Boot Loader 应用说明

HelloM3 应用笔记

北京锐鑫同创科技有限公司

www.realsense.com.cn

www.hellom3.cn

目　录

Boot Loader 是在用户应用程序开始运行之前运行的一段小程序。通过这段小程序，我们可以初始化硬件设备和软件环境，从而将系统的软硬件环境带到一个合适的工作状态，以便执行某些特定的功能。一般来说就是用来升级程序，引导程序或操作系统等。

LM3S 系列的芯片有些具有固化在内部 ROM 的 Boot Loader，有些则没有。根据芯片相应的数据手册可以很容易的确定这个问题。通过固化在 ROM 中的 Boot Loader，可以通过串口(UART0)、SSI(SSI0)、I2C(I2C0)、以太网将程序下载 FLASH 中，而不需要使用 JTAG 调试引脚。下面我们分类讨论关于 Boot Loader 的几个问题。

1、如何使用内部的 Boot Loader。

对于有内部 Boot Loader 的芯片来说，可能会遇到如下的问题：

1、 使用内部的 Boot Loader 有几种接口可供升级。

2、 如何使用内部的 Boot loader。

先回答第一个问题，关于这个问题前面提到了一句，内部 Boot Loader 支持串口(UART0)、SSI(SSI0)、I2C(I2C0)、以太网下载程序，但是并不支持 USB DFU 和 CAN 方式。

再回答第二个问题。

要想弄清楚如何使用内部 Boot Loader，我觉的有必要先说说内部 Boot Loader 是如何工作的。我们知道，在 MCU 复位后读取 0x0000 0000 地址处的数据设置堆栈，在读取 0x0000 0004 地址的数据设置 PC 值，然后用户程序开始执行了。但是，其实在上电后任何复位内核的复位操作中，MCU 内部还悄悄的执行了一些其他的操作。具体来说就是：

1、 先判断 ROM 控制寄存器(RMCTL)的 BA 位(初始状态为 1)的值。如果为 1，则将 0x0100 0000(内部 ROM 的地址)映射到 0x0000 0000 地址，如果位 0 则将 Flash 映射到 0x0000 0000 地址。

2、 因为复位后 RMCTL 的 BA 位为 1，所以此时是将 ROM 空间映射到 0 地址，执行 ROM 启动序列。而 ROM 启动序列的第一步是将 BA 位清零，将 ROM 映射到 0x0000 0000 地址处，将 Flash 映射到 0 地址出。

3、 设置好 ROM 和 Flash 的映射关系后就开始读取启动配置寄存器(BOOTCFG)的内容，如果 EN 位被置位，那么就判断指定管脚的状态与指定极性相比较。

4、 如果指定的管脚与指定的极性相匹配则执行 ROM 中的 Boot Loader。

5、 如果管脚的状态与极性不匹配，则检查 0x0000 0004 地址的内容来判断 Flash 是否已经被编程，如果该地址的数据是 0xFFFF FFFF，则表明 Flash 没有被编程过，那么执行 ROM 中的 Boot Loader。

6、 如果 0x0000 0004 地址的数据是有效的，则表明 Flash 已经被编程了，那么就从 0x0000 0000 处读取堆栈指针，从 0x0000 0004 处读取 PC 值，开始执行用户程序。

看了上述启动序列以后我们明白了如下几个问题：

1、 当给 MCU 上电以后总是要执行 ROM 中的启动序列的。

2、 ROM 的启动序列可以引导 ROM 中的 Boot Loader 的。

3、 执行内部 Boot Loader 是有两个条件的，一个是检测到了指定引脚上的指定电平状态，另一个是内部 Flash 没有被编过程。

4、 如果没有设定 BOOTCFG 寄存器来检测指定引脚的电平，而 Flash 已经被编程了则内部 ROM 的 Boot Loader 是不会再执行了，除非用 JTAG 等调试接口将芯片“解锁”。

根据以上，我们要想让内部 ROM 里的 Boot Loader 总是执行。那么最好的，可能也是唯一的方法就是给 BOOTCFG 寄存器写入特定的内容来检测某个引脚的电平极性。通过这个引脚的电平极性匹配关系来决定是否用内部 ROM 中的 Boot Loader 来升级程序还是引导用户程序。

好了，大政方针已经确定，下面就是该讨论如何付诸实施了。

稍微认真看一下芯片的数据手册，就可以得出如下的方法来给 BOOTCFG 编程。以 LM3S9B96 芯片为例，第一步是将 BOOTCFG 的地址写入到 FMA 寄存器，然后将对 ROM 内的 Boot Loader 的配置也就是要写入 BOOTCFG 的值写入 FMD 寄存器，最后给 FMC 寄存器写入相应的密钥(LM3S9B96 的密钥为 0xA442)和确认位。相应的例程可能会是这个样子的：

```
//write the BOOTCFG flash register to enable PA1 low to enable the ROM boot loader at power on
unsigned long regVal;
  regVal = HWREG(0x400FE000 + 0x1D0);        //BOOTCFG
  if (regVal & 0x80000000)                   //committed yet?
  {
      HWREG(0x400FD000 + 0x000) = 0x75100000;     //FMA=BOOTCFG "address"
      HWREG(0x400FD000 + 0x004) = FLASH_BOOTCFG_PORT_A | FLASH_BOOTCFG_PIN_1 | FLASH_BOOTCFG_DBG1;     //FMD=BOOTCFG
       value (PB5 low/DBG enabled)
      HWREG(0x400FD000 + 0x008) = 0xA4420008;     //FMC=key+commit
       SysCtlDelay(100 * SysCtlClockGet() / (3 * 1000));
  }
```

