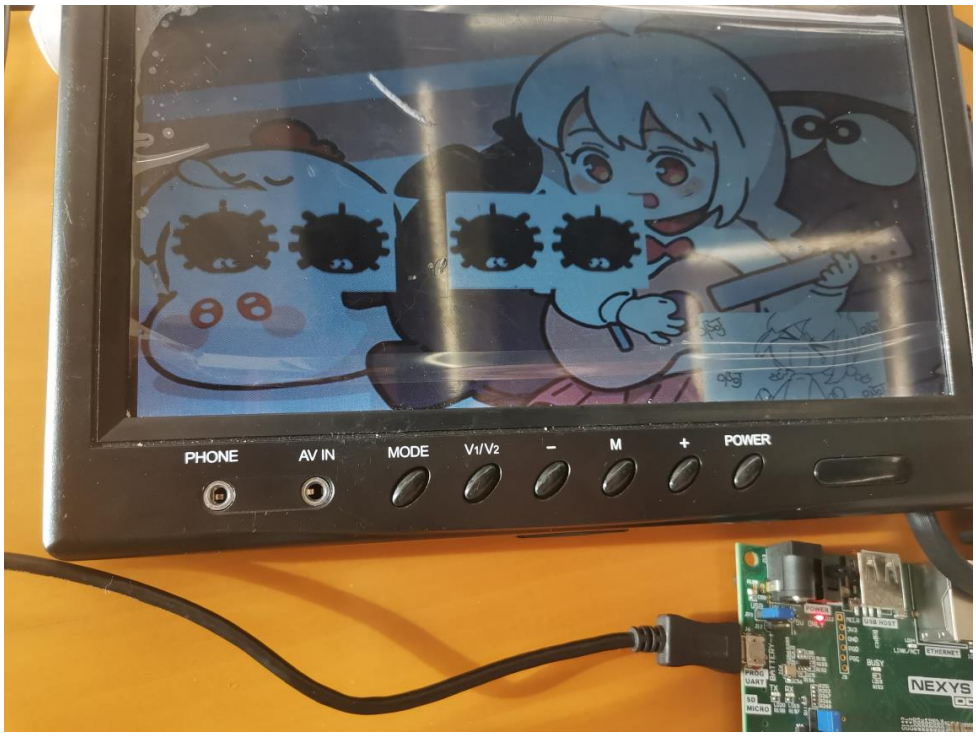
同济大学计算机系

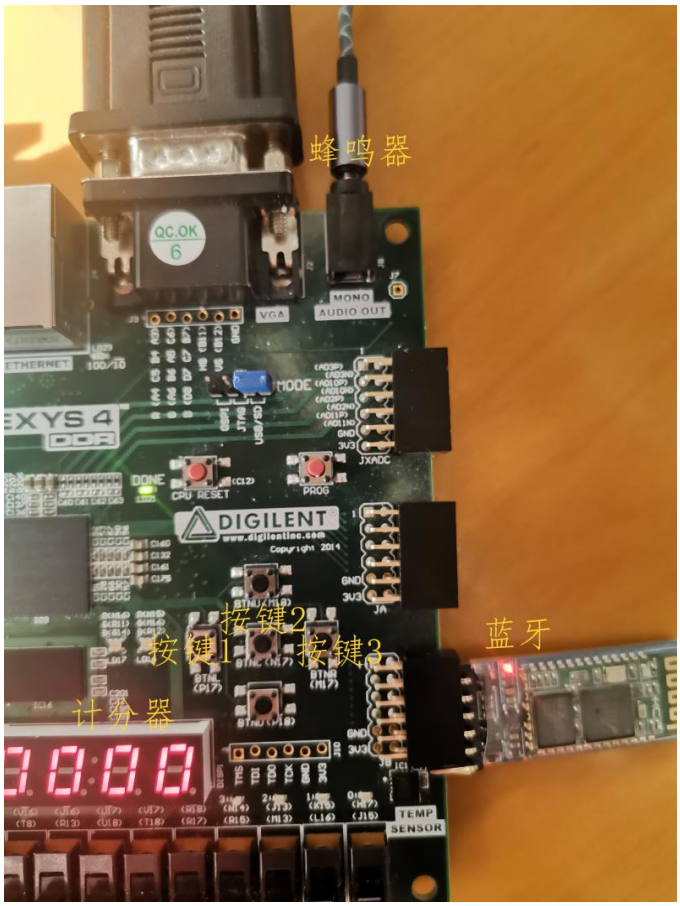
数字逻辑课程综合实验报告

目录

一、实验内容

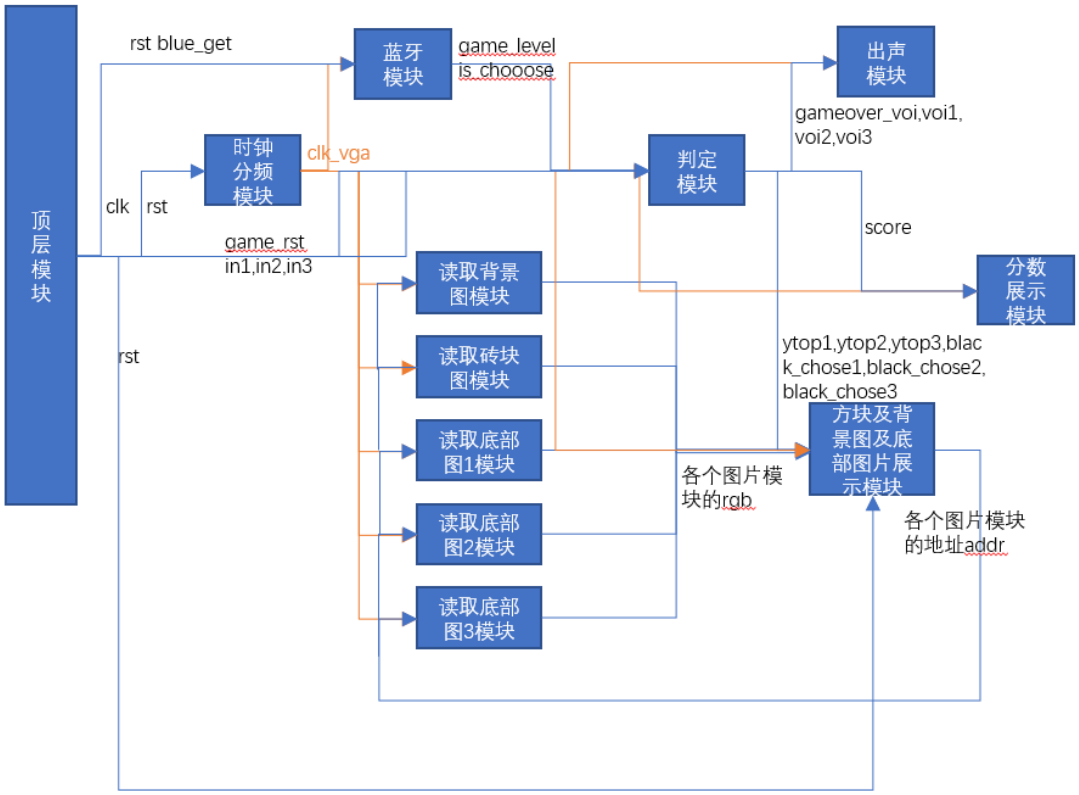
基于 VGA，蓝牙，7 短数码管，蜂鸣器完成的与别踩白块游戏规则相同的游戏。





二、数字系统总框图

1.总框图



2.子系统模块功能概述

时钟分频模块：将 100M 的时钟分频成 25M。

蓝牙模块：读取手机传来的信号，然后将读取到的信号输出。

各种读取图片的模块：根据传来的对应每个模块的地址，给出对应的 rgb 值。

判定模块：存储此时运动的 3 个蜘蛛块的最上面的 y 坐标，并根据玩家的输入来调整 y 坐标，+1，或者回到最顶层，还输出 4 个是否发声的判定，以及 3 个是否出现底层的图片的判定。

方块及背景图及底部图片展示模块：根据输入过来的各个图片的 rgb 值，以及是否出现该图片的判定，以及根据 3 个蜘蛛块的 y 坐标展示蜘蛛块。

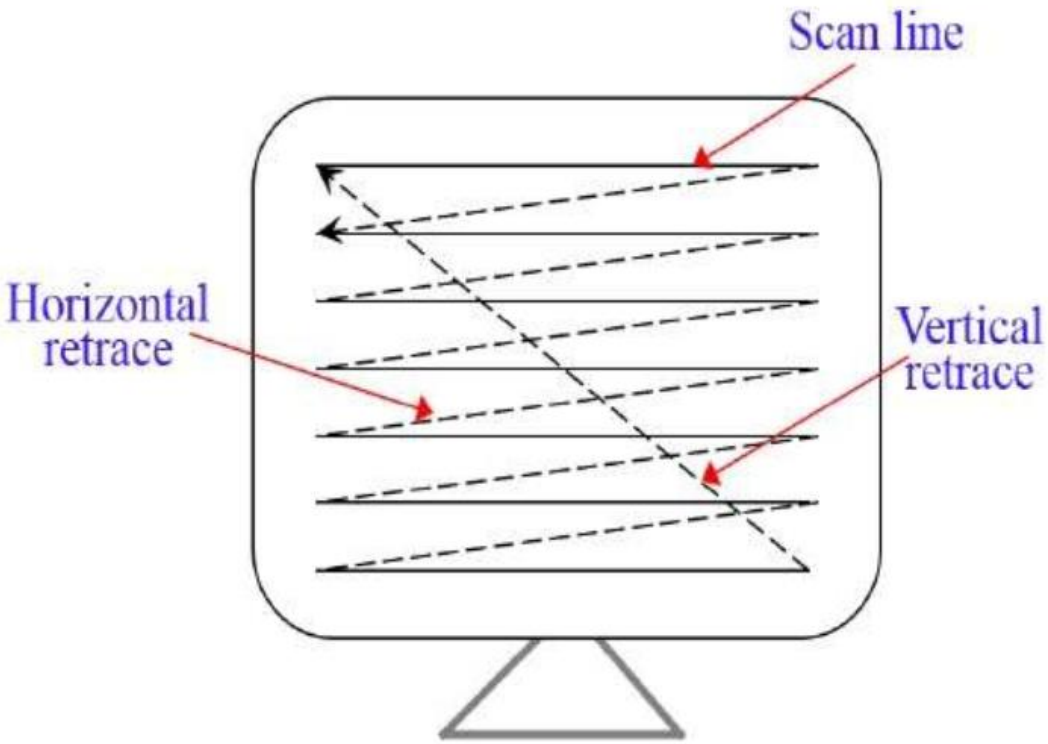
分数展示模块：根据传过来的分数对其进行对应的展示。

3.外部配件原理

VGA 使用原理

VGA 在连接过了之后会不断地进行扫描，然后它扫描时，宽度 x 是 800，高度是 525，然后 x 方向有效的位置是 144-784，总共 640 位，然后 y 方向有效的是 35-515，然后到达这一个区域的时候，然后给它赋对应的 rgb 值，才会在 VGA 上对应显示出来。

