MP3 播放器实验

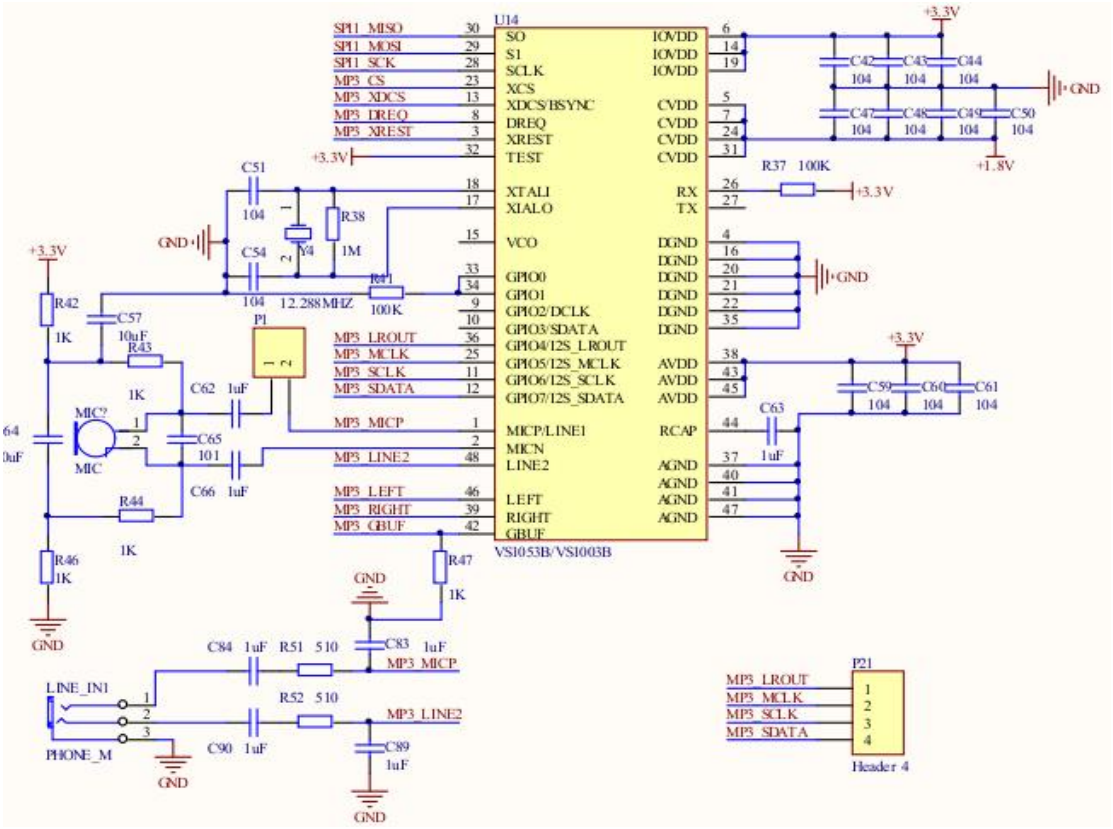
学习目标

1. 复习 STM32 硬件 SPI
2. 学会操作 VS1053 播放音乐

前几年，MP3 曾经风行一时，几乎人手一个，今天我们用开发板自己来实现一个 MP3 播放器。

31.1 VS1053 简介

VS1053 是继 VS1003 后荷兰 VLSI 公司出品的又一款高性能解码芯片。该芯片可以实现对 MP3/OGG/WMA/FLAC/WAV/AAC/MIDI 等音频格式的解码，同时还可以支持 ADPCM/OGG 等格式的编码，性能相对以往的 VS1003 提升不少。开发板上面自带了一个 VS1053 解码芯片，电气原理图为：



它使用的 IO 为：

VS1053 的 SPI 支持两种模式：1，VS1002 有效模式（即新模式）。2，VS1001 兼容模式。这里我们仅介绍 VS1002 有效模式（此模式也是 VS1053 的默认模式）。

