I²C 引导加载程序数据手册 BootLdrI2C V 2.20

Copyright © 2007-2012 Cypress Semiconductor Corporation. All Rights Reserved.

资源	PSoC® 模块			API 存储器 （字节）		引脚 （每个外部 I/O）
	数字	模拟 CT	模拟 SC	闪存	RAM	
CY7C603xx, CY7C64215, CY8C21x12, CY8C21x45, CY8C22x45, CY8C23x33, CY8C24x9x, CY8C28x43, CY8C28x52, CY8C29/27/24/21x3x, CY8CPLC20, CY8CLED04/08/16, CY8CLED0xD, CY8CLED0xG, CY8CLED16P01, CY8CTST110, CY8CTMG110, CY8CTST120, CY8CTMG120						
从器件 （完整的 API 支持）	0	0	0	2560	6-128	2
从器件 （无 API 支持）	0	0	0	2144	6-128	2

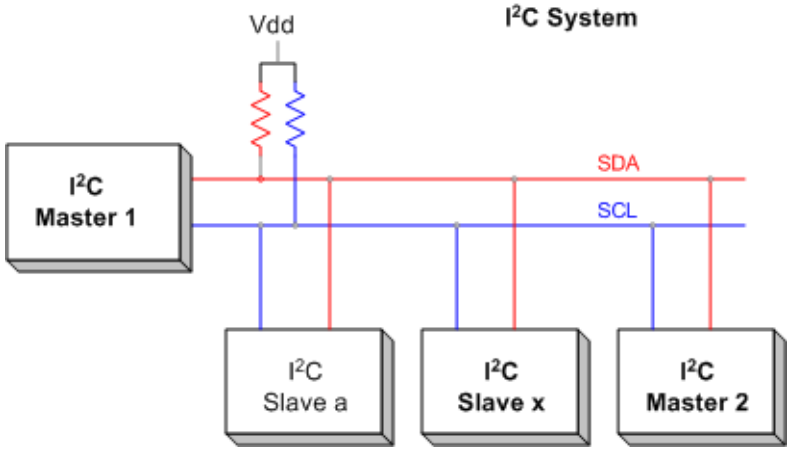
功能和概述

- 行业标准 Philips I²C 总线兼容接口
- 可使您使用 I²C 系统总线，而不是系统在线编程接口对 PSoC 器件重新编程

BootLdrI2C 用户模块实现了能够通过 I²C 接口对 PSoC 器件重新编程的引导加载程序。 PSoC 器件已经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行业标准通信接口 （如 I²C）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。引导加载信息可通过 CY3240 （USB 到 I²C 桥接器）或系统内主机处理器等 I²C 主器件进行发送。

I²C 引导加载程序需要使用 I²C 硬件用户模块。 它并不阻碍将 I²C 总线用于 PSoC 器件内的其他功能。I²C 引导加载程序为其相关联的功能使用单独的 I²C 地址。 I²C 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



快速启动

1. 查看本用户模块数据手册。 要成功实施引导加载程序项目，需要对此信息有所了解。
2. 将用户模块添加到项目中。
3. 放置用户模块，选择 “ 仅针对引导加载程序的 I^2C” 或 “ 具有引导加载功能的完整 I^2C API 支持 ”。
4. 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。
5. 右击用户模块图标，然后选择 引导加载程序工具 .
6. 单击 获取文件 . 文件 *boot.tpl*, *custom.lkp* 和 *flashsecurity.example* 文件会放在项目根目录中。
7. 关闭**引导加载程序工具** 向导。
8. 生成源代码并编译项目。
9. 查看输出文件 *<project>.mp* 和 *<project>.hex* 了解项目是如何构建的。
10. 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。
11. 详细教程位于 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**帮助 > 文档 > 支持文档**。

功能描述

引导加载程序在闪存中的位置，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 （必须被）写保护，以防止发生意外修改或损坏。 复位向量已进行了修改，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，并对照写入闪存中最后两个字节的校验和对其进行进行验证。 如果校验和匹配，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头，且用户代码可以执行。
2. 如果校验和不匹配，则引导加载程序将执行可自定义的用户代码，以执行系统的关键任务 （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代码映象。
4. 主控通过带若干编码字节的 64-byte 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。

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 项目的存储器组织之间存在显著的差异。 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 显示引导加载程序存储器组织。

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

I2C 和睡眠

当配合进入睡眠状态的项目使用 I2C 时，需特别小心。 在项目进入睡眠状态之前，遵循这些步骤，以确保正确的进入睡眠和 I2C 处理：

1. 确保所有 I2C 通信已完成
2. 通过调用 Stop API 禁用 I2C。
3. 将 I2C 引脚配置为模拟 High-Z 驱动模式。

在器件从睡眠中唤醒时遵循这些步骤：

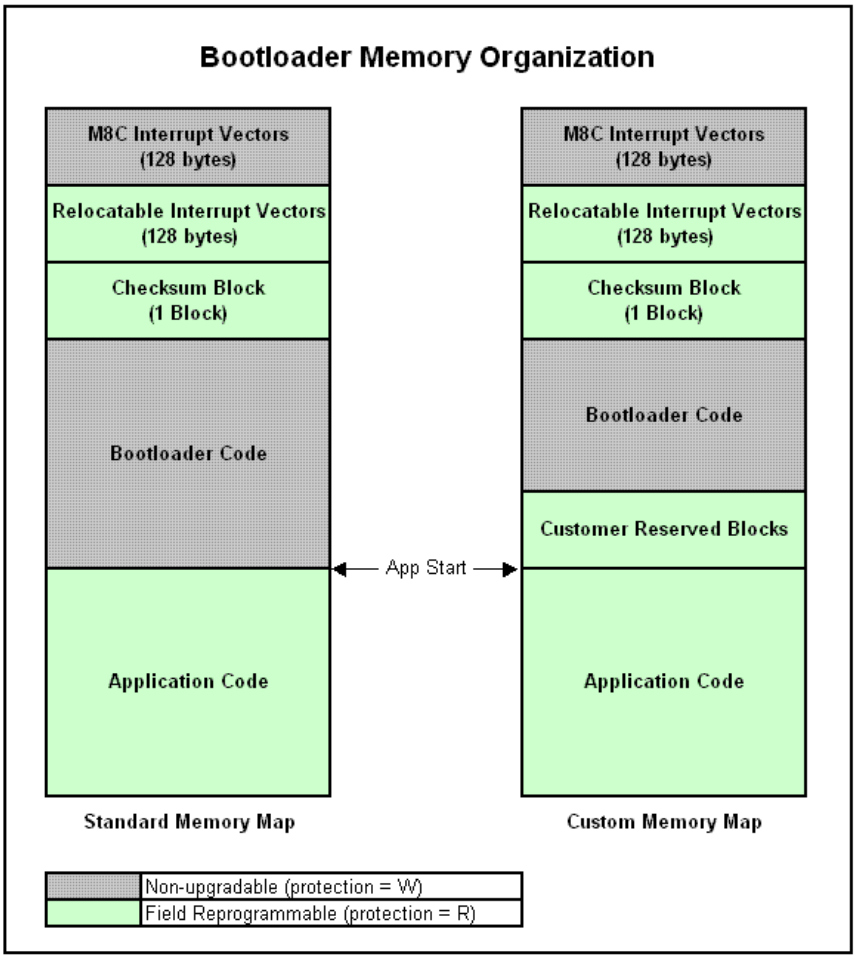
1. 确保无活动的 I2C 通信。
2. 通过调用 Start API 启用 I2C。
3. 将 I2C 引脚配置为 “ 开漏驱动低电平 ” 驱动模式。
4. 使能中断。

