黑灵—Petit Fatfs 移植

FATFS 简介

FatFs 是一个通用的文件系统模块，用于在小型嵌入式系统中实现 FAT 文件系统。FatFs 的编写遵循 ANSI C，因此不依赖于硬件平台。它可以嵌入到便宜的微控制器中，如 8051, PIC, AVR, SH, Z80, H8, ARM 等等，不需要做任何修改。

此次开发黑灵时，在文件系统选择上，我们选择了 FATFS 官网下的 Petit Fatfs 文件系统。Petit Fatfs 文件系统是 FatFS 的精简，比较适用于低端 8 位单片机中。可以用在小 RAM 的单片机中，RAM 可以小于扇区的 RAM（512bytes)中。

而且黑灵采用官网指定文件系统还有一个很重要的一点就是：大家在学会以后，可以拿过来随时移植到自己想用的项目上，而不用考虑太多的程序限制问题。

STC15F2K60S2 它的 RAM 是 2k 对于移植 Petit Fatfs 文件系统比较合适，这也算是 51 方面对 Petit Fatfs 文件系统移植的一种新跨越。

Petit Fatfs 文件系统移植详解

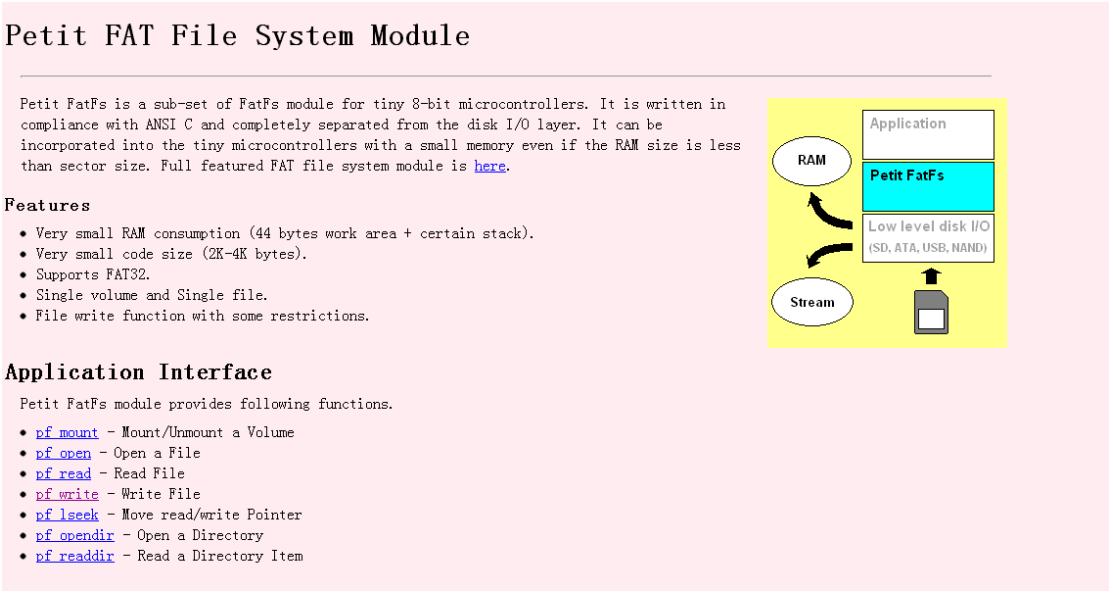
在对 Petit Fatfs 文件系统的移植上我们也找了好多的资料，下面就对此作一个详细的讲解

官网方面

FATFS 的官网主站网址是 http://elm-chan.org/fsw_e.html

进入后会出现一个界面如图

这里看第四个选项 就是 Petit Fatfs Module 意思就是小型文件系统模块，点击进入



Petit FAT File System Module

Petit FatFs is a sub-set of FatFs module for tiny 8-bit microcontrollers. It is written in compliance with ANSI C and completely separated from the disk I/O layer. It can be incorporated into the tiny microcontrollers with a small memory even if the RAM size is less than sector size. Full featured FAT file system module is here.

Features

- Very small RAM consumption (44 bytes work area + certain stack).
- Very small code size (2K-4K bytes).
- Supports FAT32.
- Single volume and Single file.
- File write function with some restrictions.

Application Interface

Petit FatFs module provides following functions.

- pf_mount - Mount/Unmount a Volume
- pf_open - Open a File
- pf_read - Read File
- pf_write - Write File
- pf_lseek - Move read/write Pointer
- pf_opendir - Open a Directory
- pf_readdir - Read a Directory Item

第一段是 **Petit Fatfs 文件系统的介绍** 主要意思：

Petit FatFs 是一个极小的 8 位微控制器 FatFs 模块。它是和控制器 IO 层完全分离的。它可以被嵌入到一个很小内存的微控制器中，即使内存大小小于扇区大小。

第二段

特征

很小的内存消耗（44 字节的工作区+一定堆栈）。

非常小的代码的大小（2k-4k 字节）。

支持 FAT32。

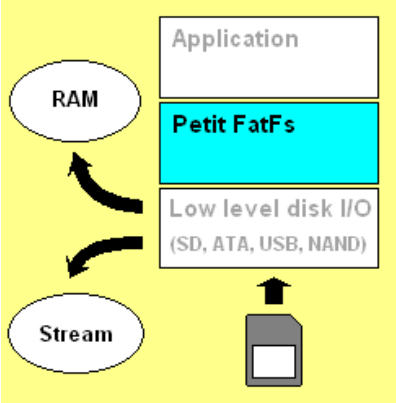
单量单文件。

写文件的一些限制写函数。

文件的限制

官方介绍的限制：1、只能挂载一个设备。2、Petit FatFs 不能创建文件，只能对已存在的文件进行操作。3、写函数只能写到原来文件的大小，不能扩展大小，不能更新文件时间信息，不能写只读文件。4 这一点比较重要，就是 Petit Fatfs 不支持长文件名比如说打开一个文件 pf_open("/机器猫.bmp"); 这里文件名字不能超过 8 个字节 机器猫 是 6 个字节 在加一个汉字也可以，如果超过了 8 个字节，就找不到了，找不到就意味着无法打开指定路径文件。这一点一定要注意！

Petit FATFs 能够在不同的单片机平台上运行，具有良好的层次结构，其层次结构，如下图



第三段

Application Interface

Petit FatFs module provides following functions.

