

数字逻辑设计 · 课程设计报告

基于数字系统的恐龙跑酷游戏

Dinosaur Game Based on Digital System



	组长	组员
姓 名	NoughtQ	gxgg
学 号		
学 院	计算机科学与技术学院	
专 业	计算机科学与技术	
指导教师	洪奇军	
报告日期	2024 年 6 月 13 日	

目录

1 游戏设计说明	3
1.1 设计背景	3
1.2 玩法介绍	3
2 设计原理	4
2.1 硬件描述语言	4
2.2 有限状态机	4
2.3 VGA 显示	5
2.4 PS2 键盘输入	7
3 设计过程	8
3.1 整体框架	8
3.2 模块详解	11
3.2.1 顶层模块	11
3.2.2 IP 核的生成	17
3.2.3 VGA 显示	21
3.2.4 PS2 键盘输入	26
3.2.5 七段数码管显示	28
3.3 仿真测试：PS2 键盘物理仿真	35
4 测试及分析	38
5 展望和心得	41
6 小组成员及分工	42

1 游戏设计说明

1.1 设计背景

《恐龙游戏》（英语：Dinosaur Game）是一款内置于 Google Chrome 的横向滚动栏形式网页游戏。当用户在离线时尝试浏览网站，浏览器会提示并未连接到互联网，并在错误页面上显示一只像素化形象的小暴龙。用户可以通过按键 SPACE 或 ↑ 开始游戏，随后玩家将操控一只像素风格的小暴龙，它会在黑白色调的沙漠场景中持续从左往右移动，玩家需要让它避开障碍物并获取分数，直至小恐龙撞到障碍物使游戏结束。恐龙移动速度会随着时间推进而逐渐加快，虽然操作简单，但速度的不断提升使游戏兼具趣味性和挑战性。

1.2 玩法介绍

在本次课程设计中，小组成员利用 SOWRD 板等硬件设备，尽可能地还原《恐龙游戏》的基本玩法；但也由于硬件设备的限制，对游戏进行了一定的简化。

本游戏主要用外接 PS2 键盘进行控制：

- 按下键盘任意键**进入游戏**
- 按下↑键使小恐龙**跳起**
- 按下↓键使小恐龙**下蹲**

小组成员通过让背景和障碍物的从右往左的运动，来形成恐龙从左往右运动的效果。当恐龙尚未撞到障碍物（包括仙人掌、翼龙等）时，分数会随时间流逝不断积累，同时画面的移动速度会逐渐提升。而当恐龙撞到障碍物时，屏幕显示游戏结束界面，同时 SWORD 板上的七段数码管显示本次游戏的分数以及最高记录。玩家可以通过点击 SWORD 板阵列键盘的复位键重新开始下一轮游戏。

2 设计原理

2.1 硬件描述语言

硬件描述语言 (hardware description language, HDL) 是用来描述电子电路 (特别是数字电路) 功能、行为的语言, 可以在寄存器传输级、行为级、逻辑门级等对数字电路系统进行描述。随着自动化逻辑综合工具的发展, 硬件描述语言可以被这些工具识别, 并自动转换到逻辑门级网表, 使得硬件描述语言可以用来进行电路系统设计, 并能通过逻辑仿真的形式验证电路功能。设计完成后, 可以使用逻辑综合工具生成低抽象级别 (门级) 的网表 (即连线表)。

硬件描述语言在很多地方可能和传统的软件编程语言类似, 但是最大的区别是, 硬件描述语言能够描述硬件电路的时序特性。硬件描述语言是电子设计自动化体系的重要部分。小到简单的触发器, 大到复杂的超大规模集成电路 (如微处理器), 都可以用硬件描述语言描述。本课程设计主要采用 **Verilog** 语言编写。

2.2 有限状态机

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

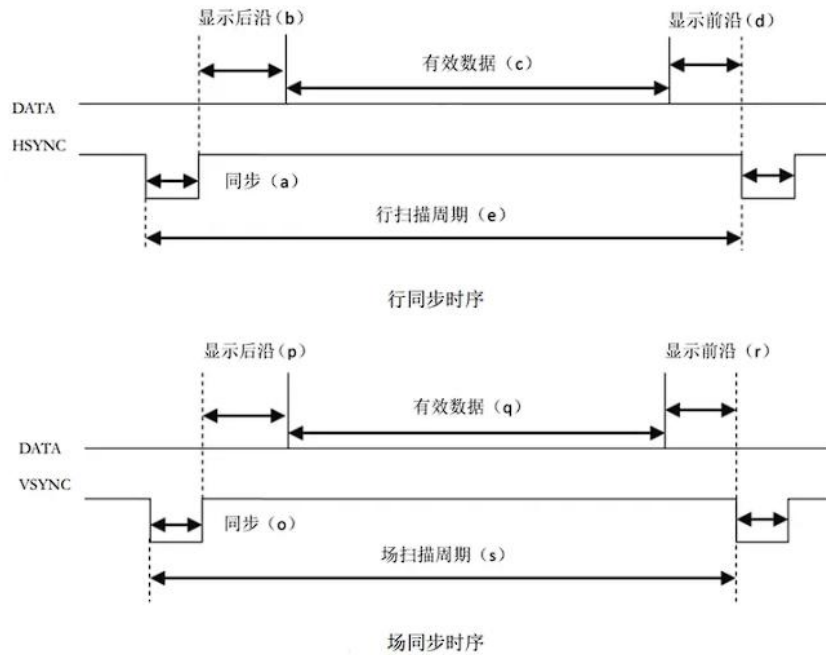
态，组合逻辑用于状态译码和产生输出信号。根据输出信号产生方法的不同，状态机可分为两类：**米里型(Mealy)**和**摩尔型(Moore)**。摩尔型状态机的输出只是当前状态的函数。米里型状态机的输出是在输入变化后立即变化的，不依赖时钟信号的同步，摩尔型状态机的输入发生变化时还需要等待时钟的到来，必须在状态发生变化时才会导致输出的变化，因此比米里型状态机要多等待一个时钟周期。

2.3 VGA 显示

VGA(Video Graphics Array) 协议是一种使用模拟信号的显示标准，我们需要提供的是数字信号（如比较重要的 RGB 三色值、扫描同步信号等），板内 DAC(Digital-to-Analog Converter, DAC/D2C) 会将其转换为 VGA 接口需要的模拟信号。

影响画面质量的因素较主要的有分辨率、刷新率以及色彩。**分辨率**指屏幕中显示的有效像素点数量，一般以 $aaa \times bbb$ 表示，前者指一行中像素点个数，后者指一列中像素点个数。**刷新率**指屏幕内容刷新速度，一般以 Hz 为单位表示一秒钟刷新多少次。**色彩**主要指色彩空间格式与“精度”，VGA 要求使用 RGB 色彩空间（即红绿蓝三色混合）且为 12 位，即一个色彩通道用 4 位表示。在我们的实验中，使用 $640 \times 480 @ 60\text{Hz}$ 的显示模式，需要接入的时钟频率为 25.175MHz。

VGA 使用**逐行扫描**的方式来打印界面，即每次图片刷新都从左上角开始，先从左到右扫描完一行，再转到下一行的最左边开始扫描，直到扫描完最后一行。我们需要处理**行时序与场时序**，时序图如下：



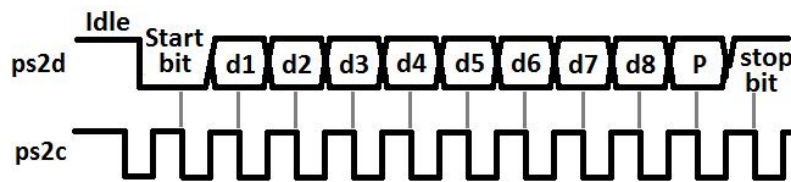
首先提供数据，对于 640×480@60Hz 的显示模式，上图中行扫描的 a, b, c, d, e 分别为 96, 48, 640, 16, 800，场扫描的 o, p, q, r, s 分别为 2, 33, 480, 10, 525。下面以行时序为例进行解释：行同步阶段(96)将行同步信号置于**低位**进行同步；显示后沿(48)将行同步信号重新拉起到**高位**但并不显示图像(将 RGB 三色通道均置为 0 即可)；有效数据(640)为每行 640 个像素点，此时将像素点对应的色彩 RGB 值放置在相应通道上进行色彩输出；显示前沿(16)不显示图像；之后将进入下一个周期的行扫描（从行同步开始）此时已经开始对下一行进行扫描，一次扫描经过的像素点数量(800)，但实际打印的有效数据数量(640)。场同步扫描与行同步扫描相似，不再赘述。

对于我们而言，只需要知道同步信号拉低与拉高的时机以及打印有效数据的时机即可（行坐标在 $[a+b, a+b+c)$ 且列坐标在 $[o+p, o+p+q)$ 时打印有效的数据）。

