PSoC 6 MCU 设备固件升级软件开发套件指南

关于本文档

范围及目的

本指南提供有关如何使用设备固件升级(DFU) 软件开发工具套件 (SDK) 为赛普拉斯 PSoC 6 MCU 产品开发 DFU 系统的全面信息。包括 SDK 应用程序编程接口 (API) 的详细描述，以及多个引导加载程序代码示例的构建说明。

相关部件系列

全部 PSoC® 6 MCU 部件

相关代码示例

参阅**相关文档**

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 视频库。

目录

关于本文档 1
目录 1
1 简介 3
2 关于设备固件升级 4
2.1 术语与定义 4
2.2 DFU 模块的使用 5
2.3 基本 DFU 功能流程 5
2.4 其它使用案例 5
3 DFU SDK 描述 7
3.1 DFU SDK 文件 7
3.1.1 用户回调函数 8
3.1.2 链接器脚本 9
3.2 DFU 生态系统 9
4 如何使用 SDK 10
4.1 确定系统中的应用 10
4.2 定位内存中的应用 11
4.3 设计应用 12
4.3.1 ModusToolbox 指南 12
4.3.2 PSoC Creator 指南 15

目录

4.4 构建和编程应用 21
5 PSoC 6 MCU DFU 代码示例 22
5.1 如何构建代码示例 23
5.2 PSoC 6 MCU 基础 DFU 23
5.2.1 ModusToolbox 指南 23
5.2.2 PSoC Creator 指南 26
5.3 PSoC 6 MCU Bluetooth LE DFU 32
5.3.1 带可升级堆栈的 Bluetooth LE DFU 37
5.4 PSoC 6 MCU 双应用程序 Bootloader 39
5.5 PSoC 6 MCU 加密 Bootloader 41
6 相关文档 44
7 Appendix A. DFU 主机工具 45
8 Appendix B. 主机命令/响应协议 46
8.1 命令/响应数据包结构 46
8.2 命令 46
8.2.1 进入 DFU 47
8.2.2 同步 DFU 47
8.2.3 退出 DFU 48
8.2.4 发送数据 48
8.2.5 发送无响应数据 48
8.2.6 编程数据 48
8.2.7 确认数据 49
8.2.8 擦除数据 50
8.2.9 确认应用 50
8.2.10 设置应用元数据 51
8.2.11 获取元数据 51
8.2.12 设置 EIVector 52
9 Appendix C. .cyacd2 文件格式 53
10 Appendix D. 应用元数据 54
10.1 元数据结构 55
11 Appendix E. 构建后批处理文件列表 56
11.1 ModusToolbox 构建后 Bash 文件 56
11.2 PSoC Creator 构建后 Batch 文件 57
文档修订记录 59

1 简介

本指南简要介绍了设备固件更新 (DFU，也称为“引导加载程序”) 的基础知识，然后详细介绍了赛普拉斯 DFU 软件开发套件 (SDK) 以及如何将其与 PSoC 6 MCU 一起使用。

Note: “bootloader”一词在业界已变得承载太多含义，并经常与设备启动(“boot-up”)过程相混淆。因此，本应用笔记和相关的代码示例正转换成“设备固件更新”或 DFU，以减少歧义。

本指南涵盖了多种开发工具，包括赛普拉斯的 ModusToolbox 集成开发环境 (IDE) 和赛普拉斯的 PSoC Creator IDE。ModusToolbox IDE 是适用于 Windows，macOS 和 Linux 的免费 IDE。它结合了 PSoC 6 MCU 和业界使用最广泛的 WiFi 和蓝牙技术，为物联网 (IoT) 设计人员提供了简单、一致且熟悉的设计体验。有关 ModusToolbox IDE 的更多信息，请单击**此处**。

PSoC Creator 是基于 Windows 的免费 IDE，可基于多个赛普拉斯 PSoC MCU 系列进行系统的并行硬件和固件设计。有关 PSoC Creator 的更多信息，请单击**此处**。

如果您是 DFU/bootloader 的新手，那么下一部分将介绍基本概念和设计原理，**关于设备固件升级**?

如果您熟悉 DFU 概念并想了解它们如何在 PSoC 6 MCU 中实现，请参见 DFU SDK 说明，如何使用 SDK 和 PSoC 6 MCU DFU 代码示例部分。有关 DFU 代码示例的列表，请参见相关文档。有关 PSoC 6 MCU 代码示例的完整列表，请单击**此处**。

Note: 在发布时，支持基于 UART，I^2C，SPI 和 Bluetooth LE 的 DFU 系统。其他通信渠道将在以后的 SDK 更新中添加。

2 关于设备固件升级

设备固件升级 (DFU) 是 MCU 系统设计的常见部分。DFU 程序使现场更新产品的固件成为可能。在典型产品中，固件存储在 MCU 的闪存中。MCU 安装在印刷电路板 (PCB) 上并嵌入产品中，如 **Figure 1** 所示。

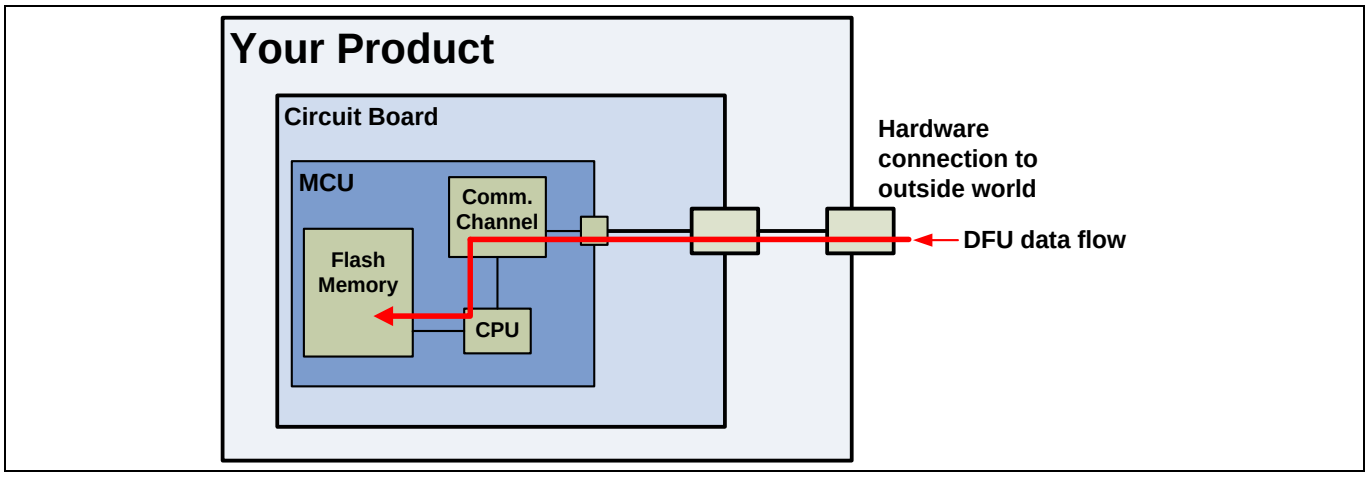


Figure 1 DFU 数据流框图

在工厂，通常使用 MCU 的 JTAG 或 SWD 接口在 PCB 组装时完成将固件初始编程到产品中。但是，这些接口通常在现场不可用，因此通常不用于固件更新。

现场更新固件的更好方法是使用产品与外部世界之间的现有连接。连接可以是标准通信端口，例如 I^2C，USB 或 UART，无线信道，例如 Bluetooth LE，或者自定义协议。

2.1 术语与定义

Figure 1 表明产品的嵌入式固件必须能够将通信端口用于两个不同的目的: 正常操作和更新闪存。知道如何更新闪存的那部分嵌入式固件称为 **DFU 模块**，或 bootloader (引导加载程序) ，如 **Figure 2** 所示。

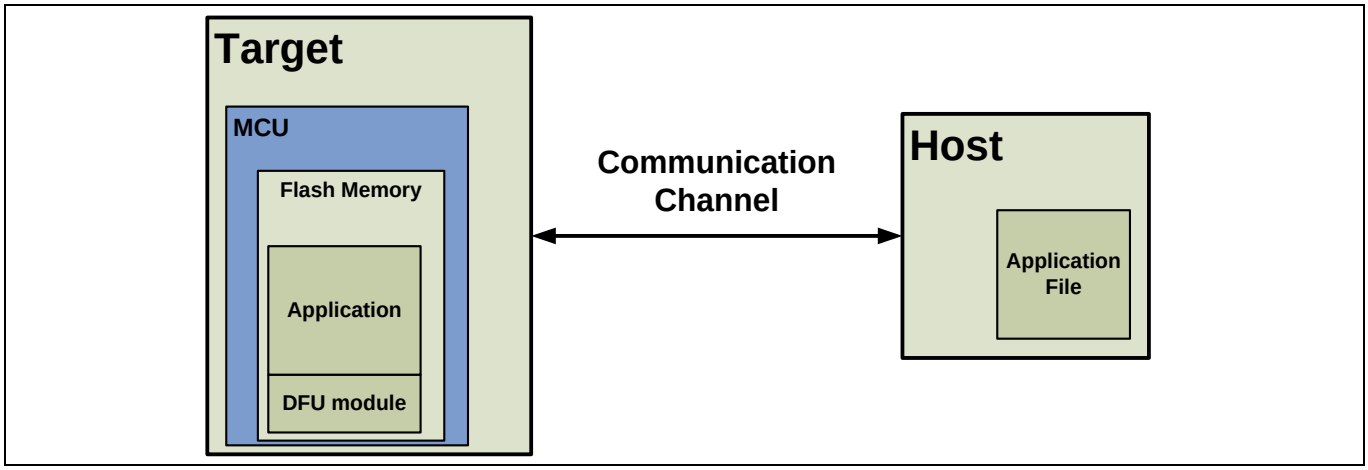


Figure 2 通用 DFU 系统

通常，提供更新闪存的数据的系统称为主机(host)，而正在更新的系统称为目标(**target**)。主机可以是外部计算机或与目标位于同一 PCB 上的另一个 MCU。

关于设备固件升级

将数据从主机传输到目标闪存的行为称为 **DFU** 或 **DFU** 操作 (在以前的文档中，赛普拉斯将 DFU 称为引导加载程序。但是，该术语经常与启动相混淆。因此，在本应用笔记和相关的代码示例中，为了减少歧义，我们将其称为 DFU。) 放在闪存中的数据称为**应用程序**或**固件映像**。

2.2 DFU 模块的使用

DFU 模块和应用程序通常共享通信端口。DFU 的第一步是将产品置于执行 DFU 模块而不是应用程序的模式。这可以通过响应诸如按钮按下或接收到的命令之类的事件来完成。该应用程序检测到此类事件，并通过将控制权转移到 DFU 模块进行响应。

DFU 模块运行后，主机可以通过通信通道发送“start DFU”命令。如果 DFU 模块发送“OK”响应，则 DFU 操作可以开始。

2.3 基本 DFU 功能流程

在 DFU 期间，主机读取新应用程序的文件，将其解析为闪存编程命令，并将这些命令发送到 DFU 模块。收到整个文件并将其安装在目标闪存中后，DFU 模块可以将控制权传递给新应用程序。

DFU 模块通常在设备重置后首先执行 [1]。然后，它可以执行以下操作:

- 在将控制转移到该应用程序之前检查新应用程序的有效性
- 管理开始主机通信的时间
- 执行 DFU 操作
- 将控制权交给新应用

Figure 3 是显示其工作原理的流程图。

2.4 其它使用案例

存在其他更复杂的 DFU 用例。一个常见的用例是应用程序正在运行，并且有一些事件警告应用程序可以获得更新。

当应用程序继续执行其正常任务时，它还会将新应用程序下载到临时位置。验证新应用程序有效性后，将其复制到闪存中的正确位置，控制权将转移给它。

[1] 在典型的 MCU 中，多个事件可能导致器件复位，例如器件上电或复位引脚上的电压电平。此外，固件可以通过写入器件寄存器来触发复位。这称为“软件重置”或 SRES。使用 SRES 功能，应用程序可以重置设备，例如响应按钮等事件。这有效地将控制转移到 DFU 模块，因为 DFU 模块首先在设备重置时执行。

关于设备固件升级

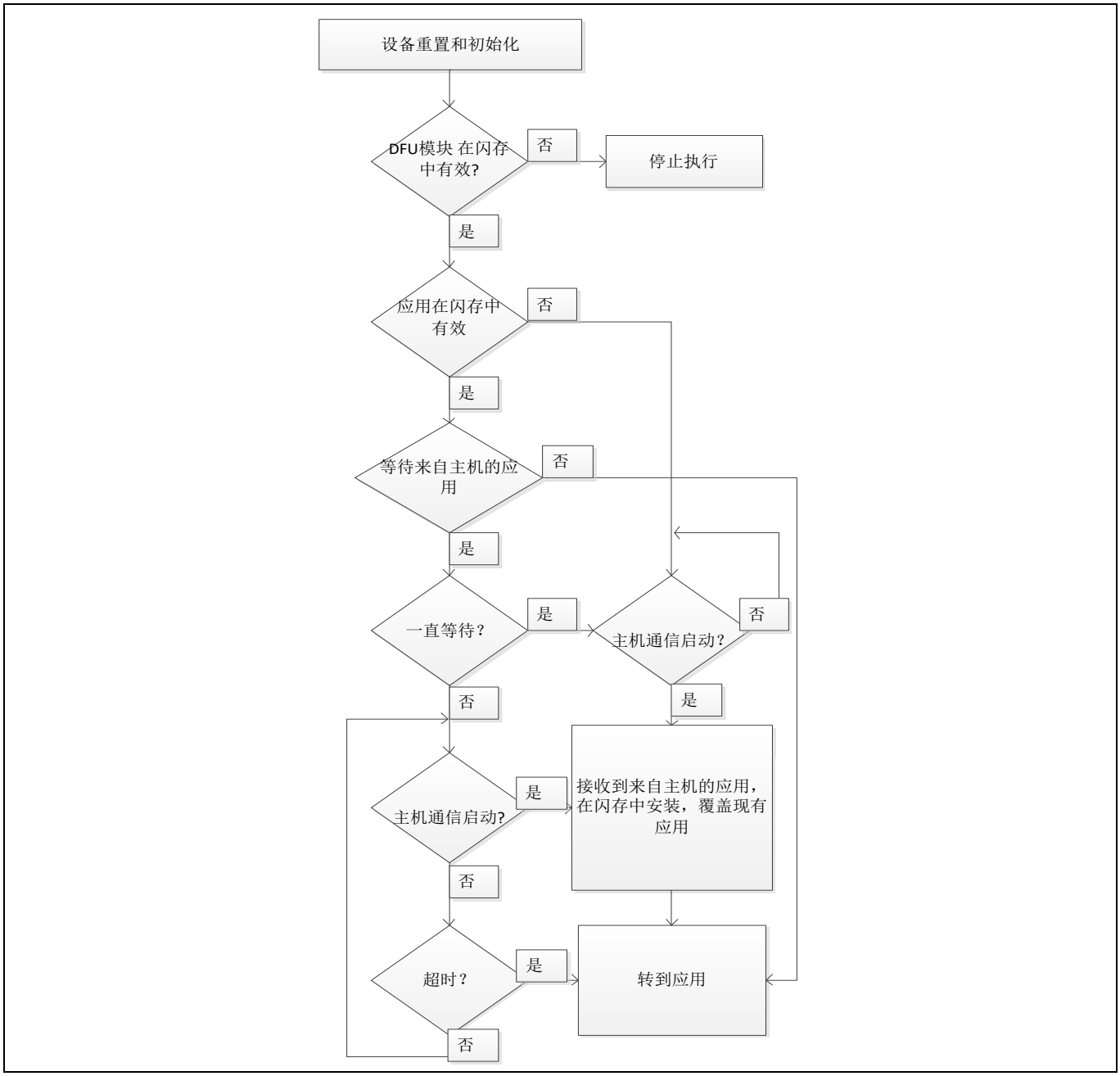


Figure 3　DFU 功能流程图

3 DFU SDK 描述

赛普拉斯 DFU SDK 是一个 API，由一组可调用的函数和其他元素组成，可实现快速 DFU 系统开发。此 SDK 是赛普拉斯 PSoC 6 SDK 的一部分。该 SDK 通常位于 *C:\ ... \ ModusToolbox_1.0 \ libraries \ psoc6sw-1.0 \ components \ psoc6mw \ dfu* 中。该 SDK 可以被其他 IDE 以及 ModusToolbox IDE 使用。

SDK 包含以下内容:

- 实现 SDK API 的源代码，通常是.*h* 和.*c* 文件、链接脚本文件、以及 mk 文件
- 文档，即本指南。
- 代码示例

Note: *您可以由于自身目的修改 DFU SDK 源代码，例如修改或添加命令到主机接口协议(请参阅* ***8 Appendix B****. 主机命令/响应协议)。*

3.1 DFU SDK 文件

Table 1 列出了 SDK 中的文件。

Note: *PSoC 6 MCU 有两个 Arm® CPU: Cortex®-M4 (CM4) 和 Cortex-M0+ (CM0+)。为每个 CPU 提供链接器脚本文件，用于多个 IDE。有关更多信息，请参见* ***AN215656****，《PSoC 6 MCU 双 CPU 系统设计》。*

Table 1 DFU SDK 文件

文件	描述	
cy_dfu.h, .c	DFU SDK 文件	
cy_dfu_bwc_macro.h	向后兼容赛普拉斯的传统 Bootloader SDK	
dfu_user.h	包含用户可编辑的`#define` 语句，这些语句控制 SDK 中的操作和启用的功能	
dfu_user.c	包含 SDK 所需的用户功能: 五个功能，用于控制与 DFU 主机的通信； 这些也称为“运输功能” 两个函数– `ReadData()` 和 `WriteData()`–控制对内部或外部存储器的访问 提供的功能是默认功能，可以针对您的应用程序进行修改	
transport_uart.h, .c *transport_... .h, .c*	包含正在使用的主机通信通道的 DFU 传输功能； 这些函数通常由 *dfu_user.c* 中的传输函数调用	
dfu_cm4.ld, *dfu_cm0p.ld*	自定义 GCC 链接描述文件	在每个应用程序中，这些链接描述文件会替换默认的链接描述文件。这些文件为每个 CPU 以及 DFU 模块和其他区域定位代码和数据部分。
dfu_cm4.scat, *dfu_cm0p.scat,* *dfu_mdk_common.h,* *dfu_mdk_symbols.c*	自定义 MDK 链接器脚本； 存在公用文件以创建必要的符号定义。	
dfu_cm4.icf, *dfu_cm0p.icf*	自定义 IAR 链接器脚本	

这些文件在 PDL DFU 文件夹中的组织，如 **Figure 4** 所示:

```
\---dfu
    |   cy_dfu.h
    |   cy_dfu.c
    |   cy_dfu_bwc_macro.h
    \---config
    |       dfu_user.h
    |       dfu_user.c
    |       transport_uart.h
    |       transport_uart.c
    |       transport_... .h  transport files for other
    |       transport_... .c  communication channels
    \---linker_scripts
        \---GCC
        |       dfu_cm0p.ld
        |       dfu_cm4.ld
        \---MDK
        |       dfu_cm0p.scat
        |       dfu_cm4.scat
        |       dfu_mdk_common.h
        |       dfu_mdk_symbols.c
        \---IAR
                dfu_cm0p.icf
                dfu_cm4.icf
```

Figure 4　DFU SDK 文件组织

3.1.1　用户回调函数

DFU SDK API 函数回调多个用户函数。这允许您自定义以下 DFU 操作:

- 主机通信，也称为传输功能或通信接口功能
- 读取和写入设备的内部闪存以及其他非易失性存储器 (NVM)，例如外部闪存

Table 2 列出了所需的用户回调函数。DFU 代码示例显示了这些函数的典型代码; 请参阅 **DFU 代码示例**和**相关文档**。

Table 2　用户回调函数

函数	描述
用于主机通信。示例在 *dfu_user.c*、*transport_xxx.h* 和 *.c* 文件中	
`Cy_DFU_TransportStart()`	打开并初始化通讯信道
`Cy_DFU_TransportStop()`	关闭通讯信道
`Cy_DFU_TransportReset()`	重新初始化信道，通常使其恢复到已知状态
`Cy_DFU_TransportRead()`	接收来自主机的数据包； 请参阅**主机数据包结构**
`Cy_DFU_TransportWrite()`	发送数据包到主机，请参阅 **Host 主机数据包结构**
用于非易失性内存 (NVM) 访问。例子在 *dfu_user.c* 中	
`Cy_DFU_ReadData()`	从设备闪存或其他 NVM 读取数据
`Cy_DFU_WriteData()`	将数据写入设备闪存或其他 NVM

3.1.2 链接器脚本

根据定义，DFU 系统是 NVM 中具有多个应用程序 (至少两个) 的系统。你决定每个应用程序驻留在 NVM 中的哪个位置。DFU SDK 包含可以适应您的应用程序的模板链接器脚本文件 (请参阅 **Table 1** 和 **Figure 4**) 。DFU 代码示例包括示例链接脚本文件。

3.2 DFU 生态系统

SDK 中不包含 DFU 系统的其他元素。它们可以在 ModusToolbox IDE，PSoC 6 SDK 中的其他地方，PSoC Creator IDE 或外围驱动程序库 (PDL) 中使用，并且可以被其他 IDE 使用:

- CyMCUElfTool: 文件夹... \ ModusToolbox_1.0 \ tools \ cymcuelftool 1.0 \ bin 中的 cymcuelftool.exe。该程序被称为构建应用程序的一部分，如 **Figure 5** 所示。 (该图与 PSoC Creator 和其他 IDE 相似。)
- 该程序将核心和应用程序映像文件组合到输出文件中。它是一个命令行程序；其选项语法记录在 Help 命令 (`-h / --help`) 输出中。您可以在以下文件夹中找到用户指南: \ cymcuelftool 1.0 \ doc。
- 其他驱动程序 - 用于通信、闪存等 - 位于... \ ModusToolbox_1.0 \ libraries \ psoc6sw-1.0 \ components \ psoc6pdl \ drivers 文件夹中。它们根据 DFU SDK API 函数的需要进行调用。
- 设备固件更新主机工具: 文件夹... \ *ModusToolbox_1.0* \ *tools* \ *dfuh tool 1.0* 中的 *dfuh-tool.exe*。这是 **Figure 2** 中所示“主机”的示例。有关更多信息，请参见附录 A，设备固件更新主机工具。

