机械与电气工程学院

综合设计性实验报告

课程名称：音频 DSP 技术与应用

实验题目：实验 6　音频信号处理实验

班　　级：电信　151

姓　　名：苏伟强

学　　号：1507400051

日　　期：2018.7.2

目录

1 实验目的

复习巩固数字信号处理和音频 DSP 的基本原理和应用方法；掌握在 DSP 上实现数字音频处理算法的方法。

2 实验设备

硬件：ADSP-21489 EZ-Borad 开发板，串口转 USB 线；软件：Visual Studio 2017, Visual DSP++, 串口调试助手，虚拟串口。

3 界面设计与通信协议

要实现终端对 *DSP* 设备的控制，首先需要一个上位机控制程序，其次是设计好一个通信协议，以保证通信双方命令的正常接收和解析。

3.1 上位机界面设计

本实验上位机程序使用 *C++* 编写，基于 *MFC*，界面如图1所示，从上到下分为三个部分，第一部分是建立串口连接部分，用于选择串口，发送特定命令，测试串口的可用性等；第二部分为音频算法部分，包括限幅器，*ButterWorth* 滤波器，*Bessel* 滤波器；第三部分为基本功能部分，主要完成诸如音量控制，通道选择，输入/输出选择，音频直通等功能。

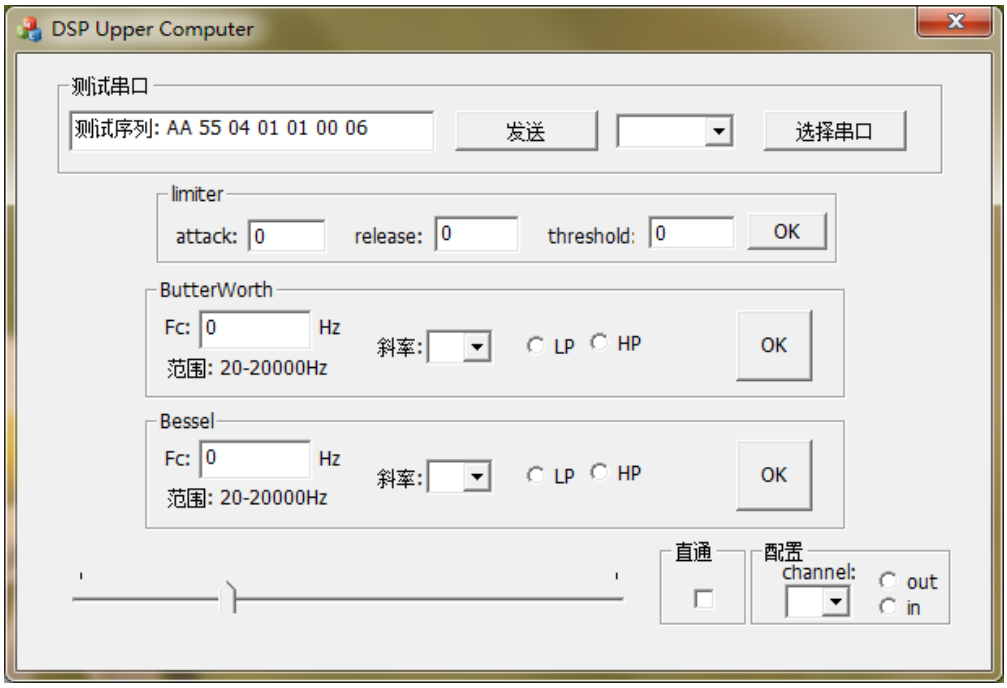

图 1: DSP 上位机程序

3.2 通信协议设计

本实验上位机与 *DSP* 板的通信方式为串口通信，所以需要编写一个串口通信协议以保证通信双方命令的正常接收和解析。基本的帧格式如图2所示：

图 2: 基本帧格式

其中，帧头部分为 0xAA, 0x55 占用 2 个字节；长度部分的计算方法为：$Byte(ID)+Byte(Paras)+1$，其中 $Byte$ 为求括号内容所占字节数，$Paras$ 为参数；ID 部分为功能的 ID 号；$Paras$ 部分为参数列表；CRC 部分为校验和，该位的计算方法为：长度 $+ID+sum(Paras)$。上述帧头，各功能命令长度，各功能 ID，已经在上位机程序中进行了宏定义，如下所示

MFCAppDlg.cpp

```
//帧头
#define   HEAD1          0xAA    //  参数的帧头1
#define   HEAD2          0x55    //  参数的帧头2
//参数个数
#define   VOLUMELEN        4       //  音量命令参数个数(长度)
#define   BYPASSLEN        3       //  直通命令参数个数
#define   VOLUMELEN        4       //  音量命令参数个数(长度)
#define   LIMITERLEN       10       //  限幅命令参数个数
#define   BUTTERLEN        8       //  巴特沃斯命令参数个数
#define   BESSELLEN        8       //  贝塞尔命令参数个数
//ID
#define   VOLUMEID         0x01    //  音量命令ID
#define   BYPASSID         0x03    //  直通命令ID
#define   VOLUMEID         0x01    //  音量命令ID
#define   LIMITERID        0x0D    //  限幅命令ID
#define   BUTTERID         0x09    //  巴特沃斯命令ID
#define   BESSELID         0x0B    //  贝塞尔命令ID
```