2.3 PS2 键盘输入

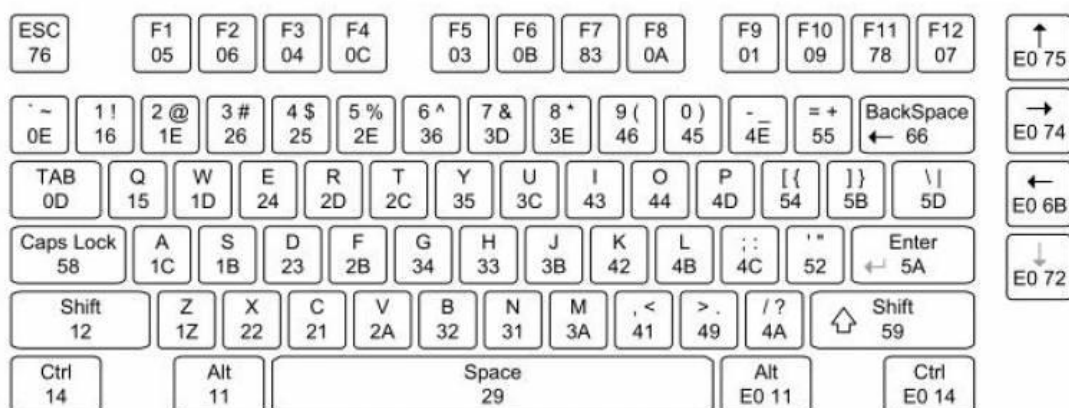
PS/2 接口(Personal System/2)是一种 PC 电脑上的接口，可用来连接键盘和鼠标。接口共有 6 个接脚，除接地与 Vcc 外，有**时钟和一位数据**（另外两脚为保留未使用）。Sword 板内有 USB-PS2 转换，因此我们可以将 USB 键盘或鼠标插入板上 USB 口来使用。

PS/2 协议中，一次传输有效数据为一字节，每次传输（一帧）为 11 位，分别为开始位（1 位，一直为 0）、有效数据（8 位）、校验位（1 位）、结束位（1 位，一直为 1），在我们简单的设计里，每一帧数据中只需要关注中间的 8 位有效数据即可。



PS/2 将键盘编码分为通码(Make)与断码(Break)，通码代表“按下”，断码代表“松开”。键盘上大部分按键的通码只有一字节（比如 WASD 等字母按键），但也有特殊按键的通码为两字节（比如上下左右，其格式为 E0 开头的两字节数据）。断码是在通码的基础上添加一字节的 F0 数据，比如 W 的通码为 1D 断码为 F0 1D，上 ↑ 的通码为 E0 75，断码为 E0 F0 75。由此我们可知，通码可能需要 1~2 帧，断码可能需要 2~3 帧（如果传输内容超过 1 帧，键盘可以保证传输内容是连续的，不会被其他信号隔开）。

