I2C 引导加载程序 (Bootloader) 数据手册 BootLdrI2C V 2.20

Copyright © 2010-2012 Cypress Semiconductor Corporation. All Rights Reserved.

资源	PSoC® 模块				API 占用存储器（字节）		引脚
	CapSense®	I2C/SPI	定时器	比较器	闪存	RAM	
CY8C20x96、CY8C20x66、CY8C20x46、CY8C20x36、CY8C20336AN, CY8C20436AN, CY8C20636AN、CY8C20xx6AS、CY8C20XX6L、CY7C643/4/5xx, CY7C60413、CY7C60424、CY7C6053、CY8CTST200、CY8CTMG200、CY8CTMA30xx、CYONS2xxx、CYONSFN2xxx、CYONSTB2010、CYONSTB2011、CYONSFN2010-BFXC、CYONSCN2024-BFXC、CYONSCN2028-BFXC、CYONSCN2020-BFXC、CYONSKN2033-BFXC、CYONSKN2035-BFXC、CYONSKN2030-BFXC							
从器件 （支持全部 API ） 12 字节		1			2134	12 字节	2
从器件 （不支持 API ） 5 字节		1			2017	5 字节	2
纯软件从器件 （不支持 API ）不可以使用 I^2C 资源给应用程序					2518	5 字节	

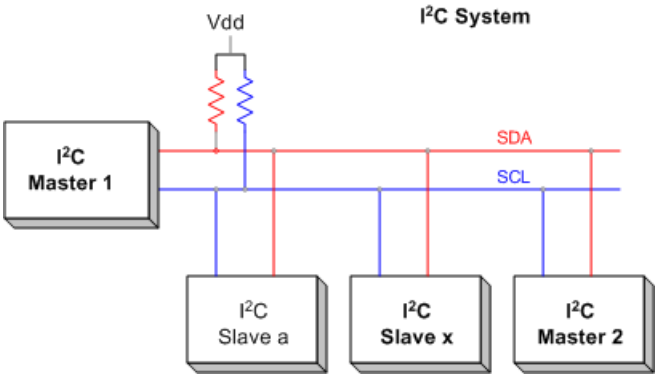
功能和概述

- 工业标准 Philips I^2C 总线兼容接口。
- 可以通过 I^2C 系统总线，而不是使用在线编程引脚对 PSoC® 器件重新编程。

I^2C Bootloader 用户模块实现了一个引导加载程序，可以通过 I^2C 接口对 PSoC 器件重新编程。 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。 然而，引导加载程序能够通过工业标准通信接口 （如 I^2C）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非常有用。 CY3240 引导加载信息可通过一个 I^2C 主器件，例如 USB 到 I^2C 转接桥或系统主机处理器等进行发送。

I^2C 引导加载程序需要使用 I^2C 硬件用户模块。 它并不防碍 I^2C 总线在 PSoC 器件内的其他功能的使用。 I^2C 引导加载程序为其相关联的功能使用单独的 I^2C 地址。 I^2C 引导加载程序的所有代码都在 EEPROM 的受保护区域进行编程，且不会被意外覆盖。

Figure 1. I²C 框图



可用配置

有三种配置可供选择。 其中的两种配置使用可用的 I²C 资源，这样可防止在同样使用 I²C 资源的应用程序中添加不同用户模块。 第三种配置用于纯软件引导加载程序，该程序不使用中断，且不会消耗 I²C 资源。 这在设计应用时提供了更多的灵活性。 纯软件用户模块的运算结构已进行了其他更改；这些更改在本用户模块数据手册的结尾进行了总结。

快速启动

1. 查看本用户模块数据手册。
2. 为确保正常运行，此用户模块改变了存储器位置以及编译器对整个项目的处理方式。 在实施引导加载程序项目之前，有必要了解一下用户数据手册中的信息。
3. 将用户模块添加到项目中。
4. 放置用户模块。 选择 “I²C Operation for Boot loader only（I²C 只支持 Bootloader）” 或者 “Full I2C API Support with Bootloader （支持全部 I²C API 的 bootloader）”。
5. 在菜单栏中，打开**项目 > 设置**对话框，然后单击**确定** 保存项目参数。
6. 右键单击用户模块图标并选择 **“Boot Loader Tools （引导加载程序工具）”**.
7. 单击 **获取文件**. 文件 *boot.tpl*, *custom.lkp* 和 *flashsecurity.example* 会放在项目根目录中。
8. 关闭 **“Boot Loader Tools （引导加载程序工具）”** 向导。
9. 一些用户模块使用大型阵列中的存储器或 page 0 的大量 RAM。 结果很可能耗尽 page 0 的内存。 如果出现此问题，请选择**项目 > 设置 > 编译器**，然后在 HI-TECH 的 “ 选项 Options” 框中添加以下行：

   ```
   --AUTOBANK=1
   ```

   此选项会将自动的 C 变量移到内存 page 1。 选择的内存页不能超过正在使用的器件可用的最大值。 有关 --AUTOBANK 选项的详细信息，请参见 HI-TECH 手册。
10. 生成源代码并编译项目。
11. 查看输出文件 *<project>.mp* 以及 *<project>.hex* 了解项目是如何构建的。
12. 在创建了编译准确无误的项目后，请转到 “ 固件源代码示例 ” 章节。 修改并调整此章节提供的示例代码。
13. 详细教程请见 PSoC Designer™ 5.1。 要访问引导加载程序教程，请转到菜单栏，然后单击 **Help > Documentation > Supporting Documents**。

功能描述

引导加载程序位于闪存部分，用户可以使用用户模块参数进行定义。 此存储器空间进行了写保护，以防止发生意外修改或损坏。 复位矢量已进行了修改，以便在处理器复位后执行引导加载程序。

引导加载程序可执行下列操作：

1. 复位后，引导加载程序计算闪存中用户代码的校验和，并对照写入闪存最后两个字节的校验和对其进行验证。 如果校验和是匹配的，则表示之前进行的编程尝试是成功的，引导加载程序将转到用户代码的开始部分。 用户代码可以执行。
2. 如果校验和不匹配，则引导加载程序将执行自定义的用户代码，以执行关键的系统任务 （例如打开风扇），然后进入引导加载程序模式，等待从主控处获取 10-byte 引导加载程序密钥。 如果之前的引导加载失败 （比如，出现电源瞬变情况），程序会由于校验和不匹配而进入引导加载程序模式。
3. 从主控处收到有效的引导加载程序密钥后，引导加载程序将以一个状态字节作为响应，告知主控它已准备好接收闪存映像。
4. 主控通过带若干编码字节的长度可变的数据包发送更新的用户代码。
5. 引导加载程序会将用户代码写入闪存中。 当所有的闪存页都成功写入，引导加载程序会执行闪存验证操作，然后进行软件复位来开始执行用户代码。

   Note I^2C 主控在每次进行块写入操作后须等待 100 ms 才能读取该块的状态字节，以便对闪存块进行写入操作。 器件正在写入闪存时无法响应中断。 带 128 字节闪存模块的器件累加数据包并验证正确的数据包结构，直到将一个完整的闪存模块累加完为止。

用户模块的引导加载程序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 ™ 项目的存储器结构有很大区别。 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，融入了引导加载程序的项目包含两个独立的程序，这两个程序支持不同的功能。 图 2 显示了引导加载程序的存储器映射。

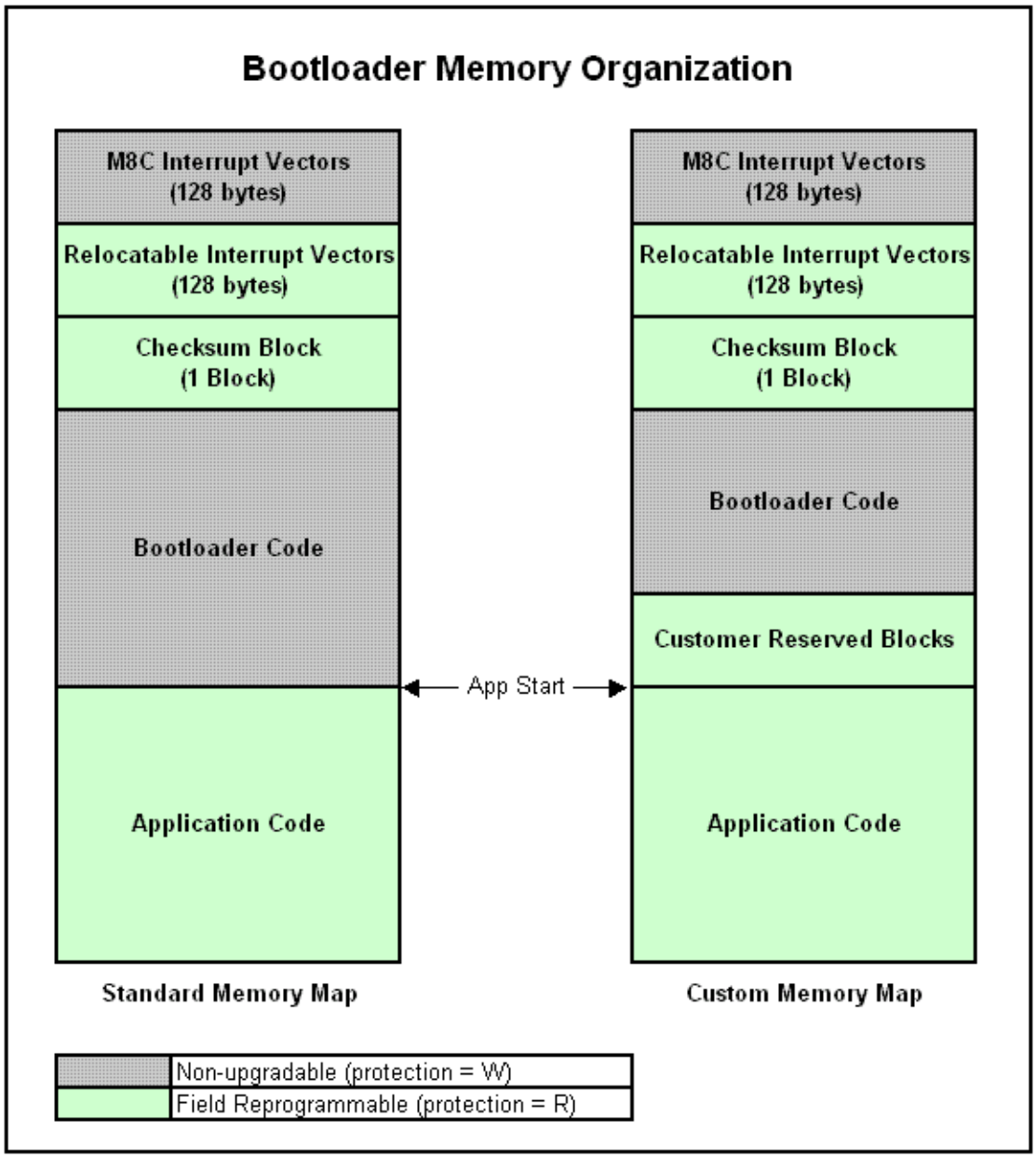
在部署融入了引导加载程序的项目之后，将无法对灰色显示的存储器位置重新进行编程。 对于绿色显示的存储器位置，可通过运行引导加载程序进行更改。

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。 为了便于此操作，BootLdrI2C 用户模块提供了自定义的文件和专用工具，能够在引导加载程序项目开发过程中为用户提供帮助。 要使用这些专用工具，请切换到 “ 器件编辑器 Device Editor” 视图，然后右键单击 “BootLdrI2C 用户模块 ” 图标。 除了作为用户模块一部分提供的工具和文件，还提供了一个 host 应用程序例程作为用户模块安装的一部分，用来展示引导加载程序的基本功能。 PC 的应用程序和 Microsoft Visual Studio® 2005 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\BootLdrI2C\BootLoaderHostApp\…

Figure 2. 引导加载程序存储器映射



概述

PSoC Designer 使用标准化文件，内置有关部件系列的数据，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。

存储器布局显示受管 ROM 的六个关键区域：

- 第一个存储器区域是 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。 前两个模块的 ROM 不得修改或放置在任何其他位置。

- 第二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。 这一区域位于第 2 和第 3 模块上。 在部署了应用程序和引导加载程序之后，这一区域中的内容可能会被覆盖，但其位置不得更改。 此区域的特性与下一部分描述的校验和区域类似。
- 定义的第三个 ROM 区域是校验和区域。 这一区域位于第 4 模块上，包含引导加载程序软件用来下载和验证前台应用程序的重要数据。 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本身的校验和。 最后两个字节是运行时应用程序的校验和。 校验和模块的结构除了包含引导加载程序使用的空间外，还包含用来自定义数据的空间。 这种结构被定义为 C 结构，而且只要不更改引导加载实用程序使用的数据或者在模块内对其重新定位，就可以对此结构进行修改。
- 定义的第四个存储器区域是包含引导加载程序代码本身的区域。 这一区域起始于第 5 模块。 在部署了包含引导加载程序的工程或器件之后，将不能对这一区域重新编程或现场升级。
- 第五个区域留作用户数据使用。 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。 如存储器映射所示，这一区域是可选的。 如果没有任何自定义数据，那么可以使用标准存储器映射。
- 第六个存储器区域是应用程序区域。 这一区域保存应用程序映像。 由于“引导加载程序代码”区域的代码大小可进行扩展，因此这一区域的起始地址可进行调整。 使用“Application_Start_Block”参数 （位于属性窗口中）可相应地设置应用程序起始地址。 该区域必须占用所有剩余的存储器。

如果应用程序具有一些必须始终启用 （包括在引导加载过程中）的代码，BootLdrI2C 用户模块的设计可提供充分的自定制来满足这一需求。 解决这个问题的最好方法是使用汇编程序 AREA 指令将此代码添加到引导加载程序 ROM 区域。 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

