同济大学计算机系

数字逻辑课程综合实验报告

学　　号 2053051

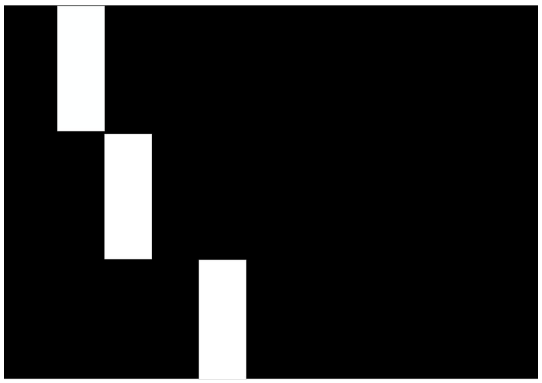
姓　　名 刘越影

专　　业 计算机科学与技术

授课老师 张冬冬

一、实验内容

1.**项目名称**：钢琴块小游戏
2.**界面样式**：如图所示，共有八条轨道，每播放一个音符，在随机一个轨道上会产生一个白色的钢琴块，若玩家需要在钢琴块掉出屏幕之前按下键盘上对应的键，则该钢琴块被消除。此外，本游戏还设计了开始界面和结束界面。

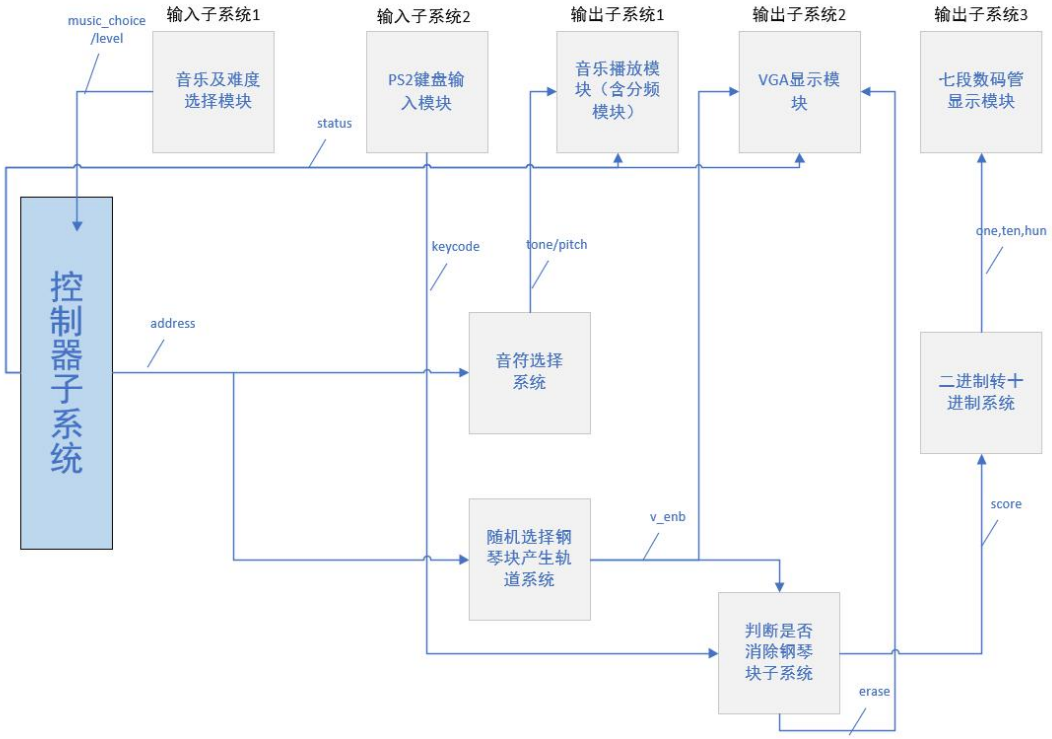


（游戏过程截图）

（游戏开始界面和结束界面）

3.**操作与规则说明**：FPGA 开发板连接 VGA,ps2 键盘和耳机（或音响）。游戏开始前，玩家需要通过开发板上的拨码开关选择要演奏的乐曲（目前有三首可供选择）和游戏模式（简单模式和挑战模式，后者比前者音乐播放速度及钢琴块下落速度更快）。游戏开始，VGA 上根据音乐节奏在随机的轨道上产生钢琴块，玩家需要在钢琴块完全落下之前点击键盘上该轨道对应的按键将其消除，每消除一个钢琴块可计一分，未消除则不得分，开发板上的数码管实时显示当前得分。通过开发板上最右侧的拨码开关可以实现全部数据复位功能。
4.**器件简介**：
①Nexys4 DDR Artix-7——由 Xilinx 公司开发出的一款现场可编程门阵列（FPGA）开发板
②VGA——VGA(Video Graphics Array)是 IBM 在 1987 年随 PS/2 机一起推出的一种视频传输标准，具有分辨率高、显示速率快、颜色丰富等优点；
③Logitech MK120 键盘——使用 PS/2 协议的一款有线键盘。
④索尼耳机——用于输出音频的设备

二、钢琴块小游戏数字系统总框图



本数字系统包含时序逻辑与组合逻辑。

输入子系统 1：音乐及难度选择子模块。由寄存器构成，通过 FPGA 上的拨码开关选择要演奏的曲目以及难度，在演奏过程中可以随时切换。寄存器是时序逻辑。

输入子系统 2：PS2 键盘输入子模块。根据外接 PS2 键盘模块的时钟读取来自键盘的数据储存到寄存器中。时钟、寄存器等是时序逻辑。

输出子系统 1：音乐播放模块，内含时钟分频模块。根据来自音符选择系统的音调与音高信号计算对应的分频系数，将系统 100MHZ 时钟分频作为音频信号输出。寄存器、分频器等是时序逻辑。

输出子系统 2：VGA 显示模块。根据来自随机选择钢琴块产生轨道子系统的信号以及来自控制器子系统行同步信号与场同步信号选择某个时间应该输出颜色的区域。时钟、寄存器等是时序逻辑，选择器等是组合逻辑。

输出子系统 3：七段数码管显示模块。根据经过二进制十进制转换系统转换得到的分数个、十、百位信号点亮数码管。依次点亮不同数码管需要时钟控制，是时序逻辑；而选择要点亮的数码管则包含组合逻辑。

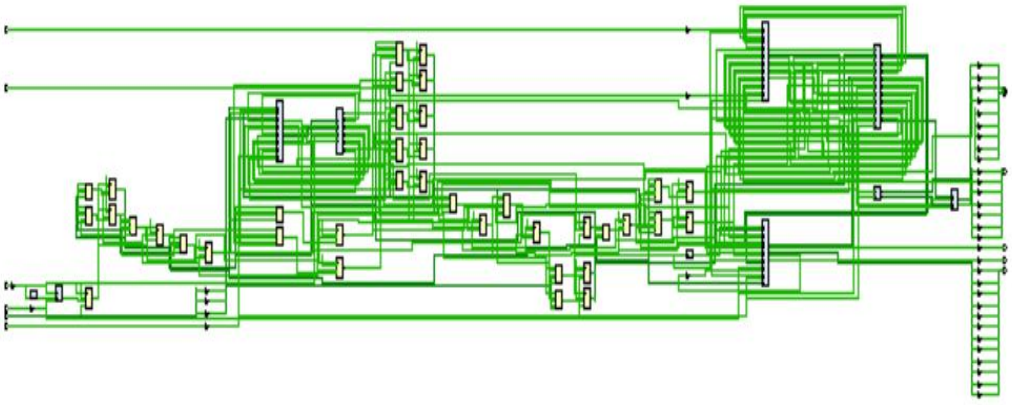
音符选择子系统：根据来自控制器的地址信号选择对应的音调音高，是组合逻辑。

随机选择钢琴块产生轨道：根据当前读取音符的地址选择钢琴块产生轨道，属于组合逻辑。

判断是否消除钢琴块子系统：将产生钢琴块的轨道信息与接收到的按键信息比较判断是否消除钢琴块。包含计数器等时序逻辑与比较器等组合逻辑。

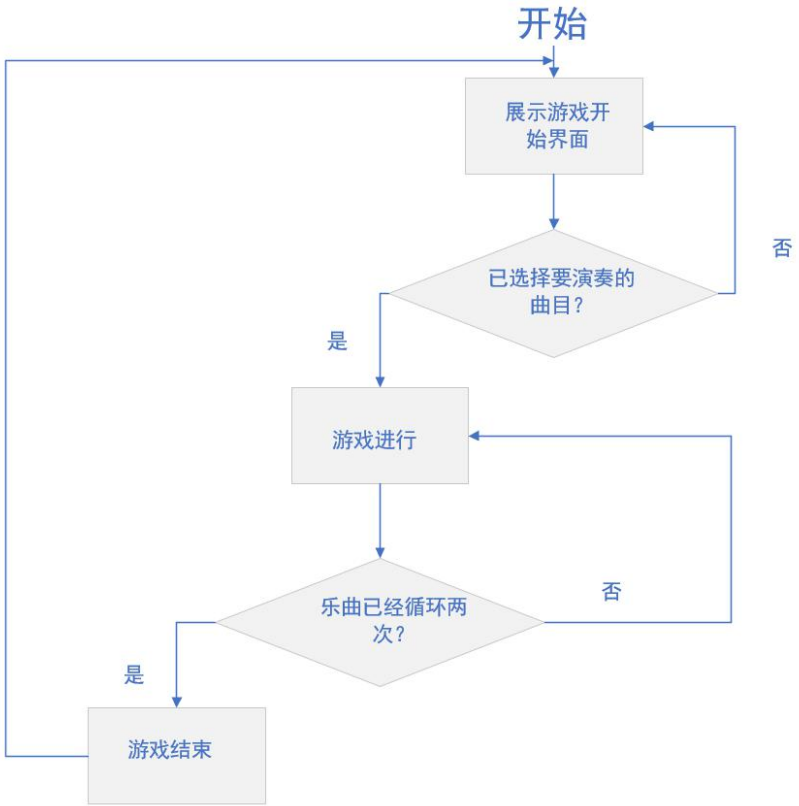
二进制转十进制系统：包含计数器等，属于时序逻辑。

RTL 图片如下方所示：



三、系统控制器设计

1. ASM 流程图



2. 状态转移真值表

现态（PS）	次态（NS）	转换条件
展示游戏开始界面（001）	游戏进行（010）	已选择要演奏的曲目（choice==3’b001||choice==3’b010||choice!=3’b100）
	展示游戏开始界面（001）	未选择要演奏的曲目（choice!=3’b001&&choice!=3’b010&&choice!=3’b100）
游戏进行（010）	游戏结束（100）	曲目循环两次（loop_times==2）
	游戏进行（010）	曲目播放未循环两次（loop_times<2）

3. 次态激励函数表达式

采用“一对一”法设计定序型控制器，设现态为 $Q_2Q_1Q_0$，次态为 $Q_2(D)Q_1(D)Q_0(D)$，设转换条件中“已选择要演奏的曲目”为 A，选择的曲目已经播放两次为 B，则有：

$Q_2(D)=Q_1 \cdot B$
$Q_1(D)=Q_0 \cdot A+Q_1 \cdot \sim B$
$Q_0(D)=Q_0 \cdot \sim A$

4. 控制命令函数表达式

在本系统中，控制器直接将状态传递给 VGA 显示系统和音乐播放系统等，所以有：$status=Q_2Q_1Q_0$

四、子系统模块建模

1. top 模块

1.1 功能描述：定义所有与外部连接的输入输出信号，连接并控制各个子系统。同时包含整个数字系统的控制器部分。

1.2 verilog 代码：

```
module top(
    input clk,//100MHZ 时钟
    input rst_n,//游戏开关
    input [2:0]choice,//乐曲选择
    input level,//难度选择
    input ps2_clk,//键盘时钟
    input ps2_data,//键盘数据
    output speaker,//扬声器
    output [6:0]SEG, //七段数码管显示数据
    output [7:0]AN, //七段数码管选择
    output hys,//行同步
```

```
    output vys,//场同步
    output [11:0]lcd_rgb//[11:8]r,[7:4]g,[3:0]b
);
    wire vga_clk;//vga 专用时钟
    wire [6:0]tone_cur;//正在播放音符的音调
    wire [2:0]pitch_cur;//正在播放音符的音高
    reg [6:0] addr = 0;//选择乐曲的第几个音符
    wire clk_2HZ;//简单模式时钟频率
    wire clk_4HZ;//挑战模式采样频率
    reg clk_chosen;//玩家选择的模式对应时钟频率
    wire [7:0] v_enb;//选择产生方块的轨道
    wire [7:0] erase;//用户是否在钢琴块下落前点击将其消除
    wire [7:0] score;//当前得分
    wire [31:0] keycode;//当前读到的键盘码
    wire [3:0] one;//得分数个位显示
    wire [3:0] ten;//得分数十位显示
    wire [3:0] hun;//得分数百位显示
    reg [1:0]loop_times;//循环播放次数
    reg [3:0] status;//当前状态（001 展示起始界面；010 展示游戏界面；100 展示
结束界面）
    always@(posedge clk)
    begin
        if(choice!=3'b001 && choice!=3'b010 && choice!=3'b100) status <=
3'b001;
        else if(loop_times==2) status<=3'b100;
        else status<=3'b010;
    end

    always@ (posedge clk)
    begin
        if(level==0) clk_chosen<=clk_2HZ;
        else clk_chosen<=clk_4HZ;
    end


    always@(posedge clk_chosen , negedge rst_n)
    begin
        if(!rst_n || status!=3'b010) begin addr<=0; loop_times <= 0; end
        else
        begin
            if(addr==127) begin addr<=0; loop_times<= loop_times+1; end
            else addr<=addr+1;
        end
    end
```

```
    //生成vga专用的25MHZIP核
    vga_ip vga_ip_inst(.clk_in1(clk),.clk_out1(vga_clk));
    //2HZ分频,简单模式
    divider #(50000000/2) d_2hz(.clk(clk),.clk_out(clk_2HZ));
    //4HZ分频,挑战模式
    divider #(25000000/2) d_4hz(.clk(clk),.clk_out(clk_4HZ));
    //随机选择钢琴块出现轨道模块
    block_random
block_random_inst(.addr(addr),.cur_tone(tone_cur),.cur_pitch(pitch_cur),.v_
enb(v_enb));
    //音乐选择模块
    music_choice
music_choice_inst(.choice(choice),.addr(addr),.pitch(pitch_cur),.tone(tone_
cur));
    //播放音乐模块
    music_creator
music_creator_inst(.clk(clk),.rst_n(rst_n),.status(status),.tone(tone_cur),.
pitch(pitch_cur),.speaker(speaker));
    //vga显示模块
    vga
vga_inst(.clk(clk),.vga_clk(vga_clk),.rst_n(rst_n),.status(status),.level(l
evel),.v_enb(v_enb),.erase(erase),.hys(hys),.vys(vys),.lcd_rgb(lcd_rgb));
    //键盘信号接收模块
    ps2_receiver
ps2_receiver_inst(.clk(clk),.kclk(ps2_clk),.kdata(ps2_data),.rst_n(rst_n),.
keycode(keycode));
    //玩家擦除操作控制模块
    erase_controller
erase_controller_inst(.clk(clk),.rst_n(rst_n),.v_enb(v_enb),.keycode(keycod
e),.level(level),.erase(erase),.score(score));
    //将分数转为十进制模块
    bin_dec
bin_dec_inst(.clk(clk),.bin(score),.rst_n(rst_n),.one(one),.ten(ten),.hun(h
un));
    //实时计分显示模块
    score_display
score_display_inst(.score(score),.clk(clk),.rst_n(rst_n),.one(one),.ten(ten)
,.hun(hun),.seg(SEG),.an(AN));
Endmodule
```

