西安交通大學

XI'AN JIAOTONG UNIVERSITY

数字电子技术开放实验报告

题目 基于 FPGA 的 Dino 游戏设计

设计所在单位 西安交通大学电气学院

2018 年 12 月

报告题目：基于 FPGA 的 Dino 游戏设计

开源地址：https://github.com/AirWSW/FPGA-Dino-Game

摘要

本文介绍了基于 Basys 2 FPGA 开发板进行 Chrome 经典彩蛋 Dino 游戏的设计的方法，实现 VGA 图像输出、绘图窗口的控制输出、字符的 ASCII 控制输出、有限状态机实现的 Dino 基本游戏逻辑等多项功能。本文通过 ISE 软件辅助模拟设计并实际测试，得到了逻辑完整，画面精致完善，玩家体验良好的 Dino 游戏。

本文介绍的实验主要分为三大部分：第一部分为原理分析，通过解构原始游戏的动画效果以及游戏状态循环，将其游戏逻辑进行整理与抽象化；第二部分为系统模块化设计，该部分依据游戏设计所需原理，设计相应的 VGA 显示驱动与游戏逻辑模块；第三部分为验证并优化模型，该部分通过实际测试游戏进而寻找出有违逻辑的地方，进行进一步的修正。由于 VGA 游戏设计较为抽象，本文采用了仿真与编译测试结合的调试方法，通过实际游戏测试与仿真数据相对比加快 Debug 的速度，使最终编译出的游戏更具可玩性。

经过对游戏的不断测试与调整，最终完成了基于 Basys 2 FPGA 开发板的 Dino 游戏设计实验。

关键词：Basys2, VGA 图像输出, ASCII, 有限状态机, ISE

目录

一、实验内容与目标

1.1 实验项目内容

利用 Basys2 板本身资源以及 VGA 显示器输出所构成的实验系统，使用 Verilog 硬件描述语言实现 Dino 游戏的设计与开发。其包含 VGA 视频信号显示、图形控制输出、字符控制输出以及游戏本身逻辑多方面内容的相互协调。VGA 视频信号的输出显示原理可以参考 Xilinx 官方的帮助文档[1] [2] [3]，或者参考数字电子技术实验指导书中相应章节的内容[4]。

要求所设计开发出基于 FPGA 的 Dino 游戏具有原 Chrome 浏览器中彩蛋游戏 Dino 的基本游戏逻辑，即能够正常显示动画效果，能够根据输入对游戏主角 Dino 进行控制，能够统计当前阶段的得分以及能够输出统计字符。此外可以在此基础上设计出绘图区块显示以及文本段区块显示等更多功能。

该实验的难点主要在于理解 VGA 信号的产生原理，相对坐标定位与绘图区域控制，字符显示系统设计，游戏逻辑设计等方面。

1.2 实验预期目标

1、在学习硬件描述语言的基础上，完成 Dino 游戏的硬件设计方案和软件设计方案；分阶段完成模块设计和调试、系统设计和调试、最终完成系统设计要求。

2、撰写合格的实验报告、演示 Dino 游戏的功能和指标测试，并拍摄相应的短视频。

3、了解 VGA 显示扫描原理、VGA 时序等相关知识。

4、掌握基于 FPGA 的数字系开发的基本过程和技术。

基本功能要求：VGA 能够正常显示，能够控制 Dino 进行跳跃，能够使用数码管进行计分（基本游戏逻辑）。

提高功能要求：（1）输出更加精致的 Dino 恐龙与云朵形象，并添加相应的动画效果；（2）在屏幕上显示当前级别与得分等文字提示信息。

需要的主要仪器设备等实验条件：

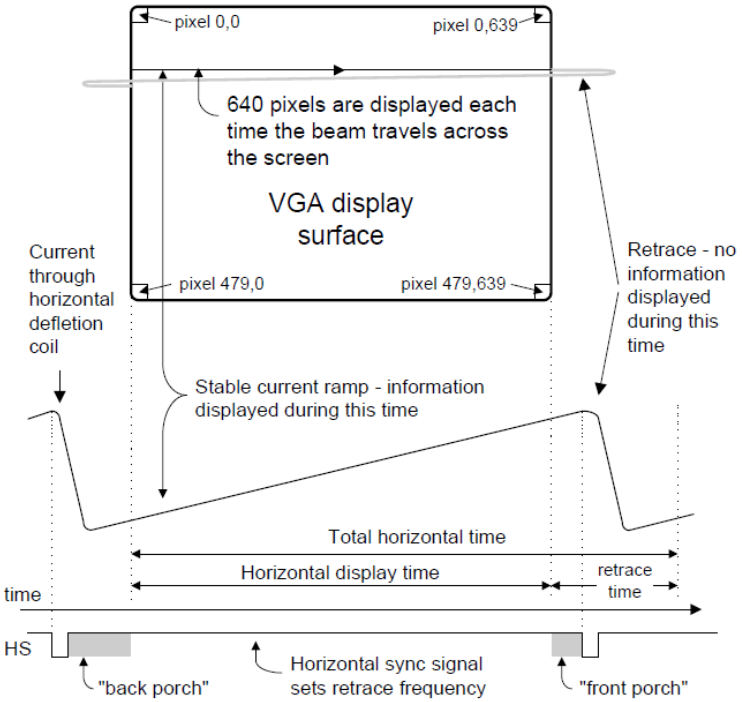
Basys2 板、VGA 连接线、VGA 显示器、ISE 及 Modelsim 仿真软件、PC 机等。

二、实验基本原理

2.1 VGA 显示原理

VGA（Video Graphics Array，视频图形阵列）是 IBM 于 1987 年提出的一个使用模拟信号的计算机显示标准。这个标准已对于现今的个人计算机市场已经十分过时。即使如此，VGA 仍然是最多制造商所共同支持的一个标准，个人计算机在加载自己的独特驱动程序之前，都必须支持 VGA 的标准。[5]

VGA 的时序主要由同步脉冲时间（Sync pulse time）、显示时间（Display time）、调整脉冲时间（VS pulse width）、显示前空余时间（VS front porch）以及显示后空余时间（VS back porch）来进行驱动。在设计 VGA 时序信号时往往将上述参数与需要显示的像素多



Symbol	Parameter	Vertical Sync			Horizontal Sync	
		Time	Clocks	Lines	Time	Clocks
T_S	Sync pulse time	16.7ms	416,800	521	32 us	800
T_{disp}	Display time	15.36ms	384,000	480	25.6 us	640
T_{pw}	VS pulse width	64 us	1,600	2	3.84 us	96
T_{fp}	VS front porch	320 us	8,000	10	640 ns	16
T_{bp}	VS back porch	928 us	23,200	29	1.92 us	48

