ale Semiconductor
应用笔记

Document Number: AN4655
Rev 0, 01/2013

Kinetis L 系列上的 IIC 引导加载程序设计

作者: Wang Peng

内容

1 概述

很多应用或产品都需要现场升级固件，以便修复某些发现的漏洞，或者用来提高性能。其中大多数的固件升级都不使用专用的调试接口，而是使用 UART、USB 和 I2C 等通信接口。这种情况下，就需要一个串行引导加载程序，以便在不使用调试器或专用编程工具的情况下，通过某个通信接口执行固件升级。

本应用笔记将指导您使用 IIC 接口在 Kinetis L 系列上设计引导加载程序。

2 简介

引导加载程序是一种内置固件，用于通过通信接口将应用程序代码编程到闪存中。

本应用笔记介绍如何使用 Kinetis L 系列 Freedom 开发平台（FRDM-KL05Z 板），将来自 PC 终端的 UART 数据转换到 IIC 总线，以及如何与目标板（L 系列电路板）通信，以实现目标应用程序代码的更新。请参见下图。

图 1. 顶层视图

引导加载程序利用了 AN2295SW_REV1 软件工具(可从 **freescale.com** 获取),这个软件工具广泛应用于当前的 Kinetis 产品，通过 UART 接口来实现应用程序升级。

转接板使用 Freedom FRDM-KL05Z 板将 UART 总线转换为 IIC 总线，将数据重新打包并传输到目标板。

目标板带有内置引导加载程序代码，作为 IIC 从设备与转接板通信。接收到命令和数据后，该引导加载程序会升级目标板上的应用程序。

与本应用笔记关联的示例代码 AN4655SW (可从 **freescale.com** 获取) 可以直接在 FRDM-KL05 板上运行，应将“引导加载程序”下载到目标板、将“UartToIIC”下载到转接板，项目“demo_bootloader”用于生成 S19 文件，该文件可使用 PC 软件下载。

3 软件架构

本应用笔记随附的软件工具 AN4655SW.zip（包含 win_hc08sprg.exe）可从 **freescale.com** 获取，用于解码 S19 文件，并通过 FC 协议与转接板通信。

3.1 转接板

转接板通过 FC 协议与 PC 终端通信。有关 FC 协议的详细信息，请参见《AN2295: 适用于 M68HC08 和 HCS08 MCU 的开发者串行引导加载程序》(可从 **freescale.com** 获取)。

转接板将初始化为 IIC 主设备并与目标板通信，从而通过 IIC 总线与目标板之间接收或发送数据包；其使用数据长度与校验和重新打包数据帧。以下是数据包的格式。

数据长度	原始数据帧	校验和

之后，转接板将从从设备读取数据；通过接收到第一个数据用于确定从设备是否就绪。如果就绪（命令|0x80），转接板将向接收器指示已接收到正确的应答，例如:

当发送给从设备的命令是 0x03，则接收到的应答必须是 0x03|0x80。

首先，它将向目标板发送 FC_CMD_HOOK(0x02)，然后从目标板读取状态，检查目标板是工作于引导加载程序模式，还是用户代码模式。

如果接收到的状态为 FC_CMD_HOOK|0x80，则发送 0xFC 和 PC 终端握手，否则一直检查目标板的状态，直到收到 FC_CMD_HOOK|0x80 为止。

软件流程显示如下:

