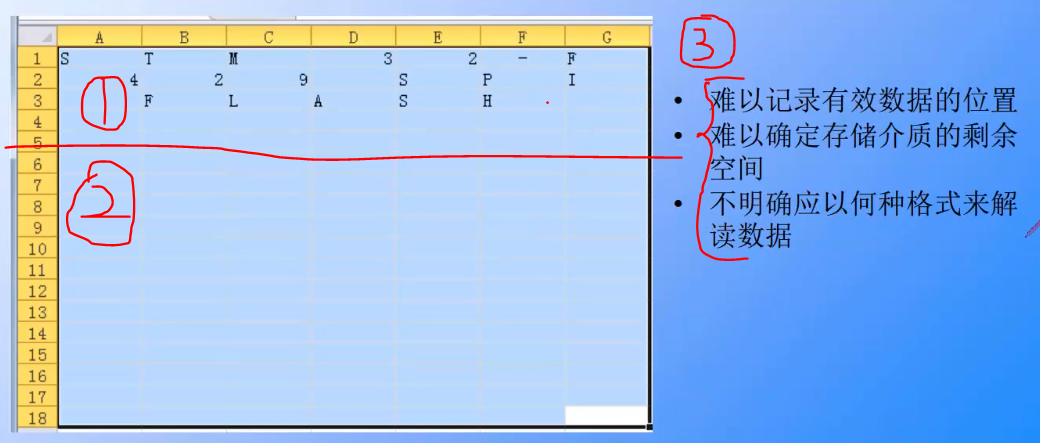
Flash的文件系统Fatfs

1. **基本概念**
2. 什么是文件系统



图中③是提出来的一些问题。如果盲目的直接向Flash存储数据，那么就会出现一些问题：数据位置在哪？存储介质剩余多少？存储的数据以何种格式解读？

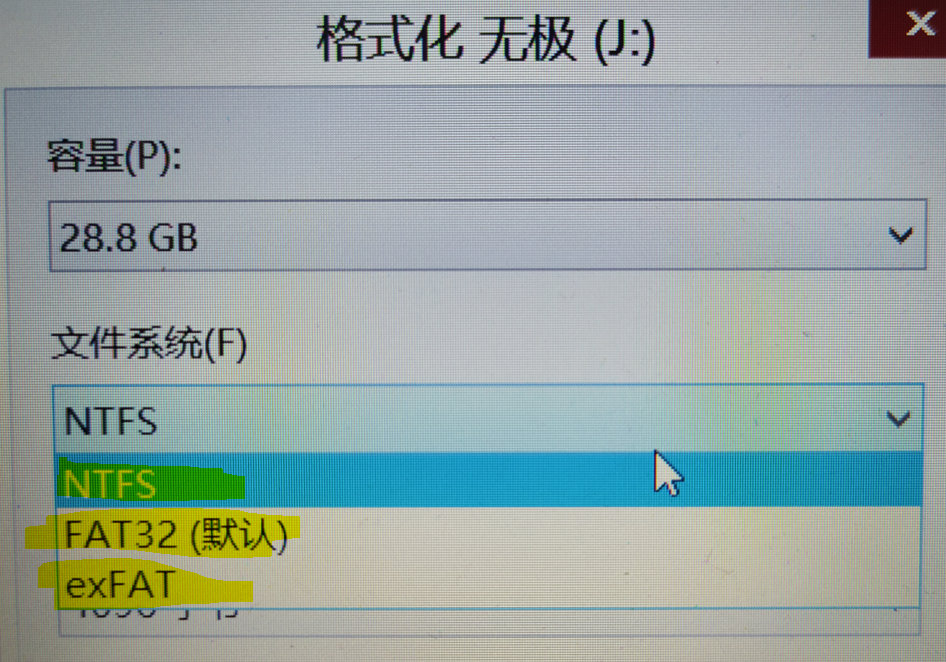
**所以提出如下解决思路：**

我们把存储介质分为两个部分，①和②。类比为一本书①是目录②是内容。把数据存储到②，然后数据的位置，格式，占用容量等信息存储到①。这样读数据的时候，先读①进行索引，知道要读的数据在哪？什么格式？多大？根据这些索引信息，再从②中读取数据。

