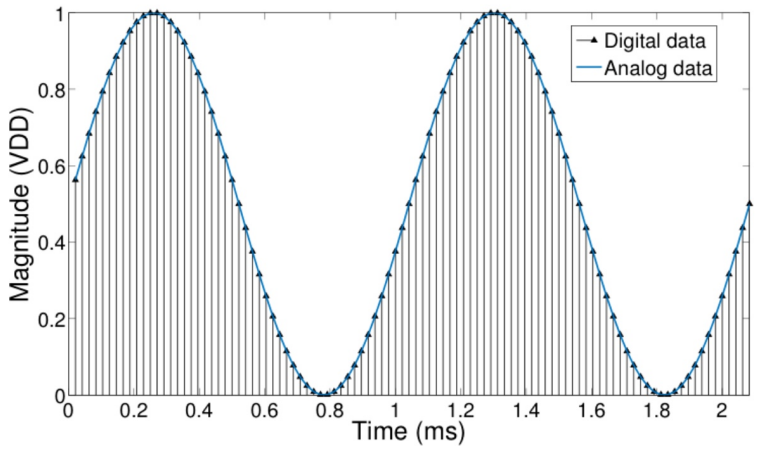
1、实验目的

学习音频信号的输出方式。

学习如何将数字信号转换为模拟信号的基本原理。

2、实验原理

人耳可以听到的声音的频率范围是 20-20kHz。音频设备如扬声器或耳机等所接收的音频信号一般是模拟信号，即时间上连续的信号。但是，由于数字器件只能以特定间隔产生数字输出，我们需要通过数字/模拟转换将数字信号转换成模拟信号输出。根据采样定律，数字信号的采样率(每秒钟产生的数字样本数量)应不低于信号频率的两倍。所以，数字音频一般采用 44.1kHz (CD 音频)或 48kHz 的采样率，以保证 20kHz 的信号不会失真。数字信号到模拟信号的转换示意图如下：



上图显示了数字信号到模拟信号转换的基本原理。在 48kHz 的采样率下，每间隔 1/48000 秒(1/48 毫秒)的时间产生一个数字输出样本点，如图中黑色小三角所示。该输出经过平滑后，会产生一个对应的模拟信号。假设需要产生一个人耳能听到的单频率 f=960Hz 的正弦波，我们需要在合适的时间点上设置(或输出)合适的数字值来形成正弦波形。对于一个正弦波信号 s(t)，其数学表达式是：$s(t)=\sin(2\pi ft)$。其中 f 为频率，t 为时间。在数字信号中，用整数来标记各个数字样本，样本编号顺序依次为 0, 1, 2, 3, 4, . . .当采样率是 48kHz 时，每两个点之间的间隔是 1/48 毫秒。此时，我们可以将 t 改写成 t=n/48000 秒。这样上式就变成：$s(n)=\sin(2\pi fn/48000)$。代入 f=960Hz，我们得到 $s(n)=\sin(2\pi\times 960n)=\sin(2\pi n)$。所以，对于整数 n 来说，每 50 个点对应正弦波的一个周期。仔细观察上图中的 960Hz 正弦波可知，其周期为 1/960 秒(1.042 毫秒)，对应 50 个样

本点。

在实际信号输出时，我们一般不采用浮点数而选用整数值来表示每个样本点的大小。这个过程一般称为量化。假设我们用带符号的 16bit 整数(补码)来表示单个样本点，此时 32767 即对应输出的最大值(例如+1V 电压)， -32767 即对应输出的最小值(例如-1V 电压)。这时，我们就可以通过循环输出 50 个点的整数值 s(n)（平均）= round (s(n) × 32767)来产生一个 sin 波形，如下表所示：

n	0	1	2	3	…	48	49	50	51
$s(n)$	0	0.125	0.249	0.368	…	-0.249	-0.125	0	0.125
$\bar{s}(n)$	0	4107	8149	12062	…	-8149	-4107	0	4107