从原理图中，我们看出 VS1053 跟单片机相连的引脚主要有 7 根，他们是：

1) VS_MISO：SPI 输入线
2) VS_MOSI：SPI 输出线
3) VS_SCK：SPI 时钟线
4) VS_XCS：器件片选
5) VS_XDCS：数据片选和字节同步
6) VS_DREQ：数据请求，输入总线。这个信号引脚是用来反馈 VS1053 的 2048 字节 FIFO 是否可以接受数据。如果它为高电平，则 VS1053 可以接收最少 32 字节的 SDI 数据或者接受 SCI 命令。当流缓冲区太满和 SCI 命令正在执行的期间，DREQ 会转换到低电平，此时应该停止想 VS1053 发送新数据和命令。
7) VS_RST：复位端

31.2 VS1053 的操作

VS1053 通信支持 SPI 协议，所以我们使用 STM32 的硬件 SPI 发送接收数据就可以了，开发板上面使用的是 STM32 的硬件 SPI1，具体操作跟之前介绍的 SPI 操作时一样的，这么我们就不详细介绍了。

1. SCI 串行命令写操作

此串行命令接口 SCI 的串行总线协议包括：一个指令字节、一个地址字节和一个 16 位的数据字，也就是 4 个字节。

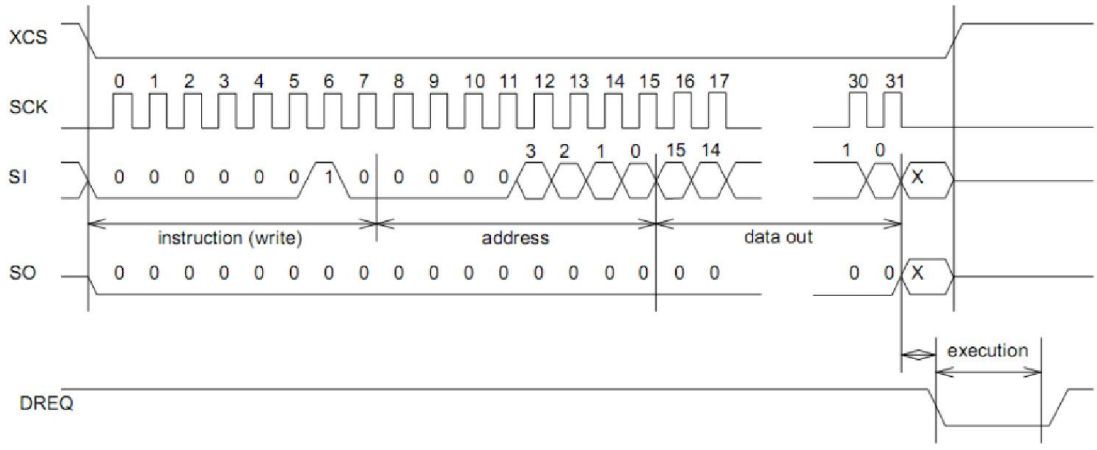


图 7：SCI 字写入操作

这里要注意的是，DREQ 要在高电平的时候，才能进行操作。在寄存器被更新的整个期间，DREQ 会被拉到低电平并维持此状态。

还有个注意的地方是，操作的时候，注意要把 XDCS 选择取消数据片选，也就是将它置为高电平。

所以从上图，我们总结的操作步骤为：

1) 等待 DREQ 为高电平。
2) 在进行命令操作的时候，我们不需要 SPI 的速度很快，所以我们使

用低速的 SPI。

3) 设置 XDCS 为高电平
4) 设置 XCS 为低电平
5) 发送一个字节的指令
6) 发送一个字节的地址
7) 发送两个字节的指令数据
8) 取消 XCS 片选

