

课程设计报告

**中文题目 ：** **基于数字系统的贪吃蛇游戏**

**英文题目：** **[Gluttonous Snake](http://dict.youdao.com/w/Gluttonous Snake/" \l "keyfrom=E2Ctranslation)**

姓名/学号： 施能/3140100193

指导教师： 施青松

参加成员： 施能

专业类别： 地理信息科学

所在学院： 地球科学学院

**论文提交日期 2017 年 1 月 9 日**

摘要

这个设计是在sword平台下用Verilog语言模仿完成的《贪吃蛇》游戏，大家应该早已熟悉这款游戏，该游戏实现的就是玩家操控一条细长的直线（俗称[蛇](https://zh.wikipedia.org/wiki/%E8%9B%87" \o "蛇)或[虫](https://zh.wikipedia.org/wiki/%E8%99%AB" \o "虫)），它会不停前进，玩家只能操控蛇的头部朝向（上下左右），一路拾起触碰到之物（或称作“豆”），并要避免触碰到自身或者其他障碍物。每次贪吃蛇吃掉一件食物，它的身体便增长一些。游戏设计大致分为四面都有墙(都不可穿越)以及某部分的墙可以穿越，以及四面墙都可以穿越的模式。本次课程设计实现的是一个最基本的贪吃蛇游戏。

**关键词：**　Verilog、蛇、食物、障碍、VGA、得分

目录

[摘要 ii](#_Toc7224)

[第1章 绪论 5](#_Toc22784)

[1.1贪吃蛇设计背景 5](#_Toc24276)

[1.2国内外现况分析 5](#_Toc13312)

[1.3 主要内容和难点 5](#_Toc28403)

[第2章 贪吃蛇设计原理 5](#_Toc873)

[2.1 贪吃蛇设计相关内容 5](#_Toc22536)

[2.1.1 VGA原理 6](#_Toc15961)

[2.1.2 按键防抖动 6](#_Toc20555)

[2.1.3 七段码显示分数 6](#_Toc14193)

[2.2 贪吃蛇设计方案 6](#_Toc10240)

[2.2.1 游戏的整体结构 6](#_Toc11647)

[2.3 贪吃蛇硬件设计 9](#_Toc31720)

[2.3.1时钟分频模块 9](#_Toc1711)

[2.3.2按键去抖动模块 10](#_Toc3142)

[2.3.3七段码显示模块 10](#_Toc12780)

[2.3.3 VGA驱动模块 12](#_Toc25897)

[第3章 贪吃蛇设计实现 15](#_Toc18583)

[3.1 实现方法 15](#_Toc10705)

[3.2 实现过程 19](#_Toc6855)

[3.2.1 Game\_Ctrl\_Unit模块 19](#_Toc9320)

[3.2.2 Snake模块 19](#_Toc25257)

[3.2.3 Snake\_Eatting\_Food模块 20](#_Toc1133)

[3.2.4 VGA\_Control模块 21](#_Toc32696)

[3.2.5 顶层模块设计 21](#_Toc18815)

[3.3仿真与调试 22](#_Toc24808)

[实现中出现的问题 22](#_Toc1380)

[最终的仿真结果 22](#_Toc15610)

[3.3.1 Game\_Control\_Unit模块 22](#_Toc14218)

[3.3.2 Snake模块 23](#_Toc29326)

[3.3.3 Snake\_Eatting\_Food模块 26](#_Toc21847)

[第4章 系统测试验证与结果分析 26](#_Toc30130)

[4.1 功能测试 26](#_Toc28143)

[4.2 技术参数测试 27](#_Toc7934)

[4.3 结果分析 27](#_Toc3437)

[4.4 系统演示与操作说明 27](#_Toc4223)

[第5章 结论与展望 30](#_Toc25244)

图目录

[图 1 贪吃蛇游戏状态机 8](#_Toc6656)

[图 2 贪吃蛇移动方向状态机 9](#_Toc31889)

[图 3 模块整体结构图 10](#_Toc16196)

[图 4 Display模块结构图 12](#_Toc18023)

[图 5 MC14495\_ZJU模块原理图 12](#_Toc26304)

[图 6 Display模块原理图 13](#_Toc31075)

[图 7 VGA控制时序 14](#_Toc3924)

[图 8 VGA有效区域 14](#_Toc19212)

[图 9 贪吃蛇降低分辨率示意图 16](#_Toc22367)

[图 10 贪吃蛇示意图 17](#_Toc28241)

[图 11 Game\_Ctrl\_Unit模块逻辑图 20](#_Toc14303)

[图 12 Snake模块逻辑图 21](#_Toc18377)

