AN1102 ATK-VS1053 MP3 模块使用说明

本应用文档（AN1102，对应 **ALIENTEK MiniSTM32 开发板扩展实验 10**）将教大家如何在 ALIENTEK STM32 开发板上使用 ATK-VS1053 MP3 模块。本文档我们将使用 ATK-VS1053 MP3 模实现音频播放，设计一个属于你自己的 MP3！

本文档分为如下几部分：

1， ATK-VS1053 MP3 模块简介
2， 硬件连接
3， 软件实现
4， 验证

1、ATK-VS1053 MP3 模块简介

ATK-VS1053 MP3 MODULE 是 ALIENTEK 推出的一款高性能音频编解码模块，该模块采用 VS1053B 作为主芯片，支持：MP3/WMA/OGG/WAV/FLAC/MIDI/AAC 等音频格式的解码，并支持：OGG/WAV 音频格式的录音，支持高低音调节以及 EarSpeaker 空间效果设置，功能十分强大。

1.1 模块资源简介

ATK-VS1053 MP3 模块是 ALIENTEK 开发的一款高性能音频编解码模块，该模块接口丰富、功能完善，仅需提供电源（3.3V/5.0V），即可通过单片机（8/16/32 位单片机均可）控制模块实现音乐播放，或者录音等功能，模块其资源图如图 1.1.1 所示：

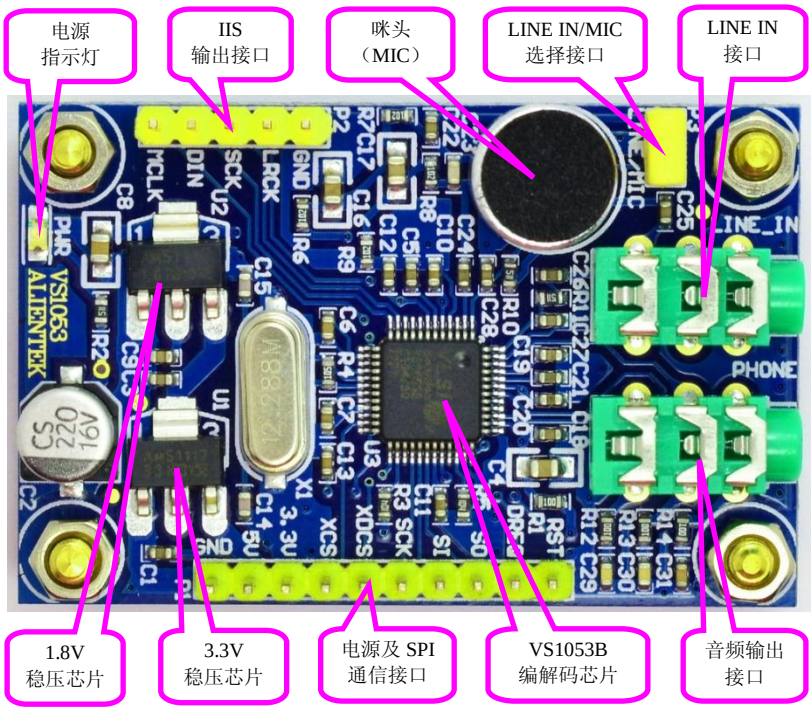

图 1.1.1 ATK-VS1053 MP3 模块资源图

从图 1.1.1 可以看出，ATK-VS1053 MP3 模块不但外观漂亮，而且功能齐全、接口丰富，

模块尺寸为 34mm*52.6mm，并带有安装孔位，非常小巧，并且利于安装，可方便应用于各种设计。

ALIENTEK ATK-VS1053 MP3 模块板载资源如下：

- 高性能编解码芯片：VS1053B
- 1 个 LINE IN/MIC 选择接口
- 1 个咪头
- 1 个电源指示灯（蓝色）
- 1 个 1.8V 稳压芯片
- 1 个 3.3V 稳压芯片
- 1 路 IIS 输出接口
- 1 路电源及 SPI 控制接口
- 1 路 3.5mm LINE IN 接口，支持双声道输入录音
- 1 路 3.5mm 音频输出接口，可直接插耳机

