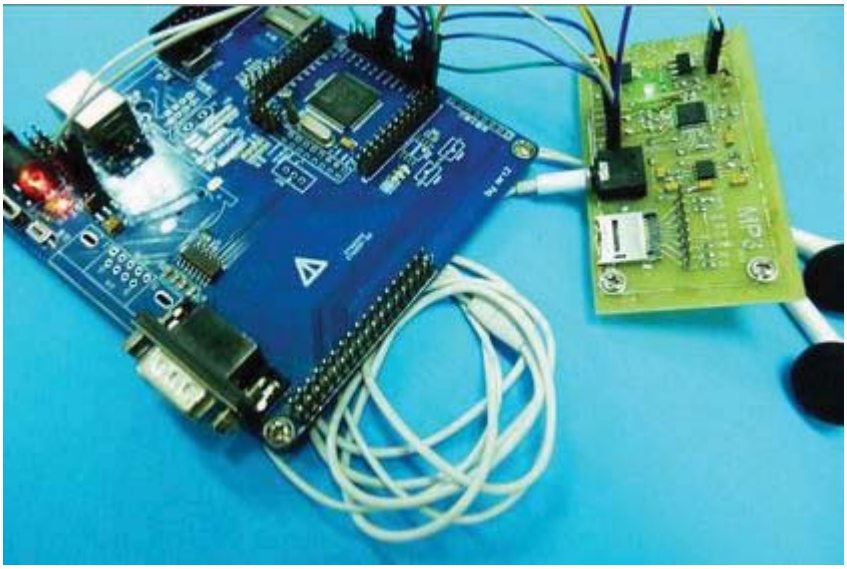
基于 STM32(ARMV7)的 MP3 播放器

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，并在 LCD 中显示曲目和艺术家，支持中英文显示。这部分需要 LCD 的支持，有关 LCD 的操作请查看 LCD 的那部分 pdf 教程。

