中国科学技术大学计算机学院

《数字电路实验》报告

实验题目：实验 10 综合实验

学生姓名：余致远

学生学号：PB18111740

完成日期：2019 年 12 月 20 日

计算机实验教学中心制

2019 年 09 月

【实验目的】

熟练掌握前面实验中的所有知识点

熟悉几种常用通信接口的工作原理及使用

独立完成具有一定规模的功能电路设计

【实验环境】

PC 一台

Windows 或 Linux 操作系统

Vivado

FPGA 实验平台（Nexys4 DDR）

Logisim

vlab.ustc.edu.cn

自选外设：VGA 屏幕一块

【实验练习】

题目 3 利用所学知识完成功能电路的设计，选题、内容、方案均由自己确定，可使用外设。 要求有一定原创性、有自己的核心代码、电路功能完整，运行稳定，文档详细。

基于 FPGA 的模仿 Chrome Dino 小游戏

作者：余致远

简介

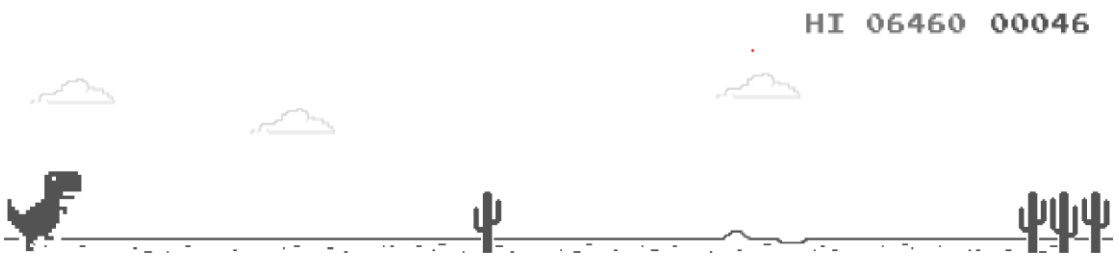


图 1 Chrome Dino 游戏

Chrome Dino 是谷歌 Chrome 浏览器在断网情况下提供的彩蛋游戏，可以通过在地址栏输入 chrome://dino 打开，玩法是通过空格键控制霸王龙跳跃越过障碍物，受到许多用户的喜爱。本次实验中我使用一块车载 VGA 屏和开发板简单模仿了这款游戏。

游戏功能

开发板上的 cpu_reset 键控制重启。开机时显示恐龙屏保，恐龙图片向屏幕四角直线运动，碰到显示器边框以后反弹。按下中央按钮可以从屏保界面/游戏结束界面开始游戏，或者在游戏中进行跳跃。游戏中的唯一障碍物是仙人掌，当恐龙碰到仙人掌时游戏结束，需要按下中央按钮重新开始。开发板下方的一排开关中，通过扳动最右边 12 个开关可以改变恐龙的颜色，12 个开关刚好分别对应开发板支持的 12 位 RGB 信号。13-15 个开关以二进制控制游

戏中恐龙前进（仙人掌前进）的速度，共 8 档。最左边的一个开关作为彩蛋可以使恐龙在游戏中散步，碰到仙人掌不会游戏结束。8 位 LED 数字显示游戏分数，左边为最高分，右边为当前得分。游戏在 VGA 屏幕上以分辨率 1024*768 显示。

STEP1 硬件选择

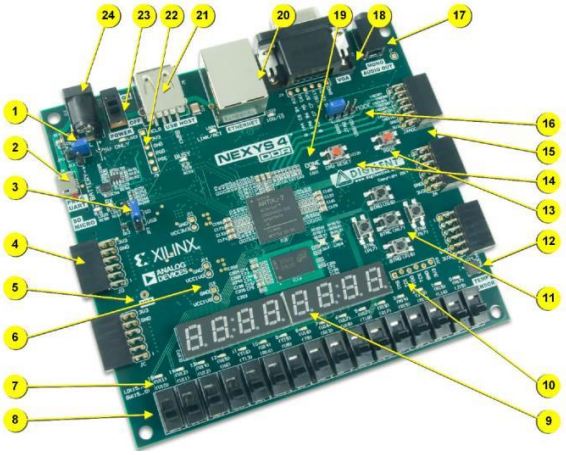
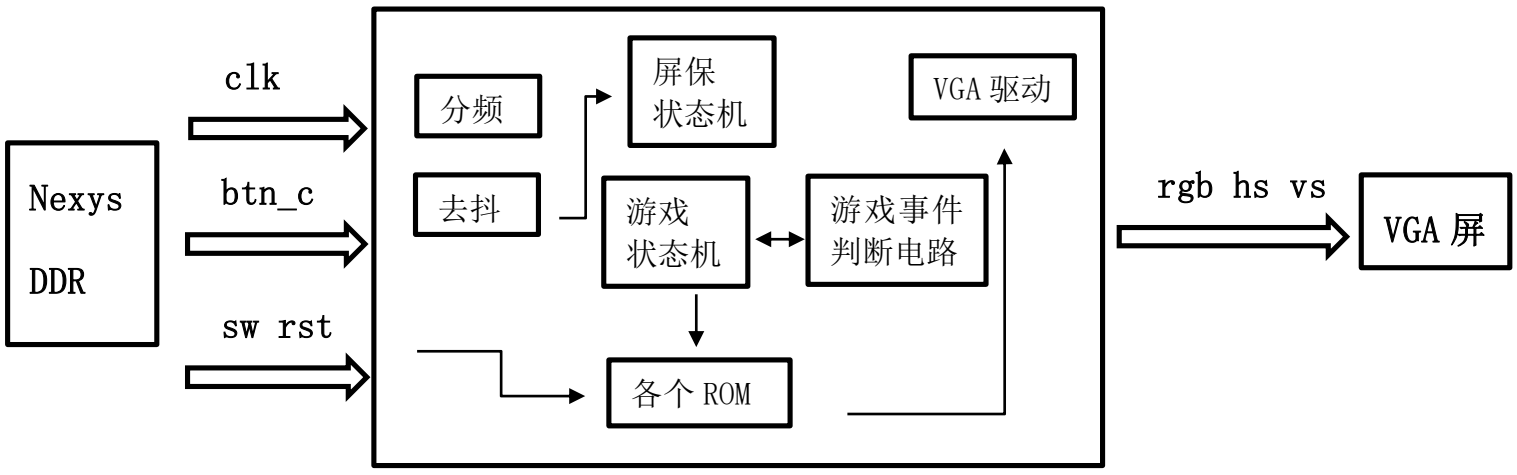


图 3 Nexys DDR 开发板

图 2 VGA 显示屏

STEP2 系统框架



系统主要由以下几个模块构成：行走时左右脚计时控制模块；跳跃模仿模块；恐龙、地面与仙人掌图像 ROM 地址控制模块；rgb 预赋值与碰撞判定模块；rgb 信号输出模块，VGA 信号控制模块；屏保与游戏状态机；计分与 LED 数字控制模块；按键去抖与下降沿检测；各个 ROM。

STEP3：程序设计

接下来分模块进行分析。

1、行走时左右脚控制

以主板时钟简单驱动 25 位 walk_cnt 大数组循环计数，行走状态 walk_state 在有效数字大于 18 位时由 0 变成 1，按 walk_state 加载行走时抬左/右脚的图片。

2、跳跃模仿

利用 walk_state 的第 18 位（经过调试，在这一位的脉冲视觉效果较好）生成脉冲 jump_pulse，驱动 7 位数组 jump_cnt 从 0 到 64 计数。恐龙的跳跃高度是 jump_cnt 的二次函数，经过调整设置为$\frac{(x-32)^2}{3}$个像素。jump_cnt 在自身不计数且有按键信号时才会开始计数。

3、VGA 信号控制

这一模块来自于实验课件，不再赘述。大致原理是根据屏幕的分辨率进行时序控制，输出行时序 vs 与场时序 hs 信号，可以简单理解为当前屏幕上显示像素点的位置。

4、恐龙、地面与仙人掌图像 ROM 及其地址控制

如果当前时序在输出图片的位置，那么就需要从 ROM 中读取图片，此前需要更新向 ROM 传递的地址。为此恐龙、地面与仙人掌都需要定义图片显示的水平与垂直位置。当前像素在图片范围内时，随时更新各图片的地址信号。

为了模拟地面运动，选取了长度为 128 像素的地面重复相连，地面在 128 像素范围内循环运动，通过在屏幕的有限区间内显示地面来体现其流畅运动。

ROM 的 coe 文件通过 matlab 转换图片得到。Matlab 程序来自实验课件引用的博客，为方便 DIY 着色，对原来的程序稍作了修改使图片只存储为黑白两色。图片截取自 Chrome Dino 游戏，稍作了锐化处理。