ATK-VS1053 模块采用高准设计，特点包括：

- 板载 VS1053B 高性能编解码芯片，支持众多音频格式解码，支持 OGG/WAV 编码。
- 板载稳压电路，仅需外部提供一路 3.3V 或 5V 供电即可正常工作；
- 板载 3.5mm 耳机插口，可直接插入耳机欣赏高品质音乐；
- 板载咪头（MIC），无需外部麦克风，即可实现录音；
- 板载 IIS 输出，可以接外部 DAC，获得更高音质；
- 板载电源指示灯，上电状态一目了然；
- 采用国际 A 级 PCB 料，沉金工艺加工，稳定可靠；
- 采用全新元器件加工，纯铜镀金排针，坚固耐用；
- 人性化设计，各个接口都有丝印标注，使用起来一目了然；接口位置设计安排合理，方便顺手。
- PCB 尺寸为 34mm*52.6mm，并带有安装孔位，小巧精致；

ATK-VS1053 MP3 模块的资源介绍，我们就介绍到这里，详细的介绍，请看《ATK-VS1053 MP3 模块用户手册》相关章节。

1.2 模块使用

模块通过SPI接口来接受输入的音频数据流，它可以是一个系统的从机，也可以作为独立的主机。这里我们只把它当成从机使用。我们通过SPI口向VS1053不停的输入音频数据，它就会自动帮我解码了，然后从输出通道输出音乐，这时我们接上耳机就能听到所播放的歌曲了。

模块（VS1053）通过7根信号线同主控芯片连接，分别是：XCS、XDCS、SCK、SI、SO、DREQ、和RST。其中RST是VS1053的复位信号线，低电平有效。DREQ是一个数据 请求信号，用来通知主机，VS1053可以接收数据与否。SCK、SI(MOSI)和SO(MISO)则是VS1053的SPI接口，他们在XCS和XDCS的控制下面来执行不同的数据通信。另外，模块需要外部提供5V/3.3V供电，推荐采用5V供电，这样，总共需要9根线来连接。

