数电实验EXP10

刘永鹏 191220070 693901492@qq.com 12.1

1. 实验内容

基本要求

1. 实现至少 8 个音符，建议可以实现 C5，D5，. . .，B5，C6 这些音符。
2. 只需要支持每次按下单个按键。按键按下后开始发音，按键期间持续发音，松开后停止发音。按无关键不发音。
3. 无杂音和爆破音等干扰。

可选扩展要求

1. 可调节音量。
2. 支持多个键同时按下的和声。例如，同时按下 C5，E5，G5 时发出大三和 弦，即对应三个音相加的结果，可以只支持同时发两个音。

2. 实验原理

1. **音频输出原理：**
    1. 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。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。
    2. 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. . .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

$$round(sin(2\pi k1024) \times 32767)$$

    3. 但是，在 FPGA 中要计算乘除法及取整操作耗费资源较多，我们实际应用中采取累加的方法。因此，生成频率为 f 的正弦波的过程如下：
        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**
    1.

Signal Name	*FPGA Pin No.*	*Description*	*I/O Standard*
AUD_ADCLRCK	PIN_AH29	Audio CODEC ADC LR Clock	3.3V
AUD_ADCDAT	PIN_AJ29	Audio CODEC ADC Data	3.3V
AUD_DACLRCK	PIN_AG30	Audio CODEC DAC LR Clock	3.3V
AUD_DACDAT	PIN_AF29	Audio CODEC DAC Data	3.3V
AUD_XCK	PIN_AH30	Audio CODEC Chip Clock	3.3V
AUD_BCLK	PIN_AF30	Audio CODEC Bit-stream Clock	3.3V
I2C_SCLK	PIN_Y24 or PIN_E23	I2C Clock	3.3V
I2C_SDAT	PIN_Y23 or PIN_C24	I2C Data	3.3V

2. **I²C接口**

    1. 我们利用 I 2C 来设置音频芯片。
    2. 本实验中，我们主要通过更改九个寄存器中的值来操作音频芯片。

3. 实验代码/实验截图

基础功能：

首先，我们用note.txt初始化了从C5到C6的24个音符。

为实现基础功能，我们只需要在按键事件发生时，从内存取出对应的频率即可。

```
always @ (posedge clk)
begin
    test[4:0] <= freq[4:0];
    test[9:6] <= data[4:0];
    case(data)
            //low
            8'h15: begin freq /*harmony[write]*/ <= notes[0]; end//C
            8'h1e: begin freq /*harmony[write]*/ <= notes[1]; end//C#
            8'h1d: begin freq /*harmony[write]*/ <= notes[2]; end//D
            8'h26: begin freq /*harmony[write]*/ <= notes[3]; end//D#
            8'h24: begin freq /*harmony[write]*/ <= notes[4]; end//E
            8'h2d: begin freq /*harmony[write]*/ <= notes[5]; end//F
            8'h2e: begin freq /*harmony[write]*/ <= notes[6]; end//F#
            8'h2c: begin freq /*harmony[write]*/ <= notes[7]; end//G
            8'h36: begin freq /*harmony[write]*/ <= notes[8]; end//G#
            8'h35: begin freq /*harmony[write]*/ <= notes[9]; end//A
            8'h3D: begin freq /*harmony[write]*/ <= notes[10]; end//A#
            8'h3c: begin freq /*harmony[write]*/ <= notes[11];end//B

            //high
            8'h2a: begin freq /*harmony[write]*/ <= notes[12]; end//C
            8'h34: begin freq /*harmony[write]*/ <= notes[13]; end//C#
            8'h32: begin freq /*harmony[write]*/ <= notes[14]; end//D
            8'h33: begin freq /*harmony[write]*/ <= notes[15]; end//D#
            8'h31: begin freq /*harmony[write]*/ <= notes[16]; end//E
            8'h3a: begin freq /*harmony[write]*/ <= notes[17]; end//F
            8'h42: begin freq /*harmony[write]*/ <= notes[18]; end//F#
            8'h41: begin freq /*harmony[write]*/ <= notes[19]; end//G
            8'h4b: begin freq /*harmony[write]*/ <= notes[20]; end//G#
            8'h49: begin freq /*harmony[write]*/ <= notes[21]; end//A
            8'h4c: begin freq /*harmony[write]*/ <= notes[22]; end//A#
            8'h4a: begin freq /*harmony[write]*/ <= notes[23]; end//B

            8'h55: begin                        // +
                if (volume >= 7'b1111111)
                    volume <= 7'b1111111;
                else volume <= volume + 1;
            end
            8'h4e: begin                        // -
                if (volume <= 7'b0110000)
                    volume <= 7'b0110000;
                else volume <= volume - 1;
            end
```

```
            default: freq <= freq;
    endcase

    if(data == 8'h55 || data == 8'h4e)
        en <= 1;
    else en <= 0;
    if (break && data != 0) begin
        freq <= 0;
        break <= 0;
    end
    if (data == 8'hf0) begin
        break <= 1;
        write = write - 1;
    end
end
```

对于键盘的控制，相当于EXP8的简化实现：通过在当前文件中调用键盘控制器模块，来获取键盘当前的按键信息：

```
ps2_keyboard k1(.clk(clk),.clrn(1),.ps2_clk(ps2_clk),
    .ps2_data(ps2_data),.data(data),.nextdata_n(0));
```