实际操作中的 ROM 选用了适用于大图片的 Block Memory IP 核，各图片的 ROM 都有各自专用的变量，以便控制。

图 4 仙人掌

图 5 跳跃的恐龙

图 6 抬左脚的恐龙

图 7 抬右脚的恐龙

图 8 撞上仙人掌的恐龙

图 9 地面

5、rgb 预赋值与碰撞判定

定义要输出的 rgb 信号为 12 位数组 W_rom_data，在这一模块中按照恐龙>仙人掌>地面的优先级，用 if-else 语句给 W_rom_data 赋值。赋值时为了使色彩活跃，将仙人掌设置为绿色 12'h0e0，地面设置为黄色 12'h3e0。界面的其他部分为白色 12'hfff。

碰撞判定比较简单，当前位置有恐龙时立起 dinosaur_flag，有仙人掌时立起 cactus_flag，检测这两个变量同时为 1 时立起 crash 信号，表示发生碰撞，其值将在状态机中调用。

6、rgb 信号输出

设定恐龙图片左上角初始位置为水平方向 128 像素，竖直方向 450 像素，地面位置为水平方向 128 像素，竖直方向 450+128-12 像素（128 是恐龙高度，12 是地面高度，这样地面与恐龙的脚会有重合，视觉效果更真实），游戏内容显示的范围为 118-728 共 610 像素长度（起始位置留出空白，这样地面向左单向运动造成的左边多余不会被看见），高度则为当前恐龙跳跃的位置到地面最下沿。

7、按键去抖与下降沿检测

按键去抖的模块包括去毛刺和产生单位时间脉冲信号，代码选取自之前的实验，因为电子元件按键时不稳定的特性，这一模块是很有必要采用的。在状态机中图片位置的改变都应该发生在每一帧结束后，可以将场时钟的下降沿为标志，因此引入了下降沿检测，代码来自引用的博客。

8、屏保与游戏状态机

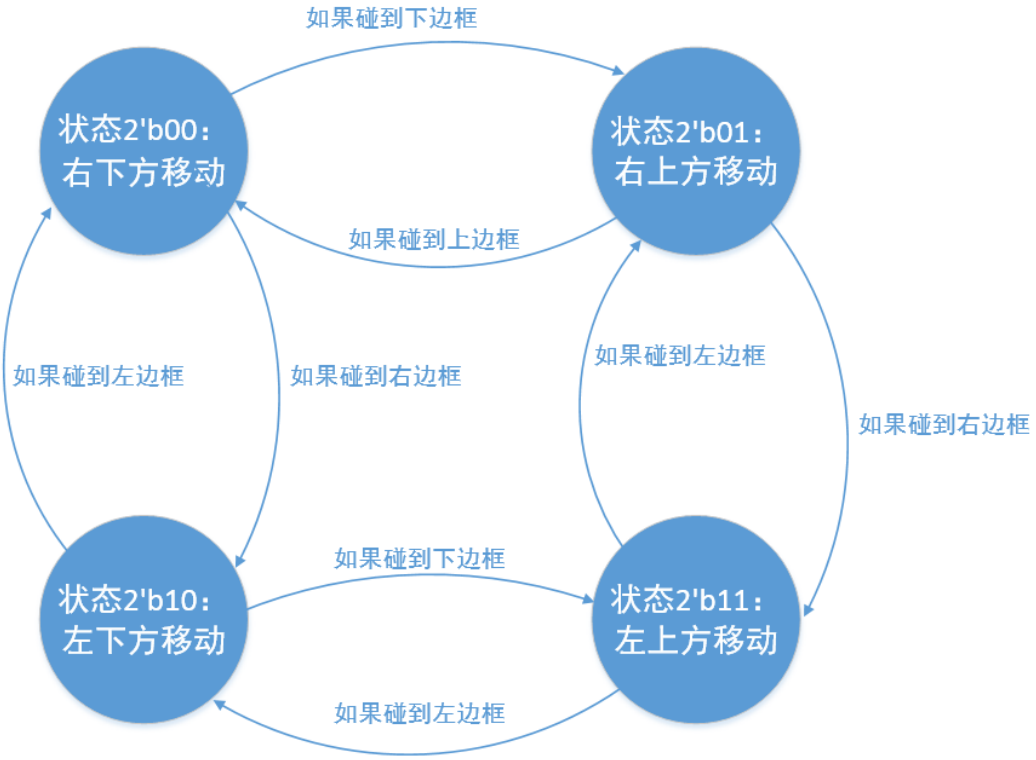


图 10 屏保状态机

屏保代码节选自实验课件引用的博客，原先作为学习调试用，顺便保留了下来。屏保的状态机是简单的向右下、向左下、向右上、向左上四个，在碰到上下左右框的时候发生相应的更改，不再赘述。

屏保与游戏状态机包含共四个状态：SCREEN_SAVER = 2'b00; GAME_START = 2'b01; GAMING = 2'b10; GAME_OVER = 2'b11。

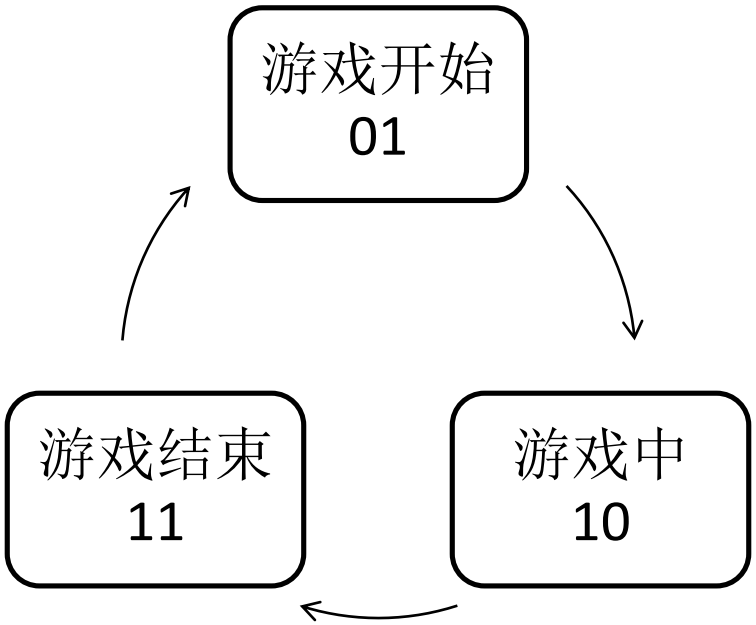
在有限状态机第一部分中，游戏状态转换的情况是：

屏保：如果收到跳跃信号，转到游戏开始；

游戏开始：因为这一状态一定是受跳跃信号激发的，跳跃计数一定也会同时启动，在跳跃计数完成（恐龙落地）后转到游戏中状态。

游戏中：当碰撞信号 crash 为真，且不在游戏彩蛋散步模式中（!sw[15]），转到游戏结束，否则继续游戏。

游戏结束：当发生碰撞时游戏结束，在新的跳跃信号激发的跳跃计数结束后，重新开始游戏。



在有限状态机第三部分中，各个状态中需要进行的主要步骤有：

屏保 SCREEN_SAVER: 屏保状态机。

游戏开始 GAME_START: 这一状态来自于屏保 SCREEN_SAVER，或收到跳跃信号准备重新开始的游戏结束 GAME_OVER 状态，没有仙人掌，地面不动，恐龙跳一下后转到游戏中 GAMING。为恐龙、地面位置设初始值；读取恐龙跳跃的图像，通过跳跃计数设定图片起始位置来模仿恐龙跳跃的动画。

游戏中 GAMING: 主要的游戏状态。有仙人掌，地面运动，恐龙走动或跳跃。如果仙人掌运动出屏幕，随机生成仙人掌（事实上没有随机生成，而是使用了有一定随机性的左右脚计数数组的第 20 位作为标志）；根据从左到右第二到第四个 switch 开关的值改变仙人掌、地面的行进速度；根据当前的左右脚计数或跳跃信号和跳跃计数选取

恐龙的图像，根据跳跃计数改变恐龙的位置。

游戏结束 GAME_OVER: 游戏结束，各图片位置不变，读取恐龙撞仙人掌图像，等待跳跃信号重新转到游戏开始 GAME_START。

9、计分与 LED 数字控制

通过 SCORE 和 HIGH 保存游戏的当前与最高分数，其中 SCORE 会在游戏结束时置零。经过调整发现左右脚计数数组 walk_cnt 的第 23 位适合作为计分的脉冲信号。而 walk_cnt[18:16]适合作为 LED 管的扫描信号。LED 控制模块修改自之前的实验。