它的扫描路径如下：

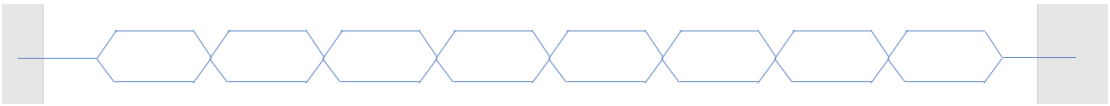


它的时序如下：

只有在 Hor Active Video 区域时才能够在 VGA 对应的区域显示出来。

蓝牙使用原理

这里的蓝牙是 9600 波特，即每秒钟会传输 9600 个数据，然后使用的时钟是 25Mhz 的频率，所以用 25M/9600=2605，也就是每 2605 个时钟沿传输一个数据，然后读取的话是在这个 2605 个时钟沿的中间位置读取的，因为这时候的信号更稳定，下图就是手机传给蓝牙信号的时序图，因为是传给蓝牙对应数字的 ASCII 码，所以总共有 8 位，然后我们在检测到信号边缘到来的时候，开始计数，在每个波在中间位置进行读取，读取 8 个信号后结束。



蜂鸣器的使用原理：

蜂鸣器是通过传递过来的方波信号的频率来控制振动装置，不同的频率对应不同的声音，原理如下：

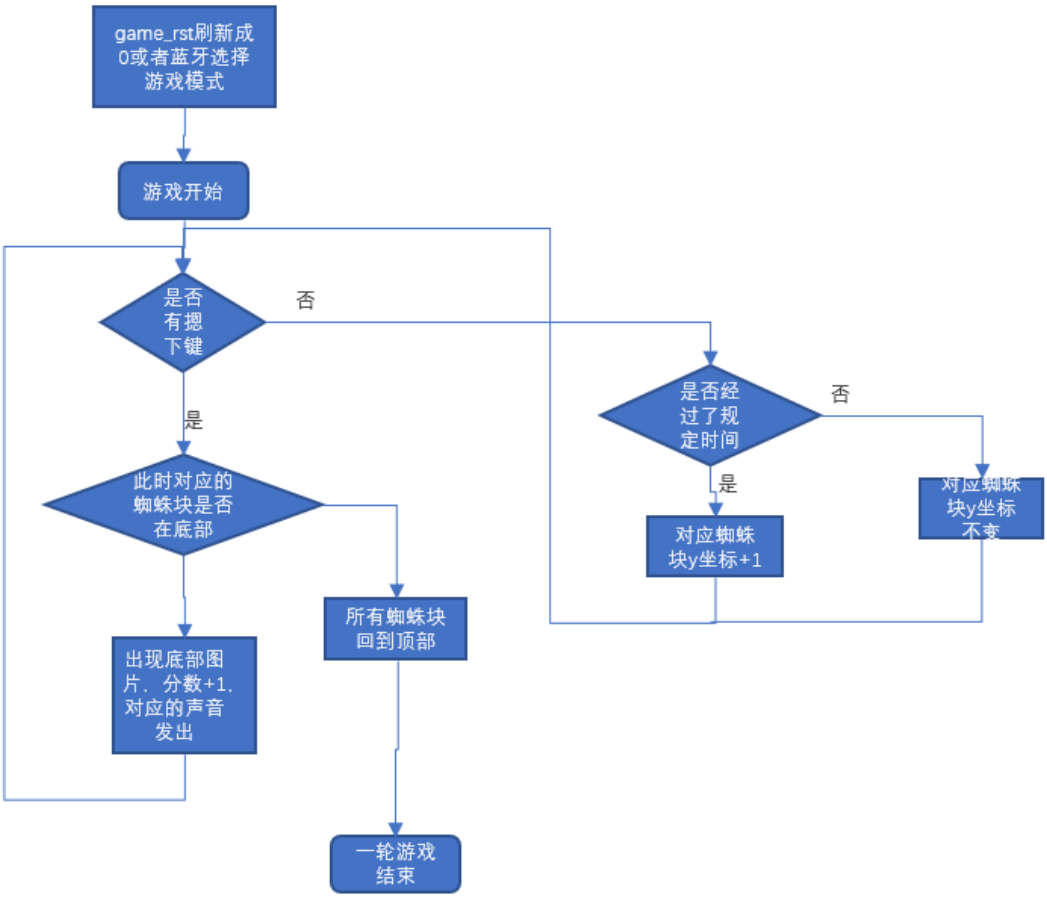
下面是不同的频率对应的不同的音阶：

音名	频率/Hz	音名	频率/Hz	音名	频率/Hz
低音 1	261.63	中音 1	532.25	高音 1	1046.50
低音 2	293.67	中音 2	587.33	高音 2	1174.66
低音 3	329.63	中音 3	659.25	高音 3	1318.51
低音 4	349.23	中音 4	698.46	高音 4	1396.92
低音 5	391.99	中音 5	783.99	高音 5	1567.98
低音 6	440	中音 6	880	高音 6	1760
低音 7	493.88	中音 7	987.76	高音 7	1975.52

三、系统控制器设计

（要求画出所设计数字系统的 ASM 流程图，列出状态转移真值表。由状态转移真值表，求出系统控制器的次态激励函数表达式和控制命令逻辑表达式，并用 Logisim 画出系统控制器逻辑方案图。）

1.整个游戏的 ASM 大致流程图



因为整个系统涉及多多个状态机以及状态转移真值表，列出整合出来的状态转移真值表比较困难，所以就主要列出各个分立的状态转移真值表。

2.对于各个系统控制器的状态转移真值表

1)判断一个键是否摁下

因为要实现只有当摁下的那一个上升沿才有效，不能使得摁着不放手后一直都有效，所以不能简单的通过那一个按键是否为 1，又因为游戏逻辑的判断是在 clk_vga 的上升沿所以摁下的那一瞬间极有可能游戏没有进行刷新判定，所以也不能够通过

状态转移真值表：

1 代表摁下,，0 代表没有被摁下，2 代表按键被摁着

现态		次态		转移条件
in1t	in1u	in1t’	in1u’	
0	0	0	0	clk_vga 上升
0	0	1	0	in1=1 即按键摁下
1	0	1	1	clk_vga 上升沿
1	1	1	2	clk_vga 上升沿
1	2	1	2	clk_vga 上升沿
1	2	0	2	in1=0 即按键松开
0	2	0	0	clk_vga 上升沿

有 3 个对应的这种判断摁下的模块，其余 2 个与它类似，不再赘述

2) 一个蜘蛛块的y 坐标的变换

根据不同的输入条件，对应那个蜘蛛块的最上方的 y 坐标要相应变化

状态转移真值表：

0 代表游戏开始，2 代表游戏结束，1 代表对应的 y+1，3 代表维持不变

现态	次态	转移条件
sta1	sta1’	
1	0	clk_vga 上升沿及 game_rst||game_level_choose 即游戏刷新了，或者重新选择了游戏模式
2		
3		
1	2	clk_vga 上升沿及 yt1>=515||sta1==2 &&!（game_rst||game_level_choose） 即不满足上述条件并且蜘蛛块移动到了屏幕外面或者游戏已经结束
2		
1	0	clk_vga 上升沿及 in1u==1&&yt1>=415 &&!(yt1>=515||sta1==2&&!（game_rst||game_level_choose）) 即不满足上述条件并且对应的按键按下了并且蜘蛛块也在底部
3		
1	3	clk_vga 上升沿及 count!=0 &&!(in1u==1&&yt1>=415&&!(yt1>=515||sta1==2 &&!（game_rst||game_level_choose）） 即不满足上述条件并且没有达到预定的计数值
3		
3	1	clk_vga 上升沿及 count==0 &&!(in1u==1&&yt1>=415&&!(yt1>=515||sta1==2 &&!（game_rst||game_level_choose）） 即不满足上述条件并且经过了预定的值（时间）

对应的其他 2 个蜘蛛块也是类似的，不再赘述

3)底部图片的展示

对应不同的输入，下方图片有不同的转移状态来决定是否展示图片

状态转移真值表：

0：不展示　1：展示

现态	次态	转移条件
b_c_sta1	b_c_sta1’	
0	0	clk_vga 上升沿及 sta1==2||sta2==2||sta3==2||b_c_count1>=12000000 即游戏结束了或者对应展示的时间达到预定值
1		
0	1	clk_vga 上升沿及 in1u==1||b_c_sta1==1 &&(sta1==2||sta2==2||sta3==2||b_c_count1>=12000000) 即不满足上述条件并且对应按键按下了或者依旧在展示图片
1		

4)声音的输出选择

对应不同的状态和输入，展示不同的声音，期中主要的是对游戏结束时的音的展示，因为如果将一轮游戏结束转变为对应的游戏结束声音，那么因为游戏结束的状态一直维持着，所以游戏结束结束的声音状态就会一直是对应的发声状态，所以需要用状态机来控制。

