7．6 WAV 声音文件的播放

7．6．1 WAV 文件及应用

1．WAV 文件概述及音频文件的获取

WAV（也称 WAVE）是 Windows 的声音文件，后缀为 wav(即*.wav)，有时也直接称它为 PCM(脉冲编码调制)声音文件，是没有经过任何压缩的声波数据文件，就是说文件中的音频数据直接对应着模拟声音被（A/D 或 D/A）量化的数字量。在 windows 操作系统下，在开机或关机，以及打开文件出错等等都会听到悦耳、熟悉的声音，这些声音就是 wav 格式的音频文件，可以在 C:\WINDOWS\Media 目录下找到，如图 7.10 所示。

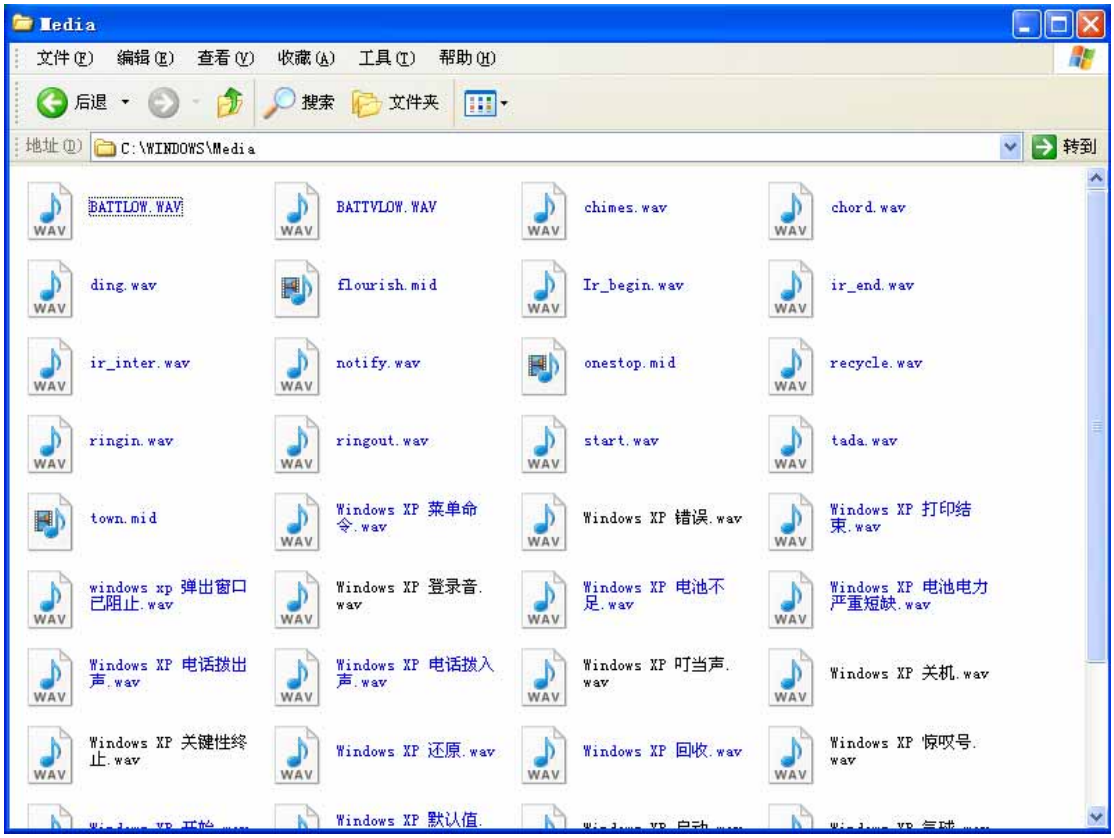

图 7。10　windows 下的 wav 文件

在实践的嵌入式产品中，我们可能希望产品在开机、关机时也能播放出像 Windows XP 那样悦耳、熟悉的一段声音；也可能希望点击触摸屏上的某个按钮时发出清澈的响声；在出现紧急情况时发出刺耳的报警声等等。这时我们都可以在图 7.10 所示的音频文件中选取，或上网查找更好的 wav 声音文件，以及亲自录制等。通常可以选择 windows 自带的录音机(当然，其他的播放器也可以播放 wav 格式文件)来播放测试和了解音频文件的属性及格式转化。录音机可以选择“开始菜单->程序->附件->娱乐->录音机”打开，如图 7.11 所示的录音机界面。在录音机下可以选择“文件->打开”菜单项中打开需要测试的音频文件，如图所示还在播放、前进、后退等按钮，控制音频文件的播放。

