Freescale Semiconductor
Application Note

Document Number:AN4655
Rev. 0, 01/2013

IIC Bootloader Design on the Kinetis L Series

by: Wang Peng

Contents

1 Overview

Many applications or products need to upgrade firmware in field to fix some bugs found or sometimes to improve performance. Most of them do not use the dedicated debug interface, but only use the communication interfaces,such as UART, USB, I2C, and so on. In this case, a serial bootloader is required to upgrade firmware upgrade via one of the communication interfaces without debugger or dedicated program tools.

This application note guides how to design bootloader on Kinetis L series with IIC interface.

2 Introduction

Bootloader is a built-in firmware which is implemented to program the application code to flash via the communication interface. This application note will introduce how to use the Kinetis L series Freedom Development platform, FRDM-KL05Z board to convert UART data from PC terminal to the IIC bus, and communicate with the target board (L series board) to implement update of the target application code. See the following figure.

Figure 1. Top level view

The bootloader takes advantage of AN2295SW_REV1 software tools, available on **freescale.com,** which is widely used in all Kinetis products to implement bootloader to update the application code through the UART interface.

The convert board uses the freedom board FRDM-KL05Z to convert UART bus to IIC bus, and repackage data transfer to the target board.

The target board has built-in bootloader code, which acts as IIC slave device to communicate with the convert board. After receiving the command and data, it will upgrade the application on the target board.

The sample code AN4655SW associated with this application note (available on **freescale.com**) can directly run on the FRDM-KL05 board, and the “bootloader” shall be downloaded to the target board, “UartToIIC” to convert board, and project “demo_bootloader” is for generating S19 file, which can be downloaded using PC software.

3 Software architecture

The software tool attached with this application note, AN4655SW.zip (containing win_hc08sprg.exe) available on **freescale.com**, decodes S19 file and communicates with the convert board through FC protocol.

3.1 Convert board

The convert board communicates with PC terminal through the FC protocol. For detail information regarding the FC protocol, see AN2295: Developer's Serial Bootloader for M68HC08 and HCS08 MCUs, available on **freescale.com**.

Convert board will be initialized to IIC master and communicates with the target board, in order to receive or transmit data package with the target board using IIC bus; it repackages data frame with data length and checksum. Below is the format of the data package.

Data length	Original data frame	Check Sum

After that, it reads data from slave; the first data received is to determine whether the slave is ready. If it is, (command |0x80), then it indicates to the receiver that the correct acknowledge is received, for example:

When the command sent to slave is 0x03, the received acknowledge must be 0x03|0x80.

At first, it will send FC_CMD_HOOK(0x02) to target board, and then reads status from the target board to check if it works in bootloader mode or user code mode.

If the received state is FC_CMD_HOOK|0x80, then it will send 0xFC to start to hook with PC terminal, otherwise, it will always check state of the target board till it receives FC_CMD_HOOK|0x80.

The following figure shows the software flow.

