AN1402 ATK-VS1053 MP3 模块录音功能

使用说明

本应用文档（AN1402，对应 **ALIENTEK MiniSTM32 开发板（V3.0）扩展实验 5**）将教大家如何通过 ALIENTEK STM32 开发板和 ATK-VS1053 MP3 模块实现一个录音机的功能。本文档我们将使用 ATK-VS1053 MP3 模块实现 WAV 录音，设计一个简单的录音机！

本文档分为如下几部分：

1， WAV 简介
2， 硬件设计
3， 软件实现
4， 验证

1、WAV 简介

WAV 即 WAVE 文件，WAV 是计算机领域最常用的数字化声音文件格式之一，它是微软专门为 Windows 系统定义的波形文件格式（Waveform Audio），由于其扩展名为"*.wav"。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！

ATK-VS1053 MP3 模块支持 2 种格式的 WAV 录音：PCM 格式或者 IMA ADPCM 格式，其中 PCM（脉冲编码调制）是最基本的 WAVE 文件格式，这种文件直接存储采样的声音数据没有经过任何的压缩。而 IAM ADPCM 则是使用了压缩算法，压缩比率为 4:1。

这里，我们主要讨论 PCM，因为这个最简单。我们将利用 VS1053 实现 16 位，8Khz 采样率的单声道 WAV 录音(PCM 格式)。要想实现 WAV 录音得先了解一下 WAV 文件的格式，WAVE 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)和 Data Chunk。每个 Chunk 由块标识符、数据大小和数据三部分组成，如图 1.1 所示：

图 1.1 Chunk 结构示意图

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。

首先，我们来看看 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-8），然后数据段为“WAVE”，表示是 wav

文件。RIFF 块的 Chunk 结构如下：

```
//RIFF 块
typedef __packed struct
{
    u32 ChunkID;        //chunk id;这里固定为"RIFF",即 0X46464952
    u32 ChunkSize ;     //集合大小;文件总大小-8
    u32 Format;         //格式;WAVE,即 0X45564157
}ChunkRIFF ;
```

接着，我们看看 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，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 块的 Chunk 结构如下：

```
//fmt 块
typedef __packed struct
{
    u32 ChunkID;        //chunk id;这里固定为"fmt ",即 0X20746D66
    u32 ChunkSize ;     //子集合大小(不包括 ID 和 Size);这里为:20.
    u16 AudioFormat;    //音频格式;0X10,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;       //字节速率;
    u16 BlockAlign;     //块对齐(字节);
    u16 BitsPerSample;  //单个采样数据大小;4 位 ADPCM,设置为 4
}ChunkFMT;
```

接下来，我们再看看 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，该块 Chunk 结构如下：

```
//fact 块
typedef __packed struct
{
    u32 ChunkID;        //chunk id;这里固定为"fact",即 0X74636166;
    u32 ChunkSize ;     //子集合大小(不包括 ID 和 Size);这里为:4.
    u32 DataFactSize;   //数据转换为 PCM 格式后的大小
}ChunkFACT;
```

DataFactSize 是这个 Chunk 中最重要的数据，如果这是某种压缩格式的声音文件，那么从这里就可以知道他解压缩后的大小。对于解压时的计算会有很大的好处！不过本文档我们使用的是 PCM 格式，所以不存在这个块。

最后，我们来看看数据块（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。紧接着就是 wav 数据。根据 Format Chunk 中的声道数以及采样 bit 数，wav 数据的 bit 位置可以分成如表 1.1 所示的几种形式：

单声道	取样 1		取样 2	
8 位量化	声道 0		声道 0	
双声道	取样 1		取样 2	
8 位量化	声道 0(左)	声道 1(右)	声道 0(左)	声道 1(右)
单声道	取样 1		取样 2	

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)
双声道	取样 1			
16 位量化	声道 0 (左, 低字节)	声道 0 (左, 高字节)	声道 1 (右, 低字节)	声道 1 (右, 高字节)

