USBFS 引导加载程序数据表 BootLdrUSBFS V 1.30

Copyright © 2007-2011 Cypress Semiconductor Corporation. All Rights Reserved.

资源	PSoC® 模块			API 内存 （字节）		引脚 （根据外部 I/O）
	数字	模拟 CT	模拟 SC	闪存	RAM	
CY8C24x94、CY8CLED04、CY7C64215、CY8C20x66、CY8C20x36、CY8C20x46、CY8C20x96、CY7C643xx、CYONS2000、CYONS2110、CY8CTST120、CY8CTMG120、CY8CTMA120、CY8CTST200、CY8CTMG2xx						
HID 支持	0	0	0	4615-4982	46	2
不含 HID 支持	0	0	0	4516-4982	46	2

Note 期望当添加额外接口、HID 类和其他 USBFS 扩展时扩展闪存和 RAM。 如果引导加载程序处于实际操作中，它将使用大量 RAM 下载程序数据，但是在退出时释放 RAM。 由于引导加载程序的操作消除了应用程序操作，此 RAM 需求实际上是可以忽略的。 ROM/ 闪存使用包括完整 USB 接口。 用于引导加载程序函数的附加代码只比 USB 本身使用的大约 1.9K 字节代码的正常需求多 2K 字节。

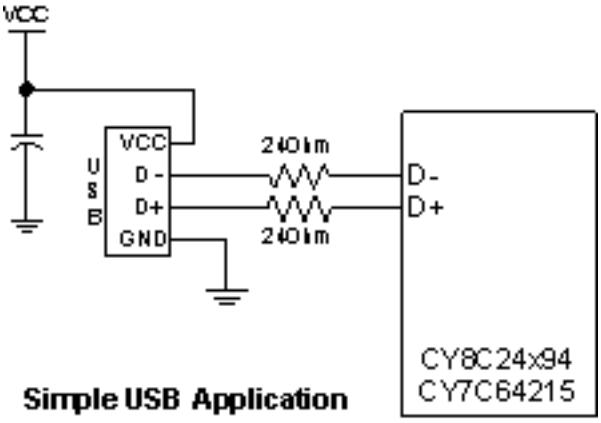
功能和概述

- 灵活存储器映射
- 不使用工程工具情况下的器件重新编程
- 产品常驻可重新编程性
- 集成了通信接口以最大程度地减少代码开销
- 固件升级的现场部署
- USB 全速器件接口驱动程序
- 支持中断和控制传输类型
- 设置向导便于轻松准确生成描述符
- 对描述符设置选择的运行时支持
- 可选 USB 字符串描述符
- 可选 USB HID 类支持

USB 引导加载程序支持全功能器件重新编程能力，内置错误检测和行业标准通信接口。

多个 USB 器件描述符在系统中共存，这样便可以命令运行器件进行自我重新配置和重新编程。 在重新配置期间维护核心 USB 函数，以便支持传输和存储程序数据时的主机通信。 在重新配置过程结束时，器件自行复位，验证新程序，并自动执行新程序。

Figure 1. USBFS 引导加载程序框图



快速启动

1. 查看此数据表。 要成功实施引导加载程序项目，需要对此信息有所了解。
2. 将用户模块添加到项目中。
3. 通过选择 HID 或非 HID 类应用程序，放置用户模块。
4. 右键单击用户模块图标。 选择 **引导加载程序工具**. 选择 **获取文件**. 当完成时，boot.tpl、custom.lkp、HTLinkOpts.lkp 和 flashsecurity.example 文件必须位于项目根目录中。
5. 右键单击用户模块图标。 选择 **器件： > 应用程序 USB 设置向导 …** 验证 **字符串** 区域中是否至少有一个字符串。 默认情况下必须至少提供一个字符串，否则请添加一个字符串。 选择 **“ 确定 ”**.

   Note 如果在步骤 3 中选择了 HID 类应用程序，则需要以下设置。 对于非 HID 应用程序，跳过以下设置并转到步骤 6。

   - 单击**导入 HID 报告模板**操作。
   - 选择 3 按钮鼠标模板。
   - 单击模板右侧的**应用**操作。
   - 编辑 HID 类描述符：为 HID 报告字段选择 3 按钮鼠标。
   - 单击**确定**保存 USB 描述符信息。
6. 右键单击用户模块图标。 选择 **器件 > 引导加载程序 USB 设置向导 …** 不需要进行任何修改。 选择 **“ 确定 ”**.
7. （仅限 Image Craft 编译器）在 **项目 > 设置 > 连接器** 对话框中，将 **可重定位代码开始地址** 设置为通过乘以 ApplicationCode_Start_Block X 器件模块大小而获得的值，以避免不经意地尝试在引导加载程序 ROM 区域中创建应用程序代码。 如果这些设置留下了未使用的 ROM 区域，可以稍后优化这些设置。 连接器在遇到存储器重叠错误时，会给出没有任何帮助的消息。 如果连接器问题降低到最低程度，则初始项目开发遇到的挫折会少一些。 对于大多数默认设置，将可重定位代码开始地址设置为大约 0x1700 是适用的： 举例而言，对于 CYONS2xxx （模块大小为 0x80 字节）： 0x2e*0x80 = 0x1700，而对于 CY8C24794 （模块大小为 0x40 字节）： 0x5c*0x40 = 0x1700。
8. 生成源代码并编译项目。
9. 查看输出文件 *<project>.mp* 以及 *<project>.hex* 查看构建项目的方式。
10. 在创建了编译准确无误的项目后，请转到 “ 固件代码示例 ” 部分。 修改并调整示例中提供的代码。

[+] Feedback

功能说明

USBFS 用户模块提供：

- 符合第 9 章的 USB 全速器件框架。
- 对 USB 主机发出的请求进行解码和调度的控制端点的低层驱动程序。
- 便于轻松构建描述符的 USBFS 设置向导。

您可以选择构建基于 HID 的器件或普通 USB 器件。 选择 USBFS 用户模块时进行您的选择。 要在初始选择后更改您的选择，请删除 USBFS 用户模块的现有实例，然后添加新实例。

部署含有引导加载程序的项目后，将不再对以红色突出显示的存储器位置进行重新编程。 可以通过运行引导加载程序来更改应用程序代码以及校验和。 程序空间的前两个模块除外，您可以将其他主要功能代码组移动到您确定的位置。

除了用户模块中调整的参数，还提供了两个其他重要功能。 可以通过右键单击器件管理器视图中的引导加载程序图标来访问一组内置工具。 另外，提供了主机应用程序 （包括源代码）以及有关如何在系统中设置和使用它以展现引导加载程序功能的指南。

有关 USB 的信息 （包括有关 USB 用法的规范、资源示例和论坛），请访问 www.usb.org。

操作原理

要创建引导加载程序项目，需要对 PSoC Designer 标准模型进行一些非标准的修改。 为了简化此过程，引导加载程序用户模块提供了自定义文件和专用工具来帮助您进行引导加载程序项目开发。 可以通过切换到器件编辑器视图，然后右键单击 USBFS 引导加载程序用户模块图标来访问这些专用工具。 除了作为用户模块一部分提供的工具和文件，还提供了带有可演示引导加载程序基本功能的用户模块安装的主机应用程序示例。 此 Microsoft Visual Studio® 2005 的基于 PC 的应用程序和源代码包含在 PSoC Programmer 3 安装目录中的一个 .zip 文件中。

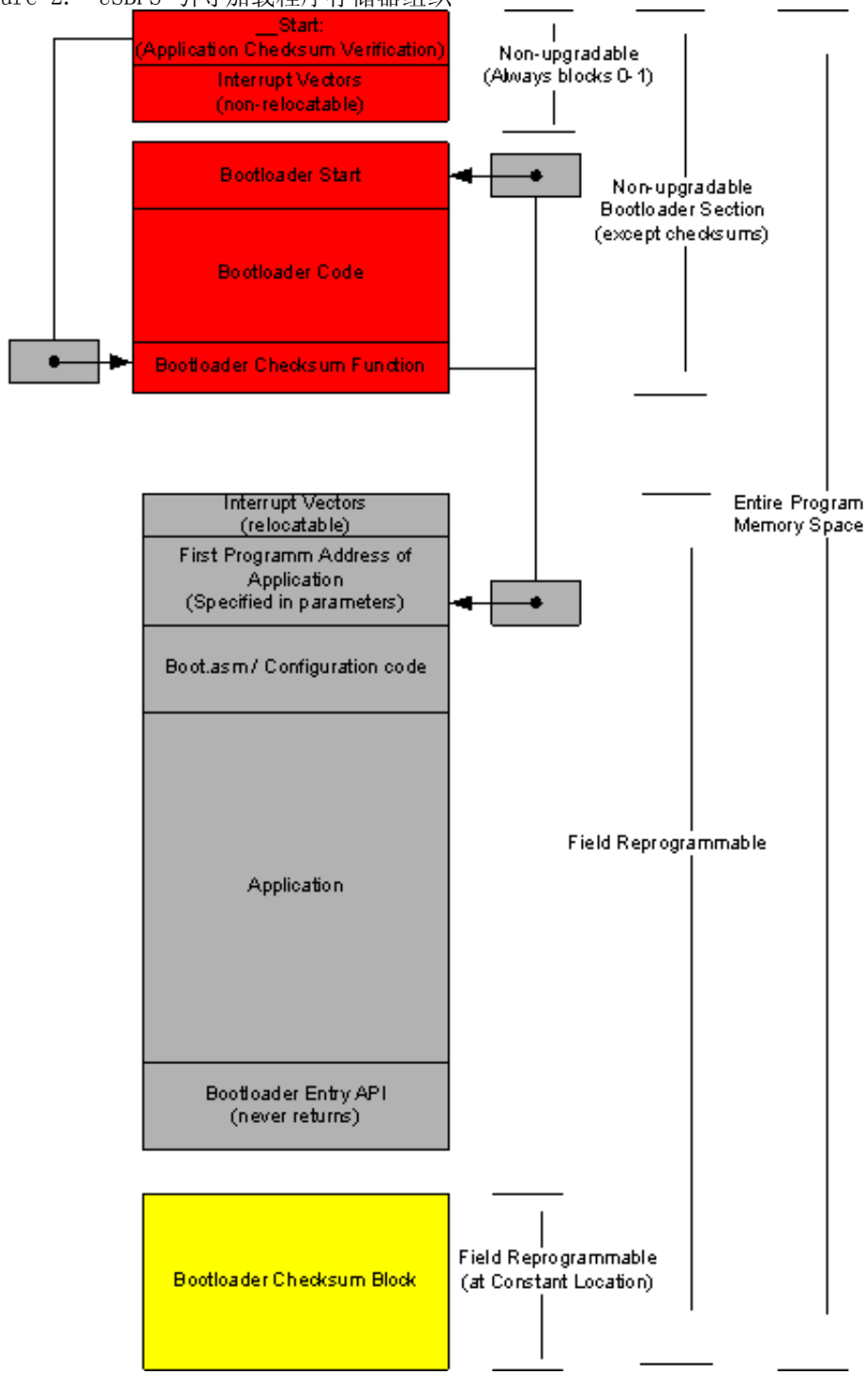
<install_path>\Cypress\Programmer\3.xx\Bootloaders\BootLdrUSBFS\USB_BootLoaderHostApp\…

使用此应用程序要求安装和有限自定义一个普通 USB 驱动程序，该驱动程序应当支持主机演示应用程序。 此文件作为安装的一部分提供，可以在初始操作引导加载程序器件时注册此文件。 使用窗口手动安装方法可以指定前面所指定位置的 “\USB driver” 目录中包含的驱动程序的位置。

[+] Feedback

必须修改包括的 driver.inf 文件才能正确指定所选引导加载程序器件的 VID 和 PID。 请注意，必须在 driver.inf 文件中的两个位置进行此更改：一个位置靠近文件顶部，另一个位置靠近底部。

Figure 2. USBFS 引导加载程序存储器组织



[+] Feedback

USBFS 引导加载程序存储器组织

PSoC Designer™ 使用标准化文件、有关部件系列的内置数据和特定器件的属性来创建可编译项目和更正 API 定义。 具有引导加载程序的项目需要的存储器映射与标准 PSoC Designer 项目的存储器映射有很大区别。 存储器区域的选择代表核心的设计决策，在设计的整个生命周期中都坚持这一决策。 不需要引导加载程序的项目仅允许编译器和连接器分配 RAM 和 ROM。 但是，引导加载程序必须将 RAM 和 ROM 分组在特定区域，以便该程序在加载新应用程序时不会崩溃。

在如下图所示的存储器布局 五图 2 中个重要受管理 ROM 区域：

- 第一个区域是 ROM 的模块 0 和 1。 这些模块包含关键的中断矢量和重启矢量。 由于几乎不可能通过任何操作器件来控制对这些模块的读取访问，因此无法将这些模块擦除和重新编程。 不能修改 ROM 的前两个模块，且不能将它们放置在任何其他位置。
- 要定义的第二个存储器区域是包含引导加载程序代码自身的区域。 该区域可以放置在不同的 ROM 位置。 但是，在部署包含引导加载程序的项目或器件后，该区域不可重新编程，且不能进行现场升级。
- 第三个存储器区域是可重定位中断表。 根据器件结构的不同，此表可能由一个或两个模块组成。 这一区域包含中断矢量以及通用矢量，为中断或使用引导加载程序加载新应用程序时可能更改的代码条目提供跳转表。 例如，这一区域包含应用程序起始地址。 在加电时验证校验和后，引导加载程序可以使用此地址启动新应用程序。 您可以在设计时确定此区域的位置。 部署应用程序和引导加载程序后，可以重新写入此区域，但是不能修改其位置。 此区域的特性类似于本节稍后描述的校验和区域。
- 第四个存储器区域是应用程序区域。 应用程序区域包含一组中断矢量，可以对这些矢量重新编程，但前提是当重新写入它们时不再访问它们。 请考虑若在执行代码期间更改代码，程序可能遇到的问题。 通过在引导加载期间关闭所有应用程序中断，可轻松满足此要求。 当引导加载程序启动时，会自动完成此操作。 除了中断矢量，应用程序区域还包含大多数器件引导代码和所有前台运行时应用程序。
- 第五个定义的 ROM 为校验和区域。 此区域包含重要数据，引导加载程序软件使用这些数据下载和验证前台应用程序。 校验和区域包含起始地址以及以模块表示的前台应用程序的大小。 校验和模块的前两个字节是校验和模块自身的校验和，后两个字节是运行时应用程序的校验和。 除了引导加载程序使用的空间，校验和模块的结构还包含供您自定义数据的空间。 此结构以 C 结构定义的形式公开，只要引导加载程序实用程序使用的数据未更改或重新放置在模块中，就可以修改此结构。

如果应用程序具有一些必须始终可运行 （包括在引导加载过程中）的代码，引导加载程序用户模块的设计可提供充分的定制来满足这一需求。 这可以通过使用汇编程序 AREA 指令将代码添加到引导加载程序 ROM 区域中来完成。 引导加载过程中代码所使用的任何 RAM 必须添加到为引导加载程序定义的 RAM 区域中。

用户模块参数中存储器区域的定义

通过 USBFS 引导加载程序用户模块参数，您可以自定义主程序元素在 ROM 中的放置位置。 用户模块中的默认值提供了有效初始设置。 用户应该使用这些设置，直到完整的项目成功编译完成。 编译项目后，可以查看程序存储器映射和 .hex 输出文件，以确定如何优化程序结构。 如果用户对参数重新进行配置并意外造成存储器区域冲突，那么在没有有效存储器映射可供查看的情况下很难确定正确的位置。