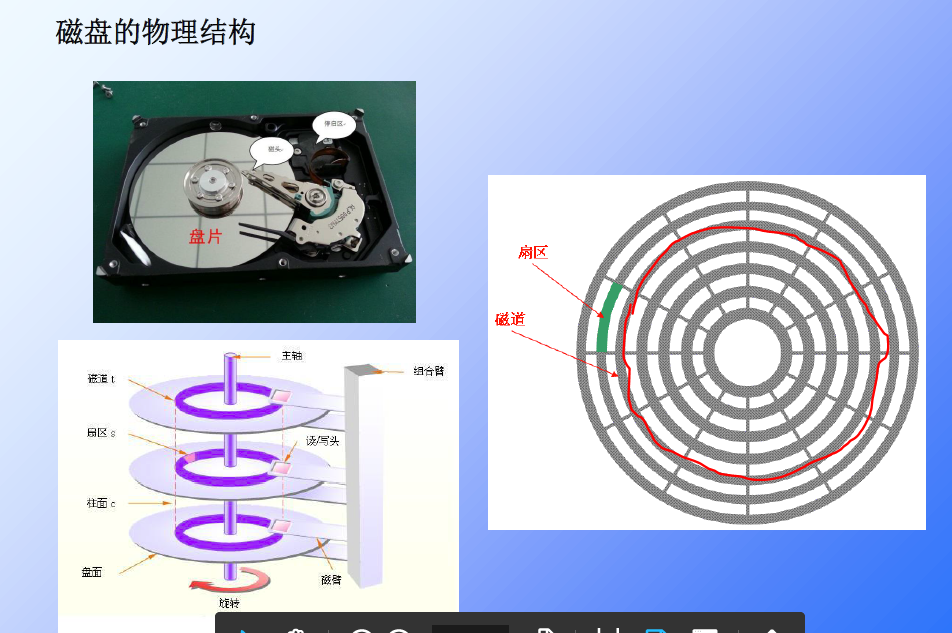
这就是**文件系统的基本思路**

1. **“文件系统”举个例子**

我们以window文件系统为例：

windows就有三个文件操作系统。NTFS,FAT32,exFAT。硬盘区如果有文件系统，那么在硬盘中存东西就会有条例会**以格式的形式存储**。步骤为：新建----各种格式的后缀的文件。而不像STM32串行写Flash那样生硬的只写一段数据。



磁道是一圈的部分

扇区是一圈的一部分

不论擦除还是写都是以扇区为最小单位

磁盘有很多层，每层都有许多扇区和磁道。指针读取到扇区部分有没有磁力，有磁力就是1没磁力就是0.

1. **FAT32**

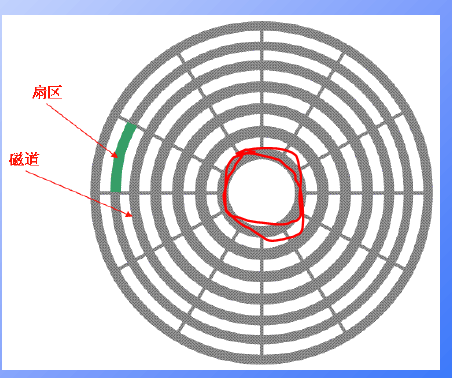
FAT32和FATFS本质是一个东西

**任何存储介质都可以移植文件操作系统**，EEPROM也可以弄。不过因为EEPROM内存太小。因此在FLASH里移植。

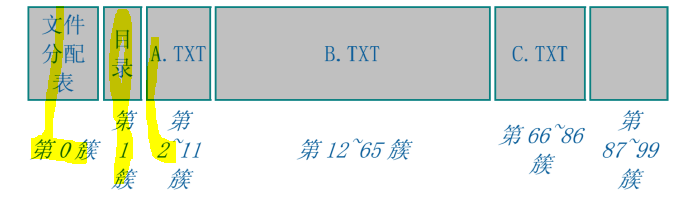
1. 文件系统是如何实现对FLASH操作

使用文件系统前必须先格式化，把存储介质格式化对应的系统。在使用文件系统前，要先对存储介质进行格式化。时会上新建一个文件分配表和目录。这样，系统就可以记数据存放的物理地址，剩余空间。如图在**红色地方建立一个文件分配表和目录。**

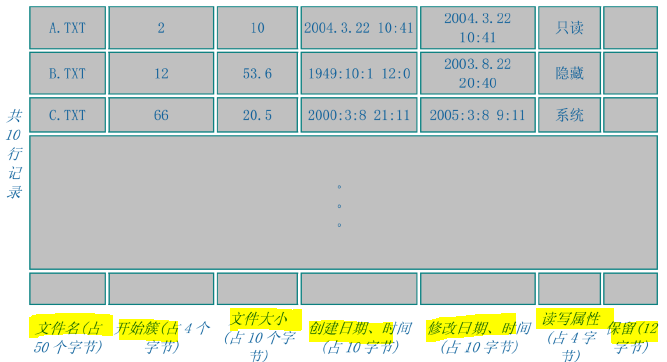
如果我们要建立一个PDF文件，那么回先在红色部分记录相应的标记位，然后再存储PDF文件。那么在读取的时候，会先读取红色的标记位，读到标记位后，就直接去相应的PDF存储地址读取PDF文件。



