Freescale Semiconductor
Application Note

Document Number:AN4775
Rev 0, 07/2013

IIC Boot Loader Design on the Kinetis E Series

by: Wang Peng

Contents

1 Overview

Many applications or products need to upgrade firmware in the 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, IIC, and so on. In this case, a serial boot loader is required to upgrade firmware via one of the communication interfaces without debugger or dedicated program tools.

This application note guides how to design boot loader on Kinetis E series MCUs with IIC interface.

2 Introduction

Boot loader is a built-in firmware which is implemented to program the application code to flash memory via the communication interface.

This application note introduces how to use KE02Z Freedom Development (FRDM-KE02Z) board to convert UART data from PC terminal to the IIC bus, and communicate with the target board (KE02Z board) to implement update of the target application code. See the following figure.

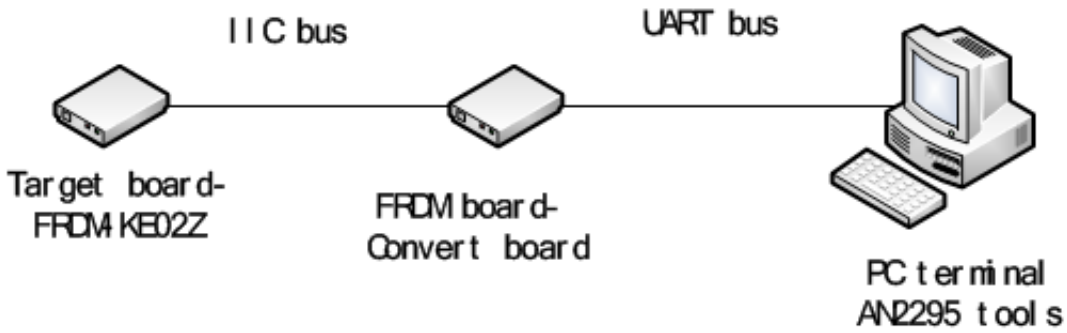

Figure 1. Top level view

The boot loader takes advantage of AN2295SW software tools, available on **freescale.com,** which is widely used in all Kinetis products to implement boot loader to update the application code through the UART interface.

The convert board uses the freedom board FRDM-KE02Z to convert UART bus to IIC bus, and repackage data transfer to the target board.

The target board has built-in boot loader code, which acts as IIC slave device to communicate with the convert board, after receiving command and data, and program the application code to flash memory on the target board.

The software attached with this application note, AN4775SW.zip, contains the following:

- a sample code, which can directly run on the FRDM-KE02Z board
- "I2C_boot loader" which shall be downloaded to the target board
- "Bridge_UartToIIC" which shall be downloaded to the convert board
- the project "RTC_demo" is for generating S19 file, which can be downloaded using PC software.

3 Software architecture

The software tool attached with this application note, AN4775SW.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	Checksum

The following steps explain the flow chart given in the following figure.

1. The convert board sends FC_CMD_HOOK(0x02) to the target board.
2. Then, it reads status from the target board to check if it works in boot loader mode or user code mode.
3. If the received state is FC_CMD_HOOK|0x80, then it will send 0xFC to start hook up with PC terminal, otherwise, it will always check state of the target board till it receives FC_CMD_HOOK|0x80.
4. After this, it receives data frame from the target board, repackages it with data length and checksum and sends it through the IIC bus.

5. 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.

