

# FRDMK64F SD Bootloader

## Contents

1、Introduction.....	1
2、Bootloader's implementation.....	1
3、Memory relocation.....	7
4、Run the demo.....	9
5、Reference .....	9

## 1、 Introduction

As is known to all, we use debugger to download the program or debug the device.

FRDMK64 have the opsenSDA interface on the board, so wo do not need other's debugger. But if we want to design a board without debugger but can download the program, we can use the bootloader. The bootloader is a small program designed to update the program with the interface such as UART,I2C,SPI and so on.

This document will describe a simple bootloader based on the FRDMK64F.The board uses SD card to update the application. User can put the binary file into the card. When the card insert to the board ,the board will update the application automatically. The bootloader code and application code are all provided so that you can test it on your own board.

## 2、 Bootloader's implementation

The schematic for SD card is shown below. The board uses SDHC module to communicate with SD card.

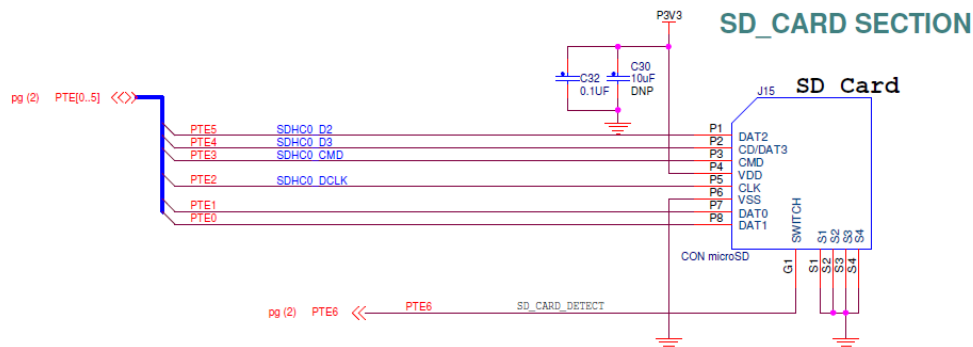
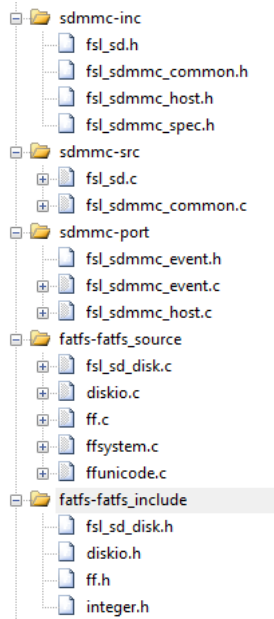


Figure 1.Schematic for SD card

We use the 2.6.0 version of FRDM-K64F's SDK. You can download the SDK in our website. The link is "mcuxpresso.nxp.com".

The bootloader uses SDHC and fatfs file system. So we should add files to support it.



**Figure 2.The support file**

In main code, the program will wait until the card has inserted. Then it will find the file named "a000.bin" in sd card to update the application. If the file do not exist, the board will directly execute the application. If there is no application, the program will end.

The following code shows how the program wait for inserting sd card. It will also check if the address has the application's address.

```
while (sdcardWaitCardInsert() != kStatus_Success)
{
    addrValue = *((uint32_t*) (0xa000));
    if(addrValue != 0xffffffff)
    {
        jumptoApp();
    }
}
```

**Figure 3.The code -- wait for inserting card**

The following code shows how the program opens the binary file. If sd card doesn't have the file, the program will go to the application.

```

error = f_open(&g_fileObject, _T("/a000.bin"), (FA_WRITE | FA_READ));
if (error)
{
    if (error == FR_EXIST)
    {
        return -1;
    }
    else
    {
        addrValue = *((uint32_t*)(0xa000)); //if the 0xa000 has the app address
        if(addrValue != 0xffffffff)
        {
            jumptoApp(0);
        }
        return -1;
    }
}

```

**Figure 4.Open the binary file**

If the program opens the file normally, the update will begin. It will erase 200k's space from 0xa000. You can adjust it according to your project.

Now I will explain update's method in detail. Our data is written to the buffer called "rBuff". The buffer size is 4K. Before write data to it, it is cleared.

Please note that when we erase or program the flash, we should disable all interrupts and when the operations finish we should enable the interrupts.

The file size will decide which way to write the data to flash.