2.音乐选择模块

2.1 功能描述：根据控制系统给的地址选择对应乐曲的对应音符，输出该音符的

音调与音高。目前有三首乐曲可供选择，乐曲内容直接列写在代码中。

2.2 verilog 代码：

```
module music_choice(
    input [2:0]choice,//要选择的乐曲(目前共有 3 首乐曲可以选择)
    input [6:0]addr,//要选择的音符地址（每首乐曲不超过 127 个音符）
    output reg [2:0] pitch,//音符的音高
    output reg [6:0] tone//音符的音调
    );
    always@(choice,addr)
    begin
        if(choice==3'b001)//乐曲"白月光与朱砂痣"
        begin
            case(addr)
                1: begin pitch=3'b010; tone=7'b000_0001; end //中音 1
                2: begin pitch<=3'b010; tone<=7'b000_0001; end //延音
                3: begin pitch<=3'b010; tone<=7'b000_0001; end //延音
                4: begin pitch<=3'b010; tone<=7'b000_0001; end //延音
                5: begin pitch<=3'b100; tone<=7'b100_0000; end //低音 7
                6: begin pitch<=3'b100; tone<=7'b100_0000; end //延音
                7: begin pitch<=3'b010; tone<=7'b000_0001; end //中音 1
                8: begin pitch<=3'b010; tone<=7'b000_0001; end //延音
                9: begin pitch<=3'b010; tone<=7'b001_0000; end //中音 5
                10: begin pitch<=3'b010; tone<=7'b001_0000; end //延音
                11: begin pitch<=3'b010; tone<=7'b001_0000; end //中音 5
                12: begin pitch<=3'b010; tone<=7'b010_0000; end //中音 6
                13: begin pitch<=3'b010; tone<=7'b010_0000; end //中音 6
                14: begin pitch<=3'b010; tone<=7'b010_0000; end //延音
                15: begin pitch<=3'b010; tone<=7'b010_0000; end //延音
                16: begin pitch<=3'b010; tone<=7'b010_0000; end //中音 6
                17: begin pitch<=3'b010; tone<=7'b001_0000; end //中音 5
                18: begin pitch<=3'b010; tone<=7'b001_0000; end //延音
                19: begin pitch<=3'b010; tone<=7'b000_0010; end //中音 2
                20: begin pitch<=3'b010; tone<=7'b000_0010; end //延音
                21: begin pitch<=3'b010; tone<=7'b000_0100; end //中音 3
                22: begin pitch<=3'b010; tone<=7'b000_0100; end //延音
                23: begin pitch<=3'b010; tone<=7'b000_1000; end //中音 4
                24: begin pitch<=3'b010; tone<=7'b000_1000; end //延音
                25: begin pitch<=3'b010; tone<=7'b000_0100; end //中音 3
                26: begin pitch<=3'b010; tone<=7'b000_1000; end //中音 4
                27: begin pitch<=3'b010; tone<=7'b000_1000; end //延音
                28: begin pitch<=3'b010; tone<=7'b000_0100; end //中音 3
                29: begin pitch<=3'b010; tone<=7'b000_0100; end //中音 3
                30: begin pitch<=3'b010; tone<=7'b000_0100; end //延音
                31: begin pitch<=3'b010; tone<=7'b000_0100; end //延音
```

```
32: begin pitch<=3'b010; tone<=7'b000_0100; end //延音
33: begin pitch<=3'b010; tone<=7'b000_0001; end //中音1
34: begin pitch<=3'b010; tone<=7'b000_0001; end //延音
35: begin pitch<=3'b010; tone<=7'b000_0001; end //延音
36: begin pitch<=3'b010; tone<=7'b000_0001; end //延音
37: begin pitch<=3'b100; tone<=7'b100_0000; end //低音7
38: begin pitch<=3'b100; tone<=7'b100_0000; end //延音
39: begin pitch<=3'b010; tone<=7'b000_0001; end //中音1
40: begin pitch<=3'b010; tone<=7'b000_0001; end //延音
41: begin pitch<=3'b010; tone<=7'b000_0100; end //中音3
42: begin pitch<=3'b010; tone<=7'b000_0100; end //延音
43: begin pitch<=3'b010; tone<=7'b000_1000; end //中音4
44: begin pitch<=3'b010; tone<=7'b000_1000; end //延音
45: begin pitch<=3'b010; tone<=7'b000_1000; end //中音4
46: begin pitch<=3'b010; tone<=7'b000_1000; end //延音
47: begin pitch<=3'b010; tone<=7'b000_1000; end //延音
48: begin pitch<=3'b100; tone<=7'b010_0000; end //低音6
49: begin pitch<=3'b010; tone<=7'b000_0001; end //中音1
50: begin pitch<=3'b010; tone<=7'b000_0001; end //延音
51: begin pitch<=3'b010; tone<=7'b000_0001; end //中音1
52: begin pitch<=3'b010; tone<=7'b000_0001; end //延音
53: begin pitch<=3'b010; tone<=7'b000_0010; end //中音2
54: begin pitch<=3'b010; tone<=7'b000_0010; end //延音
55: begin pitch<=3'b010; tone<=7'b000_0001; end //中音1
56: begin pitch<=3'b010; tone<=7'b000_0001; end //延音
57: begin pitch<=3'b100; tone<=7'b100_0000; end //低音7
58: begin pitch<=3'b100; tone<=7'b100_0000; end //延音
59: begin pitch<=3'b100; tone<=7'b100_0000; end //延音
60: begin pitch<=3'b100; tone<=7'b100_0000; end //延音
61: begin pitch<=3'b000; tone<=7'b000_0000; end //休止符
62: begin pitch<=3'b000; tone<=7'b000_0000; end //休止符
63: begin pitch<=3'b000; tone<=7'b000_0000; end //休止符
64: begin pitch<=3'b000; tone<=7'b000_0000; end //休止符
65: begin pitch<=3'b010; tone<=7'b000_0001; end //中音1
66: begin pitch<=3'b010; tone<=7'b000_0001; end //延音
67: begin pitch<=3'b010; tone<=7'b000_0001; end //延音
68: begin pitch<=3'b010; tone<=7'b000_0001; end //延音
69: begin pitch<=3'b100; tone<=7'b100_0000; end //低音7
70: begin pitch<=3'b100; tone<=7'b100_0000; end //延音
71: begin pitch<=3'b010; tone<=7'b000_0001; end //中音1
72: begin pitch<=3'b010; tone<=7'b000_0001; end //延音
73: begin pitch<=3'b010; tone<=7'b001_0000; end //中音5
74: begin pitch<=3'b010; tone<=7'b001_0000; end //延音
75: begin pitch<=3'b010; tone<=7'b001_0000; end //中音5
```

```
76: begin pitch<=3'b010; tone<=7'b010_0000; end //中音 6
77: begin pitch<=3'b010; tone<=7'b010_0000; end //中音 6
78: begin pitch<=3'b010; tone<=7'b010_0000; end //延音
79: begin pitch<=3'b010; tone<=7'b010_0000; end //延音
80: begin pitch<=3'b010; tone<=7'b010_0000; end //中音 6
81: begin pitch<=3'b010; tone<=7'b001_0000; end //中音 5
82: begin pitch<=3'b010; tone<=7'b001_0000; end //延音
83: begin pitch<=3'b010; tone<=7'b000_0010; end //中音 2
84: begin pitch<=3'b010; tone<=7'b000_0010; end //延音
85: begin pitch<=3'b010; tone<=7'b000_0100; end //中音 3
86: begin pitch<=3'b010; tone<=7'b000_0100; end //延音
87: begin pitch<=3'b010; tone<=7'b000_1000; end //中音 4
88: begin pitch<=3'b010; tone<=7'b000_1000; end //延音
89: begin pitch<=3'b010; tone<=7'b000_0100; end //中音 3
90: begin pitch<=3'b010; tone<=7'b000_1000; end //中音 4
91: begin pitch<=3'b010; tone<=7'b000_1000; end //延音
92: begin pitch<=3'b010; tone<=7'b000_0100; end //中音 3
93: begin pitch<=3'b010; tone<=7'b000_0100; end //中音 3
94: begin pitch<=3'b010; tone<=7'b000_0100; end //延音
95: begin pitch<=3'b010; tone<=7'b000_0100; end //延音
96: begin pitch<=3'b010; tone<=7'b000_0100; end //延音
97: begin pitch<=3'b010; tone<=7'b000_0001; end //中音 1
98: begin pitch<=3'b010; tone<=7'b000_0001; end //延音
99: begin pitch<=3'b100; tone<=7'b100_0000; end //低音 7
100: begin pitch<=3'b100; tone<=7'b100_0000; end //延音
101: begin pitch<=3'b100; tone<=7'b100_0000; end //低音 7
102: begin pitch<=3'b100; tone<=7'b100_0000; end //延音
103: begin pitch<=3'b010; tone<=7'b000_0001; end //中音 1
104: begin pitch<=3'b010; tone<=7'b000_0001; end //延音
105: begin pitch<=3'b010; tone<=7'b000_0001; end //中音 1
106: begin pitch<=3'b010; tone<=7'b000_0001; end //延音
107: begin pitch<=3'b010; tone<=7'b000_0010; end //中音 2
108: begin pitch<=3'b010; tone<=7'b000_0100; end //中音 3
109: begin pitch<=3'b010; tone<=7'b000_0010; end //中音 2
110: begin pitch<=3'b010; tone<=7'b000_0010; end //延音
111: begin pitch<=3'b010; tone<=7'b000_0010; end //延音
112: begin pitch<=3'b010; tone<=7'b010_0000; end //中音 6
113: begin pitch<=3'b010; tone<=7'b000_0001; end //中音 1
114: begin pitch<=3'b010; tone<=7'b000_0001; end //延音
115: begin pitch<=3'b010; tone<=7'b000_0010; end //中音 2
116: begin pitch<=3'b010; tone<=7'b000_0010; end //延音
117: begin pitch<=3'b010; tone<=7'b000_0010; end //延音
118: begin pitch<=3'b010; tone<=7'b000_0010; end //延音
119: begin pitch<=3'b010; tone<=7'b000_0001; end //中音 1
```

```
        120: begin pitch<=3'b010; tone<=7'b000_0001; end //延音
        121: begin pitch<=3'b010; tone<=7'b000_0001; end //中音1
        122: begin pitch<=3'b010; tone<=7'b000_0001; end //延音
        123: begin pitch<=3'b010; tone<=7'b000_0001; end //延音
        124: begin pitch<=3'b010; tone<=7'b000_0001; end //延音
        125: begin pitch<=3'b000; tone<=7'b000_0000; end //休止符
        126: begin pitch<=3'b000; tone<=7'b000_0000; end //休止符
        127: begin pitch<=3'b000; tone<=7'b000_0000; end //休止符
    endcase
end
else if(choice==3'b010)//乐曲"错位时空"
begin
case(addr)
        1: begin pitch=3'b100; tone=7'b010_0000; end //低音6
        2: begin pitch<=3'b100; tone<=7'b010_0000; end //延音
        3: begin pitch<=3'b010; tone<=7'b000_0100; end //中音3
        4: begin pitch<=3'b010; tone<=7'b000_0100; end //延音
        5: begin pitch<=3'b010; tone<=7'b000_0100; end //中音3
        6: begin pitch<=3'b010; tone<=7'b000_0010; end //中音2
        7: begin pitch<=3'b010; tone<=7'b000_0100; end //中音3
        8: begin pitch<=3'b010; tone<=7'b010_0000; end //中音6
        9: begin pitch<=3'b010; tone<=7'b001_0000; end //中音5
        10: begin pitch<=3'b010; tone<=7'b000_0100; end //中音3
        11: begin pitch<=3'b010; tone<=7'b000_0100; end //延音
        12: begin pitch<=3'b010; tone<=7'b000_0100; end //延音
        13: begin pitch<=3'b010; tone<=7'b001_0000; end //中音5
        14: begin pitch<=3'b010; tone<=7'b001_0000; end //延音
        15: begin pitch<=3'b010; tone<=7'b010_0000; end //中音6
        16: begin pitch<=3'b010; tone<=7'b010_0000; end //延音
        17: begin pitch<=3'b010; tone<=7'b001_0000; end //中音5
        18: begin pitch<=3'b010; tone<=7'b010_0000; end //中音6
        19: begin pitch<=3'b010; tone<=7'b010_0000; end //中音6
        20: begin pitch<=3'b010; tone<=7'b001_0000; end //中音5
        21: begin pitch<=3'b010; tone<=7'b000_1000; end //中音4
        22: begin pitch<=3'b010; tone<=7'b010_0000; end //中音6
        23: begin pitch<=3'b010; tone<=7'b001_0000; end //中音5
        24: begin pitch<=3'b010; tone<=7'b001_0000; end //延音
        25: begin pitch<=3'b010; tone<=7'b000_0001; end //中音1
        26: begin pitch<=3'b010; tone<=7'b000_0001; end //延音
        27: begin pitch<=3'b010; tone<=7'b000_0100; end //中音3
        28: begin pitch<=3'b010; tone<=7'b000_0100; end //延音
        29: begin pitch<=3'b010; tone<=7'b000_0100; end //延音
        30: begin pitch<=3'b010; tone<=7'b000_0100; end //延音
        31: begin pitch<=3'b010; tone<=7'b000_0010; end //中音2
```

```
32: begin pitch<=3'b010; tone<=7'b000_0010; end //延音
33: begin pitch<=3'b010; tone<=7'b000_0010; end //中音2
34: begin pitch<=3'b010; tone<=7'b000_0100; end //中音3
35: begin pitch<=3'b010; tone<=7'b000_1000; end //中音4
36: begin pitch<=3'b010; tone<=7'b000_1000; end //延音
37: begin pitch<=3'b010; tone<=7'b010_0000; end //中音6
38: begin pitch<=3'b010; tone<=7'b010_0000; end //延音
39: begin pitch<=3'b010; tone<=7'b000_0100; end //中音3
40: begin pitch<=3'b010; tone<=7'b000_0100; end //延音
41: begin pitch<=3'b010; tone<=7'b000_0010; end //中音2
42: begin pitch<=3'b010; tone<=7'b000_0010; end //延音
43: begin pitch<=3'b010; tone<=7'b000_0001; end //中音1
44: begin pitch<=3'b010; tone<=7'b000_0001; end //延音
45: begin pitch<=3'b010; tone<=7'b000_0001; end //延音
46: begin pitch<=3'b010; tone<=7'b000_0001; end //延音
47: begin pitch<=3'b010; tone<=7'b000_0010; end //中音2
48: begin pitch<=3'b010; tone<=7'b000_0010; end //延音
49: begin pitch<=3'b010; tone<=7'b000_0100; end //中音3
50: begin pitch<=3'b010; tone<=7'b000_0100; end //延音
51: begin pitch<=3'b010; tone<=7'b000_0010; end //中音2
52: begin pitch<=3'b010; tone<=7'b000_0010; end //延音
53: begin pitch<=3'b010; tone<=7'b000_0100; end //中音3
54: begin pitch<=3'b010; tone<=7'b000_0100; end //延音
55: begin pitch<=3'b100; tone<=7'b100_0000; end //低音7
56: begin pitch<=3'b100; tone<=7'b100_0000; end //延音
57: begin pitch<=3'b100; tone<=7'b100_0000; end //延音