引导加载实用程序

引导加载程序用户模块提供了与前台应用程序共存的完整实用程序。 当器件启动或复位时，会始终调用引导加载实用程序。 一旦在系统启动时调用引导加载程序，引导加载程序通过计算前台应用程序 ROM 区域中的校验和来验证前台应用程序。 计算出的校验和与存储在校验和模块上的校验和 （随应用程序创建）进行比较。 如果两个校验和相等，则引导加载实用程序允许前台应用程序执行。 如果两个校验和不相等，则引导加载程序进入等待循环，等待主机应用程序下载有效应用程序。 它还使自己的 USB 子系统允许主机传输数据。 如果主机系统发现此接口处于启用状态，它可能会选择执行其自己的一组应用程序。 虽然提供了与给定示例一起成功运行的默认 USB 描述符，但是您可以选择在主机或 PSoC 器件上更改任何参数。 为主机应用程序包括了 VisualStudio 2005 源代码。 PSoC Programmer 3 安装目录中提供了应用程序和源代码示例。

<install_path>\Cypress\Programmer\3.xx\Bootloaders\BootLdrUSBFS\USB_BootLoaderHostApp\…

[+] Feedback

引导加载程序工具

快捷菜单中提供了一些工具，可以通过右键单击用户模块图标访问这些工具。

特殊版本的 boot.tpl、custom.lkp 和 HTLinkOpts.lkp 可以加入项目中或从项目中除去。 从主菜单中，选择 “ 工具 ” >“ 恢复默认启动文件 ”。 如果您删除了 USBFS 引导加载程序用户模块，则恢复默认启动文件的选项移动到 PSoC Designer 的 “ 文件 ” 菜单中。

生成校验和 - 一旦项目已经正确建立，您可以使用引导加载程序工具来创建和自动验证校验和。 在进入引导加载程序工具选择屏幕时，将生成项目代码，并执行对整个项目的完整编译。 然后在所生成的十六进制文件上执行校验和计算，得出的校验和与用户模块所存储的校验和进行比较。 如果校验和不匹配，则会显示一条消息。 如果用户愿意，可以重新计算并存储一个新的校验和。 如果在由引导加载程序工具所调用的自动化生成和构建过程中发生了构建或编译错误，而且没有成功地创建十六进制文件，则将报告错误，但不在 PSoC Designer 的构建对话框内显示错误调试信息。 在从自动化界面上调用生成和构建命令时，错误报告会被抑制。 为了调试构建错误，有必要使用处于引导加载程序工具菜单以外的传统构建和生成流程。

生成 dld 文件 - 此工具项将从十六进制项目输出文件中生成一个下载文件。 该文件只包含了由引导加载程序重新编程的十六进制模块，包括校验和模块。 “ 主机演示 ” 应用程序能够读取此文件，并将其下载至包含引导加载程序的正在运行中的项目之内。 此下载文件可以部署到现场应用程序中以升级 PSoC 器件。

校验和半自动生成

一旦项目能够构建起来并且在编译时没有发生错误时，就必须生成应用程序的校验和。 在器件编辑器视图中右键单击 “ 引导加载程序用户模块 ” 图标并选择**引导加载程序工具**，即可选用其中一项实用程序来创建应用程序校验和。 应用程序校验和 （之前计算的值或默认值）作为隐藏的用户模块参数存储起来。 一旦 “ 引导加载程序工具 ” 菜单页被调用，将用以前的校验和来验证根据当前 output\<prj_name>.hex 文件所计算出来的校验和。 在成功编程以前，必定无法生成校验和。 一旦校验和已经创建，则必须将校验和集成到编译文件中。 这样就要求进行第二次编译。 最后，为了验证存储校验和的代码模块，提供了该模块自身的校验和。 这需要第三次编译。 需要多次编译，这是因为内置校验和参数是只能进行后编译计算的预编译元素。

提供的特殊文件

通过打开**引导加载程序工具**菜单并选择**获取文件**，可以访问一些重要文件。 特定于器件的 boot.tpl 文件与称为 custom.lkp (ImageCraft)、HTLinkOpts.lkp (Hi Tech) 的文件和预定义 flashsecurity.txt 文件一起放置在主项目目录中。 此处简要介绍了每个文件 （这些文件的原始版本放在项目备份目录中）：

Boot.tpl。 - 这个文件中包含了中断矢量表的不可重定位和可重定位定义以及器件具体的引导设置，此设置在 ROM 的一个可重定位区域内规定，而不是标准 boot.tpl 文件内规定的固定位置。

Custom.lkp - 在源代码生成发生后，custom.lkp 文件在依照用户模块参数所定义的自动生成的主要代码模块的 ROM 区域内放置。 不得修改 custom.lkp 文件内的代码模块，具体名称为：

- -bSSCParmBlk - 包含闪存运行时所使用的指定关键 RAM。
- -bBootloader
- -bBLChecksum
- -bUserAPP - 更改最后三行的任何一行会导致出现一个错误对话框，该对话框指出项目无法检测正确的 custom.lkp 文件。

在代码生成期间，custom.lkp 文件最后三行中的每一行均在代码生成软件的控制下进行重新编写。 在最后 3 行之内或之后所做的改动都会导致发生错误或完全被丢弃。 您可以更改 custom.lkp 文件的其余部分。 要调试项目的存储器分配，可以通过在第一个空格处插入分号，将所有上述三行注释掉。 这样将让连接器能够自动放置代码，并且可能有助于确定应用程序代码大小方面的要求。

[+] Feedback

HTLinkOpts.lkp – 在源代码生成发生后，HTLinkOpts.lkp 文件以依照用户模块参数所定义的自动生成的主要代码模块的 ROM 区域内放置。 不得修改 HTLinkOpts.lkp 文件内的代码模块。

- -L-ACODE... & -L-AROM... 行包含提供总体 ROM 大小的数据
- -L-PPD_startup... 包含用于定位引导加载程序具体 ROM 区域的连接器指令
- -L-P
- -L-Pbss0= 对最后几行中任意一行的改动将导致出现一个错误对话框，表明项目无法检测到正确的 HTLinkOpts.lkp 文件。

在代码生成期间，HTLinkOpts.lkp 文件最后几行在代码生成软件的控制下进行重新编写。 在最后 3 行之内或之后所做的改动都会导致发生错误或完全被丢弃。

Flashsecurity.example – 这是默认文件，此文件按照默认的用户模块参数所具体规定的默认存储器映射进行布置。 为了进行最终的项目创建，可能需要根据最终存储器映射和所部署器件和固件的应用程序大小手工修改此文件。 请注意，此文件并不直接由 PSoC Designer 使用。 如果出于某些原因，项目进行了更新或对过时文件做出了标记，则不覆盖 flashsecurity 文件。 您可以编辑和重命名给定的文件 flashsecurity.example。

Flashsecurity.txt – 这是一个由 PSoC Designer 所提供的默认文件。 此文件内的数据加入到 .hex 文件，并指令器件如何管理对于内部 ROM 存储器的访问。 如果存储器模块防止写访问，则引导加载程序不工作。 由读写保护内置到编程的 PSoC 中，在第一次部署引导加载程序之前，必须正确配置此文件。

USB 描述符

标准 USBFS 用户模块集成了一个工具，用来开发运行时应用程序中使用的 USB 描述符。 引导加载程序增加了一个附加工具，以允许开发或修改默认 USB_Bootloader 描述符。 这两个描述符存储在 ROM 的不同区域中。 前台应用程序的描述符可以随应用程序升级。 USB_Bootloader 描述符是引导加载程序 ROM 区域的一部分，不能在字段中更新。 为了维护核心功能，还在引导加载程序 ROM 区域中放置了关键 USB 代码。 这样可以解决正在执行的代码同时被重新编写的问题 （这绝对不是良好的编程习惯）。

自供电器件的 USB 符合性

在 *USB 符合性检查表*中，提出了一个问题，内容是 “ 器件的上拉电阻是否仅在 V_{BUS} 为高电平时有效 ?”。

该问题将*通用串行总线规范修订版 2.0* 中的第 7.1.5 部分作为参考。 该部分的节选内容为：“ 上拉电阻的电压源必须来自或由 USB 电缆的供电控制，以便当删除 V_{BUS} 时，上拉电阻不会在它连接到的数据线上提供电流。”

如果您创建的器件采用自供电，则必须通过电阻网络将 GPIO 引脚连接到 V_{BUS}，并写入固件以监控 GPIO 的状态。 应用笔记 AN15813 *监控 EZ-USB FX2LP VBUS* 介绍了必需的硬件和软件组件。 使用 USB IO 控制寄存器 1 (USBIO_CR1) 控制 D+ 线上的上拉电阻。

引导加载程序 VID 和 PID

为了完成 USB 器件的最终部署，必须分配供应商 ID 和产品 ID。 它们都是由 USB 标准化组织应 USB 开发人员的请求而分配的。 出于开发目的，可以使用不与主机上的现有 VID 和 PID 冲突的任何 VID 和 PID 来调试项目。 但是，出于项目发布或部署目的时，不能使用分配给赛普拉斯的 VID 和 PID。

参数进入模块

所有存储器参数均进入引导加载程序中编号为 0x00 到 0xFF 的模块（对于 16K 器件）或 0x00 到 0x1FF 的模块 （对于 32K 器件）。 虽然这并不是进入存储器地址最方便的格式，但这种格式防止了部分模块地址被错误地分配到存储器映射的不同节段。 有问题的 PSoC 器件只能存储 64 字节闪存模块 （对于 20x45，为 128 字节），这是正确维护不同项目代码节段之间的边界的简单方式。

[+] Feedback

主机应用程序调试

内置引导加载程序的应用程序可能较难调试。 因此，可能需要在引导加载程序用户模块文件内部进行额外调整。 这些调整包含在文件 BootLdrUSBFS_Bt_loader.inc 中。 其中一个部分包含了下列等效参数：

```
BOOT_TIMEOUT:           EQU     40    ;set to zero to make timeout infinite
CHECKSUM_ON_CKSUMBLK:   EQU      1    ;Apply a checksum to the checksum block
                                     ;(adds compile steps and code to verify)
```

BOOT_TIMEOUT 等效参数允许您增加、减少在用户命令调用引导加载程序后未从主机接收到任何通信情况下代码超时，或者使该代码超时无限大。 这点可能在开发或调试主机应用程序时有用。

第二个等式控制着校验和模块内校验和的使用。 如果将此等式设置为 0，则不对包含在校验和模块内的校验和执行任何验证。 校验和验证仍然会按照用户模块参数内的定义而对整个用户应用程序区域执行。

时序

USBFS 用户模块在 CY8C24x94、CY8CLED04、CY7C64215、CY8C20xx6、CY7C643xx、CYONS2000、CYONS2100 和 CYONS2110 器件上支持 USB 2.0 全速操作。

USBFS 设置向导

USBFS 引导加载程序用户模块不使用 PSoC Designer 参数网格显示进行个性化设置。 相反，它使用窗体驱动的 USBFS 设置向导来定义应用程序的 USB 描述符。 通过描述符，向导对用户模块进行个性化设置。

用户模块由 USBFS 设置向导生成的信息驱动。 此向导简化了 USB 描述符的构建，并将生成的信息集成到用于器件枚举的驱动程序固件中。 如果不先运行此向导，选择适当的属性并生成代码，则 USBFS 引导加载程序用户模块将无法正常工作。

使用模块而不是地址

大多数 PSoC 部件的模块大小为 64 字节。 现在介绍一些模块大小为 128 字节的 PSoC 器件。 两个重要事实是： 任何引导加载程序都需要写入闪存中，PSoC 只能逐 “ 模块 ” 写入闪存。 因此，对于引导加载程序应用，将存储器视为一组要写入的 “ 模块 ” 会更加有效。

为了从模块转换为绝对地址，需要做以下乘法： Abs_addr = block_number X 模块大小。Block_0 起始于地址 0，Block_n 起始于地址 n x Block_size.。在引导加载程序参数中，所有模块均以十六进制相互分隔，因此，乘以 0x40 （64-byte 模块）或 0x80 （128-byte 模块），即可获得十六进制的地址。

十六进制输出文件包含每行的绝对地址。 不论相应器件的模块大小如何 (0x40/0x80)，十六进制输出文件都会将代码分隔为每行 64(d)/0x40 字节。 因此，对于模块为 6 4byte 的器件，每行就代表一个代码模块。 对于 128 字节模块的器件，十六进制文件的两行代表一个模块 （由于模块 0 起始于地址 0，因此必须始终将 128 字节模块视为：一半为 “ 偶数 ” 部分，代表 （地址）下半部分；一半为 “ 奇数 ” 部分，代表 （地址）上半部分）。

请参见十六进制文件，熟悉您使用的部件的闪存模块大小。

[+] Feedback

常见问题

本节讨论创建引导加载程序项目时发生的问题，并给出如何解决这些问题的建议。

更新引导加载程序项目、服务包升级和编译器

使用引导加载程序应用时，应避免更改 PSoC 开发人员环境。 这包括未更新 PSoC Designer、引导加载程序用户模块和编译器。

要了解其原因，请记住引导加载程序和应用程序最初是一起编译的，但是在部署可引导加载的系统后，将仅对应用程序部分进行重新编程。 必须用相同版本的引导加载程序用户模块对新的或修订的应用程序进行编译，以便新应用程序与原始部署中的引导加载程序匹配。 理想情况下，开发环境中所有版本的元件都是兼容的。 但对于引导加载程序，则十分有必要保持兼容性。 如果不更改开发环境，则可以消除兼容性风险。

基于 USB 的引导加载程序将其 USB 子系统公开给 API 这样的应用程序。 这样，便可以减小代码大小。 此类函数的公开并非通过可重定向调用表来完成。 此策略意指应用程序对引导加载程序中的特定地址进行直接调用。 由于引导加载程序和应用程序是一同编译的，因此对引导加载程序的任何更改将导致对 USB API 函数地址的更改，从而导致与任何其他版本的引导加载程序不兼容。

尽管 PSoC Designer 支持多个编译器，但是，不能笼统地认为由一个编译器编译的引导加载程序能够与另一个编译器编译的应用程序兼容。 其中一个主要的差异是关于 RAM 分配的假定。 不同的编译器实现 RAM 分页的方式可能不一样。 另外，由于引导加载程序和应用程序是一同编译的，因此假如一对引导加载程序和应用程序所使用的开发工具不相符，则无法对它们进行调试。

看门狗定时器的内部使用

与看门狗定时器的协同连接至 globalparams.inc 文件中包含的全局参数 WATCHDOG_ENABLE。 如果您的项目使用看门狗定时器，它将有条件地编译连接到全局参数的代码，并在引导加载校验和以及下载操作的过程中自动设置看门狗。 CPU 时钟速度会影响看门狗定时器的更新速度。 看门狗定时器的实际最小值设置大约为 0.125 秒。

Flashsecurity.txt 中的错误设置

该文件的默认设置值是在项目创建之时设置的。 “Flashsecurity.example” 文件中提供了一个配置范例。 Flashsecurity.example 通过 “ 引导加载程序工具 ”-“ 获取文件 ” 用户模块菜单项来提供。 该映射必须允许在将要真正执行引导加载的所有位置进行闪存写操作。 其中一种策略是让所有模块均可以写入。 另一种策略是在当前花一些时间布置存储器映射，并相应地对本文件进行编辑。 无论选择何种策略，在项目开始就采取措施始终比后期进行调试的见效要快。 用户有责任对引导加载程序可执行代码所占据的代码区提供写保护。 如果未能正确地对闪存安全性进行映射，这种情况就会变成系统破坏的因素之一，并导致形成极为困难的调试任务。