这时，如果我们用一个计数器不停从 0 至 49 计数，输出对应的整数值就可以不断产生连续的 960Hz 正弦波。量化过程中采用的整数位数越多，信号的精度也就更高。

上例中的正弦波是固定频率为 960Hz 的。实际应用中，如果要产生不同频率的正弦波，就不能采用简单计数的方式，而需要采用查三角函数表的方式。假设我们在存储器中存储了一张 1024 点的 sin 函数表。即存储器中以地址 k=0…1023 存储了 1024 个三角函数值(以 16bit 补码整数表示)，地址为 k 的数值设置为 round(sin(2πk/1024)*32767)。对于任意频率 f 的正弦波，在第 n 个样本点需要输出的值为 round(sin(2πk/48000)*32767)。比较上述两式，得到 k/1024= nf/48000。因此，在函数表中最接近这个值的表项应该是 k= round(nf*1024/48000)mod1024。函数表中的项目越多，查找到的函数值越精确，但一般情况下人的耳朵往往可以容忍 1%-5%的误差。

但是，在 FPGA 中要计算乘除法取整等操作比较复杂，我们实际应用中采取累加的方法。我们观察到 n 每增加 1，对应的 k 会增加 f*1024/48000。这样，可以通过每个样本点将 k 递增 f*1024/48000 来避免乘除法。例如，f=960Hz 时，可以每个样本点对 k 递增 960*1024/48000=20.48。这样，50 个点后正好 k 增加了 1024，完成一个周期。但是，这里的 k 是整数，如何能够每次递增一个小数值呢？这里，可以采用定点小数的方式，即用 16bit 来表示 k。其中前 10bit 是整数部分，用来查三角函数表，后面 6bit 是小数部分，用来提高精度。这时，如果将此 16bit 数看成是一个无符号整数的话，每次需要递增 960*65536/48000 =1311，对应的小数值是 1311/64=20.4843。从整数角度来看，50 个样本点递增了 65550，这个值略大于 65536 一些，会对周期带来一些小的误差，但是这样的误差是可以容忍的。

因此，生成频率为 f 的正弦波的过程如下：根据频率 f 计算递增 d=f*65536/48000；在系统中维持一个 16bit 无符号整数计数器，每个样本点递增 d。根据 16 位无符号整数计数器的高 10 位获取查表地址 k。使用查表结果作为当前的数字输出。

附钢琴音高频率表：

音名/八度	3	4	5	6
C	130.81Hz	261.63 Hz	523.25 Hz	1046.5 Hz
$C^{\#}$	138.59 Hz	277.18 Hz	554.37 Hz	1108.7 Hz
D	146.83 Hz	293.66 Hz	587.33 Hz	1174.7 Hz
$D^{\#}$	155.56 Hz	311.13 Hz	622.25 Hz	1244.5 Hz
E	164.81 Hz	329.63 Hz	659.26 Hz	1318.5 Hz
F	174.61 Hz	349.23 Hz	698.46 Hz	1396.9 Hz
$F^{\#}$	185.00 Hz	369.99 Hz	739.99 Hz	1480.0 Hz
G	196.00 Hz	392.00 Hz	783.99 Hz	1568.0 Hz
$G^{\#}$	207.65 Hz	415.30 Hz	830.61 Hz	1661.2 Hz
A	220.00 Hz	440.00 Hz	880.00 Hz	1760.0 Hz
$A^{\#}$	233.08 Hz	466.16 Hz	932.33 Hz	1864.7 Hz
B	246.94 Hz	493.88 Hz	987.77 Hz	1975.5 Hz

