USBFS Bootloader 数据手册 BootLdrUSBFSe V 3.00

Copyright © 2010-2014 Cypress Semiconductor Corporation. All Rights Reserved.

资源	PSoC® 模块			API 存储器 （字节）		（每个外部 I/O 的）引脚数量
	数字	模拟 CT	模拟 SC	闪存	RAM	
CY7C64215、 CY8C24794、 CY8C24894-24LTXI、 CY8C24994、 CY8CLED04、 CY8CTMA120-100BVXI、 CY8CTMA120-56LFXI、 CY8CTMA120-56LTXI、 CY8CTMG120-56LFXI、 CY8CTMG120-56LTXI、 CY8CTST120、 CYRF89235、 CY8C24493、 CY7C69xxx						
支持 HID	0	0	0	5 871	76	2
不支持 HID	0	0	0	5 300	49	2
支持 USB-UART	0	0	0	6 380	65	2
CY7C643xx、 CY8C20396/396A/496/496A-24LQXI、 CY8C20646/646A/666/666A-24LTXI、 CY8C20xx6AS、 CY8C20XX6L、 CY8CTMG200-48LTXI、 CY8CTMG201-48LTXI、 CY8CTMG200A-48LTXI、 CY8CTMG201A-48LTXI、 CY8CTST200-48LTXI、 CY8CTST200A-48LTXI、CYONS2000、 CYONS2100、 CYONS2110、 CYONS2010、 CYONSFN2162-LBXC、 CYONSTB2010-LBXC						
支持 HID	0	0	0	6 024	84	2
不支持 HID	0	0	0	5 450	56	2

注意： 当添加额外接口、 HID 类和其他 USB 扩展时，希望扩展闪存和 RAM。如果 Bootloader 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 Bootloader 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/ 闪存的使用情况包括一个完整的 USB 接口。用于 Bootloader 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

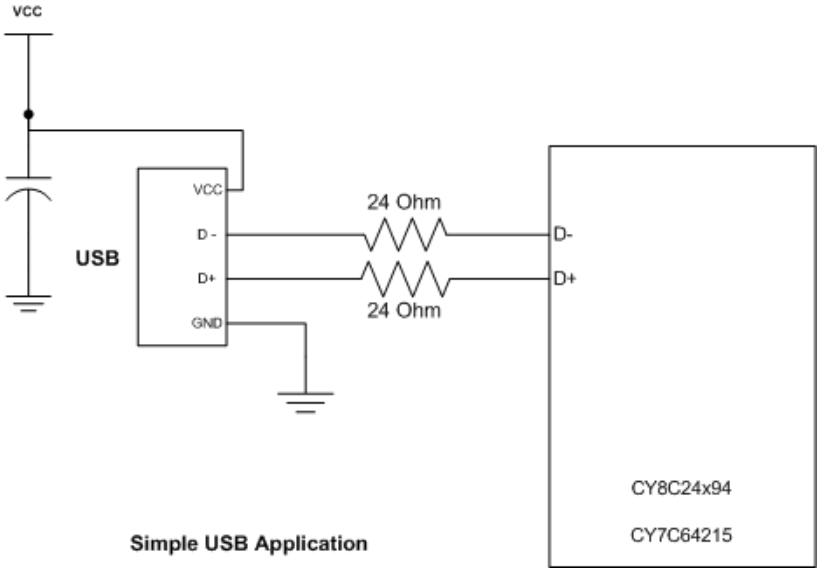
- 灵活存储器映射
- 不使用工程工具情况下的器件重新编程
- 产品固有的可重新编程性
- 集成了通信接口以尽量减少代码开销
- 固件升级的现场部署
- USB 全速器件接口驱动程序
- 支持中断和控制传输类型
- 设置向导便于轻松准确地生成描述符
- 对描述符设置选项提供运行期支持
- 可选的 USB 字符串描述符
- 支持可选的 USB HID 类
- 支持可选的 USB-UART （CDC）类

USB Bootloader 用户模块实现了能够通过 USB 接口对 PSoC 器件重新编程的 Bootloader。 PSoC 器件提供了系统内串行编程接口 （ISSP），能够将新的代码下载到器件中。但是， Bootloader 能够通过工业标准通信接口 （如 USB）进行代码更新。此用户模块对于任何需要现场进行重新编程的器件来说都非常有用。可通过赛普拉斯 USB Bootloader 主机接口发送引导加载信息。

USB Bootloader 支持全功能器件重新编程能力，内置错误检测和工业标准通信接口。

多个 USB 器件描述符在系统中共存，这样便可以指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。

图 1. USBFS Bootloader 框图



快速启动

1. 查看本用户模块数据手册。要成功实施 Bootloader 项目，需要了解此信息。
2. 将用户模块添加到项目中。
3. 通过选择 HID、非 HID 或 USB-UART （CDC）类应用程序放置用户模块。
4. 右键单击用户模块图标。选择 **Boot Loader Tools**。然后选择 **Get Files**。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。

- 如果在第三步骤中选择了 HID 或非 HID 类应用：

  右键单击用户模块图标。选择 **Device: Application USB Setup Wizard…**。验证在 **Strings** 区中只有一个字符串。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。

  对于非 HID 应用程序，跳过以下设置并按下 **OK** 按键。

  - 单击 **Import HID Report Template** （导入 HID 报告模板）报告模板操作。

  - 选择 3 按键鼠标模板。

  - 点击模板右侧的 **Apply** （应用）操作。

  - 编辑接口类：在 Interface Attributes 属性中选择 HID 类。

  - 编辑 HID 类描述符：为 HID Report （HID 报告）字段选择 3 按键鼠标。

  - 点击 **OK** 按键，保存 USB 描述符信息。

- 如果第三步骤选择了 USB-UART 应用：

  右键单击用户模块图标。选择 **Device: Application USB Setup Wizard…**。配置 CDC 属性并点击 **OK** 按键。

5. 右键单击用户模块图标。选择 **Device: Bootloader USB Setup Wizard…**。不需要进行任何修改。选择 **OK** 按钮。
6. 生成源代码并编译项目。
7. 检查输出文件 *<project>.mp* 和 *<project>.hex* 来查看项目的编译方法。
8. 在创建了编译没有错误的项目后，请转到 “ 固件代码示例 ” 部分。修改并调整示例中提供的代码。

功能说明

BootLdrUSBFSe 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 USB 设置向导。

您可以选择构建基于 HID 的通用 USB 器件或 USB-UART 器件。放置 BootLdrUSBFSe 用户模块时进行您的选择。要在初始选择后更改您的选择，请清除 BootLdrUSBFSe 用户模块的现有实例，然后添加新实例。

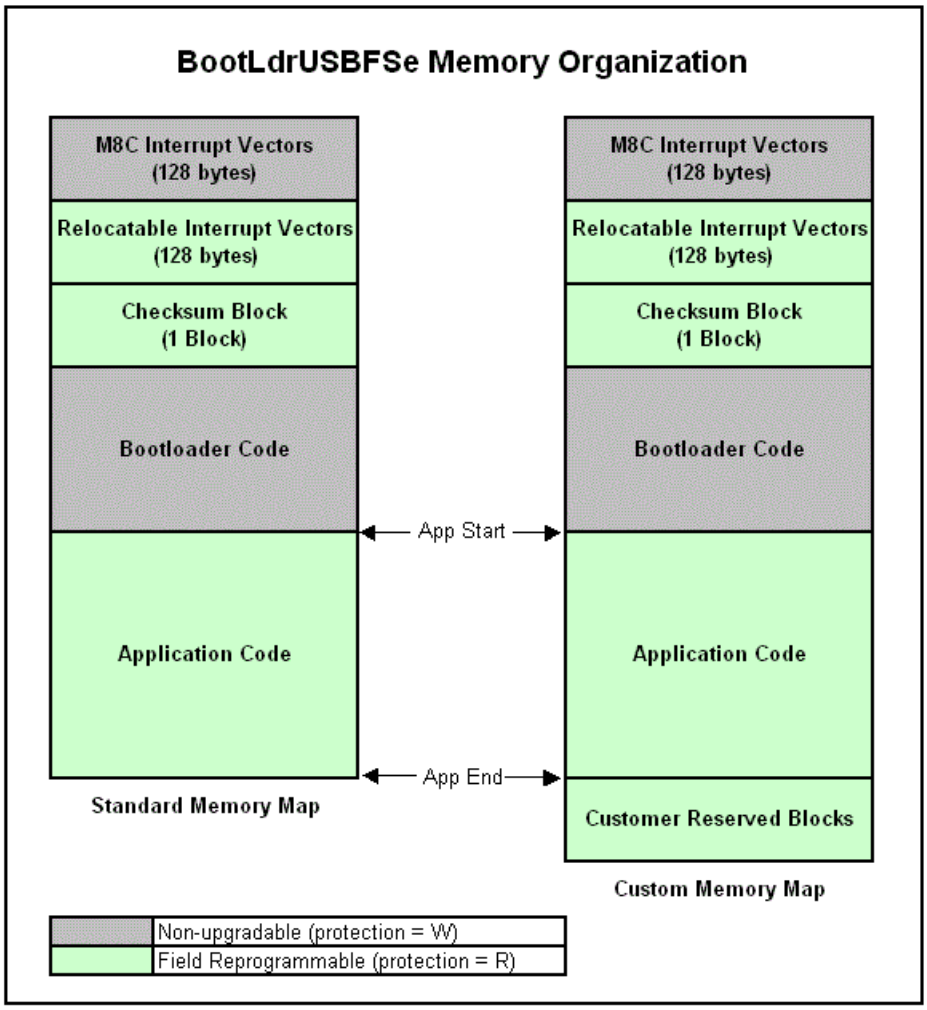
用户模块的 Bootloader 部分提供了一种方法，能够将存储器映射和主要代码功能模块组织到与器件重新编程兼容的区域。该项目与传统 PSoC Designer™ 项目的存储器组织有很大区别。有必要对存储器映射进行修改，以便在对器件应用程序重新编程时满足器件功能的最低要求。实际上，融入了 Bootloader 的项目包含两个独立的程序，这两个程序支持不同的功能。存储器的映射显示在图 2 内。

在部署融入了 Bootloader 的项目后，将无法对以灰色突出显示的存储器位置进行重新编程。应用起始地址会在参数窗口中改变。

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 Bootloader 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 Bootloader 功能的指南。

更多有关 USB 的信息 （包括有关 USB 使用情况的规范、资源示例和论坛），请访问 www.usb.org。

图 2. USBFS Bootloader 存储器组织



操作原理

要创建 Bootloader 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，Bootloader 用户模块提供了自定义文件和专用工具来帮助您进行 Bootloader 项目开发。可以通过切换到器件编辑器视图，然后右键单击 BootLdrUSBFSe 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 Bootloader 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\BootloaderUSBFS\Bootloader_Host_PSoC1\…

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 Bootloader 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。

必须修改包括的 driver.inf 文件才能正确指定所选 Bootloader 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。

USBFS Bootloader 存储器组织

PSoC Designer 使用标准化文件，与器件系列相关的内置数据以及特定器件的属性，以创建可编译的项目和正确的 API 定义。具有 Bootloader 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 Bootloader 的项目仅允许编译器和连接器分配 RAM 和 ROM。但是， Bootloader 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。

图 2 显示的存储器布局中有六个受管理 ROM 的关键区域：

- 第一个区域是 ROM 的前 128 个字节。它包含关键的中断向量和重启向量。由于几乎不可能通过任何操作器件来控制对这些区域的读取访问，因此无法将这些模块擦除和重新编程。不能修改 ROM 的前 128 个字节，且不能将它们放置在任何其他位置。
- 第二个区域是可重定位中断表。根据器件架构的不同，此表可能由一个或两个块组成。该区域包含中断向量以及通用向量，为中断或当使用 Bootloader 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， Bootloader 可以使用此地址启动新应用程序。部署应用程序和 Bootloader 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。
- 定义的第三个 ROM 区域是校验和区域。此区域包含重要数据， Bootloader 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 Bootloader 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 Bootloader 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。
- 定义的第四个存储器区域是包含 Bootloader 代码本身的区域。在部署了包含 Bootloader 的工程或器件之后，将不能对这一区域重新编程或现场升级。
- 第五个区域留作客户数据使用。这一区域可能包含通过引导加载进行升级时必须保留下来的配置数据。该区域可选。
- 第六个存储器区域是应用程序区域。该应用区域包含应用图像。可以对此区域的起始地址进行调整。使用 “Application_Start_Block” 参数 （位于属性窗口中），用户可相应地设置应用程序起始地址。这一区域应占用所有剩余存储器。

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， Bootloader 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 Bootloader 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 Bootloader 定义的 RAM 区域中。

用户模块参数中存储器区域的定义

通过 BootLdrUSBFSe 用户模块的 “Application_Start_Block” 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为止。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。

Bootloader 实用程序

Bootloader 用户模块提供了与前台应用程序共存的完整实用程序。当启动或复位器件时，会始终调用 Bootloader 实用程序。一旦在系统启动时调用 Bootloader， Bootloader 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 Bootloader 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 Bootloader 进入等待循环，等待主机应用程序下载有效应用程序。它还使能自己的 USB 子系统从而允许主机传输数据。如果主机系统发现此接口处于使能状态，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 的安装目录中提供了示例应用程序和源代码。

[Install path]\Cypress\Programmer\3.xx\Bootloaders\BootloaderUSBFS\Bootloader_Host_PSoC1\…

Bootloader 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 Tools （工具）> Restore Default Boot files （恢复默认启动文件）。如果您清除了 USBFSe Bootloader 用户模块，则恢复默认启动文件的选项移动到 PSoC Designer 的 File （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 Bootloader 工具来创建和自动验证校验和。在进入 Bootloader 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 Bootloader 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指令时，错误报告会被抑制。为了调试构建错误，有必要使用处于 Bootloader 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。该文件只包含由 Bootloader 重新编程的十六进制模块，该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 Bootloader 的正在运行中的项目。此下载文件可以部署到现场应用程序中，以升级 PSoC 器件。

dld 下载文件由 Bootloader 用户模块工具生成并列于工作区浏览器中的 “ 输出文件 ” 中。

校验和半自动生成的相关注释

编译项目没有发生错误后，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 “Bootloader 用户模块 ” 图标并选择 **Bootloader Tools**，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 “Bootloader Tools” 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。创建校验和后，必须将其集成到编译文件中。这样就要求进行第二次编译。