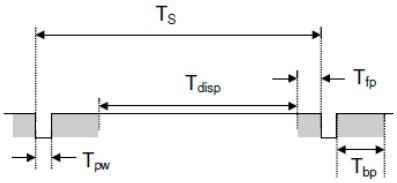


图 1 同步脉冲的时序图（输出模式 640x480）VgaRefComp.pdf

少以及时钟频率共同确定，一些常用的显示参数（时钟频率小于等于 50MHz）如表一。

表 1 常用的 VGA 信号显示参数[6]

时钟频率 分辨率刷新频率	行/列	脉冲时间	调整 脉冲时间	显示前 空余时间	显示时间	显示后 空余时间
25.175MHz	行	525	2	33	**480**	10
640X480@60Hz	列	800	96	48	**640**	16
31.5MHz	行	500	3	16	**480**	1
640X480@75Hz	列	840	64	120	**640**	16
50MHz	行	666	6	23	**600**	37
800X600@72Hz	列	1040	120	64	**800**	56
40MHz	行	628	4	23	**600**	1
800X600@60Hz	列	1056	128	88	**800**	40
49.5MHz	行	625	3	21	**600**	1
800X600@75Hz	列	1056	80	160	**800**	16

对于 VGA 图像的显示，需要在相应的显示时间内确定当前显示的位置以及与要显示的内容，在同步信号的周期内将其输出到 VGA 显示器。

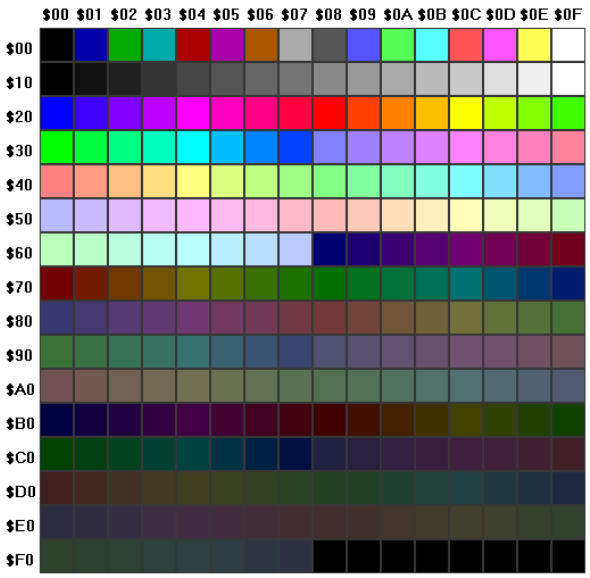


图 2 7 位 VGA 色彩信号对照表[5]

参考 Basys2 原理图可知，VGA 的色彩传输位数为 7 位数据，通过查询相应的 VGA 色彩对照表从而选择需要的颜色。

2.2 数码管显示原理

参考 Basys2 原理图可知，数码管的输入信号为共用端，因此采用循环刷新显示的方法进行数码管数字内容的显示。

2.3 图形驱动原理

图形驱动主要分为两个部分：首先是游戏场景定位问题，其次是游戏场景的渲染问题。

对于游戏场景定位，本文通过建立相对坐标系的方式进行图形

位置输出定位。

游戏中的场景与游戏角色可以通过简单的方块图形进行替代，但是为了增强游戏者的体验，将游戏进行区域判定处理，通过一个二维的 bitmap 加以 case 的形式进行查询，从而根据当前的游戏状态来对游戏场景进行精细化渲染。

对于复杂游戏角色的渲染，也使用了部分相对定位的方式。具体的方案如图 3。

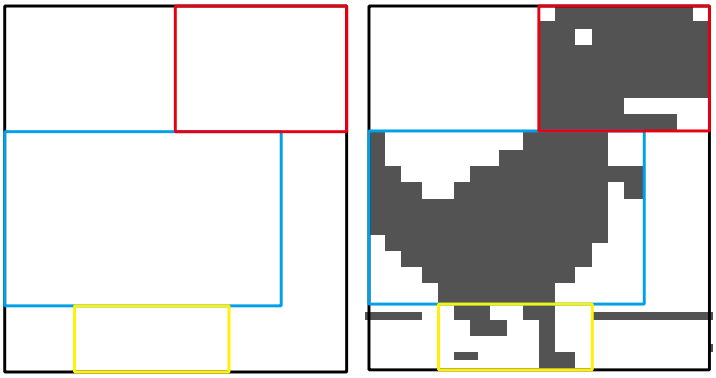


图 3 部分相对定位的方式

本文在游戏设计中将 Dino 分为三个部分，分别为红色区域、蓝色区域以及黄色区域。三个区域相对于黑色矩形的左上角进行定位，并分别进行填充绘制，通过状态转换显示为动态图形。

2.4 ASCII 字符显示驱动原理

ASCII 字符显示驱动的原理与图形显示驱动的原理较为相似。在屏幕绘图坐标系的基础上对需要字符输出的位置进行相对定位，确定字符输出区域。

本文根据 ASCII 表[7]的顺序将字符的形状存入常量中，然后通过二级转换的方式将输入的 ASCII 字符串进行转换，进而根据该区域通过循环输出不同位长 ASCII 编码的英文字符。该方式简化了英文字符输出编写时的查询复杂度，提高了开发效率。

图 4(a) 英文字符显示效果

图 4(b) 英文字符定位方式

2.5 游戏逻辑原理

本文中的 Dino 小游戏源自于 Google Chrome 中的断网彩蛋小游戏 Dino，该游戏通过键盘空格键控制小恐龙 Dino 进行跳跃来躲避前方的障碍。

经过分析得到，Dino 小恐龙一共有等待开始游戏，游戏中，跳跃以及死亡一共四种状态，并通过配合不同的键盘输入进行不同状态之间的转换。因此本文使用了一个四状态的有限状态机为主，进行游戏主逻辑的设计，状态转移图如图 5。

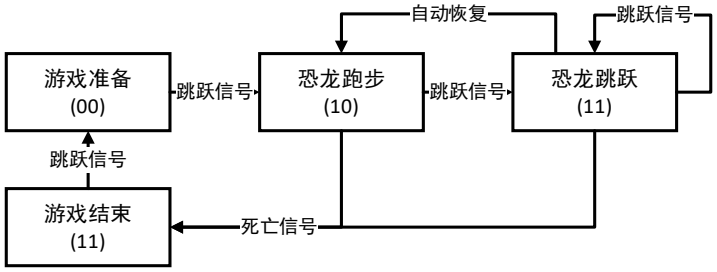


图 5 7 状态转移图

三、系统模块化程序设计

3.1 顶层模块设计

首先根据整体游戏需求确定顶层模块的接口。

经过对游戏整体设计的分析可知，对于顶层模块，其输入为主时钟信号 clk 以及分别对应重置游戏 clr、重新开始 restart、预留按键 key 和跳跃 jump 一共四个按键信号；输出则为七段 LED 灯的显示信号 dp/ an/ seg 以及 VGA 显示输出控制信号 hsync/ vsync/ red/ green/ blue。