在第一次给芯片写程序的时候将这段代码加到程序的开始部分，当程序运行的时候会首先开启内部的 Boot Loader，这样以后下载程序的时候就可以不需要调试接口来下载程序了，当然相关的头文件还是要包含进来的。

下面对程序做一下简要的说明：

首先定义一个 unsigned long 型的变量，然后读取当前 BOOTCFG 寄存器的值，并判读它的最高为是否为 1，如果为 1 则表明 BOOTCFG 没有被提交过，也就是说还没有设定好每次都从内部 ROM 中的 Boot Loader 启动，那么就要对 BOOTCFG 配置，否则就是已经配置好了 BOOTCFG，则不需要重新配置了。

对 BOOTCFG 寄存器内容提交的过程是这样的，首先要将 FMA 寄存器的值放入提交 BOOTCFG 寄存器的地址，即 0x7510 0000，有人可能会对 0x7510 0000 这个地址有疑惑，为什么不是 0x400FE1D0 这个地址呢，根据数据手册中内部存储器那一章的非易失性存储器的编程那一句里所讲，里面列出了一个表格，指出如果要提交 BOOTCFG 寄存器，那么 FMA 寄存器里的值应该是 0x7510 0000，而他的数据源则是 FMD，也就是说把要写入 BOOTCFG 寄存器的值放到 FMD 寄存器中。

If 语句的第二句就是配置 FMD 的值的，采用了很直观的方式配置，最后已经就是写入对 BOOTCFG 编程的密钥和确认信息了。

至此，对内部 Boot Loader 的配置已经完成了，应用的时候，首先给指定的管脚加上指定的电平极性，然后启动就会进入 Boot Loader，此时通过前面说过的串口连接电脑就可以用 LM Flash Programmer 来对 MCU 进行下载程序了。至于下载程序的方法和后面的要将的类似，如果不太明白的话可以参考后面的内容。

有个问题需要单独说明一下：当使用 ROM 中的 Boot Loader 时，系统时钟使用的是内部的振荡器，它的频率是 16MHz(±1%)。之所以要提这个问题，是因为以串行方式烧写程序的时候(包括 UART，I2C，SSI)。它们通信都是要一定的速率的，如果超过了可能没有办法正确执行。

使用 UART 接口的时候，系统时钟不能低于 UART 速率的 32 倍，换句话说也就是 UART 的速率不能超过 500K(16M / 32)。使用 SSI 接口的时候系统时钟不能低于 SSI 的 12 倍，也就是 SSI 的速率不能超过 1.3M(16M / 12)。I2C 稍微有点特殊，根据数据手册上说，它可以运行在 100K 和 400K 的速率模式下，因此，它的最高速率是 400K。顺便提一句，使用串行方式的时候一般都有类似主从的概念，Boot Loader 通常都运行在从机模式下，I2C 在从机模式下使用的默认地址是 0x42。

2、通过 USB DFU 方式下载程序

对于没有内部 Boot Loader 的芯片来说，那么你用起来相对来说就不需要考虑那么多了，安安心心的把 Boot Loader 下载到 Flash 里运行吧。

要说明的是内部的 Boot LoadeR 不支持 USB DFU。要想使用 USB DFU 的方式来升级程序不管内部 ROM 中有没有 Boot Loader 都得把他下载到 Flash 中。

使用 USB DFU 下载程序的方式如下：

首先打开 boot_usb 目录下面的工程文件。打开 bl_config.h 这个文件，看清楚里面的配置，这里面的配置起着至关重要的决定。它的每一行都有很长的英文解释。要做的就是记住 APP_START_ADDRESS 定义的值，它关系到将来应用程序的下载地址的修改。然后把 ENABLE_UPDATE_CHECK 的这个定义给使能，接着使能 FORCED_UPDATE_PERIPH、FORCED_UPDATE_PORT、FORCED_UPDATE_PIN、FORCED_UPDATE_POLARITY、FORCED_UPDATE_WPU 这几个的定义。这几个定义分别定义了要检查是否升级的引脚的端口模块、端口、引脚序号、匹配电平类型和上下拉电阻。完成之后就可以编译并下载程序到 MCU 了。

完成以上步骤之后，用一根 USB 线将开发板和电脑的 USB 线相连，将 bl_config.h 中设定的引脚给定升级所需要的电平，重启开发板。如果之前没有安装过 USB DUF 的驱动那么需要先安装 USB DFU 的驱动，该驱动所在位置为“光盘目录/tools/windows_drivers”。安装好驱动之后在设备管理器中应该能发现如图 1 所示的设备：

图 1　安装好 USB DFU 驱动后出现的设备

当发现该项不正常的是需要重新安装驱动确保驱动正确安装。安装好驱动之后打开 LM Flash Programmer 软件(这个软件的安装包在“关盘目录/tools/LMFlashProgrammer”下面，也可以在 TI 或流明官网去下载)，会在 configuration 选项卡里的 Interface 一栏的下拉菜单中找到 USB DFU 这一项，选择该项之后 Select Device 里面会出现一个设备，如图 2 所示：

