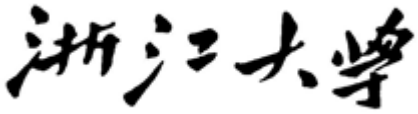


课程设计报告

中文题目： 基于数字系统的 Demon 跳跃障碍游戏

英文题目： Demon based on the Digital System

姓名/学号： 董泰佑/3140104431

指导教师： 施青松

参加成员：

专业类别： 计算机科学与技术

所在学院： 计算机科学与技术学院

联系电话： 17816878432/599335

论文提交日期 2015 年 1 月 12 日

摘要

基于数字系统的Demon小游戏是指玩家通过对button的操作实现对Demon的控制，跳跃障碍的游戏。通过调用VGA显示屏来显示游戏界面。游戏操作简单快捷，容易掌握，游戏画面精细生动。工程使用时序状态机模型，采用层次化、模块化方法，使用Verilog HDL语言编写程序，基于Xilinx Spartan-3实验平台实现工程。

本设计能够实现简单的人机交互功能，并且游戏难度会随着时间的推移而增加，来提高游戏的趣味性和挑战性。

关键词：VGA显示 FPGA Verilog语言 Demon 跳跃障碍

目录

目录

图目录

第1章 绪论

1.1 Demon 游戏设计背景

通过一个学期对数字逻辑课程的学习，对逻辑电路的设计和实现有了初步的了解，希望可以通过自己对所学知识的应用，设计出一个基于 Xilinx Spartan-3 实验平台的趣味小游戏。同时加深自己对所学知识的理解和掌握，让自己所学的知识从书面水平上升到理论高度，在实践中学习和进步。

1.2 国内外现况分析

Chrome 浏览器已经开发过类似的小游戏，有兴趣者可以上网查阅或者在 chrome 浏览器上实践。这种趣味小游戏深受广大学生和办公室工作者们的喜爱，在工作或者学习小憩时可以放松心情，愉悦身心，市场需求量仍很丰富。此外，本类游戏可以提升的地方还有很多，仍具有很大的发展空间。

1.3 主要内容和难点

实验内容：

本实验调用 VGA 显示模块，时序显示 Demon 在屏幕上的动作和位置。

Demon 会根据用户的按键操作实现跳跃、暂停等操作。如果 Demon 和障碍物相撞，游戏就会结束并且屏幕上显示“game over”的字样，这时按下复位键游戏就会重新开始。

实现功能：

Demon 可以根据用户的指令实现跳跃动作。Demon 的动态动作可以时序交替调用两个 ROM 实现。工程中背景和障碍物的时序显示实现均由调用 ROM 实现。背景和障碍物的时钟控制均由自己生成的时钟模块控制。这个时钟模块的频率会根据用户时间的变化而改变——随着用户时间的增加而不断增大，进而增加游戏难度。背景和障碍物的随机生成设置。用户在游戏过程中可以打开开关 SW[7]来控制游戏的启动和暂停，button[1]是游戏的复位键，button[2]是游戏的操作键，来控制小恐龙的跳跃操作。

重点难点分析：

本游戏的难点设计主要有如下几个方面：

1. Demon 跑动动作的时序显示。如何交替显示两张图片实现小恐龙的跑动动作。
2. Demon 跳跃时的速度控制，如何实现和现实中的情景一样的由重力加速度控制的速度变化。
3. 背景的时序显示。怎样使背景显示出动态的效果。
4. 游戏难度的增加。如何控制时钟信号，使背景和障碍物的移动速度随着时间的增加而增加
5. Demon 与障碍物的碰撞检测的控制。如何判断 Demon 与障碍物碰撞在一起。
6. 实验板的空间限制。如何合理的利用空间，可以高效的显示出游戏的效果图片。