3、实验环境 / 器材

环境：Quartus Prime 17.1；器材：DE10-Standard FPGA 开发板，ps2 键盘，耳机。

4、流程图及程序代码

考虑进行分模块设计。考虑初级功能的单键发音，以及高级功能的调音量以及和声。

实验指导中附带了一个 sound_sample 的工程，里面已经写好了在给定频率的情况下对三角函数波形设置以及向芯片传输信息的功能。因此需要在这个架构上做二次开发。就已经给出的基础架构工程来说，I2C 接口单个命令发送代码的实现在模块 I2C_Controller 中，音频芯片配置实现在模块 I2C_Audio_Config 中，该配置设置了正常音量(0dB)，使用 48kHz 双声道每个声道 16bit 的 I2S 通信设置，并且将 I2S 的 XCLK 设置为 18.432MHz(48kHz 的 384 倍)。通过调用 Quartus 提供的标准 IP 库来产生 18.432MHz 特殊频率时钟，改配置基础架构已经完成。另外，给定频率后通过计算得到 16 位数据，其高 10 位获取查表地址 k。使用查表结果作为当前的数字输出。查表功能以及作为表的存储器以及 mif 文件都已经

给出。因此需要修改的最主要的地方在于三角函数生成器 Sin_Generator 中的第三个参数，这个参数决定了开发板发出的声音的频率。因此，需要通过与键盘接口联系起来，通过按下的键决定发出的声音。

现在可以将这里需要实现的基本功能，即按下键发音，松开键不发音与实验八类比。这里发音即是根据键盘发送的键码确定频率，类比实验八中的根据键码确定 ASCII 码；这里发音以及停止发音可以类比实验八中七段数码管的显示，如果按键，则显示；如果松开，则灯灭。因此对于基本功能，可以承用实验八中对于键盘键码处理的模块，并且做一些改动。键码处理模块命名为 newps2board，定义输入数据如下：clk 为开发板系统 50Hz 时钟，reset 为清零端（高电平有效），ps2_clk 是键盘发送的时钟信号，ps2_data 是键盘 在其时钟信号下降沿发送的数据信息。中间变量如下：buffer 是接收到的连续 10 位键盘数据，其中包括开始的 0，中间的 8 位键码信息以及奇偶校验位（最后还有一个 1，被抛掉），作为缓冲信息，count 是一个计数器，负责计算连续从键盘接收到多少位数据（12 位算一组数据）。另外添加的 nowdata，作为锁存当前键码；off，原来作为时序中数码管灯亮灭的指示，现在改成作为音频输出声音还是静音的指示。isEnd，作为时序中是否接收到键盘发送的 F0（松开一个按键的信号）的指示。为实现高级功能中的和声，需要额外添加一个 pastdata，锁存当前接收到的有效且非断码的键码的上一个有效且非断码的键码，这样就可以知道和声中最先被按下的键的键码（之后被按下的那个键存在 nowdata 里）。为了能知道什么时候按下了两个键以输出和声，增添指示位 past，表示处理当前键码的时候一直有一个键已经被按下，添加指示位 dual，表示当前已经有两个键按下，可以输出和声。因此，新添加的标志位的处理相当于有一个键按下，那么 past 条件满足，已经存在一个键；再按下一个键，在满足 past 的条件下升级为满足 dual，即可以输出和声。对于实验八的状态做一些改动，得到下面本次实验的状态设计（不考虑同时按两个键以上的键，即如果当前两个键按下，如果之后状态发生了改变，那么一定是键被松开）：

