同济大学计算机系

数字逻辑课程综合实验报告

学　　号_____　　_____

姓　　名_____　　_____

专　　业_____计算机科学与技术_____

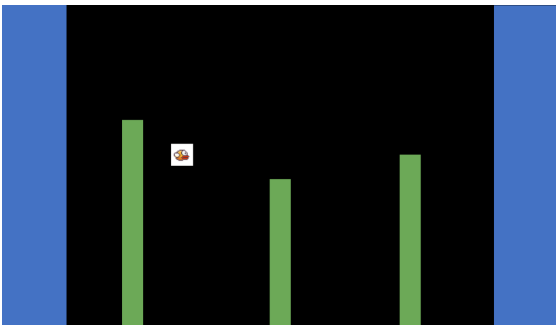
授课老师_____张冬冬_____

一、实验内容

1. 项目内容概括

通过 9”TFT 显示屏与 NEXYS 4 DDR 开发板实现 VGA 小游戏：FlappyBird。同时，实现这一项目的过程中，用到了包括三态门、分频器、译码器、编码器、七段数字管、ROM、控制器等所学内容。

2. 游戏界面展示



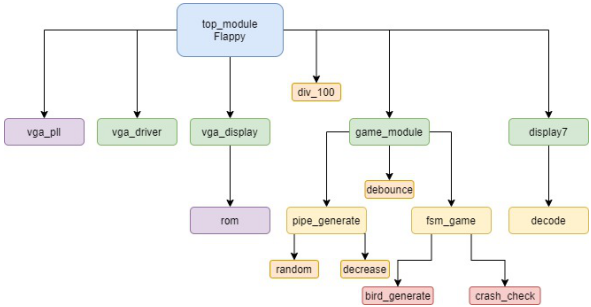
3. 游戏操作说明

玩家通过板卡上的复位按键进入游戏，按下游戏开始按键以进入游戏状态。游戏过程中，可以通过板卡上的按键控制小鸟在显示屏上的运动，玩家需要控制小鸟跳跃管道，碰到管道游戏失败，板卡上七段数码管实现计分的功能，并在游戏结束后显示相应的提醒内容。进行复位后可以再度开始游戏。

4. 器件简介

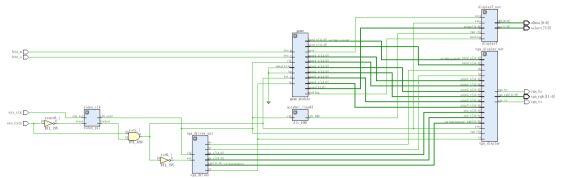
- Nexys DDR Artix-7——由 Xilinx 公司开发的一款现场可编程门阵列（FPGA）开发板
- 9”TFT VGA 显示屏。VGA——（Video Graphics Array）是 IBM 在 1987 年随 PS/2 机一同推出的视频传输标准，具有分辨率高、显示速率快、颜色丰富的优点。

二、系统框图

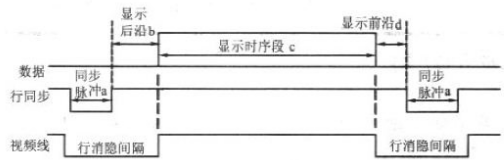


说明：

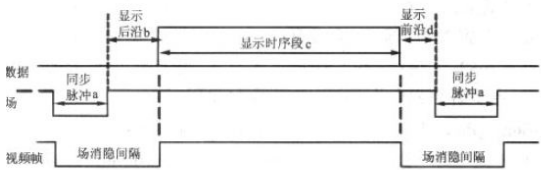
紫色模块表示调用了系统 IP 核，本次作业使用了 vivado 提供的 ROM 和 PLL 两个 IP 核。其中 ROM 用来存放小鸟图像的 coe 文件，PLL 用来将系统 100MHz 的时钟转换为 65MHz。



三、VGA 信号显示原理



VGA 的行时序



VGA 的场时序

VGA 利用了人眼视觉残留效应，实际上每一个像素点都是依次亮起，从左至右，再从上至下，显示出完整的一张图像。显示器扫描方式为逐行扫描：从屏幕左上角一点开始，从左向右逐点扫描，每扫描完一行,电子束回到屏幕的左边下一行的起始位置。每行结束时，用行同步信号进行同步；当扫描完所有的行，形成一帧，用场同步信号进行场同步，并使扫描回到屏幕左上方，开始下一帧

完成一行扫描的时间称为水平扫描时间，其倒数称为行频率；完成一帧（整屏）扫描的时间称为垂直扫描时间，其倒数称为场频率，即刷新一屏的频率，常见的有 60Hz，75Hz 等等。标准的 VGA 显示的场频 60Hz,行频 31.5KHz。本次实验用到的 VGA 场频率为 65MHz。

VGA 标准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。由 VGA 的行时序可知：每一行都有一个行同步脉冲（Sync a），是数据行的结束标志，同时也是下一行的开始标志。在同步脉冲之后为显示后沿（Back porch b），在显示时序段（Display interval c）显示器为亮的过程，RGB 数据驱动一行上的每一个像素点，从而显示一行。在一行的最后为显示前沿（Front porch d）。在显示时间段（Display interval c）之外没有图像投射到屏幕。

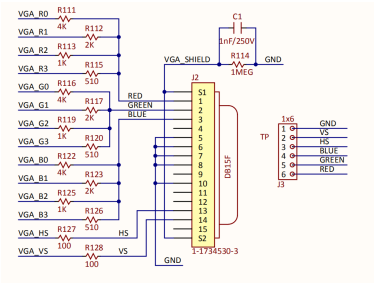
同步脉冲(Sync a)、显示后沿（Back porch b）和显示前沿（Front porch d）都是在 RGB 信号无效数据段，屏幕不显示数据。