图 7。11 windows 自带的录音机

我们也可在录音机中选择‘ 文件->属于 ’菜单项打开正在播放的音频文件的属性对话框，如图 7.12 所示。

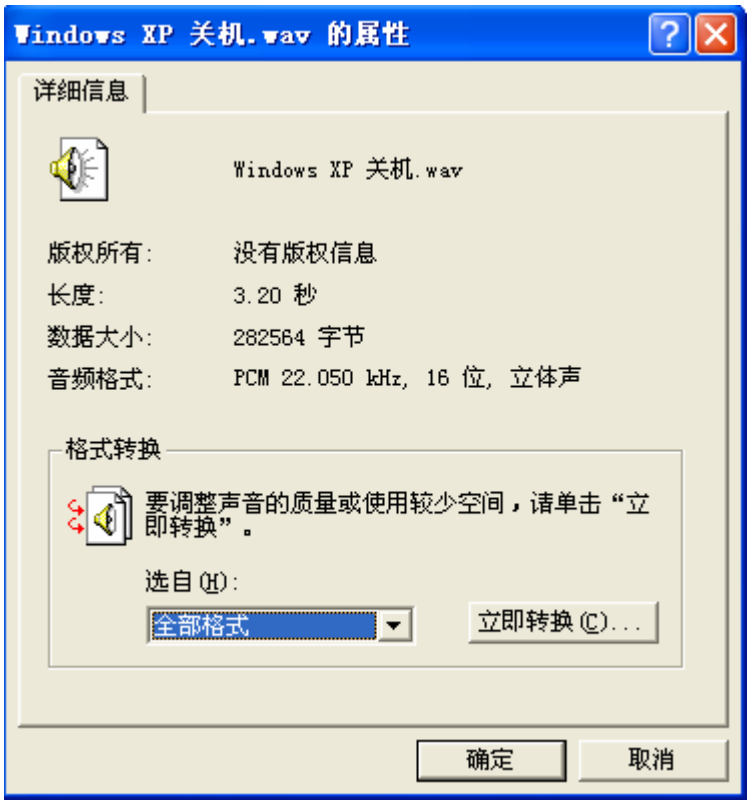

图 7。12 wav 文件属性对话框

图中的声音格式：频率为 22.050kHz，16 位及立体声。有些 wav 文件的音频格式会不一样，如频率为 44.10kHz，或为 8 位，或为单声道等，甚至有些文件数据格式也不一样。由于嵌入式系统资源的有限，如做到像 PC 机的 windows 那样支持各种模式会比较困难（如频率，因为处理器时钟进行分频设置后，不能同时支持 44.1kHz 及 8kHz 或其他等，当 44.1kH 比较准确时，计算得到的 8kHz 存在的误差可能就比较大，播放的声音就会失真），通常我们将各种格式的文件通过录音机将他们转换成统一格式的音频文件。点击图 7.12 对话框的立即转换按钮，或录音机“ 文件->另存为 ”菜单项另外为找开的文件，如图 7.13 所示格式转换对话框。我们可以根据应用程序中的音频初始化及 WAV 文件分析部分程序在属性项中选择合适的音频文件频率、位数、声道等。

图 7。13 wav 文件格式转换

2．WAV 文件格式分析

当我们有了统一音频格式 wav 文件后，我们还需要在应用程序中分析 wav 的文件格式，去掉文件的头信息，提取音频数据通过 IIS 输出给音频解码器播放。WAV 文件格式相对 MP3 或视频文件等简单的太多，网上也都可以找到它们的格式信息及分析之类的文章，其中这个网站“ http://www.moon-soft.com/program/FORMAT/ ”下载的文件格式还是比较详细的，读者可以自行去下载参考分析。通常我们可以使用 UltraEdit 等常用的编辑器打开音频文件，再对着 wav 文件格式去分析它，如图 7.14 所示为 wav 音频文件的数据。