例程函数实现为：

```
/*****************************************************************
* Function Name  : VS10XX_WriteCmd
* Description    : VS10XX 写入一个命令.
* Input          : addr：写入地址
*                  * cmd：写入命令
* Output         : None
* Return         : 0：写入成功；0xFF：写入失败。
*****************************************************************/

int8_t VS10XX_WriteCmd(uint8_t addr, uint16_t cmd)
{
    uint16_t i = 0;

    while(VS_DREQ == 0)   //等待空闲
    {
        i++;
        if(i > 0xAFFF)
        {
            return 0xFF;
        }
    }

    SPI1_SetSpeed(SPI_BaudRatePrescaler_256);   //设置 SPI1 低速
    VS_XDCS_SET;
    VS_XCS_CLR;

    SPI1_WriteReadData(VS_WRITE_COMMAND);     //发送写命令
    SPI1_WriteReadData(addr);                 //发送地址
    SPI1_WriteReadData(cmd >> 8);             //先发高 8 位
    SPI1_WriteReadData(cmd & 0x00FF);
    VS_XCS_SET;

    SPI1_SetSpeed(SPI_BaudRatePrescaler_2);
    return 0;
}
```

2. SCI 串行命令读操作

命令读的时序如下：

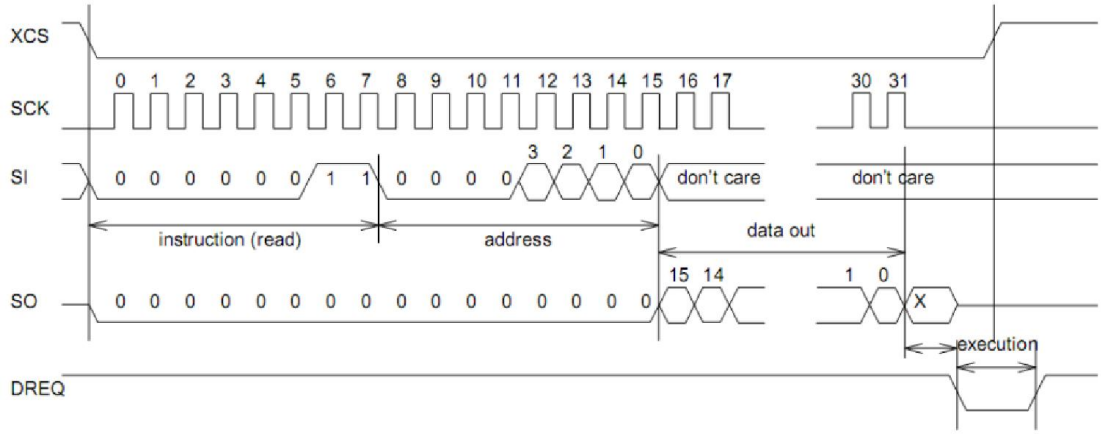


图 6: SCI 字读取操作

那么我们读取的操作步骤为:

1) 等待 DREQ 为高电平
2) 在进行命令操作的时候，我们不需要 SPI 的速度很快，所以我们使用低速的 SPI。
3) 设置 XDCS 为高电平
4) 设置 XCS 为低电平
5) 发送一个字节的指令
6) 发送一个字节的地址
7) 接收两个字节的数据
8) 取消 XCS 片选

例程函数实现如下：

```
/*****************************************************************************
* Function Name  : VS10XX_ReadData
* Description    : VS10XX 读取一个数据.
* Input          : addr：要读取的地址
* Output         : None
* Return         : 读取到的 16 位数据
*****************************************************************************/

uint16_t VS10XX_ReadData(uint8_t addr)
{
    uint16_t readValue, i;

    while(VS_DREQ == 0) //等待空闲
    {
        i++;
        if(i > 0xAFFF)
        {
            return 0xFFFF;
        }
```

```
    }

    SPI1_SetSpeed(SPI_BaudRatePrescaler_256);   //设置 SPI1 低速
    VS_XDCS_SET;
    VS_XCS_CLR;

    SPI1_WriteReadData(VS_READ_COMMAND);     //发送读命令
    SPI1_WriteReadData(addr);                //发送地址
    readValue = SPI1_WriteReadData(0xFF);    //先读高 8 位
    readValue <<= 8;
    readValue |= SPI1_WriteReadData(0xFF);

    VS_XCS_SET;
    SPI1_SetSpeed(SPI_BaudRatePrescaler_2);      //设置 SPI1 速度

    return readValue;
}
```