表 1.1 WAVE 文件数据采样格式

本文档，我们采用的是 16 位，单声道，所以每个取样为 2 个字节，低字节在前，高字节在后。数据块的 Chunk 结构如下：

```
//data 块
typedef __packed struct
{
    u32 ChunkID;            //chunk id;这里固定为"data",即 0X61746164
    u32 ChunkSize ;         //子集合大小(不包括 ID 和 Size);文件大小-60.
}ChunkDATA;
```

通过以上学习，我们对 WAVE 文件有了个大概了解。接下来，我们看看如何使用 VS1053 实现 WAV（PCM 格式）录音。

激活 PCM 录音

VS1053 激活 PCM 录音需要设置的寄存器和相关位如表 1.2 所示：

寄存器	位域	说明
SCI_MODE	2, 12, 14	开始 ADPCM 模式，选择：咪/线路1
SCI_AICTRL0	15..0	采样率 8000..48000 Hz （在录音启动时读取的）
SCI_AICTRL1	15..0	录音增益（1024 = 1×）或 0 是自动增益控制（AGC）
SCI_AICTRL2	15..0	自动增益放大器的最大值 （1024 = 1×, 65535 = 64×）
SCI_AICTRL3	1..0 2 15..3	0=联合立体声(共用 AGC)，1=双声道(各自的 AGC)，2=左通道，3=右通道 0=IMA ADPCM 模式，1=线性 PCM 模式 保留，设置为 0

表 1.2 VS1053 激活 PCM 录音相关寄存器

通过设置 SCI_MODE 寄存器的 2、12、14 位，来激活 PCM 录音，SCI_MODE 的各位描述见 VS1053 的数据手册。SCI_AICTRL0 寄存器用于设置采样率，我们本文档用的是 8K 的采样率，所以设置这个值为 8000 即可。SCI_AICTRL1 寄存器用于设置 AGC，1024 相当于数字增加 1，这里建议大家设置 AGC 在 4（4*1024）左右比较合适。SCI_AICTRL2 用于设置自动 AGC 的时候的最大值，当设置为 0 的时候表示最大 64(65536)，这个大家按自己的需要设置即可。最后，SCI_AICTRL3，我们本文档用到的是咪头线性 PCM 单声道录音，所以设置该寄存器值为 6。

通过这几个寄存器的设置，我们就激活 VS1053 的 PCM 录音了。不过，VS1053 的 PCM 录音有一个小 BUG，必须通过加载 patch 才能解决，如果不加载 patch，那么 VS1053 是不输出 PCM 数据的，VLSI 提供了我们这个 patch，只需要通过软件加载即可。

读取 PCM 数据

在激活了 PCM 录音之后，SCI_HDAT0 和 SCI_HDAT1 有了新的功能。VS1053 的 PCM 采样缓冲区由 1024 个 16 位数据组成，如果 SCI_HDAT1 大于 0，则说明可以从 SCI_HDAT0 读取至少 SCI_HDAT1 个 16 位数据，如果数据没有被及时读取，那么将溢出，并返回空的状态。

注意，如果 SCI_HDAT1≥896，最好等待缓冲区溢出，以免数据混叠。所以，对我们来说，只需要判断 SCI_HDAT1 的值非零，然后从 SCI_HDAT0 读取对应长度的数据，即完

成一次数据读取，以此循环，即可实现 PCM 数据的持续采集。

最后，我们看看本文档实现 WAV 录音需要经过哪些步骤：

1） 设置 VS1053 PCM 采样参数

这一步，我们要设置 PCM 的格式（线性 PCM）、采样率（8K）、位数（16 位）、通道数（单声道）等重要参数，同时还要选择采样通道（咪头），还包括 AGC 设置等。可以说这里的设置直接决定了我们 wav 文件的性质。

2） 激活 VS1053 的 PCM 模式，加载 patch

通过激活 VS1053 的 PCM 格式，让其开始 PCM 数据采集，同时，由于 VS1053 的 BUG，我们需要加载 patch，以实现正常的 PCM 数据接收。

3） 创建 WAV 文件，并保存 wav 头

在前两部设置成功之后，我们即可正常的从 SCI_HDAT0 读取我们需要的 PCM 数据了，不过在这之前，我们需要先在创建一个新的文件，并写入 wav 头，然后才能开始写入我们的 PCM 数据。

