北京科技大学 计算机与通信工程学院

实 验 报 告

实验名称：时序逻辑实验

一、实验目的与实验要求

1、实验目的

综合运用本课程所学习的知识，设计并实现复杂的数字系统。

2、实验要求

1） 在实验报告中提交状态图或者算法流程状态图，系统级模块图、设计代码、仿真程序、仿真结果截图、实测验证结果照片。

2） 制作复杂数字系统设计报告 PPT，验收过程除了演示自己的作品外，还需要通过 PPT 讲解数字系统设计。PPT 讲解时间 2 分钟。

3） 提交实验报告、PPT 和每个实验的完整工程文件。

二、实验设备（环境）及要求

（1） Xilinx Ego1 实验平台。

（2） OS：Win7 64 位

（3） Software：Vivado15.4 开发工具

三、实验内容与步骤

1、实验

（1）实验内容

设计一款贪吃蛇游戏。通过 VGA 接口控制液晶显示器,在液晶显示器上实现该游戏。具体要求为,在液晶显示器上有一移动的”蛇”,通过开发板上的拨码开关实现“蛇”上下左右移动。本实验刚开始的蛇身长度定位 3。由于程序书写上的限制，我们设定蛇的最大长度为 16。设计平台为 xilinx 的 EGO1 FPGA 开发板。同时会在屏幕有效区域随机出现“苹果”,“蛇”每吃到一次苹果则表示一次成功操作,得分加 1,并在数码管上显示得分。

（2）主要步骤

a) 新建工程，取名为 Greedy_Snake。

b) 将工程文件分层，经过分析修改，最终本设计含有六个子模块,分别是：

- Clk4to1
- Game_ctrl_unit
- Snake_eating_apple
- Snake1
- Key
- KeyBoard
- VGA_control
- Seg_display

c) 决定输入输出的量和类型，由于本次实验要求通过 VGA 在显示屏上显示，则需要设计 VGA 的输出量。（实验后期，为了增加实验功能，则设计了键盘的输入和输出方法）具体的输入输出见下。

- input clk0,
- input reset,
- input ps2d,
- input ps2c,
- output h_sync,
- output v_sync,
- output [2:0] color_out, //颜色输出
- output [7:0] seg_out,//数码管管脚
- output [3:0] sel

d) 顶层模块如下：

```
module Greedy_Snake (
input clk0,
input reset,
input ps2d,
input ps2c,
```

```
output h_sync,
output v_sync,
output [2:0] color_out, //颜色输出
output [7:0] seg_out,//数码管管脚
output [3:0] sel
);


wire left_key_press,right_key_press, up_key_press, down_key_press;
wire [1:0] snake;
wire [9:0] x_pos;
wire [9:0] y_pos;
wire [5:0] apple_x;
wire [4:0] apple_y;
wire [5:0] head_x;
wire [5:0] head_y;
wire add_cube;
wire [1:0] game_status;
wire hit_wall;
wire hit_body;
wire die_flash;
wire restart;
wire [6:0] cube_num;
wire left; //由于用了键盘表示，这里使用 wire 而不是 input
wire right;
wire up;
wire down;
wire key_relese; //键盘按键释放按钮

clk4to1 clk4to1(clk0,reset,clk);

Snake_one
Snake1(.clk(clk),.reset(reset),.left_press(left_key_press),.right_press(right_key_press)
,.up_press(up_key_press),.down_press(down_key_press),.snake(snake),.x_pos(x_pos)
,.y_pos(y_pos),.head_x(head_x),.head_y(head_y),.add_cube(add_cube),.game_status(
game_status),.cube_num(cube_num),.hit_body(hit_body),.hit_wall(hit_wall),.die_flas
h(die_flash) );

Game_ctrl_unit
game_ctrl_unit1(   .clk(clk),.reset(reset),.key1_press(left_key_press),.key2_press(right
_key_press),.key3_press(up_key_press),.key4_press(down_key_press),.game_status(
game_status),.hit_wall(hit_wall),.hit_body(hit_body),.die_flash(die_flash),.restart(rest
art) );
```

```
Keyboard keyboard1(clk0,reset,ps2d,ps2c,left,right,up,down,key_relese);//键盘模块

Snake_eating_apple
snake_eating_apple1(   .clk(clk),.reset(reset),.apple_x(apple_x),.apple_y(apple_y),.head_x(head_x),.head_y(head_y),.add_cube(add_cube) );

VGA_control
vga_control(.clk(clk),.reset(reset),.h_sync(h_sync),.v_sync(v_sync), .snake(snake),.color_out(color_out),.x_pos(x_pos), .y_pos(y_pos),.apple_x(apple_x),.apple_y(apple_y)
  );

Key
key1(  .clk(clk),.reset(reset),.left(left),.right(right),.up(up),.down(down),.left_key_press(left_key_press),  .right_key_press(right_key_press),.up_key_press(up_key_press),.down_key_press(down_key_press) );

Seg_display
Seg_display1( .clk(clk),.reset(reset),.add_cube(add_cube),.seg_out(seg_out),.sel(sel) );

Endmodule
```

