT_APP_PATHS=../cm0p_app
```

按照上述步骤操作后，你可以在 Project Explorer 窗口（仅在 Eclipse IDE for ModusToolbox 中）查看这些应用程序，如 Figure 4 所示。两个应用程序都有自己的一套库、配置和 make 文件。

Figure 4. Project Explorer



CM4 应用程序会自动构建依赖的 CM0+应用程序。它链接应用程序两次，一次没有 CM0+镜像，另一次有 CM0+镜像。默认镜像包括 CM0+镜像，就像使用预先构建的镜像之一构建 CM4 应用程序时那样。

在 Eclipse IDE for ModusToolbox 中，当构建或编程应用程序时，你需要选择应用程序。你可以在项目浏览器中点击属于该应用程序的任何文件或文件夹。快速面板会更新它的链接，包括基于所选应用程序的启动脚本。

建议只在其中一个 CPU 中初始化硬件。因此，只有一个 *design.modus* 文件应该被编辑。

mtb_shared 文件夹包含可能被两个应用程序共享的库和 BSP 文件。每个应用程序都有自己的 deps 文件夹。在这个文件夹中添加应用程序可能使用的任何依赖库。注意，同一个库可能会出现在两个应用程序中。

4.1.3 自定义链接脚本

CM0+和 CM4 应用程序都有自己的链接脚本。默认情况下，CM0+ CPU 只消耗 8192 [0x2000] 字节的闪存和 8192 [0x2000] 字节的 SRAM。如果你的 CM0+应用程序需要更多的内存，两个链接脚本都需要改变。下面的步骤列出了这些改变。

1. 在项目的根目录下创建一个共享文件夹来存储自定义链接脚本。例如：

   shared / linker_script / COMPONENT_CM0P

   shared / linker_script / COMPONENT_CM4

2. 将下面文件夹中的 CM0+链接脚本文件复制到 "*shared / linker_script / COMPONENT_CM0P* "文件夹中：

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

   默认情况下，BSP 支持 ARM（*.sct）、GCC_ARM（*.ld）和 IAR（*.icf）工具链。每个工具链都有自己的链接脚本。

3. 根据需要编辑 FLASH 和 SRAM 的大小。

工具链	需要修改的地方
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. 将下面文件夹中的 CM4 链接脚本文件复制到 "shared / linker_script / COMPONENT_CM4 "文件夹中

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

5. 根据 CM0+的内存大小来编辑数值：

工具链	需要修改的地方

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. 如果你在 CM0+应用 make 文件中命名的 APPNAME 与 "cy_m0p_image "不同，链接脚本中的任何引用都需要用新的名称来替换。

7. 在 CM0+和 CM4 的 Makefile 中，添加下面这一行，以引用自定义的链接脚本

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

   你需要根据复制到你在步骤（1）和（2）中创建的链接脚本文件夹中的文件来替换 FILENAME 和 EXTENSION。

8. 在 CM0+ Makefile 中，在 DEFINES 中加入 CM4 应用地址（CY_CORTEX_M4_APPL_ADDR）。例如，如果 CM0+镜像的大小是 0x8000，则设置为。

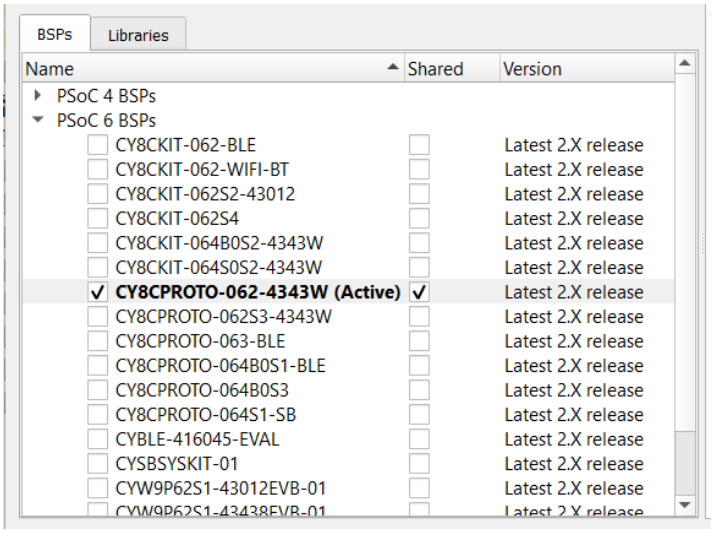
   DEFINES=CY_CORTEX_M4_APPL_ADDR=0x10008000

4.1.4 共享库和外设

如前所述，CM4 和 CM0+可以共享或不共享一些应用程序所使用的库。每个应用程序的 deps 文件夹包含了该应用程序使用的库的列表。ModusToolbox 提供了一个管理库和 BSP 的工具，叫做 Library Manager。你可以从 Eclipse IDE for ModusToolbox 的 **Quick Panel** > **Tools** > **Library Manager** 启动这个工具。或者，你也可以执行位于这个文件夹的库管理器工具。(<install_path>ModusToolbox/tools_<version>/library-manager/) 。

当选择 BSP 或库时，该工具提供了一个选项来分享它（Shared 列），如 Figure 5 所示。

Figure 5. Library Manager



在双 CPU 应用中，建议共享所有的库和 BSP。

当需要配置外设时，建议在双 CPU 应用中只有一个 design.modus，并且只有一个 CPU 可以调用 init_cycfg_all()或 cybsp_init()，最好是 CM4。如果一个外设只被其中一个 CPU 使用，你可以像在单 CPU 应用中那样使用。

如果一个外设是共享的，你应该只在其中一个 CPU 中初始化它，并使用某种互斥或同步机制来避免两个 CPU 同时使用它。不建议将 HAL 用于共享外设，因为 CM0+不能访问由 CM4 创建的 HAL 对象。

4.2 PSoC Creator 说明

PSoC 6 MCU 双 CPU 器件的 PSoC Creator 项目开发类似于 PSoC Creator 支持的任何其他器件。 要创建新项目，请选择 **File** > **New** > **Project**. A **Create Project**。 将显示“**Create Project**”对话框，如 Figure 6 所示。

选择 **Target Device** (A)和 PSoC 6（B）。 在下拉列表（C）中，选择**<Launch Device Selector...>**以查看 PSoC 6 设备列表。

Figure 6. PSoC Creator 创建项目对话框

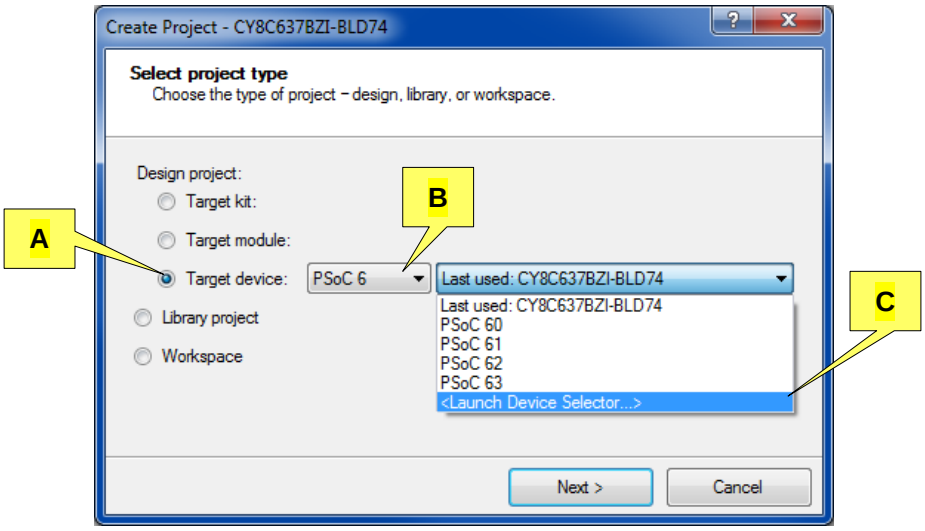
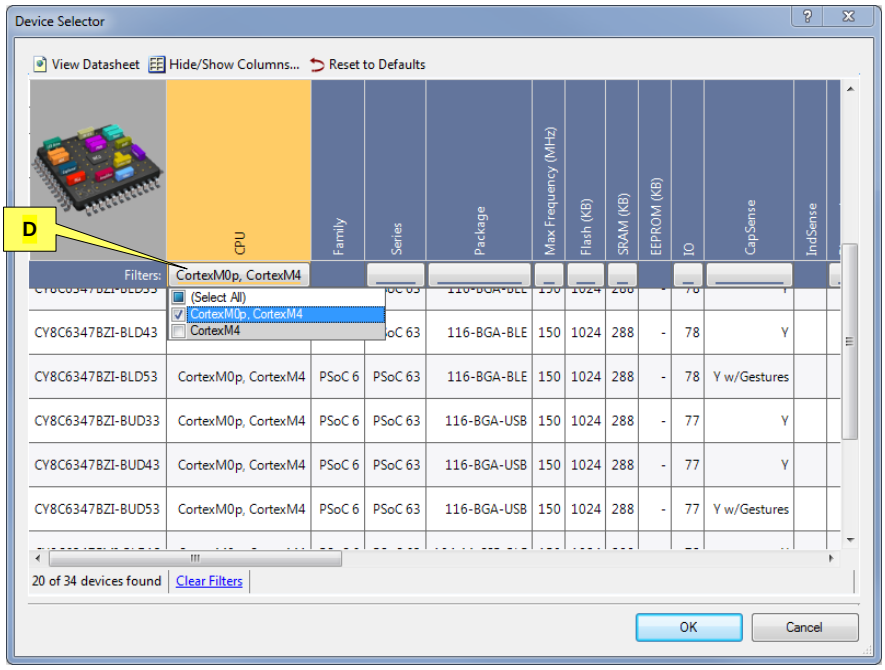


Figure 7 显示了设备选择器对话框。 要查看双 CPU 设备列表，请单击 **CPU** category (D)，然后仅选择 **CortexM0p, CortexM4**。

在 PSoC 6 BLE Pioneer 套件 CY8CKIT-062-BLE 中，PSoC 6 MCU 双 CPU 器件部件号为 CY8C6347BZI-BLD53。 在 PSoC 6 WiFi-BT Pioneer 套件 CY8CKIT-062-WiFi-BT 中，PSoC 6 MCU 双 CPU 器件部件号为 CY8C6247BZI-D54。

