青风手把手教你学 stm32f030 系列教程

---------- 库函数操作版本

出品论坛： www.qfv8.com 青风电子社区

作者： 青风
出品论坛: www.qfv8.com
淘宝店：http://qfv5.taobao.com
QQ 技术群：241364123
硬件平台：QF-STM32F030 开发板

2.15 文件系统操作 SD 卡

2.15.1 原理分析

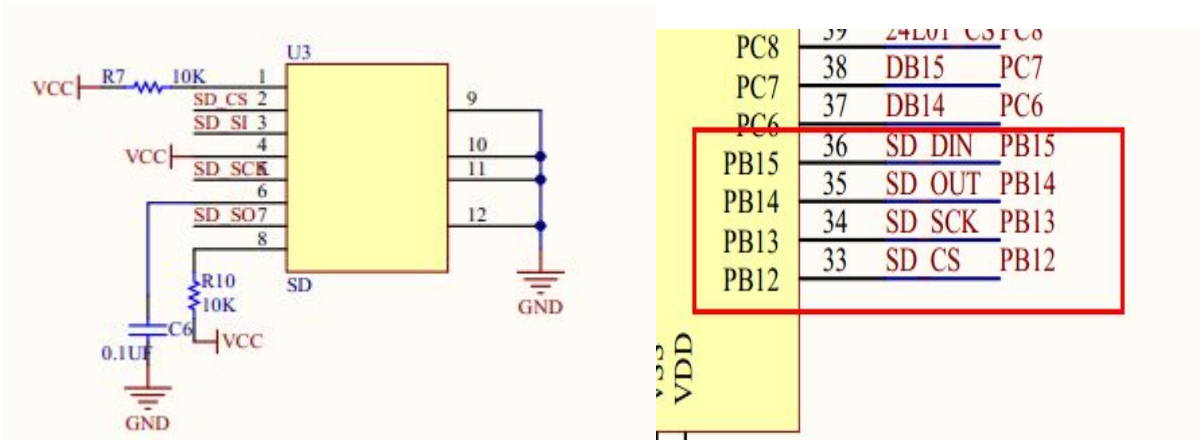
很多单片机系统都需要大容量存储设备，以存储数据。目前常用的有 U 盘，FLASH 芯片，SD 卡等。他们各有优点，综合比较，最适合单片机系统的莫过于 SD 卡了，它不仅容量可以做到很大（32Gb 以上），而且支持 SPI 接口，方便移动，有几种体积的尺寸可供选择（标准的 SD 卡尺寸，以及 TF 卡尺寸），能满足不同应用的要求。只需要 4 个 IO 口，就可以外扩一个最大达 32GB 以上的外部存储器，容量选择尺度很大，更换也很方便，而且方便移动，编程也比较简单，是单片机大容量外部存储器的首选。

FATFS 是一个专为小型嵌入式系统调计的通用文件系统模块。FATFS 是用 ANSI C（标准 C 语言）相兼容的语法书写，并且完全地实现了与磁盘 IO 层分离，因此它是与硬件无关的。它完全不需要任何修改就能被集成到低成本的微控器(MCU)中，与 WINDOWS FAT 相兼容的文件系统。与平台无关，便于移植。对代码存储器的大小和运行时所需 RAM 的大小及其它硬件性能要求很低，所以 FATFS 可以很好地运用在低成本的嵌入式系统中。