按键-通码/断码对照表如下：

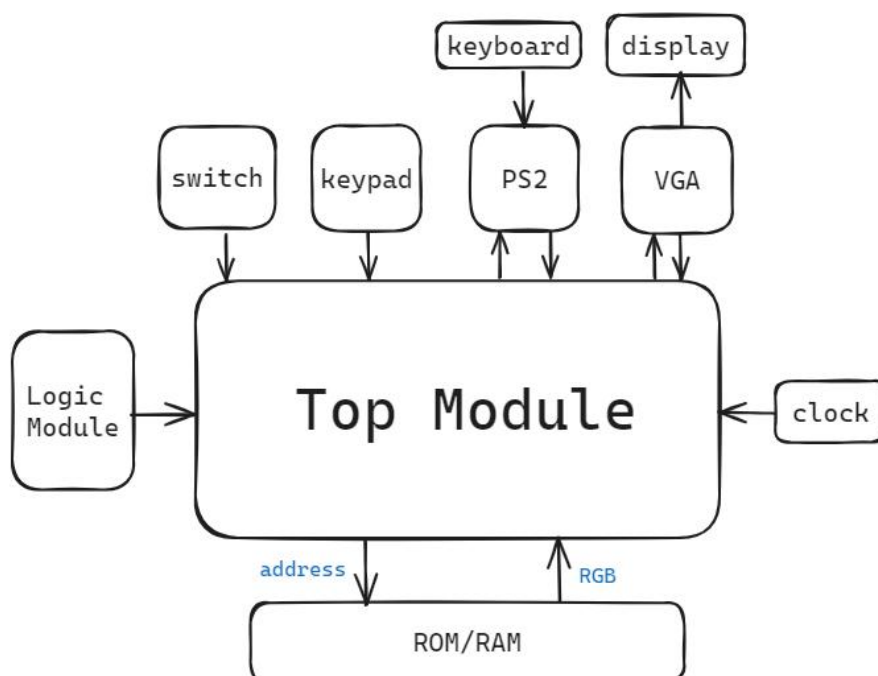


3 设计过程












3.1 整体框架

下面将从模块结构和流程图 2 个角度介绍本设计的整体框架，通过这些框架对整个设计有一个全面的认识。

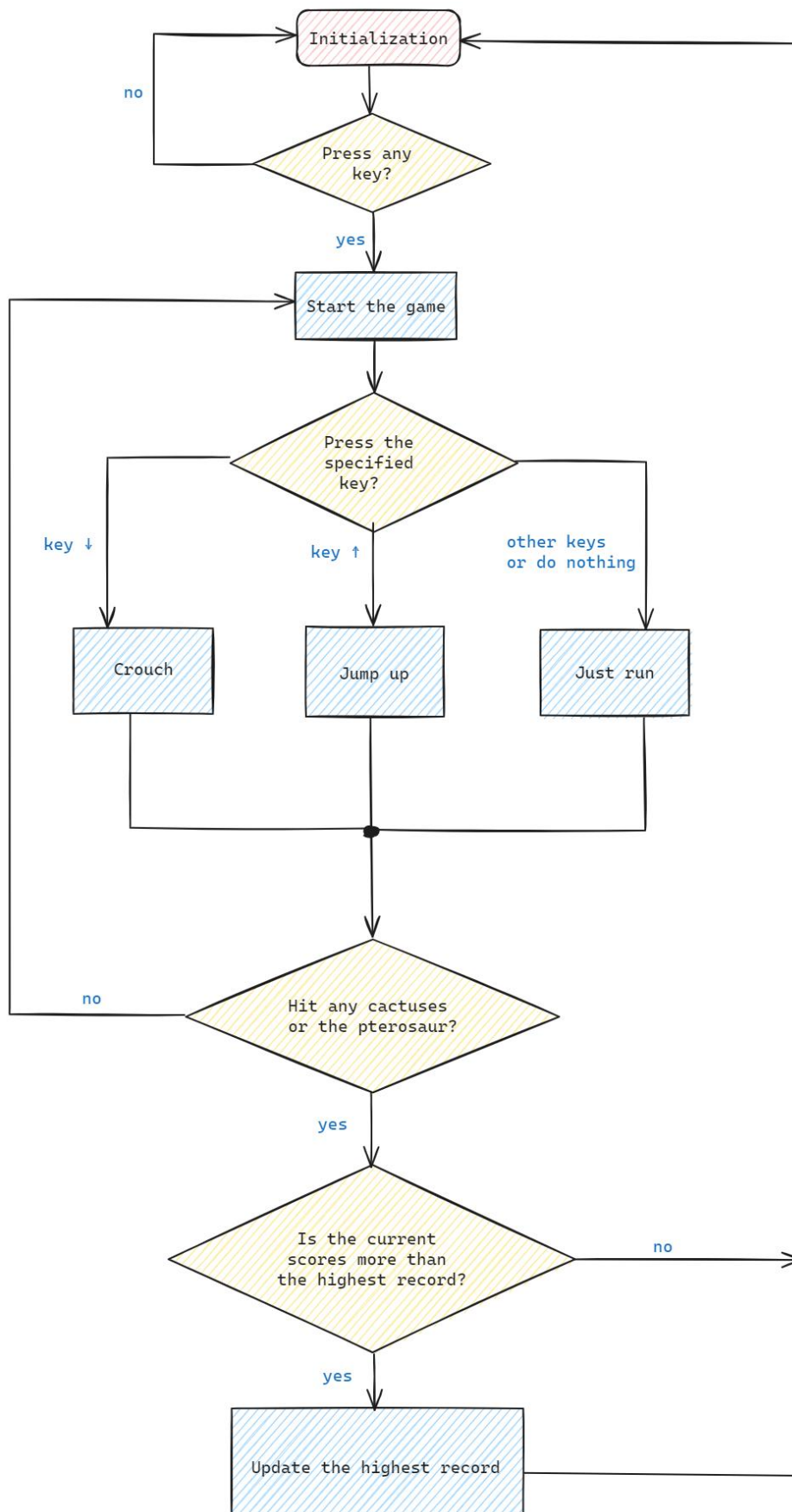
- 本设计的**模块整体结构**如下所示：



对应 Vivado 中的文件目录结构如下所示：

- ▼ ●  **Top** (Top.v) (30)
 - k1 : ps2 (ps2.v)
 - ▼ ● segDevice : Seg7Device (Seg7Device.v) (3)
 - ▼ ● U0 : Seg7Decode (Seg7Decode.v) (8)
 - U0 : SegmentDecoder (SegmentDecoder.v)
 - U1 : SegmentDecoder (SegmentDecoder.v)
 - U2 : SegmentDecoder (SegmentDecoder.v)
 - U3 : SegmentDecoder (SegmentDecoder.v)
 - U4 : SegmentDecoder (SegmentDecoder.v)
 - U5 : SegmentDecoder (SegmentDecoder.v)
 - U6 : SegmentDecoder (SegmentDecoder.v)
 - U7 : SegmentDecoder (SegmentDecoder.v)
 - U1 : Seg7Remap (Seg7Remap.v)
 - U2 : ShiftReg (ShiftReg.v)
 - v0 : vgac (vgac.v)
 - >  s1 : start_cover (start_cover.xci)
 - >  b1 : background (background.xci)
 - >  f1 : finish_cover (finish_cover.xci)
 - >  d1 : dinosaur (dinosaur.xci)
 - >  d2 : dinosaur_crouch (dinosaur_crouch.xci)
 - >  d3 : dinosaur1 (dinosaur1.xci)
 - >  d4 : dinosaur2 (dinosaur2.xci)
 - >  c11 : cactus1 (cactus1.xci)
 - >  c12 : cactus1 (cactus1.xci)
 - >  p1 : pterosaur (pterosaur.xci)
 - ledDevice : ShiftReg (ShiftReg.v)
 - a0[15] : AntiJitter (AntiJitter.v)
 - a0[14] : AntiJitter (AntiJitter.v)
 - a0[13] : AntiJitter (AntiJitter.v)
 - a0[12] : AntiJitter (AntiJitter.v)
 - a0[11] : AntiJitter (AntiJitter.v)
 - a0[10] : AntiJitter (AntiJitter.v)
 - a0[9] : AntiJitter (AntiJitter.v)
 - a0[8] : AntiJitter (AntiJitter.v)
 - a0[7] : AntiJitter (AntiJitter.v)
 - a0[6] : AntiJitter (AntiJitter.v)
 - a0[5] : AntiJitter (AntiJitter.v)
 - a0[4] : AntiJitter (AntiJitter.v)
 - a0[3] : AntiJitter (AntiJitter.v)
 - a0[2] : AntiJitter (AntiJitter.v)
 - a0[1] : AntiJitter (AntiJitter.v)
 - a0[0] : AntiJitter (AntiJitter.v)

- 游戏的流程图如下所示:



3.2 模块详解

3.2.1 顶层模块

输入输出信号

```
module Top(  
    input clk,           // 时钟输入  
    input ps2_clk,       // PS2 时钟输入  
    input ps2_data,      // PS2 数据输入  
    input rstn,          // 复位输入  
    input [15:0] SW,      // 开关，实际上只用到 SW[0]，用来控制屏幕亮灭  
    output hs,           // 水平同步信号输出  
    output vs,           // 垂直同步信号输出  
    output [3:0] r,       // 4 位宽红色信号输出  
    output [3:0] g,       // 4 位宽绿色信号输出  
    output [3:0] b,       // 4 位宽蓝色信号输出  
    output SEGLED_CLK,    // 分段时钟信号输出  
    output SEGLED_CLR,    // 分段清零信号输出  
    output SEGLED_D0,     // 分段输出信号输出  
    output SEGLED_PEN,    // 分段笔选信号输出  
    output LED_CLK,       // 下面 4 个参数均用于七段数码管显示输出  
    output LED_CLR,  
    output LED_D0,  
    output LED_PEN  
);
```

中间变量定义

```
// 实际的游戏坐标  
reg [8:0] dino_y = 7'd120;           // 恐龙纵坐标  
  
reg [9:0] cactus1_1_x;               // 仙人掌横坐标  
reg [9:0] cactus1_2_x;  
reg [9:0] cactus1_3_x;  
reg [8:0] cactus1_1_y = 9'd320;      // 仙人掌纵坐标  
reg [8:0] cactus1_2_y = 9'd320;  
reg [8:0] cactus1_3_y = 9'd320;  
  
reg [9:0] ptero_x;                   // 翼龙横坐标
```

```

reg [9:0] ptero_height = 9'd160; // 翼龙纵坐标

reg [9:0] ground_height = 9'd120; // 地面高度

// 其他中间变量
reg start = 0; // 开始界面
reg finish = 0; // 结束界面

reg [7:0] width = 6'd60; // 恐龙的宽度
reg [7:0] height = 6'd60; // 恐龙的高度
reg is_crouch = 0; // 恐龙是否下蹲
reg [5:0] leap; // 恐龙跳起高度
reg airup = 0, airdown = 0; // 判断在空中上升还是下落
reg flash = 0; // 用于切换移动恐龙的图片，实现恐龙跑动的效果

reg [4:0] width_c = 5'd20; // 仙人掌的宽度
reg [5:0] height_c = 6'd40; // 仙人掌的高度
reg start_c1[2:0]; // 仙人掌是否开始移动

reg ptero_used; // 翼龙是否开始移动

reg [3:0] speed; // 移动速度

```

所用模块（主要是一些外接设备）

```

// 防抖动模块
wire [15:0] SW_OK;
AntiJitter #(4) a0[15:0](.clk(clkdiv[15]), .I(SW), .O(SW_OK));

// PS2 键盘输入
wire up, down; // 只用到键盘的 '↑' 和 '↓'
wire ready;
ps2 k1
(.clk(clk), .rst(0), .ps2_clk(ps2_clk), .ps2_data(ps2_data), .up(up)
, .down(down), .ready(ready));

// 七段数码管输出，用于显示最高记录和实时得分
reg [15:0] score = 32'b0; // 实时得分
reg [15:0] record = 32'b0; // 最高记录
wire [3:0] sout;
Seg7Device

```

```

segDevice(.clkIO(clkdiv[3]), .clkScan(clkdiv[15:14]), .clkBlink(clkdiv[25]),
    .data({record,
score}), .point(8'h0), .LES(8'h0), .sout(sout));
    assign SEGLED_CLK = sout[3];
    assign SEGLED_D0 = sout[2];
    assign SEGLED_PEN = sout[1];
    assign SEGLED_CLR = sout[0];

    // VGA 输出
    reg [11:0] vga_data;
    wire [9:0] col_addr;
    wire [8:0] row_addr;
    vgac v0 (
        .vga_clk(clkdiv[1]), .clrn(SW_OK[0]), .d_in(vga_data), .row_
addr(row_addr),
        .col_addr(col_addr), .r(r), .g(g), .b(b), .hs(hs), .vs(vs)
    );
    // 用于显示是否开启 SW 开关
    wire [15:0] ledData;
    assign ledData = SW_OK;
    ShiftReg #(.WIDTH(16)) ledDevice
(.clk(clkdiv[3]), .pdata(~ledData), .sout({LED_CLK, LED_D0, LED_PEN, L
ED_CLR}));

```

逻辑部分（主体）

```

always @(posedge clk) begin

    if (finish == 1 && record < score)
    // 若游戏结束，且当前分数高于最高记录，则更新最高记录
        record <= score;
    if (finish == 0 && start == 1 && cnt % 25_000_000 == 0)
    // 每过 0.5s 记 1 分
        score <= score + 1;

    if (cnt % 10_000_000 == 0)
    // 每隔 0.2s 切换移动恐龙图片，实现恐龙移动的效果
        flash = ~flash;

    // 处理跳跃逻辑
    if (cnt % 500_000 == 0) begin

```

```

// 上跳/下降速度为 0.1s-1 像素
if (airup == 1) begin
    // 若恐龙正处于上跳状态
    if (dino_y == 250) begin
        // 若恐龙达到最大跳跃高度，开始下降
        airup <= 0;
        airdown <= 1;
    end
    else
        dino_y <= dino_y + 1;
    // 否则继续上跳
end
if (airdown == 1) begin
    // 若恐龙正处于下降状态
    if(dino_y == 120)
    // 若恐龙回到地面，停止下降
        airdown <= 0;
    else dino_y <= dino_y - 1;
    // 否则继续下降
end
end

if (!rstn) begin
    // 按下复位键，回到开始界面，当前分数清零
    start <= 0;
    score <= 0;
end

// 处理 PS2 键盘输入
else if (start == 1) begin
    if (up == 1) begin
        // 若按下 '↑' 键，恐龙跳起
        is_crouch <= 0;
        if (airup == 0 && airdown == 0)
            airup <= 1;
        end

        if (down == 1 && airup == 0 && airdown == 0) begin
            // 若按下 '↓' 键，且恐龙不在空中，则下蹲，并改变相应坐标
            is_crouch <= 1;
            width <= 8'd70;
            height <= 5'd30;
        end
    end
end

