浙江大学

课程设计报告

中文题目 ： Verilog 实现贪吃蛇游戏

英文题目： GreedySnake Game based on Verilog

姓名/学号： 杨启航

指导教师： 施青松

参加成员： 杨启航

专业类别： 计算机科学与技术

所在学院： 计算机学院

论文提交日期 2018 年 1 月 10 日

摘要

本课程设计是数字逻辑设计课程的课程设计，主要使用硬件描述语言 Verilog HDL 在实验室的 SWORD Kintex-7 开发板上实现了基于数字系统的贪吃蛇游戏，并对实现中的各种细节包括系统结构设计的详尽分析、硬件模块设计详尽分析过程、各模块的硬件描述、仿真测试问题与解决方案等内容进行了较为详尽的描述。本课程设计综合使用了数字逻辑设计课程中所介绍的各个知识点，包括组合电路与时序电路、状态机、ROM 等，还使用了课程中未要求的 VGA 显示等知识点。

关键词： 贪吃蛇，数字逻辑设计，Verilog,课程设计

目录

第1章 绪论

1.1 贪吃蛇游戏设计背景

贪吃蛇游戏是一款经典的益智游戏，简单又耐玩。该游戏通过控制蛇头方向吃蛋，从而使得蛇变得越来越长.

用游戏把子上下左右控制蛇的方向，寻找吃的东西，每吃一口就能得到一定的积分，而且蛇的身子会越吃越长，身子越长玩的难度就越大，不能碰墙，不能咬到自己的身体，更不能咬自己的尾巴。

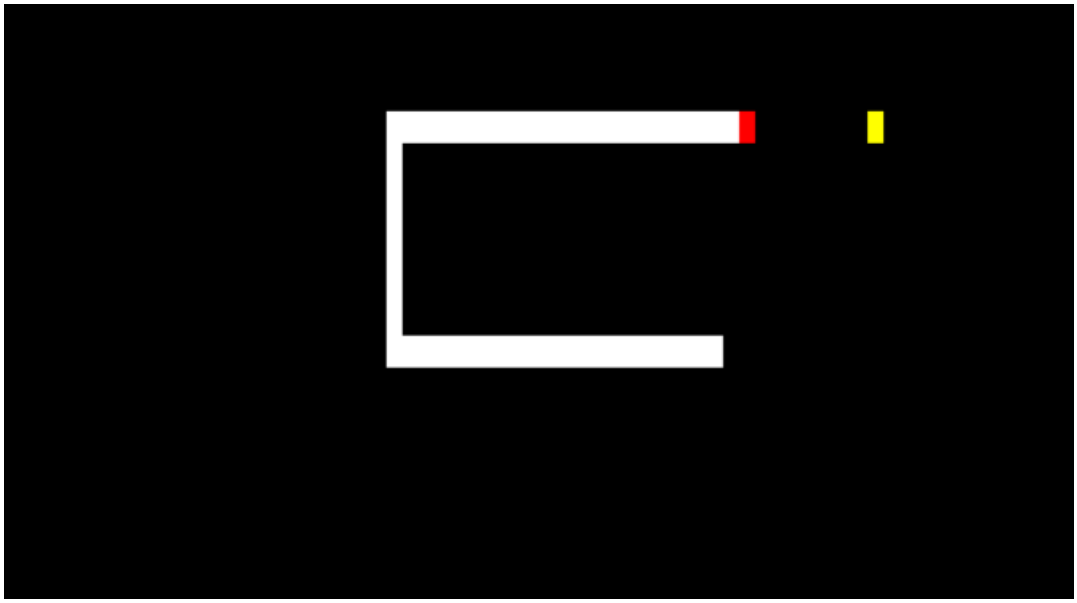

Figure 1 贪吃蛇游戏

1.2 国内外现况分析

贪吃蛇作为一款经典的益智游戏，在移动、PC 端都有各种类型的版本，至于基于数字系统的贪吃蛇我在查阅资料后发现，国内外均有相关的研究设计，但关于贪吃蛇的移动模块的设计，我有一些与以往资料不同的思路，希望能够尝试利用自己的新想法实现贪吃蛇的设计。

1.3 主要内容和难点

本课程设计希望设计基于数字系统的贪吃蛇游戏，通过 Verilog HDL 语言描述实现，硬件设备为实验室的 Sword Kintex7 开发板。游戏希望能够实现贪吃蛇的正常控制以及吃蛋、死亡判定等功能，并通过 VGA 接口在显示器上输出游戏过程的画面。

主要难点有三点，第一个是如何存储蛇以及食物的信息、第二个是如何进行蛇的移动，第三个则是如何通过 VGA 接口显示游戏画面。

第2章 贪吃蛇设计原理

2.1 贪吃蛇设计相关内容

本课程设计主要使用到了有限状态机、VGA 接口、阵列键盘等内容。有限状态机使游戏能够在几个不同的状态下切换（开始、游戏中、死亡等），VGA 接口用于游戏画面的显示，阵列键盘则用于游戏的控制。

下面简要说明：

2.1.1 有限状态机

状态机由状态寄存器和组合逻辑电路构成，能够根据控制信号按照预先设定的状态进行状态转移，是协调相关信号动作、完成特定操作的控制中心。状态机简写为 FSM（Finite State Machine），主要分为 2 大类：

第一类，若输出只和状态有关而与输入无关，则称为 Moore 状态机

第二类，输出不仅和状态有关而且和输入有关系，则称为 Mealy 状态机。

本工程的主要控制模块就利用有限状态机实现在不同的状态下切换，为了简洁，使用了 Mealy 状态机。

2.1.2VGA 接口

VGA 是 IBM 在 1987 年随推出的一种视频传输标准，具有分辨率高、显示速率快、颜色丰富等优点，在彩色显示器领域得到了广泛的应用。

信号线	定义
HS	列同步信号（3.3V 电平）
VS	行同步信号（3.3V 电平）
R	红基色（0~0.714V 模拟信号）
G	绿基色（0~0.714V 模拟信号）
B	蓝基色（0~0.714V 模拟信号）

Figure 2 VGA 信号线

为了在显示器上显示我们需要的图案，我们需要使用标准 VGA 接口中的上述信号线。HS 与 VS 用于控制显示器的显示频率，R、G 与 B 则用于控制显示器当前像素的颜色。

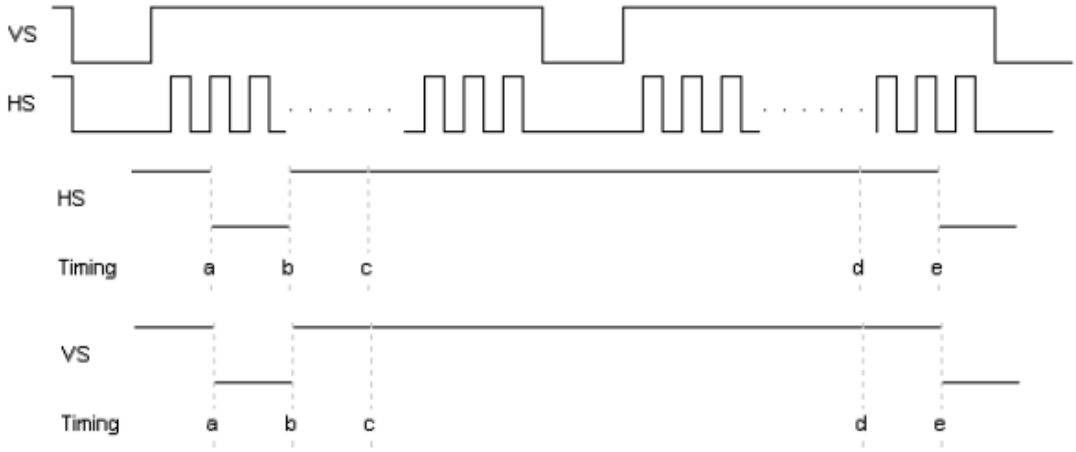

Figure 3 VGA 标准时序

上图是VGA的标准时序。不难看出，列同步信号与行同步信号的时序都分为4段。a-b段称为同步期，此时同步信号为低电平；b-c段称为消隐后肩，c-d段称为显示期，是数据的有效区域，d-e段称为消隐前肩，这三段的同步信号为高电平。显示器就是根据列同步信号与行同步信号的周期，来决定显示的分辨率、频率等参数。不同分辨率与频率对应的VS-HS数值，可以通过查表得到。