如图2所示，不同的功能有不同的参数列表，如下图3-5所示为举例音量调节，限幅器，滤波器的帧结构，可以看到他们携带着不同的参数，这些参数有些占有 2Byte，如图所示表示的颜色色块比较长，普通参数为 1Byte，这些参数一部分可以被宏定义。

图 3: 音量调节帧结构

图 4: 限幅器帧结构帧结构

图 5: 滤波器帧结构

设计好帧结构之后，还需要考虑上位机发送方式以及 DSP 端对命令的解析方式。这里上位机采用的是 16 进制发送，由于 DSP 端是一个字节一个字节接收数据的，DSP 端接收到之后可通过一个 $switch$ 来判断时候符合约定的帧结构顺序，如果是，保存，并且检测帧结构的下一位，如果不是则丢弃该帧，如此反复直到读取一个完整的帧，再解析这个帧，实现特定的动作和全局标志的修改。如下图6所示为其流程。

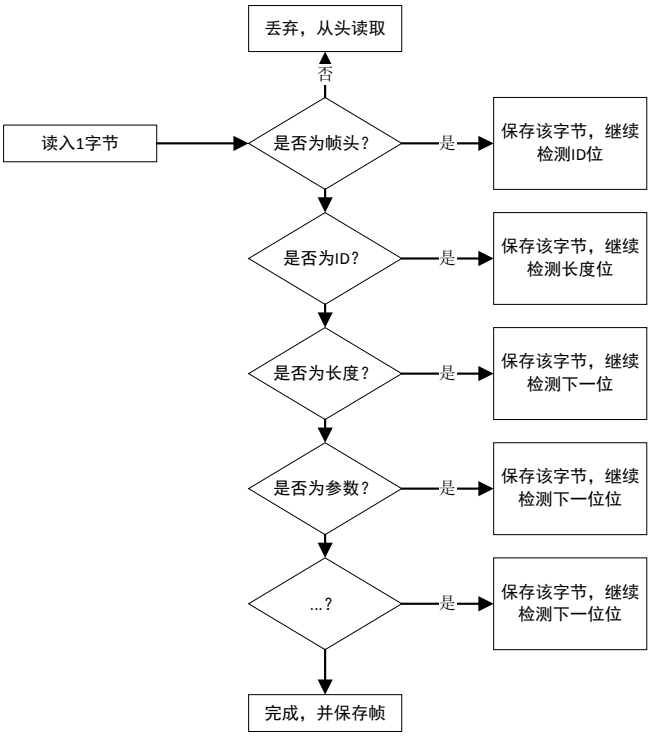

图 6: DSP 端接收数据并且保存帧，处理帧

4 算法原理和仿真结果

本实验，除基本功能外（直通，音量）外，实现的算法有 *Butterworth* 滤波器，*Bessel* 滤波器，以及限幅器 *Limiter*。以下介绍其原理及仿真结果。

4.1 Butterworth 滤波器原理及仿真

Butterworth 滤波器是一种通频带的频率响应曲线很平坦的信号处理滤波器。它也被称作最大平坦滤波器。巴特沃斯滤波器的特点是通频带内的频率响应曲线最大限度平坦，没有纹波，而在阻频带则逐渐下降为零。由于需要上位机实时控制滤波器参数，所以这里不使用直接在 *Matlab* 使用 *fdatool* 生成滤波器系数的方法，而是在嵌入式系统中实时根据上位机的设定需求来产生相应的滤波器参数。

归一化的 *Butterworth* 滤波器系统函数一般形式为 [2]：

$$H(s) = \frac{d_0}{a_0 + a_1 s + a_2 s^2 + \ldots + a_N s^N} \tag{1}$$

一般情况下会希望保持通带增益为 0dB，因此设定 $d_0 = a_0$。在 $a_0 = a_N = 1$ 情况下得到如图7归一化的巴特沃斯多项式：

n	多项式因子 $B_n(s)$
1	$(s+1)$
2	$s^2 + 1.414s + 1$
3	$(s+1)(s^2+s+1)$
4	$(s^2+0.7654s+1)(s^2+1.8478s+1)$
5	$(s+1)(s^2+0.6180s+1)(s^2+1.6180s+1)$
6	$(s^2+0.5176s+1)(s^2+1.414s+1)(s^2+1.9318s+1)$
7	$(s+1)(s^2+0.4450s+1)(s^2+1.247s+1)(s^2+1.8022s+1)$
8	$(s^2+0.3986s+1)(s^2+1.111s+1)(s^2+1.6630s+1)(s^2+1.9622s+1)$