Figure 7. PSoC Creator 器件选择对话框

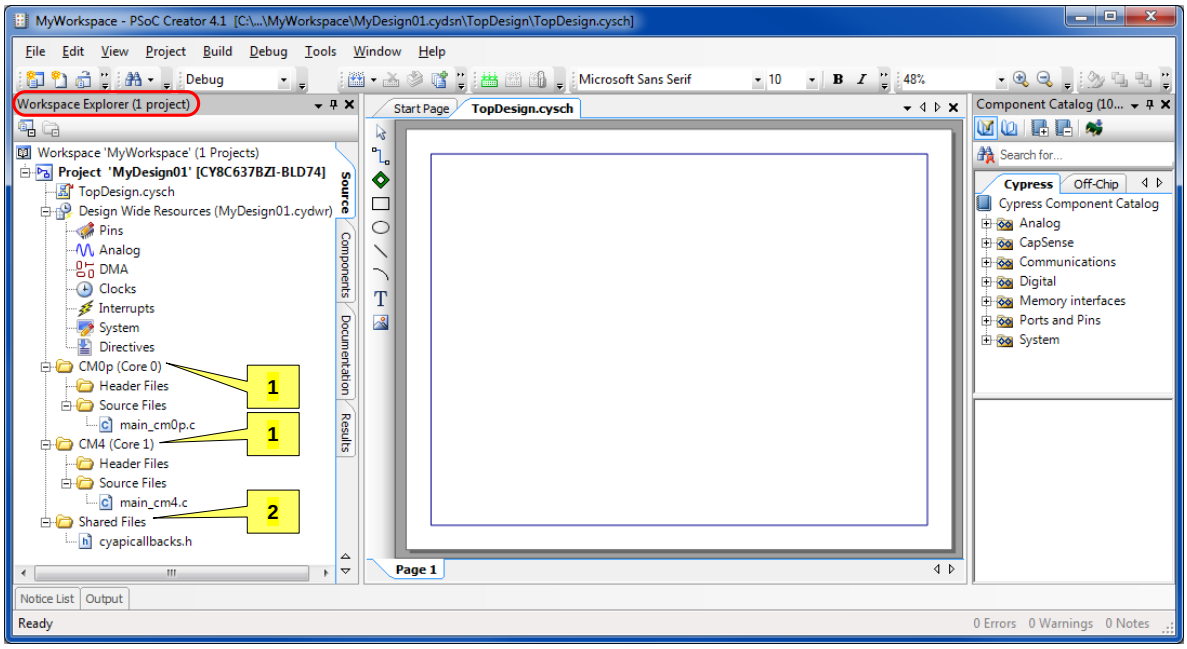


选择 PSoC 6 MCU 器件后，项目创建过程的其余部分与其他器件相同。 单击“**Create Project**”对话框的其余部分; PSoC Creator 创建项目。

初始项目窗口布局（Figure 8）包括一个 Workspace Explorer 窗口，其中包含双 CPU 设备的以下功能：

1. 为每个 CPU 分离 *main.c* 文件 - *main_cm0p.c* 和 *main_cm4.c*。 文件夹 CM0p（Core 0）和 CM4（Core 1）中的源被编译为各个 CPU 的单独二进制文件。
2. Shared Files 文件夹。 此文件夹中的源文件都被编译为二进制文件。

Figure 8. 双 CPU 器件 PSoC Creator 初始项目布局



初始项目布局还包括 TopDesign 硬件原理图以及相关的组件目录窗口。

创建项目后，通过将组件拖动到原理图上并配置和连接它们来实现硬件设计。

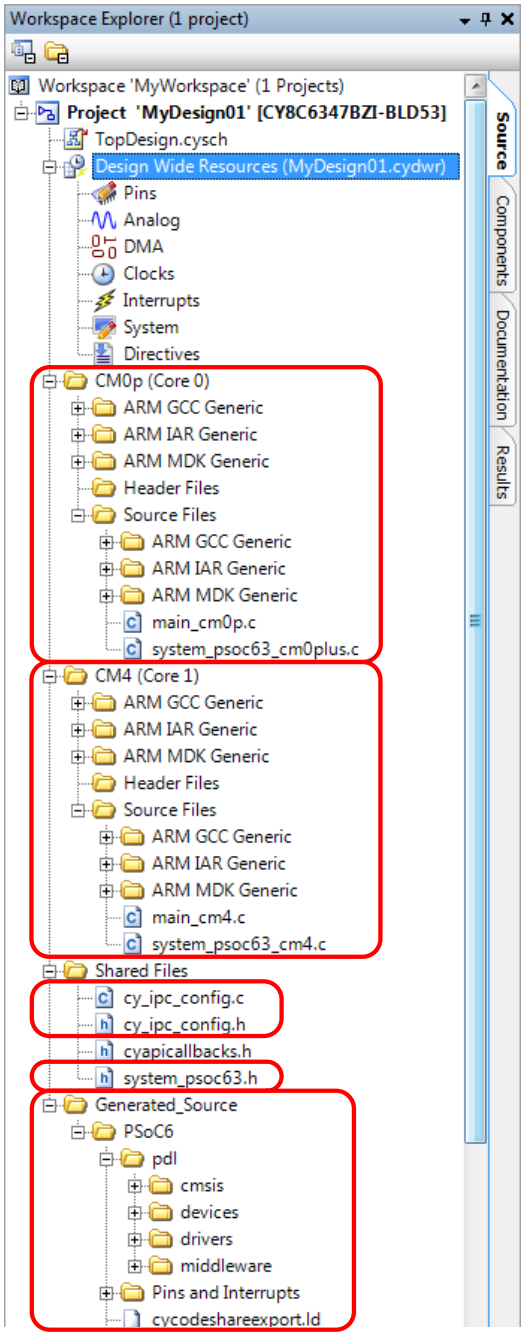
完成原理图设计输入后，选择 **Build** > **Generate Application**。这将在现有文件夹以及新文件夹 Generated Source 中创建多个系统源代码文件和文件夹，如 Figure 9 所示。

生成的源包含原理图上每个 Component 的驱动程序，以及赛普拉斯的外设驱动程序库（PDL）。PDL 是一个软件开发工具包（SDK），它集成了设备头文件，启动代码和外设驱动程序。外设驱动程序将硬件功能抽象为一组易于使用的 API。

有关 PDL 的更多信息，请选择 PSoC Creator 中 **Help** > **Documentation** > **Peripheral Driver Library**。此外，每个组件都有一个数据表，记录了该组件的驱动程序 API。右键单击 **Component** 并选择 **Open Datasheet ...**。

PSoC Creator 会创建其他几个文件和文件夹，并将它们放在现有文件夹 *CM0p（Core 0）*，*CM4（Core 1）*和 *Shared Files* 中。这些文件通常支持 PSoC Creator 以及其他 IDE 的配置，启动和链接选项。有关这些文件的更多信息，请参阅 PSoC Creator 帮助文章 *Generated Files (PSoC 6)*。

Figure 9. 添加生成源文件到项目



4.3 资源分配注意事项

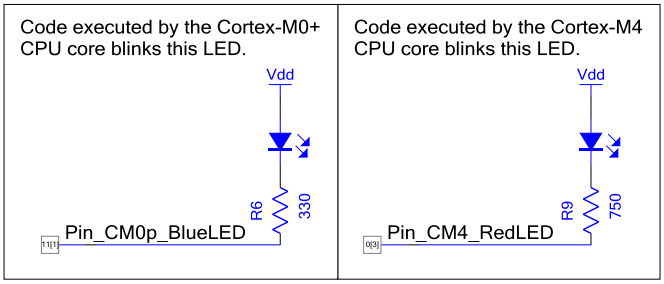
所有 PDL 驱动程序源和其他 API 文件都可供两个 CPU 使用。如果 CPU 的代码引用生成的源文件中的任何 API 元素，则该文件将编译为该 CPU 的二进制文件。可以将同一文件编译到两个二进制文件中 - 请参阅代码示例 CE216795，PSoC 6 MCU 双 CPU 基础知识(Only English)。

如果将相同的源文件编译到两个二进制文件中，则该文件中的函数将在两个二进制文件中重复。即使副本位于单独的版本和二进制文件中，在某些情况下，考虑由两个 CPU 同时执行的功能也很方便。

如前所述，两个 CPU 都可以访问外设；例如，两个 CPU 都可以通过同一个 UART 发送数据。通常，PDL 驱动程序功能是“CPU 安全的”，也就是说，这两个 CPU 可以有效地同时执行这些功能。但是，您应该做出有关为每个 CPU 分配资源的设计决策。有两种方法可以做到这一点：

- **将资源专用于一个 CPU**。包含仅在所需 CPU 的固件中使用资源的代码。此外，如果您使用的是 PSoC Creator，最好的做法是在项目原理图中指明“拥有”资源的 CPU，如 Figure 10 所示。

Figure 10. 双 CPU 控制独立引脚组件的 PSoC Creator 项目原理图



- **在 CPU 之间共享资源**。代码示例 CE216795 显示了 PSoC 6 MCU 的 IPC 模块如何用于实现互斥锁以在 CPU 之间共享存储器。使用相同的技术来共享外设资源，例如 UART。

  在 CPU 二进制文件中分配的闪存和 SRAM 通常与其他 CPU 的闪存和 SRAM 分开。如果在 CPU 的链接描述文件中定义了自定义节和节放置，则必须确保节不重叠。相反，共享内存的另一种方法是为每个 CPU 定义具有相同地址的自定义部分。

  注意：如果两个 CPU 都控制同一 GPIO 端口上的引脚，则仅使用以下 API 函数来更改引脚输出：`Cy_GPIO_Write()`, `Cy_GPIO_Set()`, `Cy_GPIO_Clr()`, 和 `Cy_GPIO_Inv()`。有关更多信息，请参阅 PDL 文档中的 GPIO API 参考。

4.4 中断分配注意事项

双 CPU 设计的一个重要考虑因素是分配和处理中断。如前所述，所有设备中断都可供 CM4 使用，并且中断子集通过多路复用器路由到 CM0 +。您必须决定哪个 CPU 将处理每个中断。

更多信息，请参见应用笔记 AN217666，PSoC 6 MCU 中断 (Only English)。

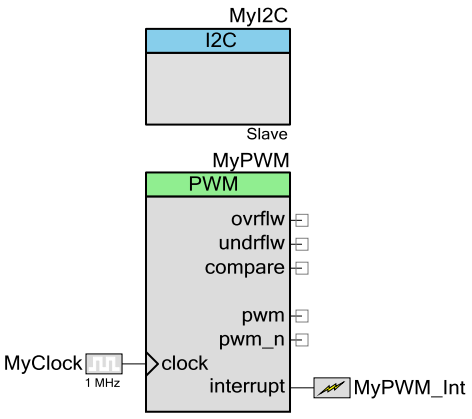
ModusToolbox 软件：在 ModusToolbox 软件中，以编程方式分配中断。目前，还没有 GUI 支持。所需代码的示例在 AN217666 中、在 PDL 文档中 Cypress BLE Middleware Library 部分和配置 BLESS 中断子部分。本小节中的代码显示了如何将 PSoC 6 MCU BLE 子系统（BLESS）中断分配给 CM0 +或 CM4。可以轻松修改代码以支持其他中断源。赛普拉斯 HAL 只支持 CM4，因此 HAL 驱动器所需的所有中断都分配给 CM4。

PSoC Creator：对于 PSoC Creator，让我们在示例设计中分配中断。Figure 11 显示了一个带有两个中断的设计; 一个来自 PWM 组件，连接到中断组件 MyPWM_Int; 另一个来自 I2C 组件。

在 **Design Wide Resources** 窗口（文件类型 *.cydwr*）中，选择 Interrupts 选项卡以查看设计中的所有中断，如 Figure 12 所示。

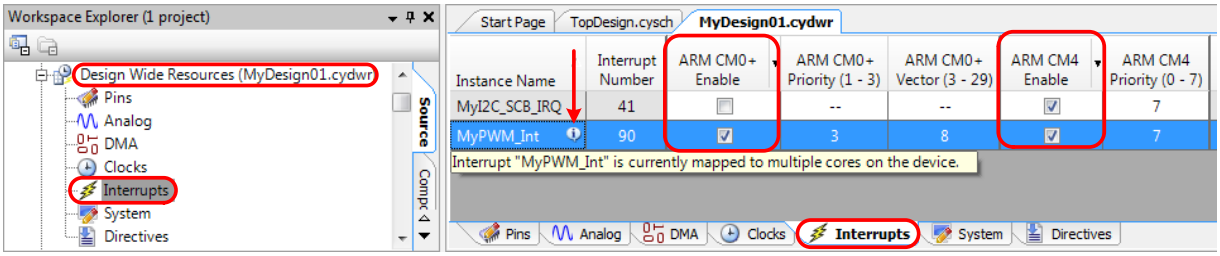
在此示例中，I2C 组件中嵌入了一个中断。 该中断没有显示在 Figure 11 的原理图中; 它在 **Design-Wide Resources** 窗口中显示为 MyI2C_SCB_IRQ 。

Figure 11.两个中断设计示例原理图



选中或取消选中 **ARM CM0 + Enable** 和 **ARM CM4 Enable** 列中的框，以便为各个 CPU 分配中断。