我们只需要在一个 VS-HS 完成一个周期的时间内，按从左上角到右下角的顺序输出各个像素点的颜色，就能在显示器上显示出对应的图案。需要注意的是，VGA 颜色的输出必须严格遵循时序，只能在 c-d 段输出非 0 数据，否则会对显示造成干扰。

实验室使用的 VGA 显示为 640*480@Hz，查表即可得到其时序。

Scanline part	Pixels	Time [µs]
Visible area	640	25.422045680238
Front porch	16	0.63555114200596
Sync pulse	96	3.8133068520357
Back porch	48	1.9066534260179
Whole line	800	31.777557100298

Figure 4 Horizontal timing (line)

Frame part	Lines	Time [ms]
Visible area	480	15.253227408143
Front porch	10	0.31777557100298
Sync pulse	2	0.063555114200596
Back porch	33	1.0486593843098
Whole frame	525	16.683217477656

Figure 5 Vertical timing (frame)

2.2 贪吃蛇游戏设计方案

因为使用 VGA 接口显示画面，所以我们需要给屏幕上的每一个像素点一个坐标，但是直接以像素点为单位来进行游戏是不合适的，因为这样会使游戏中各要素（蛇、食物、墙壁等）很难用肉眼察觉，第二则是使得整个工程的运算量增大。因此我们手动“降低”分辨率——以像素块为单位来进行游戏，由于屏幕为 640x480 分辨率，故把 16*16 的像素块作为基本单位，将屏幕变成 40*30 的点阵。

根据我们想要设计实现的效果，可以将整个工程主要分成如下 4 个模块：

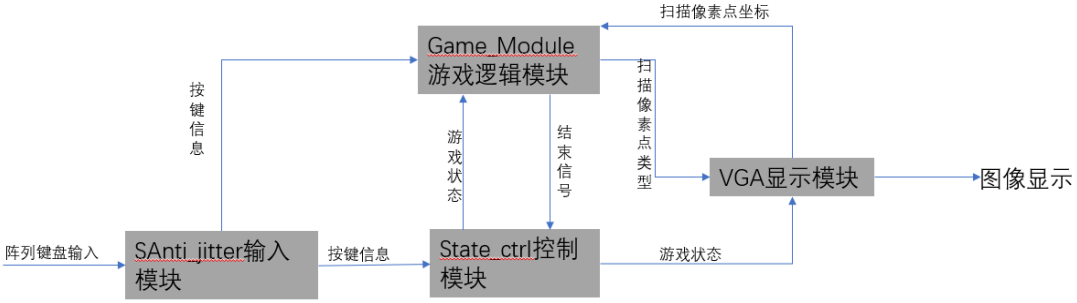

Figure 6 贪吃蛇游戏结构框图

1. State_ctrl 模块

包含状态机，用于控制在不同状态下，其他各模块按照不同的逻辑运行。游戏一共设置四种状态——开始状态、游戏状态、死亡状态、胜利状态。

在开始状态下，屏幕显示贪吃蛇的开始画面，按下任何按钮则进入游戏状态。

游戏状态是游戏的主要状态，在此状态里，屏幕上会显示出贪吃蛇的游戏画面，贪吃蛇会根据阵列键盘输入调整移动方向，每移动一步则进行游戏是否结束以及是否吃到食物的判断。

当贪吃蛇达到最大长度时候进入胜利状态。当贪吃蛇吃到身体或者撞墙时进入失败状态。

由于 IP 核 ROM 容量限制，胜利和失败都使用一副静态画面(Game Over)显示，在 Game Over 延时显示一段时间后，状态会自动切换回开始状态，主要通过一个计数器来实现延时。

2. GameModule 模块

是维护游戏正确运行的模块。GameModule 模块控制蛇的移动、增长、进行游戏是否结束的判定、控制食物的生成以及判断蛇是否吃到食物。

贪吃蛇的身体信息存储在寄存器组中，每次移动时在寄存器单元之间移位赋值，并计算新的蛇头存储在第一个寄存器中。由于贪吃蛇身体只会增长不会缩短，因此利用另外一个寄存器来存储各节身体是否显示，当蛇身增长时，增加显示的节数。判断游戏是否结束则主要通过将蛇头坐标与其他需要比较的点的坐标比较判断。

由于本模块控制贪吃蛇的移动并在移动一次后进行相关判断，因此主要使用

的时钟频率必须较小，否则一开始游戏，贪吃蛇就会很快撞墙死亡。但接受阵列键盘输入的子模块需要使用一个较快的时钟，否则容易丢失输入,导致控制不灵敏。

3.VGA_Display 模块 用于与 VGA 接口进行通信显示，完成产生扫描信号并输出每个像素点的 RGB 值的工作。它从 GameModule 模块获取当前像素点信息，从 State_ctrl 模块获取当前的状态，在非游戏状态调用 IP 核显示静态画面。
4.SAnti_jitter 模块 用于获取阵列键盘的输入并实现去抖动功能。
还有辅助的 clk_div 模块用于时钟分频。

2.3 贪吃蛇游戏硬件设计

下面简要介绍各模块功能并给出源代码，详细的介绍在 3.1 节给出。

2.3.1 顶层模块 它主要调用了 4 个子模块：SAnti_jitter 模块 IP 核调用获取阵列键盘的输入和去抖动；GameModule 模块维护游戏过程的正常运行；State_ctrl 模块保证工程在几个状态中正确切换。VGA_Display 模块调用 VGA 接口在屏幕上显示游戏画面。

```
module Snake_Top(
       input clk,
       input [15:0] SW,
       input RSTN,
       input [3:0] BTN,
       output wire [4:0]K_ROW,
       output wire H_sync,
       output wire V_sync,
       output wire [3:0] R,
       output wire [3:0] G,
       output wire [3:0] B,
       output wire Buzzer,
       output wire RDY,
       output wire CR
   );
    wire [31:0] clkDiv;//时钟信号
    wire [3:0] Button_Out;//阵列键盘去抖动输入
    wire [1:0] state;//游戏当前状态
    wire [15:0] SW_OK;
    wire [9:0] X_Pos,Y_Pos;//扫描像素点坐标
    wire rst;
    wire disValid,refresh,grow,die,win;
    wire [2:0]pixel;//扫描像素点的类型
    wire [6:0]length;//蛇的长度
    assign Buzzer=1'b1;
```

```
 SAnti_jitter(.clk(clk),
              .RSTN(RSTN),
              .readn(1),
              .Key_y(BTN),
              .Key_x(K_ROW),
              .SW(SW),
              .Key_ready(RDY),
              .BTN_OK(Button_Out),
              .SW_OK(SW_OK),
              .CR(CR),
              .rst(rst)
              );


GameModule   M3(.clk(clkDiv[24]),
                .rst(rst),
                .mov_clk(clkDiv[10]),
                .state(state),
                .Button_Control(BTN),
                .X_Pos(X_Pos),
                .Y_Pos(Y_Pos),
                .length(length),
                .pixel(pixel),
                .die(die),
                .win(win)
                );

State_ctrl   M4(.clk(clkDiv[5]),
                .rst(rst),
                .BTN(BTN),
                .die(die),
                .state(state),
                .win(win)
                );
clk_div     M5(.clk(clk),
               .clkdiv(clkDiv)
                );


VGA_Display  M6(.clk(clkDiv[1]),
                .rst(rst),
                .pixel(pixel),
                .X_Pos(X_Pos),
                .Y_Pos(Y_Pos),
```

```
                        .state(state),
                        .H_sync(H_sync),
                        .V_sync(V_sync),
                        .R(R),
                        .G(G),
                        .B(B)
                        );
endmodule
```

2.3.2 SAnti_jitter 模块

直接调用了 IP 核实现该模块的功能

```
module SAnti_jitter(input wire clk,
                            input wire RSTN,
                            input wire readn,
                            input wire [3:0]Key_y,
                            output reg[4:0] Key_x,
                            input wire[15:0]SW,
                            output reg[4:0] Key_out,
                            output reg     Key_ready,
                            output reg[3:0] pulse_out,
                            output reg[3:0] BTN_OK,
                            output reg[15:0]SW_OK,
                            output reg    CR,
                            output reg    rst
                           );

Endmodule
```