关于VS1053 SPI通信的详细介绍，请参考《ATK-VS1053 MP3模块用户手册》2.3节。这里我们不做详细阐述了。

ATK-VS1053 MP3 模块的原理图如图 1.2.1 所示：

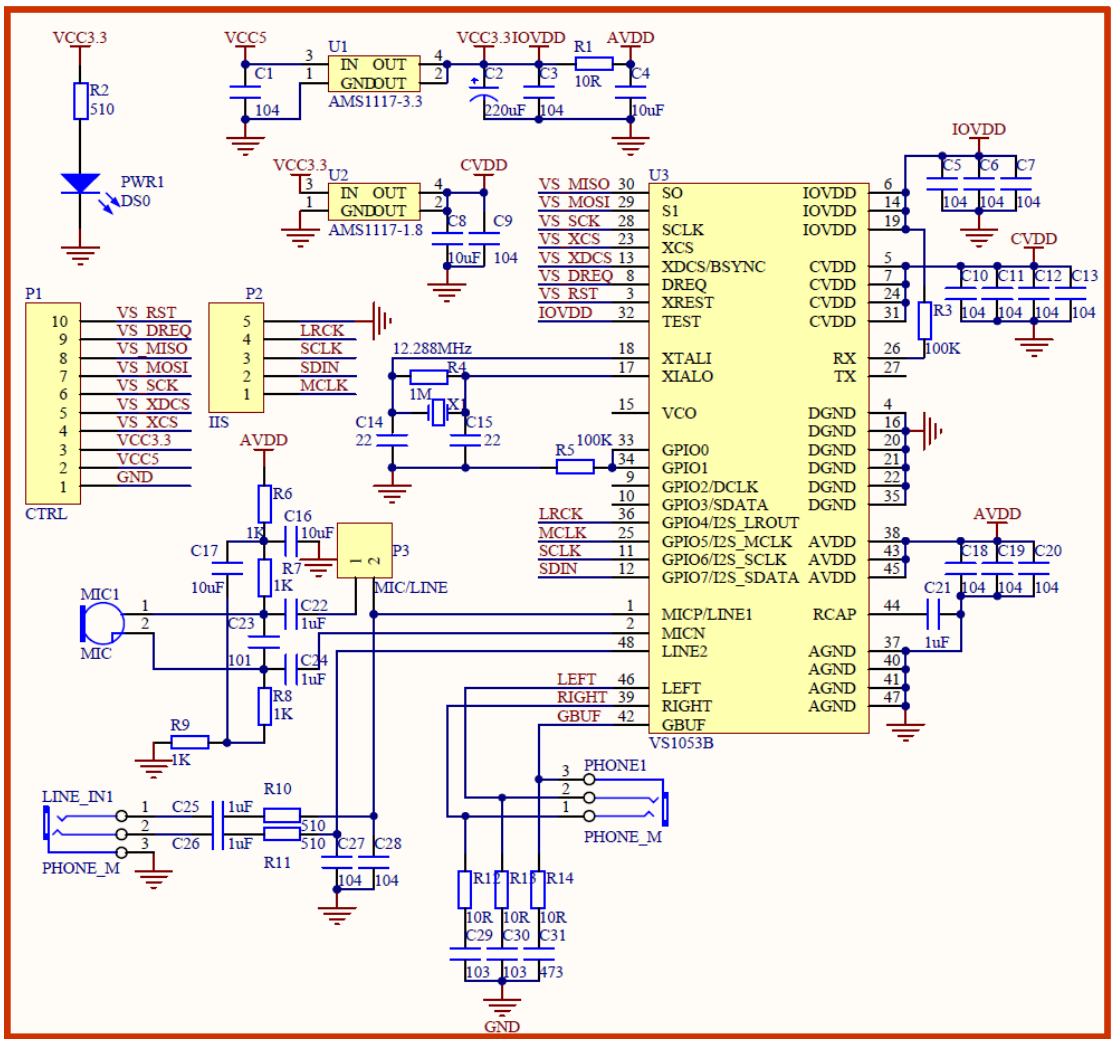

图1.2.1 ATK-VS1053 MP3模块原理图

模块通过 P1 接口与外部单片机系统连接，P1 引脚描述如表 1.2.1 所示：

序号	名称	说明
1	GND	地
2	5V	5V 供电口，只可以供电
3	3.3V	3.3V 供电口，当使用 5V 供电的时候，这里可以输出 3.3V 电压给外部使用
4	XCS	片选输入（低有效）
5	XDCS	数据片选/字节同步
6	SCK	SPI 总线时钟线
7	SI	SPI 总线数据输入线
8	SO	SPI 总线数据输出线
9	DREQ	数据请求
10	RST	复位引脚（硬复位，低电平有效）

表 1.2.1 供电与通信接口 P1 口各引脚功能表

用 ATK-VS1043 MP3 模块来播放音频文件是非常简单的，一般的音频文件（MP3/WMA/OGG/WAV/MIDI/AAC 等），只需要简单的 3 步操作即可实现音频播放。

1）复位 VS1053

这里包括了硬复位（拉低 RST）和软复位（设置 MODE 寄存器的 SM_RESET 位为 1），是为了让 VS1053 的状态回到原始状态，准备解码下一首歌曲。这里建议大家在每首歌曲播放

之前都执行一次硬件复位和软件复位，以便更好的播放音乐。

2）配置 VS1053 的相关寄存器

这里我们配置的寄存器包括 VS1053 的模式寄存器（MODE）、时钟寄存器（CLOCKF）、音调寄存器（BASS）、音量寄存器（VOL）等。

3）发送音频数据

当经过以上两步配置以后，我们剩下来要做的事情，就是往 VS1053 里面扔音频数据了，只要是 VS1053 支持的音频格式，直接往里面丢就可以了，VS1053 会自动识别，并进行播放。不过发送数据要在 DREQ 信号的控制下有序的进行，不能乱发。这个规则很简单：只要 DREQ 变高，就向 VS1053 发送 32 个字节。然后继续等待 DREQ 变高，直到音频数据发送完。