注：野火 **stm32** 开发板中板载的 **mp3** 跟这里的接口是一样的。

纯手工打造 **MP3**，**VS1003** 硬解码，外加立体耳机功放 **TDA1308**，音质堪比电脑。



MP3 与开发板的接线

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

FWlib/fsmc.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

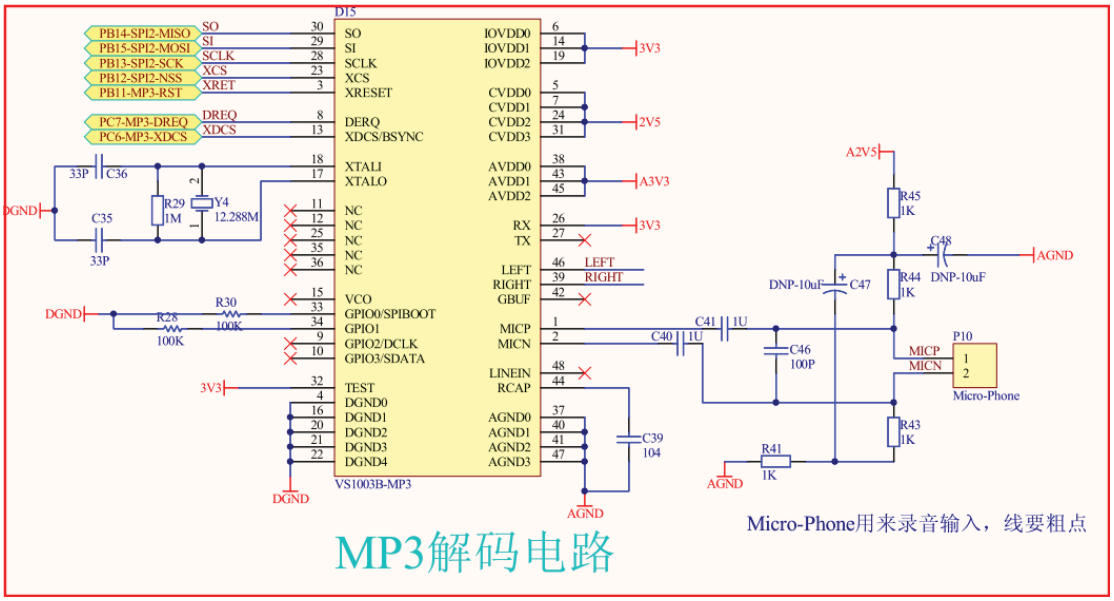
USER/lcd.c

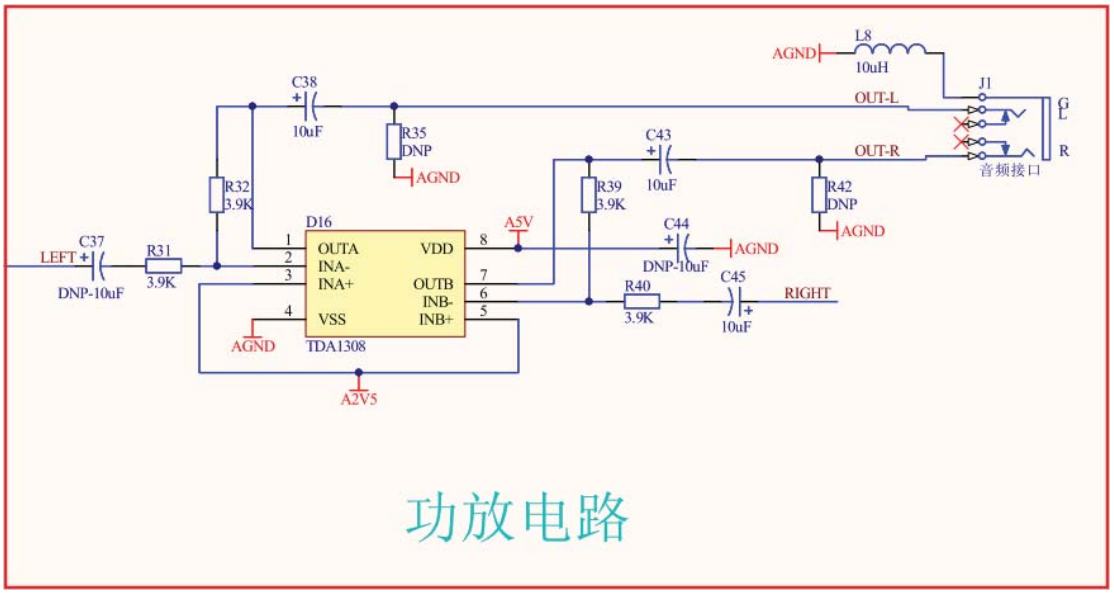
USER/sd_fs_app.c

USER/sd_bmp.c

USER/mp3play.c

野火 **STM32** 开发板中 **MP3** 的硬件原理图：





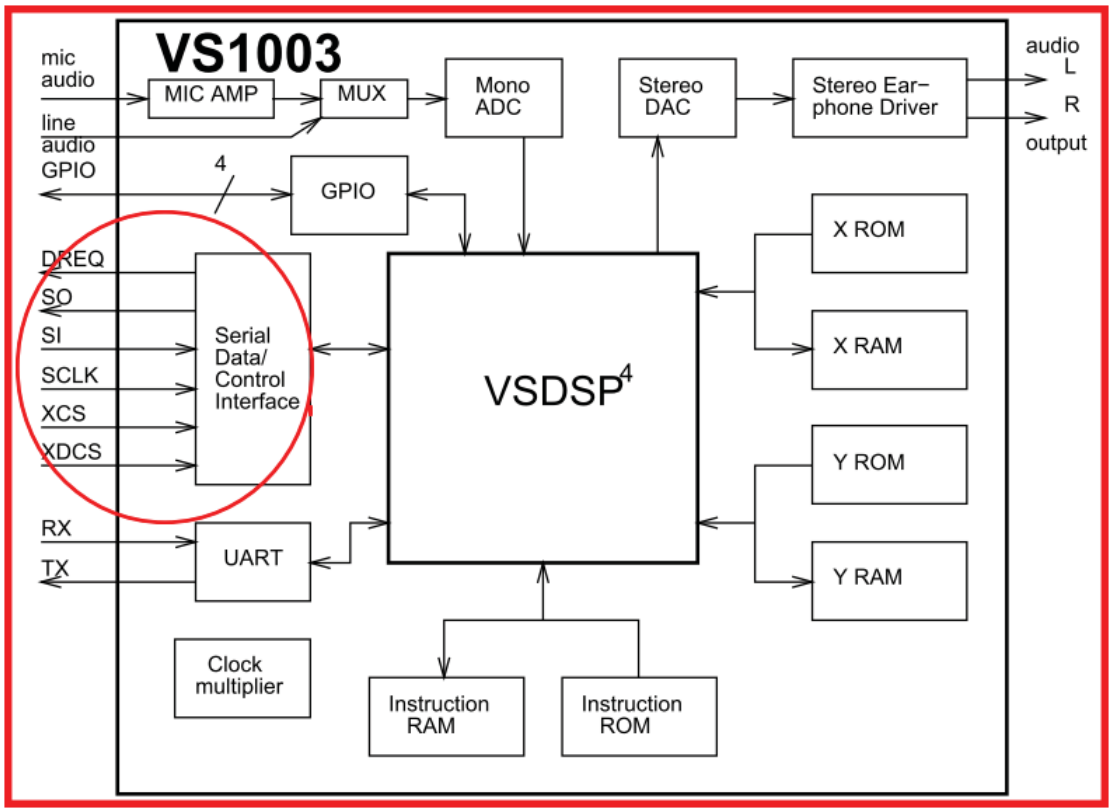
解码部分采用 VS1003-MP3/WMA 音频解码器，然后将解码后的数据送 TDA1308 放大后由音频接口外播出来。

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。

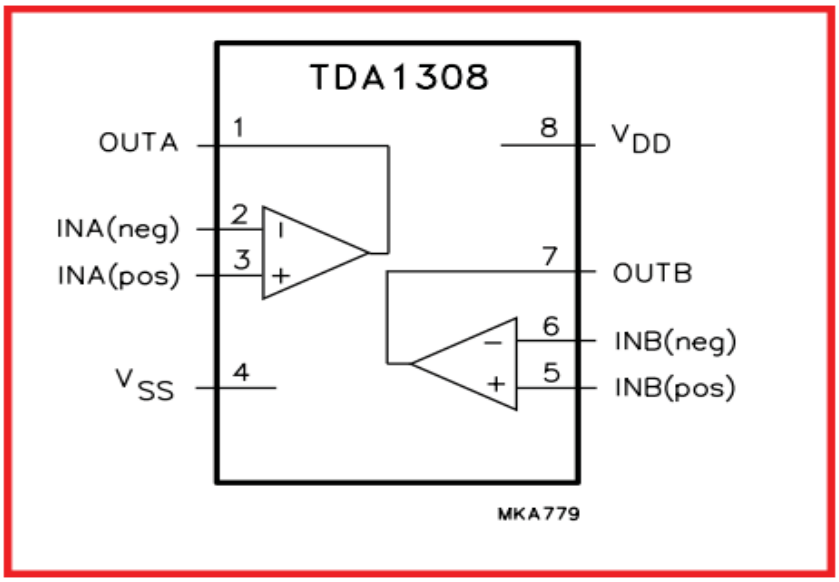
VS1003 通过一个串行接口来接收输入的比特流，它可以作为一个系统的从机。输入的比特流被解码，然后通过一个数字音量控制器到达一个 18 位过采样多位 ε - Δ DAC。通过串行总线控制解码器。除了基本的解码，在用户 RAM 中它还可以做其他特殊应用，例如 DSP 音效处理。

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。其原理框图如下：



有关 VS1003B 和 TDA1308 的详细应用，大家可参考官方的 datasheet。

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。

----------MP3 知识扫盲----------

在讲解实验之前，先大概了解下 MP3 的知识。

- MP3 文件概述->

MP3 文件是由帧(frame)构成的，帧是 MP3 文件最小的组成单位。MP3 的全称为 MPEG1Layer-3 音频文件，MPEG(Moving Pictures Experts Group)在汉语中翻译为活动图像专家组，特指活动影音压缩标准，MPEG 音频文件是 MPEG1 标准中的声音部分，也叫 MPEG 音频层，它根据压缩质量和编码复杂程度划分为三层，即 Layer1、Layer2、Layer3，且分别对应 MP1、MP2、MP3 这三种音频文件，并根据不同的用途，使用不同层次的编码。MPEG 音频编码的层次越高，编码器越复杂，压缩率也越高，MP1 和 MP2 的压缩率分别为 4：1 和 6：1-8：1，而 MP3 的压缩率则高达 10：1-12：1。不过 MP3 对音频信号采用的是有损压缩方式，未来降低声音失真度，MP3 采用了“感官编码技术”，即编码时先对音频文件进行频谱分析，然后用过滤器滤掉噪音电平，接着通过量化的方式将剩下的每一位打散排列，也就是 Huffman 无损压缩编