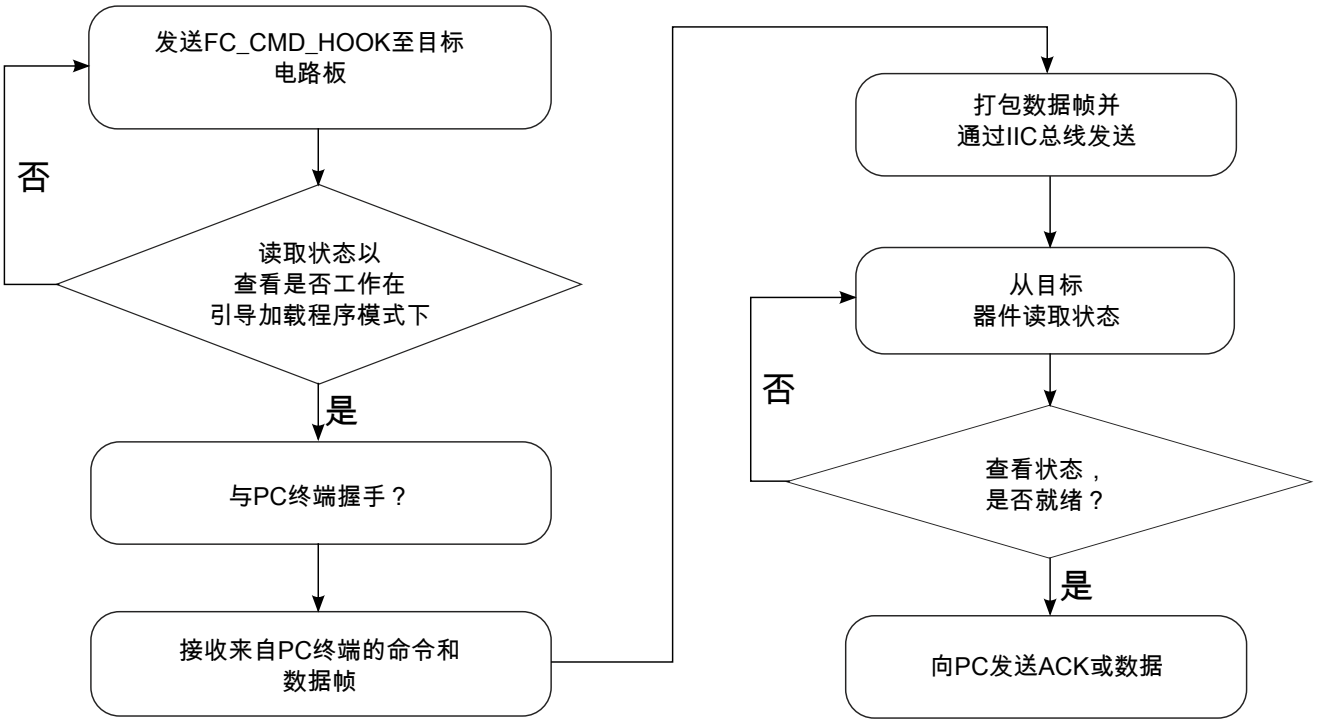

图 2. 转接板软件流程图

转接板充当 PC 终端与目标板之间的桥梁，使用转接板可以通过 PC 将 S19 文件下载到目标板。

3.2 目标板

目标板包含内置引导加载程序代码。上电后，目标板首先将检查工作模式，以确定自身是处于引导模式还是用户代码模式。可以通过几种方法来执行此类检查，例如检查外部 GPIO 的电平。

- 如果 GPIO 引脚为低电平，则目标板将进入引导模式以运行引导加载程序。
- 如果 GPIO 引脚为高电平，则目标板将进入用户代码模式以运行应用程序代码。

但是对于某些应用，可供使用的引脚是有限的，没有针对该目的提供额外的 GPIO。对于此类情况，以下几节介绍了通过闪存中的标志来确定工作模式的方法，该标志可以在应用程序代码中修改。

3.2.1 闪存中的标志

对闪存编程可以使其中的一个位从 1 变为 0，如果该位需要更改为 1，则必须首先擦除闪存。对于 KL05Z 系列芯片，一个扇区就是 1 KB。因此，扇区可以用来保存参数。为了减少擦除次数并延长闪存寿命，使用算法以实现读取-修改-写入操作，在整个扇区空间写满后，可以一次性擦除闪存扇区。

在此应用程序中，标志仅使用一个字节；因此闪存写入总次数如下：

总写入次数 = 1024 * 50k

标志定义如下：

```
#define WORK_MODE_BOOT_LOADER            0xFF
#define WORK_MODE_USER_APP               0x5a
#define WORK_MODE_INVALID_FLAG           0x00
```

