手把手教你写 STM32 的 bootloader

(SDIO 读取 TF 卡更新固件)

作者：谭建裕

1、bootloader 的简介及作用

什么是 bootloader？本人不知道该怎么说，反正会来看这篇都是知道自己要干嘛的。不过 bootloader 的作用还是要提提的，bootloader 最直观的作用就方便，比如你用单片机给人家做了一款产品，后期你的产品固件需要更新的时候，你总不能带着电脑直接去客户那里拆开产品给单片机下程序吧？也不能教客户怎么给单片机下程序吧？用户体验感太差。

其实本质上 bootloader 的也是一个完整的程序，也有 main 函数，有自己的中断向量表，栈顶指针，它可以检查有没有新的固件，如果有，则将新的固件的数据写入到我们指定的 flash 地址中，之后跳到新的程序中去就 OK 了。此时 bootloader 的优势就来了，bootloade 更新固件有很多种方式，本人在这里只详细讲解一种，搞懂一种之后，其它的都好办，因为它们的思路都是一样的。Bootloader 的主体原理是：首先将 bin 文件的数据复制到特定的地址。然后设置中断向量表，设置 MSP 主堆栈指针（具体请看 CM3 权威指南），设置复位向量。然后就没有然后了。

2、bootloader 涉及的知识

本人在此讲解的是 STM32 通过读取 TF 内的 bin 文件数据来更新固件。这里牵扯到 STM32 的 SDIO 外设，FATFS 文件系统，STM32 的 flash 读写操作。

2.1 SDIO

SDIO 是 STM32 的外设，需要注意的是只有 100 引脚及以上的才有。电路原理图如图 2-1-1 所示。

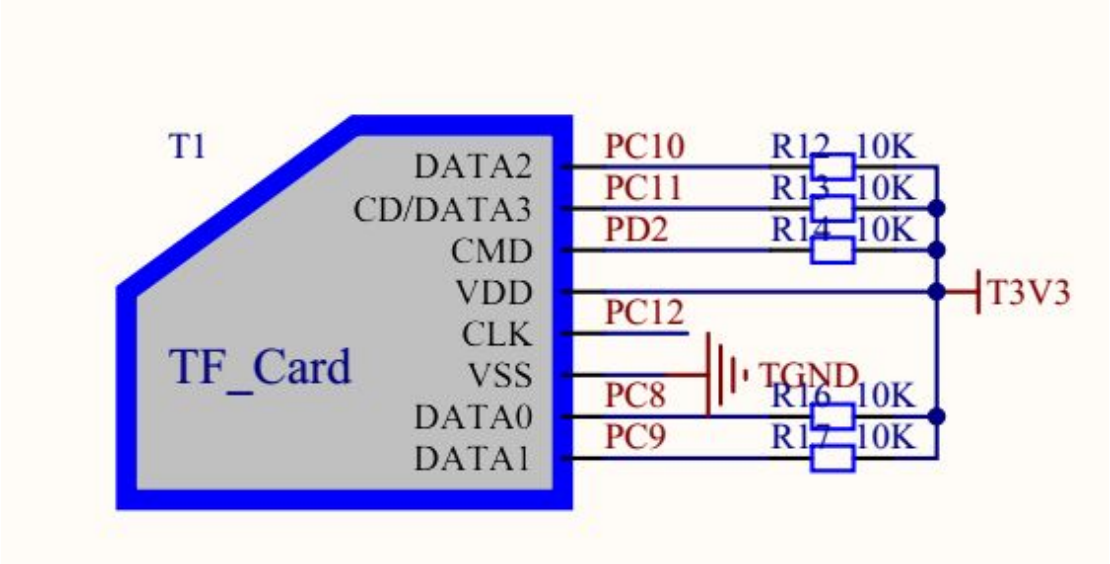


图 2-1-1

注意：在使用 TF 之前必须保证 TF 卡格式为 FAT32,单元大小为 2048。如图 2-1-2 所示。

图 2-1-2

记得在 stm32f10x_it.c 文件中添加中断函数。如图 2-1-3 所示。

stm32f10x_it.c

```
140  /*  Add here the Interrupt Handler for the used periphe
141  /*  available peripheral interrupt handler's name pleas
142  /*  file (startup_stm32f10x_xx.s).
143  /*****************************************************
144 /*
145   * 函数名: SDIO_IRQHandler
146   * 描述   : 在SDIO_ITConfig() 这个函数开启了sdio中断 ,
147         数据传输结束时产生中断
148   * 输入   : 无
149   * 输出   : 无
150   */
151  #include "sdio_sdcard.h"
152  void SDIO_IRQHandler(void)
153 {
154    /* Process All SDIO Interrupt Sources */
155    SD_ProcessIRQSrc();
156 }
```