4） 读取 PCM 数据

经过前面几步的处理，这一步就比较简单了，只需要不停的从 SCI_HDAT0 读取数据，然后存入 wav 文件即可，不过这里我们还需要做文件大小统计，在最后的时候写入 wav 头里面。

5） 计算整个文件大小，重新保存 wav 头并关闭文件

在结束录音的时候，我们必须知道本次录音的大小（数据大小和整个文件大小），然后更新 wav 头，重新写入文件，最后因为 FATFS，在文件创建之后，必须调用 f_close，文件才会真正体现在文件系统里面，否则是不会写入的！所以最后还需要调用 f_close，以保存文件。

2、硬件设计

本文档实验功能简介：开机的时候先检测 SD 卡和字库，然后测试 VS1053(正弦测试与寄存器测试)，之后，检测 SD 卡根目录是否存在 RECORDER 文件夹，如果不存在则创建，如果创建失败，则报错。在找到 SD 卡的 RECORDER 文件夹后，即设置 VS1053 进入录音模式，此时可以在耳机听到 VS1053 采集的音频。KEY0 用于停止录音并保存；KEY1 用于开始/暂停录音；WK_UP 用于播放最近一次的录音。

当我们按下 KEY1 的时候，可以在屏幕上看到录音文件的名字，以及录音时间，然后通过 KEY0 可以停止录音并保存该文件（文件名和时间也都将清零），在完成一个录音后，我们可以通过按 WK_UP 按键，来试听刚刚的录音。DS1 用于提示程序正在运行。