STEP4：程序展示与代码放在文末

【总结与思考】

1.请总结本次实验的收获

本次实验用时约有二十多小时，学习 vga 的控制用了一些时间理解代码和进行调试，之后恐龙的跳跃控制花了一些碎片时间完成，星期五连续工作了一天完成整个状态机和调试。实验中除了课件和引用的博客之外难以参考其他代码，因此造成了一些结构杂乱，大量代码集中在一个模块中，一定程度上影响了工程性。但是通过长时间的思考和调试最终成功完成了基本预期功能，此时的成就感难以言说，单看代码长度就可以感受到。

2.请评价本次实验的难易程度

很难。课件中关于 VGA 引用的博客十分有指导价值。实验完成后再回顾实验难度，觉得通过设计合理的结构框架，逐步分解复杂工程（例如分别完成状态机的各状态），结合所学其实已经足以完成比较

复杂的程序。

3.请评价本次实验的任务量

实验任务量很大，当然也取决于个人的设计。

4.请为本次实验提供改进建议

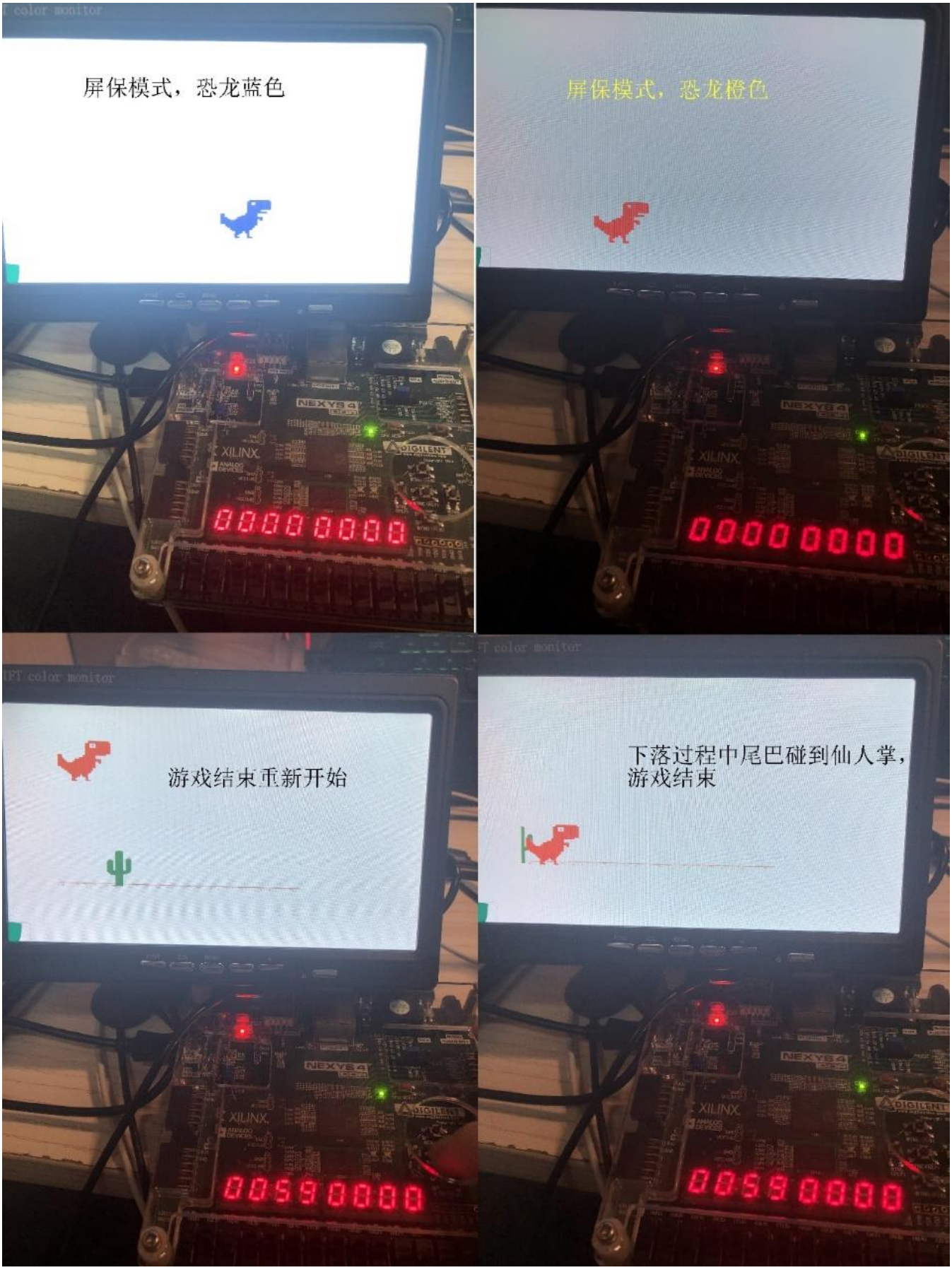
建议实验文档再提前一些给出，实验室也提前开放，给同学提供更宽松的准备时间。如果合适的话可以提供往届同学的项目与报告以供参考，同时也强烈建议引入查重。