58: begin pitch<=3'b100; tone<=7'b100_0000; end //延音
59: begin pitch<=3'b000; tone<=7'b000_0000; end //休止符
60: begin pitch<=3'b000; tone<=7'b000_0000; end //休止符
61: begin pitch<=3'b000; tone<=7'b000_0000; end //休止符
62: begin pitch<=3'b000; tone<=7'b000_0000; end //休止符
63: begin pitch<=3'b100; tone<=7'b010_0000; end //低音6
64: begin pitch<=3'b100; tone<=7'b010_0000; end //延音
65: begin pitch<=3'b010; tone<=7'b000_0100; end //中音3
66: begin pitch<=3'b010; tone<=7'b010_0100; end //延音
67: begin pitch<=3'b010; tone<=7'b000_0100; end //中音3
68: begin pitch<=3'b010; tone<=7'b000_0010; end //中音2
69: begin pitch<=3'b010; tone<=7'b000_0100; end //中音3
70: begin pitch<=3'b010; tone<=7'b010_0000; end //中音6
71: begin pitch<=3'b010; tone<=7'b001_0000; end //中音5
72: begin pitch<=3'b010; tone<=7'b001_0000; end //延音
73: begin pitch<=3'b010; tone<=7'b000_0100; end //中音3
74: begin pitch<=3'b010; tone<=7'b000_0100; end //延音
75: begin pitch<=3'b010; tone<=7'b001_0000; end //中音5
```

```
76: begin pitch<=3'b010; tone<=7'b001_0000; end //延音
77: begin pitch<=3'b010; tone<=7'b001_0000; end //延音
78: begin pitch<=3'b010; tone<=7'b001_0000; end //延音
79: begin pitch<=3'b010; tone<=7'b010_0000; end //中音 6
80: begin pitch<=3'b010; tone<=7'b010_0000; end //延音
81: begin pitch<=3'b001; tone<=7'b000_0001; end //高音 1
82: begin pitch<=3'b001; tone<=7'b000_0001; end //延音
83: begin pitch<=3'b010; tone<=7'b100_0000; end //中音 7
84: begin pitch<=3'b010; tone<=7'b010_0000; end //中音 6
85: begin pitch<=3'b010; tone<=7'b001_0000; end //中音 5
86: begin pitch<=3'b010; tone<=7'b010_0000; end //中音 6
87: begin pitch<=3'b010; tone<=7'b001_0000; end //中音 5
88: begin pitch<=3'b010; tone<=7'b001_0000; end //延音
89: begin pitch<=3'b010; tone<=7'b000_0001; end //中音 1
90: begin pitch<=3'b010; tone<=7'b000_0001; end //延音
91: begin pitch<=3'b010; tone<=7'b000_0100; end //中音 3
92: begin pitch<=3'b010; tone<=7'b000_0100; end //延音
93: begin pitch<=3'b010; tone<=7'b000_0100; end //延音
94: begin pitch<=3'b010; tone<=7'b000_0100; end //延音
95: begin pitch<=3'b010; tone<=7'b000_0010; end //中音 2
96: begin pitch<=3'b010; tone<=7'b000_0010; end //延音
97: begin pitch<=3'b010; tone<=7'b000_0010; end //中音 2
98: begin pitch<=3'b010; tone<=7'b000_0100; end //中音 3
99: begin pitch<=3'b010; tone<=7'b000_1000; end //中音 4
100: begin pitch<=3'b010; tone<=7'b000_1000; end //中音 4
101: begin pitch<=3'b010; tone<=7'b000_1000; end //中音 4
102: begin pitch<=3'b010; tone<=7'b000_0010; end //中音 2
103: begin pitch<=3'b010; tone<=7'b100_0100; end //中音 3
104: begin pitch<=3'b010; tone<=7'b000_0100; end //延音
105: begin pitch<=3'b010; tone<=7'b010_0000; end //中音 6
106: begin pitch<=3'b010; tone<=7'b010_0000; end //延音
107: begin pitch<=3'b001; tone<=7'b000_0001; end //高音 1
108: begin pitch<=3'b001; tone<=7'b000_0001; end //延音
109: begin pitch<=3'b001; tone<=7'b000_0001; end //延音
110: begin pitch<=3'b001; tone<=7'b000_0001; end //延音
111: begin pitch<=3'b001; tone<=7'b000_0010; end //高音 2
112: begin pitch<=3'b001; tone<=7'b000_0010; end //延音
113: begin pitch<=3'b001; tone<=7'b000_0010; end //延音
114: begin pitch<=3'b001; tone<=7'b000_0100; end //高音 3
115: begin pitch<=3'b001; tone<=7'b000_0010; end //高音 2
116: begin pitch<=3'b001; tone<=7'b000_0010; end //延音
117: begin pitch<=3'b001; tone<=7'b000_0100; end //高音 3
118: begin pitch<=3'b001; tone<=7'b000_0100; end //延音
119: begin pitch<=3'b010; tone<=7'b100_0000; end //中音 7
```

```
        120: begin pitch<=3'b010; tone<=7'b100_0000; end //延音
        121: begin pitch<=3'b010; tone<=7'b100_0000; end //延音
        122: begin pitch<=3'b010; tone<=7'b100_0000; end //延音
        123: begin pitch<=3'b000; tone<=7'b000_0000; end //休止符
        124: begin pitch<=3'b000; tone<=7'b000_0000; end //休止符
        125: begin pitch<=3'b000; tone<=7'b000_0000; end //休止符
        126: begin pitch<=3'b000; tone<=7'b000_0000; end //休止符
        127: begin pitch<=3'b000; tone<=7'b000_0000; end //休止符
    endcase
end
else if(choice==3'b100)//乐曲"天空之城"
begin
case(addr)
        1: begin pitch=3'b100; tone=7'b010_0000; end //低音 6
        2: begin pitch<=3'b100; tone<=7'b100_0000; end //低音 7
        3: begin pitch<=3'b010; tone<=7'b000_0001; end //中音 1
        4: begin pitch<=3'b010; tone<=7'b000_0001; end //延音
        5: begin pitch<=3'b010; tone<=7'b000_0001; end //延音
        6: begin pitch<=3'b100; tone<=7'b100_0000; end //低音 7
        7: begin pitch<=3'b010; tone<=7'b000_0001; end //中音 1
        8: begin pitch<=3'b010; tone<=7'b000_0001; end //延音
        9: begin pitch<=3'b010; tone<=7'b000_0100; end //中音 3
        10: begin pitch<=3'b010; tone<=7'b000_0100; end //延音
        11: begin pitch<=3'b100; tone<=7'b100_0000; end //低音 7
        12: begin pitch<=3'b100; tone<=7'b100_0000; end //延音
        13: begin pitch<=3'b100; tone<=7'b100_0000; end //延音
        14: begin pitch<=3'b100; tone<=7'b100_0000; end //延音
        15: begin pitch<=3'b100; tone<=7'b100_0000; end //延音
        16: begin pitch<=3'b100; tone<=7'b100_0000; end //延音
        17: begin pitch<=3'b100; tone<=7'b000_0100; end //低音 3
        18: begin pitch<=3'b100; tone<=7'b000_0100; end //延音
        19: begin pitch<=3'b100; tone<=7'b010_0000; end //低音 6
        20: begin pitch<=3'b100; tone<=7'b010_0000; end //延音
        21: begin pitch<=3'b100; tone<=7'b010_0000; end //延音
        22: begin pitch<=3'b100; tone<=7'b001_0000; end //低音 5
        23: begin pitch<=3'b100; tone<=7'b010_0000; end //低音 6
        24: begin pitch<=3'b100; tone<=7'b010_0000; end //延音
        25: begin pitch<=3'b010; tone<=7'b000_0001; end //中音 1
        26: begin pitch<=3'b010; tone<=7'b000_0001; end //延音
        27: begin pitch<=3'b100; tone<=7'b001_0000; end //低音 5
        28: begin pitch<=3'b100; tone<=7'b001_0000; end //延音
        29: begin pitch<=3'b100; tone<=7'b001_0000; end //延音
        30: begin pitch<=3'b100; tone<=7'b001_0000; end //延音
        31: begin pitch<=3'b100; tone<=7'b001_0000; end //延音
```

```
32: begin pitch<=3'b100; tone<=7'b001_0000; end //延音
33: begin pitch<=3'b100; tone<=7'b000_0100; end //低音3
34: begin pitch<=3'b100; tone<=7'b000_0100; end //延音
35: begin pitch<=3'b100; tone<=7'b000_1000; end //低音4
36: begin pitch<=3'b100; tone<=7'b000_1000; end //延音
37: begin pitch<=3'b100; tone<=7'b000_1000; end //延音
38: begin pitch<=3'b100; tone<=7'b000_0100; end //低音3
39: begin pitch<=3'b100; tone<=7'b000_1000; end //低音4
40: begin pitch<=3'b010; tone<=7'b000_0001; end //中音1
41: begin pitch<=3'b010; tone<=7'b000_0001; end //延音
42: begin pitch<=3'b010; tone<=7'b000_0001; end //延音
43: begin pitch<=3'b100; tone<=7'b000_0100; end //低音3
44: begin pitch<=3'b100; tone<=7'b000_0100; end //延音
45: begin pitch<=3'b100; tone<=7'b000_0100; end //延音
46: begin pitch<=3'b100; tone<=7'b000_0100; end //延音
47: begin pitch<=3'b100; tone<=7'b000_0100; end //延音
48: begin pitch<=3'b100; tone<=7'b000_0100; end //延音
49: begin pitch<=3'b010; tone<=7'b000_0001; end //中音1
50: begin pitch<=3'b010; tone<=7'b000_0001; end //延音
51: begin pitch<=3'b100; tone<=7'b100_0000; end //低音7
52: begin pitch<=3'b100; tone<=7'b100_0000; end //延音
53: begin pitch<=3'b100; tone<=7'b100_0000; end //延音
54: begin pitch<=3'b100; tone<=7'b000_1000; end //低音4
55: begin pitch<=3'b100; tone<=7'b000_1000; end //延音
56: begin pitch<=3'b100; tone<=7'b000_1000; end //延音
57: begin pitch<=3'b100; tone<=7'b000_1000; end //低音4
58: begin pitch<=3'b100; tone<=7'b100_0000; end //低音7
59: begin pitch<=3'b100; tone<=7'b100_0000; end //延音
60: begin pitch<=3'b100; tone<=7'b100_0000; end //延音
61: begin pitch<=3'b100; tone<=7'b100_0000; end //延音
62: begin pitch<=3'b100; tone<=7'b100_0000; end //延音
63: begin pitch<=3'b100; tone<=7'b100_0000; end //延音
64: begin pitch<=3'b000; tone<=7'b000_0000; end //休止符
65: begin pitch<=3'b000; tone<=7'b000_0000; end //休止符
66: begin pitch<=3'b100; tone<=7'b010_0000; end //低音6
67: begin pitch<=3'b100; tone<=7'b100_0000; end //低音7
68: begin pitch<=3'b010; tone<=7'b000_0001; end //中音1
69: begin pitch<=3'b010; tone<=7'b000_0001; end //延音
70: begin pitch<=3'b010; tone<=7'b000_0001; end //延音
71: begin pitch<=3'b100; tone<=7'b100_0000; end //低音7
72: begin pitch<=3'b010; tone<=7'b000_0001; end //中音1
73: begin pitch<=3'b010; tone<=7'b000_0001; end //延音
74: begin pitch<=3'b010; tone<=7'b000_0100; end //中音3
75: begin pitch<=3'b010; tone<=7'b000_0100; end //延音
```

```
76: begin pitch<=3'b100; tone<=7'b100_0000; end //低音 7
77: begin pitch<=3'b100; tone<=7'b100_0000; end //延音
78: begin pitch<=3'b100; tone<=7'b100_0000; end //延音
79: begin pitch<=3'b100; tone<=7'b100_0000; end //延音
80: begin pitch<=3'b100; tone<=7'b100_0000; end //延音
81: begin pitch<=3'b100; tone<=7'b100_0000; end //延音
82: begin pitch<=3'b100; tone<=7'b000_0100; end //低音 3
83: begin pitch<=3'b100; tone<=7'b000_0100; end //延音
84: begin pitch<=3'b100; tone<=7'b010_0000; end //低音 6
85: begin pitch<=3'b100; tone<=7'b010_0000; end //延音
86: begin pitch<=3'b100; tone<=7'b010_0000; end //延音
87: begin pitch<=3'b100; tone<=7'b001_0000; end //低音 5
88: begin pitch<=3'b100; tone<=7'b010_0000; end //低音 6
89: begin pitch<=3'b100; tone<=7'b010_0000; end //延音
90: begin pitch<=3'b010; tone<=7'b000_0001; end //中音 1
91: begin pitch<=3'b010; tone<=7'b000_0001; end //延音
92: begin pitch<=3'b100; tone<=7'b001_0000; end //低音 5
93: begin pitch<=3'b100; tone<=7'b001_0000; end //延音
94: begin pitch<=3'b100; tone<=7'b001_0000; end //延音
95: begin pitch<=3'b100; tone<=7'b001_0000; end //延音
96: begin pitch<=3'b100; tone<=7'b001_0000; end //延音
97: begin pitch<=3'b100; tone<=7'b001_0000; end //延音
98: begin pitch<=3'b100; tone<=7'b000_0100; end //低音 3
99: begin pitch<=3'b100; tone<=7'b000_1000; end //低音 4
100: begin pitch<=3'b010; tone<=7'b000_0001; end //中音 1
101: begin pitch<=3'b100; tone<=7'b100_0000; end //低音 7
102: begin pitch<=3'b100; tone<=7'b100_0000; end //延音
103: begin pitch<=3'b100; tone<=7'b100_0000; end //延音
104: begin pitch<=3'b010; tone<=7'b000_0001; end //中音 1
105: begin pitch<=3'b010; tone<=7'b000_0001; end //延音
106: begin pitch<=3'b010; tone<=7'b000_0010; end //中音 2
107: begin pitch<=3'b010; tone<=7'b000_0010; end //延音
108: begin pitch<=3'b010; tone<=7'b000_0100; end //中音 3
109: begin pitch<=3'b010; tone<=7'b000_0001; end //中音 1
110: begin pitch<=3'b010; tone<=7'b000_0001; end //延音
111: begin pitch<=3'b010; tone<=7'b000_0001; end //延音
112: begin pitch<=3'b010; tone<=7'b000_0001; end //延音
113: begin pitch<=3'b000; tone<=7'b000_0000; end //休止符
114: begin pitch<=3'b010; tone<=7'b000_0001; end //中音 1
115: begin pitch<=3'b100; tone<=7'b100_0000; end //低音 7
116: begin pitch<=3'b100; tone<=7'b010_0000; end //低音 6
117: begin pitch<=3'b100; tone<=7'b010_0000; end //延音
118: begin pitch<=3'b100; tone<=7'b100_0000; end //低音 7
119: begin pitch<=3'b100; tone<=7'b100_0000; end //延音
```

```
                120: begin pitch<=3'b100; tone<=7'b001_0000; end //低音 5
                121: begin pitch<=3'b100; tone<=7'b001_0000; end //延音
                122: begin pitch<=3'b100; tone<=7'b010_0000; end //低音 6
                123: begin pitch<=3'b100; tone<=7'b010_0000; end //延音
                124: begin pitch<=3'b100; tone<=7'b010_0000; end //延音
                125: begin pitch<=3'b100; tone<=7'b010_0000; end //延音
                126: begin pitch<=3'b000; tone<=7'b000_0000; end //休止符
                127: begin pitch<=3'b000; tone<=7'b000_0000; end //休止符
            endcase
        end
    end
endmodule
```