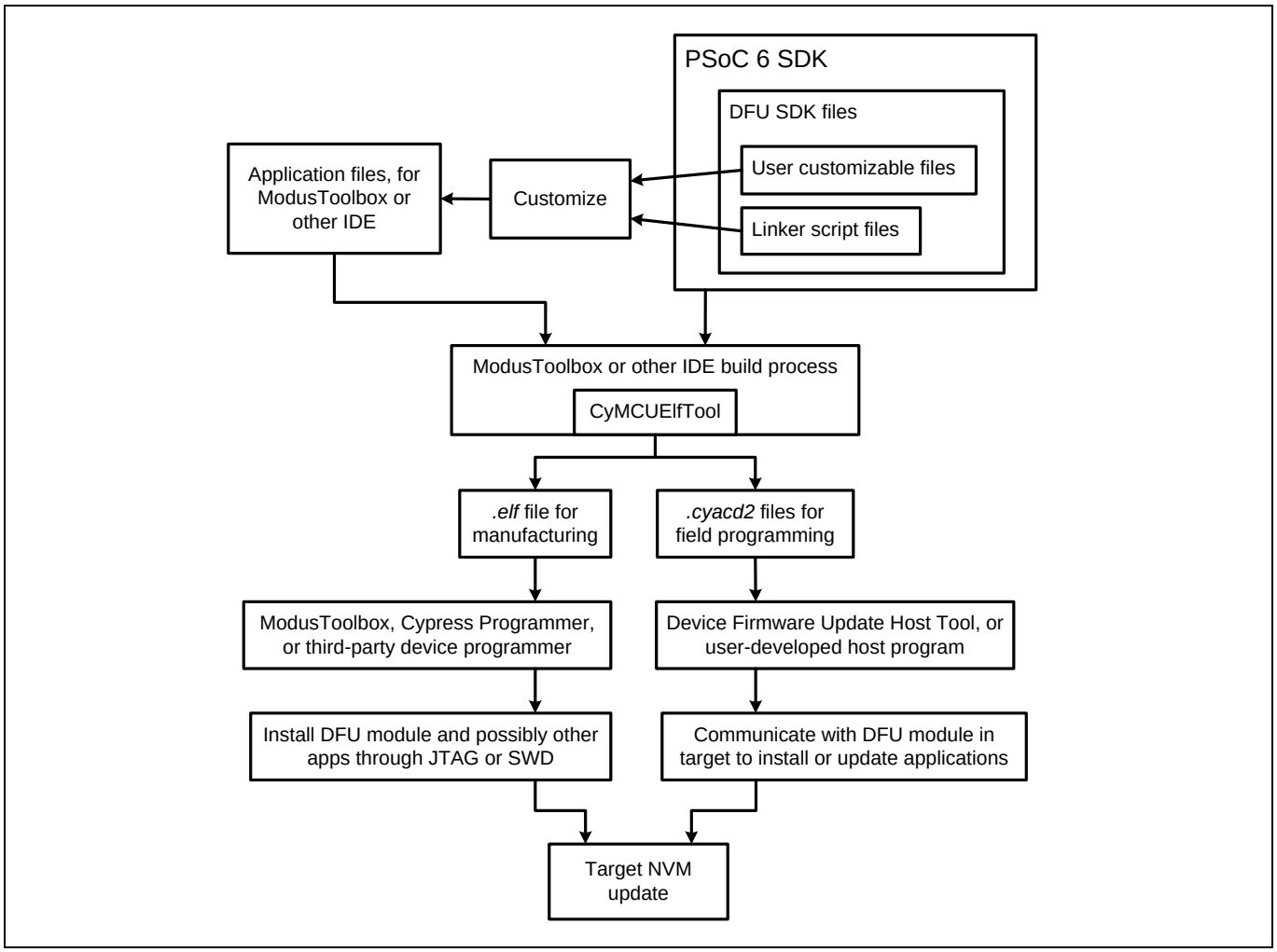


Figure 5 ModusToolbox 应用程序构建和 DFU 流程图

4 如何使用 SDK

本节介绍使用 DFU SDK 的一般过程。在较高的层面上，步骤如下:

1. **确定系统中的应用**
2. **定位存储器中的应用**
3. **设计应用**
4. **构建和编程应用**

有关如何构建 DFU SDK 代码示例的详细说明，请转至 **DFU 代码示例**。

4.1 确定系统中的应用

第一步是确定您的设计中有多少个应用程序，以及每个应用程序的用途。哪个应用程序具有 DFU 功能? **Figure 6** 显示了多个应用程序的一些典型内存映射:

- 映射 **A** 显示了一个基本情况，其中 Application＃0 下载和安装应用程序＃1。
- 映射 **B** 显示了一个更复杂的案例。有三个应用程序。所有应用程序都有 DFU 模块-它们可以下载和安装任何其他应用程 (实际上，应用程序＃0 通常不会更新。) 数据部分位于所有应用程序之外。
- 映射 **C** 显示与映射 **B** 相同的情况，除了其中一个应用程序位于外部非易失性存储器 (NVM) 。

在每个映射中，DFU 系统使用少量闪存 (通常为一行) 来存储应用程序元数据。有关信息，请参阅 **Appendix D. 应用元数据**。

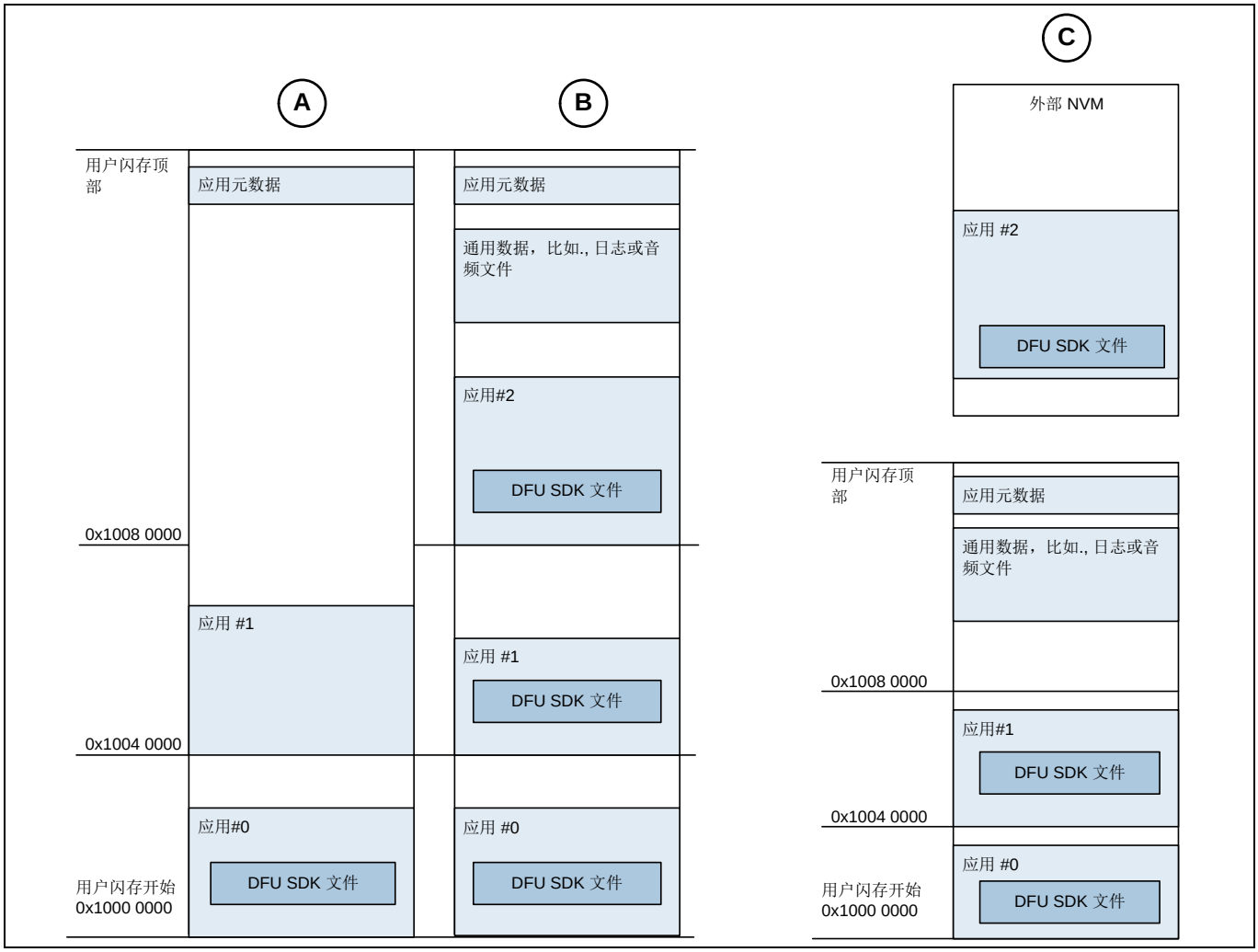


Figure 6　示例 DFU 内存映射

4.2　定位内存中的应用

下一步是确定每个应用程序在闪存中的位置。有几个因素需要考虑:

- 最多支持 63 个应用程序，具体取决于元数据大小。有关 DFU SDK 元数据的更多信息，请参阅**附录 D，应用程序元数据**。
- 应用程序可能包含一个或多个 CPU 的代码和数据，例如 PSoC 6 MCU 器件中的 CM4 和 CM0+。更多信息，请参见应用笔记 **AN215656**，*PSoC 6 MCU 双 CPU 系统设计*。

第一个应用程序 (通常称为 “App0”) 必须位于用户闪存的开头，在 PSoC 6 MCU 中的地址为 0x1000 0000，如 **Figure 6** 所示。这是设备复位后执行的第一个应用程序 [1]。

Note:　*另一个模块 (如安全系统中的安全映像模块) 可能占用地址 0x1000 0000，而 App0 可能位于闪存中的其他位置。有关更多信息，请参阅 **AN221111**，创建安全系统。*

1 CM0+ CPU 首先在设备复位时执行 SROM 和其他系统级代码，然后将控制权转移到位于 0x1000 0000 的应用程序。有关详细信息，请参阅设备技术参考手册（TRM）。

- 任何应用程序都可以包含 DFU SDK，因此能够下载和安装另一个应用程序，和/或将控制权转移到另一个应用程序。在大多数情况下，App0 应包含 DFU SDK。
- 当以非阻塞模式写入闪存时，PSoC 6 MCU 具有 512 KB 区域的边写边读 (RWW) 功能。即，DFU 代码可以在一个闪存区域中执行，而在另一区域中更新闪存。有关更多信息，请参见 PDL 闪存驱动程序文档和设备技术参考手册 (TRM)。有关详细信息，请参阅 PDL 闪存驱动程序文档和设备**技术参考手册 (TRM)**。
- 应用程序可能驻留在外部存储器以及设备的内部闪存中，如 **Figure 6** 中的映射 C 所示。外部存储器位于从 0x1800 0000 到 0x1FFF FFFF 的 128 MB 子区域中。

Note: *在外部存储器以外执行时，必须使用 XIP 模式下的 QSPI 模块将外部存储器设备映射到 PSoC 6 MCU。有关详细信息，请参阅设备技术参考手册 (TRM)。*

- PSoC6 MCU 应用程序包含验证字节，这些字节作为应用程序的一部分保存在 NVM 中。这样可以在将控制转移到应用程序之前验证应用程序。

Note: *根据应用程序所在的安全环境，可以构建应用程序以进行简单验证 (使用校验和或 CRC 数据) 以及授权 (使用加密签名数据)。DFU 系统可以直接实现验证，也可以调用 Flash Boot 模块中的函数来检查授权。有关更多信息，请参阅 **AN221111**，创建安全系统。*

- 在正常工作流程中可能更新的 NVM 数据 (例如事件日志) 不应该是应用程序的一部分，因为它会使应用程序难以验证。相反，指定一个位于任何应用程序空间之外的 NVM 区域，并将其用于数据存储。**Figure 6** 显示了映射 **B** 和 **C** 中的两个示例。
- 为应用程序元数据保留 1 行闪存，如 **Figure 6** 所示 (在极少数情况下，可能需要多行)。应用程序元数据由 DFU SDK 管理，并由应用程序用于将控制权转移到另一个应用程序。应用元数据空间应位于任何应用空间之外。

4.3 设计应用

4.3.1 ModusToolbox 指南

Note: *本部分仅包含 DFU 特定的说明。有关创建 ModusToolbox 应用程序的详细分步说明，请参见 ModusToolbox 帮助。**AN221774**，PSoC 6 MCU 入门；或本文档中的 **DFU 代码示例**。*

下一步是设计**前面步骤**中标识的每个应用程序。每个应用程序都是单独一个 ModusToolbox 应用程序，独立于任何其他应用程序。使用 ModusToolbox IDE，您可以将所有应用程序放在一个工作区中，或在单独的工作区中，也可以在计算机上的不同位置中。在开始使用 PSoC 6 MCU 之前，建议针对整体系统开发需求制定工作区和应用程序计划。

Note: *用于 PSoC 6 MCU (双 CPU) 的 ModusToolbox 应用程序由五个项目组成:*

- <project name>_config: 设备配置文件 design.modus 和设备配置源文件
- <project name>_mainapp: Cortex-M4 (CM4) CPU 的源文件
- <project name>_mainapp_cm0p: CPU Cortex-M0+ (CM0+) CPU 的源文件
- <project name>_mainapp_psoc6pdl: Cortex-M4 (CM4) CPU 的驱动程序文件
- <project name>_mainapp_cm0p_psoc6pdl: Cortex-M0+ (CM0+) CPU 的驱动程序文件

对每个要执行 DFU 操作和/或作为可安装应用程序的 ModusToolbox 应用程序，执行以下操作:

如何使用 SDK

1. 如果应用程序要执行 DFU 操作，请在 *design.modus* 文件中启用通信外设。该外设实现与 DFU 主机的通信通道。

   DFU SDK 包括对许多通信通道类型的支持，包括 UART，SPI，I2C 和 Bluetooth LE。每种通道类型都有代码示例。请参阅 **DFU 代码示例**和**相关文档**。

   您也可以创建自定义的通信通道。驱动程序必须实现用户 **回调函数**中描述的传输功能。

2. 将 DFU SDK 集成到项目中，如 **Figure 7** 所示。在项目资源管理器窗口中右键单击_mainapp 项目，然后单击 **ModusToolbox Middleware Selector**。

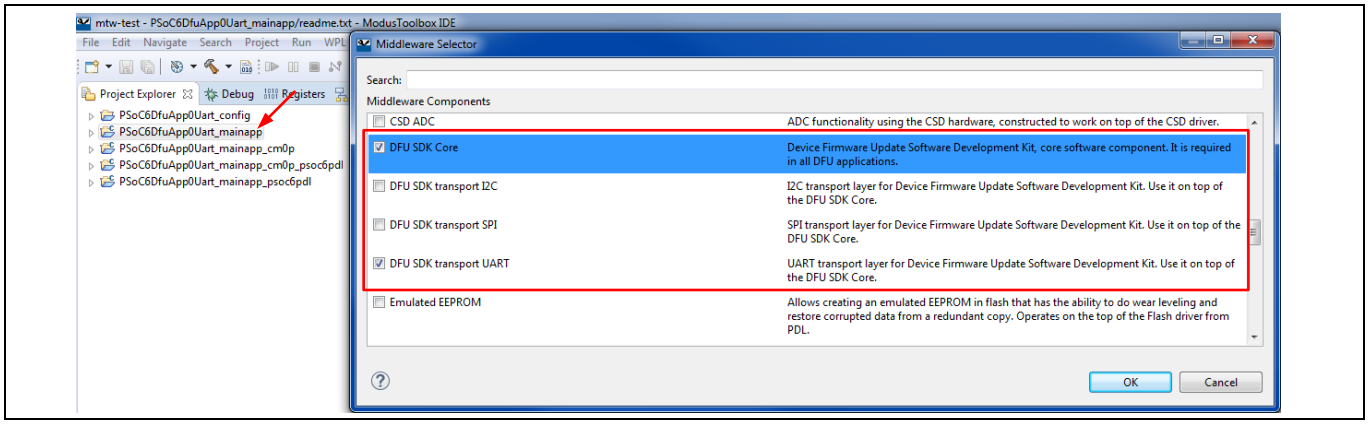


Figure 7　将 DFU SDK 合并到 ModusToolbox 项目中

如果应用程序将执行 DFU 操作，请选择 **DFU SDK Core** 和 **DFU SDK transport xxx** 框。

Note: *在大多数情况下，只会选择一个传输盒。但是，如果您想拥有一个自定义的通信通道，则可以取消选中所有传输盒，甚至选择多个传输盒。*

如果该应用程序是可下载的应用程序，并且将控制权转移到另一个应用程序，请选择 **DFU SDK Core**。将控制权转移到另一个应用程序的代码包含在 DFU SDK 核心文件中。

完成后单击 **OK**。所需的源.c，.h 和链接脚本 (例如.ld) 文件会自动添加到项目中。

对_mainapp_cm0p 项目重复此步骤； 但仅选择 **DFU SDK core**。

3. 默认情况下，已为应用程序＃0 (App0) 设置已安装的链接程序脚本文件。对于其他应用程序，通过更改应用程序编号来编辑文件。以下示例显示了 dfu_cm0p.ld 和 dfu_cm4.ld 中针对 GCC 链接器的可下载应用程序 App1 的编辑:

```
/*
* DFU SDK specific: aliases regions, so the rest of the code does not use
* application-specific memory region names
*/
REGION_ALIAS("flash_core0", flash_app1_core0);
REGION_ALIAS("flash",       flash_app1_core1);
REGION_ALIAS("ram",           ram_app1_core1);

/* DFU SDK specific: sets an app Id */
__cy_app_id = 1;
```

4. 更改项目属性以使用 DFU 链接描述文件而不是默认链接描述文件。为_mainapp 选择 *dfu_cm4.ld* 文件，如 **Figure 8** 所示。包括文件的相对路径。为_mainapp_cm0p 选择 *dfu_cm0p.ld* 文件 (图中未显示)。

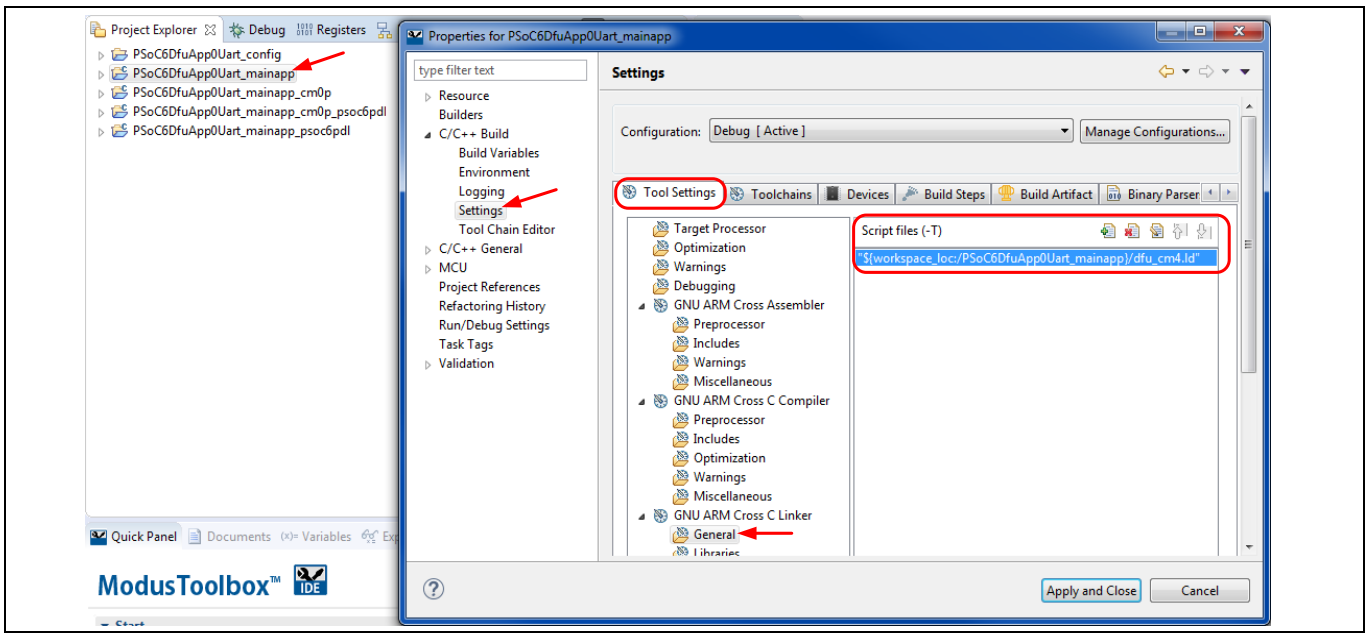


Figure 8　自定义链接描述文件的 ModusToolbox 项目属性设置

5. 对于可下载的应用程序项目，添加一个构建后命令来调用 Cypress 实用程序 *cymcuelftool.exe*，如 **Figure 9** 所示。cymcuelftool 包含在 ModusToolbox 安装中。它生成一个*.cyacd2 文件，该文件由 DFU 主机下载 (参见 **Figure 5**)。仅针对 CM4 二进制文件执行 post-build 命令。cymcuelftool 组合了 CM4 (__mainapp) 和 CM0+ (mainapp_cm0p) 项目，以创建输出.*cyacd2* 文件。

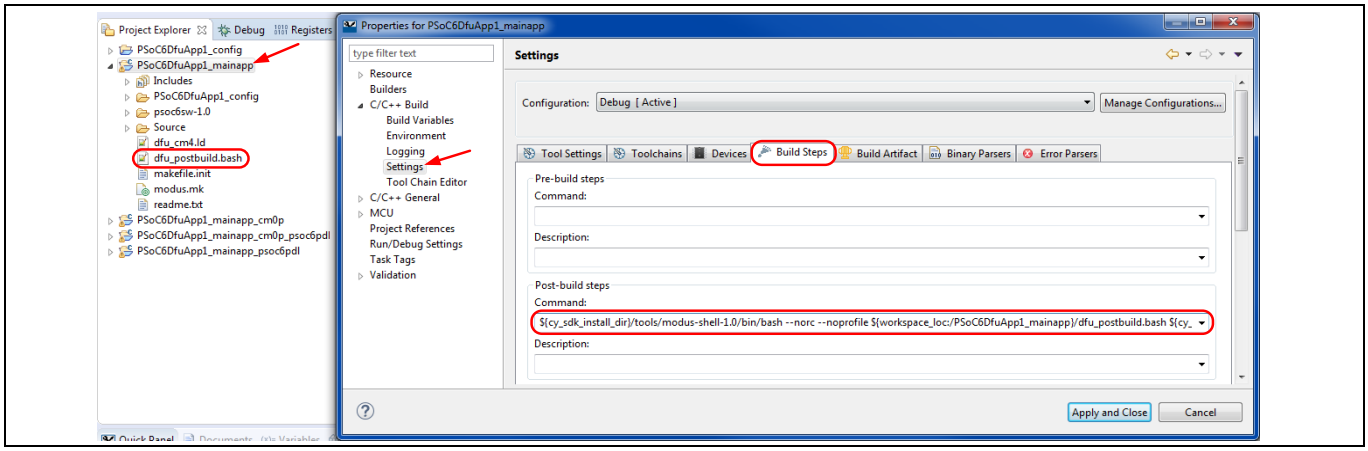


Figure 9　ModusToolbox Post-Build 命令

该命令通常在代码示例 **CE213903**，*PSoC 6 MCU Basic DFU* 中调用 bash 文件，例如 dfu_postbuild.bash。为了方便起见，将文件复制并粘贴到_mainapp 项目中。

然后更改_mainapp 项目属性以添加类似于以下内容的命令文本，如 **Figure 9** 所示。

```
${cy_sdk_install_dir}/tools/modus-shell-1.0/bin/bash --norc --noprofile
${workspace_loc:/PSoC6DfuApp1_mainapp}/dfu_postbuild.bash
${cy_sdk_install_dir}/tools/cymcuelftool-1.0/bin/cymcuelftool
```

```
${workspace_loc:/PSoC6DfuApp1_mainapp_cm0p}/${config_name:PSoC6DfuApp1_mainapp_cm0p
}/PSoC6DfuApp1_mainapp_cm0p.elf
${workspace_loc:/PSoC6DfuApp1_mainapp}/${config_name:PSoC6DfuApp1_mainapp}/PSoC6Dfu
App1_mainapp.elf ARM_CM4
```

6. 根据需要将代码添加到 CM4 (__mainapp) 和 CM0+ (mainapp_cm0p) 项目中的 main.c 和其他源文件中。具体要求是:

- 根据需要向所有源文件添加`#include "cy_dfu.h"`语句，以访问 DFU SDK API。
- 对于可下载的应用程序，在__mainapp_cm0p 项目的 *main.c* 中，将 `Cy_SysEnableCM4()`函数调用更改为: `Cy_SysEnableCM4((uint32_t)(&__cy_app_core1_start_addr));`
- 对于可下载的应用程序，在__mainapp 项目的 main.c 中，添加以下全局语句。调整所选签名类型的数组大小。

```
/* This section holds signature data for application verification. */
CY_SECTION(".cy_app_signature") __USED static const uint32_t
cy_bootload_appSignature[1];
```

在两个主文件的总体代码中，请考虑以下内容:

- 调用哪个 DFU 函数:
  - `Cy_DFU_Complete()`: 在执行整个 DFU 操作时会阻止。如果在 DFU 期间没有其他任务要做，请调用此函数。
  - `Cy_DFU_Init()`然后是对 `Cy_DFU_Continue()` 的一系列调用: `Cy_DFU_Continue()`在接收，处理和响应来自主机的一个命令包时阻止。如果必须在 DFU 中完成其他任务，请调用这些函数。
- 每个应用程序是否应将控制权传递给另一个应用程序。根据需要向 DFU SDK API 函数 `Cy_DFU_ValidateApp()`和 `Cy_DFU_ExecuteApp()`添加调用。