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 ” 视图，然后右键单击 “BootLdrI2C 用户模块 ” 图标。 除了提供工具和文件作为用户模块的一部分，还提供主机应用程序示例作为用户模块安装的一部分，可用来展示引导加载程序的基本功能。 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\3.00\Bootloaders\BootLdrI2C\BootLoaderHostApp\…

Figure 2. 引导加载程序存储器组织



概述

PSoC Designer 使用标准化文件，有关系列器件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。 无需引导加载程序的项目可以简单地允许编译器和连接器分配 RAM 和 ROM，但引导加载程序必须将 RAM 和 ROM 集中到特定的区域，以便在加载新应用程序时程序不会崩溃。

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 的模块 0 和 1。 这些模块包含关键的中断向量和重启向量。 由于正在运行的器件几乎不可能通过控制读操来访问这些块，因此这些块不会被擦除或重新编程。 不能修改 ROM 的最前面两个块，且不能将它们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。 根据器件架构的不同，此表可能由一个或两个块组成。 这一区域包含中断向量以及通用向量，为中断或当使用引导加载程序加载新应用程序时可能更改的代码入口提供跳转表。 例如，这一区域包含应用程序起始地址。 引导加载程序能够在加电时验证了校验和之后，使用此地址来启动新应用程序。 这一区域位于 Flash 第 2 和第 3 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重写，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 Flash 第 4 块 上，包含引导加载程序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Flash第5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。 如存储器映射所示，这一区域是可选的。 如果没有自定义数据，您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 “Appliaction_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。 在引导加载过程中，需要将代码使用的所有 RAM 添加到为引导加载程序定义的 RAM 区域。

用户模块参数中存储器区域的定义

通过 BootLdrI2C 用户模块参数，您可以自定义主程序元素在 ROM 中的放置位置。 用户模块中的默认参数应提供有效的初始设置。 用户应该使用这些设置，直到完整的项目成功编译完成。 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实用程序，与前台 （主要）应用程序共存。 当器件启动或复位时，会始终调用引导加载实用程序。 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，则引导加载实用程序允许前台应用程序执行。 如果两个校验和不相等，则引导加载程序将进入等待循环，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。 用户模块工具自动支持将应用程序下载到目标位置的两种下载文件格式。 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，请参阅应用笔记 AN2352 “PSoC®1 通讯 - I2C-USB 桥接器使用方法 ”。 此方法同时在本用户模块数据手册的附录中有所论述。

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\3.00\Bootloaders\BootLdrI2C\BootLoaderHostApp\…

演示 I^2C 引导加载程序会用到两个应用程序。 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。 源代码可以很容易地适应其他 PSoC 器件架构。

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。 它能够读取和解析下载文件 *<fiilename>.dld* 并将其传输到此章节之前提到的 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。 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。

Note 在某些情况下，您可能需要在您的电脑上 *mscomm32.ocx* 安装该文件。 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* 可以通过搜索以下文件名从 Microsoft 网站上下载 *mscomm32.ocx".*

引导加载程序工具

快捷菜单中提供了一些工具，可以通过右键单击用户模块图标访问这些工具。 从下拉菜单中选择 “ 引导加载程序工具 ” 。

通过 “ 获取文件 ” 选择，可添加 boot.tpl、custom.lkp 和 HTLinkOpts.lkp 的特殊版本，这些文件可以在工程中被放置或删除。 从主菜单中选择 “ 工具恢复默认引导文件 ” 将其删除。 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 - 一旦项目已经正确建立，您可以使用引导加载程序工具来创建和自动验证校验和。 在进入引导加载程序工具选择屏幕时，将生成项目代码，并执行对整个项目的完整编译。 然后在所生成的十六进制文件上执行校验和计算，得出的校验和与用户模块所存储的校验和进行比较。 如果校验和不匹配，则会显示一条消息。 如果用户愿意，可以重新计算并存储一个新的校验和。 如果在由引导加载程序工具所调用的自动化生成和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，有必要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。 该文件只包含了由引导加载程序重新编程的十六进制模块，包括校验和模块。 “ 主机演示 ” 应用程序能够读取此文件，并将其下载至包含引导加载程序的正在运行中的项目之内。 此下载文件可以部署到现场应用程序中以升级 PSoC 器件。

dld 和 txt 下载文件由 BootLdrI2C 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

一旦项目能够构建起来并且在编译时没有发生错误时，就必须生成应用程序的校验和。 在器件编辑器视图中右键单击 “BootLdrI2C 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，必定无法生成校验和。 一旦校验和已经创建，则必须将校验和集成到编译文件中。 这样就要求进行第二次编译。