经过以上三步，我们就可以利用模块来播放音乐了。

2、硬件连接

本章实验功能简介：开机检测SD卡和字库是否存在，如果检测无问题，则开始循环播放SD 卡根目录的歌曲，在TFTLCD 上显示歌曲名字、播放时间、歌曲总时间、歌曲总数目、当前歌曲的编号等信息。KEY0 用于选择下一曲，KEY1 用于选择上一曲，而KEY_UP 用于暂停/播放。本实验用DS1 来象征性的指示程序的运行（实际是指示一个簇的结束）。

本实验用到的资源如下：

1）STM32F103RBT6。

2）DS1。

3）TFTLCD 液晶模块。

4）SD 卡。

5）KEY0、KEY1、KEY_UP。

6）ATK-VS1053 MP3模块

ALIENTEK MiniSTM32 开发板与 ATK-VS1053 MP3 模块的连接关系如图 2.1 所示：

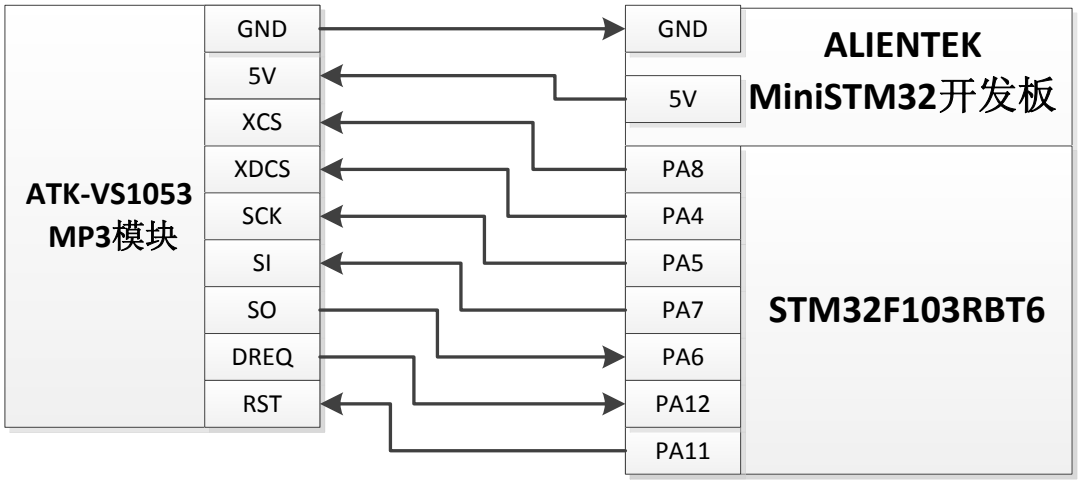