3. 音乐播放模块

3.1 功能描述：根据输入的音调和音高计算对系统时钟的分频系数，输出分频后的时钟至音频接口。

3.2 实现方法：音乐产生的原理是频率，而不同音调音高的音符的频率可以根据查表得到。

音阶		Octave0	Octave1	Octave2	Octave3
Do	C	262	523	1047	2093
	Db	277	554	1109	2217
Re	D	294	587	1175	2349
	Eb	311	622	1245	2489
Mi	E	330	659	1329	2637
Fa	F	349	698	1397	2794
	Gb	370	740	1480	2960
Sol	G	392	784	1568	3136
	Ab	415	831	1661	3322
La	A	440	880	1760	3520
	Bb	466	923	1865	3729
Si	B	494	988	1976	3951

3.3 verilog 代码：

```
module music_creator(
    input clk,//100MHZ 系统时钟
    input rst_n,//复位信号（低电平有效）
    input [3:0] status,//当前游戏状态
    input [6:0] tone, //音调
    input [2:0] pitch, //音高
    output reg speaker//扬声器
    );
```

```
wire [6:0] high, middle, low; //定义低中高音

//调用分频模块
//低音1~7(分别为low[0]~low[7])
divider #(388200/2) d_l1(.clk(clk),.clk_out(low[0]));
divider #(340500/2) d_l2(.clk(clk),.clk_out(low[1]));
divider #(303300/2) d_l3(.clk(clk),.clk_out(low[2]));
divider #(286300/2) d_l4(.clk(clk),.clk_out(low[3]));
divider #(255100/2) d_l5(.clk(clk),.clk_out(low[4]));
divider #(227200/2) d_l6(.clk(clk),.clk_out(low[5]));
divider #(202500/2) d_l7(.clk(clk),.clk_out(low[6]));

//中音1~7(分别为middle[0]~middle[7])
divider #(191100/2) d_m1(.clk(clk),.clk_out(middle[0]));
divider #(170200/2) d_m2(.clk(clk),.clk_out(middle[1]));
divider #(151600/2) d_m3(.clk(clk),.clk_out(middle[2]));
divider #(143100/2) d_m4(.clk(clk),.clk_out(middle[3]));
divider #(127500/2) d_m5(.clk(clk),.clk_out(middle[4]));
divider #(113600/2) d_m6(.clk(clk),.clk_out(middle[5]));
divider #(101200/2) d_m7(.clk(clk),.clk_out(middle[6]));

//高音1~7(分别为high[0]~high[7])
divider #(95600/2) d_h1(.clk(clk),.clk_out(high[0]));
divider #(85100/2) d_h2(.clk(clk),.clk_out(high[1]));
divider #(75800/2) d_h3(.clk(clk),.clk_out(high[2]));
divider #(71500/2) d_h4(.clk(clk),.clk_out(high[3]));
divider #(63700/2) d_h5(.clk(clk),.clk_out(high[4]));
divider #(56800/2) d_h6(.clk(clk),.clk_out(high[5]));
divider #(50600/2) d_h7(.clk(clk),.clk_out(high[6]));

always@(pitch,tone,rst_n)
begin
if(!rst_n||status!=3'b010) speaker<=0;
//高音部分
else if(pitch == 3'b001)
begin
    case(tone)
        7'b000_0000:begin speaker<=0; end
        7'b000_0001:begin speaker<=high[0]; end
        7'b000_0010:begin speaker<=high[1]; end
        7'b000_0100:begin speaker<=high[2]; end
        7'b000_1000:begin speaker<=high[3]; end
        7'b001_0000:begin speaker<=high[4]; end
```

```
            7'b010_0000:begin speaker<=high[5]; end
            7'b100_0000:begin speaker<=high[6]; end
        endcase
    end

    //中音部分
    else if(pitch == 3'b010)
    begin
        case(tone)
            7'b000_0000:begin speaker<=0; end
            7'b000_0001:begin speaker<=middle[0]; end
            7'b000_0010:begin speaker<=middle[1]; end
            7'b000_0100:begin speaker<=middle[2]; end
            7'b000_1000:begin speaker<=middle[3]; end
            7'b001_0000:begin speaker<=middle[4]; end
            7'b010_0000:begin speaker<=middle[5]; end
            7'b100_0000:begin speaker<=middle[6]; end
        endcase
    end

    //低音部分
    else if(pitch == 3'b100)
    begin
        case(tone)
            7'b000_0000:begin speaker<=0; end
            7'b000_0001:begin speaker<=low[0]; end
            7'b000_0010:begin speaker<=low[1]; end
            7'b000_0100:begin speaker<=low[2]; end
            7'b000_1000:begin speaker<=low[3]; end
            7'b001_0000:begin speaker<=low[4]; end
            7'b010_0000:begin speaker<=low[5]; end
            7'b100_0000:begin speaker<=low[6]; end
            endcase
        end

    else speaker<=0;
    end
endmodule
```

4. 分频模块

4.1 功能描述：根据传入的分频系数将系统时钟分频，输出分频后的时钟。

4.2 verilog 代码：

```
module divider(clk, clk_out);
```

```
    parameter DIV_NUM = 1; //对时钟进行分频

    input clk;
    output reg clk_out;
    reg [31:0]count;

    always@(posedge clk)
    begin
        if(count == DIV_NUM)
        begin
            clk_out <= !clk_out;
            count <= 0;
        end
        else
            count <= count + 1;
    end

endmodule
```

5.随机选择钢琴块产生轨道模块

5.1 功能描述：根据正在播放的音符的地址随机选择产生钢琴块的轨道。

5.2 verilog 代码：

```
module block_random(
    input [6:0]addr,//当前读取的音符地址
    input [6:0]cur_tone,//当前播放音符的音调
    input [2:0]cur_pitch,//当前播放的音符的音高
    output reg [7:0] v_enb//选择出现钢琴块的轨道
    );

   always@(addr)
   begin
       if(addr%8==0) v_enb = 8'b0000_0001;
       else if(addr%8==1) v_enb = 8'b0001_0000;
       else if(addr%8==2) v_enb = 8'b0100_0100;
       else if(addr%8==3) v_enb = 8'b0010_1000;
       else if(addr%8==4) v_enb = 8'b0000_0100;
       else if(addr%8==5) v_enb = 8'b0010_0010;
       else if(addr%8==6) v_enb = 8'b1000_0000;
       else if(addr%8==7) v_enb = 8'b0000_0001;
       else v_enb = 8'b0000_0000;
       end

endmodule
```