使用 BootLdrI2C 用户模块中提供的参数可自定义主要程序单元在 ROM 中的位置。 用户模块中的默认参数应提供有效的初始设置。 用户必须使用这些设置，直到完整的项目成功编译完成。 在项目编译完成之后，查看程序存储器映射以及 .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

BootLdrI2C 用户模块提供完整的子系统应用程序，与前台 （主要）应用程序共存。 在器件启动或复位时，会始终调用引导加载应用程序。 引导加载程序在系统启动时被调用后，会通过计算前台应用程序 ROM 区域上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，则引导加载程序的验证功能允许前台应用程序执行。 如果两个校验和不相等，则引导加载程序将进入等待循环，等待主机应用程序下载有效的应用程序。 它还使能自己的 I^2C 子系统允许主机传输数据。 如果主机系统发现此接口已被使能，它可能会选择执行自己的一组应用程序。 用户模块工具自动下载到目标应用程序支持两种文件格式。 第一种是名为 <project_name>.txt 的输出文件，第二种是名为 <project_name>.dld 的输出文件。 这两种下载文件格式由不同的演示工具支持。

下载方法

1. 可使用 CY3240 USB-I^2C 桥接器工具包下载 .txt 文件。 用于下载 .txt 格式文件的相关工具在 AN45683 中进行了论述。 有关如何使用 CY3240 USB-I^2C 桥接器的更多信息，请参见应用笔记“PSoC1 通信 – I2C-USB 桥接器使用” AN2352. 附录中也论述了这种下载方法。

2. 另外还支持第二种应用程序下载方法。 这种方法比前一种方法更为复杂，但对于为最终产品开发自定义 I2C 下载应用程序可提供更多信息。 在此简要讨论一下此主机应用程序。 PSoC Programmer 3 的安装目录中提供了应用程序示例和源代码。

<install_path>\Cypress\Programmer\3.xx\Bootloaders\BootLdrI2C\BootLoaderHostApp\…

演示 I^2C 引导加载程序需要用到两个应用程序。 第一个是基于 PSoC 27000 的应用程序 （包含完整的 PSoC 项目），能够将包含嵌入式引导加载程序下载记录的 RS-232 通信转化为发送到引导加载应用程序的 I^2C 数据包。 该 PSoC 项目提供有源代码，应该很容易适应其他 PSoC 器件结构。

第二个应用程序是一个 Microsoft Visual Studio 应用程序 - I^2C Bootloader Host，并提供了一个安装程序和可修改的源代码。 此程序能够读取和解析下载文件 <fiilename>.dld 并将其传输到上述的 PSoC 项目。 该应用程序还提供了可用于 Microsoft Visual Studio 2005 环境下的源代码。 提供此应用程序的目的只在于进行演示，而不准备用于生产或转售目的。

Note 在某些情况下，使用此应用，用户必须在电脑上安装文件 mscomm32.ocx。 此文件可以从 Microsoft 网站上下载，并用 Windows/ 附件 / 命令提示符窗口和以下命令来安装：

> regsvr32 mscomm32.ocx

文件 *regsvr32.exe* 位于 Windows 安装文件夹 windows/system32 (winXP) 之下。

文件 *mscomm32.ocx* 可以通过在 Microsoft 网站上搜索文件名 “mscomm32.ocx” 来进行下载。

引导加载程序工具

通过右键单击用户模块图标，可从快捷菜单中选择几项工具。

特殊版本的 boot.tpl 和 custom.lkp 可以加入项目中或从项目中除去。 从主菜单中选择 “ 工具恢复默认引导 ” 文件。 如果 BootLdrI2C 用户模块已经删除，则恢复默认引导文件的选项将不可以从本用户模块图标上选用， 但是能够从 PSoC Designer 主菜单的工具选项卡内调用。

生成校验和 - 一旦用户项目已经正确建立，用户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。 如果用户需要的话，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编译过程中发生了编译或连接错误，而且没有成功地生成十六进制文件，则将报告错误，但不在 PSoC Designer 的编译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。 为了调试编译错误，必须使用处于引导加载程序工具菜单以外的传统编译和生成流程。

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。 生成三种类型的下载文件: (<project_name>.txt、<project_name>.dld 和 <project_name>.iic)。 针对 “ 仅用于引导加载程序的 I2C” 或 “ 完整的 I2C API 支持 ” 选项，将生成 <project_name>.txt 和 <project_name>.dld。 针对 “ 纯软件引导加载程序 ” 选项，将生成 <project_name>.iic。 这些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。 上述任一下载文件都可以部署到现场应用程序中以升级 PSoC 器件。

dld、txt 和 iic 下载文件由 BootLdrI2C 用户模块工具生成，并在工作区浏览器的 “Output Files” 中列出。

Note 目前仅此用户模块数据手册包含的器件支持 <project_name>.txt 文件 （除非在另一个数据手册中指定）。

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 “Device Editor ” 视图内右键单击 “I²C 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。 在成功编译以前，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。

提供的特殊文件

通过选择**引导加载程序工具**菜单并单击 **获取文件**，可以访问一些重要文件。 一个与器件具体对应的 boot.tpl 文件放置在主项目目录下，并且还有一个称为 custom.lkp 的文件以及一个预定义的 flashsecurity.txt 文件。 所有这些文件的原始版本均放置在项目备份目录下。 每个文件用途的简要说明如下：

Boot.tpl。 - 此文件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 文件内的以下代码模块：

- -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。 在最后三行之内或之后所做的改动都会导致发生错误或被丢弃。 可以更改 custom.lkp 文件的其余部分。 如需对项目的存储器分配进行调试，可通过在第 1 个空格处插入分号对所有这三行做出注释。 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 文件包含的是自动生成的 ROM 区域，这些区域是依照用户模块参数所定义主要代码区块。 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。 如需创建最终的项目，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，这个文件并不是 PSoC Designer 所直接使用的文件。 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，将这个 flashsecurity 文件覆盖似乎并不方便，因为这样会要求开发者重复进行修改。 如有必要，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 提供的默认文件。 此文件内的数据加入到 .hex 文件，并指定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。 由于读保护和写保护均内置在编程后的 PSoC 内，因此该文件必须在引导加载程序第一次部署之前进行正确的配置。

I²C 中断处理

标准 BootLdrI2C 用户模块可以选择提供一份 I²C 中断处理模块的前台副本。 这段代码副本与 API 一起放置在可升级存储器中，能够适用于功能完整的 I²C 从器件。 引导加载程序本身维持着一个内部实用程序，此实用程序可处理传送至引导加载程序的 I²C 数据。 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）。

模块入口参数

所有存储器参数均从引导加载程序的 0x00 至 0xFF 模块进入 （模块的最大编号取决于器件系列，编号范围在 0x4f 到 0x1FF 之间）。 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。 由于某些 PSoC 器件只能存储 64 字节的闪存模块，而有些器件则可以使用 128 字节的模块。 这种方式在正确维持项目代码不同节段之间的边界时较为简单。

大多数 PSoC 部件的模块大小为 64 字节。 本数据手册中的 PSoC 器件模块为 128 个字节。 两个重要事实：

1. 所有引导加载程序都需要写入闪存。
2. PSoC 只能按 “Block （区块）” 写入闪存。

因此，对于引导加载应用程序而言，将存储器视为一组将要写入的 “Block” 会更有用。

为了从模块转换为绝对地址，需要做以下乘法： Abs_addr = block_number X 区块大小。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，所有区块均以十六进制相互分隔，因此，乘以 0x40 （64-byte 区块）或 0x80 （128-byte 区块），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为一个模块。 （由于模块 0 起始于地址 0，因此必须将 128 字节模块视为：一半为 “ 偶数 ” 部分，代表下 （地址）半部分；一半为 “ 奇数 ” 部分，代表上 （地址）半部分）。