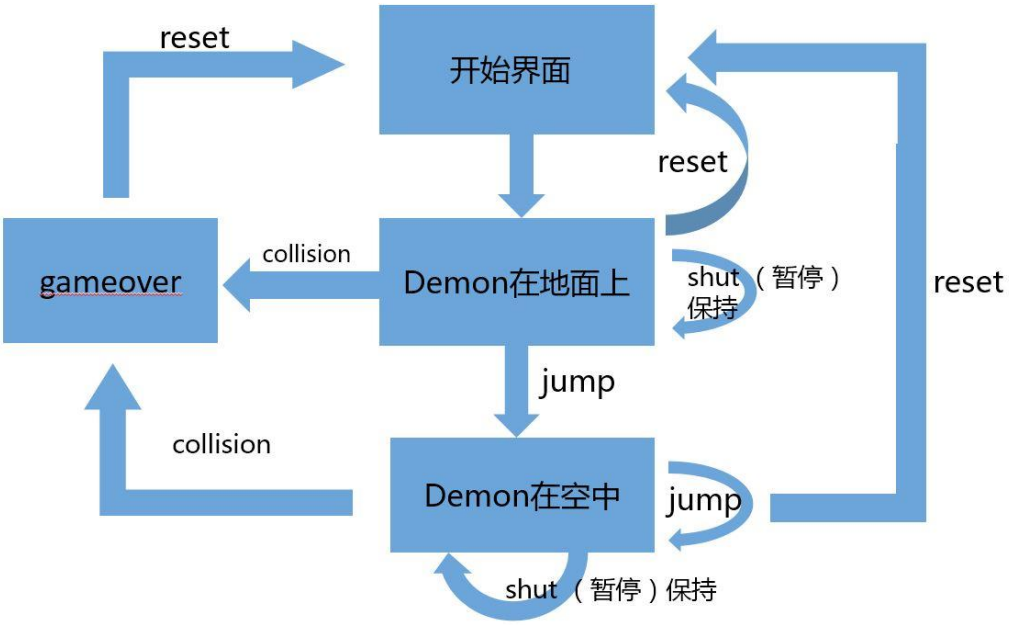
第2章 Demon 跳跃障碍游戏设计原理

2.1 Demon 设计相关内容

本次实验主要通过时钟模块的分频电路和 ROM 的调用来实现。

1. 时钟分频模块：利用系统时钟进行分频得到实验需要的时钟信号。
2. CORE ROM：通过 ROM 的调用，储存相应的像素点和 RGB 的值节省实验板的空间的占用，高效的实现实验的效果。
3. VGA 模块：调用 VGA 在屏幕上显示相应的游戏效果。由 VGA 模块产生同步信号与实验板的参数和频率同步显示。
4. 计数器模块：通过计数器实现游戏难度的增加和控制。

2.2 Demon 设计方案



图表 1 工程状态图

界面的显示由不同的信号控制，所以要根据不同的状态来显示不同的界面。

Reset 为复位操作，即游戏重新开始。Collision 为结束判断，如果 Demon 与障碍物碰撞，则游戏结束。Jump 为 Demon 跳跃操作，按下 button[0]，Demon 跳起。游戏的难度会随着用户时间的增加而增加，表现出来的就是背景和障碍物移动速度的增加。

由于板的空间资源的限制，所以关于图片显示的数据均储存在 ROM 里面，使工程实现的速度大大加快。

2.3 Demon 硬件设计

硬件方面调用 VGA 显示器显示游戏和人机交互界面。按键操作则直接在 Spartan-3 上进行操作。因为游戏操作简单，实现方便快捷，所以没有调用 ps2 键盘实现。

第3章 Demon 设计实现

3.1 实现方法

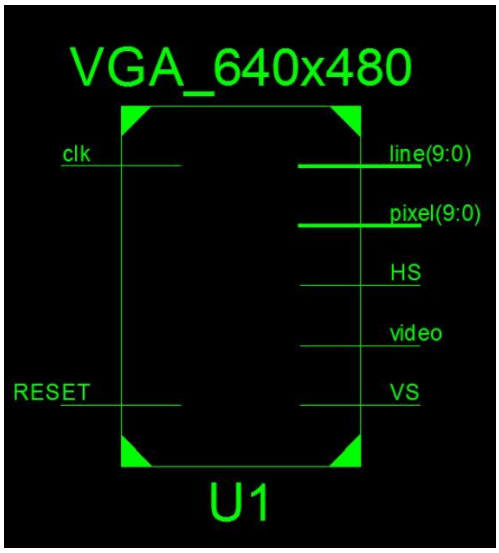
工程设计采用 Verilog 代码编程实现，画图模块调用 ROM 储存数据。

核心模块的实现依赖于状态机的建立。根据 jump、reset 等按钮开关等改变状态机的状态，进而改变 Demon 和工程的状态。背景和障碍物出现的依赖于计数器和时钟分频模块的实现。利用计数器根据时间改变背景和障碍物的移动速度，进而提升游戏难度。