错误的可重定位代码起始地址 （仅适用于连接器参数 ImageCraft 编译器）

由于引导加载程序项目的存储器映射与传统项目的存储器映射有很大不同，因此通常需要更改可重定位代码起始地址。 这是连接器在试图将多个模块代码写入到同一地址时所发生的错误现象的常见原因。 可以在 “ 项目 ”>“ 设置 ”>“ 连接器 ” 选项卡中的 “ 可重定位代码起始地址 ” 字段中更改此参数。 计算绝对十六进制起始地址时要让此地址比引导加载程序代码所使用的最高模块的地址稍大一点儿，或者让其地址占用一个未使用的 ROM 区域。 对于 USB 版本的引导加载程序，将此值设置为通过乘以下列参数而得到的值：

ApplicationCode_Start_Block X 模块大小 = 可重定位代码起始地址。

如有必要，可以选择使此值大于所述的计算结果。 请记住，某些器件的模块大小为 0x80 字节，而某些器件的模块大小为 0x40 字节。

[+] Feedback

存储器重叠

为了纠正可重定位代码起始地址 （参见上一节），可使用前加的分号将 custom.lkp 文件的最后三行以加注释的方式除去，并尝试再次构建这个文件，然后检查所生成的存储器映射。 存储器重叠问题较难诊断出来，因为这种问题会导致输出文件无法生成。 通过修改 custom.lkp 文件可以让连接器放置目标模块，而目标模块随后提供了一个用于纠正存储器重叠根本原因的起点。

电源稳定性

电源噪声、短时脉冲、电压降低、缓慢的功率斜坡以及不良的连接都可能造成闪存编程中出现难以诊断的问题。 相对于功率斜坡来说，程序执行是十分快速的，在某些情况下，在闪存编程正在进行时，可能仍有某个部件存在着功率电平正在变化的情况。 举一个例子，一些状态在加电时写入闪存。 评估您的使用模型以及在闪存操作期间电源状况发生变化的可能性。 电源稳定性不佳可能造成某些部件不能正常运行并有可能造成闪存数据保持不良。

引导加载过程中应用程序或中断未完全停止

必须先终止将要被新引导加载的应用程序替代的应用程序，然后才能执行引导加载操作。 将应用程序中断关闭尤其重要。 当引导加载过程发生时，中断矢量地址在重新写入其新地址之前更改为零。 如果未禁用，运行中断将导致随机复位 （通过将矢量复位为 0）。 请注意，这并不适用于引导加载程序所使用的特定通信中断。

有两个 USB 接口：一个由应用程序使用，另一个由引导加载程序使用。 应当实现一个明确关闭 USB 应用程序接口并启用引导加载程序接口的方法，其中也包括完全关闭正在运行的应用程序。 本数据表稍后提供的代码示例包含了此过程的示例。

下载新文件导致器件停止运行

构建应用程序也可以不利用进入引导加载程序实用工具的方便性。 很容易不经意地这样做。 例如，带有简单 while(1); 循环的 main() 函数从不返回和进入引导加载程序。 因此，一旦该程序开始执行 （只要其校验和正确），则无法对其进行重新编程。 有许多策略可以解决此问题，但是此用户模块中不包括默认方法。 以下是一些建议：

1. 采用一个复位条件，让器件第一次加电启动时在引导加载程序被启用时能够留出一段时间。 通过设置超时参数，可以将器件配置为在复位时进入引导加载程序，在超时过期时退出前台应用程序。
2. 在代码中的某些点设置能够让器件进入引导加载程序的测试。 这可以是开关闭合或保持某个端口引脚处于低 / 高电平。
3. 使用内置 USB 功能 （例如功能报告或空闲端点），定义可发送到器件的 USB 通信以指示它进入引导加载模式。 当发送此命令时，器件暂时关闭 USB 总线，当命令返回时，它应当枚举为引导加载程序。
4. 如果器件未正常得到服务，则使用看门狗定时器将器件复位。 这可以与上述策略之一合并，以允许 WDT 中断启动可引导加载状态。 在从看门狗定时器进行复位时，可以对看门狗定时器相关联的状态位进行监测，以检测看门狗定时器是否是出现复位条件的原因。 有关其他信息，请参见技术参考手册。
5. 开发了两个项目，而且每个项目的引导加载程序均在一些细微之处有所不同。 请记住 “ 正在引导加载 ” 指的是正在对器件的一部分进行编程。 这意味着这两个相互重编程应用程序中每一个的引导加载程序的实现均必须完全一致。 所有引导加载程序参数和可重定位代码起始地址应当相同 （这与第一个应用程序模块不同）。 对这个问题的调试策略包含对相应的 2 个 16 进制文件进行对比，并特别关注引导加载程序所使用的 16 进制代码区域。 另一种方法是对 <project>.lst 文件进行比较。 引导加载程序利用了一些重定向矢量以允许某些应用程序地址参数发生改变。 所有这些跳转矢量必须与应用程序和引导加载程序相匹配。 在引导加载程序部署到现场应用程序后，其中的代码将无法更改。 以后的应用程序必须仍然 “ 同意 ” 相互使用的跳转矢量所存储的位置。

[+] Feedback

参数和资源

USBFS 引导加载程序包含一个与全功能 USBFS 用户模块集成的引导加载程序实用工具。

通过为引导加载程序定义的参数，可以定义当编译和连接程序时主程序区域的位置。

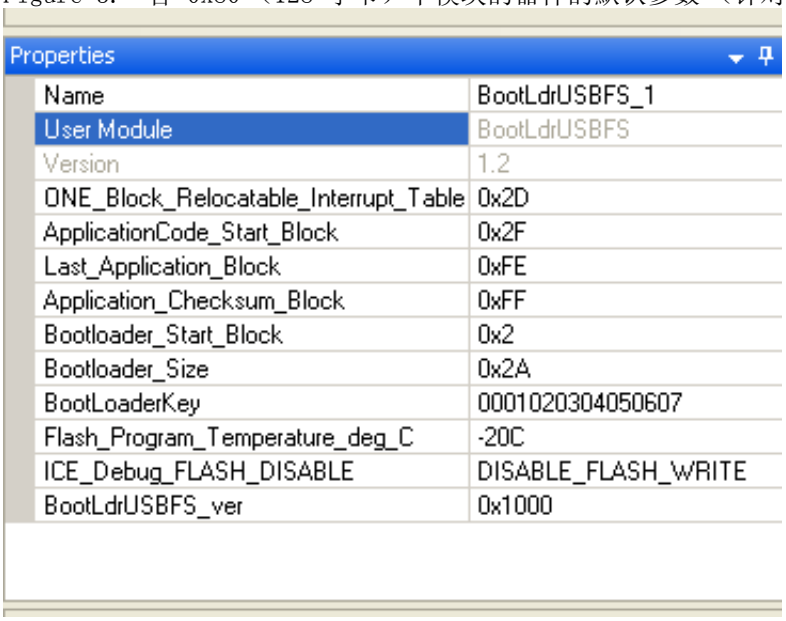
重命名用户模块

重命名用户模块可能需要对部件进行特定操作，这是因为此用户模块需要向导来填充和 / 或覆盖源文件。 在任何情况下，都不能更新向导所生成文件中的向导所生成变量名称。 您必须打开每个向导，选择 “ 确定 ” 或 “ 应用 ” 强制重新生成内部变量名称。 如果由于变量名称未更新而导致仍发生编译错误，请从项目中删除有问题的文件，重新打开向导，选择 “ 确定 ” 或 “ 应用 ”，然后重新构建项目。 将用更正的变量名称来替换文件。

默认参数

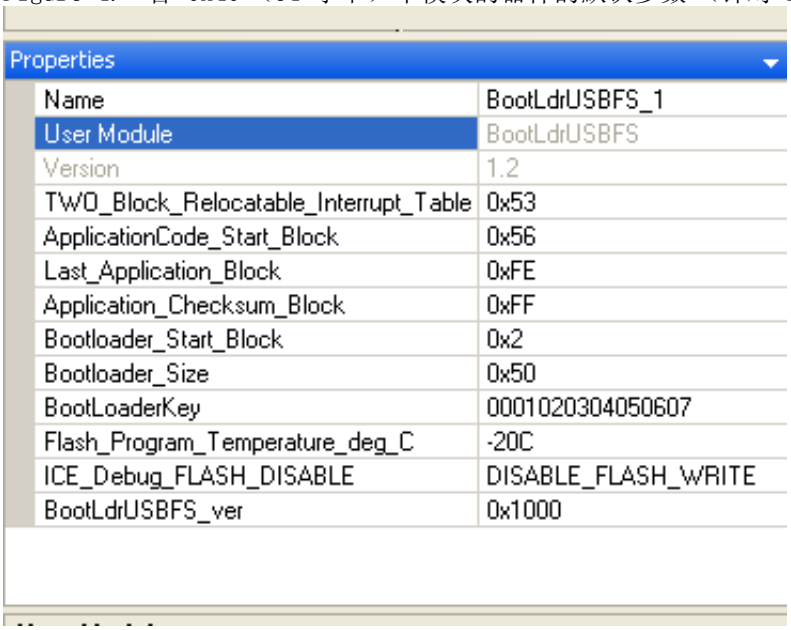
默认参数仅供参考。 项目中的默认值可以根据所使用部分的模块大小进行定制，或者可以进行调整以提供足够大小的代码区域。 项目经过编译和测试后，开发人员可以选择调整模块大小以优化存储器的使用。

Figure 3. 含 0x80 （128 字节）个模块的器件的默认参数 （针对 HID 选定选项）



[+] Feedback

Figure 4. 含 0x40 （64 字节）个模块的器件的默认参数 （针对 HID 选定选项）



TWO_Block Relocatable_Interrupt_Table

ONE_Block Relocatable_Interrupt_Table

大多数 PSoC 部件的模块大小为 64 字节。 在这些部件上，ROM 区域的两个模块用于定义一组中断表，这些表与始终在 PSoC ROM 模块 0 和 1 中提供的表相匹配。 某些 PSoC 器件具有 128 字节模块，只需要一个 ROM 模块来保存表。 由于此处重定向了模块 0 和 1 中断表，因此这些表必须一直位于存储器中的同一点。 这些可重定位表指向应用程序的可重定位中断矢量。

引导加载程序工具也使用此项参数来确定哪些代码模块用于 .dld 文件的处理，以及哪些代码模块用于校验和的计算。 此变量将传送至校验和模块，用于引导加载程序实用工具对应用程序校验和的自动验证操作。

ApplicationCode_Start_Block

这是分配给用户应用程序的第一个代码模块。 此代码是可引导加载的。 引导加载程序工具也使用此项参数来确定哪些代码模块用于 .dld 文件的处理，以及哪些代码模块用于校验和的计算。 此变量将传送至校验和模块，用于引导加载程序实用工具对应用程序校验和的自动验证操作。

Last_Application_Block

这是分配给用户应用程序的最后一个代码模块。 引导加载程序工具还使用此参数确定要为 <prj_name>.dld 文件处理哪个代码模块以及要在哪个代码模块上计算校验和。 此变量 （在转换为绝对地址后）将传送至校验和模块，用于引导加载程序实用工具对应用程序校验和的自动验证操作。 在完成该应用程序后，可以降低此值以节省校验和操作期间的处理时间并避免引导加载 ROM 的空模块。 如果升级的应用程序增加，可以增大此值，以使较大代码空间要求增加到可用 ROM 空间限制。

Application_Checksum_Block

这是校验和存储模块的位置。 如果应用程序更改但是不能在未重新部署引导加载程序的情况下移动其位置，则可以重新编写此模块应用程序。

此模块不能放在应用程序区域中，因为这样做需要在校验和模块 “ 周围 ” 连接应用程序，这超出了当前连接器的能力。 为此，模块必须放在远离应用程序的非 “ 障碍 ” 位置。 可能的位置有：引导加载程序与应用程序之间的单一模块，或者位于 ROM 的最后几个模块之一。

[+] Feedback

Bootloader_Start_Block

这是引导加载程序实用工具的第一个模块。 正常情况下，您不必更改此位置。 但是在某些情况下，可能需要移动此位置才能维护与当前实现的自定义引导加载程序的兼容性。

Bootloader_Size

这是引导加载程序应用的模块大小。 通常，此大小不需要调整。 如果您要向引导加载程序中添加代码，可以增加此大小以容纳额外代码。 其他模块也必须进行类似调整。

BootLoaderKey

这是加在发送给引导加载程序应用的数据操作前面的密钥值。 该密钥代表着一个额外的验证步骤，以确保引导加载程序升级实用工具不会意外启用。

默认值为 “0001020304050607”。

Flash_Program_Temperature_deg_C

此参数是器件重新编程时所期望的典型编程温度。 在本参数规定以外的温度对器件进行编程时，有可能对程序的保持效果造成不利的影响。

在引导加载过程中将程序温度参数与实际温度保持一致能够影响到存储器的保持性能以及写入操作次数的最大数量。 PSoC 在较低温度下可实现更为强大的闪存写入操作。 在远低于此参数设置值的温度下进行引导加载可能会导致存储器的保持性能下降。 因此，请注意确保引导加载程序从不在超过此参数值 20C 以上的温度下运行。 有关详细信息，请参见赛普拉斯器件规格。

ICE_Debug_Flash_DISABLE

此参数用于纠正在启用并运行 USB 资源的情况下执行 SSC 时 ICE 的不正常调试行为。 当调用 SSC 操作 （且是在闪存写入期间操作）时，禁用 USB SIE。 禁用闪存写入会允许在未向闪存实际写入代码的情况下对应用程序进行完整测试。

默认值为 “ 闪存写入禁用 ”。

BootLdrUSBFS_ver

此参数是引导加载程序的版本。 内部固件目前不使用此参数，但此参数可作为校验和模块的一部分提供。 您可以设置此值，使用此值验证引导加载程序可执行代码的版本是否正确。

Bootload_when_CKSUM_fails

通常，如果在复位时应用程序校验和不正确，则引导加载程序将激活，并等待引导加载新应用程序。 在使用 ICE 或调试器进行代码开发时，在每次编译后采用额外步骤来更正校验和是非常不方便的。 此功能允许您在不考虑引导加载程序操作的情况下运行和调试应用程序。 应当为应用程序现场部署而重新启用校验和功能。

[+] Feedback

应用程序编程接口

本节讨论引导加载程序的 API 和引导加载程序中包含的 USB 功能。 引导加载程序包含很有限的一组 API，这是因为引导加载程序的主要目的是完全删除和替换用户应用程序。

每次放置用户模块时，都会为其分配一个实例名称。 默认情况下，PSoC Designer 将 BootLdrUSBFS_1 分配到给定项目中此用户模块的第一个实例。 可将该值更改为符合标识符语法规则的任意唯一值。 分配的实例名称成为每个全局函数名称、变量和常量符号的前缀。 为简便起见，在以下说明中将实例名称缩写为 BootLdrUSBFS。 某些 API 的前面没有实例名称。 这些子程式 （例如 *ENTER_BOOTLOADER*）是不变的。

引导加载程序 API

ENTER_BOOTLOADER()

GenericBootloaderEntry

说明：

进入引导加载程序应用，并在没有任何引导加载程序主机开始与器件通信而导致超时 （如果定义了超时）后返回。 定义了一个泛型参数，在已部署部件的整个生命周期中，该参数都会驻留在一个固定的地址。 还可以通过直接调用此函数的已知十六进制地址来实现此函数。

此函数对 GenericBootloaderEntry 执行 ljmp，并驻留在 0x7C。

C 原型：

```
void ENTER_BOOTLOADER(void)
```

汇编：

```
lcall  ENTER_BOOTLOADER ; Call the Start Function
```

另外：

```
GenericHardDefinition: equ (0x7C)
lcall  GenericHardDefinition            ; Call the Start Function
```