2.3.3State_ctrl 模块 包含一个状态机，作为游戏逻辑的控制模块

```
`timescale 1ns / 1ps
module State_ctrl(
    input clk,
    input rst,
    input [3:0] BTN,
     input die,
     input win,
     output reg[1:0] state
    );
     reg [31:0] cnt;
     localparam StartState=2'b10,//开始状态
                 DieState=2'b00,//死亡状态
                 PlayState=2'b01,//游戏状态
                 WinState=2'b11;//胜利状态

    initial begin
```

```
    state <= StartState;//初始化为开始状态
    cnt <= 0;
end

always@(posedge clk or posedge rst)begin
    if(rst) begin
        state <= StartState;
    end
    else begin
        case(state)
        WinState:begin//死亡后延时若干时间回到初始状态
            if(cnt <= 10000000)begin
                cnt <= cnt + 1;
                state <= state;
            end
            else begin
                cnt<=0;
                state<=StartState;//延时结束，回到初始状态
            end
        end

        DieState:begin//胜利后延时若干时间回到初始状态
            if(cnt <= 10000000)begin
                cnt <= cnt + 1;
                state <= state;
            end
            else begin
                cnt <= 0;
                state <= StartState;//延时结束，回到初始状态
            end
        end
        PlayState:
            if(die) state <= DieState;//进入死亡状态
            else if(win) state <= WinState;//胜利状态
            else state <= state;

        StartState://在开始状态时，按键进入游戏状态
            if(BTN != 4'b1111) state <= PlayState;
            else state <= state;
        default:
            state <= state;
        endcase
    end
end
```

```
endmodule
```

2.3.4GameModule 模块

作为维护游戏运行的主要模块，它包含以下三个部分

1. 子模块 CreatEgg：随机产生食物，判断蛇是否吃到食物
2. 子模块 Moving_control:接受阵列键盘的输入，将当前前进方向作为输出
3. 实现蛇的移动、游戏是否结束的判断以及输出 VGA 当前扫描像素点的信息

Moving_control 模块

```
`timescale 1ns / 1ps

module Moving_control(
    input clk,
    input rst,
   input [3:0] Control,//[3]下 [2]右 [1]左 [0]上
    output reg [1:0]direction//蛇头的移动方向 00 上,01 左,10
右,11 下
   );
    localparam up=2'b00,
                left=2'b01,
                right=2'b10,
                down=2'b11;

   initial begin//初始化
      direction<=down;
   end
    always @(posedge clk or posedge rst) begin
      if(rst)begin//复位蛇头向下
          direction <= down;
          end
      else begin
          if(!Control[0]&&direction!=down) direction <=
up;//蛇头向上，且保证原方向不是向下
          else if(!Control[1]&&direction!=right) direction
<= left;//左
          else if(!Control[2]&&direction!=left) direction
<= right;//右
          else if(!Control[3]&&direction!=up) direction <=
down;//下
          else direction <= direction;//保持原方向
      end
   end


endmodule
```

CreatEgg 模块

```
`timescale 1ns / 1ps

module CreatEgg (
    input clk,
    input rst,
    input state,
    input [5:0] Head_X,
    input [5:0] Head_Y,
    input refresh,
    output reg [5:0] Food_X,
    output reg [5:0] Food_Y,
    output reg Grow
);
    localparam PlayState = 2'b01;
    reg [11:0] Random_Generator;

    initial begin//初始化
        Food_X <= 10;
        Food_Y <= 20;
        Grow <= 0;
    end

    always @( posedge clk or posedge rst ) begin
        if(rst) begin
            Food_X <= 24;
            Food_Y <= 10;
            Grow <= 0;
        end

        else  if(state==PlayState)begin//判断食物是否被吃并随机生
成食物位置
                //使用计数器取模获得食物随机位置
                Random_Generator <= Random_Generator + 666;
                if( (Food_X == Head_X && Food_Y == Head_Y) |
refresh ) begin
                    if(Food_X == Head_X && Food_Y == Head_Y)//
食物被吃
                        Grow <= 1;

                    //随机生成食物位置 已避免食物生成在墙里

                    Food_X <= (Random_Generator[11:6])%38+1'b1;
                    Food_Y <= Random_Generator[5:0]%28+1'b1;
```

```
                    end
                    else
                        Grow <= 0;
                end
            else begin//非游戏状态，初始化食物位置以及增长信号
                Food_X <= 10;
                Food_Y <= 20;
                Grow <= 0;
            end
        end
    endmodule
```