e) Keyboard 键盘模块代码：

该代码主要用来控制键盘的输入输出，通过 PS/2 键盘通码的查询和设置，完成键盘上的 WSAD 的上下左右的操作。同时，设置 key_relese 来控制键盘上按键的松开，使状态机得到正确的改变。

```
module keyboard(
input wire clk, reset,
input wire ps2d, ps2c,
output wire left,right,up,down,
output reg key_relese
);
//declare veriables
wire [7:0] dout;
wire rx_done_tick;
supply1 rx_en; // always HIGH
reg [7:0] key;

//instantiate models
// Nexys4 converts USB to PS2, we grab that data with this module
```

```
ps2_rx ps2(clk, reset, ps2d, ps2c,rx_en, rx_done_tick, dout);

//sequencial logic
always@(posedge clk)
begin
	if(rx_done_tick)
	begin
		key <= dout; //key is one rx cycle behind dout
	if(key == 8'hF0)	//check if the key has been relesed
		key_relese <= 1'b1;
	else
		key_relese <= 1'b0;
	end
end
//output control keys of interest
assign left = (key==8'h01C) & !key_relese; // 1c is the code for 'A'
assign right = (key==8'h023) & !key_relese; // 23 is the code for 'D'
assign up = (key==8'h01D) & !key_relese; // 1D is the code for 'W'
assign down = (key==8'h01B) & !key_relese;//1B is the code for 's'


endmodule

//ps/2 底层调用文件
`timescale 1ns / 1ps
module ps2_rx(
input wire clk, reset,
input wire ps2d, ps2c, rx_en,
output reg rx_done_tick,
output wire [7:0] dout
);

// symbolic state declaration
localparam [1:0]
   idle = 2'b00,
   dps    = 2'b01, //data, parity, stop
   load = 2'b10;

// signal declaration
reg [1:0] state_reg, state_next;
reg [7:0] filter_reg;
wire [7:0] filter_next;
reg f_ps2c_reg;
wire f_ps2c_next;
```

```
reg [3:0] n_reg, n_next;
reg [10:0] b_reg, b_next;
wire fall_edge;

// filter and falling-edge tick generation for ps2c
always @(posedge clk, posedge reset)
if (reset)
   begin
      filter_reg <= 0;
      f_ps2c_reg <= 0;
   end
else
   begin
      filter_reg <= filter_next;
      f_ps2c_reg <= f_ps2c_next;
   end

assign filter_next = {ps2c, filter_reg[7:1]};
assign f_ps2c_next = (filter_reg==8'b11111111) ? 1'b1 :
                              (filter_reg==8'b00000000) ? 1'b0 :
                                f_ps2c_reg;
assign fall_edge = f_ps2c_reg & ~f_ps2c_next;


// FSMD state & data registers
always @(posedge clk, posedge reset)
   if (reset)
      begin
         state_reg <= idle;
         n_reg <= 0;
         b_reg <= 0;
      end
   else
      begin
         state_reg <= state_next;
         n_reg <= n_next;
         b_reg <= b_next;
      end

// FSMD next-state logic
always @(*)
begin
   state_next = state_reg;
   rx_done_tick = 1'b0;
```

```
    n_next = n_reg;
    b_next = b_reg;
    case (state_reg)
        idle:
            if (fall_edge & rx_en)
                begin
                    // shift in start bit
                    b_next = {ps2d, b_reg[10:1]};
                    n_next = 4'b1001;
                    state_next = dps;
                end
        dps: // 8 data + 1 parity + 1 stop
            if (fall_edge)
                begin
                    b_next = {ps2d, b_reg[10:1]};
                    if (n_reg==0)
                        state_next = load;
                    else
                        n_next = n_reg - 1;
                end
        load: // 1 extra clock to complete the last shift
            begin
                state_next = idle;
                rx_done_tick = 1'b1;
            end
    endcase
end

// output
assign dout = b_reg[8:1]; // data bits

endmodule
```

f) 分频模块代码：

由于设计要求，FPGA 板上所提供的频率为 100MHz，而实际 VGA 显示刷新中则需要 25MHz，因此需要对所提供的频率进行分频操作，否则，会在液晶显示屏上显示 out range，不能正常显示图像。

```
`timescale 1ns / 1ps
module clk4to1(
input clk0,
input reset,
output clk
```

```
    );
    reg [2:0]cnt1;
    wire c0;
always @(posedge clk0 or negedge reset )
            if(!reset)
                cnt1<=0;
            else
                cnt1<=cnt1+1;
assign c0=~cnt1[1];
assign clk=~c0;
endmodule
```

g) 数码管显示数据模块：

在游戏中，我们增加了计数模块，则需要对数码管进行显示操作，通过设置 seg_out 及管脚的选择和时间脉冲的计算、控制，最后使得共 4 个数码管进行显示操作，逢十进位。最高可显示 4 位。

```
module Seg_display(

    input clk,
    input reset,

    input add_cube,

    output reg [7:0] seg_out,
    output reg [3:0] sel);

    reg [15:0] point;
    reg [31:0] clk_cnt;

    always @(posedge clk or negedge reset ) begin
            if( !reset    ) begin
            seg_out <= 0;
           clk_cnt <= 0;
            sel <= 0;

        end
        else begin

            if( clk_cnt <= 200000 )
                begin
              clk_cnt <= clk_cnt + 1;
              if( clk_cnt == 50000 ) begin
                  sel <= 4'b0001;
```

```
        case( point[3:0] )

            0:seg_out =8'b0011_1111;
            1:seg_out =8'b0000_0110;
            2:seg_out =8'b0101_1011;
            3:seg_out =8'b0100_1111;
            4:seg_out =8'b0110_0110;
            5:seg_out =8'b0110_1101;
            6:seg_out =8'b0111_1101;
            7:seg_out =8'b0000_0111;
            8:seg_out =8'b0111_1111;
            9:seg_out =8'b0110_1111;
        default;
        endcase

end

else if( clk_cnt == 100000 ) begin

        sel <= 4'b0010;

        case( point[7:4] )

            0:seg_out =8'b0011_1111;
            1:seg_out =8'b0000_0110;
            2:seg_out =8'b0101_1011;
            3:seg_out =8'b0100_1111;
            4:seg_out =8'b0110_0110;
            5:seg_out =8'b0110_1101;
            6:seg_out =8'b0111_1101;
            7:seg_out =8'b0000_0111;
            8:seg_out =8'b0111_1111;
            9:seg_out =8'b0110_1111;
        default;

        endcase
end

else if( clk_cnt == 150000 ) begin

        sel <= 4'b0100;

        case( point[11:8] )
```

```
                    0:seg_out =8'b0011_1111;
                    1:seg_out =8'b0000_0110;
                    2:seg_out =8'b0101_1011;
                    3:seg_out =8'b0100_1111;
                    4:seg_out =8'b0110_0110;
                    5:seg_out =8'b0110_1101;
                    6:seg_out =8'b0111_1101;
                    7:seg_out =8'b0000_0111;
                    8:seg_out =8'b0111_1111;
                    9:seg_out =8'b0110_1111;
                default;


                endcase
            end

            else if( clk_cnt == 200000 ) begin
                sel <= 4'b1000;

            case( point[15:12] )

                    0:seg_out =8'b0011_1111;
                    1:seg_out =8'b0000_0110;
                    2:seg_out =8'b0101_1011;
                    3:seg_out =8'b0100_1111;
                    4:seg_out =8'b0110_0110;
                    5:seg_out =8'b0110_1101;
                    6:seg_out =8'b0111_1101;
                    7:seg_out =8'b0000_0111;
                    8:seg_out =8'b0111_1111;
                    9:seg_out =8'b0110_1111;
                default;

            endcase
            end

        end
        else
            clk_cnt <= 0;
            end
end

reg addcube_state;
```

```
always @( posedge clk or negedge reset    ) begin

    if( reset == 0 ) begin

        point <= 0;
        addcube_state <= 0;

    end

    else begin

        case( addcube_state )

        0: begin
            if( add_cube ) begin

                if( point[3:0] < 9 )
                    point[3:0] <= point[3:0] + 1;
                else begin
                    point[3:0] <= 0;
                    if( point[7:4] < 9 )
                        point[7:4] <= point[7:4] + 1;
                    else begin
                        point[7:4] <= 0;
                        if( point[11:8] < 9 )
                            point[11:8] <= point[11:8] + 1;
                        else begin
                            point[11:8] <= 0;
                            point[15:12] <= point[15:12] + 1;
                        end
                    end
                end

                addcube_state <= 1;

            end
        end

        1: begin

            if(!add_cube)
                addcube_state <= 0;
```

```
            end

            endcase

        end

    end

endmodule
```

