AN73503

PSoC® USB HID Bootloader

作者： Robert Murphy, Keith Mikoleit

相关器件系列：所有支持 USB 的 PSoC 3、PSoC 4 L 系列和 PSoC 5LP 器件

相关代码示例和其他文档：请点击此处查看相关文档的完整列表。

更多代码示例？我们明白。

如需寻找包含上百 PSoC 代码示例并有不断更新的网上资源，请浏览我们的代码示例网页。您还可以在此处观看 PSoC 4 视频库。

AN73503 介绍了如何通过使用 USB 人机界面器件（HID）系列为 PSoC 器件实现一个 USB Bootloader。同时它还展示了如何创建一个基于 Windows 的 USB Bootloader 主机程序。

目录

1 简介 1
1.1 术语和定义 2
1.2 使用 Bootloader 2
1.3 Bootloader 功能流程 3
1.4 USB Bootloader 注意事项 3
1.5 代码示例 3
2 Bootloader 工程 4
2.1 创建工程 4
2.2 配置 Bootloader 组件 5
2.3 配置 Bootloader 组件 5
2.4 配置输入引脚 6
2.5 放置引脚并配置时钟 6
2.6 Bootloader 固件 8
3 Bootloadable（应用）工程 8
3.1 创建工程 8
3.2 配置 Bootloadable 组件 9
3.3 配置工程的其余部分 9
4 PSoC Creator Bootloader 主机 10
5 创建一个 Bootloader 主机 11
5.1 所需资源 11
5.2 创建一个 Bootloader 主机应用程序 12
6 总结 15
7 相关资源 16
A 附录 A — USBFS HID 配置 17
A.1 创建器件描述符 17
A.2 创建 HID 描述符 21
B 附录 B — Bootloader 和器件复位 26
B.1 为什么需要执行器件复位？ 26
B.2 对 PSoC 3 和 PSoC 5LP 器件 I/O 引脚的影响 26
B.3 对其他功能的影响 27
B.4 示例：风扇控制 27
C 附录 C — 主机内核 API 29
文档修订记录 30
全球销售和设计支持 31
产品 31
PSoC® 解决方案 31
赛普拉斯开发者社区 31
技术支持 31

1 简介

Bootloader 是 MCU 系统设计中常用的部分。通过 Bootloader，可以现场对产品的固件进行更新。在工厂里，将固件初始编程到某个产品内时，通常是通过 MCU 的联合测试行动组织（JTAG）或串行线调试器（SWD）接口实现的。但一般不能在现场访问这些接口。

这样就需要引入引导加载（Bootload）。通过引导加载程序，可以使用一个标准的通信接口（如 USB 或 I2C）对系统固件进行升级。Bootloader 通过与主机通信获取新的应用代码或数据，然后将这些代码或数据写入到器件的闪存存储器内。

通过本应用笔记，您可以了解：

- 将 USB bootloader 添加到 PSoC 3、PSoC 4 L 系列或 PSoC 5LP 器件内
- 为引导加载准备一个应用工程
- 使用 PSoC Creator 所提供的 Bootloader 主机程序
- 添加您自己的基于 Windows 的 Bootloader 主机程序

本应用笔记假定您已经熟悉了 PSoC 器件和 PSoC Creator 集成开发环境。如果您尚未了解这些产品，可通过 AN54181、AN79953 和 AN77759 等 PSoC 入门应用笔记获取相关信息。
如果您对 PSoC Creator 还不太熟悉，请参考 PSoC Creator 主页。

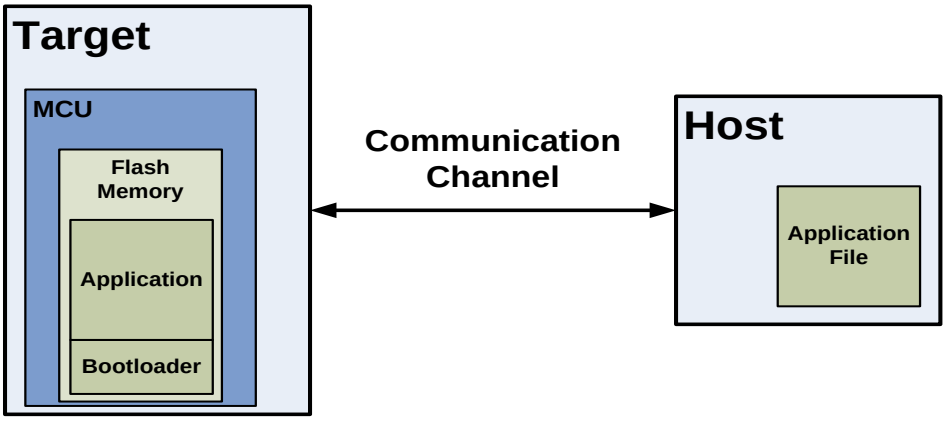
本应用笔记同时假设您已经熟悉了有关 Bootloader 的概念。如果您对这些概念还不太熟悉，请参考应用笔记 AN73854 — PSoC 介绍中有关 Bootloader 的部分。

最后，本应用笔记假设您已经熟悉了 USB。如果您刚接触 USB 或 USB HID 类别，请参考 AN57294 — USB 101：通用串行总线 2.0 简介或 AN57473 — PSoC 中的 USB HID 基本知识。

1.1 术语和定义

通过图 1 可以知道，产品中的嵌入式固件可以通过通信端口实现两种不同的操作 — 普通操作和更新应用操作。嵌入式固件中用于更新应用的部分被称为 **Bootloader**。

图 1. Bootloader 系统框图



一般情况下，提供用于更新闪存中数据的系统被称为**主机**，被更新的系统被称为**目标机**。主机可以是一个外部 PC，也可以是与目标机位于相同 PCB 上的另一个 MCU。

从主机到目标闪存的数据传输过程被称为**引导加载过程**，或**引导加载操作**，或简称为**引导加载**。放置在闪存中的数据被称为**应用程序**或 **Bootloadable**。

引导加载的另一个常见术语是**系统内编程（ISP）**。赛普拉斯有一款名称相似的产品，即系统内串行编程器（ISSP），它的操作被称为主机源串行编程（HSSP）。更多有关信息，请参考应用笔记 AN73054 或 AN84858 — 使用外部微控制器（HSSP）对 PSoC 进行编程。

1.2 使用 Bootloader

Bootloader 和应用程序通常共用一个 Bootloader 通信端口。要想使用 Bootloader，首先要对产品进行操作以执行 Bootloader，而不是执行应用程序。

一旦运行了 Bootloader，主机可以使用通信通道发送“start bootload”（开始引导加载）命令。如果目标机发送了一个“OK”响应，则引导加载便可开始。

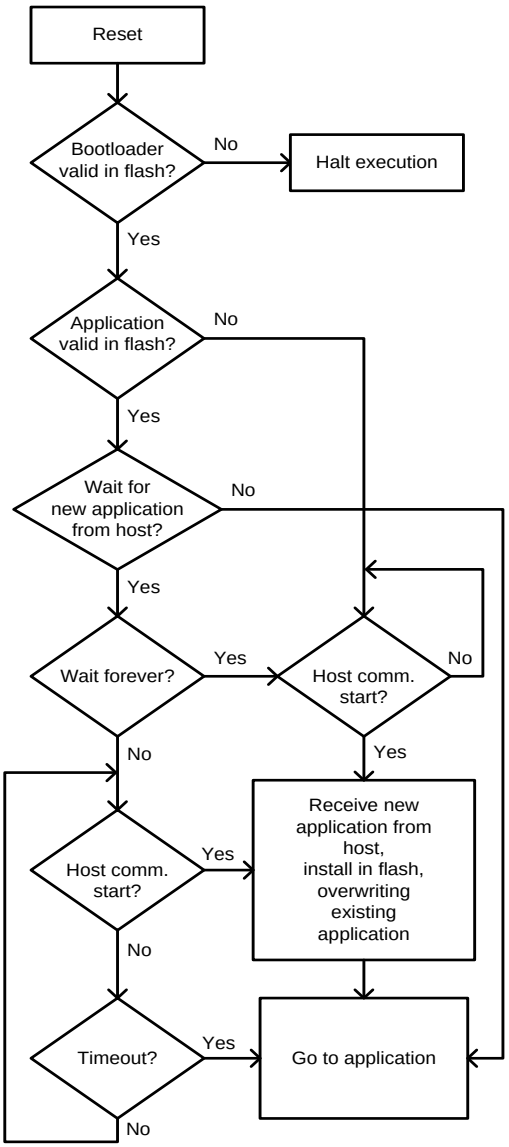
在引导加载过程中，主机将读取包含新应用程序信息的文件，并将读取的内容解析成闪存写命令，然后将这些命令发送到 Bootloader 中。发送完整个文件后，Bootloader 将控制权交给新的应用程序。

1.3 Bootloader 功能流程

一个 Bootloader 通常从复位开始。然后它可以执行以下操作，如图 2 所示：

- 在运行应用程序前，先检查它的有效性
- 管理时序，以启动主机通信
- 实现引导加载 /闪存更新操作
- 最后，将控制权交给应用程序

图 2. 引导加载过程流程图



1.4 USB Bootloader 注意事项

将一个 USB 端口用作为 Bootloader 中的通信接口使用时，需要考虑几个问题。USB HID 类别的优点是不需要驱动程序，并且该器件能够在任何主机操作系统下工作。

请注意，将控制权交给应用程序前，Bootloader 会等待一段时间，USB 器件会利用这段时间进行枚举。如果 USB 枚举时间过长，那么在目标机重启后主机可能错失启动引导加载操作的机会。

一旦应用程序获得控制权，有两种方式可使控制权返回给 Bootloader。

Bootloadable API

PSoC Creator Bootloadable 组件有一个用于启动 Bootloader 的 API 函数，即 `Bootloadable_Load()`。通过使用该函数，应用程序可以将控制权返回交给 Bootloader。对于一个产品的用户来说，这是启动固件更新的好方法。

该方法的缺点是您必须依赖应用程序代码才能执行应用程序的升级。如果应用程序存在漏洞，不能将控制权成功返回给 Bootloader 怎么办？为了确保能够升级和修复固件，在 Bootloader 工程中添加启动 Bootloader 的可靠处理方法不失为一个好的选择。

初始化时启动带无限期等待的 Bootloader

为了解决应用程序受损或 USB 枚举时间过长等问题，我们可以添加一种方法来保持停留在 Bootloader 中，直到接收到主机命令为止。在调用 Bootloader_Start()和运行其正常子程序前，您可以通过自定义 Bootloader 工程来检查某些用户输入。

更多有关信息，请参考 Bootloader 固件部分。

1.5 代码示例

下面内容介绍的是创建 PSoC Creator Bootloader 和 Bootloadable 工程的各个步骤。您可以在代码示例 CE95391 中获取完整的工程。

2 Bootloader 工程

PSoC Bootloader 系统至少包含两个 PSoC Creator 工程 — 一个 Bootloader 工程和至少一个 Bootloadable（应用程序）工程。本章节将介绍创建 Bootloader 工程的方法。下一章节说明了如何创建一个 Bootloadable 工程。

Bootloader 工程设计包含了 PSoC Creator Bootloader 和各个 USBFS 组件，如图 3 所示。USBFS 组件与电脑主机进行通信，以获取各条命令和一个新的应用镜像。Bootloader 组件会编程闪存，执行主机命令/响应协议，并启动 Bootloadable 应用。