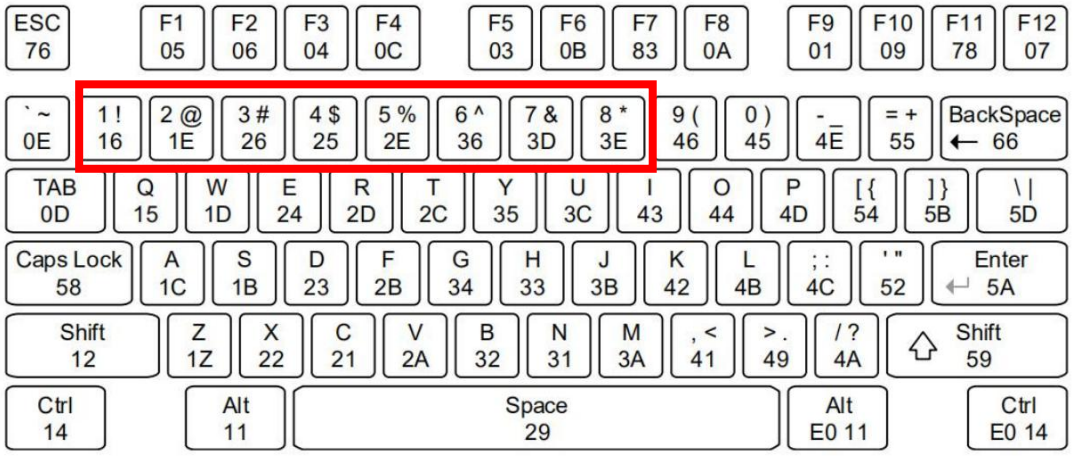
6.键盘输入模块

6.1 功能描述：接收从键盘发送过来的信号并将其存储到 keycode 中输出。

6.2 外围模块协议说明：

开发板与键盘的通信满足 PS2 通信协议。当按下一个键时，键盘送出相应键的扫描码；松开时，送出 F0 接着又是扫描码。

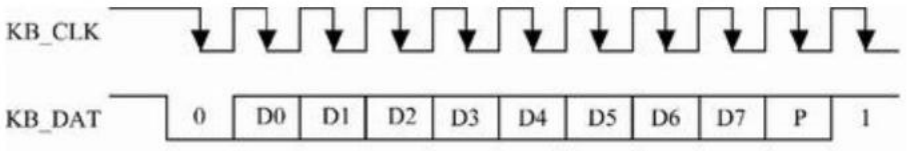
键盘各个按键对应的扫描码如下图所示（本实验用到的键用红框标出）：



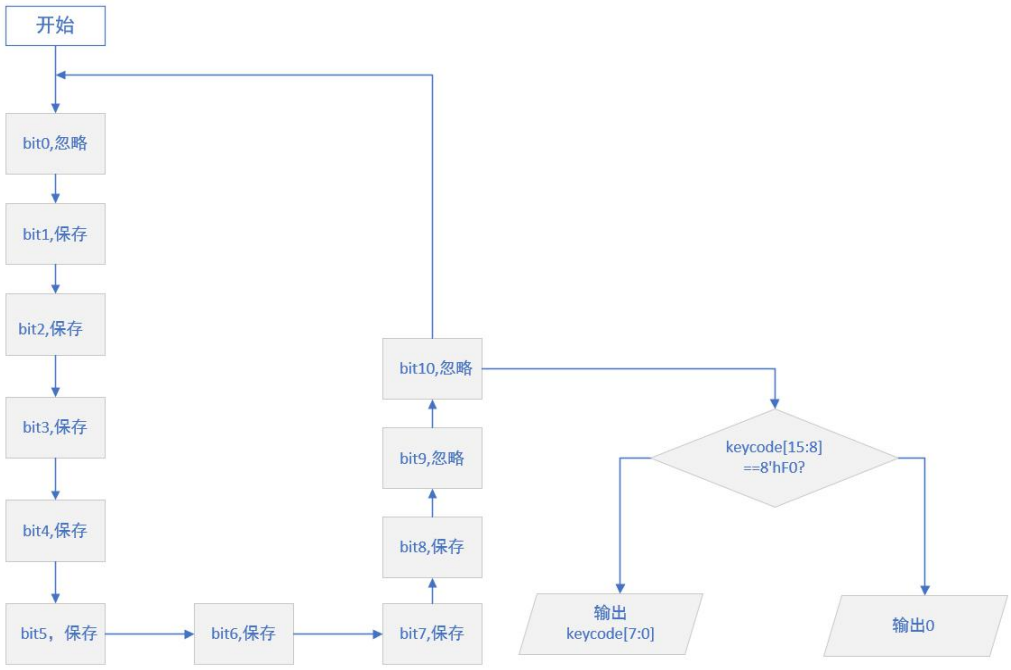
Ps2 键盘向主机发送的数据格式如下图所示：

1个起始位	总是逻辑0
8个数据位	（LSB）低位在前
1个奇偶校验位	奇校验
1个停止位	总是逻辑1

一般都是由 ps2 设备产生时钟信号。发送按帧格式。数据位在 clock 为高电平时准备好，在 clock 下降沿被 PC 读入。如下图所示：

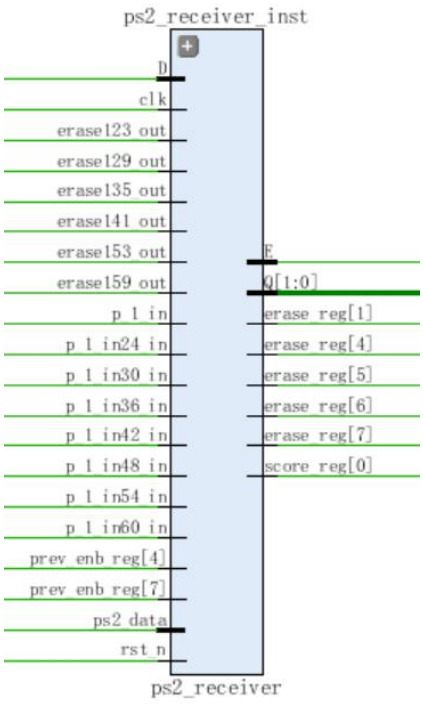


本模块对应的 ASM 图如下图所示：



在本模块中，先对键盘时钟信号进行消抖处理，将消抖后的时钟信号作为触发读取信号。读取一个 11bit 数据时，我们忽略第一个起始位和最后的奇偶校验位以及停止位，将中间的 8 位存储起来。特别需要注意的是，在本系统中，我们不能直接将当前读到的扫描码作为输出，而应该判断当前一次读到的 8 位扫描码为 F0（即此次为松开按键）时，将输出置为 0. 因为另一个模块中判断是否应该加分是以钢琴块下落阶段当前读到的键盘扫描码是否为对应轨道按键来判断。如果不采用置 0 的做法，会导致在按下某个键后即使长时间没有键按下，寄存器中记录的当前键码会一直是之前的值，分数也会一直增加。

6.3 RTL 图：



6.4 verilog 代码：

```
module ps2_receiver(
    input clk,//系统时钟
    input kclk,//键盘时钟
    input kdata,//键盘数据
    input rst_n,//清零信号
    output reg [31:0] keycode = 0//存储从键盘读到的扫描码
    );
reg [7:0] datacur;//当前读到的 8 位数据
reg [3:0] cnt;//0~10 计数，读取一个字节

//键盘按键消抖
reg [2:0] ps2_clk_sync = 3'b111;
always @ (posedge clk)
    ps2_clk_sync<={ps2_clk_sync[1:0],kclk};
wire sampling = ps2_clk_sync[2]&~ps2_clk_sync[1];

always@(posedge clk)
begin
    if (!rst_n) begin cnt=0; keycode=0; datacur=0; end
    else if(sampling)
    begin
        if(cnt==4'd10)
        begin
            keycode[31:24]=keycode[23:16];
            keycode[23:16]=keycode[15:8];
            keycode[15:8]=keycode[7:0];
            if(keycode[15:8]==8'hF0)//如果按键松开，则低四位键盘码置 0
                keycode[7:0] = 8'b0000_0000;
            else
                keycode[7:0]=datacur;
            cnt=0;
        end
        else if(cnt>= 1&& cnt<=8)
        begin
            datacur[cnt-1]=kdata;
            cnt=cnt+1;
        end
        else cnt= cnt+1;//忽略开始和结束标志以及奇偶校验位
   end
end

endmodule
```

7. VGA 显示模块

7.1 功能描述：游戏开始及游戏结束后，显示封装在 ip 核中的初始界面图片，游戏进行时，根据音乐节奏显示钢琴块，若接收到来自键盘的擦除钢琴块的信号，则将对应位置钢琴块擦除。

7.2 外围模块通信协议说明：

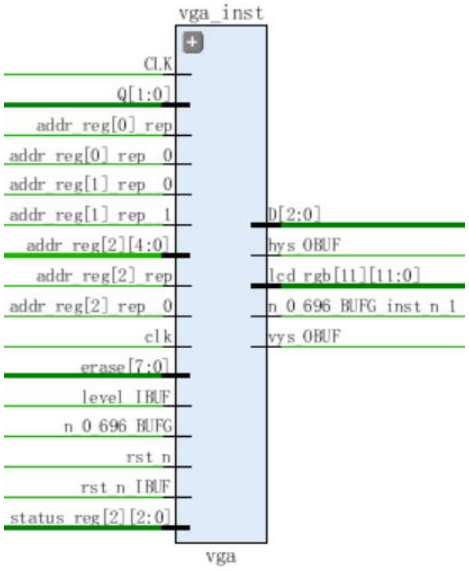
通过控制红绿蓝三基色，我们就可以控制 1 个像素的颜色。而一幅图像是由非常多的像素组成的。例如本实验用到的 640*480 分辨率的图像，是由一行 640 个、一共 480 行，这么多像素组合起来显示的图像。为了让显示器显示这么一幅图像，那么就要控制显示器的扫描枪一个一个像素地将颜色显示起来。像素变化的时间非常快，从而使人眼误认为所有像素是一起显示的。

在 VGA 模块中，我们需要关注的是行同步信号、场同步信号以及颜色信号。必须在行同步信号和场同步信号均在有效显示区域时才输出相应的颜色，否则输出黑色。

在 640*480 分辨率的图像中，每个过程对应的时间如图所示：

分辨率	行/列	同步脉冲	显示后沿	显示区域	显示前沿	帧长	单位
640*480 /60Hz	行	96	48	640	16	800	基准时钟
	列	2	33	480	10	525	行

7.3 RTL 图：



7.4 verilog 代码：

```
module vga(
    input clk,//100MHZ 系统时钟
    input vga_clk,//25MHZ 时钟（由 IP 核生成）
    input rst_n,//复位信号（低电平有效）
    input [3:0] status,//当前游戏状态
    input [7:0]v_enb,//要显示颜色的彩条对应值为 1
    input level,//根据用户选择的难度确定钢琴块下落速度
    input [7:0]erase,//用户是否点击该钢琴块
    output reg hys,//行同步
    output reg vys,//场同步
    output reg [11:0]lcd_rgb//[11:8]r,[7:4]g,[3:0]b
);
```

```
reg [9:0]h_cnt;//脉冲计数控制行同步
reg [9:0]v_cnt;//脉冲计数控制场同步
wire add_h_cnt;//控制行同步脉冲计数加 1 标志
wire end_h_cnt;//行同步颜色信号结束显示标志
wire add_v_cnt;//控制场同步脉冲计数加 1 标志
wire end_v_cnt;//场同步颜色信号结束显示标志

//行同步相关参数
parameter C_HS_SYNC_PULSE = 96;//同步脉冲
parameter C_HS_BACK_PORCH = 48;//显示后沿
parameter C_HS_ACTIVE_TIME = 640;//显示区域
parameter C_HS_FRONT_PORCH = 16; //显示前沿
parameter C_HS_LINE_PERIOD = 800;//帧长
//场同步相关参数
parameter C_VS_SYNC_PULSE = 2;//同步脉冲
parameter C_VS_BACK_PORCH = 33;//显示后沿
parameter C_VS_ACTIVE_TIME = 480;//显示区域
parameter C_VS_FRONT_PORCH = 10; //显示前沿
parameter C_VS_FRAME_PERIOD = 525;//帧长
//钢琴块相关参数
parameter width = 80;
parameter length = 160;


//每次更新一个像素点，简单模式下更新频率为 2*160HZ，挑战模式下更新频率为
4*160HZ
//根据玩家选择的难度确定钢琴块从显示区域顶部下落到底部所需的更新次数
wire clk_64MHZ;
block_drop_clock d_64mhz(.clk_in1(clk),.clk_out1(clk_64MHZ));
wire clk1;//简单模式下的控制钢琴块下落时钟
wire clk2;//挑战模式下控制钢琴块下落时钟
divider #(200000/2)d_2c160HZ(.clk(clk_64MHZ),.clk_out(clk1));
divider #(100000/2)d_4c160HZ(.clk(clk_64MHZ),.clk_out(clk2));
reg clk_chosen;//根据玩家选择的难度选择时钟
always@(level)
begin
    if(level==0) clk_chosen <= clk1;
    else clk_chosen <= clk2;
end

//设计场同步信号 hys
always @(posedge vga_clk or negedge rst_n)
begin
```

```
     if(!rst_n)
     begin
         h_cnt <= 0;
     end
     else if(add_h_cnt)
     begin
         if(end_h_cnt)
             h_cnt <= 0;
         else
             h_cnt <= h_cnt + 1;
     end
end
assign add_h_cnt = 1;
assign end_h_cnt = add_h_cnt && h_cnt==C_HS_LINE_PERIOD - 1;

always@(posedge vga_clk or negedge rst_n)
begin
     if(!rst_n)
     begin
         hys <= 0;
     end
     else if(add_h_cnt && h_cnt ==C_HS_SYNC_PULSE -1)
     begin
         hys <= 1'b1;
     end
     else if(end_h_cnt)
     begin
         hys <= 1'b0;
     end
end

//设计 vys 信号
always @(posedge vga_clk or negedge rst_n)
begin
    if(!rst_n)
    begin
        v_cnt <= 0;
    end
    else if(add_v_cnt)
    begin
        if(end_v_cnt)
            v_cnt <= 0;
    else
        v_cnt <= v_cnt + 1;
```

```
    end
end
assign add_v_cnt = end_h_cnt;
assign end_v_cnt = add_v_cnt && v_cnt==C_VS_FRAME_PERIOD - 1;

always @(posedge vga_clk or negedge rst_n)
begin
    if(!rst_n)
    begin
        vys <= 1'b0;
    end
    else if(add_v_cnt && v_cnt ==  C_VS_SYNC_PULSE - 1)
    begin
        vys <= 1'b1;
    end
    else if(end_v_cnt)
    begin
        vys <= 1'b0;
    end
end

//设计 lcd_rgb 信号
reg whole_area;//整个颜色显示有效区域标志
reg [7:0] area; //接下来七个颜色显示区域标志分别对应七个音符

reg [11:0]count0;
reg [11:0]count1;
reg [11:0]count2;
reg [11:0]count3;
reg [11:0]count4;
reg [11:0]count5;
reg [11:0]count6;
reg [11:0]count7;
reg [7:0]prev_enb=8'b0000_0000;
reg [7:0]flag;//flag 等于 1 时对应轨道钢琴块能够显示

always@(posedge clk_chosen)
begin
    if(prev_enb[0]==0 && v_enb[0]==1) begin count0<=0; flag[0] <= 1; end
    if(count0==C_VS_ACTIVE_TIME+length || erase[0] == 1) flag[0] <= 0;
    if(flag[0] == 1) count0<=count0+1;

    if(prev_enb[1]==0 && v_enb[1]==1) begin count1<=0; flag[1] <= 1; end
    if(count1==C_VS_ACTIVE_TIME+length || erase[1] == 1) flag[1] <= 0;
```

```
    if(flag[1] == 1) count1<=count1+1;

    if(prev_enb[2]==0 && v_enb[2]==1) begin count2<=0; flag[2] <= 1; end
    if(count2==C_VS_ACTIVE_TIME+length || erase[2] == 1) flag[2] <= 0;
    if(flag[2] == 1) count2<=count2+1;

    if(prev_enb[3]==0 && v_enb[3]==1) begin count3<=0; flag[3] <= 1; end
    if(count3==C_VS_ACTIVE_TIME+length || erase[3] == 1) flag[3] <= 0;
    if(flag[3] == 1) count3<=count3+1;

    if(prev_enb[4]==0 && v_enb[4]==1) begin count4<=0; flag[4] <= 1; end
    if(count4==C_VS_ACTIVE_TIME+length || erase[4] == 1) flag[4] <= 0;
    if(flag[4] == 1) count4<=count4+1;

    if(prev_enb[5]==0 && v_enb[5]==1) begin count5<=0; flag[5] <= 1; end
    if(count5==C_VS_ACTIVE_TIME+length || erase[5] == 1) flag[5] <= 0;
    if(flag[5] == 1) count5<=count5+1;

    if(prev_enb[6]==0 && v_enb[6]==1) begin count6<=0; flag[6] <= 1; end
    if(count6==C_VS_ACTIVE_TIME+length || erase[6] == 1) flag[6] <= 0;
    if(flag[6] == 1) count6<=count6+1;

    if(prev_enb[7]==0 && v_enb[7]==1) begin count7<=0; flag[7] <= 1; end
    if(count7==C_VS_ACTIVE_TIME+length || erase[7] == 1) flag[7] <= 0;
    if(flag[7] == 1) count7<=count7+1;

    prev_enb <= v_enb;
end

always @(posedge vga_clk)
begin
    whole_area      <=      (h_cnt>=(C_HS_SYNC_PULSE+C_HS_BACK_PORCH)       &&
h_cnt<(C_HS_SYNC_PULSE+C_HS_BACK_PORCH+C_HS_ACTIVE_TIME))
                &&      (v_cnt>=(C_VS_SYNC_PULSE+C_VS_BACK_PORCH)       &&
v_cnt<(C_VS_SYNC_PULSE+C_VS_BACK_PORCH+C_VS_ACTIVE_TIME));
    area[0]         <=          flag[0]         &&          whole_area          &&
(h_cnt>=(C_HS_SYNC_PULSE+C_HS_BACK_PORCH)                                       &&
h_cnt<(C_HS_SYNC_PULSE+C_HS_BACK_PORCH+width))
             && (v_cnt>=(C_VS_SYNC_PULSE+C_VS_BACK_PORCH-length+count0) &&
v_cnt<(C_VS_SYNC_PULSE+C_VS_BACK_PORCH+count0));
    area[1]         <=          flag[1]         &&          whole_area          &&
(h_cnt>=(C_HS_SYNC_PULSE+C_HS_BACK_PORCH+width)                                 &&
h_cnt<(C_HS_SYNC_PULSE+C_HS_BACK_PORCH+2*width))
             && (v_cnt>=(C_VS_SYNC_PULSE+C_VS_BACK_PORCH-length+count1) &&
```