工程中 clk_game 即为控制背景和障碍物速度的时钟，它的产生根据游戏的时间控制。

3.2 实现过程

3.2.1 VGA 显示模块



图表 2 VGA 显示模块图像

VGA 显示模块产生系统显示用的行同步信号和列同步信号：HS 和 VS

其中采用分频模块产生频率为 25mhz 的时钟信号 dclk

输出信号 pixel[9:0]与 line[9:0]为相应的行计数与列计数

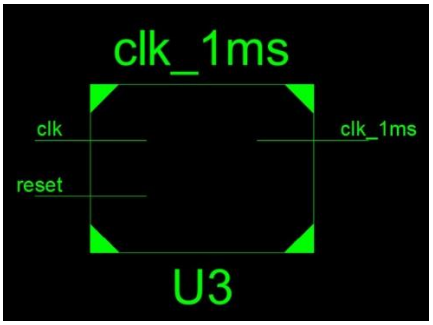
如果 video == 1，则表示现在系统扫描至屏幕显示图像的区域

```
module VGA_640x480(
  input wire clk,
  input wire RESET,
  output HS,
  output VS,
  output [9:0]pixel,
  output [9:0]line,
  output wire video
    );
reg dclk;//分频的时钟，频率为 25mhz
reg [9:0]h_count;
reg [9:0]v_count;
always@(posedge clk or posedge RESET)begin
  if(RESET == 1)
      dclk <= 1'b1;
  else
      dclk <= ~dclk;
```

```
end
//行计数: VGA horizontal counter(0-639+8+8+96+40+8 = 799)

always@(posedge dclk or posedge RESET)begin
 if(RESET == 1)
     h_count <= 10'h0;
 else if(h_count == 10'd799)
     h_count <= 10'h0;
 else
     h_count <= h_count + 10'h1;
end

assign pixel = h_count - 10'd143;
assign HS = (h_count >= 10'd96);

//帧计数: v_count:VGA vertical counter (0-524)
always@(posedge dclk or posedge RESET)begin
 if(RESET == 1)
     v_count <= 10'h0;
 else if(h_count == 10'd799)
     if(v_count == 10'd524)
         v_count <= 10'h0;
     else
     v_count <= v_count + 10'h1;
end

assign line = v_count - 10'd35;
assign VS = (v_count >= 10'd2);
assign video = (((h_count >= 10'd143)&&(h_count < 10'd783)) && ((v_count >= 10'd35)
&& (v_count < 10'd515)));
endmodule
```

3.2.2 时钟分频模块

由于时钟模块产生的原相似，并且本次工程调用的时钟模块较多，故挑选一个时钟模块进行讲解，其余产生原理相同。



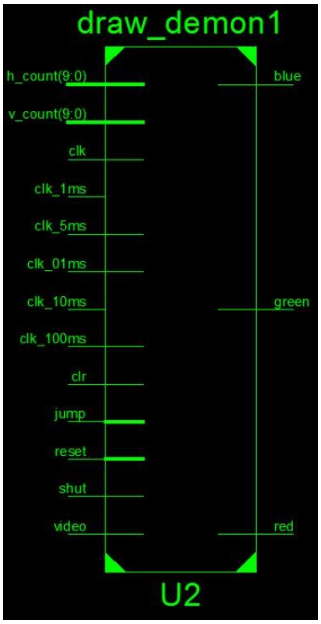
图表 3 时钟分频模块图像

输入系统时钟模块 clk：50mhz 的时钟，分频以后输出 1000hz 的时钟

Verilog 源码如下表所示：

```
module clk_1ms(
 input clk,
 input reset,
 output clk_1ms
   );
reg[20:0] count;
reg second_m;

initial count <= 0;

 always@(posedge clk)begin
     if(reset || (count == 49999))begin
         count <= 0;
         second_m <= 1;
     end
     else begin
         count <= count + 1;
         second_m <= 0;
         end
     end
assign clk_1ms = second_m;
endmodule
```

3.2.3 工程核心模块



图表 4 工程核心模块

输入：

分频时钟信号：clk_1ms, clk_01ms, clk_5ms, clk_10ms, clk_100ms;

清屏信号：clr
重置信号：reset
暂停信号：shut
Demon 跳跃信号：jump
扫描坐标信号：h_count[9:0]行扫描坐标、v_count[9:0]列扫描坐标
输出：
RGB 信号：red、green、blue

3.2.3.1 ROM 的调用

```
Demon1    Rom1(.clka(clk),.addra(addr),.douta(color1));              //Demon 图片 1
Demon2    Rom2(.clka(clk),.addra(addr),.douta(color2));
                                          //Demon 图片 2，两张图片交替显示，构成动态图
ground    Rom3(.clka(clk),.addra(show_ground1),.douta(color_ground1));
                                                                    //背景地面的显示
trees1    Rom5(.clka(clk),.addra(show_tree1),.douta(color_tree1));   //障碍物 1 显示
clouds    Rom6(.clka(clk),.addra(show_cloud),.douta(color_cloud));   //背影云彩的显示
trees2    Rom9(.clka(clk),.addra(show_tree2),.douta(color_tree2));   //障碍物 2 显示
game_over Rom10(.clka(clk),.addra(show_over),.douta(color_over));        //游戏结束图
片显示
```