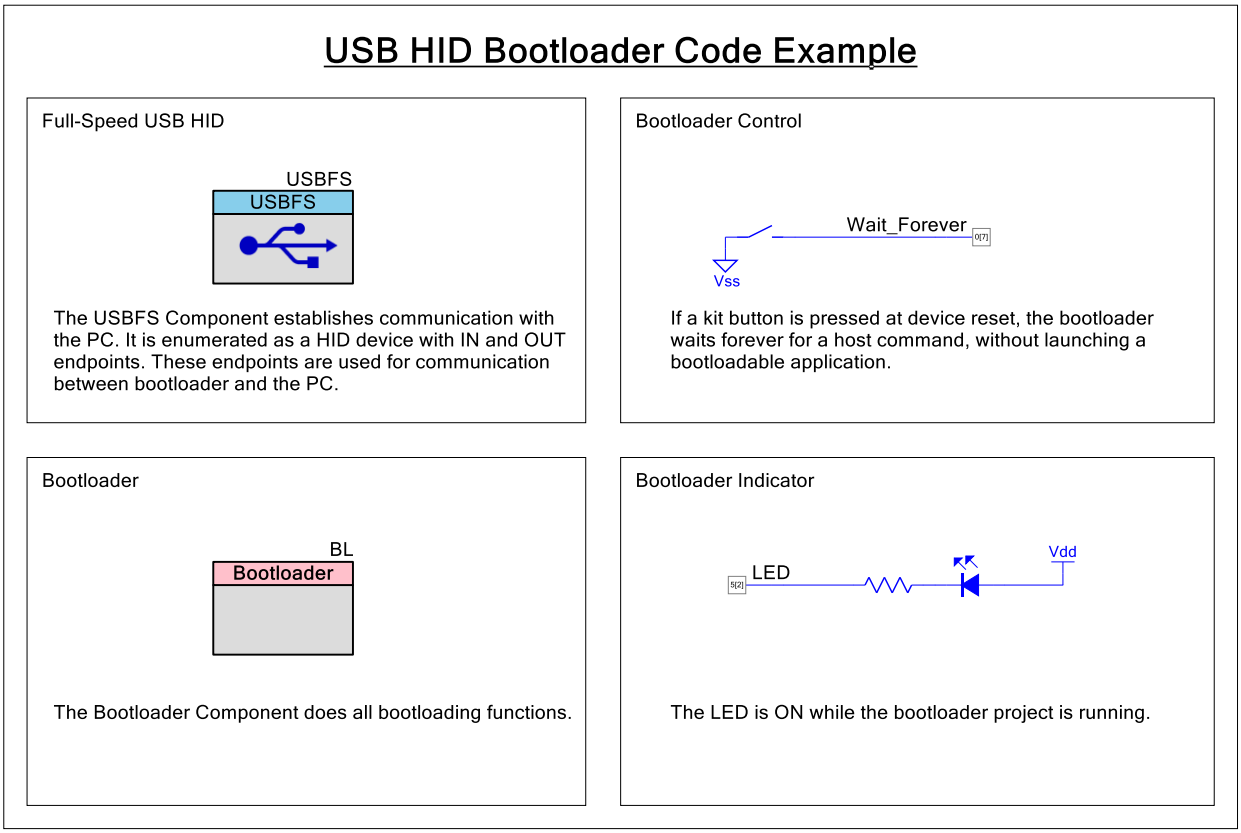
2.1 创建工程

首先，创建一个新的 PSoC Creator 工程。在 Create Project 对话框中选择您的目标硬件或器件。

为工作区命名，如“PSoCx_USB_Bootloader”。为工程命名，如“USB _Bootloader”。

现在，将各 PSoC Creator 组件（USBFS、Bootloader 和其他组件）添加到原理图中。您可以根据图 3 中显示的内容重命名各个组件。

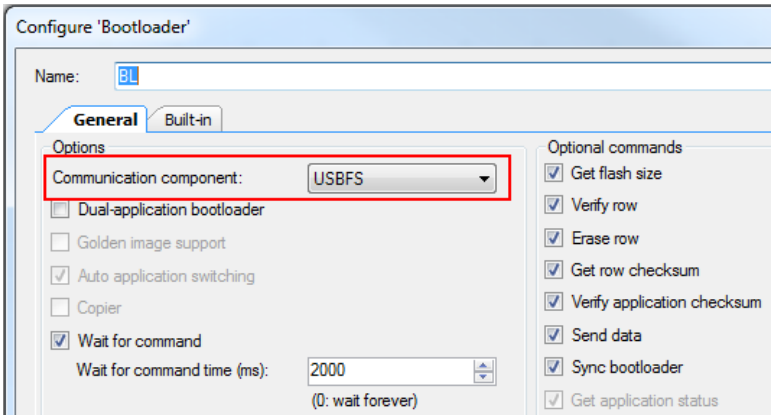
图 3. 代码示例 CE95391 中的 Bootloader 工程原理图



2.2 配置 Bootloader 组件

只需要对 Bootloader 组件执行唯一一项更改，即将通信组件设置为 USBFS，如图 4 所示。这样 PSoC Creator 会生成 Bootloader 所要求的函数，用于连接 USB。

图 4. Bootloader 组件定制向导



您还需要查看 **Wait for command**（等待命令）设置。由于需要考虑 USB 器件的枚举时间，因此对于 USB Bootloader 来说，这些设置极其重要。

- **Wait for command**：器件复位时，Bootloader 可以等待来自主机的命令，或立即切换到应用代码。如果已经使能该选项，Bootloader 将等待来自主机的命令，直到经过 **Wait for command time**（等待命令的时间）参数指定的超时时间为止。如果在超时间隔内 Bootloader 没有收到由主机发送的命令，它会切换到应用代码。
- **Wait for command time**（单位为 ms）：如果已经使能了上述选项，该参数指的是 Bootloader 切换到应用代码前的等待时间。如果该值为零，则表示等待时间无限长。默认值为两秒超时。

如果器件闪存中没有安装有效应用，则无论这些设置如何，Bootloader 都会一直等待主机命令。

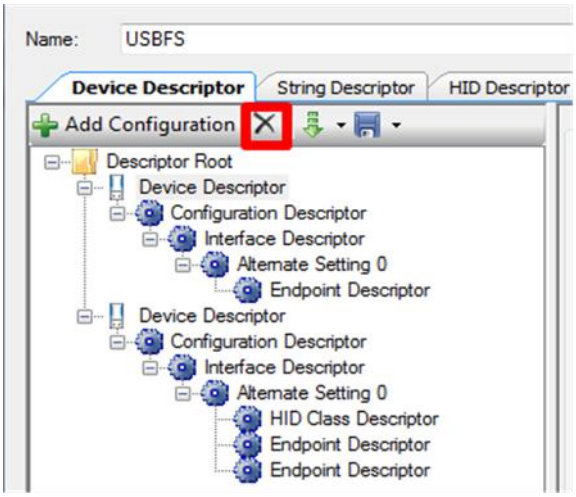
有关 Bootloader 组件设置的完整信息，请参考 Bootloader 组件数据手册中的内容。

2.3 配置 Bootloader 组件

接下来，要配置 USBFS 组件，从而支持 Bootloader 通信。谨记，我们采用的是 USB HID 类别，从而主机 PC 不再需要外部提供的驱动程序文件。

要想开始配置 USBFS 组件，请双击它。然后，在配置对话框中，删除默认的器件根描述符（Device Descriptor root）。点击 **Device Descriptor** 选项卡，然后点击 **‘X’** 按键，如图 5 所示。

图 5. 删除默认的 USB 配置

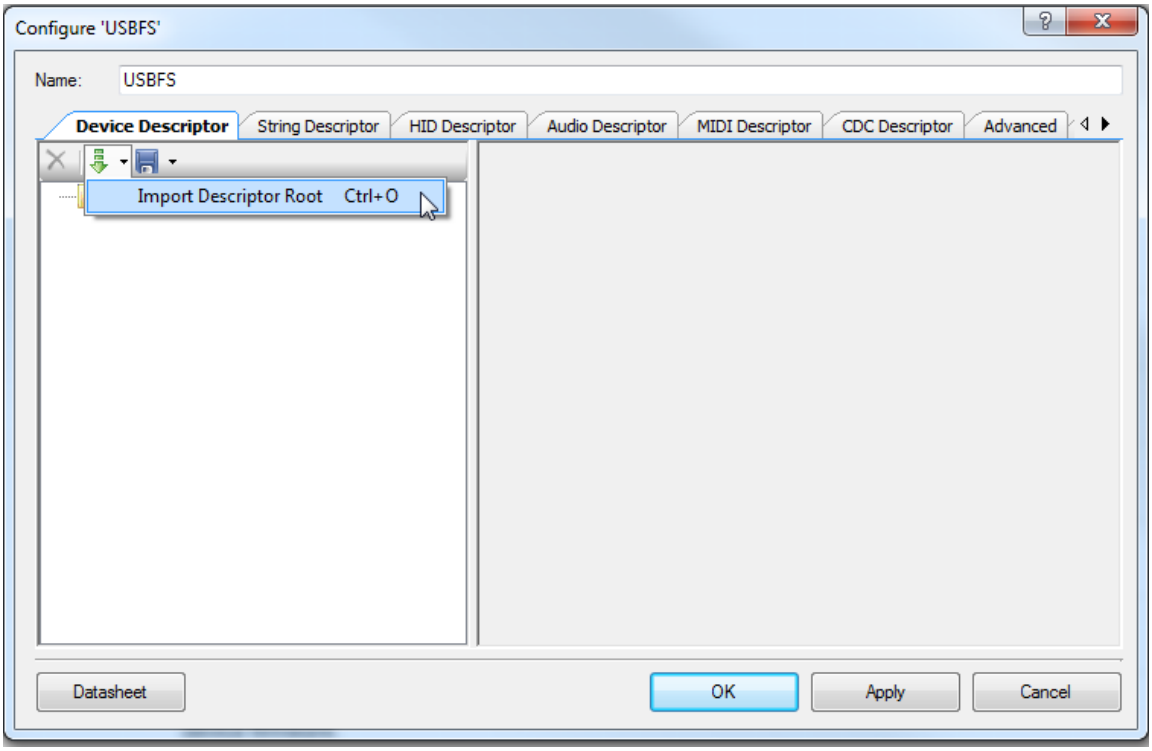


然后，使用“Insert Configuration”选项来导入 Bootloader 的 USB 配置，如图 6 所示。Bootloader 的完整 USB 配置被保存为一个*.xml* 文件，该文件保存在 PSoC Creator 中。

<PSoC Creator InstallDir> \ psoc \ content \ cycomponentlibrary \ CyComponentLibrary.cylib \ USBFS_v_2_80 \ Custom \ template \ Bootloader.root.xml

导入该配置后，PSoC Creator 会自动加载所需的 USB 描述符。请参考附录 A — USBFS HID 配置，了解有关 USB 配置的详细信息。

图 6. USB 插入配置



2.4 配置输入引脚

如前所述，这是一个提供用于启动 Bootloader 的后门选项的好方法。其中一种方法就是在启动时检查引脚的状态。如果该终端应用已经拥有了一个用户接口（如某个开关），那么该方法比较有效。

通过使用数字输入引脚组件，可以获得某个引脚的状态。在代码示例 CE95391 中，该引脚被连接到套件的一个按键上。按下该按键时，它被短路接地，因此要将引脚组件配置为电阻上拉模式。这样，按下按键时，引脚输入状态会变为 0；释放按键时其状态则变为 1。

2.5 放置引脚并配置时钟

完成配置所有组件后，可以设置时钟源并放置各个引脚。

在工作区浏览器窗口中，找出并双击 *USB_Bootloader.cydwr*。这时将出现“Pin Selection”选项卡。将 Wait_Forever 和其他引脚组件分配到在您套件中所使用的器件物理引脚上。请参考代码示例 CE95391，深入了解该过程。

注意： PSoC Creator 会自动分配 USB D+和 D-数据线。

注意： 您可以将 Bootloader 工程和应用程序中的引脚组件分配到同一个物理引脚上。甚至，您可以使用同一个引脚名称，因为 Bootloader 和应用程序是相互独立的工程。

然后，点击 **Clocks** 选项卡，并双击其中一个时钟，从而打开时钟配置向导。图 7 显示的是 PSoC 4 L 系列的 USB 时钟配置，图 8 则显示了 PSoC 3 和 PSoC 5LP 的 USB 时钟配置。