码，最后形成具有较高压缩比的 MP3 文件，并使压缩后的文件在回放时能够达到比较接近原音的声效。

- MP3 文件组成->

MP3 文件大体分为 3 部分:TAG_V2(ID3V2),Frame,TAG_V1(ID3V1)。要注意有些 MP3 文件尾是没有 ID3V1 tag 的，有的是 MP3 文件头部的 ID3V2 tag。 其中 ID3V1 存放在 MP3 文件的末尾 ，用 16 进制的编辑器打开一个 MP3 文件， 查看其末尾的 128 个顺序存放字节，数据结构定义如下：

```
1.  typedef struct tagID3V1
2.  {
3.      char Header[3];       // 标签头必须是"TAG"，否则被认为没有标签头
4.      char Title[30];       // 标题
5.      char Artist[30];      // 作者
6.      char Album[30];       // 专辑
7.      char Year[4];         // 出品年代
8.      char Comment[28];     // 备注
9.      char reserve;         // 保留
10.     char track;           // 音轨
11.     char Genre;           // 风格
12. }ID3V1, *pID3V1;
```

现在很多 MP3 文件有的只是 ID3V2，基本上都没有了尾部的 ID3V1 了，有 ID3V1 的是比较老的歌曲，野火发现像 beyond 的歌都含有 ID3V1 tag ，但信息也不如上面的结构体中描述得多，一般有的是 title 和 artist。这在接下来的代码中可看到 ID3V1 的应用。

ID3V1 的各项都是顺序存放，没有任何标志将其分开，比如标题信息不足 30 个字节，则使用'\0'补足，否则将造成信息错误。

1、 每帧的播放时间：无论帧长是多少，每帧的播放时间是 26ms。

2、 数据帧大小的计算：

```
3、*1、每帧的播放时间：无论帧长是多少，每帧的播放时间是 26ms。
4、  *2、
        FrameSize = ( (( MPEGVersion == MPEG1 ? 144 : 72 ) * Bitrate) / SampleRate
         ) + PaddingBit
5、  */
```

◆ **ID3V2**：包含了作者，作曲，专辑等信息，长度不固定，扩展了 ID3V1 的信息量。

◆ **Frame**:一系列的帧，个数由文件大小和帧长决定。每个 frame 的帧长可能固定，也可能不固定，由位率 bitrate 决定。每个帧又分为帧头和数据实体两部分。帧头记录了 mp3 的位率，采样率，版本等信息，每个帧之间相互独立。

◆ **ID3V1**：包含了作者，作曲。专辑等信息，长度为 **128 byte**。

每个 MP3 数据帧有一个帧头 FRAMHEADER，长度是 4 个 byte(32bit)，帧头后面可能有两个字节的 CRC 校验码，这两个字节的是否存在决定于 FRAMHEADER 的第 16bit，为 0 则帧头后面没校验，为 1 则有校验，校验值长度为 2 个字节，紧跟在 FRAMHEADER 后面，接着就是帧的实体数据了，格式如下：

FRAMHEADER	CRC(free)	MAIN_DATA
4byte	0 or 2 byte	长度由帧头计算得出

关于 MP3 的知识野火先讲解到这，其他的稍后再讲解，下面我们来具体看看源码的实现。

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.     /* LED 初始化，用于调试 */
19.     LED_GPIO_Config();
20.
21.     /* 2.4TFT 初始化 */
22.     LCD_Init();
23.
24.     /* 文件系统初始化-----汉字字库保存在 sd 卡中 */
25.     sd_fs_init();
26.

27.   Set_direction(0);         /* 设置 LCD 的扫描方向，这里为垂直扫描*/
28.
29.   LCD CLEAR(0,0,240,320);
30.     //Lcd show bmp(0, 0,"/PIC.bmp");
31.
32.     USART1_printf( USART1, " \r\n 这是一个 MP3 测试例程 !\r\n " );
33.
34.     /* MP3 硬件 I/O 初始化 */
35.     VS1003_SPI_Init();
36.
37.     /* SD 卡硬件初始化，并初始化盘符为 0 */
38.     //disk initialize( 0 );
39.
40.     /* MP3 就绪，准备播放，在 vs1003.c 实现 */
41.     MP3 Start();
42.
43.     LED1(ON);
44.
45.     /* 播放 SD 卡(FATFS)里面的音频文件 */
46.     MP3_Play();
47.
48.   while (1)
49.   {
50.   }
51. } /* 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，所以还是需要调用库函数 `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。

`LCD_Init();` 用于初始化我们的 2.4TFT，我们将用它来显示我们的 MP3 歌曲的信息。

sd_fs_init();用于初始化 MicroSD 卡底层硬件，并将盘符初始化为 0，还设置了 MicroSD 卡的中断优先级，这个函数里面调用了 SDIO_NVIC_Configuration(); 和 disk_initialize(0); 这两个函数。调用 sd_fs_init();函数是非常重要的，不然 MicroSD 卡是工作不了的，文件系统根本跑不起来，MP3 文件根本就读不出来。