请注意以下内容：

在 “ 使用有效校验和重新编译项目 ” 操作后，如果使用正确应用校验和编程该器件，那么启动后，该器件将进入应用。“ 使用有效校验和重新编译项目 ” 和编程操作之间不允许任何 ‘ 生成 ’ 操作。

如果在生成操作后使用错误应用校验和编程器件，或未实现 “ 使用有效校验和重新编译项目 ” 操作情况下，启动后，器件将进入 Bootloader。

注意： 用户模块使用校验和来验证 .hex 代码的完整性。通过 “ 使用有效校验和重新编译项目 ” 可以计算这些校验和，通过生成操作可以清除它们。**因此，如果在编译操作前没有执行生成操作或 “ 使用有效校验和重新编译项目 ”，那么在执行编译操作后，不应该编程器件。**

会发生以下情况：

1. 执行了生成操作
2. 更改了应用代码
3. 执行了编译操作
4. 编程了器件

如果发生这些情况，编程后，器件始终进入 Bootloader。

另外还会发生以下情况：

1. 执行了 “ 使用有效校验和重新编译项目 ”
2. 更改了应用代码
3. 执行了编译 （无生成）
4. 编程了器件

如果发生这些情况，编程后，器件始终进入应用。如果应用代码不正确， Bootloader 只能通过 Acquisition 窗口下载新的应用代码。

提供的特殊文件

通过打开 **Bootloader Tools** 菜单并选择 **Get Files**，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变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 — 对最后三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 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更改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。

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... 和 -L-AROM... 行包含提供总体 ROM 大小的数据
- -L-PPD_startup... 包含用于定位 Bootloader 特殊 ROM 区域的链接器指令
- -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更改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 Bootloader 不工作。由读写保护内置到编程的 PSoC 中，在第一次部署 Bootloader 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。 Bootloader 增加了一个工具，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 Bootloader 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 Bootloader 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

*USB 符合性检查表*中提出了一个问题：“ 器件的上拉电阻是否仅在 VBUS 为高电平时有效？ ”

通用串行总线规范修订版 2.0 中的第 7.1.5 部分内显示了该问题。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，以便当清除 VBUS 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 VBUS，并写入固件以监控 GPIO 的状态。应用笔记 *AN15813* — 监控 EZ-USB FX2LP VBUS 介绍了需要的硬件和软件组件。使用 USB IO 控制寄存器 1 （USBIO_CR1）控制 D+ 线上的上拉电阻。

Bootloader 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。

参数模块入口

所有存储器参数均进入 Bootloader 中，模块编号为 0x00 到 0xFF （对于大多数器件）。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。正在提到的 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。

主机应用程序调试

内置 Bootloader 的应用程序可能较难调试。因此，可能需要在 Bootloader 用户模块文件内部进行额外调整。这些调整包含在 BootLdrUSBFSe_Bt_loader.inc 文件中。其中一个部分包含了下列等式参数：

```
; Boot Timeout constant. The timeout is based on an 8Hz Sleep interval.
; So for a timeout of 1 second, this value has to be 8
BOOT_TIMEOUT:          EQU    200  ; set to zero to make timeout infinite,
                                   ; default is 200(dec) = ~25 seconds
CHECKSUM_ON_CKSUMBLK:  EQU     1   ; set to 1: Apply a checksum to the checksum block
                                   ;(adds compile steps and code to verify)
; set to 0: remove verification of checksum on Checksum block
```

BOOT_TIMEOUT 等式允许您增加、减少在用户指令调用 Bootloader 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

BootLdrUSBFSe 用户模块支持 USB 2.0 全速操作。

USBFS 设置向导

BootLdrUSBFSe 用户模块不使用 PSoC Designer 参数网格显示进行 USB 描述符个性化设置。相反，它使用向导来定义应用程序和 Bootloader 的 USB 描述符。通过描述符，向导对用户模块进行个性化设置。

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。如果未先运行应用和 Bootloader 向导，选择适当的属性并生成代码，则 BootLdrUSBFSe 用户模块将无法正常工作。

使用模块而不是地址

任何 Bootloader 都需要写入闪存中， PSoC 只能逐 “ 模块 ” 写入闪存。因此，对于 Bootloader 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 的起始地址为 addr 0， Block_n 的起始地址为 n x Block_size。在 Bootloader 参数中，所有模块均以十六进制相互分隔，因此，乘以 0x40 （64 字节模块）或 0x80 （128 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 64 字节的器件，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，熟悉您使用的器件的闪存块大小。

常见问题

本节讨论创建 Bootloader 项目时发生的问题，并给出如何解决这些问题的建议。

更新 Bootloader 项目、服务包升级和编译器

使用 Bootloader 应用时，应避免更改 PSoC 开发人员环境。这包括未更新 PSoC Designer、 Bootloader 用户模块和编译器。

要了解其原因，请记住 Bootloader 和应用程序最初是一起编译的，但是在部署 bootloadable 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 Bootloader 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 Bootloader 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 Bootloader，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 Bootloader 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。此类函数的公开通过可重定向调用表来完成。此策略意指应用程序对 Bootloader 中的特定地址进行间接调用。由于 Bootloader 和应用程序是一同编译的，因此对 Bootloader 进行的任何更改会使 USB API 函数的地址改变。

尽管 PSoC Designer 支持多个编译器，但不能假定由一个编译器编译的 Bootloader 能够与另一个编译器编译的应用程序相兼容。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 Bootloader 和应用程序是一同编译的，因此假如一对 Bootloader 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

创建项目后，设置该文件的默认设置值。“Flashsecurity.example” 文件中提供了一个示例配置。Flashsecurity.example 通过 Bootloader 工具 — Get File （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。其中一种策略使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 Bootloader 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。

为了进行部署和调试，建议为应用区域提供 ‘U’ （无保护）的闪存安全。对于最终生产考虑，建议在应用区域提供 ‘R’ （读取保护）的闪存安全设置，以防止外部读写。

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 Bootloader 项目的存储器映射与传统项目的存储器映射有较大的区别，因此在调用 Bootloader 工具窗口时，需要更改可重定位代码起始地址。可以在 Project > Settings > Linker 选项卡中的 “ 可重定位代码起始地址 ” 字段中查看 （更改）此参数。对这些参数进行乘以操作会得到该值：

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

注意： 取消放置 Bootloader UM 后，可重定位代码起始地址无法复位到其原始值。用户需要手动改回，以节省 ROM 空间。

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器件不能正常运行，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，中断向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 （通过将向量复位为 0）。请注意，这并不适用于 Bootloader 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 Bootloader 使用。应当实现一个明确关闭 USB 应用程序接口并使能 Bootloader 接口的方法，其中也包括完全关闭正在运行的应用程序。本数据手册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 Bootloader 实用工具的方便性。很容易不经意地这样做。例如，包含一个简单 while(1); loop 的 main() 函数将绝对不会返回且绝对不会进入 Bootloader。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 Bootloader 被使能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 Bootloader，在超时过期时退出前台应用程序。
2. 在代码中能够让器件进入 Bootloader 的某些点设置测试。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。当发送此指令时，器件暂时关闭 USB 总线，当指令返回时，它应当枚举为 Bootloader。
4. 如果器件处于非正常情况，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 Bootloadable 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 Bootloader 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新编程应用程序中每一个的 Bootloader 的实现均必须完全一致。所有 Bootloader 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两个十六进制文件进行对比，并特别关注 Bootloader 所使用的十六进制代码区域。另一种方法是对 <project>.lst 文件进行比较。 Bootloader 利用了一些重定向向量以允许某些应用程序地址参数发生改变。所有这些跳转向量必须与应用程序和 Bootloader 相匹配。在 Bootloader 部署到现场应用程序后，其中的代码将无法更改。未来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。

参数和资源

BootLdrUSBFSe 用户模块包含一个与全功能 USBFS 用户模块集成的 Bootloader 实用工具。

通过为 Bootloader 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，请从项目中清除有问题的文件，重新打开向导，选择 OK 或 APPLY，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器件的块大小进行定制，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。

图 3. 含 0x80 （128 字节）个模块的器件的默认参数 （针对 HID 支持选项）

ApplicationCode_Start_Block	0x2C
BootLoaderKey	0001020304050607
Fixed Pulse Width	ENABLE
Flash_Program_Temperature_deg_C	-20C
ICE_Debug_FLASH_DISABLE	Flash_WRITE_ENABLE
Bootload_when_CKSUM_fails	ENABLE_(deployment)
AcquisitionWindow	10

图 4. 含 0x40 （64 字节）个模块的器件的默认参数 （针对 HID 支持选项）

ApplicationCode_Start_Block	0x57
BootLoaderKey	0001020304050607
Fixed Pulse Width	ENABLE
Flash_Program_Temperature_deg_C	-20C
ICE_Debug_FLASH_DISABLE	Flash_WRITE_ENABLE
Bootload_when_CKSUM_fails	ENABLE_(deployment)
AcquisitionWindow	10

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。Bootloader 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 Bootloader 实用程序对应用程序校验和的自动验证操作。

BootloaderKey

这是加在发送给 Bootloader 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 Bootloader 升级实用工具不会意外使能。

默认值为 “0001020304050607”。

固定脉冲宽度

通过使能该选项可以使用基于 “Flash_Program_Temperature_deg_C” 的固定脉冲宽度。禁用该选项时，便可以通过调用 “BL_SetTemp(BYTE bTemp)” 函数在运行时间内设置编程温度。

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。在此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 Bootloader 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_ENABLE

此参数用于纠正在使能并运行 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 ”。

Bootload_when_CKSUM_fails

通常，如果在复位时应用程序校验和不正确，则 Bootloader 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 Bootloader 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使能校验和功能。

AcquisitionWindow

该变量表示在进行应用代码之前， PSoC 器件必须在启动过程中等待 Enter Bootloader （进入 Bootloader）指令的时间。其单位为毫秒。值范围从 0 到 255 个计时 （125 毫秒 / 计时）。

应用编程接口 （API）

使用本节介绍的应用编程接口 （API）可以对 BootLdrUSBFSe 用户模块进行编程控制。以下各节描述的是生成并集成描述符的情况。该节还列出了基本的和器件特定的 API 函数。作为开发人员，您需要了解 USB 协议基本内容，并熟悉 USB 2.0 规范 （尤其是第 9 章规范 “USB 器件框架 ”）。

BootLdrUSBFSe 用户模块支持控制传输、中断传输、批量传输和同步传输。可设计某些函数或函数组（例如 LoadInEP 和 EnableOutEP），以将它与批量和中断端点配合使用。其他函数 （如 BL_USBFS_LoadInISOCEP）则与同步端点一起使用。更多有关如何执行这些传输类型的信息，请参考技术参考手册 （TRM）。

注意：

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，因此与该用户模块一同提供的 API 不支持从中断执行这些例程。如果这是设计的需要，则请与赛普拉斯现场应用工程师联系。

对于大型储存器模型器件，调用程序需要保留 CUR_PP、 IDX_PP、 MVR_PP 以及 MVW_PP 等寄存器中的所有值。尽管一些寄存器现在不被修改，但是无法保证在将来的版本中也会如此。

Bootloader API

Bootloader 包含很有限的一组 API，这是因为 Bootloader 的主要目的是完全清除和替换用户应用程序。

ENTER_BOOTLOADER()

说明：

进入 Bootloader 应用，并在没有任何 Bootloader 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

BL_SetTemp （仅针对包含 64 字节闪存模块的器件）

说明：

该函数用于动态更新 Bootloader 的最后一次 die 温度测量值。在这种情况下，应用程序获得一个温度测量值并使用该函数将其传递到 Bootloader。然后，在发生引导加载事件时， Bootloader 根据通过该函数接收到的温度对闪存进行最佳的编程。

建议在运行期间定期测量 （或者确定）器件的 die 温度。每次测量 die 温度后，应通过该函数将其传递到 Bootloader。 Bootloader 在引导加载期间使用接收到的 die 温度更改闪存编程擦除和写入周期。这可优化闪存的保留时间和持续时间。因此，执行引导加载所需的时间随传递到 Bootloader 的温度值而定。

die 温度可通过调用器件片上温度传感器模块进行测量，或通过读取或测量其它一些外部器件或温度传感器进行测量。

该函数重写 “Flash_Program_Temperature_Deg_C” 用户模块参数设置的 Die 温度值。

C 原型：

```
void BL_SetTemp (CHAR cTemp);
```

汇编程序：

```
mov A, cTemp
lcall BL_SetTemp
```

示例代码：

```
void main(void)
{
```

```
CHAR cDieTemp = -20;  // Allocate a variable to hold the die temperature
// Use -20C as the default value
...
// Measure die temperature here and copy to cDieTemp variable
BL_SetTemp(cDieTemp); // Update Bootloader with real die temperature
ENTER_BOOTLOADER();  // Run the BootLoader
...
}
```

参数：

cTemp：Die 温度 （单位为摄氏度）。

返回值：

无

其它影响：

A 和 X 寄存器可能通过该函数的本次执行或以后执行修改。在大型存储器模型 （CY8C29xxx）中，所有 RAM 页面指针寄存器也会出现这种状况。如果需要，调用函数负责保留调用 fastcall16 函数时的各个值。

USBFS API

BootLdrUSBFSe 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 BL_USBFS_LoadInEP 和 BL_USBFS_EnableOutEP）设计为可用于批量和中断端点。其他函数 （如 BL_USBFS_LoadInISOCEP）则与同步端点一起使用。更多有关如何执行这些传输类型的信息，请参考技术参考手册 （TRM）。