图 7: 归一化 *Butterworth* 多项式

由此得到 $d_0 = a_0 = a_N = 1$ 情况下的 *Butterworth* 多项式展开的系数表 [2]，见表1 以上表达式是 s 域的表

表 1: *Butterworth* 多项式展开的系数表

N	a0	a1	a2	a3	a4	a5	a6	a7	a8	a9
1	1	1								
2	1	1.414214								
3	1	2	2							
4	1	2.613126	3.414214	2.613126						
5	1	3.236068	5.236068	5.236068	3.236068					
6	1	3.863703	7.464102	9.14162	7.464102	3.863703				
7	1	4.493959	10.09783	14.59179	14.59179	10.09783	4.493959			
8	1	5.125831	13.13707	21.84615	25.68836	21.84615	13.13707	5.125831		
9	1	5.758771	16.58172	31.16344	41.98639	41.98639	31.16344	16.58172	5.758771	
10	1	6.392453	20.43173	42.80206	64.8824	74.23343	64.8824	42.80206	20.43173	6.392453

达式，下面是变化到 z 域的方法。对于低通滤波器

$$s = \frac{1}{C_1}\frac{1-z^{-1}}{1+z^{-1}} \tag{2}$$

$$C_1 = \Omega_c \tan(\frac{w_c}{2}) \tag{3}$$

$$\Omega_c = 1, \quad w_c = \frac{2\pi f_c}{f_s} \tag{4}$$

对于高通滤波器

$$s = C_1\frac{1-z^{-1}}{1+z^{-1}} \tag{5}$$

$$C_1 = \Omega_c \tan(\frac{w_c}{2}) \tag{6}$$

$$\Omega_c = 1 \tag{7}$$

于是，对弈一阶 *Butterworth* 低通滤波器有

$$\frac{C_1}{C_1+1}H(z) = \frac{1}{\frac{1-z^{-1}}{C_1(1+z^{-1})+1}} = \frac{\frac{C_1}{C_1+1}+\frac{C_1}{C_1+1}z^{-1}}{1+\frac{C_1-1}{C_1+1}z^{-1}} \tag{8}$$

于是有

$$b_0 = b_1 = \frac{C_1}{C_1+1}, \qquad b_2 = 0 \tag{9}$$

$$a_0 = 1, \qquad a_1 = \frac{C_1-1}{C_1+1}, \qquad a_2 = 0 \tag{10}$$

同理得二阶低通滤波器的滤波器系数为

$$b_0 = GC_1^2, \qquad b_1 = 2b_0, \qquad b_2 = b_0 \tag{11}$$

$$a_0 = 1, \qquad a_1 = G(2C_1^2-2), \qquad a_2 = G(C_1^2 - 1.41421356C_1 + 1) \tag{12}$$

对于高通滤波器，结合式5，6，7，可得到一阶滤波器系数

$$b_0 = G, \qquad b_1 = -b_0, \qquad b_2 = 0 \tag{13}$$

$$a_0 = 1, \qquad a_1 = G(C_1-1), \qquad a_2 = 0 \tag{14}$$

二阶滤波器系数

$$b_0 = G, \qquad b_1 = -2G, \qquad b_2 = G \tag{15}$$

$$a_0 = 1, \qquad a_1 = 2G(C^2-1), \qquad a_2 = G(C^2 - 1.4142136C - 1) \tag{16}$$

通过以上公式得到对应滤波器的系数，然后对比使用 *Matlab* 的 *fdatool* 生成的滤波器的参数，看是否一致。若一致，则仿真结束，算法可行。

4.2 Bessel 滤波器原理及仿真

Bessel 是具有最大平坦的群延迟（线性相位响应）的线性滤波器，因而在通频带上保持了被过滤的信号波形而不参数畸变。

和 *Butterworth* 滤波器一样,*Bessel* 也可以将其转化为级联型结构进行构造,其参数的计算亦同 *Butterworth* 滤波器，这里不做赘述，需要注意的是 *Bessel* 滤波器是 *FIR* 系统，仅有参数 *b*。

同样的，通过以上公式得到对应滤波器的系数，然后对比使用 *Matlab* 的 *fdatool* 生成的滤波器的参数，看是否一致。若一致，则仿真结束，算法可行。

4.3 限幅器原理及仿真

Limiter 的作用是在把音频做最大化处理的同时限制一定音量以上的峰值，防止由于电平过大输入而造成失真，说白了就是一个压限器。参数 *Threshold* 为门限，*attack* 为起始时间，*Release* 为释放时间。