3.2.3.2 Demon 形态的选择

```
always@(posedge clk_100ms)begin//mode 用来判断小恐龙的两种形态，mode == 1 显示图片 1；mode
== 2 显示图片 2
    if(shut == 1 || collision1 == 1) //判断游戏是否结束或暂停
        mode = mode;
    else mode = ~mode;
end
always@(*)begin
if(h_count >= x1 && h_count < x2 && v_count >= Y && v_count < Y+60)begin//判断是否到
了显示恐龙的位置
        y = v_count - Y;
        x = h_count - x1;
        addr = y * 60 + x;
        if(mode == 1 || high == 1)begin//通过 mode 控制 draw_demon1 和 draw_demon2，进而
控制小恐龙的显示
            draw_demon1 = 1;
            draw_demon2 = 0;
        end
        else begin
            draw_demon2 = 1;
            draw_demon1 = 0;
        end
end
else begin
```

```
    draw_demon1 = 0;                        //如果不在显示区域，则变量为 0
    draw_demon2 = 0;
end
```

addr 为传递给 ROM 的位置信息，ROM 会返回该位置显示的颜色 color1 或者 color2。
若 draw_demon1 == 1，则绘制图像 1
若 draw_demon2 == 1，则绘制图像 2

3.2.3.3 背景及障碍物模块的移动及难度增加功能

```
always@(posedge clk_01ms)begin//clk_game 产生的频率受 diff 控制，cllk_game 控制背景的移动速度
    count = count + 1;
    if(count >= diff)begin
        count  = 0;
        clk_game = 1;
    end
    else begin
        clk_game = 0;
    end
end

always@(posedge clk_100ms)begin //diff 随游戏时间的增加而减少，从而使 clk_game 频率越来越高
    count1 = count1 + 1;            //通过对 diff（difficult）的改变来改变背景的移动速度，进而增加游戏难度
    if(reset == 1)begin
        diff = 60;
    end else
    if(count1 >= 50)begin
        count1 = 0;
        if(diff > 20)
            diff = diff - 4;
        else diff = diff;
    end
end

always@(posedge clk_game)begin//用来控制背景移动的时钟，clk_game 来控制背景的移动速度
if(clr == 1)begin                       //清屏操作
        ground = 0;
        X_tree1 = 1200;
        X_tree2 = 900;
    end
else begin
    if(shut == 1 || collision1 == 1)begin   //判断游戏是否结束或暂停
        ground = ground;
```

```
    end else if(reset == 1)begin
        ground = 64;
    end
    else if(ground <= 576)                          //ground 为控制背景显示的指针，
ground 为地面显示的开始位置
        ground = ground + 1;
    else ground = 64;

    if(reset == 1)
        X_tree1 = 1200;
    else if(shut == 1 || collision1 == 1)begin     //判断游戏是否结束或暂停
        X_tree1 = X_tree1;
    end
    else if(X_tree1 > 0 )
        X_tree1 = X_tree1 -1;                              //控制树（障碍物）的循环移
动
    else X_tree1 = 1200;

     if(reset == 1)                                          //复位操作
        X_tree2 = 900;
    else if(shut == 1 || collision1 == 1)begin     //判断游戏是否结束或暂停
        X_tree2 = X_tree2;
    end
    else if(X_tree2 > 0 )
        X_tree2 = X_tree2 -1;
    else X_tree2 = 900;

    if(reset == 1)begin                                     //复位操作
        X_cloud1 = 800;
        X_cloud2 = 700;
    end else
    if(shut == 1 || collision1 == 1)begin              //判断游戏是否结束或暂停
        X_cloud1 = X_cloud1;
        X_cloud2 = X_cloud2;
    end
    else begin
        if(X_cloud1 > 0 )
            X_cloud1 = X_cloud1 -1;                        //云彩的循环移动
        else X_cloud1 = 800;
        if(X_cloud2 > 0)
            X_cloud2 = X_cloud2 - 1;
        else X_cloud2 = 700;
    end
end
```

```
end
```

该功能由通过对 diff 的改变来改变控制背景移动的时钟 clk_game，而 diff 改变游戏开始以后的时间控制，随着时间的增加，diff 的值不断减少，所以 clk_game 的频率逐渐增加，伴随着背景和障碍物移动速度的增加，游戏难度逐渐增大。