查看十六进制文件，并了解所要操作部分的闪存模块大小。

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，可能存在要在 BootLdrI2C 用户模块文件内部进行的额外调整。 这些包含在 BootLdrI2C_Bootloader.inc 文件中，其中一段包含下列等式：

```
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 等式让用户能够加长、缩短或缩写无限长度的代码，这些代码会在一旦用户命令调用了引导加载程序后未从主机收到任何通信时发生超时。 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。

直流和交流电气特性

有关 PSoC 器件的电气特性，请参见该器件的数据手册。

放置

用户模块会消耗 I2C/SPI 通信模块。 放置用户模块时，可以选择 **“I2C Operation for Boot loader only”** 或 **“I2C Operation for Boot loader only”**. 如果计划仅将 I^2C 接口用于引导加载器件，请选择 **“Full I2C API Support with Bootloader”**. API 将限制为引导加载程序所使用的功能，同时减少闪存和 RAM 的使用。

参数和资源

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。 对项目进行汇编和测试后，用户可以选择调整模块大小，以优化内存使用情况。

Figure 3. 默认参数

Full API Support option

Parameters - BootLdrI2C_1

Name	BootLdrI2C_1
User Module	BootLdrI2C
Version	1.1
Slave_Addr_HEX	0x1
Boot_Loader_Addr_HEX	0x0
Read_Buffer_Types	RAM ONLY
Communication_Service_T	Interrupt
ApplicationCode_Start_Blo	0x14
Bootload_when_CKSUM_I	ENABLE_(deployment)
BootLoaderKey	0001020304050607
Flash_Program_Temperatu	-40C
Ignore_N_I2C_Prefix_Byte	2
BootLdrI2C_ver	0x1000
I2C Clock	100K Standard
I2C_Pin	P[1]0-P[1]1

SW Only option

Parameters - BootLdrI2C_1

Name	BootLdrI2C_1
User Module	BootLdrI2C
Version	1.1
Boot_Loader_Addr_HEX	0x0
I2C Clock	100kHz
I2C_Pin	P1[0]-P1[1]
Interrupt_out	Disable
Interrupt_out_Port	None
Interrupt_out_Pin	None
ApplicationCode_Start_Blo	0x14
Bootload_when_CKSUM_I	ENABLE_(deployment)
Run_App_if_CKSUM_OK	ENABLE
BootLoaderKey	0001020304050607
Flash_Program_Temperatu	-20C
Ignore_N_I2C_Prefix_Byte	2
BootLdrI2C_ver	0x1000

以上显示了默认参数示例。 这些参数在用户模块的源代码中可能定期进行更新，更新后的参数可能会与示例中的参数有所不同。

所有缓冲区名称的定义均体现了其对 I²C 主控的用途。 例如，I2Cs_pRead_Buf 指包含了要由 I²C 主控读取的数据的 RAM 位置。

Slave_Addr_HEX

这是一个从器件和多主控从器件 (Slave and MultiMasterSlave) 参数。 此参数选择一个 7-bit 的从器件地址，I²C 主控将采用此地址作为从器件或处于从器件模式下的多主控从器件的地址。 有效的选择项为 0x00-0x7F。由于它是地址的上 7 位，实际地址将在代码内显示为双倍。

Boot_Loader_Addr_HEX

选择 I2C 主控使用的 7-bit 从器件地址，可对 I2C 引导加载程序从器件寻址。 有效的选择项为 0 - 7Fh。 由于它是地址的上 7 位，实际地址将在代码内显示为 2 倍。 参数值必须不同于 Slave_Addr_HEX 参数值。

Read_Buffer_Types

选择数据读取所支持的缓冲区的类型。 有 2 种选择项可用： “ 仅 RAM”(RAM ONLY) 或 “RAM 或闪存 ”(RAM OR FLASH)。 “ 仅 RAM”(RAM ONLY) 选项会删除支持直接闪存 ROM 读取所要求的代码和变量。 选择 “RAM 或闪存 ”(RAM OR FLASH) 项能够提供用于读取 RAM 缓冲区或闪存 ROM 缓冲区以便将数据发送到主控的代码和变量支持。 如果选择了 “RAM 或闪存 ” 项，则可以使用 API 调用来选择是使用 RAM 读取还是使用闪存读取缓冲区。

Interrupt_out/Interrupt_out_Port/Interrupt_out_Pin

（只在纯软件引导加载程序提供此参数）当引导加载程序等待 “enter-bootloader” 或 “run-app” 命令时，会启用一个可配置的输出脉冲。 进入引导加载程序后，脉冲在每个模块传输完后的闪存写入期间为低脉冲。 占空比和脉冲周期可通过包含文件进行配置。 有关纯软件引导加载程序的详细信息，请参见此用户模块数据表中的描述。

Communication_Service_Type

Note 纯软件引导加载程序中未提供此参数。

此参数允许用户在基于中断的数据处理策略和轮询策略之间做出选择。 在基于中断的策略中，数据传输针对预先定义的缓冲区而启动。 数据随后在后台以最快速度移入或移出缓冲区。 其中还包含一个用于处理数据移动的中断服务子程序 (ISR)。 在选择轮询数据处理策略时，用户对何时执行数据移动操作拥有控制权。 为了实现轮询策略，用户必须定期调用函数 BootLdrI2C_Poll() （准确的实例名称请参见 I2C.h 文件）。 每次轮询函数被调用时，将有 1 个单字节进行传输。 其他 I²C 函数的使用完全相同。 在中断延迟极为重要 （且异步通信中断也可能导致问题）的情况下，可使用轮询通信策略。 另一种使用方式是在用户要求对何时进行数据传输拥有绝对控制权的情况下。 轮询的一个缺点是，当 I²C 状态机启用时，总线将会在每个字节后自动停顿，直到轮询函数被调用时为止。 轮询函数只在 I²C 的从器件和多主控从器件实现方式下可用。 “ 单一主控 (Single Master)” 实现方式提供了支持字节式数据传输的 API 函数。

I2C 时钟

具体指定了运行 I²C 接口所需的时钟频率。 可供选择的时钟频率有 3 种：

- 50 K 标准
- 100 K 标准
- 400 K 快速 （在 CPU_Clk_speed 大于 6 MHz 时）

如果加电时检测到无效的校验和，引导加载程序默认的时钟频率将是 100Khz。 开发人员可通过对引导加载程序 asm 做出永久性的修改来定制此频率 （搜索标签 “UserCode_BODY6”）。

I2C_Pin

从端口 1 选择用于 I^2C 信号的引脚。 无需为这些引脚选择适当的驱动模式，PSoC Designer 会自动完成这项选择。

ApplicationCode_Start_Block

这是分配给用户应用程序的第一个代码区块。 此代码必须可引导加载 / 可写入。 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。

默认值显示在上图中。

Bootloader_when_CKSUM_fails

对应用程序进行调试时，禁用校验和功能会给开发带来便利。 如果应用程序校验和与校验和模块中存储的校验和不匹配，可以使用此参数禁用加电后自动进入引导加载程序的功能。

默认值为 Enable_(Deployment)。

Bootloader_Key

这是加在发送给引导加载应用程序的数据操作前面的键值，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被调用。

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内存的保持性能下降。 基于这个原因，用户应当采取预防措施以确保引导加载程序绝对不会在本参数设置值 20 摄氏度以上的温度下运行。 有关详细信息，请参见赛普拉斯器件规格书。

Run_App_if_CKSUM_OK

（仅适用于纯软件引导加载程序）此参数允许应用程序在校验和已验证的情况下自动运行。 当此功能处于启用状态时，进入引导加载程序的唯一方法是使用应用程序中的 BootLdrI2C_bootLoaderStart() 函数。 请参见纯软件引导加载程序的相关流程图。

Ignore_N_I2C_Prefix_Bytes

Note RS-232 至 I^2C 转换器项目会将 2 个前置字节发送到器件。 因此，在使用所提供的演示应用程序时，本参数的正确设置值为 2。 此设置值也适用于一些特定的基于 I^2C 的 SM 总线协议。本参数让用户能够配置引导加载程序以忽略可变数量的前置字节。

应用于纯软件引导加载程序时，此参数具有特别的含义。 纯软件引导加载程序使用其中的两个字节作为协议字节本身。 这意味着，应该将主机协议字节添加至纯软件引导加载程序所使用的两个字节。 纯软件引导加载程序所允许的前置字节不能少于两个。

BootLdrI2C_ver

此参数是引导加载程序的版本。 内部固件目前不使用此项参数，但可作为校验和模块的一部分提供。 这项参数可以由用户设置用于验证引导加载程序可执行代码的正确版本。

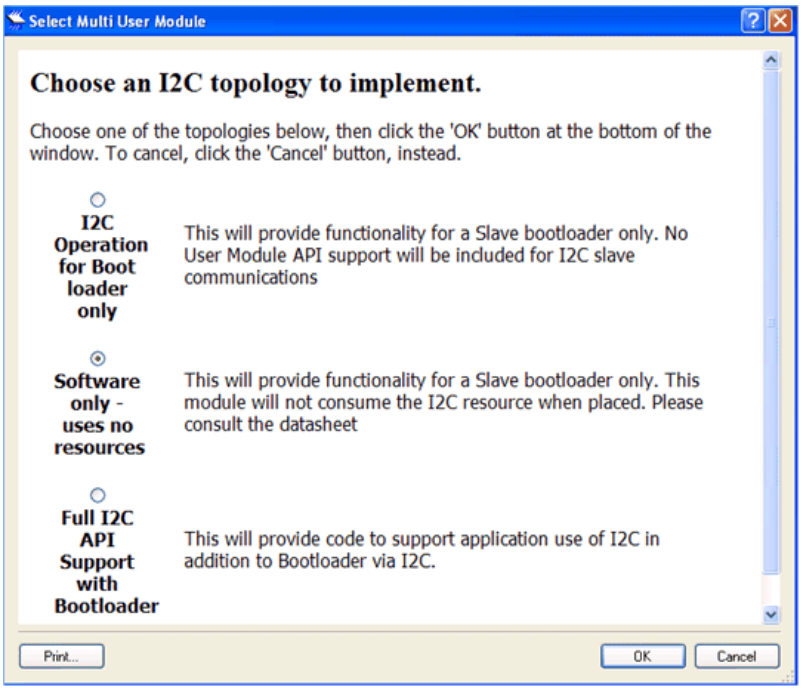
I^2C 拓扑结构选项

放置引导加载程序用户模块时，必须确定为引导加载程序项目实现哪种 I^2C 拓扑结构。

仅用于引导加载程序的 I²C 操作 - 此选项提供仅用于引导加载程序的 I²C 通信。对于 I2C 从器件通信将不提供用户模块 API 支持。如果应用程序所使用的 I²C 通信仅用于引导加载，请选择此选项。

纯软件（不占用资源）- 此选项同样提供仅用于引导加载程序的 I²C 通信。对于 I²C 从器件通信将不提供用户模块 API 支持。与第一个选项不同的是，此选项不会消耗 I²C 资源。如果计划让 I²C 通信仅用于引导加载并且让 I²C 资源用于其他用途，请选择此选项。

引导加载程序的完整 I²C API 支持 - 此选项提供代码以支持 I²C 以及通过 I²C 的引导加载程序的应用。如果计划让 I²C 通信在应用程序中除引导加载外还用于其他用途，请选择此选项。



常见问题

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。其中包括不更新 PSoC Designer 和引导加载程序用户模块，以及不更改编译器。

最初，引导加载程序和应用的编译是一同进行的。但是，在部署可引导加载的系统之后，仅对应用部分进行了重新编程。新应用或修改后的应用应与相同版本的引导加载程序用户模块一同进行编译，以使新应用与最初部署的引导加载程序相匹配。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。只要不更改开发环境，就可以消除兼容性风险。

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相互兼容。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是一同编译的，因此假如一对引导加载程序和应用所使用的开发工具不相符，则无法对它们进行调试。

HI-TECH 编译器的 RAM 分配问题

一些用户模块使用大型阵列的存储器或第 0 页上的大量 RAM：

- 用户模块存储器的默认位置是第 0 页
- 引导加载程序需要第 0 页上处于最低位置的存储器
- HI-TECH 本地变量和 InterruptRAM 的默认位置是第 0 页

由于所有这些都在争夺 RAM 第 0 页上的空间，因此第 0 页上的存储器可能会耗尽。 如果出现此问题，请选择 “ 项目 ”>“ 设置 ”>“ 编译器 ”，然后在 HI-TECH 的 “ 选项 ” 框中添加一行：

```
--AUTOBANK=1
```

此选项会将自动的 C 变量移到内存页 1。 选择的内存页不能超过正在使用的器件可用的最大值。 有关 --AUTOBANK 选项的详细信息，请参见 HI-TECH 手册。

看门狗定时器的内部使用

与看门狗定时器的协同连接至以下全局参数：包含在 globalparams.inc 文件内的 WATCHDOG_ENABLE 参数。 如果项目用到了看门狗定时器，连接至这个全局参数的有条件编译的代码会在计算校验和以及下载操作期间自动设置看门狗定时器。 CPU 时钟频率将影响看门狗定时器的更新速度。 实际可以设定的看门狗定时器最小值约为 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 “BootLoader Tools - Get Files ” 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。

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。 出于最终生产考虑，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。

不正确可重定位代码起始地址 （连接器参数），仅限 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存在很大的不同，因此可重定位代码起始地址将几乎肯定需要进行改变。 这是连接器在试图将多个模块代码连接到同一地址时所发生的错误现象的常见原因。 该参数可以在主菜单选项卡下找到：**项目 > 设置 > 连接器**。 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 I^2C 版本的引导加载程序，将此数值设置为 0x8C0 或者 0x900 应该足够了 （如果其他参数使用了默认值）。

Note 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。

存储器重叠

为了纠正可重定位代码起始地址，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区块，这样就可以开始找到内存重叠的根本原因。

电源稳定性

电源噪声、短时脉冲、电压降低、缓慢的电压上升过程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。 相对于电源上升来说，程序执行要更快，在某些情况下，正在对 flash 编程时，可能仍有机会碰到电源电平在变化。 其中一个例子是在电源加载过程中对 flash 进行写入操作。用户应当谨慎评估自己

的用户模块及其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 flash 数据保持不良。

引导加载过程中应用程序未完全停止

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。 在引导加载过程中，中断矢量地址在写入新地址前会变为零。 如果不禁用中断，运行中断将导致随机复位。 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。

使用 I^2C 接口可识别多个地址。 在应用程序未完全关闭时有可能开始与引导加载程序进行通信。 应该实施明确进入引导加载程序的方法，这包括完全关闭正在运行的应用程序。

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。这种结果很容易在无意识的情况下发生例如，包含一个简单 while(1)； 循环的 main{} 程序，将绝对不会返回且绝对不会进入引导加载程序。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。 有多种策略可以解决这一问题。 此用户模块内不包含任何默认方法。 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. 启用 I^2C 应用程序资源并创建一个能够让器件进入引导加载程序的 I^2C 命令。 通常情况下，如果主例程启用了 I^2C，就可以使用引导加载程序地址来使器件进入引导加载程序。
4. 如果器件没有定期得到处理，使用看门狗定时器可复位器件。 这种方法可以结合上述策略之一，以实现一个看门狗定时器 (WDT) 中断以启动可引导加载状态。 在根据看门狗定时器复位条件进行复位时，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。 参见 《技术参考手册》。
5. 开发两个项目，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。 请牢记，引导加载的意义就是对器件进行*部分*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应当完全一致，可重定位代码起始地址也应当完全一致 （这不同于第一个应用程序模块）。 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。
6. 基于 PSoC 的转换器应用程序在引导加载操作失败时挂起。 这是一种基于 if 定义的超时设置，可以通过配置让基于 PSoC 的转换器在可变软件循环后放弃通信尝试。 为了进行调试，用户可能希望将此软件打开或关闭。 检查项目的源代码可找到这个超时开关。
7. 电源噪声、短时脉冲、电压降低、缓慢的电压上升过程以及不良的连接。 所有这些电源问题都可能造成闪存编程中出现难以诊断的问题。 相对于电源上升来说，程序执行要更快，在某些情况下，正在对 flash 编程时，可能仍有机会碰到电源电平在变化。 其中一个例子是在电源加载过程中对 flash 进行

写入操作。用户应当谨慎评估自己的使用模型及其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表现为闪存保持性不良。

应用程序编程接口

应用程序编程接口 (API) 固件提供了高级命令来支持多字节传输的发送和接收操作。 读取缓冲区可以在 RAM 存储器或闪存中设置。 写入缓冲区只能设置在 RAM 存储器之内。

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 会为指定项目中此用户模块的第一个实例分配 BootLdrI2C_1。 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 BootLdrI2C。

Note

在这里，如同所有用户模块 API 中的一样，A 和 X 寄存器的值可能通过调用 API 函数发生更改。 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。 选择这种 “ 寄存器易失 ” 策略是为了提高效率，并且从 PSoC Designer 的 1.0 版本起已开始使用。 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。

对于大型储存器模块驱动， 保存 CUR_PP、IDX_PP、MVR_PP 以及 MVW_PP 寄存器中的所有值也是调用程序的职责。 尽管部分寄存器现在可能不可修改，但是无法保证在将来的版本中也会如此。

ENTER_BOOTLOADER

说明：

用于完全设置引导加载程序并准备下载新应用程序的例程。 一旦被调用，该子程序将不会返回，除非发生超时或复位。

GenericBootloaderEntry 函数名称始终位于同一个物理 ROM 地址，从而使以后编译的应用程序仍然可以使用此函数调用进入引导加载程序。

C 原型：

```
void ENTER_BOOTLOADER(void);
```

汇编程序：

```
lcall ENTER_BOOTLOADER
```

参数：

无

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。

BootLdrI2C_Start

说明：

为了保持一致性而提供的空例程。

C 原型：

```
void  BootLdrI2C_Start(void);
```

汇编程序：

```
lcall BootLdrI2C_Start
```

参数：

无

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。

BootLdrI2C_DisableInt

说明：

通过禁用 SDA 中断来禁用 I^2C 从器件。 执行与 I2Cs_Stop. 相同的操作

C 原型：

```
void  BootLdrI2C_DisableInt(void);
```

汇编程序：

```
lcall BootLdrI2C_DisableInt
```

参数：

无

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。

BootLdrI2C_EnableInt

说明：

启用 I^2C 中断以便实现启动条件检测。 请牢记要通过使用以下这个宏来调用这个全局中断启用函数: M8C_EnableGInt.

C 原型：

```
void  BootLdrI2C_EnableInt(void);
```

汇编程序：

```
lcall BootLdrI2C_EnableInt
```

参数：

无

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。

BootLdrI2C_Poll() 和 BootLdrI2C_BootLdr_Poll()

说明：

在 Communication_ Service_Type 参数设置为 “ 轮询 ”(Polled) 时使用。 此函数提供了一个由用户控制的进入 I/O 处理例程的入口。 如果将 Communication_Service_Type 参数设置为 “ 中断 ”(Interrupt)，则此函数不做任何操作。

C 原型：