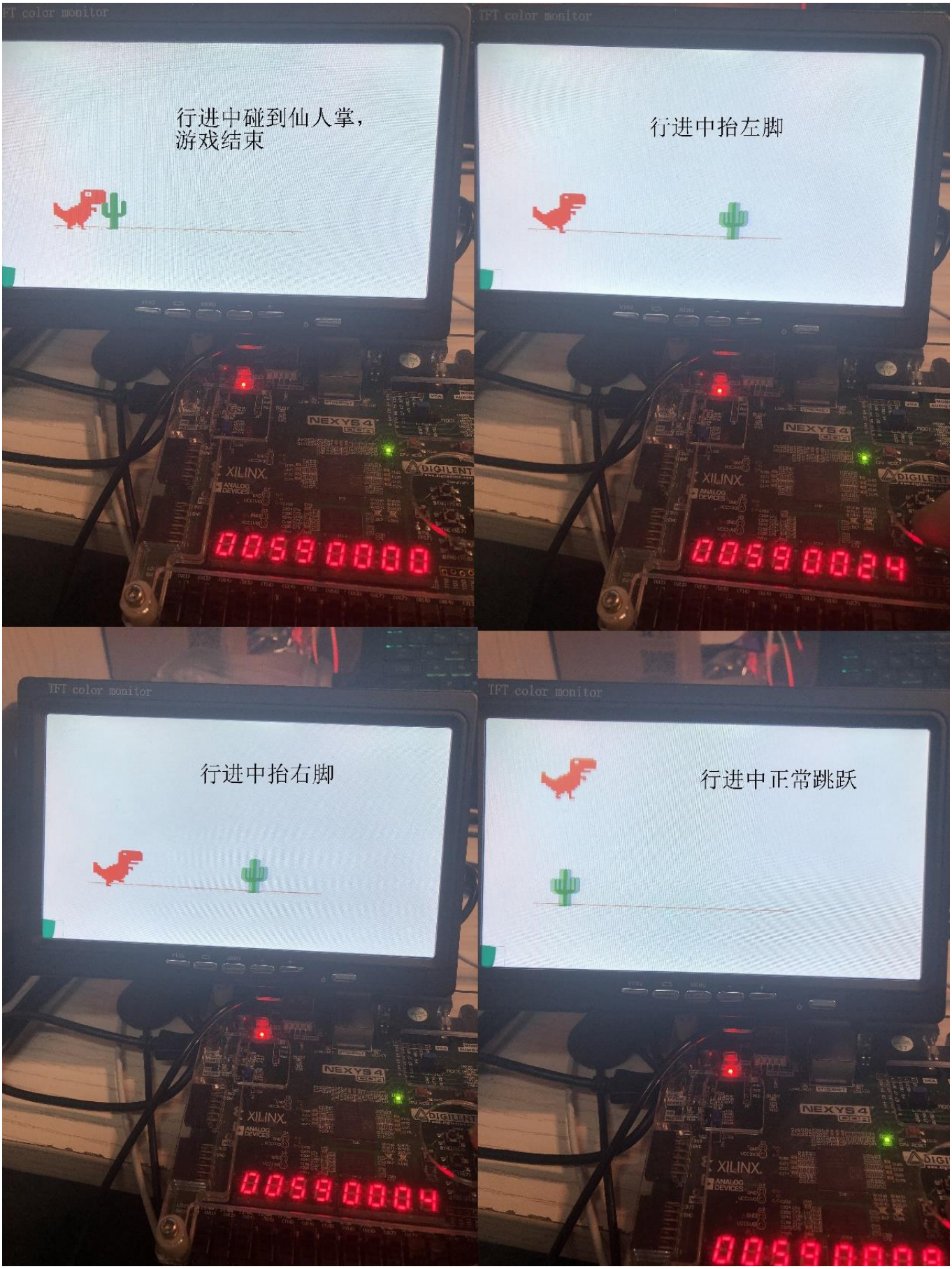
参考文献：

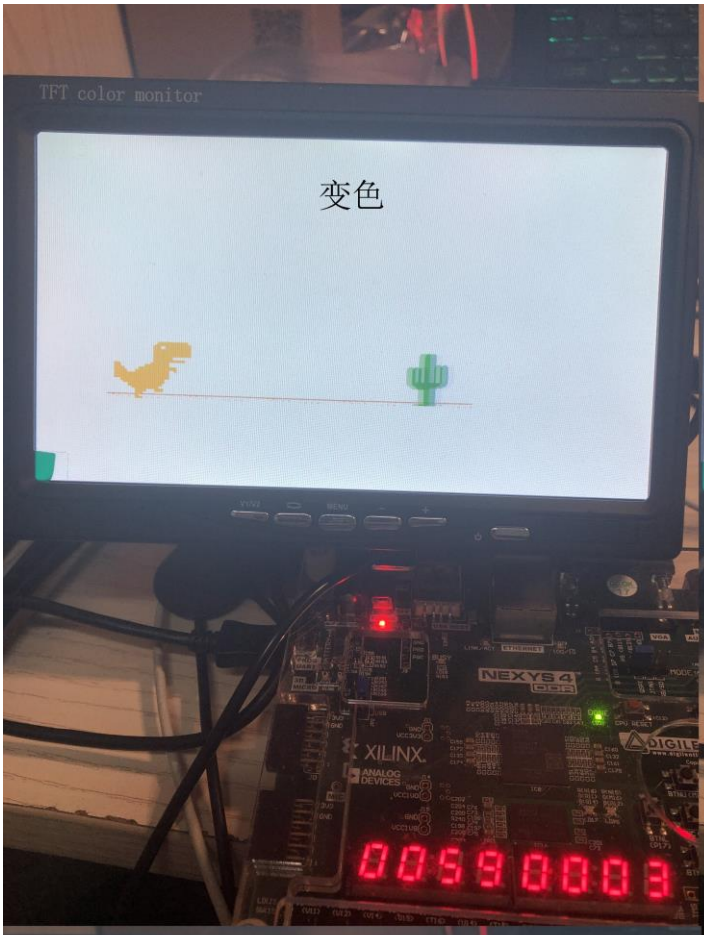
[1]jgliu, 《【接口时序】7、VGA 接口原理与 Verilog 实现》https://www.cnblogs.com/liujinggang/p/9690504.html