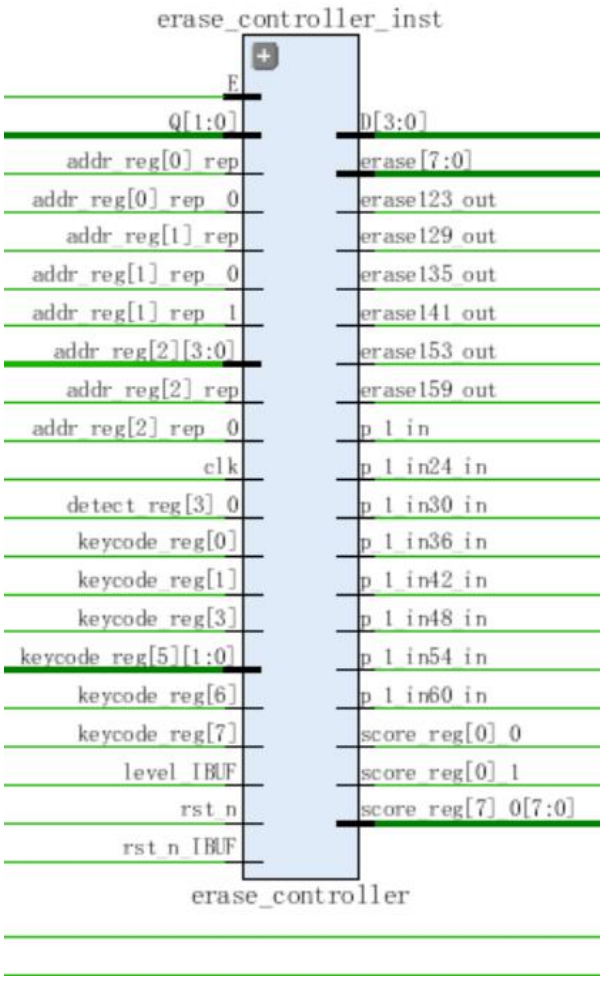
```
v_cnt<(C_VS_SYNC_PULSE+C_VS_BACK_PORCH+count1));
    area[2]         <=         flag[2]         &&         whole_area         &&
(h_cnt>=(C_HS_SYNC_PULSE+C_HS_BACK_PORCH+2*width)                              &&
h_cnt<(C_HS_SYNC_PULSE+C_HS_BACK_PORCH+3*width))
              && (v_cnt>=(C_VS_SYNC_PULSE+C_VS_BACK_PORCH-length+count2) &&
v_cnt<(C_VS_SYNC_PULSE+C_VS_BACK_PORCH+count2));
    area[3]         <=         flag[3]         &&         whole_area         &&
(h_cnt>=(C_HS_SYNC_PULSE+C_HS_BACK_PORCH+3*width)                              &&
h_cnt<(C_HS_SYNC_PULSE+C_HS_BACK_PORCH+4*width))
              && (v_cnt>=(C_VS_SYNC_PULSE+C_VS_BACK_PORCH-length+count3) &&
v_cnt<(C_VS_SYNC_PULSE+C_VS_BACK_PORCH+count3));
    area[4]         <=         flag[4]         &&         whole_area         &&
(h_cnt>=(C_HS_SYNC_PULSE+C_HS_BACK_PORCH+4*width)                              &&
h_cnt<(C_HS_SYNC_PULSE+C_HS_BACK_PORCH+5*width))
              && (v_cnt>=(C_VS_SYNC_PULSE+C_VS_BACK_PORCH-length+count4) &&
v_cnt<(C_VS_SYNC_PULSE+C_VS_BACK_PORCH+count4));
    area[5]         <=         flag[5]         &&         whole_area         &&
(h_cnt>=(C_HS_SYNC_PULSE+C_HS_BACK_PORCH+5*width)                              &&
h_cnt<(C_HS_SYNC_PULSE+C_HS_BACK_PORCH+6*width))
              && (v_cnt>=(C_VS_SYNC_PULSE+C_VS_BACK_PORCH-length+count5) &&
v_cnt<(C_VS_SYNC_PULSE+C_VS_BACK_PORCH+count5));
    area[6]         <=         flag[6]         &&         whole_area         &&
(h_cnt>=(C_HS_SYNC_PULSE+C_HS_BACK_PORCH+6*width)                              &&
h_cnt<(C_HS_SYNC_PULSE+C_HS_BACK_PORCH+7*width))
              && (v_cnt>=(C_VS_SYNC_PULSE+C_VS_BACK_PORCH-length+count6) &&
v_cnt<(C_VS_SYNC_PULSE+C_VS_BACK_PORCH+count6));
    area[7]         <=         flag[7]         &&         whole_area         &&
(h_cnt>=(C_HS_SYNC_PULSE+C_HS_BACK_PORCH+7*width)                              &&
h_cnt<(C_HS_SYNC_PULSE+C_HS_BACK_PORCH+8*width))
              && (v_cnt>=(C_VS_SYNC_PULSE+C_VS_BACK_PORCH-length+count7) &&
v_cnt<(C_VS_SYNC_PULSE+C_VS_BACK_PORCH+count7));
end

//开始界面图片导入
 reg [18:0]address_start;
 always@(posedge clk)
 begin
    if(!rst_n) address_start <=0;
    else              if(whole_area)              address_start              <=
(v_cnt-(C_VS_SYNC_PULSE+C_VS_BACK_PORCH))*640+(h_cnt-(C_HS_SYNC_PULSE+C_HS_
BACK_PORCH) );
    else address_start<=0;
 end
wire [11:0] data_start;
```

```
start_picture
m0(.clka(clk),.ena(1'b1),.wea(1'b0),.addra(address_start),.dina(12'd0),.dou
ta(data_start));

always @(posedge vga_clk or negedge rst_n)
begin
    if(!rst_n) lcd_rgb <= 12'b0000_0000_0000;
    else if(status!=3'b010 && whole_area) lcd_rgb <= data_start;
    else if (area) lcd_rgb <= 12'b1111_1111_1111;
    else lcd_rgb <= 12'b0000_0000_0000;
end

endmodule
```

8. 控制是否擦除钢琴块子系统：

4.1 功能描述：若在某轨道钢琴块下落期间接收到来自键盘的擦除该轨道钢琴块信号，则将对应轨道的擦除标志 erase 置 1，并使得分加 1.

4.2 RTL 图：



4.3 verilog 代码:

```
//通过键盘输入控制是否应该擦除钢琴块以及是否加分
module erase_controller(
    input clk,//系统时钟
    input rst_n,//复位信号（低电平有效）
    input[7:0] v_enb,//允许出现钢琴块的轨道
    input[31:0]keycode,//当前读到的键盘码
    input level,//玩家选择的难度
    output reg [7:0]erase,//是否擦除某轨道上的钢琴块
    output reg [7:0]score//当前得分
    );
    reg [7:0] prev_enb = 8'b00000000;//上一次的出现钢琴块轨道的信号
    reg [7:0] detect;//是否允许检测某轨道来自键盘消除信号的标志
    reg [27:0] count0;
    reg [27:0] count1;
    reg [27:0] count2;
    reg [27:0] count3;
    reg [27:0] count4;
    reg [27:0] count5;
    reg [27:0] count6;
    reg [27:0] count7;
    reg [31:0]count_max;

   always@(level)
    begin
       if(level==0)  count_max<=32'd250000000;
       else count_max<=32'd125000000;
    end

always@(posedge clk or negedge rst_n)
begin
    if(!rst_n) begin erase <= 0; score <= 0; end
    else
    begin
        //轨道 0
        if(prev_enb[0] == 0 && v_enb[0] == 1) //当某轨道钢琴块产生时,
该轨道对应的检测信号置 1, 计数置 0, 擦除标志置 0
            begin detect[0] <= 1; count0 <= 0; erase[0] <= 0;end
        else
        begin
            //当接收到键盘对应消除按键时, 对应 erase 位置 1,分数加 1
            if(keycode[7:0]==8'h16 && detect[0]==1) begin erase[0] <= 1;
score <= score+1; end;
            //当钢琴块完全落下或消除键已经按下时, 对应轨道检测标志置 0
```

```
            if(count0==count_max || erase[0]==1) detect[0] <= 0;
            //正在检测阶段，count 每次加 1
            if(detect[0]==1) count0 <= count0+1;
        end
        //轨道 1
        if(prev_enb[1] == 0 && v_enb[1] == 1)
            begin detect[1] <= 1; count1 <= 0; erase[1] <= 0;end
        else
        begin
            if(keycode[7:0]==8'h1e && detect[1]==1) begin erase[1] <= 1;
score <= score+1; end;
            if(count1==count_max || erase[1]==1) detect[1] <= 0;
            if(detect[1]==1) count1 <= count1+1;
        end
        //轨道 2
        if(prev_enb[2] == 0 && v_enb[2] == 1)
            begin detect[2] <= 1; count2 <= 0; erase[2] <= 0;end
        else
        begin
            if(keycode[7:0]==8'h26 && detect[2]==1) begin erase[2] <= 1;
score <= score+1; end;
            if(count2==count_max || erase[2]==1) detect[2] <= 0;
            if(detect[2]==1) count2 <= count2+1;
        end
        //轨道 3
        if(prev_enb[3] == 0 && v_enb[3] == 1)
            begin detect[3] <= 1; count3 <= 0; erase[3] <= 0;end
        else
        begin
             if(keycode[7:0]==8'h25 && detect[3]==1) begin erase[3] <=
1;  score <= score+1; end;
             if(count3==count_max || erase[3]==1) detect[3] <= 0;
             if(detect[3]==1) count3 <= count3+1;
        end
        //轨道 4
        if(prev_enb[4] == 0 && v_enb[4] == 1)
            begin detect[4] <= 1; count4 <= 0; erase[4] <= 0;end
        else
        begin
            if(keycode[7:0]==8'h2e && detect[4]==1) begin erase[4] <= 1;
score <= score+1; end;
            if(count4==count_max || erase[4]==1) detect[4] <= 0;
            if(detect[4]==1) count4 <= count4+1;
        end
```

```
        //轨道 5
        if(prev_enb[5] == 0 && v_enb[5] == 1)
            begin detect[5] <= 1; count5 <= 0; erase[5] <= 0; end
        else
        begin
            if(keycode[7:0]==8'h36 && detect[5]==1) begin erase[5] <= 1;
score <= score+1; end;
            if(count5==count_max || erase[5]==1) detect[5] <= 0;
            if(detect[5]==1) count5 <= count5+1;
        end
        //轨道 6
        if(prev_enb[6] == 0 && v_enb[6] == 1)
            begin detect[6] <= 1; count6 <= 0; erase[6] <= 0; end
        else
        begin
            if(keycode[7:0]==8'h3d && detect[6]==1) begin erase[6] <= 1;
score <= score+1; end;
            if(count6==count_max || erase[6]==1) detect[6] <= 0;
            if(detect[6]==1) count6 <= count6+1;
        end
        //轨道 7
        if(prev_enb[7] == 0 && v_enb[7] == 1)
            begin detect[7] <= 1; count7 <= 0; erase[7] <= 0; end
        else
        begin
            if(keycode[7:0]==8'h3e && detect[7]==1) begin erase[7] <= 1;
score <= score+1; end;
            if(count7==count_max || erase[7]==1) detect[7] <= 0;
            if(detect[7]==1) count7 <= count7+1;
        end
    end
    prev_enb <= v_enb;//记录前一次轨道的使能信号
end

endmodule
```