提供的特殊文件

通过打开**引导加载程序工具**菜单并选择**获取文件**，可以访问一些重要文件。 一个与器件具体对应的 boot.tpl 文件放置在主项目目录下，并且还有两个分别称为 custom.lkp (ImageCraft) 和 HTLinkOpts.lkp (Hi Tech) 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。 - 这个文件中包含了中断向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 3 行之内或之后所做的改动都会导致发生错误或完全被丢弃。 您可以更改 custom.lkp 文件的其余部分。 要调试项目的存储器分配，可以通过在第一个空格处插入分号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 大小信息的数据。
- -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。 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，则将该 flashsecurity 文件覆盖似乎并不方便。 随后可能需要开发人员进行反复修改。 在必要时，所提供的文件 flashsecurity.example 可以进行编辑和重命名。

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。 如果存储器区块采用了写访问保护，则引导加载程序无法正常工作。 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。

I^2C 中断处理

标准 BootLdrI2C 用户模块可以选择提供一份 I^2C 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，能够用来操作功能完整的 I^2C 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 I^2C 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 （这不是良好的编程习惯）。

参数模块入口

所有存储器参数均在各模块的引导加载程序中输入，16K 器件的模块编号从 0x00 到 0xFF，8K 器件从 0x00 到 0x7F，32K 器件从 0x00 到 0x1FF。 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。 大部分的 PSoC 器件只能存储 64 字节的闪存模块 （部分器件系列是 128 字节），这是正确维持项目代码不同节段之间的边界的简单方法。

大多数 PSoC 部件的模块大小为 64 字节。 部分新 PSoC 器件的区块为 128 字节。 两个重要事实：

1. 所有引导加载程序都必须写入闪存。
2. PSoC 只能按 “ 块 ” 为单位写入闪存。

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 “ 块 ” 会更有用。

要将区块转化为绝对地址，需进行乘法运算： 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。 对于区块为 128 字节的器件，十六进制文件的两行为一个区块 （由于区块 0 起始于地址 0，因此必须一直将 128 字节区块视为：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）。