图 2.1 ATK-VS1053 MP3 模块与 MiniSTM32 开发板连接关系图

上表中，就是ALIENTEK MiniSTM32开发板与ATK-VS1053 MP3模块的连接示意图。实际连接如图2.2所示：

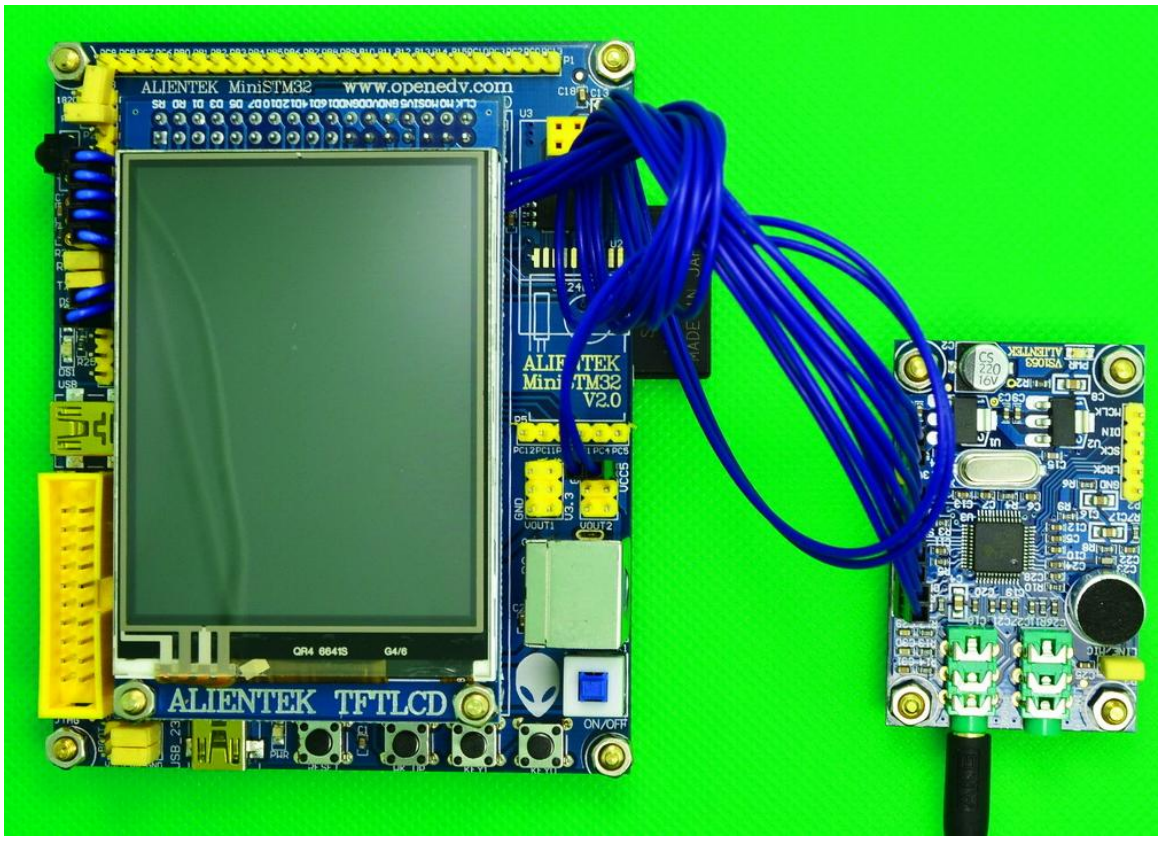

图 2.2 模块与 STM32 开发板连接实物图

图中，我们总共用了 9 个杜邦线连接 ATK-VS1053 MP3 模块与 MiniSTM32 开发板，供电采用 5V 直接供电，完全与图 2.1 所示的关系图一致。

3、软件实现

本章，我们在第二十四章实验的基础上修改，先打开第二十四章实验的工程，首先在HARDWARE文件夹所在的文件夹下新建一个MP3的文件夹。在该文件夹下新建mp3player.c和mp3player.h 两个文件。然后在HARDWARE文件夹下新建一个VS10XX的文件夹，在该文件夹里面新建VS10XX.c和VS10XX.h两个文件。