[2]中国科学技术大学 vlab 实验中心，《实验 10 综合实验》https://vlab.ustc.edu.cn/home/docs/lab10.pdf

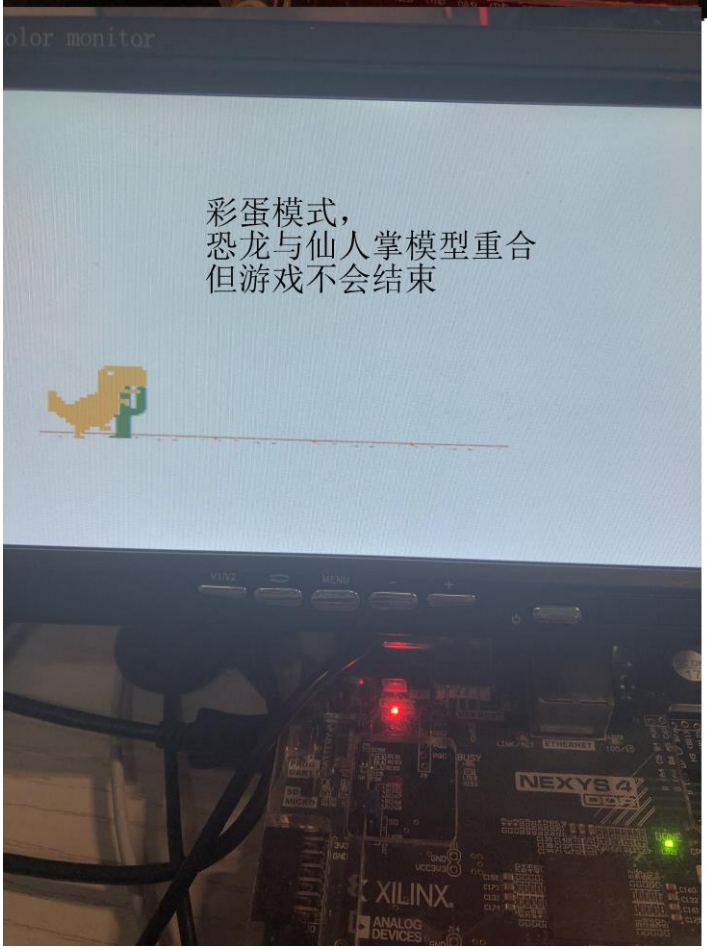
STEP4 程序展示和代码




行进中碰到仙人掌，
游戏结束
行进中抬左脚
行进中抬右脚
行进中正常跳跃


TFT color monitor
变色
NEXYS 4
XILINX
ANALOG DEVICES


TFT color monitor
当前分是最高分，
左边的HIGH马上更新
NEXYS 4
XILINX
ANALOG DEVICES


彩蛋模式，
恐龙与仙人掌模型重合
但游戏不会结束
NEXYS 4
XILINX
ANALOG DEVICES

```
module top(
    input clk,cpu_resetn,//[11:0] rd_data,
    input btnc,
    input [15:0]sw,
    output hs,vs,[11:0]vga_data,
    output [7:0]an,
    output [7:0]seg);

wire clk_65m,lock;
clk_wiz_0 clk_wiz_0(
    .clk_in1 (clk),
    .clk_out1 (clk_65m),
    .reset (~cpu_resetn),
    .locked (lock)
    );
vga_ctrl vga_ctrl(
    .clk (clk_65m),
    .rst (~lock),
 //   .rd_data (rd_data),
    .btnc(btnc),
    .sw(sw),
    .hs (hs),
    .vs (vs),
    .vga_data (vga_data),
    .an(an),
    .seg(seg)
    );
endmodule

module vga_ctrl(
    input clk,
    input rst,//clk=65MHz
    //output [9:0] h_addr,v_addr,
    //output rd_vld,
    //input [11:0] rd_data, //r[3:0],g[3:0],b[3:0]
    input btnc,
    input [15:0]sw,
    output reg hs,vs,
    output [11:0] vga_data,
    output reg [7:0]an,
    output [7:0] seg);

parameter H_CNT = 11'd1343; //136+160+1024+24=1343
parameter V_CNT = 11'd805; //6+29+768+3=806
parameter
    C_H_SYNC_PULSE      =   136   ,
    C_H_BACK_PORCH      =   160  ,
    C_H_ACTIVE_TIME     =   1024 ,
    C_H_FRONT_PORCH     =   24   ,
    C_H_LINE_PERIOD     =   1344 ;
// 分辨率为 1024*768 时场时序各个参数定义
parameter
    C_V_SYNC_PULSE      =   6    ,
    C_V_BACK_PORCH      =   29   ,
    C_V_ACTIVE_TIME     =   768  ,
    C_V_FRONT_PORCH     =   3    ,
    C_V_FRAME_PERIOD    =   806  ;
parameter
    C_IMAGE_WIDTH_dinosaur       =   128   ,
    C_IMAGE_HEIGHT_dinosaur      =   128   ,
    C_IMAGE_PIX_NUM_dinosaur     =   16384 ,

    C_IMAGE_WIDTH_ground       =   768   ,
```

```
    C_IMAGE_HEIGHT_ground      =   12   ,
    C_IMAGE_PIX_NUM_ground     =   9216 ,

    C_IMAGE_WIDTH_cactus1       =   64   ,
    C_IMAGE_HEIGHT_cactus1      =   128   ,
    C_IMAGE_PIX_NUM_cactus1     =   8192 ;

//parameter   C_COLOR_BAR_WIDTH   =   C_H_ACTIVE_TIME / 8  ;
parameter   C_H_OFFSET_dinosaur = 128;
parameter   C_V_OFFSET_dinosaur = 450;

reg [10:0] h_cnt,v_cnt;
reg h_de,v_de;//data_enable
reg [11:0]rd_data;
reg [3:0] data;

reg     [13:0]  R_rom_dinosaur_addr      ; // ROM 的地址
reg     [13:0]  R_rom_ground_addr      ; // ROM 的地址
reg     [13:0]  R_rom_cactus1_addr      ; // ROM 的地址

wire     [3:0]   speed;
assign speed = sw[14:12] * 2'd2;

reg     [11:0]  W_rom_data;
reg     [11:0]  W_rom_data_dinosaur      ; // ROM 中存储的数据
wire    [11:0]  W_rom_dinosaur_jump;
wire    [11:0]  W_rom_dinosaur_leftup;
wire    [11:0]  W_rom_dinosaur_rightup;
wire    [11:0]  W_rom_dinosaur_dead;
wire    [11:0]  W_rom_data_ground;
wire    [11:0]  W_rom_data_cactus1;

reg [3:0] vga_r;
reg [3:0] vga_g;
reg [3:0] vga_b;
wire W_active_flag;
reg ground_flag;
reg dinosaur_flag;
reg cactus_flag;

reg [13:0]HIGH;

wire crash;
assign crash = (dinosaur_flag && cactus_flag);
//////////////////////////////////////////////////////////////////
//状态机相关
//////////////////////////////////////////////////////////////////
reg     [10:0]      R_dinosaur_h_pos         ; // 图片在屏幕上显示的水平位
置，当它为 0 时，图片贴紧屏幕的左边沿
reg     [10:0]      R_dinosaur_v_pos         ; // 图片在屏幕上显示的垂直位
置，当它为 0 时，图片贴紧屏幕的上边沿
reg     [10:0]      R_ground_h_pos         ; // 图片在屏幕上显示的水平位
置，当它为 0 时，图片贴紧屏幕的左边沿
reg     [10:0]      R_ground_v_pos         ; // 图片在屏幕上显示的垂直位
置，当它为 0 时，图片贴紧屏幕的上边沿
reg     [10:0]      R_cactus1_h_pos         ; // 图片在屏幕上显示的水平位
置，当它为 0 时，图片贴紧屏幕的左边沿
reg     [10:0]      R_cactus1_v_pos         ; // 图片在屏幕上显示的垂直位
置，当它为 0 时，图片贴紧屏幕的上边沿
reg     [1:0]       R_state         ;//屏保状态机

reg cactus_come;//仙人掌出现
```

```
wire W_vs_neg;
reg                 R_vs_reg1       ;
reg                 R_vs_reg2       ;

reg [1:0] CURRENT_STATE;
reg [1:0] NEXT_STATE;
parameter SCREEN_SAVER = 2'b00;
parameter GAME_START = 2'b01;
parameter GAMING = 2'b10;
parameter GAME_OVER  = 2'b11;
wire RESTART;
wire RESTART_edge;

wire btnc_clean;
wire jumping;

//////////////////////////////////////////////////////////////////
// 功能：行走时左右脚
//////////////////////////////////////////////////////////////////
reg [24:0]  walk_cnt;
reg [13:0]   score;
wire walk_state;
wire jump_pulse;
wire score_pulse;
always@(posedge clk or posedge rst) //
begin
    if(rst) walk_cnt <= 25'h0;
    else if(walk_cnt>=25'h1ffffff) walk_cnt <= 25'h0;
    else walk_cnt <= walk_cnt + 1'b1;
end
assign walk_state = (walk_cnt[18]==1'b1) ? 1'b0 : 1'b1;
assign jump_pulse = (walk_cnt[18]==1'b1) ? 1'b0 : 1'b1;
assign score_pulse = (walk_cnt[22]==1'b1)? 1'b0 : 1'b1;
//////////////////////////////////////////////////////////////////
// 功能：记分
//////////////////////////////////////////////////////////////////
always@(posedge score_pulse or posedge rst)
begin
    if (CURRENT_STATE==GAMING)
    begin
        if(rst) score <= 14'h0;
        else if(score==14'h3fff) score <= 14'h0;
        else score <= score + 1'b1;
        if(rst) HIGH <= 14'h0;
        else if(score>HIGH) HIGH<=score;
    end
    else if(rst)
    begin
        score <= 14'h0;
        HIGH <= 14'h0;
    end
    else score <= 14'h0;
end
//////////////////////////////////////////////////////////////////
// 功能：跳跃模仿
//////////////////////////////////////////////////////////////////
reg [10:0] R_dinosaur_jump;
reg [6:0] jump_cnt;
always@(posedge rst or posedge jump_pulse or posedge jumping or posedge
 RESTART_edge)
begin
    if(rst || RESTART_edge) jump_cnt <= 7'h0;
    else if(jumping && jump_cnt==7'b0) jump_cnt<=7'b1;
```

```
    else if(jump_cnt==7'd64) jump_cnt <= 7'd0;
    else if(jump_cnt>7'b0) jump_cnt <= jump_cnt + 7'b1;
    R_dinosaur_jump <= (7'd32*7'd32)/3 - ((jump_cnt-7'd32)*(jump_cnt-
7'd32))/3;
end

//////////////////////////////////////////////////////////////////
//功能：游戏中恐龙 ROM 地址和恐龙检测
//////////////////////////////////////////////////////////////////
always @(posedge clk or posedge rst)
begin
    if(rst)
        R_rom_dinosaur_addr  <=  14'd0 ;
    else if(W_active_flag)
        ////////////////////////////////////////////////////////////////
///
        //游戏中
        ////////////////////////////////////////////////////////////////
///
        begin
            if(h_cnt >= (C_H_SYNC_PULSE + C_H_BACK_PORCH + R_dinosaur_h
_pos                        )  &&
               h_cnt <= (C_H_SYNC_PULSE + C_H_BACK_PORCH + R_dinosaur_
h_pos + C_IMAGE_WIDTH_dinosaur  - 1'b1)  &&
               v_cnt >= (C_V_SYNC_PULSE + C_V_BACK_PORCH + R_dinosaur_
v_pos                        )  &&
               v_cnt <= (C_V_SYNC_PULSE + C_V_BACK_PORCH + R_dinosaur_
v_pos + C_IMAGE_HEIGHT_dinosaur - 1'b1)  )
            begin
                if(R_rom_dinosaur_addr == C_IMAGE_PIX_NUM_dinosaur - 1'
b1)
                    R_rom_dinosaur_addr  <=  14'd0 ;
                else
                    R_rom_dinosaur_addr  <=  R_rom_dinosaur_addr  +  1'
b1 ;
                if(W_rom_data_dinosaur < 12'hfff)
                    dinosaur_flag<=1'b1;
                else
                    dinosaur_flag<=1'b0;
            end
            else dinosaur_flag<=1'b0;
        end
    else
    begin
        R_rom_dinosaur_addr  <=  R_rom_dinosaur_addr  ;
        dinosaur_flag<=1'b0;
    end
end

//////////////////////////////////////////////////////////////////
//功能：游戏中地面 ROM 地址和地面检测
//////////////////////////////////////////////////////////////////