参数：

无

GenericApplicationStart

说明：

在 boot.asm 开始处进入应用程序。 与热启动类似。

C 原型：

```
void GenericApplicationStart(void)
```

汇编：

```
lcall  GenericApplicationStart            ; Call the Start Function
```

参数：

无

[+] Feedback

bootLoaderVerify

说明：

执行应用程序存储区域的校验和验证。 如果校验和验证失败，则进入引导加载程序，此函数永不返回。 否则，将从 boot.asm 开始执行前台应用程序。

C 原型：

```
void bootLoaderVerify(void)
```

汇编：

```
lcall bootLoaderVerify              ; Call the Start Function
```

参数：

无。

副作用：

A 和 X 寄存器可以由此函数的本次执行或以后执行而被修改。 当前，仅修改了 IDX_PP 和 CUR_PP 页指针寄存器。

USBFS API

本节中的应用程序编程接口 (API) 子程式允许对 USBFS 用户模块进行编程控制。 以下各节将介绍描述符生成和集成，并列出基本 API 函数和特定于器件的 API 函数。 要开发应用程序，您需要对 USB 协议有基本了解，并熟悉 USB 2.0 规范 （尤其是第 9 章规范 “USB 器件框架 ”）。

USBFS 用户模块支持控制传输、中断传输、批量传输和同步传输。 某些函数或函数组（例如 LoadInEP 和 EnableOutEP）设计为可用于批量和中断端点。 其他函数 （例如 BootLdrUSBFS_LoadINISOCEP）设计为可用于同步端点。 有关如何执行这些传输类型的更多信息，请参考技术参考手册 (TRM)。

Note

1. ** 在所有用户模块 API 中，调用 API 函数可能会更改 A 和 X 寄存器的值。 如果在调用后需要 A 和 X 的值，则调用函数负责在调用前保留 A 和 X 的值。 选择此 “ 寄存器易失 ” 策略是为了提高效率。 C 编译器自动遵循此要求。 汇编语言程序员也必须确保其代码遵守这一策略。 虽然一些用户模块 API 函数可以保留 A 和 X 不变，但是无法保证它们将来也会如此。
2. USB 用户模块的 API 子程式是不可重新进入的。 由于它们取决于 RAM 中的内部全局变量，因此随此用户模块提供的 API 不支持从中断执行这些例程。 如果这是设计需要，请与赛普拉斯现场应用工程师联系。

功能	说明
void BootLdrUSBFS_Start(BYTE bDevice, BYTE bMode)	激活用于器件和特定电压模式的用户模块。
void BootLdrUSBFS_Stop(void)	禁用用户模块。
BYTE BootLdrUSBFS_bCheckActivity(void)	检查并清除 USB 总线活动标志。 如果自从上次检查后 USB 处于活动状态，则返回 1；否则返回 0。
void BootLdrUSBFS_SetPowerStatus(BYTE bPowerStatus)	将器件设置为自供电或总线供电
BYTE BootLdrUSBFS_bGetConfiguration(void)	返回当前分配的配置。 如果未配置器件，则返回 0。

[+] Feedback

功能	说明
BYTE BootLdrUSBFS_bGetEPState(BYTE bEPNumber)	返回指定 USBFS 端点的当前状态。 2 = NO_EVENT_ALLOWED 1 = 事件处于待处理中 0 = NO_EVENT_PENDING
BYTE BootLdrUSBFS_bGetEPAckState(BYTE bEPNumber)	通过返回非零值，识别是否已设置了 ACK。
BYTE BootLdrUSBFS_wGetEPCount(BYTE bEPNumber)	从指定 USBFS 端点返回当前字节数。
void BootLdrUSBFS_LoadInEP(BYTE bEPNumber, BYTE *pData, WORD wLength, BYTE bToggle)	加载并启用用于 IN 传输的指定 USBFS 端点。
void USB_LoadInISOCEP(BYTE bEPNumber, BYTE *pData, WORD wLength, BYTE bToggle)	
BYTE BootLdrUSBFS_bReadOutEP(BYTE bEPNumber, BYTE *pData, WORD wLength)	从端点 RAM 读取指定数量的字节，并将其放入 pSrc 指向的 RAM 阵列。 该函数返回主机发送的字节数。
void USB_EnableOutEP(BYTE bEPNumber)	使指定的 USB 端点能够接受 OUT 传输。
void USB_EnableOutISOCEP(BYTE bEPNumber)	
void BootLdrUSBFS_DisableOutEP(BYTE bEPNumber)	禁用指定的 USB 端点到 NAK OUT 的传输。
BootLdrUSBFS_Force(BYTE bState)	在 USB D+/D- 引脚上强制 J、K 或 SE0 状态。 通常用于远程唤醒。 bState 参数为： USB_FORCE_SE0 0xC0 USB_FORCE_J 0xA0 USB_FORCE_K 0x80 USB_FORCE_NONE 0x00 **注意**。 当从端口 1 (P1.2-P1.7) 使用此 API 函数和 GPIO 引脚时，应用程序使用 Port_1_Data_SHADE 影子寄存器确保一致数据处理。 在汇编语言中，可以直接访问 Port_1_Data_SHADE RAM 位置。 在 C 语言中，包括外部引用： extern BYTE Port_1_Data_SHADE;

功能	说明
BYTE BootLdrUSBFS_UpdateHIDTimer(BYTE bInterface)	更新指定接口的 HID 报告定时器，如果定时器到时，则返回 1，否则返回 0。 如果定时器到时，它将重新加载定时器。
BYTE BootLdrUSBFS_bGetProtocol(BYTE bInterface)	返回指定接口的协议。

[+] Feedback

BootLdrUSBFS _Start （用户定义的应用器件）

BootLdrUSBFS _Start （引导加载程序器件）0xFF

说明：

执行 USBFS 用户模块的所有必需初始化。 可以使用此命令启动前台 USB 器件或特定于引导加载程序的 USB 器件。 在任何时候，只能有一个 USB 器件配置处于活动状态。 要启动引导加载程序器件，请将 bDevice 的值设置为 -1 (0xFF)。

C 原型：

```
void BootLdrUSBFS_Start(BYTE bDevice, BYTE bMode)
```

汇编：

```
mov   A, 0xFF                 ; The bootloader device descriptor
mov   A, 0                    ; Select application device descriptor
mov   X, USB_5V_OPERATION     ; Select the Voltage level
lcall  BootLdrUSBFS_Start      ; Call the Start Function
```

参数：

寄存器 A： 包含的器件号来自在 USBFS 设置向导中输入的所需器件描述符集。

寄存器 X： 包含芯片运行的工作电压。 这可确定是否已针对 5V 的运行电压启用电压调节器，或者是否针对 3.3V 的运行电压使用了通过模式。 下表列出了 C 语言和汇编语言中提供的符号名称及其相关值：

掩码	值	说明
USB_3V_OPERATION	0x02	禁用电压调节器和上拉的通过 vcc
USB_5V_OPERATION	0x03	启用电压调节器并针对上拉使用调节器

返回值：

无

副作用：

A 和 X 寄存器可以由此函数的本次执行或以后执行而被修改。 当前，仅修改了 IDX_PP 和 CUR_PP 页指针寄存器。

BootLdrUSBFS _Stop

说明：

执行 USBFS 用户模块所需的所有必要的关闭任务。

C 原型：

```
void  BootLdrUSBFS_Stop(void)
```

汇编：

```
lcall BootLdrUSBFS_Stop
```

参数：

无

[+] Feedback

返回值：

无

副作用：

A 和 X 寄存器可以由此函数的本次执行或以后执行而被修改。 当前，仅修改了 CUR_PP 页指针寄存器。

BootLdrUSBFS _bCheckActivity

说明：

检查 USBFS 总线活动。

C 原型：

```
BYTE BootLdrUSBFS_bCheckActivity(void)
```

汇编：

```
lcall BootLdrUSBFS_bCheckActivity
```

参数：

无

返回值：

如果 USB 自上次检查以来一直处于处于活动状态，则会在 A 中返回 1，否则会返回 0。

副作用：

A 和 X 寄存器可以由此函数的本次执行或以后执行而被修改。

BootLdrUSBFS _bGetConfiguration

说明：

获取 USB 器件的当前配置。

C 原型：

```
BYTE BootLdrUSBFS_bGetConfiguration(void)
```

汇编：

```
lcall BootLdrUSBFS_bGetConfiguration
```

参数：

无

返回值：

在 A 中返回当前分配的配置。如果未配置器件，则会返回 0。

副作用：

A 和 X 寄存器可以由此函数的本次执行或以后执行而被修改。 在大内存模式下，所有 RAM 页面指针寄存器都会出现这种状况。 在必要时，调用函数有责任将调用前后的数值保存在 fastcall16 函数内。 当前，仅修改了 CUR_PP 页指针寄存器。

[+] Feedback

BootLdrUSBFS _bGetEPState

说明：

获取指定端点的端点状态。 该端点状态从前台应用程序的角度描述端点状态。 该端点具有三种状态之一，其中两种状态意味着 IN 和 OUT 端点的不同情况。 下表列出了可能的状态及其对于 IN 和 OUT 端点的含义。

C 原型：

```
BYTE BootLdrUSBFS_bGetEPState(BYTE bEPNumber)
```

汇编：

```
MOV A, 1                            ; Select endpoint 1
lcall BootLdrUSBFS_bGetEPState
```

参数：

寄存器 A 包含端点号。

返回值：

返回指定 USBFS 端点的当前状态。 下表列出了 C 语言和汇编语言中提供的符号名称及其相关值。 每当您编写代码 （如用于处理所发送或接收数据的 ISR 代码）以更改端点的状态时，请使用这些常量。

状态	值	说明
NO_EVENT_PENDING	0x00	指示端点正在等待 SIE 操作。
EVENT_PENDING	0x01	指示端点正在等待 CPU 操作。
NO_EVENT_ALLOWED	0x02	指示端点已锁定，无法访问。
IN_BUFFER_FULL	0x00	已加载 IN 端点，且已将模式设置为 ACK IN。
IN_BUFFER_EMPTY	0x01	已发生 IN 数据操作，可加载更多数据。
OUT_BUFFER_EMPTY	0x00	OUT 端点已被设置为 ACK OUT，且正在等待数据。
OUT_BUFFER_FULL	0x01	已发生 OUT 数据操作，可读取数据。

副作用：

A 和 X 寄存器可以由此函数的本次执行或以后执行而被修改。 在大内存模式下，所有 RAM 页面指针寄存器都会出现这种状况。 在必要时，调用函数有责任将调用前后的数值保存在 fastcall16 函数内。 当前，仅修改了 IDX_PP 页指针寄存器。

[+] Feedback

BootLdrUSBFS _bGetEPAckState

说明：

通过在端点的控制寄存器中读取 ACK 位，确认此端点上是否已发生 ACK 数据操作。 此函数不清除 ACK 位。

C 原型：

```
BYTE BootLdrUSBFS_bGetEPState(BYTE bEPNumber)
```

汇编：

```
MOV A, 1                              ; Select endpoint 1
lcall BootLdrUSBFS_bGetEPState
```

参数：

寄存器 A 包含端点号。

返回值：

如果已发生确认数据操作，则此函数会返回非零值。 否则会返回零。

副作用：

A 和 X 寄存器可以由此函数的本次执行或以后执行而被修改。 在大内存模式下，所有 RAM 页面指针寄存器都会出现这种状况。 在必要时，调用函数有责任将调用前后的数值保存在 fastcall16 函数内。

BootLdrUSBFS _wGetEPCount

说明：

此函数会返回端点计数寄存器的值。 串行接口引擎 (SIE) 在计数中包含校验和数据的两个字节。 此函数会首先从计数中减去二，然后再返回值。 在对 USB_GetEPState 的调用返回 EVENT_PENDING. 之后，仅针对 OUT 端点调用此函数

C 原型：

```
WORD BootLdrUSBFS_wGetEPCount(BYTEbEPNumber)
```

汇编：

```
MOV A, 1                              ; Select endpoint 1
lcall BootLdrUSBFS_bGetEPCount
```

参数：

寄存器 A 包含端点号。

返回值：

从 A 和 X 中的指定 USBFS 端点中返回当前字节计数。

副作用：

A 和 X 寄存器可以由此函数的本次执行或以后执行而被修改。 在大内存模式下，所有 RAM 页面指针寄存器都会出现这种状况。 在必要时，调用函数有责任将调用前后的数值保存在 fastcall16 函数内。

[+] Feedback

BootLdrUSBFS _LoadInEP

BootLdrUSBFS _LoadInISOCEP

说明：

针对 IN 中断或批量传输（.._LoadInEP）及同步传输（..._LoadInISOCEP）加载和启用指定的 USB 端点。

C 原型：

```
void BootLdrUSBFS_LoadInEP(BYTE bEPNumber, BYTE * pData, WORD wLength, BYTE bToggle)
void BootLdrUSBFS_LoadInISOCEP(BYTE bEPNumber, BYTE * pData, WORD wLength, BYTE
bToggle)
```

汇编：

```
mov A, USBFS_TOGGLE
push A
mov A, 0
push A
mov A, 32
push A
mov A, >pData
push A
mov A, <pData
push A
mov A, 1
push A
lcall BootLdrUSBFS_LoadInEP
```

参数：

bEPNumber - 介于 1 和 4 之间的端点号。

pData - 指向数据数组的指针。从由 pData 指定的数据数组中加载端点的数据。

wLength - 要从由 IN 请求得到的数组中传输的字节数。有效值范围介于 0 和 256 之间。

bToggle - 用于指示在计数寄存器中设置数据切换位之前该数据切换位是否已切换的标志。对于 IN 数据操作，在成功执行每个数据传输之后切换数据位。这样可确保不会重复或丢失相同数据包。C 语言和汇编语言中提供了标志的符号名称：

掩码	值	说明
USB_NO_TOGGLE	0x00	数据切换没有变化
USB_TOGGLE	0x01	在传输之前切换数据位

返回值：

无

副作用：

A 和 X 寄存器可以由此函数的本次执行或以后执行而被修改。在大内存模式下，所有 RAM 页面指针寄存器都会出现这种状况。在必要时，调用函数有责任将调用前后的数值保存在 fastcall16 函数内。当前，仅修改了 IDX_PP 和 CUR_PP 页指针寄存器。

[+] Feedback

BootLdrUSBFS _bReadOutEP

说明：

将指定的字节数从端点 RAM 移至数据 RAM。 实际从端点 RAM 传输至数据 RAM 的字节数为主机发送的实际字节数和由 wCount 参数请求的字节数中较少的一方。

C 原型：

```
BYTE BootLdrUSBFS_bReadOutEP(BYTE bEPNumber, BYTE * pData, WORD wLength)
```

汇编：

```
mov A, 0
push A
mov A, 32
push A
mov A, >pData
push A
mov A, <pData
push A
mov A, 1
push A
lcall BootLdrUSBFS_bReadOutEP
```

参数：

bEPNumber - 介于 1 和 4 之间的端点号。

pData - 从此指针指定的数据数组中加载端点空间。

wLength - 要从数组中传输、然后因 IN 请求而发送的字节数。 有效值范围介于 0 和 256 之间。 如果所请求的主机发送的字节数较少，则该函数移动的数量少于该数量。

返回值：

返回由主机发送至 USB 器件的字节数。 该数量可能会多余或少于所请求的字节数。

副作用：

A 和 X 寄存器可以由此函数的本次执行或以后执行而被修改。 在大内存模式下，所有 RAM 页面指针寄存器都会出现这种状况。 在必要时，调用函数有责任将调用前后的数值保存在 fastcall16 函数内。 当前，仅修改了 IDX_PP 和 CUR_PP 页指针寄存器。