```
void BootLdrI2C_Poll(void);
void BootLdrI2C_BootLdr_Poll(void);
```

汇编程序：

```
lcall BootLdrI2C_Poll
lcall BootLdrI2C_BootLdr_Poll
```

参数：

无

返回值：

无

副作用：

每次调用该例程时，将处理一个 I^2C 事件，而状态变量将得到更新。 事件的构成可以是一个故障状况、一个 I/O 字节，或在某些特定情况下，一个停止状况。 调用该例程可能有三种结果：

1. 如果没有可用数据，则不执行任何操作。
2. 如果有一个可用地址或数据字节，则对此地址或数据进行接收或发送。
3. 如果外部主控已完成其写入操作，则处理停止 “ 事件 ”。 在写入操作结束阶段处理停止状态时仅更新状态变量。 如果停止状态得到处理时 I^2C 字节处于待处理状态，则必须再次调用 I2CHW_Poll 函数对其进行处理。

I2CHW_Poll() 函数在 Communication_Service_Type 设置为 “ 中断 ”(Interrupt) 时不会产生任何作用。 在总线上检测到启动 / 重启条件和地址时，总线将处于停顿状态，直至 I2CHW_Poll() 函数被调用。 如果地址被否认，则为该数据操作发送的后续字节将会被忽略，直到检测到另一个启动 / 重启和地址，否则，I^2C 总线上的每个数据字节都将处于停顿状态，直至 I2CHW_Poll() 函数被调用。

I^2C 数据将会由 I^2C 硬件停顿，直到在 Communication_Service_Type 设置为 “ 轮询 ”(Polled) 的情况下调用此函数。 对于已接收的数据，总线将在字节末尾并在生成 ACK/NAK（通过将 SCL（时钟）线保持在低位）之前停顿。 对于已发送的数据，总线将在 ACK/NAK 位在外部生成之后立即停顿。

这两个函数符合以下限制条件。 如果引导加载程序处于活动状态并且可以通过外部应用程序 I^2C 命令进入，则应该使用 API BootLdrI2C_BootLdr_Poll()。 如果检测到不是引导加载程序地址的 I^2C 地址，则会对此地址进行测试，并将其传递到同样测试地址的前台 I^2C 中断进程中。 如果引导加载程序处于非活动状态，则使用 BootLdrI2C_Poll() API 不会将此 I^2C 地址提供给内部引导加载程序数据处理。 相反，地址和后续数据将会直接传递到前台子程序中。

BootLdrI2C_Stop

说明：

通过禁用 I^2C 中断来禁用 I2CHW。

C 原型：

```
void  BootLdrI2C_Stop(void);
```

汇编程序：

```
lcall BootLdrI2C_Stop
```

参数：

无

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。

BootLdrI2C_EnableSlave

说明：

通过设置 I2C_CFG 寄存器中的 “Enable Slave”（启用从器件）位，为 I^2C HW 模块启用 I^2C Slave 功能。

C 原型：

```
void  BootLdrI2C_EnableSlave(void);
```

汇编程序：

```
lcall BootLdrI2C_EnableSlave
```

参数：

无

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。

BootLdrI2C_DisableSlave

说明：

通过清除 I2C_CFG 寄存器中的 “Enable Slave” （启用从器件）位，禁用 I^2C Slave 功能。

C 原型：

```
void  BootLdrI2C_DisableSlave(void);
```

汇编程序：

```
lcall BootLdrI2C_DisableSlave
```

参数：

无

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。

BootLdrI2C_InitWrite

说明：

仅为 FullAPISupport 提供 API。

BootLdrI2C_InitWrite 例程可初始化一个数据缓冲区指针，让从器件用来存放数据，并将该缓冲区的计数字节归零。

C 原型：

```
void  BootLdrI2C_InitWrite(BYTE * pBootLdrI2C_WriteBuf, BYTE  BootLdrI2C_Write_Count);
```

汇编程序：

```
mov A, Write_Count
push A
move A, >pWriteBuf
push A
mov A, <pWriteBuf
push A
lcall BootLdrI2C_InitWrite
```

参数：

pWriteBuf： 指向 RAM 缓冲区位置的指针。 Write_Count: 写入缓冲区的长度。

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。

BootLdrI2C_InitRamRead

说明：

仅为 FullAPISupport 提供 API。

BootLdrI2C_InitRamRead 子程序可初始化一个数据缓冲区指针，让从器件用来从中检索数据，并将相同缓冲区计算字节的值归零。

C 原型：

```
void BootLdrI2C_InitRamRead(BYTE * pBootLdrI2C_ReadBuf, BYTE  BootLdrI2C_Read_Count);
```

汇编程序：

```
mov A, Read_Count
push A
move A, >pReadBuf
push A
mov A, <pReadBuf
push A
lcall BootLdrI2C_InitRamRead
```

参数：

pReadBuf： 指向 RAM 缓冲区位置的指针。 Read_Count: 读取缓冲区的长度。

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。

BootLdrI2C_InitFlashRead

说明：

仅为 FullAPISupport 提供的 API

BootLdrI2C_InitFlashRead 例程可初始化一个闪存数据缓冲区指针，让从器件用来从中获取数据，并将该缓冲区的计数字节归零。

C 原型：

```
void  BootLdrI2C_InitFlashRead(const BYTE * pBootLdrI2C_flashaddr,  unsigned int
BootLdrI2C_Read_CountHI);
```

汇编程序：

```
mov A, >Read_Count
push A
mov A, <Read_Count
push A
move A, >pflashaddr
push A
mov A, <pflashaddr
push A
lcall BootLdrI2C_InitFlashRead
```

参数：

pflashaddr： 指向闪存数据缓冲区位置的指针。 Read_Count: 读取缓冲区的长度。

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。

将会清除读取状态位。

BootLdrI2C_bReadI2CStatus

说明：

仅为 FullAPISupport 提供 API。

在 I2CStatus 变量内返回数值。

C 原型：

```
BYTE  BootLdrI2C_bReadI2CStatus(void);
```

汇编程序：

```
lcall BootLdrI2C_bReadI2CStatus ; Accumulator contains the status on return
```

参数：

无

返回值：

bI2CStatus - 状态数据

常量	值	说明
I2CHW_RD_NOERR	01h	主控读取数据，正常 ISR 退出
I2CHW_RD_ OVERFLOW	02h	主控所读取的数据字节超出了可用字节数
I2CHW_RD_ COMPLETE	04h	一项读取操作已启动且已完成。
I2CHW_READFLASH	08h	下一个读取操作来自闪存位置
I2CHW_WR_NOERR	10h	主控成功写入了数据
I2CHW_WR_OVERFLOW	20h	主控向写入缓冲区写入了过多字节
I2CHW_WR_COMPLETE	40h	主控写入操作完成，因为新地址或停止操作
I2CHW_ISR_ACTIVE	80h	I^2C ISR 尚未退出并处于活动状态

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。

BootLdrI2C_ClrRdStatus

说明：

仅为 FullAPISupport 提供的 API

清除控制 / 状态寄存器中的状态位，但不改变缓冲区地址、计数或闪存 /Ram 读取位。

C 原型：

```
void  BootLdrI2C_ClrRdStatus(void);
```

汇编程序：

```
lcall BootLdrI2C_ClrRdStatus
```

参数：

无。

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。

BootLdrI2C_ClrWrStatus

说明：

仅为 FullAPISupport 提供的 API

清除控制 / 状态寄存器中的状态位，但不改变缓冲区地址、计数或闪存 /Ram 读取位。

C 原型：