1. 文件系统格式



* 1. **文件目录:** 记录了文件的开始簇位置、大小等信息

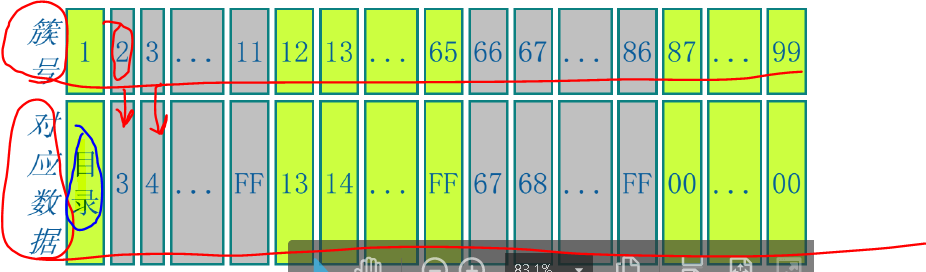


**② 文件分配表(第0簇):**记录了文件的存储的位置。先记录这个文件存储的首簇，然后首簇的末尾又记录了文件下一个存储的簇的位置。多个簇联合起来存储文件。文件分配表就按顺序记录了**文件存储的簇的顺序，指引读取文件内容的顺序。（就是链表的存储）**

**分配表的意义在于，用于查询文件的存储的不连续地址**

**第0簇存储的是文件分配表 第1簇存储的是目录**

文件a.txt 我们根据目录项中指定的a.txt 的**首簇为 2,**然后找到文件分配表的第 2 簇记录，上面登记的是3，就能**确定下一簇是3**。找到文件分配表的第件分配表的第 3 簇记录，上面登记的是4，就能 确定下一簇是 4...... 直到 指到第11 簇，发现下一个**指向是FF就是结束.**