图 7. PSoC 4 L 系列的 USB 时钟配置

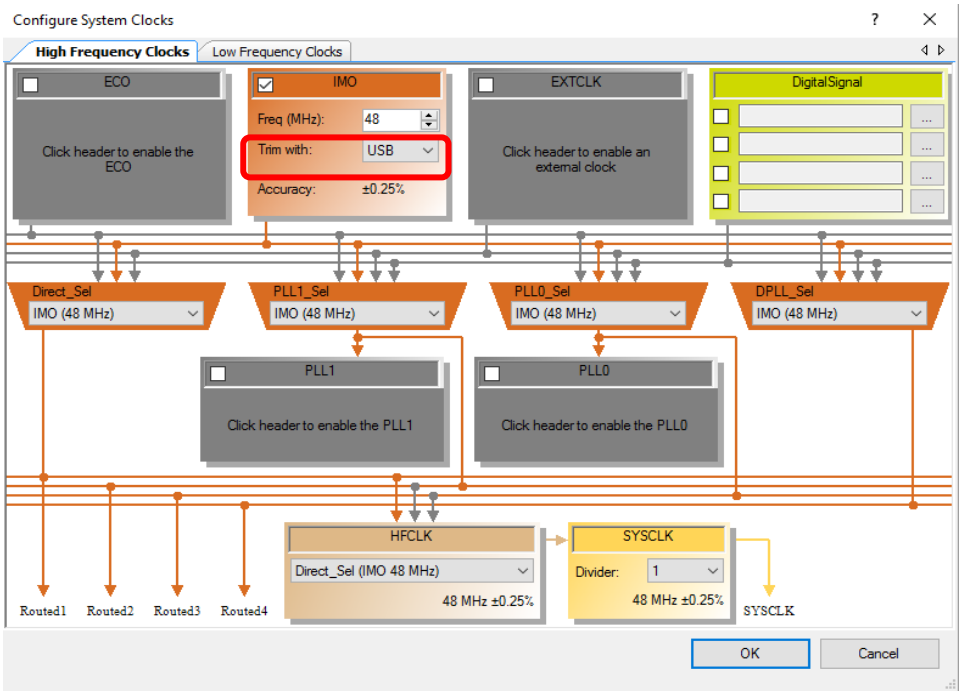
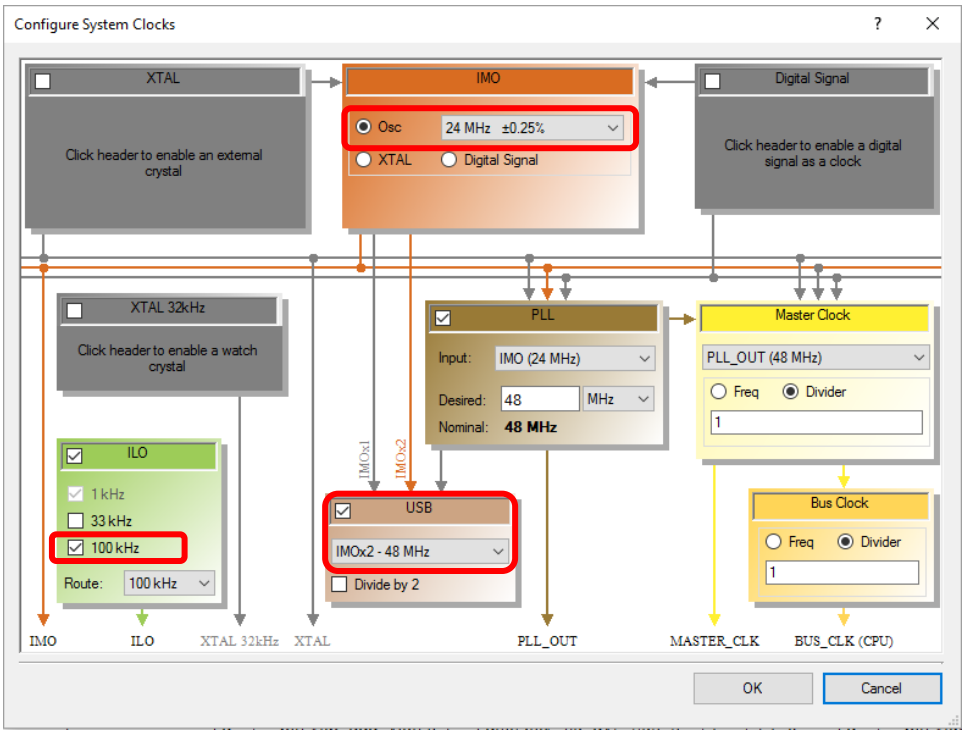


图 8. PSoC 3 和 PSoC 5LP 的 USB 时钟配置



2.6 Bootloader 固件

配置所有工程组件后，添加固件以控制 Bootloader 操作。（推荐的最佳做法是在添加您的固件前，依次选择 **Build > Generate Application** 以执行某个工程。）

在多种情况下，您只要将对 Bootloader 组件 API 函数 `Bootloader_Start()` 的调用添加到您的 *main.c* 内。（如果您已经修改函数调用的“Bootloader”部分的默认名称，那么将其替换为您 Bootloader 组件的名称。）

`Bootloader_Start()` 函数会执行整个引导加载功能。该函数没有返回值 — 将控制权转到应用程序前，它会复位 PSoC 器件。有关 Bootloader 和器件复位的更多信息，请参考附录 B。

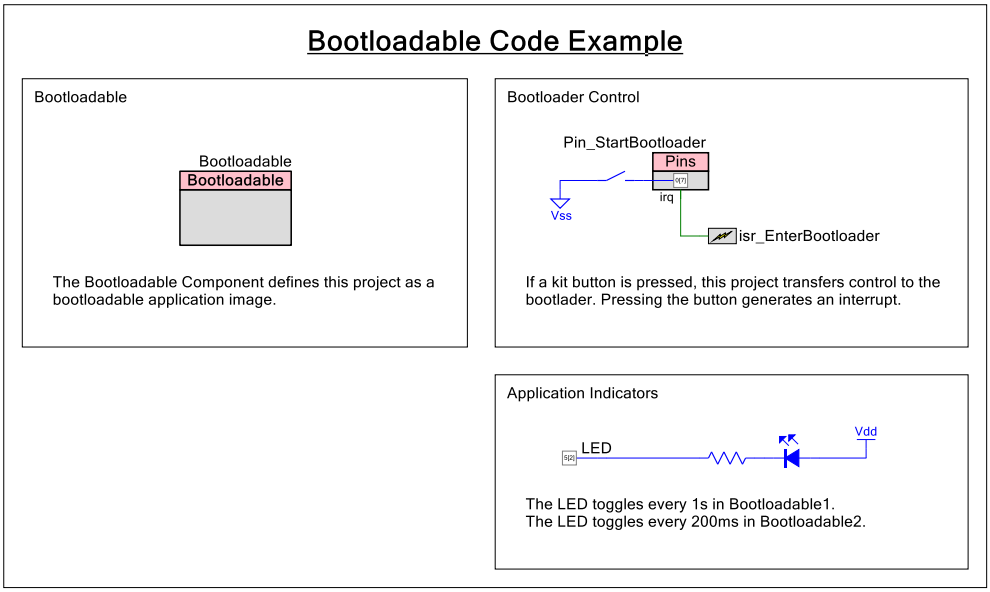
有关如何使用图 3 中显示的 bootloader 指示器和控制引脚的更复杂示例，请参考代码示例 CE95391。

3 Bootloadable（应用）工程

Bootloadable 工程包含 PSoC Creator Bootloadable 组件，如图 9 所示。其他组件用于实现所需的应用。

代码示例 CE95391 具有两个 bootloadable 工程，对于每个工程，套件 LED 会以不同的速率闪烁。这些工程还可以读取套件按键开关的状态。如果该开关处于闭合状态，工程会将控制权转给 Bootloader。

图 9. Bootloader 工程原理图



3.1 创建工程

创建另外一个新的 PSoC Creator 工程。根据 bootloader 工程，选择同样的目标硬件或器件。给该工程命名，如“USB_Bootloadable”。您可以将其添加到 bootloader 工程的工作区内或将其存入另一个工作区中。[1]

现在，将各 PSoC Creator 组件（Bootloader 和其他组件）添加到原理图中。您可以选择根据图 9 中显示的内容重命名各个组件。

[1] 一个 PSoC Creator 工作区可以具有多个工程。在多种情况下，一个 Bootloader 工程和其相关的 Bootloadable 工程全部位于同一个工作区内。但这并不是必要的，Bootloader 和 Bootloadable 工程也可以位于 PC 上单独的工作区内和单独的位置。开始使用 PSoC 前，最好对整个系统的开发需求做出一个工作区/工程计划。

3.2 配置 Bootloadable 组件

Bootloadable 工程需要始终与相关 Bootloader 工程的*.hex* 和*.elf* 输出文件相互关联，如图 10 所示。选择*.hex* 文件时，会自动选择相关的*.elf* 文件。

配置 Bootloadable 组件前，需要先完成构建相关的 Bootloader 工程。

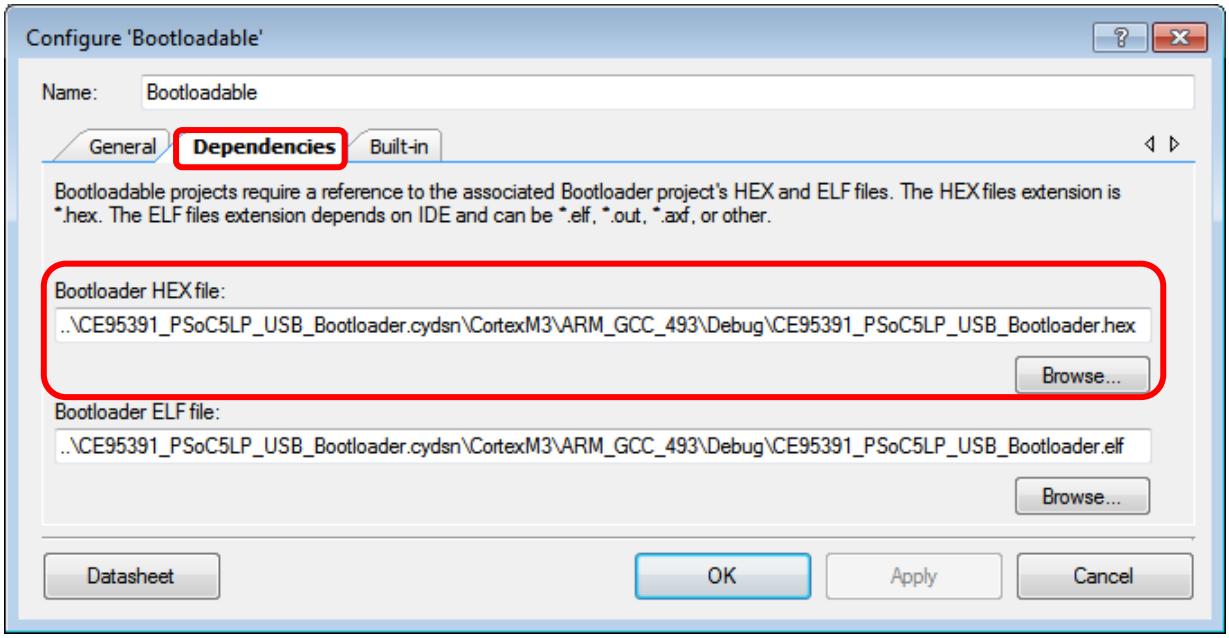
图 10 显示 Bootloader *.hex* 文件位于 PSoC Creator 工作区中相应的路径内。但这不是必要的，Bootloader 和 Bootloadable 工程可位于不同的工作区内。在这种情况下，为 Bootloader 工程的编译器输出寻找*.hex* 文件，具体进行如下选择：

<project folder>\USB_Bootloader.cydsn\DP8051\DP8051_Keil_903\Debug\（对于 PSoC 3）

<project folder>\USB_Bootloader.cydsn\CortexM3\ARM_GCC_493\Debug\（PSoC 4 L 系列和 PSoC 5LP）

更多有关 Bootloader 和 Bootloadable 文件的信息，请参考 AN73854 — PSoC 中的 Bootloader 简介。

图 10. Bootloadable 组件配置



3.3 配置工程的其余部分

配置 Bootloadable 组件后，您可以采用与标准（非 Bootloadable）工程中所使用的方式构建 Bootloadable 工程的其余部分。根据需要将其他组件添加到原理图内，配置它们并将其添加到您的固件中。请参考代码示例 CE95391，以便了解该过程。

您可以随便创建 bootloadable 工程，并使每个工程与先前创建的 Bootloadable 工程相关联。将 bootloader 工程安装在目标硬件中后，您可以下载不同的 Bootloadable 工程，以更改目标功能。

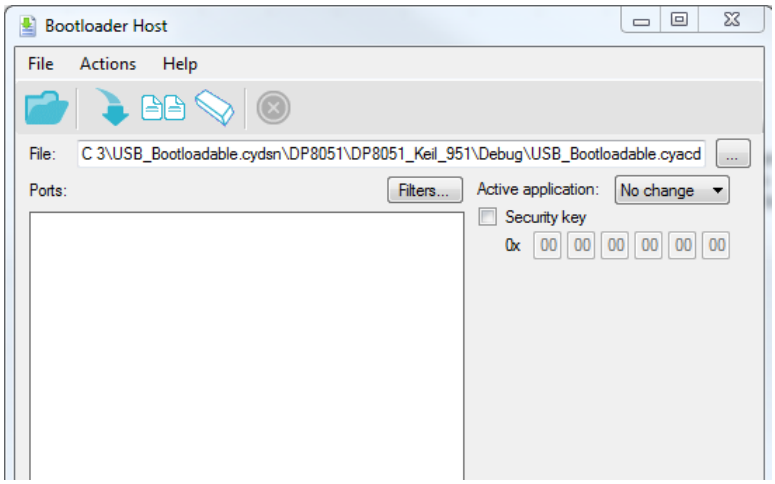
本应用笔记的其余部分介绍了如何使用 PSoC Creator 提供的 Bootloader 主机程序以及如何构建基于您 PC 的 Bootloader 主机。

4 PSoC Creator Bootloader 主机

PSoC Creator 提供了一个 Bootloader 主机程序，便于使用您的 Bootloader 和 Bootloadable 工程。请执行以下操作：

1. 编译您的 Bootloader 工程，并将它编程到目标 PSoC 内。
2. 打开 Bootloader 主机工具（图 11）。依次选择 PSoC Creator 菜单项 **Tools** > **Bootloader Host**。