图 2-1-3

2.2 FATFS 文件系统移植和使用

文件系统使用的是 FATFS9，源码在压缩包的 ff9 文件夹，如图 2-2-1 所示。

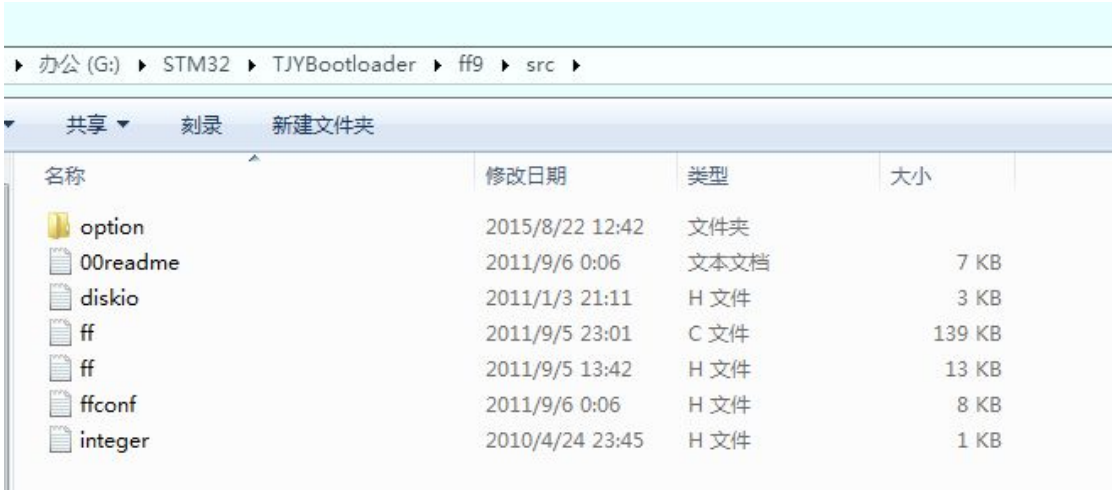
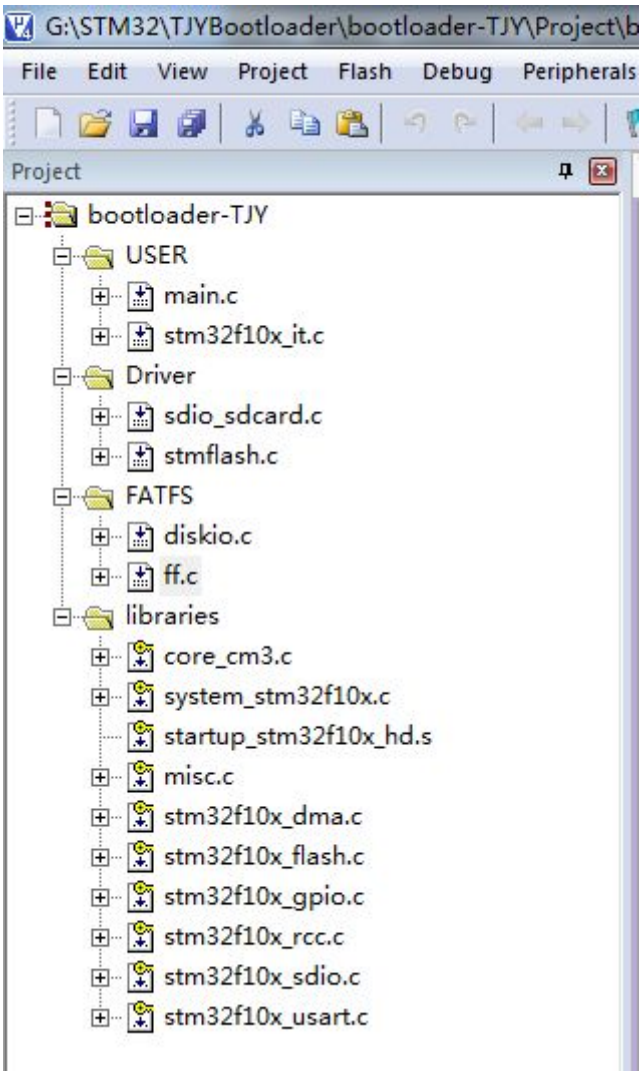


