MP3+MicroSD+FATFS（SPI2）实验

作者	fire
E-Mail	firestm32@foxmail.com
QQ	313303034
博客	firestm32.blog.chinaunix.net
硬件平台	野火 STM32 开发板
库版本	ST3.0.0

实验描述：将 MicroSD 卡(以文件系统 FATFS 访问)里面的 mp3 文件通过 VS1003B 解码，然后将解码后的数据送到功放 TDA1308 后通过耳机播放出来。

硬件连接：PB13-SPI2_SCK ：VS1003B-SCLK
PB14-SPI2_MISO：VS1003B-SO
PB15-SPI2_MOSI：VS1003B-SI
PB12-SPI2_NSS ：VS1003B-XCS
PB11 ：VS1003B-XRET
PC6 ：VS1003B-XDCS
PC7 ：VS1003B-DREQ

库文件 ：startup/start_stm32f10x_hd.c
CMSIS/core_cm3.c
CMSIS/system_stm32f10x.c
FWlib/stm32f10x_gpio.c
FWlib/stm32f10x_rcc.c
FWlib/stm32f10x_usart.c
FWlib/stm32f10x_sdio.c
FWlib/stm32f10x_dma.c

FWlib/stm32f10x_spi.c

FWlib/misc.c

用户文件: USER/main.c

USER/stm32f10x_it.c

USER/sdcard.c

USER/diakio.c

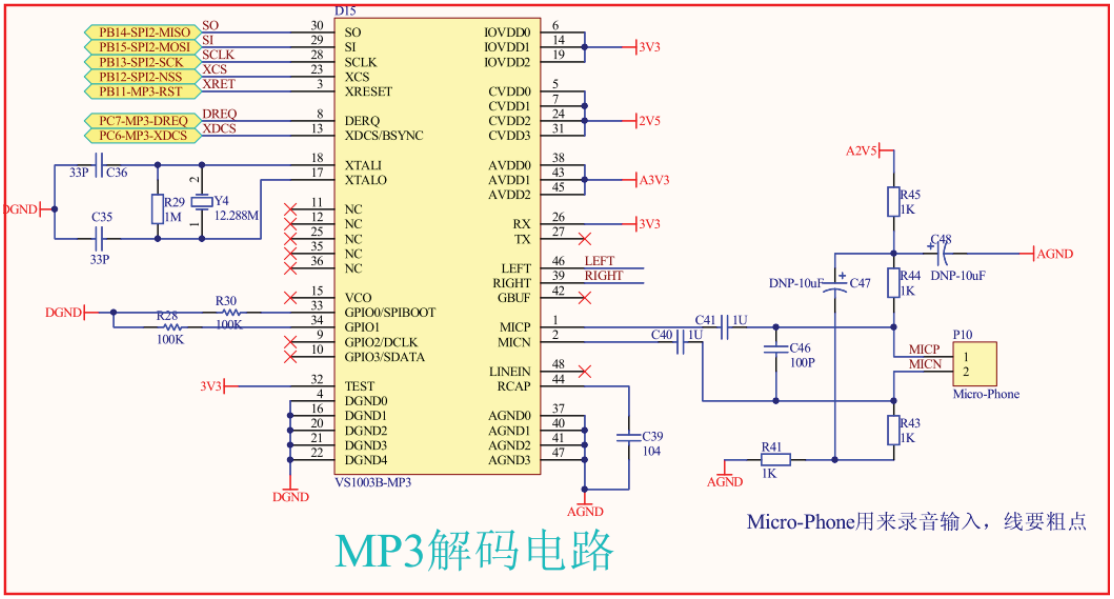
USER/ff.c

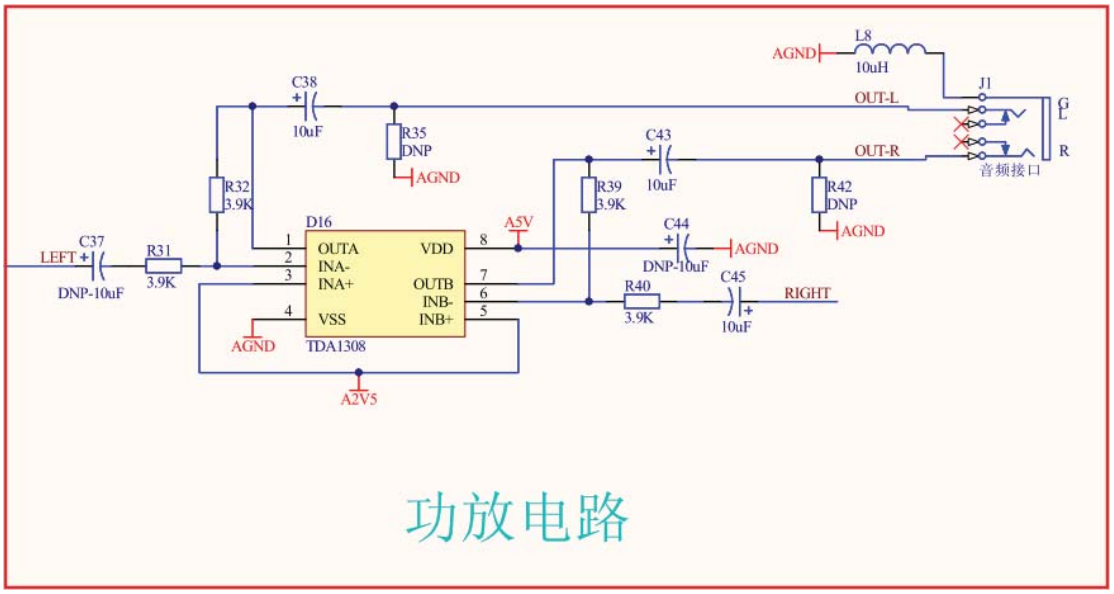
USER/usart1.c

USER/vs1003.c

USER/SysTick.c

野火 STM32 开发板中 MP3 的硬件原理图：





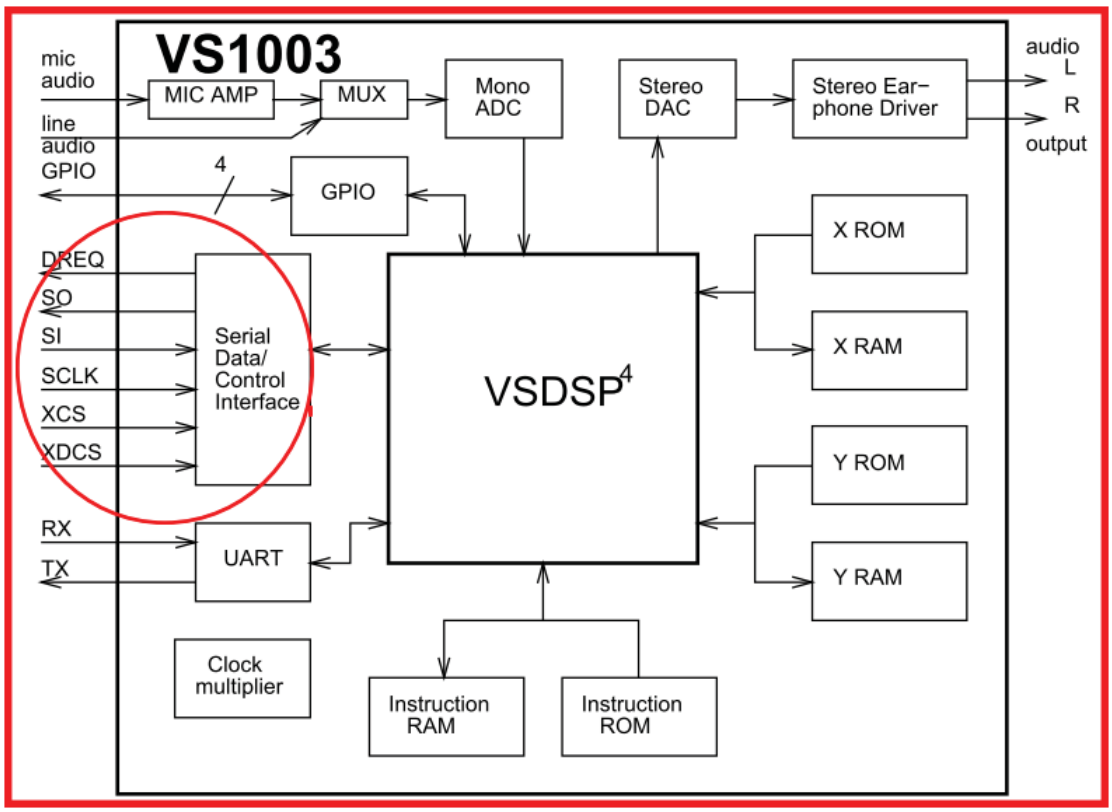
解码部分采用 VS1003-MP3/WMA 音频解码器，然后将解码后的数据送 TDA1308 放大后由音频接口外播出来。