请参见十六进制文件，熟悉您使用的器件的闪存块大小。

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。 这些调整包含在文件 BootLdrI2C_Bootloader.inc 中。 本文件有一个部分包含以下宏定义：

```
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 宏定义允许用户加长、缩短超时时间，超过该时间如果没有从主机收到任何通信，将跳出循环，也可以将超时设为无穷大。 这点可能在开发或调试主机应用程序时有用。

第二个宏定义控制着校验和模块内校验和的使用。 如果将此宏定义设置为 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。

放置

I2CHW 用户模块的 SCL 和 SDA 有两种选择，即 P1[5]/P1[7] 或 P1[0]/P1[1]，并且不需要任何数字或模拟 PSoC 模块。 不存在放置限制。 放置多个 I^2C 模块是不可能的，因为 I^2C 模块使用专用 PSoC 资源模块和中断。

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。 对项目进行编译和测试后，用户可以选择调整块大小，以优化存储器使用情况。

Figure 3. 默认参数

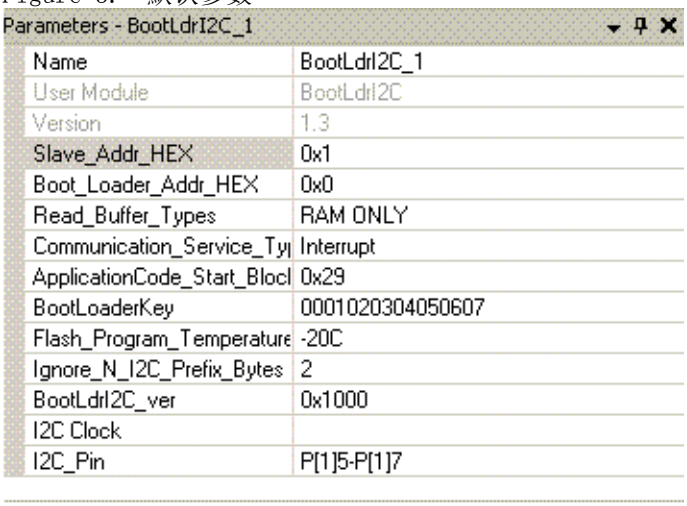


图 3 显示默认用户模块参数。 这些参数在用户模块的源代码中可能定期进行更新，更新后的参数可能会与示例中的参数有所不同。

所有缓存名称的定义均体现了其对 I^2C 主控的用途。 例如，I2Cs_pRead_Buf 指包含了要由 I^2C 主控读取的数据的 RAM 位置。

Slave_Addr_HEX

这是一个从器件和多主从器件 (Slave and MultiMasterSlave) 参数。 此参数选择一个 7-bit 的从器件地址，I^2C 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。由于它是地址的高 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双倍。 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 ”(RAM OR FLASH) 项，则可以使用 API 调用来选择是使用 RAM 读取缓冲区还是使用闪存读取缓冲区。

Communication_Service_Type

此项参数使用户能够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 I^2C 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 I^2C 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 I^2C 的从器件和多主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。

不建议在中断中使用轮询函数。 调用轮询函数的定时中断定义可能会频繁调用该函数，从而不会执行任何其他数据处理。 轮询处理不可重入，且在其完成处理之前，该函数不会被调用。

I2C 时钟

具体指定了运行 I^2C 接口所需的时钟频率。 可供选择的 I2C_Clock 速度有 3 种：

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

I2C_Pin

从端口 1 选择用于 I^2C 信号的引脚。 PSoC Designer 将自动为这些引脚选择适当的驱动模式。 请注意，在引导加载过程中，除 SCL、SDA 和可选的中断引脚以外，端口 1 上的所有引脚均设置为 HI-Z 模拟模式。

CPU_Clk_speed_(CY8C27xA)

Note 此参数仅可用于 CY8C27xxx 系列中的器件。

指定 CY8C27xxx A 版本芯片器件运行的 CPU 时钟频率范围。 CY8C27x A 是指较早的 CY27xxx 器件，现已不建议用于新设计上。 在 PSoC Designer 的器件目录中，CY8C27x A 型芯片列在目录末尾，并在其前加注了 “ 不建议用于新设计 ” 的说明。 新型器件的名称中带有字母 “X”，从而可与旧部件区别开来。 字母 X 用于标明无铅器件。 需要使用此参数，来确保以较高的 CPU 时钟频率插入专用代码处理，并在 CPU 时钟频率较低的情况下消除专用代码处理以避免不必要的开销。

CPU_Clk_speed_(CY8C27xA) 数值	使用说明
低于等于 6 MHz	在 CPU 时钟频率低于等于 6 MHz 时使用。 这消除所有不必要的代码。
高于 6 MHz （仅限 CY8C27x A）	在 CPU 时钟频率高于 6 MHz，且所使用的芯片属于 CY8C27xxx A 版本芯片系列时使用。 这可确保用户模块的正常功能，代价是增加一些额外开销。
非 CY8C27xA	用于除 CY8C27xxx A 版本芯片器件系列以外的所有芯片。 这适用于器件目录里部件编号中带有字母 “X” 的所有 CY8C29/24/22xxx 器件或 CY8C27xxx 器件。 选择这个值会消除不必要的代码。

通过为引导加载程序定义的参数，可以定义当编译和连接程序时主程序模块的位置。 在有些情况下，工程可能是在较早版本芯片的 27xxx POD 上开发的 （因此需要这些时钟频率的具体设置），而目标器件是较新版本的芯片。 目前没有发现在较新器件中包含适用于较早芯片的时钟频率参数会对项目造成重大影响。

ApplicationCode_Start_Block

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

由参数模块指定的默认地址可以通过将器件块大小 （0x40 或 0x80）乘以参数中的模块计算得出。

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。 这个步骤可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，能够影响到内存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，用户必须采取预防措施以确保引导加载程序绝对不会在高出本参数设置值 20° C 的温度下运行。 有关详细信息，请参见赛普拉斯器件规格。

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。本参数允许用户配置引导加载程序以忽略可变数量的前置字节。

BootLdrI2C_ver

此参数是引导加载程序的版本。 它当前没有用于内部固件，可作为校验和模块的一部分使用。 此参数可以由用户设置并用于验证引导加载程序可执行代码的正确版本。

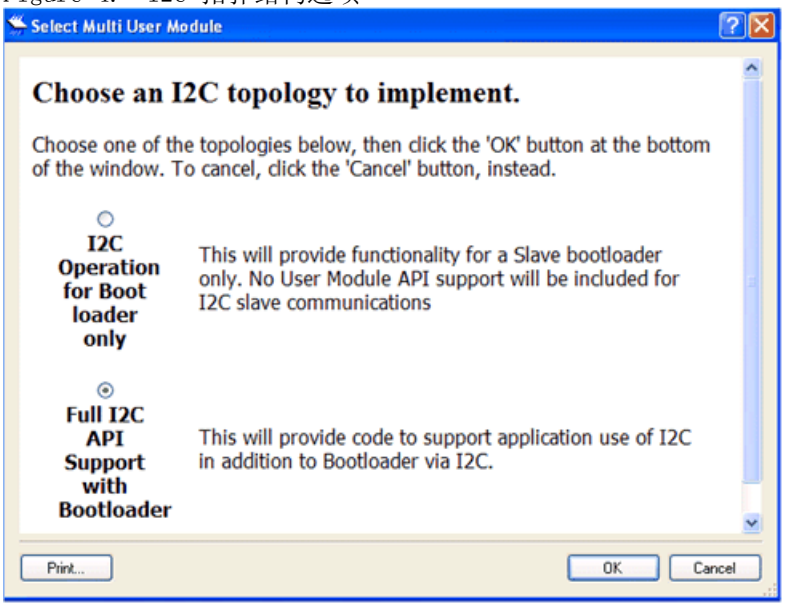
I^2C 拓扑结构选项

放置 BootLdrI2C 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。

I^2C 操作仅用于引导加载程序： 本选项仅提供用于引导加载程序的 I^2C 通信。 对于 I^2C 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 I^2C 通信仅用于引导加载，请选择此选项。

引导加载程序的完整 I^2C API 支持： 此选项提供代码以支持 I^2C 以及通过 I^2C 的引导加载程序的应用。如果计划让 I^2C 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。

Figure 4. I2C 拓扑结构选项