Figure 12. 为 CPU 分配中断



每个外设中断都硬连接到 CM4，因此在构建项目时，PSoC Creator 会自动分配中断号。 由于中断通过多路复用器路由到 CM0 +，因此可以为每个中断选择 **ARM CM0+ Vector**。

注意：如果一个中断分配给了两个 CPU，则会显示警告符号和工具提示。 通常不建议这样做; 但是，可以使用中断将一个或两个 CPU 从其 休眠模式唤醒。

4.5 调试注意事项

第三方 IDE（如 Keil µ Vision 和 IAR Embedded Workbench）支持双 CPU 调试。 Eclipse IDE for ModusToolbox 和 PSoC Creator 支持每次调试单个 CPU（CM4 或 CM0 +）。

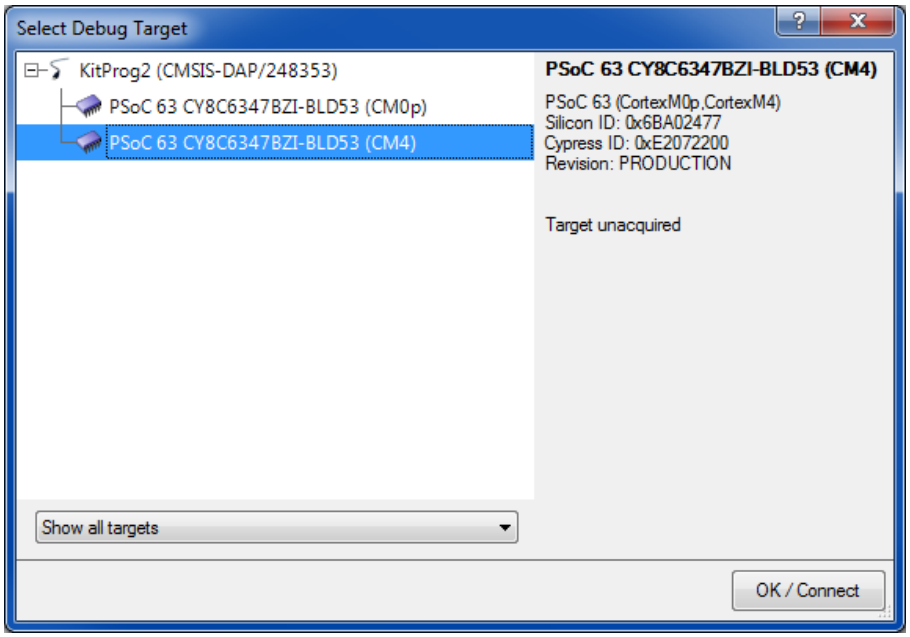
4.5.1 ModusToolbox 说明

Eclipse IDE for ModusToolbox 支持每次调试单个 CPU（CM4 或 CM0 +）。 有关详细说明， 打开 ModusToolbox User Guide 阅读 **PSoC 6 MCU Programming/Debugging** 章节。

4.5.2 PSoC Creator 说明

PSoC Creator 只支持每次调试一个 CPU。 在开始使用 PSoC 6 MCU 进行调试会话之前，请选择所需的调试目标（Debug> Select Debug Target ...），如 Figure 13 所示。 选择所需的 CPU，然后单击“**OK / Connect**”。 要调试其他 CPU，必须退出调试器，然后通过与该 CPU 的连接重新进入。

Figure 13. PSoC Creator 选择 CPU 进行调试



建议：首先开发和调试 CPU 相互通信的代码部分。之后，可以单独调试由单个 CPU 执行的代码。例如，当开发 CE216795 中的共享存储器项目时，CM0 +向 CM4 发送初始消息的部分在开发后续部分代码之前被开发和调试。

您可以使用其他 IDE（例如 µ Vision 或 IAR）同时调试两个 CPU。为此，您必须将 PSoC Creator 项目导出到另一个 IDE。 PSoC Creator 在帮助文章“集成到第三方 IDE，PSoC 6 设计”中介绍了此主题。查看帮助文章中的说明。在以下部分中，给出了总体的步骤。

4.5.3 其他 IDE 的说明

4.5.3.1 从 ModusToolbox 软件导出

ModusToolbox 软件有一个通过 make 文件导出以下 IDE 的机制：

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

关于如何导出和设置这些工具的调试的更多细节，请参阅 ModusToolbox User Guide 中的 " 导出到 IDE "一章。

所有这些 IDE 都支持双 CPU 调试；但只有 Keil µVision 的双 CPU 调试能与通过 ModusToolbox 导出的项目一起使用。

ModusToolbox 用户指南上的说明是为编程/调试 CM4 而设计的。同样的说明适用于 CM0+，但有一些变化。根据你所使用的 IDE，执行以下额外的步骤。

1. **Visual Studio Code 中打开 CM0+项目**

为 CM0+项目执行 make vscode 命令后，会创建一个名为".vscode "的新文件夹。在该文件夹中，打开 launch.json 文件并做以下修改：

a. 将 CM4 的所有实例重命名为 CM0+。

b. 将所有 targetProcessor 实例设置为 0（而不是 1）。

2. **Keil µVision 中打开 CM0+项目**

按照为 CM4 创建项目的相同步骤进行。为了进行双 CPU 调试，打开两个 Keil µVision 的实例，一个是 CM0+项目，另一个是 CM4 项目。在这两个实例中，进入 **Debug > Start/Stop Debug Session**。

3. **IAR Embedded Workbench**

ModusToolbox 中的 IAR 导出机制不支持 CM0+项目。你只能导出 CM4 项目。

4.5.3.2 Exporting from PSoC Creator

如果您使用的是 PSoC Creator，则可以将项目导出到 Keil µ Vision 或 IAR Embedded Workbench 以进行双 CPU 调试。以下是执行此操作的步骤：

1. 配置 PSoC Creator 项目
2. 创建 µVision 项目
3. 调试 µVision 项目
4. 创建 IAR-EW 项目
5. 调试 IAR-EW 项目

1. 配置 PSoC Creator 项目

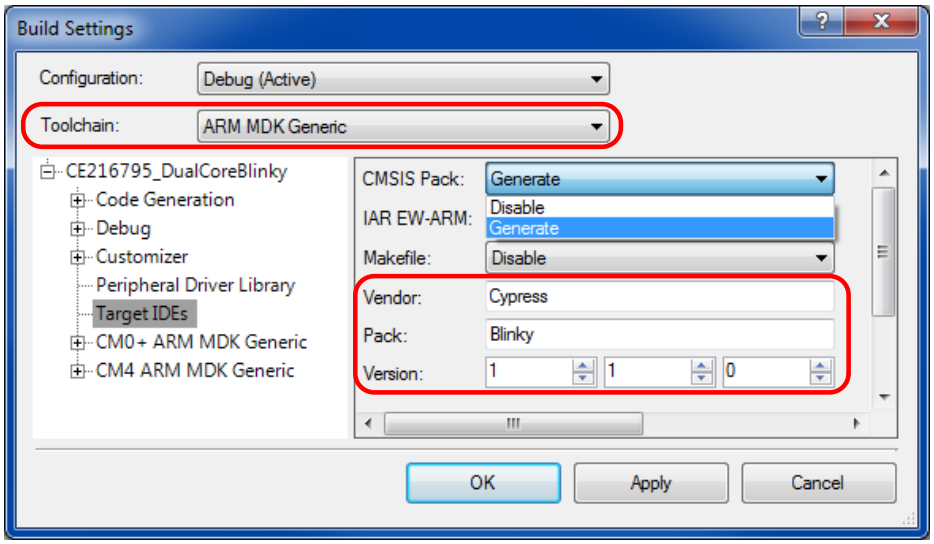
更新项目 Build Settings 中的 Target IDEs 设置，如 Figure 14 所示。

对于 µVision，请选择 **CMSIS Pack: > Generate**。在 Vendor，Pack 和 Version 字段中输入 CMSIS 包的相应标识文本。

推荐：选择 **Toolchain: > ARM MDK Generic**。

对于 IAR，您只需选择 **IAR EW-ARM: > Generate**。 （也可以使用高级选项，**Generate without copying PDL files**。）由于 IAR 有自己的编译器（PSoC Creator 不支持），因此工具链选择无关紧要。

Figure 14. 目标 IDE 构建设置

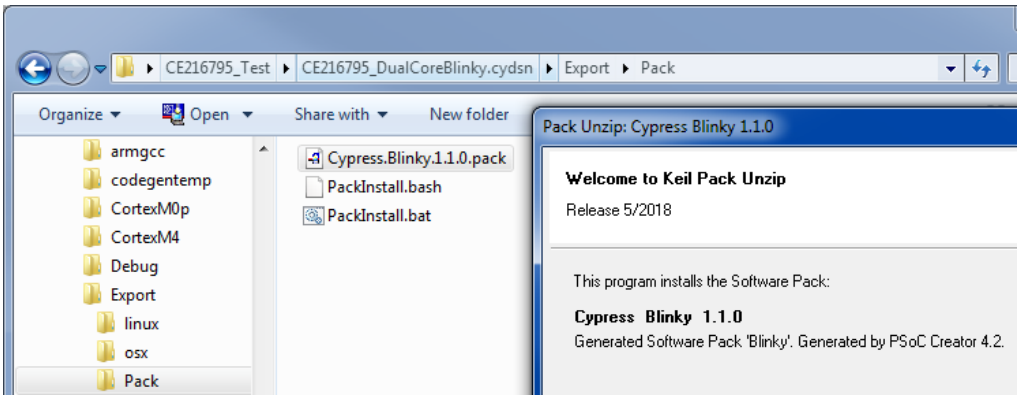


然后以常用的方式构建 PSoC Creator 项目。 将在*<project> .cydsn* 文件夹中创建一个 *Export* 文件夹，其中包含用于导出到所选 IDE 的相关文件。

对于 µVision，在构建 PSoC Creator 项目后，在 *Export \ Pack* 文件夹中找到相应的*.pack* 文件。 双击该文件将其安装为 µVision 包，如 Figure 15 所示。

注意：请勿使用 µVision Pack Installer 向导文件导入功能来安装此包。

Figure 15. 从 PSoC Creator 项目安装 µVision 包



注意：如果更新 PSoC Creator 项目，请考虑更改 µVision 包版本号（参见 Figure 14）并安装新数据包。

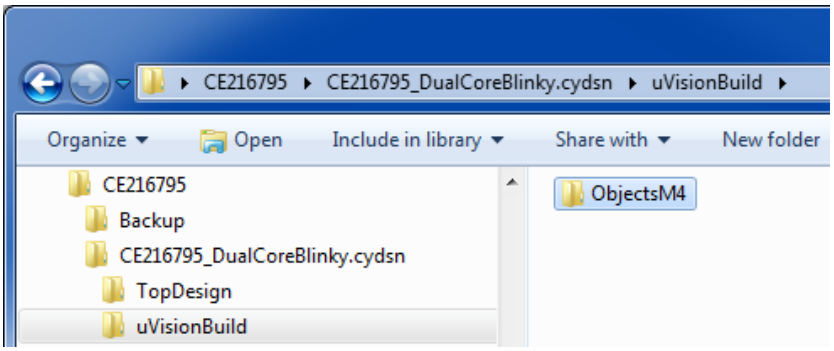
有关更多信息，请参阅 AN219434 - 将 PSoC Creator 代码导入到 PSoC 6 项目的 IDE 中

2. 创建 µ Vision 项目

