浙江大学

课程设计报告

中文题目 ： **基于数字系统的 Flappy Bird**

姓名/学号：

指导教师： 施青松 洪奇军

专业类别： 计算机科学与技术

所在学院： 计算机科学与技术学院

论文提交日期 2021 年 1 月 12 日

摘要：本次课程设计以Flappy Bird小游戏作为课题,主要完成了基于数字系统的Flappy Bird小游戏的实现，下文主要介绍了关于Flappy Bird设计的背景、功能、实现、结果验证与展望等内容，并对课题完成过程中所遇到的困难和解决方法做了分析，最后讨论课程学习的收获。

关键词：数字逻辑；FPGA；Verilog

目录

一、绪论

1.1 游戏设计背景

越南的一名独立游戏开发者开发过 flappy bird 的作品，游戏于 2013 年上线，并在 2014 年爆红。此后，开发者本人撤下游戏并于一段时间后重新上线游戏。

此后，由于开发者拒绝更新游戏以适应新的操作系统，游戏现已绝版。

如图 1.1-1，游戏的基本规则为，玩家需要通过不行按下按钮来让小鸟飞行，小鸟通过一对水管的间隙计一分。如果小鸟因为地心引力落到地上或撞到水管上，则游戏宣告失败。随着游戏的进行，小鸟的飞行速度会越来越快，使得游戏的难度越来越大。

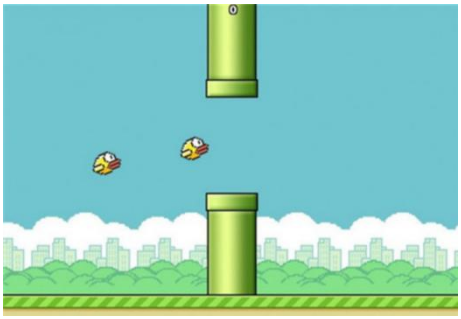

图 1.1-1 原版游戏画面

1.2 课程功能与接口

1.2.1 阵列键盘输入

如图 2-1，本次课程设计的游戏支持使用阵列键盘进行操作。利用之前实验内容中独立搭建的实验框架，最左一列第四行按键 BTN[8]控制游戏从首页进入游戏；游戏失败后，右数第二列第四行按键 BTN[10]控制从游戏结束页面进入首页，从而可以再次进入游戏，周而复始。

游戏中，左数第二列第四行按键 BTN[9]控制小鸟的飞行。按一下按键，小鸟会向上飞行一段距离。

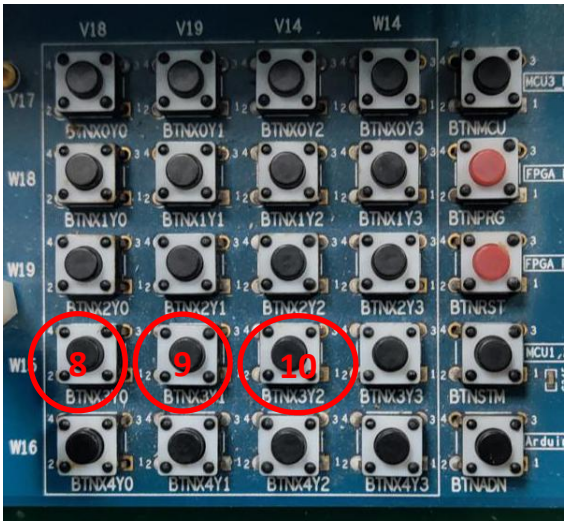

图 1.2-1 阵列键盘说明

1.2.2 开关输入

如图 2-2，本次课程设计的游戏支持使用开关。利用之前实验内容中独立搭建的实验框架，最左端 SW[15]可以控制 VGA（Video Graphics Array）是否输出图像。当 SW[7:5]=000 时，七段数码管显示当前的游戏得分；当 SW[7:5]=001 时，七段数码管显示历史最高得分。

图 1.2-2 开关说明

1.2.3 PS2 键盘输入

PS2 键盘输入功能非常简单，因为游戏的按键逻辑很简单。按要回车键来控制进入游戏和从游戏结束页面进入首页，以及空格键来控制小鸟的飞翔。

开发之初，我学习了书后的 PS2 键盘介绍，参考了一些网络上的资料以及一些课件。实际开发中，我将 PS2 键盘输入与阵列键盘输入做了兼容，二者都可以用来操控游戏。

1.3 设计难点

综合来看，本设计的难点主要有以下几点：

1．VGA 的显示：要让 VGA 显示相应的图片，并且保证没有错位。

2．按钮的功能：要让按钮能正确地发挥作用。

3．位置的控制：要确保能正确操控小鸟的位置，水管的位置要随机生成，同时要确保存在通过的可能。

4．成败的判定：要能正确判定得分与游戏失败。

二、Flappy Bird 设计原理

2.1 理论基础

2.1.1 VGA 显示

VGA（Video Graphics Array）作为一种标准的显示接口得到了广泛的应用，其信号类型为模拟类型，显示卡端的接口为 15 针母插座。VGA 在任何时刻都必须工作在某一显示模式之下，其显示模式分为字符显示模式和图形显示模式，在应用中，讨论的都是图形显示模式。

VGA 是一种 CRT（Cathode ray tube）显示器，采用光栅扫描显示——其显示原理为：电子束从屏幕左上角开始向右扫，当到达屏幕的右边缘时，电子束关闭（水平消隐），并快速返回屏幕左边缘（水平回扫），然后在下一条扫描线上开始新的一次水平扫描。一旦所有的水平扫描均告完成，电子束在屏幕的右下角结束并关闭（垂直消隐），然后迅速返回到屏幕的左上角（垂直回扫），开始下一次光栅扫描。即，阴极射线管发出电子束，电子束从左至右形成光栅，又从上向下在屏幕上形成均匀的光栅移动，间接控制电子束强度呈现显示。VGA 一次处理一个像素点，通过光栅移动的形式遍历整个屏幕的像素点；每个像素点的显示通过电子束强度来控制。

一般所使用的 VGA 显示器均为标准五输入类型，包括：列同步信号 HS、行同步信号 VS、红基色信号 R、绿基色信号 G 与蓝基色信号 B。其中 HS 和 VS 与显示频率有关，R、G 和 B 与像素点的颜色有关。

如图 2.1-1 可见，无论是行时序还是列时序都存在着图像显示区（Active video time）和图像消隐区，图像消隐区又分为消隐前肩、同步脉冲区和消隐后肩。图像消隐区的存在是由于电子束存在惯性，电子束到达屏幕边缘时会过冲到屏幕外面，产生边缘过冲。边缘过冲的区域是不显示图像的。

需要注意的是，每帧图像的显示时间间隔是固定的，不能大于 20ms，即 50 帧以上的图像刷新率，否则图像显示就会闪烁。

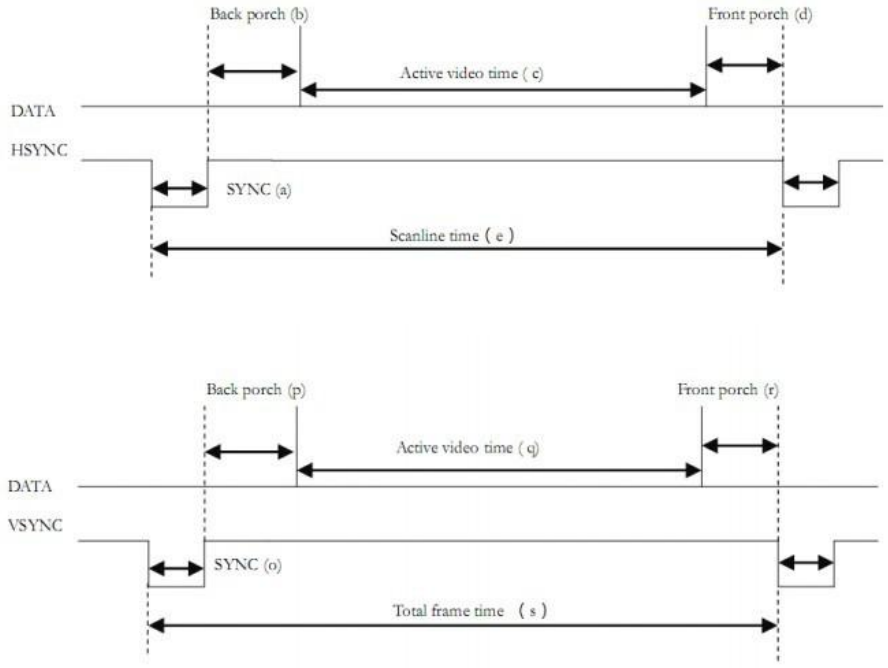

图 2.1-1 VGA 的时序

2.1.2 PS2 键盘

PS2 通信协议是一种双向同步串行通讯协议，这种通讯方式与 P2S 同步串行传输类同。PS2 通讯是标准协议，通讯双方通过 CLK 时钟信号同步，在 DATA 信号线上交换数据。PS2 协议约定通讯双方任何一方如果想抑制另外一方通讯，只需要把 CLK 信号拉到低电平。如果是计算机和 PS2 键盘之间通讯，则计算机必须做主控机。

一般两台设备间传输数据的最大时钟频率是 33kHz，大多数 PS2 设备工作在 10-20kHz。推荐值在 15kHz 左右，也就是说，CLOCK 高、低电平的持续时间都为 40us。每一数据帧包含 11—12 位，其含义见表 2.1-1。

表 2.1-1 PS2 通信数据格式

数据	意义
1 个起始位	总是逻辑 0
8 个数据位	（LSB）低位在前
1 个奇偶校验位	奇校验
1 个停止位	总是逻辑 0
1 个应答位	仅用在主控机对设备的通讯中

如图 2.1-2，数据在 PS2 时钟的下降沿读取，PS2 的时钟频率为 10-16.7kHz。对于 PS2 设备，一般来说从时钟脉冲的上升沿到一个数据转变的时间至少要有 5us；数据变化到下降沿的时间至少要有 5us，并且不大于 25us，这个时序非常重要应该严格遵循。在停止位发送后设备在发送下个包前应该至少等待 50us，给主机时间做相应的处理。不过主机处理接收到的字节时一般会抑制发送（主机在收到每个包时通常自动做这个）。在主机释放抑制后，设备至少应该在发送任何数据前等 50us。图中可见，读取数据时时钟的下降沿出现在 DATA 段的中间，这是为了确保 PC 机能正确读取到 PS2 设备的数据。因为实际情况下，会存在各种因素导致不同程度的延迟，如果时钟的下降沿与数据段的下降沿同时出现，会出现错误。