本实验用到的资源如下：

1） ALIENTEK MiniSTM32 开发板 V3.0

2） TFTLCD 液晶模块

3） ATK-VS1053 MP3 模块

4） SD 卡

5） 耳机（非必须）

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

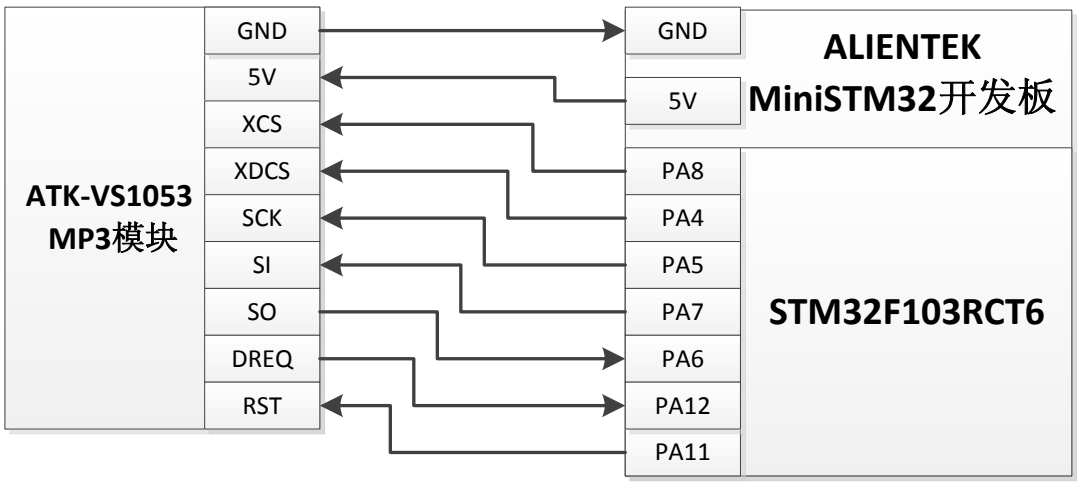

图 2.1 ATK-VS1053 MP3 模块与 MiniSTM32 开发板连接关系图

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：

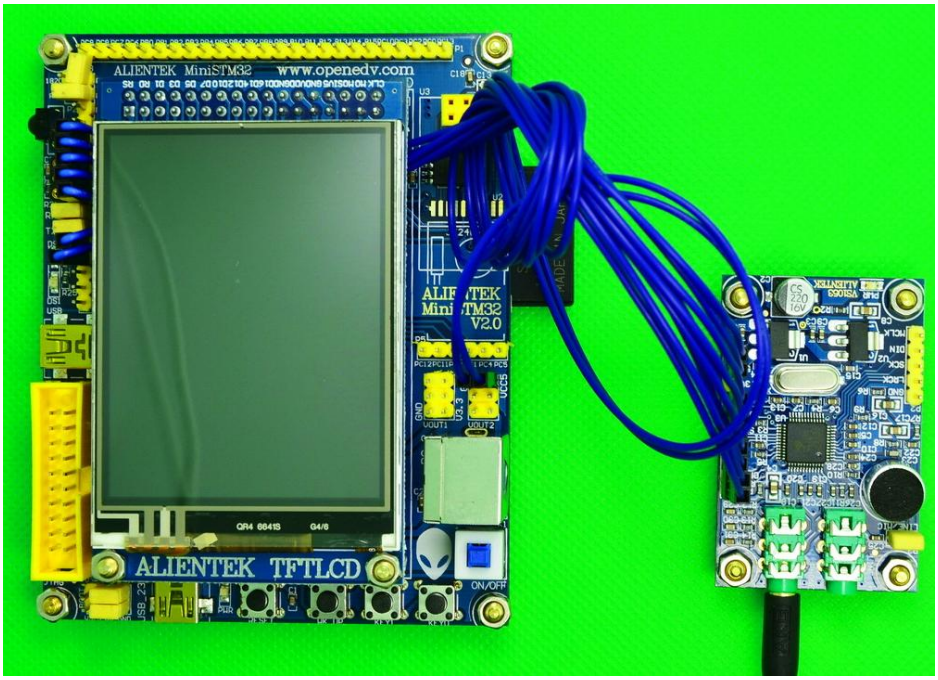

图 2.2 模块与 STM32 开发板连接实物图

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(开发板的 VOUT2)直接供电，完全与图 2.1 所示的关系图一致。

本实验，大家需要准备 1 个 SD 卡和一个耳机，分别插入 SD 卡接口和耳机接口，然后下载本实验就可以实现一个简单的录音机了。

3、软件实现

本文档，我们在 MiniSTM32 开发板 V3.0 的扩展例程 4 的基础上修改，先打开扩展例程 4 的工程，首先在 APP 文件夹下面新建 recorder.c 和 recorder.h 两个文件，然后将 recorder.c 加入到工程的 APP 组下。

因为 recorder.c 代码比较多，我们这里仅介绍其中的三个函数，首先是设置 VS1053 进

入 PCM 模式的函数：recoder_enter_rec_mode，该函数代码如下：

```
//进入 PCM 录音模式
//agc:0,自动增益.1024 相当于 1 倍,512 相当于 0.5 倍,最大值 65535=64 倍
void recoder_enter_rec_mode(u16 agc)
{
    //如果是 IMA ADPCM,采样率计算公式如下:
    //采样率=CLKI/256*d;
    //假设 d=0,并 2 倍频,外部晶振为 12.288M.那么 Fc=(2*12288000)/256*6=16Khz
    //如果是线性 PCM,采样率直接就写采样值
    VS_WR_Cmd(SPI_BASS,0x0000);
    VS_WR_Cmd(SPI_AICTRL0,8000);   //设置采样率,设置为 8Khz
    VS_WR_Cmd(SPI_AICTRL1,agc);
    //设置增益,0,自动增益.1024 相当于 1 倍,512 相当于 0.5 倍,最大值 65535=64 倍
    VS_WR_Cmd(SPI_AICTRL2,0);      //设置增益最大值,0,代表最大值 65536=64X
    VS_WR_Cmd(SPI_AICTRL3,6);      //左通道(MIC 单声道输入)
    VS_WR_Cmd(SPI_CLOCKF,0X2000);
    //设置 VS10XX 的时钟,MULT:2 倍频;ADD:不允许;CLK:12.288Mhz
    VS_WR_Cmd(SPI_MODE,0x1804);    //MIC,录音激活
    delay_ms(5);                   //等待至少 1.35ms
    VS_Load_Patch((u16*)wav_plugin,40);//VS1053 的 WAV 录音需要 patch
}
```