对于 µ Vision，您必须创建两个项目：每个 PSoC 6 MCU CPU 一个：CM0 +和 CM4。 请执行下列操作：

建议：在 PSoC Creator *<project> .cydsn* 文件夹中创建一个新文件夹（例如 uVisionBuild），以便将所有 µ Vision 项目文件与 PSoC Creator 文件分开存储（这与 IAR instructions 不同）。 在该文件夹中，为 CM4 目标文件（例如，ObjectsM4）创建另一个新文件夹，如 Figure 16 所示：

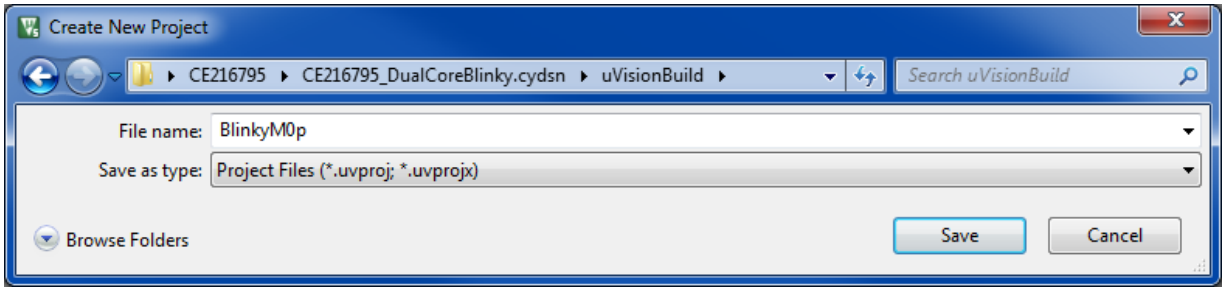
Figure 16. µVision 项目的新文件夹



打开 µ Vision 5.25 或更高版本，并在 uVisionBuild 文件夹中创建一个新项目（**Project** > **New µVision Project...**）。

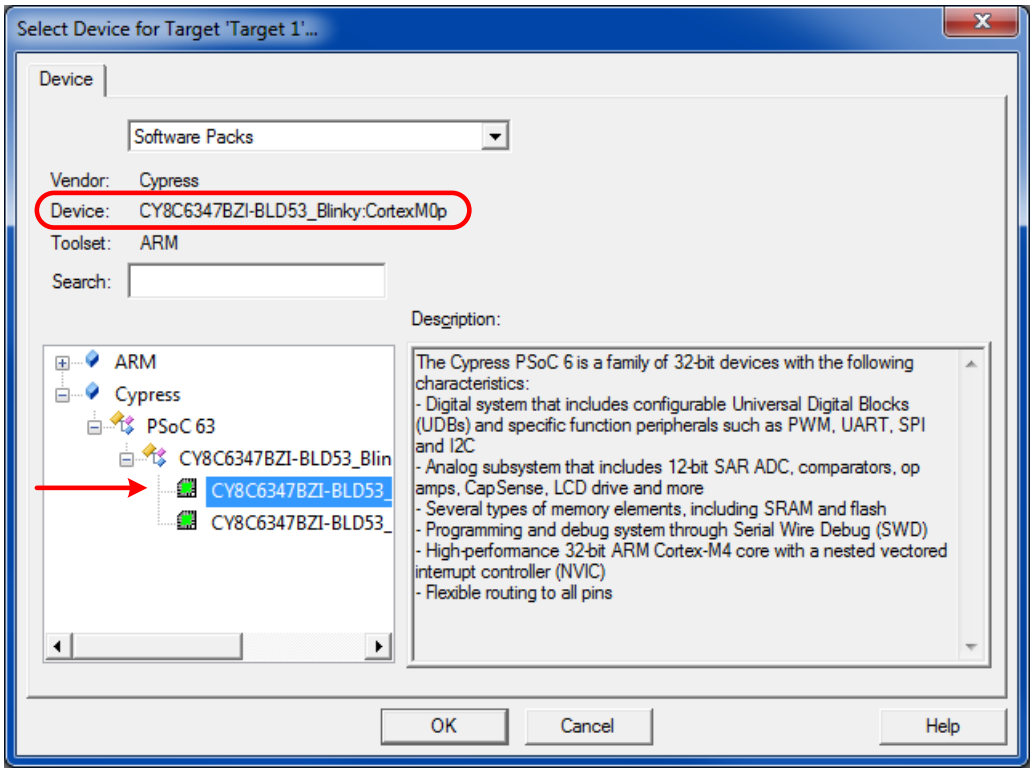
建议：根据原始 PSoC Creator 项目名称和目标 CPU 命名项目。 例如，对于 CE216795 双 CPU 闪烁项目，为 CM0 + CPU 创建一个 µ Vision 项目 BlinkyM0p，如 Figure 17 所示：

Figure 17. 为 CM0+创建 µVision 项目



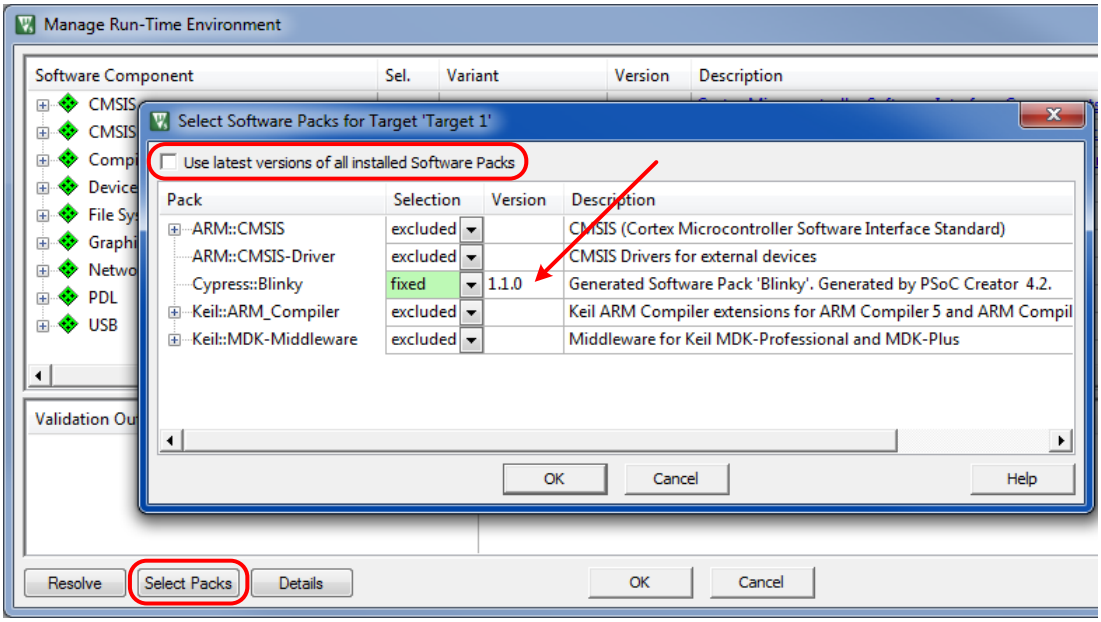
单击“**Save**”后，将显示“**Select Device for Target 'Target 1'...**”对话框。 将显示先前安装的包中定义的两个 PSoC 6 MCU CPU（Figure 15）。 选择 CM0 + CPU，如 Figure 18 所示。 单击 **OK**。

Figure 18. 选择 CM0+ 作为项目设备



接下来，将显示“**Manage Run-Time Environment**”对话框。 单击“**Select Packs**”，然后取消选中“**Use latest versions of all installed Software Packs**”。 从 PSoC Creator 项目中选择包，如 Figure 19 所示：

Figure 19. 选择 PSoC Creator 项目包



单击 **OK**; Manage Run-Time Environment 对话框的更改如 Figure 20 所示。 选择“Device Startup”和“PDL Drivers”，然后单击“**OK**”。 创建项目，包括目标 1，源组 1 以及设备启动和 PDL 文件，如 Figure 21 所示。

Figure 20. 选择数据包启动和 PDL 驱动文件

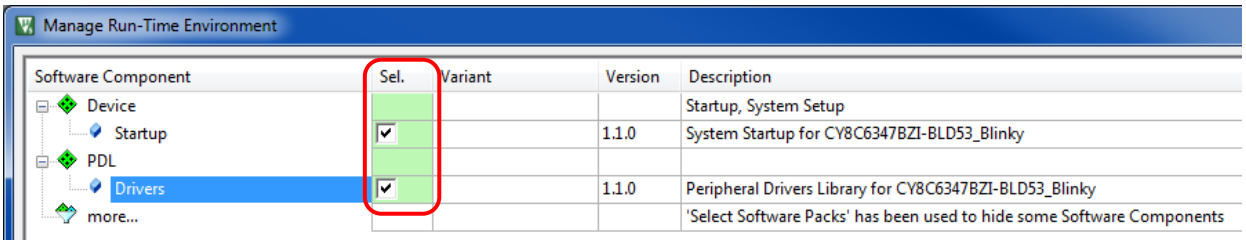
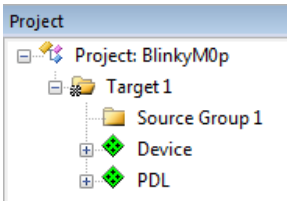
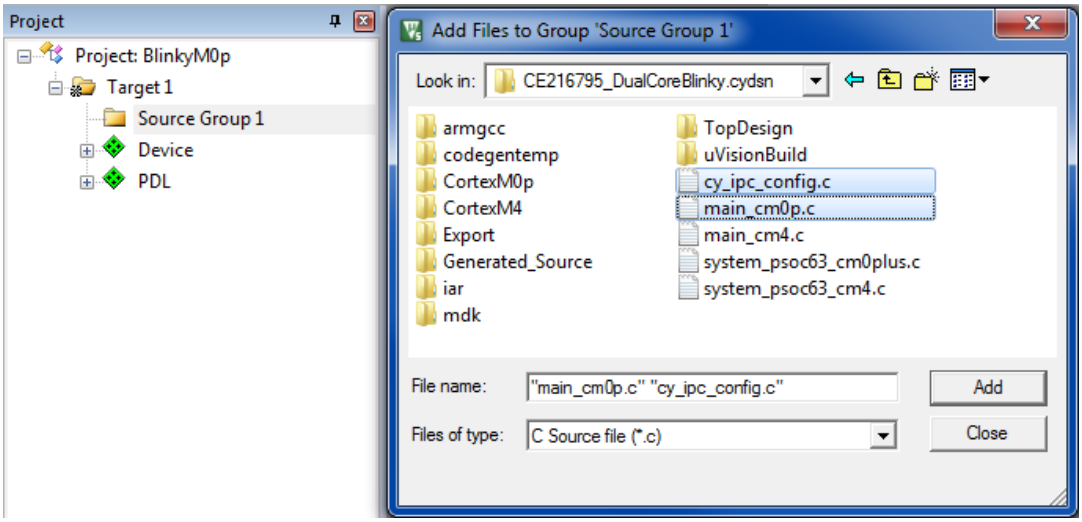


Figure 21. 初始项目创建



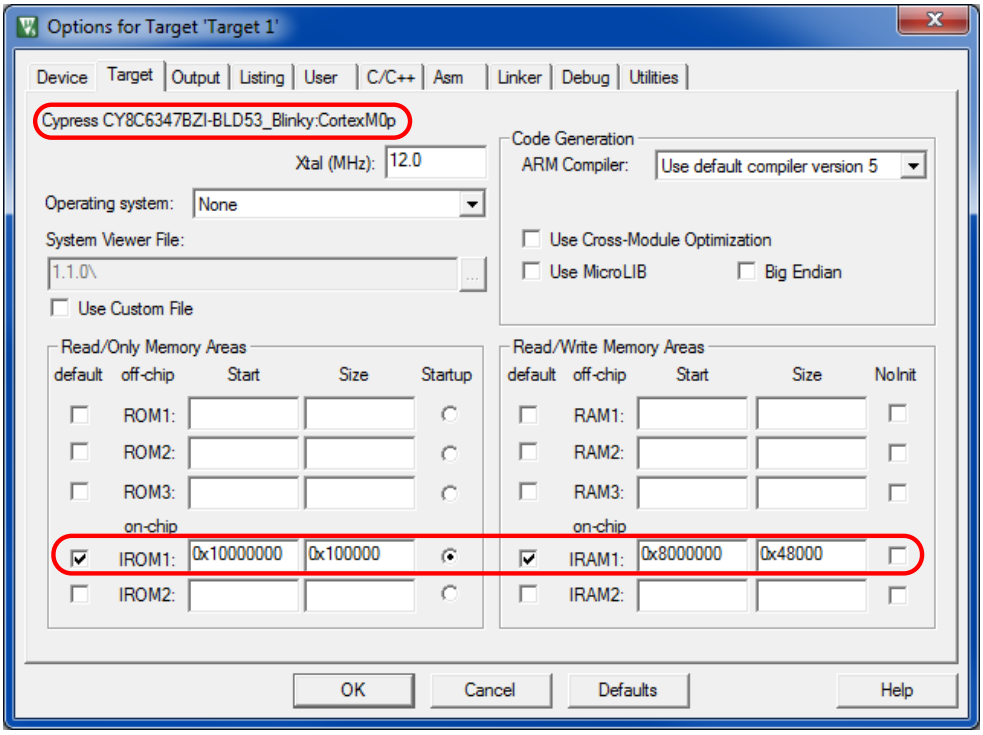
右键单击 **Source Group 1**，然后选择 **Add Existing Files to Group'Source Group 1'....**，导航到 PSoC Creator 项目文件夹并选择您的项目所需的 *main_cm0p.c*，*cy_ipc_config.c* 和所有其他非系统.*c* 和汇编程序文件，如 Figure 22 所示。 您不必添加任何.*h* 文件，启动或系统.*c* 或汇编程序文件。 单击 **Add**; 文件将添加到 µVision 项目中的源组。 单击 **Close**。

Figure 22. 将 PSoC Creator 项目 C 源文件添加到源组



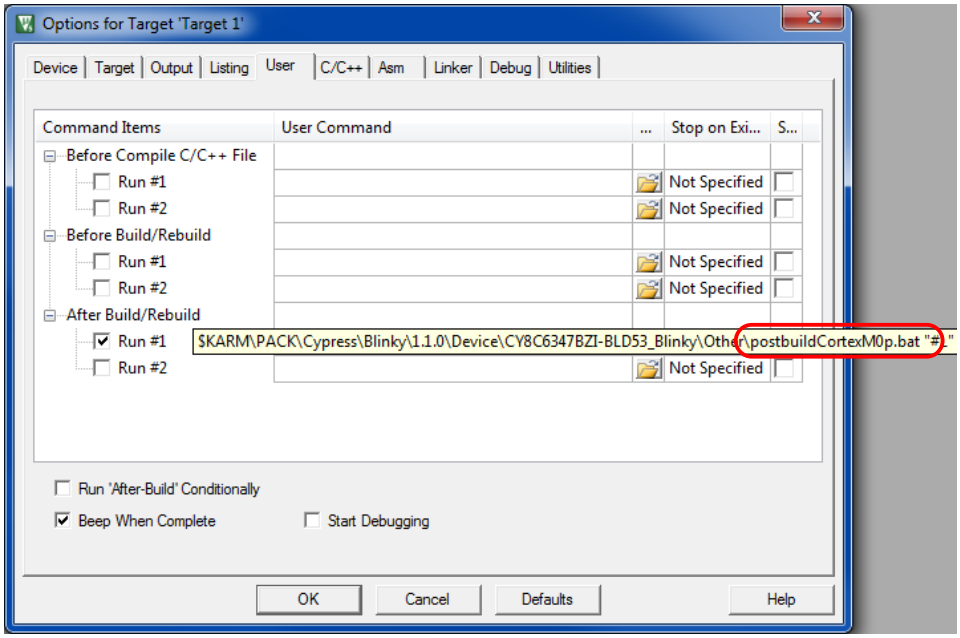
现在项目已创建，您必须设置其选项。 右键单击 **Target 1**，然后选择 **Options for Target 'Target 1'...**。在 Target 选项卡中确认设备 CPU、IROM1 和 IRAM1 对于 PSoC 6 MCU 器件是正确的，如 Figure 23 所示。 更新其他字段，比如 Xtal（MHz）和操作系统，是可选的。

Figure 23.项目目标选项



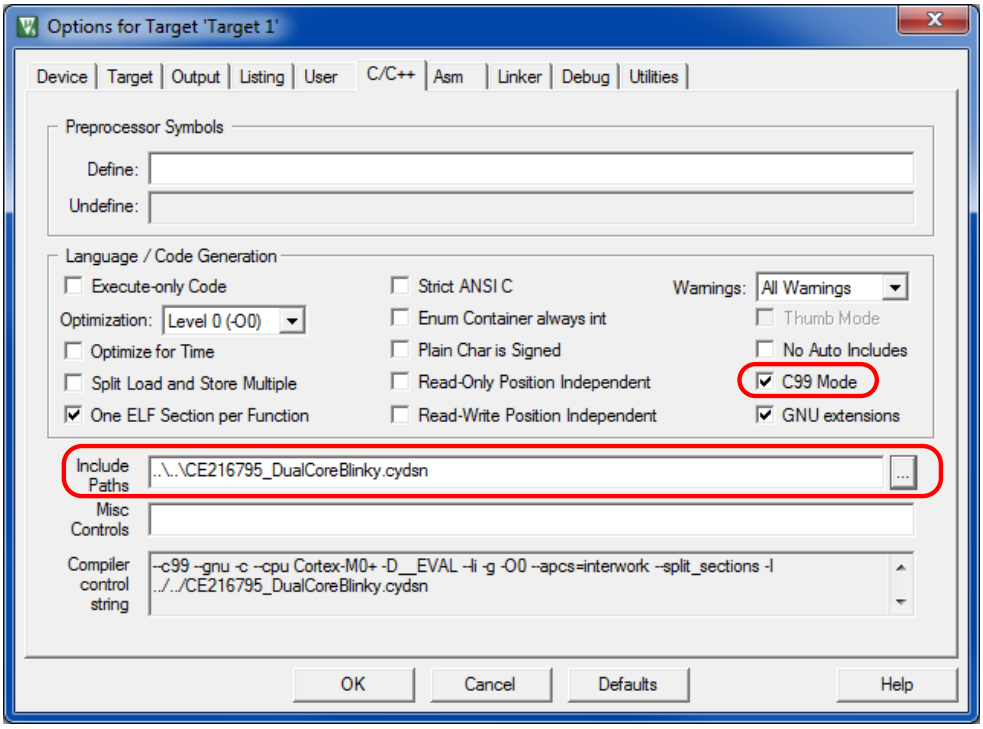
在“**User**”选项卡中，验证是否正在调用数据包中正确的编译后批处理文件。 将光标悬停在 **User Command** 字段上并确认调用了 *postbuildCortexm0p.bat*，如 Figure 24 所示。 添加其他预编译和编译后批处理文件，并根据需要选择其他选项。

Figure 24. 项目用户选项



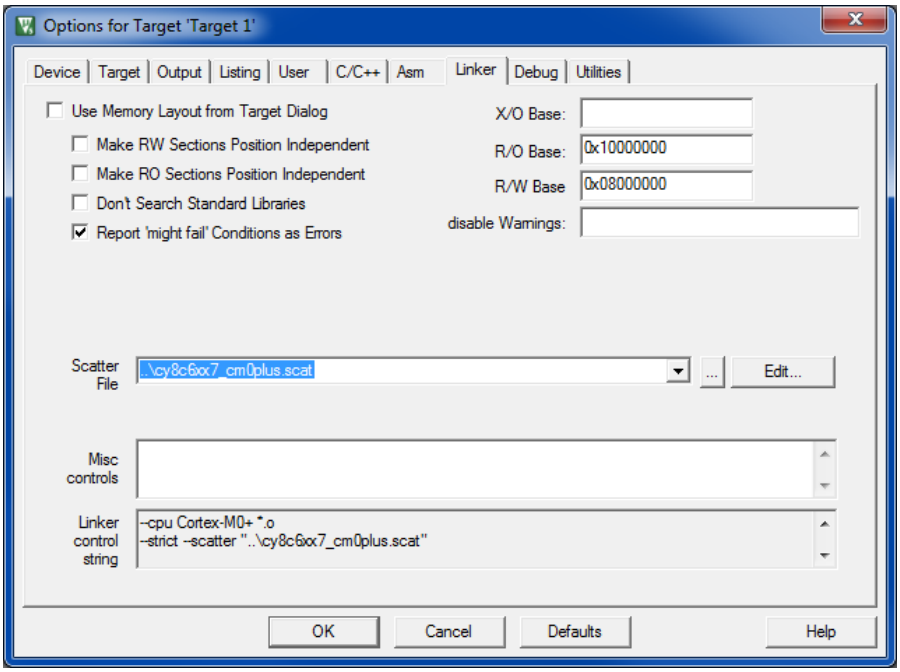
在 **C/C ++**选项卡中确认已选中 **C99 mode** 选项，如 Figure 25 所示。 （PDL 是基于 C99 开发的。）将 PSoC Creator *<project> .cydsn* 文件夹添加到 **Include Paths**; 这提供了 PSoC Creator 项目中*.h* 文件的链接。 根据需要更新其他选项和字段。

Figure 25. 项目 C/C ++选项



在 **Linker** 选项卡中确认 **R/O Base** 和 **R/W Base** 字段对于 PSoC 6 MCU 器件是否正确，如 Figure 26 所示。 从 PSoC Creator 项目文件夹中选择适当的 **Scatter File**。

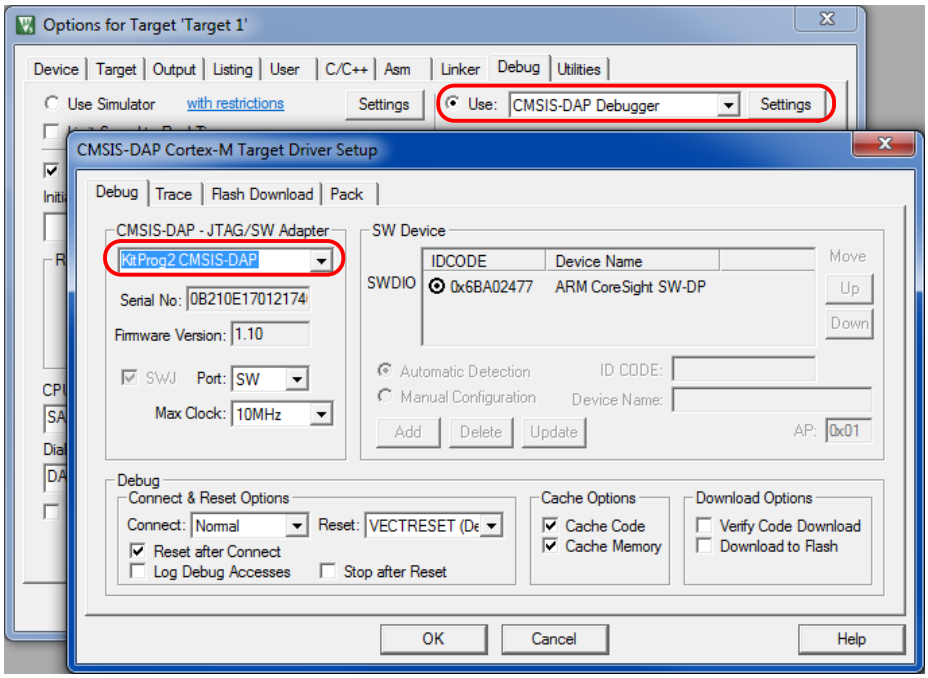
Figure 26. 项目链接器选项



将 CY8CKIT-062-BLE USB 端口连接到计算机。 按套件按钮 SW3 将 KitProg2 置于 CMSIS-DAP 模式。有关详细信息，请参阅套件指南 这允许在不使用任何外部探针的情况下进行调试。

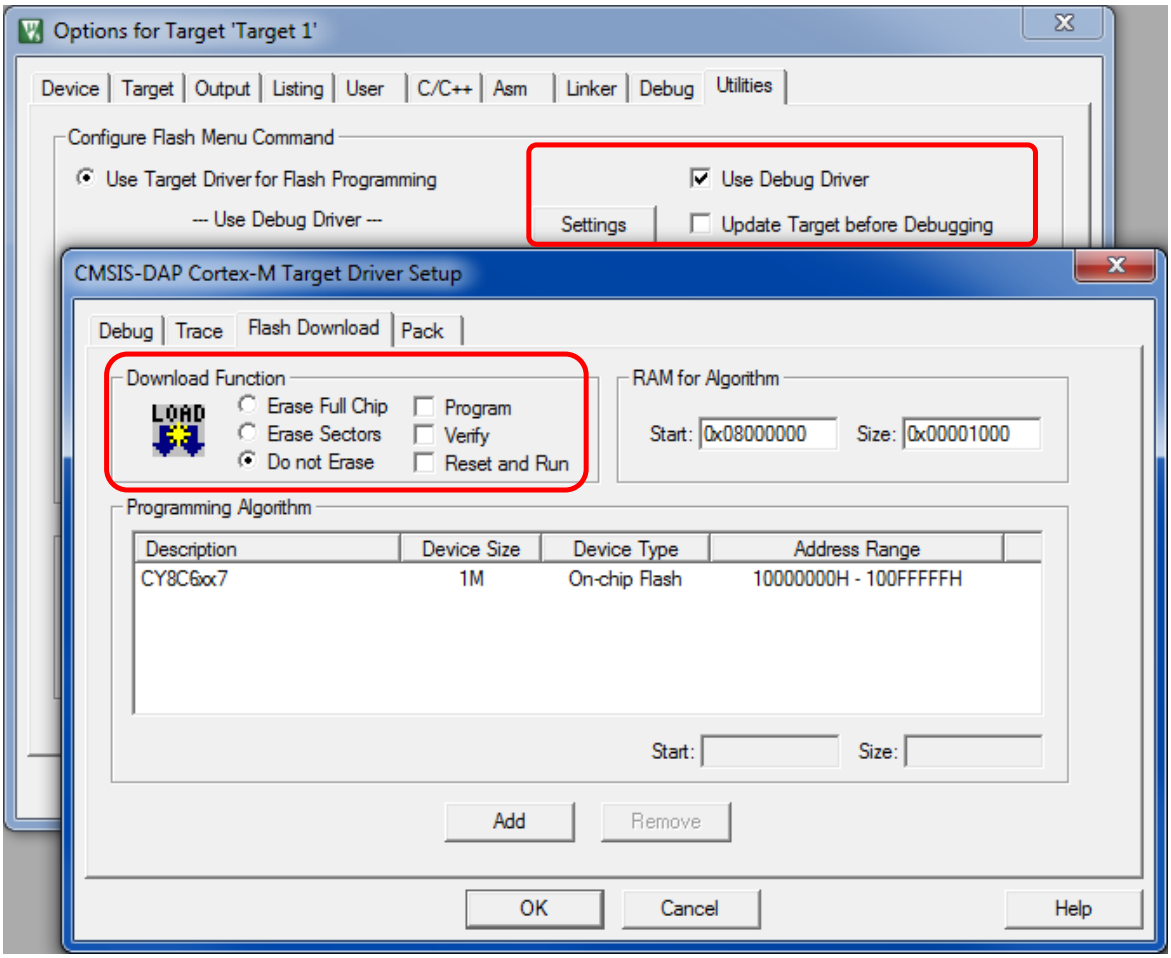
在 **Debug** 选项卡中，选择 **Use CMSIS-DAP Debugger**，如 Figure 27 所示。 单击“**Settings**”，选择“**KitProg2 CMSIS-DAP**”，然后确认所有其他设置均为默认值。 单击“**OK**”，然后返回“Options”对话框。

Figure 27.项目调试选项



在“**Utilities**”选项卡中，确认已选中“Use Debug Driver”，然后在调试前取消选中“**Update Target before Debugging**”。 单击“**Settings**”，然后取消选中所有“**Download Function**”框，如 Figure 28 所示。 单击“**Do not Erase**”。 单击“**OK**”，然后返回“**Options**”对话框。 显示警告“Nothing to do ...”；单击 **OK**。 该应用程序将由 CM4 项目加载。 单击“**OK**”保存并关闭选项设置。