游戏结束声音的状态转移真值表：

1：出声　　0、2：不出声

现态	次态	转移条件
gameover_voi	gameover_voi’	
2	0	clk_vga 上升沿及 sta1!=2&&sta2!=2&&sta3!=2 即游戏开始了
2	2	clk_vga 上升沿及 !(sta1!=2&&sta2!=2&&sta3!=2) 即游戏一直未开始，处于结束状态
1	2	clk_vga 上升沿及 !(sta1!=2&&sta2!=2&&sta3!=2) 即游戏一直未开始，处于结束状态
0	1	clk_vga 上升沿及 (sta1==2||sta2==2||sta3==2) 即游戏结束
0	0	不满足上述条件时

游戏按键声音：

游戏按键声音的状态转移真值表：

1：出声　0：不出声

现态	次态	转移条件
voi1	voi1’	
0	1	clk_vga 上升沿及 in1u==1 即按下了对应的按键
1	0	clk_vga 上升沿
0	0	clk_vga 上升沿及 in1u!=1

5)蓝牙模块内部是否读取到有效信号的标记

对于蓝牙模块，当它获取到了一次有效信号之后需要传递出去，如果不用状态机的话会一个传递出去，不利于程序的运行

1：读取完了 0：没读取

现态	次态	转移条件
is_chooose	is_chooose’	
0	1	clk_vga 上升沿及 count_2==8 即这时已经读取完了 8 个信号
0	0	clk_vga 上升沿及 count_2!=8 即这时还没有读取信号或者 还没有读取完了 8 个信号
1	0	clk_vga 上升沿 即传递标记出去后就恢复原 状

6)voice 模块内部的声音输出状态

对应多个标记，但是一次只能选择一个输出，并且触发了输出以后，需要在一定的时间内又停止，所以需要状态机来调整。

0：不响 1、2、3、4 响对应不同的频率的声音

现态	次态	转移条件
sta	sta’	
0	0	clk_vga 上升沿及 wait_time>=22000000 即已经响过了对应的时间
1		
2		
3		
4		
1	1	clk_vga 上升沿及 wait_time<22000000 即这一次还没响完
0	1	clk_vga 上升沿及 voi1==1 即需要开始响了
2	2	clk_vga 上升沿及 wait_time<22000000 即这一次还没响完
0	2	clk_vga 上升沿及 voi2==1 即需要开始响了
3	3	clk_vga 上升沿及 wait_time<22000000 即这一次还没响完
0	3	clk_vga 上升沿及 voi3==1

		即需要开始响了
4	4	clk_vga 上升沿及 wait_time<22000000 即这一次还没响完
0	4	clk_vga 上升沿及 voi4==1 即需要开始响了
0	0	clk_vga 上升沿及 wait_time<22000000 &&(voi1!=1&& voi2!=1&& voi3!=1&& voi4!=1) 即不需要发声的时间

四、子系统模块建模

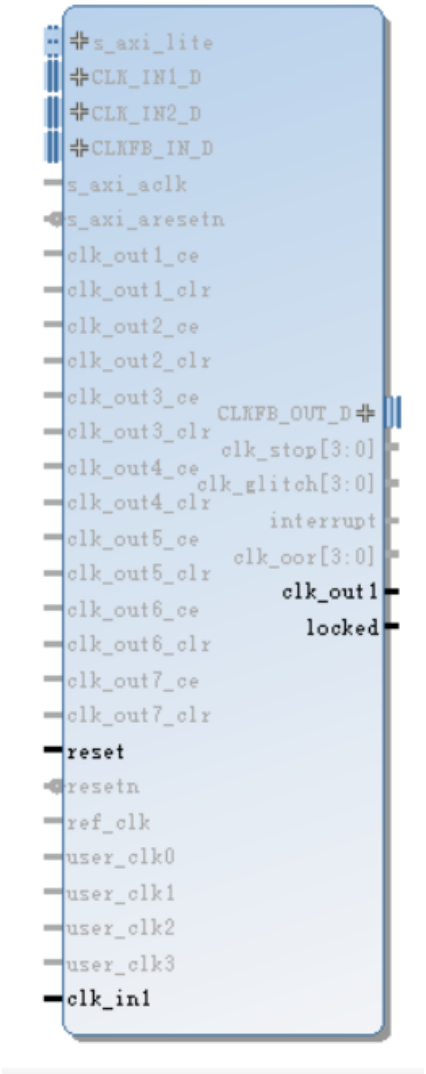
（该部分要求对实验中的所有子系统模块进行描述，给出各子系统的功能框图及接口信号定义，并列出各模块建模的 verilog 代码）

运用 IP 核器件：

1.分频器

将接口 clk 的 100M 的时钟分频成 25M 以 clk_vga 输出，对应 rst 为 1 时直接复位。

接口信号图：



2.图像内存读取

在使能端 ena 有效时，根据传来的地址 addra，将对应的 rgb 值赋值给 douta 传出。

接口信号图：

本程序当中调用了多个 Block Memory Generator，接口定义都是类似的，就是对应传出的 rgb 值位数有些不一样

3.蓝牙模块

需要一个时钟还需要一个复位键，以及蓝牙传输过来的数据，然后输出需要的读取到的信号，并且有一个是否获取了有效蓝牙信号的标记。

代码如下：

```
module bluetooth(
    input clk,
    input rst,
    input get,//读取的信号
    output reg [7:0] out=49,//输出信号，对应的是 ASCII 码
    output reg is_chooose=0
);
parameter bps=2605;//对应 9600 波特，用 25M/9600
reg [14:0] count_1;//每一位中的计数器
reg [3:0] count_2;//每一组数据的计数
reg buffer_0,buffer_1,buffer_2;//除去滤波
wire buffer_en;//检测边沿
reg add_en;//加法使能信号

always @ (posedge clk)
begin
    if(rst)
    begin
        buffer_0<=1;
        buffer_1<=1;
        buffer_2<=1;
    end
    else
    begin
        buffer_0<=get;
        buffer_1<=buffer_0;
        buffer_2<=buffer_1;
    end
```

```
end

assign buffer_en=buffer_2&~buffer_1;//1 说明检测到信号边缘

always @ (posedge clk)
begin
    if(rst)
    begin
        add_en<=0;
    end
    else if(buffer_en)
    begin
        add_en<=1;//说明可以开始相加
    end
    else if(add_en&&count_2==8&&count_1==bps-1)//已经读取到了对
应的 8 个字符
    begin
        add_en<=0;
    end
end

always @ (posedge clk)
begin
    if(rst)
    begin
        count_1<=0;
    end
    else if(add_en)
    begin
        if(count_1==bps-1)//读取了一个字的信号
        begin
            count_1<=0;
        end
        else
```

```
        begin
            count_1<=count_1+1;
        end
    end
end

always @ (posedge clk)
begin
    if(rst)
    begin
        count_2<=0;
    end
    else if(add_en&&count_1==bps-1)//读取了一个字的信号
    begin
        if(count_2==8)//读取完了 8 个
        begin
            count_2<=0;
        end
        else
        begin
            count_2<=count_2+1;
        end
    end
end

always @ (posedge clk)
begin
    if(rst)
    begin
        out<=0;
    end
    else if(add_en&&count_1==bps/2-1&&count_2!=0)//到中间的时候
进行数据读取
    begin
```

```
        out[count_2-1]<=get;
    end
end

always @(posedge clk)
begin
    if(count_2==8&&is_chooose==0)//读取了 8 个信号说明读取完毕了
    begin
        is_chooose<=1;
    end
    else if(is_chooose==0)
    begin
        is_chooose<=0;
    end

    if(is_chooose==1)
    begin
        is_chooose<=0;
    end
end

endmodule
```

4.出声模块

需要一个时钟，然后还有 4 个不同的声音的标记，然后一个然后一个输出声音，对应不同的标记输出不同的声音。

代码：

```
module gameVoice(
    input clk_vga,
    output reg voi,//输出的声音
    input [1:0]gameover_voi,//游戏结束声音
    input voi1,//3 个不同的声音
    input voi2,
    input voi3
);
reg [20:0]cnt=0;//有关不同音调的频率
reg [3:0]sta=0;//状态机
reg [28:0]wait_time=0;//等待时间
always @(posedge clk_vga)
begin
    if(wait_time>=22000000)
        sta<=0;
    else if(gameover_voi==1||sta==1)
        sta<=1;
    else if(voi1==1||sta==2)
        sta<=2;
    else if(voi2==1||sta==3)
        sta<=3;
    else if(voi3==1||sta==4)
        sta<=4;
    else
        sta<=0;
end
always @(posedge clk_vga) //相加计算需要出声的时间
begin
```

```
    if(sta==0)
        wait_time<=0;
    else  begin
        wait_time<=wait_time+1;
        if(wait_time>=22000000)
            wait_time<=0;
    end
end
always @(posedge clk_vga)//不同的状态输出不同的声音
begin
    case (sta)
        0:
        begin
            voi<=0;
            cnt<=0;
        end
        1:
        begin
            if(cnt<=47709)
                voi<=1;
            else
                voi<=0;
            if(cnt>=95419)
                cnt<=0;
            else
                cnt<=cnt+1;
        end
        2:
        begin
            if(cnt<=21294)
                voi<=1;
            else
                voi<=0;
            if(cnt>=42589)
```

```
                cnt<=0;
            else
                cnt<=cnt+1;
        end
        3:
        begin
            if(cnt<=18968)
                voi<=1;
            else
                voi<=0;
            if(cnt>=37936)
                cnt<=0;
            else
                cnt<=cnt+1;
        end
        4:
        begin
            if(cnt<=17908)
                voi<=1;
            else
                voi<=0;
            if(cnt>=35816)
                cnt<=0;
            else
                cnt<=cnt+1;
        end
        default:
            begin
                cnt<=0;
                voi<=0;
            end
    endcase
end
endmodule
```

5. 分数展示模块

需要时钟 clk_vga 以及分数的输入，然后输出是七段数码管，以及决定哪些需要量的输出，然后依据分数，获得个十百千位的数值，然后将时钟分成 4 分，对应不同的段内只亮对应位数的数码管，从而展示数据。