5 算法的 DSP 程序设计

下面列出，各个功能模块实现的函数，包括初始化函数还有功能函数的主要代码。

5.1 *Butterworth* 滤波器算法 DSP 实现

ButterWorth.c

```
/*****************************************************************************
* ButterWorth.c
*****************************************************************************/
// 头文件
#include <math.h>
#include <filter.h>
#include "ButterWorth.h"


/*****************************************************************************
* 函数名称：FirstOrderCoef
* 功能描述：根据一阶滤波器的2个参数（边界频率、系数）计算滤波器系数。
* 说明：
*          1、输入参数：
wc:边界频率
d: 一阶滤波器归一化多项式系数
FilterType: 滤波器类型（0 低通， 1 高通）

2、输出参数：
coef: 根据输入参数计算出来的滤波器系数.
按以下顺序排列系数：-a2,  -a1,   b2,   b1,   b0
3、返回值：
4、修改记录：
*                          修改内容：
*              姓     名：
*              日     期：

*
*****************************************************************************/
#pragma section ("ButterWorth_pmco ")
void FirstOrderCoef(float wc,float d,int FilterType,float pm *coef)
```

```
{

        float x,y,z;
        x = d*wc;
        y = 1+d*wc;
        z = -1+d*wc;
        if(FilterType==LOWPASS)
        {
                *coef = 0;                  // -a2
                *(coef + 1) = -z / y;       //-a1
                *(coef + 2) = 0;            //b2
                *(coef + 3) = wc / y;       //b1
                *(coef + 4) = wc / y;       //b0
        }
        else
        {
                *coef = 0 ;                                         //-a2
                *(coef + 1) = -( wc - d ) / ( wc + d );             //-a1
                *(coef + 2) = 0;                                                    //b2
                *(coef + 3) = -1 / ( d + wc );                              //b1
                *(coef + 4) =1 / ( d + wc );;                       //b0
        }

}


/*****************************************************************************
* 函数名称：SecondOrderCoef
* 功能描述：根据二阶滤波器的3个参数（边界频率、2个系数）计算滤波器系数。
* 说明：
*         1、输入参数：
wc:边界频率
k: 二阶滤波器归一化多项式系数     p2 + kp + c
c: 二阶滤波器归一化多项式系数
FilterType: 滤波器类型（0 低通， 1 高通）

2、输出参数：
coef: 根据输入参数计算出来的滤波器系数.
按以下顺序排列系数：-a2,   -a1,    b2,    b1,    b0
3、返回值：


4、修改记录：
*                       修改内容：
*             姓     名：
*             日     期：

*
*****************************************************************************/

#pragma section ("ButterWorth_pmco ")
void SecondOrderCoef(float wc,float k,float c,int FilterType,float pm *coef)
{
        float x,y,z,tmp1,tmp2;
        x = wc*wc;
        z = c*x;
        if(FilterType==LOWPASS)
        {
                tmp1 = 1 / (1+y+z);
```

```
91                  *coef = -(1-y+z) / (1+y+z);             // -a2
92                  *(coef + 1) = -(2*z-2) / (1+y+z);       //-a1
93                  *(coef + 2) = x / (1+y+z);                          //b2
94                  *(coef + 3) = (*(coef + 2)) * 2;                    //b1
95                  *(coef + 4) = *(coef + 2);                          //b0
96          }
97          else
98          {
99                  *coef = -(c-y+x)/(c+y+x);                           //-a2
100                 *(coef + 1) = -(2*x -2*c)/(c+y+x);                  //-a1
101                 *(coef + 2) = 1/(c+y+x);                            //b2
102                 *(coef + 3) = -(*(coef + 2)) * 2;       //b1
103                 *(coef + 4) = *(coef + 2);                          //b0
104         }
105
106 }
107
108
109 /*****************************************************************************
110 * 函数名称：CalulateButterWorthParam
111 * 功能描述：根据滤波器的参数计算滤波器系数。
112 * 说明：
113 *          1、输入参数：
114 f0:边界频率
115 Fs: 采样频率
116 Order: 滤波器阶数
117 FilterType: 滤波器类型（0 低通， 1 高通）
118
119 2、输出参数：
120 coef: 根据输入参数计算出来的滤波器系数.
121 按以下顺序排列系数：-a2,  -a1,   b2,   b1,   b0
122 3、返回值：
123
124
125 4、修改记录：
126 *                       修改内容：
127 *             姓    名：
128 *             日    期：
129
130 *
131 *****************************************************************************/
132 #pragma section ("ButterWorth_pmco␣")
133 int CalulateButterWorthParam(float pm *coef, int Fs, float f0, int Order,int FilterType)
134 {
135
136         // 计算滤波器系数的中间变量
137         float wc;
138         int Error = 0;
139         int k;
140         if(f0 < 20)
141         {
142                 f0 = 20;
143                 Error = 1;
144         }
145         if(f0 > 20000)
146         {
147                 f0 = 20000;
148                 Error = 1;
149         }
```

```
150
151         if(Order < 1)
152         {
153                 Order = 1;
