浙江大学

课程设计报告

中文题目 ： FPGA 纸上小鸟游戏

英文题目： “Birds on paper" Based on FPGA

姓名/学号： 蔡田 3170105781

论文提交日期 2019 年 1 月 14 日

摘要

本次课程设计题目为“纸上小鸟”，这是基于 FPGA 设计的一款原创小游戏，不像贪吃蛇以及打地鼠或者俄罗斯方块等老牌游戏，这款游戏创新之处就在于作为一款双人游戏，能够真正为玩家带来快乐，并且具有一定竞技性。在试玩过的玩家中广受好评。游戏的源代码为手敲，游戏所有图片为手绘，是一个原汁原味的原创游戏。

关键词： 数字逻辑， 游戏， Verilog， IP Core， FPGA

目录

第一章 绪论

1.1 基于 FPGA 的游戏设计背景

Verilog HDL 是一种用于数字系统设计的语言。用 Verilog HDL 描述的电路设计就是该电路的 Verilog HDL 模型也称为模块。Verilog HDL 既是一种行为描述的语言也是一种结构描述的语言。这也就是说，无论描述电路功能行为的模块或描述元器件或较大部件互连的模块都可以用 Verilog 语言来建立电路模型。如果按照一定的规矩编写，功能行为模块可以通过工具自动地转换为门级互连模块。Verilog 模型可以是实际电路的不同级别的抽象。一个复杂电路系统的完整 Verilog HDL 模型是由若干个 Verilog HDL 模块构成的，每一个模块又可以由若干个子模块构成。其中有些模块需要综合成具体电路，而有些模块只是与用户所设计的模块有交互联系的现存电路或激励信号源。利用 Verilog HDL 语言结构所提供的这种功能就可以构造一个模块间的清晰层次结构来描述极其复杂的大型设计，并对所作设计的逻辑电路进行严格的验证。Verilog HDL 作为一种高级的硬件描述编程语言，与 C 语言的风格有许多类似之处。其中有许多语句如：if 语句、case 语句等和 C 语言中的对应语句十分相似。

FPGA 即现场可编程门阵列，它是在 PAL、GAL、CPLD 等可编程器件的基础上进一步发展的产物。它是作为专用集成电路（ASIC）领域中的一种半定制电路而出现的，既解决了定制电路的不足，又克服了原有可编程器件门电路数有限的缺点。

1.2 主要内容和难点

1.1.1 主要内容： **“纸上小鸟”是一款双人竞技游戏，两位玩家通过控制键盘上的 WADS 以及上下左右按键分别控制左右两只小鸟，当一只小鸟踩过另外一只小鸟头顶，则表示获胜。注意，游戏的趣味性和竞技性在于，键盘并未进行“无冲突”设置，也就是说在某按键长按时，另一按键的插入会打断当前按键的输入，也即键盘只可以单独输入。意味着双方玩家为了获得胜利必须疯狂的不断点按操作按键，比较的是玩家的手速和策略。当玩家进行游戏时，往往伴随着劈里啪啦的键盘声响和阵阵笑声。**

1.1.2 技术要求：熟练掌握 Verilog 语言设计；掌握 SWORD 板的使用方法；掌握 VGA 显示原理和 PS2 键盘输入原理，理解并掌握状态机的设计。

1.1.3 目的：加深对模块化编程的理解，加强对硬件编程语言 Verilog 的掌握。

1.1.4 实现重点与难点：IP 核的制作；键盘的输入；VGA 的正常显示。

第二章 “纸上小鸟”设计思路与原理

2.1 基于 FPGA 的游戏设计相关内容

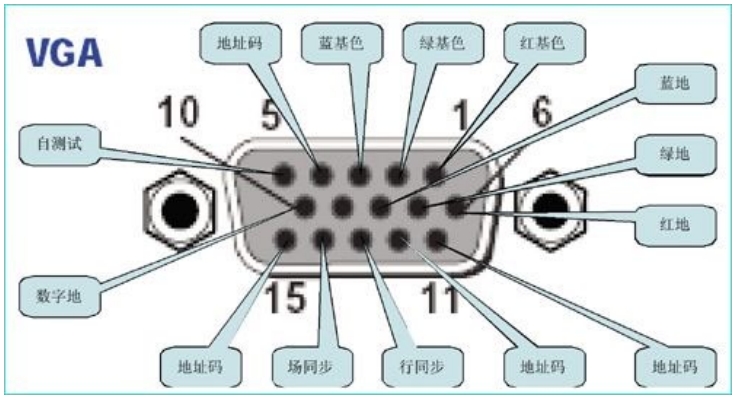
2.1.1 VGA 显示

VGA（Video Graphics Array），信号类型为模拟类型，显示卡端的接口为 15 针母插座.VGA(Video Graphics Array)作为一种标准的显示接口得到了广泛的应用。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。而在应用中，讨论的都是图形显示模式。

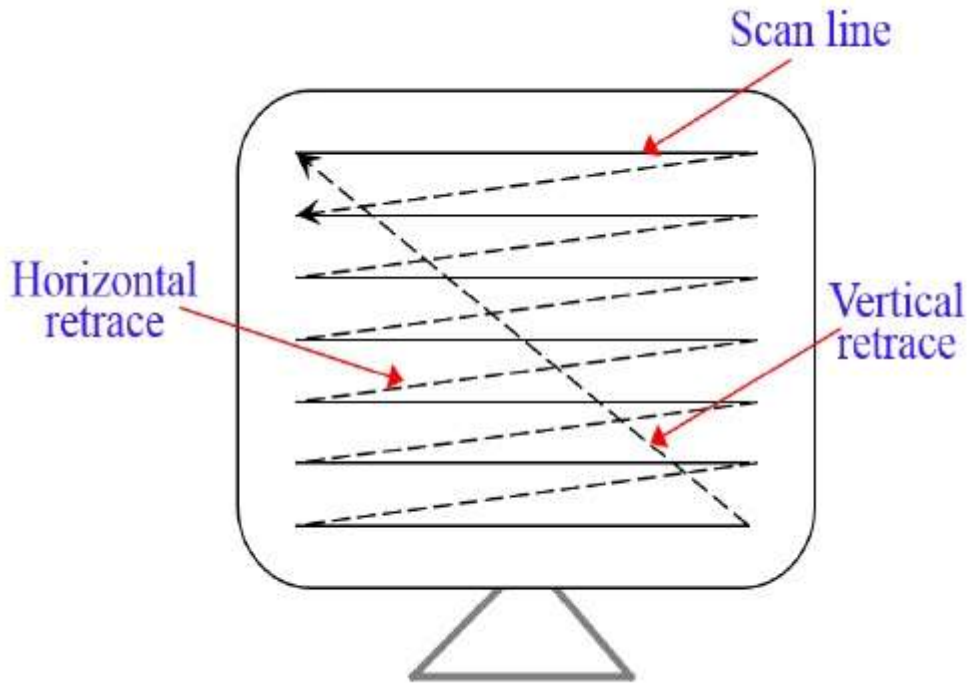
VGA 接口是一种 D 型接口，上面共有 15 针孔，分成三排，每排五个。 其中比较重要的是 3 根 RGB 彩色分量信号和 2 根扫描同步信号 HSYNC 和 VSYNC 针。其引脚编号图如下图所示：