3. VS1053 软件复位

有些情况下，解码器需要用软件复位。软软复位的方式是通过 SCI_MODE 控制寄存器中的 SM_RESET 来复位。

VS1053 的寄存器很多，我们这里就不一一贴出来了，如果大家需要可以查看数据手册。进入 SDI 测试的控制寄存器是 SCI_MODE 寄存器。它的各个位如下：

位元	名称	功能	值	说明
0	SM_DIFF	差分	0 1	正常的同相音频 左通道反相
1	SM_LAYER12	允许 MPEG layers I & II	0 1	不允许 允许
2	SM_RESET	软件复位	0 1	不用复位 复位
3	SM_CANCEL	取消当前的文件解码	0 1	不取消 取消
4	SM_EARSPEAKER_LO	EarSpeaker 低设定	0 1	关闭 激活
5	SM_TESTS	允许 SDI 测试	0 1	不允许 允许
6	SM_STREAM	流模式	0 1	不是 是
7	SM_EARSPEAKER_HI	EarSpeaker 高设定	0 1	关闭 激活
8	SM_DACT	DCLK 的有效边沿	0 1	上升沿 下降沿
9	SM_SDIORD	SDI 位顺序	0 1	MSb 在前 MSb 在后
10	SM_SDISHARE	共享 SPI 片选	0 1	不共享 共享
11	SM_SDINEW	VS1002 本地 SPI 模式	0 1	非本地模式 本地模式
12	SM_ADPCM	ADPCM 录音激活	0 1	不激活 激活
13	-	-	0 1	正确的 错误的
14	SM_LINE1	咪 / 线路1 选择	0 1	MICP LINE1
15	SM_CLK_RANGE	输入时钟范围	0 1	12..13 MHz 24..26 MHz

所以设置 SCI_MODE 的数据命令为：0x0804。
例程函数为：

```
/*****************************************************************************
* Function Name  : VS10XX_SoftReset
* Description    : 软件复位 VS10xx，同时设置时钟。
* Input          : None
* Output         : None
* Return         : 0：成功；0xFF：失败。
*****************************************************************************/

int8_t VS10XX_SoftReset(void)
{
    uint16_t i = 0;

    while(VS_DREQ == 0)
    {
        i++;
        if(i > 0x5FFF)
        {
            return 0xFF;
        }
    }
    SPI1_WriteReadData(0xFF);

    i = 0;
    while(VS10XX_ReadData(SCI_MODE) != 0x0800)   // 软件复位,新模式
    {
        VS10XX_WriteCmd(SCI_MODE, 0x0804);         // 软件复位,新模式
        VS10XX_DelayMs(2);                         //等待至少 1.35ms
        i++;
        if(i > 0x5FFF)
        {
            return 0xFF;
        }
    }

    i = 0;
    while(VS_DREQ == 0)
    {
        i++;
        if(i > 0x5FFF)
        {
            return 0xFF;
        }
```

```
    }

    i = 0;
    while(VS10XX_ReadData(SCI_CLOCKF) != 0X9800) //设置 VS10XX 的时钟,3
倍频 ,1.5xADD
    {
        VS10XX_WriteCmd(SCI_CLOCKF, 0X9800);        //设置 VS10XX 的时
钟,3 倍频 ,1.5xADD
        VS10XX_DelayMs(20);
        i++;
        if(i > 0x5FFF)
        {
            return 0xFF;
        }
    }

    return 0;
}
```

在上面的复位函数中，VS1053 软件复位完了之后又进行了时钟的设置，它的时钟设置寄存器为 SCI_CLOCKF，它的位意义为：

SCI_CLOCKF bits		
名称	位域	说明
SC_MULT	15:13	时钟乘数
SC_ADD	12:11	允许的附加乘数
SC_FREQ	10: 0	时钟频率