```
void  BootLdrI2C_ClrWrStatus(void);
```

汇编程序：

```
lcall BootLdrI2C_ClrWrStatus
```

参数：

无。

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。

纯软件引导加载程序 （不使用 I²C 资源或中断）测试版

这个新版本的 I²C 引导加载程序完全基于软件。 它在运行时不使用中断，并且在放置时不会消耗 I²C 资源。 这样可以释放资源，以便在应用程序中使用。

应该将纯软件引导加载程序的使用模型视为：作为单独的应用程序存在，在复位状态下运行并允许加载新应用程序的引导加载程序。 当应用程序运行时，引导加载程序几乎完全不可见。

纯软件引导加载程序支持的功能

通信格式匹配 EZI2C 协议： 对于 “ 写入 ” 数据操作

<I2C_WRaddr>、<EZI2CSubaddr>、<data written starting at EZI2CSubaddr>、<stop condition>

对于 “ 读取 ” 数据操作

<I2C_RDaddr>、<data read starting at EZI2CSubaddr specified in previous write>、<stop condition>

引导加载程序不消耗 I²C 资源，从而使应用程序更易于使用 I²C。

可选择在校验和正确时自动运行应用程序。

可选择在引导加载程序的引导加载操作启动时清除应用程序的内容。

可选择加快校验和进程。 验证校验和之后，值将保存在闪存中，这表示不需要在每次启动时运行该值。 这样可以加快启动过程。

当引导加载程序等待命令时，将会启用可配置的输出脉冲。 进入引导加载程序后，并且每个模块传输完后，闪存写入期间脉冲为高脉冲。 此脉冲的占空比和周期均可配置。

硬件 I²C 缓冲区可在确认 I²C 地址和数据时用于消除软件延迟。

在初始引导时，将会针对与主机的初始通信预加载状态响应。

限制 （纯软件引导加载程序）

不支持多个纯软件引导加载程序实例。 当存在多个实例时，不能保证纯软件引导加载程序用户模块正常工作。

在引导加载程序处于活动状态时将数据写入 HW I2C 缓冲区之后，主机在请求状态之前必须等待允许读取 HW 缓冲区 （约 10 毫秒）。

纯软件引导加载程序仅提供一个 API 函数。

初始状态 （16 个字节）

进入引导加载程序时将会预加载状态响应，以供主机查询。 此状态由 16 个字节组成。 最终用户可以通过修改 bootloader.inc 文件中的值定制响应。

用户可通过以下操作获取状态：首先将子地址 0 写入引导加载程序地址中，然后从 I2C 缓冲区中读取数据。

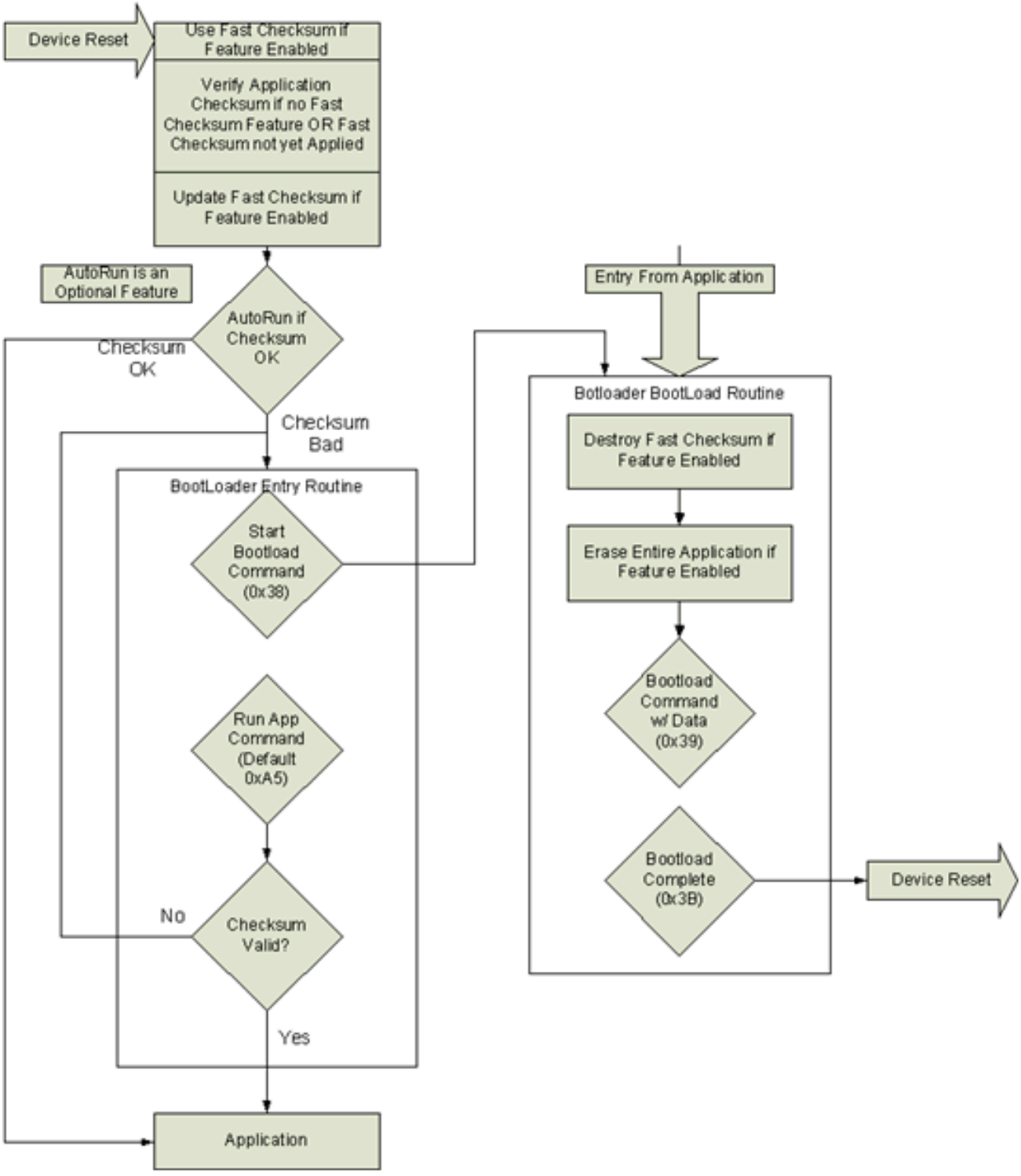
这些值在下表中进行了初始化：

address	Name	Bit 7	Bit 6		Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Master/ PSoC Access
BL,00	HST_MODE	Always 0 (defined later by host)								W/R: 00
BL,01	BootLoaderStatus	0	0	0	BootLdr Mode	1, 1 during ERASE APP		Watchdog reset	Checksum valid	R/W:00
BL,02	BootErrorCode	Inalid Command	Invalid decurity Key	BootLoad Mode	Comm Cksum Error	Flash Protection Error	Flash Checksum Error	Image Verify Error	Boot Completed OK	R/W:00
BL,03	BL_VER_Bhi	Bootloader Version Defined by Bootloader (lmsb) (default 0x10)								R/W:00
BL,04	BL_VER_Blo	Bootloader Version Defined by Bootloader (lsb) (default 0x00)								R/W:00
BL,05	BL_VER_Ah	Bootloader Version Defined by Application (msb) (default 0xff)								R/W:00
BL,06	BL_VER_Alo	Bootloader Version Defined by Application (lsb) (default 0xff)								R/W:00
BL,07	DIP_VERhi	Design IP Version (msb) (default 0xff)								R/W:00
BL,08	DIP_VERlo	Design IP Version (lsb) (default 0xff)								R/W:00
BL,09	APP_IDhi	Application ID (msb) (default 0xff)								R/W:00
BL,0a	APP_IDlo	Application ID (lsb) (default 0xff)								R/W:00
BL,0b	APP_VERhi	Application Version (msb) (default 0xff)								R/W:00
BL,0c	APP_VERlo	Application Version (lsb) (default 0xff)								R/W:00
BL,0d	CID_0	Custom ID #0 (default 0xC1)								R/W:00
BL,0e	CID_1	Custom ID #1 (default 0xC2)								R/W:00
BL,0f	CID_2	Custom ID #2 (default 0xC3)								R/W:00

此表列出了默认值。 最终用户可以修改 “bootloader.inc” 文件中的 “BL_VER***”、“DIP_VER***”、“APP_***” 和 “CID_**” 这几个值。 然而，在 “bootloader.inc” 文件中，如果用户将 “BOOT_LOADER_DEFAULTS” 更改为 0x00，则可以通过更换 “TO_DO_Developer” 变量来设置这些值。

纯软件引导加载程序操作

Figure 4. 纯软件引导加载程序操作流程图



定制纯软件引导加载程序

使用 PSoC Designer 可防止代码生成器更新代码模块。 整个引导加载程序将受到保护。 因此，用户可以更新引导加载程序代码，并且作出的更改不会被代码生成器覆盖。 这种方法是试图缓解以前问题的折中方法。

1. 部署的引导加载程序不得升级为新版本。 可能无法在字段中混合不同版本的引导加载程序。
2. 2. 可以选择修复错误，并添加增强功能或其他定制项。 可在代码中的任意位置保留此代码。

包含文件配置

某些配置可通过更改 UM_Name_bootloader.inc. 文件中的永久 EQU 值进行更改。

Bootloader.inc 文件可用于配置纯软件引导加载程序的某些操作。 用户可在受保护的区域对配置进行更改，这在重新生成代码时不会改变。

Bootloader RUN_APP

此命令的值在 booloader.inc 文件中设置为默认的 0xA5。 如果需要，用户可以通过改变包含文件中的定义修改命令值。

PULSE_COUNT: EQU 0xA00 （默认）“I’m alive” 中断输出脉冲的脉冲周期经验调节 （合法值 0-65535）。

PULSE_COMPARE: EQU 0x10 （默认）“I’m alive” 中断输出脉冲的脉冲宽度经验调节 （合法值 0-255 ）。

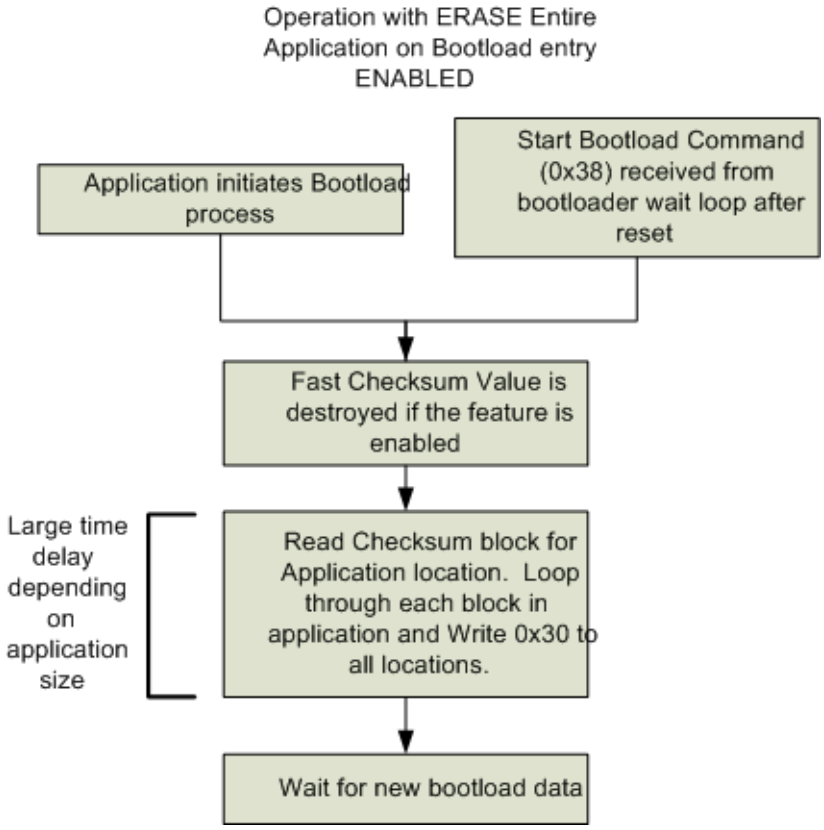
脉冲通过软件定时循环驱动，因此对于之前的两个值，脉冲位于 0xA00 循环中的 0x10 循环 (32/320) 上。

BOOT_TIMEOUT: EQU 00 （默认为 0x40）在引导加载程序循环的特定点上，计数器会进行初始化并减少计数以防止无限循环。

如果计数器的数值减少到零，则将退出循环。 将该值设置为零可以防止退出循环。 这对于调试和逐步进行引导加载过程十分有用。

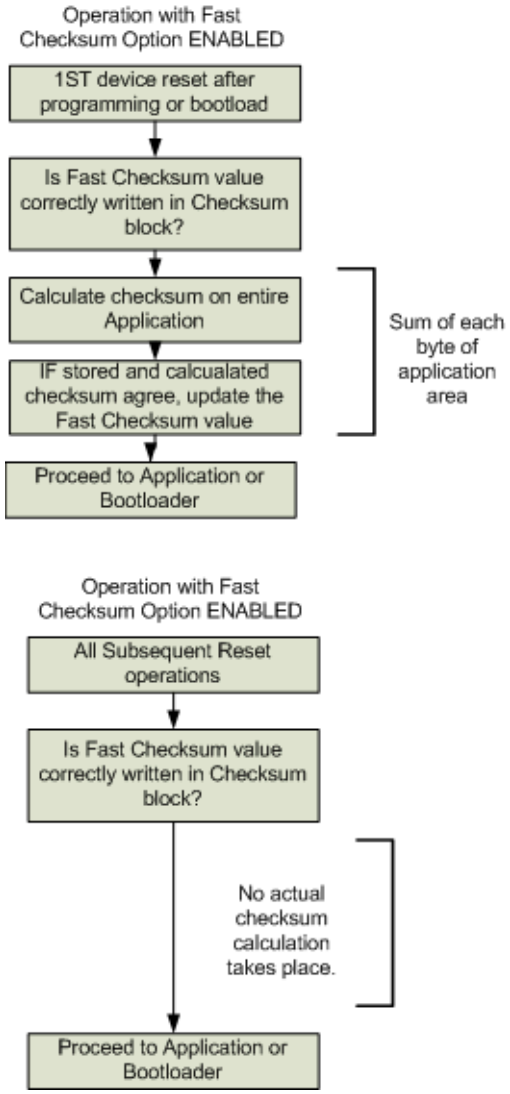
ERASE_APP_BEFORE_BOOTLOAD: EQU 0 （默认为 0/ 关闭）当收到启动引导加载命令时，将会清除整个应用程序。 这包括进入这样的循环：首先清除模块，然后通过值 0x30 （暂停）初始化被定义为应用程序空间的每个模块。 此操作可能需要花费数秒钟的时间。 在清除应用程序的过程中，器件可能并不始终对 I^2C 数据流作出响应。 当器件写入闪存时，将会激活 I2C_Interrupt_out 引脚。 由于桥接器只能延迟固定的时间段，因此，通过 USB-I2C 桥接器使用此功能特别困难。 清除应用程序可能花费的时间变化范围很大。 最佳开发方法可能是针对特定应用程序大小对清除过程进行定时，并定制用于引导加载器件的延迟时间。 如果使用智能主机传输引导加载数据，有可能要等待引脚切换才能继续这一过程。

Figure 5. 清除引导加载上的应用程序



CHECKSUM_FAST_VERIFICATION: EQU 1 （默认为 1/ 打开）在时钟脉冲速率为 3 MHz 的情况下，验证校验和需要约 1.3 秒的时间和约 30K 的 ROM 空间。 在首次验证校验和之后启用此功能时，将会在校验和模块中更新值，表示校验和验证通过。 之后的校验和验证只是为了查看是否已设置该值。 此操作可将验证时间缩短到大约 260 微秒 （指令时钟速率为 3 MHz 时）。

Figure 6. 快速校验和选项



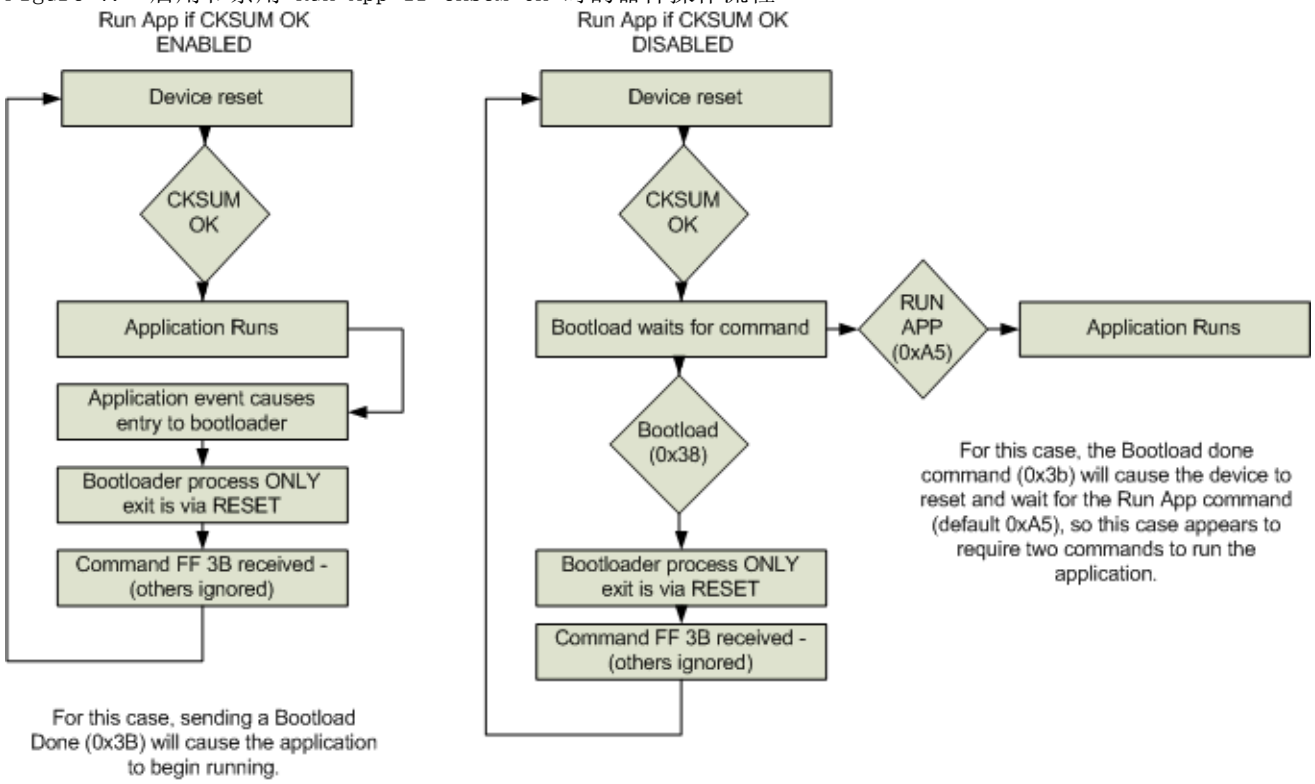
Run_App_if_CKSUM_OK: EQU 1 （默认为 1/ 打开）

如果这一部分的校验和正确，则启用此功能将允许在启动时自动运行应用程序。 此功能可与快速校验和验证功能结合使用，从而使器件快速启动。 仅可使用之前所述的 EnterBootloader() API 进入引导加载程序。 使用此功能会略微改变命令序列，从而退出引导加载程序，并在引导加载之后启动应用程序。 有关此功能提供的操作的附加说明，请参见下图。

请认真查看描述软件引导加载程序操作的流程图。

左列假设可选的 “Auto Run App on CKSUM OK ” 已启用。右例假设 “Run App if CKSUM OK” 参数已禁用。

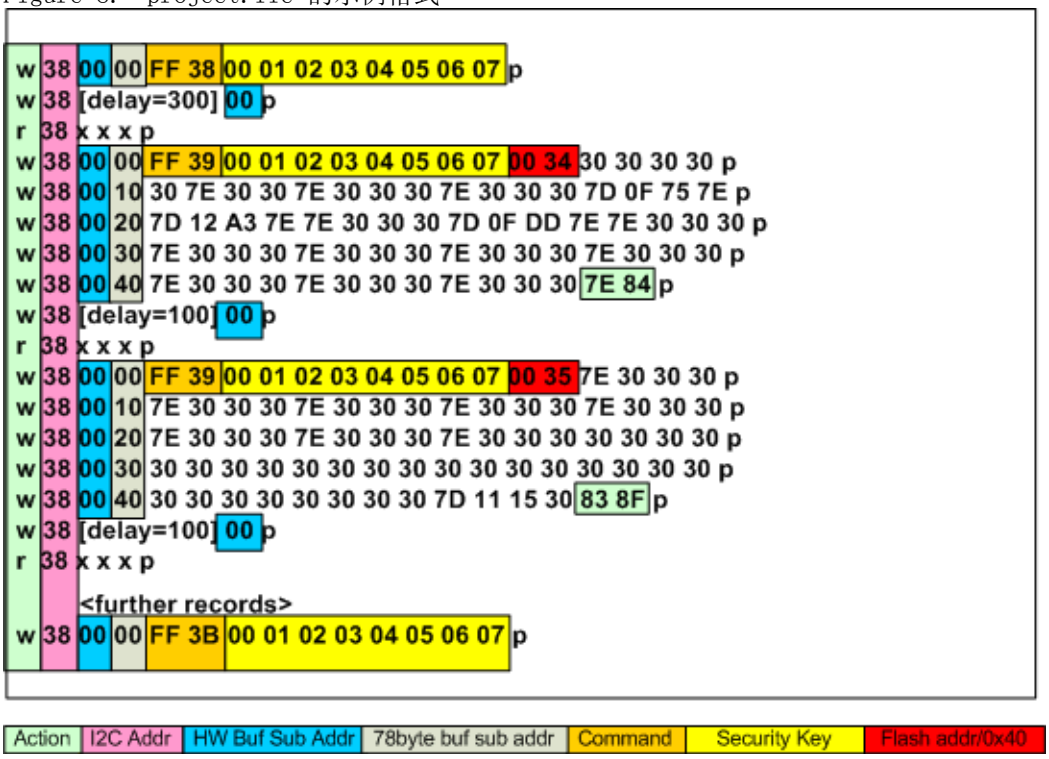
Figure 7. 启用和禁用 Run App if CKSUM OK 时的器件操作流程



为纯软件引导加载程序创建的下载文件

已经为纯软件引导加载程序创建新的下载文件格式。 此文件类似于 .txt 文件输出。 纯软件引导加载程序输出文件的扩展名为 project_name.iic。

Figure 8. project.iic 的示例格式



固件源代码示例

纯软件引导加载程序示例

使用下图中所示的值针对汇编语言和 C 语言示例配置用户模块参数：

Figure 9. 纯软件参数设置

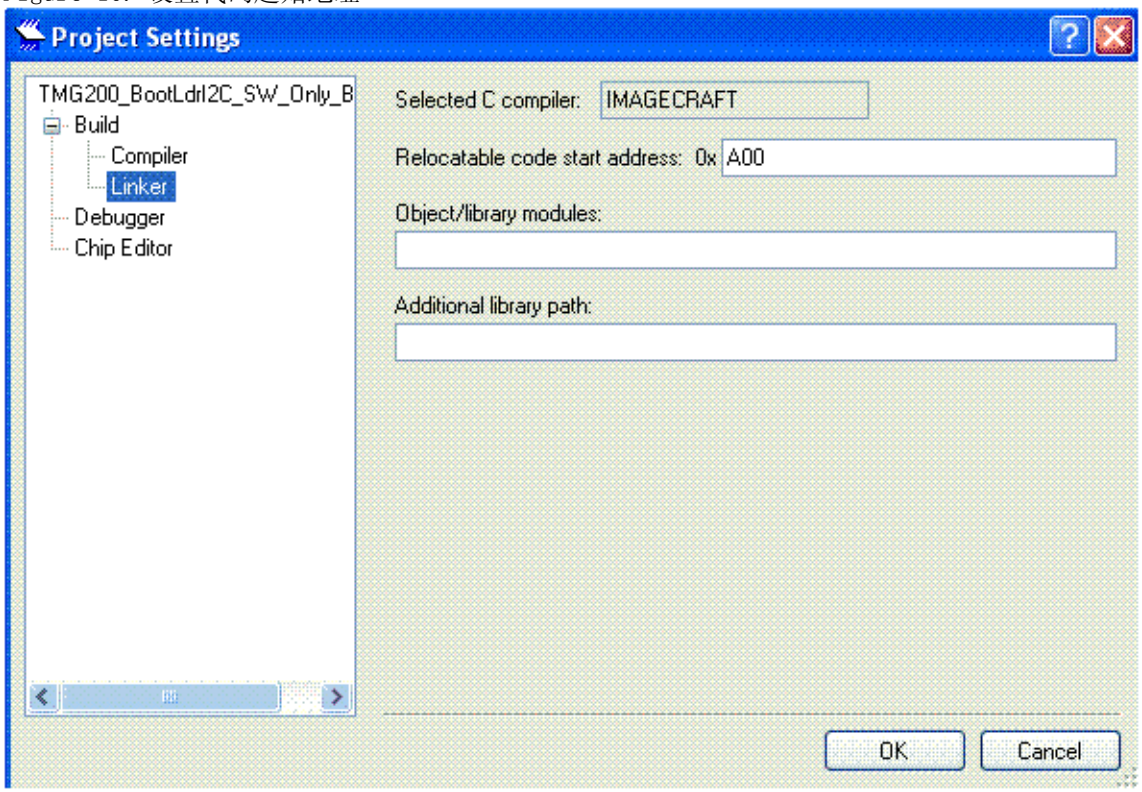
SW Only Bootloader parameters setting

Parameters - BootLdrI2C_1

Name	BootLdrI2C_1
User Module	BootLdrI2C
Version	1.1
Boot_Loader_Addr_HEX	0x0
I2C Clock	100kHz
I2C_Pin	P1[0]-P1[1]
Interrupt_out	Disable
Interrupt_out_Port	None
Interrupt_out_Pin	None
ApplicationCode_Start_Blo	0x14
Bootload_when_CKSUM_I	ENABLE_(deployment)
Run_App_if_CKSUM_OK	ENABLE
BootLoaderKey	0001020304050607
Flash_Program_Temperatu	-20C
Ignore_N_I2C_Prefix_Byte	2
BootLdrI2C_ver	0x1000

确保代码起始地址已正确设置 （仅限 IMAGECRAFT 编译器）。

Figure 10. 设置代码起始地址



以下两个示例显示了纯软件引导加载程序用户模块正常工作。 在此测试中，两个 LED 分别连接到 P0[0] 和 P0[1]。 首先，通过示例 1 对器件进行编程，从而打开 P0[0] 引脚上的 LED。 然后，通过示例 2 引导加载 （使用 USBtoIIC 桥接器 GUI）器件，从而打开 P0[1] 引脚上的 LED。 如果引导加载程序用户模块正常工作，用户将看到 P0[1] 上的 LED 打开，而 P0[0] 上的 LED 关闭。 这基于以下假设：用户已遵照此数据表中的 “ 快速启动 ” 部分将引导加载程序用户模块置于项目中。

