

# 数字电路与数字系统实验

## 实验十 音频输出

姓名:

学号:

班级:

邮箱:

实验时间: 2020. 11. 23

## 一、实验目的

学习音频信号的输出方式以及如何将数字信号转换为模拟信号的基本原理。将键盘与本实验的音频输出结合，实现一个简单的键盘电子琴功能，可调节音量，支持多个键同时按下的和声。

## 二、实验原理

假设我们需要产生一个人耳能听到的单频率  $f = 960\text{Hz}$  的正弦波信号，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： $s(t) = \sin(2\pi ft)$ 。当采样率是  $48\text{kHz}$  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。这样就变成： $s(n) = \sin(2\pi fn/48000)$ ，代入  $f = 960\text{Hz}$ ，我们得到  $s(n) = \sin(2\pi \times 960n/48000) = \sin(2\pi n/50)$ 。所以，对于整数  $n$  来说，每  $50$  个点对应正弦波的一个周期。

上例中的正弦波是固定频率为  $960\text{Hz}$  的。实际应用中，我们如果要产生不同频率的正弦波，就不能采用简单计数直接查表的方式，而需要先按频率计算出样本点对应的相位，然后查三角函数表获取对应幅度值的方式。

首先我们需要存储器中存储一张  $1024$  点的  $\sin$  函数表。即存储器中以地址  $k = 0 \dots 1023$  存储了  $1024$  个三角函数值（以  $16\text{bit}$  补码整数表示），地址为  $k$  的数值设置为

$$\text{round}\left(\sin\left(\frac{2\pi k}{1024}\right) \times 32767\right) \quad (10-3)$$

## 三、实验环境

Quartus 18.1、FPGA 开发板

## 四、实验过程

设计思路：

模仿提供的 `sound_sample`，添加自己的键盘模块即可，修改 `I2C_Audio_Config.v` 中的信息即可调节音量。

设计代码：

`audio.v`

```

58 wire [15:0] audiodata;
59 wire [15:0] freq;
60
61
62
63 //=====
64 // structural coding
65 //=====
66
67 assign reset = ~KEY[0];
68
69 audio_clk u1(CLOCK_50, reset,AUD_XCK, LEDR[9]);
70
71
72 //I2C part
73 clkgen #(10000) my_i2c_clk(CLOCK_50,reset,1'b1,clk_i2c); //10k I2C clock
74
75
76 I2C_Audio_Config myconfig(clk_i2c, KEY[0],FPGA_I2C_SCLK,FPGA_I2C_SDAT,LEDR[2:0],
77 SW[1:0]);
78
79 I2S_Audio myaudio(AUD_XCK, KEY[0], AUD_BCLK, AUD_DACDAT, AUD_DACLCK, audiodata);
80
81 Sin_Generator sin_wave(AUD_DACLCK, KEY[0], freq, audiodata);//
82
83 keyboard k(CLOCK_50, 1'b1, PS2_CLK, PS2_DAT, HEX0, HEX1, HEX2, HEX3, HEX4, HEX5,
84 freq);
85
86
87 endmodule

```

keyboard.v

```

37 ps2_keyboard P(.clk(clk), .clrn(clrn), .ps2_clk(ps2_clk), .ps2_data(ps2_data),
38 .data(data), .ready(ready), .nextdata_n(nextdata_n), .overflow(overflow));
39
40 show S0(.flag(flag1), .data(outdata[3:0]), .hex(hex0));
41 show S1(.flag(flag1), .data(outdata[7:4]), .hex(hex1));
42 show S2(.flag(flag3), .data(outdata2[3:0]), .hex(hex2));
43 show S3(.flag(flag3), .data(outdata2[7:4]), .hex(hex3));
44 show S4(.flag(flag4), .data(outdata3[3:0]), .hex(hex4));
45 show S5(.flag(flag4), .data(outdata3[7:4]), .hex(hex5));
46
47 reg [3:0] state = 4'b0000;
48
49 always @ (posedge clk2)
50 begin
51 if(flag4) freq = freq3/3 + freq2/3 + freq1/3;
52 else if (flag3) freq = freq2/2 + freq1/2;
53 else if (flag1) freq = freq1;
54 else freq = 0;
55 end
56
57 always @ (posedge clk2)
58 begin
59 if (ready)
60 begin
61 if (data == 8'hf0)
62 begin
63 outdata = data;
64 flag2 <= 0;
65 flag1 <= 0;
66 flag3 <= 0;
67 flag4 <= 0;