有关更多信息，请参见 PDL DFU SDK 文档。

Note: *您可以复制和粘贴* ***DFU 代码示例*** *和* ***相关文档*** *中列出的代码。*

4.3.2 PSoC Creator 指南

Note: *此部分仅包含特定于 Bootloader 的指令(在 ModusToolbox IDE 中，已将其更改为 DFU 操作)。有关创建 PSoC Creator 项目的详细逐步说明，请参见 PSoC Creator 帮助;* ***AN210781*** *，具有蓝牙低功耗 (Bluetooth LE) 连接的 PSoC 6 MCU 入门;或本文档中的* ***如何构建代码示例*** *。*

下一步是设计**前面步骤**中确定的每个应用程序。每个应用程序都是单个 PSoC Creator 项目，独立于任何其他项目。使用 PSoC Creator，您可以将所有项目放在一个工作区 (*.cywrk* 文件) 中，也可以放在单独的工作区中以及计算机上的不同位置。在开始使用 PSoC 6 MCU 之前，建议为您的整体系统开发需求制定工作区和项目计划。

对要作为 DFU 或可安装应用程序的每个 PSoC Creator 项目执行以下操作:

1. 如果应用程序要进行引导加载，请在项目原理图上放置一个通信组件;即 *TopDesign.cysch* 文件。此组件实现到 DFU 主机的通信通道。将组件端子连接到相应的物理引脚。

   DFU SDK 支持 PSoC Creator 组件目录中的许多通信组件，包括 UART，SPI，I2C 和 Bluetooth LE。代码示例可用于每个通信组件;请参阅 **DFU 代码示例**和**相关文档**。

Note: *为了与 SDK 中的默认传输文件兼容，请将组件命名为* ***UART，I2C，SPI 或 Bluetooth LE*** *。*

   您还可以创建自定义通信通道。驱动程序必须实现 **用户回调函数**中描述的传输函数。

2. 将 DFU SDK 合并到项目中，如 **Figure 10** 所示。在 Workspace Explorer 窗口中右键单击该项目，然后选择 **Build Settings** ....在 **Build Settings** 对话框中，选择 **Peripheral Driver Library**。

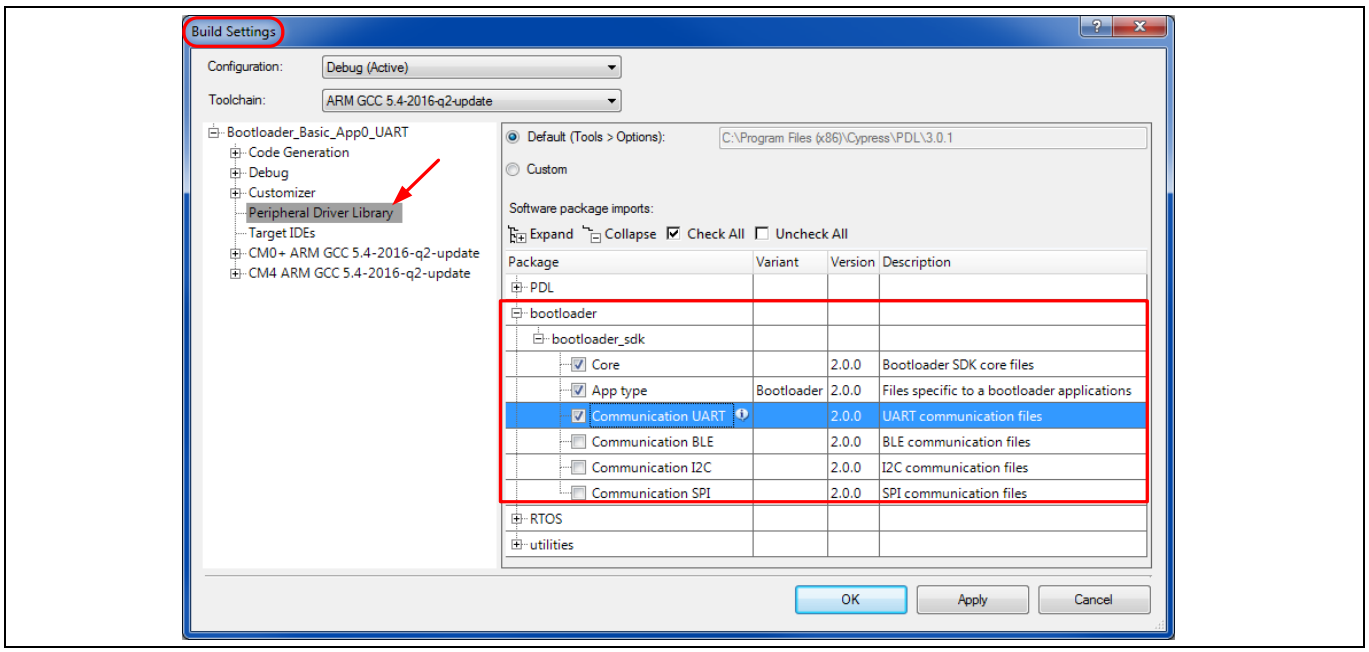


Figure 10　将 DFU SDK 合并到 PSoC Creator 项目中

如果应用程序将执行引导加载操作，请选择 **Core**, **App type bootloader,** 和 **Communication。**

Note: *在大多数情况下，只选中一个 Communication 框。但是，如果您想拥有自定义通信通道，则可以取消选中所有 Communication 框，甚至可以选中多个框。*

如果该应用程序是可下载的应用程序，并且将控制权转移到另一个应用程序，请选择 Core。SDK 核心文件中包含将控制权转移到另一个应用程序的代码。

完成后单击 **OK**。所需的源.*c*，.*h* 和链接脚本 (例如.*ld*) 文件会自动添加到项目中

3. 生成项目文件。单击 Workspace Explorer 窗口中的项目，然后选择 **Build**> **Generate Application**。文件将被添加到项目中，如 **Figure 11** 所示。

请注意，PSoC 6 MCU 有两个 Arm CPU: CM4 和 CM0+。每个 CPU 都有自己的文件夹，以及通用文件夹和共享文件文件夹。用于查找多个 IDE 的每个 CPU 的代码的链接器脚本文件会自动复制并添加到项目中。有关更多信息，请参见《**AN215656**，PSoC 6 MCU 双 CPU 系统设计》。

Note: ***Figure 11*** *显示了添加到 CM0p 文件夹的 DFU 文件。类似的文件也会添加到 CM4 文件夹中，但不会显示。*

4. 编辑文件 *bootload_user.h* 和 *bootload_user.c*。查看 *bootload_user.c* 中的**用户回调函数**，并编辑它们以进行所需的任何自定义，例如，写入和读取外部存储器。通常，功能可以保持不变。

Note: *添加的 bootload_user.c 文件包含 UART 的默认代码。对于其他通信通道 (如 I²C 和 SPI)，请按如下方式编辑 bootload_user.c:*

– 更改:

```
#include "transport_uart.h"
```

为:

```
#include "transport_i2c.h"
```

或:

```
#include "transport_spi.h"
```

或:

```
#include "transport_ble.h"
```

– 更改 “UART_Uart” 的五个实例为 “I2C_I2c”, “SPI_Spi”, 或 “CyBLE”

5. 根据您使用的编译器，编辑相应的公共链接描述文件，以编码在**定位内存中的应用**中所做的决策。有关示例，请参见**第 5.3 节第 9 步**。

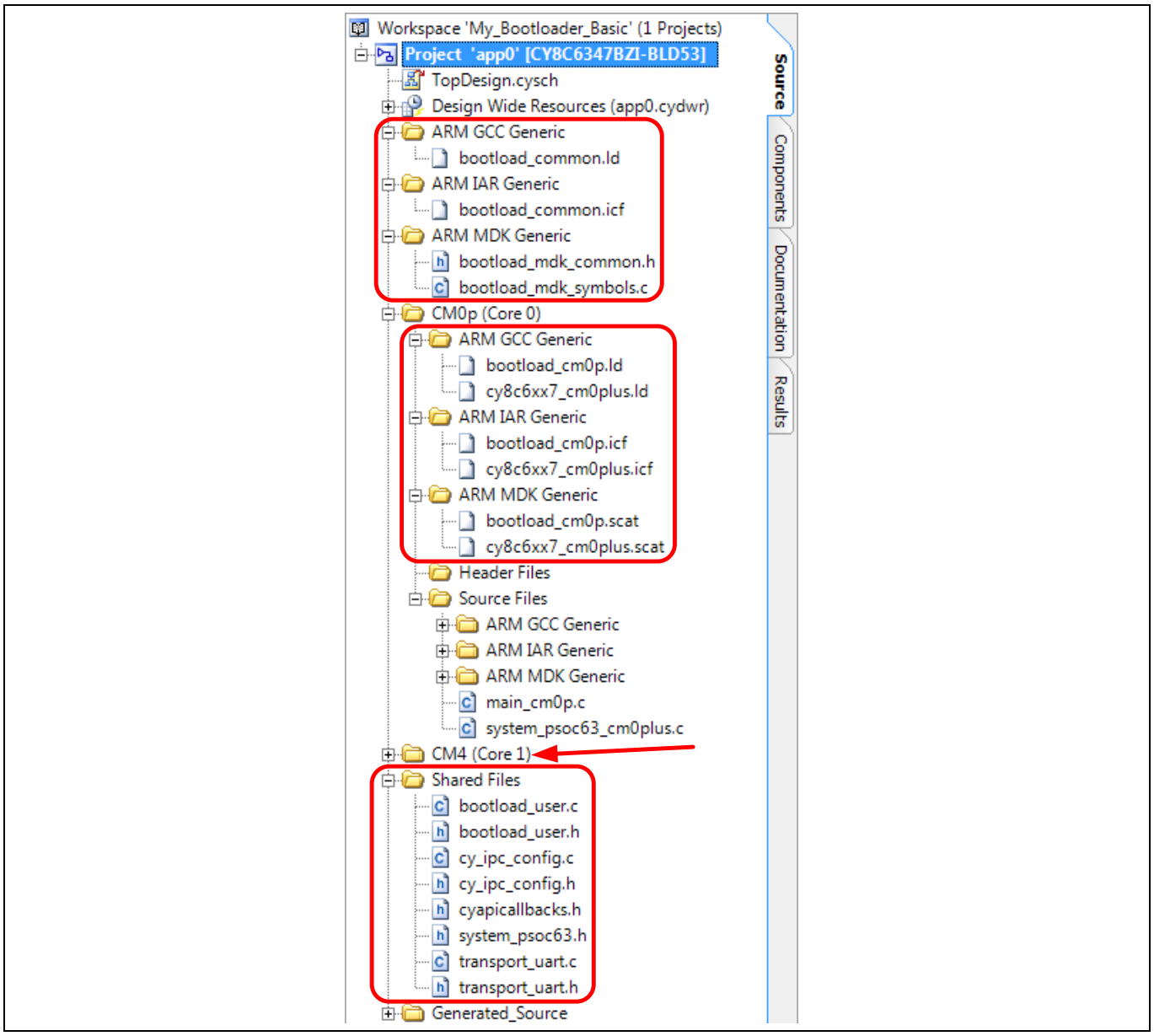


Figure 11　在 PSoC Creator 项目中添加 SDK 文件

6. 链接脚本文件默认情况下为应用程序＃0 (app0) 设置。对于其他应用程序，请通过更改应用程序编号来编辑文件。以下示例显示了针对 GCC 链接器的 app1 在 *bootload_cm0p.ld* 和 *bootload_cm4.ld* 中的编辑:

```
/*
* Bootloader SDK-specific: aliases regions, so the rest of the code does not
* use application-specific memory region names
*/
REGION_ALIAS("flash_core0", flash_app1_core0);
REGION_ALIAS("flash",       flash_app1_core1);
REGION_ALIAS("ram",           ram_app1_core1);

/* Bootloader SDK-specific: sets App ID */
__cy_app_id = 1;
```

Keil µVision MDK 链接器更复杂。以下示例显示了 *bootload_cm0p.scat* 和 *bootload_cm4.scat* 中 app1 的编辑:

```
; Flash
#define FLASH_START             CY_APP1_CORE0_FLASH_ADDR
#define FLASH_SIZE              CY_APP1_CORE0_FLASH_LENGTH

; Emulated EEPROM Flash area
#define EM_EEPROM_START         CY_APP1_CORE0_EM_EEPROM_ADDR
#define EM_EEPROM_SIZE          CY_APP1_CORE0_EM_EEPROM_LENGTH

; External memory
#define XIP_START               CY_APP1_CORE0_SMIF_ADDR
#define XIP_SIZE                CY_APP1_CORE0_SMIF_LENGTH

; RAM
#define RAM_START               CY_APP1_CORE0_RAM_ADDR
#define RAM_SIZE                CY_APP1_CORE0_RAM_LENGTH
```

bootload_mdk_symbols.c 中的 App1 编辑如下:

```
__cy_app_core1_start_addr   EQU __cpp(CY_APP1_CORE1_FLASH_ADDR)

/* Application number (ID) */
__cy_app_id                 EQU 1

/* CyMCUElfTool uses these to generate an application signature */
__cy_app_verify_start     EQU __cpp(CY_APP1_CORE0_FLASH_ADDR)
__cy_app_verify_length    EQU __cpp(CY_APP1_CORE0_FLASH_LENGTH +
                                    CY_APP1_CORE1_FLASH_LENGTH -
                                    __CY_BOOT_SIGNATURE_SIZE)
```

7. 更改“项目构建”设置以使用 DFU 链接脚本文件而不是默认链接脚本文件。选择 CM0+ CPU 的 *bootload_cm0p.ld* 文件，如 **Figure 12** 所示。包括文件的相对路径。选择 CM4 CPU 的 *bootload_cm4.ld* 文件 (**Figure 12** 中未显示) 。

Note: *为 GCC，Keil µVision MDK 和 IAR Embedded Workbench 链接器提供了链接器脚本文件。显示的示例适用于 GCC。为 IDE 使用适当的链接脚本文件。*

Note: *应当对 Debug 和 Release 配置执行此操作。*

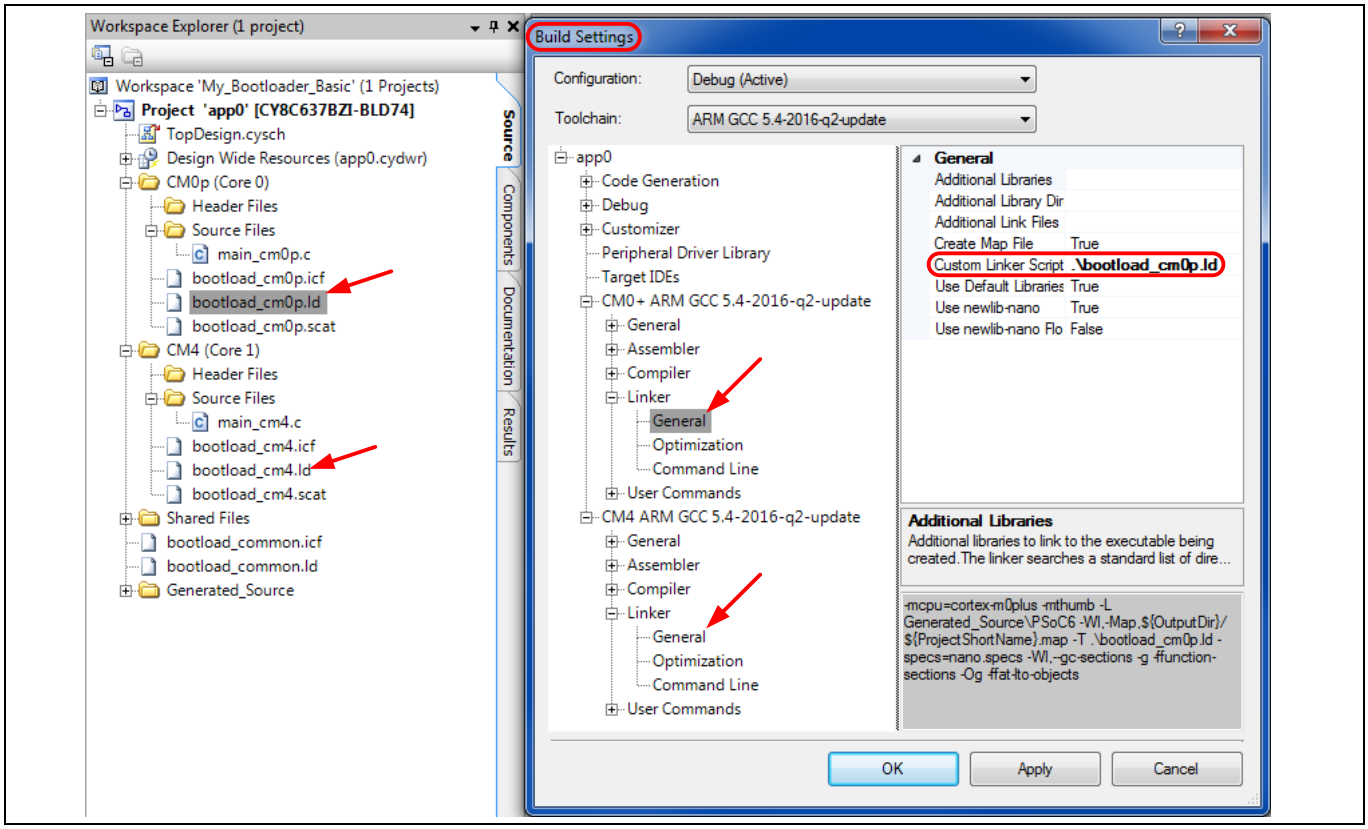


Figure 12　自定义链接脚本文件的 PSoC Creator 项目构建设置

8. 对于可下载的应用程序项目，添加生成后的批处理文件以调用赛普拉斯实用程序 *cymcuelftool.exe*。**cymcuelftool** 包含在 PSoC Creator 安装中。它生成一个*.cyacd2* 文件，由 DFU 主机下载 (参见 **Figure 5**) 。批处理文件仅应用于 CM4 二进制文件。

Note: *有关批处理文件的示例，请参阅 DFU SDK 代码示例，如 **CE213903**，PSoC 6 MCU Basic DFU。它通常位于文件夹 CE213903 \ DFU_Basic_App1.cydsn 中。批处理文件内容也可在 **Appendix E** 中找到。为方便起见，将文件复制并粘贴到 PSoC Creator 项目文件夹 app1.cydsn 中。*

然后在 PSoC Creator 的 Workspace Explorer 窗口中，将批处理文件添加到项目的 Shared Files 文件夹中，并将该文件作为构建后命令添加到 Cortex-M4 构建中，如 **Figure 13** 所示。

Note: *必须对 Debug 和 Release 配置执行此操作。*

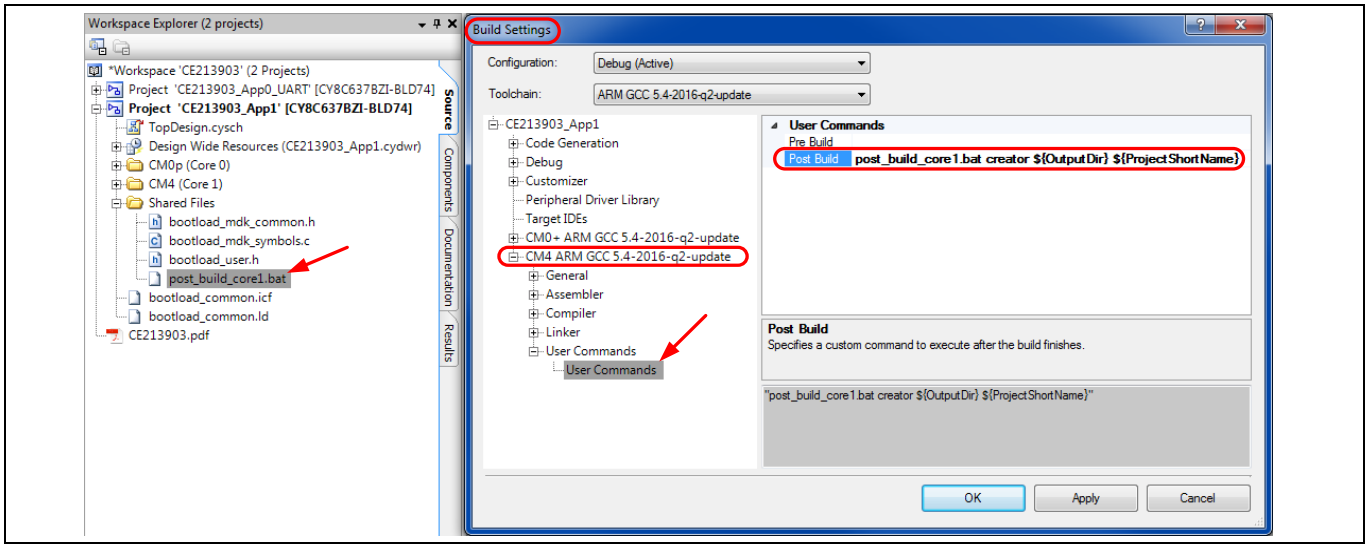


Figure 13　构建后命令

根据需要添加代码到两个 *main_cmx.c* 和其它源文件中。对于可下载的应用项目，添加到 *main_cm4.c* 文件全局阵列以存储应用签名。比如，对于校验和，最简单和最常见的签名类型，添加以下内容:

```
/* This section holds signature data for application verification.
   For checksum verification, set the number of elements in the array to 1,
   and in bootload_common.ld set __cy_boot_signature_size = 4.
*/
      CY_SECTION(".cy_app_signature") __USED static const uint32_t
                                      cy_bootload_appSignature[1];
```

在两个主文件的整体代码中，请考虑以下事项:

- 要调用哪个 DFU 函数:
  - `Cy_Bootload_DoBootload()`:在执行整个引导加载操作时阻塞。如果在引导加载时没有其他任务要做，请调用此函数。
  - `Cy_Bootload_Init()` 后跟一系列对 `Cy_Bootload_Continue()`的调用

`Cy_Bootload_Continue()` 在接收，处理和响应来自主机的一个命令包时阻塞。如果在引导加载时必须完成其他任务，请调用这些函数。

- 每个应用应将控制权转交给另一个应用。根据需要添加对 DFU SDK API 函数 `ValidateApplication()` 和 `ExecuteApplication()`的调用。

有关更多信息，请参阅 PDL DFU 文档。

*Note:　　您可以从 **DFU 代码示例**和**相关文档**中列出的代码示例中复制和粘贴代码。*

4.4 构建和编程应用

构建所有应用程序。如 **Table 3** 所示，ModusToolbox IDE 将创建多个输出文件。PSoC Creator 的输出文件与此类似。 (另请参见 **Figure 5**) 。所有输出文件都在其各自项目的 Debug 或 Release 文件夹中。

Table 3 ModusToolbox IDE 的 DFU 输出文件 (CE213903，PSoC 6 Basic DFU 中的示例)

文件名	说明
App0 配置 DFU 模块, 用于 ModusToolbox IDE	
PSoC6DfuApp0xxx_mainapp_cm0p.elf	链接器输出，CM0+项目
PSoC6DfuApp0xxx_mainapp_cm0p_signed.elf	签名过程后，cymcuelftool 输出，CM0+项目
PSoC6DfuApp0xxx_mainapp.elf	链接器输出，CM4 项目
PSoC6DfuApp0xxx_mainapp_signed.elf	签名过程后的 cymcuelftool 输出，CM4 项目
PSoC6DfuApp0xxx_mainapp_final.elf	cymcuelftool 最终输出，结合了 CM4 和 CM0+输出。这是已编程到目标设备的文件。
App1, 可下载应用, 用于 ModusToolbox IDE	
PSoC6DfuApp1_mainapp_cm0p.elf	链接器输出，CM0+项目
PSoC6DfuApp1_mainapp_cm0p_signed.elf	签名过程后，cymcuelftool 输出，CM0+项目
PSoC6DfuApp1_mainapp.elf	链接器输出，CM4 项目
PSoC6DfuApp1_mainapp_signed.elf	签名过程后的 cymcuelftool 输出，CM4 项目
PSoC6DfuApp1_mainapp_final.elf	cymcuelftool 最终输出，结合了 CM4 和 CM0+输出。
PSoC6DfuApp1_mainapp_dfu.elf	cymcuelftool 最终输出，用 CRC 字节签名。请参阅 dfu_postbuild.bash。
PSoC6DfuApp1_mainapp_dfu.cyacd2	cymcuelftool 可下载输出。请参阅 *dfu_postbuild.bash*。这是设备固件更新主机工具的输入。