代码：

```
module scoreShow(
    input clk_vga,
    input [13:0]score,//分数
    output reg[6:0]seg_data, //七段数码管
    output reg[7:0]seg_sel//选择哪些是亮的
);
reg [3:0]score_ones=0;//分数的个位数
reg [3:0]score_tens=0;//分数的十位数
reg [3:0]score_hund=0;//分数的百位数
reg [3:0]score_thou=0;//分数的千位数

reg [15:0]cnt=0;
reg [3:0]data;

always@(posedge clk_vga)
begin
    begin
        cnt<=cnt+1;
        if(cnt>=4000)
            cnt<=0;
    end
end

always@(posedge clk_vga)
begin
    score_ones<=score%10;
    score_tens<=(score/10)%10;
```

```
    score_hund<=(score/100)%10;
    score_thou<=(score/1000)%10;
end

always@(posedge clk_vga)
begin
    begin
        if(cnt<=1000)//cnt 在不同的值亮不同的管
        begin
            seg_sel<=8'b11111110;
            data<=score_ones;
        end
        else if(cnt<=2000)
        begin
            seg_sel<=8'b11111101;
            data<=score_tens;
        end
        else if(cnt<=3000)
        begin
            seg_sel<=8'b11111011;
            data<=score_hund;
        end
        else if(cnt<=4000)
        begin
            seg_sel<=8'b11110111;
            data<=score_thou;
        end
        case (data)//展示对应的数字
            4'd0:seg_data <= 7'b1000000;
            4'd1:seg_data <= 7'b1111001;
            4'd2:seg_data <= 7'b0100100;
            4'd3:seg_data <= 7'b0110000;
            4'd4:seg_data <= 7'b0011001;
            4'd5:seg_data <= 7'b0010010;
```

```
            4'd6:seg_data <= 7'b0000010;
            4'd7:seg_data <= 7'b1111000;
            4'd8:seg_data <= 7'b0000000;
            4'd9:seg_data <= 7'b0010000;
            default:
            seg_data <= 7'b1111111;
        endcase
    end
end
endmodule
```

6. 方块及背景图及底部图片展示模块

需要时钟和复位信号，以及所需要的各个图片的 rgb 值，并且将这些图片的地址进行处理后再传出去，以此来不断的更新图片的像素值。然后通过对 vga 控制模块的调用，在对应的位置通过条件判断放入需要的图片像素点，这里采用了两种更新图片地址的方式，一种是背景图片的不断累积按，一种是更具 x，y 做运算，然后给出地址，明显第二种更适合运动的物块。

代码

```
module show_test(
    input clk_vga,//输入 vga 的时钟，频率为 25Mhz
    input rst,//复位信号，高电平有效
    output x_valid,//x 方向有效
    output y_valid,//y 方向有效
    output reg[3:0] red,//对应输出给 vga 的 rgb 值
    output reg[3:0] blue,
    output reg[3:0] green,
    input [11:0]rgb,//背景图的 rgb
    output reg [20:0]addr=1,//背景图的像素地址
    input [12:0] ytop1,//蜘蛛块的最上方 y 坐标
    input [12:0] ytop2,
    input [12:0] ytop3,
    input black_chose1,//下方是否要出现对应的图片
    input black_chose2,
    input black_chose3,
    input [11:0]zz_rgb1,//蜘蛛块的 rgb
    input [11:0]zz_rgb2,
    input [11:0]zz_rgb3,
    input [3:0]jin_rgb1,////下方图片的 rgb 值
    input [3:0]jin_rgb2,
    input [3:0]jin_rgb3,
    output reg [20:0]zz_addr1=1,//蜘蛛块的像素地址
    output reg [20:0]zz_addr2=1,
    output reg [20:0]zz_addr3=1,
```

```
    output reg [20:0]jin_addr1=1,//下方图片的地址
    output reg [20:0]jin_addr2=1,
    output reg [20:0]jin_addr3=1
);
    wire [11:0] x_poi;//输出此时 x 的坐
    wire [11:0] y_poi;//输出此时 y 的坐
    wire is_display;//表征此时是否能够输出

    parameter witehigh=100;//块的高

    parameter [10:0] witex1=164;//上个蜘蛛块的左边的 x 坐标
    parameter [10:0] witex2=364;
    parameter [10:0] witex3=564;
    parameter [10:0] witewide=160;

    parameter [10:0] y_bottom=415;//底部的块的上边的坐标

    vga_control control(clk_vga,rst,x_poi,y_poi,is_display,x_v
alid,y_valid);//控制 vga

    always @(posedge clk_vga)//根据优先级显示图片,先是判断是否是底
部图片，再判断是否是蜘蛛块图片，再不是就放入背景图片
    begin
        red<=0;
        blue<=0;
        green<=0;
        if(is_display)
        begin
            if(black_chose1==1&&x_poi>=witex1&&x_poi<witex1+wi
tewide&&y_poi>=y_bottom&&y_poi<y_bottom+witehigh)//
            begin
                red <=jin_rgb1;//底部图片放入的是黑白图
                blue <=jin_rgb1;
                green <=jin_rgb1;
```

```
                if(jin_addr1==1599)
                    jin_addr1<=0;
                else
                    jin_addr1<=(x_poi-witex1)+(y_poi-y_bottom)
*witewide+1;
            end
            else
            begin
                if(black_chose2==1&&x_poi>=witex2&&x_poi<witex
2+witewide&&y_poi>=y_bottom&&y_poi<y_bottom+witehigh)//
                begin
                    red <=jin_rgb2;
                    blue <=jin_rgb2;
                    green <=jin_rgb2;
                    if(jin_addr2==1599)
                        jin_addr2<=0;
                    else
                        jin_addr2<=(x_poi-witex2)+(y_poi-y_bot
tom)*witewide+1;
                end
                else
                begin
                    if(black_chose3==1&&x_poi>=witex3&&x_poi<w
itex3+witewide&&y_poi>=y_bottom&&y_poi<y_bottom+witehigh)//
                    begin
                        red <=jin_rgb3;
                        blue <=jin_rgb3;
                        green <=jin_rgb3;
                        if(jin_addr3==1599)
                            jin_addr3<=0;
                        else
                            jin_addr3<=(x_poi-witex3)+(y_poi-y
_bottom)*witewide+1;
                    end
```

```
                    else
                    begin
                        if(x_poi>=witex1&&x_poi<witex1+witewid
e&&y_poi>=ytop1&&y_poi<ytop1+witehigh)//
                        begin
                            red[3]<=zz_rgb1[11];
                            red[2]<=zz_rgb1[10];
                            red[1]<=zz_rgb1[9];
                            red[0]<=zz_rgb1[8];
                            green[3]<=zz_rgb1[7];
                            green[2]<=zz_rgb1[6];
                            green[1]<=zz_rgb1[5];
                            green[0]<=zz_rgb1[4];
                            blue[3]<=zz_rgb1[3];
                            blue[2]<=zz_rgb1[2];
                            blue[1]<=zz_rgb1[1];
                            blue[0]<=zz_rgb1[0];
                            if(zz_addr1==1599)
                                zz_addr1<=0;
                            else
                                zz_addr1<=(x_poi-witex1)+(y_po
i-ytop1)*witewide+1;
                        end
                        else
                        begin
                            if(x_poi>=witex2&&x_poi<witex2+wit
ewide&&y_poi>=ytop2&&y_poi<ytop2+witehigh)//
                            begin
                                red[3]<=zz_rgb2[11];
                                red[2]<=zz_rgb2[10];
                                red[1]<=zz_rgb2[9];
                                red[0]<=zz_rgb2[8];
                                green[3]<=zz_rgb2[7];
                                green[2]<=zz_rgb2[6];
```

```
                                green[1]<=zz_rgb2[5];
                                green[0]<=zz_rgb2[4];
                                blue[3]<=zz_rgb2[3];
                                blue[2]<=zz_rgb2[2];
                                blue[1]<=zz_rgb2[1];
                                blue[0]<=zz_rgb2[0];
                                if(zz_addr2==1599)
                                    zz_addr2<=0;
                                else
                                    zz_addr2<=(x_poi-witex2)+(
y_poi-ytop2)*witewide+1;
                            end
                            else
                            begin
                                if(x_poi>=witex3&&x_poi<witex3
+witewide&&y_poi>=ytop3&&y_poi<ytop3+witehigh)//
                                begin
                                    red[3]<=zz_rgb3[11];
                                    red[2]<=zz_rgb3[10];
                                    red[1]<=zz_rgb3[9];
                                    red[0]<=zz_rgb3[8];
                                    green[3]<=zz_rgb3[7];
                                    green[2]<=zz_rgb3[6];
                                    green[1]<=zz_rgb3[5];
                                    green[0]<=zz_rgb3[4];
                                    blue[3]<=zz_rgb3[3];
                                    blue[2]<=zz_rgb3[2];
                                    blue[1]<=zz_rgb3[1];
                                    blue[0]<=zz_rgb3[0];
                                    if(zz_addr3==1599)
                                        zz_addr3<=0;
                                    else
                                        zz_addr3<=(x_poi-witex
3)+(y_poi-ytop3)*witewide+1;
```

```
                                end
                                else
                                begin
                                    red[3]<=rgb[11];
                                    red[2]<=rgb[10];
                                    red[1]<=rgb[9];
                                    red[0]<=rgb[8];
                                    green[3]<=rgb[7];
                                    green[2]<=rgb[6];
                                    green[1]<=rgb[5];
                                    green[0]<=rgb[4];
                                    blue[3]<=rgb[3];
                                    blue[2]<=rgb[2];
                                    blue[1]<=rgb[1];
                                    blue[0]<=rgb[0];
                                end
                            end
                        end
                    end
                end
            end
            addr=addr+1;
            if(addr>307200)
                addr=1;
        end
    end
endmodule
```

7.vga 控制模块

需要时钟和清零信号，输出现在扫描到的 x，y 坐标，以及此时是否能输出，和行有效还有列有效信号。只有当 x，y 扫描到了对应位置的时候，才能够开始输出使得 vga 屏幕上有图片。