Game_Module

```
`timescale 1ns / 1ps

module GameModule (
    input clk,//模块主要使用的时钟
    input rst,
    input mov_clk,//根据输入改变移动方向的时钟
    input [1:0]state,//游戏状态
    input [3:0] Button_Control,//控制蛇移动方向
    input [9:0] X_Pos,//VGA 当前扫描像素点的 X 坐标 0~640
    input [9:0] Y_Pos,//VGA 当前扫描像素点的 Y 坐标 0~480
    output reg [6:0] length,
    output reg [2:0]pixel,//VGA 扫描像素点的状态
    output reg die,//游戏失败
    output reg win//游戏胜利
    );

    wire [1:0] Direction;//蛇前进方向
    wire Grow;//蛇是否增长一个单位
    wire [5:0] Food_X;//食物坐标
    wire [5:0] Food_Y;
    wire [5:0] Head_X,Head_Y;//蛇头坐标
    localparam UP = 2'b00,//上下左右
                DOWN = 2'b11,
                LEFT = 2'b01,
                RIGHT = 2'b10,
                PlayState = 2'b01,//游戏状态
                NONE = 3'b000,//扫描像素点的类型，无
                HEAD = 3'b010,//蛇头
                BODY = 3'b001,//蛇身
                WALL = 3'b001,//墙壁
                FOOD = 3'b100;//食物
```

```
reg [5:0] Unit_X[15:0];//每个单元是蛇的一节身体的 X 坐标
reg [5:0] Unit_Y[15:0];//身体的 Y 坐标
reg [15:0] Unit_Exist;//判断上面各个单元是否真的存储蛇身体信息
reg refresh;//食物刷新信号
assign Head_X = Unit_X[0];//蛇头存储在第一个单元中
assign Head_Y = Unit_Y[0];

initial begin//初始化 蛇长度为 2
   Unit_X[0] <= 6'd20;
   Unit_Y[0] <= 6'd15;
   Unit_X[1] <= 6'd20;
   Unit_Y[1] <= 6'd14;
   die <= 0;
   win <= 0;
   refresh <= 0;
   Unit_Exist <= 16'b0000000000000011;
   length <= 2;
end

   CreatEgg M0(.clk(clk),//随机产生食物,判断蛇是否吃到食物,输出蛇是否
增长的信号
               .rst(rst),
               .refresh(refresh),
               .Head_X(Head_X),
               .Head_Y(Head_Y),
               .Food_X(Food_X),
               .Food_Y(Food_Y),
               .Grow(Grow)
               );

Moving_control M1(.clk(mov_clk),
                  .rst(rst),
                  .Control(Button_Control),
                  .direction(Direction)
                  );

   //当蛇吃到食物时, 蛇长度增长
always @( posedge clk or posedge rst ) begin
   if( rst ) begin//复位
      Unit_Exist <= 16'b0000000000000011;
      length <= 2;
```

```
        end
        else begin
        if(state==PlayState)//游戏状态
               if( Grow==1 ) begin
                  length <= length + 1;//长度增一
                  Unit_Exist[length] <= 1;//寄存器有效单元增一
               end
               else begin
               end
        else begin//其它状态  初始化长度信息
               Unit_Exist <= 16'b0000000000000011;
               length <= 2;
            end
        end
    end

    always @( posedge clk or posedge rst)begin
        if(rst) begin//复位
            Unit_X[0] <= 6'd20;
            Unit_Y[0] <= 6'd15;
            Unit_X[1] <= 6'd20;
            Unit_Y[1] <= 6'd14;
            die <= 0;
            win <= 0;
        end
        else begin//首先判断蛇是否达成游戏结束的条件，未达成时，进行蛇的移动
            if( state== PlayState ) begin
               if(length==16)//判断是否达到最大长度
                  win<=1;
               else if( ( Direction == UP && Unit_Y[0] == 1 )
                     | ( Direction == DOWN && Unit_Y[0] == 28 )
                     | ( Direction == LEFT && Unit_X[0] == 1 )
                     | ( Direction == RIGHT && Unit_X[0] == 38 ) )
                     die <= 1;//判断是否撞墙
               else if( ( Unit_Y[0] == Unit_Y[1] && Unit_X[0] ==
Unit_X[1] && Unit_Exist[1] == 1 )
                        | ( Unit_Y[0] == Unit_Y[2] && Unit_X[0] ==
Unit_X[2] && Unit_Exist[2] == 1 )
                        | ( Unit_Y[0] == Unit_Y[3] && Unit_X[0] ==
Unit_X[3] && Unit_Exist[3] == 1 )
                        | ( Unit_Y[0] == Unit_Y[4] && Unit_X[0] ==
Unit_X[4] && Unit_Exist[4] == 1 )
                        | ( Unit_Y[0] == Unit_Y[5] && Unit_X[0] ==
Unit_X[5] && Unit_Exist[5] == 1 )
```

```
                        | ( Unit_Y[0] == Unit_Y[6] && Unit_X[0] ==
Unit_X[6] && Unit_Exist[6] == 1 )
                        | ( Unit_Y[0] == Unit_Y[7] && Unit_X[0] ==
Unit_X[7] && Unit_Exist[7] == 1 )
                        | ( Unit_Y[0] == Unit_Y[8] && Unit_X[0] ==
Unit_X[8] && Unit_Exist[8] == 1 )
                        | ( Unit_Y[0] == Unit_Y[9] && Unit_X[0] ==
Unit_X[9] && Unit_Exist[9] == 1 )
                        | ( Unit_Y[0] == Unit_Y[10] && Unit_X[0] ==
Unit_X[10] && Unit_Exist[10] == 1 )
                        | ( Unit_Y[0] == Unit_Y[11] && Unit_X[0] ==
Unit_X[11] && Unit_Exist[11] == 1 )
                        | ( Unit_Y[0] == Unit_Y[12] && Unit_X[0] ==
Unit_X[12] && Unit_Exist[12] == 1 )
                        | ( Unit_Y[0] == Unit_Y[13] && Unit_X[0] ==
Unit_X[13] && Unit_Exist[13] == 1 )
                        | ( Unit_Y[0] == Unit_Y[14] && Unit_X[0] ==
Unit_X[14] && Unit_Exist[14] == 1 )
                        | ( Unit_Y[0] == Unit_Y[15] && Unit_X[0] ==
Unit_X[15] && Unit_Exist[15] == 1 ) )
                        die <= 1;//判断是否吃到身体
                else  begin//上述判断均为假说明游戏继续,下面先判断食物是否刷
新在蛇身体上
                    if( ( Food_Y == Unit_Y[1] && Food_X == Unit_X[1]
&& Unit_Exist[1] == 1 )
                        |  (  Food_Y  ==  Unit_Y[2]  &&  Food_X  ==
Unit_X[2] && Unit_Exist[2] == 1 )
                        |  (  Food_Y  ==  Unit_Y[3]  &&  Food_X  ==
Unit_X[3] && Unit_Exist[3] == 1 )
                        |  (  Food_Y  ==  Unit_Y[4]  &&  Food_X  ==
Unit_X[4] && Unit_Exist[4] == 1 )
                        |  (  Food_Y  ==  Unit_Y[5]  &&  Food_X  ==
Unit_X[5] && Unit_Exist[5] == 1 )
                        |  (  Food_Y  ==  Unit_Y[6]  &&  Food_X  ==
Unit_X[6] && Unit_Exist[6] == 1 )
                        |  (  Food_Y  ==  Unit_Y[7]  &&  Food_X  ==
Unit_X[7] && Unit_Exist[7] == 1 )
                        |  (  Food_Y  ==  Unit_Y[8]  &&  Food_X  ==
Unit_X[8] && Unit_Exist[8] == 1 )
                        |  (  Food_Y  ==  Unit_Y[9]  &&  Food_X  ==
Unit_X[9] && Unit_Exist[9] == 1 )
                        |  (  Food_Y  ==  Unit_Y[10]  &&  Food_X  ==
Unit_X[10] && Unit_Exist[10] == 1 )
                        |  (  Food_Y  ==  Unit_Y[11]  &&  Food_X  ==
```

```
Unit_X[11] && Unit_Exist[11] == 1 )
                        | ( Food_Y == Unit_Y[12] && Food_X ==
Unit_X[12] && Unit_Exist[12] == 1 )
                        | ( Food_Y == Unit_Y[13] && Food_X ==
Unit_X[13] && Unit_Exist[13] == 1 )
                        | ( Food_Y == Unit_Y[14] && Food_X ==
Unit_X[14] && Unit_Exist[14] == 1 )
                        | ( Food_Y == Unit_Y[15] && Food_X ==
Unit_X[15] && Unit_Exist[15] == 1 ) )
                        refresh <= 1;//判断是否吃到身体
                        else refresh <= 0;
              //在寄存器单元间进行移位赋值,然后将新蛇头信息存储在第一个单元来
实现移动
                        Unit_X[1] <= Unit_X[0];
                        Unit_Y[1] <= Unit_Y[0];
                        Unit_X[2] <= Unit_X[1];
                        Unit_Y[2] <= Unit_Y[1];
                        Unit_X[3] <= Unit_X[2];
                        Unit_Y[3] <= Unit_Y[2];
                        Unit_X[4] <= Unit_X[3];
                        Unit_Y[4] <= Unit_Y[3];
                        Unit_X[5] <= Unit_X[4];
                        Unit_Y[5] <= Unit_Y[4];
                        Unit_X[6] <= Unit_X[5];
                        Unit_Y[6] <= Unit_Y[5];
                        Unit_X[7] <= Unit_X[6];
                        Unit_Y[7] <= Unit_Y[6];
                        Unit_X[8] <= Unit_X[7];
                        Unit_Y[8] <= Unit_Y[7];
                        Unit_X[9] <= Unit_X[8];
                        Unit_Y[9] <= Unit_Y[8];
                        Unit_X[10] <= Unit_X[9];
                        Unit_Y[10] <= Unit_Y[9];
                        Unit_X[11] <= Unit_X[10];
                        Unit_Y[11] <= Unit_Y[10];
                        Unit_X[12] <= Unit_X[11];
                        Unit_Y[12] <= Unit_Y[11];
                        Unit_X[13] <= Unit_X[12];
                        Unit_Y[13] <= Unit_Y[12];
                        Unit_X[14] <= Unit_X[13];
                        Unit_Y[14] <= Unit_Y[13];
                        Unit_X[15] <= Unit_X[14];
                        Unit_Y[15] <= Unit_Y[14];
```

```
                    case( Direction ) //根据运动方向产生新的蛇头
                        UP:Unit_Y[0] <= Unit_Y[0] - 1;
                        DOWN:Unit_Y[0] <= Unit_Y[0] + 1;
                        LEFT:Unit_X[0] <= Unit_X[0] - 1;
                        RIGHT: Unit_X[0] <= Unit_X[0] + 1;
                    endcase
                end
            end
            else begin//非游戏状态，进行蛇身信息以及游戏状态信号的初始化
            Unit_X[0] <= 6'd20;
            Unit_Y[0] <= 6'd15;
            Unit_X[1] <= 6'd20;
            Unit_Y[1] <= 6'd14;
            die <= 0;
            win <= 0;
            refresh <= 0;
            end
        end
    end

    //返回VGA当前扫描坐标点的像素类型
    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 )
                pixel <= WALL;//当前像素为墙壁
            else  if(  X_Pos[9:4]  ==  Unit_X[0]  &&  Y_Pos[9:4]  ==
Unit_Y[0] && Unit_Exist[0] == 1 )
                pixel <= HEAD;//蛇头
            else if(X_Pos[9:4] == Food_X && Y_Pos[9:4] == Food_Y)
                pixel <= FOOD;//食物
            else  if(  (  X_Pos[9:4]  ==  Unit_X[1]  &&  Y_Pos[9:4]  ==
Unit_Y[1] && Unit_Exist[1] == 1 )
                    ||  (  X_Pos[9:4]  ==  Unit_X[2]  &&  Y_Pos[9:4]  ==
Unit_Y[2] && Unit_Exist[2] == 1 )
                    ||  (  X_Pos[9:4]  ==  Unit_X[3]  &&  Y_Pos[9:4]  ==
Unit_Y[3] && Unit_Exist[3] == 1 )
                    ||  (  X_Pos[9:4]  ==  Unit_X[4]  &&  Y_Pos[9:4]  ==
Unit_Y[4] && Unit_Exist[4] == 1 )
                    ||  (  X_Pos[9:4]  ==  Unit_X[5]  &&  Y_Pos[9:4]  ==
Unit_Y[5] && Unit_Exist[5] == 1 )
                    ||  (  X_Pos[9:4]  ==  Unit_X[6]  &&  Y_Pos[9:4]  ==
Unit_Y[6] && Unit_Exist[6] == 1 )
```

```
                    || ( X_Pos[9:4] == Unit_X[7] && Y_Pos[9:4] ==
Unit_Y[7] && Unit_Exist[7] == 1 )
                    || ( X_Pos[9:4] == Unit_X[8] && Y_Pos[9:4] ==
Unit_Y[8] && Unit_Exist[8] == 1 )
                    || ( X_Pos[9:4] == Unit_X[9] && Y_Pos[9:4] ==
Unit_Y[9] && Unit_Exist[9] == 1 )
                    || ( X_Pos[9:4] == Unit_X[10] && Y_Pos[9:4] ==
Unit_Y[10] && Unit_Exist[10] == 1 )
                    || ( X_Pos[9:4] == Unit_X[11] && Y_Pos[9:4] ==
Unit_Y[11] && Unit_Exist[11] == 1 )
                    || ( X_Pos[9:4] == Unit_X[12] && Y_Pos[9:4] ==
Unit_Y[12] && Unit_Exist[12] == 1 )
                    || ( X_Pos[9:4] == Unit_X[13] && Y_Pos[9:4] ==
Unit_Y[13] && Unit_Exist[13] == 1 )
                    || ( X_Pos[9:4] == Unit_X[14] && Y_Pos[9:4] ==
Unit_Y[14] && Unit_Exist[14] == 1 )
                    || ( X_Pos[9:4] == Unit_X[15] && Y_Pos[9:4] ==
Unit_Y[15] && Unit_Exist[15] == 1 ))
                pixel <= BODY;//蛇身
            else
                pixel <= NONE;//空白
        end
    end

endmodule
```