使用 ModusToolbox IDE 或 PSoC Creator 将 *App0 ..._ mainapp_final.elf* 文件编程到设备中。有关设备编程的更多信息，请参见这些工具中的帮助菜单。

以后，您可以将 Application #0 中的 DFU 模块与主机程序一起使用，以将应用程序.cyacd2 文件下载并安装到设备中。请参阅**附录 A.DFU 主机工具**

5 PSoC 6 MCU DFU 代码示例

本应用笔记中有几个代码示例。他们演示了可以完成 DFU 操作的不同方法。

有关代码示例和其他文档的完整列表，以及 **www.cypress.com** 上的下载链接，请参阅**相关文档**。

Table 4 列出了代码示例及概述:

Table 4 PSoC 6 MCU DFU 代码示例列表

CE#	标题	说明
支持 ModusToolbox IDE 和 PSoC Creator		
CE213903	PSoC 6 MCU DFU 基础	一组演示基本的 DFU 操作的示例;见 **Figure 6**，内存映射 A: 使用各种通信信道从主机下载应用程序: UART，I2C 和 SPI 将下载的应用程序安装到用户闪存中 验证应用程序，然后将控制权转移到该应用程序
只支持 PSoC Creator		
CE216767	具有蓝牙低功耗 (Bluetooth LE) 连接 DFU 的 PSoC 6 MCU	与 CE213903 相同，但使用 Bluetooth LE 作为通信通道
CE220959	具有外部存储器的 PSoC 6 MCU Bluetooth LE DFU	与 CE216767 相同，但将应用程序临时保存在外部存储器中，然后将其复制到用户闪存中的最终目标位置
CE220960	具有可升级堆栈的 PSoC 6 MCU Bluetooth LE DFU	与 CE216767 相同，此外，可以升级 Bluetooth LE 堆栈
CE221984	PSoC 6 MCU 双应用程序 DFU	与 CE213903 相同，但演示了使用出厂默认 (“黄金映像”) 模式引导加载两个应用程序，通信通道是 I2C。
CE222802	具有加密和签名的 DFU	与 CE213903 相同，但应用程序已加密并签名以进行验证，DFU 解密应用程序并验证其签名，通信通道是 UART。
以下代码示例已计划或正在开发中:		
CE2xxxxx	USB HID DFU	与 CE213903 相同，但使用 USB HID 作为通信通道
CE2xxxxx	USB 大容量存储 DFU	与 CE213903 相同，但使用 USB 大容量存储作为通信通道
CE2xxxxx	具有多个应用程序的 DFU	与 CE221984 相同； 每个应用程序可以更新另一个应用程序；参见 **Figure 6**，内存映射 B
CE2xxxxx	具有外部存储器源的 PSoC 6 MCU DFU	与 CE213903 相同，但 DFU 源是外部存储器
CE2xxxxx	嵌入式主机程序	与 **AN60317**，PSoC 3 和 PSoC 5LP I2C DFU 中描述的类似

大多数代码示例由多个单独的应用程序组成，称为“App0”和“App1”。在某些情况下，还包括其他应用程序，称为“App2”，“App3”等。每个应用程序都是 ModusToolbox IDE 中一个单独应用或 PSoC Creator 中的一个单独项目。

通常，所有应用程序都位于相同的 ModusToolbox 或 PSoC Creator 工作区中。如**设计应用程序**中所述，项目可以存在于单独的工作区中，也可以存在于计算机上的独立区域。

通常 App0 执行 DFU 操作;它下载并安装其他应用程序。一个值得注意的例外是 **CE220960**，带可升级堆栈的 Bluetooth LE DFU - 代码示例中，App1 下载并安装 App2; App0 只是将控制转移到其他应用程序之一。

PSoC 6 MCU DFU 代码示例

基本代码示例 **CE213903** 有多个'_App0_ ...'项目;每个项目都有不同的通信通道。其他代码示例演示了高级通信通道，如 Bluetooth LE 和 USB。

代码示例支持 **CY8CKIT-062-BLE PSoC 6 BLE Pioneer Kit**。应用程序通常以不同的速率或不同的颜色闪烁套件 LED，从而可以轻松判断当前正在运行的应用程序。将代码示例移植到另一个套件或您的系统是一项简单的任务。

通常，应用程序的设计使您可以通过按住工具包按钮 0.5 秒来将控制从一个应用程序转移到另一个应用程序。Bluetooth LE DFU 代码示例使用立即警报服务 (IAS) 在应用程序之间传输控制。

5.1 如何构建代码示例

以下是逐步说明，说明如何构建 **Table 4** 中列出的每个 DFU 代码示例，并为您的应用程序进行可选的修改。有关更多信息，请参阅**如何使用 SDK**。

Note: *在构建代码示例时，在许多情况下，最简单的方法是将现有代码示例的一部分复制到项目中。然后可以针对您的应用程序修改复制的部分。要复制的部分列在每组指令中。*

此指导基于赛普拉斯 ModusToolbox 和 PSoC Creator IDE; 它们可以适用于其他 IDE。

5.2 PSoC 6 MCU 基础 DFU

本节介绍如何在代码示例 **CE213903** 中构建基本 DFU。有五个一般步骤:

1. 创建 App0 和工作区。
2. 将 App0 配置为 DFU。
3. 添加 App1。
4. 将 App1 配置为可安装的应用程序。
5. 构建并测试 DFU 和应用程序。

这些步骤适用于 ModusToolbox IDE 和 PSoC Creator。

5.2.1 ModusToolbox 指南

1. **创建 App0 和工作区**

启动 ModusToolbox IDE，并为您的应用程序选择或创建工作区，如 **Figure 14** 所示:

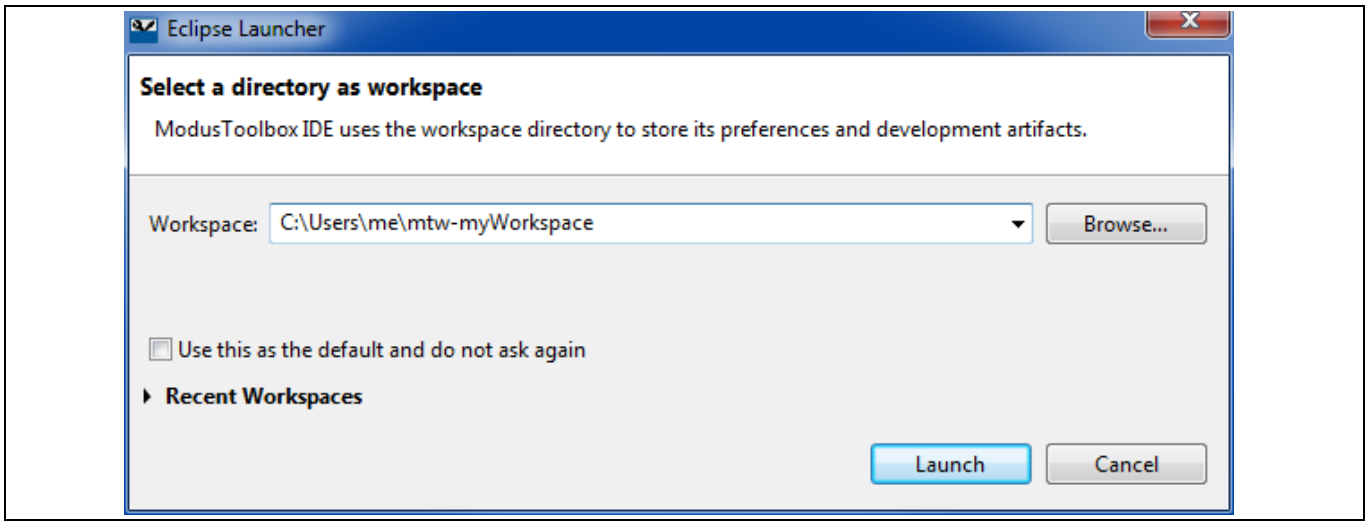


Figure 14　定义 ModusToolbox 工作区

创建一个新的应用程序。最简单的方法是在 **Quick Panel** 窗口中单击 **New Application**。在下一个对话框中，单击 **Dev/Eval Kit**，然后为 **CY8CKIT-062-BLE PSoC 6 BLE Pioneer Kit** 套件选择 CY8CKIT-062-BLE。点击 **Next**。

在下一个对话框 **Starter Application** 中，选择 **EmptyPSoC6App**，然后更改名称。CE213903 使用“PSoC6DfuApp0Uart”，“PSoC6DfuApp0I2c”和“PSoC6DfuApp0Spi”；选择一个名称，该名称指示这是执行 DFU 的应用程序。您还可以指示主机通信通道。点击 **Next**。在“Summary”对话框中，确认选择，然后单击 **Finish**。

2. **配置 App0 为 DFU**

要将 DFU 功能添加到应用程序，您必须:

- 在设计中添加一个主机通信通道
- 根据需要添加其他外设
- 添加 DFU 代码

2a. 添加一个通信通道。在项目<名称> _config 中打开 design.modus。选择一个通道类型，并将其配置为与 CY8CKIT-062-BLE 套件一起使用:

- 对于 UART: 将 UART 添加到 SCB 5，并进行以下更改: Oversample = 12，时钟=任何方便的时钟，RX = P5 [0]，TX = P5 [1]。您可以选择为 UART 和 UART 引脚分配名称。
- 对于 I²C: 将 I2C 添加到 SCB 3，并从默认值进行以下更改: 数据速率 (kbps) = 400，时钟=任何方便的时钟，SCL = P6 [0]，SDA = P6 [1]。您可以选择为 I2C 和 I2C 引脚分配名称。
- 对于 SPI: 将 SPI 添加到 SCB 6，并进行以下更改: Clock =任何方便的时钟，SCLK = P12 [2]，MOSI = P12 [0]，MISO = P12 [1]，SS1 = P12 [4]。您可以选择为 SPI 和 SPI 引脚分配名称。

2b. 根据需要添加其他外设。为了与 CE213903 兼容，请配置套件 LED 的引脚和按钮支持:

配置引脚 P0 [3] 如下: 名称= KIT_RGB_R，驱动模式=强驱动输入缓冲器关闭。确认初始驱动器状态=高 (1) 。

配置引脚 P0 [4] 如下: 名称= KIT_BTN1，驱动模式=电阻上拉输入缓冲器。确认初始驱动器状态=高 (1) 。

2c. 添加 DFU 代码。右键单击项目<名称> _mainapp，然后选择 ModusToolbox Middleware Selector。选择 Middleware DFU SDK Core，然后 DFU SDK transport UART 或其他传输选项之一。

对<name> _mainapp_cm0p 进行相同的操作，仅选择 DFU SDK 内核。所需的源.c，.h 和链接描述文件 (例如.ld) 会自动添加到两个项目中。

对于 I2C 和 SPI 通道，请在<name> _mainapp 项目的 Source 文件夹中编辑文件 dfu_user.c，如下所示:

- 更改`#include "transport_uart.h"`为:
  `#include "transport_i2c.h" or`
  `#include "transport_spi.h"`
- 将“`UART_Uart`”的 5 个实例更改为 “`I2C_I2c`” or “`SPI_Spi`”

将 DFU 代码集成到<name> _mainapp 和<name> _mainapp_cm0p 中的 main.c 文件中。最简单的方法是复制并粘贴代码示例 **CE213903** 中的代码，然后为您的应用程序修改代码。

右键单击项目<名称> _mainapp，然后选择 **Properties**。在 C/C++ **Build** > **Settings** > **Tool Settings > GNU ARM Cross C Linker** > **General** 中，将 **Script files** 条目中的文件更改为 dfu_cm4.ld，以使用项目中的自定义脚本进行链接。对项目<名称> _mainapp_cm0p 和 dfu_cm0p.ld 执行相同的操作。

2d. 构建项目。最简单的方法是选择<name> _mainapp，然后在 Quick Panel 中单击 Build <name> Applicatio。

3. **添加 App1**。

如前所述，使用 ModusToolbox IDE，您可以将应用程序添加到与 DFU 应用程序相同的工作区中，也可以将它们添加到单独的工作区、文件夹中，或同时在这两个空间中。这些指令说明如何将 App1 添加到与 DFU App0 相同的工作区中。

Note: *您可以创建与同一 DFU 系统一起使用的任意数量的可安装应用程序。在开始开发应用程序之前，建议为您的整个系统开发需求制定 DFU 和应用程序计划。请参阅**确定系统中的应用程序**。*

以与上述相同的方式创建一个新的应用程序。选择与 App0 相同的工具包。更改名称； CE213903 使用“PSoC6DfuApp1”。选择一个名称，指明这是可下载的应用程序。

4. **将 App1 配置为可安装的应用程序**

4a. 根据需要添加其他外设。编辑此应用程序的 *design.modus*。为了与 CE213903 兼容，请配置套件 LED 的引脚和按钮支持:

配置引脚 P0 [3] 如下: 名称= KIT_RGB_R，驱动模式=强驱动输入缓冲器关闭。确认初始驱动器状态=高 (1)。

配置引脚 P0 [4] 如下: 名称= KIT_BTN1，驱动模式=电阻上拉输入缓冲器。确认初始驱动器状态=高 (1)。

4b. 添加 DFU 代码。右键单击项目<名称> _mainapp，然后选择 **ModusToolbox Middleware Selector**。选择中间件 **DFU SDK Core**。对<名称> _mainapp_cm0p 执行相同的操作。所需的源.c，.h 和链接描述文件 (例如.ld) 会自动添加到两个项目中。仅将 DFU SDK Core 添加到应用程序中是为了将控制权转移到 App0。

如下编辑项目<名称> _mainapp 中的文件 dfu_cm4.ld 和项目<名称> _mainapp_cm0p 中的 dfu_cm0p.ld。这将更改 App1 的应用程序编号，并正确定位 App1 的闪存和 RAM 内存:

```
/*
* DFU SDK specific: aliases regions, so the rest of code does not use
* application specific memory region names
*/
REGION_ALIAS("flash_core0", flash_app1_core0);
REGION_ALIAS("flash",       flash_app1_core1);
REGION_ALIAS("ram",           ram_app1_core1);

/* DFU SDK specific: sets an app Id */
__cy_app_id = 1;
```

Note: *重要的是不要更改这些.ld 文件中的其他内容。除上述编辑外，它们必须与 App0 .ld 文件相同。*

将 DFU 代码集成到<name> _mainapp 和<name> _mainapp_cm0p 中的 main.c 文件中。最简单的方法是复制并粘贴代码示例 CE213903 中的代码，然后为您的应用程序修改代码。

将代码示例中的文件 *dfu_postbuild.bash* 复制到项目<名称> _mainapp 中。见**附录 E**。

PSoC 6 MCU DFU 代码示例

右键单击项目<名称> _mainapp_cm0p，然后选择 **Properties**。在 **C/C++ Build** > **Settings** > **Tool Settings** > **GNU ARM Cross C Linker** > **General** 中，将 **Script files** 条目中的文件更改为 dfu_cm0p.ld，以使用项目中的自定义脚本进行链接。

对项目<名称> _mainapp 和 dfu_cm4.ld 执行相同的操作。然后，在同一窗口中选择 **Build Steps**，然后将 **Post-build steps Command** 替换为以下内容:

```
${cy_sdk_install_dir}/tools/modus-shell-1.0/bin/bash --norc --noprofile
${workspace_loc:/PSoC6DfuApp1_mainapp}/dfu_postbuild.bash
${cy_sdk_install_dir}/tools/cymcuelftool-1.0/bin/cymcuelftool
${workspace_loc:/PSoC6DfuApp1_mainapp_cm0p}/${config_name:PSoC6DfuApp1_mainapp_cm
0p}/PSoC6DfuApp1_mainapp_cm0p.elf
${workspace_loc:/PSoC6DfuApp1_mainapp}/${config_name:PSoC6DfuApp1_mainapp}/PSoC6D
fuApp1_mainapp.elf ARM_CM4
```

创建输出.cyacd2 文件，该文件用于下载 DFU 应用程序。见**附录 C**。

4c. 构建项目。最简单的方法是选择<name>_mainapp，然后在 Quick Panel 中单击 **Build <name> Application**。

5. **测试 DFU 和应用程序**

将 App0 编程到 **CY8CKIT-062-BLE PSoC 6 BLE Pioneer Kit** 中。然后使用设备固件更新主机工具 (DHT) 将 App1 下载到套件中； 请参阅**附录 A** 中的工具使用说明。

您可以使用 **CE213903** 文档的操作部分中的说明测试 DFU 和应用程序。

5.2.2 PSoC Creator 指南

1. **创建 App0 和工作区。**

您可以同时创建 PSoC Creator 项目及其包含的工作区。在 PSoC Creator 中，选择 **File**> **New**> **Project** .... 在 Create Project 窗口将出现“选择项目类型”面板，如 **Figure 15** 显示。

A. 单击目标设备。

B. 在第一个下拉菜单中，选择 PSoC 6。

C. 在下一个下拉菜单中，选择 CY8C6347BZI-BLD53。这是安装在 **CY8CKIT-062-BLE PSoC 6 BLE Pioneer Kit** 套件中的 PSoC 6 MCU 器件。

D. 单击下一步。

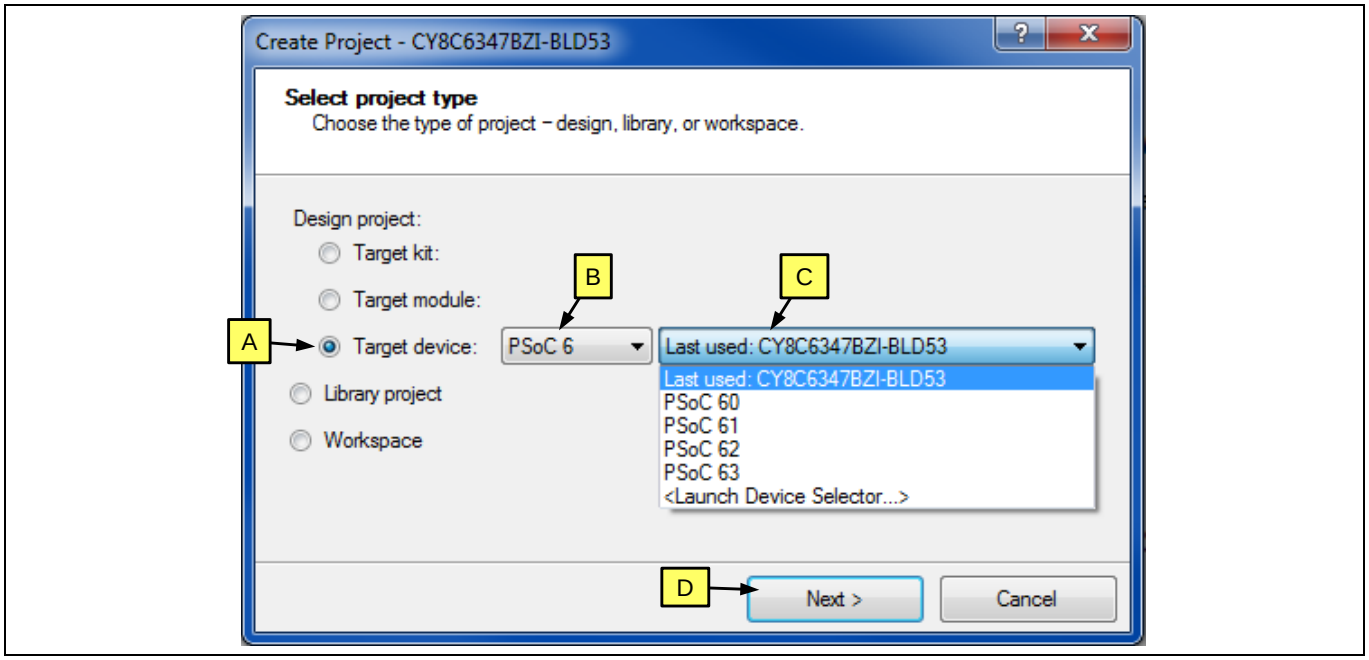


Figure 15　PSoC Creator 中选择目标器件

在接下来的两个面板上单击“**Next**”，**Select project template** 和 **Set target IDE(s)**。将出现“创建项目”面板，如 **Figure 16** 所示。

输入工作区名称，然后选择其位置。将项目名称设置为“app0”。单击 **Finish**。

在指定的位置创建一个新文件夹; 文件夹名称与工作区名称相同。在工作区文件夹中创建文件夹 *app0.cydsn*; *.cydsn* 文件夹包含所有项目文件。

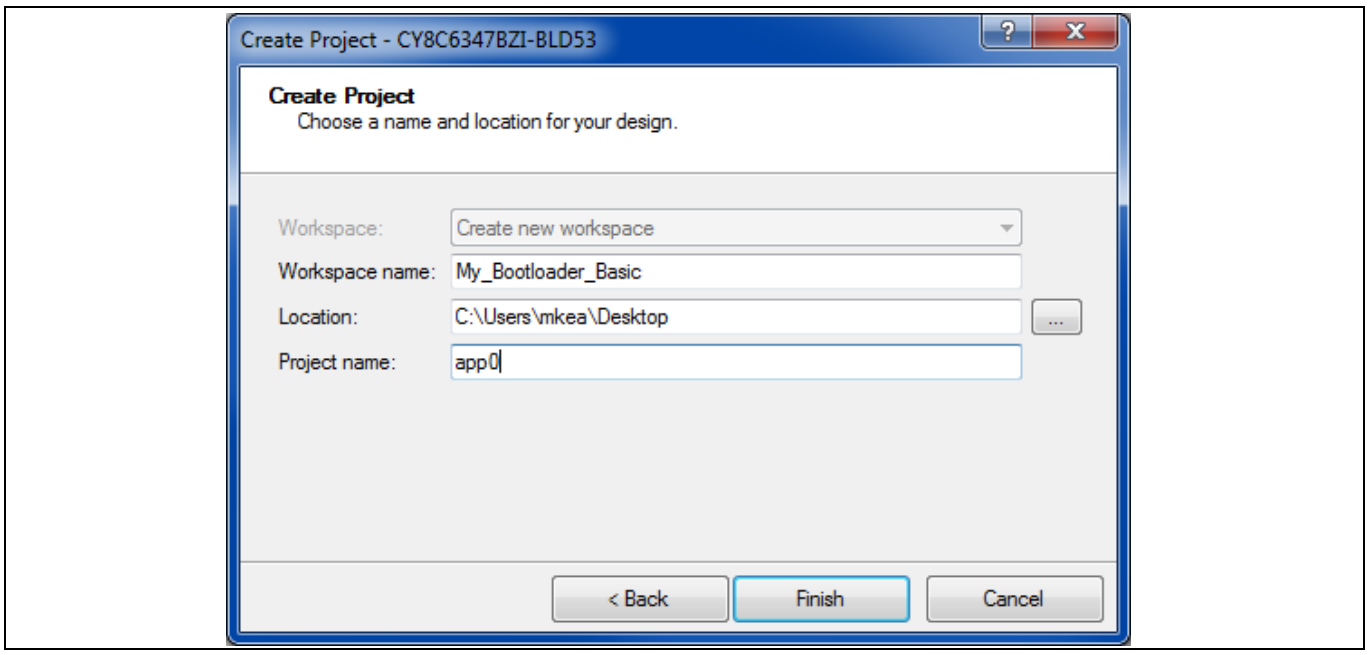


Figure 16　创建 PSoC Creator 项目

新工作区和项目文件显示在 Workspace Explorer 窗口中; 见 **Figure 11**

2. **将 App0 配置为 Bootloader。**

要将 Bootloader 功能添加到应用程序，您必须:

- 将通信组件添加到项目原理图中。使用此组件与 Bootloader 主机通信。
- 根据应用程序的需要，将其他组件添加到项目原理图中
- 将 DFU SDK 和模板文件添加到项目中