Figure 28. 项目工具选项

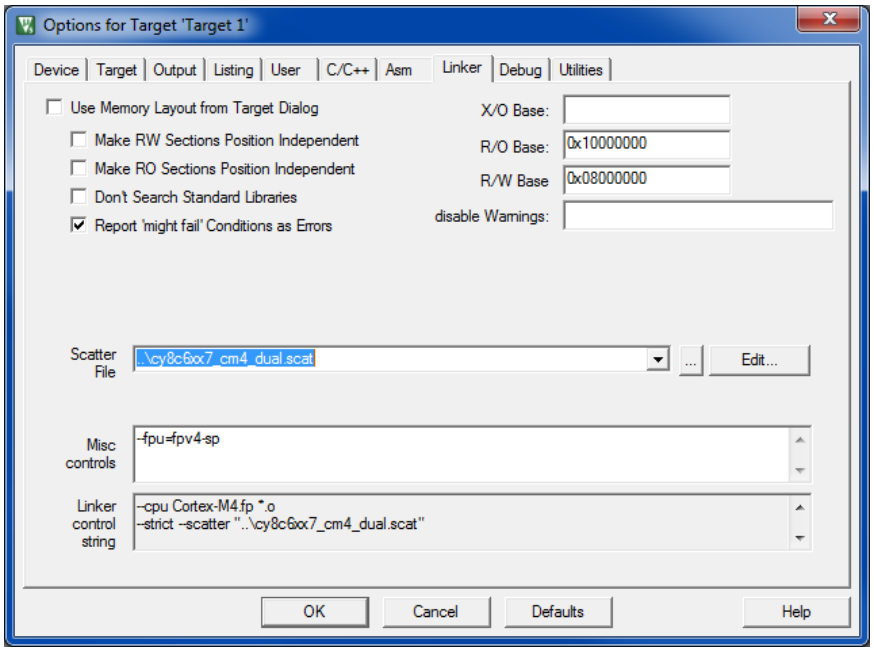


重复上述步骤，为 CM4 创建第二个项目。

建议：根据原始 PSoC Creator 项目名称和目标 CPU 命名项目。例如，对于 CE216795 双 CPU 闪烁项目，创建 µ Vision 项目 *BlinkyM4p*;请参见 Figure 17。以与 CM0 +项目相同的方式配置项目，但有以下区别：

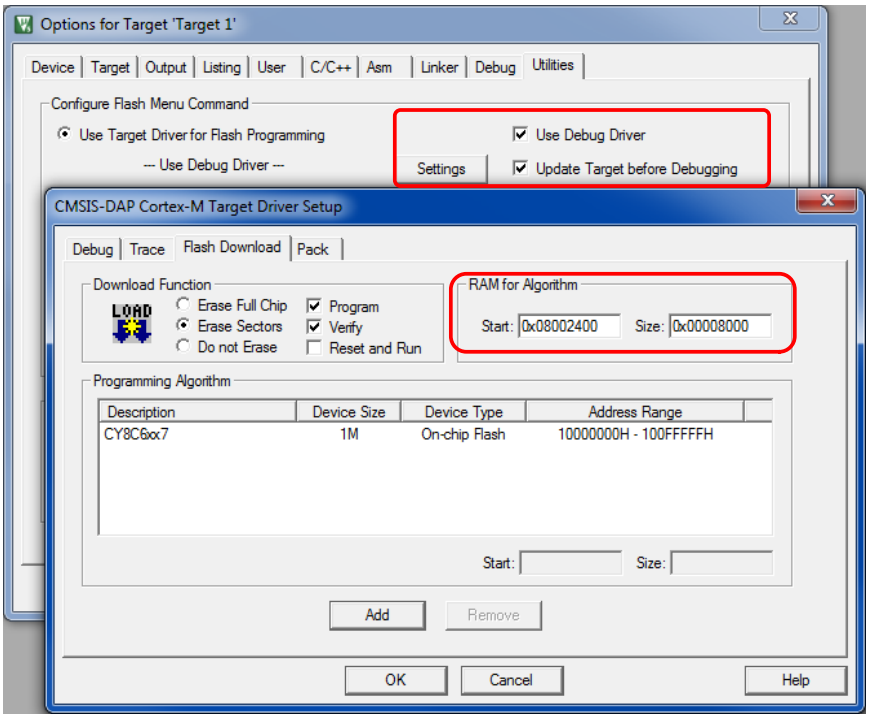
- CM4 项目必须与 CM0 +项目位于同一文件夹中，此处为 uVisionBuild。见 Figure 17。
- 从以前安装的包中选择 CM4 CPU;见 Figure 18。
- 导航到 PSoC Creator 项目文件夹，选择 *main_cm4.c*，*cy_ipc_config.c* 以及项目所需的所有其他非系统.*c* 和汇编程序文件，如 Figure 22 所示。您不必添加任何.*h* 文件、启动文件、系统.*c* 或汇编程序文件。
- 在“**Options**”对话框的“**Output**”选项卡中，单击“**Select Folder for Objects...**”，然后选择您创建的 ObjectsM4 文件夹;见 Figure 16。
- 在 **Options** 对话框的 C/C ++选项卡中，将`--fpu=fpv4-sp` 添加到 **Misc Controls**;见 Figure 25。
- 在“**Options**”对话框的“**Linker**”选项卡中，选择“cm4_dual”分散文件，如 Figure 29 所示。 CM4 项目将包含两个 CPU 的代码。将`--fpu=fpv4-sp` 添加到 **Misc Controls**。

Figure 29. CM4 项目连接器选项



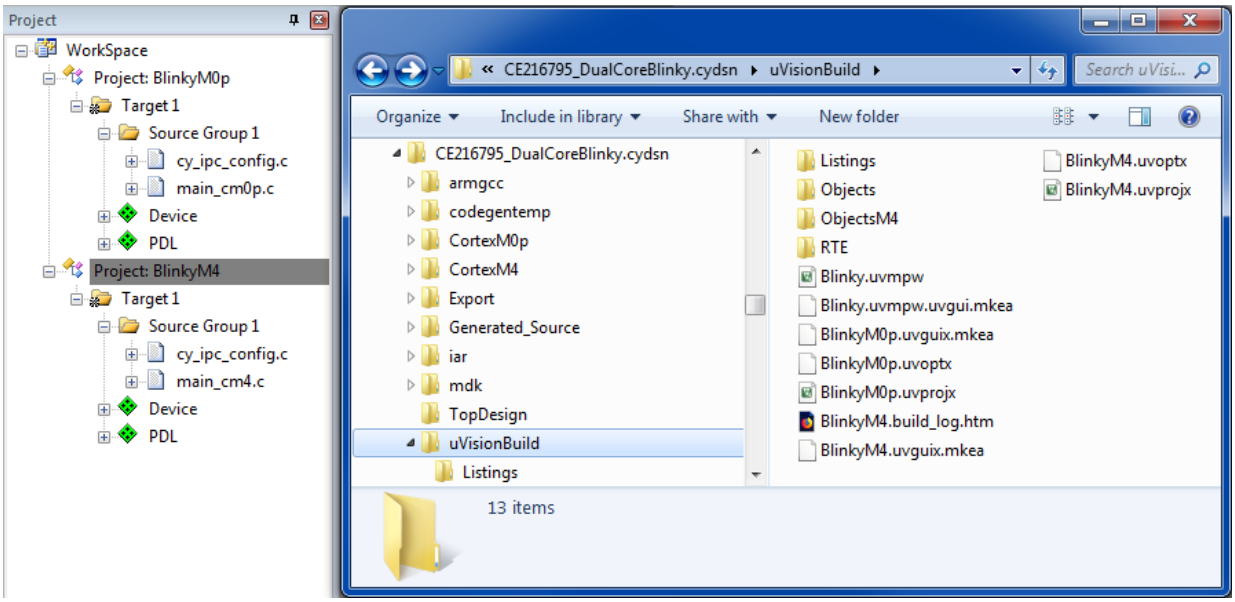
- 在 **Options** 对话框的 **Debug** 选项卡，Target Driver Setup(目标驱动程序设置)中，为 **Reset** 选项选择 VECTRESET; 见 Figure 27。
- 在 **Options** 对话框的 **Utilities** 选项卡中，确认选中 **Update Target before Debugging**，如 Figure 30 所示。 如图所示，为算法值设置 RAM。 检查 **Reset and Run** 是可选的但是方便的。

Figure 30. CM4 项目工具选项



最后，在 uVisionBuild 文件夹中创建一个 µVision 工作区（Project> New Multi-Project Workspace ...），命名为样例 *Blinky*。 将两个创建的项目添加到该工作区。 创建的工作空间和项目以及相应的文件应类似于 Figure 31。

Figure 31. 最终 µVision 项目窗口和项目文件



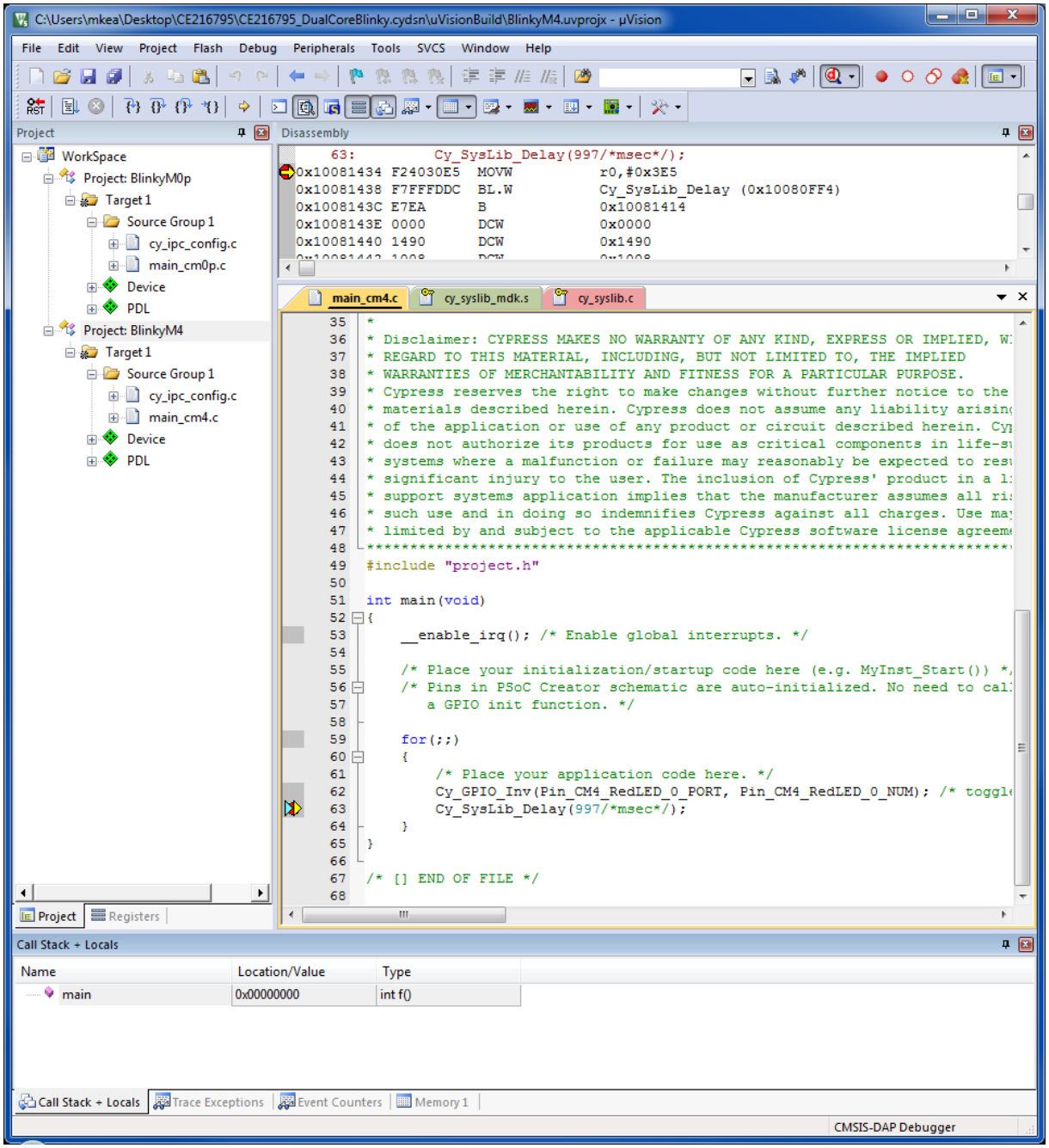
按顺序构建项目; 首先构建 CM0 +项目。 请注意， µVision 具有批量构建功能以自动执行该过程。 构建成功完成后，右键单击 BlinkyM4 项目并将其设置为活动项目。 然后通过（1）擦除闪存（**Flash > Erase**）和（2）下载项目（**Flash > Download**）并确认正确操作来测试构建选项。 如果未选择 Reset and Run（参见 Figure 30），则必须按下套件重置按钮（RST/SW1）才能开始操作。

注意：如果更改 CM0 +项目中的任何代码，则必须重建这两个项目。 请注意， µVision 具有批量构建功能以自动执行该过程。

3.调试 µ Vision 项目

从 CM4 项目开始调试 - 下载 CM4 项目会安装两个 CPU 的代码。 将 CM4 项目设置为活动项目，根据需要下载，然后单击 **Debug** > **Start/Stop Debug Session** 开始调试。 µ Vision 窗口显示类似于 Figure 32：

Figure 32. CM4 调试窗口



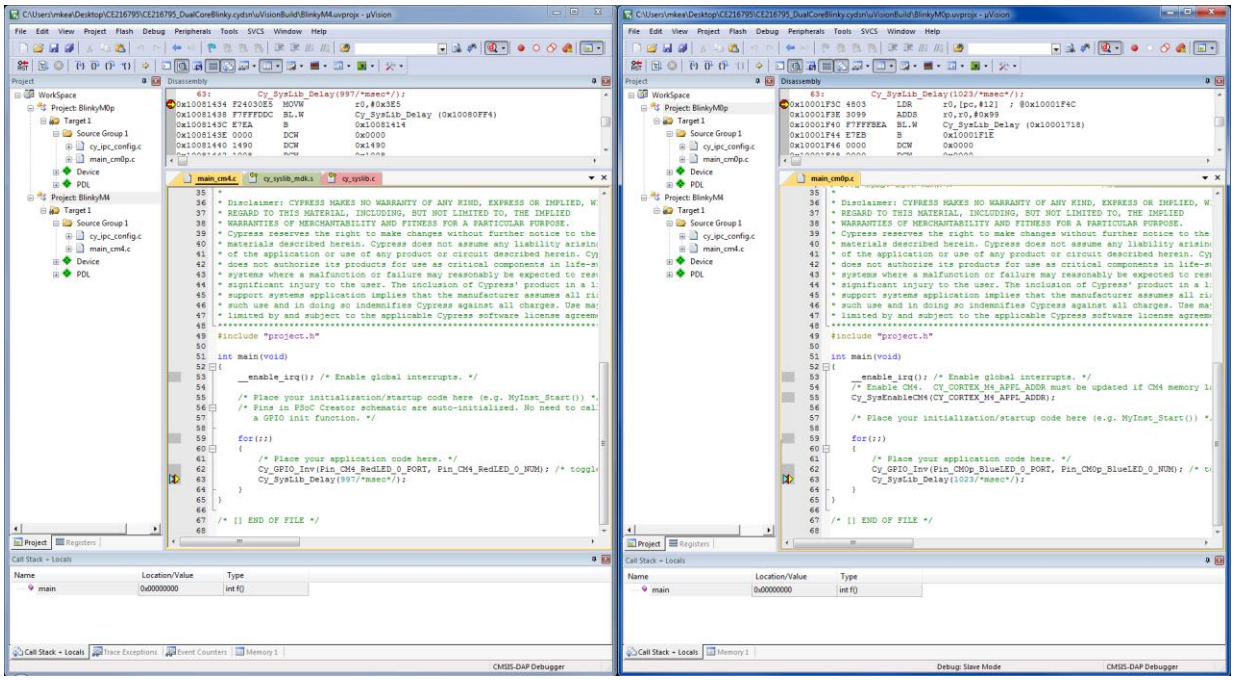
如果您正在运行 CE216795 双 CPU 闪烁项目，请在第 63 行 `Cy_Syslib_Delay()`设置断点。然后重复单击 **Debug**> **Run**，红色 LED 在断点处的每个停止点上切换。

现在打开 µ Vision 的第二个实例并加载相同的工作区。两个实例共享套件连接和 PSoC 6 MCU 调试访问端口（DAP）。使 CM0 +项目处于活动状态，然后启动调试会话。在第 63 行 Cy_Syslib_Delay（）设置断点。然后重复单击 **Debug> Run**，蓝色 LED 在断点处的每个停止点上切换。

注意：在第 55 行执行 `Cy_SysEnableCM4()`函数调用会导致 CM4 再次开始运行。转到 CM4 窗口，单击 **Debug> Stop**，然后单击 **Debug> Run**。 CM4 再次运行到断点。

它有助于将实例窗口并排放置在桌面上。窗口显示类似于 Figure 33。单击相应的窗口以在所需的 CPU 上执行调试操作。请注意，可以为每个 CPU 单独设置断点。您可以从任一窗口读取和更新相同的内存地址。

Figure 33. µVision 双 CPU 调试



4.创建 IAR-EW 项目

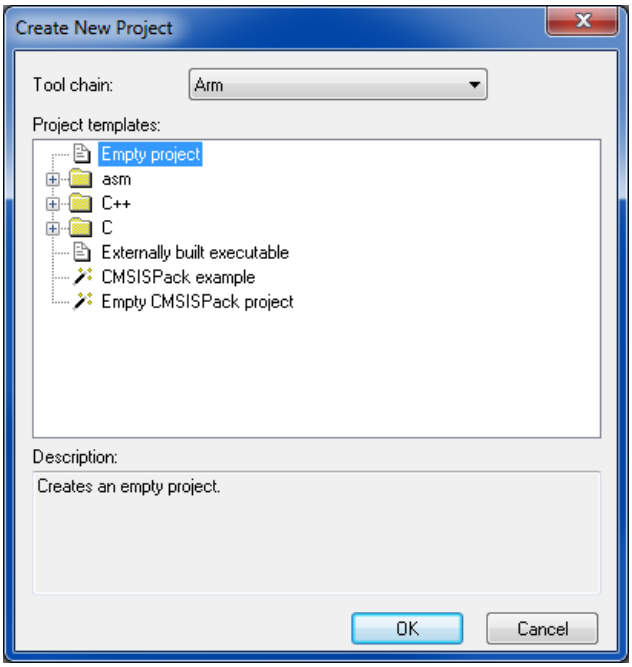
对于 IAR Embedded Workbench（IAR-EW），您必须创建两个项目：CM0 +和 CM4 每个 PSoC 6 MCU CPU 一个项目。 请执行下列操作：

注意：应在 PSoC Creator *<project> .cydsn* 文件夹中创建 IAR-EW 项目文件。 不要在 PSoC Creator *<project> .cydsn* 文件夹中创建单独的文件夹（这与 µVision instructions 不同）。

建议：在每个项目和工作区文件名中 添加“IAR_”等标记，以区分 IAR-EW 文件和同一文件夹中的 PSoC Creator 文件。

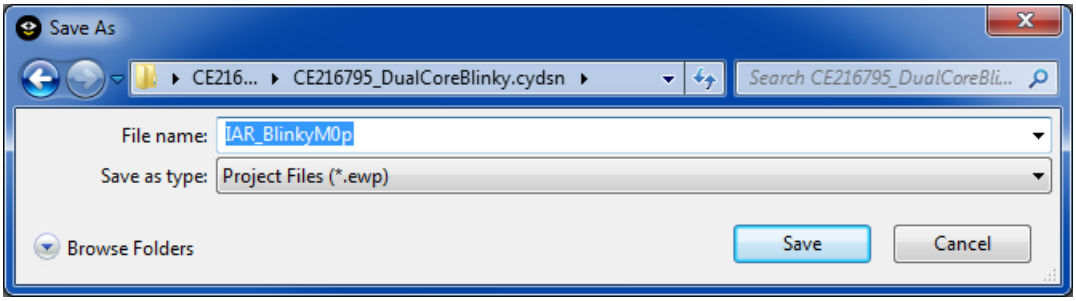
打开针对 ARM 8.22 或更高版本的 IAR Embedded Workbench，并创建一个新项目（**Project> Create New Project ...**）。 在 Create New Project 对话框中（Figure 34），确认 Tool chain 是 Arm，选择 **Empty project** 模板，然后单击 **OK**。