通过该顶层模块能够实现所设计的 Dino 游戏的基本游戏逻辑以及 VGA 动画输出显示功能。其对应的引角图如图 6 所示：

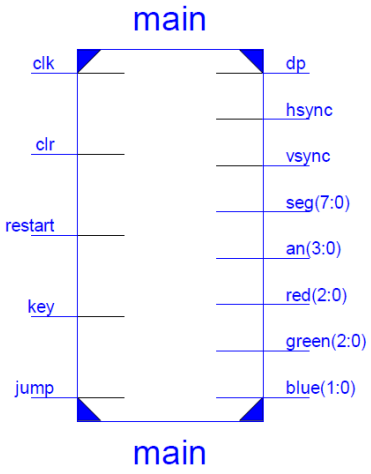


图 6 顶层模块设计接口图

其对应的引脚约束条件为：

```
// File: Basys2.ucf

// connect master clock
net clk loc = B8; // MCLK 50MHz clk

// connect keys
net clr loc = A7; // BTN3
net restart loc = M4; // BTN2
net key loc = C11; // BTN1
```

```
net jump loc = G12; // BTN0

// connect 7-segment display
net seg[6] loc = L14; // CA
net seg[5] loc = H12; // CB
net seg[4] loc = N14; // CC
net seg[3] loc = N11; // CD
net seg[2] loc = P12; // CE
net seg[1] loc = L13; // CF
net seg[0] loc = M12; // CG
net an[3] loc = K14; // AN3
net an[2] loc = M13; // AN2
net an[1] loc = J12; // AN1
net an[0] loc = F12; // AN0
net dp loc = N13; // DP

// connect VGA display
net red[0] loc = C14; // RED0
net red[1] loc = D13; // RED1
net red[2] loc = F13; // RED2
net green[0] loc = F14; // GRN0
net green[1] loc = G13; // GRN1
net green[2] loc = G14; // GRN2
net blue[0] loc = H13; // BLU0
net blue[1] loc = J13; // BLU1
net hsync loc = J14; // HSYNC
net vsync loc = K13; // VSYNC
```

3.2 时钟分频模块设计

为了得到能够使数码管显示频率 segclk 和 VGA 信号输出正常工作的频率 vgaclk 需要对主时钟信号进行分频处理。

对于时钟分频模块，其输入为主时钟信号 clk 以及分别对应重置游戏 clr 按键信号；输出为数码管的刷新显示频率 segclk 和 VGA 信号输出正常工作的频率 vgaclk。

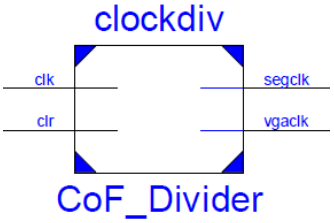


图 7 时钟分频模块设计接口图

其基本实现逻辑为：

首先通过一个计数器进行累加。

```
reg [14:0] q;

// Clock divider
always @(posedge clk or posedge clr) begin
    if (clr == 1) begin
        q <= 0;
    end
    else begin
        q <= q + 1;
    end
end
```

然后根据累加器第一位的电平信号，得到 25MHz 的 VGA 显示所需的信号频率。

```
// 50Mhz / 2^1 = 25MHz
assign vgaclk = q[0];
```

接着根据累加器第十五位的电平信号，得到 381.47Hz 的数码管显示所需的信号频率。

```
// 50Mhz / 2^15 = 381.47Hz
assign segclk = q[14];
```

3.3 数码管显示模块设计

对于七段 LED 显示模块，其输入为时钟信号 segclk 以及四位数

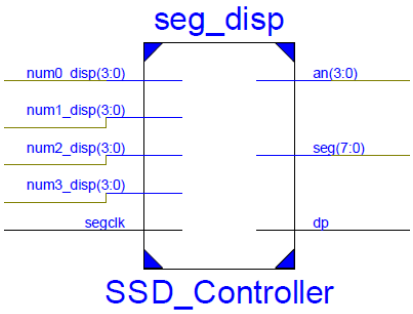


图 8 数码管显示模块设计接口图

码管需要显示的数字信号*_disp；输出信号为位使能信号 an、小数点使能信号 dp 以及七段管使能信号 seg。

其基本实现逻辑为：

首先根据 381.47Hz 时钟信号 segclk 上升沿进行操作。

```
always@(posedge segclk) begin
    ...
end
```

对四个数码管循环使能，并根据当前的数码管显示位置对其进行赋值操作。

```
an = 4'b1111;
select <= select + 1;
an[select] = 0;

case(select)
    2'b00:digit = num0_disp[3:0];
    2'b01:digit = num1_disp[3:0];
    2'b10:digit = num2_disp[3:0];
    2'b11:digit = num3_disp[3:0];
endcase
case(digit)
    'h0:seg = 7'b0000001; // disp 0
    'h1:seg = 7'b1001111; // disp 1
    'h2:seg = 7'b0010010; // disp 2
    'h3:seg = 7'b0000110; // disp 3
    'h4:seg = 7'b1001100; // disp 4
    'h5:seg = 7'b0100100; // disp 5
    'h6:seg = 7'b0100000; // disp 6
    'h7:seg = 7'b0001111; // disp 7
    'h8:seg = 7'b0000000; // disp 8
    'h9:seg = 7'b0000100; // disp 9
    'hA:seg = 7'b0001000; // disp A
    'hB:seg = 7'b1100000; // disp b
    'hC:seg = 7'b0110001; // disp C
    'hD:seg = 7'b1000010; // disp d
    'hE:seg = 7'b0110000; // disp E
    'hF:seg = 7'b0111000; // disp F
endcase
```

通过该七段 LED 显示模块能够实现所设计的根据输入四位信号，在数码管上输出数字的功能。

3.4 图形驱动模块设计

为了能够使 VGA 图像正常的显示，本文设计出针对于 Dino 游戏基本功能的图形显示驱动模块。

对于图形驱动模块，其输入为时钟信号 vgaclk、恐龙的位置 dino_pos、恐龙当前状态 dino_state、障碍物树的位置 tree_pos、游戏当前状态 game_state、当前游戏等级 level 以及四位数码管需要显示的数字信号*_disp；输出则为 VGA 显示输出控制信号 hsync/ vsync/ red/ green/ blue 以及恐龙当前存活状况 is_alive（预留信号暂时未使用）。