VS1003+TDA1308 简介->

VS1003 是一个单片 MP3/WMA/MIDI 音频解码器和 ADPCM 编码器。它包含一个高性能，自主产权的低功耗 DSP 处理器核 VS_DSP 4 ,工作数据存储器，为用户应用提供 5KB 的指令 RAM 和 0.5KB 的数据 RAM。串行的控制和数据接口，4 个常规用途的 I/O 口，一个 UART，也有一个高品质可变采样率的 ADC 和立体声 DAC，还有一个耳机放大器和地线缓冲器。

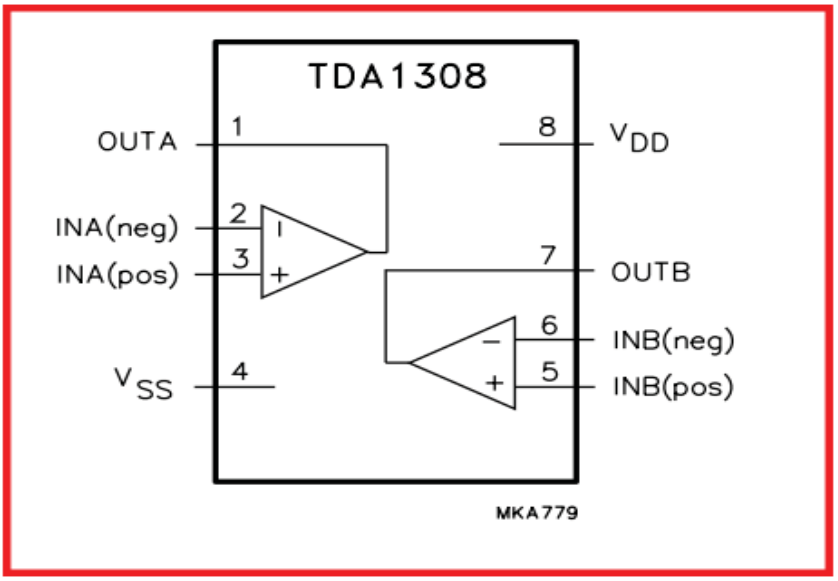
VS1003 通过一个串行接口来接收输入的比特流，它可以作为一个系统的从机。输入的比特流被解码，然后通过一个数字音量控制器到达一个 18 位过采样多位 ε - Δ DAC。通过串行总线控制解码器。除了基本的解码，在用户 RAM 中它还可以做其他特殊应用，例如 DSP 音效处理。

VS1003 原理框图：



本实验中我们只用了红色圆圈中的那几个数据口，这些数据口是串行模式的，我们用到了开发板中的 SPI2 来控制。其中数据经 SI 接口进去，经解码后由 L、R 这两个左右声道引脚出来，因为 VS1003 内部集成了一个 DA，所以出来的数据是模拟的，可直接驱动耳机，但由于功率太小，音效不佳，所以我们将信号送往 TDA1308 放大后再通过耳机外放出来，经过这样出来之后音质跟电脑上的有的比。现在市面上的 MP3 模块基于成本考虑都没加音频功放，而是直接驱动音频耳机，效果可想而知。