always @(posedge clk or posedge rst)
begin
    if(rst)
        R_rom_ground_addr  <=  14'd0 ;
    else if(W_active_flag && CURRENT_STATE!=SCREEN_SAVER)
        ////////////////////////////////////////////////////////////////
///
        //游戏中
        ////////////////////////////////////////////////////////////////
///
        begin
```

```
            if(h_cnt >= (C_H_SYNC_PULSE + C_H_BACK_PORCH + R_ground_h_p
os                      )  &&
                h_cnt <= (C_H_SYNC_PULSE + C_H_BACK_PORCH + R_ground_h_
pos + C_IMAGE_WIDTH_ground  - 1'b1)  &&
                v_cnt >= (C_V_SYNC_PULSE + C_V_BACK_PORCH + R_ground_v_
pos                     )  &&
                v_cnt <= (C_V_SYNC_PULSE + C_V_BACK_PORCH + R_ground_v_
pos + C_IMAGE_HEIGHT_ground - 1'b1)  )
            begin
                if(R_rom_ground_addr == C_IMAGE_PIX_NUM_ground - 1'b1)
                    R_rom_ground_addr  <=  14'd0 ;
                else
                    R_rom_ground_addr  <=  R_rom_ground_addr  +  1'b1 ;

                if(W_rom_data_ground < 12'hfff)
                    ground_flag <= 1'b1;
                else
                    ground_flag <= 1'b0;
            end
            else
                ground_flag <= 1'b0;
        end
    else if(!W_active_flag)
        R_rom_ground_addr  <=  R_rom_ground_addr  ;
end

//////////////////////////////////////////////////////////////////
//功能：游戏中仙人掌 1ROM 地址和仙人掌 1 检测
//////////////////////////////////////////////////////////////////
always @(posedge clk or posedge rst)
begin
    if(rst)
        R_rom_cactus1_addr  <=  14'd0 ;
    else if(W_active_flag && CURRENT_STATE!=SCREEN_SAVER)
        ////////////////////////////////////////////////////////////////
///
        //游戏中
        ////////////////////////////////////////////////////////////////
///
        begin
            if(h_cnt >= (C_H_SYNC_PULSE + C_H_BACK_PORCH + R_cactus1_h_
pos                     )  &&
                h_cnt <= (C_H_SYNC_PULSE + C_H_BACK_PORCH + R_cactus1_h
_pos + C_IMAGE_WIDTH_cactus1  - 1'b1)  &&
                v_cnt >= (C_V_SYNC_PULSE + C_V_BACK_PORCH + R_cactus1_v
_pos                    )  &&
                v_cnt <= (C_V_SYNC_PULSE + C_V_BACK_PORCH + R_cactus1_v
_pos + C_IMAGE_HEIGHT_cactus1 - 1'b1)  )
            begin
                if(R_rom_cactus1_addr == C_IMAGE_PIX_NUM_cactus1 - 1'b1
)
                    R_rom_cactus1_addr  <=  14'd0 ;
                else
                    R_rom_cactus1_addr  <=  R_rom_cactus1_addr  +  1'b1
 ;
                if(W_rom_data_cactus1 < 12'hfff)
                    cactus_flag <= 1'b1;
                else
                    cactus_flag <= 1'b0;
            end
            else
                cactus_flag <= 1'b0;
        end
```

```
    else if(!W_active_flag)
    begin
        R_rom_cactus1_addr  <=  R_rom_cactus1_addr  ;
        cactus_flag <= 1'b0;
    end
end

//////////////////////////////////////////////////////////////////
//功能：vga 预赋值
//////////////////////////////////////////////////////////////////
always @(posedge clk or posedge rst)
begin
    if(rst)
        W_rom_data  <=  12'hfff ;
    else if(W_active_flag)
    begin
        if(CURRENT_STATE==SCREEN_SAVER)
        begin
            if(W_rom_data_dinosaur==12'h000)
                W_rom_data<=sw[11:0];
            else
                W_rom_data<=W_rom_data_dinosaur;
        end
        else
        begin
            if (dinosaur_flag)
            begin
                if(W_rom_data_dinosaur==12'h000)
                    W_rom_data<=sw[11:0];
                else
                    W_rom_data<=W_rom_data_dinosaur;
            end
            else if (cactus_flag)
            begin
                if(W_rom_data_cactus1==12'h000)
                    W_rom_data<=12'h0e0;
                else
                    W_rom_data<=W_rom_data_cactus1;
            end
            else if (ground_flag)
            begin
                if(W_rom_data_ground==12'h000)
                    W_rom_data<=12'h3e0;
                else
                    W_rom_data<=W_rom_data_ground;
            end
            else
                W_rom_data<=12'hfff;
        end
    end
    else
        W_rom_data<=12'hfff;
end

//////////////////////////////////////////////////////////////////
//功能：输出 rgb 信号
//////////////////////////////////////////////////////////////////
always @(posedge clk)
begin
    //R_dinosaur_h_pos <= R_h_pos_saver;
    //R_dinosaur_v_pos <= R_v_pos_saver;
    if(rst)
        begin
```

```
            vga_r <= 4'hf;
            vga_g <= 4'hf;
            vga_b <= 4'hf;
        end
    else if(W_active_flag)
        if(CURRENT_STATE==SCREEN_SAVER)
        begin
            if(h_cnt >= (C_H_SYNC_PULSE + C_H_BACK_PORCH + R_dinosaur_h
_pos)  &&
                h_cnt <= (C_H_SYNC_PULSE + C_H_BACK_PORCH + R_dinosaur_
h_pos + C_IMAGE_WIDTH_dinosaur - 1'b1)  &&
                v_cnt >= (C_V_SYNC_PULSE + C_V_BACK_PORCH + R_dinosaur_
v_pos  )  &&
                v_cnt <= (C_V_SYNC_PULSE + C_V_BACK_PORCH + R_dinosaur_
v_pos + C_IMAGE_HEIGHT_dinosaur - 1'b1)  )
            begin
                vga_r <= W_rom_data[11:8]    ; // 红色分量
                vga_g <= W_rom_data[7:4]     ; // 绿色分量
                vga_b <= W_rom_data[3:0]      ; // 蓝色分
量
            end
            else
            begin
                vga_r <=  4'hf ;
                vga_g <=  4'hf ;
                vga_b <=  4'hf ;
            end
        end
        else //游戏中
        begin
            if(h_cnt >= (C_H_SYNC_PULSE + C_H_BACK_PORCH + R_dinosaur_h
_pos -10'd10 )  &&
                h_cnt <= (C_H_SYNC_PULSE + C_H_BACK_PORCH + R_dinosaur_
h_pos + 10'd600  - 1'b1)  &&
                v_cnt >= (C_V_SYNC_PULSE + C_V_BACK_PORCH + R_dinosaur_
v_pos  )  &&
                v_cnt <= (C_V_SYNC_PULSE + C_V_BACK_PORCH + R_ground_v_
pos + C_IMAGE_HEIGHT_ground - 1'b1)  )
            begin
                vga_r <= W_rom_data[11:8]    ; // 红色分量
                vga_g <= W_rom_data[7:4]     ; // 绿色分量
                vga_b <= W_rom_data[3:0]      ; // 蓝色分
量
            end
            else
            begin
                vga_r <=  4'hf ;
                vga_g <=  4'hf ;
                vga_b <=  4'hf ;
            end
        end
    else
        begin
            vga_r <=  4'hf ;
            vga_g <=  4'hf ;
            vga_b <=  4'hf ;
        end
end

//////////////////////////////////////////////////////////////
//功能：检测时钟下降沿
//////////////////////////////////////////////////////////////
always @(posedge clk or posedge rst)
```

```
begin
    if(rst)
        begin
            R_vs_reg1   <=  1'b0        ;
            R_vs_reg2   <=  1'b0        ;
        end
    else
        begin
            R_vs_reg1   <=  vs        ;
            R_vs_reg2   <=  R_vs_reg1   ;
        end
end
assign W_vs_neg = ~R_vs_reg1 & R_vs_reg2 ;

/*
screen_saver screen_saver(
.clk(clk),
.rst(rst),
.C_IMAGE_WIDTH_dinosaur(C_IMAGE_WIDTH_dinosaur),
.C_IMAGE_HEIGHT_dinosaur(C_IMAGE_HEIGHT_dinosaur),
.C_H_ACTIVE_TIME(C_H_ACTIVE_TIME),
.C_V_ACTIVE_TIME(C_V_ACTIVE_TIME),
.W_vs_neg(W_vs_neg),
.R_dinosaur_h_pos(R_h_pos_saver),
.R_dinosaur_v_pos(R_v_pos_saver)
);
*/

//////////////////////////////////////////////////////////////////
//有限状态机第一部分
//////////////////////////////////////////////////////////////////
always@(posedge clk)
begin
    case(CURRENT_STATE)
        SCREEN_SAVER:
        begin
            if (jump_cnt>1'b0)
                NEXT_STATE<=GAME_START;