BootLdrUSBFS _EnableOutEP

USBFS_EnableOutISOCEP

说明：

针对 OUT 批量或中断传输 (..._EnableOutEP) 及同步传输 (..._EnableOutISOCEP) 启用指定的端点。 请勿针对 IN 端点调用这些函数。

C 原型：

```
void USBFS_EnableOutEP(BYTE bEPNumber)
void USBFS_EnableOutISOCEP(BYTE bEPNumber)
```

汇编：

```
MOV A, 1
lcall USBFS_EnableOutEP
```


[+] Feedback

参数：

寄存器 A 包含端点号。

返回值：

无

副作用：

A 和 X 寄存器可以由此函数的本次执行或以后执行而被修改。 当前，仅修改了 IDX_PP 页指针寄存器。

BootLdrUSBFS _DisableOutEP

说明：

禁用指定的 USBFS OUT 端点。 请勿针对 IN 端点调用此函数。

C 原型：

```
void BootLdrUSBFS_DisableEP(BYTE bEPNumber)
```

汇编：

```
MOV A, 1                            ; Select endpoint 1
lcall BootLdrUSBFS_DisableEP
```

参数：

寄存器 A 包含端点号。

返回值：

无

副作用：

A 和 X 寄存器可以由此函数的本次执行或以后执行而被修改。 在大内存模式下，所有 RAM 页面指针寄存器都会出现这种状况。 在必要时，调用函数有责任将调用前后的数值保存在 fastcall16 函数内。

BootLdrUSBFS _Force

说明：

将 USB J、K 或 SE0 状态强制到 D+/D- 线。 此函数给予 USB 器件应用程序必要的机制，以便其可执行 USB 远程唤醒功能。 有关详细信息，请参见 USB 2.0 规范，以获取暂停和恢复功能的详情。

C 原型：

```
void BootLdrUSBFS_Force(BYTE bState)
```

汇编：

```
mov A, BootLdrUSB_FORCE_K
lcall BootLdrUSBFS_Force
```

参数：

bState 字节用于指示要启用四个总线状态中的哪一个。 下表列出了 C 语言和汇编语言代码中提供的符号名称及其相关值：

[+] Feedback

状态	值	说明
USB_FORCE_SE0	0xC0	将单端 0 强制到 D+/D- 线
USB_FORCE_J	0xA0	将 J 状态强制到 D+/D- 线
USB_FORCE_K	0x80	将 K 状态强制到 D+/D- 线
USB_FORCE_NONE	0x00	将总线返回到 SIE 控制

返回值：

无

副作用：

A 和 X 寄存器可以由此函数的本次执行或以后执行而被修改。 在大内存模式下，所有 RAM 页面指针寄存器都会出现这种状况。 在必要时，调用函数有责任将调用前后的数值保存在 fastcall16 函数内。

BootLdrUSBFS _UpdateHIDTimer

说明：

更新 HID 报告空闲定时器并返回到期状态。 在定时器过期时，重新加载定时器。

C 原型：

```
BYTE BootLdrUSBFS_UpdateHIDTimer(BYTE bInterface)
```

汇编：

```
MOV A, 1                              ; Select interface 1
lcall BootLdrUSBFS_UpdateHIDTimer
```

参数：

寄存器 A 包含接口号码。

返回值：

HID 定时器的状态在 A 中返回。下表列出了 C 语言和汇编语言代码中提供的符号名称及其相关值：

状态	值	说明
USB_IDLE_TIMER_EXPIRED	0x01	定时器已过期。
USB_IDLE_TIMER_RUNNING	0x02	定时器正在运行。
USB_IDLE_TIMER_IDEFINITE	0x00	在数据或状态更改的情况下发送了报告时所返回的值。

副作用：

A 和 X 寄存器可以由此函数的本次执行或以后执行而被修改。

[+] Feedback

BootLdrUSBFS _bGetProtocol

说明：

返回选定接口的 HID 协议值。

C 原型：

```
BYTE BootLdrUSBFS_bGetProtocol(BYTE bInterface)
```

汇编：

```
MOV A, 1                              ; Select interface 1
lcall BootLdrUSBFS_bGetProtocol
```

参数：

bInterface 包含接口号码。

返回值：

寄存器 A 包含协议值。

副作用：

A 和 X 寄存器可以由此函数的本次执行或以后执行而被修改。 在大内存模式下，所有 RAM 页面指针寄存器都会出现这种状况。 在必要时，调用函数有责任将调用前后的数值保存在 fastcall16 函数内。

BootLdrUSBFS_SetPowerStatus

说明：

设置当前电源状态。 对于自供电，将电源状态设置为 1，而对于总线供电，则将电源状态设置为 0。器件将基于此值回复 USB GET_STATUS 请求。这允许器件针对 USB 第 9 章合规性正确地报告其状态。器件可能会随时将其电源从自供电更改为总线供电，并将其当前电源报告为器件状态的一部分。 在将您的器件从自供电更改为总线供电时 （或正相反）均调用此函数，并设置相应状态。

C 原型：

```
void BootLdrUSBFS_SetPowerStatus(BYTE bPowerStaus);
```

汇编：

```
MOV   A, 1                            ; Select self powered
lcall  BootLdrUSBFS_SetPowerStatus
```

参数：

bPowerStatus 包含所需的电源状态，1 代表自供电，0 代表总线供电。 下表列出了 C 语言和汇编语言代码中提供的符号名称及其相关值：

状态	值	说明
USB_DEVICE_STATUS_BUS_POWERED	0x00	将器件设置为总线供电。
USB_DEVICE_STATUS_SELF_POWERED	0x01	将器件设置为自供电。

返回值：

无

[+] Feedback

副作用：

A 和 X 寄存器可以由此函数的本次执行或以后执行而被修改。 在大内存模式下，所有 RAM 页面指针寄存器都会出现这种状况。 在必要时，调用函数有责任将调用前后的数值保存在 fastcall16 函数内。

固件源代码示例

对于 C 语言和汇编语言示例项目，均将引导加载程序用户模块参数保留为默认状态 （图 3-4）。

确保将正确的引脚配置为下拉驱动，以识别开关闭合，进入引导加载程序。 示例取决于您可以按的按钮。此按钮拉高 Port1_7，并导致程序作为引导加载程序重新枚举。 在某些器件上，设置可能为 “ 下拉驱动 ”，而在其他器件上，正确的设置可能为 “ 漏极开路低电平 ”。 您可能需要考虑用作此示例目标的测试硬件的精确配置，并查看相关技术参考手册，以正确配置此项目。

Port_1_4	P1[4]	StdCPU	High Z Analog	DisableInt
Port_1_5	P1[5]	StdCPU	High Z Analog	DisableInt
Port_1_6	P1[6]	StdCPU	High Z Analog	DisableInt
Port_1_7	P1[7]	StdCPU	Pull Down	DisableInt
Port_2_0	P2[0]	StdCPU	High Z Analog	DisableInt
Port_2_1	P2[1]	StdCPU	High Z Analog	DisableInt

在您尝试引导加载应用程序之前，确保您修改 *flashsecurity.txt* 文件，以使应用程序、校验和及可重定位中断矢量区域变得可写入。 下图所示的 flashsecurity.txt 文件示例正是一个示例。 某些器件看起来有细微差别，但是所有器件均遵循相同的基本模式。

```
; to the end of the line.

; 0 40 80 C0 100 140 180 1C0 200 240 280 2C0 300 340 380 3C0 (+) Base Address

  W  W  W  W  W  W  W  W  W  W  W  W  W  W  W  W ;    Base Address 0
  W  W  W  W  W  W  W  W  W  W  W  W  W  W  W  W ;    Base Address 400
  W  W  W  W  W  W  W  W  W  W  W  W  W  W  W  W ;    Base Address 800
  W  W  W  W  W  W  W  W  W  W  W  W  W  W  W  W ;    Base Address C00
  W  W  W  W  W  W  W  W  W  W  W  W  W  U  U  U ;    Base Address 1000
  U  U  U  U  U  U  U  U  U  U  U  U  U  U  U  U ;    Base Address 1400
  U  U  U  U  U  U  U  U  U  U  U  U  U  U  U  U ;    Base Address 1800
  U  U  U  U  U  U  U  U  U  U  U  U  U  U  U  U ;    Base Address 1C00
  U  U  U  U  U  U  U  U  U  U  U  U  U  U  U  U ;    Base Address 2000
  U  U  U  U  U  U  U  U  U  U  U  U  U  U  U  U ;    Base Address 2400
  U  U  U  U  U  U  U  U  U  U  U  U  U  U  U  U ;    Base Address 2800
  U  U  U  U  U  U  U  U  U  U  U  U  U  U  U  U ;    Base Address 2C00
  U  U  U  U  U  U  U  U  U  U  U  U  U  U  U  U ;    Base Address 3000
  U  U  U  U  U  U  U  U  U  U  U  U  U  U  U  U ;    Base Address 3400
  U  U  U  U  U  U  U  U  U  U  U  U  U  U  U  U ;    Base Address 3800
  U  U  U  U  U  U  U  U  U  U  U  U  U  U  U  U ;    Base Address 3C00
; End 16K parts
```

请注意，所包含的主机 PC 示例项目具有供货商 ID (VID) 04b4 和产品 ID (PID) E006。 它们为赛普拉斯所拥有的 ID，可用于本地调试，但不可发布以用于生产。

以下是用 C 语言编写的 USB 引导加载程序用户模块的实现。此采样代码是针对 CY7C64215 器件系列编写的。 此用户模块支持的其他器件系列在可用端口、引脚和驱动模式方面可能有所不同。 您可能必须定制此代码，以便其可在您的器件上正常工作。

[+] Feedback

```
//
//emulate a mouse that causes the cursor to move in a square
//

#include <m8c.h>        // part specific constants and macros
#include "PSoCAPI.h"    // PSoC API definitions for all User Modules

signed char bXInc = 0;  // X-Step Size
signed char bYInc = 0;  // Y-Step Size

#define USB_INIT     0   // Initialized state
#define USB_UNCONFIG 1   // Unconfigured state
#define USB_CONFIG   2   // Configured state

// Mouse movemet states
#define MOUSE_DOWN   0
#define MOUSE_LEFT   1
#define MOUSE_UP     2
#define MOUSE_RIGHT  3

#define POSMASK      0x03 // Mouse position state mask
#define BOX_SIZE     32   // Transfers per side of the box
#define bCursorStep  4    // Step size

BYTE bConfigState = 0;    // Configuration state
BYTE bDirState = 0;       // Mouse diretion state

BYTE abMouseData[3] = {0,0,0};  // Endpoint 1, mouse packet array
BYTE bButton;                   // Used for button
BYTE boxLoop = 0;               // Box loop counter

const char abDirection[4][6] = {"DOWN "};
extern const  USB_pAppChkSumBlk;
WORD blversion;
void main(void)
{
M8C_EnableGInt;                     //Enable Global Interrupts
USB_Start(0, USB_5V_OPERATION);     //Start USB Operation usgin device 0


PRT1DR = 0;

while(1) {                                  // Main loop
   if(PRT1DR & 0x80) {
   USB_Stop();
   while(PRT1DR & 0x80);
   USB_EnterBootloader();
   }
   switch(bConfigState) {                   // Check state
   case USB_INIT:                           // Initialize state
      bConfigState = USB_UNCONFIG;

      break;
```


[+] Feedback

```
   case USB_UNCONFIG:                          // Unconfigured state
      if(USB_bGetConfiguration() != 0) {       // Check if configuration set
         bConfigState = USB_CONFIG;

         USB_LoadInEP(1, abMouseData, 3, USB_NO_TOGGLE);  // Load a dummy mouse packet
      }
      break;

  case USB_CONFIG:                            // Configured state time to move the mouse
      if(USB_bGetEPAckState(1) != 0) {
         boxLoop++;
       if(boxLoop > BOX_SIZE) {               // Change mouse direction every 32 packets
            boxLoop = 0;
            bDirState++;    // Advance box state
            bDirState &= POSMASK;

         }

        switch(bDirState) {                     // Determine current direction state

         case MOUSE_DOWN:                          // Down
            bXInc = 0;
            bYInc = bCursorStep;
            //asm("nop");
            break;

         case MOUSE_LEFT:                          // Left
            bXInc = -bCursorStep;
            bYInc = 0;
            break;

         case MOUSE_UP:                            // up
            bXInc = 0;
            bYInc = -bCursorStep;
            break;

         case MOUSE_RIGHT:                         // Right
            bXInc = bCursorStep;
            bYInc = 0;
            break;

         }
         abMouseData[1] = bXInc;                    // Load the packet array
         abMouseData[2] = bYInc;
         abMouseData[0] = 0;   // No buttons pressed
         USB_LoadInEP(1, abMouseData, 3, USB_TOGGLE); // Load and cock Endpoint1


      }  // End if Endpoint ready
      break;

   }  // End Switch
}   // End While
}
```


[+] Feedback

以下是用汇编语言代码编写的 USB 引导加载程序用户模块的实现。

此处所示汇编语言代码向您说明如何在简单 HID 应用程序中使用 BootLdrUSBFS 用户模块。 当连接到 PC 主机时，器件会枚举为 3 键鼠标。 当运行代码时，鼠标光标会呈之字形从右到左闪现。 此代码演示 BootLdrUSBFS 设置向导如何配置用户模块。 此项目与 USBFS 用户模块中的项目相同，只不过添加了一个引导加载程序。

```
;-------------------------------------------------------------------------
; Assembly main line
;-------------------------------------------------------------------------

include "m8c.inc" ; part specific constants and macros
include "memory.inc" ; Constants & macros for SMM/LMM and Compiler
include "PSoCAPI.inc" ; PSoC API definitions for all User Modules



export _main
export i
export abMouseData

area bss(RAM) // inform assembler that variables follow
abMouseData: blk 3 // USBFS data variable
i: blk 1 // count variable

area text(ROM,REL) // inform assembler that program code follows

_main:
OR F,1
; Start USBFS Operation using device 0
PUSH X
MOV X,3
MOV A,0
LCALL BootLdrUSBFS_Start
POP X

; Wait for Device to enumerate
.no_device:
PUSH X
LCALL BootLdrUSBFS_bGetConfiguration
POP X
CMP A,0
JZ .no_device
; Enumeration is completed load endpoint 1. Do not toggle the first time
; BootLdrUSBFS_LoadInEP(1, abMouseData, 3, USB_TOGGLE);
PUSH X
MOV A,1
PUSH A
MOV A,0
PUSH A
MOV A,3
PUSH A
MOV A,>abMouseData
PUSH A
```


[+] Feedback

```
MOV A,<abMouseData
PUSH A
MOV A,1
PUSH A
LCALL BootLdrUSBFS_LoadInEP
ADD SP,250
POP X

.endless_loop:



;implement bootloader entry
mov reg[PRT1DR], 0   ;load reg[PRT1DR] with  0
;   if(PRT1DR & 0x80) {
;   USB_Stop();
;   while(PRT1DR & 0x80);
;   BootLdrUSBFS_EnterBootloader();
;   }
push A
mov A, reg[PRT1DR]
and A, 0x80
jz .Exit_BOOTLOAD_TEST
;****    IMPORTANT,      configure prt0.7 as a stdcpu/pulldown     IMPORTANT     *****
;if PRT1DR.7 is pulled high, (configure for a pull down, set data to zero)
; wait for the port pin to be released (back to zero) to debounce
; immeciately un-enumerate by releasing the D+ pullup
lcall BootLdrUSBFS_Stop
.wait_for_bit_low:
tst reg[PRT1DR], 0x80
jnz .wait_for_bit_low
; once it goes low enter the bootloader
pop A
ljmp BootLdrUSBFS_EnterBootloader ;;never returns
halt

.Exit_BOOTLOAD_TEST:
pop A


;;; mouse operations



PUSH X
MOV A,1
LCALL BootLdrUSBFS_bGetEPAckState
POP X
CMP A,0
JZ .endless_loop
; ACK has occurred, load the endpoint and toggle the data bit
; BootLdrUSBFS_LoadInEP(1, abMouseData, 3, USB_TOGGLE);
PUSH X
MOV A,1
PUSH A
```