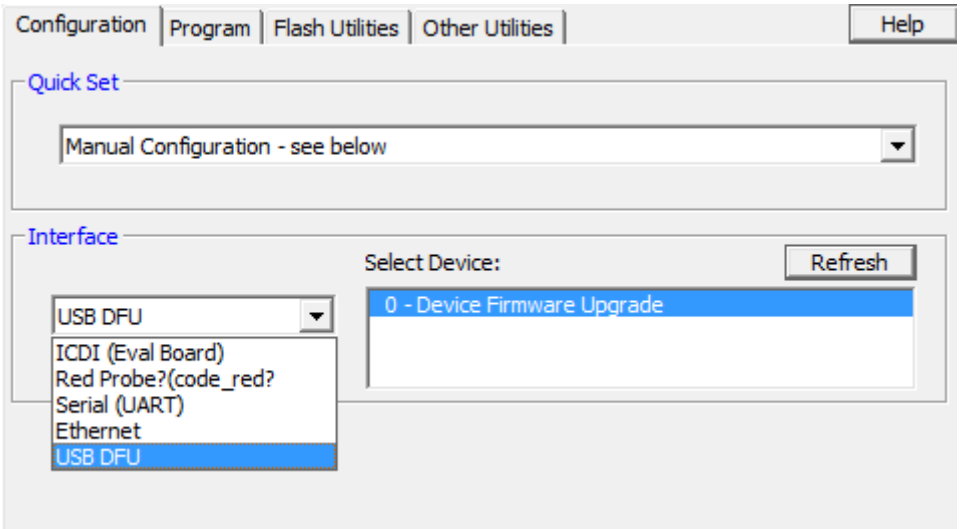

图 2　LM Flash 中的 USB DFU

如果后面没有出现设备，可以重启开发板并点 Refresh 这个按钮试试。如果能正确的走到这一步，说明可以通过 USB DFU 方式下载程序了。下载的方法是点 LM 的第二个选项卡 Program，在 Select .bin file 中通过 Browse 寻找要烧写的.bin 文件。特别要注意的是 Program Address Offset 后面的值，该值代表的是烧写的程序的起始地址。前面我们已经说过，还记得 bl_config.h 中的 APP_START_ADDRESS

这个定义吗，它就是定义的用户程序的起始地址，因此，此处一定要填写 bl_config.h 定义的这个值。上面两个是校验和编程完毕后重启 MCU，随你怎么选择，我是都选上了。相关图片如图 3 所示：

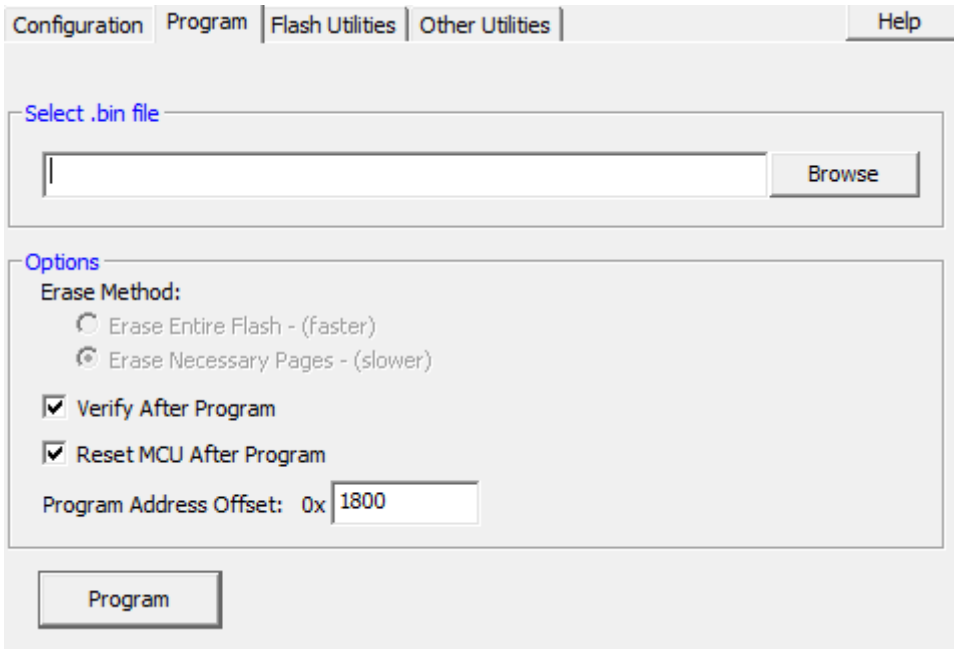

图 3　USB DFU 烧写程序的设置

设置完毕之后点击 Program 就开始对 MCU 编程了。一般来说编程和校验是不会有问题的。最多的问题还是出在应用程序中。在应用程序中要修改它的起始地址，具体来说就是将连接问的首地址改为 bl_config.h 中定义的地址，此处为 0x1800。修改的方法如下：

1、 KEIL RMDK 环境中

修改的方法是在需要修改的工程的 Options 的 Linker 选项卡中，去掉 Use Memmory Layout…前面的复选框中的勾，然后点击 Scatter File 后面的 Edit 来编辑该工程的分散加载文件。如图 4 所示：

