1.1.1 WAV 文件

（1） WAV 文件的介绍

WAV 为微软公司(Microsoft)开发的一种声音文件格式，它符合 RIFF(Resource Interchange File Format)文件规范，用于保存 Windows 平台的音频信息资源，被 Windows 平台及其应用程序所广泛支持，该格式也支持 MSADPCM，CCITT A LAW 等多种压缩运算法，支持多种音频数字，取样频率和声道，标准格式化的 WAV 文件和 CD 格式一样，也是 44.1K 的取样频率，16 位量化数字，因此在声音文件质量和 CD 相差无几!通常使用三个参数来表示声音，量化位数，取样频率和采样点振幅。量化位数分为 8 位，16 位，24 位三种，声道有单声道和立体声之分。取样频率一般有 11025Hz(11.025kHz) ， 22050Hz(22.05kHz)和 44100Hz(44.1kHz) 三种。

（2） WAV 文件的解析

WAV 文件是由若干个 Chunk 组成的，按照在文件中的出现位置包括： RIFF WAVE Chunk、Format Chunk、 Fact Chunk(可选)、LIST 块(可选)和 Data Chunk。 每个 Chunk 由块标识符、数据大小和数据三部分组成。其中块标识符由 4 个 ASCII 码构成， 数据大小则标出紧跟其后的数据的长度（ 单位为字节） ， 注意这个长度不包含块标识符和数据大小的长度， 即不包含最前面的 8 个字节。 所以实际 Chunk 的大小为数据大小加 8。

WAV 文件格式如下表 9.4.1 所示：

表 9.4.1 WAV 文件格式

偏移地址	字节数	类型	内容
00H~03H	4	字符	资源交换文件标志（RIFF），16 进制 52 49 46 46=“RIFF”的 ASCII 码
04H~07H	4	长整数	除去档头 8 个字节后，wav 文件总长度(字节) ，
08H~0BH	4	字符	WAV 文件标志（ WAVE），16 进制 57 41 56 56=“WAVE”的 ASCII 码
0CH~0FH	4	字符	波形格式标志（ FMT），16 进制 66 6D 74 20=“fmt ”的 ASCII 码
10H~13H	4	整数	描述 fmt chunk 的大小，该 chunk 去除 ID 和 size 所占的字节数后剩下的其他字节数目，0x00000010 表示后面的 16 字节用于描述 fmt chunk， 不同的格式数值可能不一样
14H~15H	2	整数	格式种类（ 值为 1 时， 表示数据为线性 PCM 编码，其他格式将是其他数值）
16H~17H	2	整数	通道数， 单声道为 1， 双声音为 2
18H~1BH	4	长整数	采样频率，描述每秒有多少个采样点
1CH~1FH	4	长整数	波形数据传输速率（ 每秒平均字节数），wav 每秒字节数=采样率*bit 解析度*通道数/8，
20H~21H	2	整数	数据的调整数（ 按字节计算），wav 每个数据块字节数，pcm 格式， 一个采样点就是一个块
22H~23H	2	整数	样本数据位数，一般取值是 8/16/32
24H~27H	4	字符	数据标志符（ data），16 进制 64 61 74 61= "data"的 ASCII 码
28H~2BH	4	长整型	采样数据总数，描述 data chunk 的大小，该 chunk 去除 ID 和 size 所占的字节数后剩下的其他字节数目，此信息后的所有字节都是音频数据
2CH~	...		从此位置开始到文件结束，全部用来记录音频信息

由表 9.4.1 可得，WAV 文件组成。

1．RIFF 块（RIFF WAVE Chunk）

该块以“ RIFF” 作为标示，紧跟 WAV 文件大小，该大小是 WAV 文件的总大小减去 8，然后数据段为“ WAVE” ，表示是 WAC 文件。RIFF 块的 Chunk 结构代码可写成如下所示：

```
//RIFF 块
//添加__packed 关键字表示结构体成员间紧密排列
__packed typedef  struct
{
    u32 RiffID;                 //RiffID;这里固定为"RIFF",即 0X46464952
    u32 ChunkSize ;             //集合大小;文件总大小减去 8
    u32 WaveID;                 //格式;WAVE,即 0X45564157
}RIFFChunk ;
```

2．Format 块（ Format Chunk）

该块以“ fmt ”作为标示（注意有个空格），一般情况下，该段的大小为 16 个字节（不包括 ID 和 Size），但是有些软件生成的 WAV 格式，该部分可能有 18 个字节，含有 2 个字节的附加信息。 Format 块结构代码可写成如下所示：

```
__packed typedef  struct
{
  u32 FmtID  ;              //FmtID ;这里固定为"fmt ",即 0X20746D66
  u32 ChunkSize ;           //数值为 16 或 18，18 则最后又附加信息
  u16 AudioFormat;          //音频格式;0X01,表示线性 PCM;0X11 表示 IMA ADPCM
      u16 NumOfChannels;    //通道数量;1,表示单声道;2,表示双声道;
      u32 SampleRate;       //采样率;0X1F40,表示 8Khz
      u32 ByteRate;         //每秒所需字节数
      u16 BlockAlign;       //数据块对齐单位(每个采样需要的字节数)
      u16 BitsPerSample;    //每个采样需要的 bit 数
// u16 ByteExtraData;       //附加信息（可选，通过 Size 来判断有无）
}FMTChunk;
```

3．Fact 块（ Fact Chunk）

该块为可选块，以“ fact”作为标示，不是每个 WAV 文件都有， 在非 PCM 格式的文件中， 一般会在 Format 结构后面加入一个 Fact 块，Fact 块结构代码可写成如下所示：

```
__packed typedef  struct
{
    u32 FACTID;             //FACTID;这里固定为"fact",即 0X74636166;
    u32 FACTSize ;          //子集合大小(不包括 ID 和 Size);这里为:4.
    u32 NumOfSamples;       //采样的数量;
}FACTChunk;
```

4．List 块（List Chunk）

```
__packed typedef  struct
{
    u32 LISTID;             //chunk id;这里固定为"LIST",即 0X74636166;
    u32 LISTSize ;          //子集合大小(不包括 ID 和 Size);这里为:4.
}LISTChunk;
```

5．Data 块 （ Data Chunk）

该块是真正保存 wav 数据的地方，以“ data”'作为该 Chunk 的标示， 然后是数据的大小。Data 块结构代码可写成如下所示：

```
__packed typedef  struct
{
    u32 DATAID;             //DATAID;这里固定为"data",即 0X5453494C
    u32 DATASize ;          //子集合大小(不包括 ID 和 Size)
```

```
}DATAChunk;
```

DATAChunk 数据块中 DATASize 之后就是音乐文件数据部分了。我们可以根据 ChunkFMT 块中的声道数和位数，可以分为如下几种形式,如表 9.4.2 所示：

表 9.4.2 WAV 文件声道及采样

单声道	采样 1	采样 2	采样 3	采样 4	采样 5	采样 6
8 位量化	声道 0	声道 0	声道 0	声道 0	声道 0	声道 0
双声道	采样 1		采样 2		采样 3	
8 位量化	声道 0（左）	声道 1（右）	声道 0（左）	声道 1（右）	声道 0（左）	声道 1（右）
单声道	采样 1		采样 2		采样 3	
16 位量化	声道 0（低字节）	声道 0（高字节）	声道 0（低字节）	声道 0（高字节）	声道 0（低字节）	声道 0（高字节）
双声道	采样 1				采样 2	
16 位量化	声道 0（低字节）	声道 0（高字节）	声道 1（低字节）	声道 1（高字节）	声道 0（低字节）	声道 0（高字节）
单声道	采样 1			采样 2		
24 位量化	声道 0（低字节）	声道 0（中字节）	声道 0（高字节）	声道 0（低字节）	声道 0（中字节）	声道 0（高字节）
双声道	采样 1					
24 位量化	声道 0（低字节）	声道 0（中字节）	声道 0（高字节）	声道 1（低字节）	声道 1（中字节）	声道 1（高字节）

如果是 16 位或 24 位，每个取样为 4 或 6 个字节，低字节在前，高字节在后。在得到这些 WAV 数据之后，通过 I2S 送给 WM8978，就可以播放音乐了。