以上代码位于piano.v中。

扩展功能：

首先，对于音量控制，从之前对I^2S芯片的分析，我们发现应当通过修改第3号寄存器来修改相应的音量值。此处代码位于I2S_Audoi_config.v中

```
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
```

注意，实验过程中发现必须同步修改九个寄存器才能正确的修改音量。

关于和声，有两种实现方式：

1. **有效位记录**：某一个音是否被按下用一个寄存器记录，遍历整个寄存器组寻找整个值为1（即被按下的键）对应的音，最后做平均处理即可。
2. **读写指针法**：本实验代码即使用这种方法。不单独记录某一个音的信息，而是维护一个读写队列，每按下一个键就往读写队列里写入一个值，处理时遍历整个读写队列。

```
always @ (clk)
begin
    test[5:0] = freq[5:0];
    test[9:6] = write[3:0];

    if(break && data != 0)
    begin
        freq <= 0;
        write <= write;
        break <= 0;
    end
    else if(write > 0 && data == harmony[write - 1])
    begin
```

```
        freq <= freq;
        write <= write;

    end

    else begin
        case(data)

            //low
            8'h15: begin /*freq*/ harmony[write] <= notes[0]; write <= write +
1;end//C
            8'h1e: begin /*freq*/ harmony[write] <= notes[1]; write <= write +
1;end//C#
            8'h1d: begin /*freq*/ harmony[write] <= notes[2]; write <= write +
1;end//D
            8'h26: begin /*freq*/ harmony[write] <= notes[3]; write <= write +
1;end//D#
            8'h24: begin /*freq*/ harmony[write] <= notes[4]; write <= write +
1;end//E
            8'h2d: begin /*freq*/ harmony[write] <= notes[5]; write <= write +
1;end//F
            8'h2e: begin /*freq*/ harmony[write] <= notes[6]; write <= write +
1;end//F#
            8'h2c: begin /*freq*/ harmony[write] <= notes[7]; write <= write +
1;end//G
            8'h36: begin /*freq*/ harmony[write] <= notes[8]; write <= write +
1;end//G#
            8'h35: begin /*freq*/ harmony[write] <= notes[9]; write <= write +
1;end//A
            8'h3D: begin /*freq*/ harmony[write] <= notes[10]; write <= write +
1;end//A#
            8'h3c: begin /*freq*/ harmony[write] <= notes[11]; write <= write +
1;end//B

            //high
            8'h2a: begin /*freq*/ harmony[write] <= notes[12]; write <= write +
1;end//C
            8'h34: begin /*freq*/ harmony[write] <= notes[13]; write <= write +
1;end//C#
            8'h32: begin /*freq*/ harmony[write] <= notes[14]; write <= write +
1;end//D
            8'h33: begin /*freq*/ harmony[write] <= notes[15]; write <= write +
1;end//D#
            8'h31: begin /*freq*/ harmony[write] <= notes[16]; write <= write +
1;end//E
            8'h3a: begin /*freq*/ harmony[write] <= notes[17]; write <= write +
1;end//F
            8'h42: begin /*freq*/ harmony[write] <= notes[18]; write <= write +
1;end//F#
            8'h41: begin /*freq*/ harmony[write] <= notes[19]; write <= write +
1;end//G
            8'h4b: begin /*freq*/ harmony[write] <= notes[20]; write <= write +
1;end//G#
            8'h49: begin /*freq*/ harmony[write] <= notes[21]; write <= write +
1;end//A
            8'h4c: begin /*freq*/ harmony[write] <= notes[22]; write <= write +
1;end//A#
```

```
            8'h4a: begin /*freq*/ harmony[write] <= notes[23]; write <= write +
1;end//B

        8'h55: begin                            // +
            if (volume >= 7'b1111111)
                volume <= 7'b1111111;
            else volume <= volume + 1;
        end
        8'h4e: begin                            // -
            if (volume <= 7'b0110000)
                volume <= 7'b0110000;
            else volume <= volume - 1;
        end
        endcase
        //f0
        if (break && data != 0) begin
            freq <= 0;
            break <= 0;
        end
        if (data == 8'hf0) begin
            if(write > 0)
                write <= write - 1;
            break <= 1;
        end
        //calculate frequency

        if(write == 0)
            freq <= 0;
        else begin
            freq <= harmony[write - 1];

            for(i=0; i<write; i=i+1)
            begin
                sum <= sum + harmony[i];
            end
            sum /= (write+1);
        end

    end
end
```

4. 实验过程

1. 本实验很难仿真调试，因此使用自己的耳机进行调试。
2. 先处理音频，确认音频代码正确后，连键盘综合调试。

5. 实验结果

已完成验收。

6. 问题与解决方案

1. 音量调节耗费大量时间：因为单独修改音量控制寄存器是无效的，这似乎不符合逻辑，但实验结果表明的确如此。

2. 和声时序处理耗费大量时间：音频采样频率高，因此使用循环风险较大，事实上，在前面的和声处理中，使用第一种方案效果更佳，但耗费了更多的寄存器资源。