常见问题

更新引导加载程序项目、服务包升级和编译器

使用引导加载应用程序时，应避免更改 PSoC 开发环境。 其中包括不更新 PSoC Designer 和 BootLdrI2C 用户模块，且不更改编译器。

其背后的原因是，引导加载程序和应用的编译最初是一同进行的。但是，在部署可引导加载的系统后，仅对应用程序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 BootLdrI2C 用户模块一同进行编译，以使新应用程序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。 而对于引导加载程序，保持兼容性是必须做到的。 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，由一个编译器编译的引导加载程序可能无法与另一个编译器编译的应用程序兼容。 另外，不同的编译器实现 RAM 分页的方式可能不一样。 另外，由于引导加载程序和应用程序是一同编译的，因此假如一对引导加载程序和应用程序所使用的开发工具不相符，则无法对它们进行调试。

HI-TECH 编译器的 RAM 分配问题

一些用户模块使用大型阵列中的内存或第 0 页上的大量 RAM。

- 用户模块内存的默认位置是第 0 页。
- 引导加载程序需要第 0 页上处于最低位置的内存。
- HI-TECH 本地变量和 InterruptRAM 的默认位置是第 0 页。

由于这些都在争夺 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会关联至以下全局参数：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。 实际可行的看门狗定时器最小设置值约为 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 “ 引导加载程序工具 ”-“ 获取文件 ” 用户模块菜单项来提供。 该映射必须允许在将要真正执行引导加载的所有位置进行闪存写操作。 其中一种策略是让所有模块均可以写入。 另一种策略是在当前花一些时间布局存储器映射，并相应地对该文件进行编辑。 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。 用户必须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。

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）

引导加载程序项目的存储器映射与传统项目有显著区别。 因此通常需要更改可重定位代码起始地址。 这是连接器在试图将多个模块代码写入到同一地址时所发生的错误现象的常见原因。 可以在 “ 项目 ”>“ 设置 ”>“ 连接器 ” 选项卡中的 “ 可重定位代码起始地址 ” 字段中更改此参数。 计算绝对十六进制起始地址时要让此地址比引导加载程序代码所使用的最高模块的地址稍大一点儿，或者让其地址占用一个未使用的 ROM 区域。 对于 I^2C 版本的引导加载程序设置，将此数值设置为 0xA40 应该足够了（如果其他参数使用了默认值）。

Note 在未放置 BootLdrI2C 用户模块时，可重定位代码地址不能重设为其原始值。 您必须手动改回，以节省 ROM 空间。

存储器重叠

如需纠正可重定位代码起始地址 （参见上一节），可使用前加的分号将 custom.lkp 文件的最后三行以加注释的方式除去，并尝试再次构建这个文件，然后检查所生成的存储器映射。 存储器重叠问题较难诊断出来，因为这种问题会导致输出文件无法生成。 通过修改 custom.lkp 文件可以让连接器放置目标模块，这样就可以开始找到存储器重叠的根本原因。

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发生变化的情况。 其中一个例子是在电源加电时对闪存的状态写入。 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造成闪存数据保持不良。

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。 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，并在超时过期时退至前台应用程序。
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。 这个条件可以是开关闭合或使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。 在看门狗定时器复位条件满足并进行复位时，可以监测一个与看门狗定时器相关联的状态位，以判断看门狗定时器溢出是出现复位的原因 （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和可重定位代码起始地址必须完全一致 （这不同于第一个应用程序区块）。 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-def 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，用户可以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。 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。 相对于供电斜坡来说，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，该器件的供电电平可能发生变化。 其中一个例子是在电源加电时对闪存的状态写入。 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。 写入缓冲区只能在 RAM 存储器中设置。

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。 选择此 “ 寄存器易失 ” 策略是为了提高效率，自 PSoC Designer 1.0 版起已强制使用此策略。 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。 此例程被调用后将不会返回，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。 这 3 个例程代表同一个地址向量。 提供了向量 “GenericBootloaderEntry”，从而有一个当该用户模块的实例名称更改时也不会发生变化的进入点。

C 原型：

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

其他 API 函数名称可以用上述名称调用。

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。 在大内存模式下 (CY8C29xxx)，所有 RAM 页面指针寄存器都会出现这种状况。 如果需要，调用函数负责通过调用 fastcall16 函数保留值。

BL_SetTemp

说明：

本函数用于动态更新引导加载程序的最后一次 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引导加载程序。 然后，在发生引导加载程序事件时，引导加载程序根据通过该函数接收到的温度对闪存进行最佳的编排。

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引导加载程序。 引导加载程序在引导加载期间使用接收到的 die 温度更改闪存编程、擦去和写入周期。 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引导加载程序的温度值而定。

