Freescale Semiconductor
Application Note

Document Number: AN4767
Rev. 0, 7/2013

UART Boot Loader Design on the Kinetis E Series

by Wang Peng

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

This document provides guidance on how to design the boot loader on Kinetis E series with a UART interface. The example code listed in this document is developed in IAR 6.50.

Contents

1 Introduction

The boot loader is a built-in firmware implemented to program the application code to on-chip nonvolatile memory (flash on Kinetis E) via the communication interface. This document introduces how to implement the UART boot loader on the FRDM-KE02Z board.

It will use the AN2295 PC side host GUI tools to decode the s19 file and transfer the application code to target MCU by UART interface. Then, it will program the flash.

2 Software architecture

Software tools on the PC, win_hc08sprg.exe, can be downloaded from the freescale website and will decode the S19 file and communicate with the target device. It is compatible with FC protocols. For more detailed information see AN2295, available at www.freescale.com.

The boot loader on built-in flash will begin to run after power up, and will determine whether the hook up will be successful. If a timeout occurs, it will jump to the entry address of the application code.

2.1 Flash operation

KE02 has advanced features for flash operation, the flash memory controller (FMC) and stalling flash control, which not only accelerates access to flash without any wait state, but is able to avoid collision when flash is busy. Here we introduce two ways to program the flash:

1. Code running in RAM programs the flash
2. Code running in flash programs the flash

2.1.1 Code running in RAM

Copy the code of the flash launch command to RAM and disable the interrupt so that the code does not access flash when the flash is busy. This method is most widely used on traditional on-chip flash devices.

A fragment of code to launch the flash command and check flash status is shown here:

```
__ramfunc  void FTMRH_LaunchCMD(uint8_t bWaitComplete)
{
    DisableInterrupts;
    if(bWaitComplete)
      {
      // Wait till command is completed
      FTMRH_FSTAT = 0x80;
      while (!(FTMRH_FSTAT & FTMRH_FSTAT_CCIF_MASK));
      }
      EnableInterrupts;
}
```

2.1.2 Code running in flash

KE02 includes the FMC feature, which is a very important peripheral, providing interconnection between the flash memory and the core. It includes the stalling flash controller, the prefetch buffer, the single entry buffer, and cache memory, so that code is able to program or erase the flash while running in flash. If the stalling flash controller feature is enabled when flash is busy, it can hold the access to flash until the flash is idle. In this case, it is not necessary to disable any interrupt. This makes code more efficient.

A fragment of the code is shown in the following:

```
void FTMRH_LaunchCMD(uint8_t bWaitComplete)
{
    /* enable stalling flash controller when flash is busy */
     MCM_PLACR |= MCM_PLACR_EFSC_MASK;
    FTMRH_FSTAT = 0x80;
    if(bWaitComplete)
```

```
    {
      // Wait till command is completed
      while (!(FTMRH_FSTAT & FTMRH_FSTAT_CCIF_MASK));
    }
}
```

2.2 Interrupt vector table relocation

The other important thing for the boot loader design is how to handle the interrupt vector table. Kinetis E series supports interrupt vector table relocation. By default, the vector table is in 0x00 to 0xBF. The user can change the vector table to any other available address, such as another flash address or RAM.

The boot loader application assigns the boot loader code and application code to different flash spaces. See the following diagram to learn the memory allocation for the boot loader and application code.

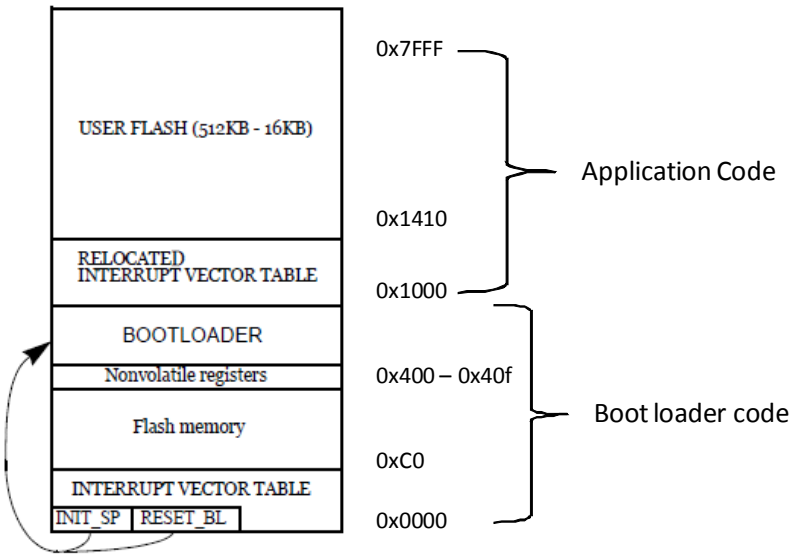

Figure 1. Memory allocation

To relocate the interrupt vector table, simply write the SCB_VTOR register with the following relocation address:

SCB_VTOR = RELOCATION_ADDRESS;

When downloading user code with the boot loader, PC tools will decode the S19 file (application code) and write the contents of the address space (0x00–0x3FF) to the relocation address (0x1000 to 0x13FF). This ensures that after reset, the boot loader will first start to run and jump to the application entry point whenever possible. The following section describes the software flow after reset.