下表列出了 USBFS 提供的 API 函数：

函数	说明
void USBFS_Start(BYTE bDevice, BYTE bMode)	激活用于器件和特定电压模式的用户模块。
void BL_USBFS_Stop(void)	禁用用户模块。
BYTE BL_USBFS_bCheckActivity(void)	检查并清除 USB 总线活动标志。如果从上次检查后 USB 保持活动状态，则返回 ‘1’ ；否则返回 ‘0’ 。
void USBFS_SetPowerStatus(BYTE bPowerStatus)	将器件设置为自供电或总线供电
BYTE BL_USBFS_bGetConfiguration(void)	返回当前分配的配置。如果未配置器件，则返回 0。
BYTE USBFS_bGetEPState(BYTE bEPNumber)	返回指定 USBFS 端点的当前状态。 2 = NO_EVENT_ALLOWED 1 = EVENT PENDING 0 = NO_EVENT_PENDING
BYTE USBFS_bGetEPState(BYTE bEPNumber)	通过返回非零值，识别是否已设置了 ACK。
BYTE USBFS_wGetEPCount(BYTE bEPNumber)	从指定 USBFS 端点返回当前字节计数。

函数	说明
void USBFS_LoadInEP(BYTE bEPNumber, BYTE *pData, WORD wLength, BYTE bToggle)	加载并使能用于 IN 传输的指定 USBFS 端点。
void USB_LoadInISOCEP(BYTE bEPNumber, BYTE *pData, WORD wLength, BYTE bToggle)	
BYTE USBFS_bReadOutEP(BYTE bEPNumber, BYTE *pData, WORD wLength)	从端点 RAM 读取指定的字节数量，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)	使能指定的 USB 端点以接受 OUT 传输。
void USB_EnableOutISOCEP(BYTE bEPNumber)	
void USBFS_DisableOutEP(BYTE bEPNumber)	禁用指定的 USB 端点，以否认 OUT 传输。

USBFS_Force(BYTE bState)	将 J、K 或 SE0 状态强制到 USB D+/D- 引脚上通常用于远程唤醒。
	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;`

人机界面 （HID）类支持 API：

函数	说明
BYTE USBFS_UpdateHIDTimer(BYTE bInterface)	更新指定接口的 HID 报告定时器，如果定时器到期，则返回 1，如果未到期，则返回 0。如果定时器到期，它将重新加载定时器。
BYTE USBFS_bGetProtocol(BYTE bInterface)	返回指定接口的协议。

BL_USBFS_Start （用户定义的应用器件）

说明：

执行 USBFS 用户模块的所有必需初始化。可以使用此指令启动前台 USB 器件或特定于 Bootloader 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 Bootloader 器件，请将 bDevice 的值设置为 -1 （0xFF）。

C 原型：

```
void BL_USBFS_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  BL_USBFS_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 的运行电压使能电压调节器，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	数值	说明
USB_3V_OPERATION	0x02	禁用电压调压器和上拉的通过 vcc
USB_5V_OPERATION	0x03	使能电压调压器并将它使用于上拉

返回值：

无

其它影响：

通过该函数的当前版本或后续版本，可以修改 A 和 X 寄存器。当前，仅修改了 IDX_PP 和 CUR_PP 页指针寄存器。

BL_USBFS_Stop

说明：

执行 USBFS 用户模块的所有必要的关闭任务。

C 原型：

```
void  BL_USBFS_Stop(void)
```

汇编：

```
lcall BL_USBFS_Stop
```

参数：

无

返回值：

无

其它影响：

通过该函数的当前版本或后续版本，可以修改 A 和 X 寄存器。当前，仅修改 CUR_PP 页面指针寄存器。

BL_USBFS_bCheckActivity

说明：

检查 USBFS 总线活动。

C 原型：

```
BYTE BL_USBFS_bCheckActivity(void)
```

汇编：

```
lcall BL_USBFS_bCheckActivity
```

参数：

无

返回值：

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

其他影响：

通过该函数的当前版本或后续版本，可以修改 A 和 X 寄存器。

BL_USBFS_bGetConfiguration

说明：

获取 USB 器件的当前配置。

C 原型：

```
BYTE BL_USBFS_bGetConfiguration(void)
```

汇编：

```
lcall BL_USBFS_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。

其他影响：

通过该函数的当前版本或后续版本，可以修改 A 和 X 寄存器。在大型存储器模型中，所有 RAM 页面指针寄存器都会出现这种状况。根据要求，所调用的函数要在调用 fastcall16 函数时保留各个值。当前，仅修改 CUR_PP 页面指针寄存器。

BL_USBFS_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。该端点可处在三种状态中的某一种，其中两种状态表示 IN 和 OUT 端点的不同情况。下表列出了各种可能的状态及其对于 IN 和 OUT 端点的含义。

C 原型：

```
BYTE BL_USBFS_bGetEPState(BYTE bEPNumber)
```

汇编：

```
mov A, 1                            ; Select endpoint 1
lcall BL_USBFS_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。当您编写代码以更改端点状态 （例如编写 ISR 代码以处理发送或接收的数据）时，使用这些常量。

状态	数值	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。

其他影响：

通过该函数的当前版本或后续版本，可以修改 A 和 X 寄存器。在大型存储器模型中，所有 RAM 页面指针寄存器都会出现这种状况。根据要求，所调用的函数要在调用 fastcall16 函数时保留各个值。当前，仅修改了 IDX_PP 页指针寄存器。

BL_USBFS_bGetEPAckState

说明：

通过读取端点控制寄存器中的 ACK 位，确定此端点上是否发生 ACK 数据操作。此函数不清除 ACK 位。

C 原型：

```
BYTE BL_USBFS_bGetEPAckState(BYTE bEPNumber)
```

汇编：

```
mov A, 1                        ; Select endpoint 1
lcall BL_USBFS_bGetEPAckState
```

参数：

寄存器 A 包含端点号。

返回值：

如果发生 ACK 数据操作，则此函数返回非零值。否则会返回零。

其他影响：

通过该函数的当前版本或后续版本，可以修改 A 和 X 寄存器。在大型存储器模型中，所有 RAM 页面指针寄存器都会出现这种状况。根据要求，所调用的函数要在调用 fastcall16 函数时保留各个值。

BL_USBFS_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。此函数先从计数减去二，然后才返回数值。只在调用 USB_GetEPState 后返回结果为 EVENT_PENDING 时，才为 OUT 端点调用此函数。

C 原型：

```
WORD BL_USBFS_wGetEPCount(BYTEbEPNumber)
```

汇编：

```
mov A, 1                            ; Select endpoint 1
lcall BL_USBFS_wGetEPCount
```

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。

其他影响：

通过该函数的当前版本或后续版本，可以修改 A 和 X 寄存器。在大型存储器模型中，所有 RAM 页面指针寄存器都会出现这种状况。根据要求，所调用的函数要在调用 fastcall16 函数时保留各个值。

BL_USBFS_LoadInEP

说明：

针对 IN 中断或批量传输 （.._LoadInEP）及同步传输 （..._LoadInISOCEP）加载和使能指定的 USB 端点。

C 原型：

```
void BL_USBFS_LoadInEP(BYTE bEPNumber, BYTE * pData, WORD wLength, BYTE bToggle)
void BL_USBFS_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 BL_USBFS_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	数值	说明
USB_NO_TOGGLE	0x00	数据切换没有变化
USB_TOGGLE	0x01	在传输前切换数据位

返回值：

无

其它影响：

通过该函数的当前版本或后续版本，可以修改 A 和 X 寄存器。在大型存储器模型中，所有 RAM 页面指针寄存器都会出现这种状况。根据要求，所调用的函数要在调用 fastcall16 函数时保留各个值。当前，仅修改了 IDX_PP 和 CUR_PP 页指针寄存器。

BL_USBFS_bReadOutEP

说明：

将指定数量的字节从端点 RAM 移动到数据 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 BL_USBFS_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 BL_USBFS_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。

其他影响：

通过该函数的当前版本或后续版本，可以修改 A 和 X 寄存器。在大型存储器模型中，所有 RAM 页面指针寄存器都会出现这种状况。根据要求，所调用的函数要在调用 fastcall16 函数时保留各个值。当前，仅修改了 IDX_PP 和 CUR_PP 页指针寄存器。

BL_USBFS_EnableOutEP

说明：

使能 OUT 批量或中断传输 （..._EnableOutEP）及同步传输 （..._EnableOutISOCEP）的指定端点。请勿为 IN 端点调用这些函数。

C 原型：

```
void BL_USBFS_EnableOutEP(BYTE bEPNumber)
void BL_USBFS_EnableOutISOCEP(BYTE bEPNumber)
```

汇编：

```
mov A, 1
lcall BL_USBFS_EnableOutEP
```

参数：

寄存器 A 包含端点号。

返回值：

无

其它影响：

通过该函数的当前版本或后续版本，可以修改 A 和 X 寄存器。当前，仅修改了 IDX_PP 页指针寄存器。

BL_USBFS_DisableOutEP

说明：

禁用指定的 USBFS OUT 端点。不要为 IN 端点调用此函数。

C 原型：

```
void BL_USBFS_DisableOutEP(BYTE bEPNumber)
```

汇编：

```
mov A, 1                            ; Select endpoint 1
lcall BL_USBFS_DisableOutEP
```

参数：

寄存器 A 包含端点号。

返回值：

无

其它影响：

通过该函数的当前版本或后续版本，可以修改 A 和 X 寄存器。在大型存储器模型中，所有 RAM 页面指针寄存器都会出现这种状况。根据要求，所调用的函数要在调用 fastcall16 函数时保留各个值。

BL_USBFS_Force

说明：

将 USB J、K 或 SE0 状态强制到 D+/D- 线上。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。欲了解有关挂起和恢复功能的详细信息，请参见 USB 2.0 规范。

C 原型：

```
void BL_USBFS_Force(BYTE bState)
```

汇编：

```
mov A, BL_USB_FORCE_K
lcall BL_USBFS_Force
```

参数：

bState 字节用于指示要使能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：

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

返回值：

无

其它影响：

通过该函数的当前版本或后续版本，可以修改 A 和 X 寄存器。在大型存储器模型中，所有 RAM 页面指针寄存器都会出现这种状况。根据要求，所调用的函数要在调用 fastcall16 函数时保留各个值。

BL_USBFS_UpdateHIDTimer

说明：

更新 HID 报告闲置定时器并返回到期状态。如果定时器到期，则重新加载定时器。

C 原型：

```
BYTE BL_USBFS_UpdateHIDTimer(BYTE bInterface)
```

汇编：

```
mov A, 1                            ; Select interface 1
lcall BL_USBFS_UpdateHIDTimer
```

参数：

寄存器 A 包含接口编号。

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	数值	说明
USB_IDLE_TIMER_EXPIRED	0x01	定时器已到期。
USB_IDLE_TIMER_RUNNING	0x02	定时器正在运行。
USB_IDLE_TIMER_IDEFINITE	0x00	如果在数据或状态更改时发送报告，则返回该值。

其他影响：

通过该函数的当前版本或后续版本，可以修改 A 和 X 寄存器。

BL_USBFS_bGetProtocol

说明：

返回所选接口的 HID 协议值。

C 原型：

```
BYTE BL_USBFS_bGetProtocol(BYTE bInterface)
```

汇编：

```
mov A, 1                              ; Select interface 1
lcall BL_USBFS_bGetProtocol
```

参数：

bInterface 包含接口编号。

返回值：

寄存器 A 包含协议值。

其他影响：

通过该函数的当前版本或后续版本，可以修改 A 和 X 寄存器。在大型存储器模型中，所有 RAM 页面指针寄存器都会出现这种状况。根据要求，所调用的函数要在调用 fastcall16 函数时保留各个值。

BL_USBFS_SetPowerStatus

说明：

设置当前电源状态。将电源状态设置为 1 表示自供电，或设置为 0 表示总线供电。器件将根据该值应答 USB GET_STATUS 请求。这允许器件针对 USB 第 9 章的合规性正确地报告其状态。器件可以随时将其电源从自供电更改为总线供电，并将其当前电源作为器件状态的一部分而报告。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 BL_USBFS_SetPowerStatus(BYTE bPowerStaus);
```

汇编：

```
mov   A, 1                             ; Select self powered
lcall  BL_USBFS_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	数值	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

其它影响：

通过该函数的当前版本或后续版本，可以修改 A 和 X 寄存器。在大型存储器模型中，所有 RAM 页面指针寄存器都会出现这种状况。根据要求，所调用的函数要在调用 fastcall16 函数时保留各个值。

USB-UART API

诸多嵌入式应用通过使用 RS-232 接口与外部系统 （如 PC）进行通信，特别是在调试过程中使用地更加广泛。但是，在 PC 领域，新型电脑很快将不再使用 RS-232 COM 端口，而会使用 USB 来代替进行串行通信。将器件移植到 USB 的最简单方法是通过 USB 总线模拟 RS-232。该方法的主要优点为 PC 应用将 USB 连接作为 RS-232 COM 连接使用，使之易于调试。该方法使用了 Windows 98SE 中所有 Microsoft® Windows 版本所附带的标准 Windows® 驱动程序。

USB 通信设备类 （CDC）规范定义了多种通信模型，其中包括了通过 USB 进行串联仿真的抽象控制模型，如第 3.6.2.1 节中所述的情况。更多详细信息，请参考 CDC 规范版本 1.1。微软 Windows USB 解调器的驱动程序 （即 usbser.sys）符合该规范。

当首次将新的器件连接到 Windows PC 时， Windows 要求提供驱动程序。需要使用 INF 文件来安装驱动程序。微软 Windows 并没有提供 usbser.sys 驱动程序的标准 INF 文件。为了安装通过 USB 模拟 RS-232 的器件，必须提供一个 INF 文件，以将连接器件映射到微软 CDC 驱动程序内。 USBUART 项目的必要 INF 文件自动被生成并位于项目 LIB 文件夹中。提供了 INF 文件后，驱动程序将 USB 器件枚举为 COM 端口。

因为是 USB 器件，并且使用 USB 协议进行控制数据流，因此终端应用中的设置内容 （波特率、数据位、奇偶校验位、停止位和流控制）不会对数据传输的性能产生任何影响。但是，如果需要，可以通过调用特定的 API 检索 RS-232 器件的设置内容 （除流控制外）。由于微软 CDC 驱动程序 （usbser.sys）并不支持流控制，所以无法检索流控制设置。

可通过调用下面 API 来检索特定设置：

- BL_UART_dwGetDTERate
- BL_UART_bGetCharFormat
- BL_UART_bGetParityType
- BL_UART_bGetDataBits
- BL_UART_bGetLineControlBitmap

