FRDMK64F SD 引导程序

URL : https://community.nxp.com/docs/DOC-344903
版本：1 最后更新：11-05-2019 更新：nxf56274

给所有社区用户的重要信息
我们在下载文件时遇到问题，这些文件已以.HTML 格式下载，支持团队正在为此寻求快速解决方案。

您可以将代码目录放在 SDK_2.6.0_FRDM-K64F\boards\frdmk64f 中使用。

1、引言

众所周知，我们使用调试器下载程序或调试设备。

FRDMK64 在板上具有 OpenSDA 调试接口，因此不需要额外的调试器。但是如果我们要设计一个没有调试器但可以下载程序的电路板，则可以使用引导加载程序。 引导加载程序是一个小程序，旨在通过 UART，I2C，SPI 等接口更新程序。

本文档将描述一个基于 FRDMK64F 的简单引导程序。该目标板使用 SD 卡更新应用程序，用户可以将二进制文件放入卡中，卡插入目标板后，板子将自动更新应用程序。本设计提供了对应的引导加载程序和应用程序代码，以便您可以在自己的板上进行测试。

2、Bootloader 的实现

SD 卡的示意图如下所示，该板使用 SDHC 模块与 SD 卡通信。

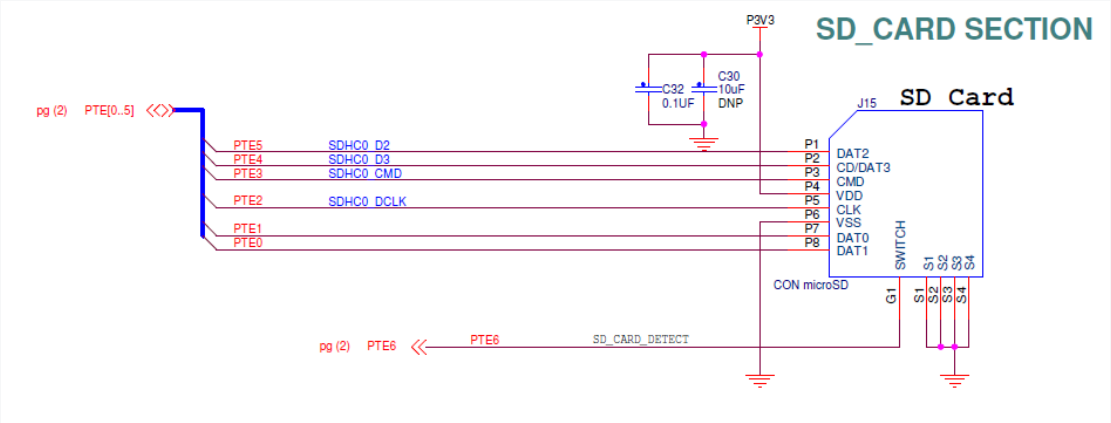

图 1. SD 卡示意图

我们使用 FRDM-K64F 的 2.6.0 版本的 SDK，您可以在我们的网站上下载该 SDK。链接是“mcuxpresso.nxp.com”。

引导加载程序使用 SDHC 和 fafts 文件系统，因此我们应该添加文件来支持它。

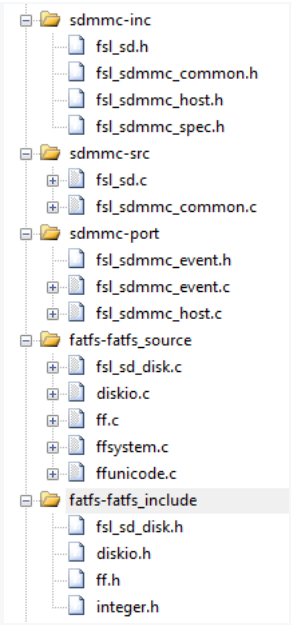

图 2. 支持文件

在主代码中，程序将循环等待直到插入卡，然后它将在 SD 卡中找到名为“ a000.bin”的文件以更新应用程序，如果文件不存在，则开发板将直接执行该应用程序，如果没有应用程序，程序将结束。 以下代码显示了程序如何等待插入 sd 卡，同时它还将检查该地址是否包含应用程序的地址。

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

图 3. 代码-等待插入卡

以下代码显示了程序如何打开二进制文件，如果 sd 卡没有该文件，则程序将跳转到该应用程序开始执行。

```
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();
        }
        return -1;
    }
}
```