1、 If the size < 4k ,we just read the file's data to buffer and judge if its size aligned with 8 byte. If not , we increase the size of "readSize" to read more data in our data buffer called "rBuffer". The more data we read is just 0.

2、 If the size > 4K, we use "remainSize" to record how much data is left. We read 4k each time until its size is smaller than 4k and then repeat step 1. When finish the operation at a time, we should clear the buffer and increase the sector numer to prepare the next transmission.

```

uint32_t fileSize = g_fileObject.obj.objsize; //get file size
uint32_t readSize;
uint32_t regPrimask = DisableGlobalIRQ();
InitAndErase(&s_flashDriver,&s_cacheDriver,0xa000,4096*50);//erase 200kb
EnableGlobalIRQ(regPrimask);
if(fileSize > 4096U) // file size >4k
{
    uint32_t remainSize = fileSize;
    uint32_t sectorNum = 10U;    //0xa000 sector_10
    while(remainSize > 0U) //the left byte
    {
        if(remainSize >= 4096U) //if the remain size >= 4k,read 4kb ,then program 4kb
        {
            f_read(&g_fileObject,rBuff,4096U,&readSize);
            regPrimask = DisableGlobalIRQ();
            programFlash(&s_flashDriver,&s_cacheDriver,sectorNum*4096U,rBuff,readSize);
            EnableGlobalIRQ(regPrimask);
        }
        if(remainSize < 4096U) //remain size <4k ,judge if align with 8 byte
        {
            f_read(&g_fileObject,rBuff,remainSize,&readSize);
            while((readSize&((uint8_t)0x07))!=0)
            {
                readSize++;
            }
            regPrimask = DisableGlobalIRQ();
            programFlash(&s_flashDriver,&s_cacheDriver,sectorNum*4096U,rBuff,readSize);
            EnableGlobalIRQ(regPrimask);
            break;
        }
        memset(rBuff,0,4096); // clear the buff
        sectorNum++;          //sector num adds
        remainSize -= 4096U;
    }
}
if(fileSize <= 4096U)
{
    f_read(&g_fileObject,rBuff,g_fileObject.obj.objsize,&readSize);
    while((readSize&((uint8_t)0x07))!=0)
    {
        readSize++;
    }
    regPrimask = DisableGlobalIRQ();
    programFlash(&s_flashDriver,&s_cacheDriver,0xa000,rBuff,readSize);
    EnableGlobalIRQ(regPrimask);
}

```

Figure 5.Write flash operation code

The way to clear the space is shown in the figure. It will initialize the flash and erase the given size from the given address. "SectorNum" is used to show which sector to erase.

```
void InitAndErase(    flash_config_t    *my_flash_config,
                    ftfx_cache_config_t *my_flash_cache,
                    uint32_t            destAddrss,
                    uint32_t            size)
{
    assert(size%4096 == 0);
    uint32_t pflashBlockBase = 0;
    uint32_t pflashTotalSize = 0;
    uint32_t pflashSectorSize = 0;
    uint32_t sectorNum = 0;
    ftfx_security_state_t my_security_state = kFTFx_SecurityStateNotSecure;
    status_t result;
    /*init flash*/
    result = FLASH_Init(my_flash_config);
    checkStatus(result);
    /*init cache*/
    result = FTFx_CACHE_Init(my_flash_cache);
    checkStatus(result);

    /*get flash property*/
    FLASH_GetProperty(my_flash_config, kFLASH_PropertyPflash0BlockBaseAddr, &pflashBlockBase);
    FLASH_GetProperty(my_flash_config, kFLASH_PropertyPflash0TotalSize,    &pflashTotalSize);
    FLASH_GetProperty(my_flash_config, kFLASH_PropertyPflash0SectorSize,    &pflashSectorSize);

    /*get security status*/
    result = FLASH_GetSecurityState(my_flash_config, &my_security_state);
    checkStatus(result);

    /*Test flash when flah is unsecure*/
    if(kFTFx_SecurityStateNotSecure == my_security_state)
    {
        FTFx_CACHE_ClearCachePrefetchSpeculation(my_flash_cache, true);
        // PRINTF("\r\n ERASE a sector");
        while(sectorNum < (size/4096))
        {
            result = FLASH_Erase(my_flash_config, destAddrss+sectorNum*4096, pflashSectorSize, kFTFx_ApiEraseKey);
            checkStatus(result);

            /*verify if erase*/
            result = FLASH_VerifyErase(my_flash_config, destAddrss+sectorNum*4096, pflashSectorSize, kFTFx_MarginValueUser);
            checkStatus(result);
            sectorNum++;
        }
        // PRINTF("Erase the sector from 0x%x---0x%x\r\n", destAddrss, destAddrss+pflashSectorSize);
    }
    else
    {
        // PRINTF("Nothing will haboen.as flash is security\r\n");
    }
}
```