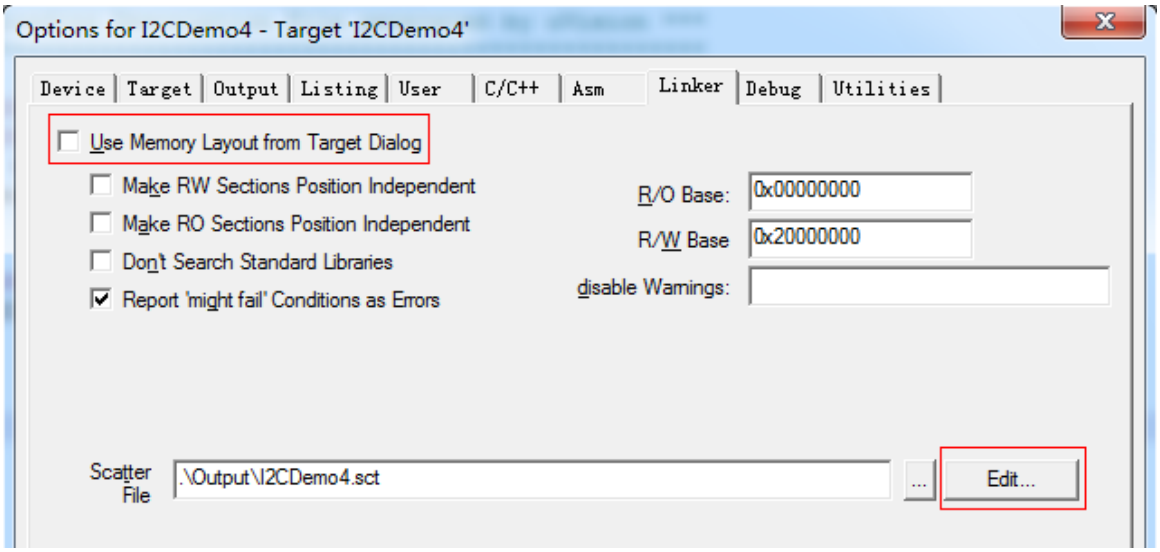

图 4　修改分散加载文件

点击 Edit 之后在弹出的分散加载文件，一般是下面的样子的，如图 5 所示：

```
; *************************************************************
; *** Scatter-Loading Description File generated by uVision ***
; *************************************************************

LR_IROM1 0x00000000 0x00040000  {    ; load region size_region
  ER_IROM1 0x00000000 0x00040000  {  ; load address = execution address
   *.o (RESET, +First)
   *(InRoot$$Sections)
   .ANY (+RO)
  }
  RW_IRAM1 0x20000000 0x00018000  {  ; RW data
   .ANY (+RW +ZI)
  }
}
```

图 5　分散加载文件

将其中用红色方框标出来的两个值都改为 0x0000 1800，然后重新编译即可生成 USB DFU 下载或用其他(非内部 ROM 中的)Boot Loader 下载程序所需的 bin 文件。

2、 在 IAR 环境中

IAR 的分散加载文件是以.icf 为后缀的文本文件，这个文件存放的位置一般在工程的根目录下面，如果没有的话，可以在工程上右击选择 Options，然后再点击 Linker，就会出现 icf 文件所在的地址了。如图 6 所示：

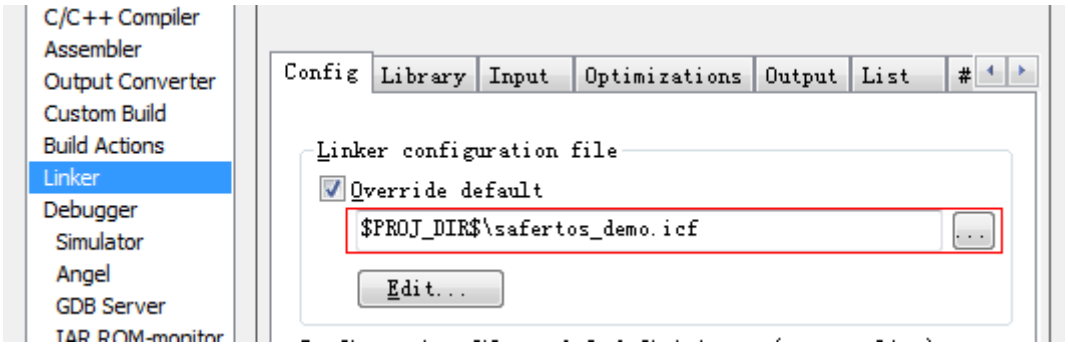

图 6　IAR 分散加载文件的地址

因为 IAR 只能用外部的记事本或其他的文本编辑器打开，所以这里点击 Edit 是没有用的(目前我的版本是这样的)，用记事本或其他文本编辑器打开这个文件之后修改 Flash 定义的那条语句，如下图所示：

```
//
// Define a region for the on-chip flash.
//
define region FLASH = mem:[from 0x00000000 to 0x0003ffff];

//
```

图 7　IAR 要修改的地方

将红色方框中的值修改为 bl_config.h 中定义的值，比方说 0x0000 1800。然后重新编译即可下载到单片机了。

如果没有使用内部 ROM 中的 Boot Loader 则它在 MCU 复位的时候首先执行，判断定义的管脚是否符合相应的电平，来确定是否需要给程序升级。如果是的话就会初始化 USB 等外设，然后等待电脑往 Flash 里下载程序，否则的话就直接引导应用程序运行。

3、以串口方式升级程序

无论是放在 ROM 中还是 Flash 的 Boot Loader 都支持以串口的方式下载程序。在第一节里已经明确的讲了内部 ROM 中的 Boot Loader 是如何运行的。无论是执行内部 ROM 中的 Boot Loader 还是执行 Flash 中的 Boot Loader 用串口来升级程序。它们执行的原理都是差不多的，都是先检查预先定义好的管脚电平(ROM 中的 Boot Loader 也可能是检查空片)，然后确定是否需要升级应用程序。当检测到了升级的条件就会设置串口及其他的外设，然后等待电脑的串口发送命令或数据。

无论你的电脑上用的是自带的串口还是用 USB 转的串口，只要有就可以用 LM Flash Programmer 下载应用程序。首先打开 LM Flash Programmer，在 configuration 选项卡里的 Interface 一栏里选择 Serial(UART)，点击后面的 Device Manager 按钮，在弹出的设备管理器里查看要使用的串口是哪一个，比方说我用的是 USB 转的串口。如下图所示：