VGA 的场时序与行时序基本一样，每一帧的场同步信号（Sync a）是一帧的结束标志，同时也是下一帧的开始标志。而显示数据是一帧的所有行数据。

我查阅了分辨率与刷新率对应的相关时序数据：

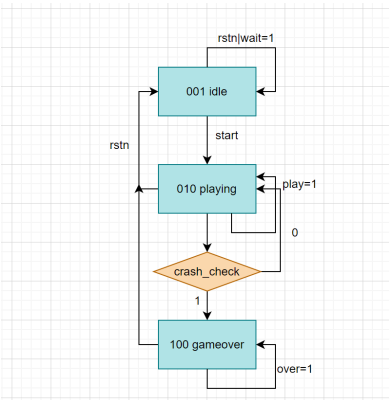
显示模式	时钟	行时序（像素数）					帧时序（行数）				
	（MHz）	a	b	c	d	e	o	p	q	r	s
640x480@60	25.175	96	48	640	16	800	2	33	480	10	525
640x480@75	31.5	64	120	640	16	840	3	16	480	1	500
800x600@60	40.0	128	88	800	40	1056	4	23	600	1	628
800x600@75	49.5	80	160	800	16	1056	3	21	600	1	625
1024x768@60	65	136	160	1024	24	1344	6	29	768	3	806
1024x768@75	78.8	176	176	1024	16	1312	3	28	768	1	800
1280x1024@60	108.0	112	248	1280	48	1688	3	38	1024	1	1066
1280x800@60	83.46	136	200	1280	64	1680	3	24	800	1	828
1440x900@60	106.47	152	232	1440	80	1904	3	28	900	1	932

学校提供的 VGA 显示器分辨率为 1024*768，可以根据上表设置相关时序参数。



上图为 NEXYS 4DDR 板卡接线原理图，每一个像素点的颜色数据由 RGB4:4:4 模式给出，即一共 12 位宽的颜色数据，此外还需连接显示器，给出 VS、HS 行场同步信号。

四、状态机模块设计



说明：为了便于查看将状态名称与状态编码写入状态框内。

1)编码为 001 即 idle 状态，此时为空闲状态，wait 信号置 1，play 和 over 信号置 0。当输入 start=1 信号时，进入游戏 playing 状态，否则保持该状态。

2)编码为 010，即 playing 状态，此时为正在游戏的状态，play 信号置 1，wait 和 over 信号置 0，并进行 crash_check 的检测，如果小鸟撞击到障碍物，置 play=0，进入 gameover 状态。

3)编码为 100，gameover 状态，游戏结束，over 信号置 1，play 和 wait 信号置 0。

4)以上三种状态均可以通过 rstn 复位信号回到 idle 态。

状态机模块代码如下：

```
module fsm_game(
    input               clk,
    input               rst_n,
    input               vs,
    input               start,
    input               move,                   //1 ip  ,0 down
    input  [11:0]       pipe1_x,                //pipe1 x
    input  [11:0]       pipe1_y,                //pipe1 y
    input  [11:0]       pipe2_x,                //pipe2 x
    input  [11:0]       pipe2_y,                //pipe2 y
    input  [11:0]       pipe3_x,                //pipe3 x
    input  [11:0]       pipe3_y,                //pipe3 y

    output  [11:0]      bird_x,
    output  [11:0]      bird_y,
    output  reg[11:0]  score,
    output  reg         over,
    output  reg         play,
    output  reg         waiting
    );

    localparam idle     = 4'b0000;
    localparam playing  = 4'b0001;
    localparam gameover = 4'b0010;
    reg [3:0]   state;
    reg [3:0]   next_state;
    reg         pass_d;
    wire        crash;
    wire        pass;
    wire        passp;
assign passp = pass &~pass_d;
always@(posedge clk)
begin
    pass_d <= pass;
end
always@(posedge clk or negedge rst_n)
begin
    if(~rst_n)
        state   <=  4'd0;
    else
        state   <=  next_state;
end
always@(*)
begin
    case(state)
    idle    :   if(start)       next_state  <=  playing;
                else            next_state  <=  idle;
    playing :   if(crash)       next_state  <= gameover;
                else            next_state  <=  playing;
    gameover:                   next_state  <= gameover;
    default :                   next_state  <=  idle;
    endcase
end
always@(posedge clk or negedge rst_n)
begin
    if(~rst_n)                  begin   over <= 1'b0;   play <= 1'b0;    waiting <= 1'b0;   end
    else if(state == idle)      begin   over <= 1'b0;   play <= 1'b0;    waiting <= 1'b1;   end
    else if(state == playing)   begin   over <= 1'b0;   play <= 1'b1;    waiting <= 1'b0;   end
    else if(state == gameover)  begin   over <= 1'b1;   play <= 1'b0;    waiting <= 1'b0;   end
    else                        begin   over <= over;   play <= play;    waiting <= waiting;   end
end

always@(posedge clk or negedge rst_n)
begin
    if(~rst_n)                          score <= 12'b0;
    else if(state == playing & passp)   score <= score + 1'b1;
```

```
    else                                    score <= score;
end

//////////小鸟生成模块//////////////////////////////////////
    bird_generate bird_generate_uut(
     //input
    .clk(clk),
    .rstn(rst_n),
    .en(vs&play),
    .move(move),
    //output
    .bird_x(bird_x),
    .bird_y(bird_y)
    );

    //////////撞击检测模块//////////////////////////////////////
    crash_check crash_check_uut(
    //input
    .bird_x(bird_x),
    .bird_y(bird_y),
    .pipe1_x(pipe1_x),
    .pipe1_y(pipe1_y),
    .pipe2_x(pipe2_x),
    .pipe2_y(pipe2_y),
    .pipe3_x(pipe3_x),
    .pipe3_y(pipe3_y),
    //output
    .pass(pass),
    .crash(crash)
    );
endmodule
```