下面给出了检查标志来确定目标板工作模式的流程图。

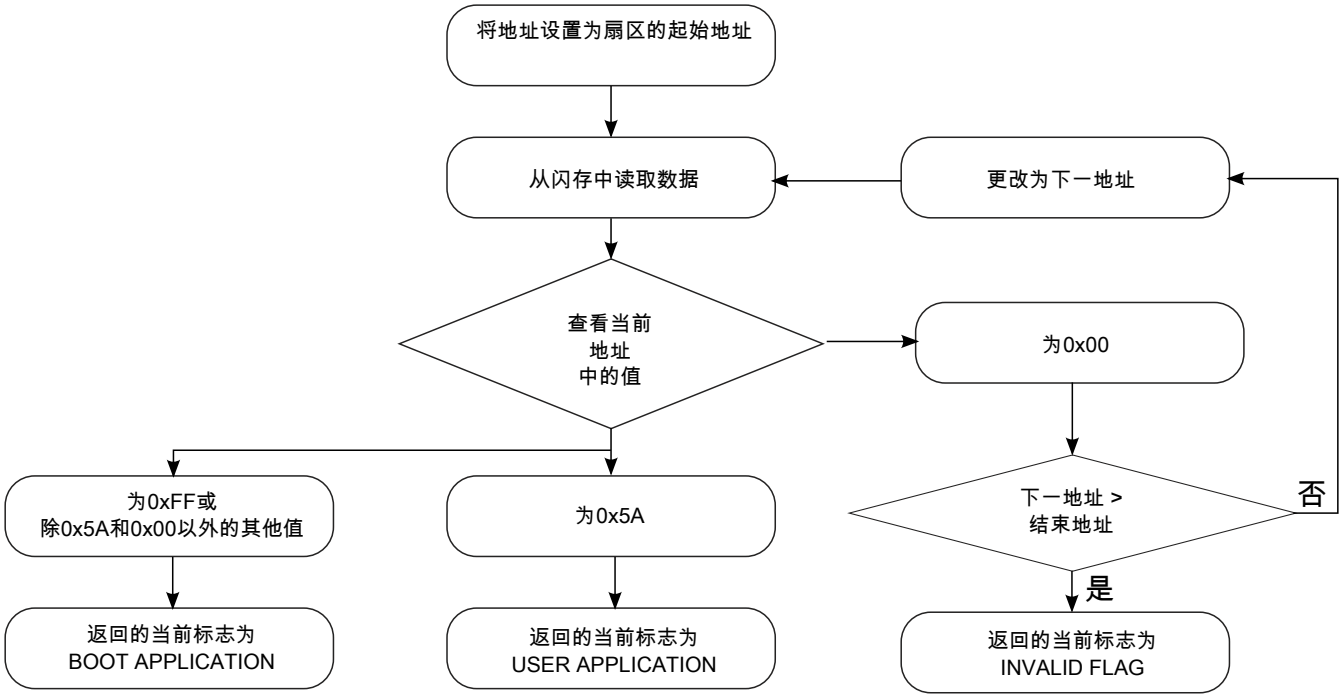

图 3. 检查标志流程图

下面给出了检查标志状态的示例代码片段。

```
uint32_t Check_WorkMode( uint32_t uiStartAddress, uint32_t uiEndAddress,uint32_t *
pCurrentAddress )
 {
uint32_t i;
   for(i=uiStartAddress;i<uiEndAddress;i++)
   {
            switch(*((uint8_t *)i))
            {
            case WORK_MODE_BOOT_LOADER:
                 *pCurrentAddress = i;
                 return WORK_MODE_BOOT_LOADER;
            case WORK_MODE_USER_APP:
                *pCurrentAddress = i;
                return WORK_MODE_USER_APP;
            case WORK_MODE_INVALID_FLAG:
                   break;
            default:
             return WORK_MODE_INVALID_FLAG;
            }
    }
*pCurrentAddress = uiStartAddress;
// if all flag is not matching, return to boot loader mode
return WORK_MODE_INVALID_FLAG;
}
```