状态名	off	isEnd	past	dual	状态解释
A	1	1	0	0	按键被松开接收到断码中的 F0
B	1	0	0	0	按键被松开接收到断码

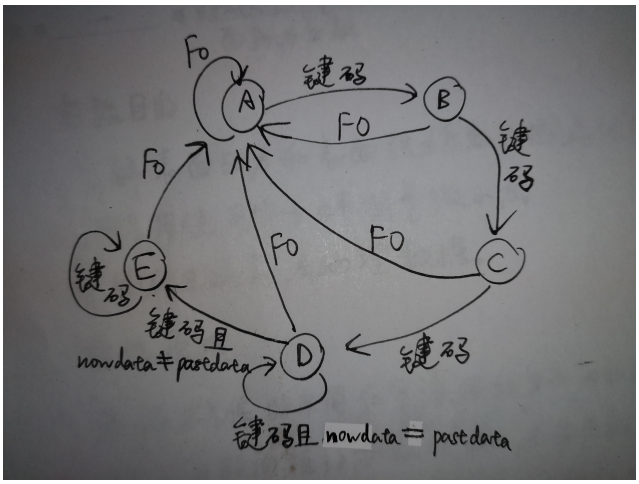
					中的键码
C	0	0	0	0	按键按下的瞬间且此时没有其它键一直被按下（即现在只有一个键被按下）
D	0	0	1	0	当前一直有一个键被按下
E	0	0	1	1	有两个键被按下

其中满足 off 为 0 的时候 isEnd 一定为 0；past 为 0 的时候 dual 一定为 0。当前接受到的数据只能是以下两种情况：1，F0；2，键码。

状态转移表：

	当前接受数据	
状态	F0	键码
A	A	B
B	A	C（nowdata）
C	A	D（pastdata）
D	A	D/E
E	A	E

其中在状态 D 接收到键码的时候可能会转移到两种不同的状态（状态转移表中表红部分）。这取决于当前 nowdata 是否与 pastdata 相等。如果相等，则转移到状态 D，说明此时接收到的还是这个键码（相当于一个键一直按下）；如果不相等，则转移到状态 E，说明在一个键按下的条件下又有一个不同的键按下了，即此时按下了两个键。括号中的操作为附加操作。nowdata 表示此时要将 nowdata 值赋为 buffer[8:1]（有效键码内容）；pastdata 表示此时要将 pastdata 值赋为 buffer[8:1]。下面是状态转移图：



根据上面的分析并参考实验八中键盘处理的写法，就可以写出用于此工程的键盘处理模块：

```
module newps2board(reset,clk,ps2_clk,ps2_data,offOut,nowdataOut,pastdataOut,dualOut);
input clk,ps2_clk,ps2_data;
input reset;
reg [7:0] nowdata;
reg [7:0] pastdata;
reg [9:0] buffer;
reg [3:0] count;
reg [2:0] ps2_clk_sync;
reg off;
reg isEnd;
reg past;
reg dual;
output [7:0] nowdataOut;
output [7:0] pastdataOut;
output offOut;
output dualOut;
initial
begin
    off=1;
    count=0;
    nowdata=0;
    pastdata=0;
    past=0;
    dual=0;
end
always @ (posedge clk)
begin
    ps2_clk_sync <= {ps2_clk_sync[1:0],ps2_clk};
end
wire sampling = ps2_clk_sync[2] & ~ps2_clk_sync[1];
always @ (posedge clk)
begin
    if(reset)
    begin
        count<=0;
    end
    else if(sampling)
    begin
        if(count == 4'd10)
        begin
            if((buffer[0] == 0) && (ps2_data) && (^buffer[9:1]))
                count <= 0;
        end
        else
        begin
            buffer[count] <= ps2_data;
            count <= count + 3'b1;
        end
    end
end
```

```
 51  always @ (posedge clk)
 52 begin
 53     if(reset)
 54     begin
 55        off<=1;
 56        nowdata<=0;
 57        pastdata<=0;
 58        past<=0;
 59        dual<=0;
 60     end
 61     else if((count==4'd10) && (sampling) && (buffer[0]==0) && (ps2_data) && (^buffer[9:1]))
 62     begin
 63        if(buffer[8:1]==8'b11110000)
 64        begin
 65           isEnd<=1;
 66           off<=1;
 67           past<=0;
 68           dual<=0;
 69        end
 70        else
 71        begin
 72           if(isEnd==0)
 73           begin
 74              off<=0;
 75              nowdata<=buffer[8:1];
 76              if(past==0&&dual==0)
 77              begin
 78                 pastdata<=buffer[8:1];
 79                 past<=1;
 80              end
 81              else if(past==1&&dual==0)
 82              begin
 83                 if(pastdata!=buffer[8:1])
 84                 begin
 85                    dual<=1;
 86                 end
 87              end
 88           end
 89           else
 90           begin
 91              isEnd<=0;
 92              past<=0;
 93              dual<=0;
 94           end
 95        end
 96     end
 97 end
 98 assign nowdataOut=nowdata;
 99 assign pastdataOut=pastdataOut;
100 assign dualOut=dual;
101 assign offOut=off;
102 endmodule
```