3.2.3.4 位置及坐标的判断功能

```
assign show_ground1 = (high_ground1*640 + (ground+h_count)%512);              //显示地面移动的指针，ROM 的输入
assign show_ground2 = (high_ground2*640 + (ground+h_count)%512);
always@(*)begin
if(h_count >= x1 && h_count < x2 && v_count >= Y && v_count < Y+60)begin//判断是否到了显示恐龙的位置
        y = v_count - Y;
        x = h_count - x1;
        addr = y * 60 + x;
        if(mode == 1 || high == 1)begin//通过 mode 控制 draw_demon1 和 draw_demon2，进而控制小恐龙的显示
            draw_demon1 = 1;
            draw_demon2 = 0;
        end
        else begin
            draw_demon2 = 1;
            draw_demon1 = 0;
        end
end
else begin
    draw_demon1 = 0;                        //如果不在显示区域，则变量为 0
    draw_demon2 = 0;
end

if(v_count >= Y_ground1 && v_count < Y_ground1+60 && h_count >= 1 && h_count <= 640)begin//判断是否到了显示地 1 的位置
    in_ground1 = 1;
    high_ground1 = v_count - Y_ground1;
end
else begin
    in_ground1 = 0;
end

if(v_count >= Y_tree1 && v_count <Y_tree1+100 && h_count >= X_tree1 && h_count <= X_tree1+70 && h_count >= 1 && h_count <= 640)begin//判断是否到了显示树 1 的位置
    in_tree1= 1;
    x_t1 = h_count - X_tree1;
    y_t1 = v_count - Y_tree1;
```

```
    show_tree1   = (y_t1 * 70 + x_t1);     //ROM 的输入端
end
else begin
    in_tree1 = 0;
end

if(v_count >= Y_tree2 && v_count <Y_tree2+60 && h_count >= X_tree2 && h_count <= X_tree2+70
&& h_count >= 1 && h_count <= 640)begin//判断是否到了显示树 1 的位置
    in_tree2= 1;
    x_t2 = h_count - X_tree2;
    y_t2 = v_count - Y_tree2;
    show_tree2  = (y_t2 * 70 + x_t2);      //ROM 的输入端
end
else begin
    in_tree2 = 0;
end

//判断是否到了显示云彩的位置
if(v_count >= Y_cloud1 && v_count < Y_cloud1 +70 && h_count >= X_cloud1 && h_count <
X_cloud1+100 && h_count >= 1 && h_count <= 640)begin
    in_cloud = 1;
    x_c1 = h_count - X_cloud1;
    y_c1 = v_count - Y_cloud1;
    show_cloud = (y_c1 * 100 + x_c1);      //ROM 的输入端
end else
if(v_count >= Y_cloud2 && v_count < Y_cloud2 +70 && h_count >= X_cloud2 && h_count <
X_cloud2+100 && h_count >= 1 && h_count <= 640)begin
    in_cloud = 1;
    x_c2 = h_count - X_cloud2;
    y_c2 = v_count - Y_cloud2;
    show_cloud = (y_c2 * 100 + x_c2);      //ROM 的输入端
end
else begin
    in_cloud = 0;
end

//判断是否到了显示结束画面的位置
if(v_count >= Y_over && v_count < Y_over+70 && h_count >= X_over && h_count <
X_over+400)begin
    in_over = 1;
    x_v = h_count - X_over;
    y_v = v_count - Y_over;
    show_over = y_v * 400 + x_v;               //ROM 的输入端
end
```

```
else in_over = 0;
end
```

利用 reg 变量 draw_demon1, draw_demon2, in_ground1, in_ground2, in_tree1, in_tree2, in_cloud, in_cloud2, in_over 来判断是否进入图像的显示区域，并返回 ROM 的输入指针，得到相应颜色的输出。

3.2.3.5 RGB 的赋值操作

```
always@(*)begin
red = 0;blue = 0;green = 0;                    //vga 显示的初始化
if(video == 1) begin
    red = 1;
    green = 1;
    blue = 1;
end

if(in_tree1 == 1)begin                              //如果在显示区域，则给 rgb 赋值
    if(color_tree1[0] == 0)begin                //ROM 输出
        red =0;
        green =0;
        blue = 0;
    end
end else if(in_tree2 == 1)begin
    if(color_tree2[0] == 0)begin
        red = 0;green = 0;blue= 0;
    end
end

if(draw_demon1 == 1 )begin                          //如果在显示区域，则给 rgb 赋值
        if(color1[0] == 0)begin                     //ROM 输出
                red = 0;
                green = 0;
                blue = 0;
        end
end else if(draw_demon2 == 1  )begin
            if(color2[0] == 0) begin
                        red = 0;
                        green = 0;
                        blue =0;
                    end
end

if(in_ground1 == 1)begin                            //如果在显示区域，则给 rgb 赋值
    if(color_ground1[0] == 0)begin                  //ROM 输出
        red = color_ground1[0];
```

```
        green = color_ground1[0];
        blue = color_ground1[0];
    end
end else if(in_ground2 == 1)begin
    if(color_ground2[0] == 0)begin
        red = color_ground2[0];
        green = color_ground2[0];
        blue = color_ground2[0];
    end
end

if(in_cloud == 1)begin                              //如果在显示区域，则给 rgb 赋值
    if(color_cloud[0] ==0)begin                  //ROM 输出
        red = 0;green = 0;blue = 0;
    end
end

if(collision1 == 1)begin      //如果游戏结束，则显示“game over”
    if(in_over == 1)begin
        red = color_over[0];
        blue = color_over[0];
        green = color_over[0];
    end
end
//将屏幕两侧涂成黑色
if(((h_count >= 0 && h_count <= 64 )||(h_count >= 576 && h_count <= 640))&& v_count >=
0 && v_count <= 480 )begin
    red = 0;
    green = 0;
    blue = 0;
end
end
```