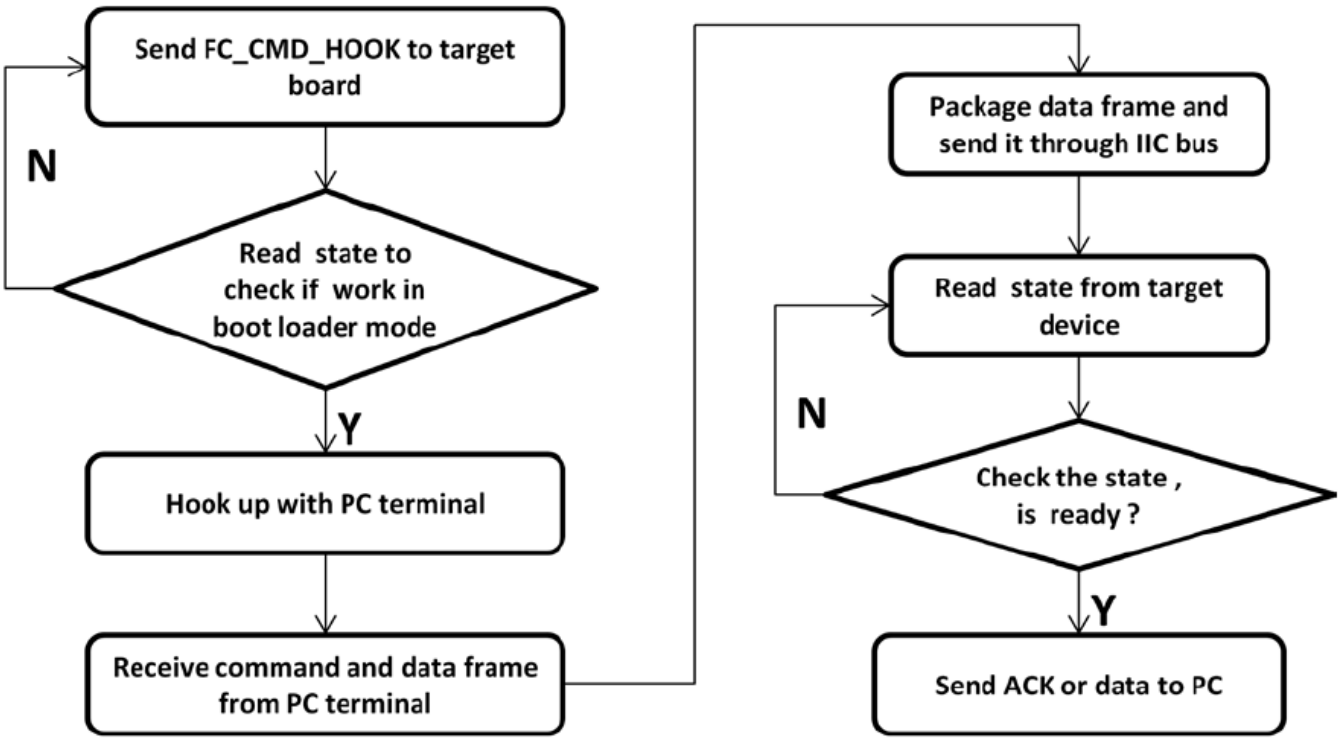

Figure 2. Convert board software flow chart

Convert board functions as a bridge between PC terminal and target board, with which a S19 file can be downloaded to the target board from PC.

3.2 Target board

The target board contains built-in bootloader code. After power up, it first checks the work mode to know whether it is in boot mode or user code mode. There are some ways to do such check, for example, checking the level of an external GPIO.

- If the GPIO pin is low, then it will enter into boot mode to run the bootloader.
- If the GPIO pin is high, then it will enter user code mode to run the application code.

But for some applications, there are limited pin/wires available and no extra GPIO for such purpose. For such cases, the following sections introduce a way to determine work mode through a flag in flash which can be modified in the application code.

3.2.1 Flag in flash

It is known that programming flash results in changing a bit from 1 to 0; if the bit needs to be changed to 1, the flash must be erased first. For KL05Z series chip, one sector is 1KB. So, one sector can be used for saving parameter, and in order to reduce the erase times, and increase the flash life, make an algorithm to implement read-modify-write, after the entire sector space is written, and then erase one time flash sector.

In this application, the flag only uses one byte; so the total number of flag write times is as below:

Total write times = 1024 * 50k

The flag is defined as below:

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

The flow chart for checking the flag to determine the work mode of the target board is given below.