**Figure 6.Erase operation code**

The following figure shows how to write the data to flash.

```
void programFlash(    flash_config_t    *my_flash_config,
                    ftfx_cache_config_t *my_flash_cache,
                    uint32_t            destAddrss,
                    uint8_t            *data,
                    uint32_t            dataSize)
{
    status_t result;
    uint32_t failAddr, failDat;
    /*program*/
    result = FLASH_Program(my_flash_config, destAddrss, data, dataSize);
    checkStatus(result);
    /*verify if program*/
    result = FLASH_VerifyProgram(my_flash_config, destAddrss, dataSize, data, kFTFx_MarginValueUser, &failAddr, &failDat);
    checkStatus(result);
    FTFx_CACHE_ClearCachePrefetchSpeculation(my_flash_cache, false);
}
```

**Figure 7.Program operation code**

Before we go to the application, we should modify the configuration we did in the bootloader.

- 1、Close the systick, clear its value.
- 2、Set the VTOR to default value.
- 3、Our bootloader runs in PEE mode. So we should change it to FEI mode.
- 4、Disable the all pins.

You should disable the global interrupt when run these codes. And don't forget to enable the global interrupt.

```
void deinit()
{
    __disable_irq();
    // Disable the timer and interrupts from it
    SysTick->CTRL = SysTick->CTRL & ~(SysTick_CTRL_CLKSOURCE_Msk | SysTick_CTRL_TICKINT_Msk | SysTick_CTRL_ENABLE_Msk);
    // Clear the current value register
    SysTick->VAL = 0;
    SCB->VTOR = 0;
    CLOCK_ExternalModeToFbeModeQuick();
    CLOCK_BootToFeiMode(kMCG_Dmx32Default, kMCG_DrsLow, NULL);
    SIM->SCGC5 = SIM_SCGC5_PORTA_MASK|SIM_SCGC5_PORTB_MASK|SIM_SCGC5_PORTC_MASK|SIM_SCGC5_PORTD_MASK|SIM_SCGC5_PORTE_MASK;
    __enable_irq();
}
```

**Figure 8.Deinitialization code**

Then we can go to the application.

```
void jumptoApp()
{
    SD_Deinit(&g_sd);
    deinit();
    static void (*farewellBootloader)(void) = 0;
    farewellBootloader = (void (*)(void)) (APP_VECTOR_TABLE[1]);
    SCB->VTOR = (uint32_t)APP_VECTOR_TABLE;
    __set_MSP(APP_VECTOR_TABLE[0]);
    __set_PSP(APP_VECTOR_TABLE[0]);
    farewellBootloader();
}
```

**Figure 9.Go to Application**

### 3、Memory relocation

The FRDMK64 has the 1M flash, from 0x00000000 to 0x00100000. As shown in figure 10, we use the 0xa000 as the application's start address.

