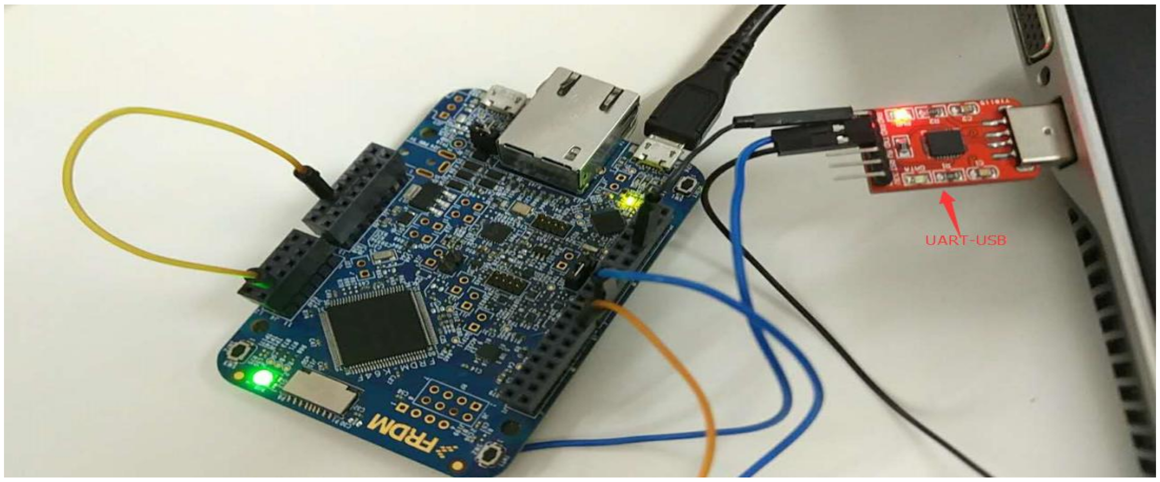
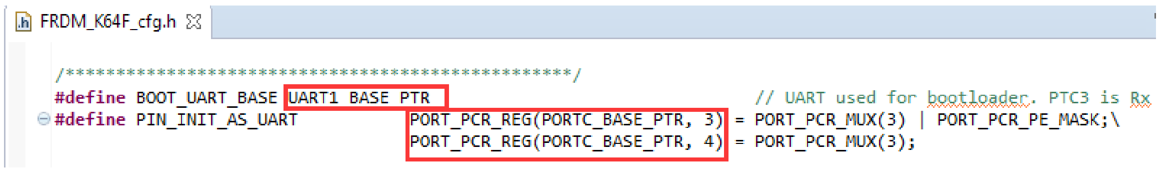
Porting Kboot to FRDM-K64

After we released the "Kinetis Bootloader to Update Multiple Devices in a Network for Cortex-M4" (Kinetis Bootloader to Update Multiple Devices in a Network for Cortex-M4), many customers want to port it to FRDM-K64 board, so here I share it.

I mainly change these points when porting:

1. The OpenSDA on board cannot meet the bootloader transmit speed requirements, so one external USB-UART board is required. In my bootloader project, I select the UART1.

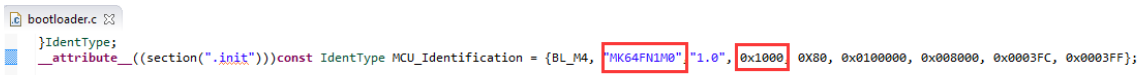




FRDM_K64F_cfg.h

```
/*****************************************************/
#define BOOT_UART_BASE UART1_BASE_PTR                     // UART used for bootloader. PTC3 is Rx
#define PIN_INIT_AS_UART          PORT_PCR_REG(PORTC_BASE_PTR, 3) = PORT_PCR_MUX(3) | PORT_PCR_PE_MASK;\
                                  PORT_PCR_REG(PORTC_BASE_PTR, 4) = PORT_PCR_MUX(3);
```

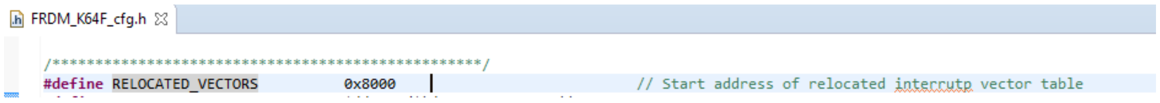
2. The sector size of MK64FN1MOVLL12 is 4Kbytes, so the minimum size of flash erase sector is 4K, I configure the MCU_Identification (About the meaning of MCU_Idendification, please refer to Kinetis Bootloader to Update Multiple Devices in a Network for Cortex-M4) :



bootloader.c

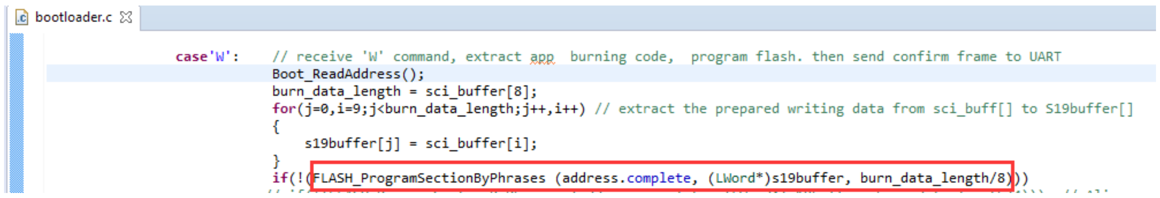
```
}IdentType;
__attribute__((section(".init")))const IdentType MCU_Identification = {BL_M4, "MK64FN1M0","1.0", 0x1000, 0X80, 0x0100000, 0x008000, 0x0003FC, 0x0003FF};
```