图 11. PSoC Creator Bootloader 主机程序



3. 点击 **Filters**，并添一个筛选项，从而确定您需要引导加载的 USB 器件（如您的 PSoC 目标），如图 12 所示。

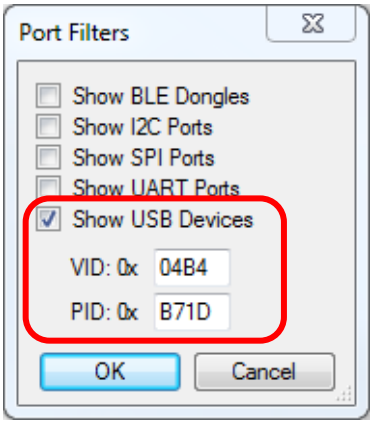
   请确保供应商 ID（VID）和产品 ID（PID）与配置 Bootloader 组件中定义的相互匹配。

4. 点击 **Open File** 图标，并浏览到您的 Bootloadable 文件所在的位置。该文件的类型为*.cyacd*，并且位于 Bootloadable 工程的编译器输出的文件夹中。

   更多有关 Bootloader 和 Bootloadable 文件的信息，请参考 AN73854 — PSoC 中 Bootloader 的简介。

5. 点击 **Program** 或按下 F5 键，以开始进行引导加载。

图 12. USB 筛选项设置



引导加载操作完成后，要等待由 Bootloader 所规定的一段超时才能将 Bootloader 的控制权交给您的 Bootloadable 应用。

5 创建一个 Bootloader 主机

本节介绍的是为实现自定义 USB Bootloader 主机的 Windows 而创建图形用户界面（GUI）应用的方式。这里只介绍了关键步骤。如果您尚不熟悉 Windows 应用开发或 Microsoft Visual Studio，请参考相关资源部分。

本应用笔记随附了一个完整的 Bootloader 主机应用 Visual Studio 工程（*USBBootloaderHost_VC2015.zip*），以供参考。您可以将工程源文件中的代码复制并粘贴到您的工程中。

5.1 所需资源

在创建 Bootloader 主机时，您需要使用下列内容：

Visual Studio

Visual Studio 版本提供了多种开发工具，并支持编程语言（例如 C#、C++）。

请注意，本应用笔记中所介绍的指导针对的是 Visual Studio Community 2015。其他版本的 Visual Studio，也许要求使用其他步骤。

Visual C# Express 2015

这是 Microsoft 的免费 IDE，它使用 C#编程语言开发 .NET 应用。本应用笔记中介绍的指导是用于免费版本，但是您可以使用任何版本（Visual Studio）。

Visual C++ Express 2015

这是 Microsoft 的另一款 IDE，它使用 C++编程语言开发 .NET 应用。在本应用笔记中，Visual C++ Express 使用 C 模块生成一个动态链接库（DLL）。

CYUSB.dll

CYUSB.dll 是赛普拉斯开发并维护的 .NET 动态链接库（DLL），该库用于将 Visual Studio 应用与赛普拉斯 USB 器件连接。该 DLL 被绑定在赛普拉斯 SuiteUSB 封装中，它是用于 Visual Studio 的 USB 开发工具中完整的一部分。可以使用下面的链接，从赛普拉斯网站上下载 SuiteUSB：Cypress SuiteUSB。

PSoC Creator 中提供的 Bootloader API

用于创建主机程序的四个 API 模块（PSoC Creator 提供）能在下面的路径内找到:

<install folder> \ PSoC Creator \ 3.3 \ PSoC Creator \ cybootloaderutils.

依照赛普拉斯 Bootloader 指令/响应协议，这些模块包含了所有用于连接主机端和某个 PSoC Creator Bootloader 组件的代码。更多有关该协议的信息，请参考 Bootloader 组件数据手册或者系统参考指南。

存在四个模块，每个都是一个 .c / .h 文件对：

- cybtldr_api.c / .h

  该模块包含底层函数，用于启动引导加载操作、编写某个 Flash 行、擦除某行、验证某行并结束引导加载操作。

- cybtldr_api2.c / .h

  此模块是更高级别的 API，用于管理整个引导加载过程。该模块具有用于编程器件、擦除器件、验证器件和终止引导加载操作的函数。

- cybtldr_command.c / .h

  该模块构建发送至 Bootloader 的指令数据包，并解析来自 Bootloader 的响应数据包。

- cybtldr_parse.c / .h

  该模块会解析*.*cyacd* 文件，该文件包含要发送至器件的 Bootloadable 镜像。

5.2 创建一个 Bootloader 主机应用程序

该流程包括以下主要阶段：

第一阶段：创建一个带有支持 Bootloader 的函数（由 PSoC Creator 提供）的 DLL（动态链接库）。

第二阶段： 创建一个 C# Windows Form 应用程序（即一个 GUI）。

第三阶段：定义通讯函数。

第四阶段：从第一阶段创建的 DLL 中导入基本 Bootloader 函数。

第五阶段：修改 Form 的函数。

第六阶段：定义主机错误代码。

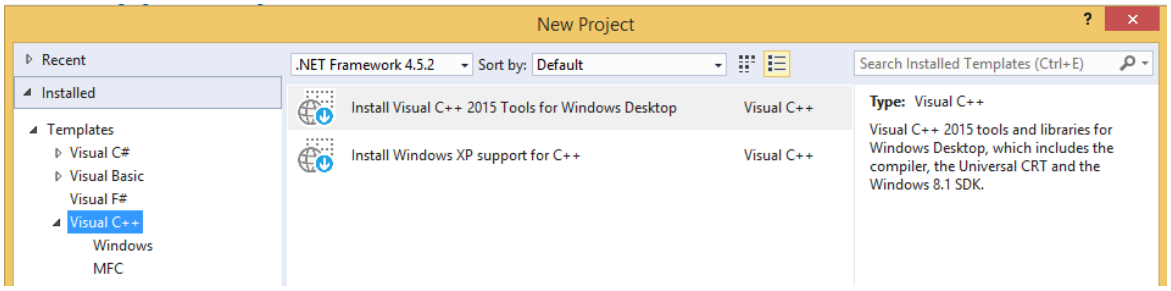
下面各章节详细介绍了各个阶段：

5.2.1 第一阶段：创建一个 DLL

首先，通过使用 PSoC Creator Bootloader C 文件创建一个 DLL。更多有关信息，请参考附录 C — 主机内核 API。

1. 运行 Visual Studio Express。
2. 创建一个新的 C++ DLL 工程。
   您可能需要为 Windows 桌面安装 Visual C++ 2015 工具，以便看到 Win32 控制台应用程序，如图 13 所示。依次选择菜单项 **File** > **New** > **Project…** > **Visual C++** > **Win32 Console Application**。

图 13. 为 Windows 桌面安装 Visual C++ 2015 工具



3. 为工程提供一个合适的名字，如 Bootloader_Utils。点击 **OK**，然后点击 **Next**。
4. 将 **Application type** 设置为 DLL，并将 **Additional options** 设置为空工程。点击 **Finish**，以应用这些设置，并创建一个新的工程。
5. 在 Solution Explorer 窗口中，右键点击工程名称（“Bootloader_Utils”），并依次选择 **Add** > **Existing Item…**。添加以下文件（位于 PSoC Creator 文件夹 *<install folder>* \ *PSoC Creator* \ *3.3* \ *PSoC Creator* \ *cybootloaderutils* 中）：
   - cybtldr_api.h、cybtldr_api.c
   - cybtldr_api2.h、cybtldr_api2.c
   - cybtldr_parse.h、cybtldr_parse.c
   - cybtldr_command.h、cybtldr_command.c
   - cybtldr_utils.h
6. 右键点击工程名称（“Bootloader_Utils”），并对工程属性添加一个预处理器定义。依次选择以下菜单项：
   Project Properties > **Configuration Properties** > **C/C++** > **Preprocessor** > **Preprocessor Definitions** > **Edit…**
   在预处理器定义后面插入“_CRT_SECURE_NO_WARNINGS”。
7. 编译工程 — 依次选择菜单项 **Build** > **Build Solution**。这样会创建 *Bootloader_Utils.dll* 文件。

5.2.2 第二阶段：创建一个 Windows Form 应用程序

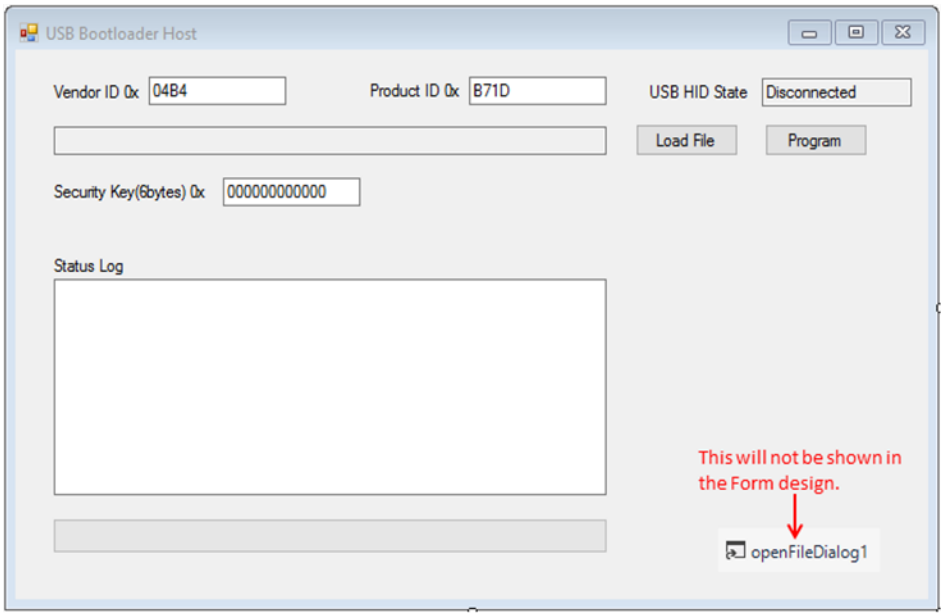
下一阶段将为 Bootloader 主机创建一个 GUI。GUI 是使用 Visual C#创建的。主要要求如下：

- 识别并连接到某个基于器件 VID 和 PID 的 HID 器件。更多相关信息，请参考相关资源中的 USB 应用笔记
- 放置一个输入框，以便用户输入一个安全密钥
- 允许用户选择一个新的.*cyacd* 文件引导加载到器件上
- 将.*cyacd* 文件编程到器件内
- 使用 x86 目标平台（**Project Properties** > **Build** = > **Platform target** > **x86**）
- 使用.NET Framework 4.x 时，一些系统可能会发生构建错误，因此应该使用.NET Framework 3.5。依次选择菜单项：**Project Properties** > **Application** > **Target framework** > **.NET Framework 3.5**

请执行以下操作：

1. 运行 Visual Studio。
2. 创建一个新的 C#工程。依次选择以下菜单项：
   File > **New** > **Project…** > **Visual C#** > **Windows Forms Application**。
3. 为工程提供一个合适的名字，如 USBBootloaderHost，然后点击 **OK**。
4. 设计 Form。将工具箱中的五个标签、五个文本框、两个按键、一个进度条以及一个 openFileDialog 放置在主要窗口中，如图 14 所示。提供一个合适的名称和默认值。

图 14. 自定义 Bootloader Host GUI



label1.Text = “Vendor ID 0x”
label2.Text = “Product ID 0x”
label3.Text = “USB HID State”
label4.Text = “Security Key(6bytes) 0x”
label5.Text = “Status Log”
textBox1.Text = “04B4”
textBox2.Text = “B71D”
textBox3.Text = “Disconnected”
textBox3.ReadOnly = true
textBox4.Text = “000000000000”
textBox5.Text = “”, textBox5.Multiline = true
textBox6.Text = “”, textBox6.ReadOnly = true

5.2.3 第三阶段：定义通讯函数

下一步将把 *CyUSB.dll* 中的 CyUSB 函数导入到 C#应用程序中。请注意，CyUSB 函数被视为非托管代码，因为它们并非被.NET 框架创建。更多有关非托管代码的信息，请参考为非托管代码进行安全编码指南。

要导入这些函数，请按下面各步进行操作：

1. 将 *CyUSB.dll* 复制到您的工程文件夹中。如果已经安装了 Cypress SuiteUSB，那么 *CyUSB.dll* 位于下面路径内：
   C:\Cypress\Cypress Suite USB 3.4.7\CyUSB.NET\lib.
2. 右键点击工程名称或 References，并选择 CyUSB.dll：
   References > **Add Reference…** > **Browse…** > **CyUSB.dll**
3. 勾选 CyUSB.dll 复选框，然后点击 **OK**。
4. 添加一个新的类文件：右键点击工程名称，然后依次选择 **Add** > **New Item…** > **Class**。为类文件提供一个合适的名称，例如 *Bootloader_Utils.cs*。
5. 将代码添加到 *Bootloader_Utils.cs* 中，如页 15 上的代码 1 所示。另外，您还可以通过 *Bootloader_Utils.cs*（位于本应用笔记随附的一个完整 Bootloader 主机 Visual Studio 工程 *USBBootloaderHost_VC2015.zip* 内）将代码复制并粘贴到您的工程中。