vga 针脚定义

VGA 15 针母插座　　VGA 15 针公插头

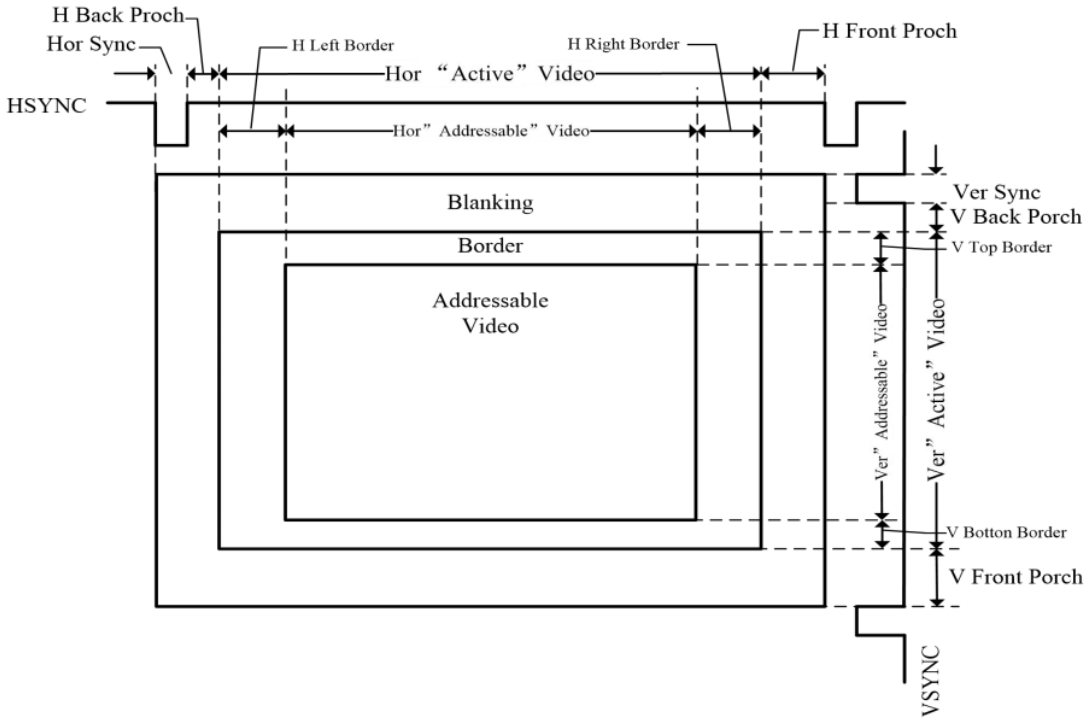


VGA 显示器扫描方式从屏幕左上角一点开始，从左向右逐点扫描，每扫描完一行,电子束回到屏幕的左边下一行的起始位置，在这期间，CRT 对电子束进行消隐，每行结束时，用行同步信号进行同步；当扫描完所有的行，形成一帧，用场同步信号进行场同步，并使扫描回到屏幕左上方，同时进行场消隐,开始下一帧。完成一行扫描的时间称为水平扫描时间，其倒数称为行频率；完成一帧（整屏）扫描的时间称为垂直扫描时间，其倒数称为场频率，即屏幕的刷新频率，常见的有 60Hz，75Hz 等等，但标准的 VGA 显示的场频 60Hz。其扫描示意图如下图所示



Raster scanning terminology for a single output frame.

VGA 的详细时序如下图所示：

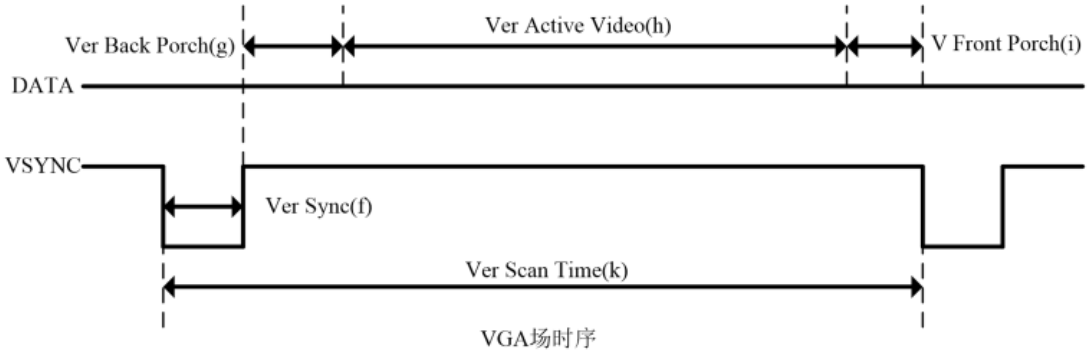


总的来说，VGA 的时序主要包括行时序与场时序两个部分。

其中行时序主要包括：行同步(Hor Sync) 、行消隐(Hor Back Porch) 、行视频有效(Hor Active Video)和行前肩(Hor Front Porch)这四个参数，行时序的时序图如下图所示

VGA行时序

而场时序主要包括：场同步(Ver Sync) 、场消隐(Ver Back Porch) 、场视频有效(Ver Active Video)和场前肩(Ver Front Porch)这四个参数，场时序的时序图如下图所示



需要注意的有三点：

1、行时序是以”像素”为单位的， 场时序是以”行”为单位的。

2、VGA 工业标准显示模式要求：行同步，场同步都为负极性，即同步脉冲要求是负脉冲。

3、VGA 行时序对行同步时间、 消隐时间、 行视频有效时间和行前肩时间有特定的规范， 场时序也是如此。 常用 VGA 分辨率时序参数如下表所示

VGA 常用分辨率时序参数

显示模式	时钟	行时序参数(单位：像素)					列时序参数(单位：行)				
	/MHz	a	b	c	d	e	f	g	h	i	k
640x480@60Hz	25.175	96	48	640	16	800	2	33	480	10	525
800x600@60Hz	40	128	88	800	40	1056	4	23	600	1	623
1024x768@60Hz	65	136	160	1024	24	1344	6	29	768	3	806
1280x720@60Hz	74.25	40	220	1280	110	1650	5	20	720	5	750
1280x1024@60Hz	108	112	248	1280	48	1688	3	38	1024	1	1066
1920x1080@60Hz	148.5	44	148	1920	88	2200	5	36	1080	4	1125

2.1.2 PS2 键盘输入

PS/2 通信协议是一种双向同步串行通信协议。通信的两端通过 CLOCK(时钟脚)同步，并通过国 DATA(数据脚)交换数据。一般两设备间传输数据的最大时钟频率是 33kHz，大多数 PS/2 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，具体含义如下图所示

数据	含义
1 个起始位	总是逻辑 0
8 个数据位	(LSB)地位在前
1 个奇偶校验位	奇校验
1 个停止位	总是逻辑 1
1 个应答位	仅用在主机对设备的通信中

PS/2 到主机的通信时序如下图所示。数据在 PS/2 时钟的下降沿读取，PS/2 的时钟频率为 10—16.7kHz。对于 PS/2 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。主机可以再第 11 个时钟脉冲停止位之前把时钟线拉低，使设备放弃发送当前字节，当然这种情况比较少见。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。

2.1.3 可编程阵列逻辑

PAL 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。因此 PLA 等于扮演（晶片）系统内含的逻辑状态图（state diagram）角色。

2.1.4 硬件描述语言

Verilog HDL 是一种用于数字系统设计的语言。用 Verilog HDL 描述的电路设计就是该电路的 Verilog HDL 模型也称为模块。Verilog HDL 既是一种行为描述的语言也是一种结构描述的语言。这也就是说，无论描述电路功能行为的模块或描述元器件或较大部件互连的模块都可以用 Verilog 语言来建立电路模型。如果按照一定的规矩编写，功能行为模块可

以通过工具自动地转换为门级互连模块。Verilog 模型可以是实际电路的不同级别的抽象。一个复杂电路系统的完整 Verilog HDL 模型是由若干个 Verilog HDL 模块构成的，每一个模块又可以由若干个子模块构成。其中有些模块需要综合成具体电路，而有些模块只是与用户所设计的模块有交互联系的现存电路或激励信号源。利用 Verilog HDL 语言结构所提供的这种功能就可以构造一个模块间的清晰层次结构来描述极其复杂的大型设计，并对所作设计的逻辑电路进行严格的验证。Verilog HDL 作为一种高级的硬件描述编程语言，与 C 语言的风格有许多类似之处。其中有许多语句如：if 语句、case 语句等和 C 语言中的对应语句十分相似。