- pf_mount - Mount/Unmount a Volume
- pf_open - Open a File
- pf_read - Read File
- pf_write - Write File
- pf_lseek - Move read/write Pointer
- pf_opendir - Open a Directory
- pf_readdir - Read a Directory Item

以上的每一项都可以单独点击进入 里面都有详细的讲解，但都是英文，下面我们就功能做一下详细介绍

应用层详细讲解

应用层位于最顶层，它提供供了以下几个函数：

① pf_mount ：

```
FRESULT pf_mount (
  FATFS*  fs  /* [IN] Pointer to the work area */
);
```

为 Petit FATFs 模块注册或者卸载一块工作区域，它包括了设备的初始化（diskio.c 中的 disk_initialize）、文件系统的设置（FA T32、FAT）等，是 Petit FATFs 能够工作的前提，

在调用其他应用层函数前应先调用此函数。

一般用法是：

```
FATFS fatfs;   ///定义一个文件系统对象
if(pf_mount(&fatfs))      printf("Failed"); //如果返回 1，则挂载失败，否则成功
```

② pf_open：

```
FRESULT pf_open (
  const char* path  /* [IN] Pointer to the file neme */
);
```

打开一个已经存在的文件，在对文件进行读操作和移动读写指针前，首先应该调用该函数。打开的文件必须是已存在的，

一般用法：

if(pf_open("MESSAGE.TXT")) printf("Failed"); //如果返回 1，则打开失败，否则成功

③ pf_read：

```
FRESULT pf_read (
  void* buff,  /* [OUT] Pointer to the read buffer */
  WORD btr,    /* [IN]  Number of bytes to read */
  WORD* br     /* [OUT] Pointer to the variable to return number of bytes read */
);
```

读一个文件。函数的三个参数分别表示读出数据存放的地址，读出数据的大小，反回真正读出的 char 数据的大小。

④ pf_write：

```
FRESULT pf_write (
  const void* buff, /* [IN]  Pointer to the data to be written */
  WORD btw,         /* [IN]  Number of bytes to write */
  WORD* bw          /* [OUT] Pointer to the variable to return number of bytes written */
);
```

写一个文件。函数的三个参数分别表示写入的数据存放的地址，要写入的数据的大小，返回真正写入的 char 数据的大小。

⑤ pf_lseek：

```
FRESULT pf_lseek (
  DWORD offset        /* [IN] File offset in unit of byte */
);
```

移动读/写指针。参数表示从第几个数据开始操作。

⑥ pf_opendir：

```
FRESULT pf_opendir (
  DIR* dp,           /* [OUT] Pointer to the blank directory object structure */
  const char* path   /* [IN]  Pointer to the directory name */
);
```

打开一个目录。第一个参数表示指向空白目录结构，第二个表示指向一个已存在的目录名。

⑦ pf_readdir：

```
FRESULT pf_readdir (
  DIR* dp,       /* [IN]  Pointer to the open directory object */
  FILINFO* fno   /* [OUT] Pointer to the file information structure */
);
```

读一个目录项。主要读取一个项目下的相关信息，如文件夹下的各文件名。

使用者在使用应用层函数时只需调用即可无须理会 Petit FATFs 的内部结构以及复杂的

FAT 协议。 中间层 Petit FATFs 包含了 FAT 的读写协议，和最底层 Low Level Disk I/O 完全分离，所以一般不用修改。 Low Level Disk I/O 位于最底层，它不是 Petit FATFs 模块的一部分，需要根据不同的单片机和不同的存储媒介进行编写，是移植过程中最重要的一部分。

第四段

接口函数

Disk I/O Interface

Since the Petit FatFs module is completely separated from disk I/O layer, it requires following functions to lower layer to read the physical disk. The low level disk I/O module is not a part of Petit FatFs module and it must be provided by user. The sample drivers are also available in the resources.

- disk_initialize - Initialize disk drive
- disk_readp - Read partial sector
- disk_writep - Write partial sector

由于 Petit FatFs 模块是从磁盘 I/O 层完全分离，它需要以下功能低层读物理磁盘。而这几个读物理层函数要由用户提供。

disk_initialize –SD 卡初始化函数

disk_readp　　SD 卡读数据

disk_writep　　SD 卡写数据

接口函数是连接底层和应用层的重要枢纽，从上面可以看错就 3 个函数需要我们来写的，而且 sd 卡初始化，sd 读数据，sd 写数据这些都是数据对 SD 卡的单一操作，我们在前面的程序里也对 SD 卡的初始化、读、写功能做了相应的讲解。文件系统移植后，按要求往里填函数即可。

到这里就应该有一个大概认识了，Petit FatFs 文件系统主要就是一个应用层，它的功能就是读写 SD 卡。对于我们来说，只要知道如何利用好应用层函数达到我们想要的效果就可以了，不用去考虑文件系统的工作过程，这样一想它就没什么那难度了！

Petit Fatfs 文件系统移植具体操作步骤

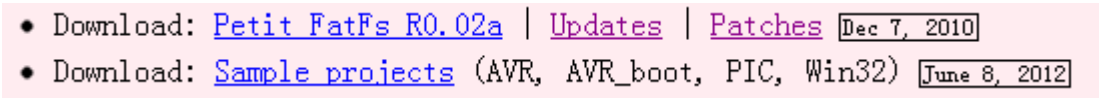
下载官网文件系统压缩文件

Resources

The Petit FatFs module is a free software and is opened for education, research and development. You can use, modify and/or redistribute it for personal, non-profit or commercial use without any restriction under your responsibility. For further information, refer to the application note.