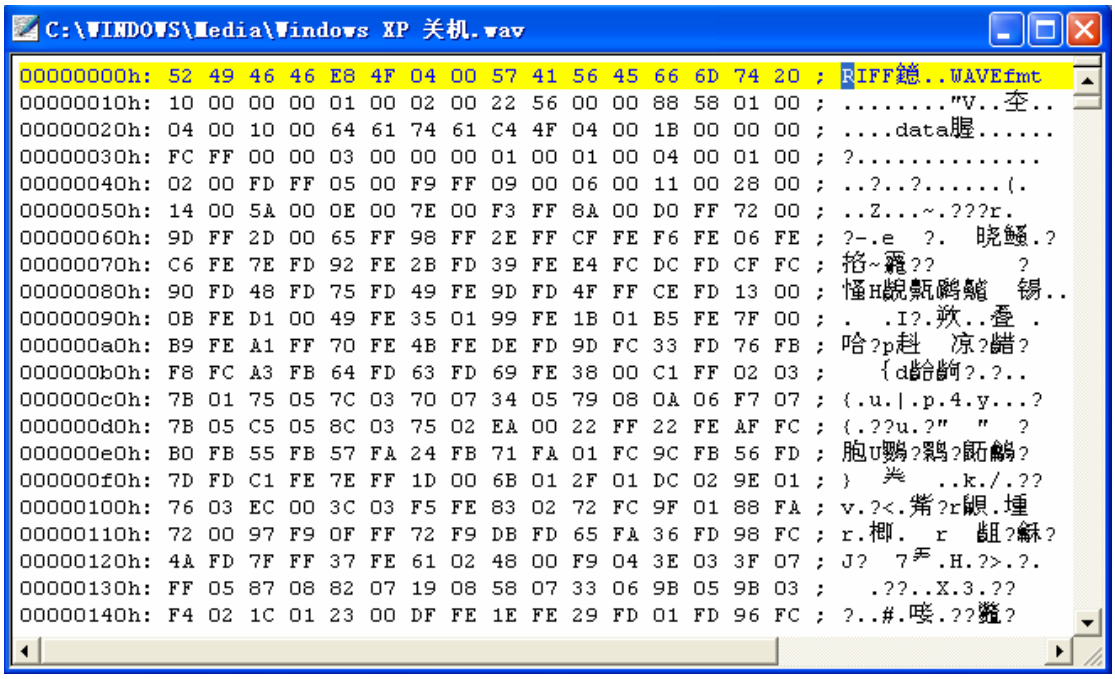

图 7。14 wav 音频文件数据

整个文件的格式如图 7.15 所示，有 RIFF header、RIFF type、fmt chunk 和音频 data chunk 四个部分，图中括号内为它们占用的字节数，下面分别介绍该四个部分内容。

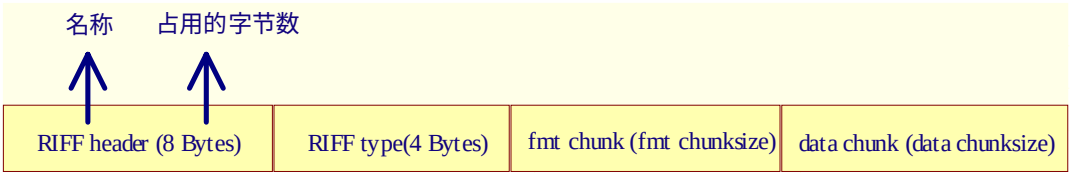

图 7。15 wav 文件格式

RIFF head

RIFF head 共 8 字节，前面 4 字节内容永远都为“RIFF”,后面 4 字节为 RIFF 文件的总字节数，不包括 RIFF header。如图 7.14 所示的“Windows XP 关机.wav”的数据，其中 4 字节的“RIFF”后为 0x00044FE8(小端模式)，读者可以查看文件的最后，会发现 0x00044FE8 正为除 RIFF header 之后的数据字节总数。

RIFF type

wav 文件的该 4 字节内容永远为“WAVE”。

fmt chunk

fmt chunk 通常为 0x18 字节，如表 7.1 所示，其中地址项的地址代表在整个 wav 文件中的地址。

表 7.1 fmt chunk 格式说明

地址	字节数	名称	说明
0x0c	4	chunkid	永远是”fmt”。
0x10	4	chunksize	指 fmt chunk 的字节数,不包括 chunkid 和 chunksize,本结构中为 0x10。
0x14	2	wformattag	格式标志，当为 01 时表示音频数据没有被压缩。
0x16	2	wchannels	声道数，1 为单声道；2 为立体声；4 为 4 声道。
0x18	4	dwsamplespersec	采样频率，如图 7。14 所示为 0x5622 即为 22050Hz。另还有 11025Hz 和 44100Hz 等。
0x1c	4	dwavgbytespersec	指每秒播放的字节数，等于 dwsamplespersec* wblockalign
0x20	2	wblockalign	表示采集一个点时的总字节数，等于(音频采样大小/8)*声道号，例如：16 位单声道为 2；16 位双声道为 4；8 位单声音为 1 等。
0x22	2	wbitspersample	指音频采样大小（即分辨率），指一个采集点所表示的位数，如 16-bit 波形则等于 16。