**文件分配表存在的意义就是存储不连续的数据(链表起的作用)**

簇号在FLASH里对应的地址

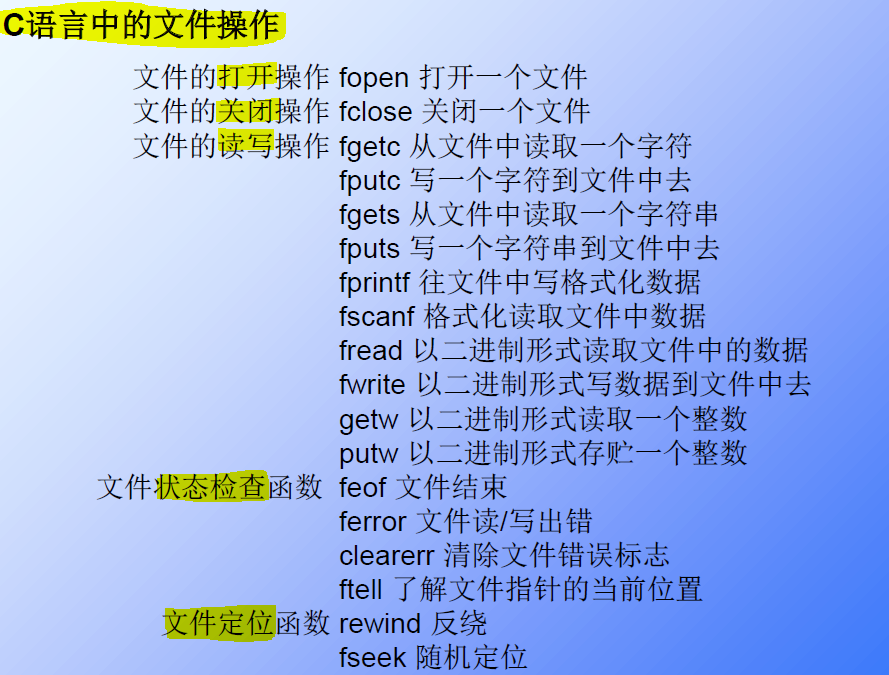
1. 解读文件格式

windows以文件后缀名来决定读取的文件格式

1. 文件系统的代码形式(如何用C语言操作文件呢)？
2. windows给出了以下C语言代码来对文件进行操作：

通过如下代码可以在**IDE中对windows文件直接操作**，可以试试。

实际中，**双击鼠标打开文件，写文件也是对这些函数的操作，只不过用鼠标点击封装了起来**



这些函数是基于windows/linux平台，但stm32并不能直接调用这些函数(32并没有windows/linux系统，因此没这些函数的库)，那则么办？

* 1. 自己写这些函数/或把这些函数移植到32上，那么就能调用了
  2. 整体移植windows/linux不太现实太大，因此只移植文件操作系统FATfs。

1. FATFS介绍
   * 1. FATFS本质上就是一段代码
     2. 由AISI 语言编写并且完全独立于底层的 I/O 介质。代码量小，可以很容易地 不加修改地移植到其他的单片机处理器当中。

FATFS与底层的IO口是独立的怎么理解

FATFS不能直接操作IO进行读写，而是

FATFS先连接你写好的驱动函数，再通过驱动函数对IO口从而对FLASH操作

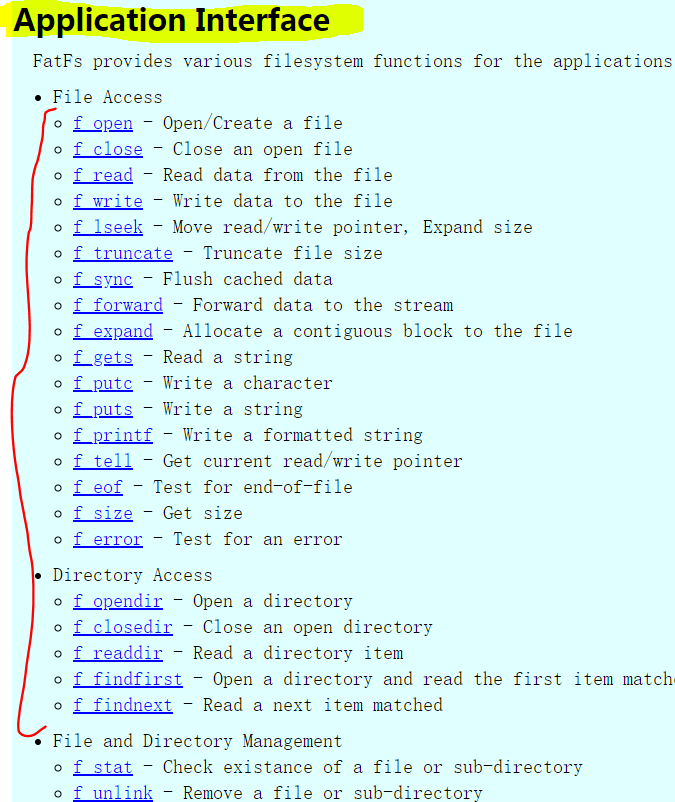
流程如下：

**FATFS+驱动函数(flash/eeprom/sd等的驱动函数单字节写/读等)+外接设备(flash/eeprom/sd)**

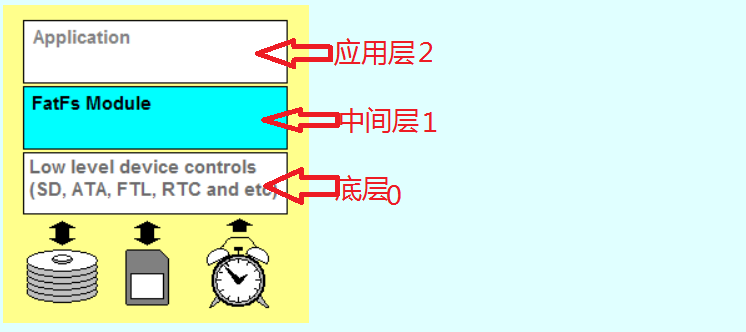
* + 1. FatFs支持FAT12、FAT16、FAT32(这三者区别在于**每次的最大存储文件**大小不同)

将**FATFS移植到32的FLASH里**后就能像操作电脑一样调用windows的函数读写文件了

FATFS的相关函数(和windows的差不多)



1. FATFS移植---综述



从上图知道，一个应用共分三层

* + 1. **应用层2：**就是直接操作的层(例如直接打开/新建/编辑一个文件等)
    2. **中间层1：**就是FATFS的函数定义，比如读取一个文件的函数f\_read，打开一个文件的函数f\_open等
    3. **底层0：**就是你自己写的一个FLASH/SD/EEPROM的驱动函数
    4. **即：**应用层 调用 🡪中间层FATFS提供的函数 调用 🡪底层驱动函数

这是一个计算机的思维，把一个大问题分成不同层的小问题去解决然后整合在一起。将这三层结合在一起就完成一个系统文件的操作。

1. FATFS移植---底层接口(Media access interface)



**中间层1和底层0想结合，**中间层提供了上图的红圈的一些**接口函数。**中间层的函数想正常运行就必须和底层连接。提供的接口函数是**需要我们自己实现**。

也就是我们把**底层0的驱动函数放到中间层1的接口函数里即可。**

1. FATFS下载压缩包的文件

参考 野火PPT串行FLASH文件系统Fatfs(第二节)

整个文件的移植就是更改ffconf.h和diskio.c文件夹

00readme.txt This file.

history.txt Revision history.

ffconf.h Configuration file for FatFs module.