die 温度可通过使用测量器件芯片温度传感器的用户模块， 或通过读取或测量其它一些外部器件或温度传感器的温度测量。

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 （摄氏度）。

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。 在大内存模式下 (CY8C29xxx)，所有 RAM 页面指针寄存器都会出现这种状况。 如果需要，调用函数负责通过调用 fastcall16 函数保留值。

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。 在大内存模式下 (CY8C29xxx)，所有 RAM 页面指针寄存器都会出现这种状况。 如果需要，调用函数负责通过调用 fastcall16 函数保留值。

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。 在大内存模式下 (CY8C29xxx)，所有 RAM 页面指针寄存器都会出现这种状况。 如果需要，调用函数负责通过调用 fastcall16 函数保留值。

BootLdrI2C_EnableInt

说明：

启用 I^2C 中断以便实现启动条件检测。 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。 在大内存模式下 (CY8C29xxx)，所有 RAM 页面指针寄存器都会出现这种状况。 如果需要，调用函数负责通过调用 fastcall16 函数保留值。

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 start/restart 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 NAK，则该数据操作发送的后续字节会被忽略，直到检测到另一个 start/restart 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 之前保持停顿，只是通过将 SCL [时钟] 线保持在低位来实现的 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能够通过外部应用程序 I^2C 命令进入，则必须使用 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，并将其传递到前台 I^2C 中断进程中，该进程也会对此地址进行测试 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例程中。

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。 在大内存模式下 (CY8C29xxx)，所有 RAM 页面指针寄存器都会出现这种状况。 如果需要，调用函数负责通过调用 fastcall16 函数保留值。

BootLdrI2C_EnableSlave

说明：

通过设置 I2C_CFG 寄存器中的 “ 使能从器件 ” 位，为 I^2C HW 模块启用 “I^2C 从器件 ” 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。 在大内存模式下 (CY8C29xxx)，所有 RAM 页面指针寄存器都会出现这种状况。 如果需要，调用函数负责通过调用 fastcall16 函数保留值。

BootLdrI2C_DisableSlave

说明：

通过清除 I2C_CFG 寄存器中的 “Enable Slave” （使能从器件）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。 在大内存模式下 (CY8C29xxx)，所有 RAM 页面指针寄存器都会出现这种状况。 如果需要，调用函数负责通过调用 fastcall16 函数保留值。

仅为 FullAPISupport 提供 API：

BootLdrI2C_InitWrite

说明：

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。 在大内存模式下 (CY8C29xxx)，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 函数保留值。

BootLdrI2C_InitRamRead

说明：

BootLdrI2C_InitRamRead 例程可初始化一个数据缓冲区指针，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_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。 在大内存模式下 (CY8C29xxx)，所有 RAM 页面指针寄存器都会出现这种状况。 如果需要，调用函数负责通过调用 fastcall16 函数保留值。

BootLdrI2C_InitFlashRead

说明：

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。 在大内存模式下 (CY8C29xxx)，所有 RAM 页面指针寄存器都会出现这种状况。 如果需要，调用函数负责通过调用 fastcall16 函数保留值。

将会清除读取状态位。

BootLdrI2C_bReadI2CStatus

说明：

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量。
I2CHW_RD_ COMPLETE	04h	一项读取操作已启动且已完成。
I2CHW_READFLASH	08h	下一个读取操作将来自某个闪存位置。
I2CHW_WR_NOERR	10h	主控成功写入了数据。
I2CHW_WR_OVERFLOW	20h	主控在写入缓冲区写入了过多字节。
I2CHW_WR_COMPLETE	40h	主控写入操作完成 （由于新的寻址或停止状况）。
I2CHW_ISR_ACTIVE	80h	I^2C ISR 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。 在大内存模式下 (CY8C29xxx)，所有 RAM 页面指针寄存器都会出现这种状况。 如果需要，调用函数负责通过调用 fastcall16 函数保留值。

BootLdrI2C_ClrRdStatus

说明：

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。 在大内存模式下 (CY8C29xxx)，所有 RAM 页面指针寄存器都会出现这种状况。 如果需要，调用函数负责通过调用 fastcall16 函数保留值。

BootLdrI2C_ClrWrStatus

说明：

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

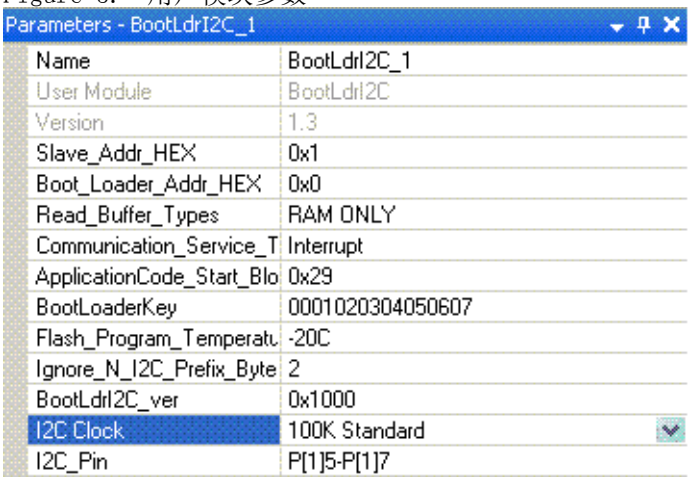
副作用：

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。 在大内存模式下 (CY8C29xxx)，所有 RAM 页面指针寄存器都会出现这种状况。 如果需要，调用函数负责通过调用 fastcall16 函数保留值。

固件源代码示例

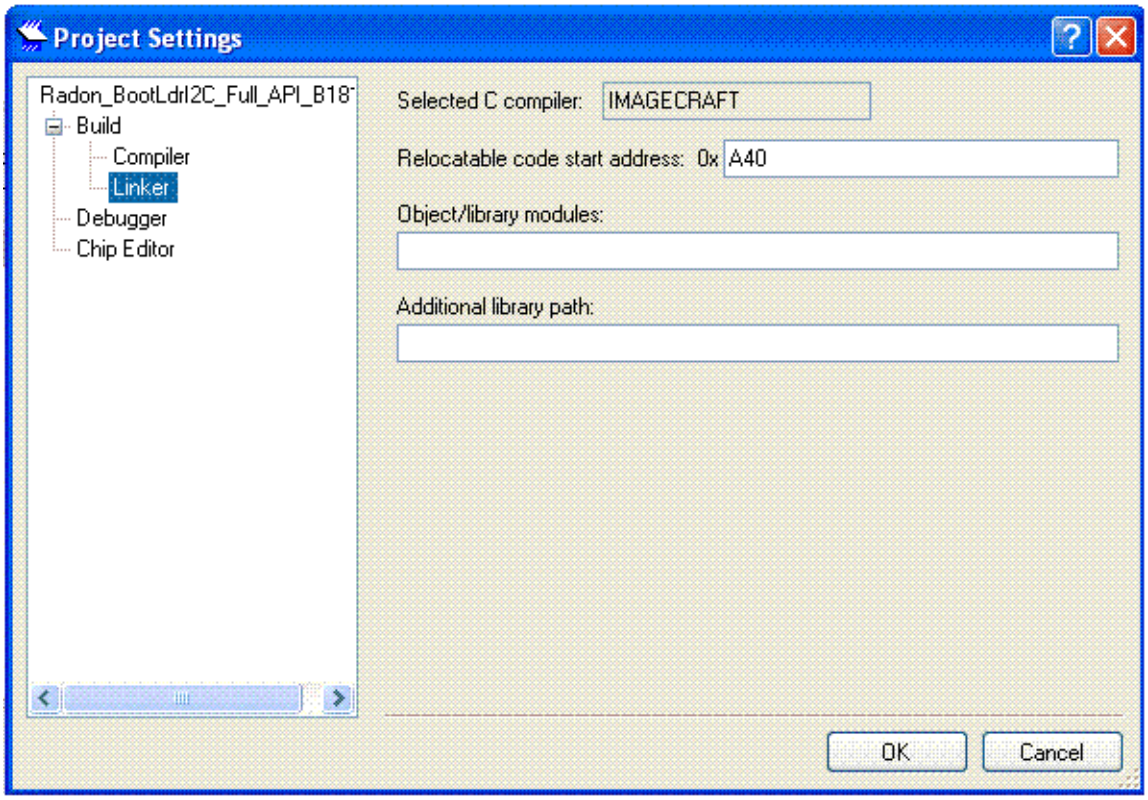
按图 5 所示，针对汇编语言和 C 语言示例配置用户模块参数：

Figure 5. 用户模块参数



确保代码起始地址已正确设置。

Figure 6. 设置代码起始地址



以下是使用 C 语言编写的 BootLdrI2C 用户模块的实现：