五、模块设计

1）顶层模块

```
module flappy(input sys_clk,//系统时钟 100MHz
              input sys_rstn,//外部复位按键
              input btn_m,                //控制小鸟移动按键，低电平 up高电平 down
              input btn_s,                //game start
              output vga_hs,//行同步
              output vga_vs,//场同步
              output [11:0]vga_rgb,//rgb444
              output [6:0]oData,//七段数码管显示数据
              output [7:0]select);//七段数码管片选信号

    wire vga_clk;//分频 65MHz 时钟
    wire locked;//分频时钟稳定信号
    wire rstn;//内部复位信号
    wire [11:0] px_data;//像素点颜色 rgb444
    wire [11:0] px_x;//像素点 x 坐标
    wire [11:0] px_y;//像素点 y 坐标
    wire [11:0] score;//得分
    wire gameover;
    wire [11:0] bird_y;
    wire [11:0] bird_x;
    wire [11:0] pipe1_x;
    wire [11:0] pipe1_y;
    wire [11:0] pipe2_x;
    wire [11:0] pipe2_y;
    wire [11:0] pipe3_x;
    wire [11:0] pipe3_y;


    wire        hs;
    wire        vs;
    wire        de;
    wire [11:0] rgb;
    wire        waiting;
    wire        over;
    wire        clk_100;
    assign rstn = sys_rstn&&locked;//时钟信号稳定后，复位信号有效

    //////////65MHz 时钟模块//////////////////////////////////////
    video_pll video_clk(
     // Clock in ports
      .clk_in1(sys_clk),
    .reset(~sys_rstn),
    //output
    .clk_out1(vga_clk),
    .locked(locked)
    );
    div_100 div_100_uut(.clk(vga_clk),.clk_100(clk_100));
```

```
    //////////vga驱动模块//////////////////////////////////
    vga_driver vga_driver_uut(
    //input
    .clk(vga_clk),
    .rst(~rstn),
    //output
    .hs(hs),
    .vs(vs),
    .de(de),
    .rgb(rgb),
    .px_x(px_x),
    .px_y(px_y)
    );
    ////////////////////game module/////////////////////////
    game_module game(
            .clk(vga_clk),
            .rst_n(rstn),
            .btn_m(btn_m),//跳跃按键
            .btn_s(btn_s),//开始按键
            .vs(vs),
            .bird_x(bird_x),
            .bird_y(bird_y),
            .pipe1_x(pipe1_x),
            .pipe1_y(pipe1_y),
            .pipe2_x(pipe2_x),
            .pipe2_y(pipe2_y),
            .pipe3_x(pipe3_x),
            .pipe3_y(pipe3_y),
            .score(score),
            .waiting(waiting),
            .over(over)
        );

    //////////vga显示模块//////////////////////////////////
    vga_display vga_display_uut(
    //input
    .vga_clk(vga_clk),
    .rstn(rstn),
    .hs(hs),
    .vs(vs),
    .de(de),
    .rgb(rgb),
    .pix_x(px_x),
    .pix_y(px_y),
    .bird_x(bird_x),
    .bird_y(bird_y),
    .pipe1_x(pipe1_x),
    .pipe1_y(pipe1_y),
    .pipe2_x(pipe2_x),
    .pipe2_y(pipe2_y),
    .pipe3_x(pipe3_x),
    .pipe3_y(pipe3_y),
    //output
    .vga_hs(vga_hs),
    .vga_vs(vga_vs),
    .vga_rgb(vga_rgb)
    );



    //////////7段数码管模块//////////////////////////////////
    display7 display7_uut(
    //input
    .waiting(waiting),
    .over(over),
    .vga_clk(clk_100),
    .rstn(rstn),
    .score(score),
    //output
    .HEX(dData),
    .sel(select)
    );
endmodule
```

2）vga_driver 模块：用于驱动 vga 显示屏，传输行场同步信号与像素点坐标、颜色信息。

```
module vga_driver(
    input                clk,           //pixel clock
    input                rst,           //reset signal high active
    output               hs,            //horizontal synchronization
    output               vs,            //vertical synchronization
    output               de,            //video valid
    output[11:0]         rgb,            //video  data
    output[11:0]         px_x,
    output[11:0]         px_y
```

```
);

parameter H_ACTIVE = 16'd1024;
parameter H_FP = 16'd24;
parameter H_SYNC = 16'd136;
parameter H_BP = 16'd160;
parameter V_ACTIVE = 16'd768;
parameter V_FP  = 16'd3;
parameter V_SYNC  = 16'd6;
parameter V_BP  = 16'd29;
parameter HS_POL = 1'b0;
parameter VS_POL = 1'b0;
parameter H_TOTAL = H_ACTIVE + H_FP + H_SYNC + H_BP;//horizontal total time (pixels)
parameter V_TOTAL = V_ACTIVE + V_FP + V_SYNC + V_BP;//vertical total time (lines)
reg hs_reg;                         //horizontal sync register
reg vs_reg;                         //vertical sync register
reg hs_reg_d0;                      //delay 1 clock of 'hs_reg'
reg vs_reg_d0;                      //delay 1 clock of 'vs_reg'
reg[11:0] pix_x;
reg[11:0] pix_y;
reg[11:0] h_cnt;                    //horizontal counter
reg[11:0] v_cnt;                    //vertical counter
reg[11:0] active_x;                 //video x position
reg[11:0] active_y;                 //video y position
reg[7:0] rgb_r_reg;                 //video red data register
reg[7:0] rgb_g_reg;                 //video green data register
reg[7:0] rgb_b_reg;                 //video blue data register
reg h_active;                       //horizontal video active
reg v_active;                       //vertical video active
wire video_active;                  //video active(horizontal active and vertical active)
reg video_active_d0;                //delay 1 clock of video_active
wire hs_p,vs_p;
assign hs_p = hs_reg & ~hs_reg_d0;
assign vs_p = vs_reg & ~vs_reg_d0;
assign hs = hs_reg_d0;
assign vs = vs_reg_d0;
assign video_active = h_active & v_active;
assign de = video_active_d0;
assign rgb = 12'h555;
assign px_x = pix_x;
assign px_y = pix_y;
always@(posedge clk or posedge rst)
begin
    if(rst)         pix_x   <=  12'd0;
    else if(de)     pix_x   <= pix_x + 1'b1;
    else if(hs_p)   pix_x   <=  12'd0;
    else            pix_x   <= pix_x;
end
always@(posedge clk or posedge rst)
begin
    if(rst)                     pix_y   <=  12'd0;
    else if(hs_p & v_active)    pix_y   <= pix_y + 1'b1;
    else if(vs_p)               pix_y   <= 12'd0;
    else                        pix_y   <= pix_y;
end

always@(posedge clk or posedge rst)
begin
    if(rst == 1'b1)
        begin
            hs_reg_d0 <= 1'b0;
            vs_reg_d0 <= 1'b0;
            video_active_d0 <= 1'b0;
        end
    else
        begin
            hs_reg_d0 <= hs_reg;
            vs_reg_d0 <= vs_reg;
            video_active_d0 <= video_active;
        end
end

always@(posedge clk or posedge rst)
begin
    if(rst == 1'b1)
        h_cnt <= 12'd0;
    else if(h_cnt == H_TOTAL - 1)//horizontal counter maximum value
        h_cnt <= 12'd0;
    else
        h_cnt <= h_cnt + 12'd1;
end

always@(posedge clk or posedge rst)
begin
    if(rst == 1'b1)
        active_x <= 12'd0;
    else if(h_cnt >= H_FP + H_SYNC + H_BP - 1)//horizontal video active
```

```
        active_x <= h_cnt - (H_FP[11:0] + H_SYNC[11:0] + H_BP[11:0] - 12'd1);
    else
        active_x <= active_x;
end

always@(posedge clk or posedge rst)
begin
    if(rst == 1'b1)
        v_cnt <= 12'd0;
    else if(h_cnt == H_FP  - 1)//horizontal sync time
        if(v_cnt == V_TOTAL - 1)//vertical counter maximum value
            v_cnt <= 12'd0;
        else
            v_cnt <= v_cnt + 12'd1;
    else
        v_cnt <= v_cnt;
end

always@(posedge clk or posedge rst)
begin
    if(rst == 1'b1)
        hs_reg <= 1'b0;
    else if(h_cnt == H_FP - 1)//horizontal sync begin
        hs_reg <= HS_POL;
    else if(h_cnt == H_FP + H_SYNC - 1)//horizontal sync end
        hs_reg <= ~hs_reg;
    else
        hs_reg <= hs_reg;
end

always@(posedge clk or posedge rst)
begin
    if(rst == 1'b1)
        h_active <= 1'b0;
    else if(h_cnt == H_FP + H_SYNC + H_BP - 1)//horizontal active begin
        h_active <= 1'b1;
    else if(h_cnt == H_TOTAL - 1)//horizontal active end
        h_active <= 1'b0;
    else
        h_active <= h_active;
end

always@(posedge clk or posedge rst)
begin
    if(rst == 1'b1)
        vs_reg <= 1'd0;
    else if((v_cnt == V_FP - 1) && (h_cnt == H_FP - 1))//vertical sync begin
        vs_reg <= HS_POL;
    else if((v_cnt == V_FP + V_SYNC - 1) && (h_cnt == H_FP - 1))//vertical sync end
        vs_reg <= ~vs_reg;
    else
        vs_reg <= vs_reg;
end

always@(posedge clk or posedge rst)
begin
    if(rst == 1'b1)
        v_active <= 1'd0;
    else if((v_cnt == V_FP + V_SYNC + V_BP - 1) && (h_cnt == H_FP - 1))//vertical active begin
        v_active <= 1'b1;
    else if((v_cnt == V_TOTAL - 1) && (h_cnt == H_FP - 1)) //vertical active end
        v_active <= 1'b0;
    else
        v_active <= v_active;
End
endmodule
```