```

```

        if (down == 0) begin
            // 若没有按下任何键，显示正常恐龙
            if (is_crouch == 1) begin
                is_crouch <= 0;
                width <= 7'd60;
                height <= 7'd60;

            end
        end
    end

// 若按下 PS2 键盘任意键，初始化数据，并进入游戏
else if (ready == 1) begin
    // 封面
    start <= 1;
    finish <= 0;
    // 恐龙
    dino_y <= 9'd120;
    width <= 7'd60;
    height <= 7'd60;
    leap <= 10'd150;
    is_crouch <= 0;
    // 仙人掌
    cactus1_1_x <= 10'd620;
    cactus1_2_x <= Ini;
    width_c <= 5'd20;
    height_c <= 6'd40;
    start_c1[0] <= 1;
    start_c1[1] <= 0;
    // 翼龙
    ptero_x <= Ini;
    ptero_used <= 0;

    speed <= 4'd5;
end

// 若仙人掌或翼龙移动到最左端，则返回至最右端，实现循环效果
if (cactus1_1_x <= 0)
    cactus1_1_x <= 10'd620;
if (cactus1_2_x <= 0)
    cactus1_2_x <= 10'd620;
if (ptero_x <= 0)
    ptero_x <= 10'd620;

// 碰撞判断：若撞到障碍物，游戏结束

```

```

if ( width >= cactus1_1_x) begin
    if (ground_height + 39 >= dino_y )
        finish <= 1;
end
if (width >= cactus1_2_x) begin
    if (ground_height + 39 >= dino_y )
        finish <= 1;
end
if (width >= ptero_x) begin
    if (is_crouch == 0 && dino_y <= ptero_height + 29)
        finish <= 1;
end

// 处理仙人掌和恐龙的位置
if(start_c1[0] && !ptero_used && cactus1_1_x <= 9'd440) begin
    ptero_used = 1;
    ptero_x = 10'd620;
end

if (start_c1[0] && !start_c1[1] && cactus1_1_x <= 9'd240) begin
    start_c1[1] <= 1;
    cactus1_2_x <= 10'd620;
end

// 每隔 4s 移动速度变快，从而逐渐提升游戏难度
if (start == 1 && cnt_time == 3 && speed < 5) begin
    speed <= speed + 1;
end

// 每隔 0.1s 移动仙人掌和翼龙，使游戏动画看起来更加“丝滑”
if(start == 1 && cnt % 5_000_000==0) begin
    if (start_c1[0])
        cactus1_1_x <= cactus1_1_x - speed;
    if (start_c1[1])
        cactus1_2_x <= cactus1_2_x - speed;
    if (ptero_used)
        ptero_x <= ptero_x-speed;
end

// 十进制计数
if (score[3:0] == 4'ha) begin
    score[7:4] <= score[7:4] + 1;
    score[3:0] <= 4'h0;
end

```



```

    if (score[7:4] == 4'ha) begin
        score[11:8] <= score[11:8] + 1;
        score[7:4] <= 4'h0;
    end
    if (score[11:8] == 4'ha) begin
        score[15:12] <= score[15:12] + 1;
        score[11:8] <= 4'h0;
    end
end
end

```

3.2.2 IP 核的生成

1. 选取或绘制图片，并将图片文件设为位图(.bmp)形式

(这里插入用到的图片)

2. 将位图转换为.coe 文件，这里我们用到了 Python 代码，如下所示：

```

bmp_to_coe.py
1  def bmp_to_coe(bmp_filename, coe_filename):
2  # 打开 BMP 文件并读取内容
3  dir = "./images/" + bmp_filename
4  with open(dir, 'rb') as bmp_file:
5  bmp_data = bmp_file.read()

6  # BMP 文件的前 54 个字节是文件头，包含文件信息
7  # 图像数据从第 55 个字节开始，每个像素占 3 个字节 (RGB)
8  image_data = bmp_data[54:]

9  # 获取图像的宽度和高度信息
10 width = int.from_bytes(bmp_data[18:22], byteorder='little')
11 height = int.from_bytes(bmp_data[22:26], byteorder='little')

12 # COE 文件的内容，包括宽度、高度和图像数据
13 coe_content = "memory_initialization_radix=2;\n"
14 coe_content += "memory_initialization_vector=\n"

15 # 将图像数据转换为 COE 格式的字符串
16 for y in range(height):
17 for x in range(width):
18 # 计算当前像素在图像数据中的偏移量
19 offset = (y * width + x) * 3

```

```

20 # 从图像数据中提取像素的 RGB 值
21 blue = bmp_data[54 + offset]
22 green = bmp_data[54 + offset + 1]
23 red = bmp_data[54 + offset + 2]

24 # 将 RGB 值转换为二进制格式，并添加到 COE 文件内容中
25 pixel_value = (red >> 4) << 8 | (green >> 4) << 4 | (blue >> 4)
26 coe_content += f"{pixel_value:012b},"

27 # 将 COE 文件内容写入 COE 文件
28 with open(coe_filename, 'w') as coe_file:
29 # 去除最后一个逗号，并添加结束符号
30 coe_content = coe_content[:-1] + ";\n"
31 coe_file.write(coe_content)

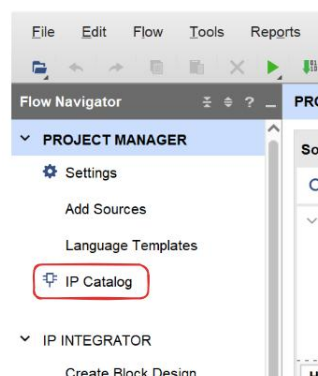
32 # 测试代码
33 if __name__ == "__main__":
34 name = input("请输入图片名称(不带文件扩展名): ")
35 bmp_filename = name + ".bmp"
36 coe_filename = name + ".coe"
37 bmp_to_coe(bmp_filename, coe_filename)
38 print("已转换")

```

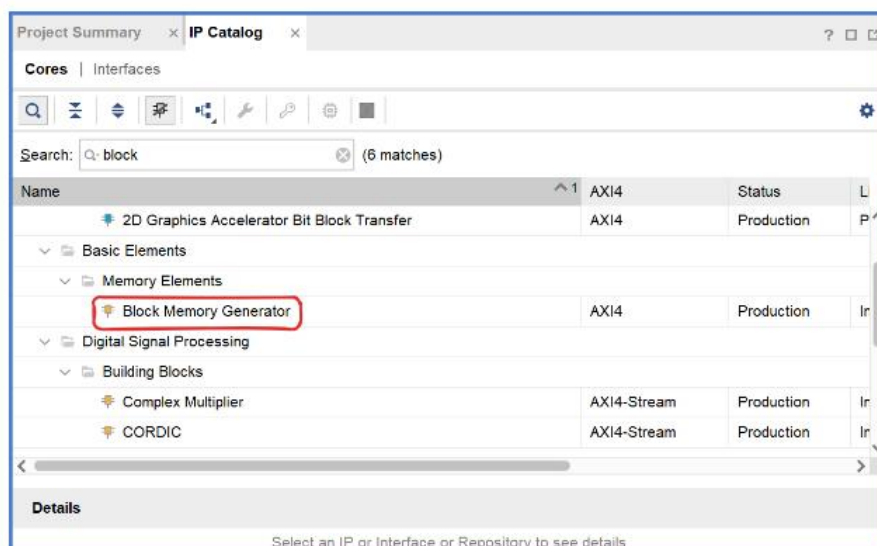
通过该程序，我们得到了所有图片的对应.coe 文件。

(这里插入所有.coe 文件的图片)