```

```

67         flag4 <= 0;
68         //freq <= 0;
69     end
70     else if(!flag2)
71     begin
72         outdata <= data;
73         flag2 <= 1;
74         flag1 <= 0;
75         flag3 <= 0;
76         flag4 <= 0;
77     end
78     else if(data != 8'hf0 && flag1 && !flag3 && data!=outdata)
79     begin
80         flag3 <= 1;
81         outdata2 <= data;
82         case(data)
83             8'h1c: freq2 <= 523.25* 65536 / 48000;
84             8'h1b: freq2 <= 587.33* 65536 / 48000;
85             8'h23: freq2 <= 659.26* 65536 / 48000;
86             8'h2b: freq2 <= 698.46* 65536 / 48000;
87             8'h34: freq2 <= 783.99* 65536 / 48000;
88             8'h33: freq2 <= 880* 65536 / 48000;
89             8'h3b: freq2 <= 987.77* 65536 / 48000;
90             8'h42: freq2 <= 1046.5* 65536 / 48000;
91             default: freq2 <= 0;
92         endcase
93         //if(freq1) freq <= freq2/2 + freq1/2;
94     end
95     else if(data != 8'hf0 && flag1 && flag3 && data!=outdata && data!=outdata2)
96     begin
97         flag4 <= 1;
98         outdata3 <= data;
99         case(data)
100             8'h1c: freq3 <= 523.25* 65536 / 48000;
101             8'h1b: freq3 <= 587.33* 65536 / 48000;
102             8'h23: freq3 <= 659.26* 65536 / 48000;
103             8'h2b: freq3 <= 698.46* 65536 / 48000;
104             8'h34: freq3 <= 783.99* 65536 / 48000;
105             8'h33: freq3 <= 880* 65536 / 48000;
106             8'h3b: freq3 <= 987.77* 65536 / 48000;
107             8'h42: freq3 <= 1046.5* 65536 / 48000;
108             default: freq3 <= 0;
109         endcase
110         //if(freq1) freq <= freq2/2 + freq1/2;
111     end
112     else if(data != 8'hf0 && flag2 && !flag3 && !flag4)
113     begin
114         outdata <= data;
115         flag1 <= 1;
116         case(data)
117             8'h1c: freq1 <= 523.25* 65536 / 48000;
118             8'h1b: freq1 <= 587.33* 65536 / 48000;
119             8'h23: freq1 <= 659.26* 65536 / 48000;
120             8'h2b: freq1 <= 698.46* 65536 / 48000;
121             8'h34: freq1 <= 783.99* 65536 / 48000;
122             8'h33: freq1 <= 880* 65536 / 48000;
123             8'h3b: freq1 <= 987.77* 65536 / 48000;
124             8'h42: freq1 <= 1046.5* 65536 / 48000;
125             default: freq1 <= 0;
126         endcase
127     end
128

```

I2C\_Audio\_Config.v

```

44 audio_reg[8] = 0; audio_cmd[8] = 0; //digital path
45
46 audio_cmd2[0] = 0; //reset
47 audio_cmd2[1] = 0; //Disable Power Down
48 audio_cmd2[2] = 0; //Sampling Control
49 audio_cmd2[3] = 0; //Left Volume
50 audio_cmd2[4] = 0; //Right Volume
51 audio_cmd2[5] = 0; //I2S format
52 audio_cmd2[6] = 0; //Active
53 audio_cmd2[7] = 0; //Analog path
54 audio_cmd2[8] = 0; //Digital path
55
56 audio_cmd3[0] = 0; //reset
57 audio_cmd3[1] = 0; //Disable Power Down
58 audio_cmd3[2] = 0; //Sampling Control
59 audio_cmd3[3] = 0; //Left Volume
60 audio_cmd3[4] = 0; //Right Volume
61 audio_cmd3[5] = 0; //I2S format
62 audio_cmd3[6] = 0; //Active
63 audio_cmd3[7] = 0; //Analog path
64 audio_cmd3[8] = 0; //Digital path
65
66
67 audio_cmd4[0] = 0; //reset
68 audio_cmd4[1] = 0; //Disable Power Down
69 audio_cmd4[2] = 0; //Sampling Control
70 audio_cmd4[3] = 0; //Left Volume
71 audio_cmd4[4] = 0; //Right Volume
72 audio_cmd4[5] = 0; //I2S format
73 audio_cmd4[6] = 0; //Active
74
75
108 if (volume == 2'b00) mi2c_data <= {audio_addr, audio_reg[cmd_count], audio_cmd[cmd_count]}
109 else if (volume == 2'b01) mi2c_data <= {audio_addr, audio_reg[cmd_count], audio_cmd2[cmd_count]}
110 else if (volume == 2'b10) mi2c_data <= {audio_addr, audio_reg[cmd_count], audio_cmd3[cmd_count]}
111 else mi2c_data <= {audio_addr, audio_reg[cmd_count], audio_cmd4[cmd_count]};
112 //mi2c_data <= {audio_addr, audio_reg[cmd_count], audio_cmd[cmd_count]};
113 mi2c_go <= 1'b1;
114

```

ModelSim 仿真：


没有进行这一步，直接在开发板上进行调试

实验结果：

音频也没法拍照啊，已经验收过了…

## 五、 实验中遇到的问题及解决办法

1、一开始调音量的时候没有注意实验手册上的提示，导致没有声音。

 在试图通过修改 `audio_cmd` 中的内容来调节音量时请注意，`audio_cmd` 是一块 RAM，系统只能综合在给定时钟沿和给定地址条件下读取或写入这个 RAM。如果试图在同一周期内读取或写入这个 RAM 中的两个地址的数据，系统将无法综合这块 RAM 的硬件，并且不会报错，直接结果就是编译通过但没有声音。请在设计时注意对 `audio_cmd` 的读取和写入需要按 RAM 的操作规范进行。

2、和声一开始对键盘按键状态的判断没有设计好，导致同时按两个键的时候会接收三个键导致音频信号多变化一次…

## 六、 启示

一定要想好了再写代码，不然容易思路混乱，反复修改，调试要有耐心，尽量每种情况都考虑到…

## 七、 意见与建议

虽然之前我并没有上过数字电路这门课，但是实验手册前面的讲解非常的清楚，由浅入深，帮助我学习和完成了这次实验。