Figure 34. IAR Embedded Workbench 创建新项目对话框



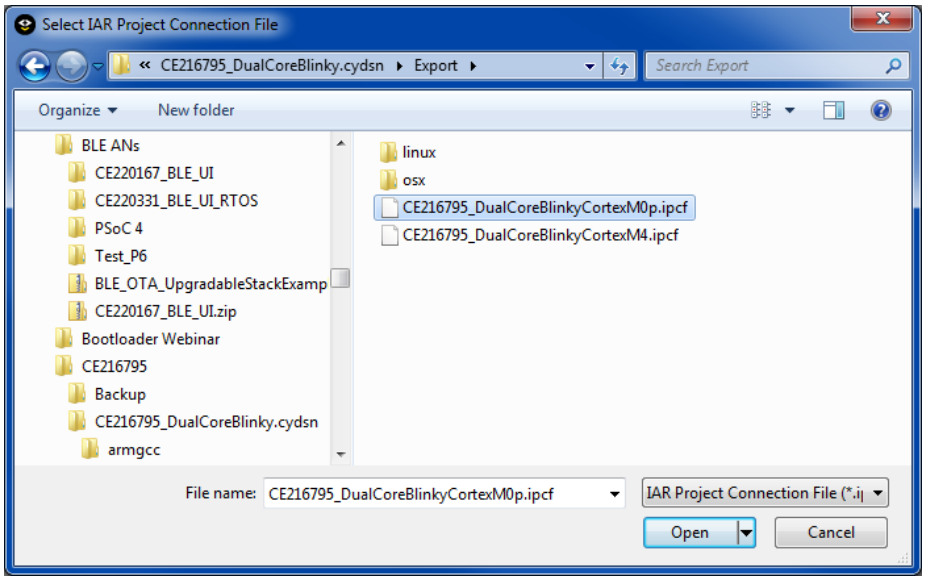
建议：在“Save As”对话框中（Figure 35），根据原始 PSoC Creator 项目名称和目标 CPU 命名项目。 例如，对于 CE216795 双 CPU 闪烁项目，为 CM0 + CPU 创建一个 µVision 项目 IAR_BlinkyM0p。

Figure 35. 为 CM0 +创建 IAR Embedded Workbench 项目



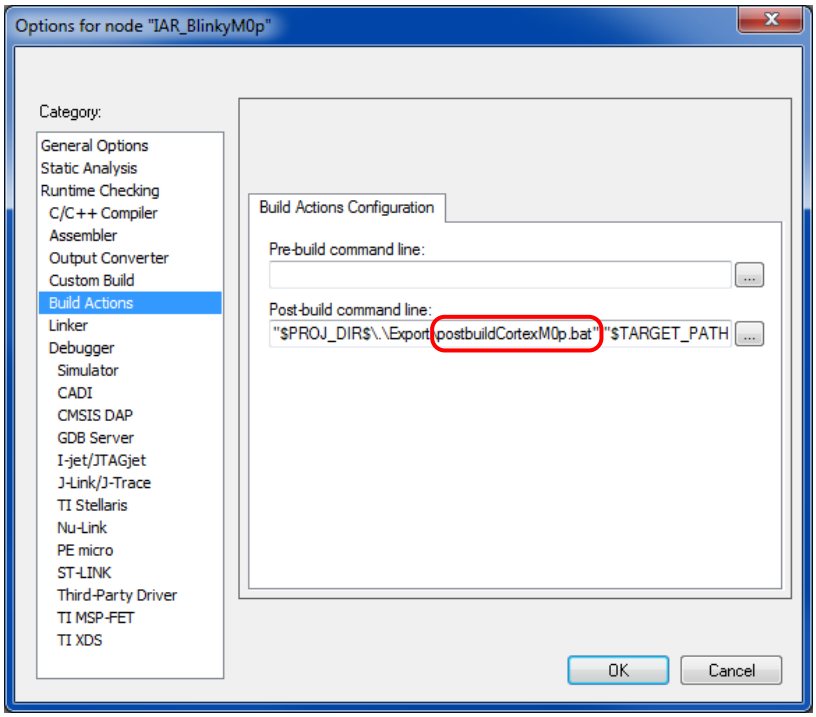
选择 **Tools** > **Options**，确保选中 **Enable project connections**。 单击 **OK**。 然后选择 **Project** > **Add Project Connection...**。在下一个对话框中，选择使用 IAR Project Connection 连接，然后单击 **OK**。 然后选择... *CortexM0p.ipcf* 文件，如 Figure 36 所示。 单击 **OK**，在工作区窗口中将多个文件夹和文件添加到项目中。

Figure 36. 从 PSoC Creator 项目导出文件夹中选择 IAR 项目连接文件



现在项目已创建，必须设置其选项。 右键单击该项目，然后选择 **Options ...**。在 Options 对话框 **Build Actions** 部分中确认调用了 *postbuildCortexM0p.bat*，如 Figure 37 所示。

Figure 37.选择 PSoC Creator 编译后批处理文件

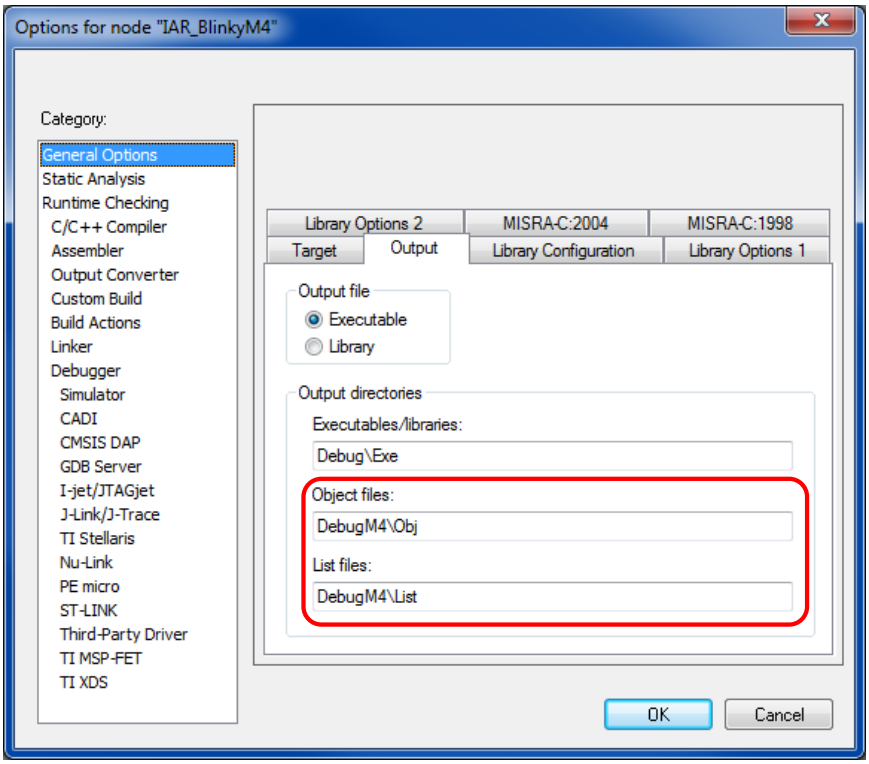


在“**Debugger**”部分的“**Setup**”选项卡中，选择 **CMSIS DAP** 驱动程序。在“**Download**”选项卡中，选中“**Suppress download**”。在 **CMSIS DAP** 部分的 **Setup** 选项卡中，将 **Reset** 设置为 **Disabled (no reset)**。该应用程序将由 CM4 项目加载。在“**Interface**”选项卡中，选择“**SWD**”。单击 **OK**。

重复上述步骤，为 CM4 创建第二个项目。**建议**：根据原始 PSoC Creator 项目名称和目标 CPU 命名项目。例如，对于 CE216795 双 CPU 闪烁项目，创建 IAR-EW 项目 IAR_BlinkyM4;请参见 Figure 35。配置类似于 CM0 +项目的项目，但有以下区别：

- CM4 项目必须与 CM0 +项目位于同一文件夹中；在此处为您的 PSoC Creator *<project> .cydsn* 文件夹。见 Figure 35。
- 选择... *CortexM4.ipcf* 文件；见 Figure 36。
- 在“**Options**”对话框的“**General Options**”部分的“**Output**”选项卡中，更改对象和列表文件的输出目录，如 Figure 38 所示。不要更改可执行文件/库输出文件夹。

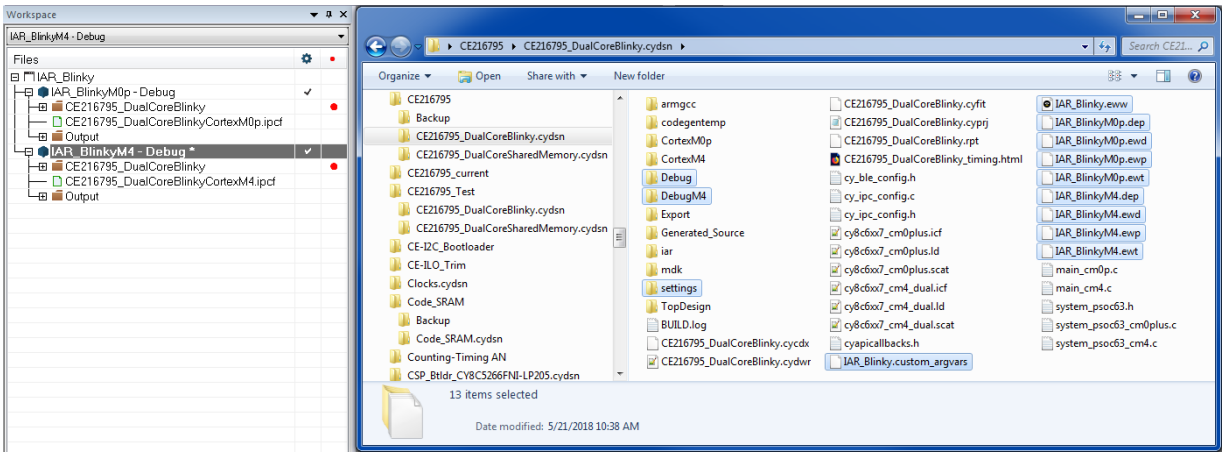
Figure 38. CM4 项目的唯一输出文件夹



- 在 **Build Actions** 部分，确认调用了 *postbuildCortexM4.bat*，参见 Figure 37。
- 在 **Debugger** 部分的 **Setup** 选项卡中，选择 **CMSIS DAP** 驱动程序。 在“**CMSIS DAP**”部分的“**Setup**”选项卡中，确认“**Reset**”设置为”**System (default)**”。 在“**Interface**”选项卡中，选择“**SWD**”。 单击”**OK**”。

选择**File > Save All**。 保存两个项目的所有文件，并自动生成工作区文件。 在Save Workspace As对话框中，在PSoC Creator *<project> .cydsn* 文件夹中创建一个名为 IAR_Blinky 的 IAR-EW 工作区。 创建的工作空间和项目以及相应的文件应与 Figure 39 类似。IAR-EW 生成的文件和文件夹将突出显示。

Figure 39. IAR Embedded Workbench 项目窗口和项目文件

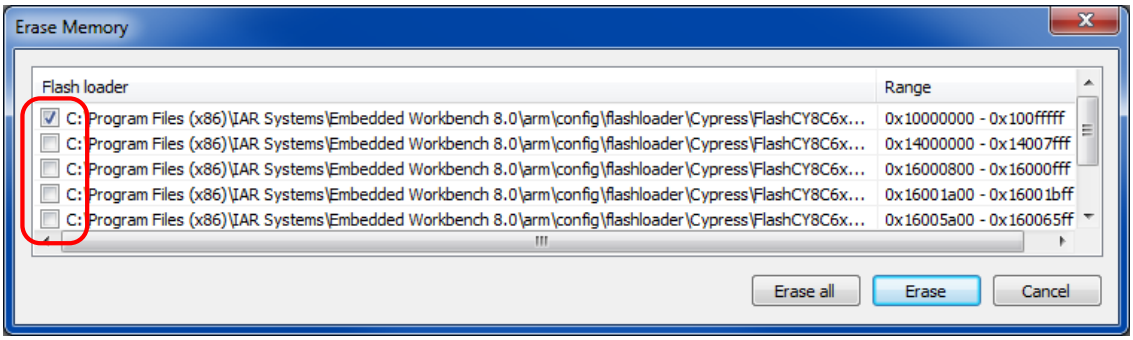


将 CY8CKIT-062-BLE USB 端口连接到计算机。 按套件按钮 SW3 将 KitProg2 置于 CMSIS-DAP 模式。 有关详细信息，请参阅套件指南。 这允许调试而无需使用任何外部调试探针。

按顺序构建项目; 首先构建 CM0 +项目。 请注意，IAR-EW 具有批量构建功能以自动执行该过程。 构建成功完成后，右键单击 BlinkyM4 项目并将其设置为活动项目。 然后通过（1）擦除闪存（**Project > Download > Erase memory**），以及（2）下载项目（**Project > Download > Download active application**）并确认正确操作来确认您的构建选项是否正确。 下载后，按套件重置按钮（RST/SW1）开始操作。

注意：擦除闪存时，通常只需要擦除 PSoC 6 MCU 应用程序闪存（0x1000 0000 – 0x100F FFFF），如 Figure 40 所示：

Figure 40. PSoC 6 MCU 的 IAR Embedded Workbench 擦除存储器对话框

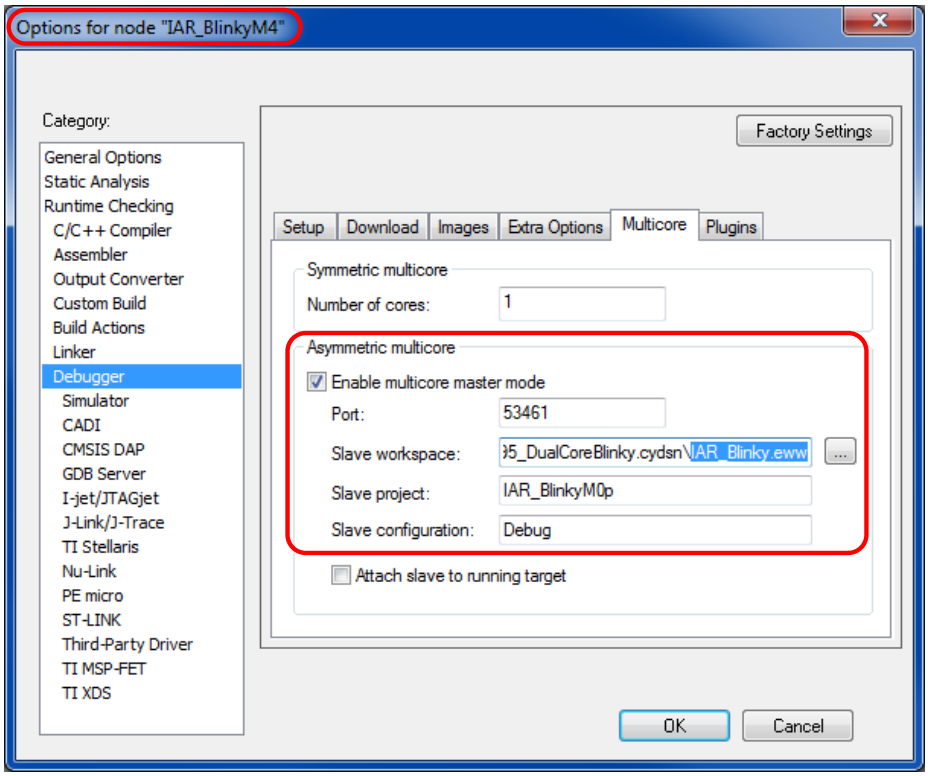


注意：如果更改 CM0 +项目中的任何代码，则必须重建这两个项目。 请注意，IAR-EW 具有批量构建功能以自动执行该过程。

5.调试 IAR-EW 项目

重新打开 CM4 项目的选项，然后转到 **Debugger** 部分的 **Multicore** 文件夹。 PSoC 6 MCU 具有不同的内核，即 CM0 +和 CM4，称为“非对称多核”。 因此，请填写非对称多核（**Asymmetric multicore**）部分中的字段，如 Figure 41 所示。 选中 **Enable multicore master mode** 可使 CM4 成为主站，以进行下载和调试。 不要更改端口。

Figure 41.设置多核调试



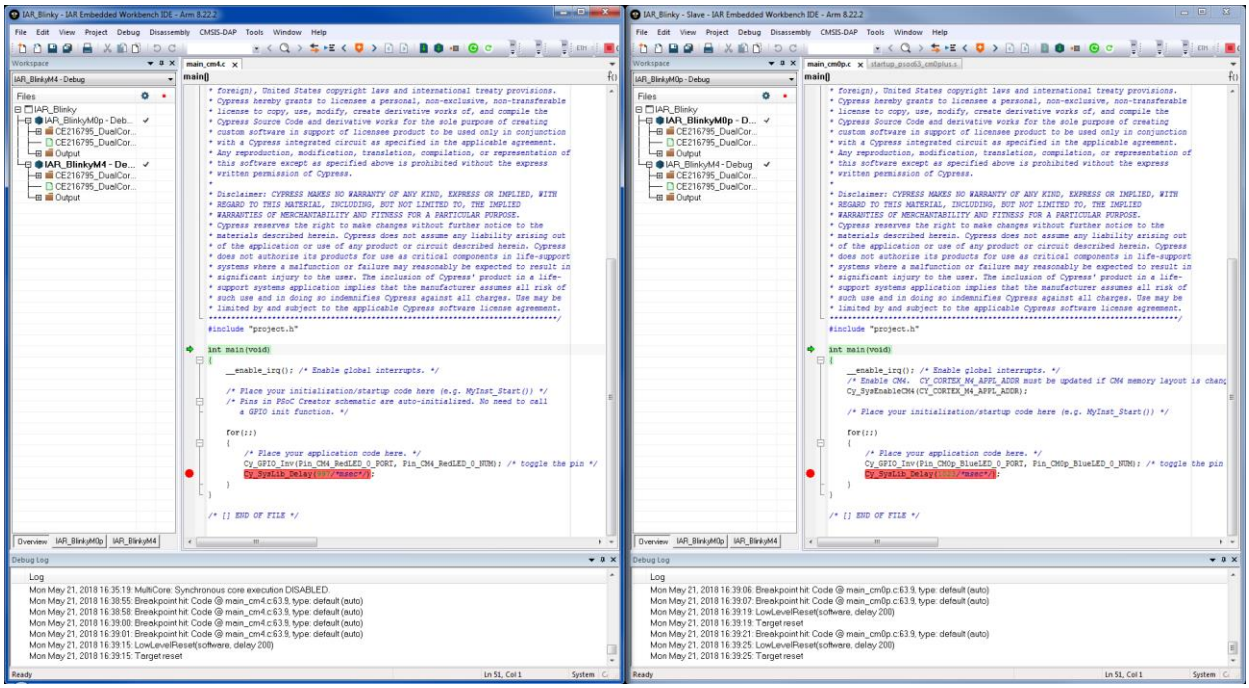