data chunk

data chunk 为整个音频文件的主体，由 3 部分组成，如表 7.2 所示。

表 7.2 data chunk 格式说明

地址	字节数	名称	说明
0x24	4	chunkid	永远是”data”。
0x28	4	chunksize	指音频数据的大小，即为后续剩余的所有字节。
0x2c	chunksize	waveformdata[]	音频数据。

data organization（数据格式）

wav 音频的所有数据（即上述所讨论的所有数据）都为小端的内存格式，如图 7.16 所示，它与 ARM 的小端格式是一样。

```
        7  6  5  4  3  2  1  0
       +-----------------------+
 char: | lsb                msb |
       +-----------------------+

        7  6  5  4  3  2  1  0 15 14 13 12 11 10  9  8
       +-----------------------+-----------------------+
short: | lsb      byte 0       |       byte 1      msb |
       +-----------------------+-----------------------+

        7  6  5  4  3  2  1  0 15 14 13 12 11 10  9  8 23 22 21 20 19 18 17 16 31 30 29 28 27 26 25 24
       +-----------------------+-----------------------+-----------------------+-----------------------+
 long: | lsb      byte 0       |       byte 1          |         byte 2        |       byte 3      msb |
       +-----------------------+-----------------------+-----------------------+-----------------------+
```

图 7。16 data organization

注：上述格式，读者都可以对照图 7.14 所示或自行打开音频文件进行分析。

3．WAV 文件在应用程序中的保存

如果音频文件较多，或需要后续更换等，往往需要在系统的 Flash 中创建文件系统，以文件的形式进行管理。但大部分的前后台系统不需要如此复杂，也不需要太多的音频文件，所以也往往直接将 WAV 音频文件以数组的变量形式存在，与代码一起被编译保存。需要播放时再使用指针的方式指向某个具体音频文件的数组以获得数据。将 wav 的音频数据转换成 C 语言格式，可以使用 WinHex 软件方便的实现。启动 WinHex，打开 wav 音频文件，然后选择“编辑->全部复制->C 源”菜单项（如图 7.17 所示）复制整个音频数据。接着，在 C 文件中粘帖即可产生如图 7.18 所示的数组，我们可以将 data 修改成合适的数组名。