发送长度为64 个字节或更长的数据包

在用户模块中， USB 端点的缓冲区大小被设置为 64 个字节，用于输入和输出数据。这样，所有读写 API 只能处理不大于 64 个字节的缓冲区长度。

另外，还限制了必须发送整 64 个字节的数据。如果有效载荷小于 64 字节或传输零长度的数据包，则 PC 驱动程序认为已经完全接收了数据包。如果接收到所有 64 个字节的数据， PC 驱动程序假定还没有接收完所有数据，并立即要求发送下一个数据包。调用函数后，下面的代码会立即将成功传输数据状态发送给 PC：

```
while (!BL_UART_bTxIsReady());
BL_UART_Write(pData, 61); //Length is less than 64 data appear on
//terminal application after this function is executed.
```

下面的代码显示的是如何发送整 64 个字节数据的示例：

```
while (!BL_UART_bTxIsReady());
BL_UART_Write(pData, 64); //Length is equal to 64
while (!BL_UART_bTxIsReady());
BL_UART_Write(pData, 0); //No actual data transfer, but zero-length packet
// indicates PC driver that data is completely received.
// 64 bytes of data appears on terminal application
// after this function is executed.
```

请注意，依次发送 64 个字节长的数据包和零长度的数据包比发送两个 32 字节长的数据包更快。这是因为整 64 个字节长的数据包会强制驱动程序继续传输。

下面代码显示的是传输 150 个字节的示例：

```
while (!BL_UART_bTxIsReady());
BL_UART_Write(pData, 64); //Length is equal to 64
while (!BL_UART_bTxIsReady());
BL_UART_Write(pData, 64); //Length is equal to 64. 128 bytes is transferred to
// PC. No data appears on terminal application at the
// moment
while (!BL_UART_bTxIsReady());
BL_UART_Write(pData, 2); //Two bytes transferred. Full 130 bytes-length packet
//appears on terminal application after this function
// is executed.
```

下表列出了 USB-UART 提供的 API 函数：

函数	说明
BOOL BL_UART_Init(void)	初始化 USB-UART 模块。如果成功初始化 USB-UART，将返回非零值。
void BL_UART_Write(BYTE * pData, BYTE bLength)	将 bLength 字节从 pData 阵列发送给 PC。
void BL_UART_CWrite(const BYTE * pData, BYTE bLength)	将 bLength 字节从常量 （ROM） pData 阵列发送给 PC。
void BL_UART_PutString(BYTE * pStr)	将空结尾字符串从 pStr 发送给 PC。
void BL_UART_CPutString(const BYTE * pStr)	将常量 （ROM）空结尾字符串从 pStr 发送给 PC。

函数	说明
void BL_UART_PutChar(BYTE bChar)	将一个字符发送给 PC
void BL_UART_PutCRLF(void)	将回车符 （0x0D）和换行符 （0x0A）发送给 PC。
void BL_UART_PutSHexByte(BYTE bValue)	将十六进制表示的两字符 bValue 发送给 PC。
void BL_UART_PutSHexInt(INT iValue)	将十六进制表示的四字符 iValue 发送给 PC。
BYTE BL_UART_bGetRxCount(void)	返回当前可读取的字节数量。
BYTE BL_UART_bTxIsReady(void)	如果 USBUART 已准备好发送数据，则返回非零值。
BYTE BL_UART_Read(BYTE * pData, BYTE bLength)	从 RX 缓冲区中读取所指定的字节数量，并将其放置在 pData 指定的 RAM 阵列中。函数返回 RX 缓冲区中剩余的字节数量和操作状态。
void BL_UART_ReadAll(BYTE * pData)	从 RX 缓冲区中读取所有可用的数据，并将其放置在 pData 指定的 RAM 阵列中。
WORD BL_UART_ReadChar(void)	返回从 RX 缓冲区中返回值的最低有效字节 （LSB）。该函数还返回操作状态和 RX 缓冲区中返回值剩余的最高有效字节。
DWORD *BL_UART_dwGetDTERate(DWORD * dwDTERate)	返回该端口所设置的数据终端速度 （单位为每秒位数）
BYTE BL_UART_bGetCharFormat(void)	返回停止位数量。
BYTE BL_UART_bGetParityType(void)	返回奇偶校验类型。
BYTE BL_UART_bGetDataBits(void)	返回数据位数量。
BYTE BL_UART_bGetLineControlBitmap(void)	返回 DTE 和 RTS 信号状态。
void BL_UART_SendStateNotify(BYTE bState)	将有关当前 UART 状态的通知发送给 PC。

BL_UART_Init

说明：

初始化 USB-UART 器件并设置该模块与 PC 的通信。

C 原型：

```
BOOL  BL_UART_Init(void)
```

汇编：

```
lcall  BL_UART_Init
```

参数：

无

返回值：

如果器件成功初始化，向累加器返回非零值。否则，将返回 0 值。仅在成功初始化后，用户模块才可操作。

其他影响：

通过该函数的当前版本或后续版本，可以修改 A 和 X 寄存器。在大型存储器模型中，所有 RAM 页面指针寄存器也会出现这种状况。如果需要，调用函数负责保留调用 fastcall16 函数时的各个值。当前，仅修改了 IDX_PP 和 CUR_PP 页指针寄存器。

BL_UART_Write

说明：

将 bLength 字符从 （RAM）指针 pData 指定的位置发送给 PC。发送大型数据包时，请参考发送长度为 64 字节或更长的数据包中的内容。

C 原型：

```
void  BL_UART_Write(BYTE * pData, BYTE bLength)
```

汇编：

```
mov   A,20              ; Load array count
push  A
mov   A,>pData          ; Load MSB part of pointer to RAM string
push  A
mov   A,<pData          ; Load LSB part of pointer to RAM string
push  A
lcall  BL_UART_Write     ; Make call to function
add   SP,253            ; Reset stack pointer to original position
```

参数：

pData 为指向数据阵列的指针。数据阵列的最大长度为 64 个字节。

bLength 为从阵列发送给 PC 的字节数量。有效值范围介于 0 至 64 之间。

返回值：

无

其它影响：

通过该函数的当前版本或后续版本，可以修改 A 和 X 寄存器。在大型存储器模型中，所有 RAM 页面指针寄存器也会出现这种状况。如果需要，调用函数负责保留调用 fastcall16 函数时的各个值。当前，仅修改了 IDX_PP 和 CUR_PP 页指针寄存器。

BL_UART_CWrite

说明：

将 bLength 字符从 （ROM）指针 pData 指定的位置发送给 PC。发送大型数据包时，请参考发送长度为 64 字节或更长的数据包中的内容。

C 原型：

```
void  BL_UART_CWrite(const BYTE * pData, BYTE bLength)
```

汇编：

```
mov   A,20              ; Load array count
push  A
mov   A,>pData          ; Load MSB part of pointer to ROM string
push  A
mov   A,<pData          ; Load LSB part of pointer to ROM string
```

```
push  A
lcall  BL_UART_CWrite    ; Make call to function
add   SP,253            ; Reset stack pointer to original position
```

参数：

pData 为指向 ROM 数据阵列的指针。数据阵列的最大长度为 64 个字节。

bLength 为从阵列发送给 PC 的字节数量。有效值范围介于 0 至 64 之间。

返回值：

无

其它影响：

通过该函数的当前版本或后续版本，可以修改 A 和 X 寄存器。在大型存储器模型中，所有 RAM 页面指针寄存器也会出现这种状况。如果需要，调用函数负责保留调用 fastcall16 函数时的各个值。当前，仅修改了 IDX_PP 和 CUR_PP 页指针寄存器。

BL_UART_PutString

说明：

将空结尾 （RAM）字符串发送给 PC。发送大型数据包时，请参考发送长度为 64 字节或更长的数据包中的内容。

C 原型：

```
void BL_UART_PutString(BYTE * pStr)
```

汇编程序：

```
mov  A,>pStr            ; Load MSB part of pointer to RAM based null
                        ; terminated string
mov  X,<pStr            ; Load LSB part of pointer to RAM based null
                        ; terminated string
lcall BL_UART_PutString  ; Call function to send string out
```

参数：

pStr：指向要发送给 PC 的字符串的指针。最高有效字节被传递到 X 寄存器中，而最低有效字节被传递到累加器中。字符串的最大长度为 64 字节，包括空结尾字符。

返回值：

无

其它影响：

通过该函数的当前版本或后续版本，可以修改 A 和 X 寄存器。在大型存储器模型中，所有 RAM 页面指针寄存器也会出现这种状况。如果需要，调用函数负责保留调用 fastcall16 函数时的各个值。当前，仅修改了 IDX_PP 和 CUR_PP 页指针寄存器。

BL_UART_CPutString

说明：

将空结尾 （ROM）字符串发送给 PC。发送大型数据包时，请参考发送长度为 64 字节或更长的数据包中的内容。

C 原型：

```
void BL_UART_CPutString(const BYTE * pStr)
```

汇编程序：

```
mov  A,>pStr            ; Load MSB part of pointer to ROM based null
                        ; terminated string
mov  X,<pStr            ; Load LSB part of pointer to ROM based null
                        ; terminated string
lcall BL_UART_PutString  ; Call function to send string out
```

参数：

pStr：指向要发送给 PC 的字符串的指针。最高有效字节被传递到 X 寄存器中，而最低有效字节被传递到累加器中。字符串的最大长度为 64 字节，包括空结尾字符。

返回值：

无

其它影响：

通过该函数的当前版本或后续版本，可以修改 A 和 X 寄存器。在大型存储器模型中，所有 RAM 页面指针寄存器也会出现这种状况。如果需要，调用函数负责保留调用 fastcall16 函数时的各个值。当前，仅修改了 IDX_PP 和 CUR_PP 页指针寄存器。

BL_UART_PutChar

说明：

将单字符写入 PC 内。

C 原型：

```
void BL_UART_PutChar(BYTE bChar)
```

汇编程序：

```
mov  A,0x33            ; Load ASCII character "3" in A
lcall BL_UART_PutChar   ; Call function to send single character to PC
```

参数：

bChar：将要发送到 PC 的字符。数据被传递给累加器。

返回值：

无

其它影响：

通过该函数的当前版本或后续版本，可以修改 A 和 X 寄存器。在大型存储器模型中，所有 RAM 页面指针寄存器也会出现这种状况。如果需要，调用函数负责保留调用 fastcall16 函数时的各个值。当前，仅修改了 IDX_PP 和 CUR_PP 页指针寄存器。

BL_UART_PutCRLF

说明：

将回车符 （0x0D）和换行符 （0x0A）发送给 PC。

C 原型：

```
void BL_UART_PutCRLF(void)
```

汇编程序：

```
lcall BL_UART_PutCRLF       ; Send a carriage return and line feed out
```

参数：

无

返回值：

无

其它影响：

通过该函数的当前版本或后续版本，可以修改 A 和 X 寄存器。在大型存储器模型中，所有 RAM 页面指针寄存器也会出现这种状况。如果需要，调用函数负责通过调用 fastcall16 函数保留寄存器的值。当前，仅修改了 IDX_PP 和 CUR_PP 页指针寄存器。

BL_UART_PutSHexByte

说明：

将 ASCII 十六进制表示的两个数据字节发送给 PC。

C 原型：

```
void BL_UART_PutSHexByte(BYTE bValue)
```

汇编程序：

```
mov  A,0x33               ; Load data to be sent
lcall BL_UART_PutSHexByte  ; Call function to output hex representation of
                          ; data. The output for this value would be "33".
```

参数：

bValue：需要转换为 ASCII 字符串 （十六进制表示）的字节。数据被传递给累加器。

返回值：

无

其它影响：

通过该函数的当前版本或后续版本，可以修改 A 和 X 寄存器。在大型存储器模型中，所有 RAM 页面指针寄存器也会出现这种状况。如果需要，调用函数负责保留调用 fastcall16 函数时的各个值。当前，仅修改了 IDX_PP 和 CUR_PP 页指针寄存器。

BL_UART_PutSHexInt

说明：

将 ASCII 十六进制表示的四个数据字节发送给 PC。

C 原型：

```
void BL_UART_PutSHexInt(INT iValue)
```

汇编程序：

```
mov  A,0x34           ; Load LSB in A
mov  X,0x12           ; Load MSB in X
lcall BL_UART_PutSHexInt ; Call function to output hex representation of data.
                      ; The output for this value would be "1234".
```

参数：

iValue：需要转换为 ASCII 字符串 （十六进制表示）的整数。最高有效字节被传递到 X 寄存器中，而最低有效字节被传递到累加器中。

返回值：

无

其它影响：

通过该函数的当前版本或后续版本，可以修改 A 和 X 寄存器。在大型存储器模型中，所有 RAM 页面指针寄存器也会出现这种状况。如果需要，调用函数负责保留调用 fastcall16 函数时的各个值。当前，仅修改了 IDX_PP 和 CUR_PP 页指针寄存器。

BL_UART_bGetRxCount

说明：

该函数返回从 PC 接收到，且在 RX 缓冲区中等待的字节数数量。

C 原型：

```
BYTE  BL_UART_bGetRxCount(void)
```

汇编：

```
lcall  BL_UART_bGetRxCount
```

参数：

无

返回值：

返回 A 寄存器中的当前字节数量。

其他影响：

通过该函数的当前版本或后续版本，可以修改 A 和 X 寄存器。在大型存储器模型中，所有 RAM 页面指针寄存器也会出现这种状况。如果需要，调用函数负责保留调用 fastcall16 函数时的各个值。当前，仅修改了 IDX_PP 页指针寄存器。

BL_UART_bTxIsReady

说明：

如果 TX 缓冲区准备好发送更多数据，则返回非零值。否则，将返回 0 值。

C 原型：

```
BYTE  BL_UART_bTxIsReady(void)
```

汇编：

```
lcall  BL_UART_bTxIsReady
```

参数：

无

返回值：

如果 TX 缓冲区可以接受数据，该函数将返回非零值，否则将返回零。

其他影响

通过该函数的当前版本或后续版本，可以修改 A 和 X 寄存器。在大型存储器模型中，所有 RAM 页面指针寄存器也会出现这种状况。如果需要，调用函数负责保留调用 fastcall16 函数时的各个值。