h) 按键控制模块

由于需要在板子上或者键盘上进行上下左右的操作，那必须设置一定的状态机来进行状态的改变，一共设置了 left_key_press、right_key_press 等 4 个按键操作，并按时间脉冲进行改变。

```
module Key (

    input clk,
    input reset,

    input left,
    input right,
    input up,
    input down,

    output reg left_key_press,
    output reg right_key_press,
    output reg up_key_press,
    output reg down_key_press);

    reg [31:0] clk_cnt;
    reg left_key_last;
    reg right_key_last;
    reg up_key_last;
    reg down_key_last;

    always @( posedge clk or negedge reset    ) begin

        if( !reset ) begin

            clk_cnt <= 0;
```

```
        left_key_press <= 0;
        right_key_press <= 0;
        up_key_press <= 0;
        down_key_press <= 0;

        left_key_last <= 0;
        right_key_last <= 0;
        up_key_last <= 0;
        right_key_last <= 0;

    end

    else begin
        if( clk_cnt == 50000 ) begin

            clk_cnt <= 0;
            left_key_last <= left;
            right_key_last <= right;
            up_key_last <= up;
            down_key_last <= down;

            if( left_key_last == 0 && left == 1 ) begin
                left_key_press <= 1;
            end

            if( right_key_last == 0 && right == 1 ) begin
                right_key_press <= 1;
            end

            if( up_key_last == 0 && up == 1 ) begin
                up_key_press <= 1;
            end

            if( down_key_last == 0 && down == 1 ) begin
                down_key_press <= 1;
            end

        end

        else begin
            clk_cnt <= clk_cnt + 1;
            left_key_press <= 0;
            right_key_press <= 0;
```

```
                up_key_press <= 0;
                down_key_press <= 0;
            end

        end

    end

endmodule
```

i) VGA 控制模块：

在设计中，VGA 的颜色显示都是需要依靠 if-else 逻辑进行实现的，因此，如果需要设置颜色，其实是相当困难的，在设计中我们遇到了全画面变色等出乎意料的情况，并且对颜色的选择和分辨也是一件较难的事情，在参考了老师的俄罗斯方块的代码后，最终调试成功

```
module VGA_control (

    input clk,
    input reset,

    input [1:0] snake,
    input [5:0] apple_x,
    input [4:0] apple_y,

    output reg [9:0] x_pos,
    output reg [9:0] y_pos,

    output reg h_sync,
    output reg v_sync,
    output reg [2:0] color_out);

    reg [19:0] clk_cnt;
    reg [9:0] line_cnt;


    localparam NONE = 2'b00;
    localparam HEAD = 2'b01;
    localparam BODY = 2'b10;
    localparam WALL = 2'b11;
```

```
localparam HEAD_COLOR = 3'b010;
localparam BODY_COLOR = 3'b011;

reg [3:0] lox;
reg [3:0] loy;



// generate display signal
always @( posedge clk or negedge reset) begin


    if( reset == 0) begin

        clk_cnt <= 0;
        line_cnt <= 0;

        h_sync <= 1;
        v_sync <= 1;

    end
    else begin

        x_pos <= clk_cnt - 144;
        y_pos <= line_cnt - 33;

        //产生水平控制信号
        if( clk_cnt == 0 ) begin
            h_sync <= 0;
            clk_cnt <= clk_cnt + 1;
        end
        else if( clk_cnt == 96 ) begin
            h_sync <= 1;
            clk_cnt <= clk_cnt + 1;
        end
        else if( clk_cnt == 799 ) begin
            clk_cnt <= 0;
            line_cnt <= line_cnt + 1;
        end
        else clk_cnt <= clk_cnt + 1;

        // 产生垂直控制信号
        if( line_cnt == 0 ) begin
            v_sync <= 0;
```

```
                end
                else if( line_cnt == 2 ) begin
                    v_sync <= 1;
                end
                else if( line_cnt == 521 ) begin
                    line_cnt <= 0;
                    v_sync <= 0;
                end

if( x_pos >= 0 && x_pos < 640 && y_pos >= 0 && y_pos < 480 )
  begin

                    lox = x_pos[3:0];
                    loy = y_pos[3:0];

                    if( x_pos[9:4] == apple_x && y_pos[9:4] == apple_y )

                        case( {loy, lox} )

                        8'b0000_0000: color_out = 3'b000;
                         default: color_out = 3'b001;
                         endcase

                    else if( snake == NONE )
                        color_out = 3'b000;

                    else if( snake == WALL )
                        color_out = 3'b101;

                    else if( snake == HEAD | snake == BODY ) begin

                        case( {lox, loy} )

                        8'b0000_0000: color_out = 3'b000;
                            default:color_out = ( snake == HEAD ) ? HEAD_COLOR :
BODY_COLOR;

                        endcase

                    end

                end

                else
```

```
                color_out = 3'b000;
            end
        end
endmodule
```