2.3.5 VGA_Display 模块

该模块主要负责 VGA 的显示，在游戏状态下，它向 Game_Module 模块发送当前扫描的像素点的坐标，得到该像素类型的返回值，从而显示游戏画面其他状态则从 IP 核生成的 ROM 中读取像素信息显示预先加载好的静态画面。

它主要包含以下两个模块:

1. VGA_signal 模块:根据 VGA 的信号时序生成扫描信号
2. VGA_Color 模块:确定当前扫描像素点的 RGB 值。

VGA_signal 模块

```
`timescale 1ns / 1ps
module VGA_signal(
        input clk,
        input rst,
        output reg[9:0] HS_cnt,//行计数
        output reg[9:0] VS_cnt,//场计数
        output H_sync,//行同步信号（低有效）
        output V_sync,//场同步信号（低有效）
```

```
output  disValid//是否为有效显示区域,1 为有效
);
localparam H_PIXEL=10'd799,//行像素 800
           VLINE=10'd520,//行数 521
           //下面是有效显示区域的行列坐标边界值
           UP_BOUND=31,
           DOWN_BOUND=510,
           LEFT_BOUND=144,
           RIGHT_BOUND=783;

reg DisValid;//displayValid 当前是否为有效显示区域
assign disValid=DisValid;
initial begin//初始化
   HS_cnt<=0;
   VS_cnt<=0;
   DisValid<=0;
end

//列计数与行同步
assign H_sync=(HS_cnt<96)?0:1;
always@(posedge clk or posedge rst)
begin
   if(rst)
      HS_cnt<=0;
   else if(HS_cnt==H_PIXEL)//当前行扫描完毕
      HS_cnt<=0;
   else
      HS_cnt<=HS_cnt+1;
end


//行计数与场同步
assign V_sync=(VS_cnt<2)?0:1;
always@(posedge clk or posedge rst)
begin
   if(rst)
      VS_cnt<=0;
   else if(HS_cnt==H_PIXEL)
   begin
      if(VS_cnt==VLINE)
         VS_cnt<=0;
      else
         VS_cnt<=VS_cnt+1;
   end
```

```
            else
                VS_cnt<=VS_cnt;
        end

        always@(posedge clk)begin

    if((HS_cnt>=LEFT_BOUND)&&(HS_cnt<=RIGHT_BOUND)&&(VS_cnt<DOWN_B
OUND)&&(VS_cnt>UP_BOUND))
                DisValid<=1;//有效显示区
            else
                DisValid<=0;//非有效显示区
        end
endmodule
```

VGA_Color 模块

```
`timescale 1ns / 1ps

module VGA_Color(
        input clk,
        input rst,
        input disValid,
        input [1:0]state,
        input [2:0]pixel,//即将显示的像素类别
        input [11:0] EggPos,//蛋的位置(以像素块为单位)
        input [9:0] X_Pos,//行计数
        input [9:0] Y_Pos,//场计数
        output reg [3:0] R,//颜色显示
        output reg [3:0] G,
        output reg [3:0] B
        );
        wire [15:0]data2;
        wire [15:0]data1;
        reg [18:0]addr;
        localparam StartState=2'b10,//开始状态
                    DieState=2'b00,//死亡状态
                    PlayState=2'b01,//游戏状态
                    WinState=2'b11;//胜利状态
GAME_END  ROM1(.clka(clk),.addra(addr),.douta(data1));//三个分别是时
钟信号、地址线、数据输出
Start_State ROM2(.clka(clk),.addra(addr),.douta(data2));
//Win ROM3(.clka(clk),.addra(addr),.douta(data3));

        always@(posedge clk or posedge rst)begin
            if(rst)begin
                        R<=0;
```

```
                        G<=0;
                        B<=0;
          end
          else begin
              if(disValid)begin
                  if( X_Pos >= 0 && X_Pos < 640 && Y_Pos >= 0 &&
Y_Pos < 480 ) begin
                      case(state)
                      PlayState://游戏状态
                         if(pixel==3'b100)begin
                                R<=4'b0000;
                                G<=4'b0000;
                                B<=4'b1111;
                         end
                         else if( pixel == 3'b000 )//00 显示没有东西的
地方，白色
                            begin
                               R<=4'b1111;
                               G<=4'b1111;
                               B<=4'b1111;
                         end
                         else if( pixel == 3'b001 )//01 显示墙壁
                         begin
                                R<=0;
                                G<=4'b1111;
                                B<=0;
                         end
                         else if(pixel == 3'b010)//10 显示蛇头
                         begin
                                R<=4'b1111;
                                G<=0;
                                B<=0;
                         end
                         else if(pixel == 3'b110)//11 显示蛇身
                         begin
                                R<=0;
                                G<=4'b1111;
                                B<=4'b1111;
                         end
                      DieState:begin//死亡状态 显示 GameOver
                         addr<=Y_Pos*640+X_Pos;
                         R<=data1[14:11];
                         G<=data1[8:5];
                         B<=data1[3:0];
```

```
                    end
                WinState:begin//胜利状态，显示 GameOver
                    addr<=Y_Pos*640+X_Pos;
                    R<=data1[14:11];
                    G<=data1[8:5];
                    B<=data1[3:0];
                    end
                StartState:begin//开始状态，显示开始游戏画面
                    addr<=Y_Pos*640+X_Pos;
                    R<=data2[14:11];
                    G<=data2[8:5];
                    B<=data2[3:0];
                    end
                endcase
            end
        else //非有效显示区域
                begin
                    R<=0;
                    G<=0;
                    B<=0;
                end
    end
  end
endmodule
```

