华中科技大学

实验名称：　　躲避游戏的设计

院（系）：　　**********

专业班级：

姓名：　　****　　****

　　****　　****

　　*****　　*****

时间：　　2018.4.26

地点：

实验成绩：

指导教师：

2018 年 4 月 26 日

一、 实验目的

1. 熟悉有限状态机的实现。
2. 熟悉 VGA 接口的设计
3. 掌握使用用 FPGA 设计游戏，键盘，接口的使用过程的方法
4. 掌握使用 FPGA 音频产生与输入，输出方法
5. 实现数字系统的设计、实现。
6. 设计熟悉 Verilog 分层次模块设计。

二、 实验要求

使用 Verilog HDL 和 FPGA 开发板实现一个具有一定功能的飞机躲避障碍物的游戏，键盘控制开始结束，配有一定的背景音乐。

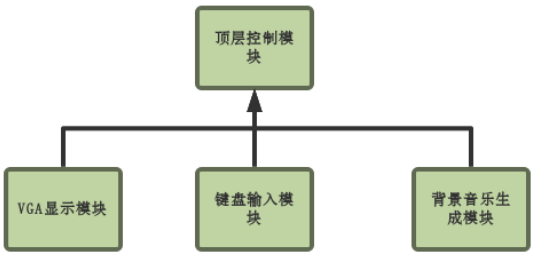
三.实验元器件

类型	型号（参数）	数量
FPGA 开发板	NEXYS4 DDR	1
显示器		1
小型音响		1
键盘		1

四.实验设计

1. 总体模块设计

总体模块总体上分为顶层模块,键盘输入输出模块，VGA 显示模块，背景音乐生成模块和控制模块五大部分。如下图：



2. 顶层模块设计

顶层模块是这个项目的主要部分，也是各个模块的综合部分，并将分数实时显示在数码管和 VGA 显示屏上。

3. 键盘输入输出模块

辅助功能微控制器（Microchip PIC24FJ128）为 Nexys4 提供 USB HID 主机功能。上电后，微控制器处于配置模式，将比特流下载到 FPGA 或等待从其他源编程。 一旦 FPGA 被编程，微控制器切换到应用模式，在这种情况下是 USB HID 主机。 微控制器中的固件可以驱动连接到标有“USB Host”的 J5 型 A 型 USB 连接器的鼠标或键盘。集线器支持目前不可用，因此只能使用单个鼠标或单个键盘。 信号进入 FPGA--两个用于实现与鼠标或键盘通信的标准 PS / 2 接口。

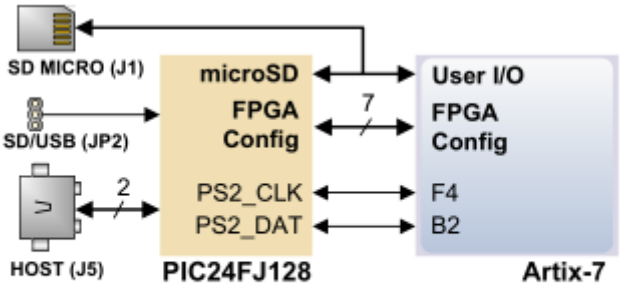
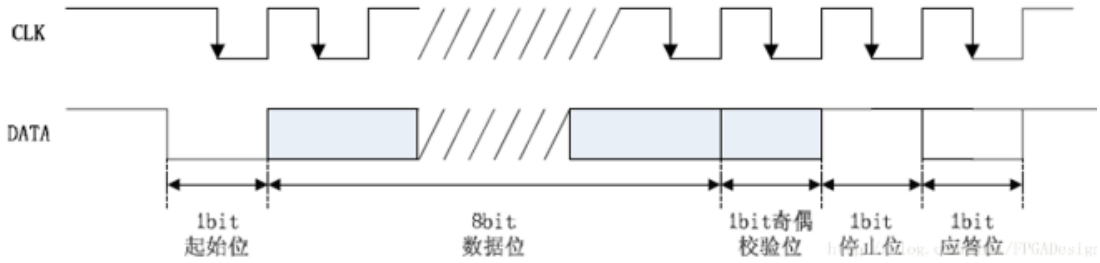


Figure 7. Nexys4 PIC24 Connections

而 PS2 键盘由一根时钟线、一根数据线完成通信，PS2 通信的帧格式如下所示，时钟的下降沿数据有效。：



按键在被按下时，会发送一个字节，这个码就是**通码**；按键在释放时，会发送两个字节，这个码就做**断码**（当然也有例外）。每一个按键都有唯一的通码和断码，断码一般为 0xF0 加通码，据此进行判断按下的是哪个键，断开哪个键，从而执行对应的功能。