2.1.5 技术工具

1. ISE 软件

   SE 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. SWORD 板

3. 计算机

4. PS 软件

用于对图片进行美术处理和加工，同时对图片像素进行适当裁剪以适应 VGA 显示。

5. Matlab 软件

通过 Matlab 编写脚本，把 jpg 图片转成 3 位 16 进制 coe 文件，使之与 VGA 显示相适应。

2.1.6 课程设计方法

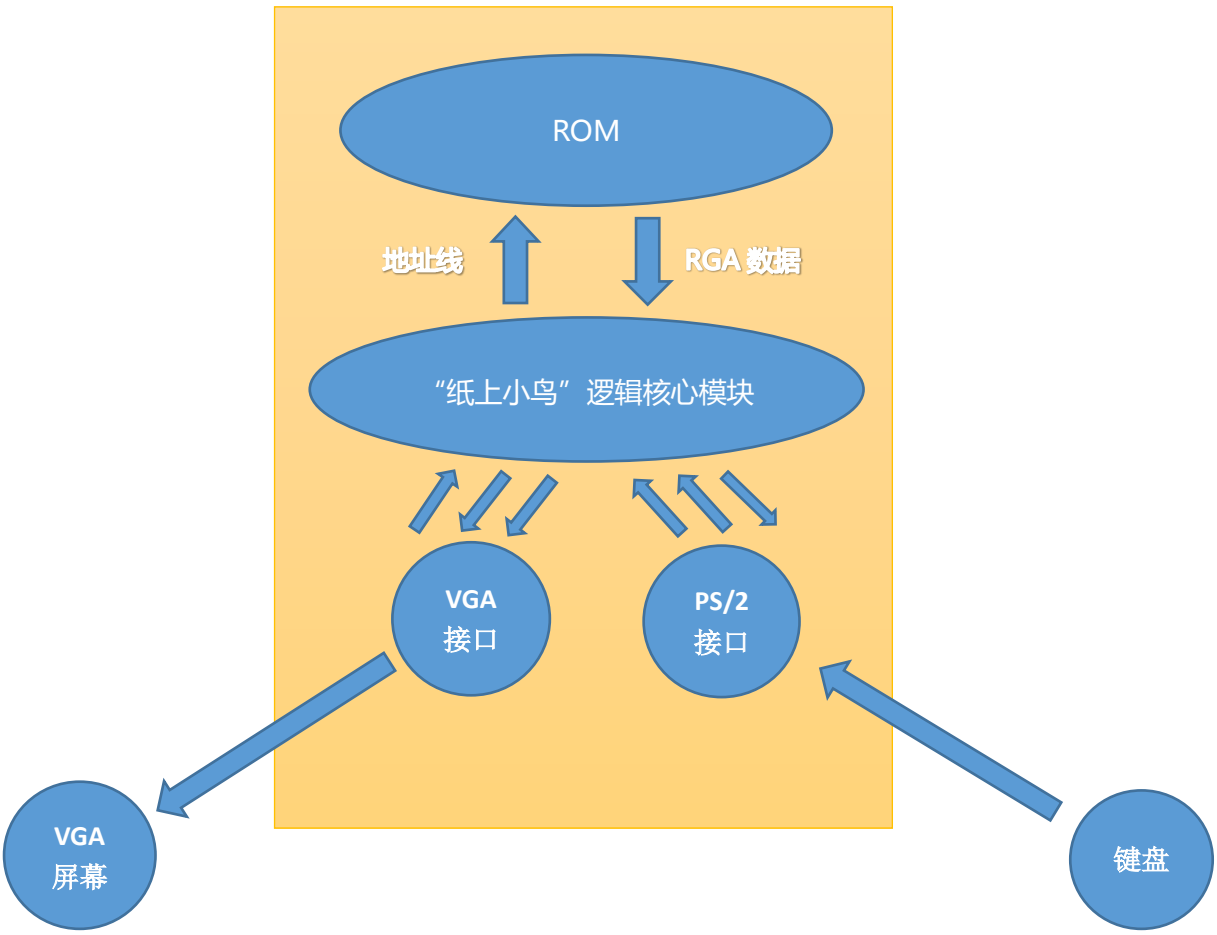
利用 Verilog HDL 硬件设计游戏，通过手绘设计游戏画面和贴图，通过 Matlab 编写脚本将图片转成 coe 并生成 IP 核，实现用 PS2 键盘输入数据，用 VGA 显示数据，最后生成 bit 文件，利用可编程阵列逻辑烧到板子上，实现最终的人机互动游戏。

2.2 基于 FPGA 的“纸上小鸟“的设计方案

2.2.1 技术需求：

VGA 扫描显示欢迎界面，游戏界面以及结束界面；PS2 键盘输入数据处理为切换界面信号、人物移动信号；核心模块对输入信号用状态机进行逻辑处理，输出。

2.2.2 整体设计方案：



“纸上小鸟“核心模块为游戏主要代码实现，VGA 接口负责核心模块与 VGA 显示器之间数据传输，PS2 接口负责键盘和核心模块之间的数据传输。ROM 存有图片 IP 核，同样与核心模块之间进行数据传输。