j) 贪吃蛇移动模块

贪吃蛇的移动中涉及了状态机的改变、食物增加的情况，因此，在程序中，我只涉及到 16 位的，即蛇身最长只会达到 16 段，因为每一段都要涉及多一个下一个状态的表示，在 Verilog 中显然用该方式描述是过于麻烦的，但是又奈何水平有限，没有想到更好的方法来处理该情况。

```
module Snake_one(

    input clk,
    input reset,

    input left_press,
    input right_press,
    input up_press,
    input down_press,

    input [9:0] x_pos,
    input [9:0] y_pos,

    input add_cube,

    input [1:0] game_status,
    input die_flash,

    output reg [1:0] snake,
    output reg [6:0] cube_num,

    output [5:0] head_x,
    output [5:0] head_y,

    output reg die_body,
    output reg die_wall);
```

```
localparam UP = 2'b00;
localparam DOWN = 2'b01;
localparam LEFT = 2'b10;
localparam RIGHT = 2'b11;

localparam NONE = 2'b00;
localparam HEAD = 2'b01;
localparam BODY = 2'b10;
localparam WALL = 2'b11;

localparam PLAY = 2'b10;

reg [31:0] cnt;

wire [1:0] direct;
reg  [1:0] direct_r;
assign direct=direct_r;
reg [1:0] direct_next;

reg change_to_left;
reg change_to_right;
reg change_to_up;
reg change_to_down;

reg [5:0] cube_x[15:0];
reg [5:0] cube_y[15:0];
reg [15:0] is_exist;

reg [2:0] color;

assign head_x = cube_x[0];
assign head_y = cube_y[0];
```

always @(posedge clk or negedge reset) //初始化刚开始时我们设定蛇向右运动

```
    if( !reset)
        direct_r<=RIGHT ;
    else
        direct_r<=direct_next;
```

```
always @( posedge clk or negedge reset    )
begin
    if(!reset )
          begin                    //设定初始化后刚开始三个蛇块的位置

        cnt <= 0;

        cube_x[0] <= 10;
        cube_y[0] <= 5;

        cube_x[1] <= 9;
        cube_y[1] <= 5;

        cube_x[2] <= 8;
        cube_y[2] <= 5;

        die_wall <= 0;
        die_body <= 0;

         end

    else
         begin

        cnt <= cnt + 1;

        if( cnt == 12500000 ) begin
            cnt <= 0;

        if( game_status == PLAY ) begin    //在 play 过程中判断是否 die_wall

                    if( ( direct == UP && cube_y[0] == 1 )     //如果方向这时是
上，并且已经到头了，这时候就头碰到就死
                        | ( direct == DOWN && cube_y[0] == 28 )//同上的思
想
                        | ( direct == LEFT && cube_x[0] == 1 )
                        | ( direct == RIGHT && cube_x[0] == 38 ) )
                    die_wall <= 1;
                //下边判断是否 die_dody
else if( ( cube_y[0] == cube_y[1] && cube_x[0] == cube_x[1] && appear[1] == 1 )
    | ( cube_y[0] == cube_y[2] && cube_x[0] == cube_x[2] && appear[2] == 1 )
    | ( cube_y[0] == cube_y[3] && cube_x[0] == cube_x[3] && appear[3] == 1 )
    | ( cube_y[0] == cube_y[4] && cube_x[0] == cube_x[4] && appear[4] == 1 )
    | ( cube_y[0] == cube_y[5] && cube_x[0] == cube_x[5] && appear[5] == 1 )
```

```
        | ( cube_y[0] == cube_y[6] && cube_x[0] == cube_x[6] && appear[6] == 1 )
        | ( cube_y[0] == cube_y[7] && cube_x[0] == cube_x[7] && appear[7] == 1 )
        | ( cube_y[0] == cube_y[8] && cube_x[0] == cube_x[8] && appear[8] == 1 )
        | ( cube_y[0] == cube_y[9] && cube_x[0] == cube_x[9] && appear[9] == 1 )
    | ( cube_y[0] == cube_y[10] && cube_x[0] == cube_x[10] && appear[10] == 1 )
    | ( cube_y[0] == cube_y[11] && cube_x[0] == cube_x[11] && appear[11] == 1 )
    | ( cube_y[0] == cube_y[12] && cube_x[0] == cube_x[12] && appear[12] == 1 )
    | ( cube_y[0] == cube_y[13] && cube_x[0] == cube_x[13] && appear[13] == 1 )
    | ( cube_y[0] == cube_y[14] && cube_x[0] == cube_x[14] && appear[14] == 1 )
    | ( cube_y[0] == cube_y[15] && cube_x[0] == cube_x[15] && appear[15] == 1 ) )

                        die_body<=1;

            else        //蛇运动过程中相邻模块之间坐标存在联系,依次代替
                    begin
                        cube_x[1] <= cube_x[0];
                        cube_y[1] <= cube_y[0];

                        cube_x[2] <= cube_x[1];
                        cube_y[2] <= cube_y[1];

                        cube_x[3] <= cube_x[2];
                        cube_y[3] <= cube_y[2];

                        cube_x[4] <= cube_x[3];
                        cube_y[4] <= cube_y[3];

                        cube_x[5] <= cube_x[4];
                        cube_y[5] <= cube_y[4];

                        cube_x[6] <= cube_x[5];
                        cube_y[6] <= cube_y[5];

                        cube_x[7] <= cube_x[6];
                        cube_y[7] <= cube_y[6];

                        cube_x[8] <= cube_x[7];
                        cube_y[8] <= cube_y[7];

                        cube_x[9] <= cube_x[8];
                        cube_y[9] <= cube_y[8];

                        cube_x[10] <= cube_x[9];
                        cube_y[10] <= cube_y[9];
```

```
cube_x[11] <= cube_x[10];
cube_y[11] <= cube_y[10];

cube_x[12] <= cube_x[11];
cube_y[12] <= cube_y[11];

cube_x[13] <= cube_x[12];
cube_y[13] <= cube_y[12];

cube_x[14] <= cube_x[13];
cube_y[14] <= cube_y[13];

cube_x[15] <= cube_x[14];
cube_y[15] <= cube_y[14];



case( direct )            //根据运动方向来决定蛇头的坐标

UP:begin
    if( cube_y[0] == 1 )
        die_wall <= 1;
    else
        cube_y[0] <= cube_y[0] - 1; //只要不是最上面，并处于 UP 状态，这时让其向下
end

DOWN: begin
    if( cube_y[0] == 28 )
        die_wall <= 1;
    else
        cube_y[0] <= cube_y[0] + 1;
end

LEFT:begin
    if( cube_x[0] == 1 )
        die_wall <= 1;
    else
        cube_x[0] <= cube_x[0] - 1;
end

RIGHT: begin
    if( cube_x[0] == 38 )
```

```
                    die_wall <= 1;
                else
                    cube_x[0] <= cube_x[0] + 1;
            end

            endcase

        end

    end

end

    always @( * ) begin

        direct_next = direct;

        case( direct )

        UP:
            begin
                if( change_to_left )
                    direct_next = LEFT;
                else if( change_to_right )
                    direct_next = RIGHT;
                else
                    direct_next = UP;
            end

        DOWN :
            begin
                if( change_to_left )
                    direct_next = LEFT;
                else if( change_to_right )
                    direct_next = RIGHT;
                else
                    direct_next = DOWN;
            end

        LEFT :
            begin
```

```
                if( change_to_up )
                    direct_next = UP;
                else if( change_to_down )
                    direct_next = DOWN;
                else
                    direct_next = LEFT;
            end

        RIGHT :
            begin
                if( change_to_up )
                    direct_next = UP;
                else if( change_to_down )
                    direct_next = DOWN;
                else
                    direct_next = RIGHT;
            end

        endcase

    end


    always @( posedge clk ) begin

        if( left_press == 1 ) begin
            change_to_left <= 1;
        end

        else if( right_press == 1 ) begin
            change_to_right <= 1;
        end

        else if( up_press == 1 ) begin
            change_to_up <= 1;
        end

        else if( down_press == 1 ) begin
            change_to_down <= 1;
        end

        else begin
            change_to_left <= 0;
            change_to_right <= 0;
```

```
            change_to_up <= 0;
            change_to_down <= 0;
        end

end


reg addcube_state; //小蛇吃方块

always @( posedge clk or negedge reset ) begin

        if( reset == 0 ) begin

            appear <= 16'b0000000000000111; //开始 3 个小方块存在

            cube_num <= 3; //初始 3 个小方块作为蛇
            addcube_state <= 0;

        end

        else begin

            case( addcube_state )

            0: begin
                if( add_cube ) begin

                    cube_num <= cube_num + 1;
                    appear[cube_num] <= 1;
                    addcube_state <= 1;

                end
            end

            1: begin

                if(!add_cube)
                    addcube_state <= 0;

            end

            endcase

        end
```

```
    end


    reg [3:0] lox;
    reg [3:0] loy;

    // determine the display
    always @( x_pos or y_pos ) begin

        if( x_pos >= 0 && x_pos < 640 && y_pos >= 0 && y_pos < 480 ) begin

    if( ( x_pos[9:4] == 0 | y_pos[9:4] == 0 | x_pos[9:4] == 39 | y_pos[9:4] == 29 ) )
                snake = WALL;

else if( x_pos[9:4] == cube_x[0] && y_pos[9:4] == cube_y[0] && appear[0] == 1 )
begin
                snake = ( die_flash == 1 ) ? HEAD : NONE;
            end
else if
( ( x_pos[9:4] == cube_x[1] && y_pos[9:4] == cube_y[1] && appear[1] == 1 )
    | ( x_pos[9:4] == cube_x[2] && y_pos[9:4] == cube_y[2] && appear[2] == 1 )
    | ( x_pos[9:4] == cube_x[3] && y_pos[9:4] == cube_y[3] && appear[3] == 1 )
    | ( x_pos[9:4] == cube_x[4] && y_pos[9:4] == cube_y[4] && appear[4] == 1 )
    | ( x_pos[9:4] == cube_x[5] && y_pos[9:4] == cube_y[5] && appear[5] == 1 )
    | ( x_pos[9:4] == cube_x[6] && y_pos[9:4] == cube_y[6] && appear[6] == 1 )
    | ( x_pos[9:4] == cube_x[7] && y_pos[9:4] == cube_y[7] && appear[7] == 1 )
    | ( x_pos[9:4] == cube_x[8] && y_pos[9:4] == cube_y[8] && appear[8] == 1 )
    | ( x_pos[9:4] == cube_x[9] && y_pos[9:4] == cube_y[9] && appear[9] == 1 )
| ( x_pos[9:4] == cube_x[10] && y_pos[9:4] == cube_y[10] && appear[10] == 1 )
| ( x_pos[9:4] == cube_x[11] && y_pos[9:4] == cube_y[11] && appear[11] == 1 )
| ( x_pos[9:4] == cube_x[12] && y_pos[9:4] == cube_y[12] && appear[12] == 1 )
| ( x_pos[9:4] == cube_x[13] && y_pos[9:4] == cube_y[13] && appear[13] == 1 )
| ( x_pos[9:4] == cube_x[14] && y_pos[9:4] == cube_y[14] && appear[14] == 1 )
| ( x_pos[9:4] == cube_x[15] && y_pos[9:4] == cube_y[15] && appear[15] == 1 ))
                snake = ( die_flash == 1 ) ? BODY : NONE;

            else
                snake = NONE;
        end

    end

endmodule
```