- FatFs User Forum
- Read first: Petit FatFs module application note Dec 7, 2010
- Download: Petit FatFs R0.02a | Updates | Patches Dec 7, 2010
- Download: Sample projects (AVR, AVR_boot, PIC, Win32) June 8, 2012
- FAT32 Specification by Microsoft (The reference document on FAT file system)
- How to Use MMC/SDC
- Benchmark 3 (ATtiny85/8MHz with MMC via USI)
- Previous versions: R0.02 | R0.01a

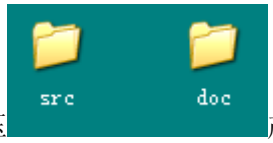
官网的最下一段，就是有关文件系统函数的下载了



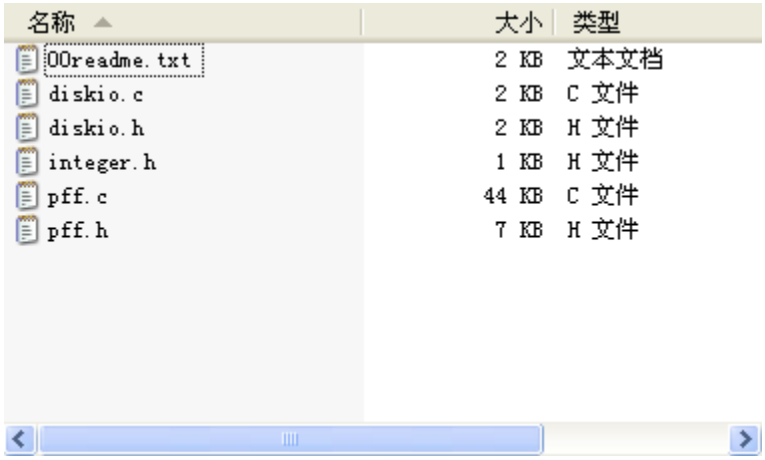
现在的地方就有这两个　第一个是文件系统应用函数 第二个是例程，例程好像是 AVR 的没有仔细研究。这里直接讲下载压缩文件。

点击后出现上图

下载到桌面，然后进行解压



产生两个文件夹，src 就是我们要用到的应用层函数，doc 应该是一些介绍（如果感兴趣请自己研究）这里不做都分析，下面我们讲解 src 文件夹



打开后可以看到一共有 6 个文本，第一个很明显就是说明了，也就是说下面 5 个文件是我们要移植到程序里的 Petit Fatfs 文件系统函数。

其中 integer.h 和 pff.c 一般情况下不需要进行改动，

需要改动的是 diskio.c 和 pff.h 。

改动 1

diskio.c disk io 意思就是接口层函数，里面主要包括三个函数的编写即 disk_initialize 、disk_readp 和 disk_writep ，这三个函数已经在上面讲过，这里只需将这三个函数写在 diskio.c 里面相应的位置即可。

改动 2

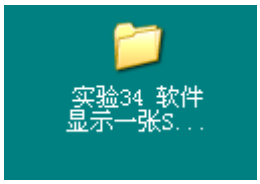
配置的工作主要在 **pff.h** 里面。在 pff.h 里面我们需要配置的有：

① #define _USE_READ 。有 0 和 1 两个值可被选择，选择 0 则不使能文件读操作，选择

1 则使能读文件操作，在这里我们选择 1，使能读文件操作。
② #define _USE_DIR。为 1 时使能打开一个目录和读一个目录项操作，为 0 时则不使能相应操作，因为我们需要打开目录也是打开文件夹的意思 所以选择 1
③ #define _USE_LSEEK。有 0 和 1 两个选择项。1 时使能移动读/写指针操作，为 0 时则禁止。。 在电子书找页数偏移时会用到，所以选择 1
④ #define _USE_WRITE。为 1 时使能写文件操作，为 0 时禁止写文件操作。
⑤ #define _FS_FAT32。为 0 时仅支持 FAT16 文件系统，为 1 时支持 FAT32 文件系统。FAT32 文件系统是 FAT16 文件系统的升级，而且现在 SD 卡在出厂时一般都默认被格式化为 FAT32 文件系统，所以设置其值为 1。
⑥ #define _WORD_ACCESS。有 0 和 1 两个值可被选择。为 0 时选择字节寻址方式，为 1 时选择字寻址方式。