2.2.3 游戏流程与操作：

1. **初始化：**

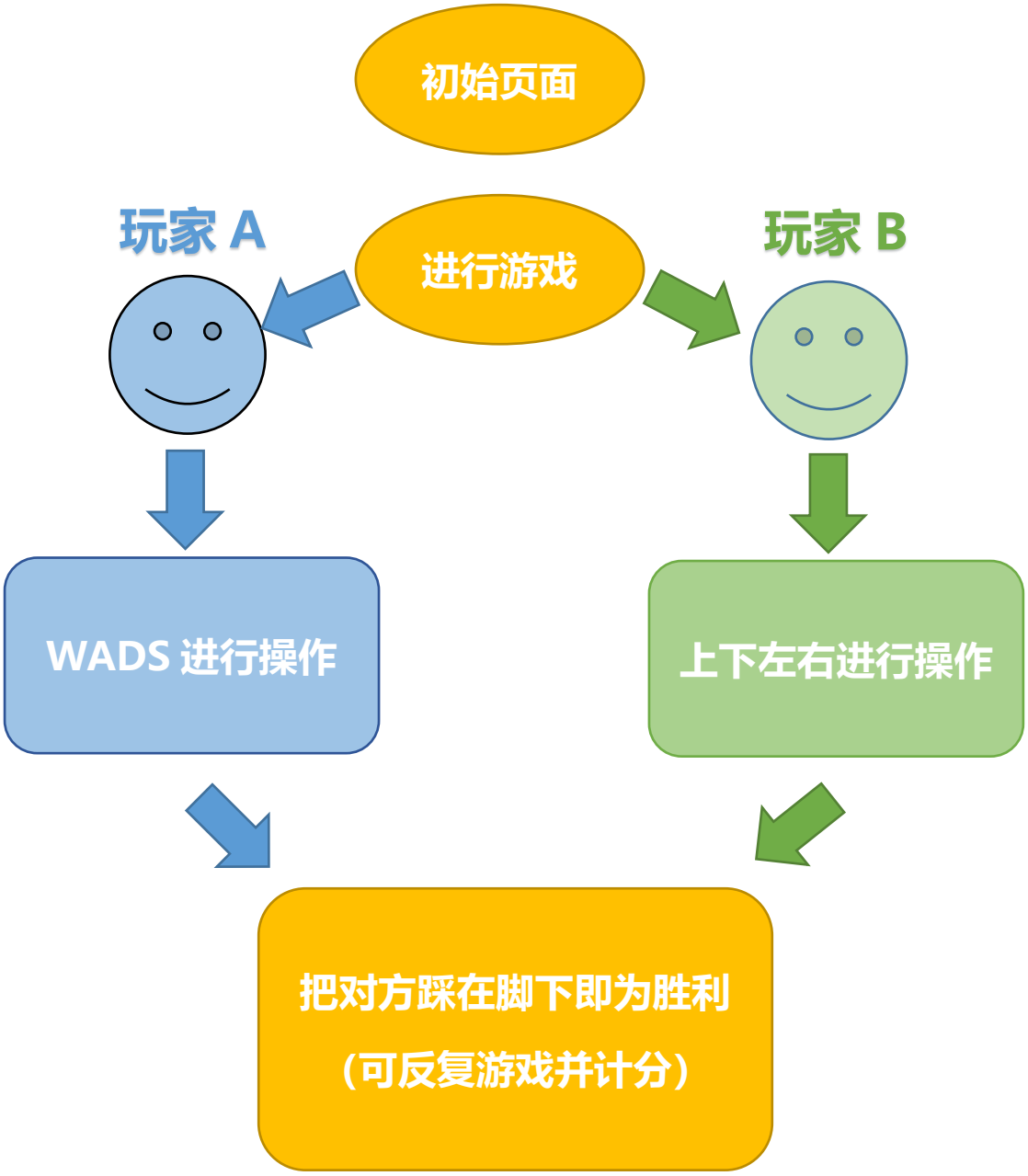
   SWORD 板上的 AA10 是整个程序的总开关。

   SWORD 板上的 AB10 控制欢迎页面的切换以及初始化。

2. **键盘输入：左侧小鸟利用键盘 WADS 控制，右侧小鸟利用上下左右方向键控制。**

3. **游戏规则：**

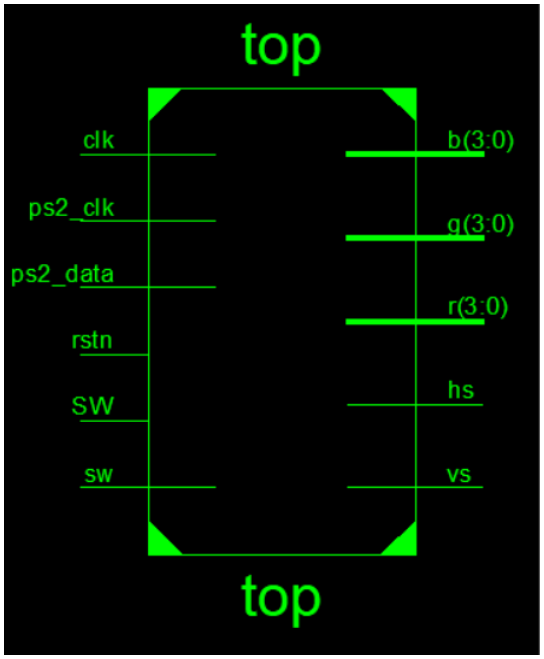
双方玩家控制自己的小鸟，当一只小鸟“踩”在另一只小鸟头上既表示获胜，同时七段码会显示得一分。要注意，游戏操作并没有想象中那么简单，由于本程序的键盘设定没有消除冲突，所以当一个玩家用键盘操作时另一个玩家的操作会打断当前玩家操作，这也就意味着双方玩家为了获得胜利必须不断“点按”键盘，只有手速足够快，并且移动路线很有策略的玩家才会取得最终胜利。玩家在游戏时会产生激烈地键盘敲击声音，这是他们在争夺操作主动权的表现。



第三章 “纸上小鸟”设计实现

3.1 实现过程

3.1.1 top 模块




```
module top(
    input wire clk,     // clock
    input wire rstn,     // reset
    input wire  SW,     // a switch
    input wire  sw,     // another switch
    input wire ps2_clk, //ps2 clock
    input wire ps2_data,
    output  wire  hs,       //synchronization
    output  wire  vs,    //synchronization
    output wire  [3:0] r,
    output wire  [3:0] g,
    output wire  [3:0] b,
    output SEGLED_CLK,  //Seg7Device
    output SEGLED_CLR,//Seg7Device
    output SEGLED_DO,//Seg7Device
```

```
    output SEGLED_PEN//Seg7Device
    );
    wire [9:0] data;
    wire [9:0] col_addr;
    wire [8:0] row_addr;
    wire [9:0] on_hc_1;
    wire [8:0] on_vc_1;
    wire [9:0] on_hc_2;
    wire [8:0] on_vc_2;
    wire rdn;
    wire [3:0]sout;
    reg [31:0]clkdiv;
    wire [11:0] vga_data;
    reg first;

    initial begin first = 1;
        end

    reg [9:0] col_temp_1 ; //remember the col position
    reg [8:0] row_temp_1 ; //remember the row position
    reg [9:0] col_temp_2 ; //remember the col position
    reg [8:0] row_temp_2 ; //remember the row position

    always@(posedge clk) begin
        clkdiv <= clkdiv + 1'b1;
    end

    vgac v0 (
        .vga_clk(clkdiv[1]), .clrn(SW), .d_in(vga_data),

    .row_addr(row_addr), .col_addr(col_addr), .r(r), .g(g), .b(b), .hs(hs),
.vs(vs),
        .rdn(rdn)
    );//use the vgac block

    vgaRGB v1(
    .rgb_clk(clkdiv[1]),.vc(row_addr),.hc(col_addr),.on_vc_1(on_vc_1),.on_h
c_1(on_hc_1),.on_vc_2(on_vc_2),.on_hc_2(on_hc_2),.sw(sw),.vga_data(vga_dat
a),.score(score)
    );//use the vgaRGB block
    ps2_ver2
v2(.clk(clk),.rst(),.ps2_clk(ps2_clk),.ps2_data(ps2_data),.data_out(data),
.ready());//use the ps2_ver2 block
```

```
   Seg7Device
segDevice(.clkIO(clkdiv[3]), .clkScan(clkdiv[15:14]), .clkBlink(clkdiv[25]
),
        .data(score), .point(8'h0), .LES(8'h0),
        .sout(sout));//use the Seg7Device block
    assign SEGLED_CLK = sout[3];
    assign SEGLED_DO = sout[2];
    assign SEGLED_PEN = sout[1];
    assign SEGLED_CLR = sout[0];

    assign on_vc_1 = row_temp_1;//remember the row position
    assign on_hc_1 = col_temp_1;//remember the col position
    assign on_vc_2 = row_temp_2;//remember the row position
    assign on_hc_2 = col_temp_2;//remember the col position


    always @(posedge clkdiv[20]) begin
        if (first == 1) begin
            col_temp_1 = col_temp_1 + 10;
            row_temp_1 = row_temp_1 + 10;
            col_temp_2 = col_temp_2 + 100;
            row_temp_2 = row_temp_2 + 120;
            first = 0;
        end

        if (data[7:0]==8'h1c && data[8] == 0 && col_temp_1 > 7)
            begin
                col_temp_1 = col_temp_1 - 1;
            end
        if (data[7:0]==8'h23 && data[8] == 0 && col_temp_1 < 592) begin
                col_temp_1 = col_temp_1 + 1;
            end
        if (data[7:0]==8'h1d && data[8] == 0 && row_temp_1 > 2)
            begin
                row_temp_1 = row_temp_1 - 1;
            end
        if (data[7:0]==8'h1b && data[8] == 0 && row_temp_1 < 450) begin
                row_temp_1 = row_temp_1 + 1;
            end
        if (data[7:0]==8'h6b && data[8] == 0 && col_temp_2 > 7)
            begin
                col_temp_2 = col_temp_2 - 1;
            end
        if (data[7:0]==8'h74 && data[8] == 0 && col_temp_2 < 592) begin
```

```
                col_temp_2 = col_temp_2 + 1;
            end
        if (data[7:0]==8'h75 && data[8] == 0 && row_temp_2 > 2)
            begin
                row_temp_2 = row_temp_2 - 1;
            end
        if (data[7:0]==8'h72 && data[8] == 0 && row_temp_2 < 450) begin
                row_temp_2 = row_temp_2 + 1;
            end
    end
endmodule
```