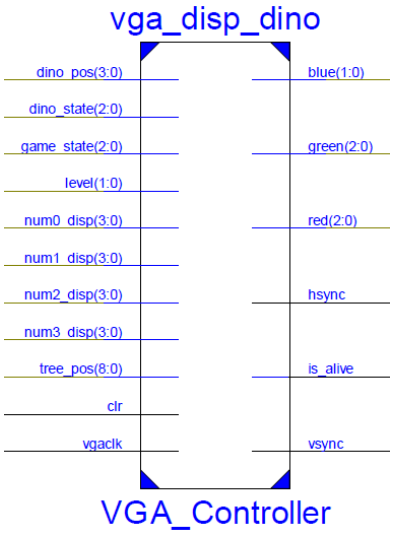


图 9 图形驱动模块设计接口图

其基本实现逻辑为：

首先确定场扫描参数以及确定动画图形显示常量。

```
// video structure constants 640*480@60Hz, 25.125MHz
parameter H_SYNC = 96; // hsync pulse length
parameter H_BACK = 48; // end of horizontal back porch
parameter H_DISP = 640;
parameter H_FRONT = 16; // beginning of horizontal front porch
parameter H_TOTAL = 800; // horizontal pixels per line
```

```
parameter V_SYNC = 2; // vsync pulse length
parameter V_BACK = 33; // end of vertical back porch
parameter V_DISP = 480;
parameter V_FRONT = 10; // beginning of vertical front porch
parameter V_TOTAL = 525; // vertical lines per frame

// char source
parameter char_line0 =
'h00000000C0000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000800;
...
parameter char_lineF =
'h00000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000;
parameter end_ascii = 'h5F;

//      0  1  2  3  4  5  6  7  8  9  A  B  C  D  E  F
// 0x20    !  "  #  $  %  &  '  (  )  *  +  ,  -  .  /
// 0x30  0  1  2  3  4  5  6  7  8  9  :  ;  <  =  >  ?
// 0x40  @  A  B  C  D  E  F  G  H  I  J  K  L  M  N  O
// 0x50  P  Q  R  S  T  U  V  W  X  Y  Z  [  \  ]  ^  _

// land and cloud source
parameter source_line0 =
'b00000000000000000000000011110000000000000000000011110000010000;
...
parameter source_lineF =
'hb51bffd0bdf74cb609ee8260f8de41a486f1fda459fa47c7ef0f50cc5f0c;

// dino source
parameter dino_line0 = 'b01111111101110011110000000001111100;
...
parameter dino_lineF = 'b11111111100000000000000000000000000;
```

然后根据 VGA 的相关参考文献完成 VGA 现实的主要驱动，确定有效的水平与垂直绘图区域(Safe area)的位置坐标 Xpos、Ypos 以及整体图像的水平位移（为了避免 9 位位移表示 512 小于屏幕宽度 640 时造成的图像偏移）。

```
// Horizontal & vertical counters
always @(posedge vgaclk or posedge clr) begin
    // reset condition
    if (clr == 1) begin
        intHcnt <= 0;
        intVcnt <= 0;
    end
    else begin
        // keep counting until the end of the line
        if (intHcnt < H_TOTAL)
            intHcnt <= intHcnt + 1;
        else begin
```

```
            // keep counting until the end of the row
            intHcnt <= 0;
            if (intVcnt < V_TOTAL)
                intVcnt <= intVcnt + 1;
            else
                intVcnt <= 0;
        end
    end
end

// generate sync pulses
assign hsync = (intHcnt < H_SYNC) ? 0 : 1;
assign vsync = (intVcnt < V_SYNC) ? 0 : 1;

// define the safe area
assign Xpos = intHcnt - H_SYNC - H_BACK;
assign Ypos = intVcnt - V_SYNC - V_BACK;

// define the average x move
parameter move_x = 50;
```

接着通过内置的计时器实现游戏中欢迎界面字符滚动显示的 ASCII 字符转换控制位 text、。

```
// Temp Clock divider
reg [31:0] q;
always @(posedge vgaclk or posedge clr) begin
    if (clr == 1) begin
        q <= 0;
    end
    else begin
        q <= q + 'b1000;
    end
end

// text for welcome screen last char's ASCII
reg [7:0] text = 8'h20;
always @(posedge q[22]) begin
    text <= q[29] ? (text<8'h5f ? text + 1 : 8'h20) : 8'h21;
end
```

对于字符 char 绘图定位区域、陆地 land 绘图定位区域、云 cloud 绘图定位区域、树 tree 绘图定位区域以及恐龙 dino 绘图定位区域，本文使用统一的定位方式通过定义相对的水平和垂直坐标、绘图区域的尺寸以及当前绘图的状态来对图形的输出进行控制。特别的，对于字符绘图区域另外定义了行数、列数以及字符串变量。

```
// char ============================================================
parameter char_lines = 16;
parameter char_rows = 1;
parameter char_scale = 2;
reg [127:0] char_disp;

reg [9:0] char_x;
reg [9:0] char_y;
reg [7:0] char_index;

assign charXpos = (Xpos - char_x)/char_scale;
assign charYpos = (Ypos - char_y)/char_scale;
// char end ========================================================

// land ============================================================
parameter land_x = 0;
parameter land_y = 376;
parameter land_scale = 2;

wire [7:0] land_state;

assign land_state = q[29:22];

assign landXpos = (Xpos - land_x)/land_scale;
assign landYpos = (Ypos - land_y)/land_scale;
// land end ========================================================

// cloud ===========================================================
parameter cloud_y = 150;
parameter cloud_scale = 2;
wire [8:0] cloud_x;
wire [2:0] cloud_state;

assign cloud_x = ~q[31:23];
assign cloud_state = {0,q[23],q[25]};

assign cloudXpos = (Xpos - cloud_x - move_x/2)/cloud_scale;
assign cloudYpos = (Ypos - cloud_y)/cloud_scale;
// cloud end =======================================================

// tree ============================================================
parameter tree_y = 300;
parameter tree_scale = 4;

wire [8:0] tree_x;
wire [2:0] tree_state;

assign tree_x = ~tree_pos;
assign tree_state = {0,q[23],q[25]};

assign treeXpos = (Xpos - tree_x - move_x)/tree_scale;
assign treeYpos = (Ypos - tree_y)/tree_scale;
// tree end ========================================================

// dino ============================================================
parameter dino_x = 50;
parameter dino_y = 300;
```

```
parameter dino_scale = 4;

// wire [2:0] dino_state;

// assign dino_state = {0,q[24],q[26]};

assign dinoXpos = (Xpos - dino_x - move_x)/dino_scale;
assign dinoYpos = (Ypos - dino_y + dino_pos * 10)/dino_scale; // Ypos
add
// dino end ============================================================
```