3. 打开 Vivado 左侧的“IP Catalog”。

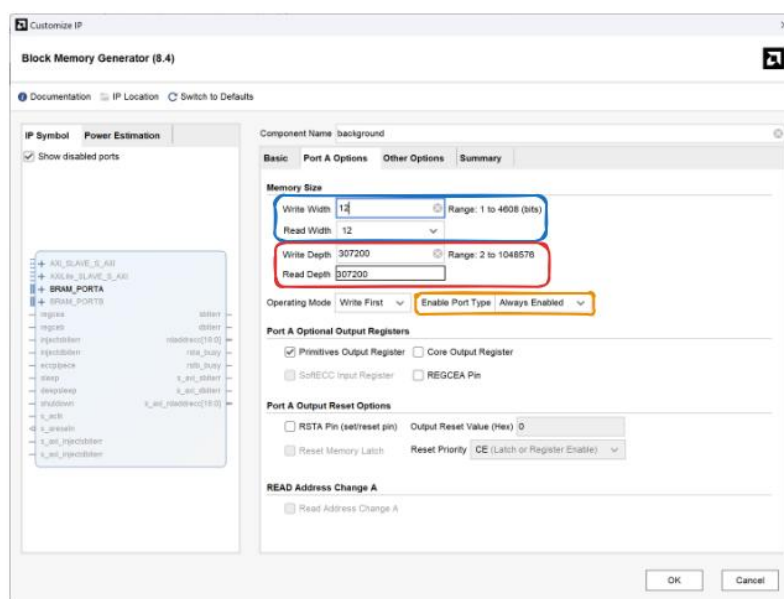


利用搜索功能找到“Block Memory Generator”，双击进入。

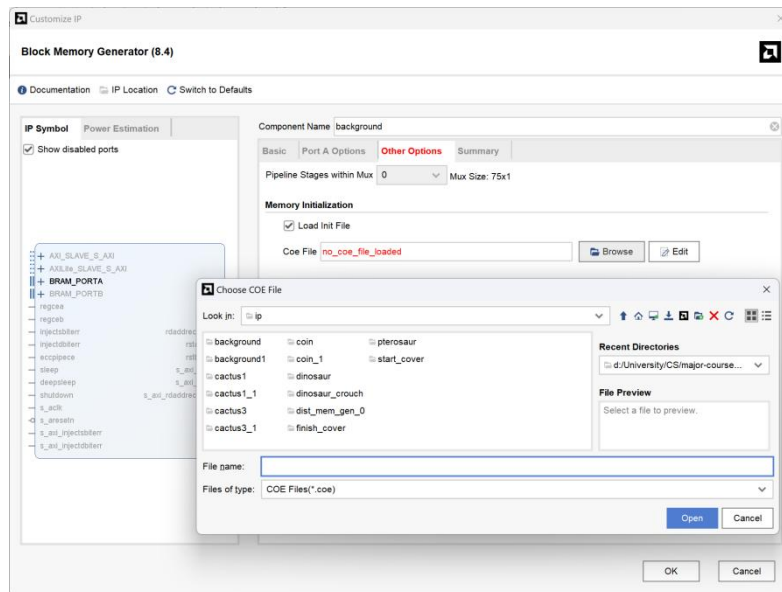


4. IP 核的配置

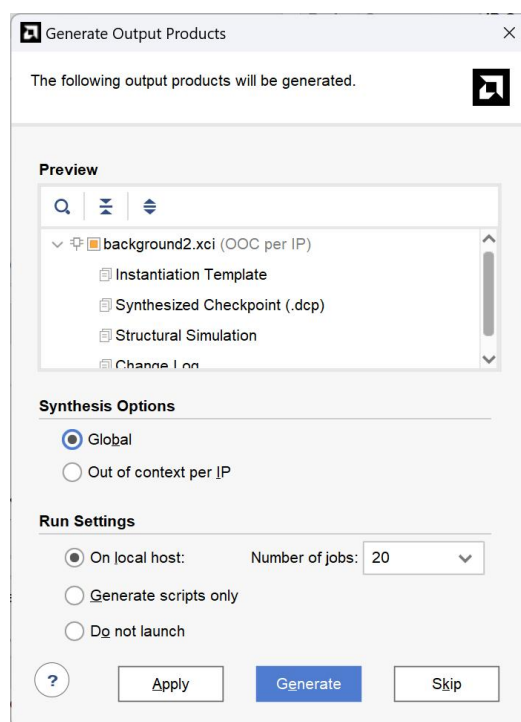
- 进入后，先在“Component Name”中给 IP 核取名字。
- 点击“Port A Options”选项，进入下图所示的界面，在蓝色方框处输入数据的位宽，这里输入 12（因为 VGA 要求 RGB 的位宽为 12bit）；在红色方框处输入数据的个数，即图片的像素总数（宽 x 高）；在橙色方框处选择“Always Enabled”。



- 然后进入“Other Options”界面，导入要生成 IP 核的 .coe 文件。接着点击下面的“OK”按钮。



- 在“Synthesis Options”中选择“Global”，最后点击“Generate”生成 IP 核。



- 最终效果如下 (cactus3.xci)：

- Top (Top.v)
- SegmentDecoder (SegmentDecoder.v)
- > ④ cactus3 (cactus3.xci)
- clk_100ms (clk_100ms.v)
- clk_10ms (clk_10ms.v)

3.2.3 VGA 显示

本设计中采用了 VGA demo 中给出的 vgac 模块，由它实现显示屏上的像素扫描，并给出了实时位置坐标。代码如下所示：

```
vgac.v
`timescale 1ns / 1ps
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
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]; // 2-bit blue
end
endmodule

```

在顶层模块中的调用：

1. IP 核的输入和输出变量的声明

```

reg [18:0] StCoverAdd;    // 开始封面
wire [11:0] StCoverOut;
reg [18:0] EdCoverAdd;    // 结束封面
wire [11:0] EdCoverOut;
reg [18:0] BgAdd;         // 游戏背景图片
wire [11:0] BgOut;

reg [11:0] DinoAdd;       // 空中的恐龙
wire [11:0] DinoOut;
reg [11:0] DinocAdd;      // 下蹲的恐龙
wire [11:0] DinocOut;
reg [11:0] Dino1Add;      // 移动的恐龙 1
wire [11:0] Dino1Out;
reg [11:0] Dino2Add;      // 移动的恐龙 2
wire [11:0] Dino2Out;

```

```

reg [9:0] Cacone1Add;          // 仙人掌 1
wire [11:0] Cacone10ut;
reg [9:0] Cacone2Add;          // 仙人掌 2
wire [11:0] Cacone20ut;

reg [10:0] PteroAdd;           // 翼龙
wire [11:0] Ptero0ut;

```

2. IP 核模块的调用

```

// 3 张封面
start_cover
s1(.addra(StCoverAdd), .douta(StCover0ut), .clka(clkdiv[1]));
background b1(.addra(BgAdd), .douta(Bg0ut), .clka(clkdiv[1]));
finish_cover
f1(.addra(EdCoverAdd), .douta(EdCover0ut), .clka(clkdiv[1]));

// 恐龙
dinosaur d1(.addra(DinoAdd), .douta(Dino0ut), .clka(clkdiv[1]));
dinosaur_crouch
d2(.addra(DinocAdd), .douta(Dinoc0ut), .clka(clkdiv[1]));
dinosaur1
d3(.addra(Dino1Add), .douta(Dino10ut), .clka(clkdiv[1]));
dinosaur2
d4(.addra(Dino2Add), .douta(Dino20ut), .clka(clkdiv[1]));

// 仙人掌
cactus1
c11(.addra(Cacone1Add), .douta(Cacone10ut), .clka(clkdiv[1]));
cactus1
c12(.addra(Cacone2Add), .douta(Cacone20ut), .clka(clkdiv[1]));

// 翼龙
pterosaur
p1(.addra(PteroAdd), .douta(Ptero0ut), .clka(clkdiv[1]));

```

3. IP 核输入赋值

```

always @(posedge clkdiv[1]) begin
    // 封面
    StCoverAdd <= (col_addr >= 0 && col_addr <= 639 && row_addr >=
0 && row_addr <= 479)? col_addr + (479 - row_addr) * 640 : 0;
    EdCoverAdd <= (col_addr >= 0 && col_addr <= 639 && row_addr >=
0 && row_addr <= 479)? col_addr + (479 - row_addr) * 640 : 0;
    BgAdd <= (col_addr >= 0 && col_addr <= 639 && row_addr >= 0
&& row_addr <= 479)? (col_addr + 620 - cactus1_1_x) % 640 + (479 -

```

```

row_addr)*640 : 0;
    // 恐龙
    DinoAdd <= (col_addr >= 0 && col_addr <= 59 && 479-row_addr >=
dino_y && 479-row_addr <= dino_y + 59)? col_addr + (479 - row_addr
- dino_y) * 60 : 0;
    Dino1Add <= (col_addr >= 0 && col_addr <= 59 && 479-row_addr >=
dino_y && 479-row_addr <= dino_y + 59)? col_addr + (479 - row_addr
- dino_y) * 60 : 0;
    Dino2Add <= (col_addr >= 0 && col_addr <= 59 && 479-row_addr >=
dino_y && 479-row_addr <= dino_y + 59)? col_addr + (479 - row_addr
- dino_y) * 60 : 0;
    DinocAdd <= (col_addr >= 0 && col_addr <= 69 && 479-row_addr >=
dino_y && 479-row_addr <= dino_y + 29)? col_addr + (479 - row_addr
- dino_y) * 70 : 0;
    // 仙人掌
    Cacone1Add <= (col_addr >= cactus1_1_x && col_addr <=
cactus1_1_x + 19 && 479-row_addr >= ground_height && 479 - row_addr
<= ground_height + 39)? (col_addr - cactus1_1_x) + (479 - row_addr
- ground_height) * 20 : 0;
    Cacone2Add <= (col_addr >= cactus1_2_x && col_addr <=
cactus1_2_x + 19 && 479-row_addr >= ground_height && 479 - row_addr
<= ground_height + 39)? (col_addr - cactus1_2_x) + (479 - row_addr
- ground_height) * 20 : 0;
    // 翼龙
    PteroAdd <= (col_addr >= ptero_x && col_addr <= ptero_x + 64
&& 479-row_addr >= ptero_height && 479-row_addr <= ptero_height + 29)?
(col_addr - ptero_x) + (479 - row_addr - ptero_height) * 65 : 0;
end