3）vga_display模块：对像素点坐标与颜色信息进行设置。

```
module vga_display
    #(          parameter BIRD_HEIGHT    = 12'd40,
                parameter BIRD_WIDTH     = 12'd40,
                parameter PIPE_WIDTH     = 12'd60,
                parameter IMG_HEIGHT     = 12'd768,
                parameter IMG_WIDTH      = 12'd1024
     )(
        //input
        input           vga_clk,
        input           rstn,
        input           hs,
        input           vs,
        input           de,
        input   [11:0]  rgb,
        input   [11:0]  pix_x,
```

```
        input   [11:0]  pix_y,
        input   [11:0]  bird_x,
        input   [11:0]  bird_y,
        input   [11:0]  pipe1_x,
        input   [11:0]  pipe1_y,
        input   [11:0]  pipe2_x,
        input   [11:0]  pipe2_y,
        input   [11:0]  pipe3_x,
        input   [11:0]  pipe3_y,
        //output
        output          vga_hs,
        output          vga_vs,
        output  [11:0]  vga_rgb
);
reg         hs_d0;
reg         hs_d1;
reg         hs_d2;
reg         vs_d0;
reg         vs_d1;
reg         vs_d2;
reg [11:0]  rgb_d0;
reg [11:0]  rgb_d1;
reg [11:0]  rgb_d2;
reg [11:0]  addr;
reg [11:0]  t_data;
reg [2:0]   flag = 0;
reg [2:0]   flag_d0;
reg [2:0]   flag_d1;
wire        vs_p;
wire [23:0]  data;
assign vs_p = vs & ~vs_d0;
assign vga_hs   = hs_d2;
assign vga_vs   = vs_d2;
assign vga_rgb  = t_data;
always@(posedge vga_clk)
begin
    hs_d0   <= hs;
    hs_d1   <= hs_d0;
    hs_d2   <= hs_d1;
    vs_d0   <= vs;
    vs_d1   <= vs_d0;
    vs_d2   <= vs_d1;
    rgb_d0  <= rgb;
    rgb_d1  <= rgb_d0;
    rgb_d2  <= rgb_d2;
    flag_d0 <= flag;
    flag_d1 <= flag_d0;
end
always@(posedge vga_clk)
begin
    if(pix_x >= bird_x  && pix_x <= bird_x + BIRD_WIDTH - 1'b1 && pix_y >= bird_y && pix_y <= bird_y
 + BIRD_HEIGHT - 1'b1)
        flag <= 3'b1;
    else if(pix_x >= pipe1_x  && pix_x <= pipe1_x + PIPE_WIDTH - 1'b1 && pix_y >= IMG_HEIGHT - pipe1
_y && pix_y <= IMG_HEIGHT - 1'b1)
        flag <= 3'd2;
    else if(pix_x >= pipe2_x  && pix_x <= pipe2_x + PIPE_WIDTH - 1'b1 && pix_y >= IMG_HEIGHT - pipe2
_y && pix_y <= IMG_HEIGHT - 1'b1)
        flag <= 3'd2;
    else if(pix_x >= pipe3_x  && pix_x <= pipe3_x + PIPE_WIDTH - 1'b1 && pix_y >= IMG_HEIGHT - pipe3
_y && pix_y <= IMG_HEIGHT - 1'b1)
        flag <= 3'd2;
    else
        flag <= 3'd0;
end

always@(posedge vga_clk or negedge rstn)
begin
    if(~rstn)                   addr    <=  12'd0;
    else if(vs_p)               addr    <= 12'd0;
    else if(flag == 3'd1)       addr    <=  addr+1'b1;
    else                        addr    <=  addr;
end

always@(posedge vga_clk)
begin
    if(flag_d1 == 1)        t_data  <=   {data[23:20],data[15:12],data[3:0]};
    else if(flag_d1 == 2)   t_data  <=   12'h0f0;
    else                    t_data  <=   rgb_d1;
end
    blk_mem_gen_0 osd_rom_m0 (                                          //delay 2 clk
    .clka                   (vga_clk),
    .addra                  (addr   ),
    .douta                  (data   )
);
endmodule
```