接着根据时钟信号 vgaclk 上升沿进行操作，并将所得到的颜色值进行输出。

```
always @(posedge vgaclk) begin
    ...
end

assign {red[2:0], green[2:0], blue[1:0]} = {vga_rgb[7:5], vga_rgb[4:2],
vga_rgb[1:0]};
```

对于每个时钟信号 vgaclk 上升沿来说，首先对文字区域进行控制输出，根据当前的游戏状态 game_state 来输出相应的欢迎信息，游戏中信息以及结束游戏的相关信息。char_disp 为 ASCII 字符串。

```
case(game_state[1:0])
    // welcome screen
    2'b00:begin char_disp <=
{112'h57_20_45_20_4c_20_43_20_4f_20_4d_20_45_20,text,8'h20}; char_x <=
10'd192; char_y <= 10'd200; end
    // play screen
    2'b01:begin char_disp <=
{28'h4c_56_20_3,2'b00,level,64'h20_20_53_43_4f_52_45_20,4'h3,num3_disp,
4'h3,num2_disp,4'h3,num1_disp,4'h3,num0_disp}; char_x <= 10'd360;
char_y <= 10'd100; end
    // jump screen
    2'b11:begin char_disp <=
{28'h4c_56_20_3,2'b00,level,64'h20_20_53_43_4f_52_45_20,4'h3,num3_disp,
4'h3,num2_disp,4'h3,num1_disp,4'h3,num0_disp}; char_x <= 10'd360;
char_y <= 10'd100; end
    // game over screen
    2'b10:begin case(q[30:28])
            'b000:begin char_disp <=
128'h47_20_41_20_4d_20_45_20_20_4f_20_56_20_45_20_52; char_x <=
10'd192; char_y <= 10'd200; end
            'b001:begin char_disp <=
{28'h4c_56_20_3,2'b00,level,64'h20_20_53_43_4f_52_45_20,4'h3,num3_disp,
4'h3,num2_disp,4'h3,num1_disp,4'h3,num0_disp}; char_x <= 10'd192;
char_y <= 10'd200; end
```

```
            'b010:begin char_disp <=
128'h47_20_41_20_4d_20_45_20_20_4f_20_56_20_45_20_52; char_x <=
10'd192; char_y <= 10'd200; end
            'b011:begin char_disp <=
128'h20_20_20_4d_49_4e_47_52_55_49_20_59_55_20_20_20; char_x <=
10'd192; char_y <= 10'd200; end
            'b100:begin char_disp <=
128'h47_20_41_20_4d_20_45_20_20_4f_20_56_20_45_20_52; char_x <=
10'd192; char_y <= 10'd200; end
            'b101:begin char_disp <=
{28'h4c_56_20_3,2'b00,level,64'h20_20_53_43_4f_52_45_20,4'h3,num3_disp,
4'h3,num2_disp,4'h3,num1_disp,4'h3,num0_disp}; char_x <= 10'd192;
char_y <= 10'd200; end
            'b110:begin char_disp <=
128'h47_20_41_20_4d_20_45_20_20_4f_20_56_20_45_20_52; char_x <=
10'd192; char_y <= 10'd200; end
            'b111:begin char_disp <=
128'h20_20_20_53_48_49_57_45_49_20_57_41_4e_47_20_20; char_x <=
10'd184; char_y <= 10'd200; end
        endcase end
endcase
```

接着对图像区域进行控制输出，根据当前的每个图象的位置以及相应状态来输出相应的图像信息，此外通过判断根据白天黑夜输出不同的颜色效果。

```
if (Xpos >= 0 && Xpos < (H_DISP - 0) && Ypos >= 0 && Ypos < (V_DISP -
0)) begin
    // draw char
    if (game_state[2]) vga_rgb <= 8'b000_000_00; else vga_rgb <=
8'b111_111_11;
    if (Xpos >= char_x && Xpos < (char_x + 8*char_scale*char_lines) &&
Ypos >= char_y && Ypos < (char_y + 16*char_scale*char_rows)) begin
        // ASCII to parameter
        case(charXpos[6:3])
            ...
        Endcase
        // parameter to display
        case(charYpos[3:0])
            ...
        endcase
    end

    // draw land
    if (Xpos >= land_x && Xpos < (land_x + 320*land_scale) && Ypos >=
land_y && Ypos < (land_y + 30*land_scale)) begin
        ...
        end
    end

    // draw cloud
```

```
    if (Xpos >= (cloud_x + move_x/2) && Xpos < (cloud_x + 44*cloud_scale
+ move_x/2) && Ypos >= cloud_y && Ypos < (cloud_y + 12*cloud_scale))
begin
        ...
    end

    // draw dino
    if (Xpos >= (dino_x + move_x) && Xpos < (dino_x + 20*dino_scale +
move_x) && (Ypos >= dino_y - dino_pos * 10) && Ypos < (dino_y +
22*dino_scale - dino_pos * 10)) begin // Ypos minus
        if (dinoXpos >= 10 && dinoXpos < 20 && dinoYpos >= 0 && dinoYpos
< 8) begin
        ...
        end
        else if (dinoXpos >= 0 && dinoXpos < 16 && dinoYpos >= 8 &&
dinoYpos < 18) begin
        ...
        end
        else if (dinoXpos >= 5 && dinoXpos < 13 && dinoYpos >= 18 &&
dinoYpos < 22) begin
            temp <= dinoYpos - 18;
        ...
        end
    end

    // draw tree
    if (Xpos >= (tree_x + move_x) && Xpos < (tree_x + 6*tree_scale +
move_x) && Ypos >= tree_y && Ypos < (tree_y + 22*tree_scale)) begin
        ...
    end

end
else vga_rgb <= 8'b000_000_00;
```

经过仿真实验以及实际编译调试，该模块能够根据输入的数据输出期望的图像。

3.5 游戏主逻辑模块设计

对于游戏主逻辑模块，其输入为主时钟信号 clk、VGA 时钟信号 vgaclk、恐龙存活状况 is_alive（预留暂未使用）以及分别对应重置游戏 clr、重新开始 restart、预留按键 key 和跳跃 jump 一共四个按键信号；输出信号恐龙的位置 dino_pos、恐龙当前状态 dino_state、