k) Game-控制模块

该模块通过按键（键盘）来设置重启、开始、进行和死亡四个状态和反应状态改变所需要的条件。

```
module Game_ctrl_unit(input clk,
input reset,
input key1_press,
input key2_press,
input key3_press,
input key4_press,
input hit_wall,
input hit_body,
output reg [1:0] game_status,
output reg die_flash,
output reg restart);
localparam RESTART = 2'b00;
localparam START = 2'b01;
localparam PLAY = 2'b10;
localparam DIE = 2'b11;
reg [31:0] clk_cnt;
always @( posedge clk or negedge reset )
  begin
    if( !reset    )
      begin
        game_status <= START;
        clk_cnt <= 0;
        die_flash <= 1;
        restart <= 0;
      end
else begin
         case( game_status )
    RESTART:begin
             if( clk_cnt <= 5 )
                      begin
               clk_cnt <= clk_cnt + 1;
               restart <= 1;
                      end
            else begin
               game_status <= START;
               clk_cnt <= 0;
               restart <= 0;
                     end
       end
    START: begin
```

```
    if( key1_press | key2_press | key3_press | key4_press )
                    begin
      game_status <= PLAY;
                     end
              end
     PLAY:begin
     if( hit_wall | hit_body )

      game_status <= DIE;
      end
    DIE:begin
            if( clk_cnt <= 200000000 ) begin
               clk_cnt <= clk_cnt + 1;
            if( clk_cnt == 25000000 )
               die_flash <= 0;
            else if( clk_cnt == 50000000 )
                  die_flash <= 1;
            else if( clk_cnt == 75000000 )
      die_flash <= 0;
            else if( clk_cnt == 100000000 )
      die_flash <= 1;
            else if( clk_cnt == 125000000 )
      die_flash <= 0;
            else if( clk_cnt == 150000000 )
      die_flash <= 1;
               end
            else
              begin
              die_flash <= 1;
              clk_cnt <= 0;
              game_status <= RESTART;
              end
          end
    endcase
end
end
endmodule
```