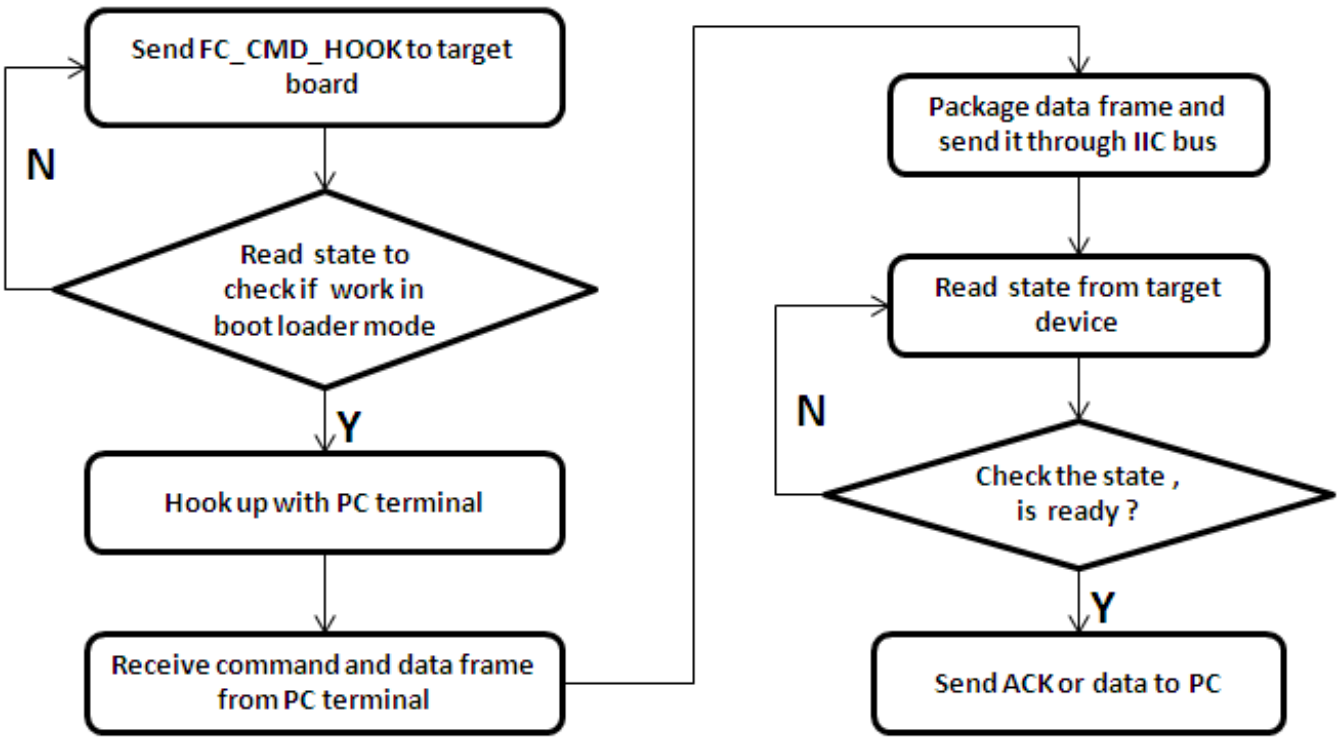

Figure 2. Convert board software flow chart

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 boot loader code. After power up, it first checks the work mode to know whether it is in boot mode or user code mode. One of the methods to identify the work mode is by checking the level of an external GPIO.

- If the GPIO pin is low, then it will enter into boot mode to run the boot loader.
- 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 hook up command is used to determine the work mode.

- If overtime occurs and hook up fails, then the board enters the user mode.
- If hook up succeeds, the board enters the boot loader mode.

This figure presents the flow chart to check the work mode.