其中 nowdataOut、pastdataOut 分别是 nowdata、pastdata 与其他模块的输出接口，dualOut、offOut 分别是 dual、off 与其他模块的输出接口。

拿到了键码数据之后，就要将其转化为音频频率。本次实验中，选择不按 shift 情况下分别为 1，2，3，4，5，6，7，8，9，0，-，=这十二个键作为可以输出声音的键（即相关键，其余为无关键），分别对应钢琴音高频率中第五个八度的从 C 到 B 的全部十二个音。每一个键对应一个频率的音。在模块 newfrequency 中，将传入的键码数据转化为对应的频率。基本思路是，对于送进来的 nowdata 和 pastdata，分别各进行一次识别，对应频率表查到对应的频率 f，之后利用公式 f*65536/48000 并进行取整（四舍五入）。之后进行判断，如果 dual 指示（输出和声）有效且 off（静音）无效，那么将两次识别得到的结果相加除以二，即求得两个发音频率的中间频率作为和声的频率；如果 dual 无效且 off 无效，那么就直接输出 nowdata 转化出的结果；如果 off 有效，说明此时停止发音，那么将频率置为全零，最后输出的结果也是零：

```
module newfrequency(nowdata,pastdata,dual,off,outfreq);
input [7:0]nowdata;
input [7:0]pastdata;
input dual;
input off;
reg [15:0]freq;
reg [15:0]pfreq;
reg [15:0]finalfreq;
output [15:0]outfreq;
initial
begin
    finalfreq=16'd0;
end
always @ (*)
begin
    if(off==1)
        finalfreq=16'd0;
    else
    begin
        case(nowdata)
            8'h16:freq=16'd357;
            8'h1E:freq=16'd378;
            8'h26:freq=16'd401;
            8'h25:freq=16'd425;
            8'h2E:freq=16'd450;
            8'h36:freq=16'd477;
            8'h3D:freq=16'd505;
            8'h3E:freq=16'd535;
            8'h46:freq=16'd567;
            8'h45:freq=16'd601;
            8'h4E:freq=16'd636;
            8'h55:freq=16'd674;
            default:freq=16'd0;
        endcase
        case(pastdata)
            8'h16:pfreq=16'd357;
            8'h1E:pfreq=16'd378;
            8'h26:pfreq=16'd401;
            8'h25:pfreq=16'd425;
            8'h2E:pfreq=16'd450;
            8'h36:pfreq=16'd477;
            8'h3D:pfreq=16'd505;
            8'h3E:pfreq=16'd535;
            8'h46:pfreq=16'd567;
            8'h45:pfreq=16'd601;
            8'h4E:pfreq=16'd636;
            8'h55:pfreq=16'd674;
            default:pfreq=16'd0;
        endcase
        if(dual)
            finalfreq=(freq+pfreq)/2;
        else
            finalfreq=freq;
    end
end
assign outfreq=finalfreq;
endmodule
```