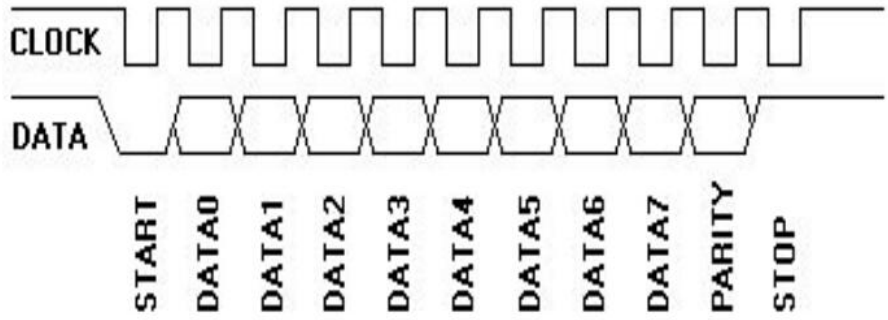

图 2.1-2 PS2 键盘时序图

2.1.3 状态机的设计

有限状态机（Finite State Machine, FSM）是时序电路设计中经常采用的一种方式，尤其适合设计数字系统的控制模块，在一些需要控制高速器件的场合，用状态机进行设计是一种很好的解决问题的方案，具有速度快、结构简单、可靠性高等优点。有限状态机非常适合用 FPGA 器件实现，用 Verilog HDL 的 case 语句能很好地描述基于状态机的设计，再通过 EDA 工具软件的综合，一般可以生成性能极优的状态机电路，从而使其在执行时间、运行速度和占用资源等方面优于用 CPU 实现的方案。

有限状态机一般包括组合逻辑和寄存器逻辑两部分，寄存器逻辑用于存储状态，组合逻辑用于状态译码和产生输出信号。根据输出信号产生方法的不同，状态机可分为两类： 米

里型（Mealy）和摩尔型（Moore）。摩尔型状态机的输出只是当前状态的函数。米里型状态机的输出是在输入变化后立即变化的，不依赖时钟信号的同步，摩尔型状态机的输入发生变化时还需要等待时钟的到来，必须在状态发生变化时才会导致输出的变化，因此比米里型状态机要多等待一个时钟周期。

2.1.4 硬件描述语言

Verilog HDL 是一种硬件描述语言（HDL: Hardware Description Language），以文本形式来描述数字系统硬件的结构和行为的语言，用它可以表示逻辑电路图、逻辑表达式，还可以表示数字逻辑系统所完成的逻辑功能。

使用 Verilog 描述硬件的基本设计单元是模块（module）。构建复杂的电子电路，主要是通过模块的相互连接调用来实现的。模块被包含在关键字 module、endmodule 之内。实际的电路元件。Verilog 中的模块类似 C 语言中的函数，它能够提供输入、输出端口，可以实例调用其他模块，也可以被其他模块实例调用。

2.1.5 可编程阵列逻辑

PAL（Programmable Array Logic）器件由可编程的与阵列、固定的或阵列和输出反馈单元组成。不同型号 PAL 器件有不同的可编程阵列逻辑输出和反馈结构，适用于各种组合逻辑电路和时序逻辑电路的设计，是一种可程式化的装置。PLA 具有一组可程式化的 AND 阶，AND 阶之后连接一组可程式化的 OR 阶，如此可以达到：只在合乎设定条件时才允许产生逻辑讯号输出。

PLA 如此的逻辑闸布局能用来规划大量的逻辑函式，这些逻辑函式必须先以积项（有时是多个积项）的原始形式进行齐一化。在 PLA 的应用中，有一种是用来控制资料路径，在指令集内事先定义好逻辑状态，并用此来产生下一个逻辑状态（透过条件分支）。举例来说，如果目前机器（指整个逻辑系统）处于二号状态，如果接下来的执行指令中含有一个立即值（侦测到立即值的栏位）时，机器就从第二状态转成四号状态，并且也可以进一步定义进入第四状态后的接续动作。

2.2 实验器材

2.2.1 ISE 软件

ISE 是使用 XILINX 的 FPGA 的必备的设计工具。它可以完成 FPGA 开发的全部流程，包括设计输入、仿真、综合、布局布线、生成 BIT 文件、配置以及在线调试等，功能非常强大。ISE 除了功能完整，使用方便外，它的设计性能也非常好，以集成的时序收敛流程整合了增强性物理综合优化，提供最佳的时钟布局、更好的封装和时序收敛映射，从而获得更高的设计性能。

2.2.2 SWORD 板

采用开放式体系构架和 32 位存储层次结构，通用性强，适用性灵活。实验方法采用基于 FPGA 实体的虚拟实验箱和 SOC 集成技术，支持个性化开发、课程设计和创新实践。系统资源可很好地支持数字电路、计算机组成、计算机体系结构、接口通讯、编译技术、操作系统、计算机网络、基于 IP 核的嵌入式系统和多媒体等课程的教学实践。

2.2.3 Photoshop 软件

进行图片编辑工作，包括修改图片大小以及查看图片像素。方便对图片作出一定修改以

适应 FPGA 开发的需要。

2.2.4 Matlab 软件

用以将图片转化为 coe 文件，便于 IP 核调用传输图像。

2.2.5 计算机

2.3 设计方案

2.3.1 整体设计

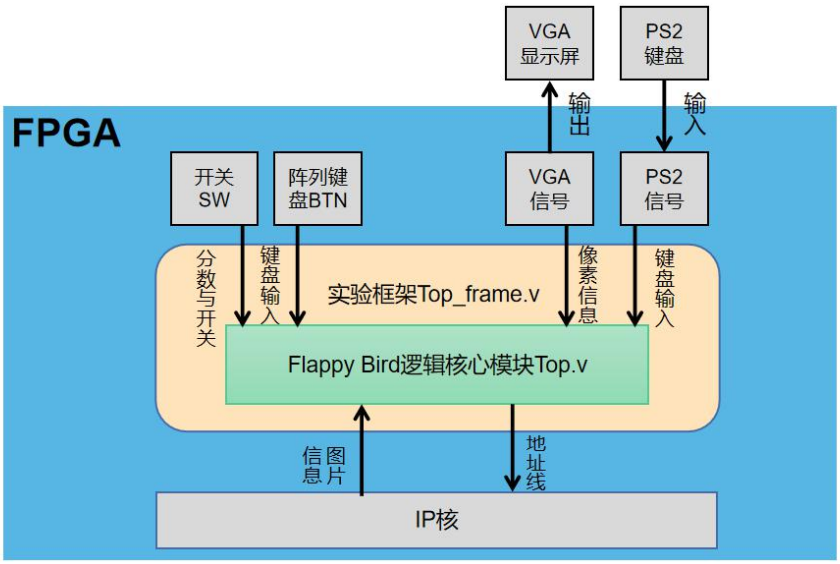

图 2.3-1 整体硬件设计框图

如图 2.3-1 所示，整个课程设计可基本如图划分。阵列键盘输入与 PS2 键盘输入都用以操控游戏的进程；开关用来控制 VGA 的显示以及七段数码管显示的分数；VGA 信号用来控制需要 VGA 显示的像素信息；IP 核为 RAM，用来保存游戏的图片素材；Flappy Bird 逻辑核心模块有处理游戏的状态、控制各元素的位置等功能。

2.3.2 Flappy Bird 逻辑核心模块设计

如图 2.3-2，Flappy Bird 逻辑核心模块设计流程基本可按照如图划分。需要说明的是，由于设计时未能充分借鉴编程通过子函数来简化主函数的思想，该逻辑核心模块 Top.v 较为臃肿。其中的键盘处理部分、欢迎页面等部分，都可以成为单独的模块从而使得逻辑核心模块更为简洁。

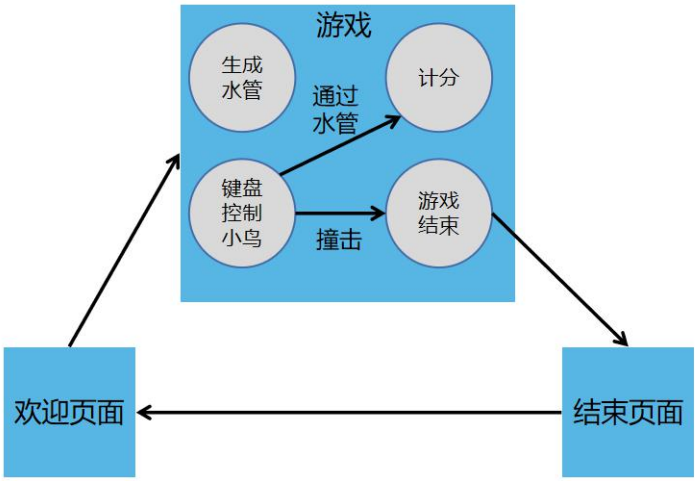

图 2.3-2 Flappy Bird 逻辑核心模块设计框图

2.4 硬件设计

2.4.1 VGA 显示

该部分首先需要的是完成 VGA 驱动，即首先要能启动 VGA。根据书后的介绍以及相关课件，不难写出如下驱动代码：

```
`timescale 1ns / 1ps

module vgac (input [11:0] d_in,        // bbbb_gggg_rrrr, pixel
               input vga_clk,           // 25MHz
               input clrn,
               output reg [8:0] row_addr, // pixel ram row address, 480
(512) lines
               output reg [9:0] col_addr, // pixel ram col address, 640
(1024) pixels
               output reg [3:0] r,g,b,      // red, green, blue colors
               output reg rdn,          // read pixel RAM (active_low)
               output reg hs,vs          //  horizontal  and  vertical
synchronization
    );

  reg [9:0] h_count; // VGA horizontal counter (0-799): pixels
  always @ (posedge vga_clk) begin
     if (!clrn) begin
        h_count <= 10'h0;
     end else if (h_count == 10'd799) begin
        h_count <= 10'h0;
     end else begin
        h_count <= h_count + 10'h1;
     end
  end

  reg [9:0] v_count; // VGA vertical counter (0-524): lines
  always @ (posedge vga_clk or negedge clrn) begin
     if (!clrn) begin
        v_count <= 10'h0;
     end else if (h_count == 10'd799) begin
        if (v_count == 10'd524) begin
           v_count <= 10'h0;
        end else begin
           v_count <= v_count + 10'h1;
        end
```

```
      end
   end
    // signals, will be latched for outputs
    wire  [9:0] row    =  v_count - 10'd35;     // pixel ram row addr
    wire  [9:0] col    =  h_count - 10'd143;    // pixel ram col addr
    wire  h_sync      = (h_count > 10'd95);    // 96 -> 799
    wire  v_sync        = (v_count > 10'd1);     // 2 -> 524
    wire  read          = (h_count > 10'd142) && // 143 -> 782
                      (h_count < 10'd783) && // 640 pixels
                      (v_count > 10'd34)  && // 35 -> 514
                      (v_count < 10'd515);   // 480 lines
    // vga signals
    always @ (posedge vga_clk) begin
       row_addr <=  row[8:0]; // pixel ram row address
       col_addr <=  col;      // pixel ram col address
       rdn      <= ~read;     // read pixel (active low)
       hs       <=  h_sync;   // horizontal synchronization
       vs       <=  v_sync;   // vertical   synchronization
       r        <=  rdn ? 4'h0 : d_in[3:0]; // 3-bit red
       g        <=  rdn ? 4'h0 : d_in[7:4]; // 3-bit green
       b        <=  rdn ? 4'h0 : d_in[11:8]; // 3-bit blue
    end
endmodule
```