此标志可以在应用程序代码或引导加载程序中更改。成功更新应用程序后，标志会更改为用户模式。

在应用程序代码中，用户可以将标志从用户模式更改为引导加载程序模式，在下次复位后，该标志将进入引导加载程序模式，准备更新应用程序代码。

下面显示了修改标志来更改工作模式的流程图。

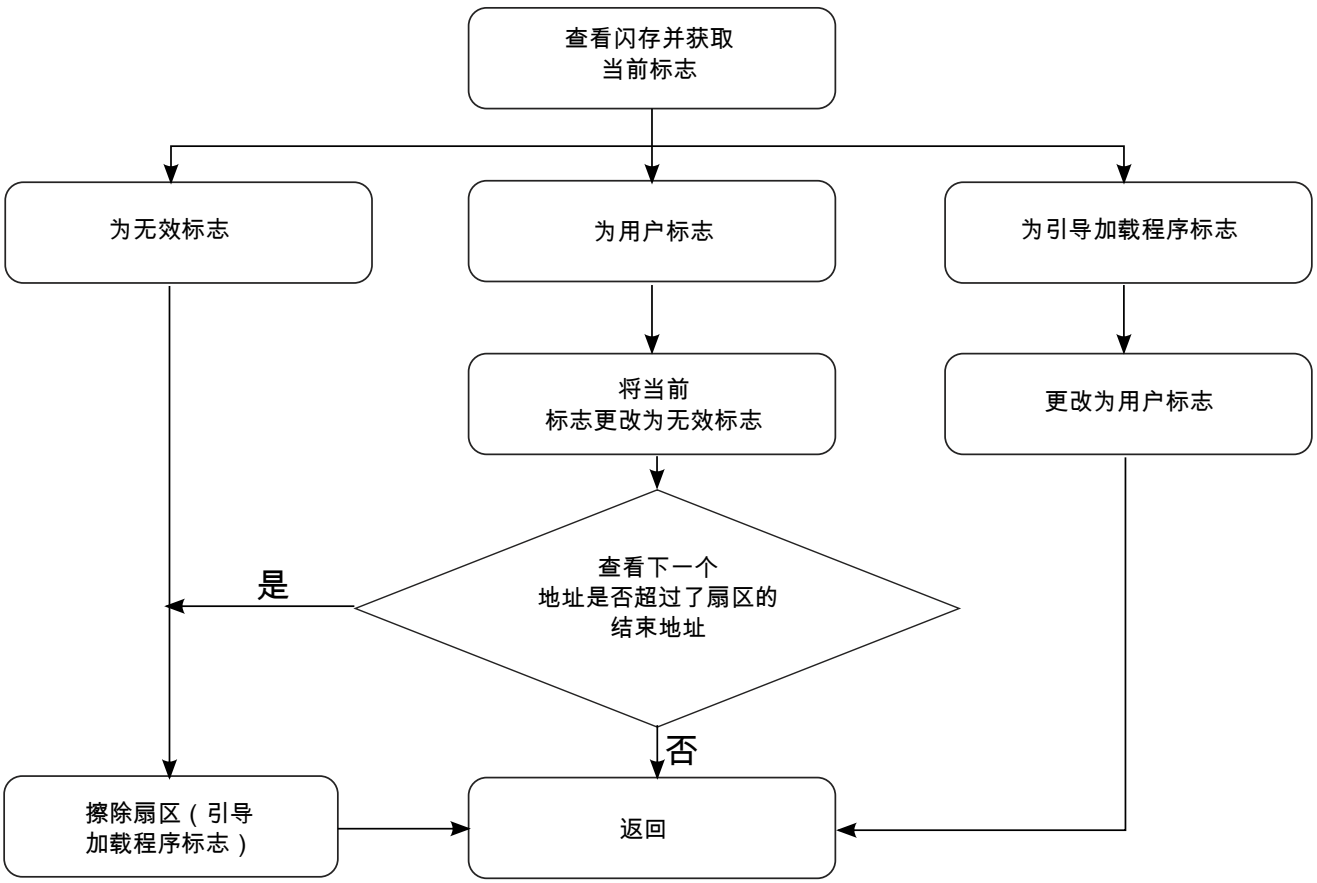

图 4. 修改标志流程图

下面显示了将标志修改为其他状态的示例代码片段:

```
/*****************************************************************************
* Function:         Write_WorkMode
* Description:     modify flag to expected status
* Returns:          result  1 - success
 *                                0 - fail
 *****************************************************************************/
 unsigned char Write_WorkMode(uint8_t WorkMode)
 {
        uint32_t uiCurrentAddress;
        uint32_t uiCurrentAddressContent;
        uint32_t ui32CurrentAddress;
        uint8_t  *pPointer;
        uint8_t uiCurrentWorkMode;
        pPointer = (uint8_t *)&uiCurrentAddressContent;
        uiCurrentWorkMode =
Check_WorkMode(FLASH_FLAG_START_ADDRESS,FLASH_FLAG_END_ADDRESS,&uiCurrentAddress);
        if( WORK_MODE_INVALID_FLAG == uiCurrentWorkMode )
        {
                // erase flash
                if(Flash_SectorErase(FLASH_FLAG_START_ADDRESS)!= Flash_OK)
                {
                        return 0;
                }
                uiCurrentWorkMode = WORK_MODE_BOOT_LOADER;
        }
        if( WorkMode == WORK_MODE_BOOT_LOADER )
        {
                 if( uiCurrentWorkMode == WORK_MODE_USER_APP )
```

```
                {
                        ui32CurrentAddress = (uiCurrentAddress/4)*4;
                        uiCurrentAddressContent = *((uint32_t*)ui32CurrentAddress);
                        pPointer[uiCurrentAddress%4] = WORK_MODE_INVALID_FLAG;
                        if( Flash_ByteProgram(ui32CurrentAddress,
&uiCurrentAddressContent,4) != Flash_OK )
                        {
                                return 0;
                        }
                }
                if( (uiCurrentAddress+1) >= FLASH_FLAG_END_ADDRESS )
                {
                          // this is the last record
                          if(Flash_SectorErase(FLASH_FLAG_START_ADDRESS)!= Flash_OK)
                          {
                                  return 0;
                          }
                }
                // after erase, default is boot loader mode
        }
        else if( WorkMode == WORK_MODE_USER_APP )
        {
                if( uiCurrentWorkMode == WORK_MODE_BOOT_LOADER )
                {
                       ui32CurrentAddress = (uiCurrentAddress/4)*4;
                       uiCurrentAddressContent = *((uint32_t *)ui32CurrentAddress);
                       pPointer[uiCurrentAddress%4] = WORK_MODE_USER_APP;
                       if( Flash_ByteProgram(ui32CurrentAddress,
&uiCurrentAddressContent,4) != Flash_OK )
                       {
                                return 0;
                       }
                }
         }