选择 **File** > **Save All** 以保存项目选项更改。然后通过选择 **Project** > **Download and Debug** 或 **Project** > **Debug without Downloading** 来开始调试。为 CM0 +项目自动打开 IAR Embedded Workbench 的第二个（从属）实例。两个实例共享套件连接和 PSoC 6 MCU 调试访问端口（DAP）。

在从属实例中，在第 63 行，即 `Cy_Syslib_Delay()`设置断点，。然后重复单击 **Debug** > **Go**，蓝色 LED 在断点处的每个停止点上切换。

单击 CM4 实例窗口中的任意位置并重复该过程。红色 LED 在断点处的每个停止点上切换。

它有助于将实例窗口并排放置在桌面上。窗口显示类似于 Figure 42。单击相应的窗口以在所需的 CPU 上执行调试操作。请注意，可以为每个 CPU 单独设置断点。您可以从任一窗口读取和更新相同的内存地址。

Figure 42. IAR Embedded Workbench 双 CPU 调试



您可以在任一窗口中停止调试。两个 CPU 的调试都已结束。按套件重置按钮（RST/SW1）重新启动套件操作。

5 总结

本应用笔记介绍了如何使用和优化 PSoC 6 MCU 中双 CPU 功能的固件和硬件设计。

优化 PSoC 6 MCU 设计的另一种方法是基于 PSoC 器件设计灵活，并使您能够在可编程模拟和数字模块中构建自定义功能。例如，PSoC 6 MCU 具有以下可用作“协处理器”的外设：

- **DMA 控制器**。请注意，C 编译器输出的最常见的 CPU 汇编程序指令是 MOV、LDR 和 STR，这意味着 CPU 花费大量周期来移动字节。让 DMA 控制器执行该操作。

  注：PSoC 6 MCU DMA 控制器具有一系列广泛的功能，使您能够构建独立于 CPU 的复杂数据传输和控制系统。ModusToolbox 设备配置器、PSoC Creator DMA 组件和 PDL 中的 API 提供了对这些功能的软件支持。有关详细信息，请参阅器件配置器帮助指南，DMA 组件数据表或 PDL 文档。赛普拉斯 HAL 只支持 CM4 CPU，因此 DMA HAL 驱动程序只能用于 CM4 应用。

- **加密块**。该模块为对称和非对称加密方法（AES，3DES，RSA 和 ECC）和散列函数（SHA-512，SHA-256）提供硬件加速。它还具有真正的随机数发生器（TRNG）功能。这些功能的软件支持由 PDL 中的 API 提供。请参阅 PDL 文档。赛普拉斯 HAL 目前还没有加密驱动。

- **通用数字模块（UDB）**。有多达 12 个 UDB，每个 UDB 都有一个 8 位数据路径，可以进行加，减，逐位运算，移位和循环冗余校验（CRC）。数据路径可以链接到字宽计算。考虑将 CPU 计算卸载到数据路径。目前，只有 PSoC Creator 支持 UDB。ModusToolbox 软件对 UDB 不提供任何支持。

  UDB 还具有可编程逻辑设备（PLD），可用于构建状态机;例如，查看查找表（LUT）组件数据表。 LUT 可以是基于硬件的有效替代方案，用于编程 CPU 中的状态机，例如使用 C 开关/ case 语句。

  此外，两个 GPIO 端口包括 Smart IO，可用于直接对进出 GPIO 引脚的信号执行布尔运算。赛普拉斯 HAL 不支持智能 I/O。

- 其他智能外设包括串行通信模块（SCB），计数器/定时器/ PWM 模块（TCPWM），蓝牙低功耗（BLE），I2S/PDM 音频，可编程模拟，和 CapSense®。使用这些外设可以进一步减轻 CPU 的处理负载。

  PSoC Creator 在 PDL 中提供了许多组件和大量的 API，以支持外设的功能。这使您可以在单个芯片中开发有效的多处理系统，从 CPU 中减轻大量功能负载。这反过来不仅可以减少代码大小，而且通过减少 CPU 必须执行的任务数量，提供降低 CPU 速度和功耗的机会。

  例如，您可以实现数字系统来控制多路复用 ADC 输入，并与 DMA 连接以将数据保存在 SRAM 中，从而创建一个高级模拟数据采集系统，而不使用 CPU。

  ModusToolbox 软件提供了一套用于设置外设的工具、用于所有赛普拉斯套件的预定义 BSP、用于 CapSense 和 emWin 等常见功能的库，以及一系列全面的示例应用程序，让你入门。

  赛普拉斯为 PSoC 外设提供广泛的应用笔记和代码示例支持，也提供详细器件数据表、PDL 文档, HAL 文档和技术参考手册（TRM）。有关更多信息，请参阅相关文档。

6 相关文档

有关 PSoC 6 MCU 资源的完整列表，请参见赛普拉斯社区的知识库文档 KBA223067。

应用笔记	
AN221774 – PSoC 6 MCU 入门	介绍 PSoC 6 MCU 器件以及如何构建第一个 ModusToolbox 或 PSoC Creator 项目
AN210781 – 带有蓝牙低功耗（BLE）连接的 PSoC 6 MCU 入门	介绍带有 BLE 连接器件的 PSoC 6 MCU 以及如何构建您的第一个 PSoC Creator 项目
AN217666-PSoC 6 MCU Interrupts	介绍 PSoC 6 MCU 中断架构以及如何配置中断
AN219434 – PSoC 6 MCU 导入生成代码到 IDE	描述如何将 PSoC Creator 生成的代码导入首选 IDE
代码示例(PSoC Creator)	
CE216795 – PSoC® 6 MCU Dual-Core Basics	演示 PSoC 6 MCU 中的两个 CPU 内核执行单独的独立任务，并使用共享内存和处理器间通信（IPC）模块相互通信
CE223820 – PSoC 6 MCU IPC Pipes	本示例演示了如何使用处理器间通信（IPC）驱动程序来实现 PSoC 6 MCU 中的消息管道。该管道用于在 CPU 之间发送消息。
CE223549 – PSoC 6 MCU IPC Semaphore	本示例演示了如何使用处理器间通信（IPC）驱动程序来实现 PSoC 6 MCU 中的信号量功能。该信号被用作锁定，以控制对 CPU 共享资源的访问。
CE226306 – PSoC 6 MCU Power Measurements	本示例展示了如何实现 PSoC 6 MCU 数据手册中所列的功率测量。
代码示例(ModusToolbox)	
mtb-example-psoc6-dual-cpu-empty-app	这是一个适用于 PSoC 6 MCU 器件的最小启动器双 CPU 应用模板。
mtb-example-psoc6-dual-cpu-ipc-sema	本示例演示了如何使用处理器间通信（IPC）驱动程序在 PSoC 6 MCU 中实现信号量功能。该信号用于锁定以控制对 CPU 共享资源的访问，并同步初始化指令
mtb-example-psoc6-dual-cpu-ipc-pipes	本示例演示了如何使用处理器间通信（IPC）驱动程序来实现 PSoC 6 MCU 中的消息管道。该管道用于在 CPU 之间发送消息