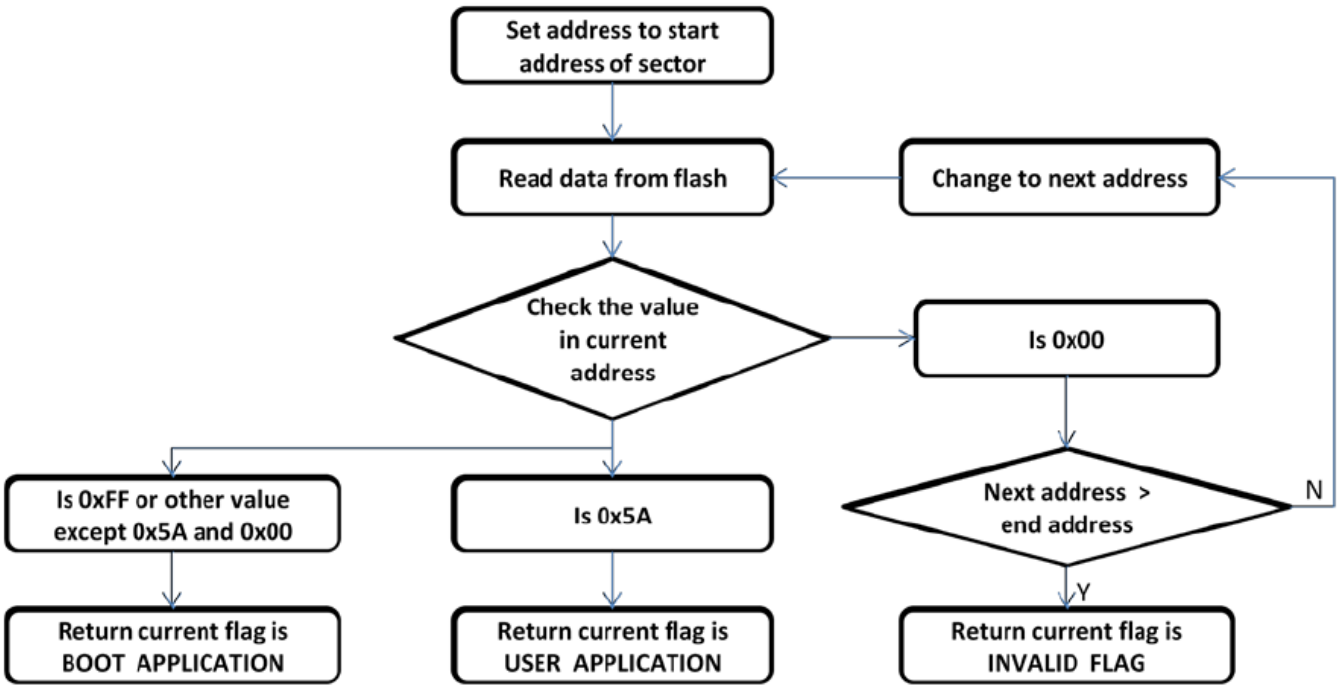

Figure 3. Check flag flow chart

The sample code snippet to check flag status is given below.

```
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;
}
```

This flag can be changed in the application code or bootloader. After successfully updating the code, it will change flag to user mode.

In application code, the user can also change the flag from user mode to bootloader mode, and on the next reset, it will enter the bootloader mode and prepare to update the application code.

The flow chart to modify the flag so as to change the work mode is shown below.