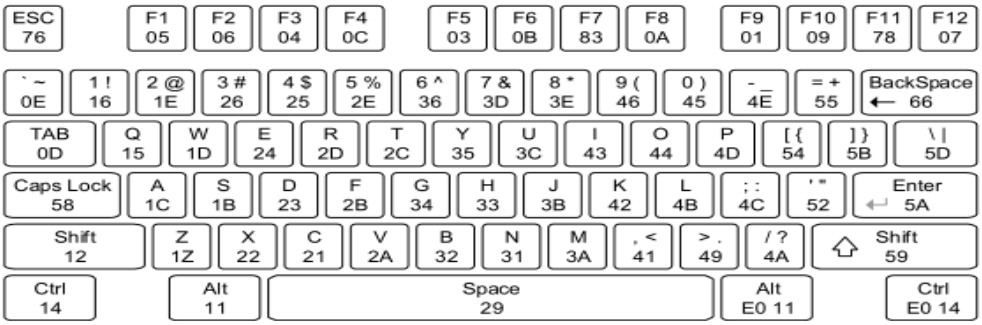
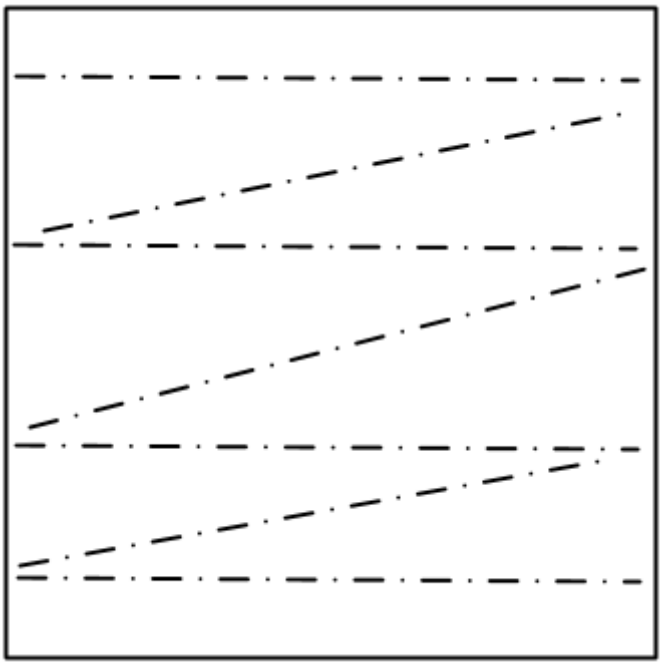


Figure 9. Keyboard scan codes

4. VGA 输入模块设计

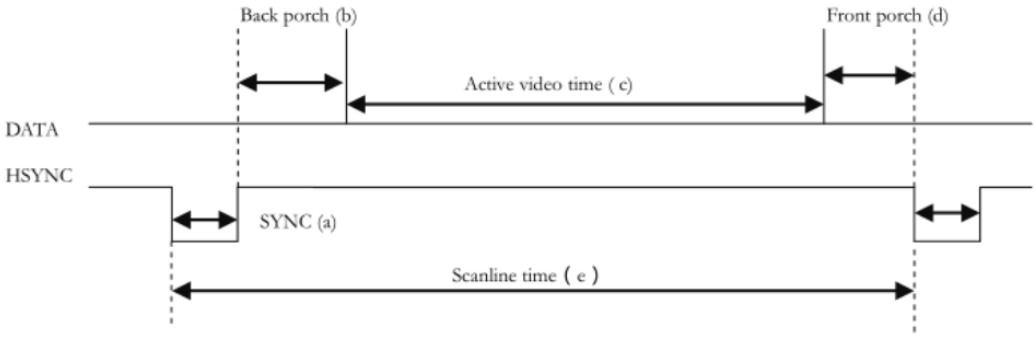
（1）VGA 显示图像

常见的彩色显示器，一般由 CRT（阴极射线管）构成。彩色是有 R,G.B（红：RED，绿：GREEN,蓝：BLUE）三基色组成。显示是用逐行扫描的方式解决，阴极射线枪发出电子束打在涂有银光粉的荧光屏幕上，产生 R,G,B 三基色，合成一个彩色像素。扫描从屏幕的左上方开始，从左到右，从上到下，进行扫描，每扫完一行，电子束回到屏幕的左边下一行的起始位置，在这期间，CRT 对电子束进行消隐，每行结束时，用行同步信号进行同步，扫描完所有行，用场同步信号进行场同步，并使扫描回到屏幕的左上方，同时进行场消隐，预备下一场的扫描。扫描示意图如下：

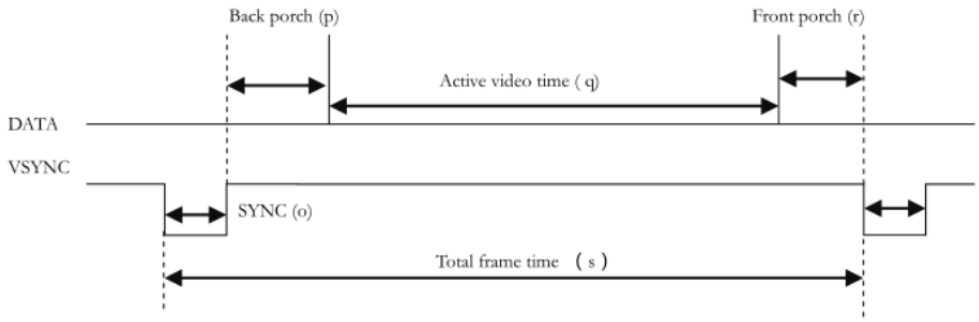


VGA 中定义行时序和场时序都需要同步脉冲（Sync a）、显示后沿（Back porch b）、显示时序段（Display interval c）和显示前沿（Front porch d）四部分。只有在显示时序段，也就是 C 区才可以信号显示出来，其他区域无法显示。