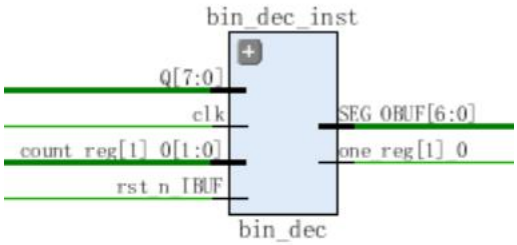
9. 二进制转十进制子系统

9.1 功能描述：将二进制数转换成十进制的个、十、百位。
9.2 实现方法：将二进制码转化为十进制 BCD 码，从低到高每四位分别作为输出的个位、十位、百位；
将二进制码转换为十进制 BCD 码的几个步骤（以 8bit 二进制码为例）：
(1)将二进制码左移一位（或者乘 2）
(2)找到左移后的码所对应的个，十，百位。

(3)判断在个位和十位的码是否大于5，如果是则该段码加3。
(4)继续重复以上三步直到移位8次后停止。
下图展示了将255的二进制转换为十进制的步骤：

Operation	Hundreds	Tens	Units	Binary	
HEX				F	F
Start				1 1 1 1	1 1 1 1
Shift 1			1	1 1 1 1	1 1 1
Shift 2			1 1	1 1 1 1	1 1
Shift 3			1 1 1	1 1 1 1	1
Add 3			1 0 1 0	1 1 1 1	1
Shift 4		1	0 1 0 1	1 1 1 1	
Add 3		1	1 0 0 0	1 1 1 1	
Shift 5		1 1	0 0 0 1	1 1 1	
Shift 6		1 1 0	0 0 1 1	1 1	
Add 3		1 0 0 1	0 0 1 1	1 1	
Shift 7	1	0 0 1 0	0 1 1 1	1	
Add 3	1	0 0 1 0	1 0 1 0	1	
Shift 8	1 0	0 1 0 1	0 1 0 1		
BCD	2	5	5		

9.3 RTL 图：



9.4 verilog 代码：

```
module bin_dec(clk,bin,rst_n,one,ten,hun);
input  [7:0] bin;//输入的二进制数
input  clk;//时钟
input rst_n;//复位信号（低电平有效）
output reg [3:0] one;//个位
output reg [3:0] ten;//十位
output reg [3:0] hun;//百位

reg [3:0] count = 0;//移位次数计数
reg [19:0]shift_reg=20'b00000000000000000000;

// 计数部分
always @ ( posedge clk or negedge rst_n )
begin
 if( !rst_n )
   count<=0;
```

```
 else if (count==8)
   count<=0;
 else
   count<=count+1;
end

// 二进制转换为十进制
always @ (posedge clk or negedge rst_n )
begin
  if (!rst_n)
       shift_reg=0;
  else if (count==0)
       shift_reg={12'b000000000000,bin};
  else if (count<=8)                  //实现 8 次移位操作
  begin
      if(shift_reg[11:8]>=5)          //判断个位是否>5, 如果是则+3
      begin
          if(shift_reg[15:12]>=5) //判断十位是否>5, 如果是则+3
          begin
              shift_reg[15:12]=shift_reg[15:12]+2'b11;
              shift_reg[11:8]=shift_reg[11:8]+2'b11;
              shift_reg=shift_reg<<1;  //对个位和十位操作结束后, 整体左移
          end
          else
          begin
              shift_reg[15:12]=shift_reg[15:12];
              shift_reg[11:8]=shift_reg[11:8]+2'b11;
              shift_reg=shift_reg<<1;
          end
      end
      else
      begin
          if(shift_reg[15:12]>=5)
          begin
              shift_reg[15:12]=shift_reg[15:12]+2'b11;
              shift_reg[11:8]=shift_reg[11:8];
              shift_reg=shift_reg<<1;
          end
          else
          begin
              shift_reg[15:12]=shift_reg[15:12];
              shift_reg[11:8]=shift_reg[11:8];
              shift_reg=shift_reg<<1;
          end
```

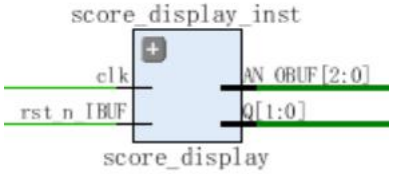
```
        end
    end
end

//输出赋值
always @ ( posedge clk or negedge rst_n )
begin
 if ( !rst_n )
  begin
    one<=0;
    ten<=0;
    hun<=0;
  end
 else if (count==8)  //此时 8 次移位全部完成，将对应的值分别赋给个,十,百位
  begin
    one<=shift_reg[11:8];
    ten<=shift_reg[15:12];
    hun<=shift_reg[19:16];
  end
 else;
end
endmodule
```

10.七段数码管显示子系统

10.1 功能描述：将输入的十进制个、十、百位数字显示在数码管上。

10.2 实现方法：与平时实验做的点亮七段数码管不同，该系统要求在不同数码管上显示不同数字。查找资料发现，多个数码管的连接并不是把每个数码管都独立的与可编程逻辑器件连接，而是把所有的 LED 管的输入连在一起。每次向 LED 写数据时，通过片选选通其中一个 LED，然后把数据写入该 LED 管，因此每个时刻只有一个 LED 管是亮的。所以为了能持续看见 LED 上面的显示内容，必须对 LED 管进行扫描，即依次并循环地点亮各个 LED 管。利用人眼的视觉暂停效应，在一定的扫描频率下，人眼就会看见好几个 LED 一起点亮。扫描频率大小必须合适才能有很好的效果。如果太小，而每个 LED 开启的时间大于人眼的视觉暂停时间，那么会产生闪烁现象。而扫描频率太大，则会造成 LED 的频繁开启和关断，大大增加 LED 功耗（开启和关断的时刻功耗很大）。经过测试得出，扫描频率选在 1000Hz 比较合适。

10.3 RTL 图：



10.4 verilog 代码：

```
module score_display(score,clk,rst_n,one,ten,hun,seg,an);
input [7:0] score;//要显示的得分(最多 255)
input clk;//系统时钟
input rst_n;//复位信号 (低电平有效)
input [3:0] one;
input [3:0] ten;
input [3:0] hun;
output reg [6:0] seg;//单个数码管显示信号
output reg [7:0] an;//数码管编号信号

wire clk_out;//提供扫描数码管的 1000HZ 时钟
divider #(100000/2)divider1000HZ(.clk(clk),.clk_out(clk_out));

reg [1:0] count;//从 0 数到 3, 依次点亮三个数码管
initial count=0;
always@(posedge clk_out)
begin
    if(count==3)count<=0;
    else count<=count+1;
end

reg [3:0]digit;//截取分数百、十、个位数, 分个显示
always @(count)
      if(!rst_n) begin digit <= 0;  an<=8'b1111_1111; end
      else
      begin
      case(count)
         0:begin digit <= one; an<=8'b1111_1110; end
         1:begin digit <= ten; an<=8'b1111_1101; end
         2:begin digit <= hun; an<=8'b1111_1011; end
         default:begin digit <= 0;  an<=8'b1111_1111; end
      endcase
      end

always @(digit)
      case(digit) //按照 gfedcba 的顺序表明各管电平, 其中低电平表示该管发光
          0:seg = 7'b1000000;
          1:seg = 7'b1111001;
          2:seg = 7'b0100100;
          3:seg = 7'b0110000;
          4:seg = 7'b0011001;
          5:seg = 7'b0010010;
          6:seg = 7'b0000010;
          7:seg = 7'b1111000;
```

```
            8:seg = 7'b0000000;
            9:seg = 7'b0010000;
            default: seg = 7'b0000000;
        endcase

endmodule
```

五、测试模块建模

1.键盘输入模块仿真测试代码

```
//ps2 键盘模块测试，依次发送“1”的通码，断码 F0 和“1”的通码，波形按预期显示
`timescale 1ns / 1ps
module ps2_tb;
//系统输入
    reg clk;//系统输入时钟
    reg rst;//系统复位
    reg ps2_clk;//ps2 的时钟
    reg ps2_data_in;//ps2 的数据
//系统输出
    wire [7:0] ps2_data_out;//按键的通、断码
    always#5 clk=~clk;//100MHZ 的时钟

    initial
    begin
    clk = 1;
    rst = 1;
    ps2_clk = 1;
    ps2_data_in = 1;

    #100 rst=0;
    //数字“1”的通码
    ps2_data_in=0;//起始位"0"
    #60 ps2_clk=0;
    #120 ps2_clk=1;
    #60 ps2_data_in=1;//"1"
    #60 ps2_clk=0;
    #120 ps2_clk=1;
    #60 ps2_data_in=0;//"0"
    #60 ps2_clk=0;
    #120 ps2_clk=1;
    #60 ps2_data_in=0;//"0"
    #60 ps2_clk=0;
```

```
#120 ps2_clk=1;
#60 ps2_data_in=1;//"1"
#60 ps2_clk=0;
#120 ps2_clk=1;
#60 ps2_data_in=0;//"0"
#60 ps2_clk=0;
#120 ps2_clk=1;
#60 ps2_data_in=1;//"1"
#60 ps2_clk=0;
#120 ps2_clk=1;
#60 ps2_data_in=1;//"1"
#60 ps2_clk=0;
#120 ps2_clk=1;
#60 ps2_data_in=0;//"0"
#60 ps2_clk=0;
#120 ps2_clk=1;
#60 ps2_data_in=1;//奇偶校验位"1"
#60 ps2_clk=0;
#120 ps2_clk=1;
#60 ps2_data_in=1;//停止位"1"
#60 ps2_clk=0;
#120 ps2_clk=1;
#2000
//断码中的 f0
ps2_data_in=0;//起始位"0"
#60 ps2_clk=0;
#120 ps2_clk=1;
#60 ps2_data_in=0;//"0"
#60 ps2_clk=0;
#120 ps2_clk=1;
#60 ps2_data_in=0;//"0"
#60 ps2_clk=0;
#120 ps2_clk=1;
#60 ps2_data_in=0;//"0"
#60 ps2_clk=0;
#120 ps2_clk=1;
#60 ps2_data_in=0;//"0"
#60 ps2_clk=0;
#120 ps2_clk=1;
#60 ps2_data_in=1;//"1"
#60 ps2_clk=0;
#120 ps2_clk=1;
#60 ps2_data_in=1;//"1"
#60 ps2_clk=0;
```

```
#120 ps2_clk=1;
#60 ps2_data_in=1;//"1"
#60 ps2_clk=0;
#120 ps2_clk=1;
#60 ps2_data_in=1;//"1"
#60 ps2_clk=0;
#120 ps2_clk=1;
#60 ps2_data_in=1;//奇偶校验位"1"
#60 ps2_clk=0;
#120 ps2_clk=1;
#60 ps2_data_in=1;//停止位"1"
#60 ps2_clk=0;
#120 ps2_clk=1;
#2000
//数字“1”的通码
ps2_data_in=0;//起始位"0"
#60 ps2_clk=0;
#120 ps2_clk=1;
#60 ps2_data_in=1;//"1"
#60 ps2_clk=0;
#120 ps2_clk=1;
#60 ps2_data_in=0;//"0"
#60 ps2_clk=0;
#120 ps2_clk=1;
#60 ps2_data_in=0;//"0"
#60 ps2_clk=0;
#120 ps2_clk=1;
#60 ps2_data_in=1;//"1"
#60 ps2_clk=0;
#120 ps2_clk=1;
#60 ps2_data_in=0;//"0"
#60 ps2_clk=0;
#120 ps2_clk=1;
#60 ps2_data_in=1;//"1"
#60 ps2_clk=0;
#120 ps2_clk=1;
#60 ps2_data_in=1;//"1"
#60 ps2_clk=0;
#120 ps2_clk=1;
#60 ps2_data_in=0;//"0"
#60 ps2_clk=0;
#120 ps2_clk=1;
#60 ps2_data_in=1;//奇偶校验位"1"
#60 ps2_clk=0;
```

```
    #120 ps2_clk=1;
    #60 ps2_data_in=1;//停止位"1"
    #60 ps2_clk=0;
    #120 ps2_clk=1;
    end

    ps2
ps2_inst(.clk(clk),.kclk(ps2_clk),.kdata(ps2_data_in),.rst(rst),.LED(ps2_da
ta_out));

    initial #13000 $finish;
endmodule
```