5.2.4 第四阶段：导入 Bootloader 函数

下一步将 DLL 中 Bootloader 函数（*Bootloader_Utils.dll*）导入 C#应用程序中。该步骤通过使用 **dllimport** 修饰符实现，请参考代码 1。

1. 将代码添加到 *Bootloader_Utils.cs* 中，如代码 1 所示。另外，您还可以通过 *Bootloader_Utils.cs* 文件（位于本应用笔记随附的一个完整 Bootloader 主机应用 Visual Studio 工程 *USBBootloaderHost_VC2015.zip* 内）将代码复制并粘贴到您的工程中。
2. 将第一阶段创建的 *BootLoader_Utils.dll* 复制到您的工程输出文件夹内：..\bin\x86\Debug

5.2.5 第五阶段：修改窗口函数

必须将几个事件和流程的代码添加到 Form 中，具体包括：

- USB HID 连接和通讯设置
- 文件管理
- 显示进度

这是将 *Form1.cs* 文件中的代码复制并粘贴到工程中的最简单方式。Form1.cs 文件位于一个完整的 Bootloader 主机应用 Visual Studio 工程（即，本应用笔记随附的 *USBBootloaderHost_VC2015.zip*）中。然后，您可以为您的应用程序修改代码。

5.2.6 第六阶段：定义主机错误代码

创建一个类文件，并给其提供一个合适的名称，如 *Bootloader_enum.cs*。枚举定义必须与 *cybtldr_utils.h* 中的相匹配。这是将 Bootlo*ader_enum.cs* 文件中的代码复制并粘贴到工程中的最简单方式；Bootloader_enum.cs 文件位于一个完整的 Bootloader 主机应用 Visual Studio 工程（即本应用笔记随附的 *USBBootloaderHost_VC2015.zip*）中。

代码 1. Bootloader_Utils.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Runtime.InteropServices;

namespace USBBootloaderHost
{
    class Bootloader_Utils
    {
        // Communication control structure
        [StructLayout(LayoutKind.Sequential)]
        public struct CyBtldr_CommunicationsData
        {
            public OpenConnection_USB OpenConnection;
            public CloseConnection_USB CloseConnection;
            public ReadData_USB ReadData;
            public WriteData_USB WriteData;
            public uint MaxTransferSize;
        };

        // Unmanaged code handling
        [UnmanagedFunctionPointer(CallingConvention.Cdecl)]
        public delegate int OpenConnection_USB();

        [UnmanagedFunctionPointer(CallingConvention.Cdecl)]
        public delegate int CloseConnection_USB();

        [UnmanagedFunctionPointer(CallingConvention.Cdecl)]
        public delegate int ReadData_USB(IntPtr buffer, int size);

        [UnmanagedFunctionPointer(CallingConvention.Cdecl)]
        public delegate int WriteData_USB(IntPtr buffer, int size);