如果进入图片显示区域的话，则根据相应 ROM 的输出对 RGB 进行赋值操作，从而在屏幕上显示出相应的图形

3.2.3.6 碰撞检测功能

```
//检测小恐龙碰撞的模块，选取坐标特征点判断（其中选取小恐龙的 5 个特征点进行识别）
if((x1+53 >= X_tree1 + 20) &&( x1 + 53 <= X_tree1 +53) &&( Y + 17 >= Y_tree1 + 14))
        collision1 = 1;
else if((Y +57 >= Y_tree1 + 14) && (x1 + 35 <= X_tree1 +53 )&& (x1 +35 >= X_tree1 + 20))
        collision1 = 1;
else if  ((Y + 57>= Y_tree1 + 14) && (x1 + 19 <= X_tree1 +53 )&&( x1+ 19 >= X_tree1 +
20))
        collision1 = 1;
```

```
else if ((Y + 36>= Y_tree1 + 14) && (x1 + 8  <= X_tree1 +53 )&& (x1 +8  >= X_tree1 + 20))
    collision1 = 1;
else if(  (Y + 37 >= Y_tree1 + 14) && (x1 + 21 <= X_tree1 + 53) && (x1 + 21 >= X_tree1
+ 20))
    collision1 = 1;
else if( (Y + 17 >= Y_tree2 + 7 ) && (x1 + 53 >= X_tree2 + 11) && (x1 + 53 <= X_tree2
+ 57))
    collision1 = 1;
else if( (Y+57 >= Y_tree2 + 7) && (x1 + 35 >= X_tree2 + 11 ) && (x1 + 35 <= X_tree2 +
57))
    collision1 = 1;
else if( (Y+57 >= Y_tree2 + 7) && (x1 + 19 >= X_tree2 + 11 ) && (x1 + 19 <= X_tree2 +
57))
    collision1 = 1;
else if( (Y+36 >= Y_tree2 + 7) && (x1 + 8 >= X_tree2  + 11)  && (x1 + 8 <= X_tree2 + 57))
    collision1 = 1;
else if( (Y+37 >= Y_tree2 + 7) && (x1 + 21 >= X_tree2 + 11) && (x1 + 21 <= X_tree2 +
57))
    collision1 = 1;
else collision1 = 0;
```

选取 Demon 的特征点，根据特征点进行相应的检测判断。如果特征点与障碍物重合，则视为碰撞，即 collision1 = 1;

3.2.3.7 Demon 在空中时速度的判断和检测

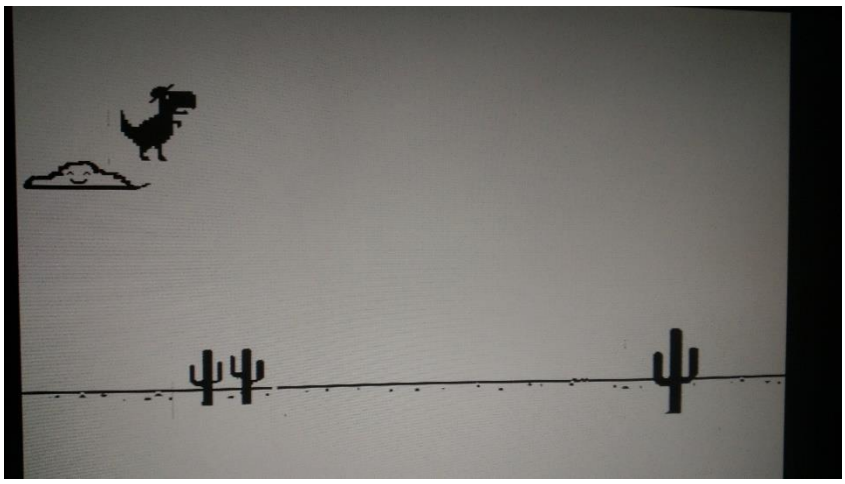
```
always@(posedge clk_5ms)begin//小恐龙跳起后在空中的时序控制
if(clr == 1)begin                    //复位与清除操作
    V = 0;
    Y = y1;
end
else begin
    if(jump == 1)begin
        if(Y == y1)begin
            V = V0;
            high = 1;
        end
    end
    else if(Y == y1 && V == 0)
        high = 0;
    if(high == 1)begin
        if(reset == 1)begin
            V = 0;
            Y = y1;
        end else
        if(shut == 1 || collision1 == 1)begin      //判断游戏是否结束或暂停
```

```
            S = S;
            Y = Y;
            V = V;
        end
        else if(V > -V0)begin
            S = (V0*V0 - V*V)/32;                    //利用公式算出小恐龙的位置和速度
            Y = y1 - S;
            V = V - delta;
        end
        else begin
            V = 0;
            Y = y1;
        end
    end
end
end
endmodule
```