代码：

```
module vga_control(
    input vga_clk,//时钟周期
    input rst,//清零信号，高电平有效
    output reg[11:0] x_poi,//输出此时 x 的坐标
    output reg[11:0] y_poi,//输出此时 y 的坐标
    output is_display,//表征此时是否能够输出
    output x_valid,//行有效信号`
    output y_valid//列有效信号
);
    //行参数
    parameter x_sync=11'd96;
    parameter x_before=11'd144;
    parameter x_beside_after=11'd784;
    parameter x_all=11'd800;
    //列参数
    parameter y_sync=11'd2;
    parameter y_before=11'd35;
    parameter y_beside_after=11'd515;
    parameter y_all=11'd525;
    //
    assign is_display=((x_poi>=x_before)&&(x_poi<x_beside_afte
r)&&(y_poi>=y_before)&&(y_poi<y_beside_after))?1:0;//y=480//x=
640 //是否可以输出
    assign x_valid=(x_poi<x_sync)?0:1;//行有效
    assign y_valid=(y_poi<y_sync)?0:1;//列有效

    always @ (posedge vga_clk)//判断此时是否可以进行绘制图像
```

```
    begin
        if(rst)//清零信号
        begin
            x_poi<=1;
            y_poi<=2;
        end
        else
        begin
            if(x_poi==x_all-1)
            begin
                x_poi<=0;
                if(y_poi==y_all-1)
                begin
                    y_poi<=0;
                end
                else
                begin
                    y_poi<=y_poi+1;
                end
            end
            else
            begin
                x_poi<=x_poi+1;
            end
        end
    end
endmodule
```

8 判定模块

需要时钟，3 个对应键的值，游戏的模式选择，还有游戏开始信息，然后输出 3 个块的顶点坐标，下面 3 个块的出现判定，还有分数，以及游戏结束的声音判定、按下 3 个键的游戏声音判定。

主要是根据状态机来实现控制，具体可参照前面的状态机描述

代码：

```
module y_top_state(
    input clk_vga,//25M
    input in1,//3 个按钮
    input in2,
    input in3,
    input game_rst,//游戏重新开始
    input [7:0]game_level,//游戏模式
    input game_level_choose,//游戏等级
    output [12:0] ytop1,//输出 3 个蜘蛛块的上方坐标
    output [12:0] ytop2,
    output [12:0] ytop3,
    output reg black_chose1,//下方图片是否需要展示
    output reg black_chose2,
    output reg black_chose3,
    output [13:0]score,//分数
    output reg [1:0]gameover_voi=0,//4 个声音
    output reg voi1=0,
    output reg voi2=0,
    output reg voi3=0
);
reg signed[12:0]yt1;
reg signed[12:0]yt2;
reg signed[12:0]yt3;

reg [28:0]count=0;
```

```
reg [2:0]sta1=2;//3 个蜘蛛块的 y 坐标
reg [2:0]sta2=2;
reg [2:0]sta3=2;

reg in1t=0;//按键的对应状态机需要的
reg in2t=0;
reg in3t=0;

reg [1:0]in1u=0;
reg [1:0]in2u=0;
reg [1:0]in3u=0;

reg b_c_sta1=0;//下方块展示状态
reg [28:0]b_c_count1=0;//下方块展示的时间
reg b_c_sta2=0;
reg [28:0]b_c_count2=0;
reg b_c_sta3=0;
reg [28:0]b_c_count3=0;

reg [13:0]rescore=0;//分数

reg [63:0]random_top=64'h82cb49afde7982c6;//y 坐标每次刷新的随机
数
always @(in1)
begin
    if(in1==0)
        in1t<=0;
    if(in1==1)
        in1t<=1;
end

always @(in2)
begin
    if(in2==0)
```

```
        in2t<=0;
    if(in2==1)
        in2t<=1;
end

always @(in3)
begin
    if(in3==0)
        in3t<=0;
    if(in3==1)
        in3t<=1;
end

always @(posedge clk_vga)
begin
    if(in1t==1&&in1u==0)
        in1u<=1;
    if(in1t==1&&in1u==1)
        in1u<=2;
    if(in1t==1&&in1u==2)
        in1u<=2;
    if(in1t==0&&in1u==2)
        in1u<=0;
    if(in1t==0&&in1u==0)
        in1u<=0;
end

always @(posedge clk_vga)
begin
    if(in2t==1&&in2u==0)
        in2u<=1;
    if(in2t==1&&in2u==1)
        in2u<=2;
    if(in2t==1&&in2u==2)
```

```
        in2u<=2;
    if(in2t==0&&in2u==2)
        in2u<=0;
    if(in2t==0&&in2u==0)
        in2u<=0;
end

always @(posedge clk_vga)
begin
    if(in3t==1&&in3u==0)
        in3u<=1;
    if(in3t==1&&in3u==1)
        in3u<=2;
    if(in3t==1&&in3u==2)
        in3u<=2;
    if(in3t==0&&in3u==2)
        in3u<=0;
    if(in3t==0&&in3u==0)
        in3u<=0;
end

always@ (posedge clk_vga)
begin
    count<=count+1;
    if(count>((game_level-48)*103600))//(game_level-48)))30720
0
    begin
        count<=0;
    end
end

always @(posedge clk_vga)
begin
    if(game_rst||game_level_choose)
```

```
    begin
        sta1<=0;
    end
    else
    begin
        if(yt1>=515||sta1==2)
        begin
            sta1<=2;
        end
        else
        begin
            if(in1u==1&&yt1>=415)
                sta1<=0;
            else
            begin
                if(in1u==1&&yt1<415)
                begin
                    sta1<=2;
                end
                else
                begin
                if(count==0)
                    sta1<=1;
                else
                    sta1<=3;
                end
            end
        end
    end
end

always @(posedge clk_vga)
begin
    if(game_rst||game_level_choose)
```

```
    begin
        sta2<=0;
    end
    else
    begin
        if(yt2>=515||sta2==2)
        begin
            sta2<=2;
        end
        else
        begin
            if(in2u==1&&yt2>=415)
                sta2<=0;
            else
            begin
                if(in2u==1&&yt2<415)
                begin
                    sta2<=2;
                end
                else
                begin
                if(count==0)
                    sta2<=1;
                else
                    sta2<=3;
                end
            end
        end
    end
end

always @(posedge clk_vga)
begin
    if(game_rst||game_level_choose)
```

```
    begin
        sta3<=0;
    end
    else
    begin
        if(yt3>=515||sta3==2)
        begin
            sta3<=2;
        end
        else
        begin
            if(in3u==1&&yt3>=415)
                sta3<=0;
            else
            begin
                if(in3u==1&&yt3<415)
                begin
                    sta3<=2;
                end
                else
                begin
                    if(count==0)
                        sta3<=1;
                    else
                        sta3<=3;
                end
            end

        end
    end
end


always @(posedge clk_vga)
```

```
begin
    if(sta1==2||sta2==2||sta3==2)
    begin
        yt1<=(random_top[63:56]+235)%235-300;//({$random} %235
)-200;
        yt2<=(random_top[55:48]+235)%235-300;
        yt3<=(random_top[47:40]+235)%235-300;
        //rescore<=0;
    end
    else
    begin
        if((sta1==0&&sta2!=0&&sta3!=0)||(sta1!=0&&sta2==0&&sta
3!=0)||(sta1!=0&&sta2!=0&&sta3==0))
        begin
            rescore<=rescore+1;
        end
        else if((sta1==0&&sta2==0&&sta3!=0)||(sta1!=0&&sta2==0
&&sta3==0)||(sta1==0&&sta2!=0&&sta3==0))
            rescore<=rescore+2;
        else if(sta1==0&&sta2==0&&sta3==0)
            rescore<=0;//rescore+3;

        case (sta1)
        0:
        begin
            yt1<=(random_top[63:56]+235)%235-300;
        end
        1:
        begin
            yt1<=yt1+1;
        end
        3:
        begin
            yt1<=yt1;
```

```
        end
        default:
            yt1<=(random_top[63:56]+235)%235-300;
        endcase

        case (sta2)
        0:
        begin
            yt2<=(random_top[55:48]+235)%235-300;
        end
        1:
        begin
            yt2<=yt2+1;
        end
        3:
        begin
            yt2<=yt2;
        end
        default:
            yt2<=(random_top[55:48]+235)%235-300;
        endcase

        case (sta3)
        0:
        begin
            yt3<=(random_top[47:40]+235)%235-300;
            random_top={random_top[0],random_top[63:1]};
        end
        1:
        begin
            yt3<=yt3+1;
        end
        3:
        begin
```

```
            yt3<=yt3;
        end
        default:
            yt3<=0;
        endcase
    end
end

always @(posedge clk_vga)
begin
    if(sta1==2||sta2==2||sta3==2||b_c_count1>=12000000)
    begin
        b_c_sta1<=0;
    end
    else
    begin
        if(in1u==1||b_c_sta1==1)
        begin
            b_c_sta1<=1;
        end
    end
end

always @(posedge clk_vga)
begin
    if(b_c_sta1==0)
    begin
        b_c_count1<=0;
    end
    else
    begin
        if(b_c_sta1==1)
        begin
            b_c_count1<=b_c_count1+1;
```

```
            if(b_c_count1>12000000)
            begin
                b_c_count1<=0;
            end
        end
    end
end

always @(posedge clk_vga)
begin
    case(b_c_sta1)
    0:
    begin
        black_chose1<=0;
    end
    1:
    begin
        black_chose1<=1;
    end
    endcase
end
////
always @(posedge clk_vga)
begin
    if(sta1==2||sta2==2||sta3==2||b_c_count2>=12000000)
    begin
        b_c_sta2<=0;
    end
    else
    begin
        if(in2u==1||b_c_sta2==1)
        begin
            b_c_sta2<=1;
        end
```

```
    end
end

always @(posedge clk_vga)
begin
    if(b_c_sta2==0)
    begin
        b_c_count2<=0;
    end
    else
    begin
        if(b_c_sta2==1)
        begin
            b_c_count2<=b_c_count2+1;
            if(b_c_count2>12000000)
            begin
                b_c_count2<=0;
            end
        end
    end
end

always @(posedge clk_vga)
begin
    case(b_c_sta2)
    0:
    begin
        black_chose2<=0;
    end
    1:
    begin
        black_chose2<=1;
    end
    endcase
```

```
end

always @(posedge clk_vga)
begin
    if(sta1==2||sta2==2||sta3==2||b_c_count3>=12000000)
    begin
        b_c_sta3<=0;
    end
    else
    begin
        if(in3u==1||b_c_sta3==1)
        begin
            b_c_sta3<=1;
        end
    end
end

always @(posedge clk_vga)
begin
    if(b_c_sta3==0)
    begin
        b_c_count3<=0;
    end
    else
    begin
        if(b_c_sta3==1)
        begin
            b_c_count3<=b_c_count3+1;
            if(b_c_count3>12000000)
            begin
                b_c_count3<=0;
            end
        end
    end
```

```
end

always @(posedge clk_vga)
begin
    case(b_c_sta3)
    0:
    begin
        black_chose3<=0;
    end
    1:
    begin
        black_chose3<=1;
    end
    endcase
end

always @(posedge clk_vga)
begin
    if(gameover_voi==2&&(sta1!=2&&sta2!=2&&sta3!=2))
        gameover_voi<=0;
    else if(gameover_voi==2)
        gameover_voi<=2;
    else if(gameover_voi==1)
        gameover_voi<=2;
    else if(sta1==2||sta2==2||sta3==2)
        gameover_voi<=1;

        if(in1u==1)
            voi1<=1;
        else
            voi1<=0;
        if(in2u==1)
            voi2<=1;
        else
```

```
            voi2<=0;
        if(in3u==1)
            voi3<=1;
        else
            voi3<=0;

end

assign score = rescore;
assign ytop1 = yt1;
assign ytop2 = yt2;
assign ytop3 = yt3;
endmodule
```