打开VS10XX.c，里面的代码我们不一一贴出了，这里挑几个重要的函数给大家介绍一下，首先要介绍的是VS_Soft_Reset，该函数用于软复位VS1053，其代码如下：

```
//软复位 VS10XX
//注意,在这里设置完后,系统 SPI 时钟被修改成 9M 左右
void VS_Soft_Reset(void)
{
    u8 retry;
    SPIx_SetSpeed(SPI_SPEED_256);      //256 分频 超低速 281.25Khz
    while(VS_DQ==0);                   //等待空闲
    SPIx_ReadWriteByte(0X00);          //启动传输
    retry=0;
    VS10XX_ID=VS_RD_Reg(SPI_STATUS);//读取状态寄存器
    VS10XX_ID>>=4;                     //得到 VS10XX 的芯片信号
    if(VS10XX_ID==VS1053)VS_WR_Cmd(SPI_MODE,0x0816);//软件复位,新模式
```

```
        else VS_WR_Cmd(SPI_MODE,0x0804);    //软件复位,新模式
        while(VS_DQ==0&&retry<200)          //等待 DREQ 为高
        {
            retry++;
            delay_us(50);
        };
        retry=0;
        while(VS_RD_Reg(SPI_CLOCKF)!=0X9800)//等待设置成功
        {
            VS_WR_Cmd(SPI_CLOCKF,0X9800); //设置 VS10XX 的时钟,3.5 倍频 ,2xADD
            if(retry++>100)break;
        }
        VS_Rst_DecodeTime();                //复位解码时间
        //向 VS10XX 发送 4 个字节无效数据，用以启动 SPI 发送
        SPIx_SetSpeed(SPI_SPEED_8);         //8 分频速度
        VS_XDCS=0;                          //选中数据传输 记得,这里一定要传送 0X00
        SPIx_ReadWriteByte(0X0);
        SPIx_ReadWriteByte(0X0);
        SPIx_ReadWriteByte(0X0);
        SPIx_ReadWriteByte(0X0);
        VS_XDCS=1;                          //取消数据传输
    }
```

该函数比较简单，先配置一下VS1053的模式顺便执行软复位操作，在软复位结束之后，再设置时钟，待时钟配置完成，就发送4个0XFF启动SPI发送。接下来，我们介绍一下VS_WR_Cmd函数，该函数用于向VS1053写命令，代码如下：

```
    //向 VS10XX 写命令
    //address:命令地址
    //data:命令数据
    void VS_WR_Cmd(u8 address,u16 data)
    {
        u16 retry=0;
        while(VS_DQ==0&&(retry++)<0xfffe)retry++;  //等待空闲
        SPIx_SetSpeed(SPI_SPEED_64);               //64 分频速度
        VS_XDCS=1;
        VS_XCS=0;
        SPIx_ReadWriteByte(VS_WRITE_COMMAND);//发送 VS10XX 的写命令
        SPIx_ReadWriteByte(address); //地址
        SPIx_ReadWriteByte(data>>8); //发送高八位
        SPIx_ReadWriteByte(data);    //第八位
        VS_XCS=1;
        SPIx_SetSpeed(SPI_SPEED_8);  //8 分频速度
    }
```

该函数用于向VS1053发送命令，这里要注意VS1053的写操作比读操作快（写1/4 CLKI，读1/6 CLKI），虽然说写寄存器最快可以到1/4CLKI，但是经实测在1/4CLKI 的时候会出错，

所以在写寄存器的时候最好把SPI 速度调慢点，然后在发送音频数据的时候，就可以1/4CLKI的速度了。有写命令的函数，当然也有读命令的函数了。VS_RD_Reg 用于读取VS1053的寄存器的内容。该函数代码如下：

```
//读 VS10XX 的寄存器
//注意不要用倍速读取,会出错
u16 VS_RD_Reg(u8 address)
{
    u16 temp=0;
    u8 retry=0;
    while(VS_DQ==0&&(retry++)<0XFE);     //等待空闲
    if(retry>=0XFE)return 0;
    SPIx_SetSpeed(SPI_SPEED_64);        //64 分频速度
    VS_XDCS=1;
    VS_XCS=0;
    SPIx_ReadWriteByte(VS_READ_COMMAND);//发送 VS10XX 的读命令
    SPIx_ReadWriteByte(address);          //地址
    temp=SPIx_ReadWriteByte(0xff);      //读取高字节
    temp=temp<<8;
    temp+=SPIx_ReadWriteByte(0xff);     //读取低字节
    VS_XCS=1;
    SPIx_SetSpeed(SPI_SPEED_8);         //8 分频速度
    return temp;
}
```