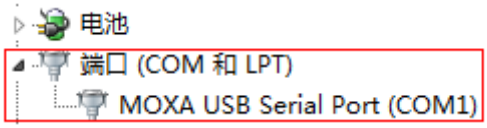

图 8　我电脑上的串口

所以在端口配置的时候 COM Port：应该选择 COM1，波特率应该选择你芯片上实际配置串口的波特率，如果是 Boot Loader 里面配置了自动适应波特率的话，则 Disable Auto Band Support 前面的复选框应该去掉。如果波特率是定死的话则应该勾选该项，然后再 Band Rate：一项里选择相应的波特率。顺便提一句，如果芯片是空片，而且 BOOTCFG 寄存器没有被修改过，此时的执行内部 Boot Loader 的时候波特率是自动适应的。至于最后一个 Transfer Size 采用默认的即可。例如我的串口配置情况是这样的：

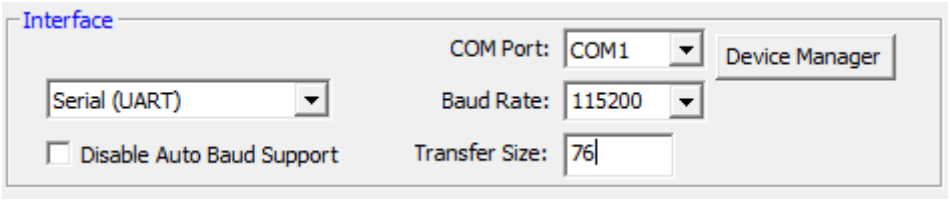

图 9　我的串口配置情况

配置好串口之后，就可以在 Program 选项卡里选择要下载的程序和要下载程序的地址了。如下图所示：

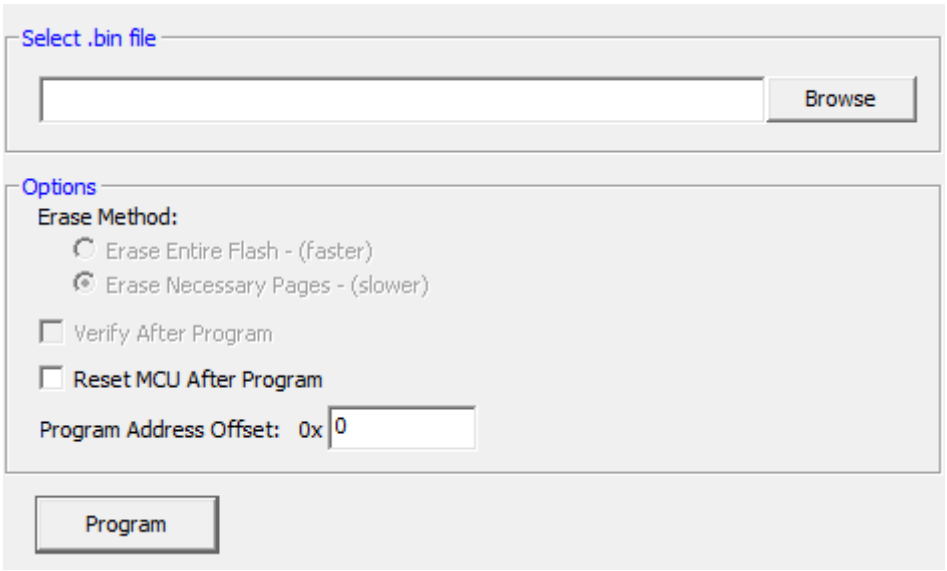

图 10　下载程序界面

这个界面和之前的 USB DFU 方式下载程序差不多，和后面要讲的下载方式也类似。就不做过多介绍了，但是有一点要着重说明一下。那就是下载地址的问题。

通过 ROM 或 Flash 中下载程序的时候，这个下载地址是不一样的。在 ROM 中运行 Boot Loader 的时候可以视为 Flash 是空的，下载地址应该是 0x0(要是这个地址上你放了别的东西如 boot_usb，此时就另当别论了)，如果用的是 Flash 里的 Boot Loader，因为你的 Boot Loader 已经在 0x0 这个地址上了，如果你不想把这个 Boot Loader 覆盖掉的话(更新 Boot Loader 除外)那么你就应该把这个地址设定为 bl_config.h 中 APP_START_ADDRESS 设定的程序起始地址：比如说 0x1800。这个要根据实际情况灵活运用，一般来说如果用 ROM 中的 Boot Loader 则该地址为 0，如果是 Flash 里的 Boot Loader 则该地址为 bl_config.h 中设定的值。

4、通过以太网接口升级程序

ROM 中的 Boot Loader 也支持以太网接口的方式升级程序，当然 Flash 中的也支持。下面就来说一说如何通过网口来升级程序。

通过网口升级程序有两个问题要说明一下：

1、地址问题：通过网口下载程序，应用程序的地址根据是 ROM 的中的 Boot Loader 还是 Flash 中的 Boot Loader 会有所区别。如果是 ROM 中的 Boot Loader，下载到 Flash 的地址必须是从 0 开始的，这个不能改变。因此编译应用程序的时候，SCT 或 ICR 中的地址要设置为 0。如果是用的 Flash 中的 Boot Loader 则下载地址位 bl_config.h 中 APP_START_ADDRESS 定义的地址，一般是 0x1800。

2、也是地址问题，不过和第一个的地址不是同一类型的地址，这个地址问题指的是芯片的 MAC 地址。我们知道，以太网通信有两个重要的地址 IP 地址和 MAC 地址。MAC 地址指的就是网卡的物理地址。如果使用内部 ROM 中的 Boot Loader 的话，这个 MAC 地址是放在 USER0/USER1 中的，它们的格式是 U0B0-U0B1-U0B2-U1B0-U1B1-U1B2，这里的 U0B0 指的是 USER0 的 7:0 位，也可以说是 Byte 0，U0B1 指的是 USER0 的 15:8 位，也可以说是 Byte 1，其他的依此类推。但是有一个重要的问题，就是如果没有设置 USER0/USER1 的时候，MAC 地址的默认值为 00-1a-b6-00-64-00，这个一定要记住。也就是说使用 ROM 中的 Boot Loader 下载程序的时候是不能把芯片 MAC 地址设为 FF-FF-FF-FF-FF-FF 的。而如果使用的是 Flash 中的 Boot Loader 的话，则芯片的 MAC 地址可以是全 F，也可以是 bl_config.h 中指定的地址。