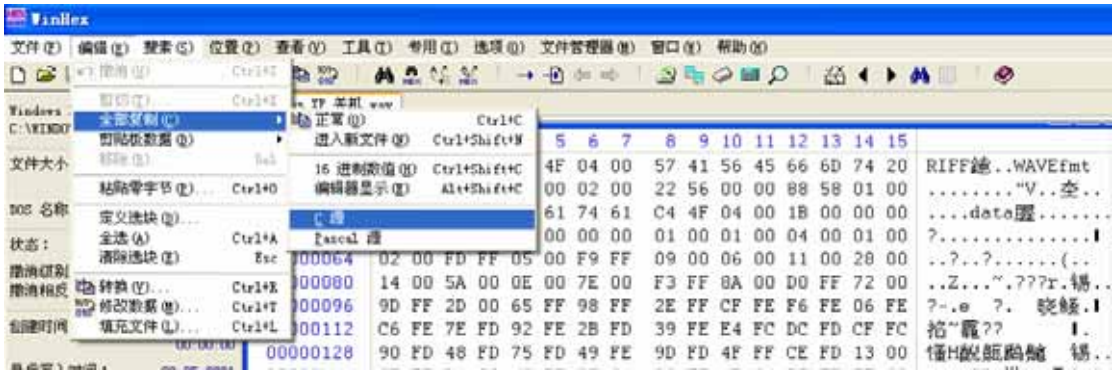

图 7。17 WinHex 复制

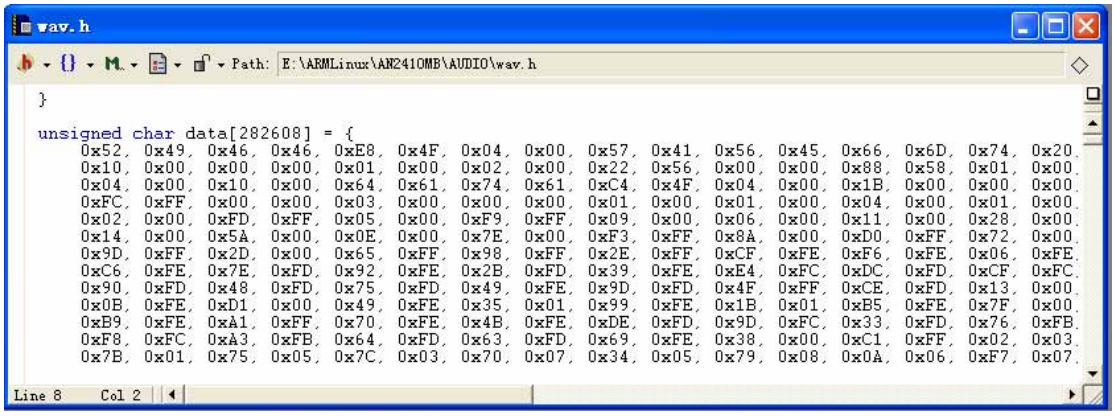

图 7。18 wav 音频文件数组

7．6．2 S3C2410A 的数字音频接口 IIS 设置

1．IIS 接口的 I/O 口端口初始化

```
//----------------------------------------------------------
```

```
//PORT E GROUP
//Ports  :  GPE4    GPE3   GPE2  GPE1    GPE0
//Signal : I2SSDO  I2SSDI CDCLK I2SSCLK I2SLRCK
//Binary :   10   ,   10     10 ,  10      10
//---------------------------------------------------------
rGPEUP  = rGPEUP  & ~(0x1f)  | 0x1f; //The pull up function is disabled GPE[4:0] 1 1111
rGPECON = rGPECON & ~(0x3ff) | 0x2aa;//GPE[4:0]=I2SSDO:I2SSDI:CDCLK:I2SSCLK:I2SLRCK
```

2．时钟设置

S3C2410A 处理器需要给 UDA1341 提供系统时钟 CODECLK，由外围时钟 PCLK 经 IISPSR 寄存器的预分频值分频等到。为得到相对准确的 CODEC 系统时钟和 fs，避免声音失真，往往需要修改原来的 FCLK、PCLK。

当 fs = 44100Hz 时，CODECLK = 256fs = 11.2896MHz

此时需要通过下述几个步骤的设置，使 CODECLK 约等于 11.2896MHz。

首先，设置锁相环控制寄存器 MPLLCON，确定 CPU 使用的时钟 FCLK。

```
FCLK=[(M_MDIV+8)*Fin]/[(M_MPIV+2)*2M_SDIV]
```

当 ChangeMPllValue(0x96,0x5,0x1);时，则：

```
FCLK = [(0x96+8)*12MHz]/[(5+2)*21] = 135.428571MHz
```

其次，设置时钟分频控制寄存器 CLKDIV 中的 HDIVN、PDIVN，确定 PCLK。

当 ChangeClockDivider(1,1);时，FCLK:PCLK = 1:4，则：

```
PCLK = FCLK/4 = 33.85714MHz
```

最后，设置 IIS 的预分频寄存器 IISPSR，确定 CODECLK。

如 rIISPSR = (2<<5) + 2;时，Prescaler control A,B = 2+1 = 3，则：

CODECLK = PCLK/3 = 11.28571MHz 约等于 11.2896MHz

3．设置 IIS 控制寄存器 IISCON

通过 IISCON 可以使能 IIS 的 DMA 传输，IIS 预分频器的使能等，如：

```
rIISCON = (1<<5) + (1<<2) + (1<<1);
//Tx DMA enable[5], Rx idle[2], Prescaler enable[1]
```

另外，IIS 接口的启动也由该寄存器控制，如下：

```
rIISCON |= 0x1;             //IIS Interface start
```

4．设置 IIS 模式寄存器 IISMOD

```
rIISMOD = (0<<8) + (1<<6) + (0<<5) + (0<<4) + (1<<3) + (0<<2) + (1<<0);
```

上述设置：主模式；传输（发送）模式；左通道低，右通道高；IIS 兼容格式；每通道串行数据位为 16 位；主系统时钟频率（CODECLK）为 256fs；串行位时钟频率为 32fs。