BL_UART_Read

说明：

从 RX 缓冲区中读取 bLength 字节数据，并将其放置在 pData 指定的数据阵列中。

C 原型：

```
BYTE  BL_UART_Read(BYTE * pData, BYTE bLength)
```

汇编：

```
mov A, 25              ; Load count
push A
mov A, >pData          ; Load MSB part of pointer to RAM array
push A
mov A, <pData          ; Load LSB part of pointer to RAM array
push A
lcall  BL_UART_Read
```

参数：

pData 为指向数据阵列的指针。数据阵列的最大长度为 64 个字节。

bLength 为需要读取并放置在阵列中的字节数量。有效值范围介于 0 至 64 之间。

返回值：

通过累加器中的位 0 到位 6 返回 RX 缓冲区中剩余的字节数量；累加器的 MSB （位 7）指示错误状态。通常，当您请求比缓冲区中可用字节数更多的数量，会发生此错误状态。 RX 缓冲区中的数据被放置在 pData 指定的数据阵列。

其他影响：

通过该函数的当前版本或后续版本，可以修改 A 和 X 寄存器。在大型存储器模型中，所有 RAM 页面指针寄存器也会出现这种状况。如果需要，调用函数负责保留调用 fastcall16 函数时的各个值。当前，仅修改了 IDX_PP 和 CUR_PP 页指针寄存器。

BL_UART_ReadAll

说明：

从 RX 缓冲区中读取所有收到的字节，并将其放置在 pData 指定的数据阵列中。

C 原型：

```
void  BL_UART_ReadAll(BYTE * pData)
```

汇编：

```
mov  A,>pData           ; Load MSB part of pointer to RAM buffer
mov  X,<pData           ; Load LSB part of pointer to RAM buffer
lcall  BL_UART_ReadAll
```

参数：

pData 为指向数据阵列的指针。最高有效字节被传递到 X 寄存器中，而最低有效字节被传递到累加器中。数据阵列的最大长度为 64 个字节。

返回值：

无

其它影响：

通过该函数的当前版本或后续版本，可以修改 A 和 X 寄存器。在大型存储器模型中，所有 RAM 页面指针寄存器也会出现这种状况。如果需要，调用函数负责保留调用 fastcall16 函数时的各个值。当前，仅修改了 IDX_PP 和 CUR_PP 页指针寄存器。

BL_UART_ReadChar

说明：

读取从 RX 缓冲区中接收到的一个数据字节。

C 原型：

```
WORD  BL_UART_ReadChar(void)
```

汇编：

```
lcall  BL_UART_ReadChar
```

参数：

无

返回值：

从累加器的返回值的 MSB 包含 RX 缓冲区中剩余的字节数量 （使用位 0 到位 6 来表示）。位 7 指示错误状态。调用该函数时，如果缓冲区为空白，位 7 被设为 1。返回值的 LSB （从 X 寄存器）包含缓冲区的一个字符。

其他影响：

通过该函数的当前版本或后续版本，可以修改 A 和 X 寄存器。在大型存储器模型中，所有 RAM 页面指针寄存器也会出现这种状况。如果需要，调用函数负责保留调用 fastcall16 函数时的各个值。当前，仅修改了 IDX_PP 和 CUR_PP 页指针寄存器。

BL_UART_dwGetDTERate

说明：

返回该端口所设置的数据终端速度 （单位为每秒位数）向函数传递指向 DWORD 的指针。该函数返回指针所参照的位置上的 DTE 率。

C 原型：

```
DWORD * BL_UART_dwGetDTERate(DWORD * dwDTERate)
```

汇编：

```
mov  A,>dwDTERate       ; Load MSB part of pointer
mov  X,<dwDTERate       ; Load LSB part of pointer
lcall  BL_UART_dwGetDTERate
```

参数：

dwDTERate：指向该函数返回值时， DTE 率所在的位置的指针。

返回值：

保存由所传递指针参照的位置上 DWORD 值的 DTE 率，然后返回指向该位置的指针。

其他影响：

通过该函数的当前版本或后续版本，可以修改 A 和 X 寄存器。在大型存储器模型中，所有 RAM 页面指针寄存器也会出现这种状况。如果需要，调用函数负责保留调用 fastcall16 函数时的各个值。当前，仅修改了 IDX_PP 和 CUR_PP 页指针寄存器。

BL_UART_bGetCharFormat

说明：

返回停止位数量。

C 原型：

```
BYTE  BL_UART_bGetCharFormat(void)
```

汇编：

```
lcall  BL_UART_bGetCharFormat
```

参数：

无

返回值：

返回累加器中的停止位数量。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	数值	说明
USBUART_1_STOPBITS	0x00	一个停止位
USBUART_1_5_STOPBITS	0x01	一个半停止位
USBUART_2_STOPBITS	0x02	两个停止位

其他影响：

通过该函数的当前版本或后续版本，可以修改 A 和 X 寄存器。在大型存储器模型中，所有 RAM 页面指针寄存器也会出现这种状况。如果需要，调用函数负责保留调用 fastcall16 函数时的各个值。当前，仅修改 CUR_PP 页指针寄存器。

BL_UART_bGetParityType

说明：

返回奇偶校验类型。

C 原型：

```
BYTE  BL_UART_bGetParityType(void)
```

汇编：

```
lcall  BL_UART_bGetParityType
```

参数：

无

返回值：

返回累加器中的奇偶校验类型。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	数值	说明
USBUART_PARITY_NONE	0x00	无奇偶校验
USBUART_PARITY_ODD	0x01	奇校验
USBUART_PARITY_EVEN	0x02	偶校验
USBUART_PARITY_MARK	0x03	标记奇偶校验
USBUART_PARITY_SPACE	0x04	空格奇偶校验

其他影响：

通过该函数的当前版本或后续版本，可以修改 A 和 X 寄存器。在大型存储器模型中，所有 RAM 页面指针寄存器也会出现这种状况。如果需要，调用函数负责保留调用 fastcall16 函数时的各个值。当前，仅修改 CUR_PP 页指针寄存器。

BL_UART_bGetDataBits

说明：

返回数据位数量。

C 原型：

```
BYTE  BL_UART_bGetDataBits(void)
```

汇编：

```
lcall  BL_UART_bGetDataBits
```

参数：

无

返回值：

返回累加器中的数据位数量。数据位数量可以是 5、 6、 7、 8 或 16。

其他影响：

通过该函数的当前版本或后续版本，可以修改 A 和 X 寄存器。在大型存储器模型中，所有 RAM 页面指针寄存器也会出现这种状况。如果需要，调用函数负责保留调用 fastcall16 函数时的各个值。当前，仅修改 CUR_PP 页指针寄存器。

BL_UART_bGetLineControlBitmap

说明：

返回包含 RS-232 式控制信号状态的位图。

C 原型：

```
BYTE  BL_UART_bGetLineControlBitmap(void)
```

汇编：

```
lcall  BL_UART_bGetLineControlBitmap
```

参数：

无

返回值：

返回包含累加器中控制信号状态的位图。可以单独处理位图中的每一位。位 D7..D2 被保留。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	数值	说明
USBUART_RTS	0x02	RTS （1：激活载波； 0：取消激活载波）
USBUART_DTR	0x01	DTR （1：有； 0：无）

其他影响：

通过该函数的当前版本或后续版本，可以修改 A 和 X 寄存器。在大型存储器模型中，所有 RAM 页面指针寄存器也会出现这种状况。如果需要，调用函数负责保留调用 fastcall16 函数时的各个值。当前，仅修改 CUR_PP 页指针寄存器。

BL_UART_SendStateNotify

说明：

将有关当前 UART 状态的通知发送给 PC。

注意： 微软 usbser.sys 驱动程序并非支持这些信号。

C 原型：

```
void  BL_UART_SendStateNotify(BYTE bState)
```

汇编：

```
mov  A, (USBUART_DCD + USBUART_DSR)
lcall  BL_UART_SendStateNotify
```

参数：

bState 位图包含累加器中控制信号的状态。可以单独处理位图中的每一位。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	数值	说明
USBUART_DCD	0x01	RS-232 DCD 信号
USBUART_DSR	0x02	RS-232 DSR 信号
USBUART_BREAK	0x04	断线检测机制的状态
USBUART_RING	0x08	环检测信号的状态。
USBUART_FRAMING_ERR	0x10	发生了帧错误。
USBUART_PARITY_ERR	0x20	发生了奇偶校验错误。
USBUART_OVERRUN	0x40	由于溢出，接收到的数据被删除。

返回值：

无

其它影响：

通过该函数的当前版本或后续版本，可以修改 A 和 X 寄存器。在大型存储器模型中，所有 RAM 页面指针寄存器也会出现这种状况。如果需要，调用函数负责保留调用 fastcall16 函数时的各个值。当前，仅修改了 IDX_PP 和 CUR_PP 页指针寄存器。

固件源代码示例

HID 器件

1. 新建一个包含 BootLdrUSBFSe 用户模块 （如 CY8C24894）支持的基本器件的项目。
2. 在用户模块目录中点击 Protocols （协议）。通过双击 BootLdrUSBFSe 图标或右键单击并选中 Place，可以添加 BootLdrUSBFSe 用户模块。
3. 选择人体界面器件 （HID）单选按键并点击 OK。
4. 按照下图配置 BootLdrUSBFSe 用户模块。默认配置可以不同，但仍适用于已给的示例代码。

ApplicationCode_Start_Block	0x57
BootLoaderKey	0001020304050607
Fixed Pulse Width	ENABLE
Flash_Program_Temperature_deg_C	-20C
ICE_Debug_FLASH_DISABLE	Flash_WRITE_ENABLE
Bootload_when_CKSUM_fails	ENABLE_(deployment)
AcquisitionWindow	10

5. 确保将正确的引脚配置为下拉驱动，以识别开关闭合，进入 Bootloader。示例取决于您可以按的按键。此按键拉高 Port1_7，并导致程序作为 Bootloader 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。

⊞ P1[6]	Port_1_6, StdCPU, High Z Analog, DisableInt, Normal, 0
⊟ P1[7]	Port_1_7, StdCPU, Pull Down, DisableInt, Normal, 0
Name	Port_1_7
Port	P1[7]
Select	StdCPU
Drive	Pull Down
Interrupt	DisableInt
AnalogMUXBus	Normal
InitialValue	0
⊞ P2[0]	Port_2_0, StdCPU, High Z Analog, DisableInt, Normal, 0

6. 右键单击用户模块图标。选择 Boot Loader Tools。然后选择 Get Files。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。
7. 右键单击用户模块图标。选择 **Device: Application USB Setup Wizard…**
   - 单击 “ 导入 HID 报告模板 ” 操作，并将名称更改为 “ 导入 HID 报告模板 ” （斜体），以表示它是一个标签。
   - 选择 3 按键鼠标模板。
   - 点击模板右侧的 Apply （应用）操作。
   - 选择 Add String （添加字符串）操作，以添加 Manufacturer （制造商）和 Product （产品）字符串。
   - 编辑器件属性：供货商 ID、产品 ID，并选择相应的字符串。

- 编辑接口属性：将 Class （类别）字段设置为 “HID”。
- 编辑 HID 类描述符：为 HID Report （HID 报告）字段选择 3 按键鼠标。
- 单击 OK 保存 USB 描述符的信息。

更多详细说明请参见下表。

8. 右键单击用户模块图标。选择 **Device: Bootloader USB Setup Wizard…** 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 Bootloader 的 VID 和 PID 不能相同。由于所包含的主机 PC 示例项目具有供货商 ID （VID）04b4 和产品 ID （PID）E006 （赛普拉斯所拥有的 ID），因此可用于本地调试，但不可发布以用于生产。单击 OK 以保存 USB Bootloader 描述符信息。
9. 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。

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

  W  W  U  U  U   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   W   W   W ;    Base Address 1000
  W  W  W  W  W   W   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
```

10. 生成应用。
11. 复制样本代码并粘贴它。
12. 执行重建全部。

描述符	数据
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

描述符	数据
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
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	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	不支持
HID 报告	3 按键鼠标
端点描述符	ENDPOINT_NAME
端点属性	
端点号	1
方向	输入
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

描述符	数据
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

示例代码

此处所示的代码向您说明如何在简单 HID 应用程序中使用 BootLdrUSBFSe 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，鼠标光标在一个正方形中移动。此代码演示 BootLdrUSBFSe 设置向导如何配置用户模块。

```
//
//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 boxLoop = 0;               // Box loop counter

void main(void)
{
    M8C_EnableGInt;                       //Enable Global Interrupts
    BL_USBFS_Start(0, USB_5V_OPERATION); //Start USB Operation using device 0

    PRT1DR = 0;
    while(1)     // Main loop
{
       if (PRT1DR & 0x80)
        {
```

```
            BL_USBFS_Stop();
            while(PRT1DR & 0x80);
            ENTER_BOOTLOADER();
        }

        switch(bConfigState)    // Check state
        {
        case USB_INIT:          // Initialize state
            bConfigState = USB_UNCONFIG;
            break;
        case USB_UNCONFIG:      // Unconfigured state
            if(BL_USBFS_bGetConfiguration() != 0)
            {     // Check if configuration set
                bConfigState = USB_CONFIG;
                // Load a dummy mouse packet
                BL_USBFS_LoadInEP(1, abMouseData, 3, USB_NO_TOGGLE);
            }
            break;
        case USB_CONFIG:        // Configured state time to move the mouse
            if(BL_USBFS_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;
                    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
                BL_USBFS_LoadInEP(1, abMouseData, 3, USB_TOGGLE); // Load and toggle
Endpoint1
            }     // End if Endpoint ready
            break;
        }         // End Switch
```

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

以下是 BootLdrUSBFSe 用户模块的汇编语言示例代码。

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 BootLdrUSBFSe 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 BootLdrUSBFSe 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 Bootloader。

```
;-------------------------------------------------------------------------
; 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

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:
    M8C_EnableGInt
    ; Start USBFS Operation using device 0
    mov X, USB_5V_OPERATION
    mov A, 0
    lcall BL_USBFS_Start