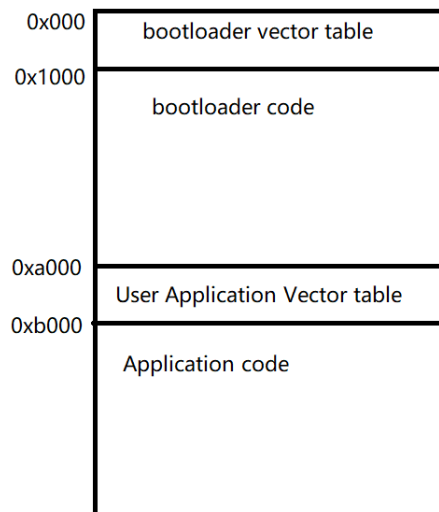


Figure 10. The memory map

Now, I will show you how to modify the link file for user application in different IDE.  
In IAR

```
/*-Memory Regions-*/
define symbol __ICFEDIT_region_FLASH_start__ = 0x0;
define symbol __ICFEDIT_region_FLASH_end__ = __ICFEDIT_region_FLASH_start__ + (1024*1024) - 1;
define symbol __ICFEDIT_region_RAM_end__ = 0x20000000;
define symbol __ICFEDIT_region_RAM_start__ = __ICFEDIT_region_RAM_end__ - (192*1024)/3;
/*-Specials-*/
define symbol __ICFEDIT_intvec_start__ = 0x0000A000; /*-User Application Base-*/

/*-Sizes-*/
define symbol __ICFEDIT_size_cstack__ = (1*1024);
define symbol __ICFEDIT_size_heap__ = (0*1024);

if (isdefinedsymbol(__stack_size__)) {
    define symbol __size_cstack__ = __stack_size__;
} else {
    define symbol __size_cstack__ = 0x0400;
}

if (isdefinedsymbol(__heap_size__)) {
    define symbol __size_heap__ = __heap_size__;
} else {
    define symbol __size_heap__ = 0x0;
}

define symbol __region_RAM2_start__ = 0x20000000;
define symbol __region_RAM2_end__ = __region_RAM2_start__ + (((192*1024)*2)/3) - 1;

define exported symbol __VECTOR_TABLE = __ICFEDIT_region_FLASH_start__;
define exported symbol __VECTOR_RAM = __ICFEDIT_region_RAM_start__;

define exported symbol __BOOT_STACK_ADDRESS = __region_RAM2_end__ - 8;

define memory mem with size = 4G;
define region FLASH_region = mem:[from __ICFEDIT_region_FLASH_start__ to __ICFEDIT_region_FLASH_end__];
define region RAM_region = mem:[from __ICFEDIT_region_RAM_start__ to __ICFEDIT_region_RAM_end__] | mem:[from __region_RAM2_start__ to __region_RAM2_end__];

define block CSTACK with alignment = 8, size = __ICFEDIT_size_cstack__ ( );
define block HEAP with alignment = 8, size = __ICFEDIT_size_heap__ ( );

initialize manually { readwrite };
initialize manually { section .data };
initialize manually { section .textw };
do not initialize { section .noinit };

define block CodeRelocate { section .textw_init };
define block CodeRelocateRam { section .textw };
define block ApplicationFlash { readonly, block CodeRelocate };
define block ApplicationRam { readwrite, block CodeRelocateRam, block CSTACK, block HEAP };
place at address mem: __ICFEDIT_intvec_start__ { readonly section .intvec, readonly section .noinit };
place in FLASH_region { block ApplicationFlash };
place in RAM_region { block ApplicationRam };
```

Figure 11. IAR's ICF

In MDK

```
#define m_interrupts_start          0x0000a000
#define m_interrupts_size          0x00000400

#define m_flash_config_start       0x0000a400
#define m_flash_config_size        0x00000010

#define m_text_start               0x0000a410
#define m_text_size                0x000FFBF0

#define m_interrupts_ram_start      0x1FFF0000
#define m_interrupts_ram_size      __ram_vector_table_size__

#define m_data_start               (m_interrupts_ram_start + m_interrupts_ram_size)
#define m_data_size                (0x00010000 - m_interrupts_ram_size)

#define m_data_2_start             0x20000000
#define m_data_2_size              0x00030000

/* Sizes */
#if (defined(__stack_size__))
    #define Stack_Size              __stack_size__
#else
```