```

4. 显示图片

```

always @(posedge(clkdiv[1])) begin

    // 显示开始封面
    if (start == 0) begin
        if (col_addr >= 0 && col_addr <= 639 && row_addr >= 0 &&
row_addr <= 479)
            vga_data <= StCoverOut;
        end

    // 显示结束封面
    else if (finish == 1) begin
        if (col_addr >= 0 && col_addr <= 639 && row_addr >= 0 &&
row_addr <= 479)
            vga_data <= EdCoverOut;
    end
end

```



```

end

// 显示游戏界面（包括背景、恐龙和障碍物）
else begin
    // 背景
    if (col_addr >= 0 && col_addr <= 639 && row_addr >= 0 &&
row_addr <= 479)
        vga_data <= Bg0ut;

    // 恐龙
    if (!is_crouch) begin                                // 若未下蹲
        if (col_addr >= 0 && col_addr <= 59 && 479 - row_addr >=
dino_y && 479 - row_addr <= dino_y + 59) begin
            if (airup == 1 || airdown == 1)
                // 若在空中，显示空中恐龙
                vga_data <= Dino0ut;
            else if (flash)
                // 否则显示移动恐龙（2 张图交错显示）
                vga_data <= Dino10ut;
            else vga_data <= Dino20ut;
        end
    end
    else begin                                            // 显示下蹲恐龙
        if (col_addr >= 0 && col_addr <= 69 && 479 - row_addr >=
dino_y && 479 - row_addr <= dino_y + 29)
            vga_data <= Dinoc0ut;
        end

    // 仙人掌
    if (start_c1[0]) begin
        if (col_addr >= cactus1_1_x && col_addr <= cactus1_1_x
+ 19 && 479 - row_addr >= ground_height && 479 - row_addr <=
ground_height + 39)
            vga_data <= Cacone10ut;
        end
        if (start_c1[1]) begin
            if (col_addr >= cactus1_2_x && col_addr <= cactus1_2_x
+ 19 && 479 - row_addr >= ground_height && 479 - row_addr <=
ground_height + 39)
                vga_data <= Cacone20ut;
            end
        end

    // 翼龙
    if (ptero_used) begin

```

```

        if (col_addr >= ptero_x && col_addr <= ptero_x + 64 &&
479 - row_addr >= ptero_height && 479 - row_addr <= ptero_height +
29)
            vga_data <= PteroOut;
        end
    end
end

```

3.2.4 PS2 键盘输入

这里也用到了 VGA demo 中的 PS/2 模块，实现键盘输入，键盘上的每一个按键会对应一个输出数据，可以据此判断键盘上哪个键被按下。在本设计中，我们只用到键盘上的 ↑ 和 ↓，分别用 up 和 down 标记是否被按下。代码如下所示：

```

ps2.v
`timescale 1ns / 1ps
module ps2(
    input clk, rst, // clk and reset
    input ps2_clk, ps2_data, // ps2_clk and ps2_data
    output reg up, down,
    output ready
);
    reg ps2_clk_flag0, ps2_clk_flag1, ps2_clk_flag2; // register for
ps2_clk
    wire negedge_ps2_clk = !ps2_clk_flag1 & ps2_clk_flag2; // detect
the fallingedge of ps2_clk
    reg negedge_ps2_clk_shift; // register for the falling edge of
ps2_clk
    reg [9:0] data; // store the 10-bit data received from ps2
    reg data_break, data_done, data_expand; // flags for data
processing
    reg [7:0] temp_data; // temporary storage of ps2 data
    reg [3:0] num; // register for counting the number of ps2 data bits
    // update the clock edge detection and auxiliary register
    always@(posedge clk or posedge rst) begin
        if(rst) begin
            ps2_clk_flag0 <= 1'b0;
            ps2_clk_flag1 <= 1'b0;
            ps2_clk_flag2 <= 1'b0;
        end
    end
end

```

```

        else begin
            ps2_clk_flag0 <= ps2_clk; //ps2_clk_flag0 is the
previousvalue of ps2_clk
            ps2_clk_flag1 <= ps2_clk_flag0;
            ps2_clk_flag2 <= ps2_clk_flag1;
        end
    end
    // update the PS2 data bit counter
    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
    // update the auxiliary register
    always@(posedge clk) begin
        negedge_ps2_clk_shift <= negedge_ps2_clk;
    end
    // logic for reading data bits and temporary storage
    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
    // logic for PS2 data processing
    always@(posedge clk or posedge rst) begin
        if(rst) begin
            data_break <= 1'b0;
            data <= 10'd0;
        end
    end

```

```

        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 ready=data_done;
// logic for PS2 data decoding
always @(posedge clk) begin
    case (data)
        10'h275: up <= 1;
        10'h375: up <= 0;
        10'h272: down <= 1;
        10'h372: down <= 0;
    endcase
end
endmodule

```

在顶层模块的调用见“顶层模块”一节的[逻辑部分](#)。

3.2.5 七段数码管显示

七段数码管用来显示游戏的当前分数和最高记录。代码如下所示：

Seg7Device.v

```

`timescale 1ns / 1ps
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

```

Seg7Decode.v

```

`timescale 1ns / 1ps
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

```

Seg7Remap.v

```

`timescale 1ns / 1ps
module Seg7Remap(
    input [31:0] I, output [31:0] O
);

    assign O[ 7: 0] = {I[24], I[12], I[5], I[17], I[25], I[16], I[4],
I[0]};
    assign O[15: 8] = {I[26], I[13], I[7], I[19], I[27], I[18], I[6],
I[1]};
    assign O[23:16] = {I[28], I[14], I[9], I[21], I[29], I[20], I[8],
I[2]};
    assign O[31:24] = {I[30], I[15], I[11],I[23], I[31], I[22],
I[10],I[3]};

endmodule

```

ShiftReg.v

```

`timescale 1ns / 1ps
module ShiftReg(clk, pdata, sout);
    parameter WIDTH = 16;
    parameter DELAY = 12;
    input clk;
    input [WIDTH - 1:0] pdata;

```

```

output [3:0] sout;
assign sout = {sck, sdat, oe, clrn};

wire sck, sdat, clrn;
reg oe;

reg [WIDTH:0] shift;
reg [DELAY-1:0] counter = -1;
wire sckEn;

assign sckEn = |shift[WIDTH - 1:0];
assign sck = ~clk & sckEn;
assign sdat = shift[WIDTH];
assign clrn = 1'b1;

always @ (posedge clk)
begin
    if(sckEn)
        shift <= {shift[WIDTH - 1:0], 1'b0};
    else
        begin
            if(&counter)
            begin
                shift <= {pdata, 1'b1};
                oe <= 1'b0;
            end
            else
                oe <= 1'b1;
                counter <= counter + 1'b1;
            end
        end
    end
endmodule

```

附：引脚约束文件

```

constraints.v
# Filename: constraints.xdc
## Constraints file for MyFinalProject

# Main clock
set_property PACKAGE_PIN AC18 [get_ports clk]
set_property IOSTANDARD LVCMOS18 [get_ports clk]

```

```

create_clock -period 10.000 -name clk [get_ports "clk"]

set_property PACKAGE_PIN W13 [get_ports {rstn}]
set_property IOSTANDARD LVCMOS18 [get_ports {rstn}]

set_property PACKAGE_PIN AA10 [get_ports {SW[0]}]
set_property PACKAGE_PIN AB10 [get_ports {SW[1]}]
set_property PACKAGE_PIN AA13 [get_ports {SW[2]}]
set_property PACKAGE_PIN AA12 [get_ports {SW[3]}]
set_property PACKAGE_PIN Y13 [get_ports {SW[4]}]
set_property PACKAGE_PIN Y12 [get_ports {SW[5]}]
set_property PACKAGE_PIN AD11 [get_ports {SW[6]}]
set_property PACKAGE_PIN AD10 [get_ports {SW[7]}]
set_property PACKAGE_PIN AE10 [get_ports {SW[8]}]
set_property PACKAGE_PIN AE12 [get_ports {SW[9]}]
set_property PACKAGE_PIN AF12 [get_ports {SW[10]}]
set_property PACKAGE_PIN AE8 [get_ports {SW[11]}]
set_property PACKAGE_PIN AF8 [get_ports {SW[12]}]
set_property PACKAGE_PIN AE13 [get_ports {SW[13]}]
set_property PACKAGE_PIN AF13 [get_ports {SW[14]}]
set_property PACKAGE_PIN AF10 [get_ports {SW[15]}]

set_property IOSTANDARD LVCMOS15 [get_ports {SW[0]}]
set_property IOSTANDARD LVCMOS15 [get_ports {SW[1]}]
set_property IOSTANDARD LVCMOS15 [get_ports {SW[2]}]
set_property IOSTANDARD LVCMOS15 [get_ports {SW[3]}]
set_property IOSTANDARD LVCMOS15 [get_ports {SW[4]}]
set_property IOSTANDARD LVCMOS15 [get_ports {SW[5]}]
set_property IOSTANDARD LVCMOS15 [get_ports {SW[6]}]
set_property IOSTANDARD LVCMOS15 [get_ports {SW[7]}]
set_property IOSTANDARD LVCMOS15 [get_ports {SW[8]}]
set_property IOSTANDARD LVCMOS15 [get_ports {SW[9]}]
set_property IOSTANDARD LVCMOS15 [get_ports {SW[10]}]
set_property IOSTANDARD LVCMOS15 [get_ports {SW[11]}]
set_property IOSTANDARD LVCMOS15 [get_ports {SW[12]}]
set_property IOSTANDARD LVCMOS15 [get_ports {SW[13]}]
set_property IOSTANDARD LVCMOS15 [get_ports {SW[14]}]
set_property IOSTANDARD LVCMOS15 [get_ports {SW[15]}]

# LED
set_property PACKAGE_PIN AF24 [get_ports {buzzer}]
set_property IOSTANDARD LVCMOS33 [get_ports {buzzer}]

set_property PACKAGE_PIN AD21 [get_ports {AN[0]}]