```
//-------------------------------------------------------------------------
// Example 1
//-------------------------------------------------------------------------

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


void main(void)
{
    PRT0DM0 = 0xff;
    PRT0DM1 = 0x00;

    while(1)
    {
        PRT0DR = 0x01;
    }
}


//-------------------------------------------------------------------------
// Example 2
//-------------------------------------------------------------------------

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


void main(void)
{
    PRT0DM0 = 0xff;
    PRT0DM1 = 0x00;

    while(1)
    {
        PRT0DR = 0x02;
    }
}
```

引导加载程序的完整 I²C API 支持示例

以下是 I²C 引导加载程序用户模块的实现，该模块对于 “ 仅针对引导加载程序的 I²C 操作 ” 和 “ 引导加载程序的完整 I²C API 支持 ” 选项使用 C 语言编写而成。

Figure 11. 完整 I²C API 支持的参数设置

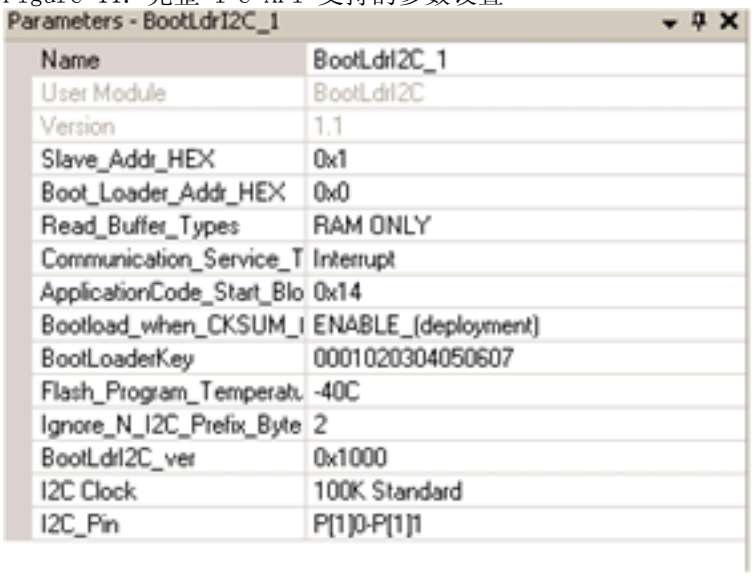
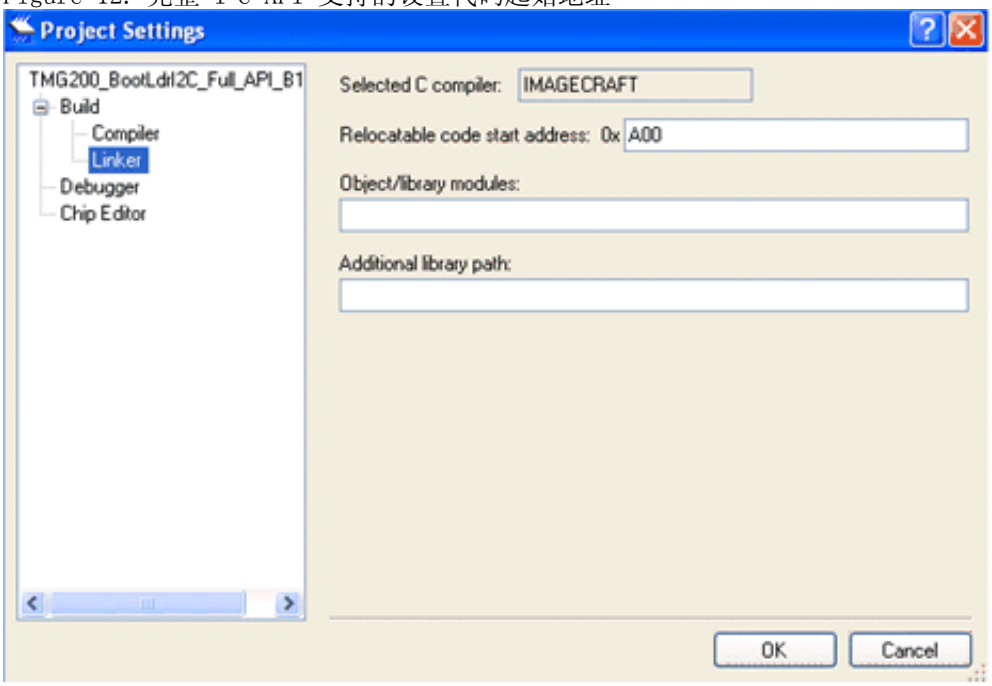


Figure 12. 完整 I²C API 支持的设置代码起始地址




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

```
BYTE result;
WORD wAddr, wByteCount, cTemperature, wByteReadCount;
BYTE pbDataDest[10], pbData[10];
void main(void)
{
    //example application consists of an EEPROM UM, an LED UM,
    //and a 16-bit timer UM.
    //the EEPROM demonstrates that the EEPROM UM can co-exist
    //with the bootloader, the timer sets a duty cycle and the
    //LED blinks at the set duty cycle.
    //The bootloader UM provides the capability to modify
    //the project (LED duty cycles are conveniently visible.)
    //and use the bootloader to download the modified project.
    //by the time the main() function is executed the project
    //has already been checksummed and verified by the bootloader.
    //init EEPROM data
    wAddr = 0;
    wByteCount = 64;
    wByteReadCount = 10;
    cTemperature = 25;

    //start the bootloader running in the background
    BootLdrI2C_1_Start();
    BootLdrI2C_1_EnableSlave();
    BootLdrI2C_1_EnableInt();

    //start blinking the LED
    Counter24_1_Start();
    Counter24_1_EnableInt();
    LED_1_Start();
    M8C_EnableGInt;

#define INCLUDE_LIB_API
#ifdef INCLUDE_LIB_API
    E2PROM_1_Start();
    result = E2PROM_1_bE2Write( wAddr, pbData, wByteCount, cTemperature);
    E2PROM_1_E2Read( wAddr, pbDataDest, wByteReadCount);
    // Insert your main routine code here.
#endif

#define BUSMODE 0xf5
    while(1)
    {
        asm("nop");
    }
}
```

以下是使用汇编语言编写的 I^2C 引导加载程序用户模块的实现。

```
;-----------------------------------------------------------------------
; 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
_main:
    ; Insert your main assembly code here.
    lcall BootLdrI2C_1_Start
    lcall BootLdrI2C_1_EnableSlave
    lcall BootLdrI2C_1_EnableInt