VGA_Display 模块

```
`timescale 1ns / 1ps

module VGA_Display(
    input clk,
    input rst,
    input [1:0]state,
    input [2:0]pixel,//即将显示的像素类别
    input [11:0] EggPos,//蛋的位置(以像素块为单位)
    output wire [9:0] X_Pos,//行计数
    output wire [9:0] Y_Pos,//场计数
    output wire H_sync,//行同步信号（低有效）
    output wire V_sync,//场同步信号（低有效）
    output wire [3:0] R,//颜色显示
    output wire [3:0] G,
    output wire [3:0] B
    );
    localparam
            UP_BOUND=31,//Y 方向有效显示区域的下界
```

```
                LEFT_BOUND=144;//X 方向有效显示区域的下界

        wire disValid;//displayValid 当前是否为有效显示区域
        wire [9:0] HS_cnt,VS_cnt;//行坐标和列坐标(包括行同步、后沿、有效数
据区、前沿)
        assign X_Pos = HS_cnt - LEFT_BOUND;//有效显示区域的 X Y 坐标
        assign Y_Pos = VS_cnt - UP_BOUND;


        VGA_signal M1(.clk(clk),
                      .rst(rst),
                      .HS_cnt(HS_cnt),
                      .VS_cnt(VS_cnt),
                      .H_sync(H_sync),
                      .V_sync(V_sync),
                      .disValid(disValid)
                      );
        VGA_Color M2(.clk(clk),
                     .rst(rst),
                     .disValid(disValid),
                     .pixel(pixel),
                     .state(state),
                     .EggPos(EggPos),
                     .X_Pos(X_Pos),
                     .Y_Pos(Y_Pos),
                     .R(R),
                     .G(G),
                     .B(B)
                     );

endmodule
```

2.3.6 辅助模块

该模块包含时钟分频模块，为其他模块提供合适的时钟信号

```
    module clk_div(
    input clk,
    output reg [31:0] clkdiv
    );
     always@(posedge clk)begin
        clkdiv<=clkdiv+1'b1;
    end
endmodule
```

第3章 贪吃蛇游戏设计实现

3.1 实现方法

下面以模块为单位详细介绍了各模块设计的思路以及方法，并给出了实现过程中难点的解决方案。

3.1.1State_ctrl 模块

该模块主要完成游戏中几个状态（开始、游戏、死亡、胜利）的正确切换。初始界面为开始界面，按下任意按键即可开始游戏，当游戏中蛇咬到自己或撞墙时，游戏结束进入死亡状态，当蛇达到最大长度时，玩家胜利，进入胜利状态。在胜利和死亡状态停留一段时间后，状态将自动转换为开始状态以便开始下一轮游戏。

该模块的实现并不困难，主要是利用到了 case 语句。利用一个寄存器来存储游戏状态 state，根据当前的 state 和输入的按键信号、死亡信号(die)、胜利信号(win)即可实现不同状态的转换。

在开始状态下，若检测到按键信号显示有按键被按下，就切换到游戏状态。

在游戏状态下，当 die 信号和 win 信号有一个为 1 时，则游戏状态结束，切换到死亡/胜利状态。

死亡/胜利状态下，延时功能的实现则利用一个计数器 cnt，当计数过程即延时过程，计数到一定数值时将计数器清零并使状态转换到开始状态。

rst 通过将状态强制转换为开始状态实现复位。

3.1.2 Clk_div 模块

该模块只是像以前实验课一样简单地将时钟分频，作为其他模块的时钟信号。

3.1.3 GameModule 模块

该模块分成了三个部分，其中 Moving_Control 模块用于根据键盘输入控制蛇的前进方向，CreatEgg 模块判断蛇是否吃到食物以及随机生成食物坐标，剩下的部分实现蛇的移动、蛇长度的增长、判断游戏是否结束（蛇吃到自己/撞墙/达到最大长度），判断食物是否生成在了蛇身上，并向 VGA 模块传输其当前扫描的像素点的信息。

Moving_Control 模块：利用一个两位的寄存器存储蛇的前进方向，将它能表示的四个值分别编码为上下左右，根据键盘的输入重新赋值，同时还需要避免蛇头一步转 180° 的情况发生（如蛇向右走，此时按下向左的按键应该是无效的），这里的实现方法是综合按键输入和当前方向寄存器的值进行判断，发生如上情况时方向寄存器保持其原来的值。需要注意的是贪吃蛇使用的主要时钟的频率很低，一般在 1Hz 左右，如果该模块也使用这个时钟，会导致阵列按键输入灵敏度降低，不能接收到按下时间较短的输入，因此需要用一个较快的时钟来驱动该模块。

CreatEgg 模块：想要判断蛇是否吃到食物，只需要判断蛇头坐标是否与食物坐标相等，相等时随机生成食物，并将蛇身增长的信号 grow 输出。

而随机生成食物，原本我使用了 Random 函数，但在下板过程中发现利用 Random

函数来随机生成食物无效——蛇头即使与食物重合，也没有食物也不会重新刷新，未能找到原因。最后解决方式是使用一个由时钟信号驱动的计数器来生成随机数，将计数器的存储值相应取模就可以得到食物的坐标。X 方向有 40 个单位，第 0 和 39 个单位是墙壁，因此存储值 Mod 38 + 1 即可得到不在墙壁内的一个随机的食物 X 坐标，同理 Y 坐标为存储值 mod 28 + 1.但是有一点需要注意，食物不生成在墙壁里，还有可能生成在蛇身内部。解决方式是在判断游戏结束的部分加上一个语句块来判断食物是否与蛇身重合，结果通过 refresh 信号输送给 CreatEgg 模块，信号为 1 时重新生成食物。

剩下的部分我都写在了 Game_Module 模块内部，主要又可以分成以下几个部分：返回 VGA 扫描像素点信息、蛇长度的增长、蛇的移动以及游戏是否结束的判定。

首先我们需要明确的是蛇的坐标信息是如何存储的。在我的第一版程序中，我是想要使用一个位宽为 MaxLength*12 的寄存器来存储蛇身信息，每 12 位表示蛇的一段身体，高六位存储 X 坐标，低六位存储 Y 坐标，蛇的移动通过寄存器内部整体左移 12 位并将新的蛇头存储在最低 12 位中来实现，关于蛇身的显示以及蛇头与蛇身的碰撞判断，则通过一个位宽为 MaxLength*12 的辅助寄存器来实现，在判断开始前，将蛇身信息赋值给辅助寄存器，然后每次判断一节蛇身，判断完当前段，将寄存器内部右移 12 位，将下一节需要判断的蛇身存储在辅助寄存器的最低 12 位。这个想法最终失败了，屏幕中甚至显示不出蛇身，只能显示蛇头。我推测是因为这个过程是移一次判断一次，而非同步判断，所以当 MaxLength 比较大时，这个过程的时延会很大，可能导致电路不稳定，故无法正确显示。

第二种解决方式是通过向量数组。因为 Verilog 对于向量数组的操作不是很方便，比如对数组某单元的向量的部分位进行操作就不够便捷。所以把一节蛇身的 X、Y12 位数据分开成 X 和 Y 分别存储在两个向量数组中。向量位宽为 6，数组大小 16，数组大小即贪吃蛇的最大长度。

想要实现蛇的移动，可以把向量数组做移位赋值，即依次把每个单元的向量赋值给它左边那个单元，然后把计算出的新蛇头位置赋值给最右边的单元的向量。

但是这样做需要考虑一个问题：向量数组中会有单元内向量存储着一个坐标信息，但它不是蛇身的有效单元（比如蛇长度不变时，在 t+1 时刻存储着 t 时刻蛇尾坐标的单元），这样的单元的坐标信息并不能作为蛇身的一部分显示。

解决方式很简单，基于这样一个原理——蛇身只能保持不变或者变长，而不会缩短。即存储蛇身信息的有效单元数量不会减少。因此我们可以把所有有效单元集中在一起，每当长度增加时，在它们后面再加上一个有效单元。实现方式是另用一个寄存器记录数组中各个单元是否为有效单元，当长度为 L 时，第 0~L-1 个单元是有效单元，蛇头为第 0 个单元。做关于蛇身的相关判断时，需要考虑的就是它们存储的坐标值。

解决了蛇身信息的存储问题后，上面想要实现的功能就比较简单了。

返回 VGA 扫描像素点信息：将像素点信息依次与墙壁、食物、蛇头、蛇身比较，出现相等时返回相关类型，否则就表示此处为无物体。需要注意的是像素点坐标组成的是 640*480 点阵，而我们游戏基于 40*30 点阵，因此对于传入的 10 位像素点坐标信息，只需要取最高的六位比较，最低四位正好组成 16*16 的像素块，无论它们的值如何，在游戏中都被认为是同一个点。