该函数就是用我们前面介绍的方法，激活 VS1053 的 PCM 模式，本章，我们使用的是 8Khz 采样率，16 位单声道线性 PCM 模式，AGC 通过函数参数设置。最后加载 patch（用于修复 VS1053 录音 BUG）。

第二个函数是初始化 wav 头的函数：recoder_wav_init，该函数代码如下：

```
//初始化 WAV 头.
void recoder_wav_init(__WaveHeader* wavhead) //初始化 WAV 头
{
    wavhead->riff.ChunkID=0X46464952; //"RIFF"
    wavhead->riff.ChunkSize=0;         //还未确定,最后需要计算
    wavhead->riff.Format=0X45564157;   //"WAVE"
    wavhead->fmt.ChunkID=0X20746D66; //"fmt "
    wavhead->fmt.ChunkSize=16;         //大小为 16 个字节
    wavhead->fmt.AudioFormat=0X01;     //0X01,表示 PCM;0X01,表示 IMA ADPCM
    wavhead->fmt.NumOfChannels=1;      //单声道
    wavhead->fmt.SampleRate=8000;      //8Khz 采样率 采样速率
    wavhead->fmt.ByteRate=wavhead->fmt.SampleRate*2;//16 位,即 2 个字节
    wavhead->fmt.BlockAlign=2;         //块大小,2 个字节为一个块
    wavhead->fmt.BitsPerSample=16;     //16 位 PCM
    wavhead->data.ChunkID=0X61746164;//"data"
    wavhead->data.ChunkSize=0;         //数据大小,还需要计算
}
```

该函数初始化 wav 头的绝大部分数据，这里我们设置了该 wav 文件为 8Khz 采样率，16 位线性 PCM 格式，另外由于录音还未真正开始，所以文件大小和数据大小都还是未知的，

要等录音结束才能知道。该函数__WaveHeader 结构体就是由前面介绍的三个 Chunk 组成，结构为：

```
//wav 头
typedef __packed struct
{
    ChunkRIFF riff;    //riff 块
    ChunkFMT fmt;    //fmt 块
    //ChunkFACT fact; //fact 块 线性 PCM,没有这个结构体
    ChunkDATA data;  //data 块
}__WaveHeader;
```