此后，需要解决的是让 VGA 显示什么的问题。利用 Matlab，可以将 png 图片转化为 coe 文件。得到 coe 文件后，在 Source 窗口单击右键，点击 New Source 选择新建 IP(CORE Generator & Architecture Wizard)，此后在如图 2.4-1 所示的目录下选择生成 RAM 或 ROM，导入 coe 文件。

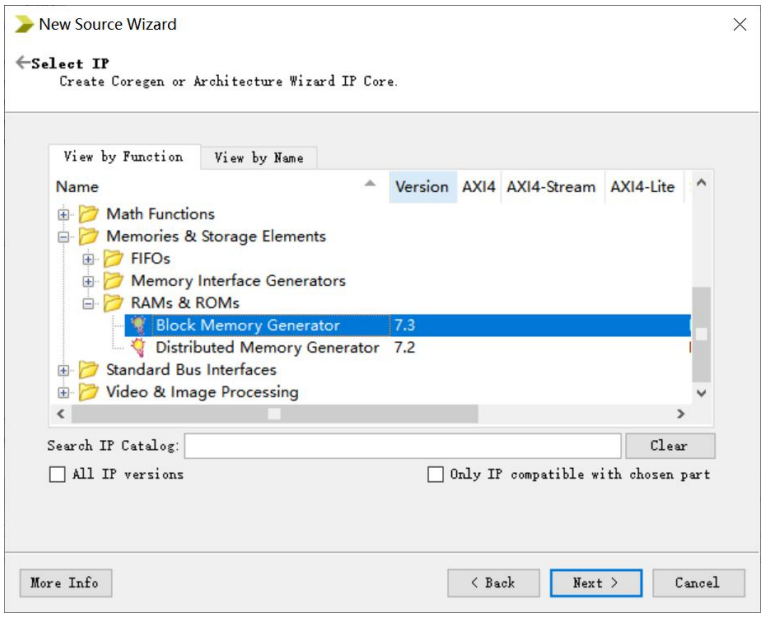

图 2.4-1 生成 IP 核

为了方便测试 VGA 驱动是否正常，我先将完全静态的欢迎页面做好。调用 IP 核，只需要将需要显示的像素点的位置通过 addra 传入，就可以通过 douta 获得该像素点的 RGB 信息。代码如下：

```
    localparam     TITLE_X=231,     TITLE_Y=182,     BUTTON_PLAY_X=262,
BUTTON_PLAY_Y=260;
    localparam NAME_X=223, NAME_Y=440;
```

```
    wire [11:0]bg_data;
    wire [11:0]button_play_data;
    wire [11:0]title_data;
    wire [18:0]bg_addr;
    wire [12:0]button_play_addr;
    wire [13:0]title_addr;
    wire [11:0]name_data;
    wire [12:0]name_addr;
    assign bg_addr=col+row*640;
    assign
button_play_addr=(col-BUTTON_PLAY_X)+(row-BUTTON_PLAY_Y)*116;
    assign title_addr=(col-TITLE_X)+(row-TITLE_Y)*178;
    assign name_addr=(col-NAME_X)+(row-NAME_Y)*194;

    background BG(.addra(bg_addr), .douta(bg_data), .clka(DIV[1]));
    button_play
BUTTON(.addra(button_play_addr), .douta(button_play_data), .clka(DIV[1
]));
    title TITLE(.addra(title_addr), .douta(title_data), .clka(DIV[1]));
    name  NAME(.addra(name_addr), .douta(name_data), .clka(DIV[1]));
```

此后，通过 case 语句来让判断在何处显示哪张图片。其中，在显示背景时，我给背景图片的 RGB 做了除以 4 的处理，这使得背景色更接近黑色，从而突出了主体的游戏名、开始游戏按钮已经作者署名。代码如下：

```
    wire BUTTON_DIS, TITLE_DIS, NAME_DIS;

    assign
BUTTON_DIS=(col>=BUTTON_PLAY_X)&&(col<=BUTTON_PLAY_X+115)&&(row>=BUTT
ON_PLAY_Y)&&(row<=BUTTON_PLAY_Y+69)&&(button_play_data!=12'h000);
    assign
TITLE_DIS=(col>=TITLE_X)&&(col<=TITLE_X+177)&&(row>=TITLE_Y)&&(row<=T
ITLE_Y+47)&&(title_data!=12'h000);
    assign
NAME_DIS=(col>=NAME_X)&&(col<=NAME_X+193)&&(row>=NAME_Y)&&(row<=NAME_
Y+29)&&(name_data!=12'h000);

     always@(posedge DIV[1])begin
       if(welcome)begin
           case(1'b1)
               TITLE_DIS: vgadata<=title_data;
               BUTTON_DIS: vgadata<=button_play_data;
               NAME_DIS:   vgadata<=name_data;
               default:   begin
                              vgadata[11:8]<=bg_data[11:8]/4;
                              vgadata[7:4]<=bg_data[7:4]/4;
                              vgadata[3:0]<=bg_data[3:0]/4;
                              end
           endcase
       end
     end
```