然后可以添加顶层文件 sound_sample（这个.v 文件在工程文件夹里，但是不在工程界面里，因此需要先在工程中添加这个文件），在原有的变量基础上新增变量：signal，对应键盘处理的 nowdata；psignal，对应键盘处理的 pastdata；off，静音指示；dual，和声指示；sound_data，键码转频率处理后输出的 16 位结果，是要传入 Sin_Generator 的关键数据。通过在顶层实体文件中调用 newps2board 获取 signal、psignal、off、dual，再传入 newfrequency，获取 sound_data，传入 Sin_Generator，就实现了可以根据键盘按键情况而改变发出声音的功能：

```
wire [15:0] sound_data;//the sound data
wire [7:0] signal;//signal from the board
wire off;//make sound or not
reg [15:0] out_sound;//the out sound data
wire dual;//harmony sound
wire [7:0] psignal;//past data
```

实现按键发声之后，考虑调音的功能。发出的声音的数据其实就是一个三角函数，这个三角函数是有振幅的。由数学原理可知，降低振幅（压缩纵坐标而不

改变横坐标)可以改变声音的大小,但是不会改变音色。因此可以通过降低振幅,压缩三角函数的高度的方式来降低音量。观察基础架构中的内容,可以发现最终要传到芯片上的数据是在模块 Sin_Generator 中的 audiodata,audiodata 传到了模块 I2S_Audio 中以对接芯片。因此可以考虑修改 audiodata 的值,再将修改后的结果作为最终的信息传入 I2S_Audio 中以对接芯片。这里的修改即改变其振幅,就是进行有符号的除法,即算术右移。在顶层实体总添加变量 out_sound,作为最终信息传入 I2S_Audio 中以对接芯片。可以在顶层实体下面单独起一个 always 模块,以开发板系统 50MHz 时钟作为激励,通过开发板上的是个拨动开关(对应输入 SW[9:0])作为调音的根据。自定义如果开关拨为“1000000000”,那么是最大声音(这里最大声音即为在 I2C_Audio_Config 中 initial 语句配置的起始声音大小,对应芯片手册上说明的左右声道最大声音);如果开关拨为“0100000000”,那么是最大声音的振幅除以 2,即将 audiodata 算术右移一位;如果开关拨为“0010000000”,那么是最大声音的振幅除以 4,即将 audiodata 算术右移两位,依此类推,直到最后拨为“0000000001”,直接静音,即设 out_sound 为全 0。其余情况,音量不变。这里有两点需要注意,一是不能直接改变 audiodata 的值,而是定义了一个中间变量,用中间变量去计算并存储改变 audiodata 后的值;二是算术右移时要将 audiodata 强制类型转换为有符号(signed)类型,之后用“>>>”符号表示算数右移:

```
always @ (posedge CLOCK_50)
begin
    if(reset)
        out_sound <= 16'd0;
    else
    begin
        case(SW[9:0])
            10'b1111111111:out_sound <= audiodata[15:0];
            10'b0111111111:out_sound <= $signed(audiodata[15:0]) >>> 1;
            10'b0011111111:out_sound <= $signed(audiodata[15:0]) >>> 2;
            10'b0001111111:out_sound <= $signed(audiodata[15:0]) >>> 3;
            10'b0000111111:out_sound <= $signed(audiodata[15:0]) >>> 4;
            10'b0000011111:out_sound <= $signed(audiodata[15:0]) >>> 5;
            10'b0000001111:out_sound <= $signed(audiodata[15:0]) >>> 6;
            10'b0000000111:out_sound <= $signed(audiodata[15:0]) >>> 7;
            10'b0000000011:out_sound <= $signed(audiodata[15:0]) >>> 8;
            10'b0000000001:out_sound <= 16'd0;
            default:out_sound <= out_sound;
        endcase
    end
end
```