```
//-----------------------------------------------------------------------
// 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 BootLdrI2C 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

本节介绍被 BootLdrI2C 用户模块使用或修改的 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 Slave。

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，生成 I^2C 总线启动，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。

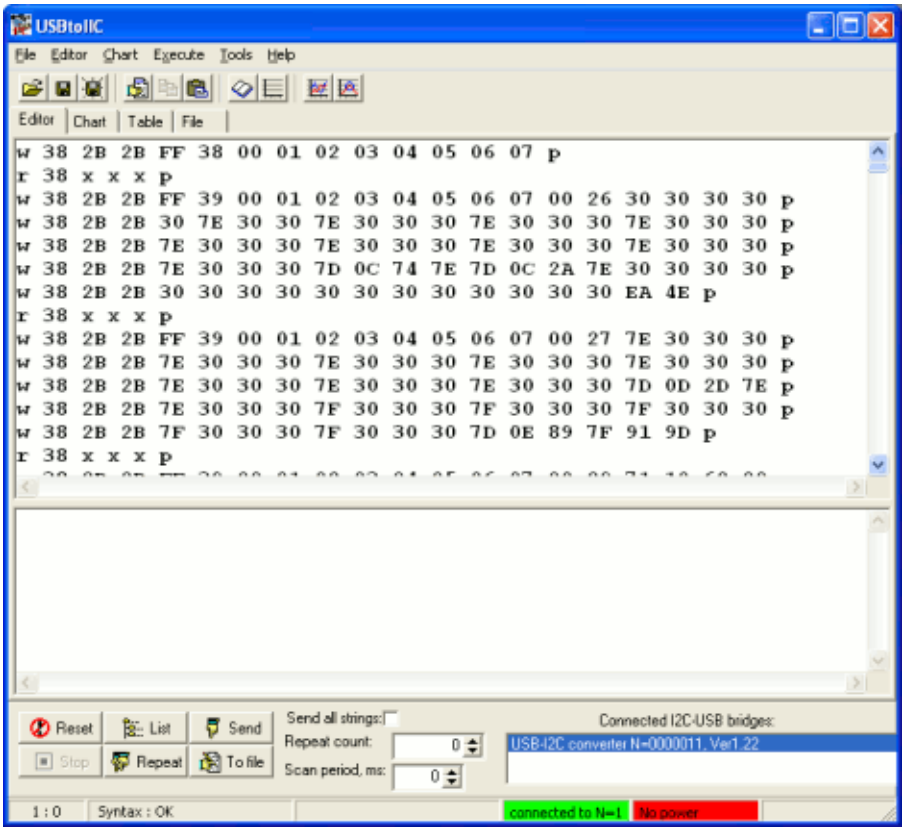
使用 USBtoIIC 桥接器 GUI 应用程序

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。

更多信息位于应用笔记 “ 使用 CY3240 I^2C-USB 桥接器的 I^2C 引导加载程序 ” AN45683。 本应用笔记探讨 <project_name>.txt 文件格式和要引导加载项目的程序。 此外，还提供了有关 .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。

下面对 USBtoIIC 桥接器进行简要的讨论：

1. 启动 CY3240 USBtoIIC 桥接器的应用程序。
2. 将 <projectname>.txt 文件导入 USBtoIIC 桥接器 GUI
3. 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 <project-name>.txt 的文件。 可能需要在文件浏览器窗口中将文件类型选择为 “ 所有文件 ”。 如果该文件不存在，则可能必须使用本数据手册中所述的引导加载程序工具重新生成该文件。 选择该文件后，文件加载可能需要数秒钟。 要检查文件是否已完成加载，请右键单击下方窗口；如果显示菜单，则 GUI 已准备就绪。



4. 将 CY3240 USBtoIIC 桥接器连接至目标系统。 使用 GUI 将电源设置为所需电平。 根据目标系统的电平需求，该电源可以用于为目标供电。 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 “ 发送所有字符串 ” 复选框，以便发送整个下载文件。 如果取消选中 “ 发送所有字符串 ” 复选框，则一小部分下载文件将高亮显示，并进行测试性发送。



5. 单击 “ 发送 ” 按钮下载新代码。 状态区域显示不同 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 （.dld 文件）格式

本节简要讨论了文件 <project_name>.dld 的格式：

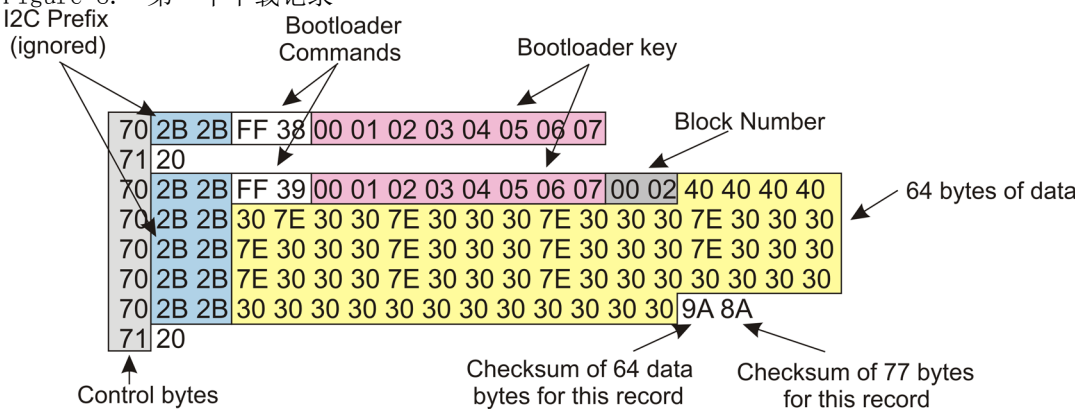
下图中显示了两个示例下载记录 – 第一个、第三个和最后一个。 这两个记录中包含了将在 I²C 主控与要引导加载的从器件之间传输的实际数据。 本节描述了这些记录的格式。

Figure 7. 示例记录

```
                First Download Record
70 2B 2B FF 38 00 01 02 03 04 05 06 07      Enter bootloader FF, 38
71 20                                       Master reads bootloader slave to acquire status
70 2B 2B FF 39 00 01 02 03 04 05 06 07 00 02 40 40 40 40   No status request  or response
70 2B 2B 30 7E 30 30 7E 30 30 30 7E 30 30 30 7E 30 30 30
70 2B 2B 7E 30 30 30 7E 30 30 30 7E 30 30 30 7E 30 30 30
70 2B 2B 7E 30 30 30 7E 30 30 30 7E 30 30 30 30 30 30 30
70 2B 2B 30 30 30 30 30 30 30 30 30 30 30 9A 8A
71 20                                       Status request and response
                ..........
                Last Download Record
70 2B 2B FF 39 00 01 02 03 04 05 06 07 00 FF 30 30 30 30   No status request  or response
70 2B 2B 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30
70 2B 2B 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30
70 2B 2B 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30 30
70 2B 2B 30 30 30 30 30 30 30 30 30 30 30 30 00 53
71 20                                       Status request and response
70 2B 2B FF 3B 00 01 02 03 04 05 06 07      Exit bootloader FF, 3B
71 20*                                      Status request and response
```

图 8 详细描述了第一个记录的格式：

Figure 8. 第一个下载记录



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。 表 5 中列出了其他可能响应：

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	闪存校验和错误

代码	含义
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入行的下面两个字节是引导加载程序忽略的 I^2C 前缀。 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。 使用了表 6 中列出的引导加载程序命令：

Table 6. 引导加载程序命令

命令	含义
FF38	进入引导加载程序
FF39	块写入
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。 您可以使用 Bootloader_Key 参数设置引导加载程序密钥。

引导加载程序块写入命令

发送到引导加载程序的命令大多数都是块写入命令。 每个块写入命令的格式都一样。 第三个块包含校验和信息。 本章节对校验和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。