行时序



场时序

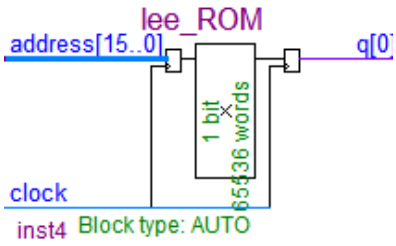


对于确定输出图像在屏幕中的位置，我们通过设置行同步计数器和列同步计数器实现，对于扫描信号，行列计数器分别计数，当计数器达到屏幕的某一固定像素，即预定图像输出位置时，给图像输出使能信号，开始输出图像。当计数器达到屏幕边界时，停止输出图像。实现如下：

```
//行扫描
always @(posedge vga_clk)
begin
    if (hcount_ov)
        hcount <= 10'd0;
     else
         hcount <= hcount + 10'd1;
end
assign hcount_ov = (hcount == hpixel_end);

//场扫描
always @(posedge vga_clk)
begin
    if (hcount_ov)
    begin
        if (vcount_ov)
            vcount <= 10'd0;
        else
            vcount <= vcount + 10'd1;
    end
end
assign vcount_ov = (vcount == vline_end);
```

（2）图片储存

图片存储需要调用 ROM 模块。ROM 模块使用宏功能模块实现，用于存储图片信息。



对于可移动的图片，先选择所需要的图片，转换为 8 位或 24 位 BMP 位图，再使用“Bmp2Mif”软件生成对应的“coe”文件，存入工程所在的文件夹。

根据图片的大小选择 ROM 的位数与存储大小。游戏的背景图片，如标题，开发者，控制界面均以图片形式保存在 ROM 中，在控制模块直接调用显示在屏幕上。

5. 背景音乐模块设计

通过向蜂鸣器输入不同频率的方波完成发声。在程序中写入指定乐谱，通过开关可调整到播放模式，自动按照音符顺序播放制定乐谱，或者演奏音阶。主要利用了有限状态机，从乐谱中依次读取乐符，然后从 A11 端口输出就可以了，通过外接音箱就可以播放出相应频率的声音。

6. 控制模块设计

控制模块是本程序的核心模块，有随机数产生，版面大小，飞机边长，模块大小等等部分，最后将各个部分画出来。

随机数产生

随机数的产生是将几个 D 触发器相互连接，使用 Verilog 进行建模，产生一定位数的随机数。

板块控制

板块控制是使用几个计数器，并且设置最大速度，其游戏的操作难度可以通过改变计数器的大小即可，这样就能控制各个障碍的移动速度。

根据产生的随机数，确定障碍的位置，这样可以使游戏更有随机性，更加有趣。

游戏设计

判定失败：当移动方块的坐标和障碍的坐标重合的时候，判定 die，结束游戏。设置三个难度，使用键盘进行选择对应的难度，之后可以控制游戏速度，分为三个难度，为：简单，中等，较困难，困难。

五.Verilog 程序设计

顶层模块

```
module top(
    clk,up,down,hsync,vsync,disp_RGB,out,ps2_state,ps2_clk,ps2_data,rst_
n,light_choose,display);
input clk;
input ps2_clk,ps2_data;
input rst_n;
output ps2_state;
input up;
input down; // 下降
output out;


output hsync; //VGA 行同步信号
output vsync; //VGA 场同步信号
output [11:0]disp_RGB; //VGA 数据输出

wire dat_act;
wire up_key_press;
wire down_key_press;
wire [9:0]hc,vc;
wire [5:0] score;
wire pause;
wire song;
wire [1:0] dengji;
output [7:0] light_choose;
output [7:0] display;
```

```
//按键输入模块
key U1(clk,start,up,down,up_key_press,down_key_press);
//控制模块
control U2( pause,clk,start,
disp_RGB,hc,vc,dat_act,up_key_press,down_key_press,dengji,score );
//VGA 输出模块
vga  U3( clk,start,hsync, vsync,hc,vc,dat_act);
// 背景音乐
song U4(clk, song, out);
//数码管
wire [3:0]num3,num2,num1,num0;
assign num3=score/1000;
assign num2=(score-num3*1000)/100;
assign num1=(score-num3*1000-num2*100)/10;
assign num0=(score-num3*1000-num2*100-num1*10);

displaynumber(clk,num0,num1,num2,num3,light_choose,display);

ps2 U5(clk,rst_n,ps2_clk,ps2_data,ps2_state,start,pause,song,dengji);

endmodule
```

VGA 扫描模块

```
module vga( clk,reset,hsync, vsync,hc,vc,dat_act);
           input clk;
           input reset;

           output hsync; //VGA 行同步信号
           output vsync; //VGA 场同步信号
           output dat_act;
           output [9:0]hc ,vc; //行，列

           reg [9:0] hcount; //VGA 行扫描计数器
           reg [9:0] vcount; //VGA 场扫描计数器

           reg flag;
           wire hcount_ov;
           wire vcount_ov;

           wire hsync;
           wire vsync;

           reg vga_clk=0;
```

```
    reg cnt_clk=0; //分频计数

    //VGA 行、场扫描时序参数表
    parameter hsync_end = 10'd95,
    hdat_begin = 10'd143,
    hdat_end = 10'd783,
    hpixel_end = 10'd799,

    vsync_end = 10'd1,
    vdat_begin = 10'd34,
    vdat_end = 10'd514,
    vline_end = 10'd524;


  //分频
    always @(posedge clk)
    begin
        if(cnt_clk == 1)
        begin
            vga_clk <= ~vga_clk;
            cnt_clk <= 0;
         end
        else
            cnt_clk <= cnt_clk +1;
    end

 //************************VGA 驱动部分
*******************************//行扫描

    always @(posedge vga_clk)
    begin
        if (hcount_ov)
            hcount <= 10'd0;
         else
             hcount <= hcount + 10'd1;
    end
    assign hcount_ov = (hcount == hpixel_end);

    //场扫描
    always @(posedge vga_clk)
    begin
        if (hcount_ov)
        begin
            if (vcount_ov)
```

```
                    vcount <= 10'd0;
                else
                    vcount <= vcount + 10'd1;
            end
        end
        assign vcount_ov = (vcount == vline_end);

        //数据、同步信号输
        assign dat_act = ((hcount >= hdat_begin) && (hcount <
hdat_end))&& ((vcount >= vdat_begin) && (vcount < vdat_end));
        assign hsync = (hcount > hsync_end);
        assign vsync = (vcount > vsync_end);

        //计数器转成 640 x 480 的样式，方便开发
        assign hc = hcount - hdat_begin;
        assign vc = vcount - vdat_begin;

endmodule
```