154                 Error = 2;
155         }
156         if(Order > 8)
157         {
158                 Order = 8;
159                 Error = 2;
160         }
161         if(FilterType < 0)
162         {
163                 FilterType = 0;
164                 Error = 3;
165         }
166         if(FilterType > 1)
167         {
168                 FilterType = 1;
169                 Error = 3;
170         }
171         wc=tanf(pi*f0/Fs);
172         switch(Order)
173         {
174                 case 1:
175                 FirstOrderCoef(wc,1,FilterType,coef);
176                 break;
177
178                 case 2:
179                 SecondOrderCoef(wc,1.4142136,1,FilterType,coef);
180                 break;
181
182                 case 3:
183                 FirstOrderCoef(wc,1,FilterType,coef);
184                 SecondOrderCoef(wc,1,1,FilterType,(coef + 5));
185                 break;
186
187                 case 4:
188                 SecondOrderCoef(wc,0.7653669,1,FilterType,coef);
189                 SecondOrderCoef(wc,1.8477591,1,FilterType,(coef+5));
190                 break;
191
192                 case 5:
193                 FirstOrderCoef(wc,1,FilterType,coef);
194                 SecondOrderCoef(wc,0.61803399,1,FilterType,(coef+5));
195                 SecondOrderCoef(wc,1.618033989,1,FilterType,(coef+10));
196                 break;
197
198                 case 6:
199                 SecondOrderCoef(wc,0.51763809,1,FilterType,coef);
200                 SecondOrderCoef(wc,1.41423562,1,FilterType,(coef+5));
201                 SecondOrderCoef(wc,1.93185165,1,FilterType,(coef+10));
202                 break;
203
204                 case 7:
205                 FirstOrderCoef(wc,1,FilterType,coef);
206                 SecondOrderCoef(wc,0.44504187,1,FilterType,(coef+5));
207                 SecondOrderCoef(wc,1.2469796,1,FilterType,(coef+10));
208                 SecondOrderCoef(wc,1.8019377,1,FilterType,(coef+15));
```

```
209                 break;
210
211                 case 8:
212                 SecondOrderCoef(wc,0.3901806,1,FilterType,coef);
213                 SecondOrderCoef(wc,1.11114046,1,FilterType,(coef+5));
214                 SecondOrderCoef(wc,1.662939224,1,FilterType,(coef+10));
215                 SecondOrderCoef(wc,1.9615705608,1,FilterType,(coef+15));
216                 break;
217
218                 default:
219                 break;
220         }
221         return Error;
222 }
223 /*****************************************************************************
224 * 函数名称：InitButterWorthFilter
225 * 功能描述：ButterWorth滤波初始化函数，将ButterWorth滤波过程中所需的中间延时缓冲区置零。
226 * 修改记录：
227 *             1、
228 *                         修改内容：
229 *           姓     名：
230 *           日     期：
231 *
232 *****************************************************************************/
233 #pragma section ("ButterWorth_pmco␣")
234 void InitButterWorthFilter(float pm *coeff,float dm *state, int nstate)
235 {
236         int i;
237         for (i = 0; i < 20; i++)
238         {
239                 *(coeff + i) = 0.0; /* initialize state array */
240         }
241         *(coeff + 4) = 1.0;     // 令b0 = 1,相当于直通
242         *(coeff + 9) = 1.0;     // 令b0 = 1,相当于直通
243         *(coeff + 14) = 1.0;     // 令b0 = 1,相当于直通
244         *(coeff + 19) = 1.0;     // 令b0 = 1,相当于直通
245
246
247         for (i = 0; i < nstate; i++)
248         {
249                 *(state + i) = 0.0; /* initialize state array */
250         }
251 }
252
253
254 /*****************************************************************************
255 * 函数名称：ButterWorthFilter
256 * 功能描述：ButterWorth滤波函数
257 * 说明：
258 *         1、输入参数：
259 in：输入数据指针
260 coeffs:滤波器系数.
261 按以下顺序排列系数：-a2,   -a1,    b2,    b1,    b0
262 state：滤波器中间缓冲区
263 samples：输入数据长度
264 order：滤波器阶数
265 2、输出参数：
266 out：经ButterWorth滤波处理后的输出数据
267 3、返回值：
```

```
268 无
269 *              4、
270 *                          修改内容：
271 *            姓      名：
272 *            日      期：
273 *
274 ***************************************************************************/
275 #pragma section ("ButterWorth_pmco ")
276 void ButterWorthFilter(float *in, float *out, float pm *coeffs,
277                         float dm *state, int samples,int order)
278 {
279         // 对输入数据进行ButterWorth滤波处理
280         int sec;
281         sec = ( 1 + order ) / 2;   // 二阶级的个数
282         biquad (in, out, coeffs, state, samples, sec);
283 }
```