该函数用于读取某个寄存器的值，我们这里不多说了。VS10XX.c 的剩余代码和VS10XX.h的代码，这里就不贴出来了。读者可以去光盘查看详细源码。

保存VS10XX.c和VS10XX.h 两个文件。把VS10XX.c 加入到HARDWARE 组下。然后我们打开mp3player.c，该文件我们仅介绍一个函数，其他代码请看光盘的源码。这里要介绍的是Play_Song 函数，该函数代码如下：

```
//播放音乐
//index:播放的歌曲编号
//返回值：0,成功；1，下一曲；2，上一曲；0xff 得到文件信息失败；0xfe，复位失败；
u8 MUSIC_BUFFER[512];
u8 Play_Song(u16 index,u16 total)
{
    u32 bfactor;
    u32 bcluster;
    u16 count;
    u8 key,n;
    u16 i;
    u8 pause=0;//不暂停
    FileInfoStruct FileInfo;
    i=Get_File_Info(Cur_Dir_Cluster,&FileInfo,T_MP3|T_WMA|T_WAV|T_MID|T_FLAC
    |T_OGG,&index);
    if(i==0)return 0xff;              //得到文件信息失败。
```

```
if(VS_HD_Reset())return 0xfe;     //硬复位
VS_Soft_Reset();                  //软复位 VS10XX
set10XX();                        //设置音量等信息
if(VS10XX_ID==VS1053)
{
    if(FileInfo.F_Type==T_FLAC)
    VS_Load_Patch((u16*)vs1053b_patch,VS1053B_PATCHLEN);
    //如果是 FLAC 文件,则加载 FLAC 用户代码.
}else //默认为 1003,其他未测试
{
   if(FileInfo.F_Type==T_FLAC||FileInfo.F_Type==T_OGG)return 0xfd;//不支持
}
LCD_Fill(0,110,239,319,WHITE);                  //整个屏幕清空
Show_Str(60,150,FileInfo.F_Name,16,0);          //显示歌曲名字
bfactor=fatClustToSect(FileInfo.F_StartCluster);//得到开始簇对应的扇区
bcluster=FileInfo.F_StartCluster;               //得到文件开始簇号
count=0;
while(1)    //播放音乐的主循环
{
   if(SD_ReadDisk(MUSIC_BUFFER,bfactor,1))break;//读取一个扇区的数据
   SPIx_SetSpeed(SPI_SPEED_8);
   //高速,对 VS1003B,最大值不能超过 36.864/6Mhz，这里设置为 4.5M
   count++;//扇区计数器
   i=0;
   do        //主播放循环
   {
      if(VS_DQ!=0&&pause==0)   // 非暂停 送数据给 VS1003
      {
         VS_XDCS=0;
         for(n=0;n<32;n++) SPIx_ReadWriteByte(MUSIC_BUFFER[i++]);
         VS_XDCS=1;
      }
      key=KEY_Scan();
      if(key)
      {
         switch(key)
         {
            case 1://下一首歌
               return 1;
            case 2://上一首歌
               return 2;
            case 3://暂停/播放
               pause=!pause;
         }
```

```
                }
            }while(i<511);//循环发送 512 个字节
            MP3_Msg_Show(FileInfo.F_Size,index,total);
            bfactor++;          //扇区加
            if(count>=SectorsPerClust)//一个簇结束,换簇
            {
                count=0;
                bcluster=FAT_NextCluster(bcluster);
                //printf("NEXT:%d\n",bcluster);
                LED1=!LED1;
                //文件结束
                if((FAT32_Enable==0&&bcluster==0xffff)||bcluster==0x0ffffff8||bcluster==
                0x0fffffff)break;//error
                bfactor=fatClustToSect(bcluster);
            }
        }
        VS_HD_Reset();  //硬复位
        VS_Soft_Reset();  //软复位
        LED1=1;       //关闭 DS1
        return 0;       //返回按键的键值!
    }
```