障碍物树的位置 tree_pos、游戏当前状态 game_state、当前游戏等级 level 以及四位数码管需要显示的数字信号*_disp；

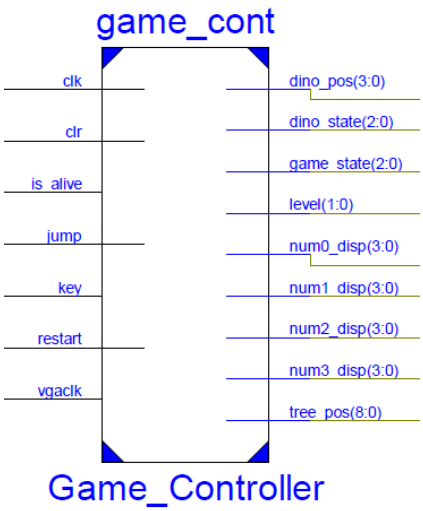


图 10 游戏主逻辑模块设计接口图

其基本实现逻辑为：

通过四个按键信号分别完成重置游戏 clr、重新开始 restart、预留按键 key 和跳跃 jump 的功能。通过使用标志字 alex 进行状态转换延时，实现消除按键的抖动输入。其 Verilog 代码片段如下：

```
if(clr) aaa_delay_count <= aaa_delay_count + 1; else aaa_delay_count <=
0;
if(clr & aaa_delay_count=='b1111010000100100000) begin alex[0] <= 1;
end // 10ms
if(clr==0) alex[0] <= 0;
alex[1] <= alex_wire[0]; // clear and reset all
if(restart) bbb_delay_count <= bbb_delay_count + 1; else
bbb_delay_count <= 0;
if(restart & bbb_delay_count=='b1111010000100100000) begin alex[2] <=
1; end // 10ms
if(restart==0) alex[2] <= 0;
alex[3] <= alex_wire[0]; // restart game
if(key) ccc_delay_count <= ccc_delay_count + 1; else ccc_delay_count <=
0;
if(key & ccc_delay_count=='b1111010000100100000) begin alex[4] <= 1;
end // 10ms
if(key==0) alex[4] <= 0;
alex[5] <= alex_wire[0]; // DEBUG ONLY or KEY
if(jump) ddd_delay_count <= ddd_delay_count + 1; else ddd_delay_count
<= 0;
```

```
if(jump & ddd_delay_count=='b1111010000100100000) begin alex[6] <= 1;
end // 10ms
if(jump==0) alex[6] <= 0;
alex[7] <= alex_wire[6]; // jump control button
```

重置游戏 clr 按下时进行重置游戏的操作。

```
if (alex_wire[1:0] == 2'b01 | alex_wire[3:2] == 2'b01) begin
    p <= 0;
    q <= 0;
    game_state <= 'b000;
    level <= 'b00;
    num3_disp_reg <= 'b0000;
    num2_disp_reg <= 'b0000;
    num1_disp_reg <= 'b0000;
    num0_disp_reg <= 'b0000;
end
```

游戏相关的有限状态机的 Verilog 代码如下：

```
case({alex_wire[7:6],game_state[1:0]})
    // welcome game
    'b0000,'b1000,'b1100:begin dino_state_reg <= 'b100; dino_pos_reg <=
'b0000; tree_pos_reg <= 'b000000000;
    num3_disp_reg <= 'b0000; num2_disp_reg <= 'b0000; num1_disp_reg <=
'b0000; num0_disp_reg <= 'b0000; end
    'b0100:begin game_state[1:0] <= 2'b01; p <= 0; q <= 0; end

    // start game
    'b0001,'b1001,'b1101:begin dino_state_reg[2] <= 0; dino_state_reg[1]
<= p[26]; p <= p + 'b1000 + level * 2; dino_pos_reg <= 0; tree_pos_reg
<= p[28:20];
        if(tree_pos_reg > 400 && tree_pos_reg < 470) begin
game_state[1:0] <= 2'b10; end
        end
    'b0101:begin game_state[1:0] <= 2'b11; end

    // jump action
    'b0011,'b1011,'b1111:begin dino_state_reg <= 'b100; p <= p + 'b1000
+ level * 2; q <= q + 'b100000; dino_pos_reg <= q[29] ? ~q[28:25] :
q[28:25]; tree_pos_reg <= p[28:20];
        if(q[30]) begin game_state[1:0] <= 2'b01; q <= 0; end
        if(tree_pos_reg > 400 && tree_pos_reg < 470 && dino_pos_reg < 8)
begin game_state[1:0] <= 2'b10; end
        end
    'b0111:begin game_state[1:0] <= 2'b11; end

    // game over
    'b0010,'b1010,'b1110:begin dino_state_reg[0] <= 'b1; end
    'b0110:begin p <= 0; q <= 0; game_state[1:0] <= 2'b00; end
endcase

game_state[2] <= p[32];
if (level != 2'b11) level <= p[34:33];
```

根据跳跃过的树来定义游戏的得分，通过十进制的累加器来实现，其每次的得分会根据当前的游戏等级进行加权处理。

```
if(q[30]) begin
    num1_disp_reg <= num1_disp_reg + 1 * (level + 1);
end

if(num0_disp_reg >= 'b1010) begin
    num0_disp_reg <= num0_disp_reg - 'b1010;
    num1_disp_reg <= num1_disp_reg + 1;
end
if(num1_disp_reg >= 'b1010) begin
    num1_disp_reg <= num1_disp_reg - 'b1010;
    num2_disp_reg <= num2_disp_reg + 1;
end
if(num2_disp_reg >= 'b1010) begin
    num2_disp_reg <= num2_disp_reg - 'b1010;
    num3_disp_reg <= num3_disp_reg + 1;
end
if(num3_disp_reg >= 'b1010) begin
    num3_disp_reg <= num3_disp_reg - 'b1010;
end
```