(FatFs的配置文件，可以通过裁剪各种宏定义，达到对函数的配置(消减/增加部分功能来尽量好的利用资源) 文件支持中文/英文/日文等版本的修改)

ff.h Common include file for FatFs and application module.

(include .h 文件)

ff.c FatFs module.

(核心的FatFs文件系统管理的相关函数)

diskio.h Common include file for FatFs and disk I/O module.

(include .h 文件)

diskio.c An example of glue function to attach existing disk I/O module to FatFs.

(连接了FATFS和存储介质底层IO的一些函数，函数要自己实现)

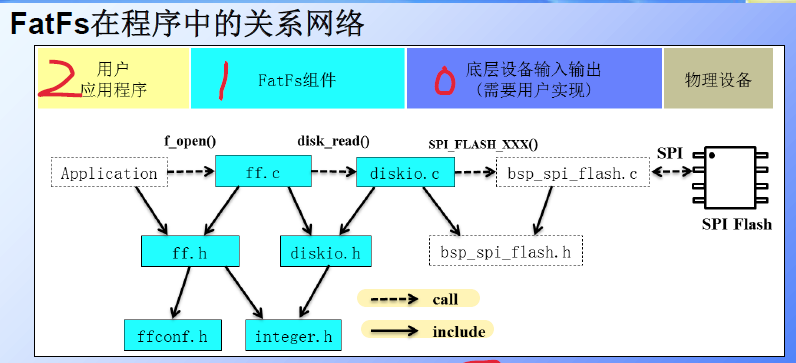
integer.h Integer type definitions for FatFs.

(整合各种类型的定义， 把各种的变量类型重新定义一下，方便使用)

option Optional external functions.

(外部的一些支持各自语言的文件 cc936支持中文 英文需要ccsbcs，syscall文件)

1. FATFS移植的**函数相关图**



1. 文件系统的进一步配置
   1. **前提知识**

**0.1关于W25Q128(正点)**

①



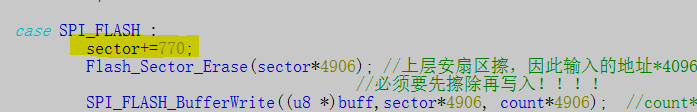
**0.2关于W25Q64(野火)**

①

|  |  |  |
| --- | --- | --- |
|  | **W25Q128(正点)** | **W25Q64(野火)** |
| 存储架构 | W25Q128前12M字节给fatfs用  12M字节后,用于存放字库字库占用3.09M  剩余部分,给客户自己用 | W25Q64前3M用于存放字库字库占用  3M字节后用于存放FATFS  再之后自己用 |
|  | 正点的SPI写函数**没有自动对齐**，因此规定disk\_read和disk\_write一次最大512，用count循环 | SPI写函数**没有自动对齐，一次**disk\_read和disk\_write数量不限 |
|  | Sector不偏移 | Sector先偏移770 |
|  | disk\_ioctl写法不同  一个GET\_SECTOR\_SIZE=4096  计算GET\_SECTOR\_COUNT=16M/4096=4096  计算GET\_BLOCK\_SIZE(最大擦出扇区个数)=1  (4096\*1=4096) | disk\_ioctl写法不同  一个GET\_SECTOR\_SIZE=512  计算GET\_SECTOR\_COUNT=16M/512= 2048\*12  GET\_BLOCK\_SIZE(最大擦出扇区个数)=8  (512\*8=4096) |
|  |  |  |

* 1. **为什么要加770(外部存储器规划)**

在diskio函数配置中，我们要在读写的底层配置中给扇区sector+=770。为什么要这么操作呢？是因为**(外部存储器规划)**的原因。



一个FLASH，他的前面的3M(770\*4k/1024=3M)内存要存储以下东西，只有后面的部分才是我们真正能用上的部分。



这些预先存储的一些字库文件，在后面的液晶屏，FSMC,操作系统中要使用到，因此要把这些预先写入。但如这些字库文件也可以存到STM32中，但为了节省空间，提高效率，就写入到外部FLASH中。所以我们要加上偏移地址，因为不能将写入的字库文件覆盖掉。

如果不小心覆盖掉，重新下载刷机程序。

读字库文件的数据用SPI的基本通讯去读，速度快些，不使用文件系统

* 1. 文件系统的**文件名支持中文(移植常文件)**

步骤：

* + 1. 需要加入cc936库文件
    2. 在ffconf里把codepage改为936
    3. 在ffconf里设置\_USE\_LFN 改为1

因为常文件(中文字库cc936)要存储在STM32的FLASH里，所以代码量增大，下载速度慢

1. 文件系统的使用
2. 编程中的一些问题
   1. **指针强制转换**

这个指针\*应该放在变量后:(DWORD\*).