键盘模块

```
module ps2(
       input clk,
       input rst_n,
       input ps2_clk,
       input ps2_data,
       output reg ps2_state,
       output reg start,pause,song,
       output reg[1:0]dengji
   );
   reg ps2_clk0, ps2_clk1, ps2_clk2;//缓冲寄存器
   wire ps2_clk_neg;  //1 表示检测到下降沿

    initial begin
       ps2_state<=0;
    end

   always @ (posedge clk or negedge rst_n)
       if (!rst_n)
           {ps2_clk0, ps2_clk1, ps2_clk2} <= 3'd0;
       else
           begin
               ps2_clk0 <= ps2_clk;
               ps2_clk1 <= ps2_clk0;
```

```
            ps2_clk2 <= ps2_clk1;
        end

    assign ps2_clk_neg = ~ps2_clk1 & ps2_clk2;


//数据接收----------------------------

    reg [7:0]data_temp;//当前接收的数据
     reg [3:0]num;   //移位控制

    always @ (posedge clk or negedge rst_n)
        if (!rst_n)
            begin
                num <= 4'd0;
                data_temp <= 8'd0;
            end

        else if (ps2_clk_neg)
            begin
                if (num == 0) num <= num + 1'b1;//跳过起始位
                else if (num <= 8)              //数据位赋值
                    begin
                        num <= num + 1'b1;
                        data_temp[num-1] <= ps2_data;
                    end
                else if (num == 9) num <= num + 1'b1;//跳过校验位
                else begin
                        num <= 4'd0;              //清 0
                        ps2_state<=~ps2_state;
                     end
            end


//按键按下/松开检测------------------------

    reg ps2_loosen;//1 表示按键松开

    always @ (posedge clk or negedge rst_n)
        if (!rst_n)
            begin
                ps2_loosen<= 1'b0;
            end
```

```
        //每接收完一个数据就进行按键检测
        else if (num == 4'd10)
            if (data_temp == 8'hf0) ps2_loosen <= 1'b1;//断码标识符
            else
                begin
                    if (ps2_loosen) //1 表示按键松开，下一次接收数据后清 0
                        begin
                            ps2_loosen<= 1'b0;
                        end
                    else            //loosen 变 0 后的下一个数据表示按键被按下
                        begin
                                    case(data_temp)
                                       8'h16:begin
start<=1'b1;pause<=1'b0; end
                                       8'h1e:begin
start<=start;pause<=1'b1; end
                                       8'h26:begin
start<=1'b0;pause<=1'b0; end
                                       8'h25:begin song<=1'b1; end
                                       8'h2e:begin song<=1'b0; end
                                       8'h22:begin dengji<=2'b00;end
                                       8'h21:begin dengji<=2'b01;end
                                       8'h2a:begin dengji<=2'b10;end
                                       8'h32:begin dengji<=2'b11;end
                                    default: begin
                                              start<=start;
                                           end
                                    endcase
                        end
                end

endmodule
```