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。

接下来的几个函数大伙看注释吧，比较简单，野火就不再啰嗦浪费大家的时间啦 ^_^。

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.   * 函数名: VS1003_SPI_Init
2.   * 描述   : VS1003 所用 I/O 初始化
3.   * 输入   : 无
4.   * 输出   : 无
5.   * 调用   : 外部调用
6.   */
7.  void VS1003_SPI_Init(void)
8.  {
9.    SPI_InitTypeDef  SPI_InitStructure;
10.   GPIO_InitTypeDef GPIO_InitStructure;
11.
12.   /* 使能 VS1003B 所用 I/O 的时钟 */
13.     RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA |RCC_APB2Periph_GPIOB | RCC_APB
    2Periph GPIOC , ENABLE);
14.     /* 使能 SPI2 时钟 */
15.   RCC APB1PeriphClockCmd(RCC APB1Periph SPI2 ,ENABLE);
16.
17.   /* 配置 SPI2 引脚: PB13-SCK, PB14-MISO 和 PB15-MOSI */
18.   GPIO_InitStructure.GPIO_Pin = GPIO_Pin_13 | GPIO_Pin_14 | GPIO_Pin_15;
19.   GPIO_InitStructure.GPIO_Speed =GPIO_Speed_50MHz;
20.   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP;
21.   GPIO Init(GPIOB, &GPIO InitStructure);
22.
23.   /* PB12-XCS(片选) */
24.   GPIO InitStructure.GPIO Pin = GPIO Pin 12;
25.   GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;
26.   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
27.   GPIO_Init(GPIOB, &GPIO_InitStructure);
28.
29.     /* PB11-XRST(复位) */
30.   GPIO InitStructure.GPIO Pin =  GPIO Pin 11;
31.   GPIO_Init(GPIOB, &GPIO_InitStructure);
32.
33.
34.   /* PC6-XDCS(数据命令选择) */
35.   GPIO_InitStructure.GPIO_Pin =  GPIO_Pin_6;
36.   GPIO_Init(GPIOC, &GPIO_InitStructure);
37.
38.   /* PC7-DREQ(数据中断) */
39.   GPIO InitStructure.GPIO Mode = GPIO Mode IPD;
40.   GPIO_InitStructure.GPIO_Pin =  GPIO_Pin_7;
41.   GPIO Init(GPIOC, &GPIO InitStructure);
42.
43.   /* SPI2 configuration */
44.   SPI_InitStructure.SPI_Direction = SPI_Direction_2Lines_FullDuplex;
45.   SPI_InitStructure.SPI_Mode = SPI_Mode_Master;
```

```
46.   SPI_InitStructure.SPI_DataSize = SPI_DataSize_8b;
47.   SPI_InitStructure.SPI_CPOL = SPI_CPOL_Low;
48.   SPI_InitStructure.SPI_CPHA = SPI_CPHA_1Edge;
49.   SPI_InitStructure.SPI_NSS = SPI_NSS_Soft;
50.   SPI_InitStructure.SPI_BaudRatePrescaler = SPI_BaudRatePrescaler_32;
51.   SPI_InitStructure.SPI_FirstBit = SPI_FirstBit_MSB;
52.   SPI_InitStructure.SPI_CRCPolynomial = 7;
53.   SPI_Init(SPI2, &SPI_InitStructure);
54.
55.   /* Enable SPI2  */
56.   SPI_Cmd(SPI2, ENABLE);
57. }
```

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.  /*
2.   * 函数名：MP3_Start
3.   * 描述   ：使 MP3 进入就绪模式，随时准备播放音乐。
4.   * 输入   ：无
5.   * 输出   ：无
6.   * 调用   ：外部调用
7.   */
8.  void MP3_Start(void)
9.  {
10.     u8 BassEnhanceValue = 0x00;       // 低音值先初始化为 0
11.     u8 TrebleEnhanceValue = 0x00;     // 高音值先初始化为 0
12.     TRST_SET(0);
13.     Delay_us( 1000 );                 // 1000*10us = 10ms
14.
15.     VS1003_WriteByte(0xff);           // 发送一个字节的无效数据，启动 SPI 传输
16.     TXDCS_SET(1);
17.     TCS_SET(1);
18.     TRST_SET(1);
19.     Delay_us( 1000 );
20.
21.     Mp3WriteRegister( SPI_MODE,0x08,0x00);      // 进入 VS1003 的播放模式
22.     Mp3WriteRegister(3, 0x98, 0x00);            // 设置 vs1003 的时钟,3 倍频
23.     Mp3WriteRegister(5, 0xBB, 0x81);            // 采样率 48k，立体声
24.     // 设置重低音
25.     Mp3WriteRegister(SPI_BASS, TrebleEnhanceValue, BassEnhanceValue);
26.     Mp3WriteRegister(0x0b,0x00,0x00);    // VS1003 音量
27.     Delay_us( 1000 );
28.
29.     while( DREQ == 0 );                 // 等待 DREQ 为高   表示能够接受音乐数据输入
30. }
```

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 SD 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] = {""}; /* MicroSD 卡根目录 */
  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 文件名为: %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 )
                {
                    PutChinese_strings22(10, 220, "开始播放", 0, 1);

            res = f_open( &fsrc, finfo.fname, FA_OPEN_EXISTING | FA_READ ); /* 以只读方式打开 */

                    /* 获取歌曲信息(ID3V1 tag / ID3V2 tag) */
                    if ( Read_ID3V1(&fsrc, &id3v1) == TRUE )
                    {// ID3V1 tag
                            printf( "\r\n 曲目    : %s \r\n", id3v1.title );
                            printf( "\r\n 艺术家  : %s \r\n", id3v1.artist );

                            PutChinese_strings22(10, 200, "曲目", 0, 1);
                            PutChinese_strings22(80, 200, id3v1.title, 0, 1);

                            PutChinese_strings22(10, 180, "艺术家", 0, 1);
                            PutChinese_strings22(80, 200, id3v1.artist, 0, 1);

                    }
                    else
```

```
55.                     {// 有些 MP3 文件没有 ID3V1 tag,只有
    ID3V2 tag
56.                             res = f lseek(&fsrc, 0);
57.                             Read_ID3V2(&fsrc, &id3v2);
58.
59.                             printf( "\r\n 曲目    : %s \r\n", id3v2.title );
60.                             printf( "\r\n 艺术家  : %s \r\n", id3v2.artist );
61.
62.                             PutChinese_strings22(10, 200, "曲目", 0, 1);
63.                             PutChinese_strings22(80, 200, id3v2.title, 0, 1);

64.
65.                             PutChinese_strings22(10, 180, "艺术家", 0, 1);
66.                             PutChinese_strings22(80, 180, id3v2.artist, 0, 1);

67.                     }
68.
69.                     /* 使文件指针 fsrc 重新指向文件头，因为在调用
    Read_ID3V1/Read_ID3V2 时，fsrc 的位置改变了 */
70.                     res = f open( &fsrc, finfo.fname, FA OPEN EXISTING | FA REA
    D );
71.                     //res = f lseek(&fsrc, 0);
72.