图 4. 打开二进制文件

如果程序正常打开文件，则更新将开始，它将从 0xa000 擦除 200k 的空间，您可以根据自己的实际代码工程大小进行调整。

现在我将详细说明更新的方法，我们的数据被写入称为“ rBUff”的缓冲区，缓冲区大小为 4K，在向其中写入数据之前，先将其清除。

请注意，在擦除和编程闪存之前应该先禁用所有中断，当操作完成后再重新使能中断。

文件大小将决定将数据写入闪存的方式。

1.如果大小小于 4k，我们只需读取文件数据进行缓冲，然后判断文件大小是否与 8 个字节对齐。如果不是，我们增加“readSize”的大小以读取称为“rBuffer”的数据缓冲区中的更多数据，这些多读出来的数据内容为 0。

2.如果大小> 4K，我们使用“remainSize”来记录剩余的数据量。每次读取 4k 直到其大小小于 4k，然后重复步骤 1。一次完成操作后，我们应清除缓冲区并增加扇区编号以准备下一次发送。

```
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);
}
```

图 5：写 Flash 操作代码

清除空间的方法如图所示，它将初始化闪存并从给定地址擦除给定大小，“ SectorNum”用于显示要擦除的扇区。

```
void InitAndErase(   flash_config_t        *my_flash_config,
                     ftfx_cache_config_t   *my_flash_cache,
                     uint32_t               destAdrss,
                     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,destAdrss+sectorNum*4096,pflashSectorSize,kFTFx_ApiEraseKey);
        checkStatus(result);

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

图 6. 擦除操作代码

下图显示了如何将数据写入闪存。

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

图 7. 程序操作代码

在转到应用程序之前，我们应该修改在引导加载程序中所做的配置。

- 关闭 Systick 时钟并清掉其计数；
- 将 VTOR 中断向量寄存器恢复为之前的默认值；
- 我们的引导程序以 PEE 模式运行。因此，我们应该将其更改为 FEI 模式；
- 禁用所有引脚。

运行这些代码时，应禁用全局中断，并且不要忘记重新使能全局中断。

```
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();
}
```

图 8. 反初始化代码

然后我们可以转到应用程序。

```
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();
}
```

图 9.转到应用程序

3，内存重定位

FRDMK64 具有 1M 闪存，从 0x00000000 到 0x00100000，如图 10 所示，我们使用 0xa000 作为应用程序的起始地址。

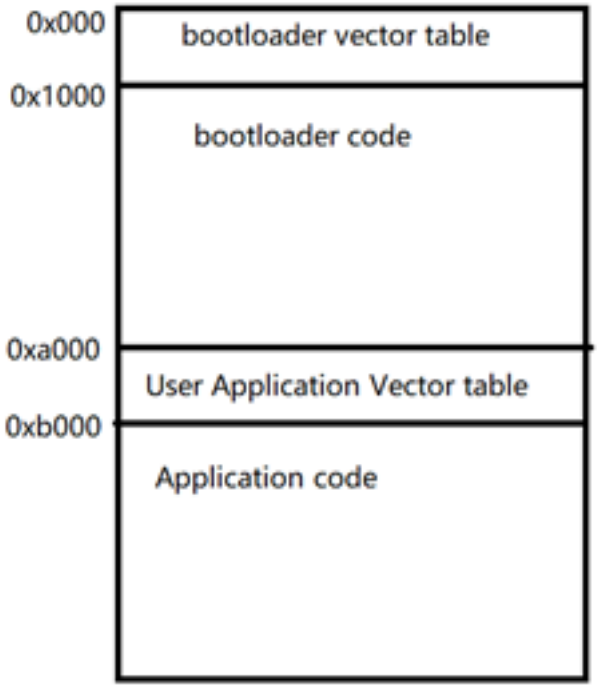

图 10：内存映射

现在，我将向您展示如何在不同的 IDE 中为用户应用程序修改链接文件。

在 IAR 中

```
/*-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 .textrw };
do not initialize  { section .noinit };

define block CodeRelocate { section .textrw_init };
define block CodeRelocateRam { section .textrw };
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 };
```

图 11：IAR 的 ICF

在 MDK 中

```
#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
```

图 12.MDK 的 SCF

在 MCUXpresso 中