4）game_module：游戏模块，包括按键操控小鸟的实现以及游戏状态机。

```
module game_module(
        input               clk,
        input               rst_n,
        input               btn_m,                          //button ,control bird height
        input               btn_s,
        input               vs,hs,
        input   [11:0]      data,

        output  [11:0]      pipe1_x,                //pipe1 x
        output  [11:0]      pipe1_y,                //pipe1 y
        output  [11:0]      pipe2_x,                //pipe2 x
        output  [11:0]      pipe2_y,                //pipe2 y
        output  [11:0]      pipe3_x,                //pipe3 x
        output  [11:0]      pipe3_y,                //pipe3 y
        output  [11:0]      bird_x,                 //bird x
        output  [11:0]      bird_y,                 //bird y
        //output  reg         o_vs,o_hs,
        //output  reg [11:0]  o_data,
        output  [11:0]      score,
        output  wire        waiting,
        output  wire        over
    );
    wire    move;
    wire    start;
    wire    playing;

    ///////////防抖动/////////////////////////////////////////////////
    debounce#(.FREQ(65))debounce_uut0(.clk(clk),.rst(~rst_n),.button_in(btn_m),.button_posedge(move))
;
    debounce#(.FREQ(65))debounce_uut1(.clk(clk),.rst(~rst_n),.button_in(btn_s),.button_posedge(start)
);
    ///////////水管生成模块/////////////////////////////////
    pipe_generate pipe_generate_uut(
    //input
    .clk(clk),
    .rstn(rst_n),
    .en(vs),
    .start(playing),
    //output
    .pipe1_x(pipe1_x),
    .pipe1_y(pipe1_y),
    .pipe2_x(pipe2_x),
    .pipe2_y(pipe2_y),
    .pipe3_x(pipe3_x),
    .pipe3_y(pipe3_y)
    );

    //========游戏状态机============================//
    fsm_game fsm(
        .clk(clk),
        .rst_n(rst_n),
        .vs(vs),
        .start(start),
        .move(move),                    //1 ip  ,0 down
        .pipe1_x(pipe1_x),
        .pipe1_y(pipe1_y),
        .pipe2_x(pipe2_x),
        .pipe2_y(pipe2_y),
        .pipe3_x(pipe3_x),
        .pipe3_y(pipe3_y),

        .bird_x(bird_x),
        .bird_y(bird_y),
        .score(score),
        .over(over),
        .play(playing),
        .waiting(waiting)
        );

endmodule
```

5）display7：七段数码管显示游戏分数。

```
////////////////////模块功能：实现七段数码管计分/////////////////////\
//根据 score 的数值输出片选信号与每片的十进制数据
module display7 (input vga_clk,
                 input rstn,
                 input  waiting,
                 input  over,
                 input [11:0] score,
                 output [6:0] HEX,
                 output [7:0] sec);
```

```
    reg [3:0] data[0:2];//score个十???
    reg [2:0] count;
    reg [6:0] oData;
    reg [7:0] select;
    reg [7:0] shift = 8'b1111_1110;
    reg [6:0] temp;
    wire[6:0] ge;
    wire[6:0] shi;
    wire[6:0] bai;
assign HEX = temp;
assign sec = shift;
always@(posedge vga_clk or negedge rstn)
begin
    if(~rstn)  shift   <= 8'b1111_1110;
    else        shift   <= {shift[6:0],shift[7]};
end
always@(*)
begin
    if(waiting)
        case(shift)
        8'b1111_1110 :  temp    <=  7'b0111111;             //t
        8'b1111_1101 :  temp    <=  7'b0111111;             //r
        8'b1111_1011 :  temp    <=  7'b0111111;             //A
        8'b1111_0111 :  temp    <=  7'b0111111;             //t
        8'b1110_1111 :  temp    <=  7'b0111111;             //S
        default      :  temp    <=  7'b0111111;             //-
        endcase
     else if(over)
        case(shift)
         8'b1111_1110 :  temp    <=  7'b0000110;             //r
         8'b1111_1101 :  temp    <=  7'b0110000;             //v
         8'b1111_1011 :  temp    <=  7'b0000110;             //E
         8'b1111_0111 :  temp    <=  7'b0110000;             //o
         8'b1110_1111 :  temp    <=  7'b0000110;             //e
         8'b1101_1111 :  temp    <=  7'b0110000;             //-
         8'b1011_1111 :  temp    <=  7'b0000110;             //a
         8'b0111_1111 :  temp    <=  7'b0110000;             //g
         default      :  temp    <=  7'b0111111;             //-
         endcase
     else if((~waiting) & (~over))
         case(shift)
         8'b1111_1110 :  temp    <=  ge;
         8'b1111_1101 :  temp    <=  shi;
         8'b1111_1011 :  temp    <=  bai;
         8'b1111_0111 :  temp    <=  7'b0111111;             //-
         8'b1110_1111 :  temp    <=  7'b0111111;             //-
         8'b1101_1111 :  temp    <=  7'b0111111;             //-
         8'b1011_1111 :  temp    <=  7'b0111111;             //-
         8'b0111_1111 :  temp    <=  7'b0111111;             //-
         default      :  temp    <=  7'b0111111;             //-
         endcase
    else
         temp    <=  7'b0111111;             //-

end
reg[3:0]    data1,data2,data3;
    always@(*)begin
    data1 = score%10;
    data2 = (score/10)%10;
    data3 = (score/100)%10;
end
decode gewei(.a(data1),.b(ge));
decode shiwei(.a(data2),.b(shi));
decode baiwei(.a(data3),.b(bai));
endmodule
```