73.             br = 1;                                 /* br 为全局变量 */
74.                     TXDCS_SET( 0 );             /* 选择 VS1003 的数据接口 */
75. /* ------------------- 一曲开始 ---------------------*/
76.           USART1 printf( USART1, " \r\n 开始播放 \r\n" );
77.                     for (;;)
78.                     {
79.                 res = f_read( &fsrc, buffer, sizeof(buffer), &br );
80.             if ( res == 0 )
81.                         {
82.                                 count = 0;                              /* 512
    字节完重新计数 */
83.                                 Delay_us( 1000 );          /* 10ms 延时 */
84.                                 while ( count < 512)           /* SD 卡读取一个
    sector，一个 sector 为 512 字节 */
85.                                 {
86.                             if ( DREQ != 0 )           /* 等待 DREQ 为高，请求数据输
    入 */
87.                                     {
88.                                         for (j=0; j<32; j++ ) /* VS1003 的 FIFO
    只有 32 个字节的缓冲 */
89.                                         {
90.                                             VS1003_WriteByte( buffer[count] );

91.                                             count++;
92.                                         }
93.                                     }
94.                                 }
95.                 }
96.                         if (res || br == 0) break;    /* 出错或者到了 MP3 文件
    尾 */
97.             }
98.                      USART1_printf( USART1, " \r\n 播放结束 \r\n" );
99. /* ------------------- 一曲结束 ---------------------*/
100.                            count = 0;
101.                              /* 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 */
102.                            while ( count < 2048 )
103.                            {
104.                        if ( DREQ != 0 )
105.                                    {
106.                                        for ( j=0; j<32; j++ )
107.                                    {
108.                                            VS1003_WriteByte( 0 );

109.                                            count++;
```

```
                                    }
                                    }
                            }
                                  count = 0;
                                  TXDCS SET( 1 );    /* 关闭 VS1003 数据端口 */
                                f_close(&fsrc);    /* 关闭打开的文件 */
                                  PutChinese_strings22(10, 220, "播放结束", 0, 1);
                             }
                    }
                  } /* while (f_readdir(&dirs, &finfo) == FR_OK) */
                } /* if (f_opendir(&dirs, path) == FR_OK)  */
              } /* end of MP3_Play */


1.
```

这个函数的代码有点长，格式代码有点乱，野火推荐大家还是配合源码一起看，源码对齐风格非常好^_^。

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，盘符可以初始化为 0~9，我们这里初始化为 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，学过 LINUX 系统编程的朋友对这点肯定非常了解。

`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。这个结构体在 ff.h 这个头文件中声明。有关具体结构成员的作用大家可直接阅读注释。

```
1.  /* File object structure */
2.
3.  typedef struct _FIL_ {
4.      FATFS*  fs;         /* Pointer to the owner file system object */
5.      WORD    id;         /* Owner file system mount ID */
6.      BYTE    flag;       /* File status flags */
7.      BYTE    csect;      /* Sector address in the cluster */
8.      DWORD   fptr;       /* File R/W pointer */
9.      DWORD   fsize;      /* File size */
10.     DWORD   org_clust;  /* File start cluster */
11.     DWORD   curr_clust; /* Current cluster */
12.     DWORD   dsect;      /* Current data sector */
13. #if !_FS_READONLY
14.     DWORD   dir_sect;   /* Sector containing the directory entry */
15.     BYTE*   dir_ptr;    /* Ponter to the directory entry in the window */
16. #endif
17. #if !_FS_TINY
18.     BYTE    buf[_MAX_SS];/* File R/W buffer */
19. #endif
20. } FIL;
```

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。

在打开音频文件之后，我们调用函数 `Read_ID3V1()`来获取 MP3 文件尾部的 ID3V1 tag，如果存在 TAG 标签的话则在电脑的超级终端和 LCD 上显示出曲目和艺术家。`Read_ID3V1()`在 `mp3play.c` 中实现：

```
/*
 * 函数名：Read_ID3V1
 * 描述  ：从 MP3 文件尾读取 ID3V1 的信息，并将这些信息保存在一个全局数组中
 * 输入  ：- FileObject: file system
 *         - info: struct tag_info
 * 输出  ：无
 * 说明  ：现在的 mp3 基本上都没有 ID3V1 tag(在文件尾部)，有的是 ID3V2 tag(在文件头部)。
 */
//void Read_ID3V1(FIL *FileObject, struct tag_info *info)
int Read_ID3V1(FIL *FileObject, struct tag_info *info)
{
  /* ID3V1 的信息包含在文件末尾的 128 个 byte 中 */
    res = f_lseek(FileObject,FileObject->fsize - 128 );

   /* 将 ID3V1 中的 128 个 byte 的信息读到缓冲区 */
    res = f_read(FileObject, &readBuf, 128 , &n_Read);

    /* ID3V1 的标签头必须是“TAG”，否则认为没有标签 */
    if (strncmp("TAG", (char *) readBuf, 3) == 0)            {
        strncpy(info->title, (char *) readBuf + 3, MIN(30, sizeof(info->title) - 1));
       /* 标题 */
        strncpy(info->artist,(char *) readBuf + 3 + 30, MIN(30, sizeof(info->artist) - 1));

        /* 作者 */
      strncpy(info->album,(char *) readBuf + 3 + 30 + 30, MIN(30, sizeof(info->album) - 1));

      /* 专辑 */
       strncpy(info->year,(char *) readBuf + 3 + 30 + 30 + 30, MIN(4, sizeof(info->year) - 1));    /* 时间 */
                return TRUE;
    }
        else
        {
            return FALSE;
        }
}
```

要想完全读懂这个函数，必须具备一定的 MP3 文件格式的知识，不然你看了也白看，完全不知所以然^_^。

我们将读取到的关于 MP3 的信息存储在 `id3v1` 这个结构体中，结构体原型在 mp3play.h 这个头文件中声明：

```
typedef struct tagID3V1

{
```

```
3.      char Header[3];       // 标签头必须是"TAG"，否则被认为没有标签头
4.      char Title[30];       // 标题
5.      char Artist[30];      // 作者
6.      char Album[30];       // 专辑
7.      char Year[4];         // 出品年代
8.      char Comment[28];     // 备注
9.      char reserve;         // 保留
10.     char track;           // 音轨
11.     char Genre;           // 风格
12. }ID3V1, *pID3V1;
```

当我们看了这个结构体的原型之后，或许代码又多看懂了一些^_^。

如果 MP3 尾部没有 TAG 标签的话则从 MP3 的头部获取 MP3 的信息，即 ID3V2 tag，这里我们要知道只要它是 MP3 文件就一定有 ID3V2 tag。所以我们在 else 中调用函数 `Read_ID3V2();`函数 `Read_ID3V2()`同样在 `mp3play.c` 中实现：