4. SDI 测试

在操作 VS1053 播放一个文件完整之前，一般我们都要先对其初始化，而初始化的过程中要进行各种 SDI 测试，这里我们只介绍会使用到的 SDI 测试。

控制 SDI 测试的寄存器也是 SCI_MODE，上面我们已经结束过 SCI_MODE，这里的设置 0x8020。

1) 正弦测试

正弦测试初始化以 8 个字节序列 0x53、0xEF、0x6E、n、0、0、0、0。其中 n 是用来定义正弦测试的地方，它的定义如下：

n 位组		
名称	位域	说明
FsIdx	7:5	采样率索引
S	4:0	正弦跳跃率

FsIdx	Fs	FsIdx	Fs
0	44100 Hz	4	24000 Hz
1	48000 Hz	5	16000 Hz
2	32000 Hz	6	11025 Hz
3	22050 Hz	7	12000 Hz

将被输出的正弦频率可以按照后面的公式计算：F = Fs * S / 128。

例如：正弦测使用值 126 激活，那将是 0b01111110。拆开 n 的部件，Fsldx = 0b011 = 3, 因此 Fs = 22050Hz。S = 0b11110 = 30，所以正弦的最终频率是 F = 22050Hz *30 / 128 =5168Hz。

而退出正弦测试的命令序列为：0x45、0x78、0x69、0x74、0、0、0、0。

我们进行正弦测试的一般过程为：

a) 进行硬件复位。

硬件复位的过程就是将通过外部 RST 引脚进行一次复位。

b) 发送进入正弦测试的命令

c) 等待 DREQ 变为高电平

d) XDCS 数据片选使能

e) 发送 8 个字节的进入正弦测试序列数据

f) 取消 XDCS 数据片选

g) 延时一段时间，进行正弦测试

h) XDCS 数据片选使能

i) 发送 8 个字节的退出正弦测试序列数据

j) 取消 XDCS 数据片选

例程函数如下：

```
/************************************************************************
* Function Name   : VS_HardwareReset
* Description     : 硬件复位 VS10XX.
* Input           : None
* Output          : None
* Return          : 0：成功；0xFF：失败。
************************************************************************/

int8_t VS10XX_HardwareReset(void)
{
    uint16_t i = 0;

    VS_RST_CLR;
    VS10XX_DelayMs(20);
    VS_XDCS_SET;   //取消数据传输
```

```
    VS_XCS_SET;    //取消数据传输
    VS_RST_SET;

    while(VS_DREQ == 0)
    {
        i++;
        if(i > 0x0FFF)
        {
            return 0xFF;
        }
    }
    VS10XX_DelayMs(20);

    return 0;
}
/*****************************************************************************
* Function Name  : VS10XX_SoftReset
* Description    : 软件复位 VS10xx，同时设置时钟。
* Input          : None
* Output         : None
* Return         : 0：成功；0xFF：失败。
*****************************************************************************/

int8_t VS10XX_SoftReset(void)
{
    uint16_t i = 0;

    while(VS_DREQ == 0)
    {
        i++;
        if(i > 0x5FFF)
        {
            return 0xFF;
        }
    }
    SPI1_WriteReadData(0xFF);

    i = 0;
    while(VS10XX_ReadData(SCI_MODE) != 0x0800)   // 软件复位,新模式
    {
        VS10XX_WriteCmd(SCI_MODE, 0x0804);         // 软件复位,新模式
        VS10XX_DelayMs(2);                         //等待至少 1.35ms
        i++;
        if(i > 0x5FFF)
```

```
        {
            return 0xFF;
        }
    }

    i = 0;
    while(VS_DREQ == 0)
    {
        i++;
        if(i > 0x5FFF)
        {
            return 0xFF;
        }
    }

    i = 0;
    while(VS10XX_ReadData(SCI_CLOCKF) != 0X9800) //设置 VS10XX 的时钟,3
倍频 ,1.5xADD
    {
        VS10XX_WriteCmd(SCI_CLOCKF, 0X9800);        //设置 VS10XX 的时
钟,3 倍频 ,1.5xADD
        VS10XX_DelayMs(20);
        i++;
        if(i > 0x5FFF)
        {
            return 0xFF;
        }
    }

    return 0;
}
```