图 2-2-1

从 bootloader 工程框图（图 2-2-2）可以看出需要添加进工程的只有 ff.c 和 disio.c。其实 disio.c 是需要我们自己编写。



FATFS 文件系统给我们提供很多库函数,我们主要只用到了一下：

FIL fnew;//定义一个文件结构体

FATFS fs;//定义一个工作区

FRESULT res; //状态标志

UINT br, bw; //读取和写入字节计数变量

f_mount(0,&fs);//在驱动器 0 中开启一个工作区 fs

res = f_open(&fnew, "0:APP.bin", FA_OPEN_EXISTING | FA_READ);//只读（FA_READ ）形式打开存在文件（FA_OPEN_EXISTING），路径为根目录下的 APP.bin 文件，文件属性存在 fnew 中。

f_lseek(&fnew, x);//跳转指令，可以跳转到 fnew 文件中第 x 个字节。

f_read(&fnew, data, 2, &br);//从 fnew 指向的文件中读取 2 个字节存在 data 中。

f_close(&fnew);//关闭文件

f_mount(0,0);//关闭工作区

2.3 STM32 的 flash 读写操作

在对 STM32 的 flash 进行写操作之前必须要先擦除要写入地址所在页的数据，而且解锁 flash。在写入时，尽管每个地址只能存放 8 位，每次写入是数据至少是 16 位，所以写 flash 时，每次是连续写入两个地址。从这里我们可以看出，以后写数据时，尽量写的地址为偶数。

用到的库函数有：

FLASH_Status FLASH_ErasePage(uint32_t Page_Address);

FLASH_Status FLASH_ProgramHalfWord(uint32_t Address, uint16_t Data);

void FLASH_Unlock(void);

void FLASH_Lock(void);

本人利用这几个库函数，封装了一下

```c
#include "stmflash.h"
#define START_ADDRESS 0x08010000
//根据文件大小来擦除flash空间
void FLASH_EraseALL(unsigned long datasize)
{
  uint8_t i;
  uint8_t page = datasize/2048 + 1;
  FLASH_Unlock();
  for(i=0; i<page; i++)
  {
    FLASH_ErasePage(START_ADDRESS+(i*2048));
  }
  FLASH_Lock();
}
//往flash中指定地址写16位数据
void Flash_WriteU16(uint32_t addr, uint16_t data)
{
  FLASH_Unlock();
  FLASH_ProgramHalfWord(addr, data);
  FLASH_Lock();
}
//往flash中指定地址连续写入num个16位数据
void FLASH_WriteBuffer(uint32_t addr, uint16_t *data, uint16_t num)
{
  uint16_t i;
  FLASH_Unlock();
  for(i=0; i<num; i++)
    FLASH_ProgramHalfWord(addr+(i*2), *(data++));
  FLASH_Lock();
}
//读取addr地址中的16位数据
uint16_t FLASH_ReadU16(uint32_t addr)
{
  return *(vu16 *)addr;
}
//读取addr的连续num个16位数据
void FLASH_ReadBuffer(uint32_t addr, uint16_t *data, uint16_t num)
{
  uint16_t i;
  for(i=0; i<num; i++)
  {
    *(data+i) = FLASH_ReadU16(addr + (i*2));
  }
}
```