2a. 添加通信组件。打开或双击项目原理图 (Workspace Explorer 窗口中的文件 *TopDesign.cysch*) 。

打开组件目录，然后导航到所需的通信组件，例如 UART，如 **Figure 17** 所示。将组件拖到原理图上。

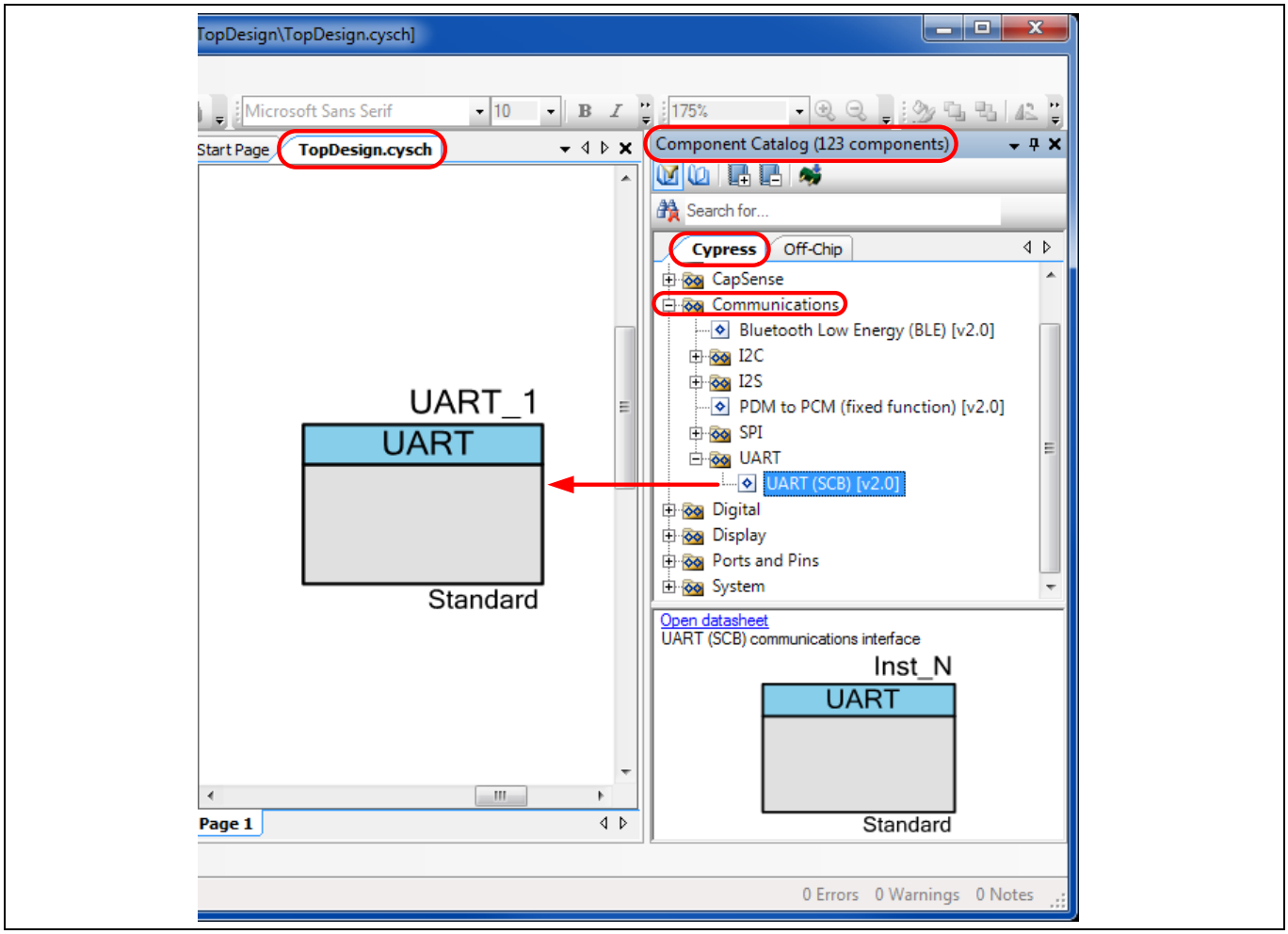


Figure 17 将 PSoC Creator 通信组件添加到 Bootloader 项目

双击原理图上的 **Component** 以配置其参数，例如波特率，位数等，如 **Figure 18** 所示。

Figure 18 显示了此代码示例的更改参数设置; 请注意，**Name** 已从 UART_1 更改为 UART，I2C_1 更改为 I2C 或 SPI_1 更改为 SPI。建议使用以后复制到项目的默认 DFU SDK 文件。

完成组件配置后，单击“**OK**”。

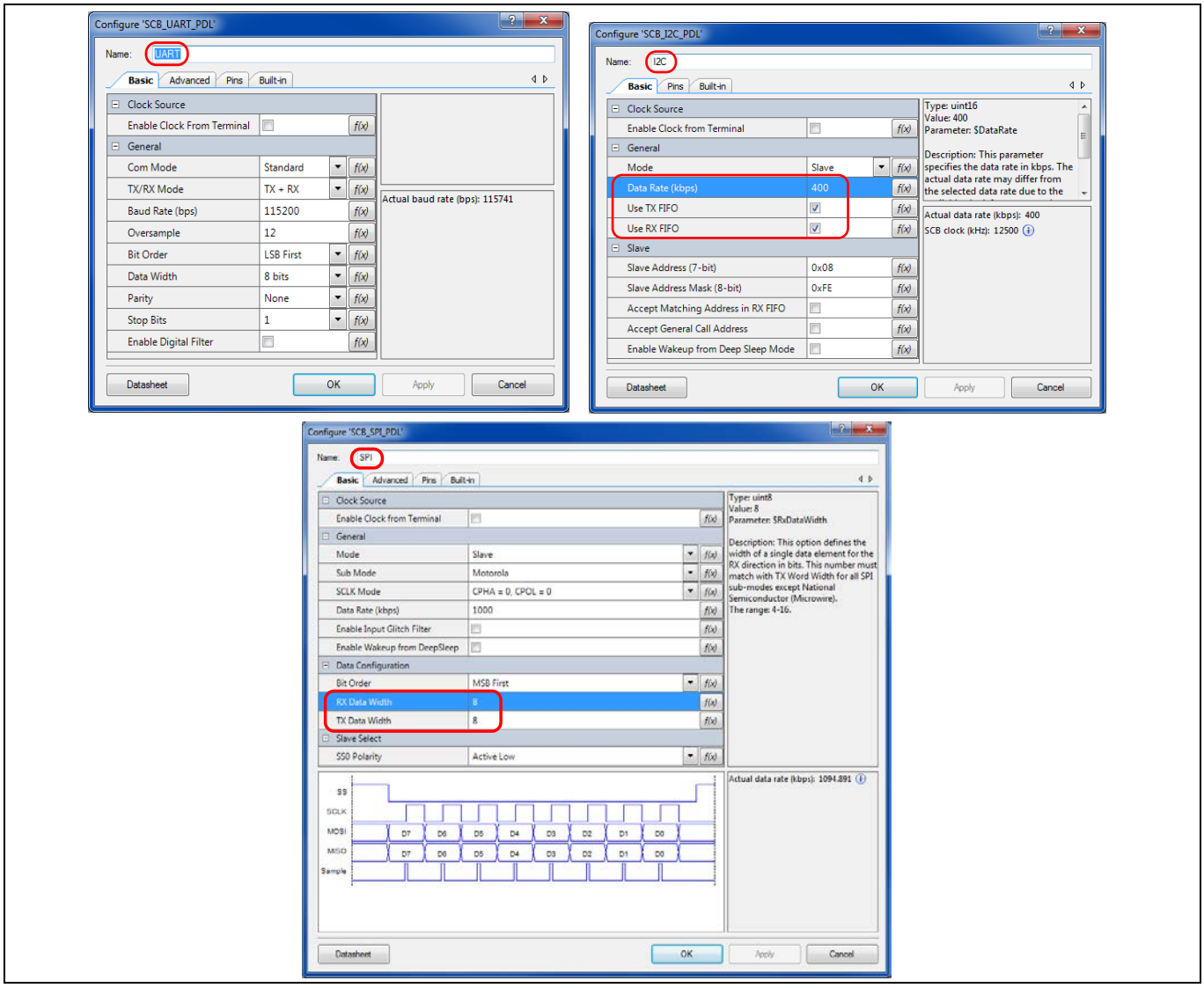


Figure 18　配置 PSoC Creator 通信组件: UART, I2C, 或 SPI

2b. 添加其他组件。将其他 PSoC Creator 组件 (如 LED 和按钮引脚) 添加到项目原理图中。最简单的方法是从代码示例 **CE213903** 复制并粘贴项目原理图的一部分，然后根据应用程序的需要修改原理图。

在 Design Wide Resources 窗口的 **Pins** 选项卡中，将 UART，I2C，SPI 和其他组件引脚连接到相应的物理引脚。使用 **CE213903** 和套件指南作为指导。

2c. 将 Bootloader 代码添加到项目中。首先，将 DFU SDK 合并到项目中 - 参见 **Figure 10**。

然后选择 **Build> Generate Application**。完成后，项目应该如 **Figure 11** 所示，添加了一些启动和其他非 DFU 文件。

对于 I2C 和 SPIDFU，请在项目 Shared Files 文件夹中编辑文件 *bootload_user.c*，如下所示:

- 更改 `#include "transport_uart.h"` 为
  `#include "transport_i2c.h" or`
  `#include "transport_spi.h"`
- 更改 “`UART_Uart`” 的 5 个实例为 “`I2C_I2c`” or “`SPI_Spi`”

PSoC 6 MCU DFU 代码示例

将 DFU 代码添加到 *main_cm0p.c* 和 *main_cm4.c* 文件中。最简单的方法是复制并粘贴代码示例 **CE213903** 中的代码，然后修改应用程序的代码。

2d. 构建项目。更改项目构建设置以使用模板链接描述文件; 请参见 **Figure 12**。然后选择 **Build**> **Build app0**。

3. **添加 App1**。

如前所述，使用 PSoC Creator，您可以将其他项目 (应用程序) 添加到与 DFU 应用程序相同的工作区，也可以将它们放在单独的工作区、文件夹或两者中。这些说明显示如何将 App1 添加到与 DFUApp0 相同的工作区。

Note: *您可以创建任意数量的可安装应用程序，这些应用程序使用相同的 DFU。在开始开发应用程序之前，推荐为引导程序和应用程序制定一个计划，以满足整个系统开发需求。请参阅**确定系统中的应用程序**。*

在 PSoC Creator Workspace Explorer 窗口中，右键单击 **Workspace**，然后选择 **Add**> **New Project** ...，如 **Figure 19** 所示:

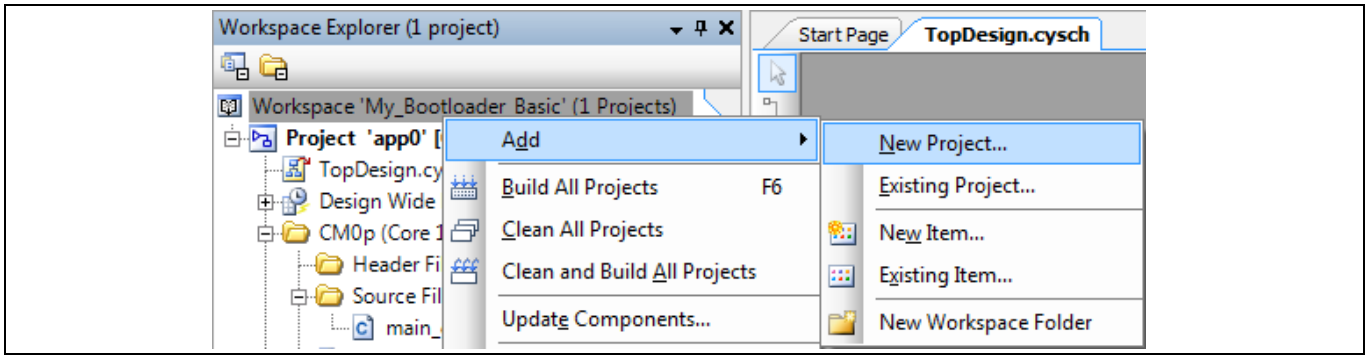


Figure 19 PSoC Creator Workspace 中添加新项目

在“创建项目”窗口将出现“选择项目类型”面板; 请参见 **Figure 15**。确保所选设备与 App0 相同。点击下一步。

在接下来的两个面板上单击下一步: 选择项目模板和设置目标 IDE。将出现“创建项目”面板。

将项目名称设置为“app1”，如 **Figure 20** 所示。默认位置是工作区文件夹。单击完成。

在工作区文件夹中创建一个新的 *app1.cydsn* 文件夹; *.cydsn* 文件夹包含所有项目文件。

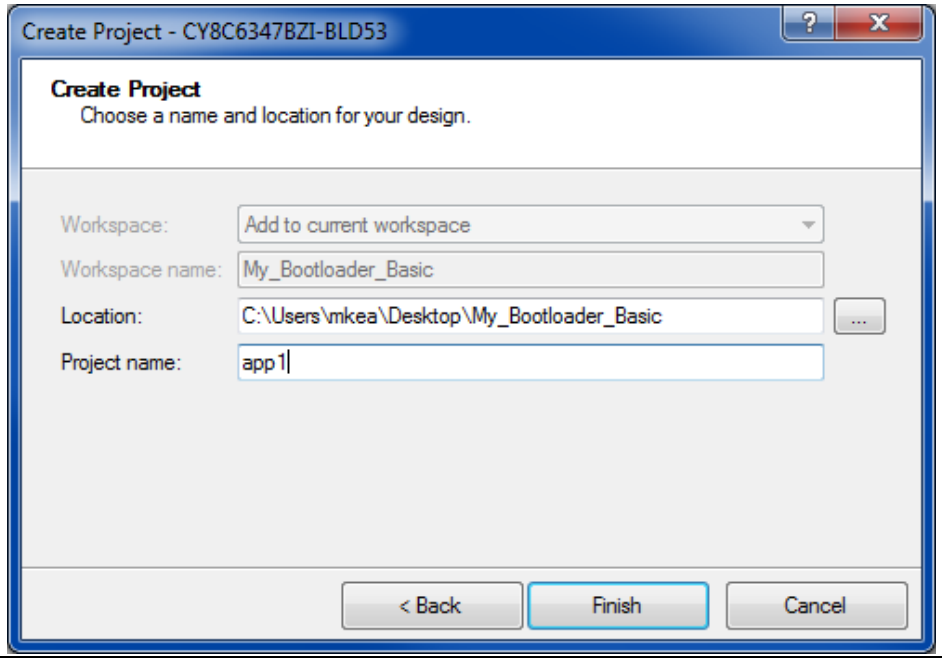


Figure 20　创建第二个项目

4. **将 App1 配置为可安装的应用程序。**

Note:　　App1 应使用与 App0 相同的工具链 (GCC 或 MDK) 构建，否则应用程序传输可能会失败。

4a. 将 LED 和按钮引脚等其他组件添加到项目原理图中。最简单的方法是从代码示例 **CE213903** 复制并粘贴项目原理图的一部分，然后修改应用程序的原理图。

在 Design Wide Resources 窗口的 **Pins** 选项卡中，将组件引脚连接到相应的物理引脚。使用 CE213903 和套件指南作为指导。

4b. 将 DFU SDK 和模板文件添加到项目中。将 DFU SDK 合并到项目中;请参见 **Figure 10**。仅检查 **Core** 框，因为 App1 是可下载的应用程序，并且没有 DFU 功能。

然后，选择 **Build**> **Generate Application**。完成后，项目应该如 **Figure 11** 所示，添加了一些启动和其他非 DFU 文件。更改项目构建设置以使用模板链接脚本文件;见 **Figure 12**。

将构建后的批处理文件添加到 Shared Files 文件夹，如第 **4.3** 节第 8 步所述。然后将一个构建后的命令添加到 Cortex-M4 构建中 - 参见 **Figure 13**。

4c. 编辑文件。按**第 4.3 节步骤 6** 中所述编辑链接脚本文件。

将 DFU 代码添加到 *main_cm0p.c* 和 *main_cm4.c* 文件中。最简单的方法是复制并粘贴代码示例 **CE213903** 中的代码，然后修改应用程序的代码。

确保已将数组添加到 *main_cm4.c* 文件中以获取应用程序签名，如**第 4.3 节步骤 8 中**所述。

4d. 构建项目。完成所有构建设置更改并完成文件编辑后，选择 **Build** > **Build app1**。

5. **测试 DFU 和应用**

将 app0 编程到 **CY8CKIT-062-BLE PSoC 6 BLE Pioneer Kit** 套件中。然后使用 DFU 主机程序 (BHP) 将 app1 引导到套件中; 请参阅附录 A 中的 BHP 使用说明。

PSoC 6 MCU DFU 代码示例

您可以使用 **CE213903** 文档“操作”部分中的说明测试 DFU 和应用程序。

5.3 PSoC 6 MCU Bluetooth LE DFU

本节介绍如何在代码示例中构建三个不同的 Bluetooth LE DFU:

- **CE216767**, PSoC 6 MCU，带蓝牙低功耗 (Bluetooth LE) 连接 DFU
- **CE220959**, 带外部存储器的 PSoC 6 MCU Bluetooth LE DFU
- **CE220960**, 带升级堆栈的 PSoC 6 MCU Bluetooth LE DFU (参见**此小节**)

这些代码示例与**基本的 DFU 代码示例**类似，具有一些其他功能:

- CE216767 将新应用程序直接下载到闪存中，如 **Figure 21** 所示。
- CE220959 将新应用程序下载到外部 NVM (套件 IC U4，赛普拉斯 512-Mbit 串行 NOR 闪存) 中的临时存储中，然后将其复制到 PSoC 6 MCU 器件闪存中的最终目标，如 **Figure 22** 所示。
- DFU (App0) 和应用程序 (App1) 各自都有自己独立的 Bluetooth LE 堆栈代码副本，如 **Figure 21** 和 **Figure 22** 显示。每个副本占用超过 128 KB 的闪存。

对于具有单个共享 Bluetooth LE 堆栈的 DFU，请**参阅具有可升级堆栈的 Bluetooth LE DFU**。

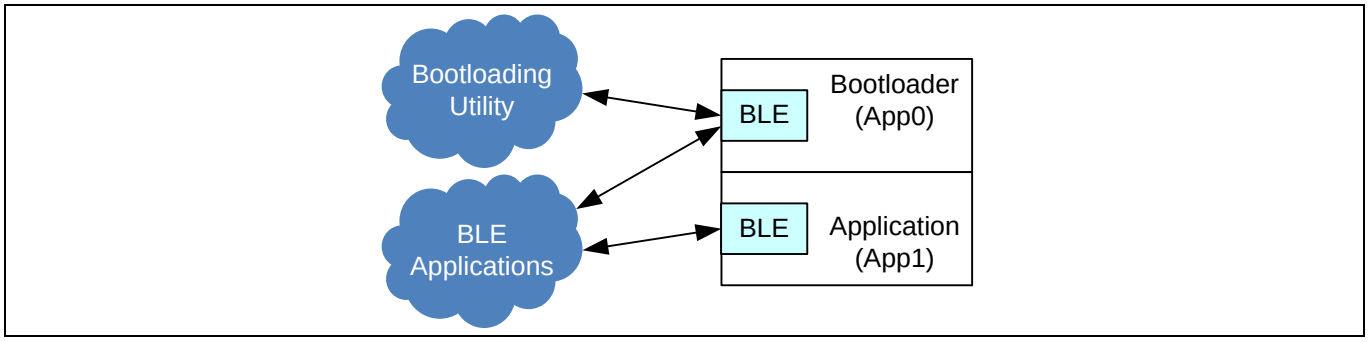


Figure 21 CE216767 数据控制流程

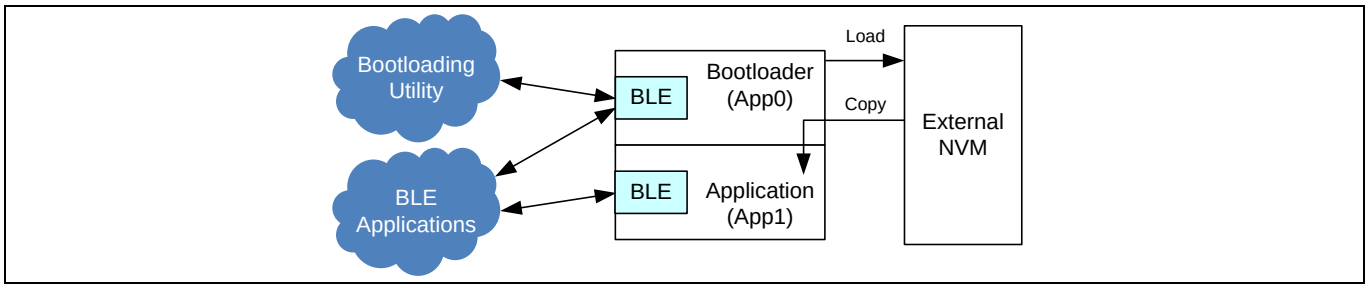


Figure 22 CE220959 数据控制流程

- Bluetooth LE 堆栈代码可由 CPU 或两个 CPU 执行; 这是 Bluetooth LE 组件配置对话框中的可选选项。代码示例演示了此功能; 在 App0 中，CM4 CPU 执行 Bluetooth LE 堆栈，在 App1 中，CM0+ CPU 执行 Bluetooth LE 堆栈。
- App0 和 App1 在其 Bluetooth LE 组件配置中启用不同的服务:
  - App0: DFU 和立即警报服务 (IAS)。当应用程序未引导时，将实现 IAS 服务。
  - App1:人机接口设备 (HID) 和即时警报服务 (IAS)。

两个应用程序都使用 IAS 将控制从一个应用程序转移到另一个应用程序。

PSoC 6 MCU DFU 代码示例

- 代码示例广泛使用 **CY8CKIT-062-BLE PSoC 6 BLE Pioneer Kit** 资源。所有三个 RGB LED，用户按钮 SW2，Bluetooth LE 子系统和 USB-UART 桥都由两个应用程序 (App0 和 App1) 使用。

要构建代码示例，请执行以下操作。有关这些步骤的更多信息，请参考 **PSoC 6 MCU 基本 DFU** 或**如何使用 SDK**。

1. 创建 App0 和工作区，如 **Figure 23** 所示。

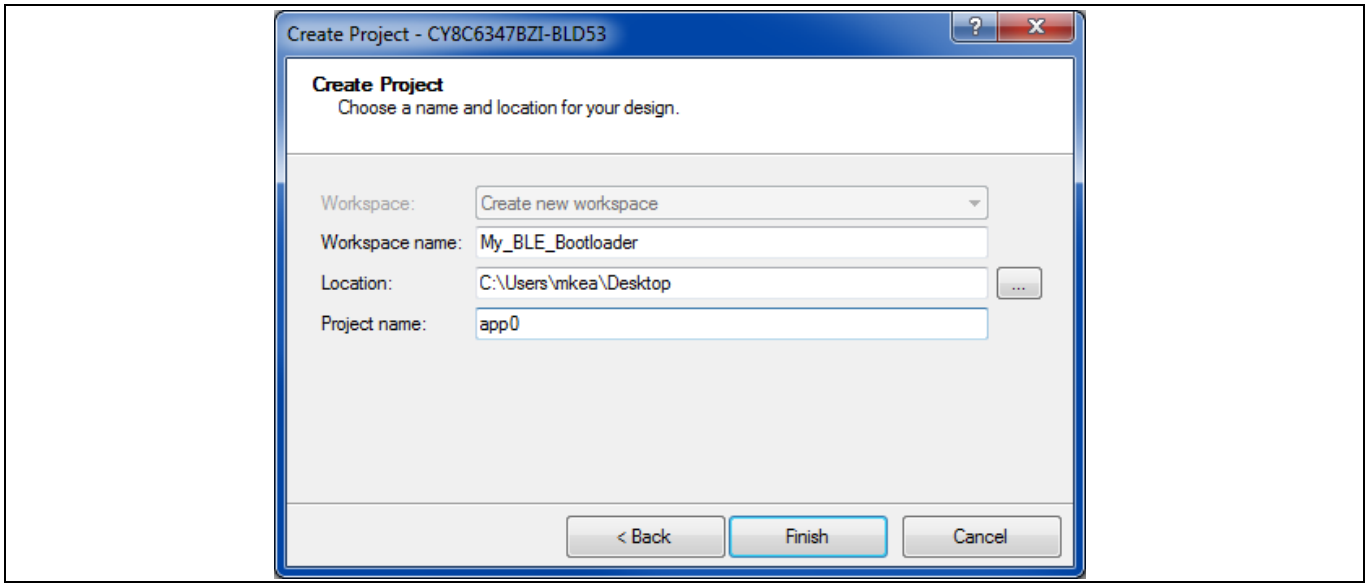


Figure 23 PSoC Creator 中创建 Bluetooth LE DFU App0

2. 将 Bluetooth LE 组件添加到 App0 顶部设计原理图中，并根据代码示例中的 Bluetooth LE 组件配置部分配置组件。具体而言，从默认配置进行以下更改:

- (可选，但建议使用默认的 DFU SDK 文件) 将组件名称更改为“BLE”。
- 常**规选项卡** (参见 **Figure 24**) : CPU 内核: 单核 (CM4 上的完整组件)

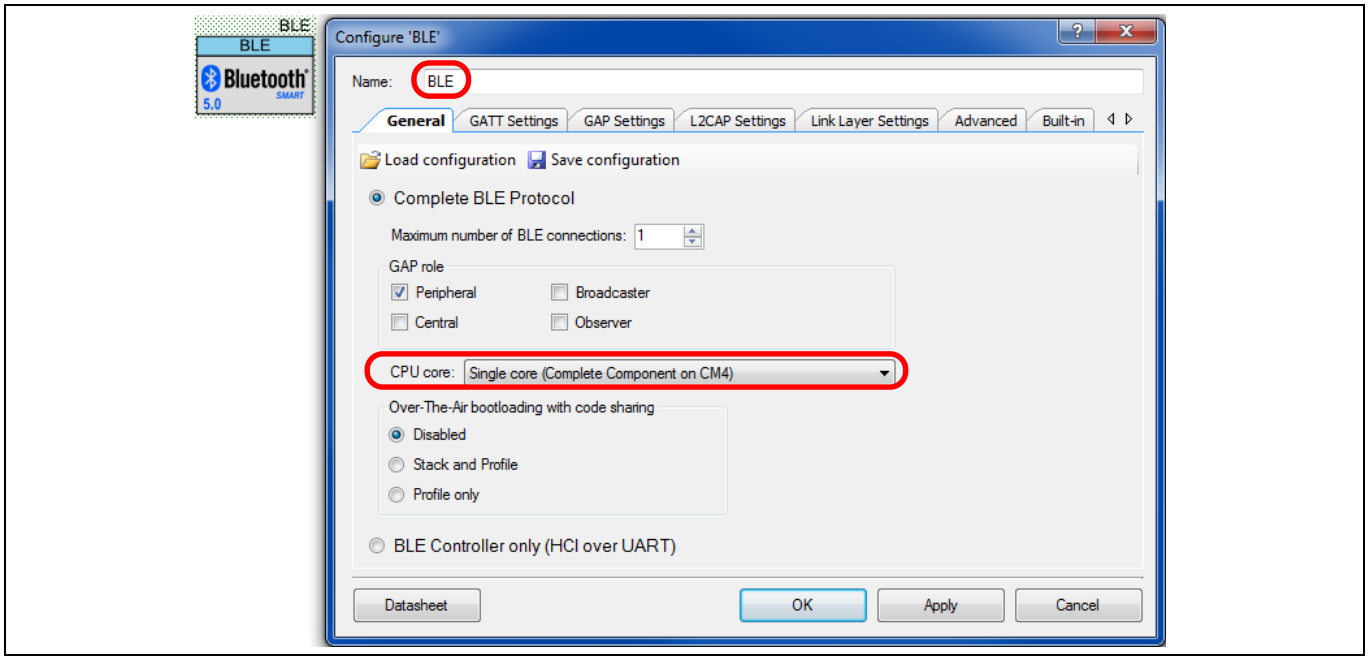


Figure 24 PSoC Creator Bluetooth LE 组件，常规选项卡配置

PSoC 6 MCU DFU 代码示例

- GATT **设置**选项卡 (参见 **Figure 25**) :
  - 通用访问，外设首选连接参数:
    - 最小连接间隔: **0x000C**
    - 最大连接间隔: **0x000C**
    - 连接监控超时倍数: **0x00C8**

  选择这些间隔以最小化引导加载时间。根据您的应用需要进行调整。

  - 右键单击 GATT 树中的“**Server**”节点，然后添加 DFU 服务。 (可选) 添加立即警报服务; 代码示例使用此服务在应用程序之间传输控制。无需编辑任何服务的特征。
  - 属性 MTU 大小 (字节) : **512**

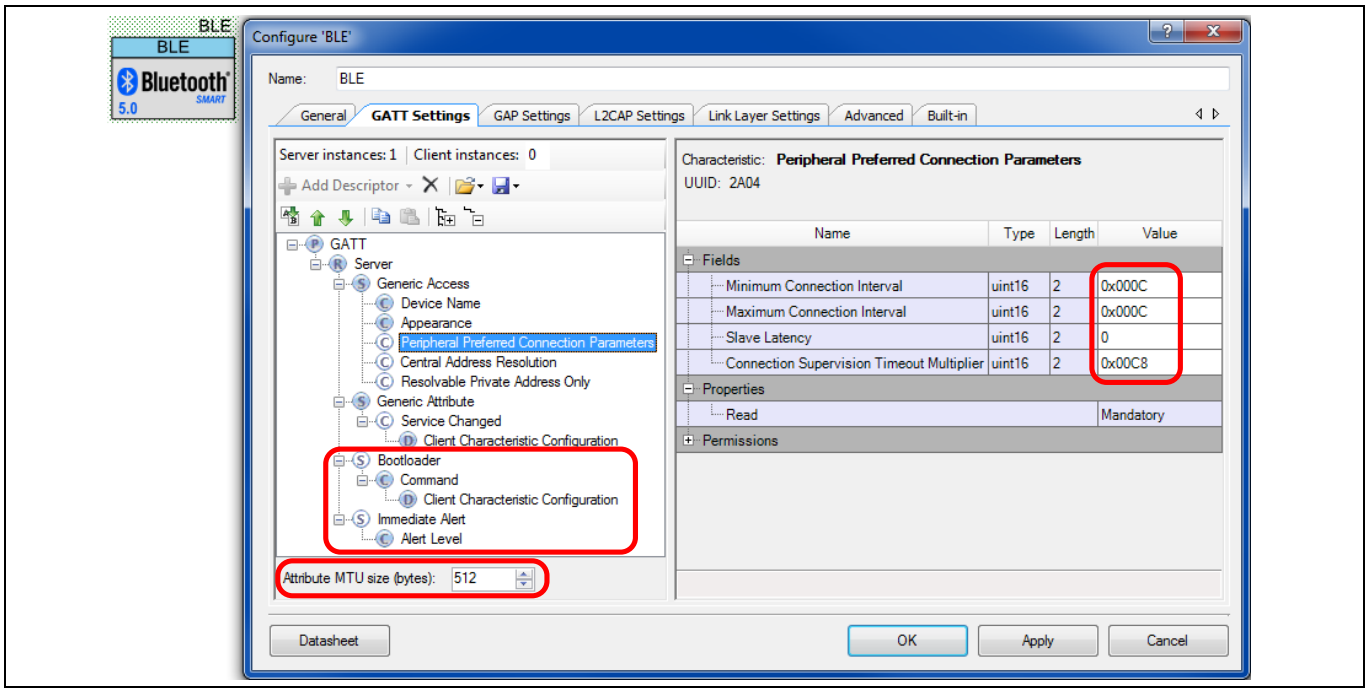


Figure 25　PSoC Creator Bluetooth LE 组件, GATT 设置选项卡配置

- **GAP 设置**选项卡:
  - 常规，设备名称: 代码示例使用“BLE DFU”; 根据您的应用需要使用不同的文本。
  - 外设配置 0，广告包: 选中本地名称并设置为完成。
  - 安全配置 0 (参见 **Figure 26**) ，安全级别: 未经身份验证的加密配对。
  - 安全配置 0，I/O 功能: 无输入无输出。
  - 安全配置 0，绑定要求: **无绑定**。

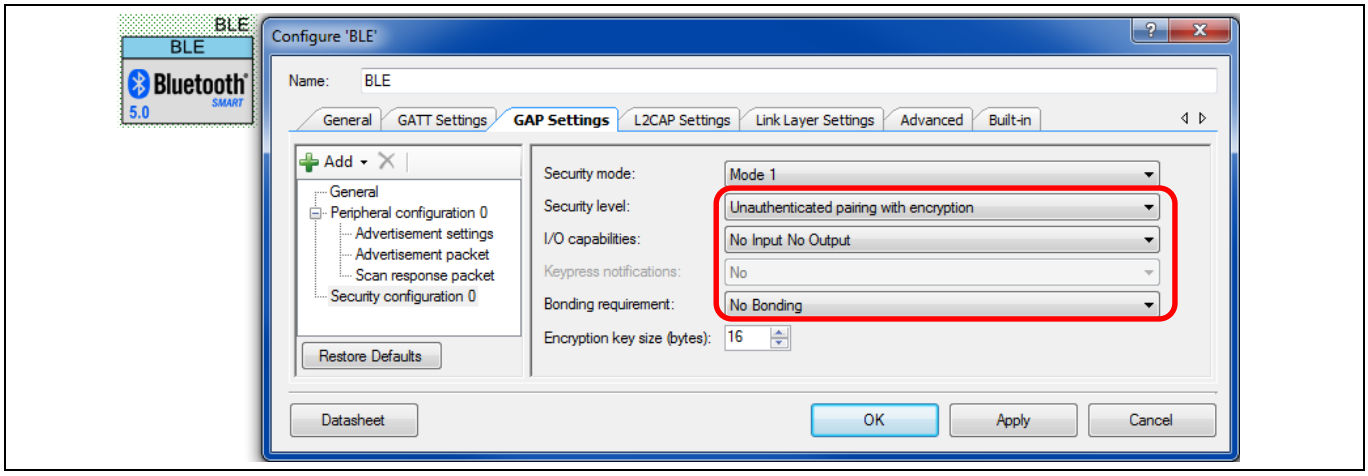


Figure 26 PSoC Creator Bluetooth LE 组件, GAP 设置选项卡配置

- 链接层设置选项卡:
  - 链路层最大 TX 和 RX 有效载荷大小 (字节) : 251，以实现更快的引导加载

3. 对于 **CE220959**，将 SMIF 组件添加到 App0 原理图，并根据 SMIF 组件配置部分配置组件。具体而言，从默认配置进行以下更改:

- (可选，但建议使用 CE220959 文件) 名称: SMIF
- 选中 **SMIF Datalines [2:3]**。外部存储器 IC 具有与 PSoC 6 MCU 器件的四 SPI (4 数据线) 接口。**SMIF Datalines [0:1]**框被默认选中。
- 取消选中 **Generate code from cy_smif.cysmif file**。文件 *cy_smif_memconfig.h* / *.c* 和 *smif_mem.h* / *.c* 已经提供了代码示例。

4. 根据需要，将其他组件 (即 LED 和按钮) 从 **CE216767** 或 **CE220959** 顶部设计原理图复制到顶部设计原理图。
5. 在 Design Wide Resources 窗口中:

- 引脚选项卡: 将组件引脚连接到相应的物理引脚。请注意，Bluetooth LE 组件不使用任何引脚。
- 时钟选项卡: 启用 WCO，从 WCO 获取 Clk_LF 和 Clk_Bak。这是在某些模式下操作 Bluetooth LE 组件所必需的;有关详细信息，请参阅 Bluetooth LE 组件数据表。
  - 对于 **CE220959**，启用 Clk_HF2 并将其频率设置为 50 MHz。最简单的方法是在 Clk_HF2 部分中选择除以 2 的选项。正确操作外部存储器接口需要频率限制。

6. 在项目构建设置中将 App0 配置为 DFU - 请参见 **Figure 10**。选择通信通道的 Bluetooth LE。
7. 选择 **Build** > **Generate Application**。这将创建 DFU 链接器文件，例如 *bootload_cm4.ld*。
8. 返回项目 Build Settings 并将 CM0+和 CM4 的 **Custom Linker Script** 设置为相应的 DFU 链接脚本文件 - 参见 **Figure 12**。
9. 在 *bootload_common.ld* 中编辑闪存和 RAM 内存区域大小，如下所示:

PSoC 6 MCU DFU 代码示例

```
MEMORY
{
    flash_app0_core0  (rx)  : ORIGIN = 0x10000000, LENGTH = 0x10000
    flash_app0_core1  (rx)  : ORIGIN = 0x10010000, LENGTH = 0x30000
    flash_app1_core0  (rx)  : ORIGIN = 0x10040000, LENGTH = 0x32000
    flash_app1_core1  (rx)  : ORIGIN = 0x10072000, LENGTH = 0x02000
. . .
    ram_app0_core0    (rwx) : ORIGIN = 0x08000100, LENGTH = 0x1F00
    ram_app0_core1    (rwx) : ORIGIN = 0x08002000, LENGTH = 0x8000
. . .
    ram_app1_core0    (rwx) : ORIGIN = 0x08000100, LENGTH = 0x1FF00
    ram_app1_core1    (rwx) : ORIGIN = 0x08020000, LENGTH = 0x20000
. . .
```

默认的 *bootload_common.ld* 为每个应用程序中的每个 CPU 分配相同数量的闪存和 RAM。

必须针对这些代码示例调整分配，因为:

- Bluetooth LE 堆栈很大
- 应最小化 App1 大小以减少引导加载时间

请注意，代码示例中对 App1 进行了相同的更改。MDK 和 IAR 链接脚本文件存在类似的内存布局。

10. 将文件 *debug.h* / *.c*，*ias.h* / *.c* 和 *bootload_user.h* / *.c* 从 **CE216767** 或 **CE220959** 文件夹复制并粘贴到 *app0.cydsn* 文件夹。覆盖现有的默认 *bootload_user.h* / *.c* 文件。此外，对于 **CE220959**，复制并粘贴 *cy_smif_memconfig.h* / *.c* 和 *smif_mem.h* / *.c* 文件。

- 调试和 ias (立即警报服务) 文件对于代码示例是唯一的，对于您的应用程序可能是可选的。
- 已编辑 bootload_user 文件以调用 Bluetooth LE 传输函数，而不是默认的 UART 传输函数。
- 对于 CE220959，SMIF 文件定义外部存储器配置并提供访问外部存储器的功能。

11. 在 PSoC Creator 工作区资源管理器窗口中，包含或移动上一步中的文件到项目 CM4 头文件和源文件文件夹，如 **Figure 27** 显示。

如果这些文件位于 Shared Files 文件夹中，它们可能包含在 CM0+ build 中，可能会导致编译错误。

12. 将 CM0+和 CM4 主代码从 CE216767 或 CE220959 复制到主文件。根据应用程序的需要更新主文件和其他源文件。
13. 构建 app0，并使用 app0 对工具包进行编程。

您可以通过安装 CE216767 或 CE220959 的 app1 来测试您的 app0。按照代码示例文档“操作”部分中的说明进行操作。

14. 如果您构建自己的 app1，请记住编辑以下文件:

- 三个链接器脚本文件，如步骤 9 和第 **4.3.2** 节步骤 6 中所述。
- 批处理文件，如第 **4.3.2** 节第 8 步所述。

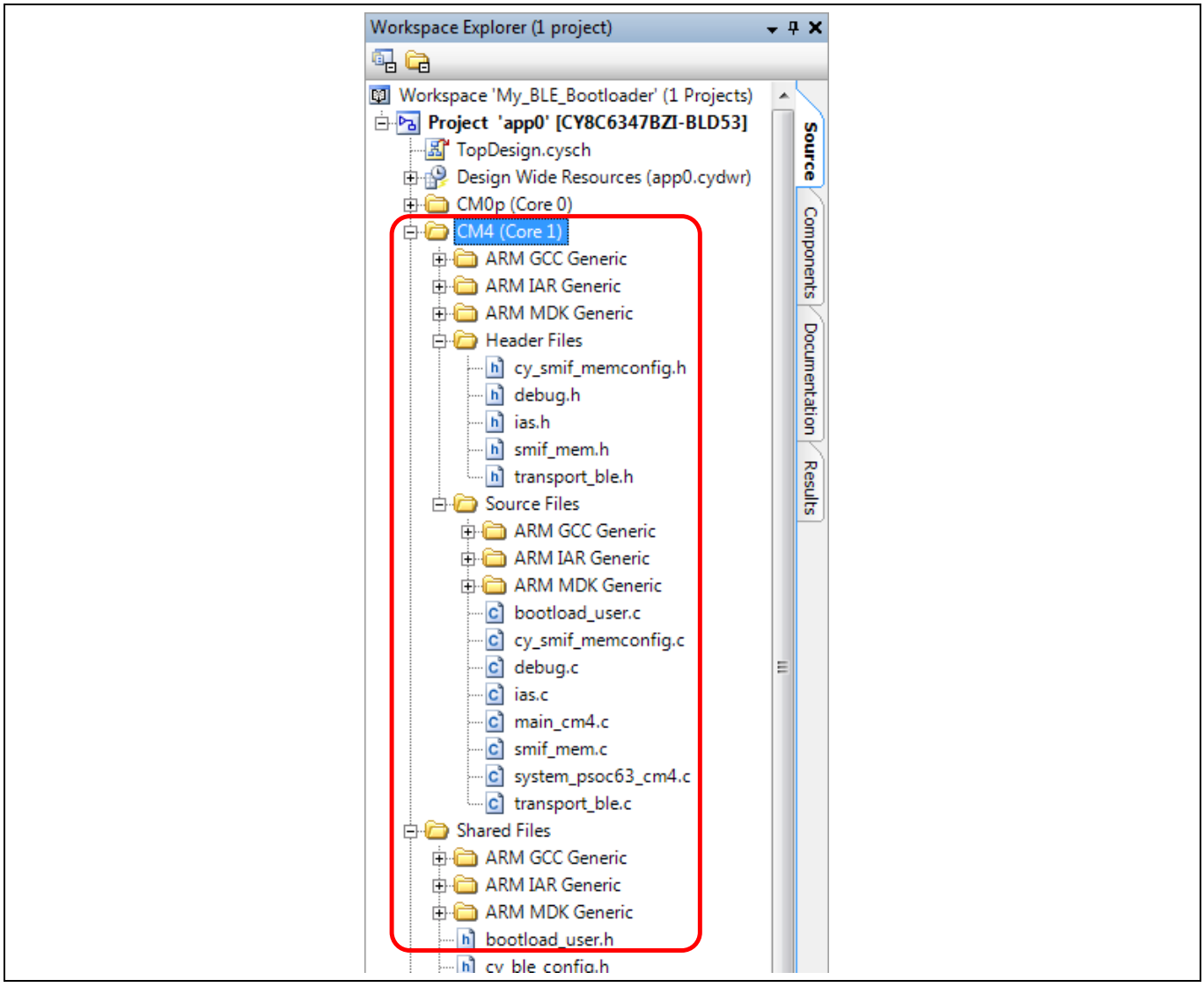


Figure 27 PSoC Creator CM4 文件夹中包含的源文件

5.3.1 带可升级堆栈的 Bluetooth LE DFU

此代码示例与**其他 Bluetooth LE DFU 代码示例**的不同之处在于它具有三个应用程序，如 **Figure 28** 所示。

在这个例子中，App0 不进行引导加载; 它只是一个发射器应用程序。它还将堆栈更新从临时位置复制到正确的位置，并启动堆栈或用户应用程序。OTA 无法通过 OTA 引导加载来更新 App0。

App1 是堆栈应用程序。它包含 DFU 和 Bluetooth LE 堆栈。DFU 可以将更新下载到用户应用程序或堆栈。更新堆栈需要将更新放在临时位置并切换到启动器进行复制，因为堆栈无法覆盖自身。

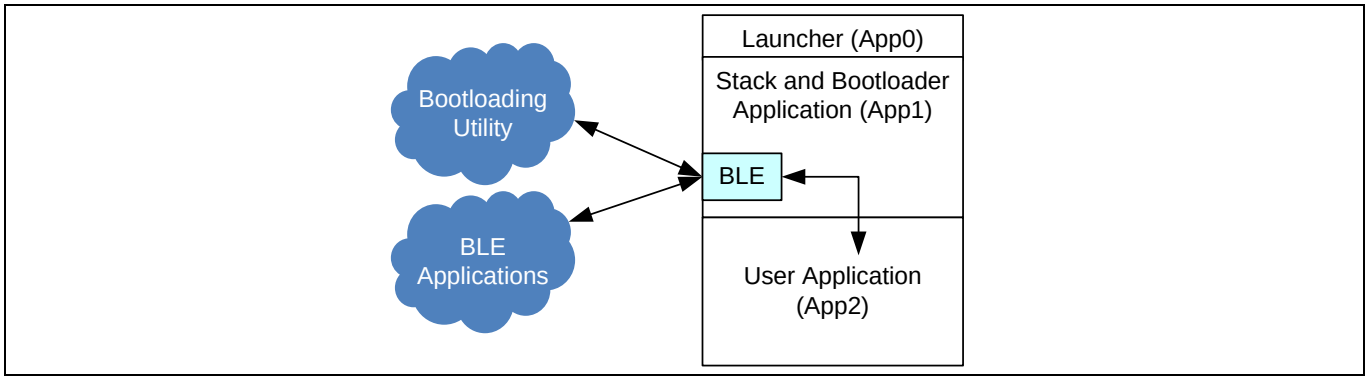


Figure 28 CE220960 数据和控制流程

App2 是用户应用程序。它有一个 PSoC Creator Bluetooth LE 组件，但它只包含没有支持堆栈代码的配置文件。从堆栈应用程序共享所需的堆栈代码和变量，大大减小了用户应用程序的大小。

构建此代码示例的步骤与其他 Bluetooth LE 代码示例的步骤类似;主要区别在于您创建了三个项目，并且 Bluetooth LE 配置不同:

1. 创建 App0 和工作区，如 **Figure 23** 所示。
2. 根据需要，将组件 (即 LED 和按钮) 从 **CE220960** App0 顶部设计原理图复制到顶部设计原理图。
3. 在 Design Wide Resources 窗口的 **Pins** 选项卡中，将组件引脚连接到相应的物理引脚。使用 **CE220960** 和套件指南作为指导。
4. 在项目构建设置中将 App0 配置为 DFU - 请参见 **Figure 10**。您必须执行此操作以启用复制应用程序和传输控制功能 - 仅选中 Core 和 App 类型框。不要选择任何通信通道盒。
5. 选择 **Build** > **Generate Application**。这会创建许多文件，包括 DFU 链接器文件，例如 *bootload_cm4.ld*。
6. 返回项目构建设置并将 CM0+和 CM4 的自定义链接脚本文件设置为相应的 DFU 链接器脚本文件 - 请参见 **Figure 12**。另外，向 CM4 用户命令添加对 *post_build_core1.bat* 文件的调用。见 **Figure 13**。
7. 更新文件，并从 **CE220960** 中的相应文件中复制 **Table 5** 中列出的新文件。根据您的应用需要添加代码。

Table 5 从 DFU SDK 默认设置中修改 CE220960 PSoC Creator App0 文件

文件	默认值的修改说明
bootload_common.ld, bootload_cm0p.ld, bootload_cm4.ld *bootload_mdk_common.h, bootload_mdk_symbols.c* *bootload_cm0p.scat, bootload_cm4.scat*	修改了三个应用程序的内存分配
bootload_user.h, bootload_user.c	修改了元数据中的应用程序数
main_cm0p.c, main_cm4.c	添加代码来执行代码示例函数
从代码示例中复制，然后添加到 Shared Files 文件夹	
post_build_core1.bat	将输出复制到 App1 和 App2 的构建位置