移植的版本为：FatFs_vR0.08a，本节是在上一节直接操作 SD 卡的基础上进行的，下面就从软硬件两个方面入手详细分析如何移植 FATFA 文件系统：

2.15.2 硬件准备：

如下图所示：



端口配置：

开发板中采用了较为小的 msd 卡作为存储卡，这是由于 msd 卡的使用较为广泛，有取代 SD 卡的趋势。通信方式采用 SPI 接口方式，连接如下：

硬件连接：

PB15-----SD_CS sd 卡片选管脚

PB14------SD_OUT sd 卡 MISO 端口

PB13------SD_DIN sd 卡 MOSI 端口

PB12------SD_SCK sd 卡 sck 管脚

2.15.3 软件准备：

下面我们来看看 FATFS 的移植过程，首先要介绍下 FATFS 的基本结构：我们下载 FATFS_VR0.08 版本如下所示：

这个文件包里包含了这个几个文件：

00readme.txt :这个文件主要是对其他的几个文件功能进行说明，以及整个 FATFS 的版本信息。这个文件可有可无。

下面 4 个函数是主要的使用函数：

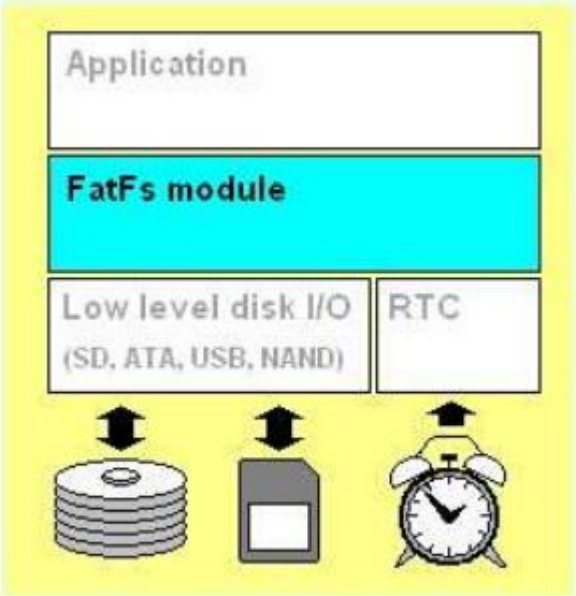
diskio.c:底层磁盘的操作函数，这些函数需要用户自己实现

ff.c 独立于底层介质操作文件的函数，完全由 ANSI C 编写

integer.h 一些数值类型的定义

ffconf.h fatfs 调用的基本配置

我们看看其文件结构图：

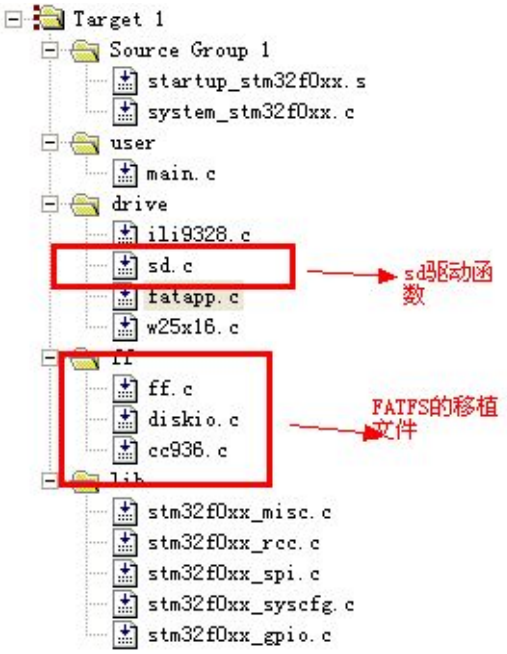


这个文件系统是分层而成的，对于用户层，我们只需要修改接口文件就可以直接和硬件相结合，

如下图所示：

我们需要修改的只有 diskio.c 和 ffconf.h 函数，这两个函数和硬件对接就 OK 了，而文件夹 option 内提供了一些文字函数，比如说 CC936 支持汉字。

我们把上面文件夹内容移植入 KEIL 建立工程树如下，我们需要加入工程树的文件为 ff.c , diskio.c 这两个函数：



我们主要需要编写 diskio.c 函数，在原来文件加入 SD 卡的操作函数，如下面蓝色所示：

```
01.  DSTATUS disk_initialize (
02.      BYTE drv                    /* Physical drive nmuber (0..) */
03.  )
04.  {
05.   SD_Error res = SD_RESPONSE_FAILURE;
06.   res =   SD_Init(); //用户直接初始化 SD
07.   return ((DSTATUS)res);
08.
```

```
09.  DSTATUS disk_status (
10.      BYTE drv          /* Physical drive nmuber (0) */
11.  )
12.  {
13.      if (drv) return STA_NOINIT;      /* Supports only single drive */
14.      return 0;                 //返回磁盘状态
15.  }
```

```
16.  DRESULT disk_read (
17.      BYTE drv,          /* Physical drive nmuber (0..) */
18.      BYTE *buff,        /* Data buffer to store read data */
19.      DWORD sector,              /* Sector address (LBA) */
20.      BYTE count         /* Number of sectors to read (1..255) */
```

```
)
{
    SD_ReadBlock(buff, sector << 9, 512);//读扇区
        return RES_OK;
}

#if _READONLY == 0
DRESULT disk_write (
    BYTE drv,                /* Physical drive nmuber (0..) */
    const BYTE *buff,  /* Data to be written */
    DWORD sector,            /* Sector address (LBA) */
    BYTE count          /* Number of sectors to write (1..255) */
)
{
     SD_WriteBlock((BYTE *)buff, sector << 9,512);//写扇区
    return RES_OK;
}
#endif /* _READONLY */
```