下面就来看看如何使用 LM Flash Programmer 通过网口来下载程序：

首先，将电脑和开发板的网口连接，然后打开开发板电源。

接着打开 LM Flash Programmer，在第一个选项卡 configuration 里的 Interface 里面选择 Ethernet，在下面的 Ethernet Adapter 里面选择和开发板相连的网口，然后在 Client IP Address 里面填写和该网卡的 IP 地址在同一个网段内的其他 IP 地址，在 Client MAC Address 里面填写相应的 MAC 地址：如果用的是 ROM 中的 Boot Loader 而且之前没有对 USER0/USER1 修改过的话此处填写芯片默认的 MAC 地址 00-1a-b6-00-64-00，否则填写修改过的 MAC 地址。如果用的是 Flash 里的 Boot Loader 而且在 bl_config.h 中没有设置 MAC 地址，此处可填写全 F，格式同前面的格式。相关图片请参考图 11.1 和图 11.2。

在第二个选项卡 Program 里选择要下载的 BIN 文件，我们注意到，这里的 Options 是灰色的，下载程序的地址是无法修改的。在写应用程序的时候注意连接地址下载就没有问题。

当使用以太网接口下载或升级程序的时候，LM Flash 会用到 BOOTP 和 TFTP 协议，它们和正常的以太网环境共存不会造成任何问题(当然会占用一点点的带宽)，它们使用的都是标准的协议。BOOTP 协议用来确定服务器端和客户端的 IP 地址，以及固件的映像名称，它使用 UDP/IP 数据包在服务器和客户端通信。TFTP 也使用 UDP/IP 数据包在服务器和客户端通信，它将固件的映像传递给客户端。这里的客户端指的就是 Boot Loader。

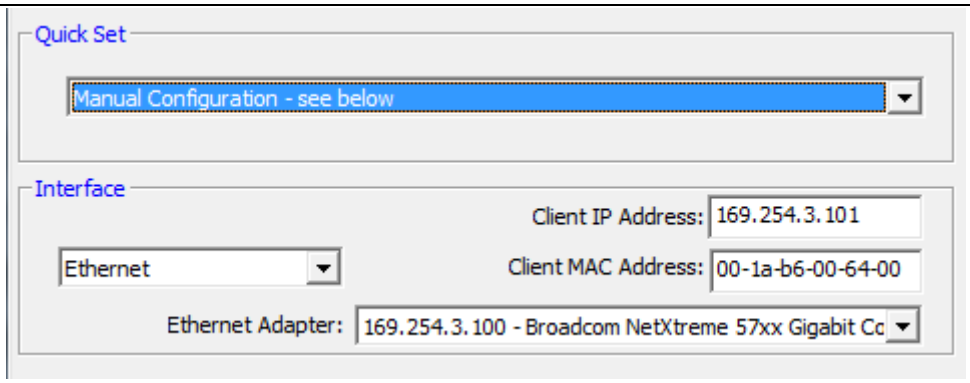

图 11.1　ROM 中运行 Boot Loader 的配置方法

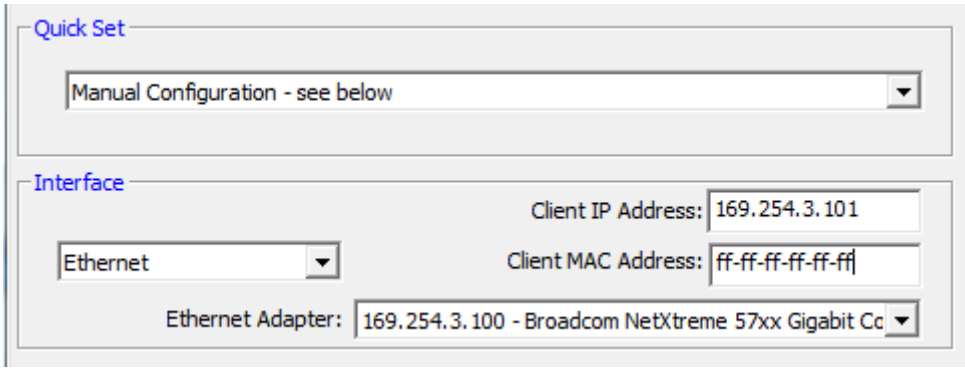

图 11.2　Flash 中运行 Boot Loader 的配置方法

关于协议更详细的信息请参考如下的网址：

http://tools.ietf.org/html/rfc951.html

http://tools.ietf.org/html/rfc1350.html

5、I2C 和 SSI 方式下载程序

前面介绍了几种通过 LM Flash Programmer 下载程序的方法，这一节将要介绍的是通过 I2C 和 SPI 接口来下载程序的方法。之所以要把他们列在一起来说是因为它们有这么几个共同点：

1、都是串行方式发生和接收数据

2、使用相同的串行协议(事实上，UART 使用的也是这样的协议)