按键控制模块

```
module key(clk,reset,up,down,up_key_press,down_key_press);
input clk;
input reset;
input up;
input down;
output reg up_key_press;
output reg down_key_press;

//难度控制
```

```
parameter T = 01_800_00;  //控制方块上升速度
parameter T_DOWN = 06_000_00; // 控制自动下降速度（add）

reg [30:0] counter;
reg [30:0] counter2;
always@(posedge clk,negedge reset )
begin
     if(!reset)
        begin
            counter <= 0;
            counter2 <= 0;
            up_key_press <= 0;
            down_key_press <= 0;
         end
     else
        begin
            if(up)
                begin
                    //计数器控制上升速度大小
                    if(counter <= T)
                        begin
                            counter <= counter + 1'b1;
                            up_key_press <= 0;
                        end
                    else
                        begin
                            counter <= 0;
                            up_key_press <= 1;
                        end
                end
             else  //下降
                begin
                    if(counter2 <= T_DOWN)
                        begin
                            if(down) //down 的按键
                                begin
                                    counter2 <= counter2 + 3; // 加速下降
                                            down_key_press <= 0; //常速下降
                                end
                            else
                                begin
                                    counter2 <= counter2 +  1'b1;
                                    down_key_press <= 0; //常速下降
                                end
```

```
                    end
                else
                    begin
                        counter2 <= 0;
                        down_key_press <= 1;
                    end
            end
        end
end

endmodule

module control( pause,clk,reset,
disp_RGB,hc,vc,dat_act,up_key_press,down_key_press,easy,score );
input pause; // 暂停信号
input clk;
input reset;
input dat_act;
input [9:0]hc,vc;
input up_key_press;
input down_key_press;

output [11:0]disp_RGB; //VGA 数据输出
input [1:0] easy;
output reg [9:0] score;
reg [11:0]data;
reg vga_clk=0; //vga 时钟

reg cnt_clk=0; //分频计数


//分频 vga_clk 25Mhz 时钟
always @(posedge clk)
begin
        if(cnt_clk == 1)
        begin
            vga_clk <= ~vga_clk;
            cnt_clk <= 0;
        end
        else
            cnt_clk <= cnt_clk +1;
end

    //定义版面的大小
```

```
     parameter game_y = 480;
     parameter game_x = 500;
    //定义飞机的边长
    parameter border = 30;
    //定义挡板的宽度
    parameter ban = 16;
    //定义挡板的长度
    parameter long = 200;
    //定义挡板的间隔
    parameter magin = 200;
     //定义分数行位置
     parameter M = 45;
     //定义分数列位置
     parameter N = 540;
     //分数各位
     wire [3:0]num3,num2,num1,num0;
     //ip 核输出
     wire[7:0] data_num;
     reg[15:0] addr;
     reg[15:0] addr2;
     wire[7:0] data_num2;
     reg [15:0]addr3;
     wire [7:0]data3;
    //ip 核调用
     score0 score0( .a(addr), .spo(data_num) );
     fenshue score1( .a(addr2), .spo(data_num2) );
     member score2( .a(addr3), .spo(data3) );
     assign num3=score/1000;
     assign num2=(score-num3*1000)/100;
     assign num1=(score-num3*1000-num2*100)/10;
     assign num0=(score-num3*1000-num2*100-num1*10);

    //VGA 扫描，画出挡板和方块，并设置挡板移动的移动变量 push
    reg [10:0] push,push1,push2,push3;
    reg stop;//用于停止游戏

    //飞机移动数据存储器
    parameter move_x = 50; //方块的初始位置
    reg [9:0]move_y;

//  随机数
reg [7:0] rand_num;
parameter seed = 8'b1111_1111;
always@(posedge clk or negedge reset)
```

```
begin
   if(!reset)
       rand_num  <= seed;
   else
       begin
           rand_num[0] <= rand_num[1] ;
           rand_num[1] <= rand_num[2] + rand_num[7];
           rand_num[2] <= rand_num[3] + rand_num[7];
           rand_num[3] <= rand_num[4] ;
           rand_num[4] <= rand_num[5] + rand_num[7];
           rand_num[5] <= rand_num[6] + rand_num[7];
           rand_num[6] <= rand_num[7] ;
           rand_num[7] <= rand_num[0] + rand_num[7];
       end
end
wire [2:0]choose;
reg [8:0]type;
//产生随机的 choose 数值
assign choose = {rand_num[3],rand_num[2],rand_num[6],rand_num[4]};

//type 是随机的
always@(posedge clk )
begin
    case(choose)
    0:type = 0;
    1:type = 40;
    2:type = 80;
    3:type = 120;
    4:type = 160;
    5:type = 280;
    6:type = 240;
    7:type = 200;
     8:type = 0;
     9:type = 280;
     10:type = 0;
    default: type = 280;
    endcase
end


//  板块移动速度控制
reg move;
reg [32:0]counter;
reg [30:0]T_move;
```

```
reg [19:0]max_speed;
reg [10:0]max_transform;
//设置难度
always @(easy)
begin
case(easy)
    2'b00: begin max_speed <= 50_0000; max_transform <= 100; end
    2'b01: begin max_speed <= 40_0000; max_transform <= 200; end
    2'b10: begin max_speed <= 40_0000; max_transform <= 500; end
    2'b11: begin max_speed <= 30_0000; max_transform <= 1000; end
endcase
end

always@(posedge clk,negedge reset)
begin
    if(!reset)
    begin
        T_move <= 30'd100_0000;
        counter <= 0;
        move <=0;
    end
    else
    begin
        if(counter >= T_move)
        begin
            move <= 1;
                //加速到一定程度
            if(T_move <= max_speed)
                T_move <=T_move;
            else//还没有加速到最大
                T_move <= T_move-max_transform;
            counter <= 0;
        end
        else
        begin
            move <= 0;
            if(!stop)
                counter <= counter + 1;
            else
                counter <= 0;
        end
    end
end
```

```
reg [8:0]rand,rand1,rand2,rand3;
wire [1:0] choose_push1,choose_push2,choose_push3;
always@(posedge clk or negedge reset)
begin
    if (!reset)
        begin
           push <= game_x;  //初始位置设定
           push1 <= game_x +  1*magin; //板之间的间隔
           push2 <= game_x + 2*magin;
           push3 <= game_x + 3*magin;
           score <= 0;
        end
    else if (move) // 板子开始动
    begin
        if(push == 0)
            begin
                 score <= score+1;
                push <= game_x;
                rand <=type; //第一块板子的位置设定
            end
        else
            begin
                push <= push-1'b1;
            end
        if(push1 == 0)
            begin
                    score <= score+1;;
                rand1 <=type; //第二块板子的位置设定

                push1 <= game_x;
                end
        else
            begin
                push1 <= push1-1'b1;
            end

        if(push2 == 0)
        begin
           score <= score+1;
            push2 <= game_x;
            rand2 <=type; //第三块板子的位置设定
        end
        else
```

```
            begin
                push2<= push2-1'b1;
            end

        if(push3 == 0)
            begin
                score <= score+1;
                push3 <= game_x;
                rand3 <=type;
                          //第四块板子的位置设定
            end