Figure 12.MDK's SCF

In MCUXpresso

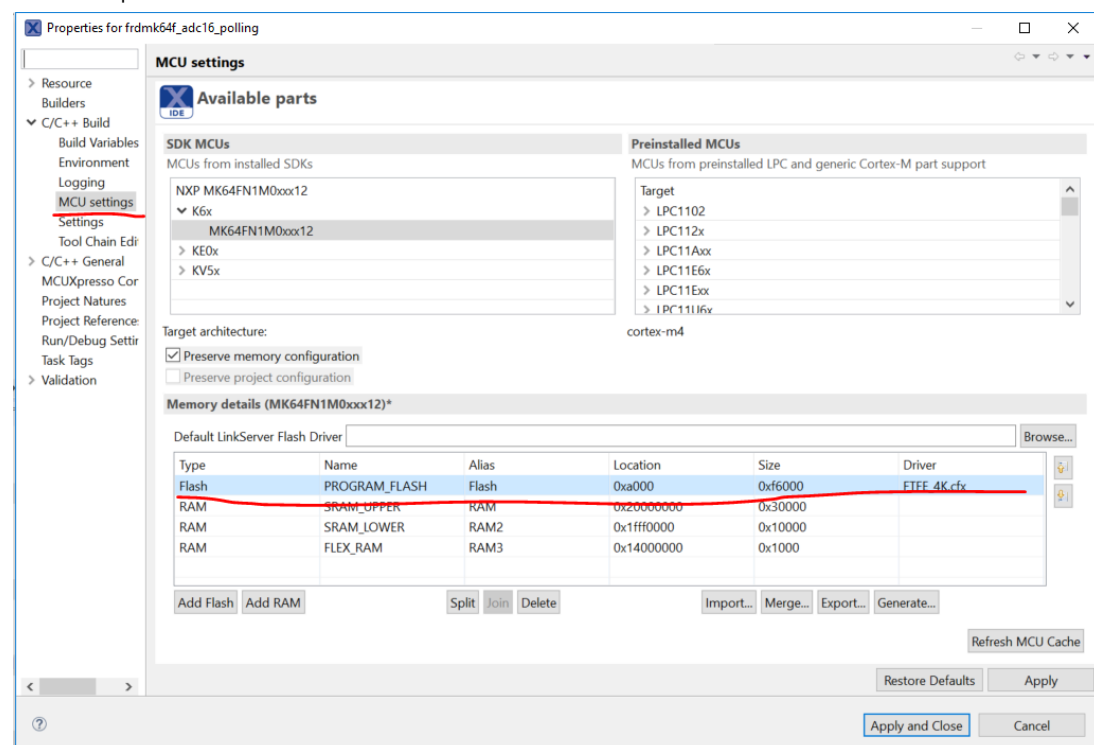
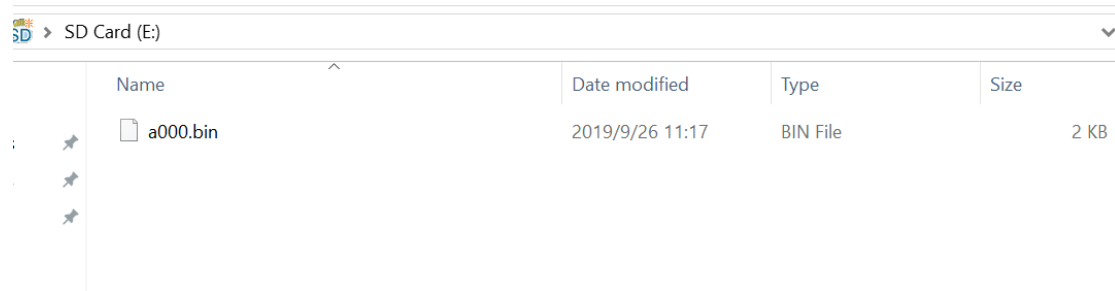


Figure 13.MCUXpresso's flash configuration



## 4、Run the demo

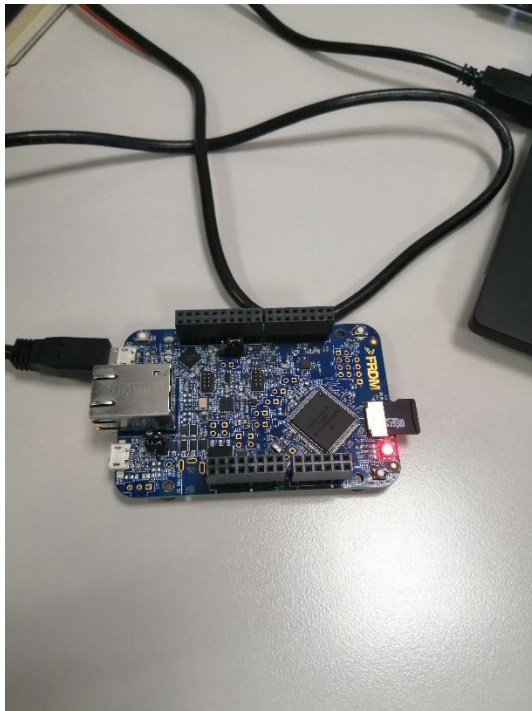
- 1) Download the bootloader first.
- 2) Prepare a user application program. We use the “led blinky” as an example.
- 3) Modify the Link file.
- 4) Generate the binary file with your IDE, please name it as “a000.bin”.
- 5) Put it into the sd card like figure 5.



SD Card (E:)				
	Name	Date modified	Type	Size
	a000.bin	2019/9/26 11:17	BIN File	2 KB

Figure 14.SD card's content

- 6) Insert the card. And power on. Wait for a moment, the application will execute automatically.



## 5、Reference

- 1) Kinetis MCU 的 bootloader 解决方案
- 2) KEA128\_can\_bootloader