2. 二进制转十进制仿真测试代码

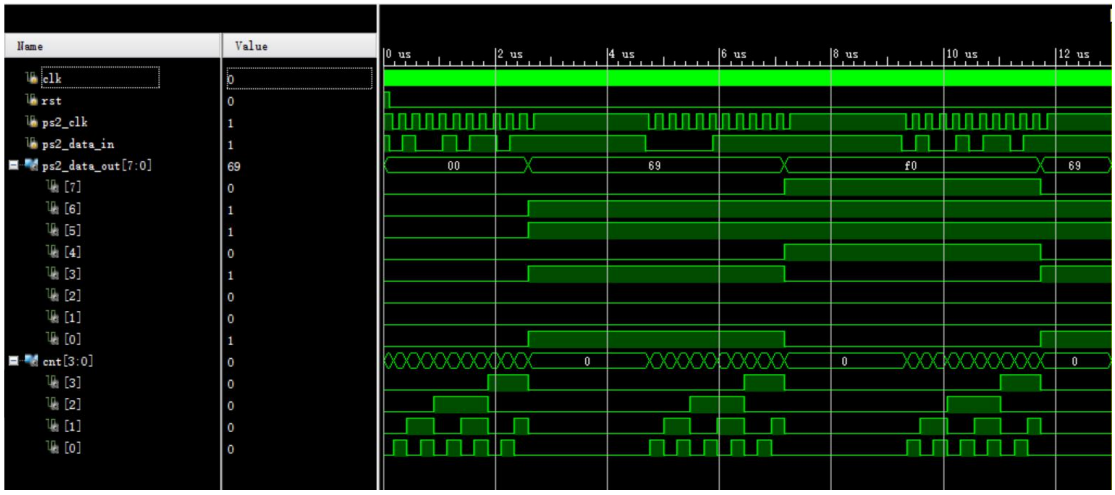
```
module bin_dec_tb();
reg [7:0]bin;
reg clk;
reg rst_n;
wire[3:0] one;
wire[3:0] ten;
wire[3:0] count;
wire[1:0] hun;
wire [17:0] shift_reg;

initial clk=0;
always#5 clk=~clk;

initial
begin
bin=8'd56;
#80 bin=8'd88;
#80 bin=8'd100;
#80 bin=8'd0;
end

initial
begin
rst_n=1;
end

initial #240 $finish;
bin_dec
bin_dec_inst(.bin(bin),.clk(clk),.rst_n(rst_n),.one(one),.ten(ten),.count(c
ount),.hun(hun),.shift_reg(shift_reg));
```

```
endmodule
```

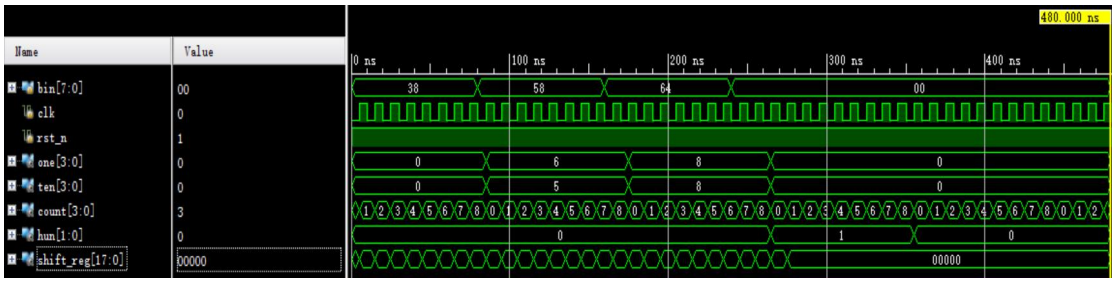
六、实验结果

（该部分可截图说明，可包含 logisim 逻辑验证图、modelsim 仿真波形图、以及下板后的实验结果贴图）

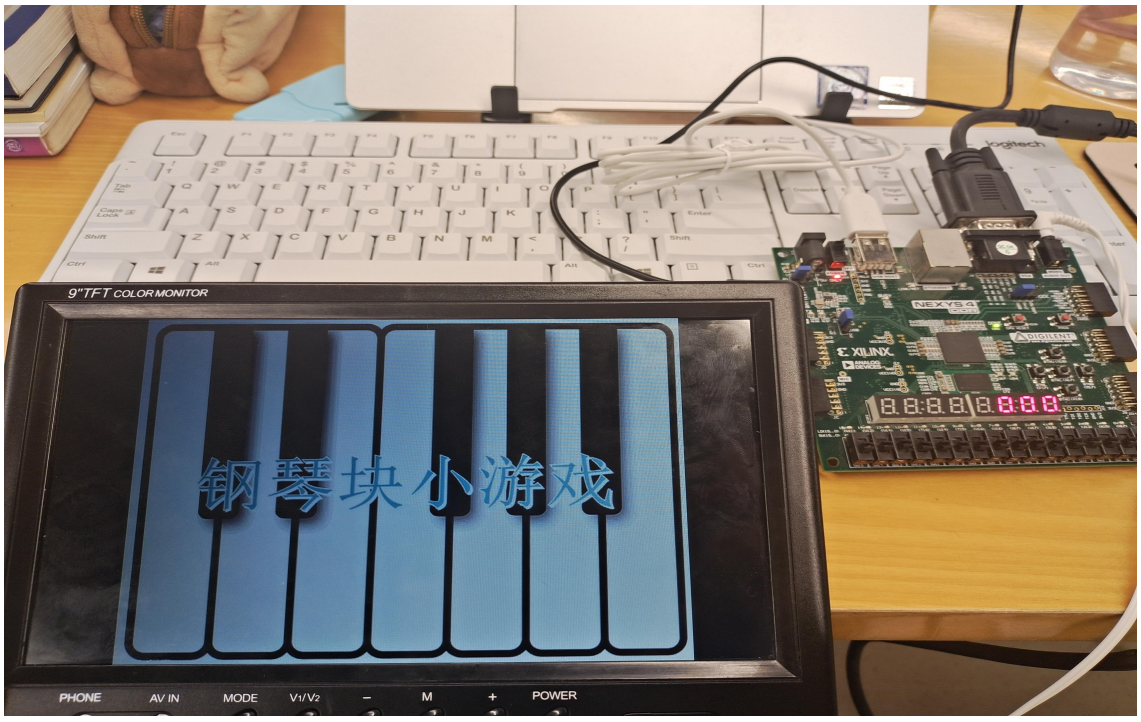
1. 键盘输入模块仿真波形图

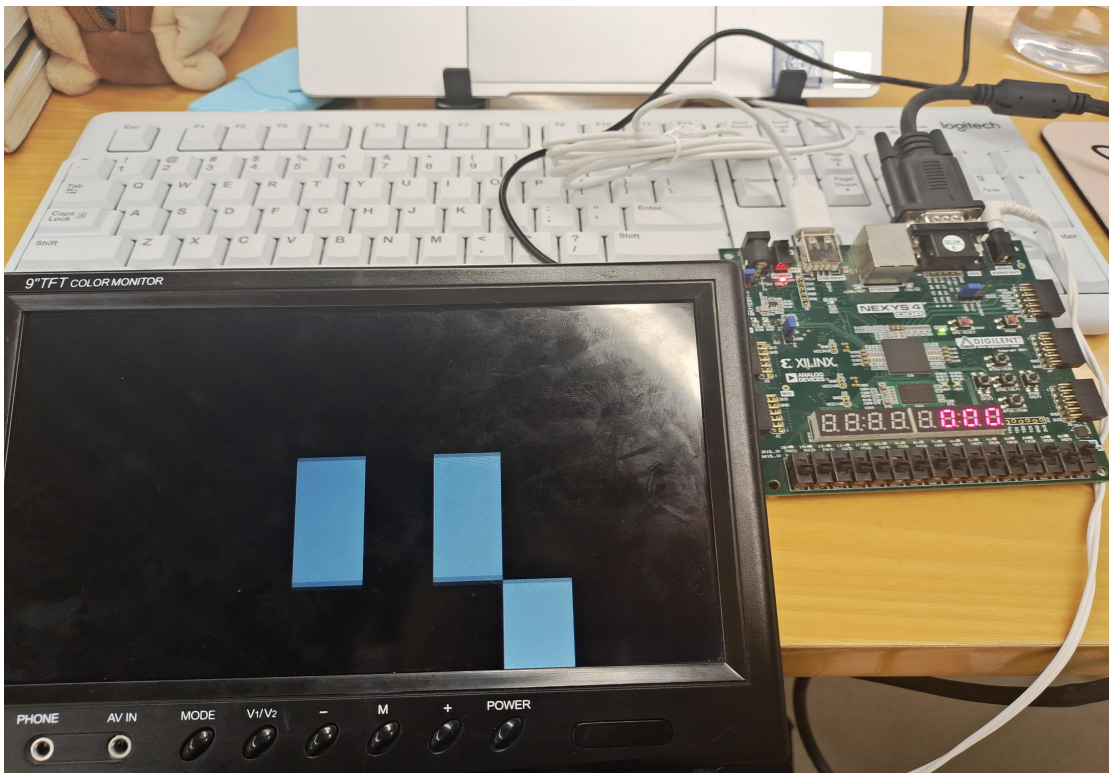


2. 二进制转十进制仿真波形图



2. 下板后的实验结果贴图





七、结论

无论是各个子模块，还是整个数字系统，经测试均能正常工作。钢琴块小游戏是一款非常简单有趣的游戏，既能体会音乐的魅力，又能感受速度的激情。

八、心得体会及建议

1. **心得体会：**对于一个较大的数字系统的设计，我们首先不应该关注过多细节，而应该从全局考虑，从数字系统要实现的功能出发将其划分成若干个相对独立的子系统，弄清楚各个子系统之间输入输出信号的关系。子系统划分越明确，实现起来越容易。在设计数字系统时，我们采用自顶向下的方法，而在实现的过程中，我们应该使用自下而上的方法。对每个模块分别编写 verilog 代码和 testbench 代码进行测试，必要时甚至可以进行下板实验，如本次实验过程中多次使用 LED 等测试信号是否正常。当确保每个模块都能正常实现相应功能时再将其拼接成大的系统。否则可能会在出现问题时难以找出问题所在，或者因为一小处改动而使整个系统瘫痪。

2. **本课程的收获：**通过本课程，我学会了从硬件的角度理解问题。硬件编程与软件编程最大的不同便在于硬件编程不单单需要考虑程序实现的功能，还要考虑编写的代码对应什么电路。比如在本次实验过程中，就多次遇到赋值冲突的问题，在多个 always 块中或在同一个 always 块中的不同 if 语句中对同一个变量进行赋值操作，会使得硬件电路不知道先处理哪个赋值的操作，也不能构建硬件电路。

3. **反思：**此次综合实验设计最难的地方在于时钟的控制，首先，根据玩家选择的游戏难度，播放音乐的频率不同，我设置的是简单模式下 1/2s 一个音符，即频率为 2HZ；挑战模式下 1/4s 一个音符，即频率为 4HZ。而钢琴块的下落速度也得随着音符出现的频率改变，本来的想法是根据人眼分辨率每 100ms 更新一次钢琴块的位置，但会遇到不到整数个像素点就要刷新一次的问题，而像素点是图像显示的最小单元，故无法实现。之后改成了每次更新一个像素点的位置，并根据玩家选择的难度决定多长时间更新一次。同时，每个外围模块本身需要不同的时钟，键盘有自己的控制时钟，而 VGA 也需要将系统时钟分频到 25HZ 才能正常使用。本次综合实验设计基本达到了预想的效果，但仍然有很多不足，比如游戏里保存的歌曲其实是写在代码里的，如果想要有更多歌曲可供选择，还需要一个一个音符记录它们的音调与音高，既增加了无意义的代码长度又浪费时间，且歌曲的质量不能保证。而由于开发板存储容量的限制，不能同时存储多张大图，所以只能将游戏初始界面和结束界面设为同一个。后期如果有时间会尝试一下用 SD 卡导入歌曲和图片，弥补这次大作业的遗憾。

4. **对本课程的建议：**建议在采购外围模块时谨慎考虑质量问题以及版本型号问题，如本次领取的 PS2 键盘大多都都不能使用，而我还一直以为是代码的问题，浪费了很多时间。以及希望定期对开发板各个接口进行质量检查。保证硬件的完好可操作才能让同学们将更多时间和精力花在程序设计中。

5. **结合数字芯片设计国内外现状谈自己的认识与体会：**

在芯片产业链中，中国与发达国家的差距是全方位的。设计方面，华为海思和紫光展锐分列国内前两名。目前，两家公司在不少领域已是世界领先水平，但一个巨大的问题是，其架构授权的核心都被外人掌握。目前，国内仅有中科院的龙芯和总参谋部的申威拥有自主架构，前者用于北斗导航，后者用于神威超级计算机，民用领域基本是空白。设备和材料是又一大短板。制造芯片的三大设备光刻机、蚀刻机和薄膜沉积，国内仅中微半导体的介质蚀刻机能跟上行业节奏，其 7 纳米设备已入围台积电名单。国内造不好高端芯片，有外部因素，也有自身原因。而芯片设计不光是现代高科技产业的基础，更是支撑和保障国家安全的战略性、基础性和先导性产业，而且重要性会越来越大。要想突破这些卡脖子技术，我们不仅需要加大科技投入，还需要加强国家之间的交流合作。抓住机遇，迎难而上，是数字芯片设计的出路所在，也是每一个高新技术产业从业人员的职责所在。