            else
            begin
                push3 <= push3-1'b1;
            end
    end
    else
    begin
        push <= push;
        push1 <= push1;
        push2 <= push2;
        push3 <= push3;
    end
end


wire die1,die2,die3,die4;
assign die1=((rand<move_y + border)&&(move_y < rand+long)&&(push <
move_x+border) && (move_x < push + ban ));
assign die2=((rand1<move_y + border)&&(move_y < rand1+long)&&(push1 <
move_x+border) && (move_x < push1 + ban ));
assign die3=((rand2<move_y + border)&&(move_y < rand2+long)&&(push2 <
move_x+border) && (move_x < push2 + ban ));
assign die4=((rand3<move_y + border)&&(move_y < rand3+long)&&(push3 <
move_x+border) && (move_x < push3 + ban ));

wire false;
assign false = die1||die2||die3||die4;

//开始画画
always@(posedge vga_clk,negedge reset)
begin
    if(!reset)
        begin
```

```
            data <= 0;
            stop <= 0;
        end
    else
        begin
           if ( (hc>move_x)
&&(hc<(move_x+border)&&(vc>move_y)&&(vc<move_y+border))) //飞机
              begin
                  if(!false) //没有失败
                      begin
                          data <= 12'b0011_000_1100;
                          stop <= 0;
                                  if(pause) // 点击暂停了
                                     stop <= 1;
                                  else
                                     stop <= 0;
                      end
                  else
                      begin
                          data <= 12'b1100_0000_0000;
                          stop <=1;
                      end
              end
            else
                if ( (hc <=game_x) && (vc <= game_y)&& (hc>push) &&
(hc<=push+ban) && (vc>=rand) && (vc<=rand+long))
                  begin
                      data <= 12'b0100_0011_0011;  //第一根横条
                  end
                else  if ((hc <=game_x) && (vc <= game_y)&&(hc>push1) &&
(hc<=push1+ban) && (vc>=rand1) && (vc<=rand1+long))
                  begin
                      data <= 12'b0011_0011_0010;  //第二根横条
                  end
                else  if ((hc <=game_x) && (vc <= game_y)&&(hc>push2) &&
(hc<=push2+ban) && (vc>=rand2) && (vc<=rand2+long))
                  begin
                      data <= 12'b0100_0100_0010;  //第三根横条
                  end
                else  if ((hc <=game_x) && (vc <= game_y)&&(hc>push3) &&
(hc<=push3+ban) && (vc>=rand3) && (vc<=rand3+long))
                  begin
                     data <= 12'b0110_0010_0100;  //第四根横条
                  end
```

```
            else
                if( (hc <= game_x) && vc <= game_y   )
                data <= 12'b1011_1010_1111;
                else

                //画数字
                begin
                if( hc >= (game_x) && hc <= 640 && (vc >= M) &&(vc <=
120) )
                begin
                        if(vc < M+29 && vc > M && hc < N+17 && hc > N)
                        begin
            //千位数字
                            addr <= (vc - M - 1) * 256 + (hc - N - 1)
+ num3 * 16;

                            data[11:8] <= {0,data_num[7:5]};
                            data[7:4] <= {0,data_num[4:2]};
                            data[3:0] <= {0,0,data_num[1:0]};
                        end
                        else if(vc < M+29 && vc > M && hc < N+33 &&
hc > N+16)
                        begin
            //百位数字
                            addr <= (vc - M - 1) * 256 + (hc - N-16 -
1) + num2 * 16;

                            data[11:8] <= {0,data_num[7:5]};
                            data[7:4] <= {0,data_num[4:2]};
                            data[3:0] <= {0,0,data_num[1:0]};
                        end
                        else if(vc < M+29 && vc > M && hc < N+49 &&
hc > N+32)
                        begin
            //十位数字
                            addr <= (vc - M - 1) * 256 + (hc - N-32 -
1) + num1 * 16;

                            data[11:8] <= {0,data_num[7:5]};
                            data[7:4] <= {0,data_num[4:2]};
                            data[3:0] <= {0,0,data_num[1:0]};
                        end
                        else if(vc < M+29 && vc > M && hc < N+65 &&
hc > N+48)
```

```
                        begin
            //个位数字
                            addr <= (vc - M - 1  ) * 256 + (hc - N-48 -
1) + num0 * 16;

                            data[11:8] <= {0,data_num[7:5]};
                            data[7:4] <= {0,data_num[4:2]};
                            data[3:0] <= {0,0,data_num[1:0]};
                        end
                        else
                        begin
                        data[11:0] <= 'h000;
                        end
                    end
                else
                    begin

                        if(hc > N && hc <= N+64 &&vc > 10 &&vc <=42 )
                        begin
                              addr2 <= (vc -10)*64 + (hc-N-1);
                               data[11:8] <= {0,data_num2[7:5]};
                               data[7:4] <= {0,data_num2[4:2]};
                               data[3:0] <= {0,0,data_num2[1:0]};
                        end

                        else
                        begin
                               if(hc > 499 && hc <= 639 &&vc > 184 &&vc
<=474 )
                               begin
                                      addr3 <= (vc -185)*140 + (hc-499)-
94;

                                      data[11:8] <= {0,data3[7:5]};
                                      data[7:4] <= {0,data3[4:2]};
                                      data[3:0] <= {0,0,data3[1:0]};
                               end
                               else
                                   data[11:0] <= 0;

                        end

                            //data[11:0] <= 0;
                    end
                    //data[11:0] <= 'h000;
```

```
                end
        end
end


//       方块移动控制
always@(posedge clk or negedge reset)
    begin

        if (!reset)
            begin
               move_y <= 240;
            end
        else if (up_key_press)
        begin
            if(~stop)
            begin
                if(move_y == 0)
                    begin
                         move_y <= move_y;
                    end
                else
                    begin
                        move_y <= move_y-1'b1;
                    end
            end
            else move_y <= move_y;
        end
      else if (down_key_press)
        if(~stop)
            begin
                if(move_y>=(game_y - border))
                begin
                     move_y <= move_y;
                end
            else
                 begin
                    move_y <= move_y+1'b1;
                 end
            end
```

```
        else move_y <= move_y;

end
// 信号输出
assign disp_RGB = (dat_act) ? data : 'h000;

endmodule
```