```
1.  /*
2.   * 函数名: Read_ID3V2
3.   * 描述   : 从 MP3 文件头部读取 ID3V2 的信息，并将这些信息保存在一个全局数组中
4.   * 输入   : - FileObject: file system
5.   *          - info: struct tag_info
6.   * 输出   : 无
7.   * 说明   : MP3 文件都有 ID3V2，包含了作者，曲目，专辑等信息，长度不固定，扩展了 ID3V1 的信息量。
8.   */
9.  void Read_ID3V2(FIL *FileObject, struct tag_info *info)
10. {
11.     uint32 t p = 0;
12.
13.     res = f_lseek(FileObject,0); /* 定位到文件的头部 */
14.     res = f_read(FileObject, &readBuf, READBUF_SIZE , &n_Read); /* 从文件头部开始读取
    READBUF_SIZE 个字节 */
15.
16.     if (strncmp("ID3", (char *) readBuf, 3) == 0)   /* ID3V3 的头部标签必须为"ID3" */
17.     {
18.         uint32_t tag_size, frame_size, i;
19.         uint8_t version_major;
20.         int frame_header_size;
21.
22.         /* 标签大小，包括标签头的 10 个字节和所有标签帧的大小 */
23.         tag_size = ((uint32_t) readBuf[6] << 21) | ((uint32_t) readBuf[7] << 14) | ((uint1
    6_t) readBuf[8] << 7) | readBuf[9];
24.
25.         info->data_start = tag_size;
26.         version_major = readBuf[3];   /* 版本号为 ID3V2.3 时就记录为 3 */
27.
28.         if (version_major >= 3)
29.         {
30.             frame_header_size = 10;     /* ID3V2.3 的标签头为 10 个字节 */
31.         }
32.         else
33.         {
34.             frame_header_size = 6;
35.         }
36.         i = p = 10;
37.
38.         /* iterate through frames */
39.         while (p < tag_size)                /* 从标签头后开始处理 */
40.         {
41.             if (version_major >= 3)
42.             {
```

```
43.                 frame_size = ((uint32_t) readBuf[i + 4] << 24) | ((uint32_t) readBuf[i + 5
    ] << 16) | ((uint16_t) readBuf[i + 6] << 8) | readBuf[i + 7];
44.             }
45.             else
46.             {
47.                 frame_size = ((uint32_t) readBuf[i + 3] << 14) | ((uint16_t) readBuf[i + 4
    ] << 7) | readBuf[i + 5];
48.             }
49.             if (i + frame_size + frame_header_size + frame_header_size >= sizeof(readBuf))

50.             {
51.                 if (frame_size + frame_header_size > sizeof(readBuf))
52.                 {
53.                     res = f_lseek(FileObject,FileObject-
    >fptr + p + frame_size + frame_header_size);
54.                     res = f_read(FileObject, &readBuf, READBUF_SIZE , &n_Read);
55.                     p += frame_size + frame_header_size;
56.                     i = 0;
57.                     continue;
58.                 }
59.                 else
60.                 {
61.                     int r = sizeof(readBuf) - i;
62.                     memmove(readBuf, readBuf + i, r);
63.                     res = f_read(FileObject, (char *) readBuf + r, i , &n_Read);
64.                     i = 0;
65.                 }
66.             }
67.
68.             /* 帧标识"TT2"/"TIT2"表示内容为这首歌的 标题 */
69.             if (strncmp("TT2", (char *) readBuf + i, 3) == 0 || strncmp("TIT2", (char *) r
    eadBuf + i, 4) == 0)
70.             {
71.                 strncpy(info-
    >title, (char *) readBuf + i + frame_header_size + 1, MIN(frame_size - 1, sizeof(info-
    >title) - 1));
72.                 if( ( info->title[0] == 0xFE && info->title[1] == 0xFF ) || ( info-
    >title[0] == 0xFF && info->title[1] == 0xFE ) )
73.                 {
74.                     /* unicode 格式*/
75.                     memset(info->title,0,sizeof(info->title));
76.                     printf( "-- MP3 title no support unicode \r\n");
77.                 }
78.             }
79.
80.                         /* 帧标识"TP1"/"TPE1"表示内容为这首歌的 作者 */
81.             else if (strncmp("TP1", (char *) readBuf + i, 3) == 0 || strncmp("TPE1", (char
     *) readBuf + i, 4) == 0)
82.             {
83.                 strncpy(info-
    >artist, (char *) readBuf + i + frame_header_size + 1, MIN(frame_size - 1, sizeof(info-
    >artist) - 1));
84.
85.                 if( ( info->artist[0] == 0xFE && info->artist[1] == 0xFF ) || ( info-
    >artist[0] == 0xFF && info->artist[1] == 0xFE ) )
86.                 {
87.                     /* unicode 格式*/
88.                     memset(info->artist,0,sizeof(info->artist));
89.                     printf( "-- MP3 artist no support unicode \r\n");
90.                 }
91.             }
92.
93.             p += frame_size + frame_header_size;
94.             i += frame_size + frame_header_size;
95.         }
96.     }
97. }
```

要想读懂这个函数，我们又要来一次 MP3 知识的扫盲了^_^……

ID3V2 到现在一共有 4 个版本，但流行的播放软件一般只支持第三版本，即 ID3V2.3。由于 ID3V1 记录在文件尾部了，故 ID3V2 就只能记录在头部了，也正是这个原因，对 ID3V2 的操作要比 ID3V1 慢，这也是我们在程序中先检测 ID3V1 的原因，

只要我们检测到 ID3V1 就不用去检测 ID3V2 了，况且检测 ID3V2 的代码实现要复杂得多。而且 ID3V2 结构比 ID3V1 结构要复杂很多，但比 ID3V1 全面且可以伸缩和扩展。

每个 ID3V2.3 的标签都有一个标签头和若干个标签帧或一个扩展标签头组成。关于曲目的信息如标题、作者等信息都存放在不同的标签帧中，扩展标签头和标签帧并不是必要的，但每个标签至少有一个标签帧。标签头和标签帧一起存放在 MP3 文件的首部。

- 标签头

在文件的首部顺序记录 10 个字节的 ID3V2.3 的头部，数据结构如下：