5．设置 FIFO 控制寄存器 IISFCON

```
rIISFCON = (1<<15) + (1<<13);       //Tx DMA,Tx FIFO --> start piling....
```

上述设置 Transmit FIFO 访问为 DMA 模式，且 Transmit FIFO 使能。

6．DMA 初始化及中断

由于音频播放很占用处理器，所以一般都使用 DMA 的方式自动从缓冲区中读取音频数据。在使用 DMA 之前，先要指定 DMA 的操作地址，就是把音频数据缓冲区的起始地址指定给它，以及其他的 DMA 相关的设置等。

```
rDISRC2 = (int)(Buf + 0x2c);//音频数据缓冲区的起始地址，去掉 0x2c 前的 wav 文件数据头
rDISRCC2 = (0<<1) + (0<<0); //设置 DMA2 的源在 AHB 总线，且地址自动加 1
rDIDST2  = ((U32)IISFIFO);  //将 IIS 的 FIFO 寄存器地址作为 DMA2 的目的地址
rDIDSTC2 = (1<<1) + (1<<0); //设置目的在 APB 总线上，且地址固定
```

```
rDCON2=(1<<31)+(0<<30)+(1<<29)+(0<<28)+(0<<27)+(0<<24)+(1<<23)+(1<<22)+(1<<20)+(size/2);
//Handshake, sync PCLK, TC int, single tx, single service, I2SSDO, I2S request,
// DMA is turned off when a current value of transfer count becomes 0, half-word, size/2
rDMASKTRIG2 = (0<<2) + (1<<1) + (0<<0);//No-stop[2], DMA2 channel On[1], No-sw trigger[0]
```

另外，还需要初始化 DMA 通道的中断等

```
pISR_DMA2  = (unsigned)DMA2_Done;
unsigned char PlayWaveEndFlag1 = 0;
void __irq DMA2_Done(void){//DMA 中断处理程序
    rSRCPND = BIT_DMA2;           //Clear pending bit
    rINTPND = BIT_DMA2;
    rIISCON = 0x0;                //IIS Interface stop
    rDMASKTRIG2 = (1<<2);         //DMA2 stop
    rIISFCON = 0x0;               //For FIFO flush
    rINTMSK = rINTMSK | (BIT_DMA2);
    PlayWaveEndFlag1 = 0;
}
```

7．6．3 UDA1341TS（L3-interface）初始化及控制

如果读者选择如 HT82V731 等普通的音频 D/A，在播放 wav 音频时只需通过 IIS 发送音频数据即可，但是 UDA1341TS 还需要通过 L3-interface 对其进行初始化及设置。初始化时需复位 UDA1341；系统时钟设置成 256fs；传输格式设置成 IIS；关闭录音功能（ADC0FF），打开放音功能（DACON）等。

```
void Init1341(char mode){//当 mode 为 1 时将 UDA1341 初始化为录音功能，0 则为播放功能
    //L3 端口初始化
    rGPECON = rGPECON & ~(3 << 22) | (1 << 22);//L3MODE(OUTPUT):GPE11
    rGPEUP  = rGPEUP  & ~(1 << 11)  | (1 << 11);//The pull up function is disabled GPE11
    rGPGCON = rGPGCON & ~(0xf << 10) | (0x5 << 10);//L3DATA、L3CLOCK (OUTPUT):GPG[6:5]
    rGPGUP  = rGPGUP  & ~(3 << 5)  | (3 << 5);//The pull up function is disabled GPG[6:5]
    //L3 Interface
    _WrL3Addr(0x14 + 2);  //STATUS (000101xx+10)
    _WrL3Data(0x60,0);    //0,1,10,000,0 : Reset,256fs,no DCfilter,iis
    _WrL3Addr(0x14 + 2);  //STATUS (000101xx+10)
    _WrL3Data(0x20,0);    //0,0,10,000,0 : No reset,256fs,no DCfilter,iis
    _WrL3Addr(0x14 + 2);  //STATUS (000101xx+10)
    _WrL3Data(0x81,0);    //1,0,0,0,0,0,01 : OGS=0,IGS=0,ADC_NI,DAC_NI,sngl speed,AoffDon
    if(mode){             //record
        _WrL3Addr(0x14 + 2);//STATUS (000101xx+10)
        _WrL3Data(0xa2,0); //1,0,1,0,0,0,10 : OGS=0,IGS=1,ADC_NI,DAC_NI,sngl speed,AonDoff
        _WrL3Addr(0x14 + 0);//DATA0 (000101xx+00)
        _WrL3Data(0xc2,0);  //11000,010  : DATA0, Extended addr(010)
        _WrL3Data(0x4d,0);  //010,011,01 : DATA0, MS=9dB, Ch1=on Ch2=off,
    }
}
```