3、它们在电脑上没有相应的接口，不能用 LM Flash 来下载。只能通过一个芯片给另一个升级。

基于以上几种原因，我把这两个综合起来将，只是大概讲讲它们通信的过程。通信就包括发送和接收数据包。

发送数据包的过程：

1、先发送要发送的包的大小，它的值时整个数据数据部分的长度+2 个字节。

2、发送数据部分的校验和。

3、发送数据部分。

4、等待一个应答信号，确认对方是否正确接收到了数据。

接收数据包的过程：

1、等待接收一个非零的数据。这非常重要，因为设备(从机)会在发送和接收一个数据之间发送一个零字节。第一个接收到的字节表示将要接收的数据包的大小。

2、接收第二个字节，该字节代表的是将要接收到的数据的校验和。

3、接收剩余的数据，长度为包的大小(即接收到的第一个字节) – 2 个字节。比方说第一个字节接收到的是 3，那么实际上接收到的数据部分长度为 3-2=1。

4、 计算接收到的数据部分的校验和，和接收到的校验(也就是第二个字节)和是否相吻合，

5、 如果校验和正确的话则返回(ACK)信号，否则返回(NAK)信号。告诉主机是否正确的接收到了数据。

以上两个过程大致描述了主机和从机是如何通信的，但是这个只是实际的数据包的通信过程，事实上主机在给从机发送数据包之前都会先发送一个命令字节。命令字节及其代表的含义如下：

COMMAND_PING =0x20	该命令长度是一个字节，用来从 Boot Loader 接收一个字节，表明它们成功建立了通信。它的格式是这样的： unsigned char ucCommand[1]; ucCommand[0] = COMMAND_PING;

COMMAND_DOWNLOAD =0x21	该命令的长度为9个字节，用来告诉Boot Loader要把数据放到哪里，共有多少数据。该命令共包含了两个 32 位的值，都是先发送 MSB。该命令同时还会擦除一段 Flash 空间，因此会导致发送 ACK/NAK 的时间会延长，所以它的后面要跟一个 COMMAND_GET_STATUS 命令，确保命令成功执行。
	该命令的格式是这样的：
	unsigned char ucCommand[9];
	ucCommand[0] = COMMAND_DOWNLOAD;
	ucCommand[1] = Program Address [31 : 24];
	ucCommand[2] = Program Address [23 : 16];
	ucCommand[3] = Program Address [15 : 8];
	ucCommand[4] = Program Address [7 : 0];
	ucCommand[5] = Program Size [31 : 24];
	ucCommand[6] = Program Size [23 : 16];
	ucCommand[7] = Program Size [15 : 8];
	ucCommand[8] = Program Size [7 : 0];

COMMAND_RUN =0x22	该命令告诉 Boot Loader 转到某一个地址开始执行。该命令包含一个 32 位的值，表示要转到的地址，先发送 MSB。
	该命令的格式是这样的：
	unsigned char ucCommand[5];
	ucCommand[0] = COMMAND_RUN;
	ucCommand[1] = Run Address[31 : 24];
	ucCommand[2] = Run Address[23 : 16];
	ucCommand[3] = Run Address[15: 8];
	ucCommand[4] = Run Address[7 : 0];

COMMAND_GET_STATUS =0x23	该命令返回最后一个命令执行的状态，一般来说每发送一个命令都要跟随这么一个命令确定上一个命令是否被正确执行等。Boot Loader 发送的相应包含当前状态的一个字节的数据包。 它的格式是这样的： unsigned char ucCommand[1]; ucCommand[0] = COMMAND_GET_STATUS; 返回的状态包括如下几种： COMMAND_RET_SUCCESS COMMAND_RET_UNKNOWN_CMD

	COMMAND_RET_INVALID_CMD COMMAND_RET_INVALID_ADD COMMAND_RET_FLASH_FALL

COMMAND_SEND_DATA =0x24	该命令跟随在 COMMAND_DOWNLOAD，如果数据很多也可以跟随在另一个 COMMAND_SEND_DATA 命令之后，如果连续发送数据则数据的地址会递增，并且续传之前编程的位置。它每次最多传输 252 个字。调用该命令之前应该先调用 COMMAND_GET_STATUS 命令，以确保数据成功编入 Flash 中。如果 Boot Loader 返回了一个 NAK 信号，则它不会增加当前的地址，允许之前的数据重新发送。
	他的格式大概是这样的：
	unsigned char ucCommand[9];
	ucCommand[0] = COMMAND_SEND_DATA;
	ucCommand[1] = Data[0];
	ucCommand[2] = Data[1];
	ucCommand[3] = Data[2];
	ucCommand[4] = Data[3];
	ucCommand[5] = Data[4];
	ucCommand[6] = Data[5];
	ucCommand[7] = Data[6];
	ucCommand[8] = Data[7];

COMMAND_RESET =0x25	该命令告诉 Boot Loader 复位。当下载成功后需要重启以执行新的程序的时候发送该命令。当然如果下载不成功想要重启也是可以的。总之，Boot Loader 接收到该命令后就会启动一个正常的启动序列。从机在启动复位序列之前会向主机返回一个 ACK 信号。表明命令正确接收，将要进入复位序列。 他的格式大概是这个样子的： unsigned char ucCommand[1]; ucCommand = COMMAND_RESET;

虽然我建议你没事不要乱改上面定义的命令，但是如果你知道你想干什么的话，想要修改增加或删除命令的话，它们在 boot_loader/bl_command.h 中，它们是 Stellaris Peripheral Driver Library 的一部分。

6、从应用程序进入 Boot Loader

在应用程序中你可以随时调用一个函数进入到 Boot Loader，想想这个有多么诱人吧。你放说你的产品外面只有一个串口接出来，其他的都封装起来了，但是你还想日后可能会给你的产品升级。这个时候你可以通过给串口发送一个特定的命令使内部的 Boot Loader 通过这个串口来给你的产品升级程序。