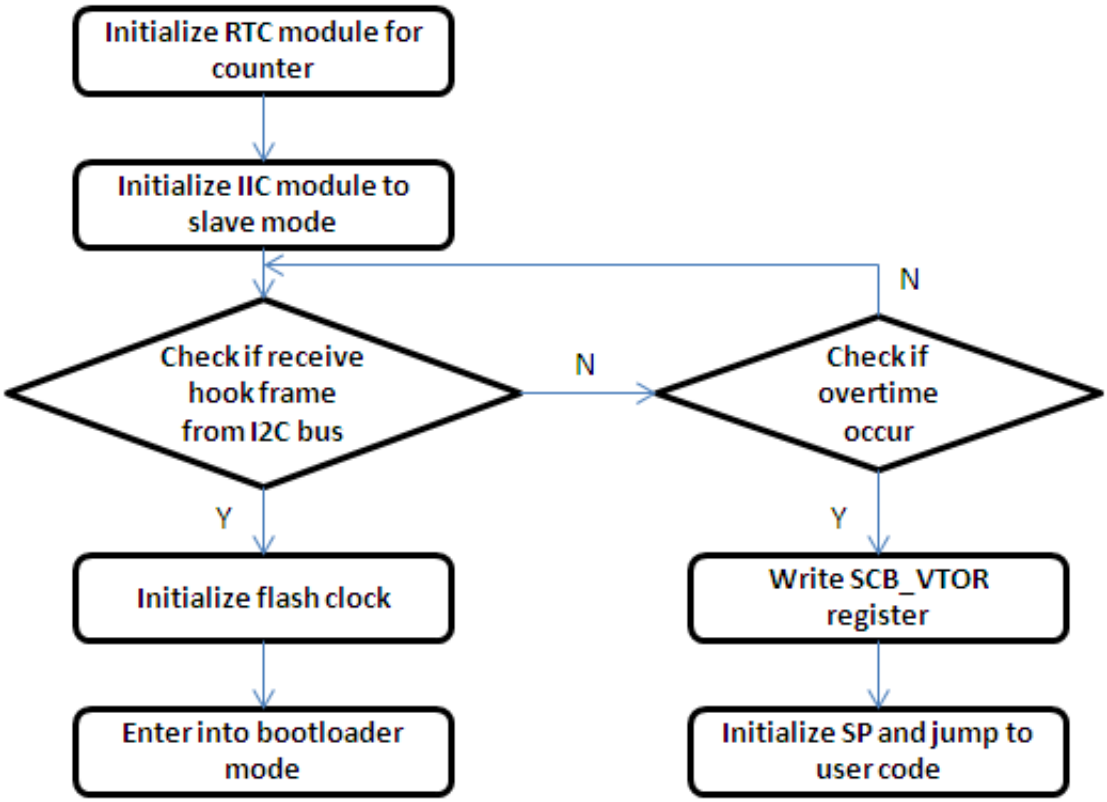

Figure 3. Flow chart to check work mode

3.2.1 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 KE02Z64P20SF0RM, available on **freescale.com**. Below is a sample code snippet:

```
void I2C_SlaveCallback( void )
 {
    I2C_ClearStatus(I2C0,I2C_S_IICIF_MASK);
    if( I2C_GetStatus(I2C0) & I2C_S_ARBL_MASK )
    {
      I2C_ClearStatus(I2C0,I2C_S_ARBL_MASK);
      if( !(I2C_GetStatus(I2C0) & I2C_S_IAAS_MASK) )
      {
         // IIAAS is 0
         return;
       }
     }
     if( I2C_GetStatus(I2C0) & I2C_S_IAAS_MASK )
     {
        I2C_SendAck(I2C0);
        gbI2CRecFrameFlag = 0;
        if( I2C_GetStatus(I2C0)& I2C_S_SRW_MASK )
        {
         // slave send data
         I2C_TxEnable(I2C0);
         u8SendIndex = 0;
         I2C_WriteDataReg(I2C0,u8SendBuff[u8SendIndex++]);
         }
         else
         {
```

```
            I2C_RxEnable(I2C0);
            I2C_ReadDataReg(I2C0);
            u8RecIndex = 0;
        }
    }
    else
    {
      if( I2C0->S & I2C_S_SRW_MASK )
      {
         // if require ACK from master
         if( I2C0->S & I2C_S_RXAK_MASK )
         {
             // no receive the ACK, switch to RX
             I2C_RxEnable(I2C0);
             I2C_ReadDataReg(I2C0);
         }
         else
         {
             if( u8SendIndex < I2C_TX_BUFF_LENGTH )
             {
                 I2C_WriteDataReg(I2C0,u8SendBuff[u8SendIndex++]);
             }
             else
             {
                 /* here do nothing, clock  stretching or send a 0xff to master. */
                 I2C_WriteDataReg( I2C0, 0xff );
             }
           }
         }
         else
         {
             if( u8RecIndex < I2C_RX_BUFF_LENGTH )
             {
              u8RecBuff[u8RecIndex++] = I2C_ReadDataReg(I2C0);
              if( u8RecIndex > sizeof(uint32_t) )
              {
               pRxFrameLength = (uint32_t *)&u8RecBuff[0];
               if( u8RecIndex >= (*pRxFrameLength) )
               {
                 // receive a frame data from master
                 gbI2CRecFrameFlag = 1;
                 Memcpy_Byte((uint8_t *)&gu8I2CRxFrameBuff[0],
                             (uint8_t *)&u8RecBuff[0],u8RecIndex);
                 // reset index counter
                 u8RecIndex = 0;
                 // change MCU state to BUSY
                 u8SendBuff[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.

3.2.2 Command description

In boot loader loop, always check the flag (g_bIICRecFrameFlag). When the flag (g_bIICRecFrameFlag) is 1, it will start to handle the received frame. Initially, use checksum 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	Checksum (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
Quit	0x51	no acknowledge	no acknowledge

- **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	Checksum (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 boot loader 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	Checksum (1 byte)
6	0x49	-	-	-	CS

  The required MCU information is given below.

  - Protocol version—1 byte
  - System Device Identification Register (SDID) content, 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	Checksum (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	Checksum (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	Checksum (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 boot loader code occupies the first region of the flash memory (the lowest memory address space). See the following figure. 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 linker files modification is as follows:

Kinetis E KE02Z

An example of modification of ICF file in IAR6.5 is given by the following code snippet.