注：_WrL3Addr()、_WrL3Data()为写 L3 Interface 的地址、数据函数，读者可以参考 2410test 内的程序。

7．6．4 功放电路的音量调节

在 IIS 音频电路原理图中所示，我们增加了一个声道的功率放大电路驱动一个小喇叭，喇叭的音量由 SPI 接口的数字电位器调节，静音由 S3C2410A 的普通 GPIO 口控制。

1．SPI 接口 I/O 口端口初始化

这里将 SPI 接口时序使用普通的 GPIO 模拟产生。

```
//SPIMOSI0 SPICLK nSS_SPI0
//GPGE12   GPGE13 GPG2
#define DP_CS   (1 << 2)//GPG2
#define DP_CLK  (1 << 13)//GPE13
#define DP_SI   (1 << 12)//GPE12
#define SP_MUTE (1 << 6)//GPB6,低电平静音
void Port_DP_Init(void){
    rGPECON = rGPECON & ~(0xf << 24) | (0x5 << 22);//SPIMOSI0、SPICLK(OUTPUT):GPE[12:13]
    rGPEUP  = rGPEUP  & ~(3 << 12)  | (0 << 11);//The pull up function is enabled GPE[11:13]
    rGPGCON = rGPECON & ~(0x3 << 4) | (0x1 << 4);//nSS_SPI0(OUTPUT):GPE[12:13]
    rGPGUP  = rGPEUP  & ~(1 << 2)  | (0 << 2);   //The pull up function is enabled GPG2
    rGPGDAT = rGPGDAT | DP_CS;                   //初始 CS = 1
    rGPEDAT = rGPEDAT & ~(3 << 12) | (1 << 12);  //CLK = 0,SI = 1;
    rGPBCON = rGPBCON & ~(3 << 12) | (1 << 12);  //SP_MUTE(OUTPUT):GPB6
    rGPBUP  = rGPBUP  & ~(1 << 6)  | (0 << 6);   //The pull up function is enabled GPB6
    rGPBDAT = rGPBDAT | SP_MUTE;                 //SP_MUTE = 1，正常操作
}
/*********************************************************************************************
函数原形：void Volume_Adj(unsigned char taps)
功    能：音量调节，即发数据调节数字电位器值
参    数：taps -- 数值，范围为 0 ~ 255,对应阻值
*********************************************************************************************/
#define COMMAND_W  0x11   //C1 C0 = 01(写数据)，P1 P0 = 01(执行电阻 0)
void Volume_Adj(unsigned char taps){
    U16 SI_DATA;
    U8 Delay_time,i;
    taps = (taps / 28) * 19;
    SI_DATA = (U16)(COMMAND_W << 8) + (0xff - taps);
    //SI_DATA = 0x5555;
    rGPGDAT = rGPGDAT &(~DP_CS);          //CS = 0;
    for (i = 0; i < 16; i++){
        if (SI_DATA & 0x8000){
            rGPGDAT |=  DP_SI;            //SI = 1;
        }
        else{
```

```
            rGPGDAT &= (~DP_SI);          //SI = 0;
        }
        SI_DATA = SI_DATA << 1;
        rGPGDAT |= DP_CLK;                //CLK = 1;
        for (Delay_time = 0; Delay_time < 0x20; Delay_time++);
        rGPGDAT &= (~DP_CLK);             //CLK = 0;
        for (Delay_time = 0; Delay_time < 0x20; Delay_time++);
    }
    rGPGDAT |= DP_CS;                     //CS = 1;
}
```