具体移植流程

这里我们随便找一个写好 SD.C 底层的函数来移植文件系统

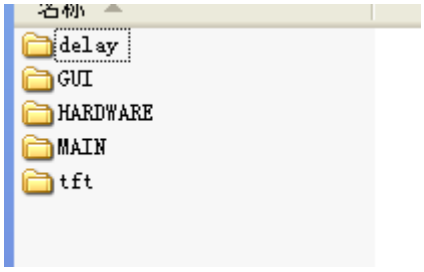


官网的文件系统 一共有 5 个文件，这里我们将这 5 个文件放到一个文件夹里

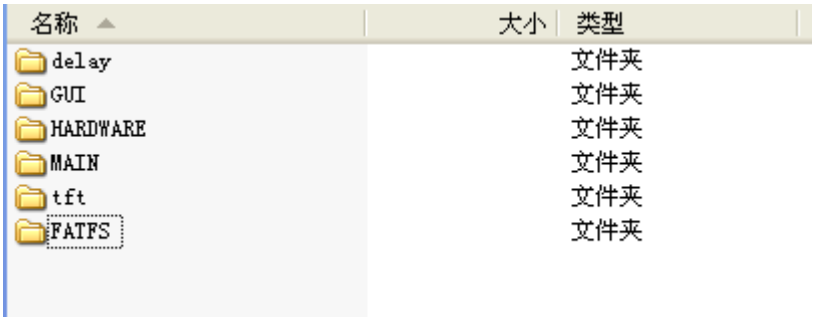
新建一个文件夹

将以上文件存入文件夹内 这样我们这个建的文件夹就是主函数调用的文件系统文件夹。

打开程序的文件夹

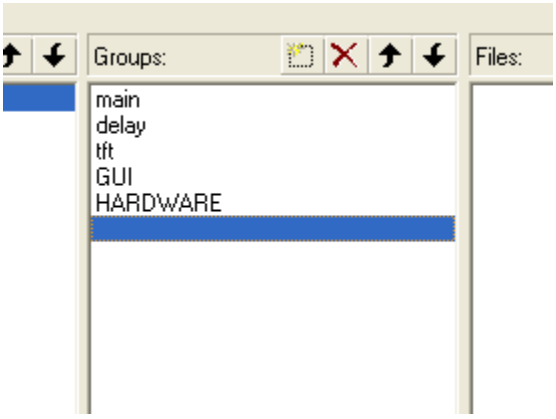


直接将刚才准备好的文件系统文件夹存入里面

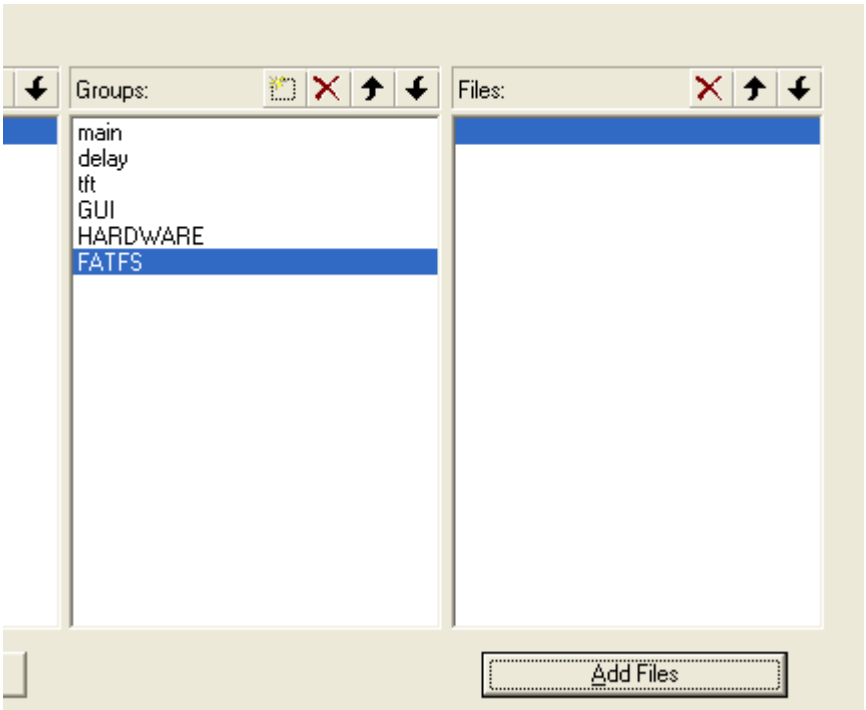


然后都开主函数 进行添加

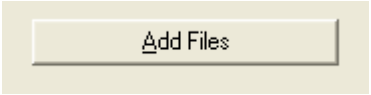
先添加文件系统的.C 函数



新建一个子程序.C 文件夹，命名为 FATFS 如下图



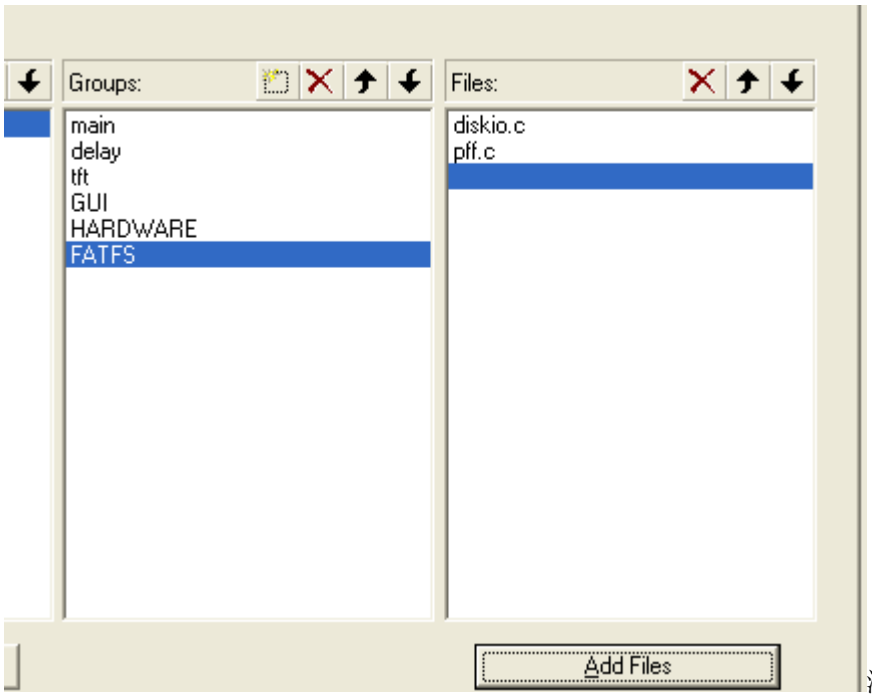
然后添加刚才存入的文件系统里的所有.C 文件




Add Files to Group 'FATFS'
查找范围(I):
程序
delay
FATFS
GUI
HARDWARE
MAIN
tft
文件名(N):
文件类型(T):
C Source file (*.c)
Add
Close


Add Files to Group 'FATFS'
查找范围(I):
FATFS
diskio.c
pff.c
文件名(N):
文件类型(T):
C Source file (*.c)
Add
Close