            else
                NEXT_STATE<=SCREEN_SAVER;
        end
        GAME_START:
        begin
            if (jump_cnt==7'd64) NEXT_STATE<=GAMING;
            else NEXT_STATE<=GAME_START;
        end
        GAMING:
        begin
            if (!crash) NEXT_STATE<=GAMING;
            else if(sw[15]==1 || jump_cnt>=7'd62) NEXT_STATE<=GAMING;
            else NEXT_STATE<=GAME_OVER;
        end
        GAME_OVER:
        begin
            if (jump_cnt==7'd64)
                NEXT_STATE<=GAME_START;
            else
                NEXT_STATE<=GAME_OVER;
        end
    endcase
end
assign RESTART = (CURRENT_STATE ==  GAME_OVER);
```

```
//////////////////////////////////////////////////////////////////
//有限状态机第二部分
//////////////////////////////////////////////////////////////////
always@(posedge clk or posedge rst)
begin
    if(rst)
        CURRENT_STATE<=SCREEN_SAVER;
    else
        CURRENT_STATE<=NEXT_STATE;
end
//////////////////////////////////////////////////////////////////
//有限状态机第三部分
//////////////////////////////////////////////////////////////////
always@(posedge clk)
begin
    if(rst)
    begin
        R_dinosaur_h_pos <= 1'b0;
        R_dinosaur_v_pos <= 1'b0;
        R_ground_h_pos <= C_H_OFFSET_dinosaur;
        R_ground_v_pos <= C_V_OFFSET_dinosaur + C_IMAGE_HEIGHT_dinosaur
 - C_IMAGE_HEIGHT_ground;
        R_state <=  2'b00;
    end
    else
    case(CURRENT_STATE)
        ////////////////////////////////////////////////////////////////
///
        //功能：显示屏保
        ////////////////////////////////////////////////////////////////
///
        SCREEN_SAVER:
        begin
            W_rom_data_dinosaur <= W_rom_dinosaur_jump;
            if(W_vs_neg)
            begin
                R_ground_h_pos <= C_H_OFFSET_dinosaur;
                R_ground_v_pos <= C_V_OFFSET_dinosaur + C_IMAGE_HEIGHT_
dinosaur - C_IMAGE_HEIGHT_ground;
                R_cactus1_h_pos <= R_dinosaur_h_pos + 10'd600 + C_IMAGE
_WIDTH_cactus1 - 1'b1 ;
                R_cactus1_v_pos <= C_V_OFFSET_dinosaur;
                case(R_state)
                    2'b00: // 图片往右下方移动
                    begin
                        R_dinosaur_h_pos     <=  R_dinosaur_h_pos + 1 ;
                        R_dinosaur_v_pos     <=  R_dinosaur_v_pos + 1 ;
                        if(R_dinosaur_h_pos + C_IMAGE_WIDTH_dinosaur ==
 C_H_ACTIVE_TIME) // 如果碰到右边框
                            R_state <=  2'b10       ;
                        else if((R_dinosaur_v_pos + C_IMAGE_HEIGHT_dino
saur) == C_V_ACTIVE_TIME) // 如果碰到下边框
                            R_state <=  2'b01       ;

                    end
                    2'b01: // 图片往右上方移动
                    begin
                        R_dinosaur_h_pos     <=  R_dinosaur_h_pos + 1 ;
                        R_dinosaur_v_pos     <=  R_dinosaur_v_pos - 1 ;
                        if(R_dinosaur_h_pos + C_IMAGE_WIDTH_dinosaur ==
 C_H_ACTIVE_TIME) // 如果碰到右边框
                            R_state <=  2'b11       ;
```

```
                        else if(R_dinosaur_v_pos == 1)     // 如果碰到上
边框
                            R_state <=  2'b00       ;
                    end
                    2'b10: // 图片往左下方移动
                    begin
                        R_dinosaur_h_pos     <=  R_dinosaur_h_pos - 1 ;
                        R_dinosaur_v_pos     <=  R_dinosaur_v_pos + 1 ;
                        if(R_dinosaur_h_pos == 1)    // 如果碰到左边框
                            R_state <=  2'b00       ;
                        else if(R_dinosaur_v_pos + C_IMAGE_HEIGHT_dinos
aur == C_V_ACTIVE_TIME) // 如果碰到下边框
                            R_state <=  2'b11       ;
                    end
                    2'b11: // 图片往左上方移动
                    begin
                        R_dinosaur_h_pos     <=  R_dinosaur_h_pos - 1 ;
                        R_dinosaur_v_pos     <=  R_dinosaur_v_pos - 1 ;
                        if(R_dinosaur_h_pos == 1)    // 如果碰到上边框
                            R_state <=  2'b01       ;
                        else if(R_dinosaur_v_pos == 1) // 如果碰到左边框
                            R_state <=  2'b10       ;
                    end
                    default:R_state <=  2'b00           ;
                endcase
            end
        end
        GAME_START:
        begin
            cactus_come<=1'b0;
            W_rom_data_dinosaur <= W_rom_dinosaur_jump;
            if(W_vs_neg)
            begin
                R_dinosaur_h_pos <= C_H_OFFSET_dinosaur;
                R_dinosaur_v_pos <= C_V_OFFSET_dinosaur - R_dinosaur_ju
mp;
                R_ground_h_pos <= C_H_OFFSET_dinosaur;
                R_ground_v_pos <= C_V_OFFSET_dinosaur + C_IMAGE_HEIGHT_
dinosaur - C_IMAGE_HEIGHT_ground;
                R_cactus1_h_pos <= R_dinosaur_h_pos + 10'd600 + C_IMAGE
_WIDTH_cactus1 - 1'b1 ;
                R_cactus1_v_pos <= C_V_OFFSET_dinosaur;
            end
        end
        GAMING:
        begin
            if(W_vs_neg)
            begin
                R_dinosaur_h_pos <= C_H_OFFSET_dinosaur;
                R_dinosaur_v_pos <= C_V_OFFSET_dinosaur - R_dinosaur_ju
mp;
                R_ground_v_pos <= C_V_OFFSET_dinosaur + C_IMAGE_HEIGHT_
dinosaur - C_IMAGE_HEIGHT_ground;
                R_ground_h_pos <= (R_ground_h_pos + C_H_OFFSET_dinosaur
 - speed) % C_H_OFFSET_dinosaur ;
                R_cactus1_v_pos <= C_V_OFFSET_dinosaur;
                if (cactus_come)
                begin
                    R_cactus1_v_pos <= C_V_OFFSET_dinosaur;
                    if (R_cactus1_h_pos > speed)
                        R_cactus1_h_pos <= (R_cactus1_h_pos - speed);
                    else
                    begin
```

```
                        cactus_come<=1'b0;
                        R_cactus1_v_pos <= C_V_OFFSET_dinosaur;
                        R_cactus1_h_pos <= R_dinosaur_h_pos + 10'd600 +
 C_IMAGE_WIDTH_cactus1 - 1'b1 ;
                    end
                end
                else
                begin
                    if (walk_cnt[20]==1'b1) cactus_come<=1'b1;
                    R_cactus1_v_pos <= C_V_OFFSET_dinosaur;
                    R_cactus1_h_pos <= R_dinosaur_h_pos + 10'd600 + C_I
MAGE_WIDTH_cactus1 - 1'b1 ;
                end
            end
/*            else
            begin
                R_ground_v_pos <= C_V_OFFSET_dinosaur + C_IMAGE_HEIGHT_
dinosaur - C_IMAGE_HEIGHT_ground;
                R_ground_h_pos <= (R_ground_h_pos + C_H_OFFSET_dinosaur
 - speed) % C_H_OFFSET_dinosaur ;
                R_cactus1_v_pos <= C_V_OFFSET_dinosaur;
            end  */
            if (jump_cnt > 7'h0) W_rom_data_dinosaur <= W_rom_dinosaur_
jump;
            else if(walk_state==1'b0) W_rom_data_dinosaur <= W_rom_dino
saur_leftup;
            else W_rom_data_dinosaur <= W_rom_dinosaur_rightup;
        end
        GAME_OVER:
        begin
            W_rom_data_dinosaur <= W_rom_dinosaur_dead;
            if(W_vs_neg)
            begin
                R_dinosaur_h_pos <= C_H_OFFSET_dinosaur;
                R_dinosaur_v_pos <= C_V_OFFSET_dinosaur - R_dinosaur_ju
mp;
                R_ground_h_pos <= C_H_OFFSET_dinosaur;
                R_ground_v_pos <= C_V_OFFSET_dinosaur + C_IMAGE_HEIGHT_
dinosaur - C_IMAGE_HEIGHT_ground;
                R_cactus1_h_pos <= R_cactus1_h_pos;
                R_cactus1_v_pos <= C_V_OFFSET_dinosaur;
            end
        end
    endcase
end

//////////////////////////////////////////////////////////////////////