我们可以用 stc-isp-15xx-v6.85.exe 软件打开一个 WAV 文件，打开后如图 9.4.1 所示：

```
00000h  52 49 46 46 AE 6D A4 00 57 41 56 45 66 6D 74 20  RIFF®m¤.WAVEfmt
00010h  10 00 00 00 01 00 02 00 11 2B 00 00 44 AC 00 00  .........+..D¬..
00020h  04 00 10 00 4C 49 53 54 82 00 00 00 49 4E 46 4F  ....LIST‚...INFO
00030h  49 41 52 54 0A 00 00 00 E7 BD 97 E7 99 BE E5 90  IART....ç½—ç™¾å
00040h  89 00 49 43 4F 50 15 00 00 00 68 74 74 70 3A 2F  ‰.ICOP....http:/
00050h  2F 77 77 77 2E 6F 69 6D 70 33 2E 63 6F 6D 00 00  /www.oimp3.com..
00060h  49 47 4E 52 04 00 00 00 50 6F 70 00 49 4E 41 4D  IGNR....Pop.INAM
00070h  0E 00 00 00 30 38 2E 49 20 4D 69 73 73 20 59 6F  ....08.I Miss Yo
00080h  75 00 49 50 52 44 0D 00 00 00 E8 88 9E E5 8A A8  u.IPRD....è‰¾å¨
00090h  E4 BA BA E7 94 9F 00 00 49 53 46 54 0E 00 00 00  äº²ç”Ÿ..ISFT....
000A0h  4C 61 76 66 35 37 2E 34 31 2E 31 30 30 00 64 61  Lavf57.41.100.da
000B0h  74 61 00 6D A4 00 00 00 00 00 00 00 00 00 00 00  ta.m¤...........
000C0h  00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

图 9.4.1 WAV 文件内容

由图 9.4.1 可得：

1. “52 49 46 46”，这个是 ASCII 字符“RIFF”，这部分是固定格式，表明这是一个 RIFF 文件头。

2. “AE 6D A4 00” ，这个是 WAV 文件的数据大小， 这个大小包括除了前面 4 个字节的所有字节， 也就等于文件总字节数减去 8。 由于 STM32 为小端模式，所以转换为 16 进制为 0xA46DAE，对应是十进制的“10775982”，即为文件的大小。

3．“57 41 56 45 66 6D 74 20”，也是 ASCII 字符“ WAVEfmt”，这部分是固定格式。

4．“10 00 00 00” ， 16 进制为 0x10， 对应十进制数为 16， 这个对应定义中的 ChunkSize 部分的大小，可以看到后面的这个段内容正好是 16 个字节。 一般情况下大小为 16， 此时最后附加信息没有，如果为 18，则上面这个文件多了两个字节的附加信息。

5．“01 00” ， 16 进制为 0x01， 对应十进制数为 1， 这个对应定义中的 AudioFormat，如果为 1 说明音频格式为 PCM，如果为其他，说明是其他格式。

6．“02 00” ， 16 进制为 0x02， 对应十进制数为 2， 这个对应定义中的 NumOfChannels，如果 1 说明是单声道，如果为 2， 说明是双声道。

7．“11 2B 00 00”，16 进制为 0x2B11，对应十进制数为 11025，代表的是采样频率 11025，采样率（每秒样本数），表示每个通道的播放速度。

8．“44 AC 00 00”，16 进制为 0xAC44，对应十进制数为 44100，代表的是每秒的数据量，波形音频数据传送速率，其值为通道数×每秒样本数×每样本的数据位数 / 8（1*22050*16/8）。播放软件利用此值可以估计缓冲区的大小。

9．“04 00”，16 进制为 0x04，对应十进制数为 4， 表示块对齐的内容。 数据块的调整数（ 按字节算的），其值为通道数×每样本的数据位值 / 8。 播放软件需要一次处理多个该值大小的字节数据， 以便将其值用于缓冲区的调整。

10．“10 00”， 16 进制为 0x10，对应十进制数为 16，即采样大小为 16Bits，每样本的数据位数，表示每个声道中各个样本的数据位数。如果有多个声道，对每个声道而言，样本大小都一样。

11．“4C 49 53 54”，这个是 ASCII 字符“LIST”，这部分是固定格式，表明这是 LIST 文件部分。

12．“64 61 74 61”，这个是 ASCII 字符“DATA”，表示文件头结束，开始数据区域。

13．“00 6D A4 00” 十六进制数是“0xA46D00” ,对应十进制 10775808， 是数据区的开头，之后是数据总数，看一下前面正好可以看到，文件大小是 10775982， 从（2） 到（12） 包括（12）正好是 10775982－10775808=174 字节。

14．之后就是真正的 Wave 文件的数据。

1.1.2 WM8978 详解

（1） WM8978 的介绍

WM8978 是一个低功耗、高质量的立体声多媒体数字信号编译码器。它主要应用于便携式应用，比如数码照相机、可携式数码摄像机。

它结合了立体声差分麦克风的前置放大与扬声器、耳机和差分、立体声线输出的驱动， 减少了应用时必需的外部组件，比如不需要单独的麦克风或者耳机的放大器。

高级的片上数字信号处理功能，包含一个 5 路均衡功能，一个用于 ADC 和麦克风或者线路输入之间的混合信号的电平自动控制功能， 一个纯粹的录音或者重放的数字限幅功能。另外在 ADC 的线路上提供了一个数字滤波的功能，可以更好的应用滤波，比如“减少风噪声”。

WM8978 可以被应用为一个主机或者一个从机。基于共同的参考时钟频率，比如 12MHz 和 13MHz，内部的 PLL 可以为编译码器提供所有需要的音频时钟。

WM8978 工作在模拟电源电压 2.5V 到 3.3V，尽管它的数字核心部分为了节省电能可以把工作电压下降到 1.62V。如果需要增大输出功率， 扬声器和 OUT3/4 线输出可以在 5V 电源运行。芯片的个别部分也可以通过软件进行断电控制。