经测试，VGA 可以正常显示欢迎页面。结束页面测试同理。

2.4.2 PS2 键盘

在理解原理的基础上，该部分代码大同小异，并且书后给出了样例。PS2 键盘驱动代码如下：

```
`timescale 1ns / 1ps
module
ps2_keyboard_driver(clk,clk_delay,rst_n,ps2k_clk,ps2k_data,data_out);
```

```
input clk;    //50M时钟信号
input clk_delay; //防抖动延迟时钟
input rst_n;  //复位信号
input ps2k_clk;   //PS2接口时钟信号
input ps2k_data;     //PS2接口数据信号
output [15:0] data_out;
//-----------------------------------------
reg ps2k_clk_r0,ps2k_clk_r1,ps2k_clk_r2;  //ps2k_clk状态寄存器
//wire pos_ps2k_clk; // ps2k_clk上升沿标志位
wire neg_ps2k_clk;   // ps2k_clk下降沿标志位
//设备发送向主机的数据在下降沿有效，首先检测PS2k_clk的下降沿
//利用上面逻辑赋值语句可以提取得下降沿，neg_ps2k_clk为高电平时表示数据可以被采集

always @ (posedge clk or negedge rst_n) begin //判断连续低电平
    if(!rst_n) begin
           ps2k_clk_r0 <= 1'b0;
           ps2k_clk_r1 <= 1'b0;
           ps2k_clk_r2 <= 1'b0;
       end
    else begin                              //锁存状态，进行滤波
           ps2k_clk_r0 <= ps2k_clk;
           ps2k_clk_r1 <= ps2k_clk_r0;
           ps2k_clk_r2 <= ps2k_clk_r1;
       end
end

assign neg_ps2k_clk = ~ps2k_clk_r1 & ps2k_clk_r2;    //下降沿

//-----------------数据采集-------------------------
reg[7:0] ps2_byte_r=0;       //PC接收来自PS2的一个字节数据存储器
reg[7:0] temp_data=0;       //当前接收数据寄存器
reg[3:0] num=0;              //计数寄存器

always @ (posedge clk or negedge rst_n) begin
    if(!rst_n) begin
           num <= 4'd0;
           temp_data <= 8'd0;
       end
    else if(neg_ps2k_clk) begin //检测到ps2k_clk的下降沿
           case (num)
               4'b0000:   num <= 4'b0001;
               4'b0001:   begin
                          num <= 4'b0011;
                          temp_data[0] <= ps2k_data;  //bit0
                      end
               4'b0011:   begin
                          num <= 4'b0010;
                          temp_data[1] <= ps2k_data;  //bit1
                      end
               4'b0010:   begin
                          num <= 4'b0110;
                          temp_data[2] <= ps2k_data;  //bit2
                      end
               4'b0110:   begin
                          num <= 4'b0111;
                          temp_data[3] <= ps2k_data;  //bit3
                      end
               4'b0111:   begin
```

```
                        num <= 4'b1111;
                        temp_data[4] <= ps2k_data;  //bit4
                    end
                4'b1111:  begin
                        num <= 4'b1110;
                        temp_data[5] <= ps2k_data;  //bit5
                    end
                4'b1110:  begin
                        num <= 4'b1100;
                        temp_data[6] <= ps2k_data;  //bit6
                    end
                4'b1100:  begin
                        num <= 4'b1000;
                        temp_data[7] <= ps2k_data;  //bit7
                    end
                4'b1000:  begin
                        num <= 4'b1010;   //奇偶校验位，不做处理
                    end
                4'b1010: begin
                        num <= 4'b0000;   // num 清零
                    end
                default: ;
                endcase
        end
end

reg key_f0=1'b0;        //松键标志位，置 1 表示接收到数据 8'hf0，再接收到下一个
数据后清零
reg judge=1'b0;   //判断大写锁定键位是否按下
reg record=1'b0, done=1'b0;
//+++++++++++++++数据处理开始+++++++++++++++++=============
always @ (posedge clk or negedge rst_n) begin //接收数据的相应处理，这里只
对 1byte 的键值进行处理
    if(!rst_n) begin
           key_f0 <= 1'b0;
       end
    else  if(record & !done) begin  record<=1'b0; done<=1'b1; end
    else  if(num==4'd10) ///一帧数据是否采集完。
           begin  //刚传送完一个字节数据
               if(!done)
                   if(temp_data == 8'hf0) begin
                       key_f0 <= 1'b1;//判断该接收数据是否为断码
                   end
                   else begin
                       if(!key_f0)            //说明有键按下
                       begin
                            if(temp_data==8'h58) begin
                                 judge <= !judge;//大写锁定值改变
                                  done<=1'b1;
                             end
                             else begin
                                  ps2_byte_r <= temp_data;//锁存当前键值
                                  record<=1'b1;
                             end
                       end
                       else begin
                           if(neg_ps2k_clk) key_f0 <= 1'b0;
                       end
                   end
```

```
            end
    else done<=0;
end

wire ret;
pbdebounce_Once Once(
                  .clk(clk),
                  .clk_delay(clk_delay),
                  .in(record),
                  .pbout(ret)
                  );

reg[15:0] ps2_asci=16'h0;   //接收数据的相应 ASCII 码
always @ (clk) begin
if(judge) begin        //如果大写锁定开启，转换为大写字母
    case (ps2_byte_r)    //键值通码转换为 ASCII 码
       8'h00: ps2_asci <= 16'h0; //清零位
        8'h15: ps2_asci <= 16'h8051; //Q
        8'h1d: ps2_asci <= 16'h8057;    //W
        8'h24: ps2_asci <= 16'h8045;    //E
        8'h2d: ps2_asci <= 16'h8052;    //R
        8'h2c: ps2_asci <= 16'h8054;    //T
        8'h35: ps2_asci <= 16'h8059;    //Y
        8'h3c: ps2_asci <= 16'h8055;    //U
        8'h43: ps2_asci <= 16'h8049;    //I
        8'h44: ps2_asci <= 16'h804f;    //O
        8'h4d: ps2_asci <= 16'h8050;    //P
        8'h1c: ps2_asci <= 16'h8041;    //A
        8'h1b: ps2_asci <= 16'h8053;    //S
        8'h23: ps2_asci <= 16'h8044;    //D
        8'h2b: ps2_asci <= 16'h8046;    //F
        8'h34: ps2_asci <= 16'h8047;    //G
        8'h33: ps2_asci <= 16'h8048;    //H
        8'h3b: ps2_asci <= 16'h804a;    //J
        8'h42: ps2_asci <= 16'h804b;    //K
        8'h4b: ps2_asci <= 16'h804c;    //L
        8'h1a: ps2_asci <= 16'h805a;    //Z
        8'h22: ps2_asci <= 16'h8058;    //X
        8'h21: ps2_asci <= 16'h8043;    //C
        8'h2a: ps2_asci <= 16'h8056;    //V
        8'h32: ps2_asci <= 16'h8042;    //B
        8'h31: ps2_asci <= 16'h804e;    //N
        8'h3a: ps2_asci <= 16'h804d;    //M
        8'h16: ps2_asci <= 16'h8021;    //!
        8'h1e: ps2_asci <= 16'h8040;    //@
        8'h26: ps2_asci <= 16'h8023;    //#
        8'h25: ps2_asci <= 16'h8024;    //$
        8'h2e: ps2_asci <= 16'h8025;    //%
        8'h36: ps2_asci <= 16'h805e;    //^
        8'h3d: ps2_asci <= 16'h8026;    //&
        8'h3e: ps2_asci <= 16'h802a;    //*
        8'h46: ps2_asci <= 16'h8028;    //(
        8'h45: ps2_asci <= 16'h8029;    //)
        8'h4c: ps2_asci <= 16'h803a;    //:
        8'h52: ps2_asci <= 16'h8022;    //"
        8'h41: ps2_asci <= 16'h803c;    //<
        8'h49: ps2_asci <= 16'h803e;    //>
        8'h4a: ps2_asci <= 16'h803f;    //?
        8'h5d: ps2_asci <= 16'h807c;    //|
        8'h54: ps2_asci <= 16'h807b;    //{
```

```
        8'h5b: ps2_asci <= 16'h807d;    //}
        8'h4e: ps2_asci <= 16'h805f;    //_
        8'h55: ps2_asci <= 16'h802b;    //+
        8'h29: ps2_asci <= 16'h8020;    //空格
        8'h5A: ps2_asci <= 16'h800d;  //回车
        default: ps2_asci <=16'h0;
        endcase
        //judge<=1'b0;
    end
    else begin          //使用小写字母
    case (ps2_byte_r)    //键值通码转换为ASCII码
       8'h0:  ps2_asci <= 16'h0; //清零位
        8'h15: ps2_asci <= 16'h8071; //q
        8'h1d: ps2_asci <= 16'h8077;    //w
        8'h24: ps2_asci <= 16'h8065;    //e
        8'h2d: ps2_asci <= 16'h8072;    //r
        8'h2c: ps2_asci <= 16'h8074;    //t
        8'h35: ps2_asci <= 16'h8079;    //y
        8'h3c: ps2_asci <= 16'h8075;    //u
        8'h43: ps2_asci <= 16'h8069;    //i
        8'h44: ps2_asci <= 16'h806f;    //o
        8'h4d: ps2_asci <= 16'h8070;    //p
        8'h1c: ps2_asci <= 16'h8061;    //a
        8'h1b: ps2_asci <= 16'h8073;    //s
        8'h23: ps2_asci <= 16'h8064;    //d
        8'h2b: ps2_asci <= 16'h8066;    //f
        8'h34: ps2_asci <= 16'h8067;    //g
        8'h33: ps2_asci <= 16'h8068;    //h
        8'h3b: ps2_asci <= 16'h806a;    //j
        8'h42: ps2_asci <= 16'h806b;    //k
        8'h4b: ps2_asci <= 16'h806c;    //l
        8'h1a: ps2_asci <= 16'h807a;    //z
        8'h22: ps2_asci <= 16'h8078;    //x
        8'h21: ps2_asci <= 16'h8063;    //c
        8'h2a: ps2_asci <= 16'h8076;    //v
        8'h32: ps2_asci <= 16'h8062;    //b
        8'h31: ps2_asci <= 16'h806e;    //n
        8'h3a: ps2_asci <= 16'h806d;    //m
        8'h16: ps2_asci <= 16'h8031;    //1
        8'h1e: ps2_asci <= 16'h8032;    //2
        8'h26: ps2_asci <= 16'h8033;    //3
        8'h25: ps2_asci <= 16'h8034;    //4
        8'h2e: ps2_asci <= 16'h8035;    //5
        8'h36: ps2_asci <= 16'h8036;    //6
        8'h3d: ps2_asci <= 16'h8037;    //7
        8'h3e: ps2_asci <= 16'h8038;    //8
        8'h46: ps2_asci <= 16'h8039;    //9
        8'h45: ps2_asci <= 16'h8030;    //0
        8'h4c: ps2_asci <= 16'h803b;    //;
        8'h52: ps2_asci <= 16'h8027;    //'
        8'h41: ps2_asci <= 16'h802c;    //,
        8'h49: ps2_asci <= 16'h802e;    //.
        8'h4a: ps2_asci <= 16'h802f;    // /
        8'h5d: ps2_asci <= 16'h805c;    //\
        8'h54: ps2_asci <= 16'h805b;    //[
        8'h5b: ps2_asci <= 16'h805d;    //]
        8'h4e: ps2_asci <= 16'h802d;    //-
        8'h55: ps2_asci <= 16'h803d;    //=
        8'h29: ps2_asci <= 16'h8020;    //空格
        8'h5A: ps2_asci <= 16'h800d;  //回车
```

```
        default: ps2_asci <=16'h0;
        endcase
     end
end
assign data_out = ret?ps2_asci:16'h0;
endmodule
```

其中的 pbdebounce_Once 模块用以在一定的延迟后产生一个短脉冲，代码如下：

```
`timescale 1ns / 1ps
module pbdebounce_Once(
    input wire clk,
    input wire clk_delay,
    input wire in,
    output reg pbout
    );

    reg start=0, done=0;
    reg [15:0] pbshift=16'b0;

    always@(posedge clk) begin
        if(in & !start)begin
            start<=1'b1;
        end
        if(done & start)begin
            start<=1'b0;
            pbout<=1'b1;
        end else
            pbout<=1'b0;
    end

    always@(posedge clk_delay or negedge start)begin
        if(!start)
        begin pbshift<=16'h0; done<=0; end
        else   begin
            if(pbshift==16'hFFFF)begin
                pbshift<=16'h0;
                done<=1'b1;
            end
            else begin
                pbshift<={pbshift[14:0],1'b1};
                done<=1'b0;
            end
        end
    end
endmodule
```

2.4.3 水管显示模块

水管显示模块有两个，分别控制在上方的水管以及在下方的水管。二者原理相似，故只需要讨论其中一个即可。以上方的水管为例，由于所有水管的位置都由 Top.v 模块操控，因此该模块只需要处理显示的信息。模块根据当前 VGA 处理像素点的位置以及当前水管的位置，计算出要显示像素点的地址，获取其 RGB 信息并传回。代码如下：

```
`timescale 1ns / 1ps
module pillar_down(input wire [31:0]div,
                       input wire [9:0]pillar_down_x,        //
coordinates of bottom-left corner
                       input wire [8:0]pillar_y,
                       input wire [9:0]col,                  //
coordinates of current pixel
                       input wire [8:0]row,
```

```
                        output wire [11:0]pillardown_data // color data
    );

    wire [9:0]x;
    wire [8:0]y;
    wire [14:0]address;

    assign x=51+col-pillar_down_x;
    assign y=320-(pillar_y-row);
    assign address=x+y*52;                 // calculate the address

    pipe_down
PILLARDOWN(.addra(address), .douta(pillardown_data), .clka(div[1]));

endmodule
```

2.4.4 小鸟控制模块

该模块负责处理小鸟的信息。其考虑了重力的设定：在没有按键的情况下，小鸟的高度会随时间下降；按下按键时，小鸟会向上飞行一段距离；游戏不在进行时，小鸟的位置要复位。该模块完成了按键的判定以及调用相关的 IP 核。在该模块中，我设置了每隔一段时间即切换输出的图片，充分利用三张小鸟挥动翅膀的素材，使得其在运动过程中看起来更像是在飞翔。

代码如下：

```
module bird_con(input wire [31:0]div,
                    input wire playing,
                    input wire fly,
                    input wire [9:0]col,              // coordinates of
current pixel
                    input wire [8:0]row,
                    output reg [9:0]bird_x,           // coordinates of
top-left corner
                    output reg [8:0]bird_y,
                    output reg [11:0]bird_data
    );

    localparam gravity=3;     // the speed to fall

    localparam BIRD_INI_X=35; // the top-left corner
    localparam BIRD_INI_Y=192;

    localparam BIRD_INI=1'b0; // 2 states -> bird_state
    localparam BIRD_PLAY=1'b1;

    wire [9:0]x;
    wire [8:0]y;
    wire [11:0]address;
    wire [11:0]bird_data_up;
    wire [11:0]bird_data_hor;
    wire [11:0]bird_data_down;

    assign x=col-bird_x;
    assign y=row-bird_y;
    assign address=x+y*48;    // calculate the address

    bird_up
ANIUP(.addra(address), .douta(bird_data_up), .clka(div[1]));
    bird_hor
ANIHOR(.addra(address), .douta(bird_data_hor), .clka(div[1]));
```

```
    bird_down
ANIDOWN(.addra(address), .douta(bird_data_down), .clka(div[1]));

    reg [3:0]flyct=0;
    always@(posedge div[20])begin          // control the position of the
bird
        if(playing==1'b0)begin
            bird_x<=BIRD_INI_X;
            bird_y<=BIRD_INI_Y;
            flyct<=0;
        end
        else if((fly && flyct!=7) || (flyct!=0 && flyct!=7))begin
            bird_y<=bird_y-7;
            flyct<=flyct+1;
        end
        else begin
            bird_y<=bird_y+gravity;
            flyct<=0;
        end
    end

    always@(posedge div[1])begin           // change the animation and so
make the bird flying
        if(div[25:24]==2'b11)begin
            bird_data<=bird_data_up;
        end
        else if(div[25:24]==2'b01)begin
            bird_data<=bird_data_down;
        end
        else if(div[24]==1'b0)begin
            bird_data<=bird_data_hor;
        end
    end

endmodule
```