五、测试模块建模

（要求列写各建模模块的 test bench 模块代码）

由于 vga 是扫描的信号，所以用 test bench 进行模拟比较困难，所以一般还是直接下板或动笔运算验证。

但是 MP3 虽然没有做出来，但是在做的过程中使用了很多 test bench，甚至用到了逻辑探测仪，但是还是没弄好。

MP3 的 test bench 代码：

```
`timescale 1ns/1ns
module set_tb;
reg clk,rst,miso,dreq;
wire sclk,mosi,xrst,xcs,xdcs;
reg[8:0] i=0;
reg[7:0] mod1=8'b00000_0010;
reg[7:0] addr1=8'hb;
reg[15:0] set1=16'h33;
reg clk_mp;
wire[15:0] data_m;
wire [20:0] addr;
wire [15:0] data;
wire [2:0] sta;
//Divider2 uut(clk,rst,clk_mp);
mp3_top uut(clk,rst,miso,dreq,sclk,mosi,xrst,xcs,xdcs);//,data
,addr);
//mp3_mem_test mp3_data(.clka(!clk_mp),.ena(1),.addra(addr),.d
outa(data_m));
/*
initial begin
    addr=0;
    clk_mp=1;
    #5;
    clk_mp=0;
    #5;
```

```
    clk_mp=1;
        addr=addr+1;
        #5;
        clk_mp=0;
        #5;
    clk_mp=1;
        addr=addr+1;
        #5;
        clk_mp=0;
        #5;
    clk_mp=1;
        addr=addr+1;
        #5;
        clk_mp=0;
        #5;
    clk_mp=1;
        addr=addr+1;
        #5;
        clk_mp=0;
        #5;
    clk_mp=1;
        addr=addr+1;
        #5;
        clk_mp=0;
        #5;
    clk_mp=1;
        addr=addr+1;
        #5;
        clk_mp=0;
        #5;
    clk_mp=1;
        addr=addr+1;
        #5;
        clk_mp=0;
```

```
        #5;
    clk_mp=1;
        addr=addr+1;
        #5;
        clk_mp=0;
        #5;
    clk_mp=1;
        addr=addr+1;
        #5;
        clk_mp=0;
        #5;
    clk_mp=1;
        addr=addr+1;
        #5;
        clk_mp=0;
        #5;
    clk_mp=1;
        addr=addr+1;
        #5;
        clk_mp=0;
        #5;
    clk_mp=1;
        addr=addr+1;
        #5;
        clk_mp=0;
        #5;
    clk_mp=1;
        addr=addr+1;
        #5;
        clk_mp=0;
        #5;
end
*/
initial
```

```
begin
    clk=1;
    rst=0;
    miso=0;
    dreq=1;
    #5;
    clk=!clk;
    #5;
    clk=!clk;
    #5;
    clk=!clk;
    #5
    rst=1;
    for(i=0;i<64;i=i+1)
    begin
    clk=!clk;
    #3;
    end
    dreq=0;
    for(i=0;i<6;i=i+1)
    begin
    clk=!clk;
    #3;
    end
    dreq=1;
    for(i=0;i<64;i=i+1)
    begin
    clk=!clk;
    #3;
    end
    dreq=0;
    for(i=0;i<6;i=i+1)
    begin
    clk=!clk;
```

```
    #3;
    end
    dreq=1;
    for(i=0;i<64;i=i+1)
    begin
    clk=!clk;
    #3;
    end
    dreq=0;
    for(i=0;i<6;i=i+1)
    begin
    clk=!clk;
    #3;
    end
    dreq=1;
    for(i=0;i<64;i=i+1)
    begin
    clk=!clk;
    #3;
    end
    dreq=0;
    for(i=0;i<6;i=i+1)
    begin
    clk=!clk;
    #3;
    end
    dreq=1;
    for(i=0;i<64;i=i+1)
    begin
    clk=!clk;
    #3;
    end
    dreq=0;
    for(i=0;i<6;i=i+1)
```

```
    begin
    clk=!clk;
    #3;
    end
    dreq=1;
    for(i=0;i<64;i=i+1)
    begin
    clk=!clk;
    #3;
    end
end
endmodule
```

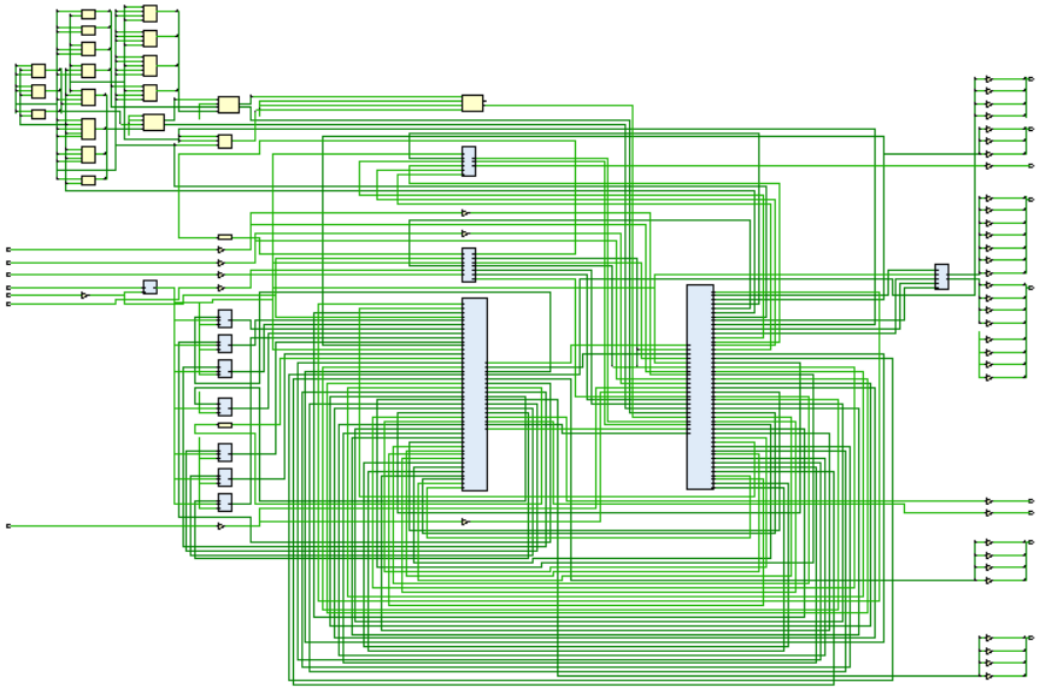
对于 block mem Generator 的读取数据验证的 test bench

```
`timescale 1ns/1ns
module mem_tb ;
reg clk;
reg [50:0]addr=1;
wire [11:0] data_m;
reg[20:0] i=1;
pic_mem uut(.clka(clk),.ena(1),.addra(addr),.douta(data_m));
initial
begin
    clk=0;
    #10;
    while(1)
    begin
        clk=!clk;
        if(clk==0)
            addr=addr+1;
        #10;
    end
end
endmodule  //mem_tb
```