（2） WM8978 的特点

- DAC 的信噪比为 98dB， 总谐波失真为-84dB
- ADC 的信噪比为 90dB， 总谐波失真为-80dB
- 立体声差分或者单声道麦克风接口
- 为驻极体麦克风提供低噪音偏置
- 增强的 3D 功能用于提高立体声分离
- 5 路均衡器（录音或者重放）
- 可调的 ADC 高通滤波器（减少风噪声）
- 低功耗、低电压
- I2S 接口，支持最高 192K，24bit 音频播放

- 左右声道音量独立调节

（3） WM8978 接口

WM8978 的控制通过 IIS 接口（数字音频接口）跟 MCU 进行音频数据传输（支持音频接收和发送）， 通过两线（MODE=0，IIC 接口）或三线（ MODE=1）接口进行配置。 WM8978 的 I2S 接口，由 4 个引脚组成：

1．ADCDAT：ADC 数据输出
2．DACDAT：DAC 数据输入
3．LRC：数据左/右对齐时钟
4．BCLK：位时钟，用于同步

WM8978 芯片可以作为 IIS 主机，用于输出 LRC 和 BLCK 时钟，本实验中，我们把 WM8978 作为从机使用，用于接收 LRC 和 BLCK。

此外， WM8978 的 IIS 接口一共支持 5 中不同的音频数据模式：

1．左对齐标准（ MSB）
2．右对齐标准（ LSB）
3．飞利浦标准（ IIS）
4．DSP 模式 A
5． DSP 模式 B

本实验中，使用的是飞利浦标准来传输 I2S 数据。

飞利浦标准模式（IIS），指的是数据在跟随 LRC 传输的 BCLK 的第二个上升沿时传输 MSB，其他位一直到 LSB 按顺序传输。传输依赖于字长、BCLK 频率和采样率，在每个采样的 LSB 和下一个采样的 MSB 之间都应该有未用的 BCLK 周期。飞利浦标准模式（IIS）数据传输协议如图 9.4.2 所示：

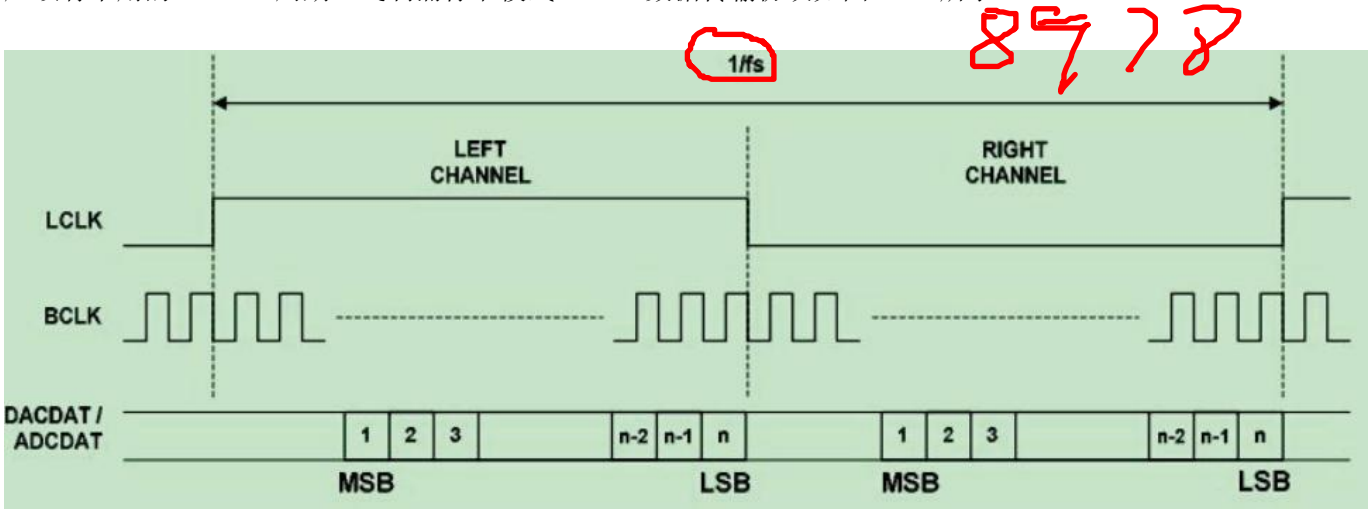


图 9.4.2 飞利浦标准模式时序图

从图中可知，fs 即音频信号的采样率，比如 44.1Khz，因此， LRC 的频率就是音频信号的采样率。 此外，WM8978 还需要一个 MCLK，本实现中采用 STM32F4 为 WM8978 提供一个 MCLK 时钟， MCLK 的频率为音频采样率的 256 倍，即 MCLK=256 * fs。

（4） **WM8978 框图**

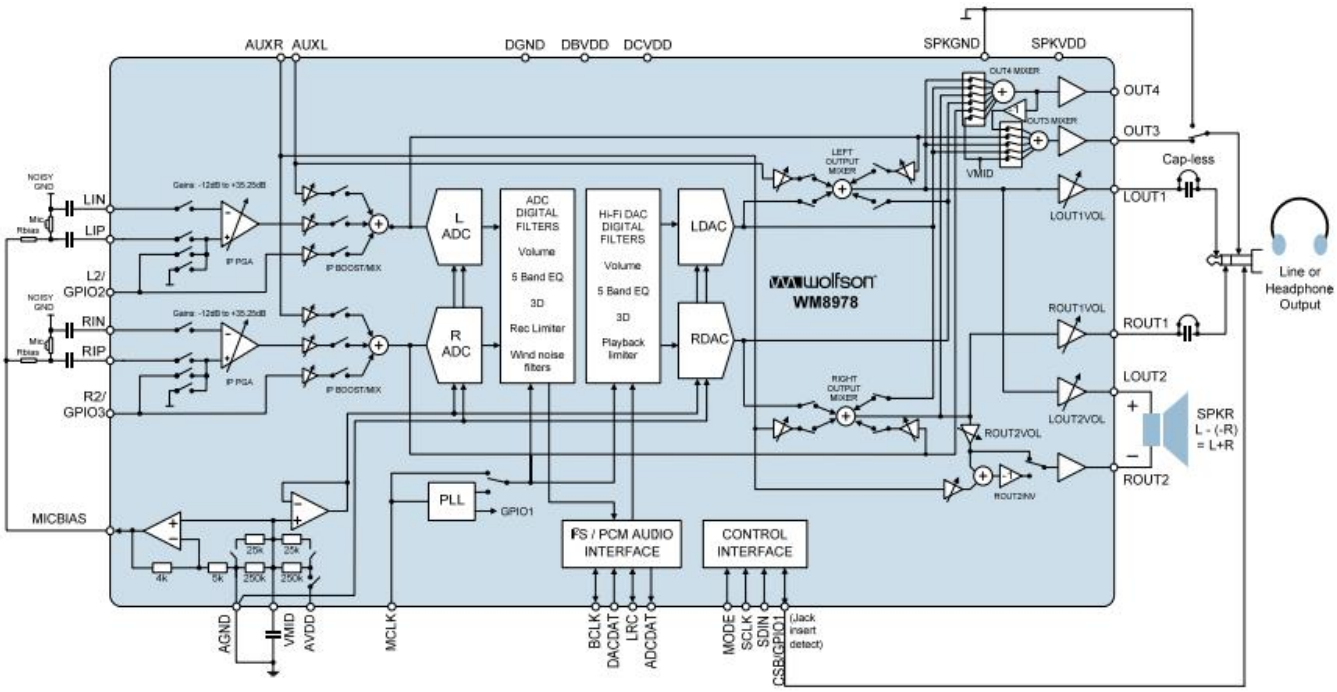


图 9.4.3 WM8978 框图

由图 9.4.3 可知，WM8978 内部有很多的模拟开关，用来选择通道，同时还有很多调节器，用来设置增益和音量的。本实现中，使用的是 IIC 接口（ MODE=0）来跟 WM8978 进行通信，不过，WM8978 的 IIC 接口跟我们之前学习的 IIC 接口有一些不同之处：

1．只可以写数据，不可以读数据
2．寄存器的位宽为 7 位，数据的位宽为 9 位
3．寄存器字节的最低位用于传输数据的最高位的数据
4．WM8978 的器件地址为：0011010 b（0X1A）

（5） **WM8978 寄存器**

WM8978 芯片有很多的寄存器，在这里，我们只介绍比较常用的那些寄存器。

1．R0 寄存器（00H）。此寄存器功能用于控制 WM8978 的软件复位，写任意值到该寄存器地址，即可实现 WM8978 的软件复位。

2．R1 寄存器（01H）。此寄存器需要使能 BIASEN（Bit3）位，该位置 1，模拟部分的放大器才会工作，只有使能了该位才可以听到声音。

3．R2 寄存器（02H）。此寄存器需要设置 ROUT1EN （Bit8）位、LOUT1EN（Bit7）和 SLEEP（Bit6）这三个位， ROUT1EN 和 LOUT1EN 位设置为 1，使能耳机输出， SLEEP 设置为 0，进入正常工作模式。

4．R3 寄存器（ 03H）。此存器需要要设置 LOUT2EN （Bit6）， ROUT2EN （Bit5）， RMIXER（Bit3），LMIXER（Bit2）， DACENR（Bit1）和 DACENL（Bit0）这 6 个位。其中， LOUT2EN 和 ROUT2EN 位设置为 1，使能喇叭输出； LMIXER 和 RMIXER 位设置为 1， 使能左右声道混合器； DACENL 和 DACENR 位设置为 1，使能则左右声道的 DAC。

5． R4 寄存器（ 04H）。此寄存器需要要设置 WL（Bit6:5）和 FMT（Bit4:3）这 4 个位。 WL（Bit6:5）用于设置字长，即音频数据有效位数。00 表示 16 位音频，10 表示 24 位音频；FMT（Bit4:3）位用于设置 IIS 音频数据模式，这里设置为 10，表示 使用飞利浦模式（IIS）。

6． R6 寄存器（ 06H）。此寄存器全部设置为 0 ，即设置 MCLK 和 BCLK 时钟源都来源于外部，本实验来源于 STM32F4 芯片。

7． R10 寄存器（0AH）。此寄存器需要设置 SOFTMUTE（Bit6）和 DACOSR128（Bit3）这两个位，其中 SOFTMUTE 位设置为 0，即关闭软件静音；DACOSR128 位设置为 1，即 DAC 得到最好的 SNR。

8． R43 寄存器（2BH）。此寄存器只需要把 INVROUT2 位置 1 ，即反转 ROUT2 输出，这样以使于更好的驱动喇叭。

9. R49 寄存器（31H）。此寄存器需要设置 SPKBOOST（Bit2）和 TSDEN（Bit1）这两个位。SPKBOOST 位用于设置喇叭的增益，可以设置为 0 ，如果想获得更大的声音，则设置为 1； TSDEN 位用于设置过热保护，如果需要开启过热保护，则设置为 1，不开启设置为 0 即可。

10. R50 寄存器（32H）。此寄存器只需要设置 DACL2LMIX（Bit0），该位用于设置将左声道的 DAC 输出接入左声道混合器里面，才能在耳机或者喇叭听到音乐。

11. R51 寄存器（33H）。此寄存器只需要设置 DACR2RMIX（Bit0），该位用于设置将右声道的 DAC 输出接入右声道混合器里面，才能在耳机或者喇叭听到音乐。

12. R52 寄存器（34H）。此寄存器用于设置耳机左声道的音量，HPVU（Bit8）用于设置是否更新左声道的音量。LOUT1VOL（Bit5:0）位用于设置左声道的音量。