2.4.5 随机数模块

该模块负责生成随机数，系我在参考了一些网上的资料后写出。该模块可用于随机生成水管的位置，增加游戏的可玩性。但为了方便验证以及演示，我在提交的版本中利用时钟分频信号来生成水管的位置，暂且降低了其难度。经仿真验证与实际物理验证，随机数模块完全可用，只是由其生成的游戏难度会更大一点。代码如下：

```
`timescale 1ns / 1ps
module random(input clk,
                input [7:0]seed,
                input load,
                output reg [7:0]random
    );

    always@(posedge clk)begin
        if(load)begin
            random<=seed;
        end
        else begin
            random[0]<=random[7];
            random[1]<=random[0];
            random[2]<=random[1];
            random[3]<=random[2];
            random[4]<=random[3]^random[7];
            random[5]<=random[4]^random[7];
            random[6]<=random[5]^random[7];
            random[7]<=random[6];
        end
    end

endmodule
```

2.4.6 逻辑核心模块 Top.v

该模块内容较多，因此分块叙述。一些简单的定义略去不谈，因为源文件中都有对变量意义的注释。

首先是水管移动速度的的变化，当游戏不在进行时，将移动速度设置为初始值。此后，利用时钟分频信号，移动速度随着时间推移逐渐变大，游戏难度逐渐升高。

```
    always@(posedge DIV[27] or negedge playing)begin
        if(playing!=1)begin
            speed<=4;
        end
        else if(speed!=8)begin
            speed<=speed+1;
        end
    end
```

调用 VGA 驱动，当开关 SW[15]为 0，VGA 被关闭，不显示任何信号。

```
  wire cas;
   vgac VGA_DRI(.d_in(vgadata), .vga_clk(DIV[1]), .clrn(SW[15]),
        .row_addr(row), .col_addr(col), .rdn(cas),
        .hs(hs), .vs(vs), .r(R), .g(G), .b(B));
```

欢迎页面的绘制，需要用到各图片的坐标、地址计算以及取出的像素信息。在理解 RAM 的输入输出后，不难直接写出。此后的游戏结束页面大同小异，因此不再特地写出。带 TITLE/title 的变量标记游戏标题图片的相关信息，带 BUTTON/button 的变量标记开始游戏按钮图片的相关信息，带 NAME/name 的变量标记署名图片的相关信息；而背景图片是铺满整个屏幕的，因此不需要特地标注其位置。

```
localparam      TITLE_X=231,      TITLE_Y=182,      BUTTON_PLAY_X=262,
BUTTON_PLAY_Y=260;
    localparam NAME_X=223, NAME_Y=440;

    wire [11:0]bg_data;
    wire [11:0]button_play_data;
```

```
    wire [11:0]title_data;
    wire [18:0]bg_addr;
    wire [12:0]button_play_addr;
    wire [13:0]title_addr;
    wire [11:0]name_data;
    wire [12:0]name_addr;
    assign bg_addr=col+row*640;
    assign
button_play_addr=(col-BUTTON_PLAY_X)+(row-BUTTON_PLAY_Y)*116;
    assign title_addr=(col-TITLE_X)+(row-TITLE_Y)*178;
    assign name_addr=(col-NAME_X)+(row-NAME_Y)*194;

    background BG(.addra(bg_addr), .douta(bg_data), .clka(DIV[1]));
    button_play
BUTTON(.addra(button_play_addr), .douta(button_play_data), .clka(DIV[1
]));
    title TITLE(.addra(title_addr), .douta(title_data), .clka(DIV[1]));
    name  NAME(.addra(name_addr), .douta(name_data), .clka(DIV[1]));
```

调用 PS2 键盘模块，此时我先用一个很高的时钟频率从模块中读取输入的 ascii，赋给 keyascii。此后，再用一个较低的频率，轮询 keyascii，并由之给相关的变量赋值。这样可以避免一次按键产生的脉冲太短。clr 是一个标记清零的变量，在判断已经处理了输入的 ascii 之后，clr 可以指示是否将 keyascii 清零。

```
    reg kenter, kspace;
    reg clr=0;
    wire [15:0]ascii;
    reg  [15:0]keyascii;
    ps2_keyboard_driver
PS2(.clk(DIV[0]), .clk_delay(DIV[5]), .rst_n(1'b1), .ps2k_clk(PS2C), .p
s2k_data(PS2D), .data_out(ascii));

    // read the input of keyboard
    always@(posedge DIV[0])begin
        if(ascii[15])begin
            keyascii<=ascii;
        end
        if(clr)begin
            keyascii<=0;
        end
    end

    always@(posedge DIV[20])begin
        clr<=0;
        if(keyascii[15])begin
            clr<=1;
            if(keyascii==16'h8020)begin
                kspace<=1;
            end
            else if(keyascii==16'h800D)begin
                kenter<=1;
            end
        end
        else begin
            kspace<=0;
            kenter<=0;
        end
    end
```

调用控制小鸟的模块，此处 flyup 是一个兼容了阵列键盘输入与 PS2 键盘输入的变量。

当有相应的按键被按下，其产生一个正脉冲。bird_x 与 bird_y 是小鸟图片左上角的坐标。

```
    wire [9:0]bird_x;
    wire [8:0]bird_y;
    wire [11:0]bird_data;
    wire flyup;                          // make the bird fly
    assign flyup=kspace || keyD;

    bird_con BIRD(.div(DIV), .playing(playing), .fly(flyup),
      .col(col), .row(row), .bird_x(bird_x), .bird_y(bird_y), .bird_data
(bird_data));
```

控制水管的移动，由于总共有八对水管，而每一对的控制都大同小异，因此仅分析其中一对。调用此前完成的模块，传入当前像素的地址以及水管的地址，可以很容易得到水管的像素信息。此处的 BREADTH 是初始的水管开口宽度。游戏不在进行状态时，还要对各水管的位置赋初值。若水管到达了屏幕之外，则让它回到上一对水管后方；否则，根据速度移动。

```
    localparam BREADTH=150;
    wire [11:0]p0d_data;
    wire [11:0]p0u_data;

    pillar_down
PILLARDOWN0(.div(DIV), .pillar_down_x(p0_x[9:0]), .pillar_y(p0_dy), .c
ol(col), .row(row), .pillardown_data(p0d_data));
    pillar_up
PILLARUP0(.div(DIV), .pillar_up_x(p0_x[9:0]), .pillar_y(p0_uy), .col(c
ol), .row(row), .pillarup_data(p0u_data));

  always@(posedge DIV[20])begin
        if(p0_x>=-52)begin
            p0_x<=p0_x-speed;
            if(p0_x>=639 && score>0)begin
                if(DIV[5])begin
                    p0_dy<=184-DIV[19:15];
                    p0_uy<=330-DIV[17:15];
                end
                else begin
                    p0_dy<=184+DIV[19:15];
                    p0_uy<=330+DIV[17:15];
                end
            end
        end
        else begin
            p0_x<=p7_x+90;
        end

        if(playing!=1)begin
            p0_x<=640; p0_dy<=184; p0_uy<=184+BREADTH;
            p1_x<=730;
            p2_x<=820;
            p3_x<=910;
            p4_x<=1000;
            p5_x<=1090;
            p6_x<=1180;
            p7_x<=1270;

        end
    end
```

此处为了方便验证与演示，我利用时钟分频信号来随机生成水管的位置。这种方法其实随机性并不太强，实际上可以以前一对水管的位置为基础、利用随机数模块来生成水管的位

置，该方法随机性更强故会使得游戏难度更大。代码如下：

```
  always@(posedge DIV[20])begin
       if(p0_x>=-52)begin
           p0_x<=p0_x-speed;
           if(p0_x>=639 && score>0)begin
               if(DIV[5])begin
                   p0_dy<=p7_dy-random[7:3];
                   p0_uy<=p7_dy+BREADTH-random[4:0];
               end
               else begin
                   p0_dy<=p7_dy+random[7:3];
                   p0_uy<=p7_dy+BREADTH+random[4:0];

               end
           end
       end
       else begin
           p0_x<=p7_x+90;
       end
    end
```

接下来是游戏的判定以及页面转换部分。该部分定义了两个变量，ishit 变量用以判定小鸟是否撞到了水管或者屏幕的上下边缘。inbetween 用来辅助实现计分功能：当小鸟处在水管开口中间，其为 1，此后如果小鸟的左边缘越过了水管的右边缘，则计一分。