5.2 *Beseel* 滤波器算法 DSP 实现

Bessel.c

```
1  /***************************************************************************
2  * Bessel.c
3  ***************************************************************************/
4  // 头文件
5  #include <math.h>
6  #include <filter.h>
7  #include "Bessel.h"
8  /***************************************************************************
9  * 函数名称：FirstOrderCoef
10 * 功能描述：根据一阶滤波器的2个参数（边界频率、系数）计算滤波器系数。
11 * 说明：
12 *          1、输入参数：
13 wc:边界频率
14 d: 一阶滤波器归一化多项式系数
15 FilterType: 滤波器类型（0 低通， 1 高通）
16
17 2、输出参数：
18 coef: 根据输入参数计算出来的滤波器系数.
19 按以下顺序排列系数：-a2,   -a1,    b2,    b1,    b0
20 3、返回值：
21
22 4、修改记录：
23 *                          修改内容：
24 *              姓      名：
25 *              日      期：
26 *
27 ***************************************************************************/
28 #pragma section ("Bessel_pmco ")
29 void FirstOrderCoef(float wc,float d,int FilterType,float pm *coef)
30 {
31         float x,y,z;
32         x = d*wc;
33         y = 1+d*wc;
34         z = -1+d*wc;
35         if(FilterType==LOWPASS)
36         {
37                 *coef = 0;              // -a2
```

```
38                  *(coef + 1) = -z / y;    //-a1
39                  *(coef + 2) = 0;         //b2
40                  *(coef + 3) = wc / y;    //b1
41                  *(coef + 4) = wc / y;    //b0
42          }
43          else
44          {
45                  *coef = 0 ;                                  //-a2
46                  *(coef + 1) = -( wc - d ) / ( wc + d ); //-a1
47                  *(coef + 2) = 0;                                                  //b2
48                  *(coef + 3) = -1 / ( d + wc );                      //b1
49                  *(coef + 4) =1 / ( d + wc );;             //b0
50          }
51  }
52  /*****************************************************************************
53  * 函数名称：SecondOrderCoef
54  * 功能描述：根据二阶滤波器的3个参数（边界频率、2个系数）计算滤波器系数。
55  * 说明：
56  *          1、输入参数：
57  wc:边界频率
58  k: 二阶滤波器归一化多项式系数     p2 + kp + c
59  c: 二阶滤波器归一化多项式系数
60  FilterType: 滤波器类型（0 低通， 1 高通）
61
62  2、输出参数：
63  coef: 根据输入参数计算出来的滤波器系数.
64  按以下顺序排列系数：-a2,   -a1,    b2,    b1,    b0
65  3、返回值：
66  4、修改记录：
67  *                          修改内容：
68  *              姓     名：
69  *              日     期：
70
71  *
72  *****************************************************************************/
73  #pragma section ("Bessel_pmco␣")
74  void SecondOrderCoef(float wc,float k,float c,int FilterType,float pm *coef)
75  {
76          float x,y,z,tmp1,tmp2;
77          x = wc*wc;
78          y = k*wc;
79          z = c*x;
80          if(FilterType==LOWPASS)
81          {
82                  tmp1 = 1 / (1+y+z);
83                  *coef = -(1-y+z) / (1+y+z);              // -a2
84                  *(coef + 1) = -(2*z-2) / (1+y+z);        //-a1
85                  *(coef + 2) = x / (1+y+z);                          //b2
86                  *(coef + 3) = (*(coef + 2)) * 2;                    //b1
87                  *(coef + 4) = *(coef + 2);                          //b0
88          }
89          else
90          {
91                  *coef = -(c-y+x)/(c+y+x);                           //-a2
92                  *(coef + 1) = -(2*x -2*c)/(c+y+x);                  //-a1
93                  *(coef + 2) = 1/(c+y+x);                            //b2
94                  *(coef + 3) = -(*(coef + 2)) * 2;        //b1
95                  *(coef + 4) = *(coef + 2);                          //b0
96          }
```

```
97 }
98 /*****************************************************************************
99 * 函数名称：CalulateBesselParam
100 * 功能描述：根据滤波器的参数计算滤波器系数。
101 * 说明：
102 *         1、输入参数：
103 f0:边界频率
104 Fs: 采样频率
105 Order: 滤波器阶数
106 FilterType: 滤波器类型（0 低通， 1 高通）
107
108 2、输出参数：
109 coef: 根据输入参数计算出来的滤波器系数.
110 按以下顺序排列系数：-a2,  -a1,   b2,   b1,   b0
111 3、返回值：
112 4、修改记录：
113 *                          修改内容：
114 *             姓     名：
115 *             日     期：
116
117 *
118 *****************************************************************************/
119 #pragma section ("Bessel_pmco␣")
120 int CalulateBesselParam(float pm *coef, int Fs, float f0, int Order,int FilterType)
121 {
122         // 计算滤波器系数的中间变量
123         float wc;
124
125         int Error = 0;
126         int k;
127
128         if(f0 < 20)
129         {
130                 f0 = 20;
131                 Error = 1;
132         }
133         if(f0 > 20000)
134         {
135                 f0 = 20000;
136                 Error = 1;
137         }
138         if(Order < 1)
139         {