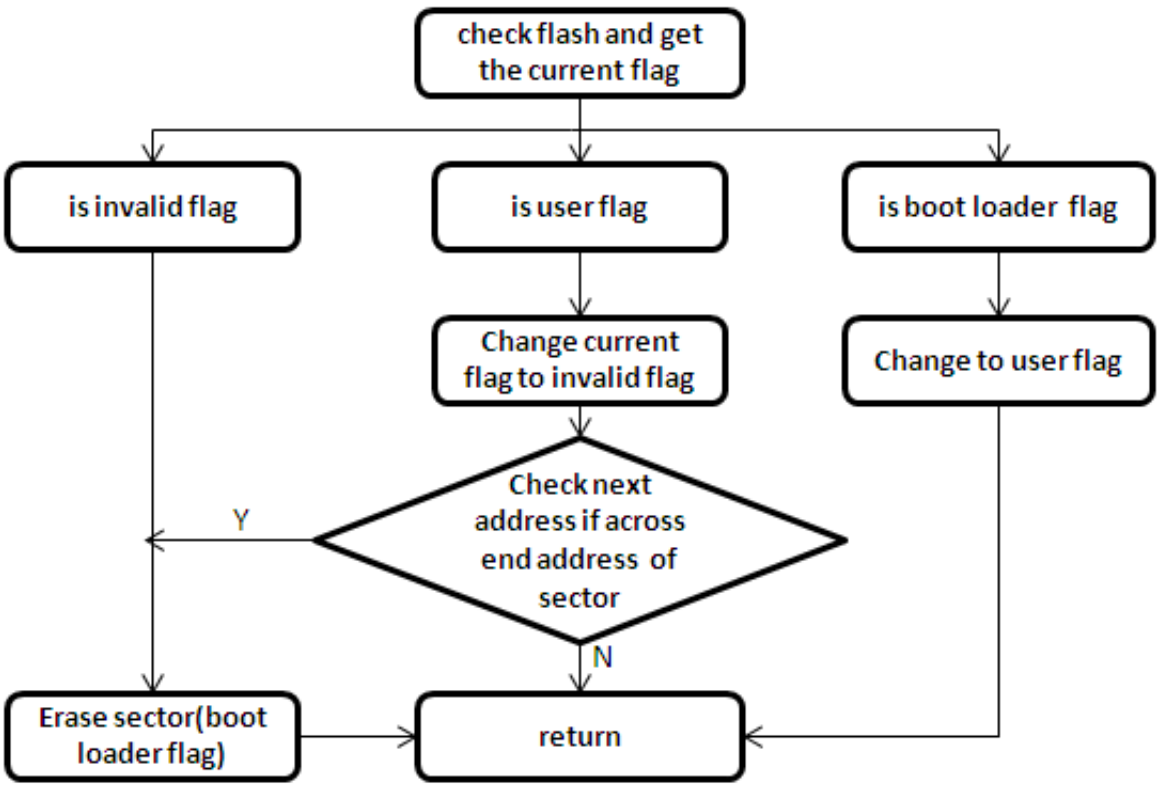

Figure 4. Modify flag flow chart

Sample code snippet to modify flag to other status is shown below:

```
/*****************************************************************************
* 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 slave driver

The target board configures IIC as a slave. It receives and transmits data to the master in IIC interrupt service routine. For detailed interrupt flow, see KL05P48M48SF1RM: KL05 Sub-Family Reference Manual, available on **freescale.com**. Below is a sample code snippet:

```
  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;
                                  }
                          }
                 }
         }
    }
}
```

After IIC receives a data frame, it will set the flag (g_bIICRecFrameFlag) so that the application code can further process data frame.

3.2.3 Command description

In bootloader loop, always check the flag (g_bIICRecFrameFlag). When the flag (g_bIICRecFrameFlag) is 1, it will start to handle the received frame. At first, use check sum to verify if frame received is correct and after the verification, unpackage the frame and handle the appropriate command. Below is the format of the received frame.

Total data length(4 bytes)	Command (1 byte)	Address (4 bytes)	Number of data (1 byte)	Data	Check Sum (1 byte)

A brief summary of commands is given in the following table.

Command Function	Command	Loader positive acknowledge	Loader negative acknowledge
Hook up	0x02	0x82, 0xFC	0x82, 0x03
Ident	0x49	0xC9, ident information	0xC9, 0x03
Erase sector	0x45	0xC5, 0xFC	0xC5, 0x03
Write	0x57	0xD7, 0xFC	0xD7, 0x03
Read	0x52	0xD2, data	0xD2, 0x03

- **Hook up Command**

  The received data package of Hook up command (coded as 0x02) is as given below.

Total data length (4 bytes)	Command (1 byte)	Address(4 bytes)	Number of data (1 byte)	Data	Check Sum (1 byte)
6	0x02	-	-	-	CS

  The Command acknowledge is given below.

Command (1 byte)	Data
0x82	0xFC/0x03

  - If the status received is 0xFC, it indicates that the target board is working in bootloader mode, and gets ready to communicate with the convert board.
  - If the status received is 0x03, it indicate that it is in the user mode, and can't receive other command.

- **Ident command**

  The received data package of Ident command (coded as 0x49), is shown in the following table.

Total data length (4 bytes)	Command (1 byte)	Address (4 bytes)	Number of data (1 byte)	data	Check Sum (1 byte)
6	0x49	-	-	-	CS

  The required MCU information is given below.

  - Protocol version— 1 byte
  - System Device Identification Register (SDID) content ($14A for the K60 Family), r(13-16 bits) is the chip revision number reflecting the current silicon level — 2 bytes
  - Number of reprogrammable memory areas—4 bytes
  - Start address of the reprogrammable area—4 bytes

- End address of reprogrammable memory area—4 bytes
- Address of the original vector table (1KB)—4 bytes
- Address of the new vector table (1KB)—4 bytes
- Length of the MCU erase blocks—4 bytes
- Length of the MCU write blocks—4 bytes
- Identification string, zero terminated—n bytes

One structure body for ident information is shown in the following code snippet.