[+] Feedback

```
MOV A,0
PUSH A
MOV A,3
PUSH A
MOV A,>abMouseData
PUSH A
MOV A,<abMouseData
PUSH A
MOV A,1
PUSH A
LCALL BootLdrUSBFS_LoadInEP
ADD SP,250
POP X

; When our count hits 128
CMP [i],128
JNZ .move_left
; Start moving the mouse to the right
MOV [abMouseData+1],5
JMP .increment_i
; When our counts hits 255
.move_left:
CMP [i],255
JNZ .increment_i
; Start moving the mouse to the left
MOV [abMouseData+1],251

.increment_i:
INC [i]

JMP .endless_loop

.terminate:
jmp .terminate
```


[+] Feedback

与示例代码相对应的 USBFS 设置

1. 新建一个包含 BootLdrUSBFS 用户模块 （如 CY8C24894）支持的基本部件的项目。
2. 在器件编辑器中，单击**协议**。 通过双击 BootLdrUSBFS 图标或右键单击并选中**选择**，从而添加 BootLdrUSBFS 用户模块。
3. 选择人体学接口器件 (HID) 单选按钮。 可选步骤：将用户模块从 BootLdrUSBFS_1 重命名为 BootLdrUSBFS，以便与采样代码相匹配，方法为：右键单击该模块，并选中**重命名**。
4. 在器件编辑器中右键单击 “USBFS 用户模块 ” 图标，以打开 “ 器件 : 应用程序 USB 设置向导 ” 。
    - 单击 “ 导入 HID 报告样本 ” 操作，并将名称更改为 “ 导入 HID 报告样本 ” （斜体），以显示它是一个标签。
    - 选择 3 按钮鼠标模板。
    - 单击样本右侧的 “ 应用 ” 操作。
    - 选择 “ 添加字符串 ” 操作，以添加 “ 制造商 ” 和 “ 产品 ” 字符串。
    - 编辑器件属性： 供货商 ID、产品 ID，并选择字符串。
    - 编辑接口属性：针对 “ 类别 ” 字段选择 “HID”。
    - 编辑 HID 类描述符：为 HID 报告字段选择 3 按钮鼠标。
5. 单击**确定**保存 USB 描述符信息。
6. 在器件编辑器中右键单击 “USBFS 用户模块 ” 图标，以打开 “ 器件 : BootLoader USB 设置向导 ” 。
7. 将正确的 VID（供货商 ID）和 PID（产品 ID）输入到向导中。 请注意，应用程序和引导加载程序的 VID 和 PID 不能相同。
8. 单击**确定**以保存 USB 引导加载程序描述符信息。
9. 生成应用程序。
10. 复制样本代码并将其粘贴在 main.c 中。
11. 执行**重建全部**。

描述符	Data
USB 用户模块描述符根	器件名称
器件描述符	器件
器件属性	
供货商 ID	使用公司 VID
产品 ID	使用产品 PID
器件发布 (bcdDevice)	0000
器件类别	在接口描述符中定义
子类	无子类
制造商字符串	我的公司
产品字符串	我的鼠标
序列号字符串	无字符串

[+] Feedback

描述符	Data
配置描述符	配置
配置属性	
配置字符串	无字符串
最大功率	100
器件电源	总线供电
远程唤醒	已禁用
接口描述符	接口
接口属性	
接口字符串	无字符串
类别	HID
子类	无子类
HID 类别描述符	
描述符类型	报告
国家 / 地区代码	不受支持
HID 报告	3-button 鼠标
端点描述符	ENDPOINT_NAME
端点属性	
端点号	1
方向	IN
传输类型	INT
间隔	10
最大数据包大小	8
字符串 /LANGID	
字符串描述符	USBFS
LANGID	
字符串	我的公司
字符串	我的鼠标
描述符	
HID 报告描述符根	USBFS
HID 报告描述符	USBFS

[+] Feedback

附录 A - USBFS 主题

USB 标准器件请求

以下章节介绍了 USB 用户模块支持的请求。 如果某个请求不受支持， 则 USB 用户模块通常将以一个 STALL 作为响应，包括请求错误。

标准器件请求	USB 用户模块支持说明	USB 2.0 规范章节
CLEAR_FEATURE	器件：	9.4.1
	接口： 不受支持。	
	端点	
GET_CONFIGURATION	返回当前器件配置值。	9.4.2
GET_DESCRIPTOR	返回指定的描述符。	9.4.3
GET_INTERFACE	为指定的接口返回选定的备选接口设置。	9.4.4
GET_STATUS	器件：	9.4.5
	接口：	
	端点：	
SET_ADDRESS	设置所有后续器件访问的器件地址。	9.4.6
SET_CONFIGURATION	设置器件配置。	9.4.7
SET_DESCRIPTOR	不支持此可选请求。	9.4.8
SET_FEATURE	器件： DEVICE_REMOTE_WAKEUP 支持由 bRemoteWakeUp 用户模块参数选定。 不支持 TEST_MODE。	9.4.9
	接口： 不受支持。	
	端点： 已暂停指定端点。	
SET_INTERFACE	不受支持。	9.4.10
SYNCH_FRAME	不受支持。 用户模块的进一步实现可以为此请求提供支持，从而可使用重复的帧模式进行同步传输。	9.4.11

[+] Feedback

HID 类别请求

类别请求	USB 用户模块支持说明	HID 器件类别定义 - 章节
GET_REPORT	可让主机经由控制管接收报告。	7.2.1
GET_IDLE	读取特定输入报告的目前空闲率。	7.2.3
GET_PROTOCOL	读取目前处于活动状态的协议 （引导协议或报告协议）。	7.2.5
SET_REPORT	可以让主机给器件发送报告，可以是设置输入、输出或特性控制的状态。	7.2.2
SET_IDLE	在新事件发生或经过指定时间段之前，不再发送有关中断型输入管的特定报告。	7.2.4
SET_PROTOCOL	在引导协议和报告协议之间相互切换。	7.2.6

USBFS 设置向导

本部分详细介绍 USBFS 用户模块指定的所有 USBFS 描述符。 这些描述包括描述符格式以及用户模块参数如何映射到描述符数据。

USBFS 设置向导是赛普拉斯提供的一种工具，可协助工程师设计 USB 器件。 此安装向导会显示器件描述符树；展开后，就会显示以下属于标准 USB 描述符定义的文件夹：

- 器件属性
- 配置描述符
- 接口描述符
- HID 类别描述符
- 端点描述符
- 字符串 /LANGID
- HID 描述符

要访问此安装向导，请右击器件编辑器中的 “USB 用户模块 ” 图标，然后单击 “USB 安装向导 ...” 菜单项目。

完全展开器件描述符树后，就会显示所有安装向导选项。 左侧显示描述符的名称，中间显示数据，右侧显示可用于特定描述符的操作。 某些情况下，描述符有一个下拉菜单，可显示提供的选项。

描述符	Data		操作
USBFS 用户模块描述符根	“ 器件名称 ”		添加器件
器件描述符	DEVICE_1		删除 | 添加配置
器件属性			
供货商 ID	FFFF		
产品 ID	FFFF		
器件发布 (bcdDevice)	0000		
器件类别	未定义	下拉	

[+] Feedback

描述符	Data		操作
子类	无子类	下拉	
协议	无	下拉	
制造商字符串	无字符串	下拉	
产品字符串	无字符串	下拉	
序列号字符串	无字符串	下拉	
配置描述符	CONFIG_NAME		删除 | 添加接口
配置属性			
配置字符串	无字符串	下拉	
最大功率	100		
器件电源	总线供电	下拉	
远程唤醒	已禁用	下拉	
接口描述符	INTERFACE_NAME		删除 | 添加端口
接口属性			
接口字符串	无字符串	下拉	
类别	供货商特定	下拉	
子类	无子类	下拉	
HID 类别描述符			
描述符类型	报告	下拉	
国家 / 地区代码	不受支持	下拉	
HID 报告	无	下拉	
端点描述符	ENDPOINT_NAME		删除
端点属性			
端点号	0		
方向	IN	下拉	
传输类型	CNTRL	下拉	
间隔	10		
最大数据包大小	8		
字符串 /LANGID			
字符串描述符	器件名称		添加字符串

[+] Feedback

描述符	Data		操作
LANGID		下拉	
字符串	所选字符串名称		删除
描述符			
HID 描述符	器件名称		导入 HID 报告样本

了解 USB 设置向导

“USB 设置向导 ” 窗口是一个表格，其中显示用于编程的三个主要区域。 第一个区域是 “ 描述符 USBFS” 用户模块，第二个区域是 “ 字符串 /LANGID”，第三个区域是 “ 描述符 HID” 报告。 使用表格下面的两个按钮执行所选的命令。

第一部分显示 “ 描述符 ”。 第二部分显示 “ 字符串 /LANGID” ；如果需要字符串 ID，那么此区域用于输入该字符串。 要为 USB 器件添加字符串，请单击**添加字符串**操作。 软件即会添加一行，提示您在此编辑字符串。 键入新字符串，然后单击**保存 / 生成**。 保存字符串之后，就可以在下拉菜单的 “ 描述符 ” 部分使用它了。 如果在未保存的情况下关闭，则将丢失该字符串。

第三个区域显示 “HID 报告描述符根 ”。 从这个区域中，您可以为所选器件添加或导入 HID 报告。

USB 用户模块描述符报告

第一列显示要展开和折叠的文件夹。 为了进行此讨论，您必须完全展开该树，以显示所有选项。 安装向导可让您在中间的 “ 数据 ” 列中输入数据；如果有下拉菜单，可以使用下拉菜单选择其他选项。 如果没有下拉菜单但有数据，可以使用光标突出显示数据并选择数据，然后用其他值或文字选项覆盖该数据。 所有值必须满足 USB 2.0 第九章的规范。

顶部显示的第一个文件夹是 *USB User Module Descriptor Root*. 它在 “ 数据 ” 列中有一个用户模块名称 （这是软件为其指定的用户模块名称）。 这个用户模块就是在互连视图中置入的用户模块。 右侧列中的 “ 添加器件 ” 操作可以添加配有描述器件所需的所有不同字段的其他 USB 器件。 新的 USB 器件描述符列在底部，在端点描述符的后面。 单击**确定**保存。 如果不保存新添加的器件，就不能使用该器件。

器件描述符将 DEVICE_NUMBER 作为数据；可能会将其删除，也可能会添加配置。 有关特定 USB 器件的所有信息都可以通过取代现有数据或使用下拉菜单来输入。

如果输入完数据 （无论是通过使用下拉菜单还是通过在相应的地方键入字母数字文字），请单击**确定**保存。

USB 挂起、恢复和远程唤醒

USBFS 用户模块支持 USB 挂起、恢复和远程唤醒。 由于这些功能与用户应用紧密相连，因此 USBFS 用户模块提供了一组 API 函数。

监控 USFS 活动

利用 BootLdrUSBFS_bCheckActivity API 函数可检查是否有 USB 总线活动发生。 应用程序使用此函数可确定是否满足进入 USB 挂起的条件。

USBFS 挂起

满足进入 USB 挂起的条件之后，应用程序会采取适当的步骤来减少电流消耗，以满足暂停电流要求。 为了将 USB SIE 和收发器置于断电模式，应用程序会调用 BootLdrUSBFS_Suspend API 和 BootLdrUSBFS_bCheckActivity API 函数，以检测 USB 活动。 此函数会禁用 USBFS 模块，但是会保持当前的 USB 地址 （在 USBIO_CR0 寄存器中）。 器件利用睡眠功能来减少功耗。

[+] Feedback

USBFS 恢复

当器件挂起时，它会定期检查以确定是否满足保持挂起状态的条件。 检查恢复条件的一种方法是利用睡眠定时器定期唤醒器件。 如果满足恢复条件，则应用程序会调用 BootLdrUSBFS_Resume API 函数。 此函数会启用 USBFS SIE 和收发器，从而使它们摆脱断电模式。 这样做不会更改 USBCR 寄存器的 USB 地址字段，从而可保持以前由主机分配的 USB 地址。

USBFS 远程唤醒

如果器件支持远程唤醒，则应用程序能够确定主机是否已借助 BootLdrUSBFS_bRWUEnabled API 函数启用远程唤醒。 当器件挂起且它确定满足用于启动远程唤醒的条件时，应用程序会使用 BootLdrUSBFS_Force API 函数强制使 USB 总线进入到相应的 J 和 K 状态，从而标志着远程唤醒成功。

创建供货商特定的器件请求并覆盖现有请求

USBFS 用户模块通过提供用于处理设置数据包请求的调度子程序，来支持供货商特定的器件请求。 您也可以写入覆盖任何所提供的标准及类别特定子程序的您自己的子程序，也可以启用不受支持的请求类型。

处理 USBFS 器件请求

所有控制传输 （包括供货商特定的和覆盖的器件请求）均由以下内容构成：

- 一个设置阶段，在此阶段中，将请求信息从主机移至器件。
- 数据阶段，该阶段包含零或更多数据操作，会按照在设置阶段中指定的方向发送数据。
- 状态阶段，该阶段将结束传输。

在 BootLdrUSBFS 用户模块中，所有的控制传输均由端点 0 中断服务子程序 (BootLdrUSBFS_EP0_ISR).) 处理。

端点 0 中断服务子程序将所有设置数据包的控制传输给调度子程序，而调度子程序会将请求路由到基于 bmRequestType 字段的相应处理程序。 处理程序会初始化特定用户模块数据结构，并将控制传输回端点 0 中断服务子程序 (ISR)。 供货商特定或覆盖器件请求的处理程序由应用程序提供。 用户模块会处理传输的数据和状态阶段，而不会涉及您的应用程序。 在传输完成之后，用户模块会更新完成状态模块。 应用程序会监控该状态模块，以确定供货商特定的器件请求是否已完成。

所有设置数据包均会进入 BootLdrUSBFS_EP0_ISR，它会将设置数据包路由到 BootLdrUSBFS_bmRequestType_Dispatch 子程序。 所有的标准器件请求和供货商特定的器件请求均从这里调度。 器件请求处理程序必须准备用于接收进行控制写入的数据的应用程序，或准备用于传输到主机以进行控制读取的数据。 对于无数据控制传输，处理程序会从设置数据包自身中提取信息。

USBFS 用户模块处理数据和状态阶段的方式与处理所有请求的方式完全相同。 对于数据阶段，会将数据复制到控制端点缓冲区 （寄存器 EP0DATA0-EP0DATA7）或从该缓冲区中复制数据，具体取决于数据操作的方向。

[+] Feedback

供货商特定器件请求调度子程序

根据应用程序要求的不同，USBFS 用户模块可基于设置数据包的 bmRequestType 字段，调度多达八种供货商特定器件请求。 请参见 USB 2.0 规范的第 9.3 节，以获取有关 USB 器件请求和 bmRequestType 字段的说明。 表 1 中列出了 USBFS 用户模块调度的八种供货商特定器件请求：