TDA1308 是一款双通道的立体耳机驱动器，是一款专门用于声音驱动的功放。其原理框图如右：



有关 VS1003B 和 TDA1308 的详细应用，大家可参考官方的 datasheet，野火就不在这里罗嗦啦^_^。

本实验是在 《MicroSD 卡+FATFS》这个实验基础上进行的。 没做过这个实验的话可参考前面的教程，否则有些代码会让您犯糊涂。

实验讲解->

首先需要将需要用到的库文件添加进来，有关库的配置可参考前面的教程，这里不再详述。在配置好库的环境之后我们从 main 函数开始分析：

```
1.  /*
2.   * 函数名: main
3.   * 描述  : 主函数
4.   * 输入  : 无
5.   * 输出  : 无
6.   */
7.  int main(void)
8.  {
9.      /* 配置系统时钟为 72M */
10.     SystemInit();
11.
12.     /* 配置 SysTick 为 10us 中断一次 */
13.     SysTick Init();
14.
15.     /* 配置串口 1 115200 8-N-1 */
16.     USART1_Config();
17.
18.     /* SD 卡中断配置 */
19.     NVIC_Configuration();
20.
21.     USART1_printf( USART1, " \r\n 这是一个 MP3 测试例程 !\r\n " );
22.
23.     /* MP3 硬件初始化 */
24.     VS1003_SPI_Init();
25.
26.     /* SD 卡硬件初始化，并初始化盘符为 0 */
27.     disk initialize( 0 );
28.
29.     /* MP3 就绪，准备播放 */
30.     MP3_Start();
31.
32.     /* 播放 SD 卡(FATFS)里面的音频文件 */
33.     MP3_Play();
34.
35.   while (1)
36.   {
37.   }
38.
39. } /* end of main */
```

库函数 `SystemInit();`将我们的系统时钟设置为 72MHZ，在所有工作之前首先要做的就是先设置系统时钟，这可千万别忘了。在 ST3.0.0 版本之后的库中，这部分工作都放在了启动文件中了，由汇编实现，只要用户代码一进入 main 函数就表示已经初始化好系统时钟了，完全不用用户考虑，用户不知道这点的话还以为不需要初始化系统

时钟呢。但我们这里用的库版本是 ST3.0.0，所以还需要调用库函数 SystemInit();来初始化我们的系统时钟。至于 ST3.0.0 和之后高版本的库有什么区别，我想说的是没什么大的区别，代码的目录结构基本没有改变，只是在代码的功能增多了，支持更完善的外设。