3、booloader 的工作流程

```
#include "stm32f10x.h"
#include "sdio_sdcard.h"
#include "ff.h"
#include "stmflash.h"
#define  FLASH_APP_ADDR   0x08010000    //定义新程序所占flash空间的起始地址
FIL fnew;                              /* file objects */
FATFS fs;                              /* Work area (file system object) for logical drives */
FRESULT res;
UINT br, bw;                           /* File R/W count */

void BootLoader_Jump(uint32_t Sect, uint32_t Msp, uint32_t Reset);

int main()
{
  u32  i = 0;
  uint8_t data[2];
  uint16_t Buf[4];
  uint32_t msp;
  uint32_t reset;
  /* Sdio Interrupt Config */
  NVIC_Configuration();
  f_mount(0,&fs);
  res = f_open(&fnew, "0:APP.bin", FA_OPEN_EXISTING | FA_READ );//只读BIN文件
  if ( res == FR_OK )//文件打开成功
  {
    FLASH_EraseALL(fnew.fsize/2048 + 2);//擦除所有数据
    f_lseek(&fnew, 0);
    br=0;
    for(i=0; i<fnew.fsize; i++)
    {
      f_read(&fnew, data, 2, &br);
      Flash_WriteU16(FLASH_APP_ADDR + (i*2), data[0] + (data[1]<<8));
    }
    f_lseek(&fnew, 0);
    br=0;
    for(i=0; i<20; i++)
    {
      f_read(&fnew, data, 2, &br);
    }
    f_close(&fnew);
    f_mount(0,0);
  }
  /*直接从FLASH中读取*/
  FLASH_ReadBuffer(FLASH_APP_ADDR, Buf, 4);
  msp = Buf[0] + (Buf[1]<<16);//新程序的堆栈栈顶指针
  reset = Buf[2] + (Buf[3]<<16);//获得复位向量
  BootLoader_Jump(FLASH_APP_ADDR, msp, reset);//跳到新程序中执行
  while(1);
}
```

```
void BootLoader_Jump(uint32_t Sect, uint32_t Msp, uint32_t Reset)
{
  uint32_t base;
  uint32_t offset;

  base = (Sect > NVIC_VectTab_FLASH) ? NVIC_VectTab_FLASH : NVIC_VectTab_RAM;
  offset = Sect - base;

  NVIC_SetVectorTable(base, offset);//设置总断向量表

  __set_MSP(Msp); //设置主堆栈指针
  ( (void (*)())(Reset))();//跳到指定flash地址
}
```

3.1 程序流程

1）读取出 BIN 文件的数据，并写到指定的地址中。

2）设置中断向量表、主堆栈指针和复位向量（具体为什么要设置这个可以去看 CM3 权威指南）。

3.2 BIN

BIN 只是二进制文件，不含有地址信息，纯粹的程序文件。HEX 文件是带有地址信息的，在烧写 hex 文件时，需要一边转化一些写入。BIN 文件的开头的前 32 位是主堆栈指针，接着 32 位是复位向量指针。具体如何生成 BIN 文件，在 4 节。

3.3 查看 bootloader 所占内存大小

双击图 3-3-1 中的 bootloader-TJY，便会弹出 map 文件，找到如图 3-3-2 中的内容，可以看出 bootloader 所占的 flash 地址是从 0x0800000-0x08004c48。我们在放置新的程序时，就不能放在这段区域内，否则会出问题。

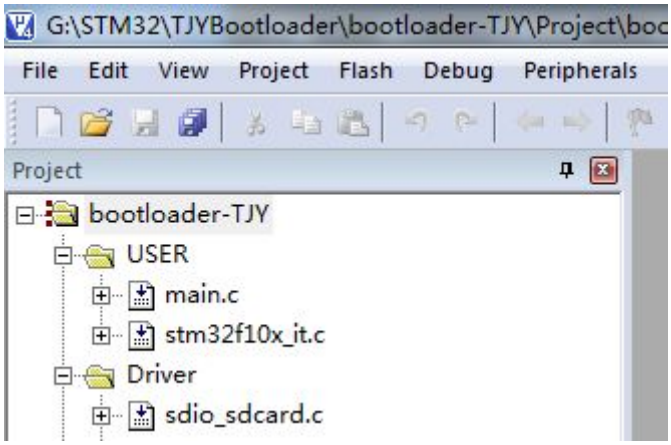


图 3-3-1

```
Memory Map of the image

  Image Entry point : 0x08000131

  Load Region LR_IROM1 (Base: 0x08000000, Size: 0x00004c48, Max: 0x00040000, ABSOLUTE)

    Execution Region ER_IROM1 (Base: 0x08000000, Size: 0x00004bf8, Max: 0x00040000, ABSOLUTE)

    Base Addr    Size         Type   Attr      Idx    E Section Name        Object
```