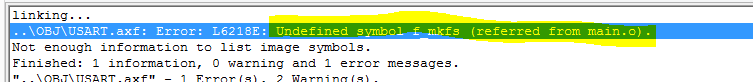
下图1正确





1. Undefined symbol f\_mkfs (referred from main.o).

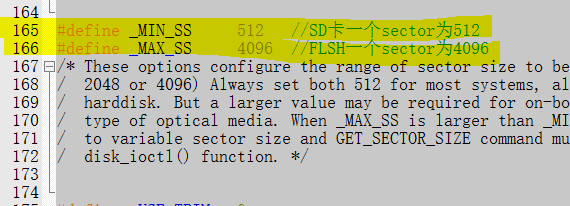
找不到函数的定义，这个函数没有定义。查相关的.c文件，看函数的定义是否被宏定义注释掉了



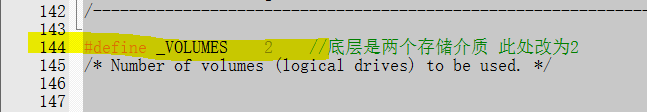
1. 挂载文件系统，调用f\_mount()函数

只要f\_mount()的返回值不等于0，我们要在ffconf.h文件中做如下更改。

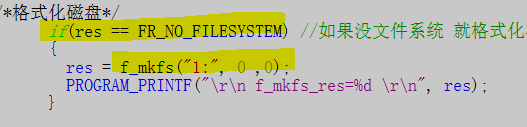
* + 1. 更改165行 \_MAX\_SS为4096，因为FLASH的一个扇区为4096



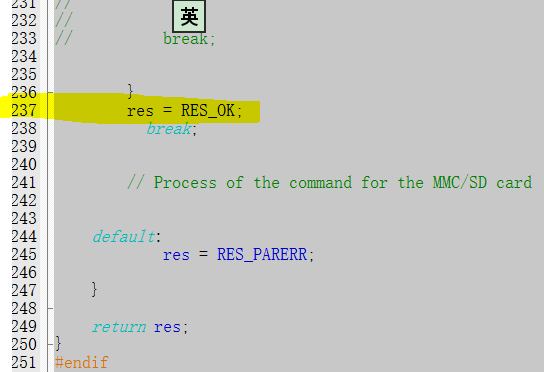
* + 1. 更改144行 \_VOLUMES的个数。(diskio函数定义了几个存储介质就改为几)



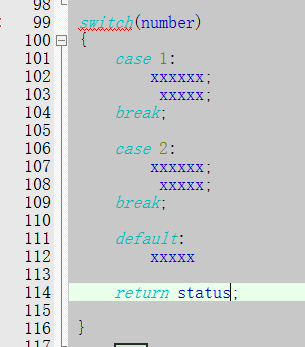
* + 1. 如果f\_mount()的返回值为13，FR\_NO\_FILESYSTEM。则证明存储介质没文件系统，需要调用f\_mkfs格式化。**先格式化再挂载**



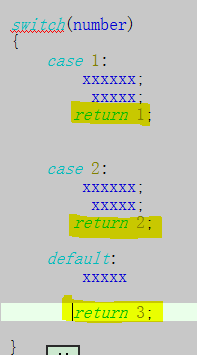
* + 1. 如果f\_mount()的返回值为1，FR\_DISK\_ERR。证明底层有问题，即diskio.c函数的底层接口有问题。我这次的问题是Miscellaneous Functions函数接口没给返回值



1. 关于**开关函数switch:**
   * 1. 开关函数的标志写法如下：



* + 1. **case后必需要有break，最后必须要有default;**
    2. **这种没有break，直接return的写法虽然也对。但是野路子，不规范！**