2) 存储器测试

存储器测试方式以 8 个字节序列 0x4D、0xEA、0x6D、0x54、0、0、0、0 来初始化。测试结果可以从 SCI 寄存器 SCI_HDAT0 读取。SCI_HDAT 各个为的意义如下：

位域	屏蔽	含义
15	0x8000	测试完成
14:10		不使用
9	0x0200	Mux 测试成功
8	0x0100	好的 MAC RAM
7	0x0080	好的 I RAM
6	0x0040	好的 Y RAM
5	0x0020	好的 X RAM
4	0x0010	好的 I ROM 1
3	0x0008	好的 I ROM 2
2	0x0004	好的 Y ROM
1	0x0002	好的 X ROM 1
0	0x0001	好的 X ROM 2
	0x83ff	全部都是好的

例程函数测试为：

```
/*********************************************************************
* Function Name  : VS10XX_RAM_Test
* Description    : 对芯片进行内存测试，读取到 0x8000 表示测试完成，读取到 0x807F
*                    * 是 VS1003 完好，读取到 0X83FF 是表示 VS1053 完好。
*                    * 测试初始化序列为：0x4D,0xEA,0x6D,0x54,0,0,0,0。
* Input          : None
* Output         : None
* Return         : 0xFFFF：测试失败。或返回测试状态
*********************************************************************/

uint16_t VS10XX_RAM_Test(void)
{
    uint16_t i = 0;

    VS10XX_HardwareReset();
    VS10XX_WriteCmd(SCI_MODE, 0x0820); // Allow SCI tests

    while(VS_DREQ == 0)
    {
        i++;
        if(i > 0x0FFF)
        {
            return 0xFFFF;
        }
    }
    SPI1_SetSpeed(SPI_BaudRatePrescaler_256);  //设置 SPI1 低速
    VS_XDCS_CLR;                               // xDCS = 1，选择 VS10XX 的数据接口
```

```
    SPI1_WriteReadData(0x4D);
    SPI1_WriteReadData(0xEA);
    SPI1_WriteReadData(0x6D);
    SPI1_WriteReadData(0x54);
    SPI1_WriteReadData(0x00);
    SPI1_WriteReadData(0x00);
    SPI1_WriteReadData(0x00);
    SPI1_WriteReadData(0x00);

    VS10XX_DelayMs(150);
    VS_XDCS_SET;

    return VS10XX_ReadData(SCI_HDAT0); // VS1003 得到的值为 0x807F;VS1053
为 0X83FF;
}
```

5. VS1053 基本设置

基本设置主要是设置 VS1053 高低频的上限和高低频的增益上限，它要设置的寄存器为：SCI_BASS。它的位意义如下：

名称	位域	说明
ST_AMPLITUDE	15:12	高音控制，步长为 1.5 dB (-8..7, 0 = 关闭)
ST_FREQLIMIT	11:8	下限频率，步长为 1000 Hz (1..15)
SB_AMPLITUDE	7:4	低音增强，步长为 1 dB (0..15, 0 = 关闭)
SB_FREQLIMIT	3:0	频率上限注，步长为 10 Hz (2..15)

这里大家可以根据需要设置，我们例程函数如下：

```
/*****************************************************************************
* Function Name  : MP3_BaseSetting
* Description    : 基本设置：设置 SCI_BASS 寄存器
* Input          : amplitudeH：高频增益 0~15(单位:1.5dB,小于 9 的时候为负数)
*                  * freqLimitH：高频上限 2~15(单位:10Hz)
*                  * amplitudeL：低频增益 0~15(单位:1dB)
*                  * freqLimitL：低频下限 1~15(单位:1Khz)
* Output         : None
* Return         : None
*****************************************************************************/

void MP3_BaseSetting(
    uint8_t amplitudeH, uint8_t freqLimitH,
    uint8_t amplitudeL, uint8_t freqLimitL
)
{
```

```
    uint16_t bassValue = 0;

    /* 高频增益是 12 ： 15 位 */
    bassValue = amplitudeH & 0x0F;
    bassValue <<= 4;

    /* 频率下限是 11 ： 8 位 */
    bassValue |= freqLimitL & 0x0F;
    bassValue <<= 4;

    /* 低频增益是 7 ： 4 位 */
    bassValue |= amplitudeL & 0x0F;
    bassValue <<= 4;

    /* 频率上限是 3 ： 0 位 */
    bassValue |= freqLimitH & 0x0F;

    VS10XX_WriteCmd(SCI_BASS, bassValue);

}
```