6）div_100 模块：产生七段数码管所需的 1000Hz 时钟。

```
module div_100(
    input   clk,
    output  reg clk_100
    );
    reg [19:0] cnt;
    always@(posedge clk )
    begin
        if(cnt == 64999)   cnt <= 20'd0;
        else                cnt <= cnt +1'b1;
    end

    always@(posedge clk )
    begin
        if(cnt == 64999)   clk_100 <= ~clk_100;
        else                clk_100 <= clk_100;
    end
endmodule
```

7）decode 模块：数码管译码器。

```
module decode(
    input   [3:0]   a,
    output  [6:0]   b
    );
    reg [6:0]   oData;
    assign b = oData;
        always@(*)begin
        case(a)
            4'h0:oData    = 7'b1000000;
            4'h1:oData    = 7'b1111001;
            4'h2:oData    = 7'b0100100;
            4'h3:oData    = 7'b0110000;
            4'h4:oData    = 7'b0011001;
            4'h5:oData    = 7'b0010010;
            4'h6:oData    = 7'b0000010;
            4'h7:oData    = 7'b1111000;
            4'h8:oData    = 7'b0000000;
            4'h9:oData    = 7'b0010000;
            4'ha:oData    = 7'b1110111;
            4'hb:oData    = 7'b0000011;
            4'hc:oData    = 7'b1000110;
            4'hd:oData    = 7'b1000001;
            4'he:oData    = 7'b0000110;
            4'hf:oData    = 7'b0001110;
            default:oData = 7'b0000000;
        endcase
        end
endmodule
```

8）pipe_generate 模块：生成水管并且左移。

```
//////////////////////////////模块功能：生成管道并左移//////////////////////
module pipe_generate(input          clk,
                     input          rstn,
                     input          en,
                     input          start,
                     output  [11:0] pipe1_x,
                     output  [11:0] pipe1_y,
                     output  [11:0] pipe2_x,
                     output  [11:0] pipe2_y,
                     output  [11:0] pipe3_x,
                     output  [11:0] pipe3_y
);
wire change1,change2,change3;
random #(.seed(10'd699) )pipy_gen1(.clk(clk),.rst_n(rstn),.en(change1 & start),.rand(pipe1_y));
random #(.seed(10'd799) )pipy_gen2(.clk(clk),.rst_n(rstn),.en(change2 & start),.rand(pipe2_y));
random #(.seed(10'd299) )pipy_gen3(.clk(clk),.rst_n(rstn),.en(change3 & start),.rand(pipe3_y));
decrease #(.seed(10'd999) )pipx_gen1(.clk(clk),.rst_n(rstn),.en(en & start),.change(change1),.data(p
ipe1_x));
decrease #(.seed(10'd599) )pipx_gen2(.clk(clk),.rst_n(rstn),.en(en & start),.change(change2),.data(p
ipe2_x));
decrease #(.seed(10'd199) )pipx_gen3(.clk(clk),.rst_n(rstn),.en(en & start),.change(change3),.data(p
ipe3_x));
endmodule
```

9）bird_generate 模块：给小鸟附加跳跃控制上升，否则自由下落的属性。

```
module bird_generate(
    //input
    input               clk,
    input               rstn,
    input               en,
    input               move,
    //output
    output    [11:0]    bird_x,
    output    [11:0]    bird_y
    );
    reg                 en_d;
    reg     [11:0]      bird_y_temp;
    wire                en_p;
    assign en_p     = en & ~en_d;
    assign bird_x   = 12'd500;
    assign bird_y   = bird_y_temp;
    always@(posedge clk)
    begin
        en_d <= en;
    end
    always@(posedge clk or negedge rstn)
    begin
        if(~rstn)
            bird_y_temp  <= 12'd380;
        else if(move)
            bird_y_temp  <=  bird_y_temp - 12'd70;
```

```
        else if(en_p)
            bird_y_temp  <=  bird_y_temp + 12'd3;
        else
            bird_y_temp  <=  bird_y_temp;
    end
endmodule
```

10）Crash_check 模块：每一帧刷新后检查小鸟像素是否与水管像素重叠，从而反馈给状态机，决定是否进入 gameover 状态。

```
//////////////////////////////模块功能：判断是否小鸟与管道相撞//////////////////////
module crash_check
    #(                  parameter BIRD_HEIGHT    = 12'd40,
                        parameter BIRD_WIDTH     = 12'd40,
                        parameter PIPE_WIDTH     = 12'd60,
                        parameter IMG_HEIGHT     = 12'd768,
                        parameter IMG_WIDTH      = 12'd1024


    )(
                    input [11:0]     bird_x,
                    input [11:0]     bird_y,
                    input [11:0]     pipe1_x,
                    input [11:0]     pipe2_x,
                    input [11:0]     pipe3_x,
                    input [11:0]     pipe1_y,
                    input [11:0]     pipe2_y,
                    input [11:0]     pipe3_y,
                    output reg       crash,
                    output           pass);

    ////////////////////////////////MAIN CODE/////////////////////////////////////////////////////////////////
//////////
always@(*)
begin
        if(bird_x + BIRD_WIDTH>= pipe1_x & bird_x <= pipe1_x + PIPE_WIDTH )
           if(bird_y >= IMG_HEIGHT - pipe1_y )      crash = 1'b1;
           else                                     crash = 1'b0;
    else if(bird_x + BIRD_WIDTH>= pipe2_x & bird_x <= pipe2_x + PIPE_WIDTH )
           if(bird_y >= IMG_HEIGHT - pipe2_y )      crash = 1'b1;
           else                                     crash = 1'b0;
    else if(bird_x + BIRD_WIDTH>= pipe3_x & bird_x <= pipe3_x + PIPE_WIDTH )
           if(bird_y >= IMG_HEIGHT - pipe3_y )      crash = 1'b1;
           else                                     crash = 1'b0;
    else if(bird_y == IMG_HEIGHT )                 crash = 1'b1;
    else                                             crash = 1'b0;
end
assign pass = (bird_x == pipe1_x + PIPE_WIDTH  )|
              (bird_x == pipe2_x + PIPE_WIDTH  )|
              (bird_x == pipe3_x + PIPE_WIDTH  );
endmodule
```