经过仿真实验以及实际编译调试，该模块能够实现基本的游戏逻辑控制操作。

3.6 模块综合与 RTL 级顶层逻辑图

将上述的所有模块按照连接关系进行综合得到基于 FPGA 的 Dino 游戏，其 RTL 级顶层逻辑图如图 11 所示。

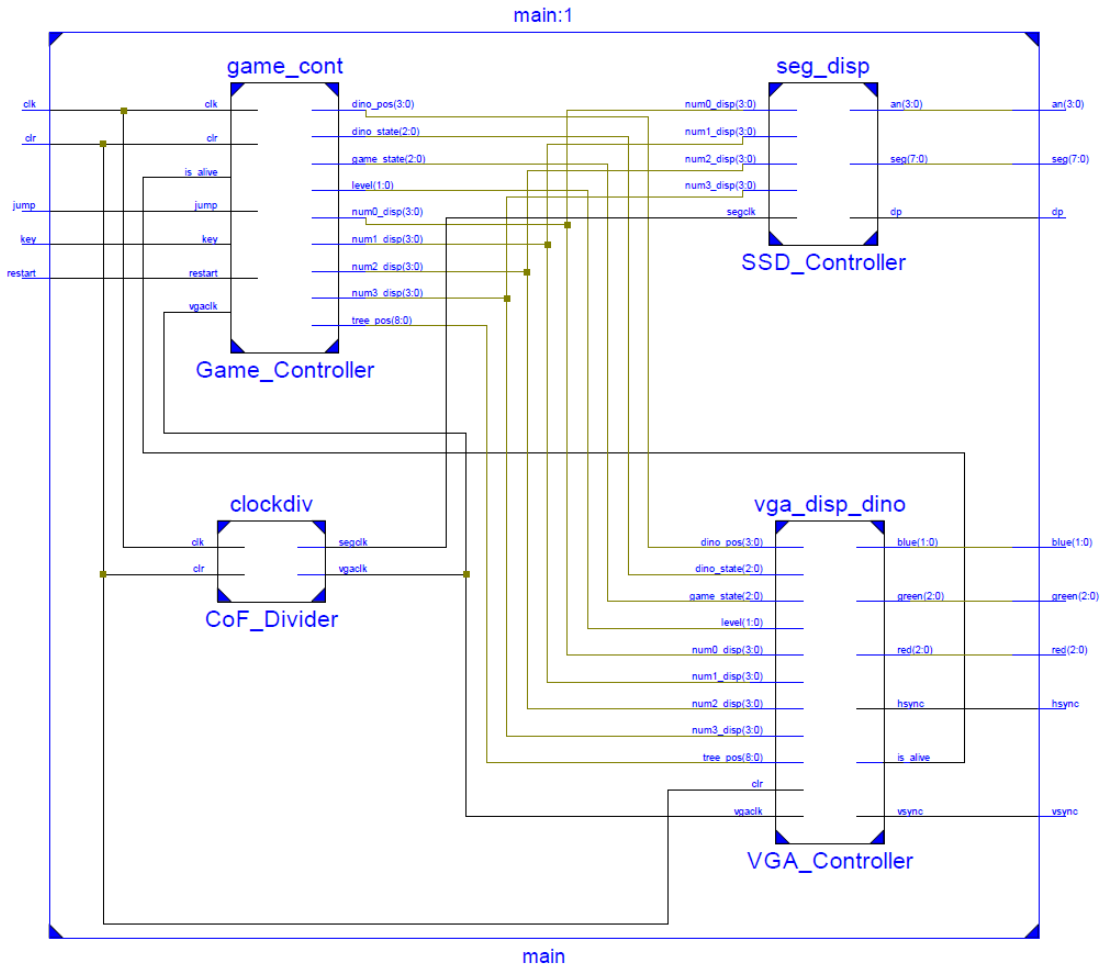


图 11 RTL 级顶层逻辑图

四、系统仿真与实验验证

4.1 系统仿真

4.1.1 延时按键防抖动仿真

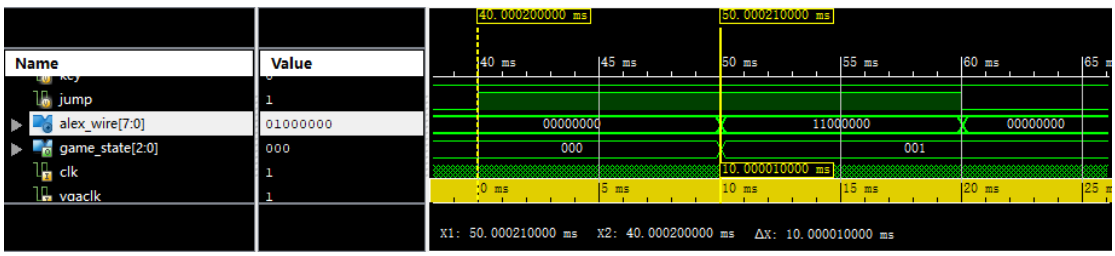


图 12 延时按键防抖动仿真结果

由图可知，当按键 jump 按下后经过 10ms，状态位 alex 由 00 变

为 01，在该时钟周期内进行相应的操作，下一个时钟周期状态位变为 11 以确保只运行一次操作。

4.1.2 图形输出场同步时钟信号仿真

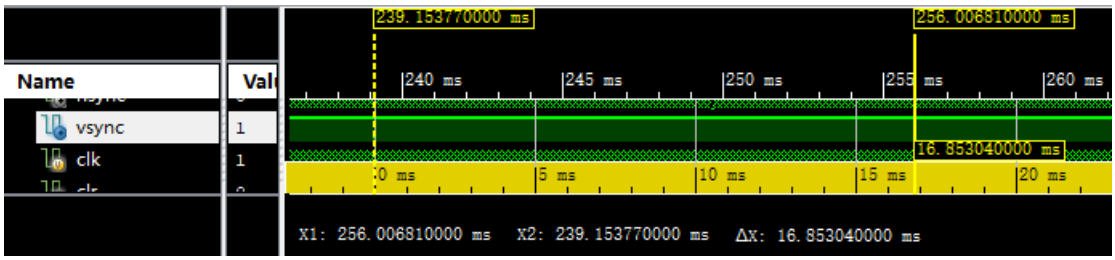


图 13 图形输出场同步时钟信号仿真结果

由仿真结果可知图形输出场同步时钟信号 vsync 频率为

$$\frac{1}{16.85\text{ms}} \approx 59.35\text{Hz} \approx 60\text{Hz}$$

与显示器识别的频率一致。

4.1.3 游戏逻辑控制信号仿真

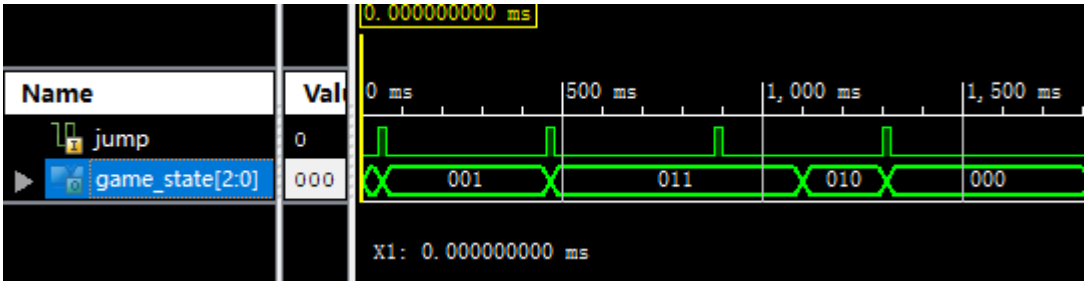


图 14 游戏逻辑控制信号仿真结果(game_state 开始为 000)

开始游戏状态为游戏欢迎 00，按下 jump 键后游戏状态变为游戏中 01，此时再次按下 jump 键游戏状态转变为恐龙跳跃 11，一定时间后再次变回 01 状态，如果在该状态下按 jump 状态将不会发生