3. I also change the RELOCATED_VECTORS :



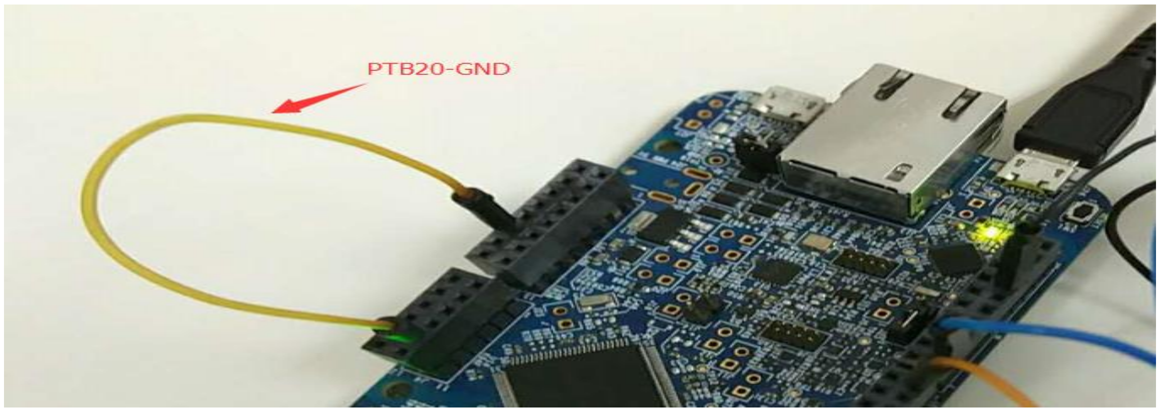
```
/***************************************************/
#define RELOCATED_VECTORS          0x8000                  // Start address of relocated interrutp vector table
```

4. In flash program of of MK64FN1MOVLL12, the size is 8 bytes in a program flash block or a data flash block, so when program, I use the function of FLASH_ProgramSectionByPhrases():



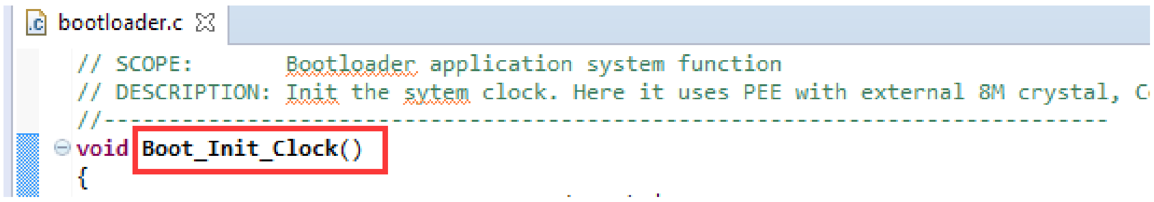
```
        case'W':    // receive 'W' command, extract app  burning code,  program flash. then send confirm frame to UART
                    Boot_ReadAddress();
                    burn_data_length = sci_buffer[8];
                    for(j=0,i=9;j<burn_data_length;j++,i++) // extract the prepared writing data from sci_buff[] to S19buffer[]
                    {
                        s19buffer[j] = sci_buffer[i];
                    }
                    if(!(FLASH_ProgramSectionByPhrases (address.complete, (LWord*)s19buffer, burn_data_length/8)))
```

5. About the “BOOT_PIN_ENABLE_GPIO”, because all buttons on FRDM-K64 are non-maskable interrupt signal, so I choose the PTB20(J4-9 on frdm-k64) as enable boot pin. Connect it to GND(the below picture) when reset, it can run into bootloader mode, if disconnect it, the board will run into user application.



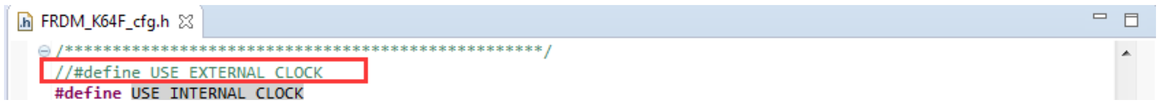
6. In this project, I use the FEI mode, if you need PEE or other modes, please configure it by yourself:

(1)Write the clock configuration code in the function “**Boot_Init_Clock**()”(This function code on my project is the configure for frdm-k22, please do not use it):



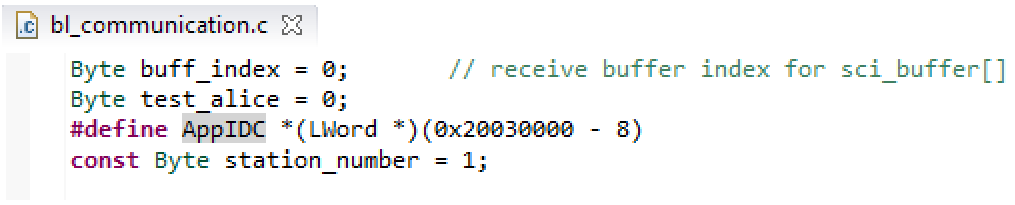
```
// SCOPE:        Bootloader application system function
// DESCRIPTION: Init the sytem clock. Here it uses PEE with external 8M crystal, C
//-----------------------------------------------------------------------------
void Boot_Init_Clock()
{
```

(2)Enable the definition “USE_INTERNAL_CLOCK”:



```
/***************************************************/
//#define USE_EXTERNAL_CLOCK
#define USE_INTERNAL_CLOCK
```

7. In application project, change AppIDC:



```
Byte buff_index = 0;        // receive buffer index for sci_buffer[]
Byte test_alice = 0;
#define AppIDC *(LWord *)(0x20030000 - 8)
const Byte station_number = 1;
```