         else
         {
                //
         }
 return 1;
 }
```

3.2.2 IIC 从驱动器

目标板将 IIC 配置为从设备。与主机通过 IIC 中断服务程序来接收和发送数据。关于中断流程的详细信息，请参见《KL05P48M48SF1RM：KL05 子系列参考手册》(可从 **freescale.com** 获取)。以下是示例代码片段：

```
  void IIC_Irq( void )
  {
    volatile  unsigned char Dummy;
    if( I2C0_S & I2C_S_IICIF_MASK )
    {
         I2C0_S |= I2C_S_IICIF_MASK;
         if( I2C0_S & I2C_S_ARBL_MASK )
         {
         I2C0_S |= I2C_S_ARBL_MASK;
         if( !(I2C0_S & I2C_S_IAAS_MASK) )
         {
                  // IIAAS is 0
                  return;
         }

     }
         if( I2C0_S & I2C_S_IAAS_MASK )
         {
                 I2C0_C1 &= ~I2C_C1_TXAK_MASK;
```

```
                // clear index counter
                m_ucSendIndex = 0;
                m_ucRecIndex = 0;
                // clear frame rec flag
                g_bIICRecFrameFlag = 0;
                if( I2C0_S & I2C_S_SRW_MASK )
                {
                        // slave send data
                        I2C0_C1 |= I2C_C1_TX_MASK;
                        I2C0_D = m_ucSendBuff[m_ucSendIndex++];
                }
                else
                {
                  I2C0_C1 &= ~I2C_C1_TX_MASK;
                  Dummy = I2C0_D;
                }
        }
        else
        {
                if( I2C0_S & I2C_S_SRW_MASK )
                {
                        // if require ACK from master
                        if( I2C0_S & I2C_S_RXAK_MASK )
                        {
                                // no receive the ACK
                                I2C0_C1 &= ~I2C_C1_TX_MASK;
                        Dummy = I2C0_D;
                        // switch to RX
                         }
                         else
                         {
                                I2C0_D = m_ucSendBuff[m_ucSendIndex++];
                         }
                }
                else
                {
                         m_ucRecBuff[m_ucRecIndex++] = I2C0_D;
                         if( m_ucRecIndex > sizeof(uint32_t) )
                         {
                                 m_pRxFrameLength = (uint32_t *)&m_ucRecBuff[0];
                                 if( m_ucRecIndex >= (*m_pRxFrameLength) )
                                 {
                                         // receive a frame data from master
                                         g_bIICRecFrameFlag = 1;
    Memcpy_Byte((uint8_t*)&g_ucIICRxFrameBuff[0],(uint8_t*)&m_ucRecBuff[0],m_ucRecIndex);
                                         // reset index counter
                                         m_ucRecIndex = 0;
                                         // change MCU state to BUSY
                                         m_ucSendBuff[0] = SLAVE_MCU_STATE_BUSY;
                                  }
                          }
                 }
         }
    }
}
```