Table 1. 供货商特定请求调度子程序名称

方向	接收方	调度子程序输入点	启用标志
主机至器件 （控制写入）	器件	USB_DT_h2d_vnd_dev_Dispatch	USB_CB_h2d_vnd_dev
	接口	USB_DT_h2d_vnd_ifc_Dispatch	USB_CB_h2d_vnd_ifc
	端点	USB_DT_h2d_vnd_ep_Dispatch	USB_CB_h2d_vnd_ep
	其他	USB_DT_h2d_vnd_oth_Dispatch	USB_CB_h2d_vnd_oth
器件至主机 （控制读取）	器件	USB_DT_d2h_vnd_dev_Dispatch	USB_CB_d2h_vnd_dev
	接口	USB_DT_d2h_vnd_ifc_Dispatch	USB_CB_d2h_vnd_ifc
	端点	USB_DT_d2h_vnd_ep_Dispatch	USB_CB_d2h_vnd_ep
	其他	USB_DT_d2h_vnd_oth_Dispatch	USB_CB_d2h_vnd_oth

您必须针对应用程序执行以下步骤，从而为供货商特定的器件请求提供汇编语言调度子程序。

1. 在 *USBFS.inc* 在 USBFS.inc 文件中，启用对供货商特定的调度子程序的支持。 查找调度子程序启用标志并将 EQU 设置为 1。
2. 写入适当命名的汇编语言子程序，以处理器件请求。 使用在表 1 中列出的输入点。

覆盖现有请求子程序

要覆盖标准或类别特定的器件请求，或启用某个不受支持的器件请求，您必须执行以下操作：

1. 在 *USBFS.inc* 在 USBFS.inc 文件中，将特定的器件请求重新定义为 USB_APP_SUPPLIED.。
2. 写入适当命名的汇编语言函数，以处理器件请求。 该汇编语言函数的名称为 APP_ 加上器件名称。

例如，要覆盖所提供的 HID 类别设置报告请求 USB_CB_SRC_h2d_cls_ifc_09，请启用带有这些更改的子程序为 USBFS.inc： *USBFS.inc*:

```
;@PSoC_UserCode_BODY_1@ (Do not change this line.)
   ;---------------------------------------------------
   ; Insert your custom code below this banner
   ;---------------------------------------------------
   ;   NOTE: interrupt service routines must preserve
   ;   the values of the A and X CPU registers.

; Enable an override of the HID class Set Report request.
USB_CB_SRC_h2d_cls_ifc_09: EQU USB_APP_SUPPLIED

   ;---------------------------------------------------
   ; Insert your custom code above this banner
   ;---------------------------------------------------
   ;@PSoC_UserCode_END@ (Do not change this line.)
```


[+] Feedback

然后，写入汇编语言子程序，名称为 APP_USB_CB_SRC_h2d_cls_ifc_09.。器件请求名称派生自 USB bmRequestType 和 bRequest 值 （USB 规范表 9-2）。

此代码是上个示例的汇编子程序的一个存根：

```
export APP_USB_CB_SRC_h2d_cls_ifc_09
APP_USB_CB_SRC_h2d_cls_ifc_09:

;Add your code here.

; Long jump to the appropriate return entry point for your application.
LJMP BootLdrUSBFS_InitControlWrite
```

附录 B - 引导加载程序主题

以下部分包含了用户在创建 USB 引导加载程序时可能用到的附加信息。

调度和覆盖子程序要求

调度或覆盖子程序至少必须通过利用至表 2 中列出的其中一个端点 0 ISR 返回点的 LJMP，将控制返回到端点 0 ISR。 子程序可能会破坏 A 和 X 寄存器，但是首先必须恢复堆栈指针 (SP) 和任何其他相关上下文，然后再将控制返回到 ISR。

Table 2. 端点 0 ISR 返回点

返回输入点	所需的数据项	说明
USB_Not_Supported	当请求不受支持时，使用此返回点。 它会停止请求。	
	数据项： 无	
USB_InitControlRead	此返回点用于启动控制读取传输。	
	BootLdrUSBFS_DataSource (BYTE)	数据源为 RAM 或 ROM（USB_DS_RAM 或 USB_DS_ROM)。。这很有必要，因为不同的指令用于从源 ROMX 或 MOV 移动数据。
	BootLdrUSBFS_TransferSize (WORD)	要传输的数据字节数目。
	BootLdrUSBFS_DataPtr (WORD)	数据的 RAM 或 ROM 地址。
	BootLdrUSBFS_StatusBlockPtr (WORD) 可选	利用 USB_XFER_STATUS_BLOCK 宏分配的状态模块的地址。
BootLdrUSBFS_InitControlWrite	此返回点用于启动控制写入传输。	
	BootLdrUSBFS_DataSource (BYTE)	USB_DS_RAM （控制写入的目标必须为 RAM）。
	BootLdrUSBFS_TransferSize (WORD)	用于接收数据的应用程序缓冲区的大小

[+] Feedback

返回输入点	所需的数据项	说明
	BootLdrUSBFS_DataPtr (WORD)	用于接收数据的应用程序缓冲区的 RAM 地址
	BootLdrUSBFS_StatusBlockPtr (WORD) 可选	利用 USB_XFER_STATUS_BLOCK 宏分配的状态模块的地址。
USB_InitNoDataControlTransfer	此返回点用于启动无数据控制传输。	
	BootLdrUSBFS_StatusBlockPtr (WORD) 可选	利用 USB_XFER_STATUS_BLOCK 宏分配的状态模块的地址。

状态完成模块

状态完成模块包含两个数据项，即一个一字节的完成状态代码和一个二字节的数据传输长度。 “ 主要的 ” 应用程序会监控完成状态，以确定如何继续。 可在下表中找到完成状态代码。 数据传输长度为所传输数据字节的实际数目。

Table 3. USBFS 传输完成代码

完成代码	说明
USB_XFER_IDLE (0x00)	USB_XFER_IDLE 指示相关数据缓冲区无有效数据，应用程序不应使用该缓冲区。实际数据传输发生在当完成代码为 USB_XFER_IDLE 时，但是它不会指示某个传输正在进行中。
USB_XFER_STATUS_ACK (0x01)	USB_XFER_STATUS_ACK 指示控制传输状态阶段已成功完成。 此时，应用程序使用相关数据缓冲区及其内容。
USB_XFER_PREMATURE (0x02)	USB_XFER_PREMATURE 指示控制传输被后续控制传输的 SETUP 中断。 对于控制写入，相关数据缓冲区的内容包含直到提前完成时的数据。
USB_XFER_ERROR (0x03)	USB_XFER_ERROR 指示未收到预期的状态阶段令牌。

自定义 HID 类别报告存储区域

如果您启用可选 HID 类别支持，设置向导会为来自 HID 类别器件的数据报告创建固定大小的报告存储区域。 它会为 IN、OUT 和 FEATURE 报告创建单独的报告区域。 如果没有报告 ID 项目标志出现在报告描述符中，从而只存在一个输入、输出和特性报告结构，则此区域是足够的。 如果要更好地控制报告存储器容量或要支持多个报告 ID，则您需要执行以下操作：

1. 使用向导指定您的器件描述、端点和 HID 报告，然后生成应用程序。
2. 在 USB_descr.asm 中禁用向导定义的报告存储区域。 *USB_descr.asm*
3. 复制用于定义报告存储区域的向导创建的代码。
4. 将它粘贴到 USB_descr.asm 中的受保护用户代码区域中 *USB_descr.asm* 或单独的汇编语言文件中。
5. 自定义代码，以定义报告存储区域。

指定您的器件和生成应用程序

使用 USB 设置向导，以指定您的器件说明、端点和 HID 报告。 在 PSoC Designer 中单击**生成应用程序**按钮。

[+] Feedback

禁用向导定义的报告存储区域

在 *USB_descr.asm* 在 USB_descr.asm 文件中，禁用向导定义的存储区域，方法为：在自定义代码区域中，取消对 WIZARD_DEFINED_REPORT_STORAGE 行的注释，如下列代码中所示：

```
WIZARD: equ 1
WIZARD_DEFINED_REPORT_STORAGE: equ 1
   ;---------------------------------------------------
   ;@PSoC_UserCode_BODY_1@ (Do not change this line.)
   ;---------------------------------------------------
   ; Insert your custom code below this banner
   ;---------------------------------------------------
   ; Redefine the WIZARD equate to 0 below by
   ; uncommenting the WIZARD: equ 0 line
   ; to allow your custom descriptor to take effect
   ;---------------------------------------------------

   ; WIZARD: equ 0
   WIZARD_DEFINED_REPORT_STORAGE: equ 0
   ;---------------------------------------------------
   ; Insert your custom code above this banner
   ;---------------------------------------------------
   ;@PSoC_UserCode_END@ (Do not change this line.)
```

复制创建代码向导

在 USB_descr.asm 中查找此代码。 *USB_descr.asm*.

```
;-------------------------------------------------------------------------
; HID IN Report Transfer Descriptor Table for ()
;-------------------------------------------------------------------------
IF WIZARD_DEFINED_REPORT_STORAGE
AREA  UserModules     (ROM,REL,CON)
.LITERAL
USB_D0_C1_I0_IN_RPTS:
  TD_START_TABLE 1                                ; Only 1 Transfer Descriptor
  TD_ENTRY       USB_DS_RAM, USB_HID_RPT_3_IN_RPT_SIZE, USB_INTERFACE_0_IN_RPT_DATA,
NULL_PTR
.ENDLITERAL
ENDIF ; WIZARD_DEFINED_REPORT_STORAGE
```

有三个部分，分别针对 IN、OUT 和 FEATURE 报告。 复制这三部分。

[+] Feedback

将代码粘贴到受保护的用户代码区域

您可以将代码粘贴到 USB_descr.asm 中显示的受保护用户代码区域中 *USB_descr.asm* 或一个单独的汇编语言文件中：

```
   ;-------------------------------------------------
   ;@PSoC_UserCode_BODY_2@ (Do not change this line.)
   ;-------------------------------------------------
   ; Redefine your descriptor table below. You might
   ; cut and paste code from the WIZARD descriptor
   ; above and then make your changes.
   ;-------------------------------------------------

   ;-------------------------------------------------
   ; Insert your custom code above this banner
   ;-------------------------------------------------
   ;@PSoC_UserCode_END@ (Do not change this line.)
; End of File USB_descr.asm
```

自定义用于定义报告存储区域的代码

要定义报告存储区域，必须编写您自己的传输描述符表条目。 表中包含用于定义所需数据项的存储空间的条目。 表中的每个传输描述符条目都会创建一个新的报告 ID。 ID 是以 0 开始的连续数字编号。 报告 ID 0 保留在 USB 规范中；您不能使用报告 ID 0，但是，当报告描述符中未出现报告 ID 时，可以使用为 ID 0 指定的传输描述符条目。 为了代码效率，您必须按顺序使用报告 ID，以 ID 1 开始。

Table 4. 传输描述符表条目

表条目	所需的数据项	说明
TD_START_TABLE	USB_NumberOfTableEntries	定义的报告 ID 编号。 ID 是以 0 开始的连续数字编号。 不使用报告 ID 0。
TD_ENTRY		
	USB_DataSource	数据源是 RAM 或 ROM （USB_DS_RAM 或 USB_DS_ROM).）。
	USB_TransferSize	数据传输的大小以字节计。 第一个字节是报告 ID。
	USB_DataPtr	数据传输的 RAM 或 ROM 地址。
	USB_StatusBlockPtr	利用 USB_XFER_STATUS_BLOCK 宏分配的状态模块的地址。

[+] Feedback

以下示例设置未使用的报告 ID 0，以及其他两个不同大小的 IN 报告。 只有在您将代码置于 USB_descr.asm 的受保护用户代码区域中时，才需要注释条件汇编语句。 *USB_descr.asm.*

```
;-------------------------------------------------------------------------
; HID IN Report Transfer Descriptor Table for ()
;-------------------------------------------------------------------------
IF WIZARD_DEFINED_REPORT_STORAGE
ELSE

_ID0_RPT_SIZE:   EQU 0       ; 7 data bytes + report ID = 8 bytes (unused)
_SM_RPT_SIZE:    EQU 3       ; 2 data bytes + report ID = 3 bytes
_LG_RPT_SIZE:    EQU 5       ; 4 data bytes + report ID = 5 bytes

AREA data (RAM, REL, CON)

EXPORT _ID0_RPT_PTR
_ID0_RPT_PTR: BLK 0         ; Allocates space for report ID0 (unused)
EXPORT _SM_RPT_PTR
_SM_RPT_PTR:    BLK 3       ; Allocates space for report ID1
EXPORT _LG_RPT_PTR
_LG_RPT_PTR:    BLK 5       ; Allocates space for report ID2

AREA bss (RAM, REL, CON)

EXPORT _SM_RPT_STS_PTR
_SM_RPT_STS_PTR: USB_XFER_STATUS_BLOCK
EXPORT _LG_RPT_STS_PTR
_LG_RPT_STS_PTR: USB_XFER_STATUS_BLOCK

AREA  UserModules     (ROM,REL,CON)
.LITERAL
EXPORT USB_D0_C1_I0_IN_RPTS:
  TD_START_TABLE 3
  TD_ENTRY  USB_DS_RAM, _ID0_RPT_SIZE, _ID0_RPT_PTR, NULL_PTR ; ID0 unused
  TD_ENTRY  USB_DS_RAM, _SM_RPT_SIZE, _SM_RPT_PTR, _SM_RPT_STS_PTR ; ID1
  TD_ENTRY  USB_DS_RAM, _LG_RPT_SIZE, _LG_RPT_PTR, _LG_RPT_STS_PTR ; ID2
.ENDLITERAL

ENDIF ; WIZARD_DEFINED_REPORT_STORAGE
```


[+] Feedback

引导加载程序 USB 下载协议

下图显示了两个示例下载记录 – 第一个和最后一个。 这两个记录中包含了将在 USB 主控与要引导加载的从器件之间传输的实际数据。 记录格式如下图所示：

Figure 5. “输入引导加载程序”命令和第一个数据模块

Packet data (BULK OUT):

FF 38 00 01 02 03 04 05 06 07 00 00 00 00 00 00 — Enter bootloader FF,38
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Status data (BULK IN):

20 20 02 00 00 00 00 AA 00 00 00 00 00 00 00 00 — Status response
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Packet data (BULK OUT):

FF 39 00 01 02 03 04 05 06 07 00 4E 00 30 30 30 — Write block command FF,39
First half of the data block
30 30 7E 30 30 7E 30 30 30 7E 30 30 30 7E 30 30
30 30 30 30 30 30 30 30 30 7E 30 30 30 28 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Status data (BULK IN):

20 20 03 00 00 4E 00 AA 00 00 20 00 00 00 00 00 — Status response
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
30 30 30 30 30 7E 30 30 7E 30 30 30 7E 30 30 30

Packet data (BULK OUT):