11）Random 模块：产生随机数。

```
///////////////模块功能：实现随机数/////////////////////////////
module random
            #(
            parameter seed = 12'b0000_1111_1111
            )
            (input          clk,
             input          rst_n,
             input          en,
             output  [11:0] rand);

    reg [11:0] rand_r;
    reg [11:0] rand_new;
    reg        en_d;
    wire       en_p;
    wire       w0;
    wire       w1;
    assign w0 = rand_r[11]^~rand_r[9];
    assign w1 = rand_r[0]^rand_r[10];
    assign rand   = {2'b0,rand_r[8:0]};
    initial rand_r = seed;

    always@(*)
    begin
        rand_new = {rand_r[10:0],w0};
    end
    always@(posedge clk)
    begin
        en_d <= en;
    end
```

```
    assign en_p = en & ~ en_d;
    always@(posedge clk)
    begin
        if(~rst_n)
            rand_r <= seed;
        else if(en_p)
            rand_r <= rand_new;
    end

endmodule
```

12）Decrease模块：利用Random模块产生的随机数，使得水管具有高低变化。

```
module decrease
    #(
        parameter seed  =  500
    )(
        input               clk,
        input               rst_n,
        input               en,
        output reg          change,
        output reg [11:0]  data
    );
    reg en_d;
    wire en_p;

always@(posedge clk)
begin
    en_d <= en;
end
assign en_p = en & ~en_d;
always@(posedge clk)
begin
    if(data == 12'd1023)    change <= 1'b1;
    else                    change <= 1'b0;
end
always@(posedge clk or negedge rst_n)
begin
    if(~rst_n)          data <= seed;
    else if(en_p)       data <= data - 1'b1;
    else if(data == 1)  data <= 12'd1023;
    else                data <= data;
end
endmodule
```

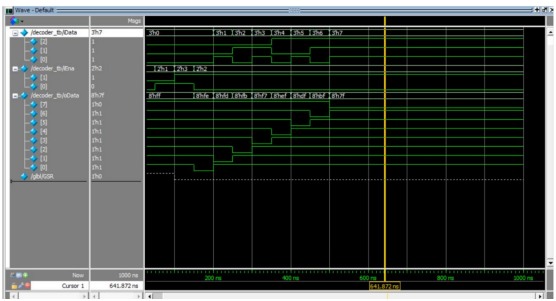
13）debounce模块：按键消抖模块。

```
module  debounce
(
    input       clk,
    input       rst,
    input       button_in,
    output reg  button_posedge,
    output reg  button_negedge,
    output reg  button_out
);
//// ---------------- internal constants --------------
parameter N = 32 ;              // debounce timer bitwidth
parameter FREQ = 50;            //model clock :Mhz
parameter MAX_TIME = 20;    //ms
localparam TIMER_MAX_VAL =   MAX_TIME * 1000 * FREQ;
////---------------- internal variables --------------
reg  [N-1 : 0]  q_reg;      // timing regs
reg  [N-1 : 0]  q_next;
reg DFF1, DFF2;             // input flip-flops
wire q_add;                 // control flags
wire q_reset;
reg button_out_d0;
//// -------------------------------------------------------------

////contenious assignment for counter control
assign q_reset = (DFF1  ^ DFF2);          // xor input flip flops to look for level chage to reset c
ounter
assign q_add = ~(q_reg == TIMER_MAX_VAL); // add to counter when q_reg msb is equal to 0

//// combo counter to manage q_next
always @ ( q_reset, q_add, q_reg)
begin
    case( {q_reset , q_add})
        2'b00 :
                q_next <= q_reg;
        2'b01 :
                q_next <= q_reg + 1;
        default :
                q_next <= { N {1'b0} };
    endcase
```

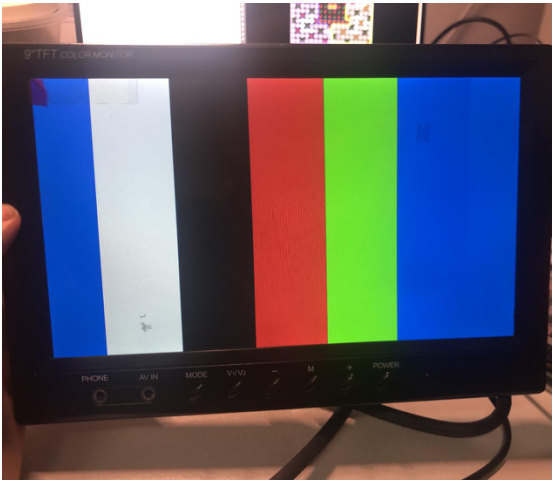
```
end

//// Flip flop inputs and q_reg update
always @ ( posedge clk or posedge rst)
begin
    if(rst == 1'b1)
    begin
        DFF1 <= 1'b0;
        DFF2 <= 1'b0;
        q_reg <= { N {1'b0} };
    end
    else
    begin
        DFF1 <= button_in;
        DFF2 <= DFF1;
        q_reg <= q_next;
    end
end

//// counter control
always @ ( posedge clk or posedge rst)
begin
    if(rst == 1'b1)
        button_out <= 1'b1;
    else if(q_reg == TIMER_MAX_VAL)
        button_out <= DFF2;
    else
        button_out <= button_out;
end

always @ ( posedge clk or posedge rst)
begin
    if(rst == 1'b1)
    begin
        button_out_d0 <= 1'b1;
        button_posedge <= 1'b0;
        button_negedge <= 1'b0;
    end
    else
    begin
        button_out_d0 <= button_out;
        button_posedge <= ~button_out_d0 & button_out;
        button_negedge <= button_out_d0 & ~button_out;
    end
end
endmodule
```

六、实验结果

1）七段数码管的测试：



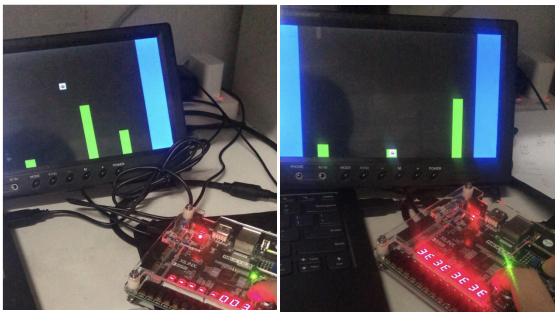
VGA 显示屏的测试：因为 VGA 信号并不容易直接从 modelsim 上得出，这里使用了 VGA 彩条实验进行显示测试。