该函数，就是我们解码音乐的核心函数了，该函数得到要播放的歌曲的一些信息，然后初始化VS1053，最后在死循环里面等待DREQ 信号的到来，每次VS_DQ变高，就向VS1053发送32 个字节，知道整个文件读完。此段代码还包含了对按键的处理（暂停/播放、上一首、下一首）及当前播放的歌曲的一些状态（编号、码率、播放时间、总时间、歌曲名字等）。mp3player.c 的其他代码和mp3player.h在这里就不详细介绍了，请大家直接参考本例程源码。最后，我们在test.c里面修改main 函数如下：

```
int main(void)
{
    u8 i;
    Stm32_Clock_Init(9);    //系统时钟设置
    delay_init(72);         //延时初始化
    uart_init(72,9600);     //串口 1 初始化
    LCD_Init();             //初始化液晶
    KEY_Init();             //按键扫描初始化
    LED_Init();             //LED 初始化
    SPI_Flash_Init();       //SPI FLASH 初始化
    usmart_dev.init(72);
    POINT_COLOR=RED;
    LCD_ShowString(60,70,"Font checking...");
    //字体更新
    if(Font_Init())//字库不存在,则更新字库
    {
        POINT_COLOR=RED;
```

```
        LCD_Clear(WHITE);
        LCD_ShowString(60,50,"Mini STM32");
        LCD_ShowString(60,70,"Font Updating...");
        //字体更新
        SD_Initialize();              //初始化 SD 卡
        while(FAT_Init())//FAT  错误
        {
            LCD_ShowString(60,90,"FAT SYS ERROR");
            i= SD_Initialize();
            if(i)//SD 卡初始化
            {
                LCD_ShowString(60,110,"SD_CARD ERROR");
            }
            delay_ms(500);
            LCD_Fill(60,90,240,126,WHITE);//清除显示
            delay_ms(500);
            LED0=!LED0;
        }
        LCD_Fill(60,90,240,126,WHITE);//清除显示
        while(Update_Font()!=0)//字体更新出错
        {
            LCD_ShowString(60,90,"SYSTEM FILE LOST");
            delay_ms(500);
            LCD_ShowString(60,90,"Please Check....");
            delay_ms(500);
            LED0=!LED0;
        };
        LCD_Clear(WHITE);
    }
    Show_Str(30,70,"ALIENTEK MiniSTM32 开发板",16,0);
    SD_Initialize();              //初始化 SD 卡
    while(FAT_Init())//FAT  错误
    {
        LCD_ShowString(60,90,"FAT SYS ERROR");
        i= SD_Initialize();
        if(i)//SD 卡初始化
        {
            LCD_ShowString(60,110,"SD_CARD ERROR");
        }
        delay_ms(500);
        LCD_Fill(60,90,240,126,WHITE);//清除显示
        delay_ms(500);
        LED0=!LED0;
    }
```

```
        VS_Init();
        while(1)
        {
            Show_Str(60,90,"存储器测试...",16,0);
            LED0=0;
            VS_Ram_Test();
            Show_Str(60,90,"正弦波测试...",16,0);
            LED1=0;
            VS_Sine_Test();
            Show_Str(60,90,"<<MP3 播放器>>",16,0);
            Play_Music();
        }
    }
```

该函数先检测外部flash 是否存在字库，然后对SD 卡和文件系统进行初始化，最后初始化VS1053，并在正弦测试和寄存器测试之后，开始播放SD卡根目录上的音乐。软件部分就介绍到这里。

4、验证

在代码编译成功之后，我们下载代码到ALIENTEK MiniSTM32 开发板上，当检测到SD卡（里面已经有歌曲了）后，执行完两个测试（RAM测试和SIN测试）之后，就开始自动播放歌曲了，如图4.1所示：

图4.1 MP3播放中

从上图可以看出，当前正在播放第1首歌曲，总共3首歌曲，歌曲名、播放时间、总时长及码率等信息等也都有显示。此时DS1 会随着音乐的播放而闪烁，音乐的码率越高，DS1闪烁的越快。

只要我们在解码模块插入耳机，就能听到歌曲的声音了。同时，我们可以通过按KEY0和KEY1 来切换下一曲和上一曲，通过WK_UP 按键来控制播放和暂停。

至此，我们就完成了一个简单的MP3 播放器了，在此基础上进一步完善，就可以做出一个比较实用的MP3了。大家可以自己发挥想象，做出一个你心仪的MP3。

正点原子@ALIENTEK
2013-11-13
开源电子网：www.openedv.com
星翼电子官网：www.alientek.com