13. R53 寄存器（35H）。此寄存器用于设置耳机右声道的音量，HPVU（Bit8）用于设置是否更新右声道的音量。ROUT1VOL（Bit5:0）位用于设置右声道的音量。

14. R54 寄存器（36H）。此寄存器用于设置喇叭左声道的音量，HPVU（Bit8）用于设置是否更新左声道的音量。LOUT1VOL（Bit5:0）位用于设置左声道的音量。

15. R55 寄存器（37H）。此寄存器用于设置喇叭右声道的音量，HPVU（Bit8）用于设置是否更新右声道的音量。ROUT1VOL（Bit5:0）位用于设置右声道的音量。

1.1.3 IIS 详解

（1）IIS 的介绍

IIS，Inter IC Sound 的简称，又称集成电路内置音频总线，是飞利浦公司为数字音频设备之间的音频数据传输而制定的一种总线标准，该总线专责于音频设备之间的数据传输，广泛应用于各种多媒体系统。它采用了沿独立的导线传输时钟与数据信号的设计，通过将数据和时钟信号分离，避免了因时差诱发的失真，为用户节省了购买抵抗音频抖动的专业设备的费用。

（2） IIS 的特点

- 全双工通信或半双工通信
- 主模式或从模式操作
- 8 位可编程线性预分频器，可实现精确的音频采样频率（从 8 kHz 到 192 kHz）
- 数据格式可以是 16 位、 24 位或 32 位
- 数据包帧由音频通道固定为 16 位（可容纳 16 位数据帧）或 32 位（可容纳 16 位、24 位、32 位数据帧）
- 可编程的时钟极性
- 发送和接收使用同一个 16 位数据寄存器
- 支持的 IIS 协议：
  - I2S Phillps 标准
  - MSB 对齐标准（左对齐）
  - LSB 对齐标准（右对齐）
  - PCM 标准
- 数据方向始终为 MSB 在前
- 用于发送和接收的 DMA 功能（ 16 位宽）
- 可输出主时钟以驱动外部音频元件。比率固定为 256 * F_S（其中 F_S 为音频采样频率）

（3） IIS 的框图

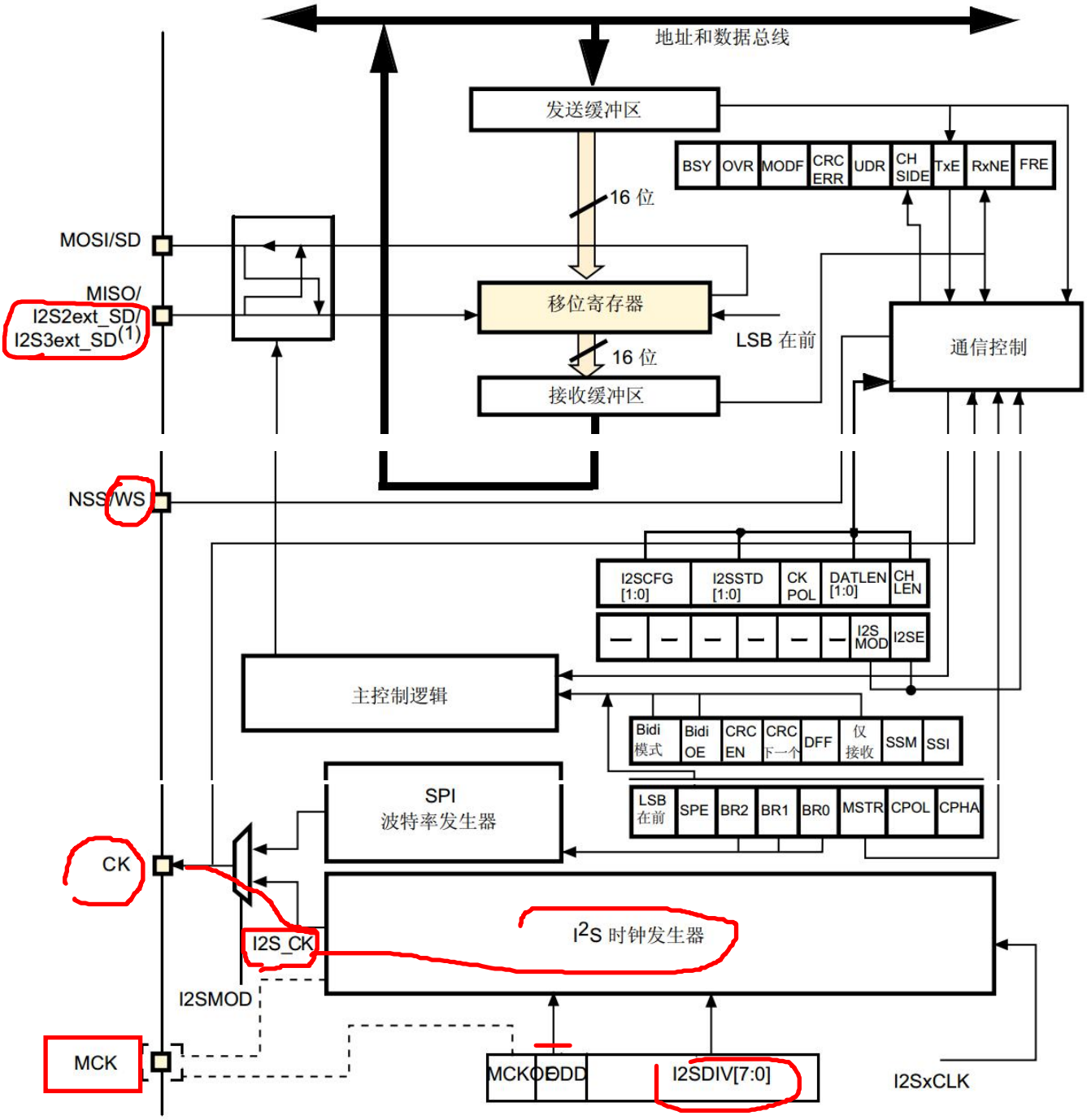


图 9.4.4 IIS 框图

STM32F4 的 IIS 是与 SPI 部分共用的，我们可以通过设置 SPI_I2SCFGR 寄存器的 I2SMOD 位来开启 IIS 功能， IIS 接口使用了几乎与 SPI 相同的引脚、标志和中断。由图 9.4.4 可知，IIS 使用的信号线主要有：

1．SD：串行数据。此信号线映射到 SPI 中的 MOSI 引脚，用于发送或接收两个时分复用的数据通道上的数据，仅在半双工模式下使用。

2．I2S2ext_SD / I2S3ext_SD：用于控制 I2S 全双工模式的附加引脚，此引脚映射到 SPI 中的 MISO 引脚。

3．WS：字选择。此信号线映射到 SPI 中的 NSS 引脚，此引脚在 IIS 中为帧时钟，用于切换左右声道的数据。WS 频率等于音频信号采样率（F_S）。

4． CK：串行时钟。此信号线映射到 SPI 中的 SCK 引脚，此引脚在 IIS 中为位时钟，是主模式下的串行时钟输出以及从模式下的串行时钟输入。 CK 频率=WS 频率（F_S） * 2 * 16（16 位宽），如果是 32 位宽，则是： CK 频率=WS 频率（F_S）* 2 * 32（ 32 位宽）。

5．MCK：主时钟输出。 当 IIS 配置为主模式，并且 SPI I2SPR 寄存器中的 MCKOE 位设置为 1 时，则使用该时钟，该时钟输出频率为 256 * F_S，其中 F_S 为采样频率。

为支持 IIS 全双工模式，除了 IIS2 和 IIS3，还可以使用两个额外的 IIS，它们称为扩展 IIS（IIS2_ext、IIS3_ext）如图 9.4.5 所示。

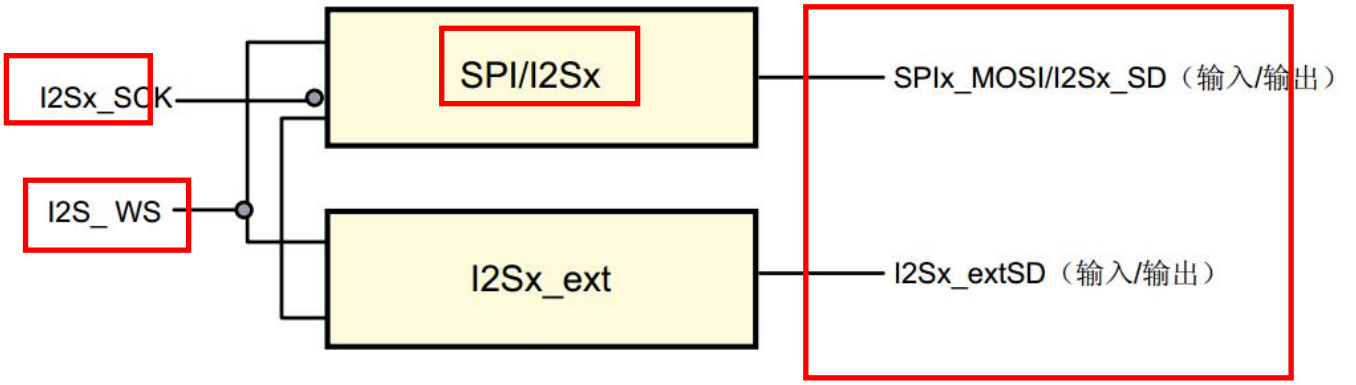


图 9.4.5 IIS 全双工框图

由图 9.4.5 可知，第一个 IIS 全双工接口基于 IIS2 和 IIS2_ext，第二个基于 IIS3 和 IIS3_ext。IIS2_ext 和 IIS3_ext 仅用于全双工模式。

IISx 可以在主模式下工作。因此：

- 只有 IISx 可在半双工模式下输出 SCK 和 WS
- 只有 IISx 可在全双工模式下向 IIS2_ext 和 IIS3_ext 提供 SCK 和 WS。

扩展 IIS (IISx_ext) 只能用于全双工模式。 IISx_ext 始终在从模式下工作。IISx 和 IISx_ext 均可用于发送和接收。

（4） 音频协议

STM32F4 有四种数据和帧格式组合，可采用下列格式发送数据：

1. 将 16 位数据封装在 16 位帧中
2. 将 16 位数据封装在 32 位帧中
3. 将 24 位数据封装在 32 位帧中
4. 将 32 位数据封装在 32 位帧中

当使用 32 位数据包中的 16 位数据时，高 16 位 (MSB) 为有效位， 低 16 位 LSB 被强制清零，无需任何软件操作或 DMA 请求（只需一个读/写操作）。如果应用程序首选 DMA，则 24 位和 32 位数据帧需要对 SPI_DR 执行两次 CPU 读取或写入操作，或者需要两次 DMA 操作。 24 位的数据帧，硬件会将 8 位非有效位扩展到带有 0 位的 32 位。对于所有数据格式和通信标准而言，始终会先发送最高有效位（ MSB 优先）。

（5） IIS Philips 标准

使用 WS 信号来指示当前正在发送的数据所属的通道，该信号从当前通道数据的第一个位(MSB) 之前的一个时钟开始有效。以 24 位为例 IIS Philips 协议波形图如图 9.4.6 所示：

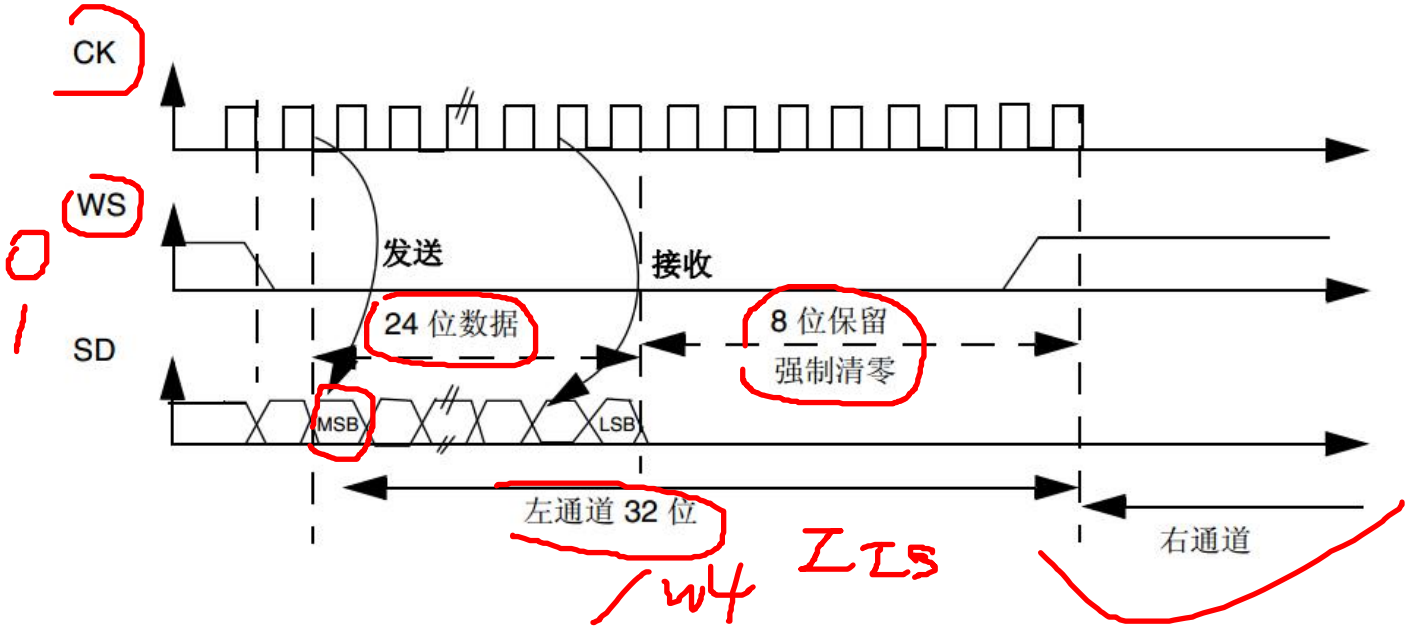


图 9.4.6 IIS Philips 标准波形（ 24 位帧， CPOL = 0）

由图 9.4.6 可知，在 24 位模式下数据传输，需要对 SPI_DR 执行两次读取或写入操作，比如我们要发送 0X8EAA33 这个数据，就要分两次写入 SPI_DR，第一次写入：0X8EAA，第二次写入 0X33xx（ xx 可以为任意数值），这样就把 0X8EAA33 发送出去了。

需要注意的是， 当我们从 SD 卡读取 24 位 WAV 数据流时，是低字节在前，高字节在后的，例如，当我们读到一个声道的数据（24bit）存储在一个 buf[3]的数据中，那么需要通过 SPI_DR 寄存器来发送这 24 位数据，那么就需要分两次来发送，第一次发送高 16 位数据，第二次发送低 8 位数据。代码可按以下过程来操作：