要想使应用程序能够跳到 Boot Loader 中执行升级程序，它必须具备一个 Boot Loader，无论是 ROM 中固化的还是 Flash 中的。最常见的就是通过 USB、以太网和串口来升级程序，但是也可以通过 I2C 和 SSI 来升级程序。和前面讲的一样，ROM 中的 Boot Loader 不支持 USB 的方式。

先说说如何使应用程序进入 Flash 中的 Boot Loader 执行程序。首先要将 Boot Loader 烧如 0 开始的 Flash 空间，然后把应用程序烧如到 Boot Loader 的 bl_config.h 中 APP_START_ADDRESS 所定义的地址。这样可以由 Boot Loader 引导应用程序。应用程序中包含有跳转到 Boot Loader 的代码，通过

外部给它的特殊命令、屏幕操作、按键……，随你怎么操作，只要你能执行到这个跳转函数它就会跳到 Boot Loader 中运行。一个示例性的代码可能会是这个样子的：

```
//*****************************************************************************
//
// Passes control to the bootloader and initiates a remote software update.
//
// This function passes control to the bootloader and initiates an update of
// the main application firmware image via UART0, Ethernet or USB depending
// upon the specific boot loader binary in use.
//
// \return Never returns.
//
//*****************************************************************************
void
JumpToBootLoader(void)
{
    //
    // We must make sure we turn off SysTick and its interrupt before entering
    // the boot loader!
    //
    SysTickIntDisable();
    SysTickDisable();

    //
    // Disable all processor interrupts.  Instead of disabling them
    // one at a time, a direct write to NVIC is done to disable all
    // peripheral interrupts.
    //
    HWREG(NVIC_DIS0) = 0xffffffff;
    HWREG(NVIC_DIS1) = 0xffffffff;

    //
    // Return control to the boot loader.  This is a call to the SVC
    // handler in the boot loader.
    //
    (*((void (*)(void))(*(unsigned long *)0x2c)))();
}
```

这段代码是直接从 TI 提供的例程里摘出来的。太多详细代码的分析不属于本文的内容，有兴趣的可以自己去研究一下。根据其注释可以很清楚的知道，进入该函数后首先关闭 systick 及其中断，然后关闭所有的中断，最后跳到了 Boot Loader 的 SVC 中断处理函数(注意，不是应用程序的 SVC 中断)中执行了。再看看 Boot Loader 的 SVCall 中断处理函数做了写什么呢，在源代码中分析，我们发现是它引导了 Boot Loader 的运行，首先把 Boot Loader 复制到 SRAM 中，然后跳到 SRAM 中初始化 bl_config.h 中所配置的硬件信息，然后就是等着 LM Flash Programmer 来升级程序了。

如果芯片有内置的 Boot Loader 的话，应用程序也可以执行 ROM 中的 Boot Loader 来升级应用程序。但是内部 ROM 中的 Boot Loader 不支持 USB。

和执行 Flash 中的 Boot Loader 有所不同，具体来说表现在以下两个方面：

1、链接时的起始地址不同。我们知道，Flash 中的 Boot Loader 要占用一定的 Flash 空间，所以应用程序链接的起始地址是定义在 bl_config.h 中的 APP_START_ADDRESS 这个地址。而 ROM 中的 Boot Loader 不占用 Flash 空间，这时候应用程序是从 0 地址开始的。

2、如何执行 Boot Loader 的问题。如果 Boot Loader 在 Flash 中，0x2C 代表的是 Boot Loader 的中断向量。如果 Boot Loader 在 ROM 中则 0x2C 代表的是应用程序的中断向量。

应用程序执行 ROM 中的 Boot Loader，上述函数中最后一句是不对的，应该修改为 ROM 中执行 Boot Loader 的入口，也就是调用 ROM_UpdateEthernet、ROM_UpdateI2C、ROM_UpdateSSI、ROM_UpdateUART 这几个函数，它们的定义在 Rom.h 中。

有关应用程序调用 Boot Loader 有几个问题，现在总结一下应用程序调用 Boot loader，包括前面可能讲过的，也有没讲过的。

1、 调用 Flash 中的 Boot Loader 注意修改程序的开始地址为 Boot Loader bl_config.h 中的地址。
2、 调用 ROM 中的 Boot Loader 注意修改程序的开始地址为 0。
3、 调用 ROM_UpdateI2C、ROM_UpdateSSI、ROM_UpdateUART 和 ROM_UpdateEthernet 不同。前三个函数没有参数，而以太网的应该写作 ROM_UpdateEthernet(ROM_SysCtlClockGet());
4、 应用程序通过以太网升级的时候，必须在应用程序里配置好网口，并且 MAC 地址已经被编程而且正在使用。(注：此时的 MAC 地址可以全是 F，只要 IP 地址在同一网段内即可)。
5、 这个网址可能有一些有价值的东西：
http://e2e.ti.com/support/microcontrollers/stellaris_arm_cortex-m3_microcontroller/f/471/p/72308/309419.aspx

结语

好了，关于 Boot Loader 的问题就介绍这么多了。不同的项目中可能用法也会有所不同，刚开始可能大家都是先看懂 TI 的例程，然后慢慢修改，摸索出适合自己项目的 Boot Loader，最后慢慢融会贯通，使用起来就能随心所欲了。这个不是一篇文章看了就能全懂的。

附录 A

北京锐鑫同创是 TI 第三方合作伙伴，专注于 TI Stellaris M3 产品的方案设计、市场推广和技术服务，公司以“把握市场脉搏，专注技术创新，提供诚信服务，实现共赢发展！”为核心价值理念，为客户提供实时、高效的技术和服务。

电话：010-82418301
传真：010-82418302
Email：support@realsense.com.cn
网站：www.realsense.com.cn
技术论坛: www.hellom3.cn