2.3 Software flow chart

After the power is on, the software will first run the boot loader to check whether a hook up will be successful. If overtime occurs, initialize SP and write SCB_VTOR register with the user interrupt vector address. Then, jump to the user code.

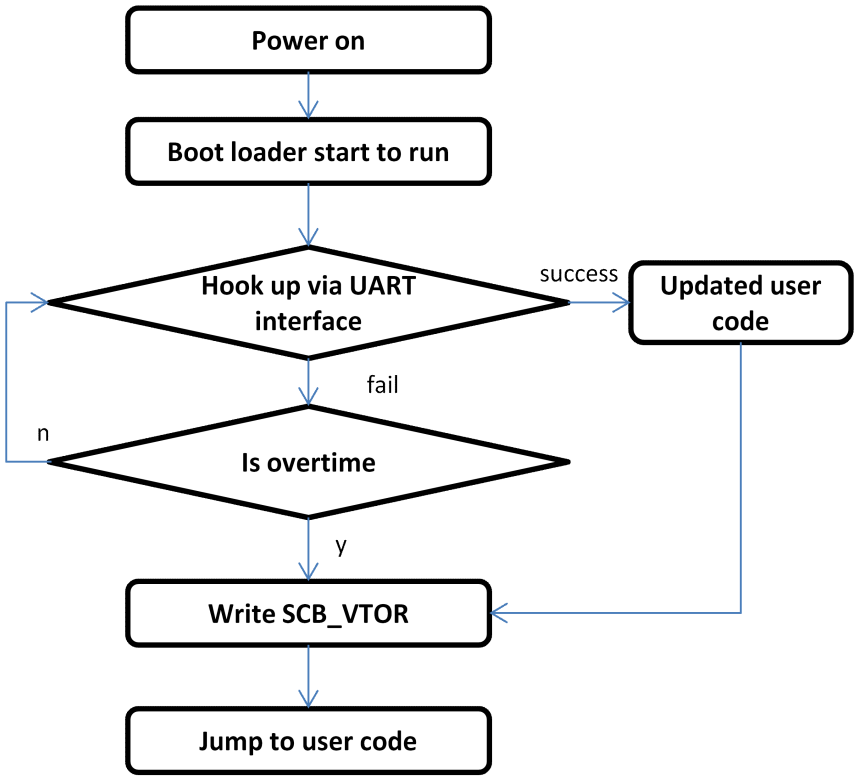

Figure 2. Software flow chart

It is recommended to modify the flash protection region to avoid the boot loader being erased unexpectedly.

2.4 User code

To generate code to be able to downloaded by boot loader, some changes must be made when compared to the normal project.

2.4.1 Link file

The normal project is available to define the code start address to any acceptable address. For example, in IAR:

define symbol __code_start__ = __ICFEDIT_region_ROM_start__ + 0x410;

Here, the defined code start address is offset to 0x410.

For the user code that uses the boot loader to download code, it is necessary to define the code start address to the specified relocation address by file FC_protocol.h, which is in the boot loader project.

Please see the following macro definition from FC_protocol.h.

#define RELOCATION_VERTOR_ADDR 0x1000

The code start address should be RELOCATION_VERTOR_ADDR plus 0x410. The address space from RELOCATION_VERTOR_ADDR to RELOCATION_VERTOR_ADDR + 0x410 for the relocated interrupt vector table.

2.4.2 Interrupt vector table

Before entering into the user code, the boot loader has to write the relocation address to SCB_VTOR register. If the user code needs to copy the vector table into RAM, copy the contents of RELOCATION_VECTOR_ADDR to the target RAM address before writing to SCB_VTOR. No other values should be written to this register.

2.4.3 Flash configuration region

The flash configuration region is located in 0x400 to 0x40F, which is in the boot loader region. If it is protected, then it cannot be modified. Otherwise, it is available for erasing and programming. The user code will first read out all of contents of the sector that contain a flash configuration field at the address 0x400 to 0x40F. Then, erase this sector, modify the buffer, and then write back to this sector.

NOTE

Do not directly define constant variable whose address is from 0x400 to 0x40F in the user code, because it is unavailable to download the code through this boot loader.

3 Conclusion

This document introduces a way of implementing the UART boot loader by using AN2295 PC side host software. It is convenient for the user to update the user code in some special applications without other programming tools.

4 References

KE02 Sub-Family Reference Manual (MKE02Z64M20SF0RM)

Developer's Serial Boot Loader (AN2295)

Kinetis L Peripheral Module Quick Reference User Guide (KLQRUG)

5 Glossary

UART	Universal Asynchronous Receiver/Transmitter
FCCOB	Flash Common Command Object
WDOG	Watchdog
MCG	Multipurpose Clock Generator
FMC	Flash Memory Controller

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Document Number: AN4767
Rev. 0
7/2013