计分时，为了让七段数码管显示十进制的分数，需要对变量每隔 4 位做一次判断。

当游戏失败，需要更新最高分。同时我设置了 LED 灯的数据，游戏失败时 Arduino 的八个 LED 灯会亮大约 1 秒而 SWORD 板上的 16 个 LED 灯会灭；其他状态下则是 SWORD 板上的 16 个 LED 灯会亮而 Arduino 上的 8 个 LED 灯会灭。代码如下：

```
  wire ishit, inbetween;    // whether the bird hits the pipe or the
boundary & whether the bird is between a pair of pipes
    assign ishit= (bird_y ==0) || (bird_y+36>=479 && bird_data!=12'h000)
||
                          (bird_x+41>=p0_x    &&    bird_x<=p0_x+44    &&
bird_y+12<=p0_dy) ||
                          (bird_x+41>=p1_x    &&    bird_x<=p1_x+44    &&
bird_y+12<=p1_dy) ||
                          (bird_x+41>=p2_x    &&    bird_x<=p2_x+44    &&
bird_y+12<=p2_dy) ||
                          (bird_x+41>=p3_x    &&    bird_x<=p3_x+44    &&
bird_y+12<=p3_dy) ||
                          (bird_x+41>=p4_x    &&    bird_x<=p4_x+44    &&
bird_y+12<=p4_dy) ||
                          (bird_x+41>=p5_x    &&    bird_x<=p5_x+44    &&
bird_y+12<=p5_dy) ||
                          (bird_x+41>=p6_x    &&    bird_x<=p6_x+44    &&
bird_y+12<=p6_dy) ||
                          (bird_x+41>=p7_x    &&    bird_x<=p7_x+44    &&
bird_y+12<=p7_dy) ||
                          (bird_x+41>=p0_x    &&    bird_x<=p0_x+44    &&
bird_y+36>=p0_uy) ||
                          (bird_x+41>=p1_x    &&    bird_x<=p1_x+44    &&
bird_y+36>=p1_uy) ||
                          (bird_x+41>=p2_x    &&    bird_x<=p2_x+44    &&
bird_y+36>=p2_uy) ||
                          (bird_x+41>=p3_x    &&    bird_x<=p3_x+44    &&
bird_y+36>=p3_uy) ||
                          (bird_x+41>=p4_x    &&    bird_x<=p4_x+44    &&
bird_y+36>=p4_uy) ||
```

```
                          (bird_x+41>=p5_x    &&    bird_x<=p5_x+44    &&
bird_y+36>=p5_uy) ||
                          (bird_x+41>=p6_x    &&    bird_x<=p6_x+44    &&
bird_y+36>=p6_uy) ||
                          (bird_x+41>=p7_x    &&    bird_x<=p7_x+44    &&
bird_y+36>=p7_uy) ;
    assign inbetween= (bird_x+41>=p0_x && bird_x<=p0_x+44) ||
                          (bird_x+41>=p1_x && bird_x<=p1_x+44) ||
                          (bird_x+41>=p2_x && bird_x<=p2_x+44) ||
                          (bird_x+41>=p3_x && bird_x<=p3_x+44) ||
                          (bird_x+41>=p4_x && bird_x<=p4_x+44) ||
                          (bird_x+41>=p5_x && bird_x<=p5_x+44) ||
                          (bird_x+41>=p6_x && bird_x<=p6_x+44) ||
                          (bird_x+41>=p7_x && bird_x<=p7_x+44) ;

    //wellcome - playing - over  & count the score
    reg [17:0]ledct=0;
    always@(posedge DIV[10])begin
        if(ledct!=0 && ledct[17]!=1)begin
            ledct<=ledct+1;
        end
        else begin
            LED[15:0]<=16'hFFFF;
            ledct<=0;
        end

        if(welcome==1 && (kenter || keyC))begin          // welcome ->
playing
            welcome<=0;
            over<=0;
            playing<=1;
        end
        else if(over==1 && (kenter || keyE))begin     // over -> welcome
            welcome<=1;
            over<=0;
            playing<=0;
        end
        else if(playing==1 && ishit)begin      // playing -> over
            welcome<=0;
            over<=1;
            playing<=0;
            LED[15:0]<=16'h0000;
            ledct<=ledct+1;
            if(score>maxscore)begin
                maxscore<=score;
                score<=0;
            end
            else begin
                maxscore<=maxscore;
                score<=0;
            end
        end
        else if(playing==1)begin
            if(inbetween)begin
                scorer<=1;
            end
            else if(scorer==1 && !inbetween)begin
                if(score[27:0]==28'h9999999)begin
                    score[31:28]<=score[31:28]+1;
                    score[27:0]<=0;
                end
```

```
                else if(score[23:0]==24'h999999)begin
                    score[27:24]<=score[27:24]+1;
                    score[23:0]<=0;
                end
                else if(score[19:0]==20'h99999)begin
                    score[23:20]<=score[23:20]+1;
                    score[19:0]<=0;
                end
                else if(score[15:0]==16'h9999)begin
                    score[19:16]<=score[19:16]+1;
                    score[15:0]<=0;
                end
                else if(score[11:0]==12'h999)begin
                    score[15:12]<=score[15:12]+1;
                    score[11:0]<=0;
                end
                else if(score[7:0]==8'h99)begin
                    score[11:8]<=score[11:8]+1;
                    score[7:0]<=0;
                end
                else if(score[3:0]==4'h9)begin
                    score[7:4]<=score[7:4]+1;
                    score[3:0]<=0;
                end
                else begin
                    score[3:0]<=score[3:0]+1;
                end
                scorer<=0;
            end
        end
    end
```

最后是画面的显示部分，这一部分的重点在于每个图片在每个像素点是否显示的判定。首先，我给每个图片一个变量，根据图片的坐标来计算其是否应该显示；其次，在 always 块内，我还对 welcome、over 和 playing 三个指示画面的变量作了判定；最后才是利用 case 语句来让图片显示在屏幕上。

由于代码与 2.4.1 中的代码大同小异，故不再附源代码。

2.4.7 实验框架

该部分为实验内容，此前已由本人根据课件独立完成，此处只做简略介绍。图 2.4-2 为本次设计的设计层次图。

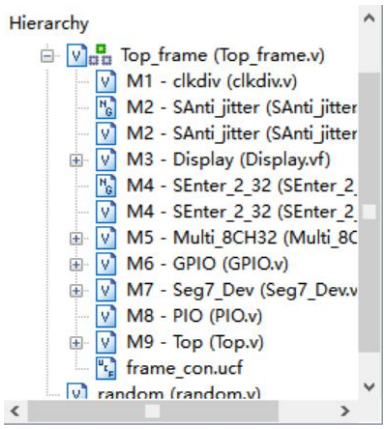

图 2.4-2 设计层次图

M1 模块为时钟分频模块，用以将 clk 时钟信号分频为更低频率的时钟信号。

M2 模块为阵列键盘去抖动模块，当按下阵列键盘的按键，其产生一个正脉冲。

M3 模块为辅助显示模块，实现将八位 16 进制数映射为七段码、选择七段数码管的显示模式、信号并行转串行以及在 Arduino 板的七段数码管上显示数据等功能。

M4 模块为阵列键盘功能模块，本模块与本设计关系不大，不做详细介绍。

M5 模块为选择器，选择八个数据中的其中一个显示在七段数码管上。

M6 模块为 SWORD 板上的 LED 控制模块，SWORD 上的 LED 采用高有效控制。

M7 模块为七段数码管显示模块，负责在七段数码管上显示数据。

M8 模块为 Arduino 上的 LED 控制模块，Arduino 上的 LED 采用低有效控制。

M9 模块即为本次设计中自主完成的 Flappy Bird 逻辑核心模块。

其中最高级的 Top_frame.v 代码如下：

```
    module Top_frame(input clk,
                     input RSTN,
                     input [3:0]K_COL,
                     input PS2C,
                     input PS2D,
                     output [4:0]K_ROW,
                     input [15:0]SW,
                     output CR,
                     output [3:0]AN,
                     output [7:0]SEGMENT,
                     output [7:0]LED,
                     output RDY,
                     output readn,
                     output SEGCLR,
                     output SEGEN,
                     output SEGDT,
                     output SEGCLK,
                     output LEDCLK,
                     output LEDDT,
                     output LEDCLR,
                     output LEDEN,
                     output wire hs,
                     output wire vs,
                     output wire [3:0]R, G, B
    );

    wire rst;
    wire [31:0]Div, Ai, Bi, Disp_num, score, maxscore;
    wire [4:0]Key_out;
    wire [3:0]Pulse, BTN_OK;
    wire [15:0]SW_OK;
    wire [7:0]blink, point_out, LE_out;
    clkdiv M1(.clk(clk), .rst(rst), .clkdiv(Div));

    SAnti_jitter
M2(.RSTN(RSTN), .clk(clk), .Key_y(K_COL), .Key_x(K_ROW), .SW(SW),

     .readn(readn), .CR(CR), .Key_out(Key_out), .Key_ready(RDY),

     .pulse_out(Pulse), .BTN_OK(BTN_OK), .SW_OK(SW_OK), .rst(rst));

    Display
M3(.flash(Div[25]), .Hexs(Disp_num), .point(point_out), .LES(LE_out),

      .SW0(SW_OK[0]), .Start(Div[10]), .rst(rst), .clk(clk), .seg_clrn(S
```

```
EGCLR),
                .SEG_PEN(SEGEN), .seg_sout(SEGDT), .seg_clk(SEGCLK));

    wire [4:0]SW_M4;
    assign SW_M4={SW_OK[7:5], SW_OK[15], SW_OK[0]};
    SEnter_2_32
M4(.clk(clk), .Din(Key_out), .D_ready(RDY), .BTN(BTN_OK[2:0]),

    .Ctrl(SW_M4), .readn(readn), .Ai(Ai), .Bi(Bi), .blink(blink));

    wire [63:0]M5point, M5les;
    assign M5point={Div[31:0], Div[31:0]};
    assign M5les={48'h000000000000, blink[7:0], blink[7:0]};
    Multi_8CH32
M5(.clk(clk), .rst(rst), .EN(1'b1), .Test(SW_OK[7:5]), .point_in(M5poin
t),

    .LES(M5les), .data0(score), .data1(maxscore), .data2(Ai), .data3(Bi)
,

    .point_out(point_out), .LE_out(LE_out), .Disp_num(Disp_num));

    wire [31:0]LEDdata;
    GPIO
M6(.clk(clk), .rst(rst), .Start(Div[20]), .EN(1'b1), .P_Data(LEDdata),
 .led_clk(LEDCLK),
            .led_sout(LEDDT), .led_clrn(LEDCLR), .LED_PEN(LEDEN));

    wire [2:0]M7_sc;
    assign M7_sc={SW_OK[1], Div[19:18]};
    Seg7_Dev
M7(.Scan(M7_sc), .Hexs(Disp_num), .point(point_out), .LES(LE_out), .SW0
(SW_OK[0]),
                .flash(Div[25]), .AN(AN), .SEGMENT(SEGMENT));

    PIO
M8(.clk(clk), .rst(1'b0), .EN(1'b1), .PData_in(LEDdata), .LED(LED));

    Top
M9(.DIV(Div), .rst(rst), .SW(SW_OK), .BTN_OK(BTN_OK), .Key(Key_out), .hs
(hs), .vs(vs), .R(R), .G(G), .B(B),

     .PS2C(PS2C), .PS2D(PS2D), .score(score), .maxscore(maxscore), .LED(
LEDdata));
endmodule
```

三、设计实现

3.1 VGA 显示

3.1.1 VGA 显示原理

在开发之初，一大疑惑就是如何理解 VGA 的工作原理。在研究书本的介绍以及相关代码后，不难发现，其实原理不难。

扫描的功能已经在驱动内实现，只要有合理的时钟信号以及清零信号 clrn 不为 0，VGA 就处于正常工作状态。而其输出 col_addr 与 row_addr，其实是在告诉调用者，VGA 驱动当前正在处理哪一个像素点。调用者只需要根据 VGA 驱动给出的当前处理的像素点的坐标，告诉 VGA 驱动要显示什么颜色，即把希望呈现的 RGB 信息赋值给 VGA 驱动的输入 d_in，即

可实现 VGA 的正常显示。

3.1.2 多图片显示

多个图片显示在 VGA 上教材其实已经给出了一个很好的答案，即利用 case 语句。如下例子，TITLE_DIS、BUTTON_DIS 与 NAME_DIS 是各个图片是否显示的条件。当赋给 case 的变量为 1'b1 时，程序自上而下寻找第一个值为 1'b1 的条件（事实上，这代表着各个图片显示的优先级）。若找到一个条件为 1'b1，则程序将该图片的像素信息赋值给 VGA，让 VGA 显示出来。

```
always@(posedge DIV[1])begin
    if(welcome)begin
        case(1'b1)
            TITLE_DIS:    vgadata<=title_data;
            BUTTON_DIS: vgadata<=button_play_data;
            NAME_DIS:   vgadata<=name_data;
            default:   begin
                           vgadata[11:8]<=bg_data[11:8]/4;
                           vgadata[7:4]<=bg_data[7:4]/4;
                           vgadata[3:0]<=bg_data[3:0]/4;
                       end
        endcase
    end
end
```

