2020-11-23

数字电路实验十 音频输出实验

秦嘉余

191220088

1348288404@qq.com

目录

1.实验目的

本实验的主要目的是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。

2.实验原理

1.生成正弦波

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 件只能以固定的时间间隔产生数字输出，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。
由以下四步生成频率为 f 的正弦波:
1. 根据频率 f 计算递增值 d = f*65536/48000
2. 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
3. 根据 16 位无符号整数计数器的高 10 位来获取查表地址 k，并查找 1024 点 的正弦函数表
4. 使用查表结果作为当前的数字输出

2. 音频接口

DE10-Standard 开发板上集成了一块 WM8731 音频编解码芯片，其参考手册可以在课程网站上找到。该音频编解码芯片提供 24bit 的音频接口，支持 8kHz 到 96kHz 的采样率。

在音频发送的实现上分为两个步骤。
第一步，是配置音频编解码芯片。在数字系统中，大量的设置是通过读写设备上或芯片上的寄存器来 实现的。每个寄存器会有一个独立的地址，并会有自己特定的功能。CPU 或主 控单元通过在寄存器上写入特定的值来将设备设置到特定状态，或者让设备执 行特定的动作。例如，要对音频芯片进行 reset，只需要在地址 0F 的寄存器中写入 00 即可。
第二步是音频流发送。如 10.1 所述，我们的音频流是一串音频的电压数字。我们需要通过 I2S 接口将这串数字传输给音频芯片。传输的具体格式是在 第一步中通过寄存器设置的。一般音频是包括左右两个声道的，我们需要以 48kHz 的速率发送两个声道的数据给音频芯片。在设置正确时，音频芯片就会将对应的数字值转换为耳机的输出，放出声音来。

3.实验环境/器材

本次实验的环境为 Quartus17.1 版本
本次实验的器材为 DE10 Standard 开发板

4. 键盘电子琴功能

1. 设计思路

本次实验中，主要与音频有关的配置输入输出文件都已给出，只需要根据键盘的键码输入设置不同的 freq 参数传入 sin_wave 模块即可。对于键盘输入部分，在实验 8 的基础上重写部分代码即可。拓展部分中，设置音量需要对配置寄存器修改，并重置音频芯片，此时音量参数 volume 也同时由键盘控制，和音部分需要记录当前按下的键，对按下的每个键对应的 freq 取平均值即可。

2. 实验代码

顶层模块如下，增加了 freq_ctrl 模块来控制 freq 参数与 volume 参数，除此之外，也修改了一些参数。例如将按下哪些键码用 LED 灯显示，利用七段显示管显示 freq 与 volume 便于观察。

```
wire clk_i2c;
wire reset;
wire [15:0] audiodata;
wire [15:0] freq;
wire [6:0] volume;
wire [2:0] state;

//=======================================================
//  Structural coding
//=======================================================

assign reset = ~KEY[0];

audio_clk u1(CLOCK2_50, reset,AUD_XCK);


//I2C part
clkgen #(10000) my_i2c_clk(CLOCK2_50,reset,1'b1,clk_i2c);  //10k I2C clock

freq_ctrl i(CLOCK2_50,reset,PS2_CLK,PS2_DAT,volume,freq,HEX2,HEX3,HEX4,HEX5,LEDR[9:2]);

I2C_Audio_Config myconfig(clk_i2c, KEY[0],FPGA_I2C_SCLK,FPGA_I2C_SDAT,state,volume,HEX0,HEX1);

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

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

以下为控制模块的基本初始化设置与参数设置，其中 rom 存储 8 种音对于的 freq 值，count 记录当前按下的按键数量，tag 为 8 位标记位，tag[i]为 1 代表 i 号音对应的按键按下，否则未按下，具体逻辑代码见压缩包对应文件

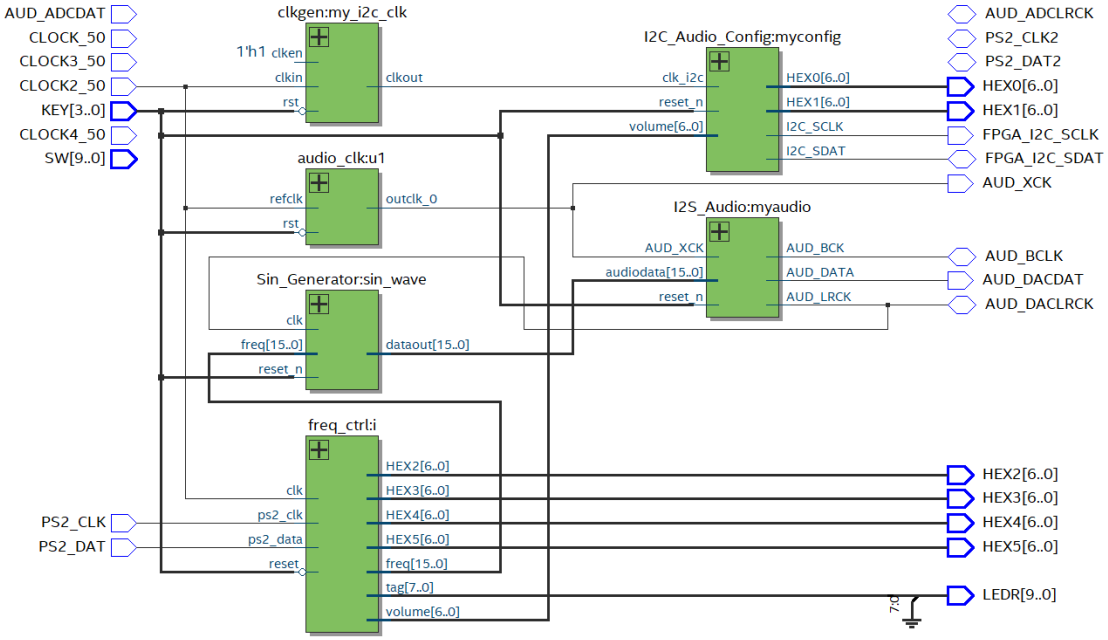
```
module freq_ctrl(clk,reset,ps2_clk,ps2_data,volume,freq,HEX2,HEX3,HEX4,HEX5,tag);
   input clk,reset;
   input ps2_clk,ps2_data;
   output reg [6:0] HEX2,HEX3,HEX4,HEX5;
   output reg [6:0] volume;
   output reg [15:0] freq;
   wire [7:0] kbdata;
   wire [7:0] data;
   wire ready;
   reg nextdata_n;
   reg break;
   reg [15:0] rom [7:0];
   reg [2:0] count;