        // DLL importing
        [UnmanagedFunctionPointer(CallingConvention.Cdecl)]
        public delegate int CyBtldr_ProgressUpdate(byte arrayID, ushort rowNum);
        [DllImport("BootLoader_Utils.dll", CallingConvention =
                   CallingConvention.Cdecl)]
        public static extern int CyBtldr_Program(string file, byte[] securityKey,
                                                 byte appId,
                                                 ref CyBtldr_CommunicationsData comm,
                                                 CyBtldr_ProgressUpdate update);
    }
}
```

6 总结

Bootloader 是一种用于执行产品领域升级的有用方法。每个 Bootloader 都需要一个硬件通讯接口，用于与主机连接。USB 是一个广泛使用的通信接口。

本应用笔记和代码示例 CE95391 中介绍的 PSoC USB HID Bootloader 提供了一种可靠的解决方案，这样您便能够快速启动和运行它。USBFS 组件被配置为 HID 类别，因此它不需要自定义驱动程序，但仍能够在任何操作系统中工作。

PSoC Creator 提供了 Windows Bootloader 主机程序。另外，本应用笔记还显示了如何使用赛普拉斯所提供的文件来创建您自己的 Windows Bootloader 主机应用。

7 相关资源

代码示例

- CE95391 — USB HID Bootloader 代码示例

应用笔记

- AN54181 — PSoC 3 入门
- AN79953 — PSoC 4 入门
- AN77759 — PSoC 5LP 入门
- AN73854 — PSoC 3、PSoC 4 和 PSoC 5LP Bootloader 简介
- AN60317 — PSoC 3 和 PSoC 5LP I^2C Bootloader
- AN86526 — PSoC 4 I^2C Bootloader
- AN68272 — PSoC 3、PSoC 4 和 PSoC 5LP UART Bootloader
- AN84401 — PSoC 3 和 PSoC 5LP SPI Bootloader
- AN57473 — PSoC 3 和 PSoC 5LP 中 USB HID 基础知识（Mouse 和 Joystick 基本原理）
- AN58726 — PSoC 3 和 PSoC 5LP 的 USB HID 中级应用笔记（键盘和复合设备）
- AN56377 — PSoC 3 和 PSoC 5LP：简述实现 USB 数据传输

其他信息

- SuiteUSB — 适用于 Visual Studio 的赛普拉斯 USB 开发工具
- EZ-USB FX3 软件开发套件

关于作者

姓名： Robert Murphy
职务： 应用工程师
背景信息： Robert Murphy 毕业于美国普渡大学，并获得了电气工程技术学士学位

姓名： Keith Mikoleit
职务： 高级应用工程师
背景信息： Keith Mikoleit 毕业于西华盛顿大学，获得电气工程技术学士学位

A 附录 A — USBFS HID 配置

该部分详细介绍了如何设置一个 USB bootloader 描述符。请注意，您可以根据 bootloader 模板文件（*bootloader.root.xml*）输入描述符，如配置 Bootloader 组件一节的内容。

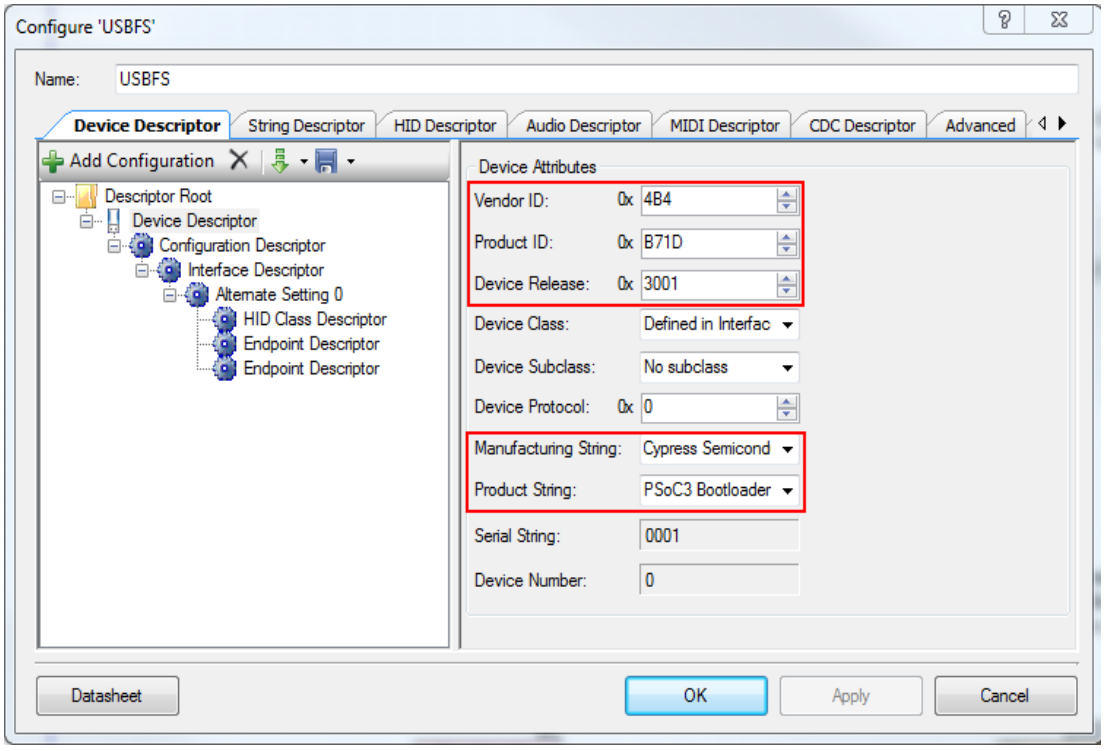
谨记，该应用采用的是 USB HID 类别，这样便不再需要外部提供的驱动程序。

开始配置 USBFS 组件时，请双击您的 PSoC Creator 工程原理图中的组件符号。

A.1 创建器件描述符

1. 在**描述符**树中，点击 **Device Descriptor**。按照下面显示的内容配置 **Device Attributes** 中的各个选项（请参考图 15）：
   - **Vendor ID**：0x04B4
   - **Product ID**：0xB71D
   - **Device Release**：0x0101
   - **Manufacturing String**：Cypress Semiconductor
   - **Product String**：PSoCx Bootloader

图 15. USBFS 器件描述符

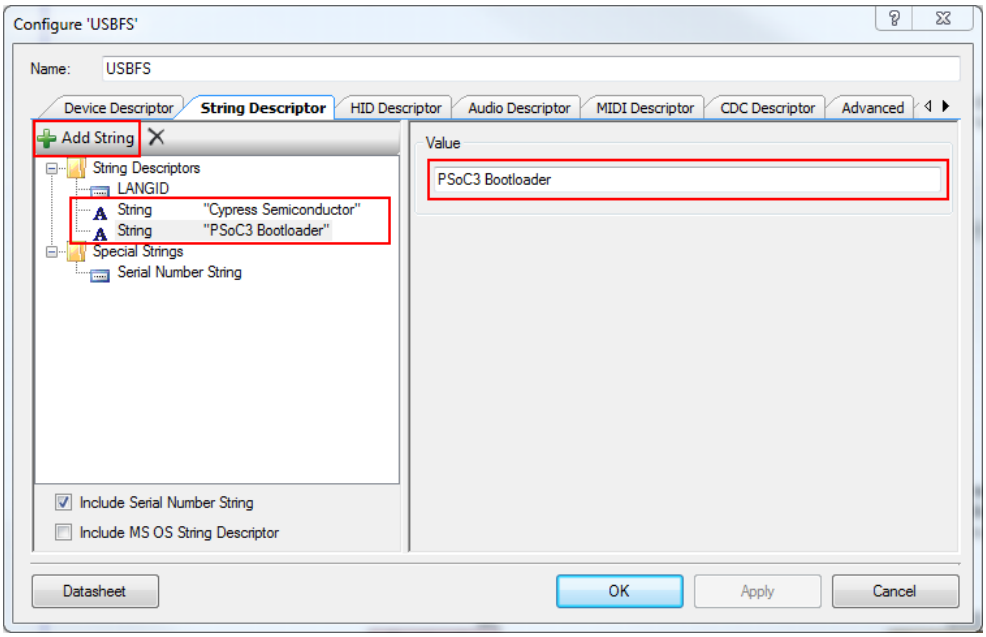


这里唯一的硬件要求是修改供应商 ID（VID）和产品 ID（PID）。VID 使用的（0x04B4）是一个特定的赛普拉斯半导体 VID。可将其用于该实例。但是当您开发的应用用于生产是，您必须使用分配给您公司的 VID。该应用中所选用的 PID 是唯一的。Bootloader 主机应用通过 VID 和 PID 来识别器件。

您可以选择更改各个字符串，如制造商字符串和产品字符串。

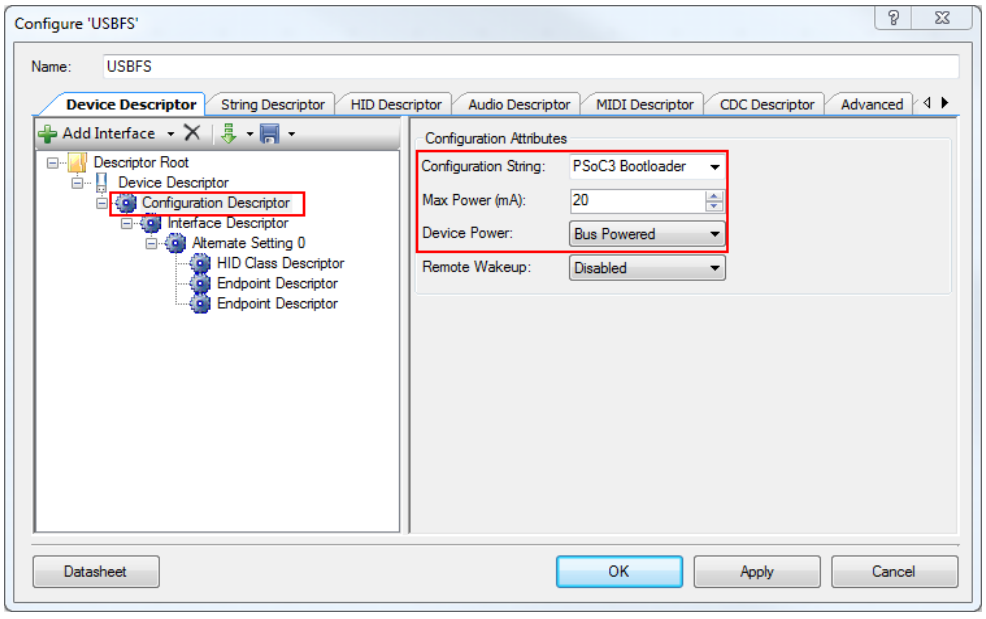
2. 请在 **StringDescriptor** 树中点击 **Add String** 选项。具体信息，请参考下面的图 16：

图 16. 字符串描述符



3. 在 **Device Descriptor** 树中，点击 **Configuration Descriptor** 项。按照下面显示的内容配置 **Configuration Attributes** 中的各选项（请参考图 17）：

图 17. USBFS 配置描述符

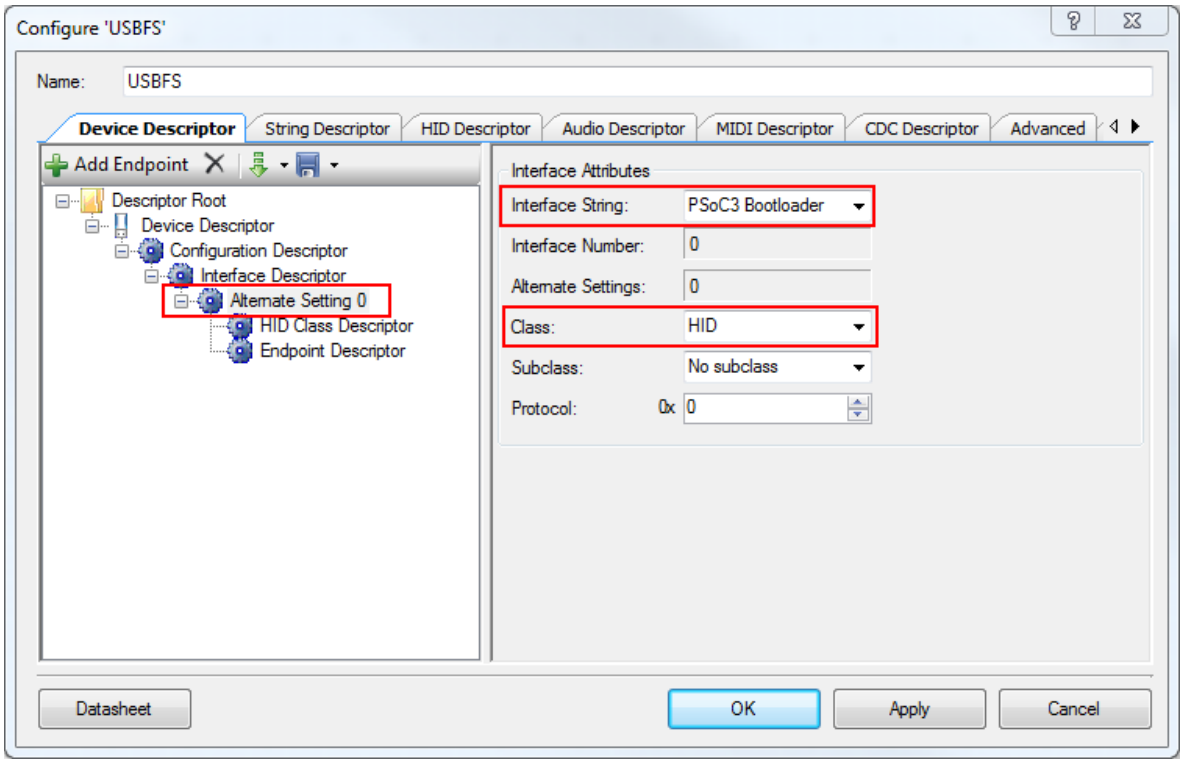


该步骤中重要的设置是将 USB 器件定义为总线供电，以及从主机请求 20 mA 功耗的电源预算。小于 20 mA 的应用可接受，但不能超过这个要求。

由于该步骤不要远程唤醒功能，因此禁用了它。

4. 在 **Interface Descriptor** 树中，点击 **Alternate Setting 0** 项。请按照图 18 中所示的内容配置 **Interface Attributes**：
   - **接口字符串**：PSoCx Bootloader
   - **类别**：HID
   - **子类别**：无子类别

图 18. USBFS 的备选设置



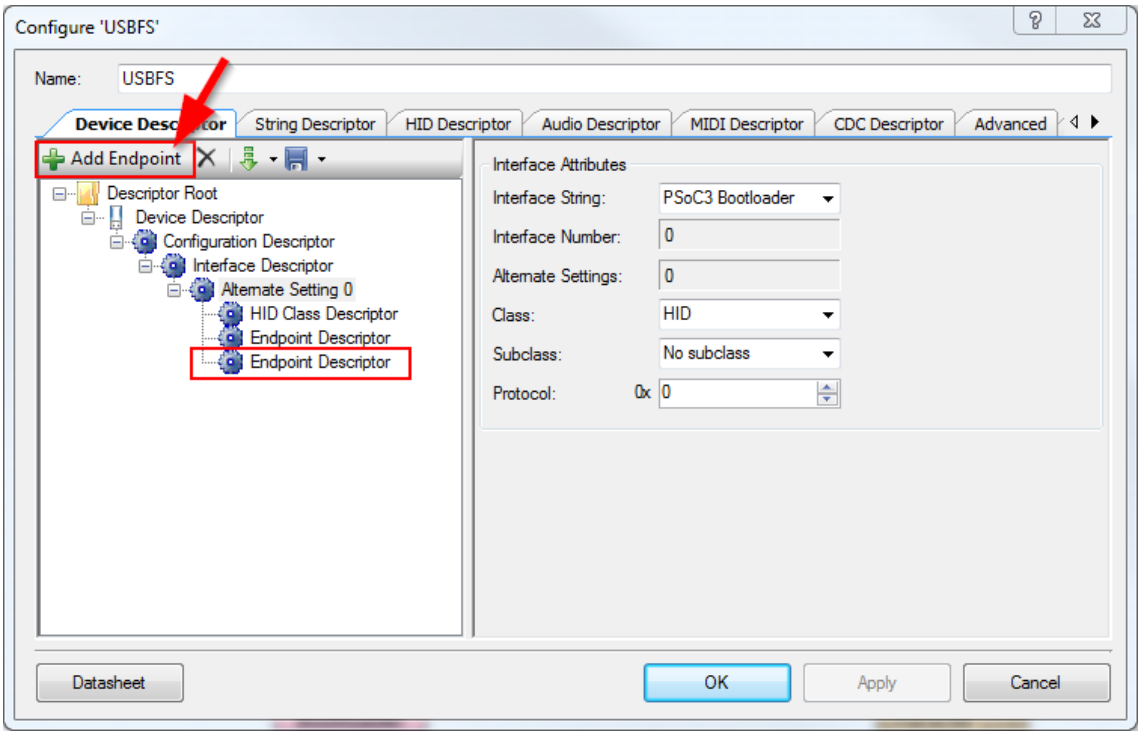
每个接口可以有多个“Alternate Settings”（备选设置），这样允许多终端配置。在这里，我们只有一个备选设置（默认）。

这也是我们将器件配置指定的 HID 类别。请注意，如果您将“Class”从 **Undefined** 修改为 **HID**，那么在“Descriptor Root”树中会显示一个附加的描述符，即 HID 类别的描述符。

请注意，我们尚未设置 HID 报告。如果这时候您点击“OK”或“Apply”，您会收到一条错误信息，这是因为尚未定义 HID 报告。请按照下面几个步骤设置该报告：

5. 点击 **Add Endpoint** 按键一个附加的 **Endpoint Descriptor** 将出现在 **Alternate Setting 0** 树中，如图 19 所示。