    ; Wait for Device to enumerate
.no_device:
    lcall BL_USBFS_bGetConfiguration
    cmp A, 0
    jz .no_device
    ; Enumeration is completed load endpoint 1. Do not toggle the first time
    ; BL_USBFS_LoadInEP(1, abMouseData, 3, USB_NO_TOGGLE);
    mov A, USB_NO_TOGGLE
    push A
    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 BL_USBFS_LoadInEP
    add SP, -6
```

```
.endless_loop:

;implement bootloader entry
    mov reg[PRT1DR], 0   ;load reg[PRT1DR] with  0
    mov A, reg[PRT1DR]
    and A, 0x80
    jz .Exit_BOOTLOAD_TEST
    lcall BL_USBFS_Stop
.wait_for_bit_low:
    tst reg[PRT1DR], 0x80
    jnz .wait_for_bit_low
    ; once it goes low enter the bootloader
    ljmp ENTER_BOOTLOADER ;;never returns
    halt
.Exit_BOOTLOAD_TEST:

;;; mouse operations

    mov A, 1
    lcall BL_USBFS_bGetEPAckState
    cmp A, 0
    jz .endless_loop
    ; ACK has occurred, load the endpoint and toggle the data bit
    ; BL_USBFS_LoadInEP(1, abMouseData, 3, USB_TOGGLE);
    mov A, USB_TOGGLE
    push A
    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 BL_USBFS_LoadInEP
    add SP, -6

; 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
```

USB-UART （CDC）

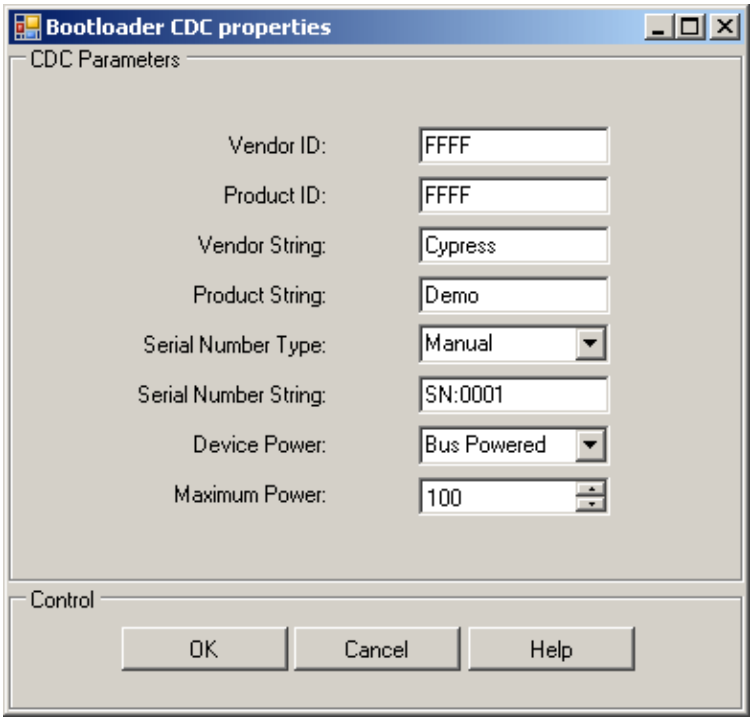
1. 新建一个包含基本器件的项目，该器件受 BootLdrUSBFSe 用户模块 （如 CY8C24894）中的 USBUART 选项的支持。
2. 在器件编辑器中，单击 **Protocols**。通过双击 BootLdrUSBFSe 图标或右键单击并选中 **Select**，可以添加 BootLdrUSBFSe 用户模块。
3. 选择 USB-UART 器件 （CDC）单选按键并点击 **OK。**
4. 按照下图配置 BootLdrUSBFSe 用户模块。默认配置可以不同，但仍适用于已给的示例代码。

ApplicationCode_Start_Block	0x60
BootLoaderKey	0001020304050607
Fixed Pulse Width	ENABLE
Flash_Program_Temperature_deg_C	-20C
ICE_Debug_FLASH_DISABLE	Flash_WRITE_ENABLE
Bootload_when_CKSUM_fails	ENABLE_(deployment)
AcquisitionWindow	10

5. 确保将正确的引脚配置为下拉驱动，以识别开关闭合，进入 Bootloader。示例取决于您可以按的按键。此按键拉高 Port1_7，并导致程序作为 Bootloader 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。

⊞ P1[6]	Port_1_6, StdCPU, High Z Analog, DisableInt, Normal, 0
⊟ P1[7]	Port_1_7, StdCPU, Pull Down, DisableInt, Normal, 0
Name	Port_1_7
Port	P1[7]
Select	StdCPU
Drive	Pull Down
Interrupt	DisableInt
AnalogMUXBus	Normal
InitialValue	0
⊞ P2[0]	Port_2_0, StdCPU, High Z Analog, DisableInt, Normal, 0

6. 右键单击用户模块图标。选择 Boot Loader Tools。然后选择 Get Files。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。
7. 右键单击用户模块图标。选择 **Device: Application USB Setup Wizard…** 按照下图配置 “Bootloader CDC properties”。



单击 OK 保存 CDC 描述符的信息。

8. 右键单击用户模块图标。选择 **Device: Bootloader USB Setup Wizard…** 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 Bootloader 的 VID 和 PID 不能相同。由于所包含的主机 PC 示例项目具有供货商 ID （VID）04b4 和产品 ID （PID）E006 （赛普拉斯所拥有的 ID），因此可用于本地调试，但不可发布以用于生产。单击 OK 以保存 USB Bootloader 描述符信息。
9. 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。

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

  W  W  U  U  U   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   W   W   W ;    Base Address 1000
  W  W  W  W  W   W   W   W   W   W   W   W   W   W   W   W ;    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
```

10. 生成应用。
11. 复制样本代码并粘贴它。
12. 执行重建全部。

示例代码

下面代码说明了如何在简单的应用中使用 BootLdrUSBFSe — USBUART 用户模块。

```
#include <m8c.h>        // part specific constants and macros
#include "PSoCAPI.h"    // PSoC API definitions for all User Modules
BYTE Len;
BYTE pData[32];
void main(void)
{
    M8C_EnableGInt; //Enable Global Interrupts
    BL_USBFS_Start(0, USB_5V_OPERATION); //Start USBUART Operation usgin device 0
    while(!BL_UART_Init()); //Wait for Device to initialize
    while(1)
    {
    if(PRT1DR & 0x80)
        {
            BL_USBFS_Stop();
            while(PRT1DR & 0x80);
            ENTER_BOOTLOADER();
        }

        Len = BL_UART_bGetRxCount(); //Get count of ready data

        if (Len)
        {
            BL_UART_ReadAll(pData); //Read all data rom RX
            while (!BL_UART_bTxIsReady()); //If TX is ready
            BL_UART_Write(pData, Len); //Echo
        }
    }
}
```

写入汇编语言的等效代码为：

```
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

AREA bss (RAM, REL)

Len:   blk 1
pData: blk 32

export _main

AREA text (ROM, REL)
_main:
    M8C_EnableGInt         ; Enable Global Interrupts
    mov A, 0
    mov X, USB_5V_OPERATION
```

```
    lcall BL_USBFS_Start    ; Start USBUART 5V operation
deviceInit:                 ; Wait for Device to initialize
    lcall BL_UART_Init
    cmp A, 0
    jz deviceInit
mainLoop:
    ; implement bootloader entry
    mov A, reg[PRT1DR]
    and A, 0x80
    jz  contLoop
    lcall BL_USBFS_Stop
wait:
    tst reg[PRT1DR], 0x80
    jnz wait
    ljmp ENTER_BOOTLOADER  ; never returns
    halt

contLoop:
    lcall BL_UART_bGetRxCount
    RAM_SETPAGE_CUR >Len
    mov [Len], A ; Get count of ready data
    cmp [Len], 0 ; Check if Len is 0
    jz mainLoop
    mov A, >pData ; Load MSB part of pointer to RAM buffer
    mov X, <pData ; Load LSB part of pointer to RAM buffer
    lcall BL_UART_ReadAll ; Read all data rom RX
txReady:
    lcall BL_UART_bTxIsReady ; Check to see if TX is ready
    cmp A, 0
    jz txReady
    ; Echo data
    RAM_SETPAGE_CUR >Len
    mov A, [Len] ; Load array count
    push A
    mov A, >pData ; Load MSB part of pointer to RAM string
    push A
    mov A, <pData ; Load LSB part of pointer to RAM string
    push A
    lcall BL_UART_Write
    add SP, -3 ; Reset stack pointer to original position
    jmp mainLoop
```

附录 A — USBFS 主题

USB 标准器件请求

以下章节介绍了 BootLdrUSBFSe 用户模块支持的请求。如果某个请求不受支持，则 BootLdrUSBFSe 用户模块通常响应一个 STALL，以表示请求错误。

标准器件请求	BootLdrUSBFSe 用户模块支持说明	USB 2.0 规范章节
CLEAR_FEATURE	器件： 接口：不支持。 端点	9.4.1
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	返回指定接口的所选备用接口设置。	9.4.4
GET_STATUS	器件： 接口： 端点：	9.4.5
SET_ADDRESS	对将来所有的器件访问设置器件地址。	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 用户模块参数选择。 不支持 TEST_MODE。 接口：不支持。 端点：暂停指定的端点。	9.4.9
SET_INTERFACE	不受支持。	9.4.10
SYNCH_FRAME	不支持。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

HID 类请求

类请求	BootLdrUSBFSe 用户模块支持说明	HID 的器件类定义 — 章节
GET_REPORT	允许主机通过控制管道接收报告。	7.2.1
GET_IDLE	读取特定输入报告的当前闲置速率。	7.2.3
GET_PROTOCOL	读取当前有效的协议 （引导协议或报告协议）。	7.2.5
SET_REPORT	允许主机向器件发送报告，可能设置输入、输出或功能控制的状态。	7.2.2
SET_IDLE	使中断输入管道上的特定报告进入闲置状态，直到发生新事件或经过指定的时长。	7.2.4
SET_PROTOCOL	在引导协议和报告协议之间进行切换。	7.2.6

USB 设置向导

本部分详细介绍了 BootLdrUSBFSe 用户模块所指定的所有 USB 描述符。介绍的内容为描述符格式以及用户模块参数如何映射到描述符数据。

USB 设置向导是赛普拉斯提供的一个工具，用于帮助工程师设计 USB 器件。此设置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

要访问设置向导，请右击工作区浏览器中的 “BootLdrUSBFSe 用户模块 ” 图标，然后点击 **Device: Application USB Setup Wizard...** 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，提供可用的选项。

描述符	数据		操作
USB 用户模块描述符根	“ 器件名称 ”		添加器件
器件描述符	DEVICE_1		清除 | 添加配置
器件属性			
供货商 ID	FFFF		
产品 ID	FFFF		
器件发布 （bcdDevice）	0000		

描述符	数据		操作
器件类别	未定义	下拉	
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		清除 | 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	禁用	下拉	
接口描述符	INTERFACE_NAME		清除 | 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		清除
端点属性			
端点号	0		
方向	输入	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

描述符	数据		操作
LANGID		下拉	
字符串	所选字符串名称		清除
描述符			
HID 描述符	器件名称		导入 HID 报告模板

了解 USB 设置向导

“USB Setup Wizard” 窗口是一个表格，它显示了用于编程的三个主要区域。第一个区域是 USB 描述符，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键执行所选的指令。

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，请点击 **Add String** （添加字符串）操作。软件即会添加一行，提示您在此处编辑字符串。键入新字符串，然后单击 **Save/Generate** （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。为了进行内容说明，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，然后用其他值或文本选项覆盖该数据。所有值必须满足 USB 2.0，第九章的规范。

在顶部显示的第一个文件夹是 *USB User Module Descriptor Root* （USB 用户模块描述符根）。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 Interconnect （互连）视图中放置的用户模块。右侧列中的 Add Device （添加器件）操作可以添加配有描述器件时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。点击 **OK** （确认）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；可能会将其清除，也可能会添加配置。通过覆盖现有数据或使用下拉菜单，可以输入有关特定 USB 器件的所有信息。

在输入完数据后，可以通过使用下拉菜单或通过在相应的地方键入字母数字文本，然后点击 **OK** （确定）保存。

USB 挂起、恢复、远程唤醒和监控 USB 活动

BootLdrUSBFSe 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 BootLdrUSBFSe 用户模块提供了一组 API 函数。

BL_USBFS_bCheckActivity API 函数提供一种方法来检查是否发生任何 USB 总线活动。如果器件支持远程唤醒，则应用能够确定主机是否使用 BL_USBFS_bRWUEnabled API 函数使能了远程唤醒。当器件挂起且确定满足用于启动远程唤醒的条件时，应用会使用 BL_USBFS_Force API 函数将相应的 J、 K 状态强制到 USB 总线上，从而通知远程唤醒。

创建供货商特定的器件请求并覆盖现有请求

BootLdrUSBFSe 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。您也可以写入您自己的子程序，以覆盖任何所提供的标准及类别特定子程序，也可以使能不受支持的请求类型。

处理 USB 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 BootLdrUSBFSe 用户模块中，所有的控制传输均由端点 0 中断服务子程序（BootLdrUSBFSe_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）。供货商特定的器件请求或覆盖的器件请求的处理程序由应用提供。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 BootLdrUSBFSe_EP0_ISR，它会将设置数据包路由到 BootLdrUSBFSe_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。在控制写操作时，器件请求处理程序必须对应用做好准备，以接收数据。在控制读操作时，则要准备好数据，以将其传输到主机。对于无数据控制传输，处理程序会从设置数据包中提取信息。

BootLdrUSBFSe 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，根据数据操作的方向，决定将数据复制到控制端点缓冲区或从该缓冲区中复制数据 （寄存器 EP0DATA0-EP0DATA7）。

供货商特定器件请求调度子程序

根据应用要求的不同， USBFS 用户模块可根据设置数据包的 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：

表 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	使能标志
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 *BootLdrUSBFSe.inc* 文件中，使能对供货商特定的调度子程序的支持。查找调度子程序使能标志并将 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，或使能某个不受支持的器件请求，您必须执行以下操作：