在 IIC 接收到数据帧后，会将标志置 1（g_bIICRecFrameFlag），以便应用程序代码能够进一步处理该数据帧。

3.2.3 命令说明

引导加载程序会始终检查标志（g_bIICRecFrameFlag）。当标志（g_bIICRecFrameFlag）为 1 时，引导加载程序开始处理接收到的帧。首先，使用校验和来验证接收到的帧是否正确；验证后，解包并处理相应的命令。以下是接收帧的格式。

总数据长度（4 字节）	命令（1 字节）	地址（4 字节）	数据长度（1 字节）	数据	校验和（1 字节）

下表提供了简要的命令汇总。

命令功能	命令	加载程序肯定应答	加载程序否定应答
握手	0x02	0x82, 0xFC	0x82, 0x03
标识	0x49	0xC9，标识信息	0xC9, 0x03
擦除扇区	0x45	0xC5, 0xFC	0xC5, 0x03
写入	0x57	0xD7, 0xFC	0xD7, 0x03
读取	0x52	0xD2，数据	0xD2, 0x03

- 握手命令

  握手命令（编码为 0x02）的接收数据包如下所示。

总数据长度（4 字节）	指令（1 字节）	地址（4 字节）	数据长度（1 字节）	数据	校验和（1 字节）
6	0x02	-	-	-	CS

  命令应答如下。

命令（1 字节）	数据
0x82	0xFC/0x03

  - 如果接收到的状态为 0xFC，则表示目标板正在引导加载程序模式下工作，并已准备好与转接板通信。
  - 如果接收状态为 0x03，则表示目标板处于用户模式，且无法接收其他命令。

- 标识命令

  标识命令（编码为 0x49）的接收数据包如下表所示。

总数据长度（4 字节）	命令（1 字节）	地址（4 字节）	数据长度（1 字节）	数据	校验和（1 字节）
6	0x49	-	-	-	CS

  需要提供的 MCU 信息如下所示。

  - 协议版本 - 1 字节
  - 系统设备识别寄存器(SDID)内容（$14A 适用于 K60 系列），r（13-16 位）为反映当前芯片等级的芯片版本编号 - 2 字节
  - 可重新编程存储器区域的编号 - 4 字节
  - 可重新编程区域的起始地址 - 4 字节
  - 可重新编程存储器区域的结束地址 - 4 字节
  - 原始向量表的地址（1 KB） - 4 字节
  - 新向量表的地址（1 KB）- 4 字节
  - MCU 擦除块的长度 - 4 字节
  - MCU 写入块的长度 - 4 字节
  - 标识符信息，以零结尾的字符串 - n 字节

  下面代码片段是标识信息的结构体。

```
typedef uint32_t addrtype;
typedef struct
{
```

```
            unsigned char Reserve ;          //  reserve bytes for 4  bytes align
            unsigned char Version;               /** version */
            uint16_t Sdid;                       /** Sd Id */
            addrtype BlocksCnt;                  /** count of flash blocks */
            addrtype FlashStartAddress;          /** flash blocks descriptor */
            addrtype FlashEndAddress;
            addrtype RelocatedVectors;           /** Relocated interrupts vector
table */
            addrtype InterruptsVectors;          /** Interrupts vector table */
            addrtype EraseBlockSize;             /** Erase Block Size */
            addrtype WriteBlockSize;             /** Write Block Size */
            char IdString[ID_STRING_MAX];        /** Id string */
}FC_IDENT_INFO;
```