SysTick 为 10us 中断一次用于 SysTick 为 10us 中断一次，用于后面的延时函数。

USART1_Config(); 配置串口 1 波特率为 115200 ，8 个数据位，1 个停止位，无硬件流控制。

NVIC_Configuration(); 用于配置 MicroSD 卡的中断优先级。

VS1003_SPI_Init();用于初始化 MP3 解码芯片 VS1003B 需要用到的 I/O 口，包括数据口(SPI2)和控制 I/O。VS1003_SPI_Init();由用户在 vs1003.c 中实现：

```
1.  /*
2.   * 函数名：VS1003 SPI Init
3.   * 描述   ：VS1003 所用 I/O 初始化
4.   * 输入   ：无
5.   * 输出   ：无
6.   * 调用   ：外部调用
7.   */
8.  void VS1003_SPI_Init(void)
9.  {
10.   SPI_InitTypeDef  SPI_InitStructure;
11.   GPIO_InitTypeDef GPIO_InitStructure;
12.
13.   /* 使能 VS1003B 所用 I/O 的时钟 */
14.     RCC APB2PeriphClockCmd(RCC APB2Periph GPIOA |RCC APB2Periph GPIOB | RCC APB
    2Periph_GPIOC , ENABLE);
15.     /* 使能 SPI2 时钟 */
16.   RCC APB1PeriphClockCmd(RCC APB1Periph SPI2 ,ENABLE);
17.
18.   /* 配置 SPI2 引脚：PB13-SCK, PB14-MISO 和 PB15-MOSI */
19.   GPIO_InitStructure.GPIO_Pin = GPIO_Pin_13 | GPIO_Pin_14 | GPIO_Pin_15;
20.   GPIO_InitStructure.GPIO_Speed =GPIO_Speed_50MHz;
21.   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
22.   GPIO Init(GPIOB, &GPIO InitStructure);
23.
24.   /* PB12-XCS(片选) */
25.   GPIO InitStructure.GPIO Pin = GPIO Pin 12;
26.   GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
27.   GPIO InitStructure.GPIO Mode = GPIO Mode Out PP;
28.   GPIO_Init(GPIOB, &GPIO_InitStructure);
29.
30.     /* PB11-XRST(复位) */
31.   GPIO InitStructure.GPIO Pin =  GPIO Pin 11;
32.   GPIO_Init(GPIOB, &GPIO_InitStructure);
33.
34.
35.   /* PC6-XDCS(数据命令选择) */
36.   GPIO InitStructure.GPIO Pin =  GPIO Pin 6;
37.   GPIO_Init(GPIOC, &GPIO_InitStructure);
```

```
38.
39.   /* PC7-DREQ(数据中断) */
40.   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IPD;
41.   GPIO_InitStructure.GPIO_Pin =  GPIO_Pin_7;
42.   GPIO_Init(GPIOC, &GPIO_InitStructure);
43.
44.   /* SPI2 configuration */
45.   SPI_InitStructure.SPI_Direction = SPI_Direction_2Lines_FullDuplex;
46.   SPI_InitStructure.SPI_Mode = SPI_Mode_Master;
47.   SPI_InitStructure.SPI_DataSize = SPI_DataSize_8b;
48.   SPI_InitStructure.SPI_CPOL = SPI_CPOL_Low;
49.   SPI_InitStructure.SPI_CPHA = SPI_CPHA_1Edge;
50.   SPI_InitStructure.SPI_NSS = SPI_NSS_Soft;
51.   SPI_InitStructure.SPI_BaudRatePrescaler = SPI_BaudRatePrescaler_32;
52.   SPI_InitStructure.SPI_FirstBit = SPI_FirstBit_MSB;
53.   SPI_InitStructure.SPI_CRCPolynomial = 7;
54.   SPI_Init(SPI2, &SPI_InitStructure);
55.
56.   /* Enable SPI2  */
57.   SPI_Cmd(SPI2, ENABLE);
58. }
```

假如我们要将数据口换成 SPI1 或者改变其他控制 I/O，只需改变这个函数即可，移植性非常强。

`disk_initialize( 0 );` 用于初始化 MicroSD 卡的底层硬件，并将卡的盘符初始化为 0，盘符的范围为 0~9，用户可选，在这里我们选为 0。

`MP3_Start();` 使 MP3 进入就绪模式(standby)，随时播放音乐。`MP3_Start();`在 vs1003.c 中实现：

```
1.   * 函数名: MP3_Start
2.   * 描述   : 使 MP3 进入就绪模式，随时准备播放音乐。
3.   * 输入   : 无
4.   * 输出   : 无
5.   * 调用   : 外部调用
6.   */
7.  void MP3_Start(void)
8.  {
9.      u8 BassEnhanceValue = 0x00;       // 低音值先初始化为 0
10.     u8 TrebleEnhanceValue = 0x00;     // 高音值先初始化为 0
11.     TRST_SET(0);
12.     Delay_us( 1000 );                 // 1000*10us = 10ms
13.
14.     VS1003_WriteByte(0xff);           // 发送一个字节的无效数据，启动 SPI 传输
15.     TXDCS_SET(1);
16.     TCS_SET(1);
17.     TRST_SET(1);
18.     Delay_us( 1000 );
19.
20.     Mp3WriteRegister( SPI_MODE,0x08,0x00);      // 进入 VS1003 的播放模式
21.     Mp3WriteRegister(3, 0x98, 0x00);            // 设置 vs1003 的时钟,3 倍频
22.     Mp3WriteRegister(5, 0xBB, 0x81);            // 采样率 48k，立体声
23.     // 设置重低音
24.     Mp3WriteRegister(SPI_BASS, TrebleEnhanceValue, BassEnhanceValue);
25.     Mp3WriteRegister(0x0b,0x00,0x00);    // VS1003 音量
26.     Delay_us( 1000 );
27.
28.     while( DREQ == 0 );                 // 等待 DREQ 为高   表示能够接受音乐数据输入
29. }
```