图 3-3-2

4、APP 程序的编写

Bootloader 写好之后，我们如何写新的程序呢？其实很简单，就和你平时写程序一样，唯一不同是，你要修改两个地方：

第一：将 IROM1 修改成如图 4-1 所示的数据。原因在 3.3 小节讲解了，为了不让新程序覆盖 bootloader 而导致问题。本人的芯片是 VCT6，flash 地址是 0x0800 000 - 0x0804 0000。Bootloader 末地址在 0x08004c48，给够 bootloader 的余量，所以我们新程序 start:0x0801 0000 ,size:0x3 0000

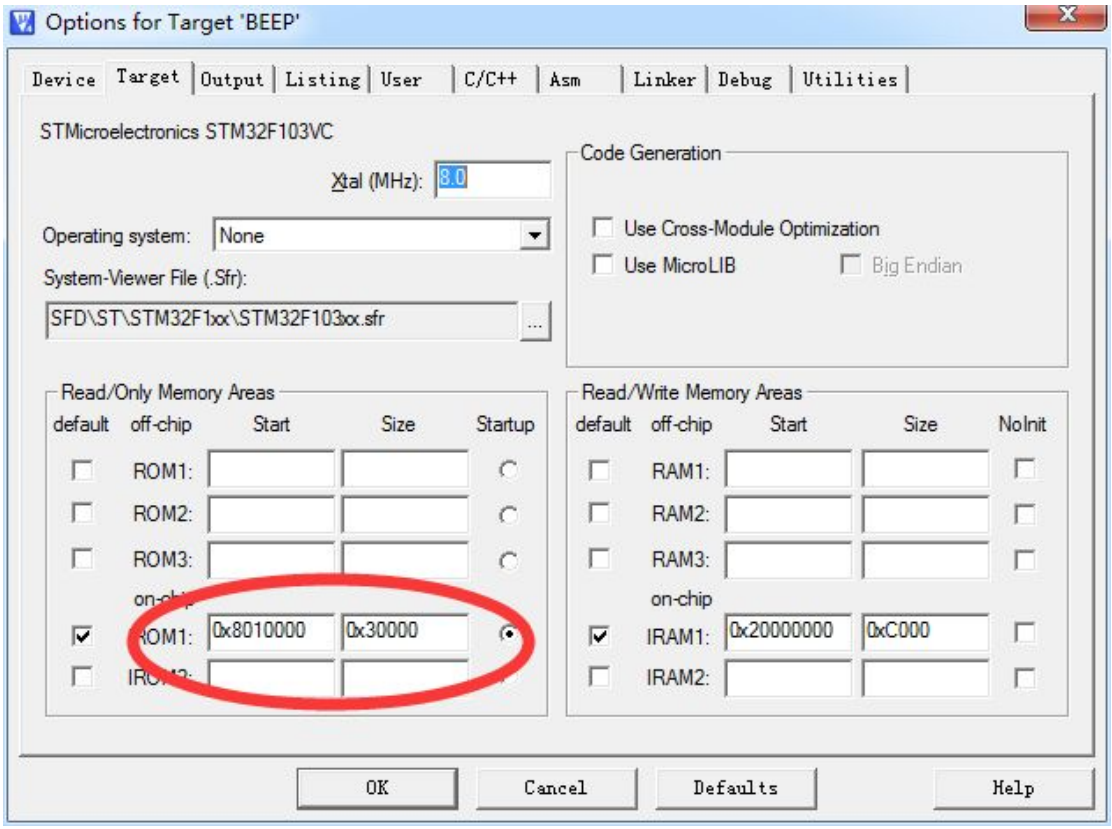


图 4-1

第二：勾选 Run #1，在里面填入 D:\MDK\ARM\ARMCC\bin\fromelf.exe --bin -o..\OUTPUT\APP.bin ..\OUTPUT\APP.axf