上面加入的函数在 **SD.H** 函数中编写，上一节内容里面已经讲过，所示在函数的头文件里，我们需要加入**#include "sd.h"**，如下所示：

```
#include "diskio.h"
#include "stm32f0xx.h"
#include "ffconf.h"
#include "sd.h"
```

同时我们进入到 **ffconf.h** 文件中，在这个文件中，我们需要加入支持中文字符，修改

```
#define _CODE_PAGE  936
```

936 为汉字码，修改的同时我们在工程中加入 **cc936.c** 函数。

那么这时 **FATFS** 系统文件基本移植完成，那么我们使用的时候就可以不管硬件问题，直接调用 **FATFS** 接口函数：

应用层程序结口如下

f_mount - 注册和取消工作区

f_open - 打开或者创建文件

f_close -关闭一个文件

f_read - 从文件中读取数据

f_write -向 文件中写入数据

f_lseek - 移动文件读指针（RP），扩展文件大小

f_truncate - 缩减文件大小

f_sync - 清空缓存数据，实现数据同步

f_opendir - 打开一个目录

f_readdir - 列举目录中的条目（包括文件和子目录）。

f_getfree - 获取可用簇

f_stat - 获取文件属性

f_mkdir - 创建文件

f_unlink - 删除文件或者目录

f_chmod - 更改属性
f_utime -更改时间戳
f_rename - 重命名或者移动一个文件或目录
f_mkfs - 格式化磁盘
f_forward - 将文件数据直接送入数据流中
f_chdir - 改变当前所在目录
f_chdrive - 改变当前所在驱动器
f_gets -读字符串
f_putc - 写一个字符
f_puts - 写字符串
f_printf - 写入一个格式化字符串
磁盘 I/O 接口
disk_initialize - 磁盘初始化
disk_status - 获取磁盘属性
disk_read - 读扇区
disk_write - 写扇区
disk_ioctl - 独立的磁盘控制资源
get_fattime - 获取时间
这些接口命令在 **ff.c** 函数中被定义了。也就是说这里面用户可以不用改变其中内容，它与硬件层没有关系，是独立的文件系统，也就是说只有移植过来就可以了。这些函数基本上就可以完成 SD 卡的相关操作，可以在主函数里进行使用，如下程序所示：

```
43.         LCD_init();             // 液晶显示器初始化
44.         LCD_Clear(ORANGE);  // 全屏显示白色
45.         POINT_COLOR =BLACK; // 定义笔的颜色为黑色
46.         BACK_COLOR = WHITE    ;       // 定义笔的背景色为白色
47.           /*------------------------- SD Init ---------------------------- */
48.         disk_initialize(0);
49.         LCD_ShowString(20,20,       "mmc/sd  演示");
50.      res = f_mount(0, &fs);
51.         if(res == FR_OK)
52.         LCD_ShowString(20,40,       "mmc/sd  初始化成功  ");
53.    else
54.    LCD_ShowString(20,40,     "mmc/sd  初始化失败");
55.
56.    res=f_open(&fsrc,"12-29.txt",FA_CREATE_ALWAYS | FA_WRITE);
57.    if (res == FR_OK)
58.       LCD_ShowString(20,60,  "文件创建成功  ");
59.    else
60.     LCD_ShowString(20,60,    "文件创建失败");
61.
62.    res = f_write(&fsrc, &w_buffer, countof(w_buffer), &bw);
63.
64.    if (res == FR_OK)
65.     LCD_ShowString(20,80,    "SD 卡写成功  ");
```

```
66.   else
67.     LCD_ShowString(20,80,   "SD 卡写失败");
68.
69.   res=f_close(&fsrc);
70.   if (res == FR_OK)
71.   LCD_ShowString(20,100,   "文本关闭成功  ");
72.   else
73.     LCD_ShowString(20,100, "文本关闭失败");
74.
75.   res=f_open(&fsrc,"12-29.TXT",FA_READ);
76.   if (res == FR_OK)
77.       LCD_ShowString(20,120,     "打开文本成功  ");
78.   else
79.     LCD_ShowString(20,120, "打开文本失败");
80.         res = f_read(&fsrc, &buffer, 1024, &br);
81.         if (res == FR_OK)
82.         {
83.         LCD_ShowString(20,140,   "文件读取成功");
84.               LCD_ShowString(20,160,   buffer);     }
85.         else
86.           LCD_ShowString(20,140, "读文件失败  ");
```

所以这样看来，SD 卡的 FATFS 文件的移植和使用还是很简单的，不过要详细理解文件系统的编写机制还需要读者多多研究了。

2.15.4 实验下载现象：

MINISD 卡插入后的测试结果在触摸屏 TFT 上显示如下所示：