```
1.  cha Header[3];     // 必须为"ID3"，否则认为标签不存在
2.  char Ver            // 版本号为 ID3V2.3 就记录为 3
3.  char Revision      // 副版本号此版本记录为 0
4.  char Flag           // 存放标志的字节，这个版本只定义了三位
5.  char Size[4]        // 标签大小，包括标签头的 1 个字节和所有的标签帧的大小
```

标签大小的计算：

一共四个字节，但每个字节只用 7 位，最高位恒为 0，格式如下：

0xxxxxxx, 0xxxxxxx, 0xxxxxxx, 0xxxxxxx

计算的时候要将 0 去掉，得到一个 28 位的二进制数，就是标签大小，

计算公式如下：

```
1.  int total_size;
2.  total_size = (Size[0]&0x7f) << 21
3.                 +(Size[1]&0x7f) << 14
4.                 +(Size[3]&0x7f) << 7
5.                 +(Size[4]&0x7f);
```

看到这对代码的实现是不是有点感觉了呀，别急，下面我们继续来拨开那层层迷雾^_^……

- 帧标识

用四个字符标识一个帧，说明一个帧的内容含义，常用的对照如下：

TIT2：标题，表示内容为这首歌的标题

TPE1：作者

TALB：专辑

TRCK：音轨(格式：N/M，其中 N 为专辑中的第 N 首，M 为专辑中共 M 首，N 和 M 为 ASCII 码表示的数字)

TYPE：年代　是 ASCII 码表示的数字

TCON：类型　直接用字符串表示

COMM：备注　格式：“eng\0 备注内容”，其中 eng 表示备注所使用的自然语言

但要注意，这里面有些帧标识 ID3V2 里面是没有的，但标题和作者一般都会有。要想知道 ID3V2 是否包含有这些信息，我们可以用 16 进制的编辑器来查看 MP3 文件，ID3V2 在头部，ID3V1 在尾部。考虑到大家可能没有用过 16 进制的编辑器，所以野火在光盘资料《常用代码编辑软件》这个文件夹下提供了 Uedit32 这个 16 进制编辑器，这个编辑器还可以阅读 c/c++代码，功能非常强大。

- 帧大小的计算

这个可没有标签头的算法那么麻烦，每个字节的 8 位全用，格式如下：

xxxxxxxx, xxxxxxxx, xxxxxxxx, xxxxxxxx

代码如下：