    //start blinking the LED
    lcall Counter24_1_Start
    lcall Counter24_1_EnableInt
    lcall LED_1_Start
    M8C_EnableGInt
.terminate:
    jmp .terminate
```

配置寄存器

此部分描述了 I^2C 引导加载程序用户模块所使用或修改的 PSoC 资源寄存器。

Table 1. 资源 I2C_CFG：Bank 0 reg[D6] 配置寄存器

位	7	6	5	4	3	2	1	0
值	Reserved	PinSelect	Bus Error IE	Stop IE	Clock Rate[1]	Clock Rate[0]	Enable Master	Enable Slave

Pin Select：选择 SCL 和 SDA 作为 P1[5]/P1[7] 或 P1[0]/P1[1]。

Bus Error Interrupt Enable：在总线发生错误时允许 I^2C 中断的产生。

Stop Error Interrupt Enable：在 I^2C 停止条件下允许 I^2C 中断。

Clock Rate[1,0]：从 3 种有效时钟频率中选择：50、100 和 400 Kbps（在 CPU_Clk_speed 大于 6 MHz 时为 400 Kbps）。

Enable Master：启用 I^2C HW 模块作为总线主控。

Enable Slave：启用 I^2C HW 模块作为总线从器件。

Table 2. 资源 I2C_SCR：Bank 0 reg[D7] 状态控制寄存器

位	7	6	5	4	3	2	1	0
值	Bus Error	Lost Arb	Stop Status	ACK out	Address	Transmit	Last Recd Bit (LRB)	Byte Complete

Bus Error：表示检测到总线错误条件。

Lost Arbitration：在多主控模式下，表示此器件的仲裁失败，（器件没有控制总线）。

Stop Status：检测到 I^2C 停止条件。

ACK out：指示 I^2C 模块对所收到的字节进行确认 (1) 或否认 (0)。

Address： 所接收或所发送的字节是一个地址。

Last Received Bit (LRB)： 在发送序列中最后收到的一个位 （第 9 位）的值，来自目标器件的确认 / 否认状态。

Byte Complete： 接收到了 8 个数据位。 对于接收模式，总线在等待确认 / 否认应答时停顿。 在发送模式下，已经接收了确认 / 否认应答 （参见 LRB），而且总线停顿以等待下一项操作的执行。

Table 3. 资源 I2C_DR： Bank 0 reg[D8] 数据寄存器

位	7	6	5	4	3	2	1	0
值	Data							

所接收或所发送的数据。 若要发送数据，该寄存器必须在写入到 I2C_SCR 寄存器之前已经加载。 所接收的数据从此寄存器中读取。 寄存器中可能包含地址或数据。

Table 4. 资源 I2C_MSCR： Bank 0 reg[D9] 主控状态控制寄存器

位	7	6	5	4	3	2	1	0
值	Reserved	Reserved	Reserved	Reserved	Bus Busy	Master Mode	Restart Gen	Start Gen

Bus Busy： 仅用于主控，在检测到任何总线 “ 启动 ”(Start) 条件时置位，在检测到 “ 停止 ”(Stop) 条件时清除。

Master Mode： 表示此器件当前作为总线主控运行。

Restart Gen： 仅用于主控，可以进行设置，生成 I^2C 总线的重复启动。

Start Gen： 仅用于主控，在总线闲置时，产生 I^2C 总线启动信号，并使用数据寄存器 (I2C_DR).) 内的数据发送 I^2C 地址

附录

以下部分包含了用户在创建 I^2C 引导加载程序时可能用到的附加信息。

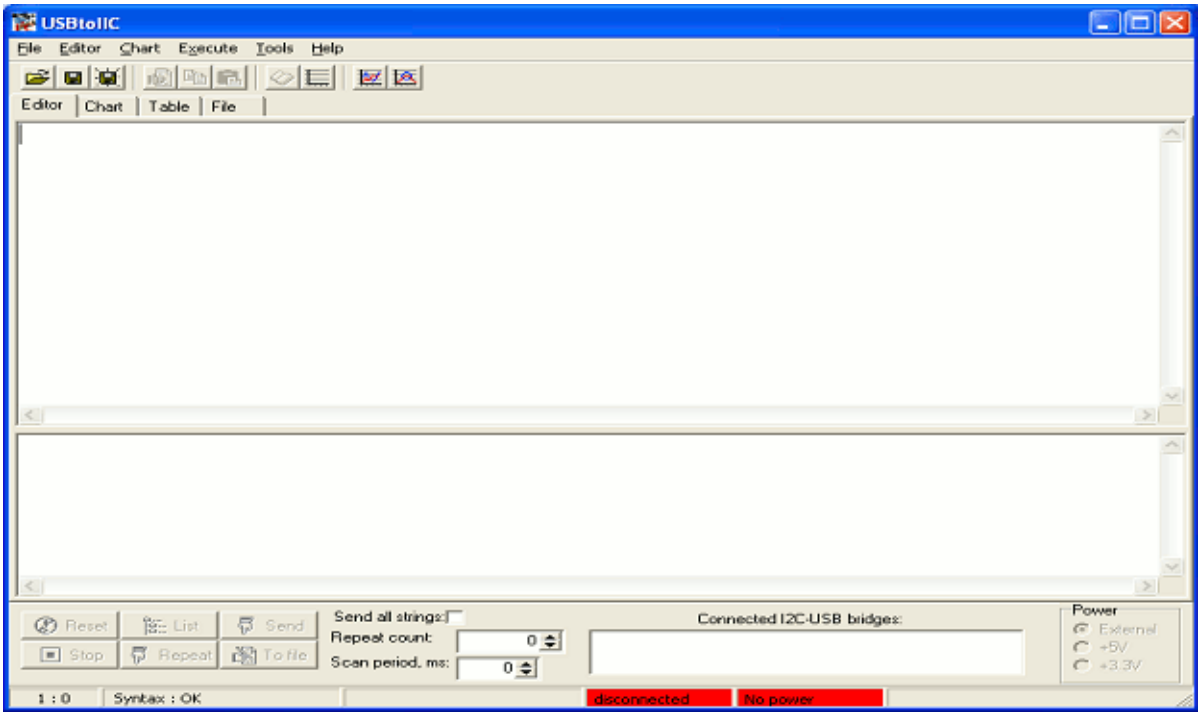
使用 USBtoIIC 桥接器 GUI 应用程序

USBtoIIC 桥接器和相关联的 GUI 是下载到引导加载程序的首选方法。

有关详细信息，请参见 “ 使用 CY3240 I^2C-USB 桥接器的 I^2C 引导加载程序 ” AN45683 应用笔记，其中论述了 <project_name>.txt 文件的格式以及使用该文件引导加载项目的步骤。 该应用笔记中提供了有关 .dld 格式到 .txt 格式转换工具的信息。 这并不是本数据手册所述器件的必要信息。 <project_name>.txt 文件会自动生成。

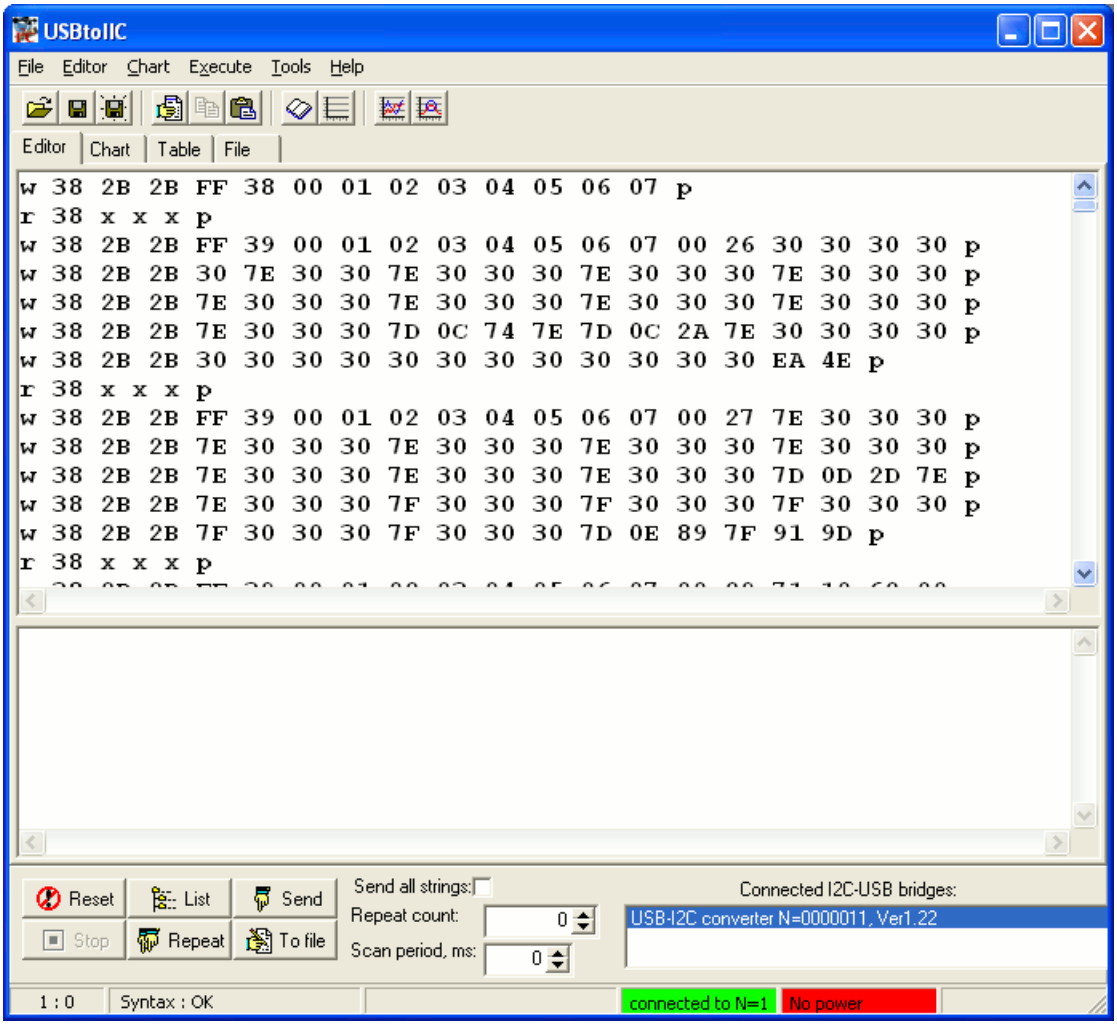
下面将简要说明 USBtoIIC 桥接器的使用。

启动 CY3240 USBtoIIC 桥接器的应用程序。 GUI 外观如下图所示：



将 <projectname>.txt 文件导入 USBtoIIC 桥接器 GUI

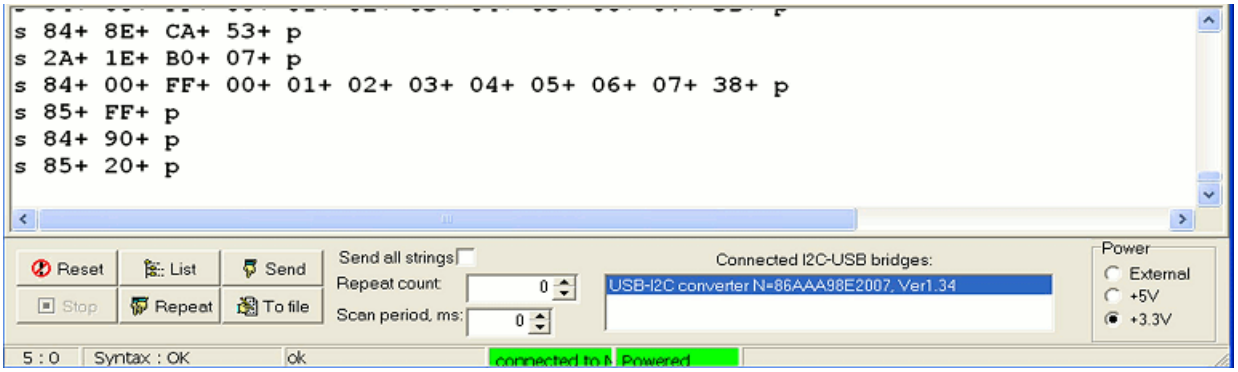
选择 **文件 > 打开**，然后浏览到要引导加载的项目的输出目录。 查找名为 <projectname>.txt 的文件。 在文件浏览器窗口中可能需要将文件类型选择为 “ 所有文件 ”。 如果该文件不存在，则可能需要使用本文档其他部分所述的引导加载程序工具重新生成该文件。 选择打开该文件后，文件加载可能需要数秒钟。 要确保文件已完全加载，请右键单击下方窗口，如果显示菜单，则 GUI 已准备就绪。



将 CY3240 USBtoIIC 桥接器连接至目标系统。 使用 GUI 将电源设置为所需水平。根据目标系统的电力需求，该电源可能会用于为目标供电。 GUI 底部的状态栏必须指示桥接器已连接且已通电。 此外，确保选中 USBtoIIC GUI 底部的 “Send all strings” 复选框，以便发送整个下载文件。 如果不选中 “Send all strings” 复选框，则一小部分下载文件将高亮显示，可以用于试验性的发送。



单击 “ 发送 ” 按钮下载新的代码，状态区域会显示各种 I2C 数据操作的状态。 请注意，“+” 表示数据操作成功。



引导加载程序 I²C 下载协议

“ 使用 CY3240 I

2

C-USB 桥接器的 *I²C 引导加载程序*” AN45683 应用笔记论述了 <project_name>.txt 文件的格式以及使用该文件引导加载项目的步骤。 该应用笔记中提供了有关 .dld 格式到 .txt 格式转换工具的信息。 这并不是本数据手册所述器件的必要信息。 <project_name>.txt 文件会自动生成。

下面将说明 <project_name>.dld 文件的格式： 请注意，闪存模块大小为 64-byte 和 128-byte 的器件的文件格式没有变化。 要编程的模块的大小在引导加载程序内部受到支持。

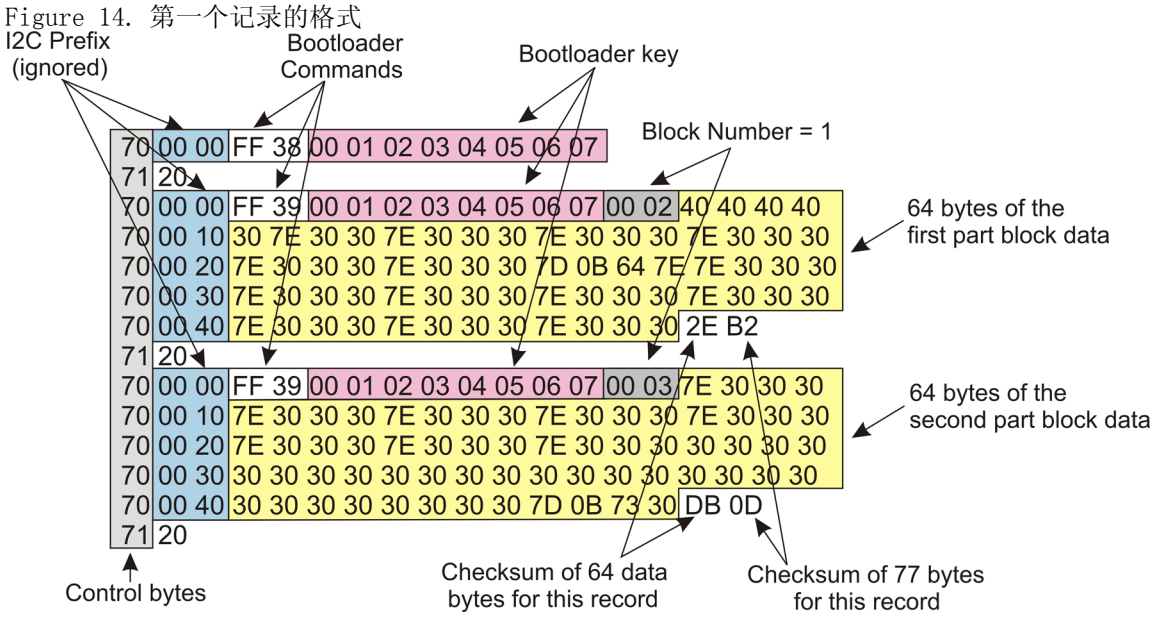
此处显示了两个示例下载记录。 这两个记录中包含了将在 I²C 主控与要引导加载的从器件之间传输的实际数据。 记录格式如下图所示：

Figure 13. 记录格式

First Download Record	
70 00 00 FF 38 00 01 02 03 04 05 06 07	Enter bootloader FF, 38
71 20	Master reads bootloader slave to acquire status
70 00 00 FF 39 00 01 02 03 04 05 06 07 00 02 40 40 40 40	No status request or response
70 00 10 30 7E 30 30 7E 30 30 30 7E 30 30 30 7E 30 30 30	
70 00 20 7E 30 30 30 7E 30 30 30 7D 0B 64 7E 7E 30 30 30	
70 00 30 7E 30 30 30 7E 30 30 30 7E 30 30 30 7E 30 30 30	
70 00 40 7E 30 30 30 7E 30 30 30 7E 30 30 30 2E B2	