PSoC Creator 组件数据表	
Interrupt	支持从硬件信号生成中断
设备文档	
PSoC 6 MCU Datasheets	PSoC 6 Technical Reference Manuals
开发套件文档	
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
工具文档	
ModusToolbox 软件	ModusToolbox IDE 简化了物联网设计人员的开发。它为 Windows，macOS 和 Linux 提供易于使用的工具和熟悉的微控制器（MCU）集成开发环境（IDE）。

PSoC Creator	PSoC Creator 支持同时进行 PSoC 器件的硬件和固件编辑、编译和调试。应用程序使用原理图捕获和 150 多个预先验证的备产外设组件创建。查看下载选项卡中的“快速入门”和“用户指南”。
Peripheral Driver Library (PDL)	由 ModusToolbox IDE 或 PSoC Creator 4.3 安装的外设驱动程序库（PDL）。访问 https://github.com/cypresssemiconductorco/psoc6pdl;或, 对于 PSoC Creator，在 *<PDL install folder>\doc* 查看用户指南和 API 参考
Hardware Abstraction Layer (HAL)	只能通过 GitHub 获取。请访问 https://github.com/cypresssemiconductorco/psoc6hal

关于作者

姓名: Mark Ainsworth

职务: 高级主任应用工程师

教育背景: Mark Ainsworth 拥有 Syracuse 大学计算机工程学士学位和华盛顿大学电子工程硕士学位，以及多年设计和构建嵌入式系统的经验。

文档修订记录

文档名称: AN215656 - PSoC 6 MCU 双 CPU 系统设计

文档编号: 002-27241

版本	ECN	变更者	提交日期	变更说明
**	6574829	XITO	05/21/2019	翻译自 002-15656 Rev *F。
*A	7238582	XITO	08/23/2021	翻译自 002-15656 Rev *H。 完成日落审查。

销售、解决方案以及法律信息

全球销售和设计支持

赛普拉斯公司具有一个由办事处、解决方案中心、厂商代表和经销商组成的全球性网络。要想查找离您最近的办事处，请访问赛普拉斯所在地。

产品

Arm® Cortex® 微控制器	cypress.com/arm
汽车级产品	cypress.com/automotive
时钟与缓冲器	cypress.com/clocks
接口	cypress.com/interface
物联网	cypress.com/iot
存储器	cypress.com/memory
微控制器	cypress.com/mcu
PSoC	cypress.com/psoc
电源管理 IC	cypress.com/pmic
触摸感应	cypress.com/touch
USB 控制器	cypress.com/usb
无线连接	cypress.com/wireless

PSoC®解决方案

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

赛普拉斯开发者社区

社区 | 项目 | 视频 | 博客 | 培训 | 组件

技术支持

cypress.com/support

Arm and Cortex are registered trademarks of Arm Limited (or its subsidiaries) in the US and/or elsewhere.

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

© 赛普拉斯半导体公司，2017-2021 年。本文件是赛普拉斯半导体公司及其子公司，包括 Spansion LLC（“赛普拉斯”）的财产。本文件，包括其包含或引用的任何软件或固件（“软件”），根据全球范围内的知识产权法律以及美国与其他国家签署条约由赛普拉斯所有。除非在本款中另有明确规定，赛普拉斯保留在该等法律和条约下的所有权利，且未就其专利、版权、商标或其他知识产权授予任何许可。如果软件并不附随有一份许可协议且贵方未以其他方式与赛普拉斯签署关于使用软件的书面协议，赛普拉斯特此授予贵方属人性质的、非独家且不可转让的如下许可（无再许可权）（1）在赛普拉斯特软件著作权项下的下列许可权（一）对以源代码形式提供的软件，仅出于在赛普拉斯硬件产品上使用之目的且仅在贵方集团内部修改和复制软件，和（二）仅限于在有关赛普拉斯硬件产品上使用之目的将软件以二进制代码形式的向外部最终用户提供（无论直接提供或通过经销商和分销商间接提供），和（2）在被软件（由赛普拉斯公司提供，且未经修改）侵犯的赛普拉斯专利的权利主张项下，仅出于在赛普拉斯硬件产品上使用之目的制造、使用、提供和进口软件的许可。禁止对软件的任何其他使用、复制、修改、翻译或汇编。

在适用法律允许的限度内，赛普拉斯未对本文件或任何软件作出任何明示或暗示的担保，包括但不限于关于适销性和特定用途的默示保证。没有任何电子设备是绝对安全的。因此，尽管赛普拉斯在其硬件和软件产品中采取了必要的安全措施，但是赛普拉斯并不承担任何由于使用赛普拉斯产品而引起的安全问题及安全漏洞的责任，例如未经授权的访问或使用赛普拉斯产品。此外，本材料中所介绍的赛普拉斯产品有可能存在设计缺陷或设计错误，从而导致产品的性能与公布的规 格不一致。（如果发现此类问题，赛普拉斯会提供勘误表）赛普拉斯保留更改本文件的权利，届时将不另行通知。在适用法律允许的限度内，赛普拉斯不对因应用或使用本文件所述任何产品或电路引起的任何后果负责。本文件，包括任何样本设计信息或程序代码信息，仅为供参考之目的提供。文件使用人应负责正确设计、计划和测试信息应用和由此生产的任何产品的功能和安全性。赛普拉斯产品不应被设计为、设定为或授权用作武器操作、武器系统、核设施、生命支持设备或系统、其他医疗设备或系统（包括急救设备和手术植入物）、污染控制或有害物质管理系统中的关键部件，或产品植入之设备或系统故障可能导致人身伤害、死亡或财产损失其他用途（“非预期用途”）。关键部件指，若该部件发生故障，经合理预期会导致设备或系统故障或会影响设备或系统安全性和有效性的部件。针对由赛普拉斯产品非预期用途产生或相关的任何主张、费用、损失和其他责任，赛普拉斯不承担全部或部分责任且贵方不应追究赛普拉斯之责任。贵方应赔偿赛普拉斯因赛普拉斯产品任何非预期用途产生或相关的所有索赔、费用、损失和其他责任，包括因人身伤害或死亡引起的主张，并使之免受损失。

赛普拉斯、赛普拉斯徽标、Spansion、Spansion 徽标，及上述项目的组合，WICED，及 PSoC、CapSense、EZ-USB、F-RAM 和 Traveo 应视为赛普拉斯在美国和其他国家的商标或注册商标。请访问 cypress.com 获取赛普拉斯商标的完整列表。其他名称和品牌可能由其各自所有者主张为该方财产。