l) 苹果出现和被吃模块：

我们选择使用 random 的方式在 640*480 的显示屏上随机出现苹果，当蛇头遇到苹果时，add_cube。但是由于苹果的随机显示，很有可能发生苹果出现在蛇身上的情况，此时游戏虽然不会暂停，但是还是有一定的 BUG，需要进一步优化改良。

```
module Snake_eating_apple (

    input clk,
    input reset,

    input [5:0] head_x,
    input [5:0] head_y,

    output reg [5:0] apple_x,
    output reg [4:0] apple_y,

    output reg add_cube

);

    reg [31:0] clk_cnt;
    reg [10:0] random_num;

    always @( posedge clk )
        random_num <= random_num + 927;

    always @( posedge clk ) begin

        if( reset == 0 ) begin

            clk_cnt <= 0;

            apple_x <= 24;
            apple_y <= 10;

            add_cube <= 0;

        end

        else begin

            clk_cnt <= clk_cnt + 1;
```

```
                if( clk_cnt == 250000 ) begin

                    clk_cnt <= 0;
if( apple_x == head_x && apple_y == head_y )
          begin
  add_cube <= 1;
apple_x <= ( random_num[10:5] > 38 ) ? ( random_num[10:5] - 25 ) :( random_num[10:5] == 0 ) ? 1:random_num[10:5];
apple_y <= ( random_num[4:0] > 28 ) ? ( random_num[4:0] - 3 ) :( random_num[4:0] == 0 ) ? 1 : random_num[4:0];

          end
else
                    add_cube <= 0;

                end

          end

     end

endmodule
```

m) 管脚约束模块代码

```
set_property IOSTANDARD LVCMOS33 [get_ports clk0]
set_property IOSTANDARD LVCMOS33 [get_ports h_sync]
set_property IOSTANDARD LVCMOS33 [get_ports reset]
set_property PACKAGE_PIN P17 [get_ports clk0]
set_property PACKAGE_PIN D7 [get_ports h_sync]
set_property PACKAGE_PIN C4 [get_ports v_sync]
set_property IOSTANDARD LVCMOS33 [get_ports v_sync]
set_property PACKAGE_PIN R2 [get_ports reset]
set_property PACKAGE_PIN H1 [get_ports {sel[0]}]
set_property PACKAGE_PIN C1 [get_ports {sel[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sel[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sel[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sel[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {sel[3]}]
set_property PACKAGE_PIN B7 [get_ports {color_out[2]}]
set_property PACKAGE_PIN D8 [get_ports {color_out[1]}]
set_property PACKAGE_PIN E7 [get_ports {color_out[0]}]
set_property PACKAGE_PIN D5 [get_ports {seg_out[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg_out[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg_out[1]}]
```

```
set_property IOSTANDARD LVCMOS33 [get_ports {seg_out[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg_out[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg_out[4]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg_out[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg_out[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {seg_out[7]}]
set_property PACKAGE_PIN B2 [get_ports {seg_out[6]}]
set_property PACKAGE_PIN B3 [get_ports {seg_out[5]}]
set_property PACKAGE_PIN A1 [get_ports {seg_out[4]}]
set_property PACKAGE_PIN B1 [get_ports {seg_out[3]}]
set_property PACKAGE_PIN A3 [get_ports {seg_out[2]}]
set_property PACKAGE_PIN A4 [get_ports {seg_out[1]}]
set_property PACKAGE_PIN B4 [get_ports {seg_out[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {color_out[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {color_out[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {color_out[0]}]

set_property PACKAGE_PIN G2 [get_ports {sel[3]}]
set_property PACKAGE_PIN C2 [get_ports {sel[2]}]

set_property IOSTANDARD LVCMOS33 [get_ports ps2c]
set_property IOSTANDARD LVCMOS33 [get_ports ps2d]

set_property PACKAGE_PIN K5 [get_ports ps2c]
set_property PACKAGE_PIN L4 [get_ports ps2d]
```