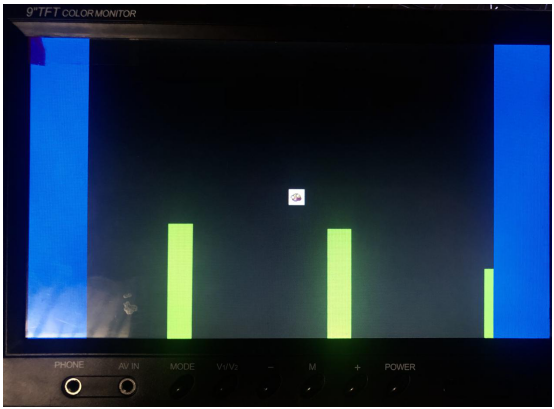
3）下板图：

左图可以观察到当前游戏分数；

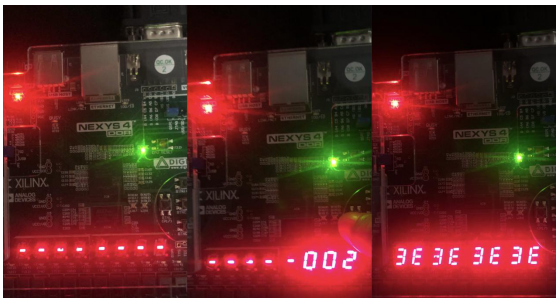
右图进入 gameover 状态，游戏界面停止，计分管显示特殊图案提示游戏结束。



高清游戏界面：



高清数码管：



六、心得体会与建议

本学期是我初次接触硬件设计的课程，通过数字逻辑教学，我对计算机功能的物理实现有了更深刻的理解。此外，学习并掌握了 Verilog HDL 这一门硬件设计语言，让我们了解到现在前沿，针对硬件设计开发的工具已经达到了比较高的、抽象的描述层次，而传统通过门级部件调用的设计方法已经被更高效率的硬件描述语言所替代了。

学习的过程中有很多不适应的地方，虽然 Verilog HDL 的语法与 C 语言相仿，但是思考的逻辑却不尽相同，比如说代码块并行这一点。还有些代码是不可综合实现的，这里也需要特别的留心。

最开始学习 Verilog 的时候，因为实践课使用的描述性语言与理论课所学内容脱节比较大，理论课所学内容更多直观对应着门级调用这种书写方式。因此学习初期完全把 Verilog 当成 C 语言来写，但是通过学习的加深，逐渐能把描述性语言和电路组成对应起来，虽说还做不到“手中是代码，心中是电路”这种层次，但是也具有了一定感觉，相信下个学期能进一步提高。

完成大作业的过程，给我了很深刻的感受，断断续续的工作进行了大半个月，实际上最初看到往届学长的工作时，都感觉与我们课上所做的那些项目脱离甚远，以至于毫无头绪。但是通过自己对 VGA 的调查学习；我先后实现了用 VGA 显示色条、用 VGA 显示动态色块、用 VGA 显示字符、用 VGA 显示 ROM 中的图像这四个小项目，工作有了一些眉目，之后也参考了很多设计的项目和书籍，花了很长时间调试项目，最后才得以完成这样一个看似简陋但是对我而言已经很不容易的项目，也真正了解到外围部件和我们板子上所用的资源并没有什么太大差异，平时所做的控制器、译码器、编码器都是数字系统不可或缺的基本部件，而了解通讯协议、管脚信息，我们可以尝试更多的拓展。

最后是一些建议，我个人感觉最终大作业的要求与平时我们所练习的项目差别还是很大，以至于很多基础薄弱的同学（比如我）完成起来相对吃力，虽然平时教学中的编码器、译码器、多路开关这些都是与理论课程紧密联系的基础模块。但对基础模块相互配合的训练显然不足，印象比较深刻的可能就只有寄存器堆实验着重训练了这一点，之后的编程教学还是希望能够在组合模块的方面增加比重。可以将每一次的课后作业设计成一个难度从浅入深的小项目，用项目包括基

础模块的应用和选择，而非仅仅根据真值表什么的做一个基础模块。这样可能趣味性会更高，也能更好的培养大家的设计能力。

之后是对于数字芯片设计领域，当前我国在国际上现状的一些体会：美国在芯片等高新技术领域已经开始了对我国的封锁，因此在高端芯片领域实现国产化是具有战略性意义的一步。

芯片分为设计与生产，当前国际芯片设计市场上，手机芯片公司主要有高通、博通，电脑芯片基本上由英特尔和 AMD 垄断，这些公司均来自美国。=而我国台湾的台积电则为芯片生产公司的领头羊，2017 年台积电包下了全世界晶圆代工业务的 56%，规模和技术均列全球第一，市值甚至超过了英特尔，成为全球第一半导体企业。

可以说无论是芯片的设计与生产，我国暂时都远远无法与美国相提并论，甚至台积电已经具备了 7nm 工艺的生产能力，而国内排名前列的中芯国际堪堪实现了 28nm 工艺，14nm 工艺还在研究当中。可见，比之台湾，我们在半导体领域都有所差距。

不过乐观的一面是，我国已经充分意识到发展数字芯片产业的重要性，快速追赶与加大投入力度的过程中，涌现了华为海思、中芯国际等后起之秀，CPU 和 SoC 领域上我们国家已经卓有进步，而而更广阔的模拟、数模混合芯片领域，我们国家的基础却几乎为零，而且差距在越来越大。因此，我想作为国内优秀高校的计算机专业学生，应当软件硬件两手抓，数字逻辑课程和后续的计算机组成原理就是为我们今后报效祖国、奉献智慧打好坚实的基础，我们更应该勤奋学习，刻苦钻研。

最后，感谢张冬冬老师与助教老师一个学期的辛勤付出与专业指教，学生浅见拙识，望您不吝赐教！