Top 模块主要负责一些输入输出的准备，并且定义必要的变量和一些必要的分频工作。其中调用的模块才是重点。

3.1.2 vgac 模块

```
module vgac (vga_clk,clrn,d_in,row_addr,col_addr,rdn,r,g,b,hs,vs); // 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
   // h_count: VGA horizontal counter  (0-799)
   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
   // v_count: VGA vertical counter (0-524)
   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
       b       <=  rdn ? 4'h0 : d_in[3:0]; // 2-bit blue
       g       <=  rdn ? 4'h0 : d_in[7:4]; // 3-bit green
       r       <=  rdn ? 4'h0 : d_in[11:8]; // 3-bit red
    end
endmodule
```

Vgac 模块主要负责行扫描，列扫描等 vga 显示的基本操作，在了解 VGA 显示原理后这部分不难编写。

3.1.3 vgaRGB 模块

```
module vgaRGB(
    input wire rgb_clk,
    input wire [9:0] hc,//horizontal position
    input wire [8:0] vc,//vertical position
```

```
    input wire [9:0] on_hc_1,//left bird
    input wire [8:0] on_vc_1,//left bird
    input wire [9:0] on_hc_2,//right bird
    input wire [8:0] on_vc_2,//right bird
    input wire sw,
    output reg [11:0] vga_data,
    output reg [31:0] score = 0//score
);

    wire [11:0] rom_data_p1;//picture that i draw myself
    wire [11:0] rom_data_p2;//picture that i draw myself
    wire [11:0] rom_data_d1;//picture that i draw myself
    wire [11:0] rom_data_d2;//picture that i draw myself
    wire [11:0] rom_data_b;//picture that i draw myself
    wire [11:0] rom_data_w;//picture that i draw myself
    reg [18:0] true_coord;

    always @(*)

if(!sw)
        begin//begin picture
            true_coord <= vc*640+hc;
            vga_data <= rom_data_w;
            score<=1'b0;
        end
    else
    begin// show background
            true_coord <= vc*640+hc;
            vga_data <= rom_data_b;
        if(on_vc_1 + 50 >= on_vc_2 && on_vc_1 <= on_vc_2 && ((on_hc_1 <=
on_hc_2 + 50 && on_hc_2 <= on_hc_1 )||(on_hc_2 <= on_hc_1 + 50 && on_hc_2 >=
on_hc_1)))
            begin//left bird is higher
                score <= score + 1'b1;
                if(vc>=on_vc_1  &&  vc<on_vc_1+50  &&  hc  >=  on_hc_1  &&  hc
<on_hc_1+50 )
                begin
                true_coord <= (vc-on_vc_1)*50+(hc-on_hc_1);
                vga_data <= rom_data_p1;
                end
                else if(vc>=on_vc_2 && vc<on_vc_2+50 && hc >= on_hc_2 && hc
<on_hc_2+50 )
                begin
                true_coord <= (vc-on_vc_2)*50+(hc-on_hc_2);
```

```
                vga_data <= rom_data_d2;
                end
                else
                begin
                true_coord <= vc*640+hc;
                vga_data <= rom_data_b;
                end
            end

        else if(on_vc_2 + 50 >= on_vc_1 && on_vc_2 <= on_vc_1 && ((on_hc_2
<=   on_hc_1 + 50 && on_hc_1 <= on_hc_2 )||(on_hc_1 <= on_hc_2 + 50 &&
on_hc_1 >= on_hc_2)))
            begin//right bird is higher
                if(vc>=on_vc_1  &&  vc<on_vc_1+50  &&  hc  >=  on_hc_1  &&  hc
<on_hc_1+50 )
                begin
                true_coord <= (vc-on_vc_1)*50+(hc-on_hc_1);
                vga_data <= rom_data_d1;
                end
                else if(vc>=on_vc_2 && vc<on_vc_2+50 && hc >= on_hc_2 && hc
<on_hc_2+50 )
                begin
                true_coord <= (vc-on_vc_2)*50+(hc-on_hc_2);
                vga_data <= rom_data_p2;
                end
                else
                begin
                true_coord <= vc*640+hc;
                vga_data <= rom_data_b;
                end
            end
        else
            begin//draw two live birds
            if(vc>=on_vc_1  &&  vc<on_vc_1+50  &&  hc  >=  on_hc_1  &&  hc
<on_hc_1+50 )
                begin
                true_coord <= (vc-on_vc_1)*50+(hc-on_hc_1);
                vga_data <= rom_data_p1;
                end
            else  if(vc>=on_vc_2  &&  vc<on_vc_2+50  &&  hc  >=  on_hc_2  &&  hc
<on_hc_2+50 )
                begin
                true_coord <= (vc-on_vc_2)*50+(hc-on_hc_2);
                vga_data <= rom_data_p2;
```

```
                end
            else
                begin //draw the background
                true_coord <= vc*640+hc;
                vga_data <= rom_data_b;
                end
            end
    end

        //my pictures
    background m2(.clka(rgb_clk),.addra(true_coord),.douta(rom_data_b));
    leftman m1(.clka(rgb_clk),.addra(true_coord),.douta(rom_data_p1));
    rightman m3(.clka(rgb_clk),.addra(true_coord),.douta(rom_data_p2));
    deadleftman m4(.clka(rgb_clk),.addra(true_coord),.douta(rom_data_d1));
    deadrightman m5(.clka(rgb_clk),.addra(true_coord),.douta(rom_data_d2));
    wellcome m6(.clka(rgb_clk),.addra(true_coord),.douta(rom_data_w));
endmodule
```

在本模块中，主要调用了几个图片的 IP 核，本程序中所有的图片 IP 核都是我自己用纸和笔绘出的。并且我们的游戏主逻辑也在本模块给出。对于每种不同的状态，我们经过判断调用不同的 IP 核，从而实现图片的变换。