```
SPI_DR=((u16)buf[2]<<8)+buf[1]; //发送高 16 位数据
SPI_DR=(u16)buf[0]<<8;          //发送低 8 位数据
```

（6） IIS 时钟

IIS 比特率用来确定 IIS 数据线上的数据流和 IIS 时钟信号频率。

IIS 比特率 = 每个通道的位数 * 通道数 * 音频采样频率。

对于 16 位双通道音频， IIS 比特率的计算公式如下：

IIS 比特率 = 16 × 2 × F_S

如果数据包为 32 位宽，则 IIS 比特率 = 32 x 2 x F_S。

IIS 时钟发生器架构如图 9.4.7 所示：

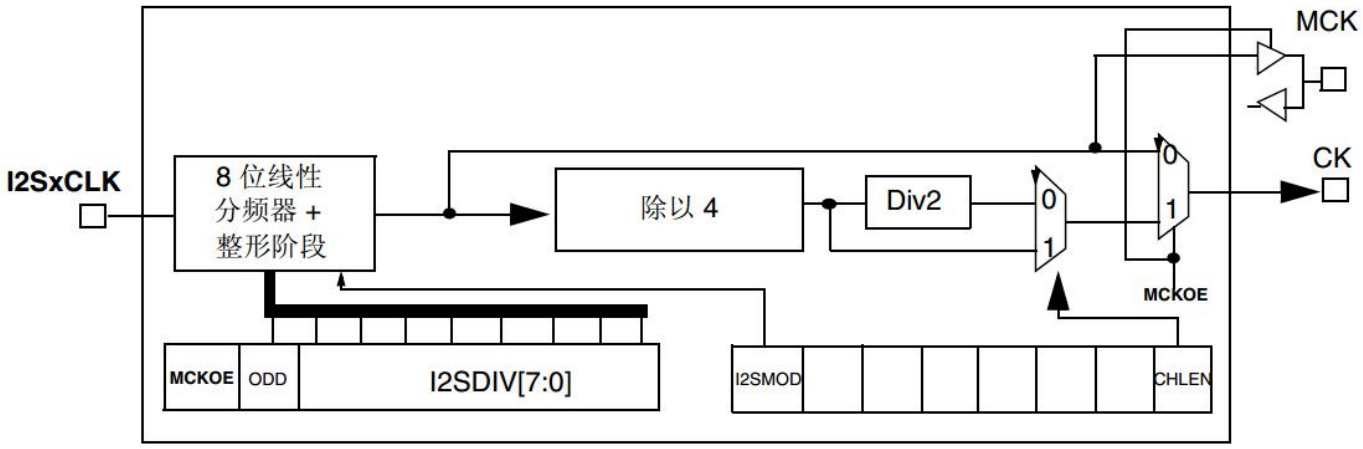


图 9.4.7 IIS 时钟发生器架构

由图 9.4.7 可知， I2SxCLK 时钟源来自于 PLLI2S 输出（通过 R 分频系数）或外部时钟（映射到 I2S_CKIN 引脚）。本实验中使用 PLLI2S 输出作为 I2SxCLK 的时钟。

我们需要根据音频采样率（ Fs，即 CK 的频率）来计算各个分频器的值，常用的音频采样率有： 22.05Khz、44.1Khz、 48Khz、 96Khz、 196Khz 等。为达到所需频率，需要根据以下公式对线性分频器进行编程：

输出主时钟（ SPI_I2SPR 寄存器中的 MCKOE 置 1)时：

Fs = I2SxCLK / [(16*2)*((2*I2SDIV)+ODD)*8)]（通道帧宽度为 16 位时）

Fs = I2SxCLK / [(32*2)*((2*I2SDIV)+ODD)*4)]（通道帧宽度为 32 位时）

关闭主时钟输出（ MCKOE 位清零）时：

Fs = I2SxCLK / [(16*2)*((2*I2SDIV)+ODD))]（通道帧宽度为 16 位时）

Fs = I2SxCLK / [(32*2)*((2*I2SDIV)+ODD))]（通道帧宽度为 32 位时）

输出主时钟时，音频频率精度可按表 9.4.3 所示：

表 9.4.3 音频频率精度

主时钟	目标 Fs (HZ)	数据格式	PLLI2SN	PLLI2SR	I2SDIV	I2SODD	实时 Fs (HZ)	误差
	8000	无关	256	5	12	1	8000	0.0000%
	16000	无关	213	2	13	0	16000.60059	0.0038%
	32000	无关	213	2	6	1	32001.20117	0.0038%

输出使能	48000	无关	258	3	3	1	47991.07031	0.0186%
	96000	无关	344	2	3	1	95982.14063	0.0186%
	22050	无关	429	4	9	1	22049.75391	0.0011%
	44100	无关	271	2	6	0	44108.07422	0.0183%

1.1.4 STM32F4 IIS 寄存器介绍

（1） SPI_I2S 配置寄存器 (SPI_I2SCFGR)

偏移地址： 0x1C
复位值： 0x0000

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved				I2SMOD	I2SE	I2SCFG		PCMSYNC	Reserved	I2SSTD		CKPOL	DATLEN		CHLEN
				rw	rw	rw	rw	rw		rw	rw	rw	rw	rw	rw

位15：12	保留，始终为0
位11	I2SMOD： I2S 模式选择 (I2S mode selection) 0：选择 SPI 模式 1：选择 I2S 模式 注意： 应在 SPI 或 I2S 禁止时配置此位。
位 10	I2SE： I2S 使能 (I2S Enable) 0：关闭 I2S 外设 1：使能 I2S 外设 注意： 不适用于 SPI 模式
位 9:8	I2SCFG： I2S 配置模式 (I2S configuration mode) 00：从模式 - 发送 01：从模式 - 接收 10：主模式 - 发送 11：主模式 - 接收 注意： 应在 I2S 禁止时配置此位。 不适用于 SPI 模式
位 7	PCMSYNC： PCM 帧同步 (PCM frame synchronization) 0：短帧同步 1：长帧同步 注意： 只有在 I2SSTD = 11（使用 PCM 标准）时，此位才有意义 不适用于 SPI 模式
位 6	保留，始终为0
位 5:4	I2SSTD： I2S 标准选择 (I2S standard selection) 00： I2S Philips 标准。 01： MSB 对齐标准（左对齐） 10： LSB 对齐标准（右对齐） 11： PCM 标准
位 3	CKPOL：空闲状态的时钟电平 (Steady state clock polarity) 0：空闲状态时钟为低电平 1：空闲状态时钟为高电平 注意： 为确保正确运行，应在 I2S 关闭时配置此位。
位 2:1	DATLEN：传输的数据长度 (Data length to be transferred) 00： 16 位数据长度 01： 24 位数据长度

	10： 32 位数据长度 11：不允许 注意： 为确保正确运行，应在 I2S 关闭时配置这些位。
位 0	CHLEN：通道长度（每个音频通道的位数） (Channel length (number of bits per audio channel)) 0： 16 位 1： 32 位 只有在 DATLEN = 00 时，此位的写操作才有意义，否则无论填入何值，通道长度始终由硬件固定为 32 位。 不适用于 SPI 模式。 注意： 为确保正确运行，应在 I2S 关闭时配置此位。

（2） SPI_I2S 预分频器寄存器 (SPI_I2SPR)

偏移地址： 0x20
复位值： 0000 0010 (0x0002)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved						MCKOE	ODD	I2SDIV							
						rw	rw	rw							

位15：10	保留，始终为0
位 9	MCKOE：主时钟输出使能 (Master clock output enable) 0：禁止主时钟输出 1：使能主时钟输出 注意： 应在 I2S 禁止时配置此位。只有在 I2S 为主模式时，才会使用此位。 不适用于 SPI 模式。
位 8	ODD：预分频器的奇数因子 (Odd factor for the prescaler) 0：实际分频值为 = I2SDIV *2 1：实际分频值为 = (I2SDIV * 2)+1
位 7:0	I2SDIV： I2S 线性预分频器 (I2S Linear prescaler) I2SDIV [7:0] = 0 或 I2SDIV [7:0] = 1 为禁用值。

（3） RCC PLLI2S 配置寄存器 (RCC_PLLI2SCFGR)

偏移地址： 0x84
复位值： 0x2000 3000

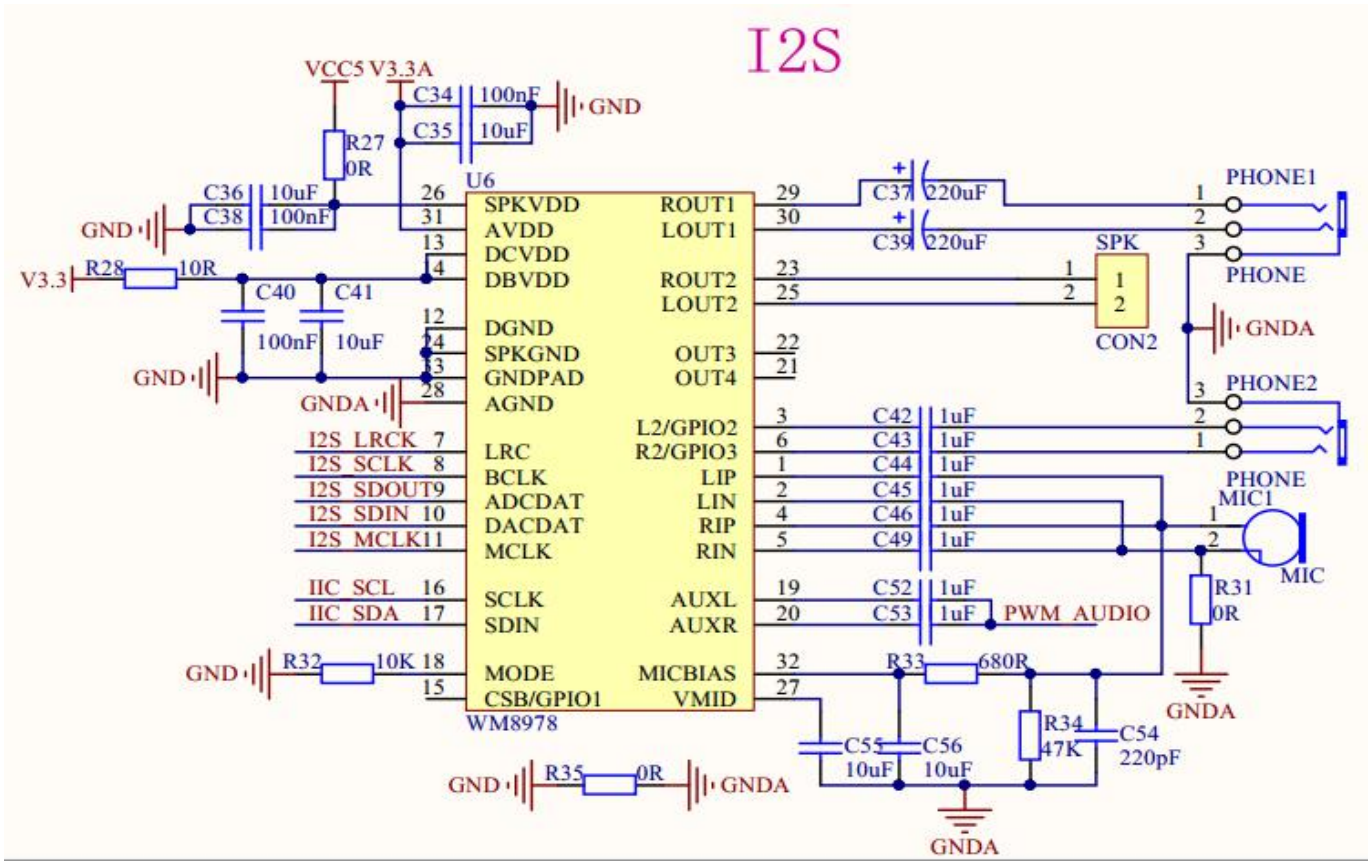
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved	PLLI2SR2	PLLI2SR1	PLLI2SR0	Reserved											
	rw	rw	rw												

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Reserved	PLLI2SN8	PLLI2SN7	PLLI2SN6	PLLI2SN5	PLLI2SN4	PLLI2SN3	PLLI2SN2	PLLI2SN1	PLLI2SN0	Reserved					
	rw	rw	rw	rw	rw	rw	rw	rw	rw						

位 31	保留，始终为0
位 30:28	PLLI2SR： 适用于 I2S 时钟的 PLLI2S 分频系数 (PLLI2S division factor for I2S clocks) 由软件置 1 和清零，用于控制 I2S 时钟频率。这些位应仅在 PLLI2S 已禁止时写入。选择的系数必须与 I2S 外设中的预分频值一致，以达到使用标准晶振时误差不超过 0.3%，使用音频晶振时误差为 0%。 小心：I2S 的正常工作频率要求低于或等于 192 MHz。 I2S 时钟频率 = VCO 频率 × PLLR 并且 2 PLLR 7 000： PLLR = 0，错误配置 001： PLLR = 1，错误配置

	010： PLLR = 2 ... 111： PLLR = 7
位 27:15	保留，始终为0
位 14:6	PLLI2SN： 用于 VCO 的 PLLI2S 倍频系数 (PLLI2S multiplication factor for VCO) 由软件置 1 和清零，用于控制 VCO 的倍频系数。这些位只能在 PLLI2S 已禁止时写入。写入这些位时只允许使用半字和字访问。 小心： 软件必须正确设置这些位，确保 VCO 输出频率介于 192 和 432 MHz 之间。 VCO 输出频率 = VCO 输入频率 × PLLI2SN 并且 192 PLLI2SN 432 000000000： PLLI2SN = 0，错误配置 000000001： PLLI2SN = 1，错误配置 ... 011000000： PLLI2SN = 192 011000001： PLLI2SN = 193 011000010： PLLI2SN = 194 ... 110110000： PLLI2SN = 432 110110000： PLLI2SN = 433，错误配置 ... 111111111： PLLI2SN = 511，错误配置
位 5:0	保留，始终为0

1.1.5 音乐播放硬件设计



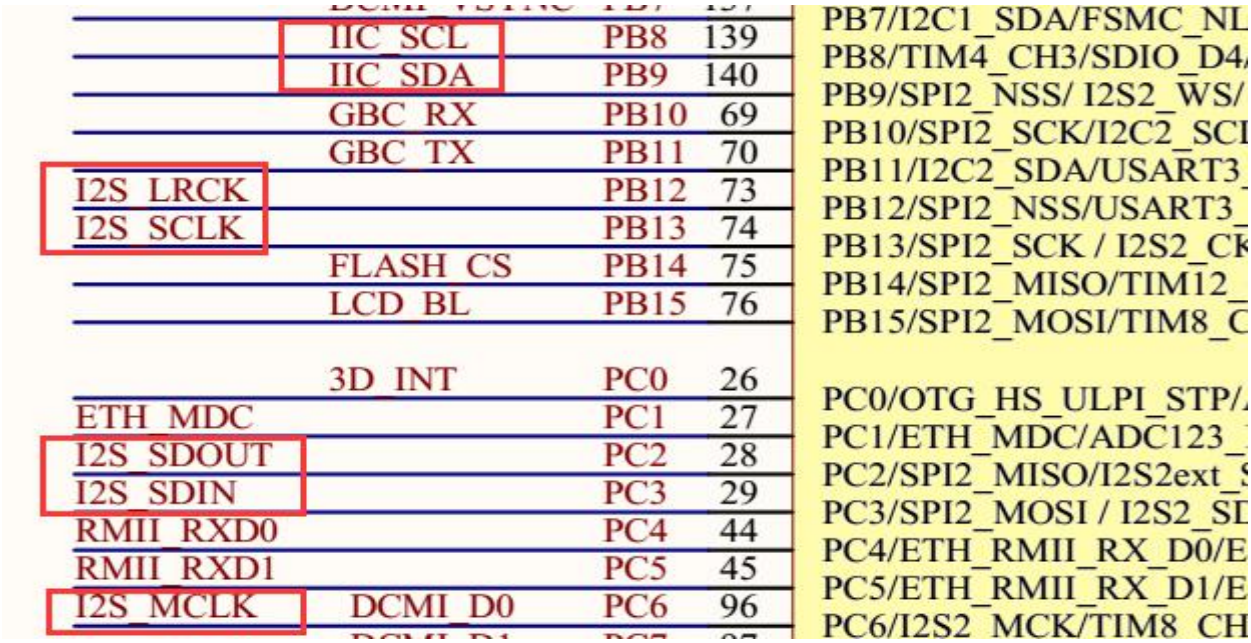


图 9.4.8 音乐播放硬件原理图

1.1.6 音乐播放软件设计

（1） 软件设计思想

1．初始化 SD 卡。本实验中需要播放 SD 卡中的 WAV 格式的音乐文件，因此需要初始化 SD 卡。

2．初始化 WM8978 芯片。配置 9.5.3 章节介绍的 WM8978 芯片的相关寄存器。主要包括软件复位、DAC 设置，输出设置和音量设置等。

3．初始化 IIS。主要设置 SPI_I2SCFGR 寄存器，设置 I2S 模式、 I2S 标准、时钟空闲电平和数据帧长等，最后开启 I2S TX DMA，使能 I2S 外设。

4．解析 WAV 文件。解析文件主要是获取音频信号采样率和位数并设置 I2S 时钟分频器。

5．设置 DMA。本实验中使用到 DMA 来传输数据，并且我们使用 IIS2，其 TX 是使用的 DMA1 数据流 4 的通道 0 来传输的。STM32F4 的 DMA 支持双缓冲功能，这样可以提高数据传输的效率。本实验中我们需要将 DMA1 数据流 4 设置为：双缓冲循环模式，外设和存储器都是 16 位宽，并开启 DMA 传输完成中断。

6．FATFS 文件系统打开播放文件。

7．启动 DMA 传输数据。播放音乐。

（2） 部分示例程序编程及分析

1．wm8978.c 文件中。

a) WM8978 芯片初始化。