蛇长度的增长：如同上文提到的，利用一个寄存器控制记录蛇身信息的向量数组中的有效单元数，当蛇身增长时，增加有数单元数。

蛇的移动：上文已提到，向量数组做移位赋值，将新蛇头计算后存入第一个单

元。

游戏是否结束：分成三部分，达到最大长度——胜利、撞墙——失败、吃到自己——失败。前面两个好解决，第一个比较当前长度和最大长度，第二个比较蛇头坐标和墙壁坐标。第三个的实现方式是将“蛇头坐标与向量数组每个单元的坐标做是否相等的判断”与上“该单元是否有效”，然后将各单元运算结果做或运算，当最后结果为 1 时，说明吃到了自己，否则没吃到自己。

3.1.4VGA_Display 模块

VGA_Display 模块主要包含 VGA_signal 模块和 VGA_Color 模块，前者生成 VGA 时序，后者生成当前扫描 RGB 的颜色信息。

VGA_signal 模块:首先查表得到有效显示区域的行列坐标边界值。根据 VGA 协议来实现生成场同步信号和行同步信号。一般可以根据模板代码根据自己的需要进行修改。

VGA_Color 模块：首先根据输入的 state 模块判断当前属于电路属于什么状态，这决定了 RGB 值是从 ROM 中获取还是根据 Game_Module 的输入生成。在非游戏状态，简单地把地址输入 ROM，然后获取输出并相对应地传递给 RGB。在游戏状态，则根据 Game_Module 输入的像素类型，利用 case 语句为 RGB 赋值。在静态图片生成时，我是将图片转换成 RGB565 的 16 位 coe 文件，但是只打听到了 12 位 RGB 引脚的定义，所以图片颜色可能不太正常。

3.2 实现过程

在设计贪吃蛇游戏的过程中，我的实现步骤主要如下：

1. 设计完成 VGA_Display 模块以及时钟分频辅助模块，并完成两个 IP 和生成的 ROM 的调用，调试知道屏幕能正常显示图形，这样方便后续设计的验证调试。
2. Game_Module 及其子模块、State_ctrl 模块的初步设计。
3. 完成 SAnti_jitter 和 State_ctrl 模块的设计，保证阵列键盘能够成功地输入，控制模块能正确地完成状态转换。
4. 进一步利用屏幕显示来验证、改进 Game_Module 模块的设计。
5. 测试各种功能，修改小 Bug.

   主要需要仿真的是 State_ctrl 模块,VGA 模块只需要物理验证显示器是否正常就可以得到结果，Game_Module 模块因为情况比较复杂，而且由于食物是随机生成的原因，食物坐标不好确定，关于长度、吃食物判定等的仿真验证比较困难，而通过物理验证已经可以验证我们想要验证的情况，因此不需要进行仿真模拟。

   最终的顶层模块的 RTL 逻辑图如下：

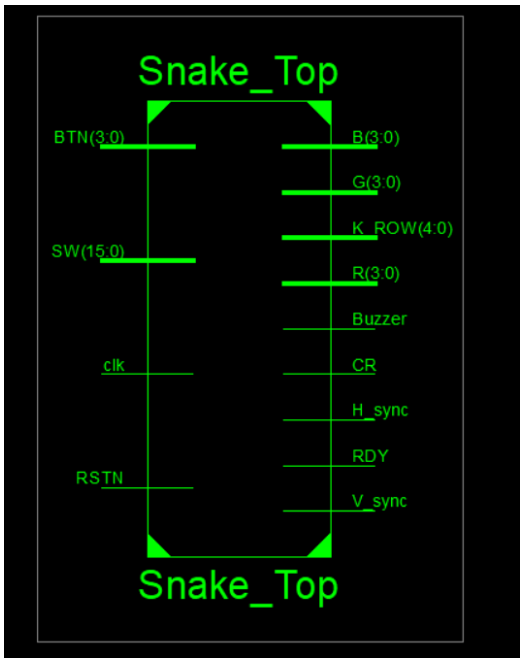

Figure 7 顶层模块的 RTL 逻辑图

因为 SAnti_jitter 调用了 IP 核，有些不需要的输出我没有进行改动。

clk 表示输入的 100MHz 时钟信号，RSTN 表示重置按钮信号， H_sync 与 V_sync 是 VGA 接口的行同步信号与场同步信号，R、G、B 是颜色信号，Buzzer 用于关闭 Arduino 子板上的蜂鸣器。SW 可用于后续改进功能的加入——如调整贪吃蛇速度、暂停游戏等，因为时间比较紧张，我来不及完成这些小功能的加入。

3.3 仿真与调试

3.3.1 State_ctrl 模块的仿真调试

State_ctrl 模块仿真调试的重点在于该模块是否能根据输入完成整个工程状态的正确切换与输出以及想要实现的游戏结束延时一定时间后切换回开始状态的功能。

因此根据需求输入以下仿真代码

```
fork
forever #5 clk=~clk;
begin
    #10;
    BTN=4'b0010;//验证按按钮从开始状态切换到游戏状态
    #20;
    BTN=4'b0000;
    rst=1;//验证 reset
    #10;
    rst=0;
    BTN=4'b0001;
    #10;
    die=1;//验证游戏状态切换死亡状态
    #10;
```

```
                die=0;//验证从死亡状态延时后切回开始状态(这里为了仿真改成
cnt<=2)
                #40;
                BTN=4'b0001;
                #10;
                win=1;//验证游戏状态切换到胜利状态
                #10;
                win=0;//验证胜利状态延时后切换回开始状态
                #10;
            end
              join
    end
```

仿真结果如下：

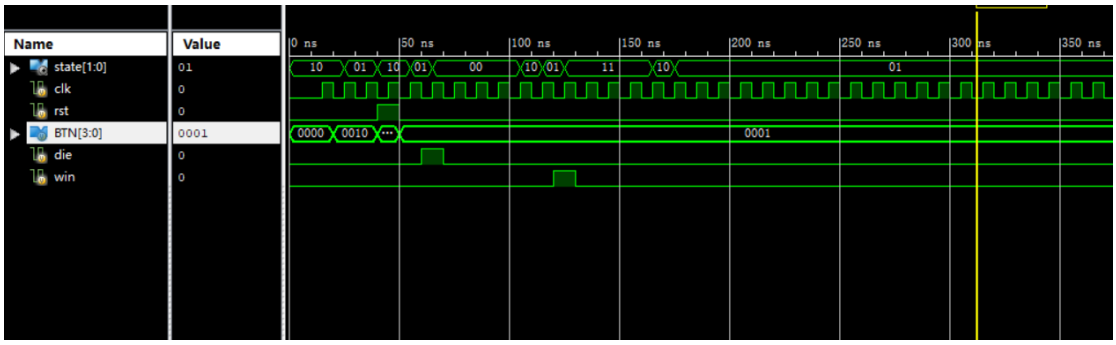

Figure 8 state_ctrl 仿真结果

可以看到仿真结果是符合理论预期的，该模块根据输入正确的进行了状态转换与输出。

在仿真过程中出现的问题主要集中在延时切换的问题。第一次仿真时，结果显示进入死亡状态后，没有延时就直接再次进入开始状态，而进胜利状态后，则切换不回开始状态。经过检查思考后，发现了问题所在：死亡状态无延时是因为我定义延时计数的 cnt 后未给它赋初值，导致第一次进入 DieState 的计数判断时，判断 cnt<1000000 为假，直接完成切换。而其后的 WinState 由于之前 DieState 将 cnt 赋值为 0，故开始计数，但由于我计数结束值设置过大，因此在有限的仿真周期里未能看到状态切换。因此做出的改动就是 1.初始化时将 cnt 赋初值 0　2.仿真时将计数结束值改为 2，便于观察验证，之后再改回。完成这些修改后，仿真的结果就符合理论预期了。