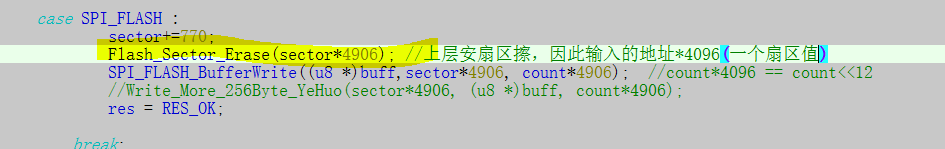


1. 调用文件系统函数，输入存储介质标号

必须以 “number：”这种形式输入存储介质标号。如“1：” “2：” 为啥我也不知道就是规定



1. disk\_write()函数，底层配置必须要**先擦除再写入**

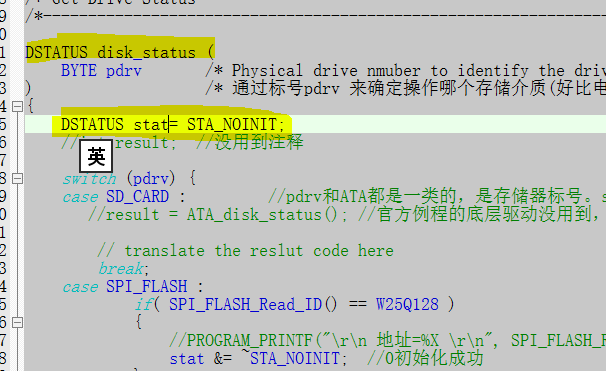


1. 上层调用函数后，必须在ffconf.h里把要调用的函数打开(即宏定义变为1)
2. 调用f\_open()函数，状态位返回值为10？？？

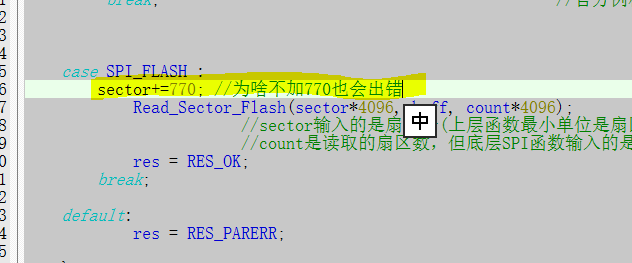


状态位返回值为10，写保护。猜测还是底层diskio.c原因

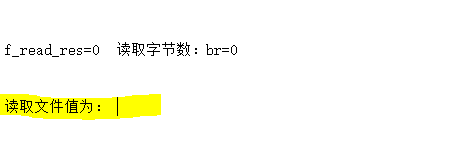
因为写diskio函数时，**初始化状态变量DSTATUS stat没有赋初值**，因此电脑会随机赋值导致返回值出错.切记定义变量尽量赋一个初始值，这属于玄学问题.



9. 为啥写扇区地址不加770也会出错

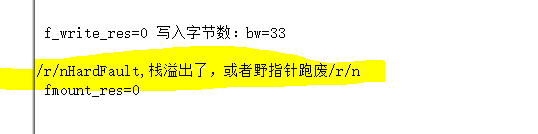


1. 读不出来写入的值 或者 一读值栈溢出
2. 读取函数返回值正常0，但读不出来值



**光标的原因**。写完函数之后，光标就会跑到写完的数据后头。在读取函数，光标后头根本没值，读出来的当然是空的。所以再读取之前，要**调用函数f\_lseek();让光标跳到数据的最开始，**这样读就会有数据了。

1. 栈溢出，程序出错



调用FATFS文件系统写文件的话，相关**存储数据的变量就必须要用FATFS文件系统中的"变量类型"定义变量**。如果不按照FATFS文件系统中的"变量类型"定义变量，就会卡死了。因为**分配的栈的大小不匹配就卡死了**。

FATFS文件系统的**函数是按照文件系统中的"变量类型"定义变量来操作**，而定义的变量如果不匹配，那么**函数操作的变量的栈大小 与 自己定义变量的栈大小不匹配**，所以就卡死了，则就会出现栈溢出

1. 我的主函数，出现**第二次打开失败**。野火的函数第二次却能打开成功。

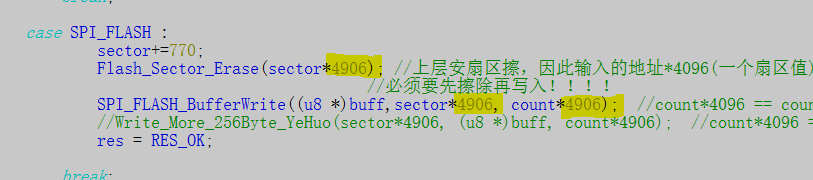
排除法：

我写文件-----我再读 失败

野火写文件----野火读 成功

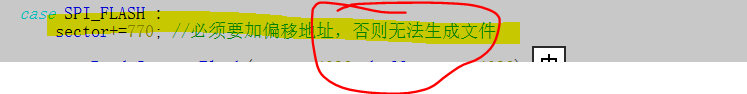
野火写----我读 成功

我的写文件就有问题！！！diskio写/SPI写有问题？经检查不是SPI写函数的问题，是diskio写接口写函数的问题，问题如下：



因为SPIFLASH函数是按字节写，而文件系统是按照sector写。所以调用SPIFLASH函数**要给sector\*4096.我写成\*4906了，所以错误！！打错了！！**