   output reg [7:0] tag;      //state


   initial begin
      rom[0]=16'd714;
      rom[1]=16'd801;
      rom[2]=16'd900;
      rom[3]=16'd953;
      rom[4]=16'd1070;
      rom[5]=16'd1201;
      rom[6]=16'd1348;
      rom[7]=16'd1428;
      count<=0;
      volume<=7'h79;
      freq<=16'h0;
      break<=1'b0;
      tag<=8'b0;
   end

   ps2_keyboard i(.clk(clk),.clrn(1'b1),.ps2_clk(ps2_clk),.ps2_data(ps2_data),.data(data),.nextdata_n(1'b0),.ready(ready));
```

3.RTL 视图



4.引脚分配

该实验为 system 软件自动分配的引脚，所以不再展示、

5.实验结果

本实验已在开发板上验收

5.拓展功能

1.修改音量

在 Audio_Config 模块中，修改如下：

```
always @ (posedge clk_i2c or negedge reset_n)
begin
    if(!reset_n)
    begin
        audio_reg[0]= 7'h0f; audio_cmd[0]=9'h0;   //reset
        audio_reg[1]= 7'h06; audio_cmd[1]=9'h0;   //Disable Power Down
        audio_reg[2]= 7'h08; audio_cmd[2]=9'h2;   //Sampling Control
        audio_reg[3]= 7'h02; audio_cmd[3]={2'b11,volume}; //Left Volume
        audio_reg[4]= 7'h03; audio_cmd[4]={2'b11,volume}; //Right Volume
        audio_reg[5]= 7'h07; audio_cmd[5]=9'h1;   //I2S format
        audio_reg[6]= 7'h09; audio_cmd[6]=9'h1;   //Active
        audio_reg[7]= 7'h04; audio_cmd[7]=9'h16; //Analog path
        audio_reg[8]= 7'h05; audio_cmd[8]=9'h06;  //Digital path
       cmd_count  <= 4'b0;
       mi2c_state <= 4'b0;
       mi2c_go    <= 1'b0;
    end
```

查阅手册得知，audio_cmd 中低 7 位为音量，最高位需要设置为 1 才可以修改音量，所以在命令执行的时序逻辑中修改寄存器，按下 reser 后完成音量修改，volume 的修改同样在控制模块中，如下：

```
8'h72:
begin
    if(volume==7'h0)
        volume<=volume;
    else
        volume<=volume-7'h1;
    tag<=tag;
end
8'h75:
begin
    if(volume==7'h7f)
        volume<=volume;
    else
        volume<=volume+7'h1;
    tag<=tag;
end
default: tag<=tag;
```

2.和音

和音部分的实现首先利用一个标识变量 tag，记录 8 个音的按下与否，tag[i]为 1 代表 i 号音按下，否则未按下。count 记录当前按下的按键数。这两个变量在键盘按键按下或者松开时修改，在上述图片中已经展示，下面为根据 count 和 tag 计算 freq 的部分

```
always @(tag)
begin
    if(tag==0)
        freq<=0;
    else
    begin
        freq<=((tag[0]==1?rom[0]:0)+(tag[1]==1?rom[1]:0)+(tag[2]==1?rom[2]:0)+(tag[3]==1?rom[3]:0)
        +(tag[4]==1?rom[4]:0)+(tag[5]==1?rom[5]:0)+(tag[6]==1?rom[6]:0)
        +(tag[7]==1?rom[7]:0))/count;
    end
end
```

对应按键按下取当前按键对于的 freq，否则取 0，最后取平均数即可

6.实验实验中遇到的问题及解决办法

本次实验的难点在于对音频芯片的理解以及对于键盘代码的复用与重写。在实验中遇到了以下一些问题：

1.

最开始，将 freq 修改为固定之后，耳机连接音频始终没有声音，后来发现是当前开发板的音频芯片损坏，更换开发板测试后正常。由此见得，实验前的检测准备工作尤为重要，否则在功能复杂后再在开发板验证将难以排查出问题

2.

修改音量部分始终无法生效，在查阅手册后，发现，对于 9 位的 audio_cmd，其中低 7 位为数据位，最好位为 0 时，使用默认设置，为 1 时，才会采用后面 7 位作为音量来设置

3.

计算和音的 freq 时，始终无法正确的设置，freq 更改过快或累计过大，在 7 段显示管上无法正常显示，由此判断是时序逻辑发生了问题，所以将 count，tag 的改变放在键码的判断的 always 语句块中，使得键码按下与 count，tag 的改变同步，此时 counr 与 tag 都改变正确。freq 的计算也需要时钟同步，直接循环计算由于硬件电路的限制，难以办到先对 freq 清零再累加取平均这样的操作次序，导致 freq 异常。后来使用一个较长的表达式直接计算 freq 即可使得 freq 正确，这在硬件电路上就不会因为时序的关系导致逻辑不同步，更加符合硬件语言的设计。