```
/*****************************************************************
*   函数名：    u8 WM8978_Init(void)
*   功能描述: WM8978 芯片初始化
*   输入参数: 无
*   输出参数: 无
*   返回值:    WM8978 初始化结果。0：初始化成功。1：初始化不成功
*   其他:
*   作者:
*****************************************************************/
u8 WM8978_Init(void)
{
    u8 res;
    RCC->AHB1ENR|=1<<1;             //使能外设 PORTB 时钟
    RCC->AHB1ENR|=1<<2;             //使能外设 PORTC 时钟
```

```
    //配置 PB 口  PB12/13  复用功能输出
    GPIOB->MODER    &= ~( 0x03<<24 | 0x03<<26 );
    GPIOB->MODER    |=    ( 0x02<<24 | 0x02<<26);
    GPIOB->OTYPER &= ~( 0X01<<12 | 0X01<<13);    //输出类型----推挽
    GPIOB->OSPEEDR |= ( 0x03<<24 | 0x03<<26);//输出速度----100M
    //配置 PC 口  PC2/PC3/PC6 复用功能输出
    GPIOC->MODER    &= ~( 0x03<<4 | 0x03<<6 | 0x03U<<12);
    GPIOC->MODER    |=    ( 0x02<<4 | 0x02<<6 | 0x02U<<12);
    GPIOC->OTYPER &= ~( 0X01<<2 | 0X01<<3 | 0X01<<6);      //输出类型----推挽
    GPIOC->OSPEEDR |= ( 0x03<<4 | 0x03<<6 | 0x03U<<12);//输出速度----100M