```
// default linker file

define symbol __ICFEDIT_region_ROM_start__ = 0x00;

// modified Linker file for KE02Z 64k flash

define symbol __ICFEDIT_region_ROM_start__ = 0x1000;
```

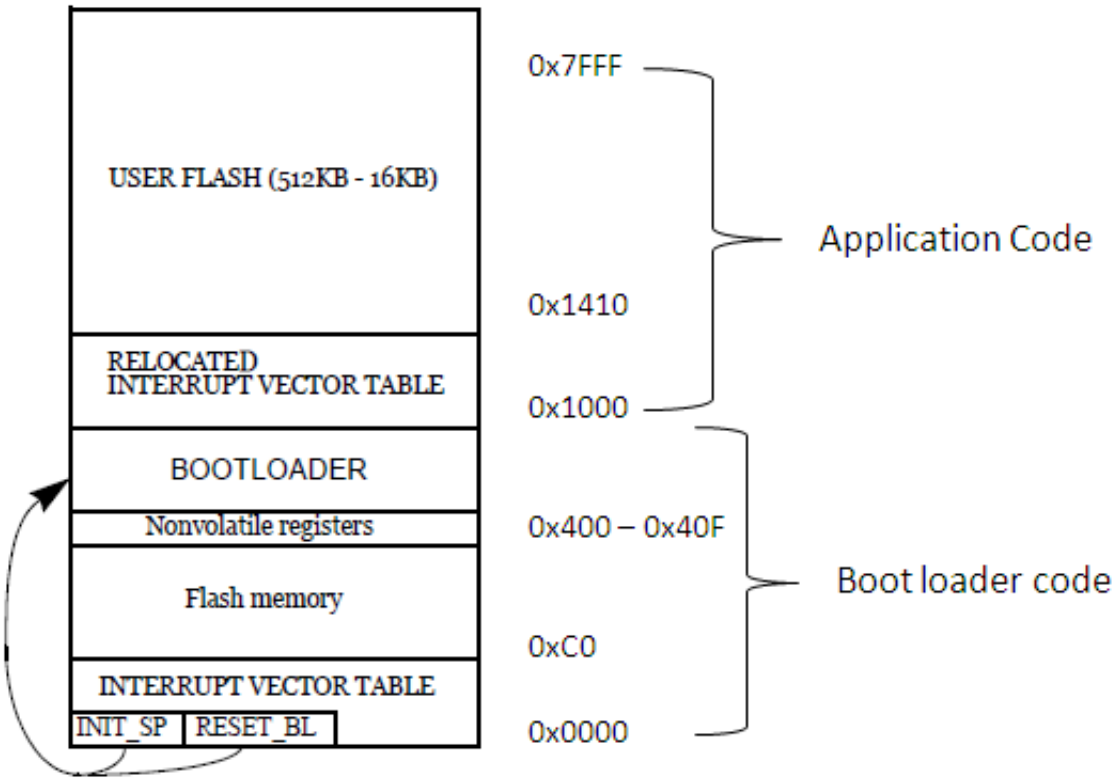

Figure 4. Memory allocation

5 Conclusion

This document introduces a way of implementing IIC boot loader on Kinetis E series MCUs using a bridge board as the convert board, and the other board as target board (KE02Z-FRDM). The users can also add boot loader by themselves in the application software.

6 References

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

- KE02Z64M20SF0RM: KE02 Sub-Family Reference Manual
- AN2295: Developer's Serial Bootloader for M68HC08 and HCS08 MCUs

7 Glossary

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

8 Revision history

Revision number	Date	Substantial changes
0	07/2013	Initial release

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Document Number AN4775
Revision 0, 07/2013