```
#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
```

图 13. MCUXpresso 的闪存配置

4，运行演示

1）首先下载引导程序；

2）准备一个用户应用程序。 我们以“led blinky”为例；

3）修改链接文件；

4）用您的 IDE 生成二进制文件，请将其命名为“a000.bin”；

5）将其放入 SD 卡中，如图 5 所示。

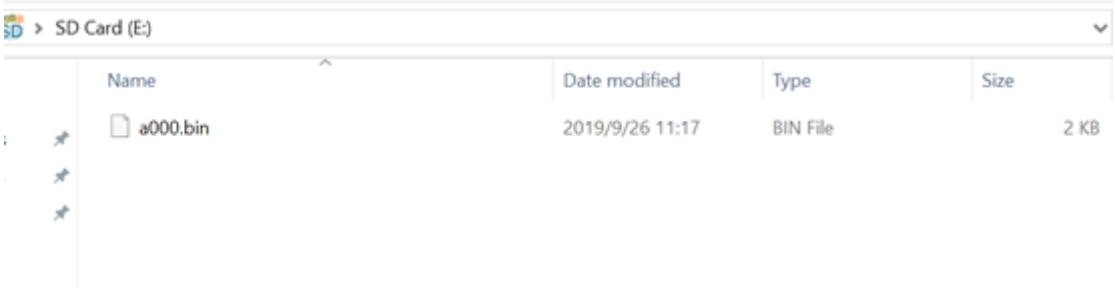

图 14：SD 卡的内容

6）插入卡，并打开电源。请稍等片刻，该应用程序将自动执行。

5.参考

1) Kinetis MCU 的 bootloader 解决方案

2) KEA128_can_bootloader

Labels

- Kinetis K Series MCUs

评论

你好

还有一个用于 FRDM-K64F 的 SD 卡引导加载程序，可在 SDHC 或 SPI 上运行-也可在其他任何 Kinetis 部件上运行-http://www.utasker.com/kinetis/FRDM-K64F.html 它在 Serial Loader 中有完整记录：http

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

//www.utasker.com/docs/uTasker/uTaskerSerialLoader.pdf，并且可以与其他加载方法混合使用。 SD 加载程序可以使用通配符文件匹配和加密（包括 AES256）进行 IP 保护，并在加载完成后清理 SD 卡。 它可以使用 Visual Studio 中的 KDS，MCUXpresso，IAR，Keil，S32，Rowely Crossworks，Green

Hills，Atollic，CooCox，GCC make 的任何 Kinetis 零件进行构建并进行完全仿真（包括 Kinetis 装置和 SD 卡）。 SD 卡加载器的大小为 14k（具有 FAT16，FAT32 和 LFN 支持），因此代码效率更高（应用程序起始地址可以为 0x4000-在演示中节省 32k）。 它还允许通过 USB 主机接口对记忆棒进行相同操作（可与 SD 卡并行使用）：精选视频：
https://www.youtube.com/watch?v=VJ7YveF4_8g&list=PLWKlVb_MqDQFZAulrUywU30v869JBYi9Q&index=26 加载程序大小为 26k（链接地址 0x7000） 它是免费的开放源代码，自 2011 年以来已在许多基于 Kinetis 的工业产品中使用，因此已完全成熟并得到证明。 问候 标记 在 Visual Studio 中运行的模拟 FRDM-K64F SD 卡和记忆棒加载器： pastedImage_3.png 完整的 Kinetis 解决方案，可满足更快/更有效的专业需求，培训和支持：http://www.utasker.com/kinetis.html i.MX RT 项目兼容性：
http://www.utasker.com/iMX.html 包括所有 Kinetis 部件的 FreeRTOS 集成 Kinetis K64： -http://www.utasker.com/kinetis/FRDM-K64F.html -
http://www.utasker.com/kinetis/TWR-K64F120M.html -
http://www.utasker.com/kinetis/TEENSY_3.5.html -
http://www.utasker.com/kinetis/Hexiwear-K64F.html uTasker：支持> 1 000 名 Kinetis 注册用户，使产品更快，更便宜地推向市场 在
http://www.utasker.com/services.html 上请求免费的紧急远程桌面咨询。 开源版本位于 https://github.com/uTasker/uTasker-Kinetis
https://community.nxp.com/thread/512558
https://community.nxp.com/thread/352862
https://community.nxp.com/thread/498809