最后,修改顶层文件里函数调用,增添对于键盘处理的两个模块函数,将 sound_data 作为参数传入 sin_wave,将 out_sound 作为参数传入 myaudio:

```
//I2C part
clkgen #(10000) my_i2c_clk(CLOCK_50,reset,1'b1,clk_i2c);  //10k I2C clock

I2C_Audio_Config myconfig(clk_i2c, KEY[0],FPGA_I2C_SCLK,FPGA_I2C_SDAT,LEDR[2:0]);

I2S_Audio myaudio(AUD_XCK, KEY[0], AUD_BCLK, AUD_DACDAT, AUD_DACLRCK, out_sound);

Sin_Generator sin_wave(AUD_DACLRCK, KEY[0], sound_data, audiodata);
```

```
109 newps2board board(
110     .clk(CLOCK_50),
111     .reset(reset),
112     .ps2_clk(PS2_CLK),
113     .ps2_data(PS2_DAT),
114     .nowdataOut(signal),
115     .pastdataOut(psignal),
116     .offOut(off),
117     .dualOut(dual)
118 );
119
120 newfrequency frequent(
121     .nowdata(signal),
122     .pastdata(psignal),
123     .off(off),
124     .dual(dual),
125     .outfreq(sound_data)
126 );
```

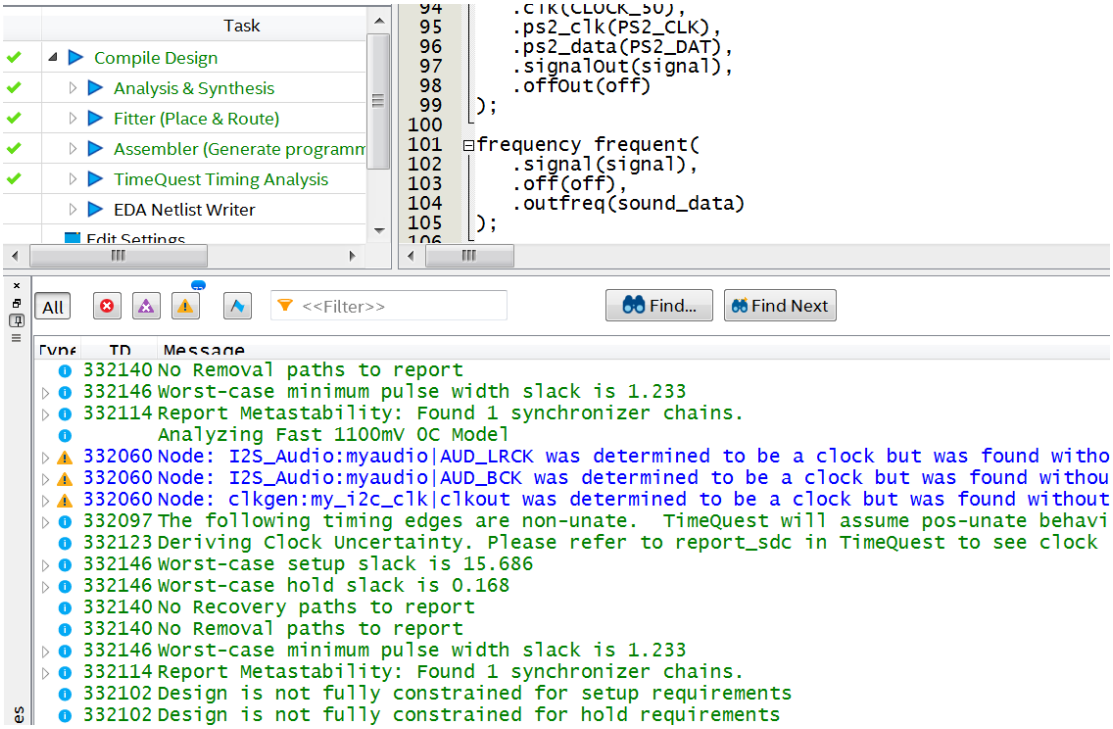
5、实验流程

下载基础架构已经给出的工程 sound_sample，将顶层文件 sound_sample.v 添加到工程里。

按照上面提到的代码流程编写各个模块的代码，并编译通过。

该工程已经给出所有的引脚分配。

生成二进制文件 sound_sample.sof，之后将二进制文件写入开发板，验证功能：



6、测试方法

开发板接耳机和键盘直接验证。

7、实验结果

插入耳机，将开关拨至“1000000000”（最大音量），按下‘1’～‘=’键，

发出各自的声音，长按一直发音，松开停止发音。按其他键不发音。

同时按两个有效按键，发出与两个按键单独按下不同的声音（和声）。

拨动开关可以调节音量。

上述功能均得到验证。

8、实验中遇到的问题及解决办法

一开始在设计调节音量的时候，方案是在 I2C_Audio_Config 中的 mi2c_state 状态机中对于 mi2c_data 赋值的时候先改变 audio_cmd[3]和 audio_cmd[4]（存放左右声道音量信息）的值，之后再赋值的操作。具体实现如下：

```
always @ (posedge clk_i2c or negedge reset_n)
begin
    if(!reset_n)
    begin
       cmd_count  <= 4'b0;
       mi2c_state <= 4'b0;
       mi2c_go    <= 1'b0;
    end
    else
    begin
       case(mi2c_state)
       2'd0: begin  //stop
                  if(cmd_count ==4'b0)
                     mi2c_state <= 2'd1;
             end
       2'd1: begin
                  //implement
                  if(switch==10'b1000000000)
                  begin
                     audio_cmd[3]<=9'd121;
                     audio_cmd[4]<=9'd121;
                  end
                  else if(switch==10'b0100000000)
                  begin
                     audio_cmd[3]<=9'd113;
                     audio_cmd[4]<=9'd113;
                  end
                  else if(switch==10'b0010000000)