第4章 系统测试验证与结果分析

4.1 功能测试

根据本课程设计希望达成的功能，我对最终成果进行了测试。以下为测试项目与结果。

测试项目	结果
在开始界面按下任意按键进入游戏状态	PASS
在任意时刻长按复位按钮重新开始游戏	PASS
在游戏界面通过阵列键盘正确控制蛇的移动	PASS
蛇无法通过一次按键转头180°	PASS
蛇撞墙进入游戏结束画面	PASS
蛇吃到自己进入游戏结束画面	PASS
蛇吃到食物时，长度增1，食物重新刷新位置	PASS
蛇达到最大长度16时游戏结束	PASS
结束界面显示一段时间后自动切换回开始界面	PASS
食物刷新在蛇身上时，重新刷新	PASS

Figure 9 测试项目与结果

4.2 技术参数测试

本设计不涉及关于技术参数相关的设计，不进行该测试。

4.3 结果分析

从上述的测试结果可以看到，本设计成功实现了贪吃蛇游戏的基本功能，能够成功地进行一盘简单的贪吃蛇游戏。但是确实还有改进的空间，比如可以利用 SW 开关来实现一些其他增加游戏性的小功能，比如改变贪吃蛇速度（改变传入 GameModule 模块的时钟即可），还可以利用 LED 管加入计分功能等。但是因为期末时间很紧张，加上我之前设计第一版花时间很多，最后走入死胡同需要推倒重来，所以没有时间来完善这一些功能，这也是一点遗憾。

4.4 系统演示与操作说明

4.4.1 操作说明

游戏只有 5 个输入，分别是阵列键盘的四列用于控制贪吃蛇的方向，红色的 RSTN 按钮则作为复位按钮。

各按键对应方向如下：

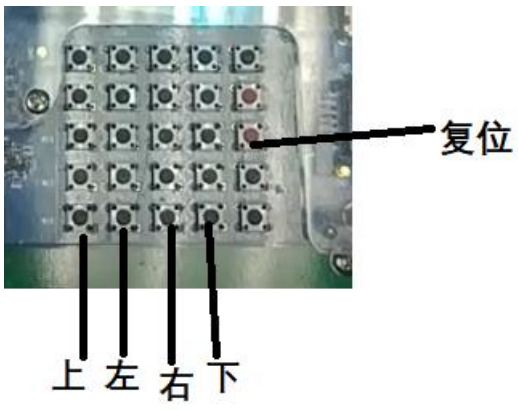

Figure 10 操作说明

4.4.2 系统演示

下面是游戏开始界面：

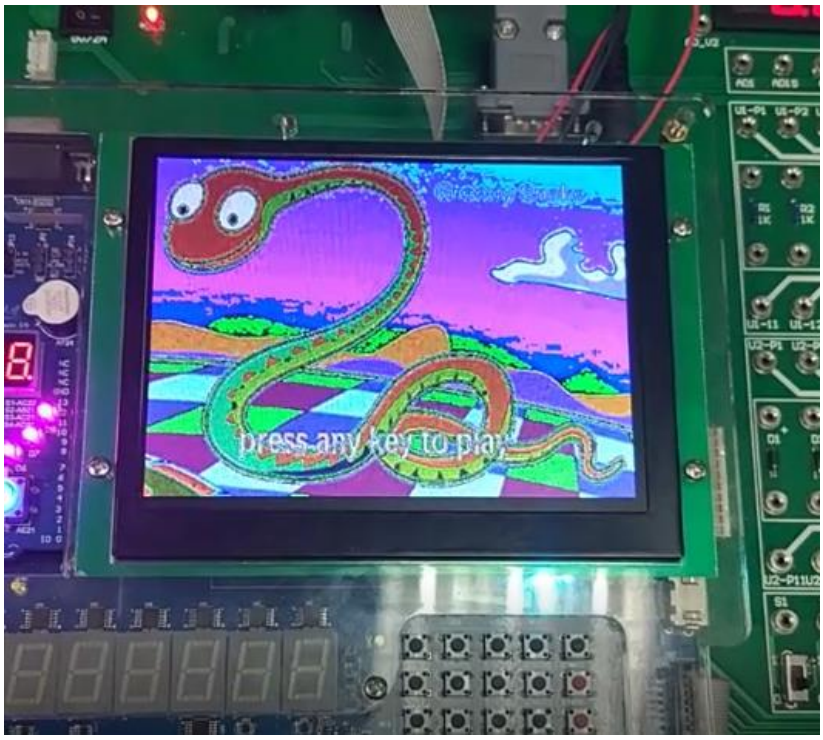

Figure 11 游戏开始界面

下面是游戏画面：

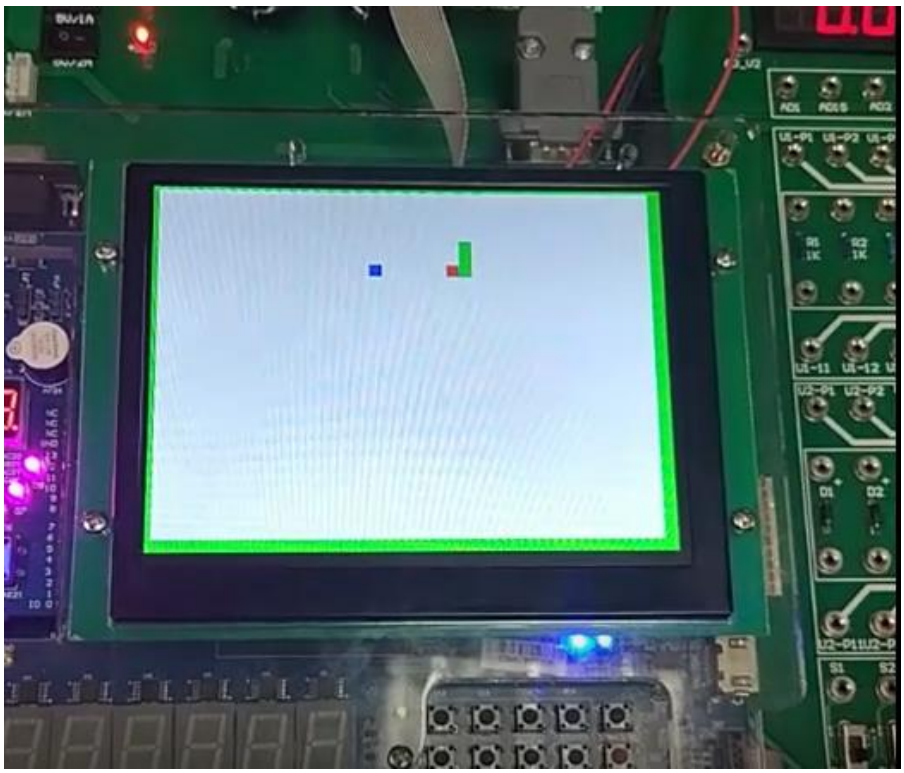

Figure 12 游戏画面

蓝色的是食物，周围一圈绿色是墙壁，红色为蛇头，蛇头后面绿色是蛇身。

下面是游戏结束画面:

Figure 13 游戏结束画面

该画面持续一段时间后会自动切换回游戏开始画面。

第5章 结论与展望

这份课程设计确确实实带给我和平时实验课不一样的感受。平时实验课往往照着PDF，依葫芦画瓢，最多再加一些思考就能够完成。但是这份课程设计却不一样，老师提供的东西很少很少，大多数都需要自己去寻找——创意、设计思路、需要使用的模块等等。“实践是检验真理的唯一标准”，这份课程设计正是对课堂所讲理论知识很好的补充，它甚至还推动着我们去学习课外的东西,例如VGA接口的使用。

这份课程设计也让我对硬件描述语言和软件设计的区别有了更深的理解，我们往往以软件设计的思维模式去描述电路，结果就是过于理想化导致无法得到正确的结果。硬件描述需要考虑的东西更多，比如时延、电路的计算资源等等。

除此以外，在完成这份课程设计后，我对Verilog的理解也更加深入了，之前哪怕做了十多次实验，我也依然未能完全明白wire和reg的区别，所以在设计的初期吃了很多苦头，不过现在已经弄懂了。

这样的课程设计不是每门课都有，想要很好地完成一份设计难度并不低，但是就是在这一步步自己的探索、失败、尝试之中，我们的水平逐渐地提高了，这是比依葫芦画瓢地做几次实验更有价值。

这份报告模板有些啰嗦，许多内容重复出现，希望能简化。