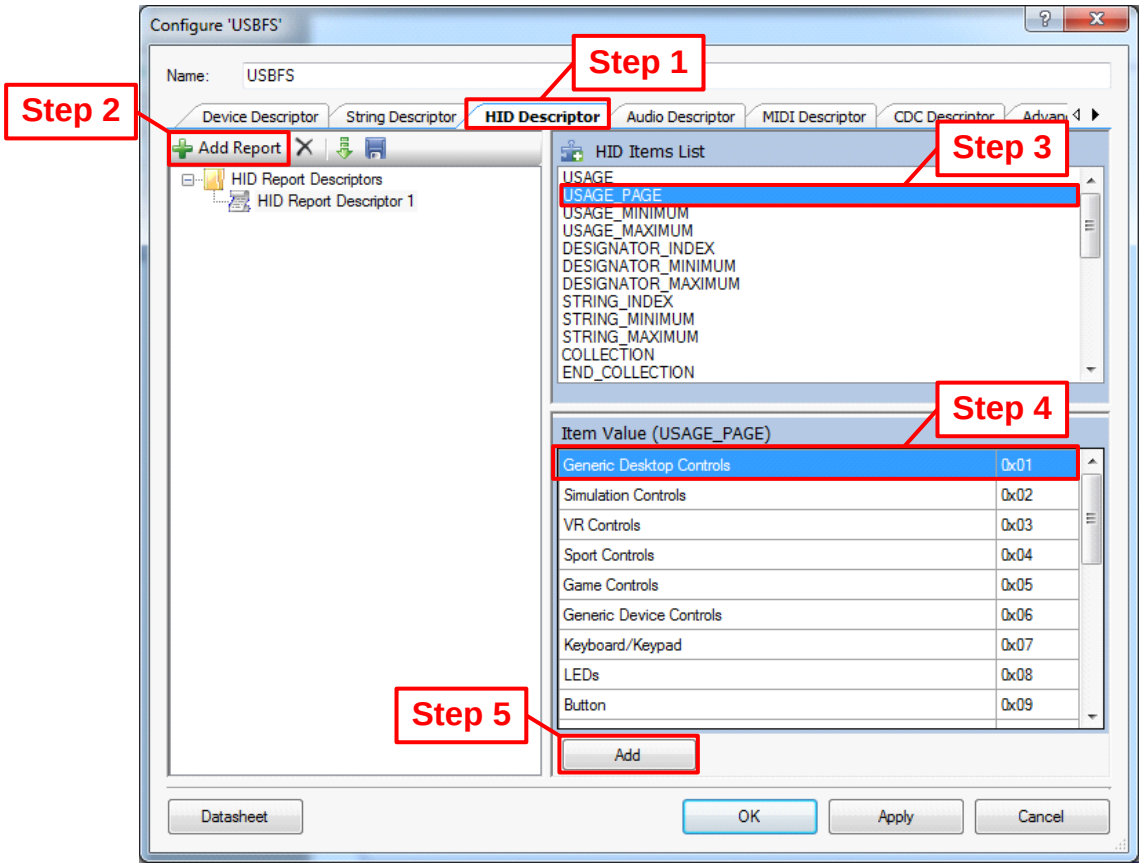
图 19. USBFS 端点添加



A.2 创建 HID 描述符

1. 使用 **HID 描述符**接口创建一个 HID 报告描述符，如图 20 所示：

图 20. HID 描述符



a. 点击 **HID Descriptor** 选项卡。

b. 点击 **Add Report** 按建。
按照第 22 页中显示的表 1 执行步骤 3 到步骤 5 的操作。必须将报告项按顺序添加到表中。

c. 从 **HID Item** 列表中选择某一项。

d. 根据该表，从 **Item Value** 列表中选择某个值。

e. “Item Value”框中包含了某个用于所选特殊项的字段时，请点击“Decimal”或“Hexadecimal”单选项，并在字段中输入所需值。

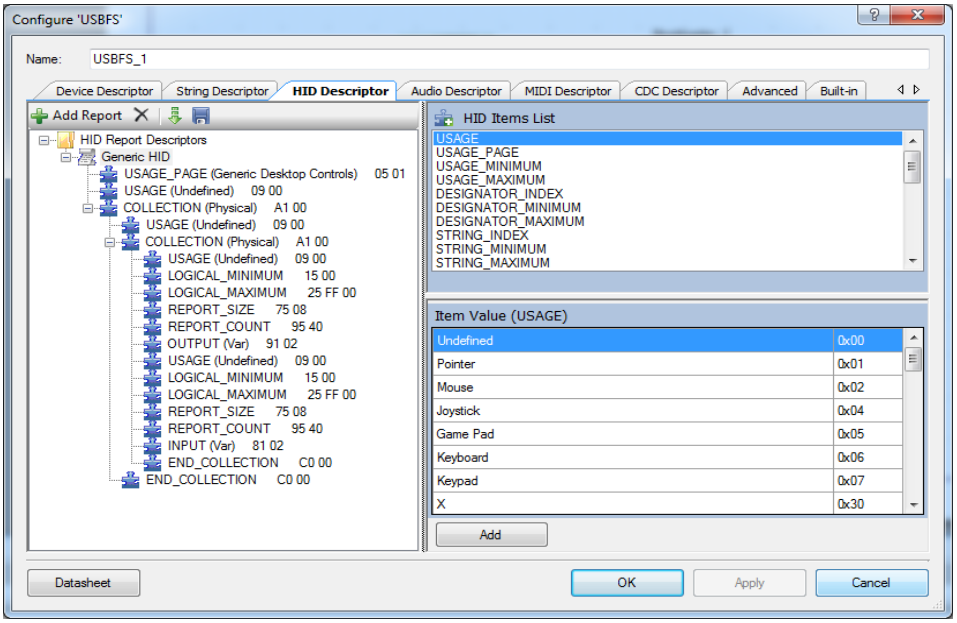
f. 重复执行步骤 3 到步骤 5，直到报告描述符与第 22 页中所显示的图 21 中的内容相似为止。

g. 将 HID 描述的名称改为“Generic HID”。

表 1. HID 描述符项

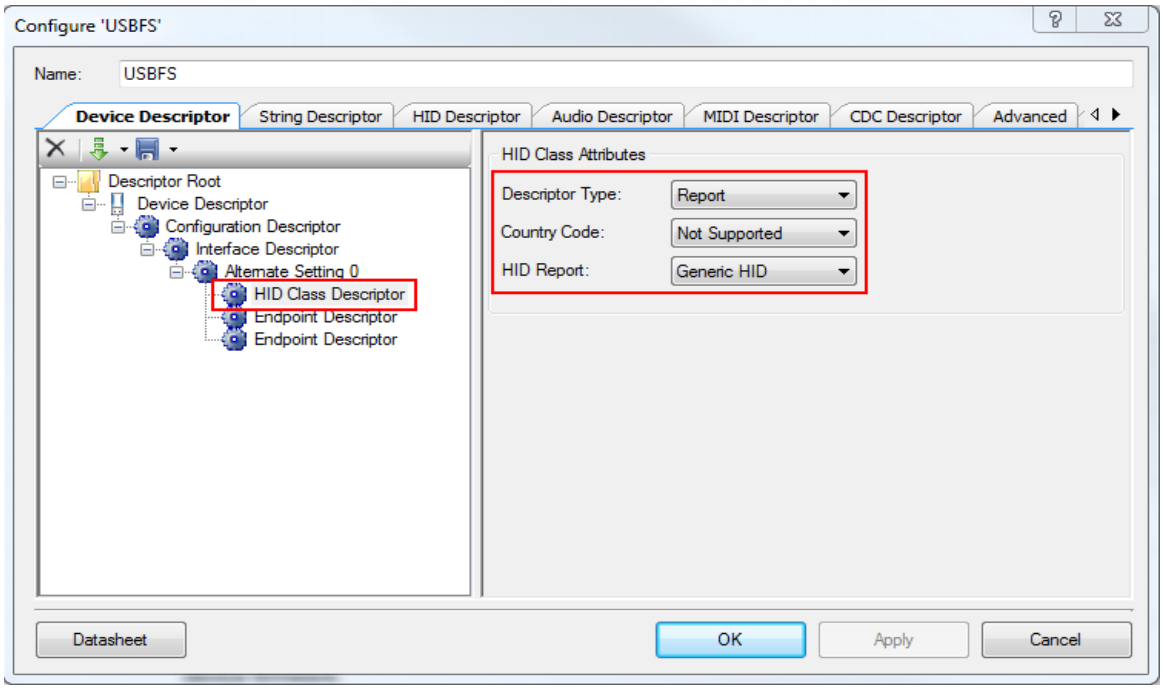
HID 项	项目值（USAGE）
USAGE PAGE (05)	通用的桌面控制地址 0x01
USAGE (09)	未定义地址 0x00
COLLECTION (A1)	物理地址 0x00
USAGE (09)	未定义 0x00
COLLECTION (A1)	物理地址 0x00
USAGE (09)	未定义地址 0x00
LOGICAL_MINIMUM (15)	0x00
LOGICAL_MAXIMUM (25)	0xFF
REPORT_SIZE (75)	0x08
REPORT_COUNT (95)	0x40
OUTPUT (91)	位 1 — 变量
USAGE (09)	未定义地址 0x00
LOGICAL_MINIMUM (15)	0x00
LOGICAL_MAXIMUM (25)	0xFF
REPORT_SIZE (75)	0x08
REPORT_COUNT (95)	0x40
INPUT (81)	位 1 — 变量
END_COLLECTION (C0)	无
END_COLLECTION (C0)	无

图 21. 完成 HID 描述符



2. 点击 **Device Descriptor** 选项卡。在**描述符**树中，点击 **HID Class Descriptor**。请按照下面的图 22 中所示的内容配置 **Device Attributes** 中的各个选项：
   - **描述符类型**：Report
   - **国家/地区代码**：不支持
   - **HID 报告**：通用 HID

图 22. USBFS HID 器件属性



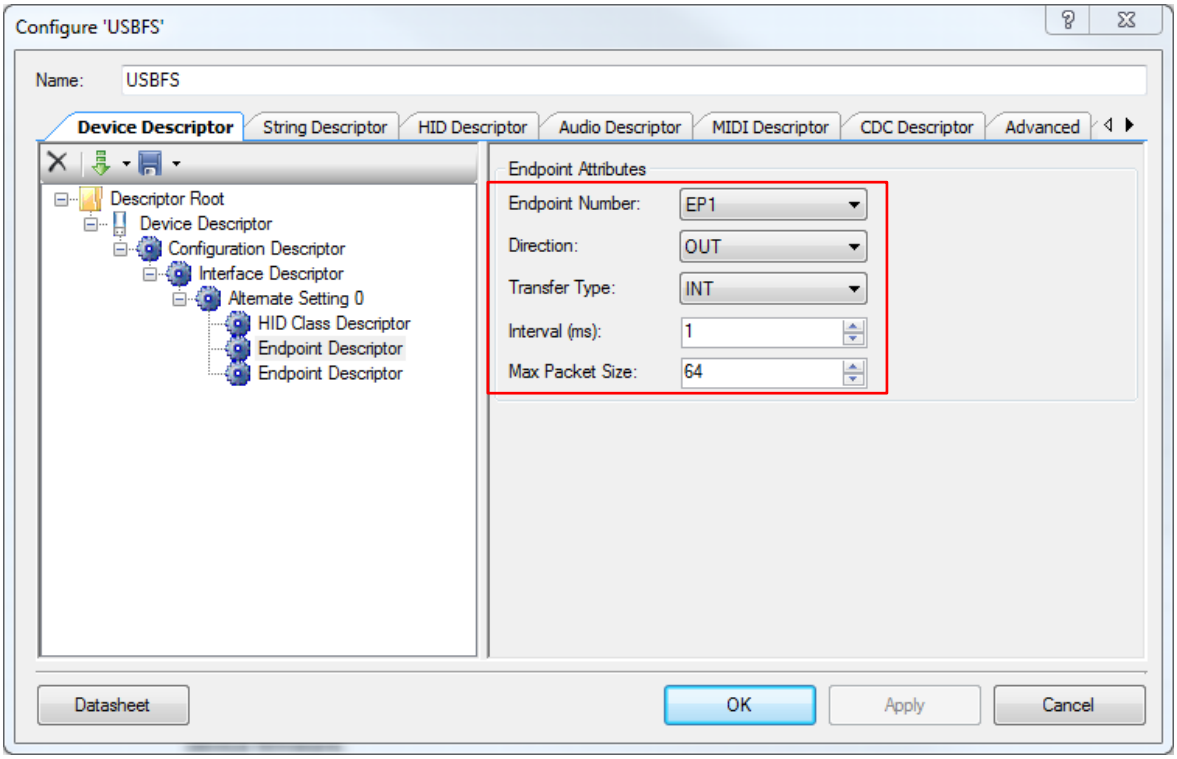
由于该工程采用的是 Report Descriptor，因此我们要将“Descriptor Type”设置为“Report”。

由于该工程并不特定于某些国家/地区，因此我们将“Country Code”设置为“Not Supported”。

最后，“HID Class Descriptor”必须指向我们前一步中所创建的“HID Report Descriptor”。为了创建该链接，需要将“HID Report”匹配为“HID Report Descriptor”（它在该实例中被标记为“Generic HID”）。

3. 在 **Descriptor** 树中，点击第一个 **Endpoint Descriptor** 项。请按照图 23 中所示的内容配置 **Endpoint Attributes** 中的各选项：
   - **Endpoint Number**：EP1
   - **Direction**：OUT
   - **Transfer Type**：INT
   - **Interval**：10
   - **Max Packet Size**：64

图 23. USBFS 端点 1



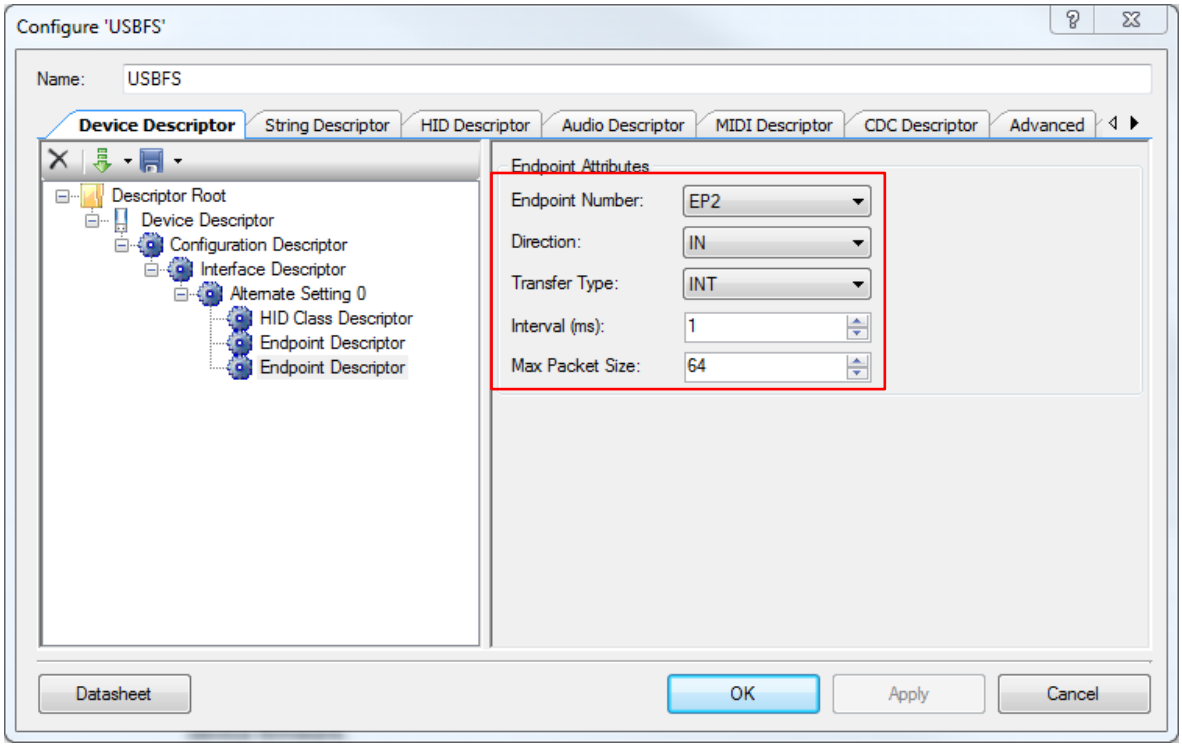
第一个端点（EP）作为“OUT”端点使用。它作为接受来自主机的数据的缓冲器。术语 USB IN 和 OUT，始终针对的是 USB 主机。

首先，我们将该端点设置为“Endpoint 1”（EP1）。然后，将端点的方向设置为“OUT”。

由于该应用是一个 HID，规范要求我们使用中断（INT）传输。

4. EP2 作为将要发送给主机的数据的缓冲器。为第二个端点重复执行步前面步骤，从而将其配置为一个具有以下特性的 IN 端点（参考图 24）：
   - **Endpoint Number**：EP2
   - **Direction**：IN
   - **Transfer Type**：INT
   - **Interval**：10
   - **Max Packet Size**：64

图 24. USBFS 端点 2



5. 点击 **Apply** 按键，然后点击 **OK** 按键，以关闭配置向导。

B 附录 B — Bootloader 和器件复位

如本应用笔记中另外指出的内容，从 Bootloader 交给 Bootloadable 控制权（反之亦然）始终要通过器件复位来执行。如果您的系统必须继续执行关键的任务，同时要将一个程序更改为另一个，则需要考虑这项内容。本节详细说明了使用复位功能的原因，以及在您的应用中复位对器件性能产生的影响。

B.1 为什么需要执行器件复位？

要想了解需要进行器件复位的原因，需要注意在您的系统中 Bootloader 和 Bootloadable 工程是完全独立的 PSoC Creator 工程。每个工程都有自己的器件配置内容。因此，将一个工程更改为另一个工程时，您可以完全重新定义 PSoC 器件的硬件功能。

为了实现复杂的自定义功能，器件配置可能包含对数千个 PSoC 寄存器的设置。对于 PSoC 的数字和模拟路由功能会发生这种情况。配置寄存器和路由时，除了要设置新配置的位之外，您还要对旧配置的位进行复位。否则，不会执行新的配置，甚至会损坏器件。

因此，对 Bootloader 工程和 Bootloadable 工程互相进行切换时，需要执行器件软件复位（SRES）操作。这样会使所有 PSoC 寄存器被复位为默认状态。然后可以开始对新工程进行配置。请注意，假设所有 PSoC 寄存器均被初始化，从而将它们复位为器件默认状态，我们可以缩短配置时间并减少闪存使用量。

B.2 对 PSoC 3 和 PSoC 5LP 器件 I/O 引脚的影响

如应用笔记 AN61290：PSoC 3 和 PSoC 5LP 硬件设计中介绍的注意事项和 AN60616：PSoC 3 和 PSoC 5LP 启动过程中介绍的内容，在复位和启动过程中 PSoC 5LP I/O 引脚处于三种不同的驱动模式，如表 2 所示。PSoC 4 L 系列的 I/O 引脚特性也与此相似。

表 2. 器件复位时 PSoC 3 和 PSoC 5LP I/O 引脚驱动模式

启动事件	I/O 引脚驱动模式	持续时间（典型值）		备注
		低速 IMO（12 MHz）	快速 IMO（48 MHz）	
器件复位（SRES）被使能 器件复位被禁用	高阻模拟	40 µs		复位被使能时，将 I/O 保持为高阻模拟模式。
非易失性锁存器（NVL）复制到 I/O 端口 开始执行代码	NVL 设置： 高阻模拟、 上拉或下拉	~12 ms	~4 ms	持续时间取决于代码执行的速度和配置的复杂性。
配置 I/O 端口和引脚	PSoC Creator 工程配置	n/a		8 种可行的驱动模式。有关详细信息，请参考器件数据手册。
代码达到 main()函数	代码可以更改 I/O 引脚功能	n/a		