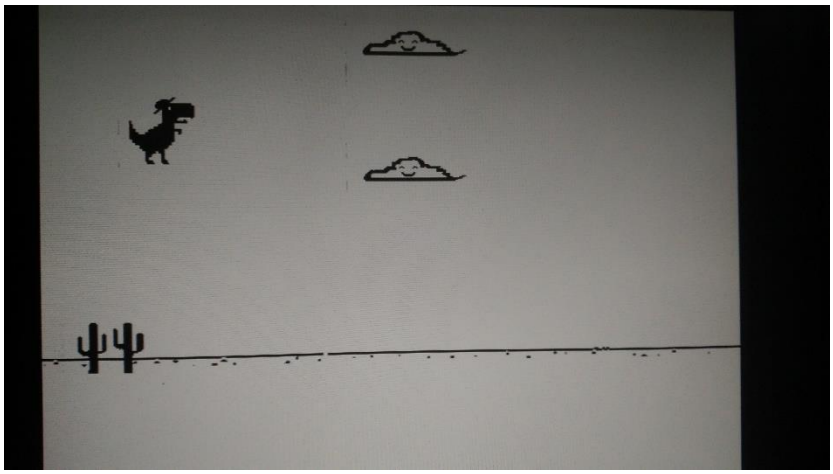
利用时钟来时序控制 Demon 在空中的位置，因为时钟变化较快，导致 Demon 坐标变化速度过快，故采用缩小的方法，将 Demon 位置的变化的幅度缩小 15 倍，得到变化速率适中的画面显示。