```
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;
```

Command acknowledge is shown below.

Command (1 byte)	Data
0xC9	Ident information

- **Erase command**

The received data package of the Erase command (coded as 0x45) is shown in the following table.

Total data length (4 bytes)	Command (1 byte)	Address (4 bytes)	Number of data (1 byte)	Data	Check Sum (1 byte)
10	0x45	Address	-	-	CS

The command acknowledge is given below.

Command (1 byte)	Status
0xC5	0xFC/0x03

- **Write command**

The received data package of the Write command (coded as 0x57), is given below.

Total data length (4 bytes)	Command (1 byte)	Address (4 bytes)	Number of data (1 byte)	Data	Check Sum (1 bytes)
Total length	0x57	Address	-	-	CS

The command acknowledge is shown in the following table.

Command (1 byte)	Status
0xD7	0xFC/0x03

- **Read command**

The received data package of the Read command (coded as 0x52), is given below.

Total data length (4 bytes)	Command (1 byte)	Address(4 bytes)	Number of data (1 byte)	Data	Check Sum (1 byte)
11	0x52	Address	Data length to be read	-	CS

Command acknowledge is given below.

Command (1 byte)	Data
0xD2	data

- **Quit command**

  This command does not need any acknowledge.

  After receiving this command, it is required to modify flag and jump to the start address of new interrupt vector table.

4 Memory allocation

The bootloader code occupies the first region of the FLASH memory (the lowest memory address space).This placement moves the beginning of the available memory space and it is necessary to shift this address in the user application linker files (ICF file in IAR and in LCF file in CodeWarrior). An example of the ICF and LCF linker files modification is as follows:

Kinetis L KL05Z

An example of modification ICF file in IAR6.4 is given by the following code snippet.

```
// default linker file

define symbol __ICFEDIT_region_ROM_start__ = 0x00;