函数中涉及到的宏定义都在 vs1003.h 这个头文件中实现。关于函数中为什么要这样操作寄存器，或者为什么要按照这个顺序来操作寄存器，请大家查阅 vs1003 的 pdf，里面讲得很详细，有 e 文跟中文资料。

`MP3_Play();` 这个函数逐个扫描我们卡里面的音频文件，并将这些音频文件都通过耳机播放出来，最后停止，暂时还没实现循环播放的功能。`MP3_Play();`在 vs1003.c 中实现：

```
/*
 * 函数名：MP3_Play
 * 描述  ：读取 SD 卡里面的音频文件，并通过耳机播放出来
 *         支持的格式：mp3,mid,wav,wma
 * 输入  ：无
 * 输出  ：无
 * 说明  ：暂不支持长文件名，不支持中文。
 */
void MP3_Play(void)
{
  FRESULT res;
  FILINFO finfo;
  DIR dirs;
  u16 count = 0;
    char j = 0;
  char path[50] = {""};
  char *result1, *result2, *result3, *result4;

  f_mount(0, &fs);                                        /* 挂载文件系统到 0 区 */

    if (f_opendir(&dirs, path) == FR_OK)                  /* 打开根目录 */
  {
    while (f_readdir(&dirs, &finfo) == FR_OK)          /* 依次读取文件名 */
    {
      if ( finfo.fattrib & AM_ARC )                   /* 判断是否为存档型文档 */
      {
        if( !finfo.fname[0] )                                 /* 文件名为空即到达了目录的末尾，退出 */
          break;
        USART1_printf( USART1, " \r\n the music file name is: %s \r\n", finfo.fname );

                result1 = strstr( finfo.fname, ".mp3" );     /* 判断是否为音频文件 */
                result2 = strstr( finfo.fname, ".mid" );
                result3 = strstr( finfo.fname, ".wav" );
                result4 = strstr( finfo.fname, ".wma" );

                if ( result1!=NULL || result2!=NULL || result3!=NULL || result4!=NULL )
                {
            res = f_open( &fsrc, finfo.fname, FA_OPEN_EXISTING | FA_READ ); /* 以只读方式打开 */
            br = 1;                                 /* br 为全局变量 */
                    TXDCS_SET( 0 );                /* 选择 VS1003 的数据接口  */
/* ------------------- 一曲开始 --------------------*/
          USART1_printf( USART1, " \r\n 开始播放 \r\n" );
                    for (;;)
                    {
                res = f_read( &fsrc, buffer, sizeof(buffer), &br );
            if ( res == 0 )
                            {
                                    count = 0;                                    /* 512 字节完重新计数 */
```

```
                                    Delay_us( 1000 );          /* 10ms 延时 */
                                    while ( count < 512)           /* SD 卡读取一个
    sector，一个 sector 为 512 字节 */
                                    {
                              if ( DREQ != 0 )           /* 等待 DREQ 为高，请求数据输
    入 */
                                        {
                                            for (j=0; j<32; j++ ) /* VS1003 的 FIFO
    只有 32 个字节的缓冲 */
                                            {

                                                VS1003_WriteByte( buffer[count] );

                                                count++;
                                            }
                                        }
                                    }
                    }
                              if (res || br == 0) break;    /* 出错或者到了文件
    尾 */
                }
                          USART1_printf( USART1, " \r\n 播放结束 \r\n" );
    /* ------------------- 一曲结束 ---------------------*/
                         count = 0;
                          /* 根据 VS1003 的要求，在一曲结束后需发送 2048 个 0 来确保下一首的正
    常播放 */
                         while ( count < 2048 )
                         {
                     if ( DREQ != 0 )
                                 {
                                    for ( j=0; j<32; j++ )
                               {
                                        VS1003_WriteByte( 0 );

                                        count++;
                               }
                               }
                    }
                          count = 0;
                          TXDCS_SET( 1 );  /* 关闭 VS1003 数据端口 */
                         f_close(&fsrc);    /* 关闭打开的文件 */
                     }
           }
         } /* end of while */
       }
    } /* end of MP3_Play */
```