```



```

set_property PACKAGE_PIN AC21 [get_ports {AN[1]}]
set_property PACKAGE_PIN AB21 [get_ports {AN[2]}]
set_property PACKAGE_PIN AC22 [get_ports {AN[3]}]
set_property PACKAGE_PIN AB22 [get_ports {SEGMENT[0]}]
set_property PACKAGE_PIN AD24 [get_ports {SEGMENT[1]}]
set_property PACKAGE_PIN AD23 [get_ports {SEGMENT[2]}]
set_property PACKAGE_PIN Y21 [get_ports {SEGMENT[3]}]
set_property PACKAGE_PIN W20 [get_ports {SEGMENT[4]}]
set_property PACKAGE_PIN AC24 [get_ports {SEGMENT[5]}]
set_property PACKAGE_PIN AC23 [get_ports {SEGMENT[6]}]
set_property PACKAGE_PIN AA22 [get_ports {SEGMENT[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {AN[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {AN[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {AN[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {AN[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SEGMENT[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SEGMENT[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SEGMENT[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SEGMENT[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SEGMENT[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SEGMENT[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SEGMENT[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {SEGMENT[7]}]

set_property PACKAGE_PIN M24 [get_ports {SEGLED_CLK}]
set_property PACKAGE_PIN M20 [get_ports {SEGLED_CLR}]
set_property PACKAGE_PIN L24 [get_ports {SEGLED_D0}]
set_property PACKAGE_PIN R18 [get_ports {SEGLED_PEN}]
set_property IOSTANDARD LVCMOS33 [get_ports {SEGLED_CLK}]
set_property IOSTANDARD LVCMOS33 [get_ports {SEGLED_CLR}]
set_property IOSTANDARD LVCMOS33 [get_ports {SEGLED_D0}]
set_property IOSTANDARD LVCMOS33 [get_ports {SEGLED_PEN}]

set_property PACKAGE_PIN N26 [get_ports {LED_CLK}]
set_property PACKAGE_PIN N24 [get_ports {LED_CLR}]
set_property PACKAGE_PIN M26 [get_ports {LED_D0}]
set_property PACKAGE_PIN P18 [get_ports {LED_PEN}]
set_property IOSTANDARD LVCMOS33 [get_ports {LED_CLK}]
set_property IOSTANDARD LVCMOS33 [get_ports {LED_CLR}]
set_property IOSTANDARD LVCMOS33 [get_ports {LED_D0}]
set_property IOSTANDARD LVCMOS33 [get_ports {LED_PEN}]

set_property PACKAGE_PIN AF24 [get_ports {LED[0]}]
set_property PACKAGE_PIN AE21 [get_ports {LED[1]}]

```

```

set_property PACKAGE_PIN Y22 [get_ports {LED[2]}]
set_property PACKAGE_PIN Y23 [get_ports {LED[3]}]
set_property PACKAGE_PIN AA23 [get_ports {LED[4]}]
set_property PACKAGE_PIN Y25 [get_ports {LED[5]}]
set_property PACKAGE_PIN AB26 [get_ports {LED[6]}]
set_property PACKAGE_PIN W23 [get_ports {LED[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {LED[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {LED[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {LED[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {LED[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {LED[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {LED[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {LED[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {LED[7]}]

set_property PACKAGE_PIN V17 [get_ports {BTN_X[0]}]
set_property PACKAGE_PIN W18 [get_ports {BTN_X[1]}]
set_property PACKAGE_PIN W19 [get_ports {BTN_X[2]}]
set_property PACKAGE_PIN W15 [get_ports {BTN_X[3]}]
set_property PACKAGE_PIN W16 [get_ports {BTN_X[4]}]
set_property PACKAGE_PIN V18 [get_ports {BTN_Y[0]}]
set_property PACKAGE_PIN V19 [get_ports {BTN_Y[1]}]
set_property PACKAGE_PIN V14 [get_ports {BTN_Y[2]}]
set_property PACKAGE_PIN W14 [get_ports {BTN_Y[3]}]
set_property IOSTANDARD LVCMOS18 [get_ports {BTN_X[0]}]
set_property IOSTANDARD LVCMOS18 [get_ports {BTN_X[1]}]
set_property IOSTANDARD LVCMOS18 [get_ports {BTN_X[2]}]
set_property IOSTANDARD LVCMOS18 [get_ports {BTN_X[3]}]
set_property IOSTANDARD LVCMOS18 [get_ports {BTN_X[4]}]
set_property IOSTANDARD LVCMOS18 [get_ports {BTN_Y[0]}]
set_property IOSTANDARD LVCMOS18 [get_ports {BTN_Y[1]}]
set_property IOSTANDARD LVCMOS18 [get_ports {BTN_Y[2]}]
set_property IOSTANDARD LVCMOS18 [get_ports {BTN_Y[3]}]

set_property PACKAGE_PIN T20 [get_ports {b[0]}]
set_property PACKAGE_PIN R20 [get_ports {b[1]}]
set_property PACKAGE_PIN T22 [get_ports {b[2]}]
set_property PACKAGE_PIN T23 [get_ports {b[3]}]
set_property PACKAGE_PIN R22 [get_ports {g[0]}]
set_property PACKAGE_PIN R23 [get_ports {g[1]}]
set_property PACKAGE_PIN T24 [get_ports {g[2]}]
set_property PACKAGE_PIN T25 [get_ports {g[3]}]
set_property PACKAGE_PIN N21 [get_ports {r[0]}]
set_property PACKAGE_PIN N22 [get_ports {r[1]}]

```

```

set_property PACKAGE_PIN R21 [get_ports {r[2]}]
set_property PACKAGE_PIN P21 [get_ports {r[3]}]

set_property IOSTANDARD LVCMOS33 [get_ports {b[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {b[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {b[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {b[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {g[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {g[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {g[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {g[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {r[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {r[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {r[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {r[3]}]

set_property PACKAGE_PIN M22 [get_ports {hs}]
set_property PACKAGE_PIN M21 [get_ports {vs}]
set_property IOSTANDARD LVCMOS33 [get_ports {hs}]
set_property IOSTANDARD LVCMOS33 [get_ports {vs}]

set_property PACKAGE_PIN N18 [get_ports {ps2_clk}]
set_property PACKAGE_PIN M19 [get_ports {ps2_data}]
set_property IOSTANDARD LVCMOS33 [get_ports {ps2_clk}]
set_property IOSTANDARD LVCMOS33 [get_ports {ps2_data}]

```

3.3 仿真测试

3.3.1 PS2 键盘物理仿真

为了了解和判断键盘按动放开的具体输出数据,方便后续电路设计中按键的判断,小组成员进行了键盘物理验证,主要测试按键‘↑’和‘↓’的键值,以确保 PS2 模块的正确性,测试代码如下所示。

```

test_ps2.v
module test_ps2(
    input wire clk,
    input ps2_clk,
    input ps2_data,

```

```

    input [0:0] SW,
    output wire [3:0] AN,
    output wire [7:0] SEGMENT,
    output reg [7:0] LED
);

    reg [15:0] data_out;
    wire up, down;
    wire [4:0] keyCode;
    wire keyReady;

    ps2 m0(
        .clk(clk),
        .rst(SW[0]),
        .ps2_clk(ps2_clk),
        .ps2_data(ps2_data),
        .up(up),
        .down(down)
    );

    always @(posedge clk) begin
        if (up == 1) begin
            data_out <= 16'h1001;
            LED <= 8'b11000011;
        end
        else if (down == 1) begin
            data_out <= 16'h0110;
            LED <= 8'b00111100;
        end
        else begin
            data_out <= 16'h0;
            LED <= 8'b00000000;
        end
    end



    DisplayNumber m1( .clk(clk),
        .hexs({data_out[15:0]}),
        .LEs(4'b0),
        .points(4'b0),
        .rst(1'b0),
        .AN(AN),
        .SEGMENT(SEGMENT)
    );