3.1.4 ps2_ver2 模块

```
module ps2_ver2(
    input clk,
    input rst,
    input ps2_clk,
    input ps2_data,
    output [9:0] data_out,
    output ready
   );
     reg ps2_clk_falg0, ps2_clk_falg1, ps2_clk_falg2;
     wire negedge_ps2_clk;

     always@(posedge clk or posedge rst)begin
        if(rst)begin
            ps2_clk_falg0<=1'b0;
            ps2_clk_falg1<=1'b0;
            ps2_clk_falg2<=1'b0;
        end
        else begin
            ps2_clk_falg0<=ps2_clk;
            ps2_clk_falg1<=ps2_clk_falg0;
            ps2_clk_falg2<=ps2_clk_falg1;
        end
```

```
end

assign negedge_ps2_clk=!ps2_clk_falg1&ps2_clk_falg2;
reg[3:0]num;
always@(posedge clk or posedge rst)begin
    if(rst)
    num<=4'd0;
    else if (num==4'd11)
        num<=4'd0;
    else if (negedge_ps2_clk)
        num<=num+1'b1;
end
reg negedge_ps2_clk_shift;
always@(posedge clk)begin
    negedge_ps2_clk_shift<=negedge_ps2_clk;
end
reg[7:0]temp_data;
always@(posedge clk or posedge rst)begin
    if(rst)
        temp_data<=8'd0;
    else if (negedge_ps2_clk_shift)begin
        case(num)
            4'd2 : temp_data[0]<=ps2_data;
            4'd3 : temp_data[1]<=ps2_data;
            4'd4 : temp_data[2]<=ps2_data;
            4'd5 : temp_data[3]<=ps2_data;
            4'd6 : temp_data[4]<=ps2_data;
            4'd7 : temp_data[5]<=ps2_data;
            4'd8 : temp_data[6]<=ps2_data;
            4'd9 : temp_data[7]<=ps2_data;
            default: temp_data<=temp_data;
        endcase
    end
    else temp_data<=temp_data;
end
reg [9:0] data;
reg data_break, data_done, data_expand;
always@(posedge clk or posedge rst)begin
    if(rst)begin
        data_break<=1'b0;
        data<=10'd0;
        data_done<=1'b0;
        data_expand<=1'b0;
    end
```

```
        else if(num==4'd11)begin
            if(temp_data==8'hE0)begin
                data_expand<=1'b1;

            end
            else if(temp_data==8'hF0)begin
                data_break<=1'b1;

            end
            else begin
                data<={data_expand,data_break,temp_data};
                data_done<=1'b1;
                data_expand<=1'b0;
                data_break<=1'b0;
            end
        end
        else begin
            data<=data;
            data_done<=1'b0;
            data_expand<=data_expand;
            data_break<=data_break;
        end
    end

    assign data_out=data;
    assign ready=data_done;

endmodule
```

这一部分负责键盘部分输入。

3.1.5 七段码的显示部分

```
module Seg7Device(
    input clkIO, input [1:0] clkScan, input clkBlink,
    input [31:0] data, input [7:0] point, input [7:0] LES,
    output [3:0] sout, output reg [7:0] segment, output reg [3:0] anode
);
    wire [63:0] dispData;
    wire [31:0] dispPattern;

    Seg7Decode U0(.hex(data), .point(point),
        .LE(LES & {8{clkBlink}}), .pattern(dispData));
    Seg7Remap U1(.I(data), .O(dispPattern));
```

```
    ShiftReg #(.WIDTH(64)) U2(.clk(clkIO), .pdata(dispData), .sout(sout));

    always @*
        case(clkScan)
        2'b00: begin segment <= ~dispPattern[ 7: 0]; anode <= 4'b1110; end
        2'b01: begin segment <= ~dispPattern[15: 8]; anode <= 4'b1101; end
        2'b10: begin segment <= ~dispPattern[23:16]; anode <= 4'b1011; end
        2'b11: begin segment <= ~dispPattern[31:24]; anode <= 4'b0111; end
        endcase


endmodule
```

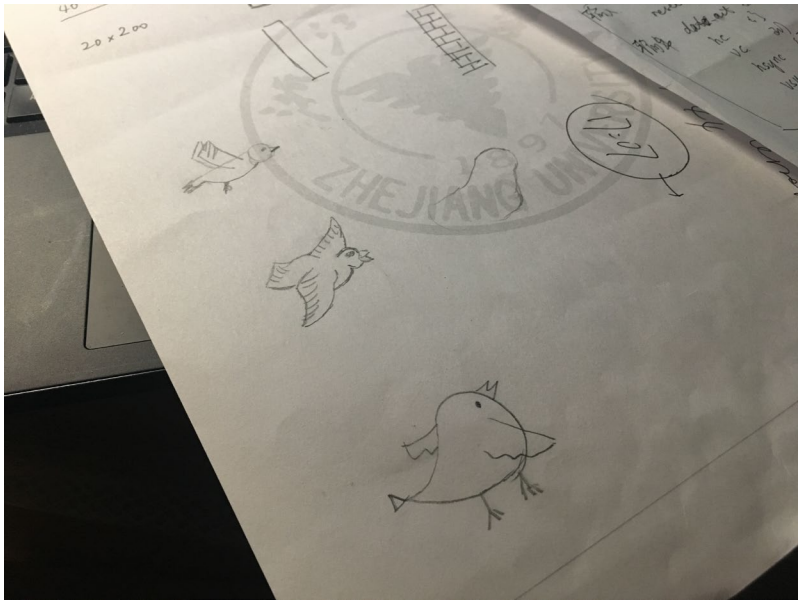
```
module Seg7Decode(
    input [31:0] hex, input [7:0] point, input [7:0] LE,
    output [63:0] pattern
);
    wire [63:0] digits;

    SegmentDecoder
        U0(.hex(hex[ 3: 0]), .segment(digits[ 6: 0])),
        U1(.hex(hex[ 7: 4]), .segment(digits[14: 8])),
        U2(.hex(hex[11: 8]), .segment(digits[22:16])),
        U3(.hex(hex[15:12]), .segment(digits[30:24])),
        U4(.hex(hex[19:16]), .segment(digits[38:32])),
        U5(.hex(hex[23:20]), .segment(digits[46:40])),
        U6(.hex(hex[27:24]), .segment(digits[54:48])),
        U7(.hex(hex[31:28]), .segment(digits[62:56]));

    assign {digits[63], digits[55], digits[47], digits[39], digits[31],
digits[23], digits[15], digits[7]} = ~point;

    assign pattern[ 7: 0] = digits[ 7: 0] | {8{LE[0]}};
    assign pattern[15: 8] = digits[15: 8] | {8{LE[1]}};
    assign pattern[23:16] = digits[23:16] | {8{LE[2]}};
    assign pattern[31:24] = digits[31:24] | {8{LE[3]}};
    assign pattern[39:32] = digits[39:32] | {8{LE[4]}};
    assign pattern[47:40] = digits[47:40] | {8{LE[5]}};
    assign pattern[55:48] = digits[55:48] | {8{LE[6]}};
    assign pattern[63:56] = digits[63:56] | {8{LE[7]}};

endmodule
```

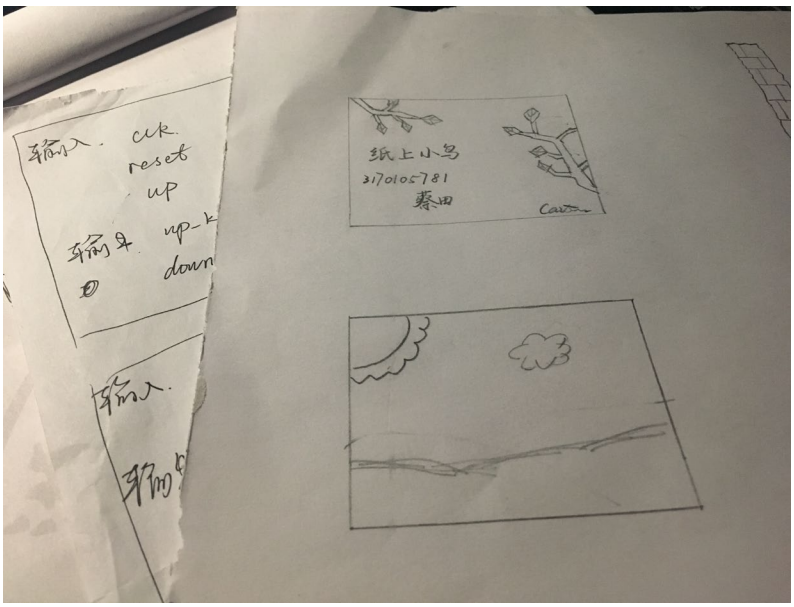
以上两个部分为七段码的显示，代码依据为老师提供的 demo 小样。

3.1.6 引脚约束部分

```
    NET "clk"                   LOC = AC18      | IOSTANDARD = LVCMOS18 ;
    NET "rstn"                  LOC = W13       | IOSTANDARD = LVCMOS18 ;
    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 "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 "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 "hs"                    LOC = M22       | IOSTANDARD = LVCMOS33 | SLEW
= FAST ;
    NET "vs"                    LOC = M21       | IOSTANDARD = LVCMOS33 | SLEW
= FAST ;

    NET "SEGLED_CLR"            LOC = M20       | IOSTANDARD = LVCMOS33 | SLEW
= FAST ;
    NET "SEGLED_DO"             LOC = L24       | IOSTANDARD = LVCMOS33 | SLEW
= FAST ;
    NET "SEGLED_PEN"            LOC = R18       | IOSTANDARD = LVCMOS33 | SLEW
= FAST ;
    NET "clk"                   LOC = AC18      | IOSTANDARD = LVCMOS18 ;
    NET "rstn"                  LOC = W13       | IOSTANDARD = LVCMOS18 ;
```