6. VS1053 音量设置

音量的设置寄存器是 SCI_VOL 寄存器，它的低 8 位控制右通道，高 8 位控制左通道，所以音量控制大小是只有 1 一个字节长度。这里要注意的是，最大音量是 0，最小音量是 0xFE。

例程函数如下：

```
/*****************************************************************************
* Function Name  : MP3_AudioSetting
* Description    : 设置声音的大小，设置 SCI_VOL 寄存器
* Input          : vol：声音的大小（0~0xFF）
* Output         : None
* Return         : None
*****************************************************************************/

void MP3_AudioSetting(uint8_t vol)
{
    uint16_t volValue = 0;

    /* 0 是最大音量，0xFE 是无声，低 8 字节控制右通道，高 8 字节控制左通道 */
    vol = 254 - vol;
    volValue = vol | (vol << 8);

    VS10XX_WriteCmd(SCI_VOL, volValue);
}
```

31.2 VS1053 播放一个音乐文件

上面我们学习了 VS1053 的一下操作方式，那么我们现在来看一个使用 VS1053 播放一个音乐文件的过程。这个过程可以分为两步：

1. 初始化

这里的初始化包括单片机本身的初始化和 VS1053，使用我们开发板的初始化过程如下：

1) 初始化 STM32 的 IO 口和 SPI
2) VS1053 进行存储器测试
3) 初始化设置音频输出（因为我们的音频输入是通过多路选择器来选择输出的音频是 VS1053 还是 FM 收音机，所以这里的音频输入要设置成 VS1053 进行输出）。
4) VS1053 进行正弦波测试
5) VS1053 进行一些基本设置（如果使用音效还可以设置音效模式）

例程函数如下：

```
/*****************************************************************************
* Function Name  : MP3_Init
* Description    : 初始化 MP3
* Input          : None
* Output         : None
* Return         : None
*****************************************************************************/

int8_t MP3_Init(void)
{
    uint16_t id;
    VS10XX_Config();            //GPIO 和 SPI 初始化
    id = VS10XX_RAM_Test();
    if(id != 0x83FF)            //检测存储器测试的结果
    {
        return 0xFF;
    }

    HC4052_Init();              //初始化多路选择的 GPIO
    VS10XX_SineTest();          //正弦测试
    HC4052_OutputSelect(AUDIO_MP3_CHANNEL); //多路选择器选择 MP3 作为音频输出
    MP3_BaseSetting(0,0,0,0);                   //基本设置
    MP3_EffectSetting(0);                       //音效设置

    return 0;
}
```

2. 输入音频文件数据

在我们例程中，我们播放的是 SD 卡里面的音频数据，所以这里要使用的 FATFS 进行数据的读写，而 FATFS 文件系统，在之前，我们已经学习过了，这里就不详细讲了。

VS1053 可以字节最少 32 字节的 SDI 数据或者接受 SCI 命令。所以这里我们可以选择每次发送 32 个字节的数据。

所以我们这里总结发送音频文件的过程如下：

1) 通过 FATFS 打开音频文件
2) VS1053 进行软件复位（为了保险我们每次播放一个音乐文件之前都进行一次软件复位）
3) 使用 FATFS 读取音频文件数据
4) 发送文件文件，每次发送 32 个字节，直到音频文件发送完成