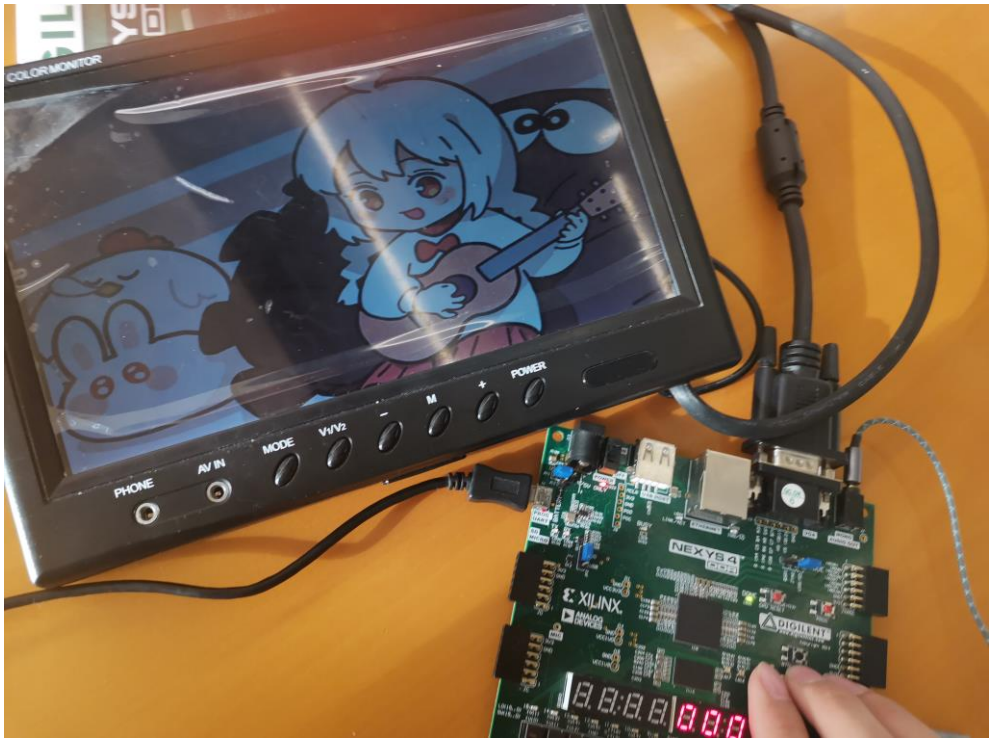
六、实验结果

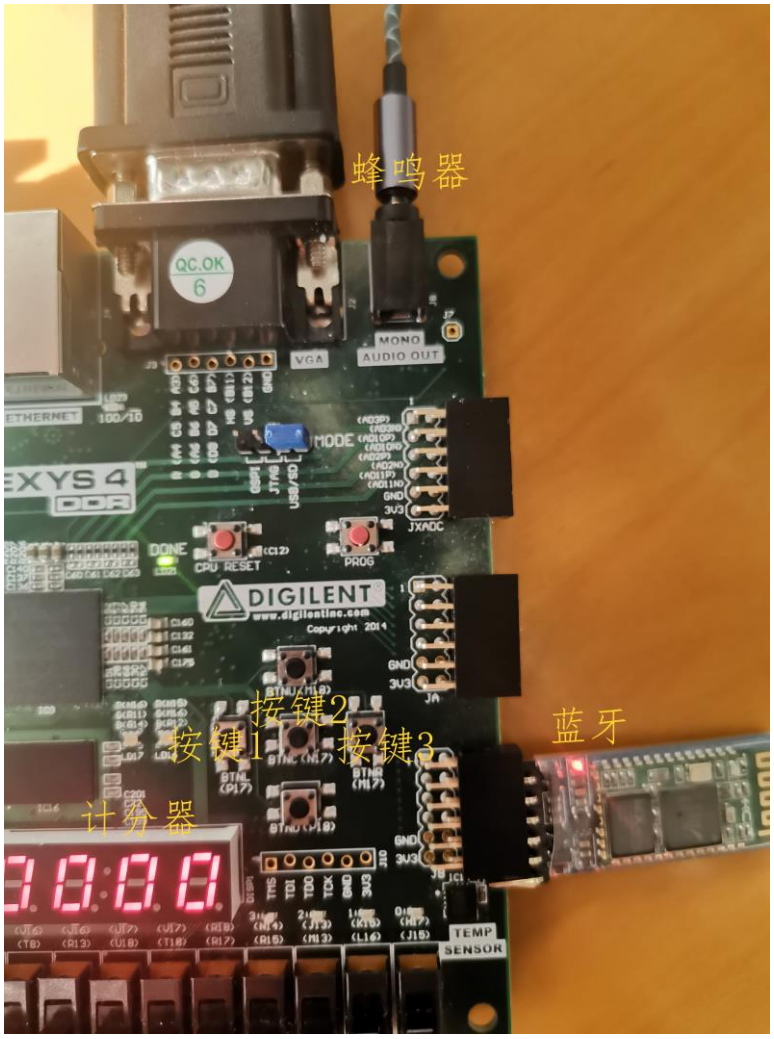
（该部分可截图说明，可包含 logisim 逻辑验证图、modelsim 仿真波形图、以及下板后的实验结果贴图）