背景音乐模块

```
module song(
    input clk,
     input sel,
     output out
    );
     reg [8:0] count_mus;
     reg [5:0] count;
     wire clk_4hz;

     reg [10:0]music_frequency;
     reg[3:0]high;
     reg[3:0]med;
     reg[3:0]low;

     frequencydiv frediv(clk,6,clk_4hz);

     always@(posedge clk_4hz)
     begin
        case(sel)
        1'b0:

        begin
            high <= 0;
            low  <= 0;
            med  <= 0;
            count <= 0;
            count_mus <= 0;
        end

        1'b1:
            begin
                if(count_mus < 266)
```

```
    count_mus <= count_mus+1;
else
    count_mus <= 0;
case(count_mus)
  0 : {high,med,low}<='h010;
  1 : {high,med,low}<='h007;
  2 : {high,med,low}<='h006;
  3 : {high,med,low}<='h006;
  4 : {high,med,low}<='h030;
  5 : {high,med,low}<='h030;
  6 : {high,med,low}<='h030;
  7 : {high,med,low}<='h030;
  8 : {high,med,low}<='h010;
  9 : {high,med,low}<='h006;
 10 : {high,med,low}<='h007;
 11 : {high,med,low}<='h007;
 12 : {high,med,low}<='h040;
 13 : {high,med,low}<='h040;
 14 : {high,med,low}<='h030;
 15 : {high,med,low}<='h020;
 16 : {high,med,low}<='h020;
 17 : {high,med,low}<='h010;
 18 : {high,med,low}<='h020;
 19 : {high,med,low}<='h010;
 20 : {high,med,low}<='h020;
 21 : {high,med,low}<='h030;
 22 : {high,med,low}<='h030;
 23 : {high,med,low}<='h010;
 24 : {high,med,low}<='h007;
 25 : {high,med,low}<='h006;
 26 : {high,med,low}<='h006;
 27 : {high,med,low}<='h020;
 28 : {high,med,low}<='h020;
 29 : {high,med,low}<='h010;
 30 : {high,med,low}<='h007;
 31 : {high,med,low}<='h007;
 32 : {high,med,low}<='h007;
 33 : {high,med,low}<='h010;
 34 : {high,med,low}<='h010;
 35 : {high,med,low}<='h006;
 36 : {high,med,low}<='h006;
 37 : {high,med,low}<='h030;
 38 : {high,med,low}<='h030;
 39 : {high,med,low}<='h010;
```

```
40 : {high,med,low}<='h010;
41 : {high,med,low}<='h020;
42 : {high,med,low}<='h020;
43 : {high,med,low}<='h060;
44 : {high,med,low}<='h060;
45 : {high,med,low}<='h050;
46 : {high,med,low}<='h050;
47 : {high,med,low}<='h040;
48 : {high,med,low}<='h040;
49 : {high,med,low}<='h030;
50 : {high,med,low}<='h020;
51 : {high,med,low}<='h030;
52 : {high,med,low}<='h030;
53 : {high,med,low}<='h030;
54 : {high,med,low}<='h030;
55 : {high,med,low}<='h030;
56 : {high,med,low}<='h030;
57 : {high,med,low}<='h030;
58 : {high,med,low}<='h010;
59 : {high,med,low}<='h007;
60 : {high,med,low}<='h006;
61 : {high,med,low}<='h006;
62 : {high,med,low}<='h030;
63 : {high,med,low}<='h030;
64 : {high,med,low}<='h030;
65 : {high,med,low}<='h030;
66 : {high,med,low}<='h030;
67 : {high,med,low}<='h010;
68 : {high,med,low}<='h006;
69 : {high,med,low}<='h007;
70 : {high,med,low}<='h007;
```