// modified Linker file for KL05Z 32k flash

define symbol __ICFEDIT_region_ROM_start__ = 0x1000;
```

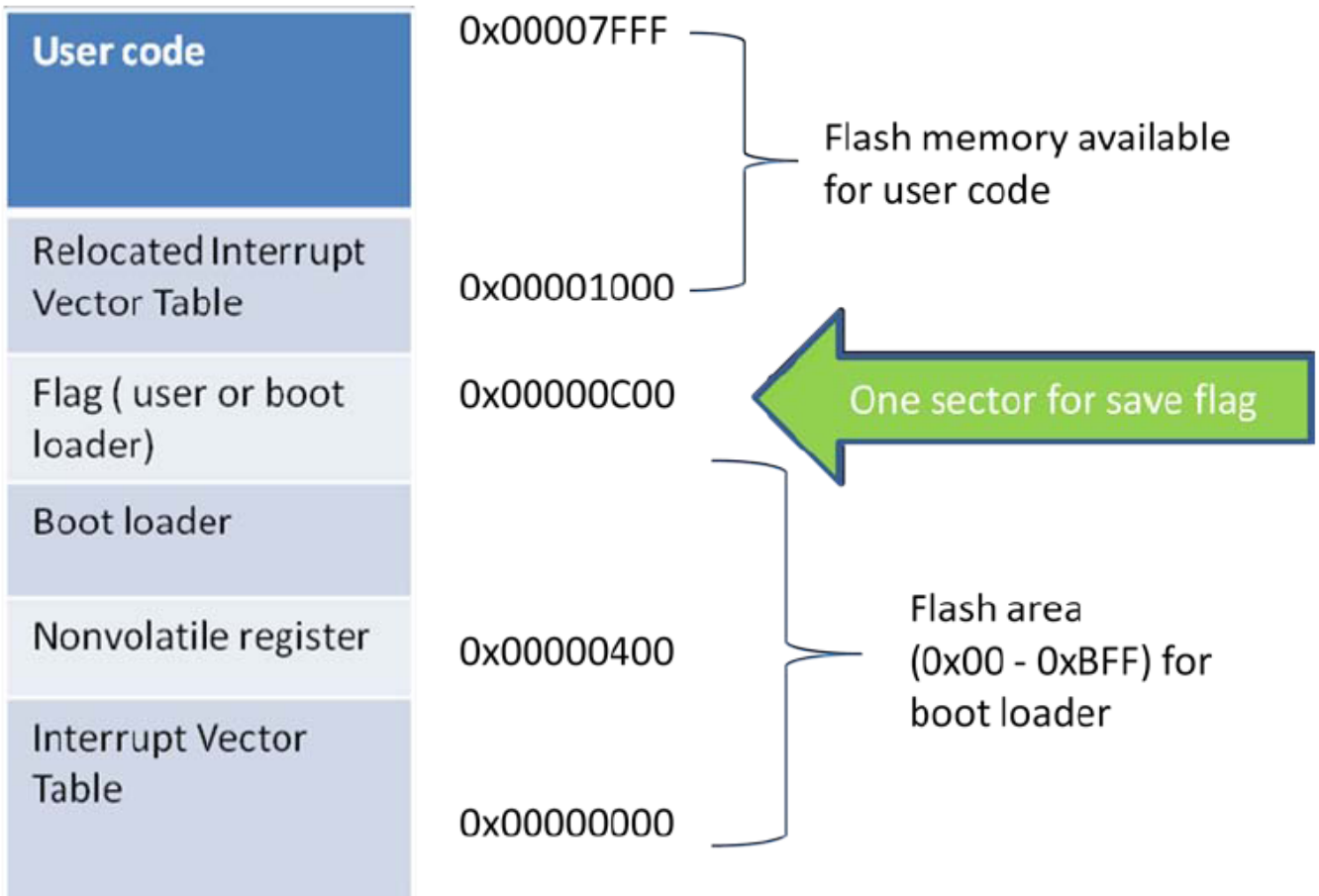

Figure 5. Memory allocation

In the user code, the user can change work mode to bootloader mode so that it can update code again at the next reset. In sample code, whenever a "b" is received from UART, it will change flag to bootloader mode. Following is the sample code snippet.

```
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 Conclusion

This document introduces a way of implementing IIC bootloader by using a bridge board as the convert board, and the other board as target board. The users can also add bootloader by themselves in the application software.

6 References

The following reference documents are available on **freescale.com**

- KL05P48M48SF1RM: KL05 Sub-Family Reference Manual
- AN2295: Developer's Serial Bootloader for M68HC08 and HCS08 MCUs
- KLQRUG, Kinetis L Peripheral Module Quick Reference User Guide

7 Glossary

UART	Universal Asynchronous Receiver/Transmitter
IIC	Inter-Integrated Circuit
FCCOB	Flash Common Command OBject
WDOG	Watchdog
MCG	Multipurpose Clock Generator

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
+1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
www.freescale.com/support

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor China Ltd.
Exchange Building 23F
No. 118 Jianguo Road
Chaoyang District
Beijing 100022
China
+86 10 5879 8000
support.asia@freescale.com

Document Number: AN4655
Rev. 0, 01/2013