由于代码比较长，在格式编排上不是很好，野火建议大家还是配合源代码一起阅读^_^。

现在我们来大概分析下 `MP3_Play();` 这个函数，这里边涉及到一些文件系统操作的函数，关于这部分函数的操作大家可参考前面的教程或者阅读 FATFS 的官方文档，其实我的教程也不完全正确，阅读官方的文档才是最可靠的。

函数 `f_mount(0, &fs);` 为我们在文件系统中注册一个工作区，并初始化盘符的名为 0。

函数 `f_opendir(&dirs, path)` 用于打开卡的根目录，并将这个根目录关联到 dirs 这个结构指针，然后我们就可以通过这个结构指针来操作这个目录了，其实这个结构指

针就类似 LINUX 下系统编程中的文件描述符，不论是操作还是目录都得通过文件描述符才能操作。

`f_readdir(&dirs, &finfo)` 函数通过刚刚的 dirs 结构指针来读取目录里面的信息，并将目录的信息储存在 `finfo` 这个结构体变量中。

紧接着判断文件的属性，如果是存档型文件的话就将文件名打印出来，然后比较文件的后缀名，查看是否为音频文件，支持的音频格式有 mp3、mid、wav、wma。

如果是音频文件的话则调用 `f_open( &fsrc, finfo.fname, FA_OPEN_EXISTING | FA_READ );`打开这个音频文件。

`TXDCS_SET( 0 );` 用于选择 vs1003 的数据端口，准备往 vs1003 中输入数据。其中 `TXDCS_SET( 0 );`是在 vs1003.h 中实现的一个宏：

```
1.  #define XDCS    (1<<6)  // PC6-XDCS
2.
3.  #define TXDCS_SET(x)  GPIOC->ODR=(GPIOC->ODR&~XDCS)|(x ? XDCS:0)
```

紧接着进入一个大循环中播放我们的 mp3 文件。

函数 `f_read( &fsrc, buffer, sizeof(buffer), &br );`从文件中读取 512 个字节的数据到缓冲区中，至于为什么是 512 个字节，而不是 1024 或者更多，这是因为卡的一个 sector 是 512 个字节，一次只能读取一个 sector。

函数 `VS1003_WriteByte( buffer[count] );`将缓冲区中的数据写入 vs1003 的数据缓冲区。注意，这里一次只能写入 32 个字节，这是因为 vs1003 的 FIFO 的大小为 32 个字节，写多了无效。