                  begin
                     audio_cmd[3]<=9'd105;
                     audio_cmd[4]<=9'd105;
                  end
                  else if(switch==10'b0001000000)
                  begin
                     audio_cmd[3]<=9'd97;
                     audio_cmd[4]<=9'd97;
                  end
                  else if(switch==10'b0000100000)
                  begin
                     audio_cmd[3]<=9'd89;
                     audio_cmd[4]<=9'd89;
                  end
```

```
                else if(switch==10'b0000010000)
                begin
                   audio_cmd[3]<=9'd81;
                   audio_cmd[4]<=9'd81;
                end
                else if(switch==10'b0000001000)
                begin
                   audio_cmd[3]<=9'd73;
                   audio_cmd[4]<=9'd73;
                end
                else if(switch==10'b0000000100)
                begin
                   audio_cmd[3]<=9'd65;
                   audio_cmd[4]<=9'd65;
                end
                else if(switch==10'b0000000010)
                begin
                   audio_cmd[3]<=9'd57;
                   audio_cmd[4]<=9'd57;
                end
                else if(switch==10'b0000000001)
                begin
                   audio_cmd[3]<=9'd49;
                   audio_cmd[4]<=9'd49;
                end
                else if(switch==10'b0000000000)
                begin
                   audio_cmd[3]<=9'd0;
                   audio_cmd[4]<=9'd0;
                end
                else
                begin
                   audio_cmd[3]<=audio_cmd[3];
                   audio_cmd[4]<=audio_cmd[4];
                end
                //end
                mi2c_data <= {audio_addr, audio_reg[cmd_count], audio_cmd[cmd_count]};
                mi2c_go   <= 1'b1;
                mi2c_state<= 2'd2;
            end
```

里面对音量的配置完全按照音频芯片手册上的规定。后来在实际验证的时候不论怎么操作，都不能发出任何声音。之后才改为最终的压缩振幅的方法，才得以解决问题。

9、实验中得到的启示

这次实验可以说相当于一个二次开发，更是要求我们会运用之前的一些素材去类比运用，比如将实验八的键盘ASCII码显示功能与这里的键盘发声功能类比，就可以更好的理解实验的流程。因此在有一些架构的基础上再进行进一步的开发的能力以及运用之前学过的知识，将它们灵活变通得以运用于当前问题的能力是本次实验所要重点考察的，也是我们要加强的地方。

10、意见和建议

暂时还没有。