3.1.3 VGA 驱动验证结果

如前所述，为了验证 VGA 驱动设计的正确性，首先尝试让 VGA 显示欢迎页面。如图 3.1-1 所示，显示正常。

图 3.1-1 欢迎页面测试

3.2 按钮功能

3.2.1 阵列键盘与 PS2 键盘兼容

实际开发中，这一部分处理较为简单。通过上文的代码不难看出，我新增了一个变量

flyup，并且令它为 PS2 键盘空格输入与阵列键盘输入取或运算的结果。回车输入与阵列键盘输入也可以做类似处理而实现 PS2 键盘输入与阵列键盘输入的兼容。

```
    wire flyup;                          // make the bird fly
    assign flyup=kspace || keyD;
```

3.2.2 小鸟飞翔优化

在小鸟飞翔时，由于小鸟一次向上飞的距离比较长，若在一帧之内变化完成，则会有强烈的瞬移感。为了让飞翔的动作看起来更连贯，我将一次飞翔拆分至 8 帧，每一帧上移 7 个像素，使得上移看起来更加连贯。利用一个计数变量 flyct，当 0≤flyct<7，小鸟每帧上移 7 个像素。在代码中体现如下：

```
  reg [3:0]flyct=0;
    always@(posedge div[20])begin          // control the position of the
bird
        if(playing==1'b0)begin
            bird_x<=BIRD_INI_X;
            bird_y<=BIRD_INI_Y;
            flyct<=0;
        end
        else if((fly && flyct!=7) || (flyct!=0 && flyct!=7))begin
            bird_y<=bird_y-7;
            flyct<=flyct+1;
        end
        else begin
            bird_y<=bird_y+gravity;
            flyct<=0;
        end
    end
```

3.3 位置控制

位置控制较为简单，主要方法是给每一个需要移动的图片都设置坐标变量，如给小鸟图片设置的坐标变量即为 bird_x 与 bird_y，这两个变量标记小鸟图片的左上角。当图片需要移动，只需要变化这两个变量，即可实现移动。

而要让图片在 VGA 上动起来，还需要设置是否显示该图片的条件，如显示小鸟需要判断当前像素点是否在小鸟的图片范围内。需要注意的是，由于所用图片不规则，我手动给每一张图片设置了黑色的背景色。在判断是否显示图片时，还需要判断当前图片像素点是不是纯黑的 FFF，如果是，说明当前像素点处在图片的边缘部分，不显示；如果不是纯黑，说明当前像素点为图片主题的一部分，应该显示。以小鸟的显示条件为例：

```
  assign
BIRD_DIS=(col>bird_x)&&(col<=bird_x+48)&&(row>=bird_y)&&(row<=bird_y+
47)&&(bird_data!=12'h000);
```

前四个不等式比较即是在判断当前像素点是否在图片范围内，最后一个不等式即是在判断图片图片像素点是否为纯黑。

3.4 成败判定

该部分不难实现，只需要设置一个变量，标记小鸟的图片主体与水管的图片主体是否有重合，以及小鸟的坐标是否超出了屏幕范围。所设置的变量即为 ishit。

```
assign ishit= (bird_y ==0) || (bird_y+36>=479 && bird_data!=12'h000) ||
                          (bird_x+41>=p0_x    &&    bird_x<=p0_x+44    &&
bird_y+12<=p0_dy) ||
```

```
                        (bird_x+41>=p1_x    &&    bird_x<=p1_x+44    &&
bird_y+12<=p1_dy) ||
                        (bird_x+41>=p2_x    &&    bird_x<=p2_x+44    &&
bird_y+12<=p2_dy) ||
                        (bird_x+41>=p3_x    &&    bird_x<=p3_x+44    &&
bird_y+12<=p3_dy) ||
                        (bird_x+41>=p4_x    &&    bird_x<=p4_x+44    &&
bird_y+12<=p4_dy) ||
                        (bird_x+41>=p5_x    &&    bird_x<=p5_x+44    &&
bird_y+12<=p5_dy) ||
                        (bird_x+41>=p6_x    &&    bird_x<=p6_x+44    &&
bird_y+12<=p6_dy) ||
                        (bird_x+41>=p7_x    &&    bird_x<=p7_x+44    &&
bird_y+12<=p7_dy) ||
                        (bird_x+41>=p0_x    &&    bird_x<=p0_x+44    &&
bird_y+36>=p0_uy) ||
                        (bird_x+41>=p1_x    &&    bird_x<=p1_x+44    &&
bird_y+36>=p1_uy) ||
                        (bird_x+41>=p2_x    &&    bird_x<=p2_x+44    &&
bird_y+36>=p2_uy) ||
                        (bird_x+41>=p3_x    &&    bird_x<=p3_x+44    &&
bird_y+36>=p3_uy) ||
                        (bird_x+41>=p4_x    &&    bird_x<=p4_x+44    &&
bird_y+36>=p4_uy) ||
                        (bird_x+41>=p5_x    &&    bird_x<=p5_x+44    &&
bird_y+36>=p5_uy) ||
                        (bird_x+41>=p6_x    &&    bird_x<=p6_x+44    &&
bird_y+36>=p6_uy) ||
                        (bird_x+41>=p7_x    &&    bird_x<=p7_x+44    &&
bird_y+36>=p7_uy) ;
```

3.5 随机数仿真结果

建立仿真文件，输入如下仿真代码。

```
  initial begin
        // Initialize Inputs
        clk = 0;
        seed = 1;
        load = 0;

        // Wait 100 ns for global reset to finish
        #100;

        // Add stimulus here

        fork
            forever #10 clk<=~clk;
            begin
            #100; load<=1;
            #100; load<=0;
            end
        join
    end
```

从图 3.5-1 可见，该模块产生的数分布较为均匀，没有规律。在给定较高时钟频率的情况下，随机性要比直接利用时钟分频信号要好。

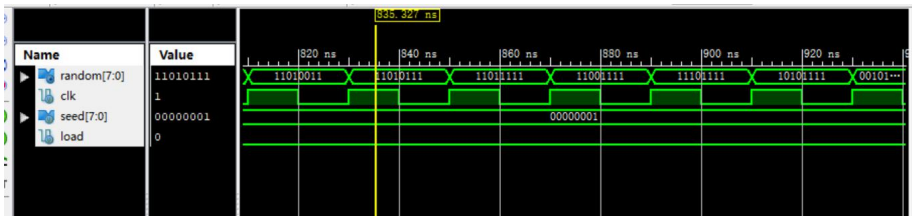

图 3.5-1 随机数仿真结果

3.6 时钟频率

开发中遇到的最大的难题，应该是时钟频率的问题。利用时钟分频模块，可以得到一个32 位的时钟分频信号 DIV。

首先是开发之初，我希望能尽量提高 VGA 的刷新率，因此给 VGA 驱动选择的时钟信号为 DIV[0]。物理验证时发现，此时的 VGA 会提示“OUT OF RANGE”，指选取的时钟信号频率过高，VGA 无法正常工作。因此，给 VGA 选择的时钟信号最高只能为 DIV[1]。

其次，我的程序利用变量 vgadata 给 VGA 输入像素的 RGB 信息。给其赋值的 always 块选取的时钟信号的频率不可低于给 VGA 驱动选取的时钟信号。如我的程序中二者都使用了 DIV[1]。否则，如果 always 块选择的时钟信号频率过低，则 VGA 处理的像素点发生变化时，vgadata 仍为经过的像素点的 RGB 信息，体现为屏幕显示许多水平长条，没有正常的图像。

再次，经过计算可以得知 VGA 显示一帧的时间为 307200 个时钟周期。当赋给其的时钟信号为 DIV[1]时，DIV[20]是能指示一帧周期的最低时钟信号。图像变化的频率不应高于 DIV[20]的频率太多，否则可能出现图像拉长、重复等现象。