71 20	Status request and response
.....	
Last Download Record	
70 00 00 FF 39 00 01 02 03 04 05 06 07 01 FF 30 30 30 30	No status request or response
70 00 10 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30	
70 00 20 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30	
70 00 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30	
70 00 40 30 30 30 30 30 30 30 30 30 30 30 30 00 54	
71 20	Status request and response
70 00 00 FF 3B 00 01 02 03 04 05 06 07	Exit bootloader FF, 3B
71 20*	Status request and response

下图显示的是第一个记录的格式：

Figure 14. 第一个记录的格式



每行都以控制字节开头。 下载协议使用的两个控制字节为：

- 70 - 从器件地址 38 写入。 该地址不被视为字节计数的一部分。
- 71 - 从器件地址 38 读取。 对从器件地址读取的预期响应为 0x20，表示成功。 其他的可能响应有：

Table 5. 从器件地址读取响应

代码	含义
0x20	引导加载模式 （成功）
0x02	映像验证错误
0x04	Flash 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

从器件地址写入命令不需要响应，因此每个从器件地址写命令行之后的两个字节是引导加载程序 I^2C 的 前缀，它们可以被引导加载程序忽略。 使用 Ignore_N_I2C_Prefix_Bytes 参数可设置应用程序中所用前缀字节的数量。

示例下载记录的第一行和第三行包含了引导加载程序命令。 使用的引导加载程序命令如下：

Table 6. 引导加载程序命令

命令	含义
FF38	进入引导加载程序
FF39	写入模块
FF3B	退出引导加载程序

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。 可以使用 Bootloader_Key 参数设置引导加载程序密钥。

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。 第三和第四 64-byte 模块包含将写入第二 128-byte 闪存模块的校验和信息。 本章节对校验和模块的格式进行了论述。 所有其他的写入区块命令都会将 64-byte 十六进制记录以五个数据包 （共 78 字节，忽略地址和丢弃的前缀）传输至引导加载程序。 对于模块大小为 128 字节的器件，累加两个这样的 64byte 传输。 模块组织经验证为一个偶数模块后接一个奇数模块。 累加完整个模块后，将写入闪存。

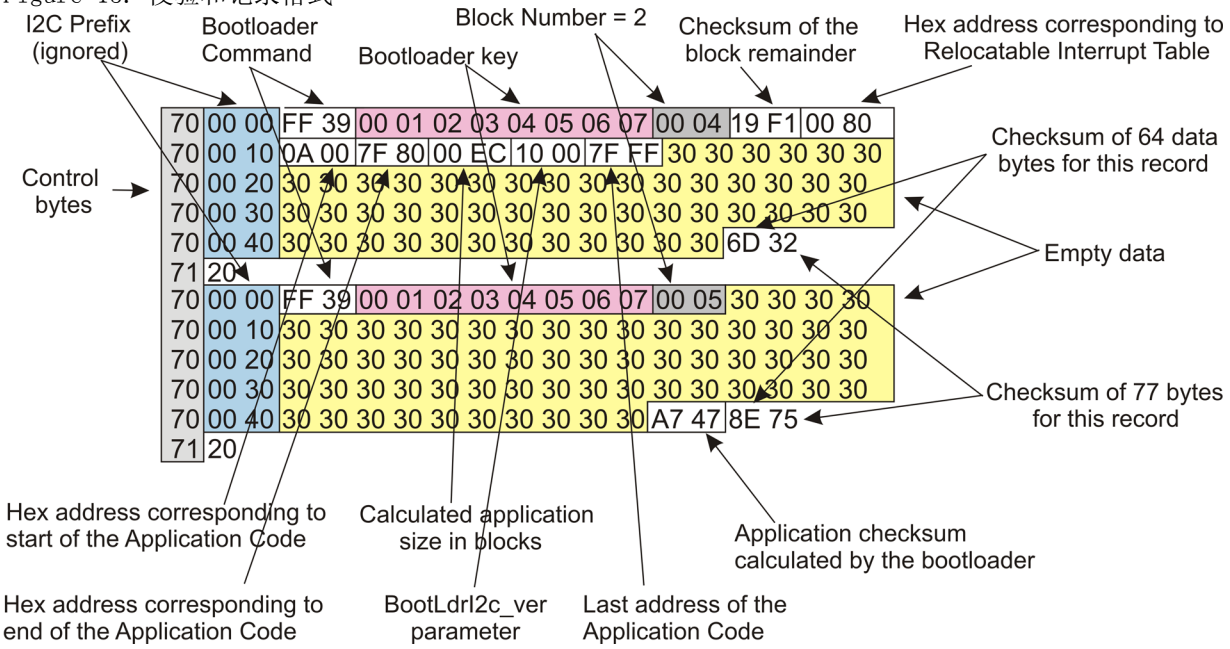
写入区块命令的第一行包含一个控制字节、一个被忽略的 2-byte I^2C 前缀、写入模块命令、引导加载程序密钥、传输的模块编号以及前四个字节的数据。 本示例中的模块编号为 0x0002，对应于 ROM 地址 0x0080。

后续三行仅包含控制字节、I^2C 前缀和 16 个字节的数据。 写入模块命令的最后一行包含控制字节、I^2C 前缀、最后 12 个字节的数据以及两个 1-byte 校验和。 第一个校验和 （本示例中为 0x2E）是该记录数据字节的校验和。 第二个校验和 （本示例中为 0xB2）是整个 77 字节记录的校验和，不包括地址字节和前缀。 地址字节和前缀在其被接收时由引导加载程序进行内部验证。

在写入区块命令结束时，系统将发送另一状态请求，该请求的结果为图示响应。

所有传输的模块具有相同的格式。 下图显示的是校验和记录的格式：

Figure 15. 校验和记录格式



第三和第四的记录包含校验和模块 （注意模块数始终为 0x0002）。 此章节提供了这些记录中数据的说明。

与其他记录相同，第一行包含了控制字节、I²C 前缀、引导加载程序写入模块命令、引导加载程序密钥以及模块编号。 紧接的两个字节包含了用于模块剩余部分的可选校验和，本示例中为 0x19F1。 该行的随后两个字节包含可重定位中断矢量表的十六进制地址 （更多信息，请参见图 2）。

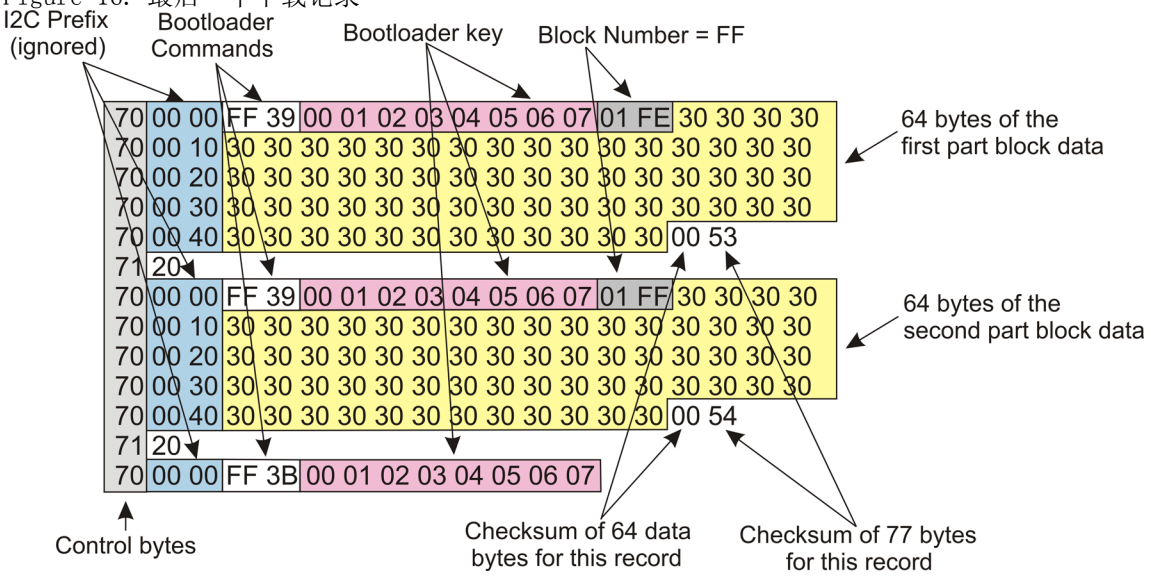
该记录的第二行包含了一个控制字节和一个 I²C 前缀，后接双字节值，代表根据区块 0x14 计算得出的 ApplicationCode_Start_Block 用户模块参数的十六进制地址。 下两个字节是从模块 0xFF 计算所得的应用代码末端的十六进制地址。 此后的两个字节是模块中应用程序的大小。 该行的最后两个字节为实际数据值，代表来自 BootLdrI2C_ver 参数的引导加载程序版本号。 此行的最后的两个字节的真实数据值是应用程序代码的最后地址。 该行的剩余部分为空的数据空间。

下面的两行包含的是额外的空数据空间。 校验和区块的最后一行包含的是空数据空间，但该行的最后四个字节除外。 这最后四个字节包含：引导加载程序计算的双字节应用程序校验和、来自 Intel 十六进制记录格式的单字节区块校验和以及整个 77 字节记录的校验和，地址字节和前缀除外。 地址字节和前缀在其被接收时由引导加载程序进行内部验证。

下一行包含了另一个状态请求和响应。

图 16 显示了最后一条纪录。

Figure 16. 最后一个下载记录



引导加载程序退出命令包含了控制字节 70、一个双字节前缀、引导加载程序退出命令 0xFF3B 以及引导加载程序密钥。

最后一行是最终状态请求和可能的响应。 在收到退出引导加载程序命令时，目标系统将立即执行内部复位并开始使用校验和模块中给出的参数验证所下载应用程序的校验和。 根据主机系统的速度，引导加载程序在主控能够接收到有效状态字节之前可能已经开始其复位进程。 因此，状态 (0x20) 可能不会始终存在。 而地址 0x71 可能会被完全否认。

BootLdrI2C 和 E2PROM 用户模块共存

当放置在一个引导加载程序项目的 E2PROM UM 中后，在客户保留模块区域中分配 E2PROM 模块。 这一区域位于引导加载程序代码区和应用程序代码区之间 （具体信息参见引导加载程序存储器组织）。 因此，E2PROM 模块并非应用程序代码区的一部分，且不得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	添加了版本历史
2.00	DHA	1. 重新组织了内存映射。 2. 已将浮动中断矢量表放置于地址 0x0080。 3. 已将校验和模块放置于地址 0x0100。 4. 已将引导加载程序启动放置于地址 0x0180。 5. 确认的应用程序模块大小低于 256。 6. 添加了引导加载程序 API 跳转表。 7. 更新了用户模块参数表。
2.10	DHA	1. 围绕 SSC 调用，用 .nocc 替换 .Literal 和 .EndLiteral 语句。 2. 删除了导出 `@INSTANCE_NAME`_EnterBootloader 语句。 3. 为 I2C 地址比较定制项添加了用户代码章节。
2.20	DHA	1. 固定了 Application_Checksum_Block 和 TWO_Block_Relocatable_Interrupt_Table. 的初始化。 2. 添加了 CYONSFN3050 LP 器件支持。 3. 添加了支持以在工作区浏览器中显示引导加载程序输出文件。 4. 在 “ Flashsecurity.txt 中的错误设置 ” 部分添加了说明。

Note PSoC Designer 5.1 在所有用户模块数据手册中都引入了 “ 版本历史 ”。 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 © 2010-2012 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.