例程函数如下：

```
/*********************************************************************
* Function Name  : MP3_PlaySong
* Description    : 播放一首歌曲
* Input          : addr：播放地址和歌名（歌曲名记得加.mp3 后缀）
* Output         : None
* Return         : None
*********************************************************************/

void MP3_PlaySong(uint8_t *addr)
{
    FIL file;
    UINT br;
    FRESULT res;
    uint8_t musicBuff[512];
    uint16_t k;

    /*open file*/
    res = f_open(&file, (const TCHAR*)addr, FA_READ);

    VS10XX_SoftReset();
    if(res == FR_OK)
    {

        while(1)
        {
            res = f_read(&file, musicBuff, sizeof(musicBuff), &br);
            k = 0;

            do
            {
                /* 发送歌曲信息 */
```

```
                        if(VS10XX_SendMusicData(musicBuff+k) == 0)
                        {
                            k += 32;
                        }
                        else
                        {
                            /* 按键扫描 */
                            switch(KEY_Scan())
                            {
                                case(KEY_UP):
                                    MP3_Volume++;
                                    MP3_AudioSetting(MP3_Volume);
                                    MP3_ShowVolume(MP3_Volume);
                                    break;
                                case(KEY_DOWN):
                                    MP3_Volume--;
                                    MP3_AudioSetting(MP3_Volume);
                                    MP3_ShowVolume(MP3_Volume);
                                    break;
                                case(KEY_RIGHT):
                                    return;
                                default:
                                    break;
                            }

                        }

                    }
                    while(k < br);

                    if (res || (br == 0))
                    {
                        break;      // error or eof
                    }
                }
                f_close(&file);   //不论是打开，还是新建文件，一定记得关闭
            }
        }
```

31.4 例程主函数

```
FileNameTypeDef filename[30];
int main(void)
{
    FATFS fs;
```

```
uint32_t sdCapacity, free;
uint8_t dat[9] = {"0:/音乐"}, i, j, k;   //要显示的图片的文件地址
uint8_t misicFile[30];

/* 初始化 */
TFT_Init();
while(FLASH_Init() != EN25Q64);
MP3_Init();
KEY_Config();

TFT_ClearScreen(BLACK);
while(SD_Init())
{
    GUI_Show12Char(0, 0, "SD card error!", RED, BLACK);
}

f_mount(0, &fs);
while(FATFS_GetFree("0", &sdCapacity, &free) != 0)
{
    GUI_Show12Char(0, 21, "FATfs error!", RED, BLACK);
}
/* 设置音量 */
MP3_Volume = 230;
MP3_AudioSetting(MP3_Volume);
/* 初始化显示 */
GUI_DisplayInit();

/*  扫描文件地址里面所有的文件 */
FATFS_ScanFiles(dat, filename);

while(1)  //扫描 30 个文件
{
    /* 判断是否是 PM3 图片文件 */
    if((filename[i].type[1] == 'm') && (filename[i].type[2] == 'p') &&
       (filename[i].type[3] == '3'))
    {
        /* 处理文件路径,先添加文件路径 */
        k = 0;
        while(*(dat + k) != '\0')
        {
            *(misicFile + k) = *(dat + k);
            k++;
        }
```

```
                /*  路径之后加上一斜杠  */
                *(misicFile + k) = '/';
                k++;

                /*  添加文件名字名字  */
                j = 0;
                while(filename[i].name[j] != '\0')
                {
                    *(misicFile + k) = filename[i].name[j];
                    k++;
                    j++;
                }

                /*  添加文件后缀  */
                j = 0;
                while(filename[i].type[j] != '\0')
                {
                    *(misicFile + k) = filename[i].type[j];
                    k++;
                    j++;
                }

                /*  文件最后添加一个结束符号  */
                *(misicFile + k) = '\0';

                /*  显示播放的歌曲并播放歌曲  */
                GUI_Box(0, 126, TFT_XMAX, 141, BLACK);       //清除显示位置
                GUI_Show12Char(0, 126, misicFile, RED, BLACK);
                MP3_PlaySong(misicFile);
            }
            i++;
            if(i > 30)
            {
                i = 0;
            }
        }
    }
```