8. 将新项目 App1 添加到工作区。将组件和文件添加到项目中，并以与**其他 Bluetooth LE 代码示例**中的 App0 相同的方式对其进行配置;除了在 Bluetooth LE 配置对话框中，为 **Over-The-Air bootloading with code sharing** 选择 **Stack and Profile**。
9. 使用 CE220960 App1 作为指导。请注意，在 **CE220960** 中，App1 有两个构建后的批处理文件，每个内核一个。
10. 使用 CE220960 中的说明进行测试。

11. 将新项目 App2 添加到工作区。将组件和文件添加到 App2，并以**其他 Bluetooth LE 代码示例**中的 App0 相同的方式对其进行配置，但在 Bluetooth LE 配置对话框中，选择通过代码共享进行空中引导加载 (**Over-The-Air bootloading with code sharing**) 的配置文件 (**Profile Only**) 。使用 CE220960 App2 作为指导。

5.4 PSoC 6 MCU 双应用程序 Bootloader

本节介绍如何在代码示例 **CE221984** 中构建 Bluetooth LE Bootloader。此代码示例与**基本 DFU 代码示例**类似，只是有两个可下载的应用程序而不是一个。有三个应用; App0 是 DFU，App1 和 App2 是可下载的应用程序。

DFU 将控制权转移到基本模式或出厂默认 (“黄金映像”) 模式下的其中一个应用程序。在出厂默认模式下，DFU (App0) 不会覆盖已安装且有效的 App1;尝试这样做会导致错误消息。在基本模式下，任何一个应用程序都可以被覆盖。

在此代码示例中，DFU 使用 I2C 通信通道。很容易改变到不同的通信通道; 有关更多信息，请参阅**上一节**。

要构建此代码示例，请执行以下步骤。有关这些步骤的更多信息，请参考 **PSoC 6 MCU 基本 DFU** 或**如何使用 SDK**。

1. 创建 App0 和工作区，如 **Figure 29** 所示。

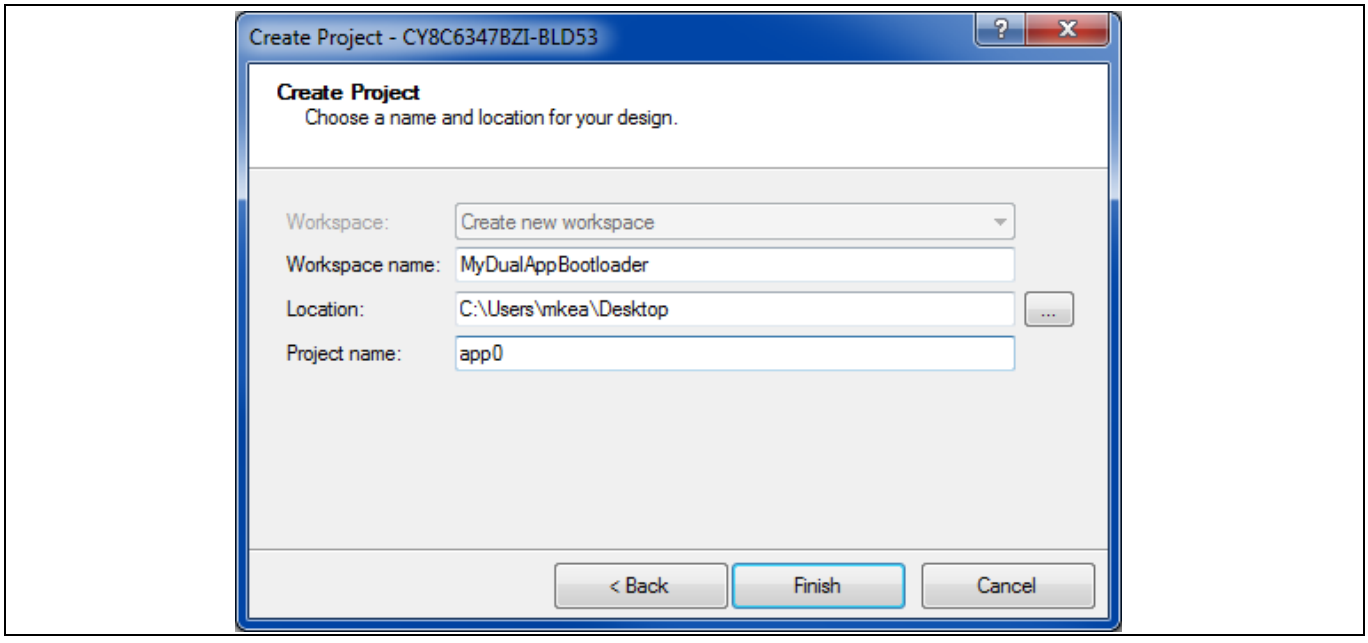


Figure 29 PSoC Creator 创建双应用 DFUApp0

2. 在项目构建设置中将 App0 配置为 DFU - 请参见 **Figure 10**。为通信通道选择 **I2C**。
3. 选择“**Build** > **Generate Application**。这将创建 DFU 链接器文件，例如 *bootload_cm4.ld*。
4. 返回项目构建设置，并将 CM0+和 CM4 的 **Custom Linker Script** 设置为相应的 DFU 链接脚本文件 - 请参见 **Figure 12**。
5. 将 I2C 组件添加到 App0 原理图中，并根据 **Figure 18** 配置组件。此时，您还可以添加 **CE221984** 使用的引脚组件; 最简单的方法是从代码示例原理图中复制并粘贴组件。
6. 在 Design-Wide Resources 窗口的 **Pins** 选项卡中，将组件引脚连接到相应的物理引脚。
7. 使用 **CE221984** 中相应文件的内容更新 **Table 6** 中列出的文件。根据您的应用需要添加代码。

Table 6 从 DFU SDK 默认设置修改 CE221984 App0 文件

文件	默认修改说明
bootload_common.ld *bootload_mdk_common.h*	为 App2 添加内存区域
bootload_user.h	设置 CY_BOOTLOAD_GOLDEN_IMAGE_IDS 宏以将 App1 确认为出厂默认图像。 要更改代码示例模式，请将 CY_BOOTLOAD_OPT_GOLDEN_IMAGE 宏更改为非零值。
bootload_user.c	从 UART 将传输功能更改为 I2C。 添加了 Cy_Bootload_WriteData()代码以执行出厂默认（“黄金映像”）检查。
main_cm0p.c *main_cm4.c*	添加代码来执行代码示例函数

8. 构建 app0 项目，并将其编程到目标工具包中。测试应用程序下载 (引导加载) 过程。您可以使用 **CE221984** 文档“操作”部分中列出的测试步骤。尝试将 CY_BOOTLOAD_OPT_GOLDEN_IMAGE 宏更改为非零值，并记下下载 App1 时 DFU 的不同行为。
   您可以使用代码示例中已有的 app1 和 app2 项目测试 DFU，或者创建自己的应用程序，如以下步骤所示:
9. 添加 app1 项目和 app2 项目。您可以将两个项目配置为可安装的应用程序。所需的详细步骤见**第 5.2.2 节第 3 步**和**第 4 步**。重要的任务是:

- 更新项目构建设置:
  - 选择 Peripheral Driver Library > DFU SDK。只选择核心框。 (**Figure 10**)
  - 链接器自定义脚本。 (**Figure 12**)
  - 构建后的批处理文件 (**Figure 13**)
- 将 *post_build_core1.bat* 文件添加到项目的 Shared Files 文件夹中。有关典型批处理文件内容，请参阅 **Appendix E**。
- 使用 **CE221984** 中相应文件的内容更新项目原理图和设计范围的资源文件。根据您的应用需要修改设计。
- 使用 CE221984 中相应文件的内容更新 **Table 7** 中列出的文件。根据您的应用需要添加代码。

Table 7 从 Bootloader SDK 默认设置修改 CE221984 PSoC Creator App1 和 App2 文件

文件	默认设置修改说明
bootload_common.ld *bootload_mdk_common.h*	为 App2 添加内存区域
bootload_cmp0.ld *bootload_cm4.ld* *bootload_cmp0.scat* *bootload_cm4.scat*	将内存区域和 App ID 指定为 app1 或 app2
bootload_user.h	设置 CY_BOOTLOAD_GOLDEN_IMAGE_IDS 宏以将 App1 标识为出厂默认图像。要更改代码示例模式，请将 CY_BOOTLOAD_OPT_GOLDEN_IMAGE 宏更改为非零值。
main_cm0p.c *main_cm4.c*	添加代码来执行代码示例函数

5.5 PSoC 6 MCU 加密 Bootloader

本节介绍如何构建具有安全功能的引导加载系统，如代码示例 **CE222802** 所示。此代码示例类似于**基本 DFU 代码示例**，其中包含安全引导加载所需的其他功能 (签名和加密) 。App0 是 DFU;可下载的 App1 可以签名，加密或两者兼而有之。

在此代码示例中，DFU 使用 UART 通信通道。很容易改变到不同的通信渠道;有关更多信息，请参阅**前面的**章节。

要构建此代码示例，请执行以下步骤。有关这些步骤的更多信息，请参考 **PSoC 6 MCU 基本 DFU** 或**如何使用 SDK**。

Note: *签名和加密需要密钥。代码示例中提供了包含默认密钥的文件，但是对于您的应用程序，您将需要创建新密钥。代码示例支持创建新密钥，但为此，您必须首先在计算机中安装 OpenSSL 和 Python。有关详细信息，请参阅代码示例文档*

1. 创建 App0 和工作区，如 **Figure 30** 所示。

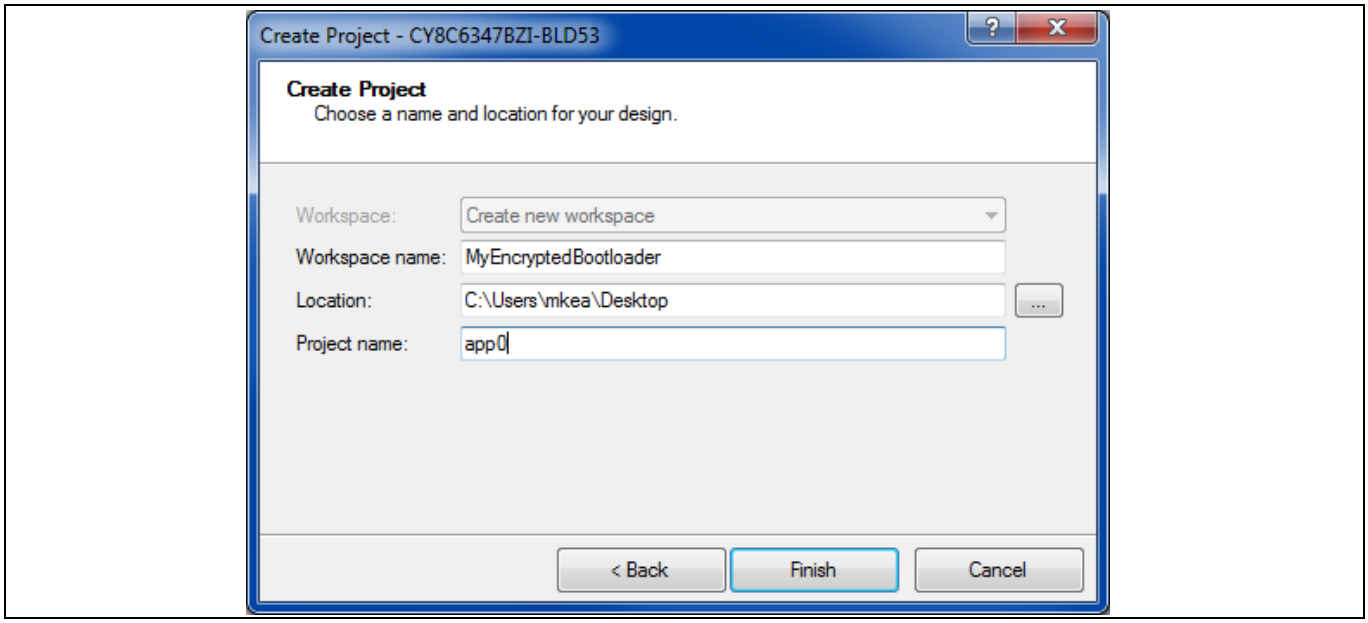


Figure 30 创建加密 DFU App0

2. 在项目构建设置中将 App0 配置为 DFU - 参见 **Figure 10**。为通信通道选择 UART。
3. 同样在 Build Settings 中，选中 Peripheral Driver Library> PDL> crypto，然后在 Variant 下选择 Extra library 以启用对 AES 解密的支持。
4. 选择 **Build** > **Generate Application。**这将创建 DFU 链接器文件，例如 *bootload_cm4.ld*。
5. 返回项目构建设置并将 CM0+和 CM4 的 **Custom Linker Script** 设置为相应的 DFU 链接器脚本文件 - 请参见 **Figure 12**。此外，还要向 **CM4 User Commands** 添加对 *post_build_core1.bat* 文件的调用。见 **Figure 13**。
6. 将 UART 组件添加到 App0 原理图中，并根据 **Figure 18** 配置组件。此时，您还可以添加 **CE222802** 使用的引脚组件;最简单的方法是从代码示例原理图中复制并粘贴组件。
7. 在 Design-Wide Resources 窗口的 Pins 选项卡中，将组件引脚连接到相应的物理引脚。使用 **CE222802** 和套件指南作为指导。
8. 更新文件，并从 **CE222802** 中的相应文件中复制 **Table 8** 中列出的新文件。根据您的应用需要添加代码。

PSoC 6 MCU DFU 代码示例

Table 8 从 DFU SDK 默认设置修改的 CE222802 App0 文件

文件	默认设置修改说明
bootload_common.ld *bootload_mdk_common.h*	不同的签名大小
bootload_user.h	设置#defines 以启用加密模块和安全验证
bootload_user.c	增加了关键变量和解密功能
main_cm0p.c *main_cm4.c*	添加代码来执行代码示例函数
新文件; 从代码示例中复制，然后添加到 Shared Files 文件夹	
cy_si_config.h *cy_si_keystorage.h, .c*	密钥存储
post_build_core1.bat	对输出文件进行签名和加密

9. 从 **CE222802** 将项目文件夹 *CE222802_Keys.cylib* 复制到工作区文件夹中。此文件夹包含 *post_build_core1.bat* 使用的密钥文件。您可以用自己的钥匙替换这些钥匙;有关说明，请参阅代码示例文档。
10. 该文件夹还包含 PSoC Creator 库项目文件 *CE222802_Keys.cylib*。您可以选择将此库项目添加到工作区。
11. 构建 app0 项目，并将其编程到目标套件中。测试应用程序下载 (引导加载) 过程。您可以使用代码示例文档的“操作”部分中列出的测试步骤。

Note: *DFU 项目的格式使得它可以通过 PSoC 6 MCU 监控闪存 (Sflash) 中的闪存引导模块进行验证，作为器件初始化的一部分。如果您重新编程设备以将 app0 替换为另一个项目，则验证将失败，设备将无法启动。如果要使用其他项目重新编程设备，必须首先禁用 app0 的验证，然后将其重新编程到设备中 - 请参阅代码示例文档中的说明。*

您可以使用代码示例中已有的 app1 项目测试 DFU，或者创建自己的应用程序，如以下步骤所示:

12. 添加 app1 项目。您可以配置为可安装的应用程序。所需的详细步骤见**第 5.2.2 节第 3 步**和**第 4 步**。重要的任务是:

- 更新项目构建设置:
  - Peripheral Driver Library > DFU SDK。只选中 Core 框。 (**Figure 10**)
  - 链接器自定义脚本。 (**Figure 12**)
  - 构建后的批处理文件 (**Figure 13**)
- 使用 **CE222802** 中相应文件的内容更新项目原理图和设计范围的资源文件。根据您的应用需要修改设计。
- 更新文件，并从 CE222802 中的相应文件中复制 **Table 9** 中列出的新文件。根据您的应用需要添加代码。

Table 9 从 DFU SDK 默认设置修改的 CE221984 App1 文件

文件	默认设置修改说明
bootload_common.ld *bootload_mdk_common.h*	指定内存区域
bootload_cmp0.ld *bootload_cm4.ld* *bootload_cmp0.scat* *bootload_cm4.scat*	指定内存区域和应用程序 ID

PSoC 6 MCU DFU 代码示例

文件	默认设置修改说明
bootload_user.h	设置#defines 以启用加密模块和安全验证
main_cm0p.c *main_cm4.c*	添加代码来执行代码示例函数
新文件; 从代码示例中复制，然后添加到 Shared Files 文件夹	
cy_si_config.h	密钥存储
post_build_core1.bat	对输出文件进行签名和加密

6 相关文档

应用笔记	
AN221774 –PSoC 6 MCU 入门	介绍 PSoC 6 MCU 器件和如何构建您的第一个 ModusToolbox 或 PSoC Creator 项目。
AN210781 – 具有蓝牙低功耗 (Bluetooth LE) 连接的 PSoC 6 MCU 入门	介绍带有 Bluetooth LE 连接器件的 PSoC 6 MCU 以及如何构建您的第一个 PSoC Creator 项目
AN215656 – PSoC 6 MCU: 双 CPU 系统设计	介绍 PSoC 6 MCU 中的双 CPU 架构，并展示如何构建简单的双 CPU 设计
AN219434 – 将 PSoC Creator 代码导入到 PSoC 6 MCU 项目的 IDE	描述如何将 PSoC Creator 生成的代码导入首选 IDE
代码示例	
CE213903 – PSoC 6 MCU 基本 DFU	描述 PSoC 6 MCU 的 UART，I2C 和 SPI DFU
CE216767 – 具有蓝牙低功耗 (Bluetooth LE) 连接 DFU 的 PSoC 6 MCU	介绍 PSoC 6 MCU 的 Bluetooth LE DFU
CE220959 – 具有外部存储器的 PSoC 6 MCU Bluetooth LE DFU	与 Bluetooth LE DFU 类似;下载的应用程序暂时保存在外部存储器中，然后复制到其最终目标
CE220960 – 具有可升级堆栈的 PSoC 6 MCU Bluetooth LE DFU	与 Bluetooth LE DFU 类似;除了应用程序之外，还可以更新 Bluetooth LE 堆栈
CE221984 – PSoC 6 MCU 双应用 I2C DFU	与基本 I2CDFU 类似;管理两个下载的应用程序而不是一个
CE222802 – PSoC 6 MCU 加密 DFU	与基本的 UARTDFU 类似;该应用程序经过数字签名和加密
PSoC Creator 组件数据表	
UART	支持基于 UART 的通信
I2C	支持基于 I2C 的通信
SPI	支持基于 SPI 的通信
Bluetooth LE	支持 Bluetooth LE 通信
设备文档	
PSoC 6 MCU: PSoC 63 BLE 数据手册	**PSoC 6 MCU:带 Bluetooth LE 架构 PSoC 63 技术参考手册**
开发套件文档	
CY8CKIT-062-BLE	PSoC 6 Bluetooth LE Pioneer Kit
CY8CKIT-062-WiFi-BT	PSoC 6 WiFi-BT Pioneer Kit
CY8CPROTO-063-BLE	PSoC 6 Bluetooth LE Prototyping Kit
CY8CPROTO-062-4343W	PSoC 6 Wi-Fi Prototyping Kit
工具文档	
ModusToolbox	ModusToolbox IDE 简化了 IoT 设计人员的开发。它为 Windows、macOS 和 Linux 提供了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。使用原理图捕获和 150 多个预先验证的，可用于生产的外设组件来创建应用程序。在下载页签中查找 Quick Start 和 User Guides。
外设驱动库 (PDL)	由 PSoC Creator 4.2 安装。在<PDL 安装文件夹> / doc 中查找 User Guide 和 API Reference。

7 Appendix A. DFU 主机工具

DFU 主机工具 (DHT) 是随 Modus Toolbox IDE 提供的独立图形工具 (PSoC Creator 也提供了一个 Bootloader 主机程序 (BHP)。其操作与 DHT 大致相同。)。此工具用于与安装了 DFU 的 PSoC 器件通信。使用此工具，您可以:

- 下载应用程序并将其安装到设备
- 验证设备上已安装的应用程序
- 从设备中删除应用程序

Note: 您无法使用 DHT 将 DFU 应用安装到设备中。相反，您必须使用其他工具(如 Modus Toolbox IDE 或 Cypress Programmer) 通过器件 SWD/JTAG 端口将 DFU 编程到闪存。安装 DFU 应用后，您可以使用 DHT 来安装可下载的应用程序。

DHT 支持通过 UART，I2C，SPI 或 USB 与赛普拉斯 MCU 器件通信，如 **Figure 31** 所示。您可以看到所有可用于连接的设备。对于 UART 或 USB，可以通过连接适当的电缆直接从计算机进行通信。对于 I2C 和 SPI，需要一个特殊的通信端口，例如 KitProg 模块。端口配置字段根据所选端口而变化。

Note: 目前，PSoC 6 MCU 仅支持基于 UART，I2C 和 SPI 的 DFU。

*Note: DHT 不支持蓝牙低功耗(Bluetooth LE)。请使用赛普拉斯的 **CySmart**™产品;请参阅“CySmart 用户指南”部分“更新外设固件”。*

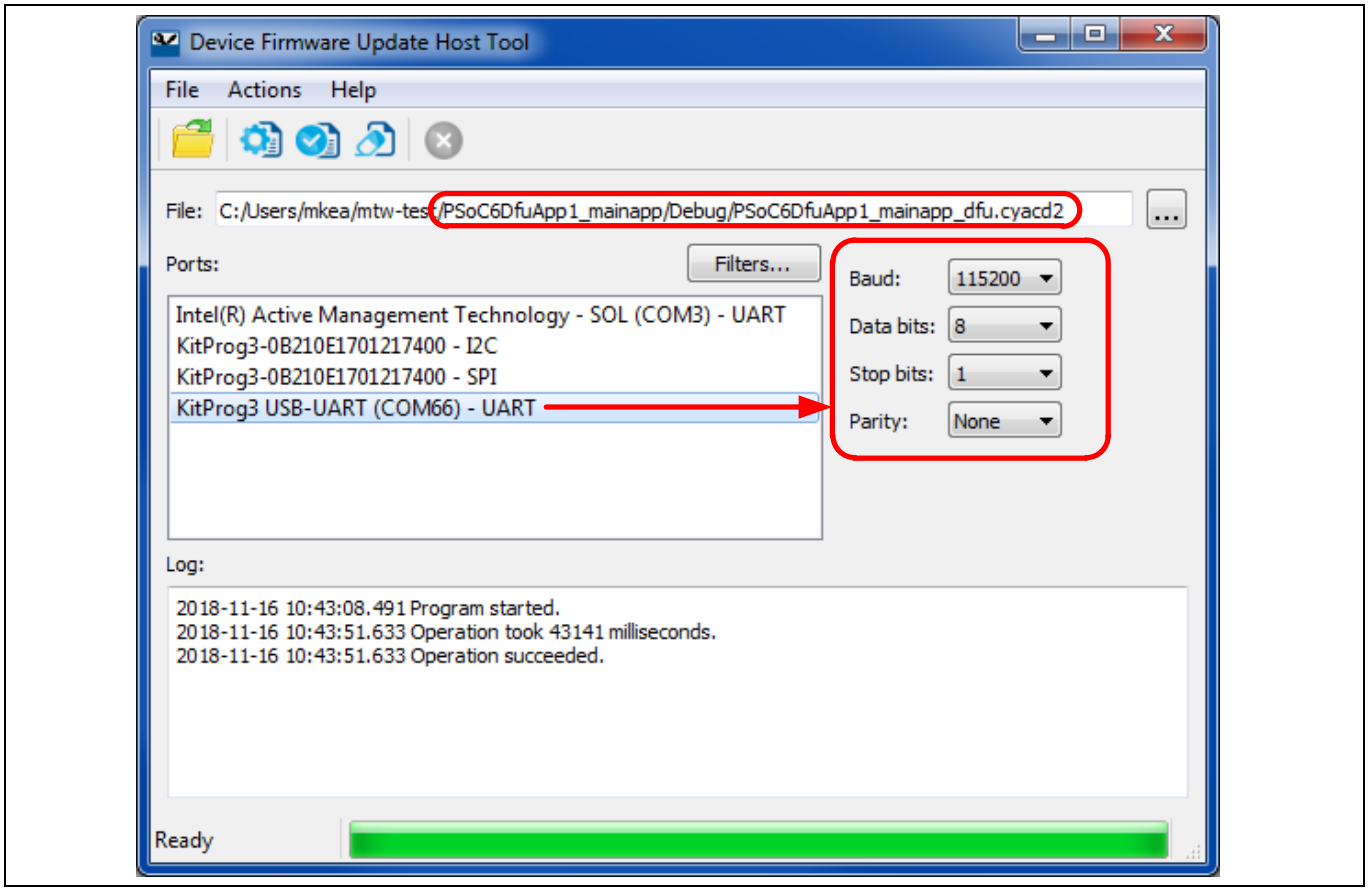


Figure 31 DFU 主机程序图形用户界面 (GUI)