命令应答如下所示。

命令（1 字节）	数据
0xC9	标识信息

- 擦除命令

  擦除命令（编码为 0x45）的接收数据包如下表所示。

总数据长度（4 字节）	指令（1 字节）	地址（4 字节）	数据长度（1 字节）	数据	校验和（1 字节）
10	0x45	地址	-	-	CS

  命令应答如下所示。

命令（1 字节）	状态
0xC5	0xFC/0x03

- 写命令

  写命令（编码为 0x57）的接收数据包如下所示。

总数据长度（4 字节）	命令（1 字节）	地址（4 字节）	数据长度（1 字节）	数据	校验和（1 字节）
总长度	0x57	地址	-	-	CS

  命令应答如下表所示。

命令（1 字节）	状态
0xD7	0xFC/0x03

- 读命令

  读命令（编码为 0x52）的接收数据包如下所示。

总数据长度（4 字节）	命令（1 字节）	地址（4 字节）	数据长度（1 字节）	数据	校验和（1 字节）
11	0x52	地址	要读取的数据长度	-	CS

  命令应答如下所示。

命令（1 字节）	数据
0xD2	数据

- 退出命令

  该命令不需要任何应答。

  接收到该指令后，需要修改标志并跳转到新中断向量表的起始地址。

4 存储器分配

引导加载程序代码占据闪存存储器的第一个区域（最低的存储器地址空间）。这种放置方式使可供用户使用的闪存空间的开始地址发生了变化，所以需要在用户的应用程序中修改链接文件（IAR 中的 ICF 文件，CodeWarrior 中的 LCF 文件）的应用程序开始地址。ICF 和 LCF 链接器文件修改示例如下：

Kinetis L KL05Z

以下代码片段提供了在 IAR6.4 中修改 ICF 文件的示例。

```
// default linker file

define symbol __ICFEDIT_region_ROM_start__ = 0x00;

// modified Linker file for KL05Z 32k flash

define symbol __ICFEDIT_region_ROM_start__ = 0x1000;
```