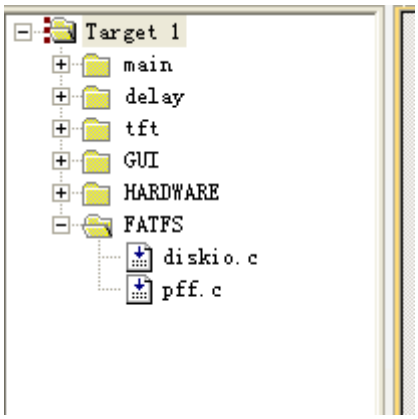
其实就 2 个.C 文件



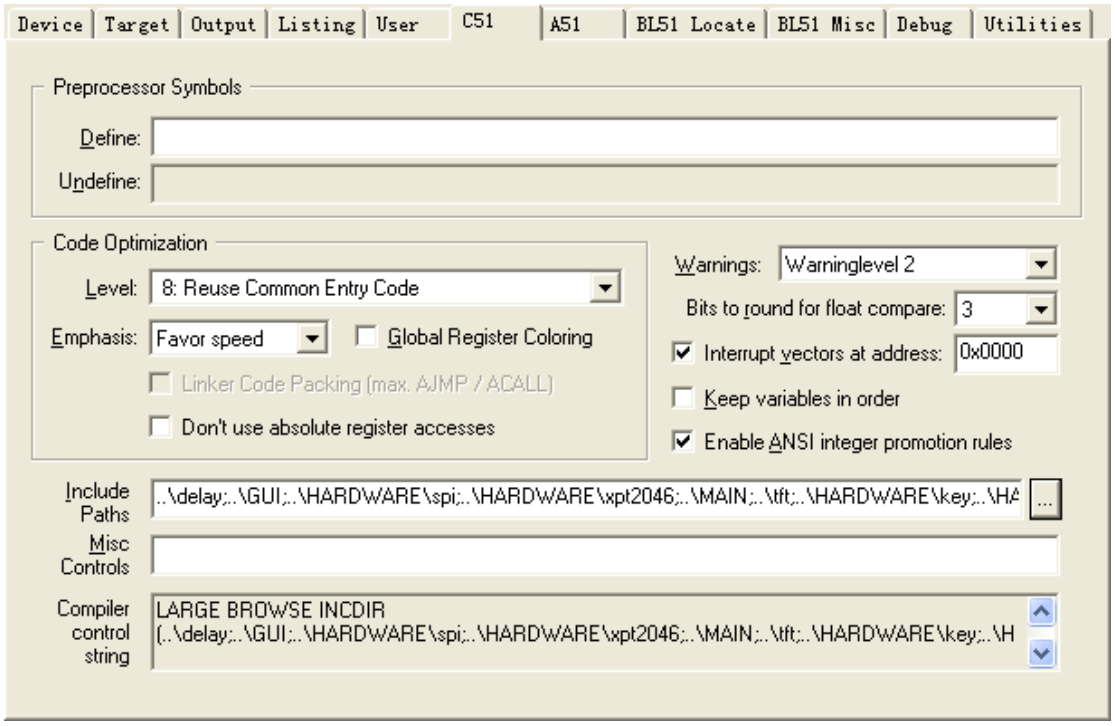
添加后效果如图

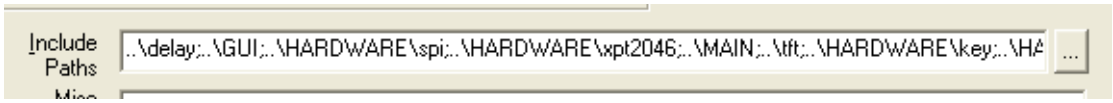
确定退出

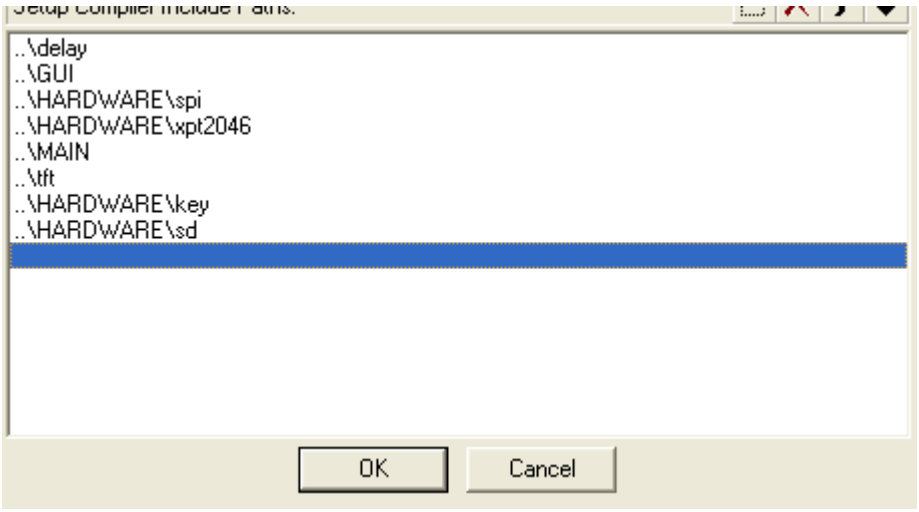
这样在右侧就可以看到添加的文件系统.C 函数

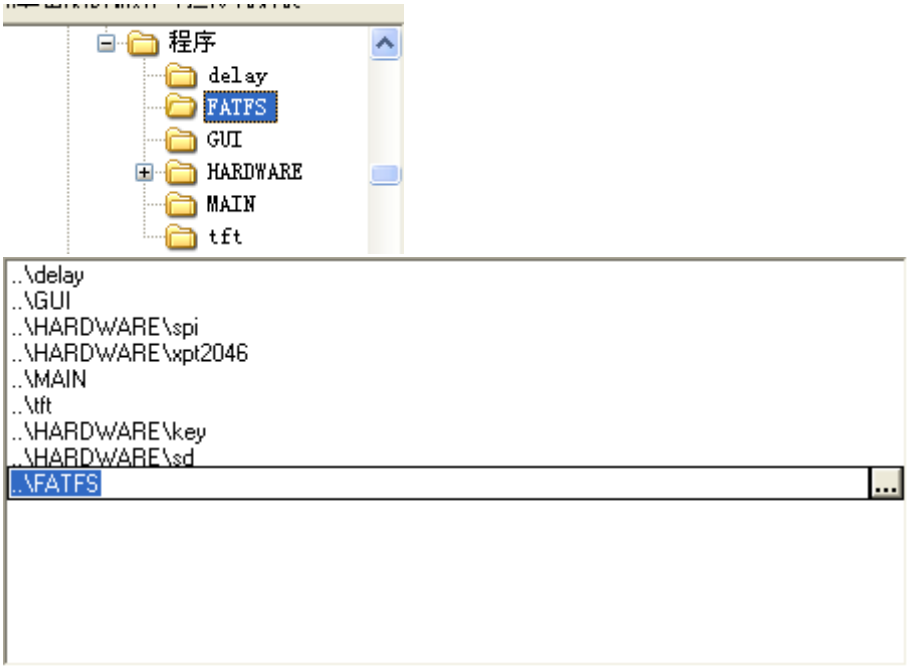


然后添加文件系统的.H 文件


Device
Target
Output
Listing
User
C51
A51
BL51 Locate
BL51 Misc
Debug
Utilities
Preprocessor Symbols
Define:
Undefine:
Code Optimization
Level:
8: Reuse Common Entry Code
Emphasis:
Favor speed
Global Register Coloring
Linker Code Packing (max. AJMP / ACALL)
Don't use absolute register accesses
Warnings:
Warninglevel 2
Bits to round for float compare:
3
Interrupt vectors at address:
0x0000
Keep variables in order
Enable ANSI integer promotion rules
Include Paths
..\delay;..\GUI;..\HARDWARE\spi;..\HARDWARE\xpt2046;..\MAIN;..\tft;..\HARDWARE\key;..\HA
Misc Controls
Compiler control string
LARGE BROWSE INCDIR
(..\delay;..\GUI;..\HARDWARE\spi;..\HARDWARE\xpt2046;..\MAIN;..\tft;..\HARDWARE\key;..\H







确定退出

到这里 文件系统的文件移植就完成了，接下来就是写接口函数和配置 pff.h 的功能项

A 写接口函数

接口函数在 diskio.c 里面 ， 直接在程序上双击打开

diskio.c

```c
/*-----------------------------------------------------------------------*/
/* Low level disk I/O module skeleton for Petit FatFs (C)ChaN, 2009      */
/*-----------------------------------------------------------------------*/

#include "diskio.h"



/*-----------------------------------------------------------------------*/
/* Initialize Disk Drive                                                 */
/*-----------------------------------------------------------------------*/

DSTATUS disk_initialize (void)
{
    DSTATUS stat;

    // Put your code here

    return stat;
}



/*-----------------------------------------------------------------------*/
/* Read Partial Sector                                                   */
/*-----------------------------------------------------------------------*/

DRESULT disk_readp (
    BYTE* dest,         /* Pointer to the destination object */
    DWORD sector,       /* Sector number (LBA) */
    WORD sofs,          /* Offset in the sector */
    WORD count          /* Byte count (bit15:destination) */
)
{
    DRESULT res;

    // Put your code here

    return res;
}

```

```
/*-----------------------------------------------------------------------*/
/* Write Partial Sector                                                  */
/*-----------------------------------------------------------------------*/

DRESULT disk_writep (
    BYTE* buff,     /* Pointer to the data to be written, NULL:Initiate/Finalize write
    DWORD sc        /* Sector number (LBA) or Number of bytes to send */
)
{
    DRESULT res;


    if (!buff) {
        if (sc) {

            // Initiate write process

        } else {

            // Finalize write process

        }
    } else {

        // Send data to the disk

    }

    return res;
}
```