// 功能：产生行时序
//////////////////////////////////////////////////////////////////////
always@(posedge clk)
begin
    if(rst)
        h_cnt <= 11'd0;
    else if(h_cnt >= C_H_LINE_PERIOD - 1'd1)
        h_cnt <= 11'd0;
    else
        h_cnt <= h_cnt + 11'd1;
end
always@(posedge clk)
begin
    if(rst)
        h_de <= 1'b0;
    else if((h_cnt>=296)&&(h_cnt<=1319))
```

```
        h_de <= 1'b1;
    else
        h_de <= 1'b0;
end
//////////////////////////////////////////////////////////////////
// 功能：产生场时序
//////////////////////////////////////////////////////////////////
always@(posedge clk)
begin
    if(rst)
        v_cnt <= 11'd0;
    else if(h_cnt == C_H_LINE_PERIOD - 1'd1)
        begin
            if(v_cnt >= C_V_FRAME_PERIOD - 1'd1)
                v_cnt <= 11'd0;
            else
                v_cnt <= v_cnt + 11'd1;
        end
end
always@(posedge clk)
begin
    if(rst)
        v_de <= 1'b0;
    else if((v_cnt>=35)&&(v_cnt<=802))
        v_de <= 1'b1;
    else
        v_de <= 1'b0;
end

//////////////////////////////////////////////////////////////////
//功能：输出 hs 和 vs 信号
//////////////////////////////////////////////////////////////////
always@(posedge clk)
begin
    hs <= ((h_cnt > C_H_SYNC_PULSE)||rst)? 1'b1 : 1'b0 ;
    vs <= ((v_cnt > C_V_SYNC_PULSE)||rst)? 1'b1 : 1'b0 ;
end

assign W_active_flag =  (h_cnt >= (C_H_SYNC_PULSE + C_H_BACK_PORCH
                ))  &&
                        (h_cnt <= (C_H_SYNC_PULSE + C_H_BACK_PORCH + C_
H_ACTIVE_TIME  ))  &&
                        (v_cnt >= (C_V_SYNC_PULSE + C_V_BACK_PORCH
                ))  &&
                        (v_cnt <= (C_V_SYNC_PULSE + C_V_BACK_PORCH + C_
V_ACTIVE_TIME  ))  ;
assign vga_data = ((v_de==1)&&(h_de==1))? {vga_r,vga_g,vga_b} : 12'h0;
//assign vga_data = ((v_de==1)&&(h_de==1))? 12'hfff : 12'h0;

//////////////////////////////////////////////////////////////////
//功能：取跳跃信号 jumping
//////////////////////////////////////////////////////////////////
jitter_clr jitter_clr(
.clk(clk),
.button(btnc),
.button_clean(btnc_clean)
);

signal_edge signal_edge(
.clk(clk),
.button(btnc_clean),
.button_redge(jumping)
);
```

```
signal_edge(
.clk(clk),
.button(RESTART),
.button_redge(RESTART_edge)
);

blk_mem_gen_jump blk_mem_gen_jump (
  .clka(clk), // input clka
  .addra(R_rom_dinosaur_addr), // input [13 : 0] addra
  .douta(W_rom_dinosaur_jump) // output [11 : 0] douta
);

blk_mem_gen_leftup blk_mem_gen_leftup (
  .clka(clk), // input clka
  .addra(R_rom_dinosaur_addr), // input [13 : 0] addra
  .douta(W_rom_dinosaur_leftup) // output [11 : 0] douta
);

blk_mem_gen_rightup blk_mem_gen_rightup (
  .clka(clk), // input clka
  .addra(R_rom_dinosaur_addr), // input [13 : 0] addra
  .douta(W_rom_dinosaur_rightup) // output [11 : 0] douta
);
//leftup walk
blk_mem_gen_dead blk_mem_gen_dead (
  .clka(clk), // input clka
  .addra(R_rom_dinosaur_addr), // input [13 : 0] addra
  .douta(W_rom_dinosaur_dead) // output [11 : 0] douta
);

blk_mem_gen_ground blk_mem_gen_ground (
  .clka(clk), // input clka
  .addra(R_rom_ground_addr), // input [13 : 0] addra
  .douta(W_rom_data_ground) // output [11 : 0] douta
);

blk_mem_gen_cactus1 blk_mem_gen_cactus1 (
  .clka(clk), // input clka
  .addra(R_rom_cactus1_addr), // input [13 : 0] addra
  .douta(W_rom_data_cactus1) // output [11 : 0] douta
);

always@(posedge clk) //分时复用
begin
    case(walk_cnt[18:16])
        3'h0:   begin
                    an <= 8'b1111_1110;
                    data <= score % 14'd10;
                end
        3'h1:   begin
                    an <= 8'b1111_1101;
                    data <= (score % 14'd100) / 14'd10;
                end
        3'h2:   begin
                    an <= 8'b1111_1011;
                    data <= (score % 14'd1000) / 14'd100;
                end
        3'h3:   begin
                    an <= 8'b1111_0111;
                    data <= score / 14'd1000;
                end
        3'h4:   begin
                    an <= 8'b1110_1111;
```

```
                    data <= HIGH % 14'd10;
                    //data <= CURRENT_STATE[0];
                end
        3'h5:   begin
                    an <= 8'b1101_1111;
                    data <= (HIGH % 14'd100) / 14'd10;
                    //data <= CURRENT_STATE[1];
                end
        3'h6:   begin
                    an <= 8'b1011_1111;
                    data <= (HIGH % 14'd1000) / 14'd100;
                end
        3'h7:   begin
                    an <= 8'b0111_1111;
                    data <= HIGH / 14'd1000;
                end
        default:begin
                    an <= 8'b1111_1110;
                    data <= score % 14'd10;
                end
    endcase
end

dist_mem_gen_0 dist_mem_gen_0(
.a (data),
.spo (seg));
endmodule


module jitter_clr(
    input clk,
    input button,
    output button_clean
);

reg [17:0] cnt;
always@(posedge clk)
begin
    if(button==1'b0)
        cnt <= 18'h0;
    else if(cnt<18'b1111_1111_1111_1111_11)
        cnt <= cnt + 1'b1;
end
assign button_clean = cnt[17];
endmodule

module signal_edge(
    input clk,
    input button,
    output button_redge);

reg button_r1,button_r2;
always@(posedge clk)
    button_r1 <= button;
always@(posedge clk)
    button_r2 <= button_r1;
assign button_redge = button_r1 & (~button_r2);
endmodule
```

```
%matlab，以游戏结束图像 dead 为例
clear
clc

% 利用 imread 函数把图片转化为一个三维矩阵
image_array = imread('dead.jpg');

% 利用 size 函数把图片矩阵的三个维度大小计算出来
% 第一维为图片的高度，第二维为图片的宽度，第三维为图片的 RGB 分量
[height,width,z]=size(image_array);    % 128*128*3

 imshow(image_array); % 显示图片

red   = image_array(:,:,1); % 提取红色分量，数据类型为 uint8
green = image_array(:,:,2); % 提取绿色分量，数据类型为 uint8
blue  = image_array(:,:,3); % 提取蓝色分量，数据类型为 uint8

% 把上面得到了各个分量重组成一个 1 维矩阵，由于 reshape 函数重组矩阵的
% 时候是按照列进行重组的，所以重组前需要先把各个分量矩阵进行转置以
% 后在重组
% 利用 reshape 重组完毕以后，由于后面需要对数据拼接，所以为了避免溢出
% 这里把 uint8 类型的数据扩大为 uint32 类型
r = uint32(reshape(red'   , 1 ,height*width));
g = uint32(reshape(green' , 1 ,height*width));
b = uint32(reshape(blue'  , 1 ,height*width));

% 初始化要写入.coe 文件中的 RGB 颜色矩阵
rgb=zeros(1,height*width);

% 因为导入的图片是 24-bit 真彩色图片，每个像素占用 24-bit，其中 RGB 分别占用 8-
bit
% 而我这里需要的是 12-bit，其中 RGB 分别为 4-bit，所以需要在这里对
% 24-bit 的数据进行重组与拼接
% bitshift()函数的作用是对数据进行移位操作，其中第一个参数是要进行移位的数
据，第二个参数为负数表示向右移，为
% 正数表示向左移，更详细的用法直接在 Matlab 命令窗口输入 doc bitshift 进行查
看
% 所以这里对红色分量先右移 4 位取出高 4 位，然后左移 8 位作为 ROM 中 RGB 数据的第
11-bit 到第 8-bit
% 对绿色分量先右移 4 位取出高 4 位，然后左移 4 位作为 ROM 中 RGB 数据的第 7-bit 到
第 4-bit
% 对蓝色分量先右移 4 位取出高 4 位，然后左移 0 位作为 ROM 中 RGB 数据的第 3-bit 到
第 0-bit
for i = 1:height*width
```

```
    rgb(i) = bitshift(bitshift(r(i),-4),8) + bitshift(bitshift(g(i),-
4),4) + bitshift(bitshift(b(i),-4),0);
end

fid = fopen( 'dead.coe', 'w+' );

% .coe 文件的最前面一行必须为这个字符串，其中 16 表示 16 进制
fprintf( fid, 'memory_initialization_radix=16;\n');

% .coe 文件的第二行必须为这个字符串
fprintf( fid, 'memory_initialization_vector =\n');

% 把 rgb 数据的前 height*width-1  个数据写入.coe 文件中，每个数据之间用逗号隔
开
% fprintf( fid, '%x,\n',rgb(1:end-1));
for i = 1:height-1
    for j = (i-1)*width+1:i*width
        if (rgb(j)~=4095)
            fprintf( fid, '000,');
        else
            fprintf( fid, 'fff,');
        end
    end
    fprintf( fid, '\n');
end

for i = (height-1)*width+1:height*width-1
    if (rgb(j)~=4095)
        fprintf( fid, '000,');
    else
        fprintf( fid, 'fff,');
    end
end

% 把 rgb 数据的最后一个数据写入.coe 文件中，并用分号结尾
if (rgb(end)~=4095)
    fprintf( fid, '000,');
else
    fprintf( fid, 'fff;');
end

fclose( fid ); % 关闭文件指针
```