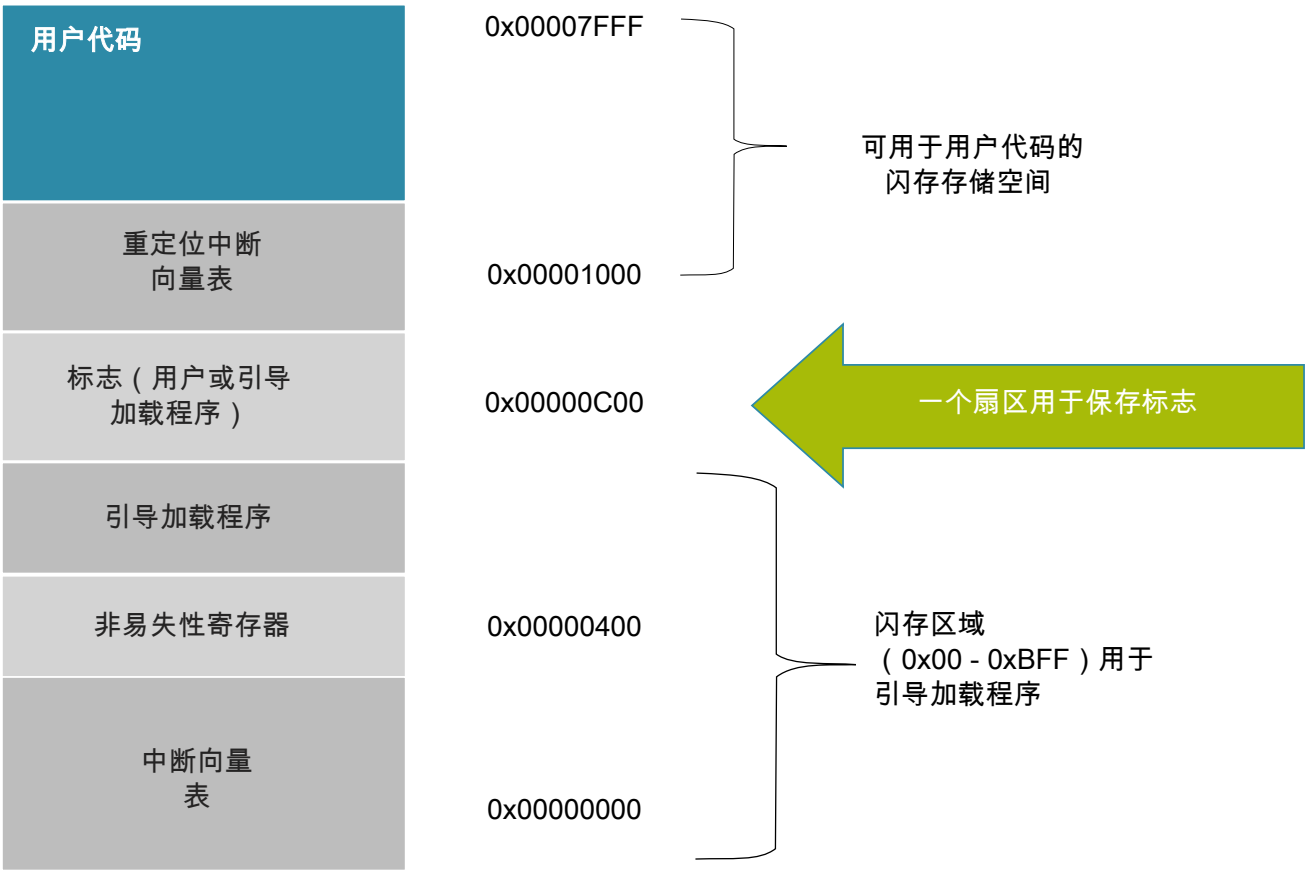

图 5. 存储器分配

在用户代码中，用户可以将工作模式更改为引导加载程序模式，使之能在下次复位时再次更新代码。在示例代码中，从 UART 接收到“b”时，其标志将更改为引导加载程序模式。示例代码片段如下所示。

```
if( UART0_S1 & UART0_S1_RDRF_MASK )
{
ch = UART0_D;
     if( ch == 'b' )
     {
        // change flag
         Write_WorkMode(WORK_MODE_BOOT_LOADER);
        // generate a software reset, wait MCU reset and enter into boot loader mode
        SCB_AIRCR = SCB_AIRCR_SYSRESETREQ_MASK|0x05fa0000;
        while(1);
      }
}
```

5 结论

本文档介绍如何利用转换板作为转接桥，另一块电路板作为目标板，从而实现 IIC 引导加载程序。用户也可以在应用软件中自行添加引导加载程序。

6 参考

可从 **freescale.com** 获取以下参考文档:

- 《KL05P48M48SF1RM：KL05 子系列参考手册》
- 《AN2295：适用于 M68HC08 和 HCS08 MCU 的开发者串行 Boot Loader》
- 《Kinetis L 外围模块快速参考用户指南》(KLQRUG)

7 术语表

UART	通用异步接收器/发送器
IIC	IIC
FCCOB	闪存通用命令对象
WDOG	看门狗
MCG	多用途时钟发生器

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

本文档中的信息仅供系统和软件实施方使用 Freescale 产品。 本文并未明示或者暗示授予利用本文档信息进行设计或者加工集成电路的版权许可。 Freescale 保留对此处任何产品进行更改的权利，恕不另行通知。

Freescale 对其产品在任何特定用途方面的适用性不做任何担保、表示或保证，也不承担因为应用程序或者使用产品或电路所产生的任何责任，明确拒绝承担包括但不局限于后果性的或附带性的损害在内的所有责任。 Freescale 的数据表和/或规格中所提供的“典型”参数在不同应用中可能并且确实不同，实际性能会随时间而有所变化。 所有运行参数，包括“经典值”在内，必须经由客户的技术专家对每个客户的应用程序进行验证。 Freescale 未转让与其专利权及其他权利相关的许可。 Freescale 销售产品时遵循以下网址中包含的标准销售条款和条件：freescale.com/SalesTermsandConditions.

Document Number AN4655
Revision 0, 01/2013