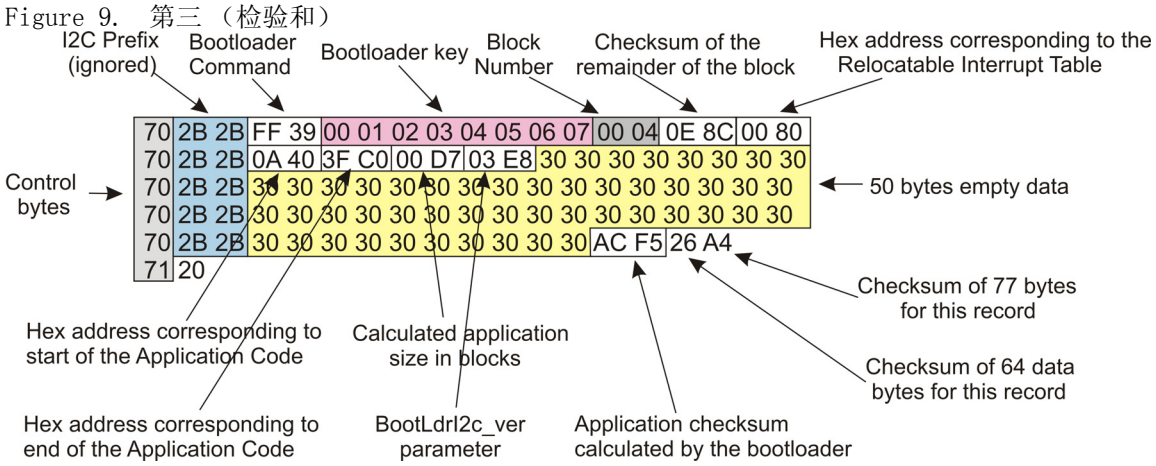
写入区块命令的第一行包含一个控制字节、一个被忽略的 2-byte I^2C 前缀、写入块命令、引导加载程序密钥、传输的块编号以及前四个字节的数据。 本示例中的块编号为 0x0002，对应于 ROM 地址 0x0080。

后续三行仅包含控制字节、I^2C 前缀和 16 个字节的数据。 块写入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 0x9 A）是该记录数据字节的校验和。 第二个校验和 （本示例中为 0x8）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。 第三个块包含校验和信息。 图 9 显示本记录的格式：

Figure 9. 第三 （检验和）



第三个记录包含校验和块 （注意块编号始终为 0x0004）。 本节中说明了该记录中的数据。

与其他记录相同，第一行包含了控制字节、I^2C 前缀、引导加载程序块写入命令、引导加载程序密钥以及块编号。 紧接的两个字节包含了用于块剩余部分的可选校验和，本示例中为 0x0E8C。 该行的随后两个字节包含可重定位中断向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 I^2C 前缀，后接双字节值，代表根据区块 0x29 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。 该行的剩余部分为空的数据空间。

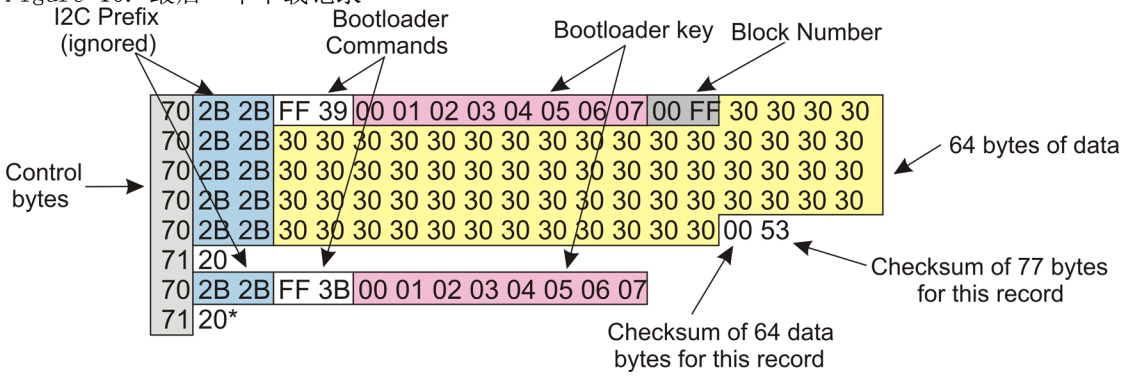
下面的两行包含的是额外的空数据空间。 校验和区块的最后一行包含的是空数据空间，但该行的最后四个字节除外。 这最后四个字节包含：引导加载程序计算的双字节应用程序校验和、来自 Intel 十六进制记录格式的单字节块校验和以及整个 77 字节记录的校验和，地址字节和前缀除外。 地址字节和前缀在其被接收时由引导加载程序进行内部验证。

下一行包含了另一个状态请求和响应。

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。

图 10 显示最后的下载记录：

Figure 10. 最后一个下载记录



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验证所下载应用程序的校验和。 此验证使用校验和块中给出的参数进行。 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

将 E2PROM 用户模块放置入引导下载程序项目中时，将 E2PROM 模块分配进客户保留模块区域。 该区域位于引导加载程序代码区域和应用代码区域之间 （有关信息，请参阅引导加载程序储存器组织）。 这确保 E2PROM 模块不是应用代码区域的一部分，也不是作为应用校验和而计算。

版本历史记录

版本	创作者	说明
1.2	DHA	添加了版本历史
2.00	DHA	1. 重新组织了内存映射。 2. 已将浮动中断向量表放置于地址 0x0080。 3. 已将校验和块放置于地址 0x0100。 4. 已将引导加载程序启动放置于地址 0x0140。 5. 添加了 SetTemperature() 函数。 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 替换了 .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. 更新了 Application_Checksum_Block 和 TWO_Block_Relocatable_Interrupt_Table. 的初始化。 2. 添加对在工作区浏览器中显示引导加载程序输出文件的支持。 3. 更新了 “Flashsecurity.txt” 章节中 “ 错误设置 ” 中的描述。 4. 添加了 “I2C 和睡眠 ” 章节。 5. 为 CY8C27x43 器件更正了到 flashsecurity.example, custom.lkp 和 boot.tpl 模板的路径。

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 © 2007-2012 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.