最后，我们介绍 recoder_play 函数，是录音机实现的主循环函数，该函数代码如下：

```
//录音机
//所有录音文件,均保存在 SD 卡 RECORDER 文件夹内.
u8 recoder_play(void)
{
    u8 res; u8 key; u8 rval=0;
    __WaveHeader *wavhead=0;
    u32 sectorsize=0; u16 w; u16 idx=0;
    FIL* f_rec=0;                    //文件
    DIR recdir;                      //目录
    u8 *recbuf;                      //数据内存
    u8 rec_sta=0;                    //录音状态
                                     //[7]:0,没有录音;1,有录音;
                                     //[6:1]:保留
                                     //[0]:0,正在录音;1,暂停录音;
    u8 *pname=0;
    u8 timecnt=0;                    //计时器
    u32 recsec=0;                    //录音时间
    u8 recagc=4;                     //默认增益为 4
    u8 playFlag=0;                   //播放标志
    while(f_opendir(&recdir,"0:/RECORDER"))//打开录音文件夹
    {
        Show_Str(60,230,240,16,"RECORDER 文件夹错误!",16,0); delay_ms(200);  、
        LCD_Fill(60,230,240,246,WHITE); delay_ms(200);  //清除显示
        f_mkdir("0:/RECORDER");                         //创建该目录
    }
    f_rec=(FIL *)mymalloc(sizeof(FIL));  //开辟 FIL 字节的内存区域
    if(f_rec==NULL)rval=1;               //申请失败
    wavhead=(__WaveHeader*)mymalloc(sizeof(__WaveHeader));//申请内存
    if(wavhead==NULL)rval=1;
    recbuf=mymalloc(512);
    if(recbuf==NULL)rval=1;
    pname=mymalloc(30);//申请 30 个字节内存,类似"0:RECORDER/REC00001.wav"
    if(pname==NULL)rval=1;
```

```
if(rval==0)                                    //内存申请 OK
{
    recoder_enter_rec_mode(1024*recagc);
    while(VS_RD_Reg(SPI_HDAT1)>>8);            //等到 buf 较为空闲再开始
    recoder_show_time(recsec);                 //显示时间
    recoder_show_agc(recagc);                  //显示 agc
    pname[0]=0;                                //pname 没有任何文件名
    while(rval==0)
    {
        key=KEY_Scan(0);
        switch(key)
        {
            case KEY0_PRES: //STOP&SAVE
                if(rec_sta&0X80)//有录音
                {
                    wavhead->riff.ChunkSize=sectorsize*512+36;//文件的大小-8;
                    wavhead->data.ChunkSize=sectorsize*512; //数据大小
                    f_lseek(f_rec,0);                       //偏移到文件头.
                    f_write(f_rec,(const void*)wavhead,
                    sizeof(__WaveHeader),&bw);//写入头数据
                    f_close(f_rec);
                    sectorsize=0;
                }
                rec_sta=0;
                recsec=0;
                LED1=1;                            //关闭 DS1
                LCD_Fill(60,230,240,246,WHITE);    //清除之前显示的文件名
                recoder_show_time(recsec);         //显示时间
                break;
            case KEY1_PRES: //REC/PAUSE
                if(rec_sta&0X01)//原来是暂停,继续录音
                {
                    rec_sta&=0XFE;//取消暂停
                }else if(rec_sta&0X80)//已经在录音了,暂停
                {
                    rec_sta|=0X01;      //暂停
                }else                   //还没开始录音
                {
                    rec_sta|=0X80;      //开始录音
                    recoder_new_pathname(pname);           //得到新的名字
                    Show_Str(60,230,240,16,pname+11,16,0); //显示文件名字
                    recoder_wav_init(wavhead); //初始化 wav 数据
                    res=f_open(f_rec,(const TCHAR*)pname,
                   FA_CREATE_ALWAYS|FA_WRITE);
```

```
                    if(res)                 //文件创建失败
                    {
                        rec_sta=0;      //创建文件失败,不能录音
                        rval=0XFE;      //提示是否存在 SD 卡
                    }else res=f_write(f_rec,(const void*)wavhead,
                    sizeof(__WaveHeader),&bw);//写入头数据
                }
                break;
            case WKUP_PRES://播放录音（仅在非录音状态下有效）
                if(rec_sta==0)playFlag=1;
        }
        if(rec_sta==0X80)//已经在录音了
        {
            w=VS_RD_Reg(SPI_HDAT1);
            if((w>=256)&&(w<896))
            {
                idx=0;
                while(idx<512)      //一次读取 512 字节
                {
                    w=VS_RD_Reg(SPI_HDAT0);
                    recbuf[idx++]=w&0XFF;
                    recbuf[idx++]=w>>8;
                }
                res=f_write(f_rec,recbuf,512,&bw);//写入文件
                if(res)
                {
                    printf("err:%d\r\n",res);
                    printf("bw:%d\r\n",bw);
                    break;//写入出错.
                }
                sectorsize++;//扇区数增加 1,约为 32ms
            }
        }else//没有开始录音，则检测 KEY1 按键，播放最近的录音
        {
            if(playFlag&&pname[0])//如果 wk_up 按键被按下,且 pname 不为空
            {
                Show_Str(60,230,240,16,"播放:",16,0);
                Show_Str(60+40,230,240,16,pname+11,16,0);//显示当播放的名字
                rec_play_wav(pname);                    //播放 pname
                LCD_Fill(60,230,240,246,WHITE);         //清除显示
                recoder_enter_rec_mode(1024*recagc);    //重新进入录音模式
                while(VS_RD_Reg(SPI_HDAT1)>>8);//等到 buf 较为空闲再开始
                recoder_show_time(recsec);              //显示时间
                recoder_show_agc(recagc);               //显示 agc
```

```
                    playFlag = 0;
                }
                delay_ms(5); timecnt++;
                if((timecnt%20)==0)LED1=!LED1;//DS1 闪烁
            }
            if(recsec!=(sectorsize*4/125))//录音时间显示
            {
                LED1=!LED1;                 //DS1 闪烁
                recsec=sectorsize*4/125;
                recoder_show_time(recsec);  //显示时间
            }
        }
    }
    myfree(wavhead);
    myfree(recbuf);
    myfree(f_rec);
    myfree(pname);
    return rval;
}
```