    //配置 IO 口复用功能
    Set_GPIO_AF(GPIOB,12,5);
    Set_GPIO_AF(GPIOB,13,5);
    Set_GPIO_AF(GPIOC,3,5);
    Set_GPIO_AF(GPIOC,2,6);
    Set_GPIO_AF(GPIOC,6,5);

    res=WM8978_Write_Reg(0,0);    //软复位 WM8978
    if(res)return 1;                        //发送指令失败,WM8978 异常

    WM8978_Write_Reg(1,0X1B);     //配置 R1 寄存器
    WM8978_Write_Reg(2,0X1B0);    //配置 R2  寄存器
    WM8978_Write_Reg(3,0X6C);     //配置 R3  寄存器
    WM8978_Write_Reg(6,0);          //配置 R4  寄存器
    WM8978_Write_Reg(43,1<<4);    //配置 R43  寄存器
    WM8978_Write_Reg(47,1<<8);   //配置 R47  寄存器
    WM8978_Write_Reg(48,1<<8);   //配置 R48  寄存器
    WM8978_Write_Reg(49,1<<1);   //配置 R49  寄存器
    WM8978_Write_Reg(10,1<<3);   //配置 R10  寄存器
    WM8978_Write_Reg(14,1<<3);   //配置 R10  寄存器
    return 0;
}
```

b) WM8978 写寄存器功能。

```
/**********************************************************************
*   函数名：    u8 WM8978_Write_Reg(u8 reg,u16 val)
*   功能描述: WM8978 芯片写寄存器的值
*   输入参数: reg:寄存器地址
                        val:要写入寄存器的值
*   输出参数: 无
*   返回值:    0,写成功   1，写放失败
*   其他:
*   作者:
**********************************************************************/
u8 WM8978_Write_Reg(u8 reg,u16 val)
{
    IIC_Start();
    IIC_Send_Byte((WM8978_ADDR<<1)|0);//发送器件地址+写命令
    if(IIC_Wait_Ack())              //等待应答
    {
```

```
        return 1;
    }
    IIC_Send_Byte(((reg<<1)& 0xFE)|((val>>8)&0X01));//写寄存器地址+数据的最高位
    if(IIC_Wait_Ack())              //等待应答
    {
        return 1;
    }
    IIC_Send_Byte(val&0XFF);  //发送数据
    if(IIC_Wait_Ack())
    {
        return 1;                   //等待应答
    }
    IIC_Stop();
    wm8978_RegCash[reg]=val; //保存寄存器值到本地
    return 0;
}
```

c) WM8978 开启 DAC 功能。

```
/*****************************************************************
*   函数名：  void WM8978_DAC_Init(void)
*   功能描述: 开启 WM8978 芯片 DAC 功能
*   输入参数: 无
*   输出参数: 无
*   返回值:    无
*   其他:
*   作者:
*****************************************************************/
void WM8978_DAC_Init(void)
{
    u16 regval;
    regval=WM8978_Read_Reg(3);   //读取 R3  寄存器的值
    regval|=3<<0;                //R3 寄存器最低 2 个位设置为 1,即开启 DACR&DACL
    WM8978_Write_Reg(3,regval);   //配置 R3  寄存器
    regval=WM8978_Read_Reg(2);   //读取 R3  寄存器的值
    WM8978_Write_Reg(50,0x01);   //配置 R50  寄存器
    WM8978_Write_Reg(51,0x01);   //配置 R51  寄存器
}
```

d) WM8978 IIS 模式。

```
/*****************************************************************
*   函数名：  void WM8978_I2S_Mode(u8 fmt,u8 len)
*   功能描述: 设置 WM8978 芯片 IIS 工作模式
*   输入参数: fmt: 0,LSB(右对齐);1,MSB(左对齐);2,飞利浦标准 I2S;3,PCM/DSP;
              len: 0,16 位;1,20 位;2,24 位;3,32 位
*   输出参数: 无
*   返回值:    无
*   其他:
*   作者:
*****************************************************************/
void WM8978_I2S_Mode(u8 fmt,u8 len)
```

```
{
    fmt&=0X03;//限定范围
    len&=0X03;//限定范围
    WM8978_Write_Reg(4,(fmt<<3)|(len<<5)); //R4,WM8978 工作模式设置
}
```

e) WM8978 设置耳机音量功能。