最后，善用不同频率的时钟分频信号有利于优化开发的项目。如若要生成随机数，可以采用高频率的信号，增强其随机性；若是移动某个图像，则其移动的频率不可以高于帧率。

四、完整测试

开始物理验证之前，先附上引脚约束代码如下：

```
NET "clk" LOC = AC18 | IOSTANDARD = LVCMOS18;
NET"RSTN"LOC=W13 | IOSTANDARD=LVCMOS18;
NET"PS2C"LOC=N18 | IOSTANDARD=LVCMOS33 | PULLUP;
NET"PS2D"LOC=M19 | IOSTANDARD=LVCMOS33 | PULLUP;

NET "K_ROW[0]"LOC=V17 | IOSTANDARD=LVCMOS18;
NET "K_ROW[1]"LOC=W18 | IOSTANDARD=LVCMOS18;
NET "K_ROW[2]"LOC=W19 | IOSTANDARD=LVCMOS18;
NET "K_ROW[3]"LOC=W15 | IOSTANDARD=LVCMOS18;
NET "K_ROW[4]"LOC=W16 | IOSTANDARD=LVCMOS18;

NET "K_COL[0]"LOC=V18 | IOSTANDARD=LVCMOS18;
NET "K_COL[1]"LOC=V19 | IOSTANDARD=LVCMOS18;
NET "K_COL[2]"LOC=V14 | IOSTANDARD=LVCMOS18;
NET "K_COL[3]"LOC=W14 | IOSTANDARD=LVCMOS18;

NET"readn"LOC=U21 | IOSTANDARD=LVCMOS33;
NET"RDY"LOC=U22 | IOSTANDARD=LVCMOS33;
NET"CR"LOC=V22 | IOSTANDARD=LVCMOS33;
NET"SEGCLK"LOC=M24 | IOSTANDARD=LVCMOS33;
NET"SEGCLR"LOC=M20 | IOSTANDARD=LVCMOS33;
NET"SEGDT"LOC=L24 | IOSTANDARD=LVCMOS33;
NET"SEGEN"LOC=R18 | IOSTANDARD=LVCMOS33;
```

```
NET"LEDCLK"LOC=N26 | IOSTANDARD=LVCMOS33;
NET"LEDCLR"LOC=N24 | IOSTANDARD=LVCMOS33;
NET"LEDDT"LOC=M26 | IOSTANDARD=LVCMOS33;
NET"LEDEN"LOC=P18 | IOSTANDARD=LVCMOS33;

NET"SW[0]"LOC=AA10 | IOSTANDARD=LVCMOS15;
NET"SW[1]"LOC=AB10 | IOSTANDARD=LVCMOS15;
NET"SW[2]"LOC=AA13 | IOSTANDARD=LVCMOS15;
NET"SW[3]"LOC=AA12 | IOSTANDARD=LVCMOS15;
NET"SW[4]"LOC=Y13 | IOSTANDARD=LVCMOS15;
NET"SW[5]"LOC=Y12 | IOSTANDARD=LVCMOS15;
NET"SW[6]"LOC=AD11 | IOSTANDARD=LVCMOS15;
NET"SW[7]"LOC=AD10 | IOSTANDARD=LVCMOS15;
NET"SW[8]"LOC=AE10 | IOSTANDARD=LVCMOS15;
NET"SW[9]"LOC=AE12 | IOSTANDARD=LVCMOS15;
NET"SW[10]"LOC=AF12 | IOSTANDARD=LVCMOS15;
NET"SW[11]"LOC=AE8 | IOSTANDARD=LVCMOS15;
NET"SW[12]"LOC=AF8 | IOSTANDARD=LVCMOS15;
NET"SW[13]"LOC=AE13 | IOSTANDARD=LVCMOS15;
NET"SW[14]"LOC=AF13 | IOSTANDARD=LVCMOS15;
NET"SW[15]"LOC=AF10 | IOSTANDARD=LVCMOS15;

#NET"Buzzer"LOC=AF25 | IOSTANDARD=LVCMOS33;
NET"SEGMENT[0]"LOC=AB22 | IOSTANDARD=LVCMOS33;
NET"SEGMENT[1]"LOC=AD24 | IOSTANDARD=LVCMOS33;
NET"SEGMENT[2]"LOC=AD23 | IOSTANDARD=LVCMOS33;
NET"SEGMENT[3]"LOC=Y21 | IOSTANDARD=LVCMOS33;
NET"SEGMENT[4]"LOC=W20 | IOSTANDARD=LVCMOS33;
NET"SEGMENT[5]"LOC=AC24 | IOSTANDARD=LVCMOS33;
NET"SEGMENT[6]"LOC=AC23 | IOSTANDARD=LVCMOS33;
NET"SEGMENT[7]"LOC=AA22 | IOSTANDARD=LVCMOS33;

NET"AN[0]"LOC=AD21 | IOSTANDARD=LVCMOS33;
NET"AN[1]"LOC=AC21 | IOSTANDARD=LVCMOS33;
NET"AN[2]"LOC=AB21 | IOSTANDARD=LVCMOS33;
NET"AN[3]"LOC=AC22 | IOSTANDARD=LVCMOS33;


NET "LED[0]"LOC=W23 | IOSTANDARD=LVCMOS33;
NET "LED[1]"LOC=AB26 | IOSTANDARD=LVCMOS33;
NET "LED[2]"LOC=Y25 | IOSTANDARD=LVCMOS33;
NET "LED[3]"LOC=AA23 | IOSTANDARD=LVCMOS33;
NET "LED[4]"LOC=Y23 | IOSTANDARD=LVCMOS33;
NET "LED[5]"LOC=Y22 | IOSTANDARD=LVCMOS33;
NET "LED[6]"LOC=AE21 | IOSTANDARD=LVCMOS33;
NET "LED[7]"LOC=AF24 | IOSTANDARD=LVCMOS33;

NET "R[0]" LOC = N21 | IOSTANDARD = LVCMOS33 | SLEW = FAST;
NET "R[1]" LOC = N22 | IOSTANDARD = LVCMOS33 | SLEW = FAST;
NET "R[2]" LOC = R21 | IOSTANDARD = LVCMOS33 | SLEW = FAST;
NET "R[3]" LOC = P21 | IOSTANDARD = LVCMOS33 | SLEW = FAST;
NET "G[0]" LOC = R22 | IOSTANDARD = LVCMOS33 | SLEW = FAST;
NET "G[1]" LOC = R23 | IOSTANDARD = LVCMOS33 | SLEW = FAST;
NET "G[2]" LOC = T24 | IOSTANDARD = LVCMOS33 | SLEW = FAST;
NET "G[3]" LOC = T25 | IOSTANDARD = LVCMOS33 | SLEW = FAST;
NET "B[0]" LOC = T20 | IOSTANDARD = LVCMOS33 | SLEW = FAST;
NET "B[1]" LOC = R20 | IOSTANDARD = LVCMOS33 | SLEW = FAST;
NET "B[2]" LOC = T22 | IOSTANDARD = LVCMOS33 | SLEW = FAST;
NET "B[3]" LOC = T23 | IOSTANDARD = LVCMOS33 | SLEW = FAST;
NET "hs" LOC = M22 | IOSTANDARD = LVCMOS33 | SLEW = FAST;
NET "vs" LOC = M21 | IOSTANDARD = LVCMOS33 | SLEW = FAST;
```

欢迎页面已经在上文测试并展示，在此处不再展示。

如图 4-1，可见游戏页面正常。

图 4-1 游戏页面

如图 4-2 可见，利用阵列键盘操控游戏时，七段数码管可以正常显示分数。这同时测试了分数显示功能与阵列键盘的操控功能。

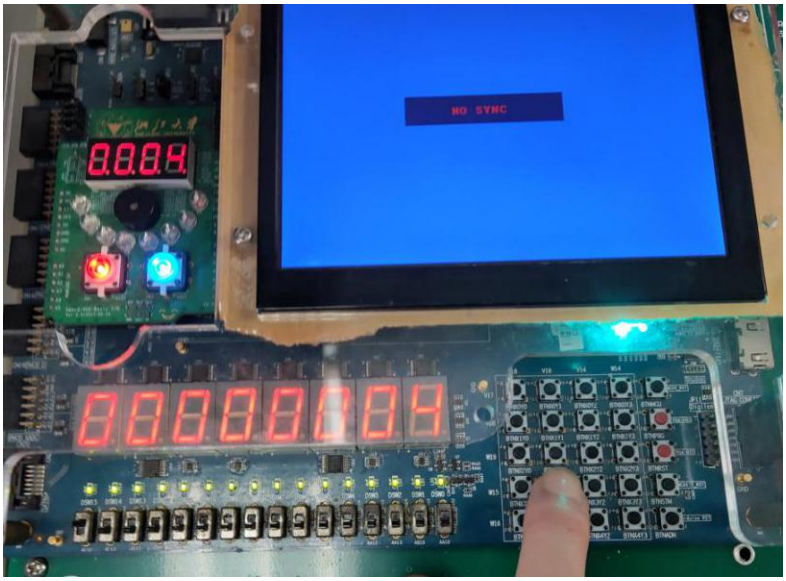

图 4-2 阵列键盘操控

如图 4-3，利用键盘操控时，游戏也可以正常进行，七段数码管可以正常显示当前分数。可知 PS2 键盘功能正常。

图 4-3 键盘操控

如图 4-4 可见，SW[7:5]=001 时，七段数码管显示游戏最高分 27 分，为十进制。此时，Arduino 上的 8 个 LED 灭而 SWORD 上的 16 个 LED 亮。

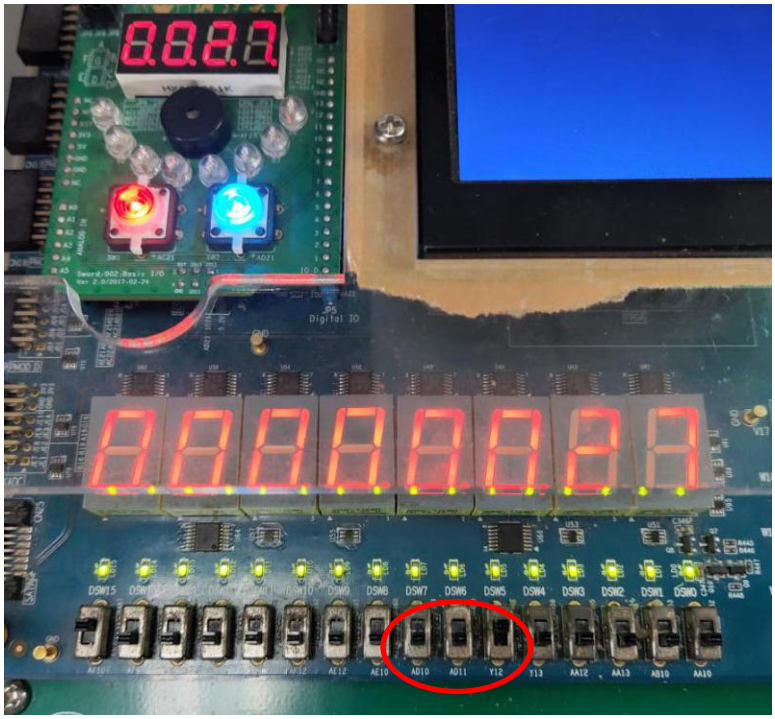

图 4-4 显示最高分

如图 4-5 可见，游戏失败时，Arduino 上的 8 个 LED 亮起约 1 秒而 SWORD 上的 16 个 LED 熄灭约 1 秒，并且当前游戏分数清零。此后按下回车或者 BTN[10]，又可以回到欢迎页面。

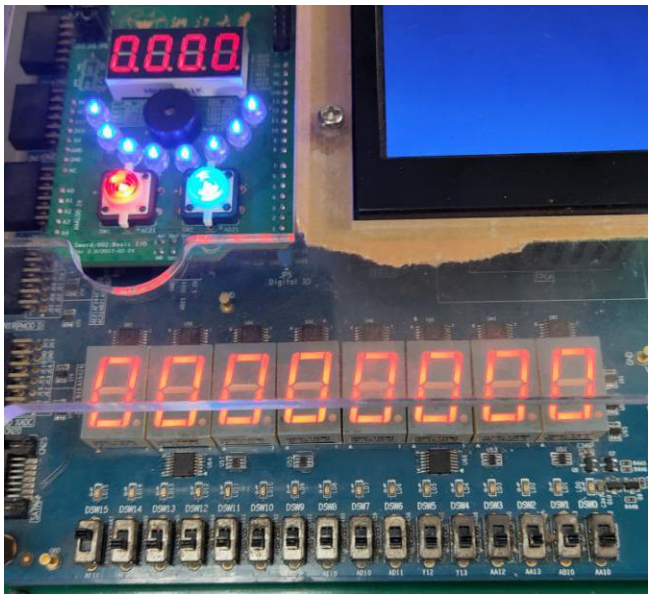

图 4-5 游戏失败

综上所述，可知希望设计并测试的功能一切正常。

五、结论与展望

本次课程设计可谓具有一定难度，主要体现在完全陌生的 VGA 以及 PS2 键盘的运用。当然，还有很大一部分困难来自于自主设计。以往的实验内容，大多是对着老师的课件，一边敲代码一边理解代码。而这一次，是要完全出于自己的理解，敲下自己的代码，实现自己期望的功能。

开发下来，收获很大。开发过程中遇到的许多问题，增进了我对 Verilog 代码的理解，如<=赋值符号。一次课程设计，增强了我学好硬件课的信心。

也必须指出，本次设计并不完美。譬如受限于 VGA 的刷新率，当游戏难度加大，水管移动速度加快时，水管会出现较明显的断层；又比如，游戏增加难度的方式只有加快速度，而还可以考虑改变水管的密度。这都是未来可以改进的地方。