该函数实现了我们在硬件设计时介绍的功能，我们就不详细介绍了，请大家自己分析代码。recorder.c 的其他代码和 recorder.h 的代码我们这里就不再贴出了，请大家参考本例程源码（扩展实验 5）。保存 recorder.c，最后，我们在 test.c 里面修改 main 函数如下：

```
int main(void)
{
    u8 key,fontok=0;
    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
    LED_Init();             //LED 初始化
    KEY_Init();             //按键初始化
    VS_Init();              //初始化 VS1053
    usmart_dev.init(72);    //usmart 初始化
    mem_init();             //初始化内存池
    exfuns_init();          //为 fatfs 相关变量申请内存
    f_mount(fs[0],"0:",1);  //挂载 SD 卡
    f_mount(fs[1],"1:",1);  //挂载 FLASH.
RST:
    POINT_COLOR=RED;//设置字体为红色
    LCD_ShowString(60,30,200,16,16,"Mini STM32");
    LCD_ShowString(60,50,200,16,16,"RECORDER TEST");
    LCD_ShowString(60,70,200,16,16,"ATOM@ALIENTEK");
    LCD_ShowString(60,90,200,16,16,"KEY0:STOP&SAVE");
    LCD_ShowString(60,110,200,16,16,"KEY1:REC/PAUSE");
```

```
        LCD_ShowString(60,130,200,16,16,"WK_UP:PLAY ");
        LCD_ShowString(60,150,200,16,16,"2014/3/26");
        while(SD_Initialize())
        {
            LCD_ShowString(60,170,200,16,16,"SD Card Error"); delay_ms(200);
            LCD_Fill(20,170,200+20,170+16,WHITE); delay_ms(200);
        }
        fontok=font_init();         //检查字库是否 OK
        if(fontok)                  //需要更新字库
        {
            LCD_Clear(WHITE);               //清屏
            POINT_COLOR=RED;                //设置字体为红色
            LCD_ShowString(60,50,200,16,16,"ALIENTEK STM32");
            LCD_ShowString(60,70,200,16,16,"SD Card OK");
            LCD_ShowString(60,90,200,16,16,"Font Updating...");
            key=update_font(20,110,16);//从 SD 卡更新字库
            while(key)//更新失败
            {
                LCD_ShowString(60,110,200,16,16,"Font Update Failed!"); delay_ms(200);
                LCD_Fill(20,110,200+20,110+16,WHITE); delay_ms(200);
            }
            LCD_ShowString(60,110,200,16,16,"Font Update Success!"); delay_ms(1500);
            LCD_Clear(WHITE);//清屏
            goto RST;
        }
        while(1)
        {
            Show_Str(60,170,200,16,"存储器测试...",16,0);
            printf("Ram Test:0X%04X\r\n",VS_Ram_Test());//打印 RAM 测试结果
            Show_Str(60,170,200,16,"正弦波测试...",16,0);
            VS_Sine_Test();
            Show_Str(60,170,200,16,"<<录音机实验>>",16,0);
            recoder_play();
        }
    }
```

该函数先检测SD卡是不是在位（初始化SD卡），存在则继续，不存在在报错，然后检测外部flash是否存在字库，如果不存在字库，则更新字库，如果存在则继续下面的操作：对VS1053进行正弦测试和寄存器测试，随后调用录音机测试函数：recoder_play，开始录音测试，此时可以在耳机听到MIC的采集的声音了。

最后，我们将VS_WR_Cmd，这个函数加入usmart控制，这样我们就可以利用usmart来设置VS1053的一些参数了，这里主要是设置AGC，比如通过发送：VS_WR_Cmd(0X0D,4096)，将设置AGC的放大倍数为4。

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卡后，执行完两个测试（SIN测试和RAM测试）之后，就可以开始录音了，我们按下KEY1按键，就可以开始录音（再按KEY1可以暂停/继续录音）。如图4.1所示：