```
/*****************************************************************
*  函数名：  void WM8978_Set_Vol(u8 voll,u8 volr)
*  功能描述: 设置耳机音量
*  输入参数:  voll ，左声道音量
              volr ，右声道音量
*  输出参数: 无
*  返回值:   无
*  其他:
*  作者:
*****************************************************************/
void WM8978_Set_Vol(u8 voll,u8 volr)
{
    voll&=0X3F;
    volr&=0X3F;//限定范围
    WM8978_Write_Reg(52,voll);          //R52,耳机左声道音量设置
    WM8978_Write_Reg(53,volr|(1<<8));  //R53,耳机右声道音量设置,同步更新(HPVU=1)
}
```

2．I2S.c 文件中。

a) IIS 初始化。

```
/*****************************************************************
*  函数名：  void I2S2_Init(u8 std,u8 mode,u8 cpol,u8 datalen)
*  功能描述: IIS 初始化
*  输入参数:  std ，I2S 标准,00,飞利浦标准;01,MSB 对齐标准(右对齐);10,LSB 对齐标准(左对齐);11,PCM 标准
                mode，I2S 工作模式,00,从机发送;01,从机接收;10,主机发送;11,主机接收
                    cpol，0,时钟低电平有效;1,时钟高电平有效
                    datalen，数据长度,0,16 位标准;1,16 位扩展;2,24 位;3,32 位.
*  输出参数: 无
*  返回值:   无
*  其他:
*  作者:
*****************************************************************/
void I2S2_Init(u8 std,u8 mode,u8 cpol,u8 datalen)
{
    RCC->APB1ENR|=1<<14;  //使能 SPI2 时钟
    RCC->APB1RSTR|=1<<14;    //复位 IIS
    RCC->APB1RSTR&=~(1<<14);//结束复位
    SPI2->I2SCFGR=0;              //全部设置为 0
    SPI2->I2SPR=0X02;             //分频寄存器为默认值
    SPI2->I2SCFGR|=1<<11;          //选择:I2S 模式
    SPI2->I2SCFGR|=(u16)mode<<8;//I2S 工作模式设置
    SPI2->I2SCFGR|=std<<4;         //I2S 标准设置
```

```
    SPI2->I2SCFGR|=cpol<<3;          //空闲时钟电平设置
    if(datalen)                      //非标准 16 位长度
    {
        SPI2->I2SCFGR|=1<<0;         //长度为 32 位
        datalen-=1;
    }
    else
    {
        SPI2->I2SCFGR|=0<<0;         //长度为 16 位
    }
    SPI2->I2SCFGR|=datalen<<1;       //I2S 标准设置
    SPI2->CR2|=1<<1;                 //SPI2 TX DMA 请求使能.
    SPI2->I2SCFGR|=1<<10;            //SPI2 I2S EN 使能.
}
```

b) IIS 的 DMA 功能初始化。

```
/**************************************************************************
*   函数名：  void I2S2_TX_DMA_Init(u8* buf0,u8 *buf1,u16 num)
*   功能描述: IIS 的 DMA 功能配置，设置为双缓冲模式,并开启 DMA 传输完成中断
*   输入参数:  buf0:M0AR 地址
                    buf1:M1AR 地址.
                        num:每次传输数据量
*   输出参数: 无
*   返回值:    无
*   其他:
*   作者:
**************************************************************************/
void I2S2_TX_DMA_Init(u8* buf0,u8 *buf1,u16 num)
{
    RCC->AHB1ENR|=1<<21;             //DMA1 时钟使能
    while(DMA1_Stream4->CR&0X01);    //等待 DMA1_Stream4 可配置
    DMA1->HIFCR|=0X3D<<6*0;          //清空通道 4 上所有中断标志
    DMA1_Stream4->FCR=0X0000021;     //设置为默认值

    DMA1_Stream4->PAR=(u32)&SPI2->DR;  //外设地址为:SPI2->DR IIS 和 SPI 共用一个数据寄存器
    DMA1_Stream4->M0AR=(u32)buf0;    //内存 1 地址
    DMA1_Stream4->M1AR=(u32)buf1;    //内存 2 地址
    DMA1_Stream4->NDTR=num;          //设置长度
    DMA1_Stream4->CR=0;              //先全部复位 CR 寄存器值
    DMA1_Stream4->CR|=1<<6;          //存储器到外设模式
    DMA1_Stream4->CR|=1<<8;          //循环模式
    DMA1_Stream4->CR|=0<<9;          //外设非增量模式
    DMA1_Stream4->CR|=1<<10;         //存储器增量模式
    DMA1_Stream4->CR|=1<<11;         //外设数据长度:16 位
    DMA1_Stream4->CR|=1<<13;         //存储器数据长度:16 位
    DMA1_Stream4->CR|=2<<16;         //高优先级
    DMA1_Stream4->CR|=1<<18;         //双缓冲模式
    DMA1_Stream4->CR|=0<<21;         //外设突发单次传输
    DMA1_Stream4->CR|=0<<23;        //存储器突发单次传输
```

```
    DMA1_Stream4->CR|=0<<25;          //选择通道 0 SPI2_TX 通道

    DMA1_Stream4->FCR&=~(1<<2);     //不使用 FIFO 模式
    DMA1_Stream4->FCR&=~(3<<0);     //无 FIFO  设置

    DMA1_Stream4->CR|=1<<4;           //开启传输完成中断

    NVIC_SetPriorityGrouping(7-2); //设置为分组 2
    NVIC_SetPriority(DMA1_Stream4_IRQn,NVIC_EncodePriority(7-2,1,0));     //配置抢占优先级和响应优先级
    NVIC_EnableIRQ(DMA1_Stream4_IRQn);//中断使能

}
```

3．音乐播放函数。

```
/**********************************************************************
*   函数名：  void WAV_Play(u8* fname)
*   功能描述: 播放音乐
*   输入参数:   fname：带路径文件名
*   输出参数: 无
*   返回值:    无
*   其他:
*   作者:
**********************************************************************/
void WAV_Play(u8* fname)
{
    u8 res;
    u32 fillnum;                //填充数目
    wavbuf.file=(FIL*)MemIn_Malloc(sizeof(FIL));
    wavbuf.dmabuf0=MemIn_Malloc(DMA_Tx_Size);
    wavbuf.dmabuf1=MemIn_Malloc(DMA_Tx_Size);
    wavbuf.buff         =MemIn_Malloc(DMA_Tx_Size);
    res=WAV_Decode(fname,&wavinfo);         //wav 解码得到文件的信息
    if(res==0)//解析文件成功
    {
        if(wavinfo.BitCount==16)
        {
            WM8978_I2S_Mode(2,0);    //飞利浦标准,16 位数据长度
            I2S2_Init(0,2,0,1);              //飞利浦标准,主机发送,时钟低电平有效,16 位扩展帧长度
        }else if(wavinfo.BitCount==24)
        {
            WM8978_I2S_Mode(2,2);    //飞利浦标准,24 位数据长度
            I2S2_Init(0,2,0,2);              //飞利浦标准,主机发送,时钟低电平有效,24 位扩展帧长度
        }
        else if(wavinfo.BitCount==32)
        {
            res=0xff;
            goto end_play_song;
        }
        I2S2_SampleRate_Set(wavinfo.SampleRate);//设置采样率
```

```
            I2S2_TX_DMA_Init(wavbuf.dmabuf0,wavbuf.dmabuf1,DMA_Tx_Size/2); //配置 TX DMA
            WAV_Stop();
        res=f_open(wavbuf.file,(TCHAR*)fname,FA_READ);      //打开文件
        if(res==0)
        {
            f_lseek(wavbuf.file, wavinfo.OffBytes);             //跳过文件头
            fillnum=WAV_Fill_Buf(wavbuf.dmabuf0,DMA_Tx_Size,wavinfo.BitCount);//填充 wavbuf.dmabuf0
            fillnum=WAV_Fill_Buf(wavbuf.dmabuf1,DMA_Tx_Size,wavinfo.BitCount);//填充 wavbuf.dmabuf1
            WAV_Start();
            while(res==0)
            {
                while(wav_transfer_flag==0);//等待 wav 传输完成;
                wav_transfer_flag=0;
                if(fillnum!=DMA_Tx_Size)//播放结束
                {
                    res=3;
                    break;
                }
                if(wav_dma_buf_flag)
                {
                    fillnum=WAV_Fill_Buf(wavbuf.dmabuf1,DMA_Tx_Size,wavinfo.BitCount);//填充 buf2
                }
                else
                {
                    fillnum=WAV_Fill_Buf(wavbuf.dmabuf0,DMA_Tx_Size,wavinfo.BitCount);//填充 buf1
                }
            }
            WAV_Stop();
        }
        f_close(wavbuf.file);
    }
    end_play_song:
    MemIn_Free(wavbuf.buff);   //释放内存
    MemIn_Free(wavbuf.dmabuf0);//释放内存
    MemIn_Free(wavbuf.dmabuf1);//释放内存
    MemIn_Free(wavbuf.file);    //释放内存
}
```

1.1.7 音乐播放程序仿真，下载，测试

把示例代码下载到开发板中，并且 SD 卡文件中 music 文件夹下有 wav 格式的音频文件，然后插上耳机就可以听到音乐了。