8 Appendix B. 主机命令/响应协议

无论使用何种通信通道，DFU 模块都使用简单的命令响应协议与主机通信。DFU 模块从通信信道接收命令，并通过向通信信道发送一个或多个字节来响应每个命令。见 **Figure 2**。

命令和响应采用字节流的形式，以确保传输数据完整性的方式打包。包括数据包有效性检查方法，也包含 2 的补码 16 位校验和。

8.1 命令/响应数据包结构

从主机发送到 DFU 模块的通信包具有如 **Figure 32** 所示的结构:

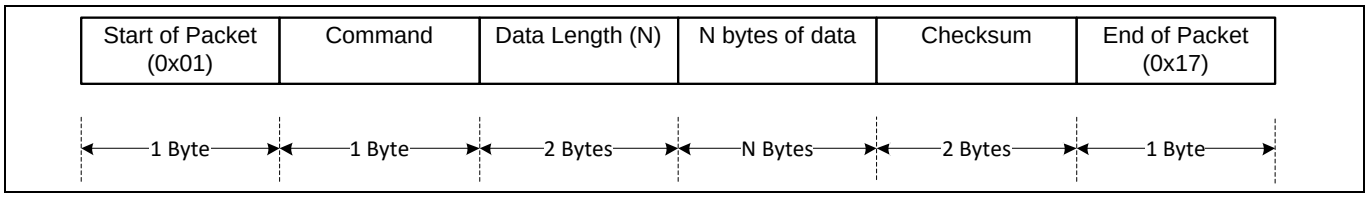


Figure 32 DFU 命令包结构

从 DFU 模块发送到主机的响应数据包具有如 **Figure 33** 所示的结构:

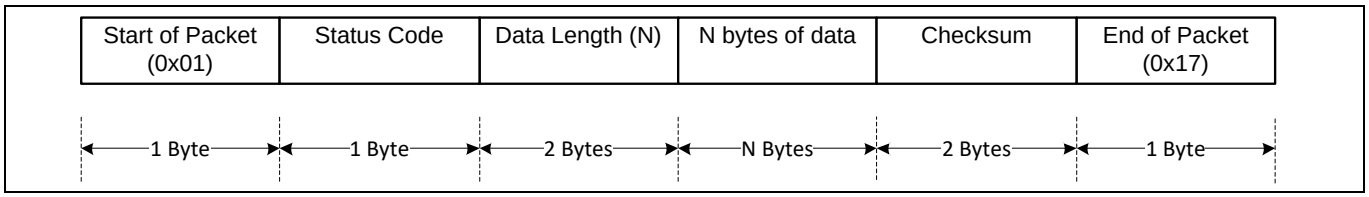


Figure 33 DFU 响应包结构

所有多字节字段都是 LSB 优先。

8.2 命令

Table 10 显示了 DFU SDK 支持的所有命令的列表。除了 Exit DFU 之外的所有命令都将被忽略，直到收到 Enter DFU 命令。

Table 10 DFU 命令列表

DFU 命令		
进入/退出	**启动引导操作**	**其它**
进入 DFU	**发送数据**	**验证应用**
同步 DFU	**发送无响应数据**	**设置应用元数据**
退出 DFU	**编程数据**	**获取元数据**
	验证数据	**设置 EIVector**
	擦除数据	

命令执行时间没有特定要求。

Table 11 显示了 DFU SDK 支持的所有状态和错误代码的列表。

Appendix B. 主机命令/响应协议

Table 11 DFU 状态和错误代码列表

状态/错误代码	值	说明
CY_DFU _SUCCESS	0x00	该命令已成功接收并执行
CY_ DFU _ERROR_VERIFY	0x02	写入失败后验证非易失性存储器 (NVM)
CY_ DFU _ ERROR_LENGTH	0x03	发送的数据量大于预期
CY_ DFU _ ERROR_DATA	0x04	分组数据的形式不正确
CY_ DFU _ ERROR_CMD	0x05	该命令无法识别
CY_ DFU _ ERROR_CHECKSUM	0x08	数据包校验和或 CRC 与预期值不匹配
CY_ DFU _ ERROR_ROW	0x0A	闪存行号无效
CY_ DFU _ ERROR_ROW_ACCESS	0x0B	无法访问闪存行号，例如由于 MPU 保护
CY_ DFU _ERROR_UNKNOWN	0x0F	出现未知错误

8.2.1 进入 DFU

开始启动加载操作。除了 Exit DFU 之外的所有其他命令都将被忽略，直到收到此命令。响应设备信息和 DFU SDK 版本。

- 输入
  - 命令字节: 0x38
  - 数据字节:
    - 4 个字节 (可选)：产品 ID。如果包含这些字节，并且它们不是 00 00 00 00，则将它们与设备产品 ID 数据进行比较。
- 输出
  - 状态/错误代码:
    - 成功
    - 错误命令
    - 错误数据，用于产品 ID 不匹配
    - 错误长度
    - 错误校验和
  - 数据字节:
    - 4 个字节: 设备 JTAG ID
    - 1 个字节: 设备修改
    - 3 个字节: DFU SDK 版本

8.2.2 同步 DFU

将 DFU 重置为已知状态，使其准备好接受新命令。缓冲的任何数据都将被丢弃。仅当 DFU 和主机彼此不同步时才需要此命令。

- 输入
  - 命令字节: 0x35
  - 数据字节: N/A
- 输出: N/A - 未确认此命令

8.2.3 退出 DFU

退出 DFU。结束 DFU 操作。

- 输入
  - 命令字节: 0x3B
  - 数据字节: N/A
- 输出: N/A – 此命令未得到确认

8.2.4 发送数据

将数据块传输到 DFU 模块。在预期编程数据或验证数据命令时缓冲该数据。如果发送了多个发送数据命令的序列，则将数据附加到前一个块。此命令用于将大量数据传输分成更小的部分，以防止某些通信协议中的信道不足。

- 输入
  - 命令字节: 0x37
  - 数据字节:
    - n 字节: 要写入和验证的数据
- 输出
  - 状态/错误代码:
    - 成功
    - 错误命令
    - 错误数据
    - 错误长度
    - 错误校验和
  - 数据字节: N/A

8.2.5 发送无响应数据

与 Send Data 命令相同，但 DFU 不生成响应。这减少了某些应用程序的 DFU 时间。

- 输入
  - 命令字节: 0x47
  - 数据字节:
    - n 字节: 要写入和验证的数据
- 输出: N/A

8.2.6 编程数据

将数据写入设备内部闪存的一行或外部非易失性存储器 (NVM) 的页面。可以遵循一系列**发送数据**或**发送无响应数据命令**

- 输入
  - 命令字节: 0x49
  - 数据字节:
    - 4 个字节: 地址。必须在正确的内存地址空间内，并进行适当的对齐。对于内部闪存，它必须与闪存行边界对齐。对于外部存储器，它必须符合外部存储器对齐要求。
    - 4 个字节: 要写入的整个数据的 CRC-32C。在编程之前和之后验证数据。

    - n 字节: 写入闪存行或外部 NVM 页面的数据。
- 输出
  - 状态/错误代码:
    - 成功
    - 错误命令
    - 错误数据
    - 错误长度
    - 错误校验和
    - 错误闪存行
    - 错误闪存行访问
  - 数据字节: N/A

8.2.7 确认数据

将数据与设备内部闪存或 SMIF 页面的一行进行比较。可以遵循一系列发送数据或发送无响应数据命令。

该命令是可选的; 它的存在取决于 *dfu_user.h* 中的用户配置宏。

- 输入
  - 命令字节: 0x4A
  - 数据字节:
    - 4 个字节: 地址。必须在正确的内存地址空间内，并进行适当的对齐。对于内部闪存，它必须与闪存行边界对齐。对于外部存储器，它必须符合外部存储器对齐要求。
    - 4 个字节: 要验证的整个数据的 CRC-32C。
    - n 字节: 与闪存行或 SMIF 页面进行比较的数据。
- 输出
  - 状态/错误代码:
    - 成功
    - 错误验证
    - 错误命令
    - 错误数据
    - 错误长度
    - 错误校验和
    - 错误闪存行
    - 错误闪存行访问
  - 数据字节: N/A
- 执行细节
  - 如果所有数据字节与从指定闪存地址开始的字节匹配，则该命令返回“成功”状态代码，否则返回“错误验证”。

8.2.8 擦除数据

删除指定的内部闪存行或 SMIF 页面的内容。

该命令是可选的; 它的存在取决于 *dfu_user.h* 中的用户配置宏。

- 输入
  - 命令字节: 0x44
  - 数据字节:
    - 4 个字节: 地址。必须在正确的内存地址空间内，并进行适当的对齐。对于内部闪存，它必须与闪存行边界对齐。对于外部存储器，它必须符合外部存储器对齐要求。
- 输出
  - 状态/错误代码:
    - 成功
    - 错误命令
    - 错误数据
    - 错误长度
    - 错误校验和
    - 错误闪存行
    - 错误闪存行访问
  - 数据字节: N/A

8.2.9 确认应用

报告闪存或外部 NVM 中应用程序的校验和是否有效。

- 输入
  - 命令字节: 0x31
  - 数据字节:
    - 1 个字节: 要验证的应用程序的应用程序编号。范围从 0 到应用程序数减 1。
- 输出
  - 状态/错误代码:
    - 成功
    - 错误命令
    - 错误数据
    - 错误长度
    - 错误校验和
    - 错误闪存行访问
  - 数据字节:
    - 1 个字节: 1/0 表示应用有效或无效

8.2.10 设置应用元数据

此命令用于设置给定应用程序的元数据。请参阅 **10**, **Appendix** D. 应用元数据。

Note: *如果用户配置 DFU SDK 以保持元数据不变，则此命令不会更新元数据。*

- 输入
  - 命令字节: 0x4C
  - 数据字节:
    - 1 个字节: 应用程序 #
    - 8 个字节: 每个附录 D 的元数据字段格式
- 输出
  - 状态/错误代码:
    - 成功
    - 错误命令
    - 错误长度
    - 错误数据
    - 错误校验和
    - 错误闪存行访问
  - 数据字节: N/A

8.2.11 获取元数据

报告选定的元数据字节

该命令是可选的; 它的存在取决于 *dfu_user.h* 中的用户配置宏。

- 输入
  - 命令字节: 0x3C
  - 数据字节:
    - 2 个字节: 来自行内的偏移; 0 - 511
    - 2 个字节: 在行内偏移; 0 - 511 (含)
- 输出
  - 状态/错误代码:
    - 成功
    - 错误命令
    - 错误长度
    - 错误数据
    - 错误校验和
    - 错误闪存行访问
  - 数据字节:
    - N 个字节 - 每个字节来自和去往偏移字节 (包括)

8.2.12 设置 EIVector

设置加密初始化向量 (EIV) 。这使 DFU 模块能够在将数据写入闪存之前解密数据。

该命令是可选的; 它的存在取决于 *dfu_user.h* 中的用户配置宏。

- 输入
  - 命令字节: 0x4D
  - 数据字节:
    - n 字节: 向量; 0,8 或 16 个字节，小端原始数据
- 输出
  - 状态/错误代码:
    - 成功
    - 错误命令
    - 错误长度
    - 错误数据
    - 错误校验和
  - 数据字节: N/A

9 Appendix C. .cyacd2 文件格式

.cyacd2 文件包含可下载的应用程序数据。它由 CyMCUElfTool 创建，并由主机程序 (如 Cypress 的 **DFU Host Program** 和 **CySmart**) 用于将应用程序发送到目标 DFU，如 **Figure 5** 所示。文件数据采用 ASCII 十六进制数字的形式，类似于 Intel 十六进制格式。每个数据字节由两个字符表示。例如，字节 0x1E 由字符 0x31 (ASCII “1”) 表示，后跟 0x45 (ASCII “E”)。

所有多字节字段都是小端。

该文件由一系列行或行组成。每行以 ASCII CR，LF 字符结尾。行是以下类型之一:

- 加密初始向量: 加密初始向量行的格式为@EIV: <bytes>。<bytes>中的数据由主机程序在 **SetEIV 命令**中用于 DFU 模块。
- 应用程序验证信息: 应用程序验证信息行的格式如下:

  ```
  @APPINFO:[__cy_app_verify_start],[__cy_app_verify_length].
  ```

  主机程序在设置应用元数据命令中使用 start 和 length 数据到 DFU。
- 标题: 标题行的结构如 **Table 12** 所示:

Table 12 .cyacd2 标题行结构

文件版本	芯片 ID	芯片版本	校验和类型	App ID	产品 ID
1 字节	4 字节	1 字节	1 字节	1 字节	4 字节

  - 文件版本: 从 1 开始编号。
  - 芯片 ID, 芯片版本,产品 ID: 用于防止应用程序被下载到错误的设备。
  - 校验和类型: 用于验证 DFU 包的方法 (请参阅 **8**, **Appendix B**. 主机命令/响应协议)。0 =校验和，1 = CRC。
  - 应用程序 ID: 请参见 **Figure 6**。这还可以控制为此应用程序更新应用程序元数据的哪一部分。
- 数据: 数据行的结构如 **Table 13** 所示。

N 的值等于要通过一系列 “发送数据” 或 “发送无响应数据” 命令以及 “编程数据” 或 “验证数据” 命令发送的数据总量。

N 的值通常但不一定等于 NVM 行的长度。例如，如果下载到 RAM 中，则 N 可以是任意值。

Table 13 .cyacd2 数据行结构

数据头	地址	数据
1 字符: “:”	4 字节	N 字节

10 Appendix D. 应用元数据

DFU SDK 使用指定的 NVM 区域 (或某些情况下的 RAM) 来存储有关应用程序的信息 - 请参见 **Figure 6**。元数据信息通常用于以下目的:

- 验证应用
- 将控制权从一个应用程序转移到另一个
- 将应用程序映像从临时位置复制到指定位置

如 **Figure 6** 所示，元数据通常占用一个闪存行或 NVM 页面。 (在具有少量闪存的设备中，可以使用多行或多页。) **Figure 6** 还显示元数据位于任何应用程序之外。

Table 14 包含用于定义 DFU 元数据的位置、大小和用法的符号。符号在 SDK 链接脚本文件和 C 源文件中定义。

Note: *显示的所有示例均适用于 GCC 编译器和链接器。MDK 和 IAR 编译器和链接器的源和链接脚本文件中存在类似的语句。*

Table 14 与元数据相关的符号

符号	定义位置	目的
`flash_boot_meta`	*dfu_cm0p.ld, dfu_cm4.ld*	定义包含元数据的物理内存区域
示例用法	`flash_boot_meta (rw) : ORIGIN = 0x100FFA00, LENGTH = 0x400` 定义 1 MB PSoC 6 MCU 用户闪存的最后 1 KB 区域 (从 0x1000 0000 开始)。	
`__cy_boot_metadata_addr` `__cy_boot_metadata_length`	*dfu_cm0p.ld, dfu_cm4.ld*	这些符号定义了与编译器无关的内存地址范围，用于存储元数据。这些符号用于 DFU SDK 代码，可用于用户代码。

示例用法	`/* DFU SDK metadata limits */`
	`/* Note that __cy_memory_0_row_size equals the row length in bytes of PSoC 6 user flash. */`
	`__cy_boot_metadata_addr = ORIGIN(flash_boot_meta);`
	`__cy_boot_metadata_length = __cy_memory_0_row_size;`	

`.cy_boot_metadata`	*dfu_cm0p.ld, dfu_cm4.ld*	DFU 元数据存储在此部分中。在构建时，CyMCUElfTool 计算此部分的校验和，并将其放在该部分的最后四个字节中。如果不需要校验和，请将该部分重命名为任何其他名称。

示例用法	`.cy_boot_metadata :`
	`{`
	`  KEEP(*(.cy_boot_metadata))`
	`} > flash_boot_meta`	

`CY_DFU_MAX_APPS`	*dfu_user.h*	允许用户控制 DFU 元数据中支持的最大应用程序数
示例用法	`/* The smallest metadata size is CY_DFU_MAX_APPS * 8 bytes per app + an optional 4 bytes for metadata checksum */` `#define  CY_DFU_MAX_APPS  (2u)`	

Appendix D. 应用元数据

符号	定义位置	目的
CY_DFU_METADATA_WRITABLE	*dfu_user.h*	DFU 不一定要写元数据 - 它可以由应用程序或其他一些用户代码完成。或者可以使用应用程序内的编译时常量来设置元数据。应用程序可以包含小于 NVM 行的元数据。 在所有这些情况下，将此宏设置为 0 以防止 DFU 写入元数据。 但请注意，DFU 要求存在元数据区域并正确初始化。这可以由 DFU 本身或应用程序完成。
示例用法	`/* A non-zero value allows writing metadata with the SetAppMetadata command. */` `#define CY_DFU_METADATA_WRITABLE (1)`	

10.1 元数据结构

应用程序元数据每个应用程序有 8 个字节的数据，后跟校验和的 4 个字节，如 **Table 15** 所示:

Table 15 元数据结构

应用程序 0	应用程序 1	...	应用程序 N – 1	校验和
8 bytes	8 bytes		8 bytes	4 bytes

Application Start Address	Application Length (bytes)
4 bytes	4 bytes

您可以在 *dfu_user.h* 文件中设置应用程序数 N:

```
#define CY_DFU_MAX_APPS (N)
```

N 的默认值为 2。

每个应用程序起始地址必须与闪存行或 NVM 页面边界对齐。应用程序长度必须是闪存行或 NVM 页面长度的倍数。

校验和的计算方法与 DFU 命令“编程数据”和“验证数据”中使用的算法相同。默认算法是 CRC-32C。

11 Appendix E. 构建后批处理文件列表

以下是来自 **CE213903** 的 App1 的构建后的 bash 和批处理文件的列表。在其他代码示例中这些文件即使不相同也是相似的。相同的 bash 或 batch 文件可以与多个可下载的应用程序一起使用，例如 PSoC 6 MCU 双应用程序 **Bootloader** 中的 app2。

11.1 ModusToolbox 构建后 Bash 文件

```
#!/bin/bash

############################################################################################################################
# This script is designed to post process a PSoC 6 application. It performs sign and merge.
#
# usage:
#   dfu_postbuild.bash <MCUELFTOOL_LOC> <CM0P_LOC> <CM4_LOC> <MCU_CORE>
#
############################################################################################################################

MCUELFTOOL_LOC=$1
CM0P_LOC=$2
CM4_LOC=$3
MCU_CORE=$4

echo Script: cymcuelftool_postbuild
echo 1: MCUELFTOOL_LOC : $MCUELFTOOL_LOC
echo 2: CM0P_LOC       : $CM0P_LOC
echo 3: CM4_LOC        : $CM4_LOC
echo 4: MCU_CORE       : $MCU_CORE
echo

filenameNoExt_cm0p="${CM0P_LOC%.*}"
filenameNoExt_cm4="${CM4_LOC%.*}"

if [ "$MCU_CORE" == "ARM_CM4" ]; then
$MCUELFTOOL_LOC --sign $CM4_LOC --output $filenameNoExt_cm4"_signed.elf"
$MCUELFTOOL_LOC --merge $filenameNoExt_cm4"_signed.elf" $filenameNoExt_cm0p"_signed.elf" --output
$filenameNoExt_cm4"_final.elf"
```

```
#DFU
$MCUELFTOOL_LOC --sign $filenameNoExt_cm4"_final.elf" CRC --output $filenameNoExt_cm4"_dfu.elf"
$MCUELFTOOL_LOC -P $filenameNoExt_cm4"_dfu.elf" --output $filenameNoExt_cm4"_dfu.cyacd2"

else
$MCUELFTOOL_LOC --sign $CM0P_LOC --output $filenameNoExt_cm0p"_signed.elf"
  if [ -e $filenameNoExt_cm4"_signed.elf" ]; then
  $MCUELFTOOL_LOC --merge $filenameNoExt_cm4"_signed.elf" $filenameNoExt_cm0p"_signed.elf" --output
$filenameNoExt_cm4"_final.elf"
  fi
fi
```

11.2 PSoC Creator 构建后 Batch 文件

```
@rem Usage:
@rem Call post_build_core1.bat <tool> <output_dir> <project_short_name>
@rem E.g. in PSoC Creator 4.2:
@rem      post_build_core1.bat creator ${OutputDir} ${ProjectShortName}

@echo ----------------------------------------
@echo   Post-build commands for Cortex-M4 core
@echo ----------------------------------------

@rem Set proper path to your PDL 3.x and above installation
@set PDL_PATH="C:\Program Files (x86)\Cypress\PDL\3.0.1"

@set CY_MCU_ELF_TOOL=%PDL_PATH%"\tools\win\elf\cymcuelftool.exe"

@set IDE=%1

@if "%IDE%" == "creator" (
    @set OUTPUT_DIR=%2
    @set PRJ_NAME=%3
    @set ELF_EXT=.elf
)

@if "%IDE%" == "uvision" (
```

```
    @set OUTPUT_DIR=%2
    @set PRJ_NAME=%3
    @set ELF_EXT=.axf
)

@if "%IDE%" == "iar" (
    @set OUTPUT_DIR=%2
    @set PRJ_NAME=%3
    @set ELF_EXT=.out
)

@if "%IDE%" == "eclipse" (
    @set OUTPUT_DIR=%2
    @set PRJ_NAME=%3
    @set ELF_EXT=
)

%CY_MCU_ELF_TOOL% -S %OUTPUT_DIR%\%PRJ_NAME%%ELF_EXT% CRC
%CY_MCU_ELF_TOOL% -P %OUTPUT_DIR%\%PRJ_NAME%%ELF_EXT% --output %OUTPUT_DIR%\%PRJ_NAME%.cyacd2
```

文档修订记录

版本	提交日期	变更说明
**	2019-02-28	翻译自: 002-13924 Rev. *E
*A	2020-03-10	翻译自: 002-13924 Rev. *F
*B	2021-06-03	更新至英飞凌模板 翻译自: 002-13924 Rev. *G

Trademarks
All referenced product or service names and trademarks are the property of their respective owners.

Edition 2021-06-03
Published by
Infineon Technologies AG
81726 Munich, Germany

© 2021 Infineon Technologies AG. All Rights Reserved.

Do you have a question about this document?
Go to www.cypress.com/support

Document reference
002-26595 Rev. *B

重要提示

本文档所提供的任何信息**绝不应当**被视为针对任何条件或者品质而做出的保证（质量保证）。英飞凌对于本文档中所提及的任何事例、提示或者任何特定数值及/或任何关于产品应用方面的信息均在此明确声明其不承担任何保证或者责任，包括但不限于其不侵犯任何第三方知识产权的保证均在此排除。

此外，本文档所提供的任何信息均取决于客户履行本文档所载明的义务和客户遵守适用于客户产品以及与客户对于英飞凌产品的应用所相关的任何法律要求、规范和标准。
本文档所含的数据仅供经过专业技术培训的人员使用。客户自身的技术部门有义务对于产品是否适宜于其预期的应用和针对该等应用而言本文档中所提供的信息是否充分自行予以评估。

如需产品、技术、交付条款和条件以及价格等进一步信息，请向离您最近的英飞凌科技办公室接洽(**www.infineon.com**)。

警告事项

由于技术所需产品可能含有危险物质。如需了解该等物质的类型，请向离您最近的英飞凌科技办公室接洽。

除非由经英飞凌科技授权代表签署的书面文件中做出另行明确批准的情况外，英飞凌科技的产品不应当被用于任何一项一旦产品失效或者产品使用的后果可被合理地预料到可能导致人身伤害的任何应用领域。