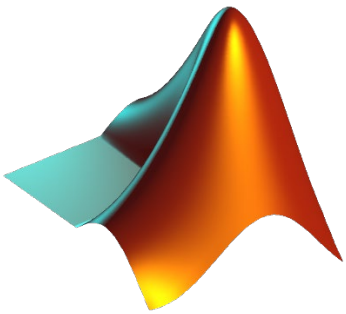
3.1.6 图像部分





本游戏内所有显示内容皆为手绘完成

在纸上绘画完成后利用手机拍照并传至电脑，下面我们需要把 jpg 格式图片转化成 coe 文件，并且是 3 位 16 进制 coe 文件，于是我选择利用 Matlab 编写脚本。

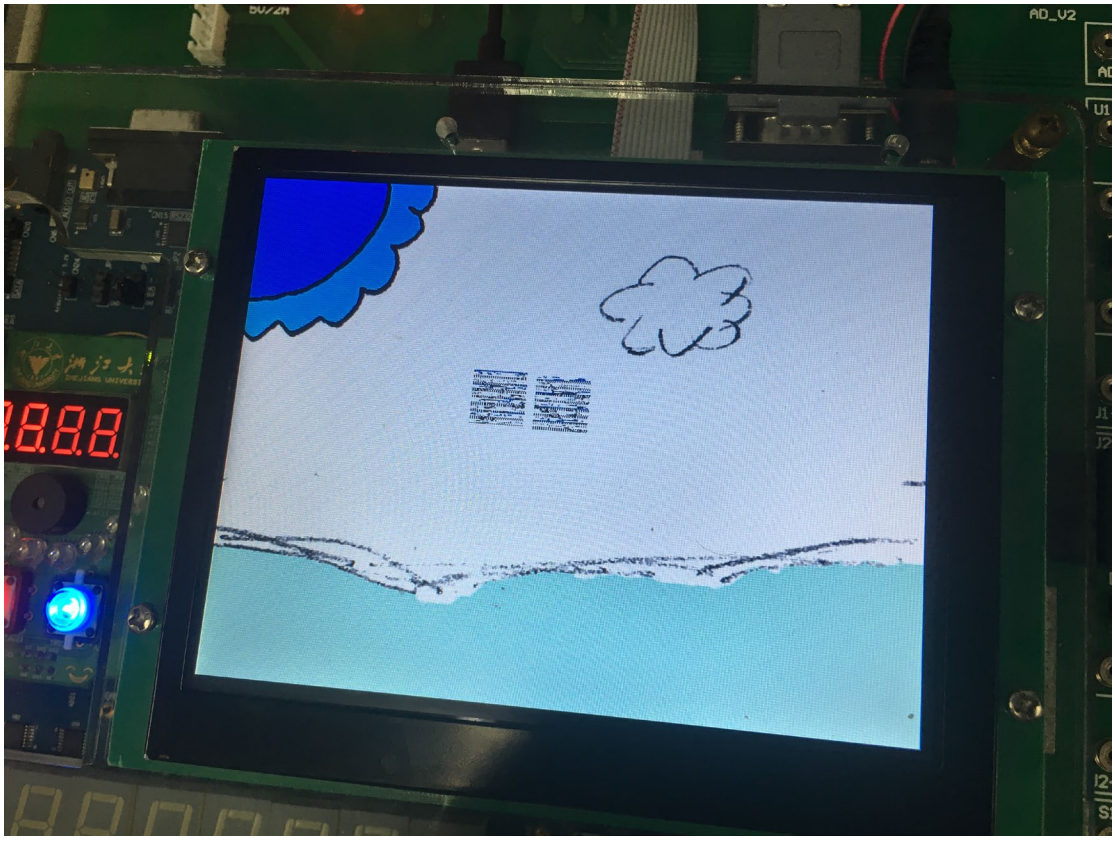


```
function res = RGB888toRGB444(imgdata, filename)
    cnt = 3;
    [h,l,c] = size(imgdata);
    res(1:h,1:l,1:c)=0;
    file = fopen(filename,"w");
    fprintf(file,"MEMORY_INITIALIZATION_RADIX=16;\nMEMORY_INITIALIZATION_VECTOR=\n");
    res(1,1,1) = floor((imgdata(1,1,1)+1) / 16);
    if(res(1,1,1)==16)res(1,1,1) = res(1,1,1)-1;end
    res(1,1,2) = floor((imgdata(1,1,2)+1) / 16);
    if(res(1,1,2)==16)res(1,1,2) = res(1,1,2)-1;end
    res(1,1,3) = floor((imgdata(1,1,3)+1) / 16);
    if(res(1,1,3)==16)res(1,1,3) = res(1,1,3)-1;end

fprintf(file,"%c%c%c",dec2hex(res(1,1,1),1),dec2hex(res(1,1,2),1),dec2hex(res(1,1,3),
1));
    for i = 1: h
        for j = 1 : l
            if(i~=1 || j~= 1)
                for k = 1 : c
                    res(i,j,k) = floor((imgdata(i,j,k)+1) / 16);
                    if(res(i,j,k)==16)res(i,j,k) = res(i,j,k)-1;end
                    cnt = cnt + 1;
                end

fprintf(file,",\n%c%c%c",dec2hex(res(i,j,1),1),dec2hex(res(i,j,2),1),dec2hex(res(i,j,
3),1));
            end
        end
    end

    imshow(res);
%     for i = 1 : h
%         for j = 2 : l
%             fprintf(file,",\n%c%c%c",resH(i,j,1),resH(i,j,2),resH(i,j,3));
%         end
%     end
    fprintf(file,";\n");
    fprintf(file,"%d",cnt);
    fclose(file);
```