可以看到就 3 块，虽然是英文描述的，但是也比较好理解

下面我们分部写入，在写入之前别忘了添加函数里要用到的 SD.c 里的函数、单片机头文件等如图

```
#include "diskio.h"
#include "stc15f2k60s2.h"
#include "spi.h"
#include "sd.h"
#include "def.h"
```

从图中可以看出还是设计到 SPI 底层，SD 卡的最终通讯方式就是用的 SPI，在函数的读写都要用到 SPI 底层，在下面的调用上，大家就可以看到它的作用了。

第 1 部 SD 卡初始化

在前面的程序里已经把 SD 卡的底层写出来了，而且还有写了 **SD 卡简介 elh.pdf** 的文档，所以接下来就不讲解关于 SD 卡的底层是怎么来的，如果有不明白的地方请参考前面的资料！

写入前效果

```
/*-----------------------------------------------------------------------*/
/* Initialize Disk Drive                                                 */
/*-----------------------------------------------------------------------*/

DSTATUS disk_initialize (void)
{
    DSTATUS stat;

    // Put your code here

    return stat;
}
```

```
/*-----------------------------------------------------------------------*/
/* Initialize Disk Drive                                                 */
/*-----------------------------------------------------------------------*/

DSTATUS disk_initialize (void)
{
    DSTATUS stat;

    stat=SD_Init();

    return stat;
}
```

写入后 这一步就是初始化 SD 卡 stat 是状态参数，读取的就是 SD 初始化的状态并返回给文件系统。

第 2 部 读 SD 卡数据

```
/*-----------------------------------------------------------------------*/
/* Read Partial Sector                                                   */
/*-----------------------------------------------------------------------*/

DRESULT disk_readp (
    BYTE* dest,         /* Pointer to the destination object */
    DWORD sector,       /* Sector number (LBA) */
    WORD sofs,          /* Offset in the sector */
    WORD count          /* Byte count (bit15:destination) */
)
{
    DRESULT res;

    // Put your code here

    return res;
}
```

变量翻译

```
    BYTE* dest,         /* Pointer to the destination object 存放数据的地址*/
    DWORD sector,       /* Sector number (LBA)  要读的扇区数*/
    WORD sofs,          /* Offset in the sector 扇区偏移值*/
    WORD count          /* Byte count (bit15:destination) 读出的数据个数*/
```

这个函数的主要要求：在指定的扇区上（1 个扇区 512 个字节），读取上去上的数据，并将读到的数据存入缓存区，以便后面文件系统对其的调用。

函数的特点是：可以在指定的扇区上读取任何一段区域的数据量，当然这个数据量最大是 512 个字节，所以在程序上我们就分步骤写。下面是我们给大家提供的参考函数。

```
{
    DRESULT res;

    // Put your code here    把你的代码写在这里

    u16 tmp,i;
    u8 r1;
    res=RES_ERROR;

    r1=SD_SendCmd(CMD17,sector<<9,0X01);//读命令

    if(r1==0)//指令发送成功
        {
            while(SPI_SendByte(0xff)!=0xfe);  //等待接收数据
             if(sofs)  //如果有偏移量
             {
             for(i=0;i<sofs;i++)SPI_SendByte(0xff);  //将偏移值循环掉

             for(i=0;i<count;i++) *(dest++)=SPI_SendByte(0xff);  //接收要读出来的数据
             }
             else   //没有偏移值 即偏移值为0
             {
               //要读的数据个数复制给tmp
               for(i=0;i<count;i++) *(dest++)=SPI_SendByte(0xff);  //接收要读出来的数据
             }
             tmp=512-sofs-count;  //剩余为读数据    扇区数-偏移值-读取数据
             for(i=0;i<tmp;i++)SPI_SendByte(0xff);// 将剩余数据循环掉 以免对其他函数有干扰


            //下面是2个伪CRC（dummy CRC）
              SPI_SendByte(0xff);
              SPI_SendByte(0xff);
        }
      else  return res;  //读指令失败

    SD_DisSelect();//取消片选
    res=RES_OK;     //数据读取成功
    return res;
}
```

函数写了很多，其实意思很简单，就是为了读取一个扇区上的任意块区域的数据。如果有对操作不明白就请仔细看程序了，注释写的也比较清楚。

第 3 部 写 SD 卡数据

```
43
44  /*-----------------------------------------------------------------------*/
45  /* Write Partial Sector                                                  */
46  /*-----------------------------------------------------------------------*/
47
48  DRESULT disk_writep (
49      BYTE* buff,     /* Pointer to the data to be written, NULL:Initiate/Finalize write operation */
50      DWORD sc        /* Sector number (LBA) or Number of bytes to send */
51  )
52  {
53      DRESULT res;
54
55
56      if (!buff) {
57          if (sc) {
58
59              // Initiate write process
60
61          } else {
62
63              // Finalize write process
64
65          }
66      } else {
67
68          // Send data to the disk
69
70      }
71
72      return res;
73  }
74
```

disk_writep：写部分扇区，里面只有两个参数，
写入的数据地址* buff 与 第几个扇区 sc 两个数据。

但在编写这个程序的时候要注意，由于 FatFs 内部调用函数的需要，必须按照以下的顺序来，
当 buff 指向一个空指针，当 sc 不为 0 时，则表示对这个扇区的写操作进行初始化；
当 sc 为 0 时，则表示对这个扇区的写操作进行结束操作；
当 buff 指向一个内存缓冲区，则是进行正常的读写。*/