转变。如果发生碰撞状态将直接变为游戏结束 10，此时需要按下 jump 键回到状态 00 重新进行游戏。

仿真的 Verilog 代码片段为：

```
initial begin
    // Initialize Inputs
    clk = 0;
    clr = 0;
    restart = 0;
    key = 0;
    jump = 0;

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

    // Add stimulus here
    clr = 0;#100;
    clr = 1;#20000000;
    clr = 0;#20000000;
    jump = 1;#20000000;
    jump = 0;#400000000;
    jump = 1;#20000000;
    jump = 0;#400000000;
    jump = 1;#20000000;
    jump = 0;#400000000;
    jump = 1;#20000000;
    jump = 0;
end

always#10 clk = ~clk;
```

4.2 实验结果与分析

结合实验仿真内容与不断地调试，使得最终的游戏运行效果如图 15 所示。该游戏具有四个主要场景，主要分别为

游戏欢迎界面，如图 15(a)，中间由字符 WELCOME!构成，其中“!”字符有随机字符滚动显示的动画效果。

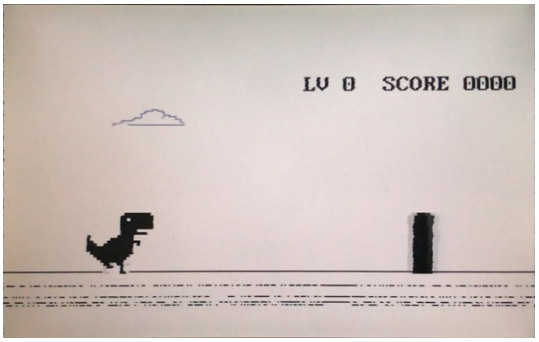
游戏中界面，如图 15(b) (d)，Dino 恐龙将会一直向前跑，右上

角会显示当前的游戏级别以及当前的得分情况，游戏得分会根据当前游戏的等级倍乘，游戏会交替白天黑夜进行。

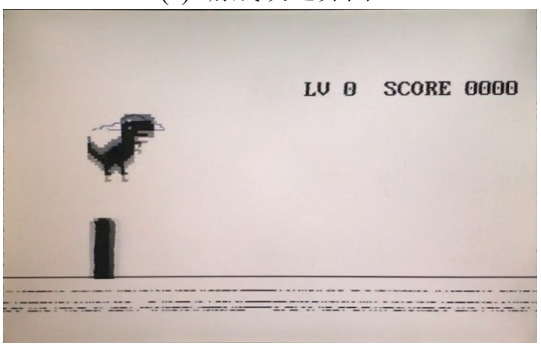
恐龙跳跃界面，如图 15(c) (e)，Dino 恐龙将会向前跳跃，越过前面的障碍，右上角显示当前的游戏级别以及当前的得分情况。

游戏结束界面，如图 15(f)，画面保持在恐龙发生碰撞的那一帧动画，此外中间由字符 GAME OVER 构成，此外会循环显示最终的游戏得分信息以及开发者的姓名。

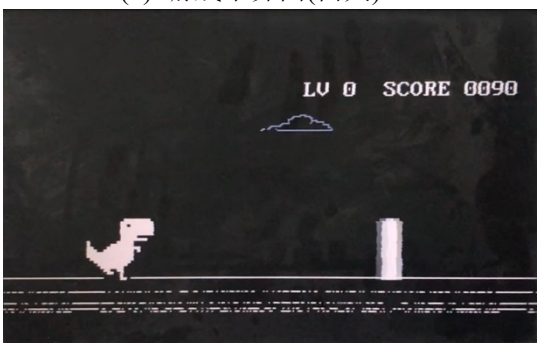
(a) 游戏欢迎界面



(b) 游戏中界面(白天)

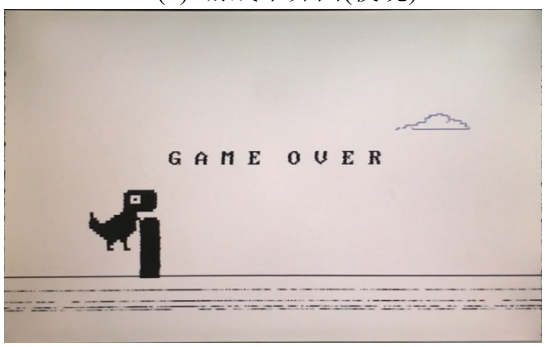


(c) 恐龙跳跃界面(白天)



(d) 游戏中界面(夜晚)

(e) 恐龙跳跃界面(夜晚)



(f) 游戏结束界面

图 15(续) 部分游戏界面

4.3 实验结论

本文能够实现 Dino 游戏的基本游戏操作以及游戏逻辑功能。本文在此基础上对 Dino 游戏加以完善，加入按键去抖的操作、优化了游戏界面的输出显示以及完善了游戏内的信息显示功能。游戏整体更具可玩性。

4.4 设计过程中出现的问题及其解决办法

1、资源占用过大，导致编译时 map 占用过大无法通过。

本文通过调整常量储存的模型，调整优化资源与功能所占比例，将部分未使用的 ACSII 字符的模型进行删减。本文采用的 ASCII 译码模型仅包含编码表中 0x20~0x5F 中的所有字符，据此平衡资源占用与实现功能之间的关系。

2、开始时 VGA 输出到显示器无信号。

开始时 VGA 输出到显示器无信号，VGA 显示器无法识别到视频信号。本文通过实际的调试，更换了多台显示器进行测试，并通过实验仿真确定场同步信号的频率。由于开始时分频出现问题，时钟频率不正确，导致了信号不稳定显示器无法识别的现象。除此之外也可以使用示波器或者仿真观察场扫描信号的频率。

3、VGA 驱动颜色表出现问题。

部分 VGA 显示时出现部分画面变暗的情况，出现色彩带的现象，经过检查是由于部分代码判断的问题所致，进行修改后可以正常显示颜色。

4、部分 warning 和 error 的状况，参考参考文献[8]进行解决。

五、参考文献

[1] Basys 2™ FPGA Board Reference Manual. basys2_rm.pdf

[2] Basys 2™ FPGA Board Schematic. basys2_sch.pdf

[3] VGA Refference Componentt. VgaRefComp.pdf

[4] 数字电子技术与接口技术实验教程

[5] Video Graphics Array - Wikipedia

https://en.wikipedia.org/wiki/Video_Graphics_Array

[6] 常用的 VGA 信号显示参数

https://www.cnblogs.com/fhyfhy/p/5381233.html

[7] ASCII - Wikipedia

https://en.wikipedia.org/wiki/ASCII

[8] FPGA 常用调试技术

https://wenku.baidu.com/view/344f93a2a1116c175f0e7cd184254b35eefd1a95.html

[9] 菜鸟 0 号.FPGA 成神之路 ----- 菜鸟的武器（ise 开发使用）

https://www.cnblogs.com/FPGAroom/p/3740862.html