1. 为啥变成u盘，**无法建立文件？？？**



1. 为啥写完**打开文件没有值？？？**

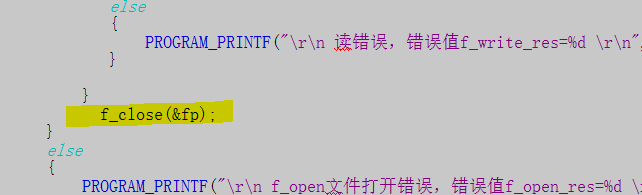
缺这个玩意，每次操作完之后，要**记得f\_close关闭文件**。关闭了值才会保存，**否则光有文件名却没值.**

**流程如下： 打开文件f\_open---**

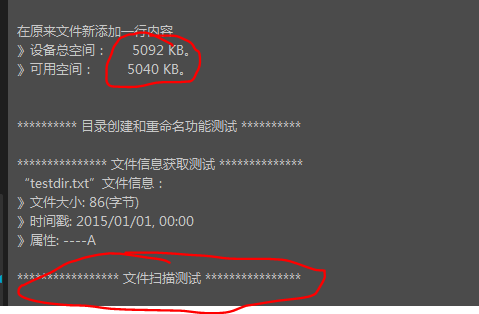
**对文件光标进行定位f\_lseek---**

**读写文件---**

**关闭文件**



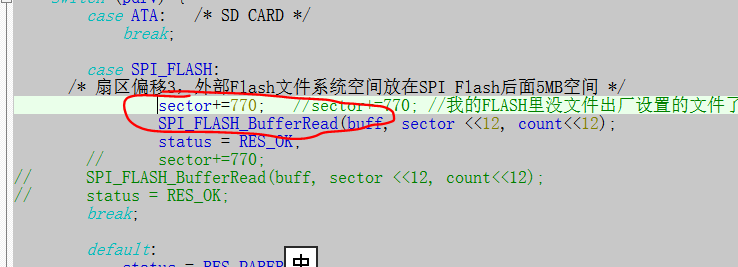
1. 读取的**内存空间是错的** **扫描不出来文件**



用正点的板子，野火的程序。卡着蛋了。

目前的原因：前770sector给字库文件让出来了了，但FLASH格式化过字库文件没了，所以就从770后开始读，读字库根本没字库就一直找所以啥都没有了。(重新放入sysytem文件就不会卡到文件扫描测试)

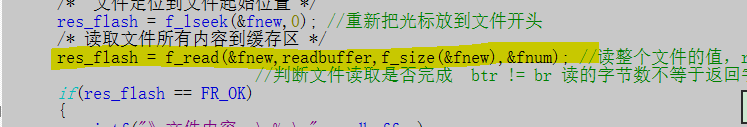
怀疑：文件系统放置的不一样。其实字库文件在最后3m放着呢，没格式化？ 是格式化了的，FLASH没系统文件，综合实验跑不了。



**目前解决方法，偏移770删除了就好了**

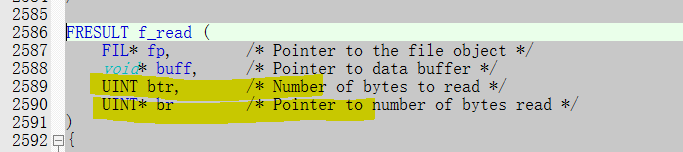
**太难了 又对又不对的 不知道为啥**

1. f\_read循环读取数据（判断数据是否读完）

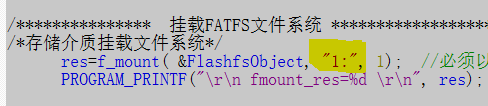


有时候因为readbuffer的大小不够，要循环的调用f\_read，往readbuffer存数据，那么如何判断数据读完了呢？

当btr(要读的字节数)和br(返回的读取的字节个数)不同时，证明数据已经读完了，那么我们就可以跳出循环调用f\_read



1. **FLASH变U盘时候，要记得打开串口才正常**
2. 这种写法**"1:"**代表什么？



要输入的数据是1：，带“”