```

endmodule

其中 `DisplayNumber` 模块在 lab7 中已设计过，故不再赘述。

上板验证的结果如图所示：

图片	说明
	什么都不按或者按下除 ‘↑’或 ‘下’ 外的其他按键时，显示数字 0000，且 LED 灯不亮
	当按下 ‘↑’键时，显示数字 1001，两边的 LED 灯亮

	<p>当按下 ‘↓’键时，显示数字 0110，中间 4</p> <p>个 LED 灯亮</p>
---	---

实验结果符合预期，这证明了 PS2 模块的正确性。

由于本次的设计中，代码的逻辑部分相对比较复杂，而且涉及到 IP 核内存的读取，采用仿真波形图的方式不太直观。所以所有的调试都通过生成 bit 文件烧录之后，通过 VGA 输出，具体实践来调试。

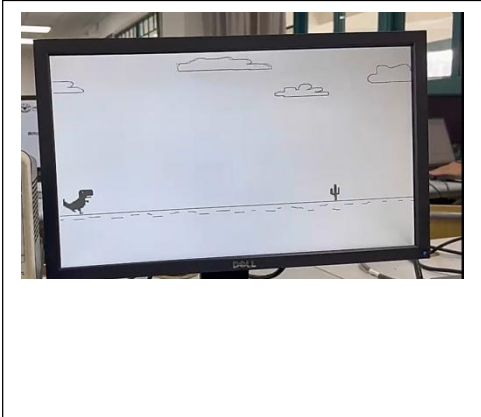
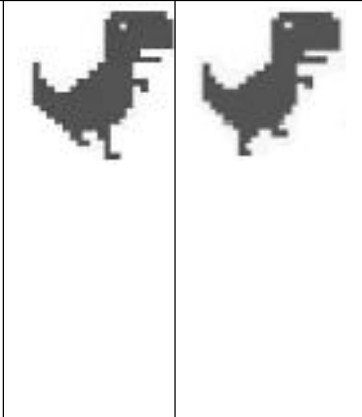

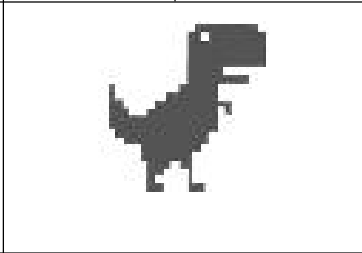
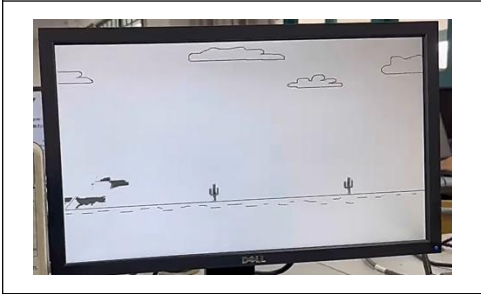

4 测试结果及分析

4.1 开始状态

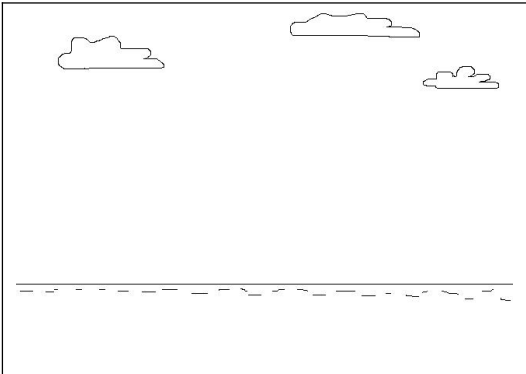
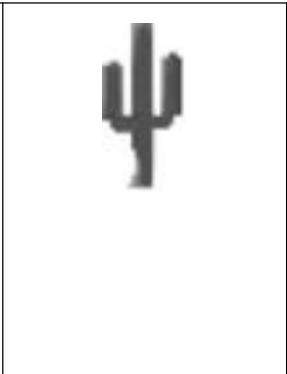

图片	原图对比	说明
		开始封面

	无	七段数码管的分数全部为 0
---	---	---------------


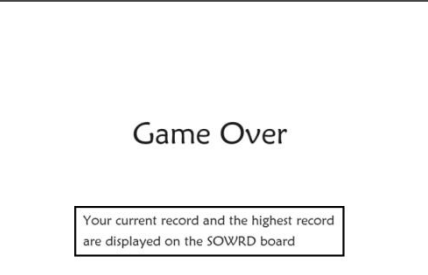
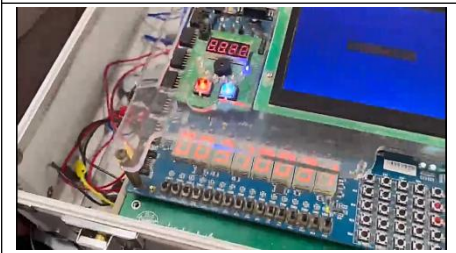
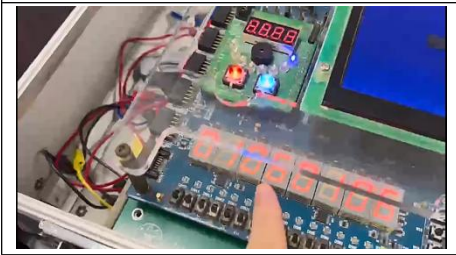
4.2 游戏运行时

图片	原图对比	说明
		不按任何键，或按动无效键，恐龙正常前进，不断切换两张图片，形成跑动的效果
		按下‘↑’键后，恐龙跳起时的状态
		按下‘↓’键后，恐龙下蹲时的状态

背景及障碍物的图片如下所示：


		
背景图片	仙人掌	翼龙

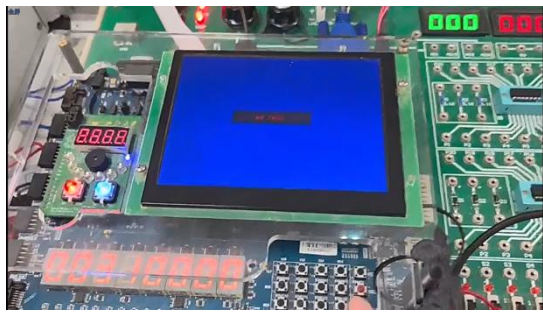
4.3 游戏结束

图片	原图对比	说明
		结束封面
	无	第一轮游戏结束，显示当前得分(31)，并更新最高记录(上一次最高记录：0)
	无	第二轮游戏结束，显示当前得分(106)，并更

		新最高记录(上一次最高记录: 31)
--	--	---------------------------

4.4 游戏重新开始





按下左图所示的复位键后，当前分数清零，并存储最高分数（如右图所示）

5 展望和心得

5.1 展望

- 修复恐龙下蹲和翼龙图片显示的 bug
- 为游戏添加背景音乐
- 设计奖励机制，比如达到一定分数可以为恐龙更换新皮肤等等
- 加大难度，设计更巧妙的障碍，同时也为恐龙设计更多的动作
-

5.2 心得

这次大程设计让我们感慨万千——经过两周多的摸爬打滚，我们从几乎什么

都不会的状态，到最后终于制作出一个像模像样的小游戏，虽然中间的过程比较痛苦，但是当看到小恐龙能够在画面“活蹦乱跳”，成功地躲过一个又一个障碍时，内心充满了激动、喜悦和满满的成就感。

遥想最开始的时候，看到前辈们的大程报告时还是一头雾水的，看了几眼就看不下去了；然而在ddl和考试周的“压迫”下，还是倒逼着自己，恶补Verilog代码语法，同时还要赶快掌握VGA和PS2协议的大致原理，学习曲线可谓是相当的陡峭。虽然很快我们就弄出了一个初版的代码，但当我们上板验证时，呈现出来的结果可令人哭笑不得。所以在后来，我们的时间大部分便在测试、发现bug、调试这三个环节中“死循环”，有时能够成功解决一些bug，有时花了大半个下午都找不出什么原因，给我们带来了很大的挫败感。尽管如此，但我们还是咬牙坚持着，一直期盼着我们理想中的结果。最后“功夫不负有心人”，我们成功设计出了一个可以正常运行的恐龙跑酷小游戏。虽然游戏机制并不复杂，但没想到我们竟然花费了如此长的时间进行设计和调试，可见硬件知识的“博大精深”。通过本次设计，我们对Verilog语言和硬件设计的理解“更上一层楼”了，这为我们学好之后的硬件课程打下更加坚实的基础。

最后感谢洪老师的悉心指导，以及前辈们留下的宝贵经验！

6 小组成员及分工

- NoughtQ (50%)：游戏的初步整体设计、报告撰写
- gxgg (50%)：游戏的细节改进和调试、视频拍摄