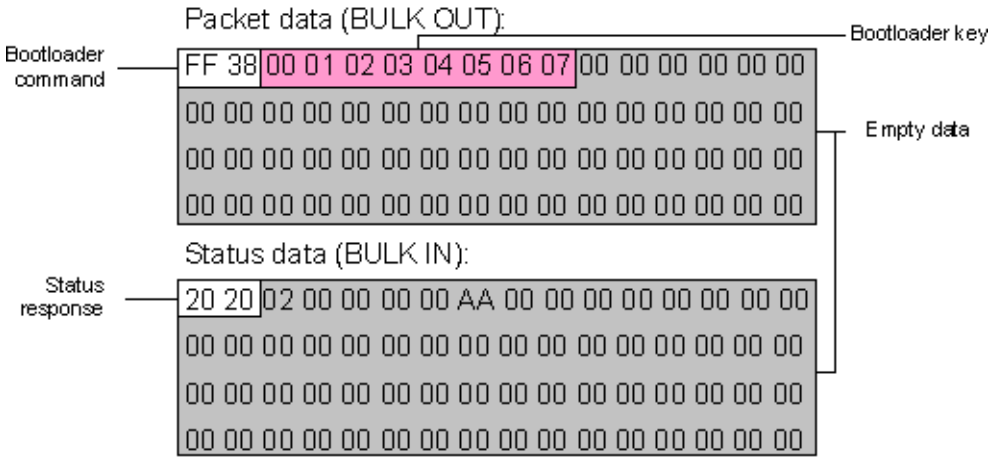
FF 39 00 01 02 03 04 05 06 07 00 4E 01 7E 30 30 — Write block command FF,39
Second half of the data block
30 7E 30 30 30 7E 30 30 30 7E 30 30 30 30 30 30
30 30 30 30 30 30 30 30 30 30 30 30 DB 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

Status data (BULK IN):

20 20 03 01 00 4E 00 AA 00 00 40 00 00 00 00 00 — Status response
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
30 30 30 30 30 7E 30 30 7E 30 30 30 7E 30 30 30

[+] Feedback

针对引导加载程序的每个命令后面都跟随着来自引导加载程序的响应。 下图显示 “ 输入引导加载程序 ” 命令的格式：



第一行以引导加载程序命令 FF38 开始，输入引导加载程序。 此命令后面跟随引导加载程序密钥。 所有引导加载程序命令在发送时必须有引导加载程序密钥。 引导加载程序将忽略发送时没有正确密钥的命令。 您可以使用 Bootloader_Key 参数设置引导加载程序密钥。 其他的引导加载程序命令有：

命令	含义
FF38	进入引导加载程序
FF39	写入模块
FF3A	验证闪存
FF3B	退出引导加载程序
FF3C	更新校验和

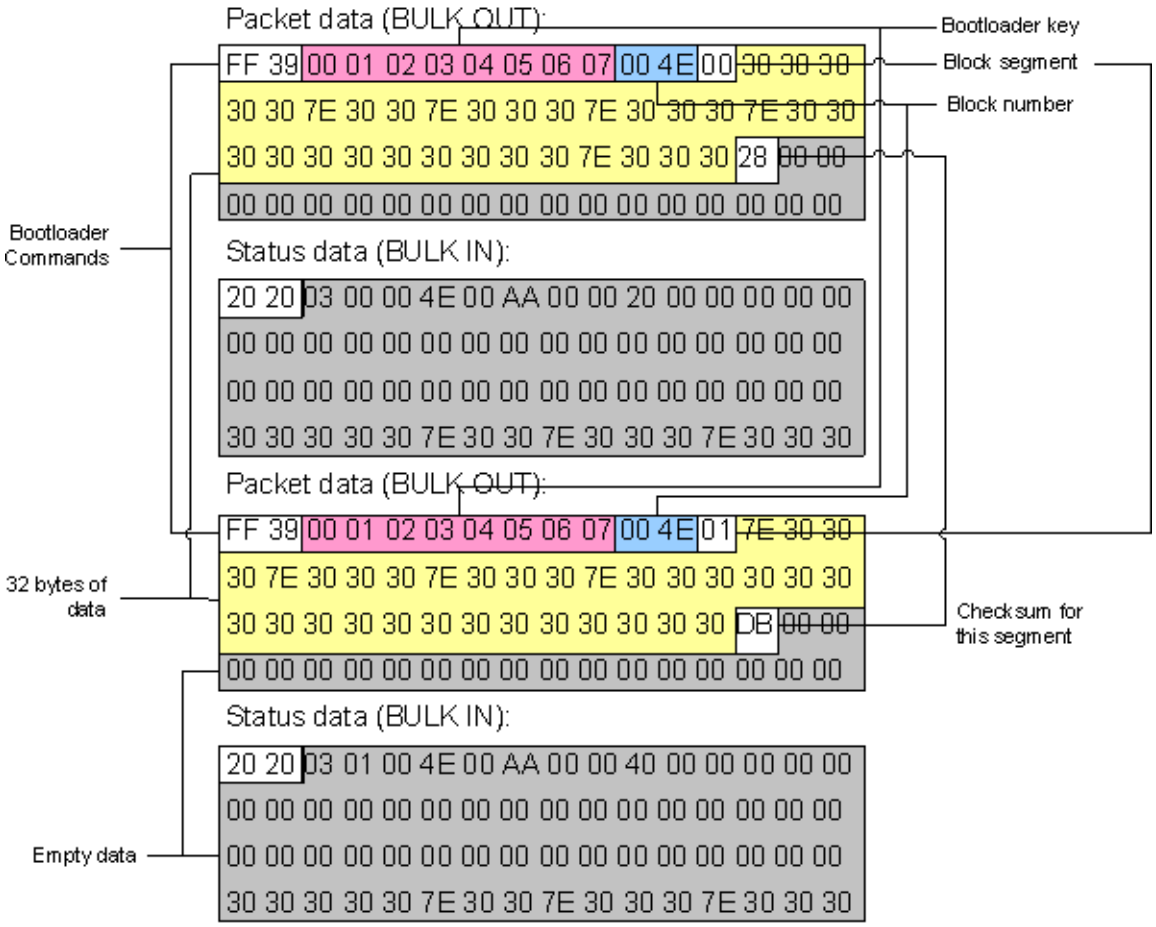
此命令后面跟随来自引导加载程序的状态响应。 所发送的代码 0x20 指示引导加载程序已成功启动。 其他的状态响应有：

代码	含义
0x20	引导加载模式 （成功）
0x01	已成功完成引导
0x02	映像验证错误
0x04	闪存校验和错误
0x08	闪存保护错误
0x10	通信校验和错误
0x40	引导加载程序密钥无效
0x80	无效命令错误

[+] Feedback

引导加载程序写入模块命令

发送到引导加载程序的命令大多数都是写入模块命令。 每个写入模块命令的格式都一样。 每个 64-byte 模块均分为两个 32-byte 数据包。 每个命令均需要来自从器件的状态响应。 64-byte 模块的传输如下图所示：

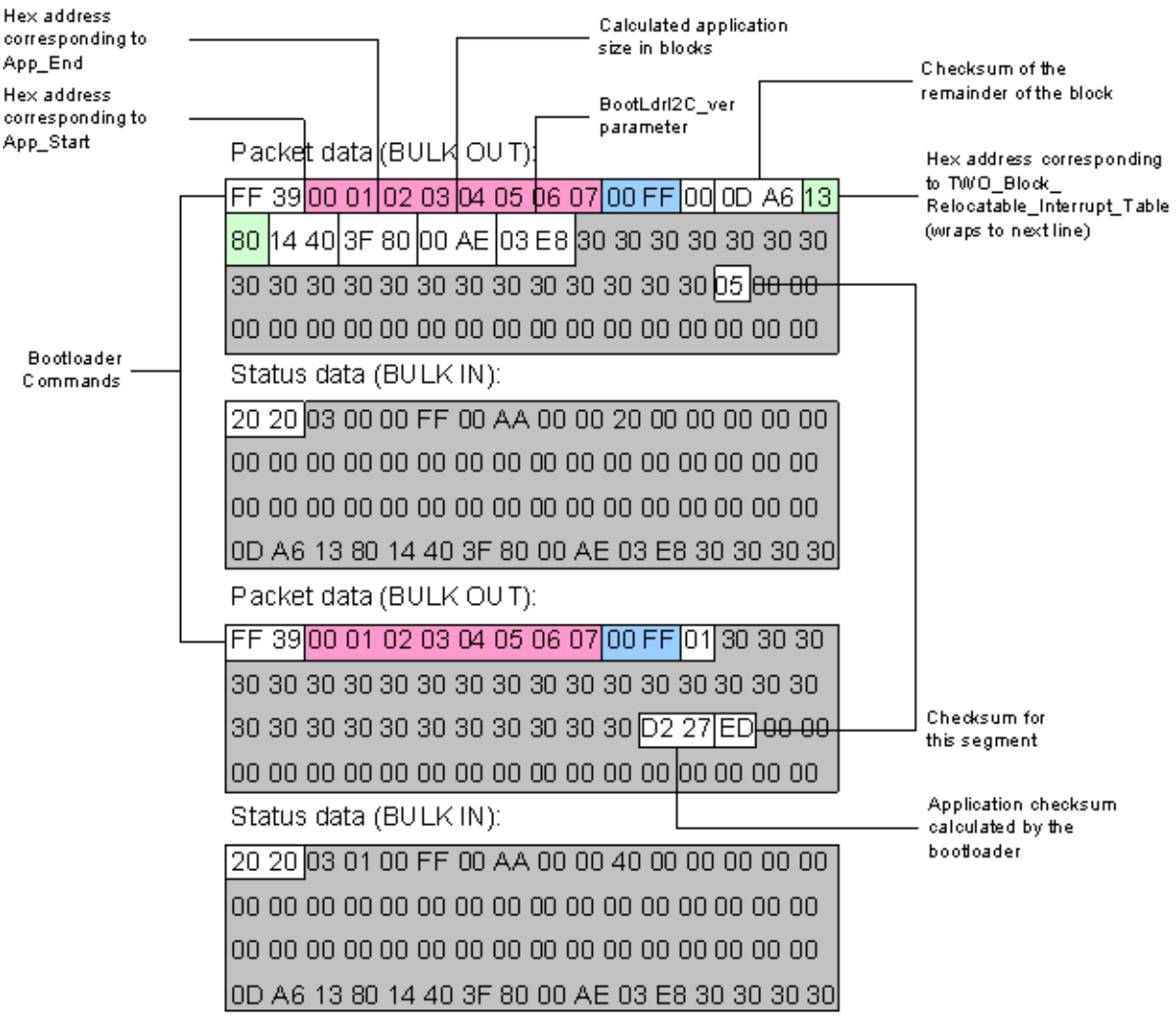


第一个数据包的第一行包含一个写入模块命令和引导加载程序密钥，后跟所传输的模块编号。 由于每个模块均分为两个，则模块编号后跟模块段编号 （第一段为 0x00，第二段为 0x01）。 第一行的最后 3 个字节、第二行的全部 16 个字节以及第二行的前 13 个字节代表有效数据的 32 个字节，后跟段数据的校验和。 模块的剩余部分为空数据，用于将段扩充为 64 个字节。

状态响应包含传输两倍的状态字节和用于将段扩充为 64 个字节的 62 字节空数据。

[+] Feedback

模块第二段的格式与第一段完全相同。 所有传输的数据模块均采用此格式 （最后一个模块除外）。 最后一个模块包含校验和以及用于引导加载程序操作的其他数据。 最后一个数据模块的格式如下图所示：



在本示例中，最后一条记录包含了校验和模块 （请注意，该示例的模块编号为 0x00ff，但是最后一个模块不必为 0x00ff）。

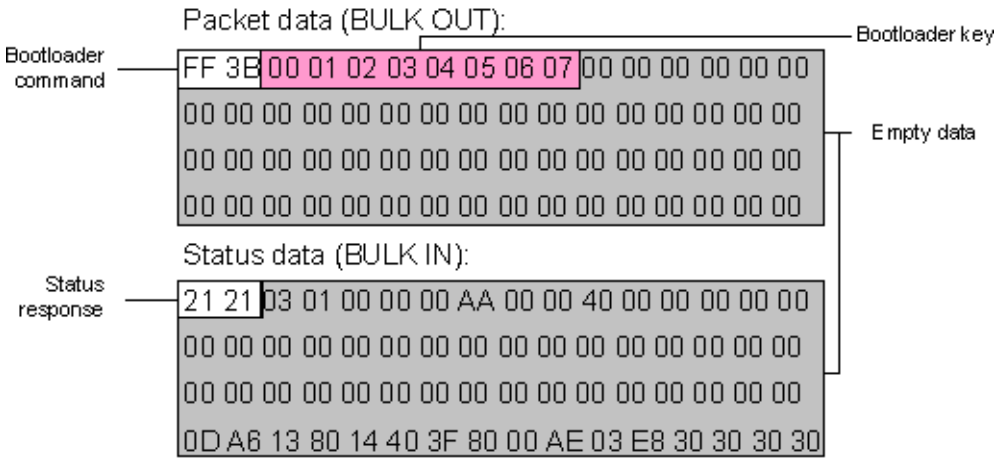
与其他记录相同，第一行包含引导加载程序写入模块命令、引导加载程序密钥、模块编号和模块段。 紧接的两个字节包含了用于模块剩余部分的校验和，本示例中为 0x0DA6。 第一行的最后一个字节和第二行的第一个字节包含了根据 TWO_Block_Relocatable_Interrupt_Table 参数的模块 0x4E 计算得出的十六进制地址。

接着，第二行包含了一个双字节值，该值代表根据模块 0x45 计算得出的 App_Start 用户模块参数的十六进制地址。 紧接的两个字节是根据模块 0xFE 计算得出的 App_End 用户模块参数的十六进制地址。 此后的两个字节是模块中应用程序的大小。 该行的最后两个字节的实际数据值为来自 BootLdrI2C_ver 参数的引导加载程序版本号。 该行的的剩余部分和下一行的大部分为空的数据空间。 段的校验和在数据包中占用的位置与其在其他数据包中占用的位置相同。 数据包的剩余部分为空的空间。

校验和模块的第二个数据包与所有其他数据包一起开始，但是它所包含的唯一数据为应用程序的校验和及第三行中的段校验和。

[+] Feedback

校验和模块后紧接引导加载程序退出命令：



引导加载程序退出命令包含引导加载程序退出命令 0xFF3B 和引导加载程序密钥。

最后的数据包为最终状态响应。

版本历史记录

版本	创作者	说明
1.30	DHA	已删除 .Literal/.Endliteral 指令围绕 jmp 指令。 USB_cls_hid.asm 中的 USBFS_bGetProtocol 和 USBFS_UpdateHIDTimer 函数的已删除指令 .SECTION 和 .ENDSECTION。

Note PSoC Designer 5.1 在所有的用户模块数据表中提供版本历史记录。 本数据表详细介绍了当前和先前用户模块版本之间的区别。

Copyright © 2007-2011 Cypress Semiconductor Corporation. The information contained herein is subject to change without notice. Cypress Semiconductor Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in a Cypress product. Nor does it convey or imply any license under patent or other rights. Cypress products are not warranted nor intended to be used for medical, life support, life saving, critical control or safety applications, unless pursuant to an express written agreement with Cypress. Furthermore, Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress products in life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

PSoC Designer™ and Programmable System-on-Chip™ are trademarks and PSoC® is a registered trademark of Cypress Semiconductor Corp. All other trademarks or registered trademarks referenced herein are property of the respective corporations.

Any Source Code (software and/or firmware) is owned by Cypress Semiconductor Corporation (Cypress) and is protected by and subject to worldwide patent protection (United States and foreign), United States copyright laws and international treaty provisions. Cypress hereby grants to licensee a personal, non-exclusive, non-transferable license to copy, use, modify, create derivative works of, and compile the Cypress Source Code and derivative works for the sole purpose of creating custom software and or firmware in support of licensee product to be used only in conjunction with a Cypress integrated circuit as specified in the applicable agreement. Any reproduction, modification, translation, compilation, or representation of this Source Code except as specified above is prohibited without the express written permission of Cypress.

Disclaimer: CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Cypress reserves the right to make changes without further notice to the materials described herein. Cypress does not assume any liability arising out of the application or use of any product or circuit described herein. Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress' product in a life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

Use may be limited by and subject to the applicable Cypress software license agreement.