当文件出错或者一曲播放完毕时就跳出 for 循环，并打印出“播放结束”的调试信息。

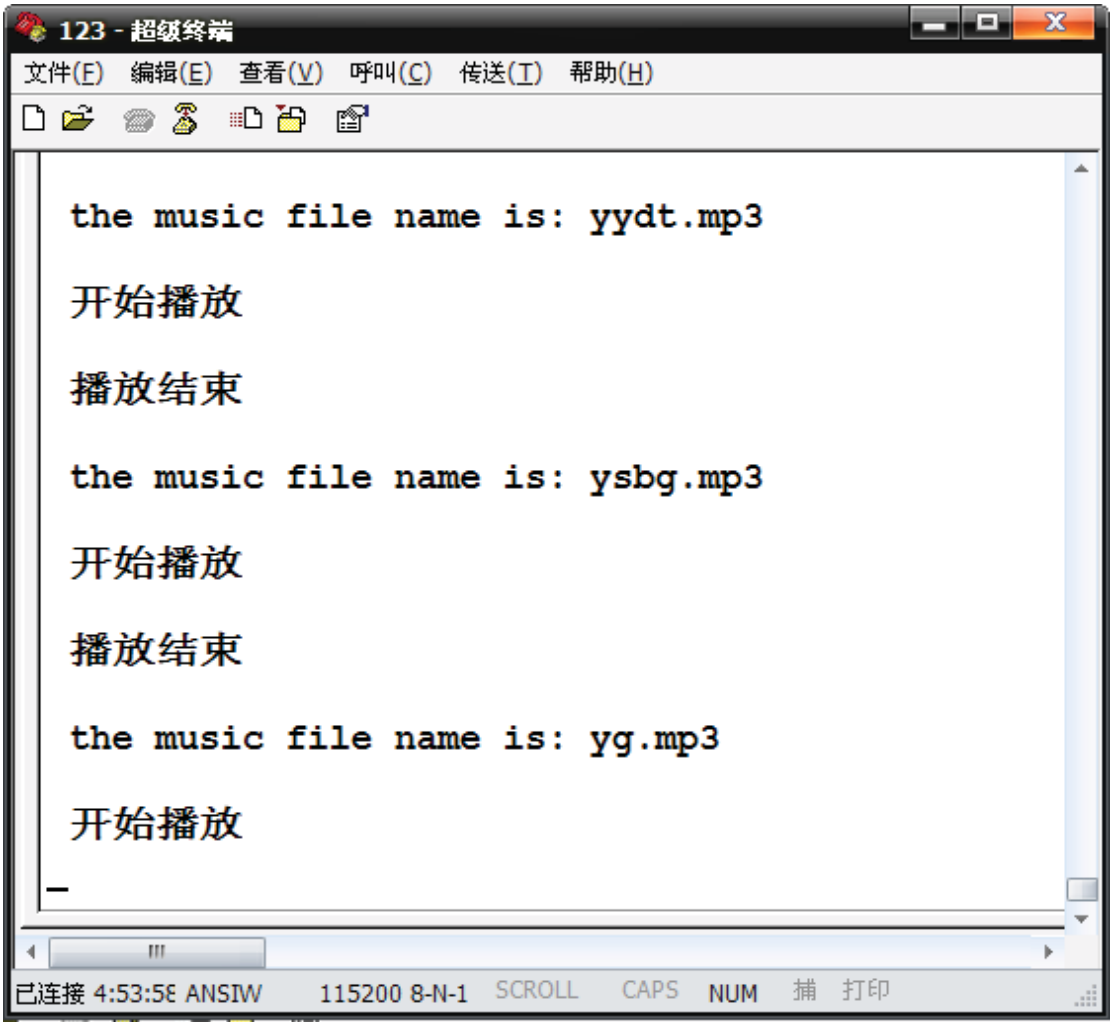
根据 VS1003 的要求，在一曲结束后需发送 2048 个 0 来确保下一首的正常播放。

一曲播放完毕我们关闭 vs1003 的数据端，关闭打开的文件，等待下一曲的播放，直到目录下的音频文件播放完为止。还暂不支持循环播放的功能。

这里面涉及到了 vs1003 操作的一些特性，需大家参考 vs1003 的 datasheet 来帮助理解。

实验现象->

将野火 STM32 开发板供电(DC5V)，插上 JLINK，插上串口线(两头都是母的交叉线)，插上 MicroSD 卡(我用的是 1G)，在卡的根目录下要有 mp3 文件，文件名要是英文，暂不支持中文和长文件名，打开超级终端，配置超级终端为 115200 8-N-1，将编译好的程序下载到开发板，即可看到超级终端打印出如下信息：



我的卡的根目录下放了 3 个 mp3 文件，都是 Eason 的歌(因为个人非常喜欢陈奕迅的歌^_^)，分别是遥远的她(yydt.mp3)、一丝不挂(ysbg.mp3)、预感(yg.mp3)。插上耳机，音质堪比电脑，音量可通过耳机来调，前提是你的耳机要能调节音量才行。

实验讲解完毕，野火祝大家学习愉快^_^。