7．6．5 WAV 文件播放

上几节把 wav 文件播放相关的各部分内容进行了叙述，这里列出 wav 文件播放的接口函数，即可以直接被其他应用程序调用来播放 wav 音频文件。

```
unsigned char *Buf,*_temp;
/*****************************************************************************
函数原形：void PlayWave(unsigned char *wavebuffer)
功    能：播放 wav 文件
参    数：wavebuffer —— 指向需播放的音频文件的起始地址
*****************************************************************************/
void PlayWave(unsigned char *wavebuffer){
    rINTMSK  = rINTMSK & ~(BIT_DMA2);//开 DMA2 中断
    Buf   = wavebuffer;
    _temp = Buf;
    fs   = *(Buf + 0x18) | *(Buf + 0x19)<<8 | *(Buf + 0x1a)<<16 | *(Buf + 0x1b)<<24;
    //读取 wav 文件的音频采集频率
    if (fs==44100){               //11.2896MHz(256fs)
        rIISPSR = (4<<5) + 4;     //Prescaler A,B=2 <- FCLK 135.4752MHz(1:2:4)
        size = *(Buf + 0x36) | *(Buf + 0x37)<<8 | *(Buf + 0x38)<<16 | *(Buf + 0x39)<<24;
        rDISRC2 = (int)(Buf + 0x3a);//音频数据缓冲区的起始地址
    //这里的 0x3a 需要查看具体音频文件进行确认，因为有些可能是 0x2c，在上一节的 wav 文件格式中也是介绍为 0x2c，所以如果读者的音频文件也为 0x2c，则此处该修改为 0x2c
    }
    else if (fs == 22050){        //fs=22050, 5.6448MHz(256fs)
        rIISPSR = (9<<5) + 9;     //Prescaler A,B=2 <- FCLK 202.8MHz(1:2:4)
        size = *(Buf + 0x28) | *(Buf + 0x29)<<8 | *(Buf + 0x2a)<<16 | *(Buf + 0x2b)<<24;
        rDISRC2 = (int)(Buf + 0x2c);
    //音频数据缓冲区的起始地址，这里同样需要确认
    }
    rIISCON = (1<<5) + (1<<2) + (1<<1);//Tx DMA enable[5], Rx idle[2], Prescaler enable[1]
    rIISMOD = (0<<8) + (2<<6) + (0<<5) + (1<<4) + (1<<3) + (0<<2) + (1<<0);
    rIISFCON = (1<<15) + (1<<13);     //Tx DMA,Tx FIFO --> start piling....
    rDISRCC2 = (0<<1) + (0<<0);       //The source is in the system bus(AHB), Increment
```

```
    rDIDST2   = ((U32)IISFIFO);        //IISFIFO
    rDIDSTC2 = (1<<1) + (1<<0);        //The destination is in the peripheral bus(APB), Fixed
    rDCON2=(1<<31)+(0<<30)+(1<<29)+(0<<28)+(0<<27)+(0<<24)+(1<<23)+(1<<22)+(1<<20)+(siz
e/2);//使能发送完成中断且关闭 DMA 通道,设置 DMA 每次操作的数据单位(Half Word)和总操作数(size/2)
    rDMASKTRIG2 = (0<<2) + (1<<1) + (0<<0);//No-stop[2], DMA2 channel On[1], No-sw trigger[0]
    rIISCON |= 0x1;                    //IIS Interface start
}
```

PlayWave()函数的开始打开了 DMA2 的中断；然后读取 wav 文件的头信息，有些 wav 文件格式可能不一样，所以需要事先确认，否则将会解析出错。由于实践应用中往往不希望处理器的内核时钟 FCLK，外围时钟 PCLK 变化(因为它们的变化会给串口的波特率、定时器等带来影响)，当同时支持采样频率为 44.1kHz 和 22.050kHz 或更多时，时钟应分频后误差会较大，这里的 44.1kH 的误差就是如此，所以在实践的应用中，尽量将音频文件通过 windows 自带的录音机转换成同一采样频率 fs 的文件；再接下来是初始化 FIFO 和 DMA；最后是启动 IIS 开始往 CODEC 发送音频数据。当发送完毕后会产生停止 DMA2 通道且产生中断。

我们可以在应用程序中的任意处调用上述的函数，如开机时播放的声音：

```
void Main(void){
    ......
    PlayWave1(Mozart);
    ......
}
```

也可以提供下述函数，供应用程序调用，如下：

```
/*********************************************************************************
函数原形：unsigned char PlayBeep(unsigned char soundtype)
功    能：播放语音信息
参    数：soundtype —— 声音类型，提供序号给其他应用程序调用
*********************************************************************************/
unsigned char PlayBeep(unsigned char soundtype){
    if (PlayWaveEndFlag1){
        return NULL;
     }//防止连续打开
     PlayWaveEndFlag1 = 1;
    switch (soundtype){
        case 0:
                PlayWave1(Drip3);
                break;
        case 1:
                PlayWave1(annu4_22);
                break;
        ......
        default:
                PlayWave1(Drip3);
                break;
    }
    return TRUE;
```

```
}
```