四：实验结果与分析

（1） Game_unit_control 状态机如下所示：

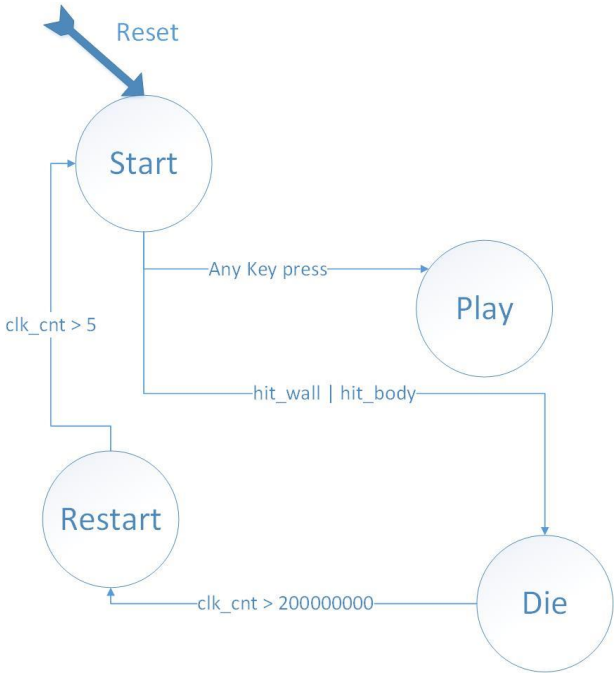

图 1： Game_unit_Control 状态转移图

（2） 贪吃蛇 RTL 结构图如下所示：

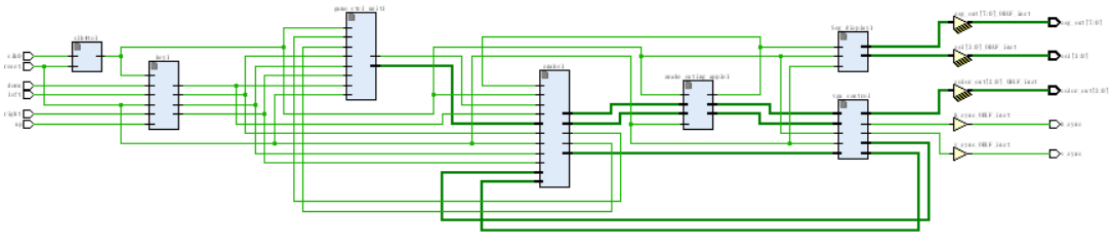

图 2：贪吃蛇游戏 RTL 结构图

（3） VGA 显示情况和板子显示情况：

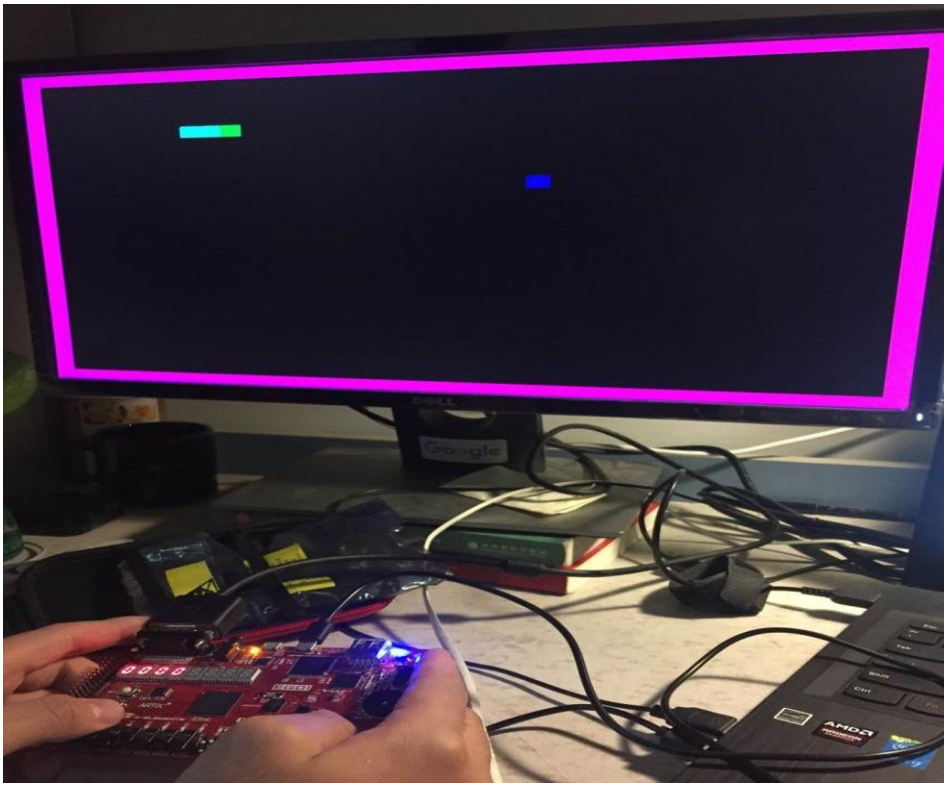

图 3： 游戏开始界面

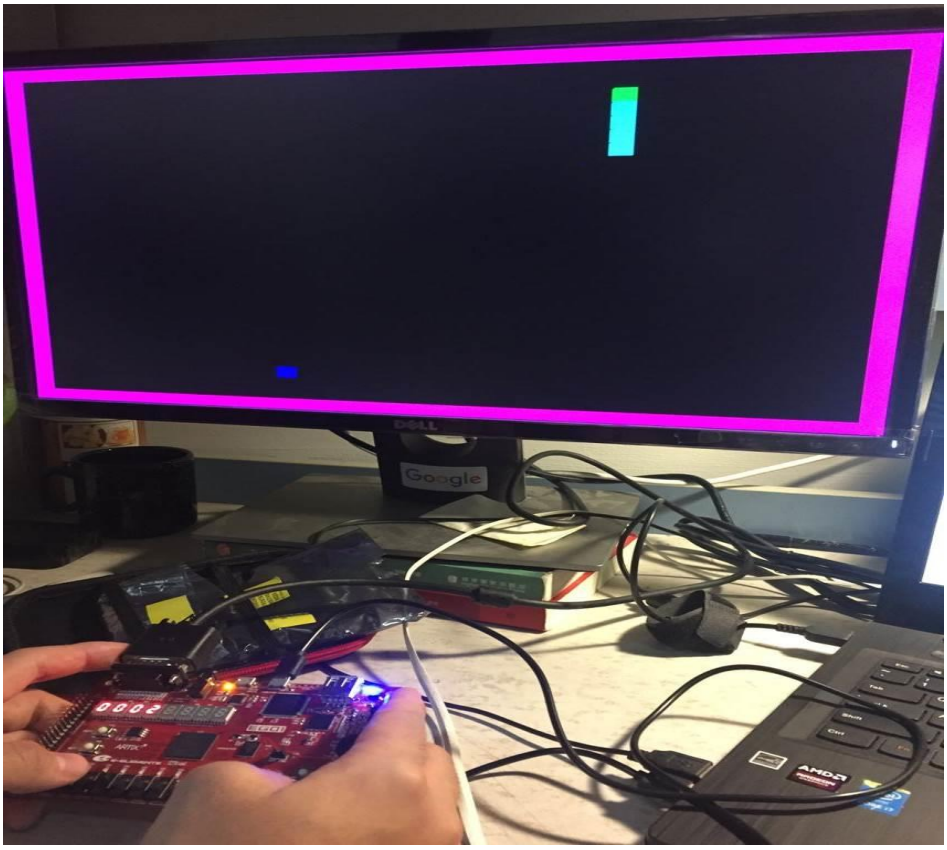

图 4：吃完“苹果”身体增长

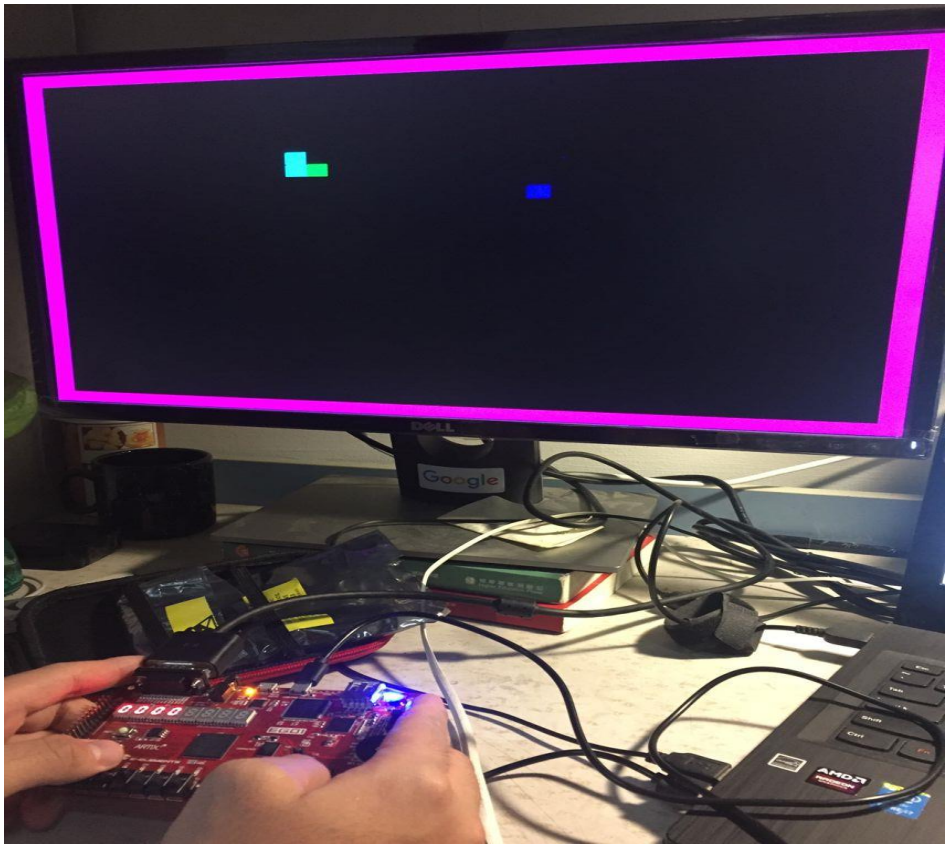

图 6：贪吃蛇转弯情况

图 7：贪吃蛇 hit wall 而 die

（4） Snake 状态改变状态机：

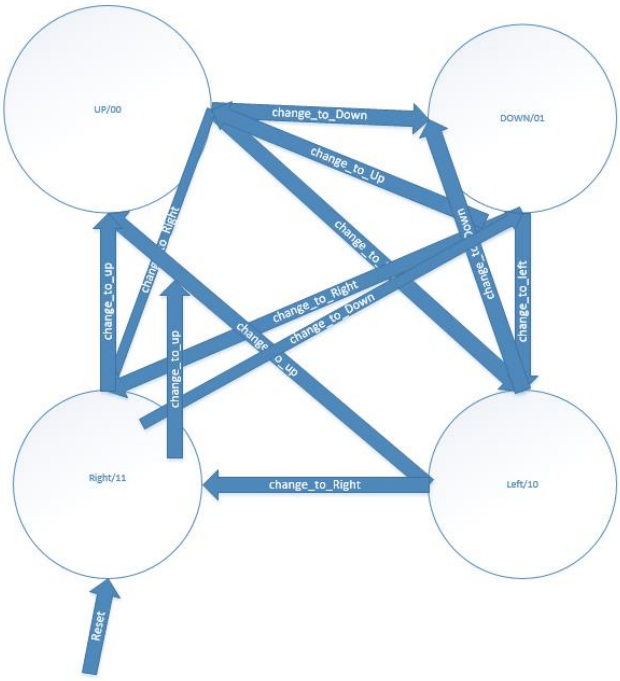

图 ： Snkae 状态改变状态机

五：结论（讨论）

1、实验结论

此实验成功在液晶显示器（LED 屏）上通过 4 拨码开关及 ps/2 标准键盘，来完成贪吃蛇的基本操作，并在 ego1 FPGA 的板子中的高 4 位数码管中显示得分，但由于程序不能反复迭代，只能设置贪吃蛇的长度最长为 16，并且由于食物是在整个画面中随机出现的，会有一定的可能出现在蛇的身上，造成一定的 bug，而且，其他复杂要求比如加快游戏进程等尚未涉及，需要进一步完善代码。但总体上，我们完成了整个贪吃蛇的基本内容，并成功在显示器及板子数码中显示，有一定的娱乐功能，达到了实验目的中设计一个复杂的数字逻辑系统的要求。同时，自己在游戏过程中，也发现自己的兴趣和价值。

2、讨论

《数字逻辑》是我们北京科技大学计算机专业的基础课，是一门实践性很强的课程，不仅要求我们掌握概念，而且还要求我们动手编程并进行上机调试、运行。