1. 在 *BootLdrUSBFSe.inc* 文件中，将特定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），请使能子程序并在 *BootLdrUSBFSe.inc* 中进行以下更改：

```
;@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.)
```

然后，写入一个名为 APP_USB_CB_SRC_h2d_cls_ifc_09 的汇编语言程序。器件请求名称由 USB bmRequestType 和 bRequest 值定义 （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 BootLdrUSBFSe_InitControlWrite
```

附录 B — Bootloader 主题

以下部分包含了用户在创建 USB Bootloader 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。

表 2. 端点 0 ISR 返回点

返回输入点	所需的数据项	说明
BootLdrUSBFSe_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	
BootLdrUSBFSe_InitControlRead	此返回点用于启动控制读取传输。	
	BootLdrUSBFSe_DataSource (BYTE)	数据源为 RAM 或 ROM（USB_DS_RAM 或 USB_DS_ROM）。这非常重要，因为要使用不同的指令从源 ROMX 或 MOV 移动数据。
	BootLdrUSBFSe_TransferSize (WORD)	要传输的数据字节数。
	BootLdrUSBFSe_DataPtr (WORD)	数据的 RAM 或 ROM 地址。
	BootLdrUSBFSe_StatusBlockPtr (WORD) 可选	利用 USB_XFER_STATUS_BLOCK 宏分配的状态模块的地址。
BootLdrUSBFSe_InitControlWrite	此返回点用于启动控制写入传输。	
	BootLdrUSBFSe_DataSource (BYTE)	USB_DS_RAM （控制写入的目标必须为 RAM）。
	BootLdrUSBFSe_TransferSize (WORD)	用于接收数据的应用缓冲区的大小
	BootLdrUSBFSe_DataPtr (WORD)	用于接收数据的应用缓冲区的 RAM 地址
	BootLdrUSBFSe_StatusBlockPtr (WORD) 可选	利用 USB_XFER_STATUS_BLOCK 宏分配的状态模块的地址。
BootLdrUSBFSe_InitNoDataControl Transfer	此返回点用于启动无数据控制传输。	
	BootLdrUSBFSe_StatusBlockPtr (WORD) 可选	利用 USB_XFER_STATUS_BLOCK 宏分配的状态模块的地址。

通过 USB Bootloader 更新 USB 应用请求子程序

通过 BootLdrUSBFSe 用户模块，您可以覆盖现有 USB 应用或添加新 USB 应用 （非 Bootloader 应用）。

要想使能该特性，需要在 BootLdrUSBFSe.inc 文件中将 UPDATE_USB_APP_HANDLERS 常量值重新定义为 USB_UM_SUPPLIED。请注意，必须在 Bootloader 部署前使能该特性。调用自定义请求子程序的代码应该位于 BootLdrUSBFSe.asm 文件中的 UserDispatchCode 和 UMDispatchCode 标签之间。通过 <UM_Name>_ReqReturnInstruction 变量返回的值可用于控制 Bootloader 的下个操作，表 3 介绍了这些值。

表 3. UserDispatchCode 代码执行后 Bootloader 执行的操作

数值	说明
USB_UM_DISPATCH	调度用户模块的请求
USB_NO_DATA_STAGE_CONTROL_TRANSFER	为无数据控制写操作准备状态阶段
USB_GET_TABLE_ENTRY	传输数据结构
USB_INIT_CONTROL_READ	初始化控制读
USB_INIT_CONTROL_WRITE	初始化控制写
USB_NOT_SUPPORTED_REQUEST	将该请求作为 “ 不受支持 ” 进行处理

注意：不能从 UserDispatchCode 直接跳转到 Bootloader 子程序。如果您需要从子程序跳转到 Bootloader 代码 （表 3 未定义该代码）中的任何位置，那么在 BootLdrUSBFSe_drv.asm 文件中，您必须将跳转位置添加到 BootLdrUSBFSe_DT_App_Req 调度表内。然后跳转到您刚定义的位置。再次，请注意必须在 USB Bootloader 部署前进行这些更改。

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。数据传输长度是指所传输数据字节的实际数目。

表 4. 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没有有效数据，并应用不应使用该缓冲区。当完成代码为 USB_XFER_IDLE 时，进行传输实际数据，但该数据并不指示某个传输正在运行中。
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，应用使用相关数据缓冲区及其内容。
USB_XFER_PREMATURE （0x02）	USB_XFER_PREMATURE 指示后续控制传输的 SETUP 中断了控制传输。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

如果您使能可选 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。
2. 在 *USB_descr.asm* 中禁用向导定义的报告存储区域。
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* 中受保护的用户代码区域内或到独立的汇编语言文件内。
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

使用 USB 设置向导，以指定您的器件说明、端点和 HID 报告。点击 PSoC Designer 中的 **Generate Application** 按键。

禁用向导定义的报告存储区域

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

请在 *USB_descr.asm* 中查找该代码。

```
;-------------------------------------------------------------------------
; HID IN Report Transfer Descriptor Table for ()
;-------------------------------------------------------------------------
IF WIZARD_DEFINED_REPORT_STORAGE
AREA  lit    (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。

将代码粘贴到受保护的用户代码区域

您可将代码粘贴到 *USB_descr.asm* 中受保护的用户代码区域内，或到独立的汇编语言文件内。

```
   ;-------------------------------------------------
   ;@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。

表 5. 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。报告 ID 0 不被使用。
TD_ENTRY		
	USB_DataSource	数据源为 RAM 或 ROM （USB_DS_RAM 或 USB_DS_ROM）。
	USB_TransferSize	数据传输的大小以字节计算。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。

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。注意：只有在您将代码置于 *USB_descr.asm* 的受保护用户代码区域中时，才需要有条件的汇编语句。

```
;-------------------------------------------------------------------------
; 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  lit     (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
```

Bootloader USB 下载协议

针对 Bootloader 的每个指令后面都跟随着来自 Bootloader 的响应。下图显示 “ 输入 Bootloader” 指令的格式：

Enter Bootloader Packet

Packet data (BULK OUT):

0000	FF	38	00	01	02	03	04	05	06	07	00	00	00	00	00	00
0010	00	80	01	00	01	40	15	40	00	AB	3F	FF	01	00	15	40
0020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0030	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Status data (BULK IN):

0000	01	00	00	00	04	00	00	00	00	00	00	00	00	00	00	00
0010	00	80	01	00	01	40	15	40	00	AB	3F	FF	01	00	15	40
0020	00	00	00	00	00	00	00	00	00	00	40	00	3B	20	81	01
0030	FF	01	00	00	00	00	00	00	00	00	00	01	00	00	00	00

	BootLoader Commands
	BootLoader Key
	Status Byte
	Error Byte
	BootLoader Configuration Data (from dld file)
	BootLoader Configuration Data (from Bootloader code)

Bootloader 配置数据用于以下格式：

字节编号	说明	在引导加载过程中验证
0x10-0x11	可重定位中断表的地址	+
0x12-0x13	校验和模块的地址	+
0x14-0x15	Bootloader 区域的地址	+
0x16-0x17	应用区域的地址	–
0x18-0x19	闪存模块中的应用区域大小	–
0x2A-0x2B	最大 ROM 地址	+
0x2C-0x2D	用户模块版本，例如 0x0100、 0x0200	+
0x2E-0x2F	使用 Bootloader 区域 ImageCraft compilerAddress 的应用区域的地址（Bootloader 区域使用 HI-TECH 编译器）	–

第一行以 Bootloader 指令 FF38 （输入 Bootloader）开始。随后此指令是 Bootloader 密钥。所有 Bootloader 指令在发送时必须有 Bootloader 密钥。 Bootloader 将忽略发送时没有正确密钥的指令。您可以使用 Bootloader_Key 参数设置 Bootloader 密钥。其他的 Bootloader 指令有：

指令	含义
FF38	进入 Bootloader
FF39	块写入
FF3A	验证闪存
FF3B	退出 Bootloader

此指令由一个来自 Bootloader 的状态响应随后。在状态响应中，第一个字节是状态字节，第二个字节是错误字节。下面显示的是每位状态字节的定义：

代码	含义
0x01	引导加载正在进行
0x02	已成功完成引导
0x04	等待进入 Bootloader 指令

下表介绍的是错误字节中的每一位。有关详细信息，请参阅本文档末尾的流程图。

代码	含义
0x01	超时错误。如果由于主机在长期内进入不活动状态而使读或写超时发生，则将设置标志并执行软件复位。只在 BOOT_TIMEOUT 常量大于 0 时，才能执行超时验证。
0x02	收到了 ‘Exit Bootloader’ （退出 Bootloader）指令。由 Bootloader 计算的应用程序和可重定位中断向量区域的校验和与从主机中收到的校验和不匹配。
0x04	闪存校验和错误。闪存模块的内容与从数据接收到的数据不匹配。
0x08	闪存保护错误。不可重新写入闪存模块因为闪存保护级不允许该操作。
0x10	通信校验和错误。接收了含不正确校验和的数据包。
0x20	配置数据错误。收到了 ‘Enter Bootloader’ （进入 Bootloader）指令，但数据包包含错误配置数据。
0x40	无效的 Bootloader 密钥。已经接收到拥有不正确 BootloaderKey 值的数据包。
0x80	无效指令错误。接收了未知指令。

Bootloader 写入模块指令

发送到 Bootloader 的大多数指令都是写入模块指令。每个写入模块指令的格式都一样。每个 64 字节模块均分为两个 32 字节数据包。每个指令均需要来自从器件的状态响应。 64 字节模块的传输如下图所示：

First Download Packet

Packet data (BULK OUT):

0000	FF	39	00	01	02	03	04	05	06	07	00	02	00	7F	30	30
0010	30	30	7E	30	30	7E	30	30	30	7E	30	30	30	7E	30	30
0020	30	7F	30	30	30	7F	30	30	30	7E	30	30	30	C9	00	00
0030	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Status data (BULK IN):

0000	01	00	00	00	02	20	00	00	00	00	00	00	00	00	00	00
0010	00	80	01	00	01	40	15	40	00	AB	3F	FF	01	00	15	40
0020	00	00	00	00	00	00	00	00	00	00	40	00	3B	20	81	01
0030	FF	01	00	00	00	00	00	00	00	00	00	01	00	00	00	00

Packet data (BULK OUT):

0000	FF	39	00	01	02	03	04	05	06	07	00	02	01	7E	30	30
0010	30	7E	30	30	30	7E	30	30	30	7E	30	30	30	7F	30	30
0020	30	7F	30	30	30	7F	30	30	30	7F	30	30	30	CB	00	00
0030	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00

Status data (BULK IN):

0000	01	00	01	00	02	40	00	00	00	00	00	00	00	00	00	00
0010	00	80	01	00	01	40	15	40	00	AB	3F	FF	01	00	15	40
0020	00	00	00	00	00	00	00	00	03	00	40	80	3B	00	00	01
0030	FF	01	00	00	00	00	00	00	00	00	00	01	00	00	00	00

	BootLoader Commands
	BootLoader Key
	Block Number
	Block Segment
	Status Byte
	Error Byte
	64 bytes of data
	Segment Checksum for 32 bytes of data

第一个数据包的第一行包含一个写入模块指令和 Bootloader 密钥，后面为所传输的模块编号。由于每个模块均分为两段，因此模块编号由模块段编号随后 （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，后面是段数据的校验和。模块的剩余部分为空数据，用于将段扩充为 64 个字节。

状态响应包含状态字节、错误字节和用于将段扩充为 64 个字节的 62 字节空数据。

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 （校验和模块除外）。校验和模块包含校验和以及用于 Bootloader 操作的其他数据。校验和数据模块的格式如下图所示：

Checksum Block Packet

Packet data (BULK OUT):

```
0000 FF 39 00 01 02 03 04 05 06 07 00 04 00 0D 34 B2
0010 3F 00 00 00 80 01 00 01 40 15 40 00 AB 3F FF 01
0020 00 15 40 30 30 30 30 30 30 30 30 30 30 8F 00 00
0030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Status data (BULK IN):

```
0000 01 00 00 00 04 20 00 00 00 00 00 00 00 00 00 00
0010 00 80 01 00 01 40 15 40 00 AB 3F FF 01 00 15 40
0020 00 00 00 00 00 00 00 00 03 00 40 80 3B 00 00 01
0030 FF 01 00 00 00 00 00 00 00 00 00 01 00 00 00 00
```

Packet data (BULK OUT):

```
0000 FF 39 00 01 02 03 04 05 06 07 00 04 01 30 30 30
0010 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30
0020 30 30 30 30 30 30 30 30 4D C0 B2 3F 97 00 00 00
0030 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

Status data (BULK IN):