140                 Order = 1;
141                 Error = 2;
142         }
143         if(Order > 8)
144         {
145                 Order = 8;
146                 Error = 2;
147         }
148
149         if(FilterType < 0)
150         {
151                 FilterType = 0;
152                 Error = 3;
153         }
154         if(FilterType > 1)
155         {
```

```
156                 FilterType = 1;
157                 Error = 3;
158         }
159         wc=tanf(pi*f0/Fs);
160         switch(Order)
161         {
162                 case 1:
163                 FirstOrderCoef(wc,1,FilterType,coef);
164                 break;
165
166                 case 2:
167                 SecondOrderCoef(wc,1.7320508,1,FilterType,coef);
168                 break;
169
170                 case 3:
171                 FirstOrderCoef(wc,0.9416,FilterType,coef);
172                 SecondOrderCoef(wc,1.4912808,1.0620221,FilterType,(coef + 5));
173                 break;
174
175                 case 4:
176                 SecondOrderCoef(wc,1.31442234,1.12109453,FilterType,coef);
177                 SecondOrderCoef(wc,1.8095176,0.891985,FilterType,(coef+5));
178                 break;
179
180                 case 5:
181                 FirstOrderCoef(wc,0.92644208,FilterType,coef);
182                 SecondOrderCoef(wc,1.181151889,1.17180405,FilterType,(coef+5));
183                 SecondOrderCoef(wc,1.70310724,0.9211423,FilterType,(coef+10));
184                 break;
185
186                 case 6:
187                 SecondOrderCoef(wc,1.0771054,1.2148822,FilterType,coef);
188                 SecondOrderCoef(wc,1.8187814,0.861474582,FilterType,(coef+5));
189                 SecondOrderCoef(wc,1.599308372,0.9558388,FilterType,(coef+10));
190                 break;
191
192                 case 7:
193                 FirstOrderCoef(wc,0.91948716,FilterType,coef);
194                 SecondOrderCoef(wc,0.99338345,1.25172598,FilterType,(coef+5));
195                 SecondOrderCoef(wc,1.760005868,0.87787371,FilterType,(coef+10));
196                 SecondOrderCoef(wc,1.50547109,0.98972154,FilterType,(coef+15));
197                 break;
198
199                 case 8:
200                 SecondOrderCoef(wc,0.92434808,1.2835648,FilterType,coef);
201                 SecondOrderCoef(wc,1.819366309,0.8474733,FilterType,(coef+5));
202                 SecondOrderCoef(wc,1.69465016,0.899352096,FilterType,(coef+10));
203                 SecondOrderCoef(wc,1.422276362,1.02217782,FilterType,(coef+15));
204                 break;
205
206                 default:
207                 break;
208         }
209
210 return Error;
211 }
212
213 /*****************************************************************************
214 * 函数名称：InitBesselFilter
```

```
215 * 功能描述：Bessel滤波初始化函数，将Bessel滤波过程中所需的中间延时缓冲区置零。
216 * 修改记录：
217 *            1、
218 *                        修改内容：
219 *           姓    名：
220 *           日    期：
221 *
222 ***************************************************************************/
223 #pragma section ("Bessel_pmco ")
224 void InitBesselFilter(float pm *coeff,float dm *state, int nstate)
225 {
226         int i;
227         for (i = 0; i < 20; i++)
228         {
229                 *(coeff + i) = 0.0; /* initialize state array */
230         }
231         *(coeff + 4) = 1.0;     // 令b0 = 1,相当于直通
232         *(coeff + 9) = 1.0;     // 令b0 = 1,相当于直通
233         *(coeff + 14) = 1.0;     // 令b0 = 1,相当于直通
234         *(coeff + 19) = 1.0;     // 令b0 = 1,相当于直通
235         for (i = 0; i < nstate; i++)
236         {
237                 *(state + i) = 0.0; /* initialize state array */
238         }
239 }
240
241
242 /***************************************************************************
243 * 函数名称：BesselFilter
244 * 功能描述：Bessel滤波函数
245 * 说明：
246 *         1、输入参数：
247 in: 输入数据指针
248 coeffs:滤波器系数.
249 按以下顺序排列系数：-a2,   -a1,    b2,    b1,    b0
250 state: 滤波器中间缓冲区
251 samples: 输入数据长度
252 order: 滤波器阶数
253 2、输出参数：
254 out: 经Bessel滤波处理后的输出数据
255 3、返回值：
256 无
257 *             4、
258 *                        修改内容：
259 *           姓    名：
260 *           日    期：
261 *
262 ***************************************************************************/
263 #pragma section ("Bessel_pmco ")
264 void BesselFilter(float *in, float *out, float pm *coeffs, float dm *state, int samples,int order)
265 {
266         // 对输入数据进行Bessel滤波处理
267         int sec;
268         sec = ( 1 + order ) / 2;   // 二阶级的个数
269         biquad (in, out, coeffs, state, samples, sec);
270 }
```