在这为期半个多月的时间内，我们完成了贪吃蛇的程序设计，这次设计是我们在本阶段学完理论课程之后对自己该方面的能力的一次很好的检验。我一直觉得，之前我学的知识最多也就是在做作业的时候才会用到，平时没有什么练习的机会，可是要学好这种语言，仅仅学习课本上的知识和做那种枯燥乏味的习题是远远不够的，更重要的是自己动手，动脑去思考去想方设法的为解决一个问题而不断进行调整和设计。经常编写程序，才能发现我们程序设计上的漏洞和自己的不足，并且在将来的应用中解决这些问题，不断提高自己转化知识的能力。 这次的课程设计是我第一次通过自己构思，和室友讨论并且不断查阅资料和咨询百度贴吧的大神与相关技术人员，最终完成了一项较大规模的程序。

在这次学生信息管理系统的设计中，我们首先对系统的整体功能进行了构思，然后用结构化分析方法进行分析，将整个系统清楚的划分为几个模块，再根据每个模块的功能编写代码。而且尽可能的将模块细分，将我们的任务准确的分开，以及商量我们在各个程序中是大概用什么方法，最后在进行函数的调用。在编写过程中，我观察到室友和其他同学编写一个相同过程的不同方法，才发现自己编写的程序仍存在很大优化空间，这更使我们的程序真正能在实际中的应用和发挥。

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，并且，由于 Verilog 语言是用逻辑来实现 vga 中的颜色显示，在显示中，出现了很多时候意想不到的颜色情况，于是便反复重新检查自己的逻辑关系，所以花了将近整整一天的时间才能在屏幕上显示出正确的图像。

通过这次设计，不仅巩固了我以前所学的知识，还更深刻掌握了Verilog语言。不论是算法思路到运行调试以及令人兴奋的可用程序，使我真正的体会到程序员生活的不易，想着以后每天面对电脑编程，不禁冒了一身冷汗，可是，当我们最终看见error为0，程序成功运行时，那喜悦更是让我们觉得欣慰，看见自己的程序能起到用处，心中满是自豪感！

六、教师评审

教师评语	实验成绩
签名： 日期：	