上面是这部分的英文解释，其实主要就是把写入分成了 3 部分，第一是写入前进行准备，第二是写入 512 个字节数据，第三是写入结束。也可以理解为文件系统在写入的时候要提供写入过程中的三个状态的信息。
下面是我们提供的参考代码 其实就是把 SD.C 里的写入扇区拆开了！

```
DRESULT disk_writep (
    BYTE* buff,     /* Pointer to the data to be written, NULL:Initiate/Finalize write operation 指针的数据被写入
    DWORD sc        /* Sector number (LBA) or Number of bytes to send 扇区数（LBA）或要发送的字节数*/
)
{
    DRESULT res;

     u8 r1;
     u16 t;
     u32 m;
    res = RES_ERROR;

    if (!buff) {                        //如果指向空指针
        if (sc) {                       //sc不为0 进行写操作初始化

            // Initiate write process         开始写过程
                r1=SD_SendCmd(CMD24,sc>>9,0X01);
                if(r1!=0) return res;  //应答不正确
                do
                {
                  if(SPI_SendByte(0XFF)==0XFF)break;//OK    放空数据等待SD卡准备好
                    m++;
                 }while(m<0XFFFFFF);//等待

                SPI_SendByte(0XFE);         //发开启符
                res=RES_OK;

        } else {                            //sc为0  表示写操作结束

            // Finalize write process          最后写过程

                    //下面是2个伪CRC（dummy CRC）
                    SPI_SendByte(0xFF);
                    SPI_SendByte(0xFF);
                    r1 = SPI_SendByte(0xFF);

                    if( (r1&0x1f) != 0x05)//等待SD卡应答
                    {
                        SD_DisSelect();
                        return res;
                    }
                    //等待操作完
                    while(!SPI_SendByte(0xFF));

                    SD_DisSelect();
                    res = RES_OK;


        }
    } else {

        // Send data to the disk              发送数据到磁盘
            for(t=0;t<512;t++)SPI_SendByte(*buff++);//提高速度,减少函数传参时间
            res = RES_OK;


    }

    return res;
}
```

到这里就把接口函数的这部分彻底讲完，如果有不明白的地方请仔细阅读程序。

B pff.h 配置

这个的具体配置 和各项的意思在上面已经介绍了 下面直接截程序的配置图

```
/----------------------------------------------------------------------------*/
#ifndef _FATFS
#define _FATFS

#define _USE_READ   1   /* 1:Enable pf_read()
                           有0 和1 两个值可被选择，
                           选择 0 则不使能文件读操作，
                           选择1 则使能读文件操作，在这里我们选择 1，使能读文件操作*/

#define _USE_DIR    1   /* 1:Enable pf_opendir() and pf_readdir()
                           为1 时使能打开一个目录和读一个目录项操作，
                           为 0 时则不使能相应操作，
                           因为我们只测试文件的读/写操作，所以设置其值为 0。 */

#define _USE_LSEEK  1   /* 1:Enable pf_lseek()
                           有 0 和1 两个选择项。
                           1 时使能移动读/写指针操作，为 0 时则禁止。 */

#define _USE_WRITE  0   /* 1:Enable pf_write()
                            为 1 时使能写文件操作，
                            为 0 时禁止写文件操作。
                            这里设置其值为1，使能写文件操作。
                            */

#define _FS_FAT12   1   /* 1:Enable FAT12 support */
#define _FS_FAT32   1   /* 1:Enable FAT32 support
                           为 0 时仅支持FAT16 文件系统，
                           为1 时支持 FAT32 文件系统。
                           FAT32 文件系统是FAT16 文件系统的升级，
                           而且现在 SD 卡在出厂时一般都默认被格式化为FAT32 文件系统，所以设置其值为 1。
                             */
```

```
#define _CODE_PAGE  1
/* Defines which code page is used for path name. Supported code pages are:
/  932, 936, 949, 950, 437, 720, 737, 775, 850, 852, 855, 857, 858, 862, 866,
/  874, 1250, 1251, 1252, 1253, 1254, 1255, 1257, 1258 and 1 (ASCII only).
/  SBCS code pages except for 1 requiers a case conversion table. This
/  might occupy 128 bytes on the RAM on some platforms, e.g. avr-gcc.

/ pdefines代码页用来路径名。支持的代码页:
932 936 949 950 437 720 737 775 850 852 855 857 858 862 866。
874, 1250, 1251, 1252, 1253, 1254, 1255, 1257, 1258和1（纯ASCII）。
/ SBCS代码页，除1例转换表的要求。这
也许在某些平台上占用128个字节的RAM，例如AVR单片机GCC。 */


#define _WORD_ACCESS    0    //有0 和1 两个值可被选择。为 0 时选择字节寻址方式，为1 时选择字寻址方式
/* The _WORD_ACCESS option defines which access method is used to the word
/  data in the FAT structure.
/
/   0: Byte-by-byte access. Always compatible with all platforms.
/   1: Word access. Do not choose this unless following condition is met.
/
/  When the byte order on the memory is big-endian or address miss-aligned
/  word access results incorrect behavior, the _WORD_ACCESS must be set to 0.
/  If it is not the case, the value can also be set to 1 to improve the
/  performance and code efficiency.
```

上面的中午注释很多都是我们拿翻译软件编过来的，可能会有错误，请理解！

总结

到这里 Petit Fatfs 文件系统的移植就介绍完了。

其实主要就是以下 4 个步骤：

1 下载文件系统文件压缩包，这个我们资料里有提供，直接用就可以；

2 将文件系统导入到自己的工程中，并添加其.C .H 文件

3 填写对应的接口函数，这个可以参考我们的程序

4 配置 pff.h 里的功能（主要就是看你用那个功能，就开哪个就可以）！