…… 因程序行数过多省去 100 多个音符
曲子是名侦探柯南的主题曲
……..

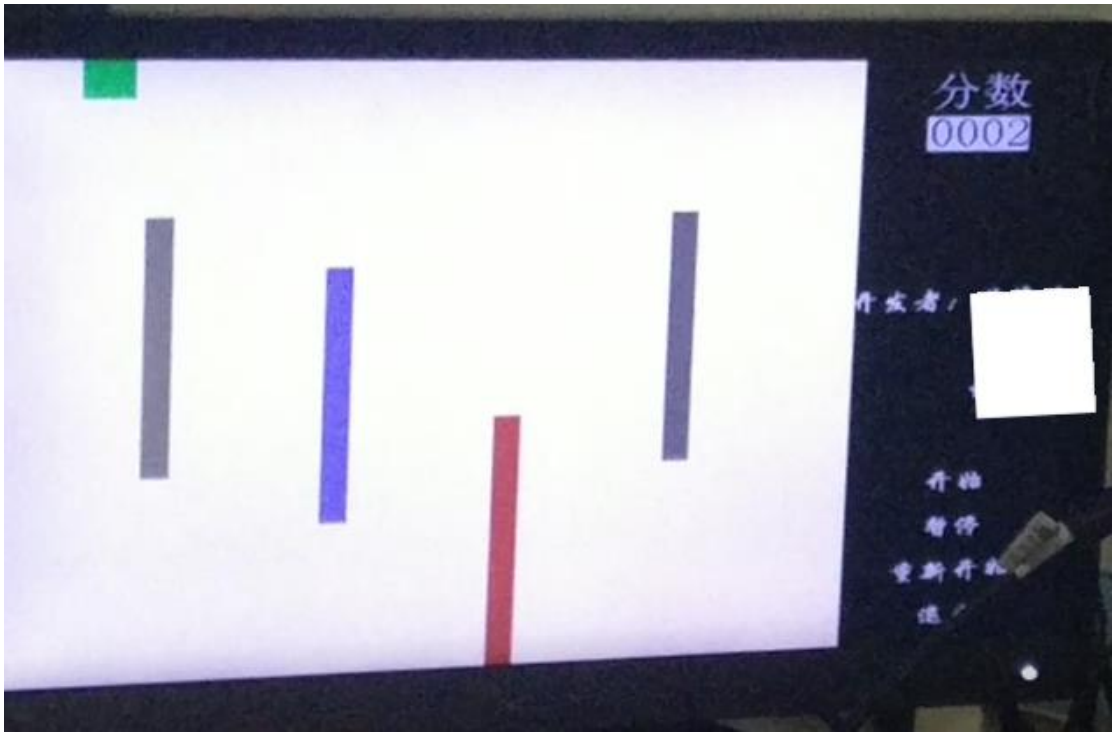
```
251 : {high,med,low}<='h050;
252 : {high,med,low}<='h050;
253 : {high,med,low}<='h060;
254 : {high,med,low}<='h060;
255 : {high,med,low}<='h060;
256 : {high,med,low}<='h060;
257 : {high,med,low}<='h060;
258 : {high,med,low}<='h060;
```

```
            259 : {high,med,low}<='h060;
            260 : {high,med,low}<='h000;
            261 : {high,med,low}<='h000;
            262 : {high,med,low}<='h000;
            263 : {high,med,low}<='h000;
            264 : {high,med,low}<='h000;
            265 : {high,med,low}<='h000;
            266 : {high,med,low}<='h000;


            default:{high,med,low}<='h000;
           endcase
       end
   default:
   begin
       high <= 0;
       low  <= 0;
       med  <= 0;
       count <= 0;
       count_mus <= 0;
   end
endcase
end

always@(posedge clk)
begin
case ({high,med,low})
 'h000:music_frequency<=0;
 'h001:music_frequency<=262;
 'h002:music_frequency<=294;
 'h003:music_frequency<=330;
 'h004:music_frequency<=349;
 'h005:music_frequency<=392;
 'h006:music_frequency<=440;
 'h007:music_frequency<=494;
 'h010:music_frequency<=532;
 'h020:music_frequency<=587;
 'h030:music_frequency<=659;
 'h040:music_frequency<=698;
 'h050:music_frequency<=784;
 'h060:music_frequency<=880;
 'h070:music_frequency<=988;
 'h100:music_frequency<=1046;
 'h200:music_frequency<=1175;
```

```
     'h300:music_frequency<=1319;
     'h400:music_frequency<=1397;
     'h500:music_frequency<=1568;
     'h600:music_frequency<=1760;
     'h700:music_frequency<=1976;
     endcase
    end

    frequencydiv frespeak(clk,music_frequency,out);
    initial begin
    high <= 0;
    low <= 0;
    med <= 0;
    count <= 0;
    music_frequency<=11'b0;
    count_mus <= 0;
    end
endmodule
```

六.实际测试结果

按顺序接入各个模块，连接 FPGA 和键盘，测试游戏各个模块功能，图片见下图，其余操作见视频附件。



七、故障与问题分析

实验过程中我们遇到了很多问题，首先是键盘的问题，我们发现小组同学自带的键盘并不能让 FPGA 识别，在 xilinx 官网上查询了才发现，支持的键盘类型不能是特别的键盘，必须是通用的键盘，但是在学校实验室键盘上测试成功了。

游戏测试过程中，发现速度增大到一定的值后就不再增大了，后来进行调整后才测试成功。

其次是我们在进行 VGA 的实验中图片是通过 ip 核的 ROM 进行储存操作的时候，发现使用的工具转换的 coe 文件有问题，因此我们进行处理，调整图片的相对位置和大小，最终做出了显示图片，显示分数的效果。

八、实验小结

经过这次实验，我们熟悉了 Verilog 游戏的设计，使用方法，VGA 的输出模块的方法，虽然遇到了很多困难，但是在不断调试的过程中也学到了很多。这次实验也让我们意识到了 FPGA 的强大之处，支持很多功能，我们使用的只是冰山一角。在存储 rom 的时候，我们遇到了很多问题，如 rom 中存储的图片只支持.coe 格式，而我们通过 bmp2mif 软件转换出的 coe 格式的文件，存在显示乱码，我们进行了大量的参数调试，才将背景图完整地显示在屏幕上，并精确定位图片显示在屏幕的位置。