有关 PSoC 3 和 PSoC 5LP 中 NVL 的使用情况，请参考器件数据手册。在您的 PSoC Creator 工程中，NVL 设置是在两个位置中进行的：

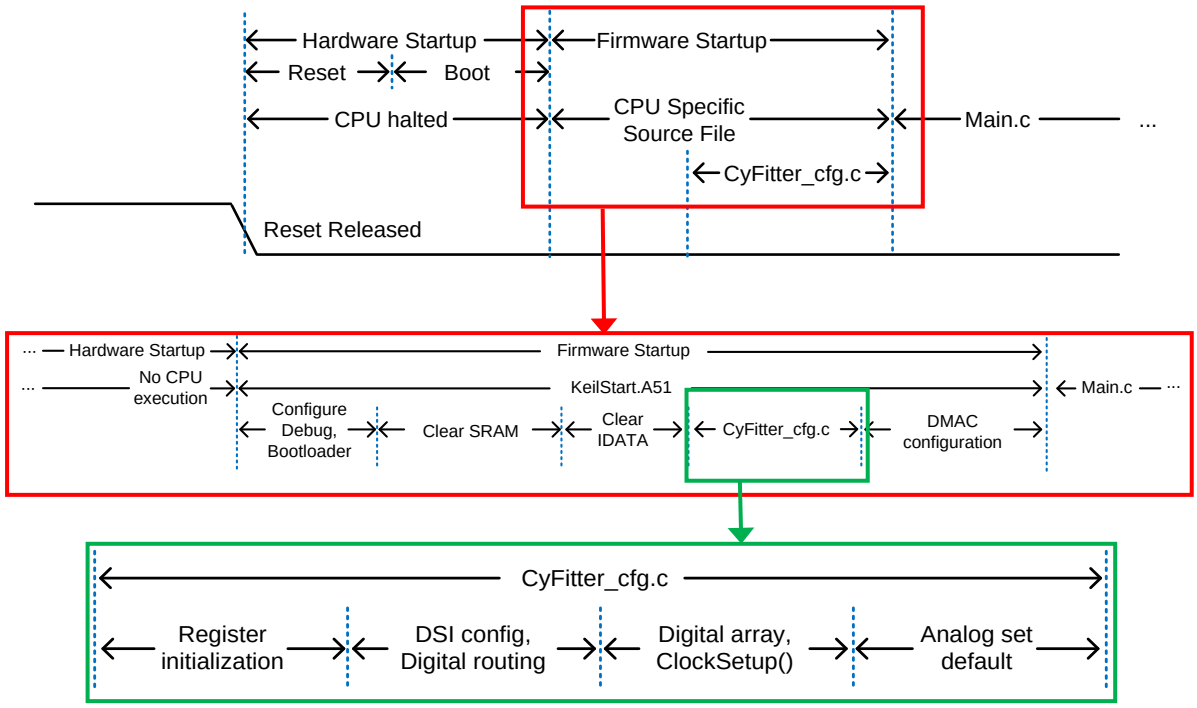
- 用于 I/O 端口的 **Reset** 选项卡（该选项卡位于单独引脚组件配置中）
- 用于所有其他 NVL 的 **System** 选项卡（该选项卡位于设计范围资源（DWR）窗口中）

当使用您的工程来编程器件时，NVL 被更新。请注意，Bootloadable 工程不可设置 NVL；其 DWR 设置必须与相关 Bootloader 工程中的 DWR 设置相匹配。

最后 I/O 驱动模式通过单独引脚组件配置进行设置。

图 25 显示的是器件启动和配置的时序图。中间的图片显示的是 PSoC 3 示例；PSoC 4 L 系列和 PSoC 5LP 的流程与其相似。有关详细信息，请参见 AN60616：PSoC 3 和 PSoC 5LP 启动流程。

图 25. 器件启动流程图



B.3 对其他功能的影响

器件复位时，通用数字模块（UDB）寄存器被复位，因此不存在任何基于 UDB 的组件，并且它们的功能均被禁用。对于基于可配置的 SC/CT 模块（PSoC 3 和 PSoC 5LP）和 CTBm 模块（PSoC 4 L 系列）的模拟组件也存在相同的情况。

所有固定的外设（数字和模拟）都被复位为它们的闲置状态。这些固定外设包括 DMA、DFB、定时器（TCPWM）、I^2C、USB、CAN、ADC、DAC、比较器以及运算放大器。除 IMO 外，所有其他时钟均被停止。

所有数字和模拟路由控制寄存器都被复位。这样会打开所有数字和模拟开关，使器件的所有连接断开。并且包括与 I/O 相关的全部连接（除与 NVL 的连接外）。

配置后，所有硬件功能都得到恢复（请参考图 25）。开始执行工程的 main()函数时，所有固件功能都得到恢复。

B.4 示例：风扇控制

通过该示例，我们可以了解 Bootloader 和其相关联的器件复位是如何集成到一个典型的应用（如，风扇控制）内的。PSoC Creator 提供了一个风扇控制器组件，该组件安装所有必要的硬件模块，包括 PWM、转速计输入捕捉定时器、控制寄存器、状态寄存器和 DMA 通道或中断。有关详细信息，请参考风扇控制器应用页面所介绍的内容。

风扇控制应用位于 Bootloadable 工程中。另外，还可以定制 Bootloader，从而能在引导加载过程中保持风扇运行。

在互相切换 Bootloader 和 Bootloadable 之间的过程中，如果器件进行复位，风扇仍可以保持运行状态，如表 3 所示。

表 3. 器件复位时风扇控制器的 PSoC I/O 引脚驱动模式

I/O 引脚驱动模式	备注
高阻模拟	在 PWM 引脚上可以加上外部上拉或下拉电阻，使占空比达到 100%。但这样做不是很必要的，因为风扇借助于它的惯性可以保持转动。
NVL 设置：高阻模拟、上拉或下拉	可以将 PWM 引脚组件的复位值设置为上拉或下拉，使占空比达到 100%。但这样做不是很必要的，因为风扇借助于它的惯性可以保持转动。
PSoC Creator 工程配置	设置 PWM 引脚组件驱动模式和初始状态，从而得到 100%的占空比。 PWM 组件变为有效，但不运行。
开始执行 Main()	PWM_Start()被调用时，PWM 会以组件的默认占空比来开始驱动 PWM 引脚。 固件可以读取转速计数据，并主动控制占空比。

C 附录 C — 主机内核 API

cybtldr_api2.c / .h

这是级别更高的 API，用于处理所有的引导加载操作。其函数可以打开和关闭文件。它为引导加载操作调用 cybtldr_api.c / .h API 的函数。可以使用这些 API 来构建基于 GUI 的 Bootloader 主机。

cybtldr_api.c / .h

这是一个行级的 API 文件，用于一次性将单行数据发送到 Bootloader 目标。该 API 文件具有用于设置引导加载操作、擦除行、编程行、验证行和结束引导加载操作的函数。表 4 详细描述了该 API 文件的函数。

表 4.*cybtldr_api.c /.h* 的函数

函数	说明
CyBtldr_StartBootloadOperation	• 使能通信接口并将 Enter Bootloader 指令发送到目标。 • 根据接收到的响应数据包，可以验证芯片 ID、目标器件的芯片修订版以及 Bootloader 版本。
CyBtldr_ProgramRow	• 首先要验证某个行，即将 Get Flash Size（获取闪存大小）指令发送到目标，以获取目标闪存的特定阵列 ID。响应该指令时，目标将返回该阵列的 Bootloadable 闪存部分起始和结束的行编号。主机会读取该响应，并检查指定行是否位于闪存的 Bootloadable 区域内。 • 如果成功验证该行，主机会将行数据分为适当的大小，并使用 Send Data（发送数据）指令将它们发送到目标。 • Program Row（编程行）指令随行数据的最后部分发送到目标。
CyBtldr_VerifyRow	• 该函数先验证由特定阵列 ID 和行编号确定的数据行。 • 如果验证行成功，将为已验证的闪存行发送一个 Verify Row（验证行）指令。响应该指令时，目标将返回该行的校验和。 • 根据期望校验和的值验证所返回的校验和。
CyBtldr_EraseRow	• 该函数先验证由特定阵列 ID 和行编号确定的数据行。 • 如果验证行成功，为已验证的闪存行发送一个 Erase Row（擦除行）指令。
CyBtldr_EndBootloadOperation	发送 Exit Bootload（退出引导加载）指令，并禁用通信接口。

cybtldr_command.c / .h

该 API 用于处理发送到目标的指令数据包结构，并解析来自目标的响应数据包。cybtldr_api.c / .h 调用该 API 的函数。例如，为了发送 Enter Bootload（进入引导加载）指令，CyBtldr_StartBootloadOperation() 将调用该 API 的 CyBtldr_CreateEnterBootloadCmd()函数。同时它还包含用于发送到目标前计算指令数据包校验和的函数。

cybtldr_parse.c / .h

该模块用于解析*.cyacd* 文件，该文件包含要发送给器件的 Bootloadable 镜像。另外，该模块具有用于设置文件访问、读取头文件、读取行数据和关闭文件的函数。

cybtldr_utils.h

该头文件提供了 Bootloader 的版本信息以及状态和错误常量。

文档修订记录

文档标题：AN73503 — PSoC® USB HID Bootloader

文档编号：002-11095

版本	ECN	变更者	提交日期	变更说明
**	5169934	RZZH	03/11/2016	本文档版本号为 Rev**，译自英文版 001-73503 Rev*I。
*A	5814739	AESATMP8	07/13/2017	更新徽标和版权。

全球销售和设计支持

赛普拉斯公司拥有一个由办事处、解决方案中心、原厂代表和经销商组成的全球性网络。如欲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

赛普拉斯开发者社区

论坛 | WICED IoT 论坛| 项目 | 视频 | 博客 | 培训 | 组件

技术支持

cypress.com/support

此处引用的所有其他商标或注册商标归其各自所有者所有。

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

赛普拉斯半导体公司，2011-2017 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。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。