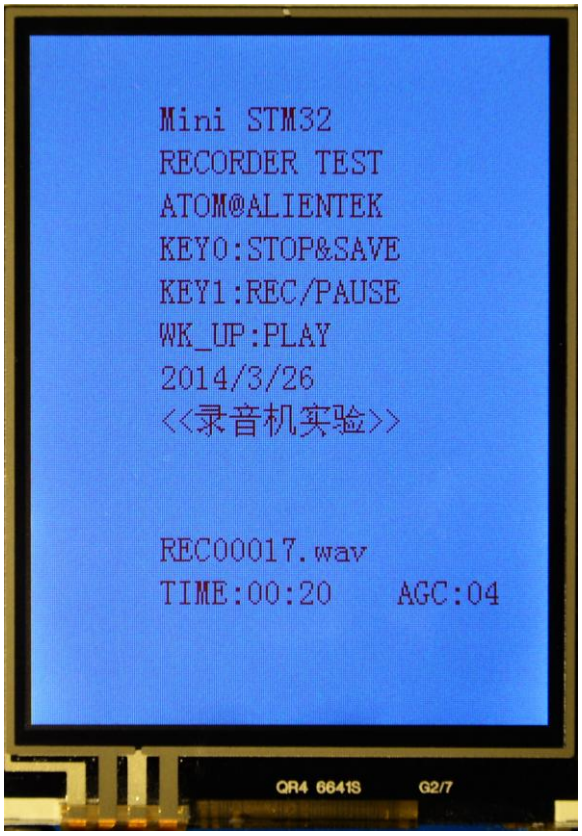

图4.1 WAV录音中

从上图可以看出，当前录音的文件为：REC00017.wav，已经录制了20秒，当前AGC设置为4，也就是4倍放大。此时我们按KEY0按键即可停止当前录音，并保存，然后再按WK_UP按键，就可以在耳机听到刚刚录制的音频文件了（LCD会显示：播放:REC00017.wav）。

我们将刚刚录制的音频文件放到电脑上面，可以通过属性查看该文件的属性，如图 4.2 所示：

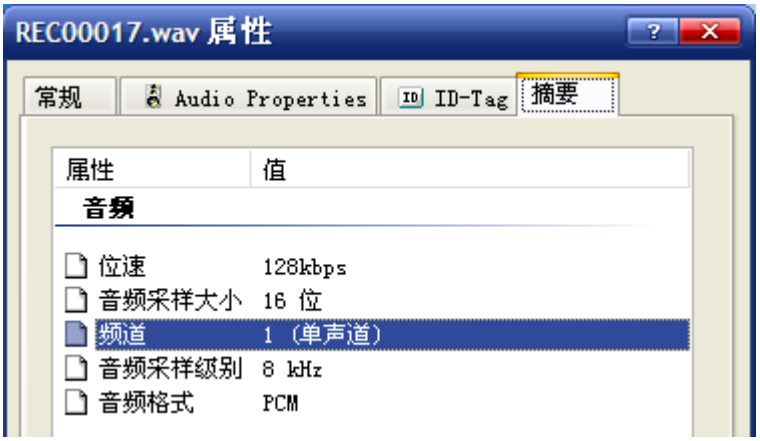

图 4.2 录音文件属性

这和我们预期的效果一样，通过电脑端的播放器（winamp/千千静听等）可以直接播放

我们所录的音频。效果还是非常不错的。

最后，本例程一定要自备SD卡一个，以存储录制的音频文件，然后**如果需要更新字库，请大家拷贝MiniSTM32开发板光盘：5，SD卡根目录文件 文件夹下面的SYSTEM文件夹到SD卡根目录，这样，才可以更新字库！**

至此，本例程结束，我们利用ATK-VS1053 MP3模块实现了一个录音机的功能，另外该模块还支持OGG录音，不过需要打补丁（加载patch），大家可以去vlsi官网（www.vlsi.fi ）了解相关信息。

正点原子@ALIENTEK
2014-03-26
公司网址：www.alientek.com
技术论坛：www.openedv.com
电话：020-38271790
传真：020-36773971