```
1.  int FSize;
2.  FSize = Size[0] * 0x100000000
3.            +Size[1] * 0x10000
4.            +Size[2] * 0x100
5.            +Size[3];
```

讲到这里，`Read_ID3V2()`函数里面的代码大伙也应该理解的差不多了^_^。下面我们再接再厉，趁着势头把后面的代码也一口气读完吧。

`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。

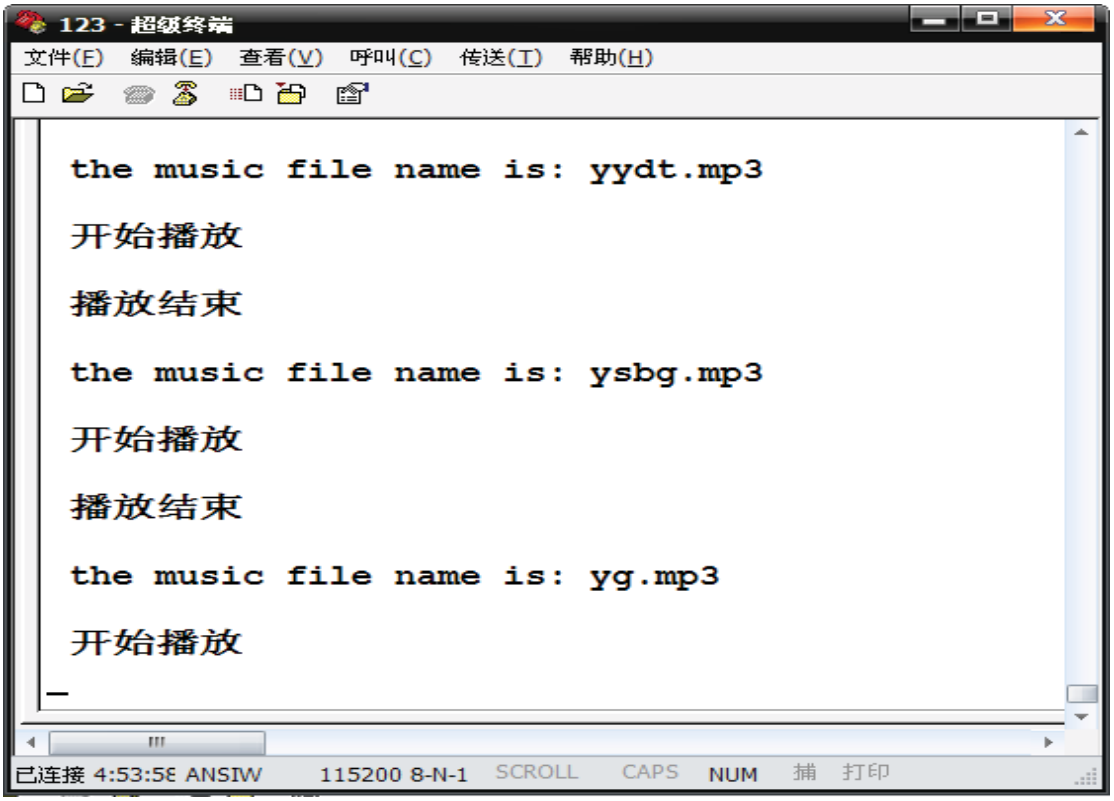
根据 VS1003 的要求，在一曲结束后需发送 2048 个 0 来确保下一首的正常播放。

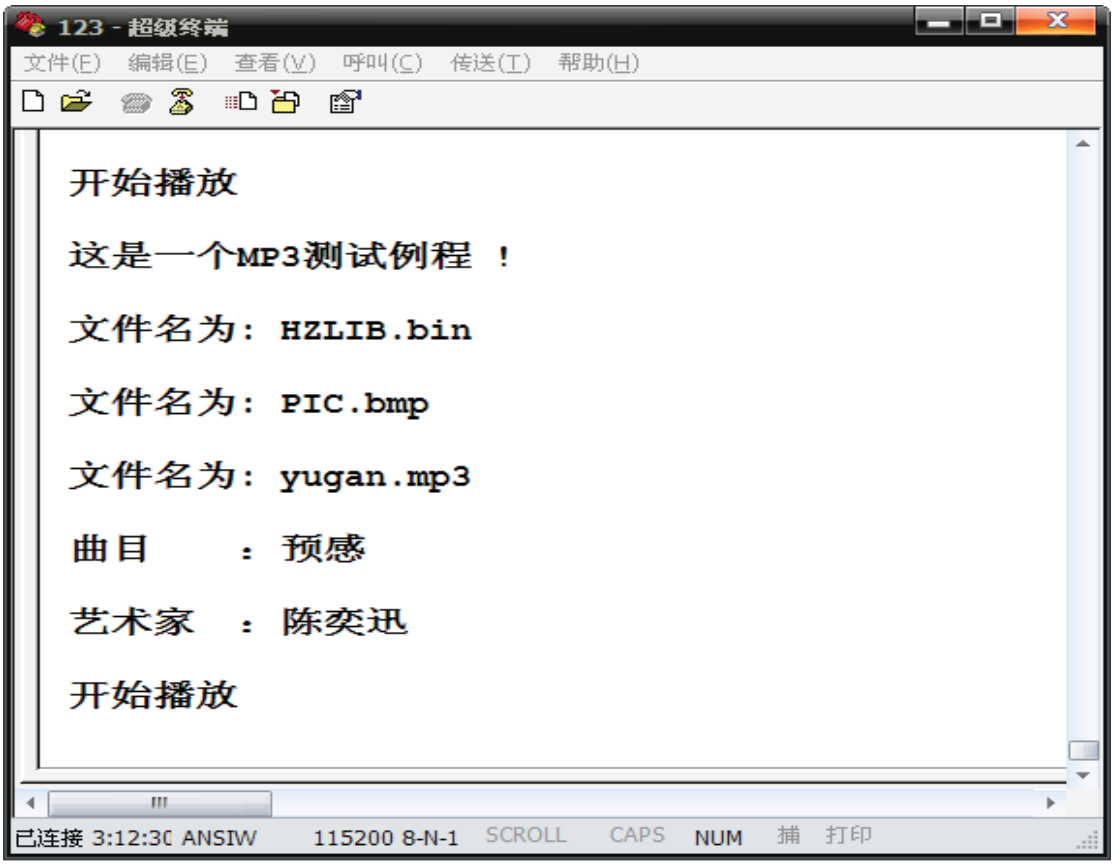
一曲播放完毕我们关闭 vs1003 的数据端，关闭打开的文件，等待下一曲的播放，直到目录下的音频文件播放完为止。还暂不支持循环播放的功能。

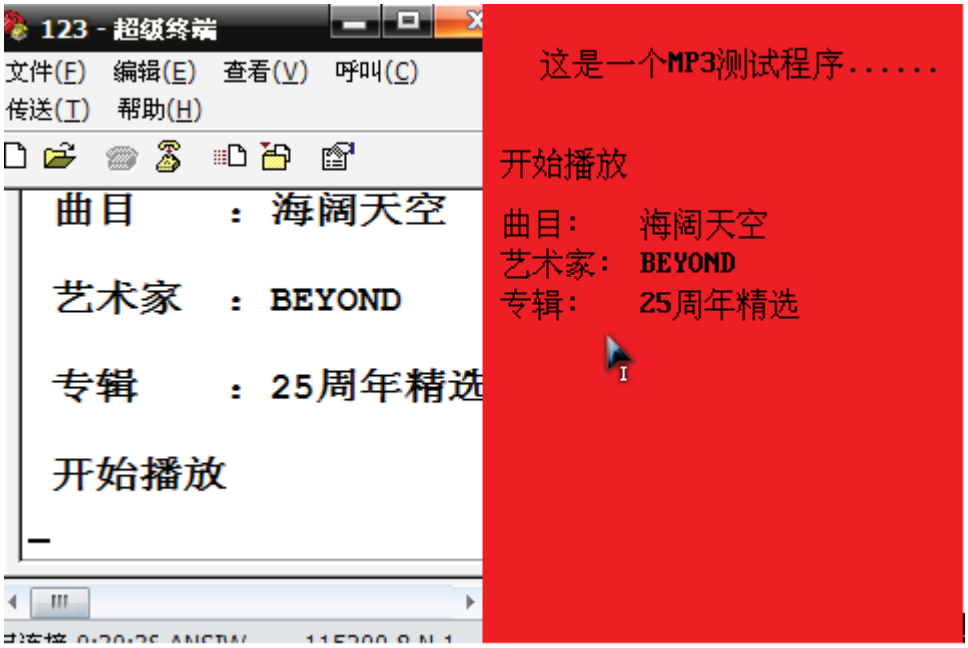
这里面涉及到了 vs1003 操作的一些特性，需大家参考 vs1003 的 datasheet 来帮助理解，野火建议大家仔仔细细地阅读，不要因为难度或者是繁琐而退缩，要知道作为一个嵌入式软件开发工程师，阅读 ic datasheet（中文/英文）是一种必备的技能。

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（在 LCD 里面的效果图这里就没有拍照了，大家见谅^_^）：


123 - 超级终端
文件(F) 编辑(E) 查看(V) 呼叫(C) 传送(T) 帮助(H)
the music file name is: yydt.mp3
开始播放
播放结束
the music file name is: ysbg.mp3
开始播放
播放结束
the music file name is: yg.mp3
开始播放
已连接 4:53:58 ANSIW 115200 8-N-1 SCROLL CAPS NUM 捕 打印


123 - 超级终端
文件(F) 编辑(E) 查看(V) 呼叫(C) 传送(T) 帮助(H)
开始播放
这是一个MP3测试例程 !
文件名为: HZLIB.bin
文件名为: PIC.bmp
文件名为: yugan.mp3
曲目 : 预感
艺术家 : 陈奕迅
开始播放
已连接 3:12:30 ANSIW 115200 8-N-1 SCROLL CAPS NUM 捕 打印



这两张是 LCD 里面显示的信息，不过在测试时野火换了首歌曲^_^。

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(yugan.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^_^。