```
0000 01 00 01 00 04 40 00 00 00 00 00 00 00 00 00 00
0010 00 80 01 00 01 40 15 40 00 AB 3F FF 01 00 15 40
0020 00 00 00 00 00 00 00 00 03 00 40 80 3B 00 00 01
```

	Checksum of the remainder of 58 bytes of data
	Reloc Int Vecs address
	Checksum block address
	Bootloader start address
	Application start address
	Application size (in blocks)
	Device memory size
	Bootloader version number
	Reloc code start addresss (for Image Craft only)
	Application checksum 2's complement
	Application checksum

根据器件系列，可将校验和模块定位在模块 0x0004 或 0x0002 上 （在该示例中，模块编号为 0x0004）。

与其他记录相同，第一行包含 Bootloader 写入模块指令、 Bootloader 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 58 个字节的校验和，本示例中该值为 0x0D34。第一行中最后的字节和第二行的第一个字节包含的是辅助应用校验和。如果应用代码正确，则辅助应用校验和应等于应用校验和，否则表示应用代码有误。

下面各行包含 2 字节值，它代表可重定位中断向量的十六进制地址、校验和模块、 Bootloader 启动、应用程序启动、应用程序大小、器件的最大存储器、 Bootloader 版本编号以及可重定位代码的起始地址。

几乎所有下面的各行都是空数据空间。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。数据包中剩余部分为空白空间。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的数据是应用程序校验和及第三行中的段校验和。

最后下载数据包是 Bootloader Exit （Bootloader 退出）指令：

Last Download Packet

Packet data (BULK OUT):															
0000	FF	38	00	01	02	03	04	05	06	07	00	00	00	00	00	00
0010	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0020	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
0030	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
Status data (BULK IN):																
0000	02	00	01	00	00	40	01	00	80	00	00	00	00	00	00	00
0010	00	80	01	00	01	40	15	40	00	AB	3F	FF	01	00	15	40
0020	00	00	00	00	00	00	00	00	03	00	40	80	3B	00	00	01
0030	FF	01	00	00	00	00	00	00	00	00	00	01	00	00	00	00

	BootLoader Exit Commands
	BootLoader Key
	Status Byte
	Error Byte
	BootLoader Configuration Data (from Bootloader code)

Bootloader 退出指令包含 Bootloader 退出指令 0xFF3B 以及 Bootloader 密钥。

最后状态响应包含一个 0x02 状态字节 （已成功完成引导）和一个 0x00 错误字节 （无错误）。如果状态和错误字节存有其他数值，则表示引导加载过程失败。

图 5. BootLdrUSBFSe 用户模块操作流程图 （64 字节的闪存模块器件）

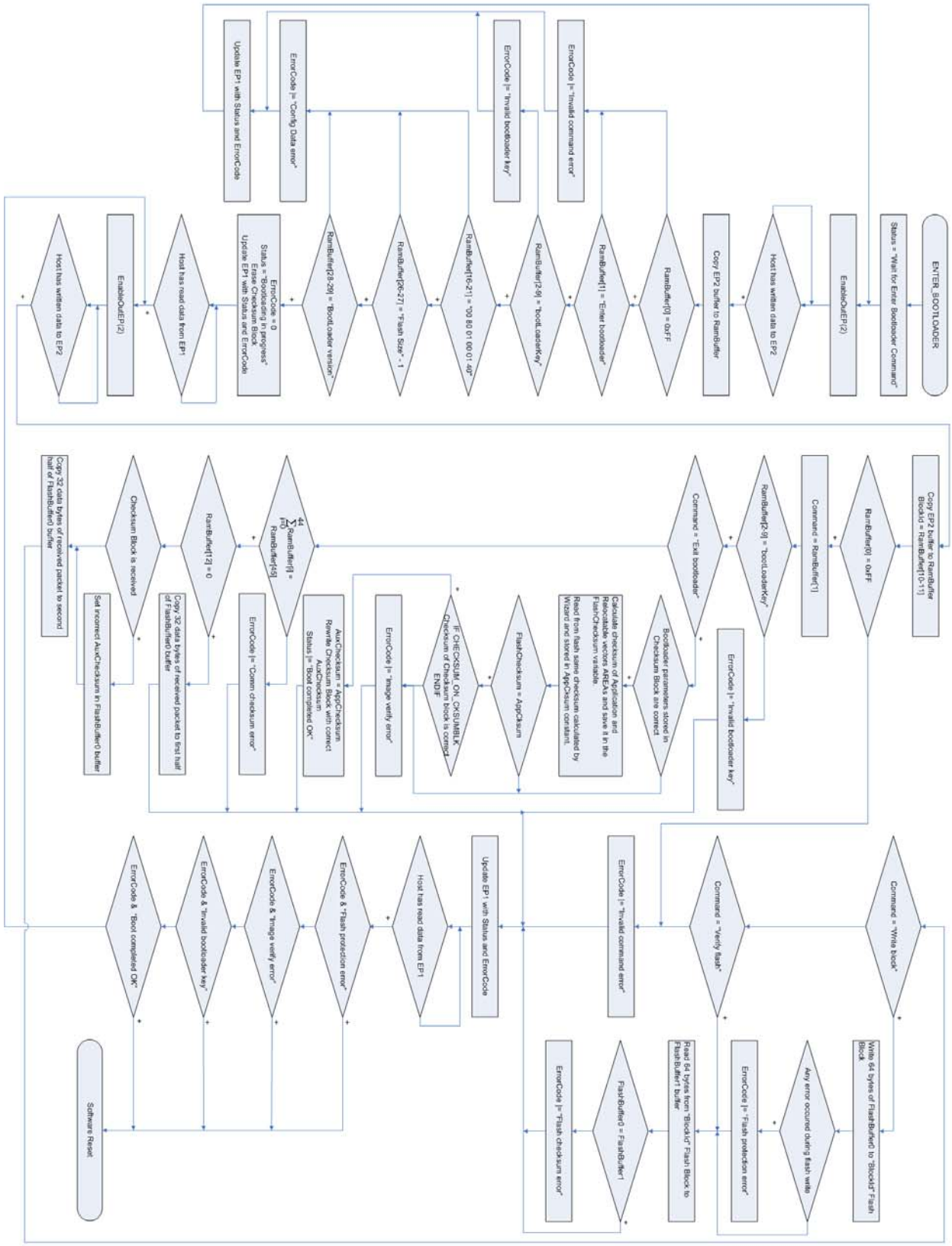
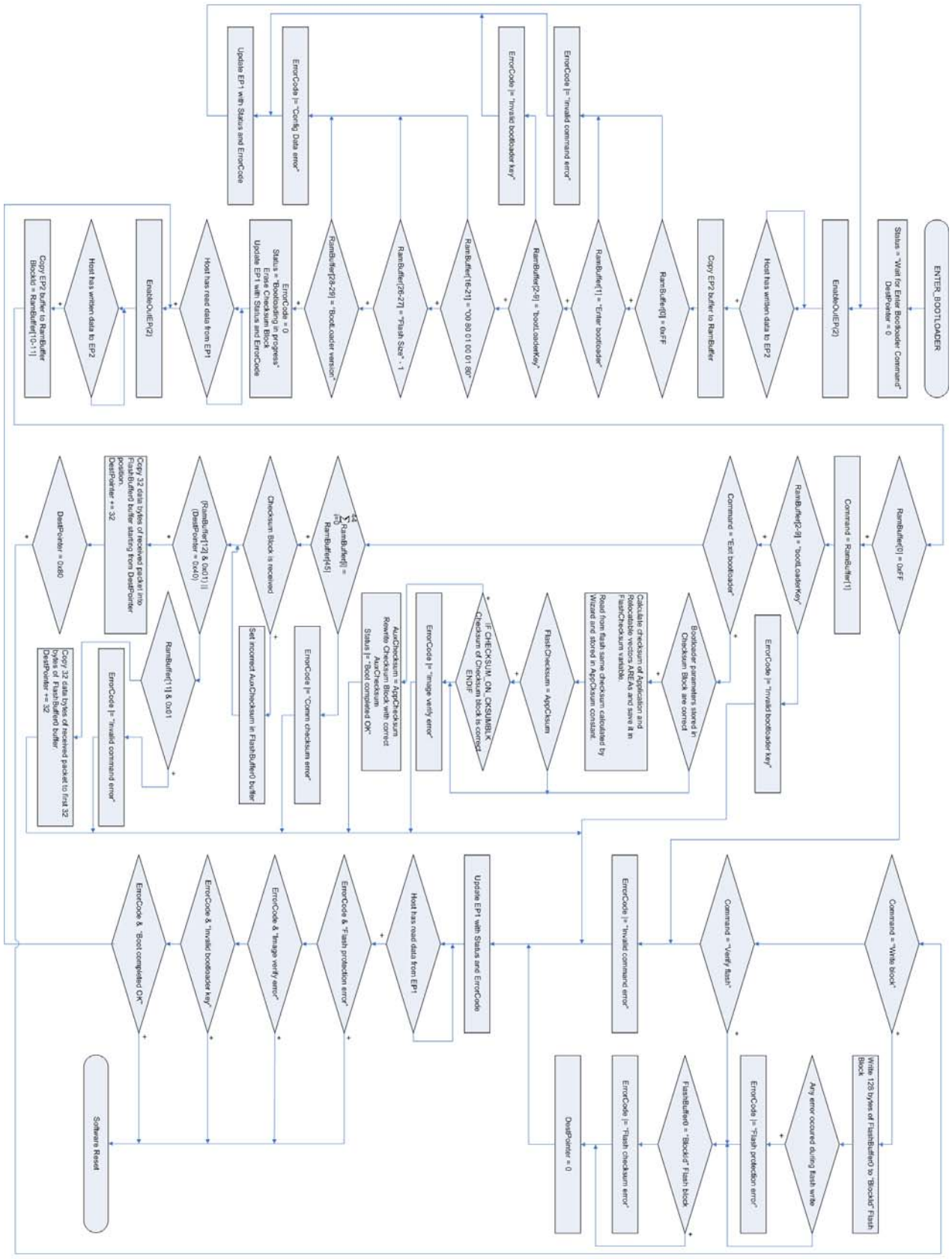


图 6. BootLdrUSBFSe 用户模块操作的流程图 （128 字节的闪存模块器件）



BootLdrUSBFSe 和 E2PROM 用户模块共存

将 E2PROM 用户模块放置入 Bootloader 项目中时，需要将 E2PROM 模块分配给客户保留模块区域。这一区域位于 Bootloader 代码区和应用程序代码区之间 （具体信息参见 Bootloader 存储器组织）。因此，E2PROM 模块并非应用程序代码区的一部分，且将不作为应用校验和的一部分计算。

版本历史记录

版本	创作者	说明
1.0	DHA	初始版本
1.10	DHA	1. 在 jmp 指令集中清除 .Literal/.Endliteral 指令。 2. 在 USB_cls_hid.asm 内，分别为 USBFS_bGetProtocol 和 USBFS_UpdateHIDTimer 函数清除 Directives .SECTION 和 .ENDSECTION。
1.20	DHA	1. 添加了对写 EP0_CR 的验证。 2. 添加了对 SIE MODE 的验证，并在 EP0 ISR 中验证 ACK 位。 3. 加大了 Bootloader 区域。 4. 在向导中添加了帮助文件。 5. 在 SSC 调用中，将 .Literal 和 .EndLiteral 语句替换为 .nocc。 6. 清除了导出 `@INSTANCE_NAME`_EnterBootloader 语句。
1.30	DHA	1. 将 “AREA UserModules” 中的应用描述符转移到 “AREA lit” 中。 2. 添加了初始化 USB_Protocol 变量，从而使之符合 HID 标准。 3. 添加了在 Workspace Explorer （工作区浏览器）中显示 Bootloader 输出文件的支持。 4. 纠正了大型存储器模型模式指令，以指出编译错误。 5. 对于 Encore III 器件，将 USB 通讯的内部振荡器自动频率锁定功能添加到 bootResetIsr() 函数内。 6. 向 “Flashsecurity.txt 中的错误设置 ” 部分添加了说明。

版本	创作者	说明
1.40	DHA	1. 更新了图 3 和图 4。
1.50	DHA	1. 修改了 `@INSTANCE_NAME`_bCheckActivity API 函数，以防止丢失数据包。 2. 添加了 CYRF89435 器件支持。 3. 将 BootloaderKey 参数限制为 16 个符号。
2.00	HPHA	1. 添加了通过 USB Bootloader 覆盖 USB 应用请求子程序的功能。 2. 添加了覆盖 Not_Supported 请求子程序的功能。 3. 校验和验证从启动移动到引导加载完成阶段，以加快应用启动过程并满足 USB 合规性。
3.00	MYKZ	1. 对于 CY8C20xx6 系列，修正了 BL_USBFS_bReadOutEP 函数，以防止 CPU 时钟改变。 2. 添加了 CY8C24x93 和 CY7C69000 支持。 3. 添加了一个警告信息，以向用户通知用户模块从项目中被清除的时候。 4. 修正了存储器重叠问题：Bootloader 和应用程序使用的各变量被锁定在 RAM 第 0 页的结尾上。

注意： PSoC Designer 5.1 在所有用户模块数据手册中介绍了 “ 版本历史 ”。本数据手册详细介绍当前和先前用户模块版本之间的区别。

Copyright © 2010-2014 赛普拉斯半导体公司。此处所包含的信息可能会随时更改，恕不另行通知。除赛普拉斯产品内嵌的电路外，赛普拉斯半导体公司不对任何其他电路的使用承担任何责任。也不会根据专利权或其他权利以明示或暗示的方式授予任何许可。除非与赛普拉斯签订明确的书面协议，否则赛普拉斯产品不保证能够用于或适用于医疗、生命支持、救生、关键控制或安全应用领域。此外，对于合理预计会发生运行异常和故障并对用户造成严重伤害的生命支持系统，赛普拉斯将不批准将其产品用作此类系统的关键组件。若将赛普拉斯产品用于生命支持系统，则表示制造商将承担因此类使用而招致的所有风险，并确保赛普拉斯免于因此而受到任何指控。

PSoC Designer™ 和 Programmable System-on-Chip™ 是赛普拉斯半导体公司的商标， PSoC® 是赛普拉斯半导体公司的注册商标。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。

所有源代码 （软件和 / 或固件）均归赛普拉斯半导体公司 （赛普拉斯）所有，并受全球专利法规 （美国和美国以外的专利法规）、美国版权法以及国际条约规定的保护和约束。赛普拉斯据此向获许可者授予适用于个人的、非独占性、不可转让的许可，用以复制、使用、修改、创建赛普拉斯源代码的派生作品、编译赛普拉斯源代码和派生作品，并且其目的只能是创建自定义软件和 / 或固件，以支持获许可者仅将其获得的产品依照适用协议规定的方式与赛普拉斯集成电路配合使用。除上述指定用途外，未经赛普拉斯的明确书面许可，不得对此类源代码进行任何复制、修改、转换、编译或演示。

免责声明：赛普拉斯不针对该材料提供任何类型的明示或暗示保证，包括 （但不仅限于）针对特定用途的适销性和适用性的暗示保证。赛普拉斯保留在不另行通知的情况下对此处所述材料进行更改的权利。赛普拉斯不对此处所述之任何产品或电路的应用或使用承担任何责任。对于合理预计可能发生运转异常和故障，并对用户造成严重伤害的生命支持系统，赛普拉斯不授权将其产品用作此类系统的关键组件。若将赛普拉斯产品用于生命支持系统，则表示制造商将承担因此类使用而导致的所有风险，并确保赛普拉斯免于因此而受到任何指控。

产品使用可能受适用于赛普拉斯软件许可协议的限制。