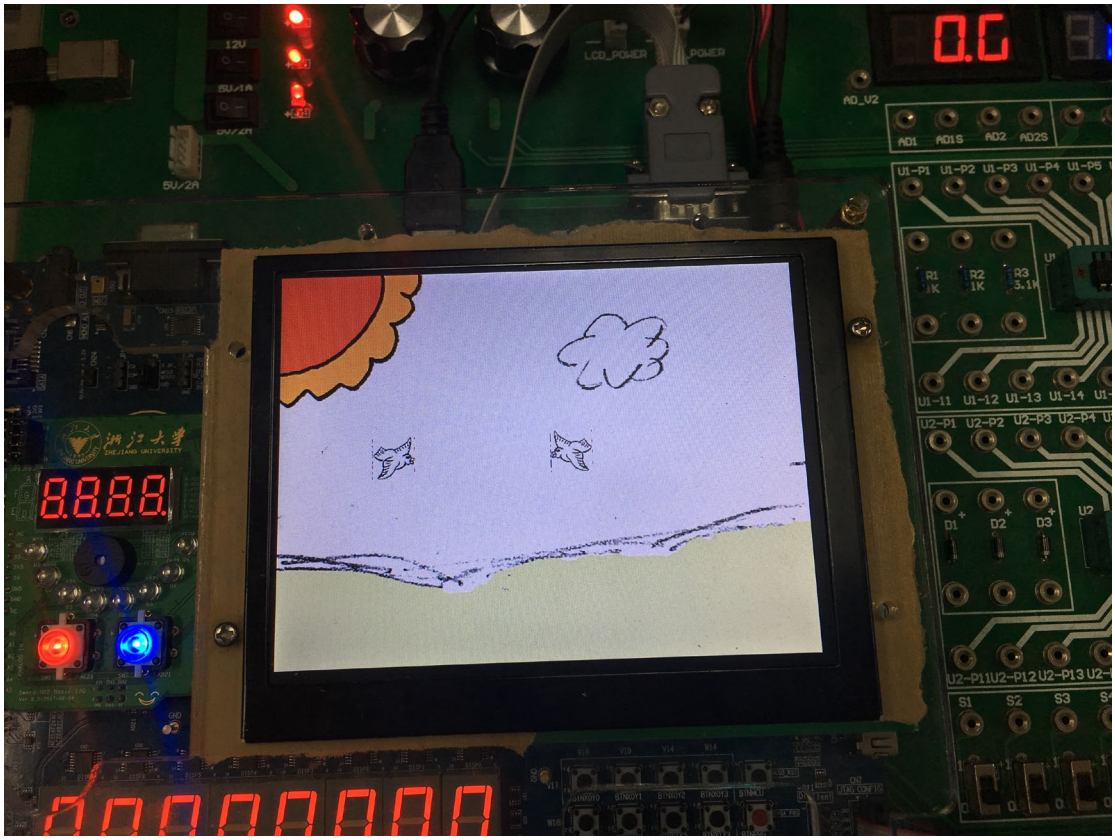
3.2 调试过程

在调试过程中先后出现了很多种问题，问题主要集中在 VGA 显示上，最开始 VGA 是这样显示的：

不断研究发现，是我生成 IP 核的时候宽度设置有误，而且 coe 文件生成的是 6 位 16 进制 coe，所以我根据实际情况编写了 Matlab 脚本，使得图像能够正常显示。但是同时又出现了新的问题：

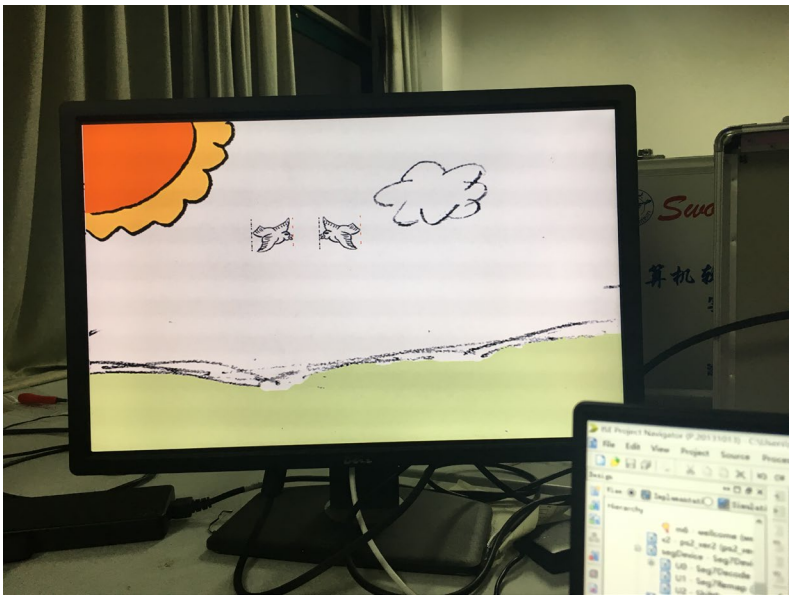


其实相较于最开始的那种根本显示不出来正常图像的问题，这种仅仅是颜色上的差异要好多了，经过观察发现，该图片仅仅是红色和蓝色反了，即 r 和 b 颠倒了，所以我在程序中将 r 和 b 的位置互换，这样就得到了正常结果。



第四章 结果分析与用户反馈

4.1 结果分析



初始页面和游戏画面都能正常显示，初始页面出现后拨动 AB10 进入游戏界面，并且经过验证，操作正常。两只小鸟运动情况正常，游戏规则也能生效，小鸟死亡时也会有相应显示，并且七段码会进行计分。

键盘的输入如我们所愿，玩家 A 的输入会受到玩家 B 输入的打断，使得双方必须不停“点按”来达到争抢主动权的目的。完全符合我对游戏设计的初衷，说明程序成功。

4.2 用户体验与反馈

程序制作完毕后，我邀请多名同学进行游戏，因为游戏规则较为简单所以大家很容易接受，最开始大家以为游戏很容易，只需要简单的控制方向就可以，但是实际上并不简单，因为键盘的冲突设定，导致一个玩家不可能如愿移动，另一个玩家的操作势必会影响当前的操作。在游戏过程中，双方玩家为了获得胜利必须不断“点按”键盘，只有手速足够快，并且移动路线很有策略的玩家才会取得最终胜利。玩家在游戏时会产生激烈地键盘敲击声音，这是他们在争夺操作主动权的表现。

在进行游戏时，双方玩家都拼命敲击键盘，投入其中而且全神贯注，充满竞技性。

体验过游戏之后，玩家们纷纷表示这款游戏画面制作极有创意，并且游戏规则简单易于理解，同时键盘的冲突是游戏的亮点，因为只有冲突的键盘才会让双方都拼命点按键盘，否则这款游戏就失去了自己的灵魂。

在提供的视频中也有用户体验部分。体验结束他们也对这款游戏进行了评价。相比于传统的贪吃蛇或是打地鼠等游戏，同学们认为这款游戏操作简单但是思路独特，另辟蹊径，能在繁重的课业负担下给人以简单的快乐，这一点从同学们试玩时的笑声中也可以得到验证。

同学们还表示这款游戏具有增强玩家间友谊的功效。

第五章 总结与致谢

5.1 总结

作为一个完全一个人制作，完全原创的小游戏（甚至为了原创，图片都是我自己画的），在制作过程中，虽然游戏看似简单但还是遇到不少困难。游戏原理逻辑的实现其实并没有出现太多问题，问题最多还是出现在 VGA 显示以及键盘输入上面，但是经过不断地学习和不断地调试，这些问题都得以较好地解决。

在解决问题的过程中，锻炼了我的思考和动手能力，比如当图片不能很好地从 jpg 转化为 3 位 16 进制的 coe 文件时，我通过编写 Matlab 脚本解决了问题。VGA 的显示也在不断地实践过程中最终成功显示。

值得一提的是，整个大程序感觉时间消耗最大的环节其一是 IP 核的生成，其二是整个工程最后的 generate 的过程。一张背景的 IP 核从开始生成到生成完毕大概需要 1 小时左右时间，当然这个时间也和电脑配置有关。Generate 的耗时虽然比不上 IP 核的生成，但是也是极为漫长的，并且你可能会在 generate 的等待过程中发现新的 bug，往往修改完 bug 后又需要重新 generate。

总之，整个大程序制作下来收获还是很多的，只是因为没有队友所以做起来比较艰难，不过切切实实学到了很多东西，当然，我的游戏还存在着一些需要完善的地方，而且由于原创缘故，可能有些游戏规则还不够完善，后续如果有了更好的游戏规则的灵感也会再次补充

5.2致谢

最后由衷感谢在我完成大程序过程中，

在我提问时极有耐心解答的同学们，谢谢大家帮助！

感谢那些在 512 陪我一起通宵奋斗的小伙伴们！

感谢施老师理论课上传授的知识以及人生道理，是啊，数逻课上学到真本事才是硬道理，而通过本次数逻大作业，我感觉我个人确实学会了好多。

感谢实验室洪老师的关怀和指导。

也感谢在群里为大家答疑解惑的助教！