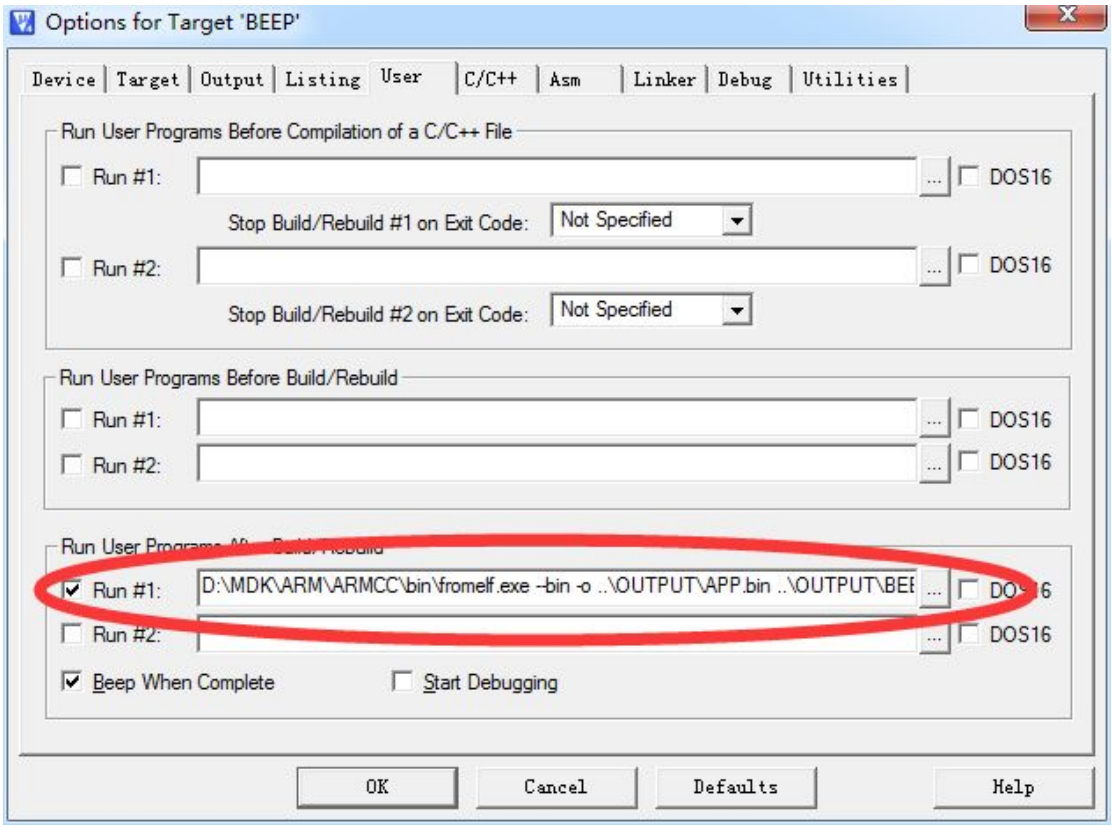


图 4-2

只有这样，编译器才会生成 BIN 文件。D:\MDK\ARM\ARMCC\bin 是 fromelf.exe 的路径，在安装 MDK 的安装文件夹下，每个人的不同，自己可以搜索一下， --bin -o 生成 BIN 的命令，了解一点 gcc 的人会好理解点。..\OUTPUT\是存放 BIN 的路径。其中..是工程的同级文件夹（工程存在 Project 中，BIN 放在 Output 文件夹中，它们是同级的），和.有区别，大家注意一下。APP.axf 是工程名字，生成的 HEX 文件就是这个名字，不要弄错了。

BIN 的名字可以随便改，但是 bootloader 程序中也要做出相应的修改，否则 bootloader 找不到新程序的。

最后将 BIN 放到 TF 中就可以愉快地升级啦！我的邮箱是 641533882@qq.com。有什么问题，欢迎交流。


机 ▸ 办公 (G:) ▸ STM32 ▸ TJYBootloader ▸ bootloader-TJY ▸
刻录
新建文件夹
名称
修改日期
类型
大小
Driver 2015/8/22 13:39 文件夹
ff9 2015/8/22 11:23 文件夹
Listing 2015/8/22 11:51 文件夹
Output 2015/8/22 13:36 文件夹
Project 2015/8/22 14:39 文件夹
stm32_Libraries 2015/8/22 11:23 文件夹
USER 2015/8/22 14:07 文件夹