5.3 压限器算法 DSP 实现

Limiter.c

```
/*****************************************************************************
* Limiter.c
*****************************************************************************/
// 头文件
#include <math.h>
#include <filter.h>
#include "Limiter.h"

#pragma section ("LimiterVar_dmda")
float T;
#pragma section ("LimiterVar_dmda")
float AT;
#pragma section ("LimiterVar_dmda")
float RT;
#pragma section ("LimiterVar_dmda")
float TAV;
#pragma section ("LimiterVar_dmda")
float LastDynamicGain;
#pragma section ("LimiterVar_dmda")
float Slope_log;
#pragma section ("LimiterVar_dmda")
float Threshold_log;
#pragma section ("LimiterVar_dmda")
float Thresh_amp;
#pragma section ("LimiterCoef_pmda")
float pm Coeffs[5]={0, -0.0068, 0, 0, 0.9932};  // 一个二阶节，用于电平RMS
/*****************************************************************************
* 函数名称：InitLimiter
* 功能描述：Limiter滤波初始化函数，将Limiter滤波过程中所需的中间延时缓冲区置零。
* 修改记录：
*            1、
*                       修改内容：
*           姓    名：
*           日    期：
*
*****************************************************************************/
#pragma section ("Limiter_pmco␣")
void InitLimiter(float dm *state, int nstate)
{
        int i;
        for (i = 0; i < nstate; i++)
        {
                *(state + i) = 0.0; /* initialize state array */
        }

}
int CalulateLimiterParam(float Threshold, float CompRatio, float Attack,
                 float Release,float RefAmp,float *LimiterParam,int fs)
{
        // 判断输入参数是否合法
        int Error = 0;
        float T;
        if(Threshold < -40)
        {
                Threshold = -40;
```

```
56                  Error = 1;
57          }
58          if(Threshold > 20)
59          {
60                  Threshold = 20;
61                  Error = 1;
62          }
63
64          if(CompRatio < 1)
65          {
66                  CompRatio = 1;
67                  Error = 2;
68          }
69          if(CompRatio > 10)
70          {
71                  CompRatio = 10;
72                  Error = 2;
73          }
74          if(Attack < 0.0001)
75          {
76                  Attack = 0.0001;
77                  Error = 3;
78          }
79          if(Attack > 0.2)
80          {
81                  Attack = 0.2;
82                  Error = 3;
83          }
84          if(Release < 0.001)
85          {
86                  Release = 0.001;
87                  Error = 4;
88          }
89          if(Release > 3)
90          {
91                  Release = 3;
92                  Error = 4;
93          }
94          T= (float)1/fs;
95          *LimiterParam = T;
96          *(LimiterParam + 1) = 1-expf(-2.2*T/Attack);
97          *(LimiterParam + 2) = 1-expf(-2.2*T/Release);
98          *(LimiterParam + 3) = 1 - (1/CompRatio);
99          Thresh_amp = powf(10,(Threshold/20)) * RefAmp;
100         *(LimiterParam + 4) = Thresh_amp;
101         *(LimiterParam + 5) = log10f(Thresh_amp);
102         return Error;
103 }
104
105 /*****************************************************************************
106 * 函数名称：Limiter
107 * 功能描述：Limiter滤波函数
108 * 修改记录：
109 *            1、
110 *                        修改内容：
111 *           姓     名：
112 *           日     期：
113 *
114 *****************************************************************************/
```

```
115
116 #pragma section ("Limiter_pmco ")
117 void Limiter(float *in, float *out,float *f, float *g,float
118                         *CompState,float *state,float *LimiterParam,int samples)
119 {
120         int i;
121         T = *LimiterParam;
122         AT = *(LimiterParam + 1);
123         RT = *(LimiterParam + 2);
124         Slope_log = *(LimiterParam + 3);
125         Thresh_amp  = *(LimiterParam + 4);
126         Threshold_log = *(LimiterParam + 5);
127         LastDynamicGain = g[samples-1];
128         for( i = 0; i < samples; i++ )
129         {
130                 CompState[i] = in[i] * in[i];
131         }
132         biquad (CompState, out, Coeffs, state, samples, 1);      // 计算电平均值
133         for( i = 0; i < samples; i++ )
134         {
135                 CompState[i] = log10f(out[i]);
136         }
137         // determine static gain f = 10^(-LS*(X-LT))
138
139         for ( i = 0; i < samples; i++ )
140         {
141                 if (out[i] >= Thresh_amp )
142                 {
143                         f[i]=powf(10,(-Slope_log*(CompState[i]-Threshold_log)));
144                 }
145                 else
146                 {
147                         f[i] = 1.0;
148                 }
149         }
150 //---------------- dynamic gain ----------------
151
152         if (out[0] >= Thresh_amp )
153         {
154                 g[0] = (1 - AT)*LastDynamicGain + AT*f[0]; //dynamic gain during attack
155         }
156         else
157         {
158                 g[0] = (1 - RT)*LastDynamicGain + RT*f[0];  // dynamic gain during release
159         }
160
161         for ( i = 1; i < samples; i++ )
162         {
163                 if (out[i] >= Thresh_amp )
164                 {
165                         g[i] = (1 - AT)*g[i-1] + AT*f[i]; //dynamic gain during attack
166                 }
167                 else
168                 {
169                         g[i] = (1 - RT)*g[i-1] + RT*f[i];  // dynamic gain during release
170                 }
171         }
172         LastDynamicGain = g[samples-1];
173         // gain adjusted output
```

```
174         for ( i = 0; i < samples; i++ )
175         {
176                 out[i] = in[i] * g[i];
177         }
178 }
```

6 实验结果分析

7 结论

在使用上位机控制 *DSP* 板块，每次控制一个通道，也就是说每一发送命令，只能对一个通道起作用，要实现两个通道都要一样的效果，还需要选择另一个通道，然后发送同样的配置命令。这样处理的好处是方便对音频信号的精准处理，可以处理多个单独通道的信号。实验的结论是，Butterworth 和 Beseel 都能实现高通和低通的滤波，理论上讲，*Bessel* 在通带内邮线性相位，波形不失真，但是实际上并没有听出区别，高通滤波的时候，高频分量被保留，低频分量被滤去，音频音调较高，声音较为尖锐，低通滤波的时候低频分量被保留，高频分量被滤去，声音低沉。

大部分结论已经在实验过程的叙述中给出，这里不做赘述。

参考文献

[1] Butterworth S. On the theory of filter amplifiers[J]. Wireless Engineer, 1930, 7(6): 536-541.

[2] 程佩青. 数字信号处理教程 [M]. 清华大学出版社有限公司, 2001.

[3] 陈后金. 信号与系统 [M]. 清华大学出版社有限公司, 2003.

[4] 陈后金. 数字信号处理.2 版 [M]. 高等教育出版社, 2008.