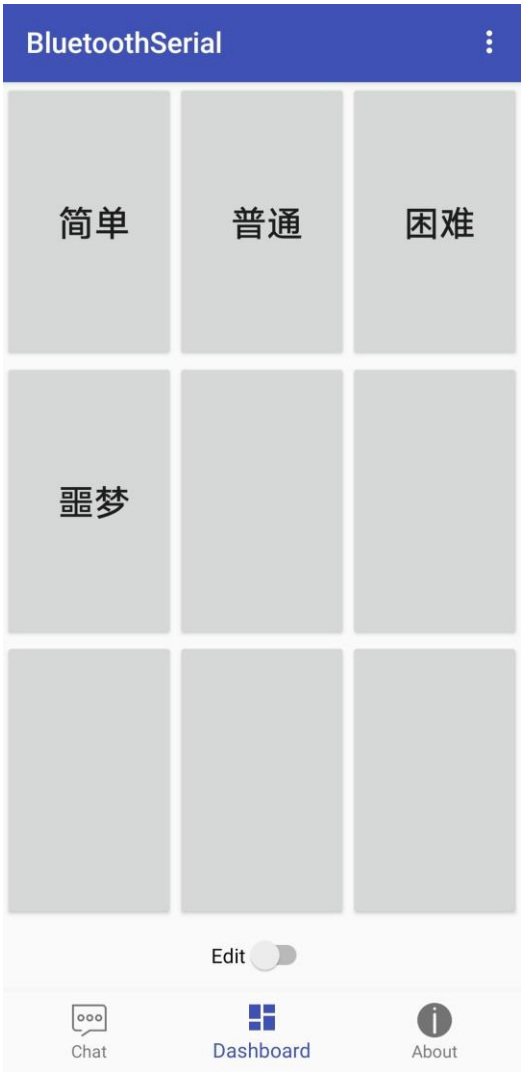
vivado 中的模拟电路图



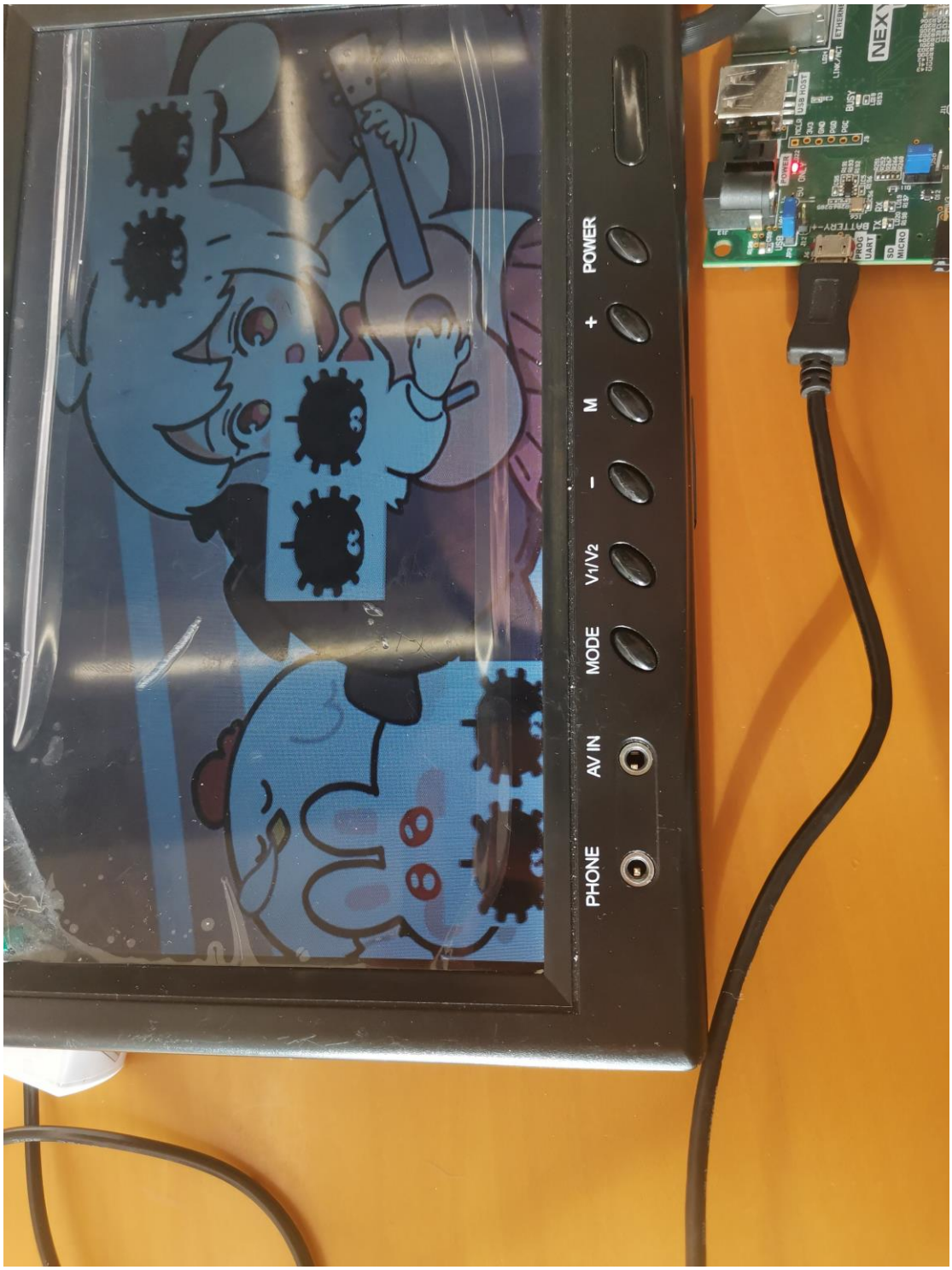
游戏开始界面：


COLOR MONITOR
PHONE
AV IN
MODE
V1/V2
POWER
NEXYS 4
XILINX

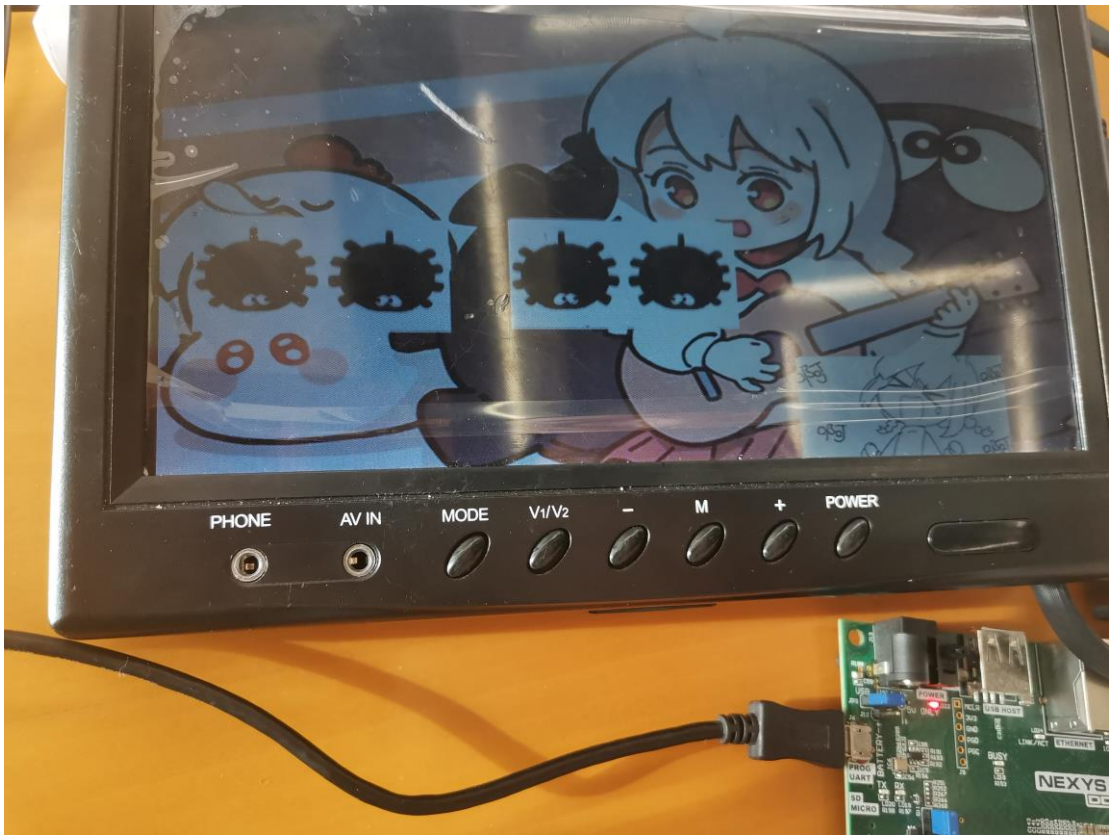

蜂鸣器
VGA
MONO
AUDIO OUT
MODE
JXADC
CPU RESET
PROG
DONE
DIGILENT
www.digilentinc.com
JA
按键2
按键1
按键3
蓝牙
计分器
TEMP
SENSOR


BluetoothSerial
简单
普通
困难
噩梦
Edit
Chat
Dashboard
About

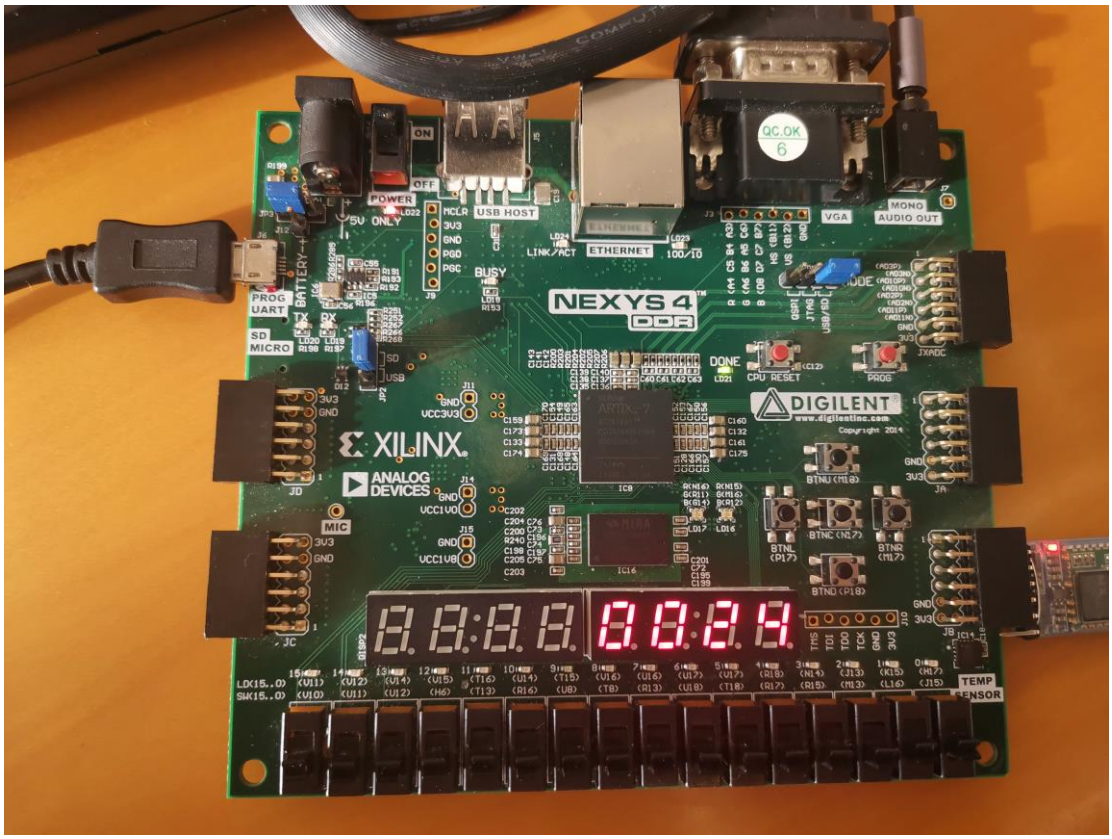
选择游戏模式或者重置游戏后开始游戏：



在蜘蛛块接触到了最下沿的时候点击对应的十字按钮的键，就可以消除，并且会在对应的底部出现图片，共 3 张：

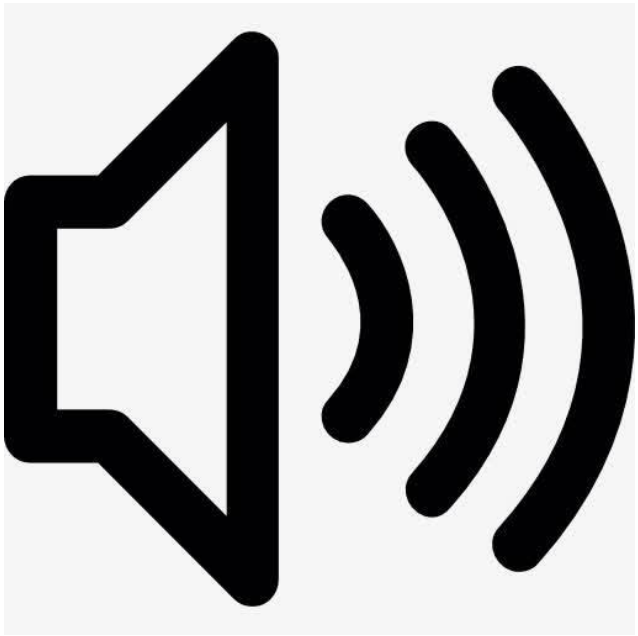


如果没有在对应的时候按下按键，让蜘蛛块划过或者提前按了，都是使得游戏失败，失败后无蜘蛛块滑落，分数显示不动：



再开始游戏需要将底部按钮最左边上拉，再下拉。将底部按钮最右边上拉会将 VGA 熄屏，游戏停止。

还有声音会根据按键或游戏结束而发出。



七、结论

根据上述可以使得游戏运行起来。并将已知 bug 修复了。

八、心得体会及建议

这次数字逻辑大作业真的花费了很多时间，一开始我是用 C 语言的思想去写这个游戏逻辑的，但是当变化一多的话就很容易出现冲突，然后综合不了，这也就使得我大概懂得了为什么要使用状态机，因为每个模块都是同步运行的，所以我们需要把一个变量的变化约束在一个 module 里面，这就需要一个状态量来做衔接了，以防止出现这个 module 里面输出因为输入在变，而另一个 module 里面也在变它。所以要完成一个复杂的系统就需要一定量的状态机了。

每个外设的使用也很需要琢磨，需要弄清楚它的输入输出，内部的时序逻辑，这也很需要资料的辅导，网上的资料有些良莠不齐，这就很需要甄别了，如果能找到好的资料就可以很快速的调配好。其实这些外设中我花费最长时间弄的是 MP3，大概花了一个多星期，但是还是没弄好，虽然对它的原理已经很熟悉了，但是因为 MP3 对于时序的要求很严格，有一点差错就播不出来了，所以调试起来也很痛苦，还动用了逻辑分析仪，最后也还是没弄好，也没有时间再弄了，真的是遗憾了。硬件的 debug 真的是很麻烦，有很多情况也不适合用 modelism 来模拟，需要自己多想想，多下板，但是下板又真的很要时间，会拖慢效率，所以也还需要在设计前就将这些时序全部想清楚，不要轻易下板，不然就很容易因为一些小 bug 而浪费很多时间。

还有什么可以综合什么不可以综合也很需要想清楚，要避免竞争冒险，同时还有些不可综合的语句也要避免使用，我之前想用{random}来产生随机数，虽然能下板，但是最后并没有随机数产生，而是一直都是 0，这就是不可综合语句了，所以最后采用的是用一大串数据，然后不断循环位移，将对应的位置赋值给每个蜘蛛块的顶部 y 的数据，当作伪随机数使用。

建议：

希望如果老师是想锻炼大家的系统设计能力的话，可以把这些外设的资料再给多一点，可以从每一届同学中挑选出一些解释的比较好的，当作资料流传下去，这样就不用再去网上漫无头绪的找资料了，大家也就能够将从怎么做出这个外设跳到能用这个外设做出什么了。