第4章 系统测试验证与结果分析

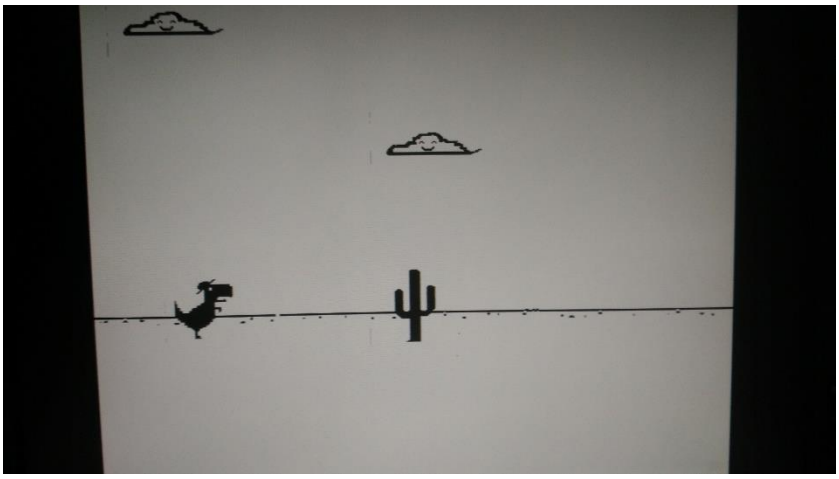
4.1 功能测试与结果分析



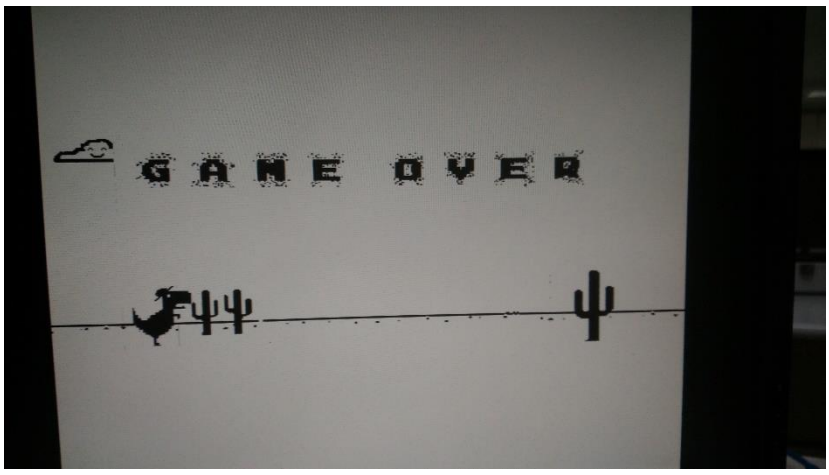
图表 5 Demon 跳跃操作 1



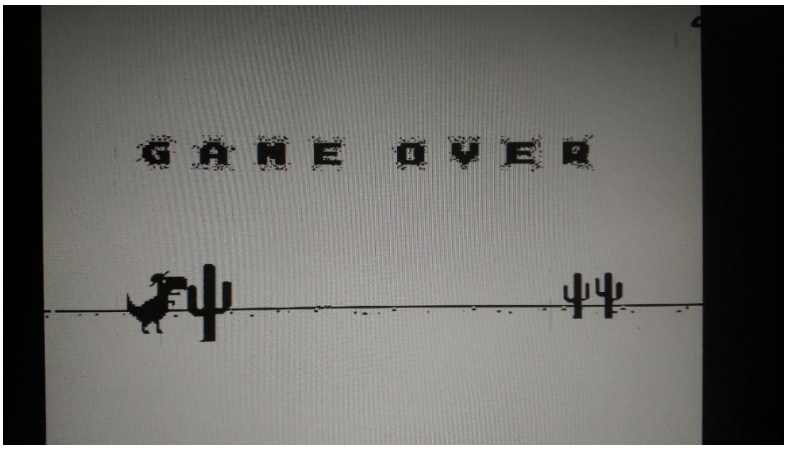
图表 6 Demon 的跳跃操作 2



图表 7 Demon 跑动状态



图表 8 结束界面 1



图表 9 结束界面 2

游戏操作方法：

按下 btn[0]小恐龙实现跳跃操作

按下 btn[1]为界面刷新操作，即游戏重新开始

打开 SW[7]游戏暂停，否则游戏继续进行

可以观察到 Demon 在撞到障碍物后游戏就会结束。在此期间，背景和障碍物的移动速度会不断加快，使游戏的难度逐渐增加。对用户的反应能力和对游戏的熟练程度的要求也会不断提高。

4.2 设计工程遇到的困难

1.Demon 跳跃的实现。

要实现速度具有加速的效果，就要实现速度和位置的时序变化。实验时设置好恒定的初速度和加速度。因为实验的分辨率并不高，所以如果速度过快，则游戏无法进行。位置变化过慢，就会降低游戏的真是性和趣味性。在实现是，位置没 5ms 变化一次，利用视觉暂留效果，感觉 Demon 的位置是连续变化的。而位移的实现，通过等比缩小的方法，使小恐龙位移受速度改变的影响等比减小，使屏幕上变化更为自然。

2.资源

初始时显示的图片都储存在 reg 变量里，后来发现在工程综合的过程中占用的时间过长，故后面采用 ROM 储存的方法，实验时通过调用 ROM 获得相应的 RGB 值。

3.Demon 碰撞的检测

实验是原本想法是采用检测颜色冲突的方法来检测碰撞，随后发现检测颜色的过程中因为时序的问题会出现 Demon 提前停止的现象，所以后来采用检测坐标的方法。通过检测 Demon 的 5 个特征点的坐标来检测 Demon 的碰撞效果。

4.地图的动态显示

由于 verilog 语言余除只能余除 2 的幂次，而屏幕的宽度为 640，所以画面如果铺满整个屏幕，背景的时序显示就会出现问题。所以屏幕用于显示的实际宽度为 512。通过地图指针的时序变化来实现地图的循环显示的效果。

第5章 结论与展望

自己完成这个项目确实投入了很多时间。怎么说呢，如果逻辑课不布置这次 project

的话，我应该永远不会埋头在实验室去攻克这个游戏。首先感谢老师能给我这个机会让我发现自己的兴趣和价值，自己也渐渐喜欢上在实验室和大家一起完成一个项目的感觉了。在这个期间自己也结识了跟多的朋友，学到了更多的东西，真是非常感谢老师这半年的教导。

在完成这个工程的过程中确实遇到了很多困难。首先是对 verilog 语言的不熟悉，自己首先学习了 verilog 的语法知识。其次就是对 vga 的调用，因为自己以前并没有接触过这类的硬件，所以花了大概一下午的时间才能在屏幕上显示出图像。

其次就是对 Demon 功能的实现了。知道真正去实践才可以学到东西，发现理论和实践的不同。自己也是在实验过程中不断改进方法，从屏幕上显示不出图像到正常显示、从调用 reg 变量到利用 ROM、从没有背景到背景丰富、从碰撞检测一直存在 bug 到 bug 改进、从没有难度到游戏难度会自动变化。我觉得这个过程不仅是我们学习的过程，更是我们兴趣建立的过程、自信心增长的过程。虽然这个工程自己实现的时间较晚，由于 deadline 的限制，有的功能还没有添加，但是这次的经历抵得上自己在自习室自习双倍的时间学到的东西。

通过这次实验，自己更有自信做出更好的游戏，相信自己会在这条路上越走越远。再次感谢老师这学期的辛勤教导啦，提前祝老师新年快乐！