[图 13 Snake\_Eatting\_Food模块逻辑图 21](#_Toc31449)

[图 14 VGA\_Control模块逻辑图 22](#_Toc14143)

[图 15 Game\_Control\_Unit仿真结果 23](#_Toc28923)

[图 16 Snake仿真结果1 24](#_Toc6765)

[图 17 Snake仿真结果2 25](#_Toc10880)

[图 18 Snake仿真结果3 26](#_Toc27912)

[图 19 Snake\_Eating\_Food仿真结果 26](#_Toc14164)

[图 20 游戏控制 28](#_Toc10749)

[图 21 游戏开始 28](#_Toc12920)

[图 22 吃到食物 29](#_Toc20987)

[图 23 游戏结束 30](#_Toc7337)

# 第1章 绪论

## 1.1贪吃蛇设计背景

我实现的是一个有趣的小游戏——贪吃蛇。这个游戏在出现之初就很受玩家喜爱，网上也涌现了各种各样的版本，通过设置迷宫和改变蛇的速度来增大难度。限于能力与精力所限，我做的这个游戏只是这个游戏的简易版，即只有四面的墙不能触碰，同时游戏虽然简单，画面也比较简单，但可玩性非常好，就蛇的移动速度也没有变化。虽然游戏非常简单，但是这是一个非常经典的游戏，相信玩家会一次又一次地玩。因此我做的这个游戏希望勾起大家童年的回忆，也希望给玩家带来快乐。

## 1.2国内外现况分析

贪吃蛇是一个产生于1970年代中后期的[计算机游戏](https://zh.wikipedia.org/wiki/%E8%AE%A1%E7%AE%97%E6%9C%BA%E6%B8%B8%E6%88%8F" \o "计算机游戏)。此类游戏在1990年代由于一些小屏幕设备引入而再度流行起来，在现在的手机上基本都可安装此小游戏。版本亦有所不同。 此类游戏有较多分类和改版。 著名分支有[Nibble](https://zh.wikipedia.org/w/index.php?title=Nibble&action=edit&redlink=1" \o "Nibble（页面不存在）)、[TRON](https://zh.wikipedia.org/wiki/%E9%9B%BB%E5%AD%90%E4%B8%96%E7%95%8C%E7%88%AD%E9%9C%B8%E6%88%B0" \o "电子世界争霸战)等。这款游戏在FLASH、安卓、IOS等平台都有很完善的版本，而在sword中用verilog实现并不多。

### 1.3 主要内容和难点

在本游戏中，玩家操控一条细长的直线（俗称[蛇](https://zh.wikipedia.org/wiki/%E8%9B%87" \o "蛇)或[虫](https://zh.wikipedia.org/wiki/%E8%99%AB" \o "虫)），它会不停前进，玩家只能操控蛇的头部朝向（上下左右），一路拾起触碰到之物（或称作“豆”），并要避免触碰到自身或者其他障碍物。每次贪吃蛇吃掉一件食物，它的身体便增长一些。当蛇撞到地图的边界或者撞到了自己的身体之后，玩家失败，游戏即结束。当贪吃蛇长度到达16之后，玩家即胜利，游戏结束。当游戏结束后，玩家可以按reset键重新开始，此时得分清零，地图重置。

游戏实现的难点主要是对接VGA，以及游戏中蛇状态转换的一些判断条件。我们需要保证蛇能够正常移动，在碰到墙壁或者碰到自己的身体之后程序能够检测到，并且判定死亡。

# 第2章 贪吃蛇设计原理

## 2.1 贪吃蛇设计相关内容

我的设计用到了ISE14.7软件、SP3平台、VGA屏幕。

### 2.1.1 VGA原理

电子枪发出的电子束经过偏转电场或磁场打到屏幕上的荧光粉使其发出亮光。由于显像管内荧光粉受到电子束击打后发光的时间很短，所以电子束必须不断击打荧光粉使其持续发光。电子枪从屏幕的左上角开始扫描，从左到右扫描完后从下一行左边重新开始，一直到扫描完整个屏幕。由于扫描的速度极快，人眼无法分辨出同一针的先后顺序，达到显示一定图形的目的。

## 2.1.2 按键防抖动

游戏要用到五个BTN键分别控制reset、向左、向右、向上、向下，采用按键防抖动模块，在之前实验七实现的顶层模块中已经得到了实现。

## 2.1.3 七段码显示分数

将得分用七段码分频显示，我们将寄存器当中的值读出来之后放到顶层模块空闲的通道七进行显示。具体的讲解见下文。

## 2.2 贪吃蛇设计方案

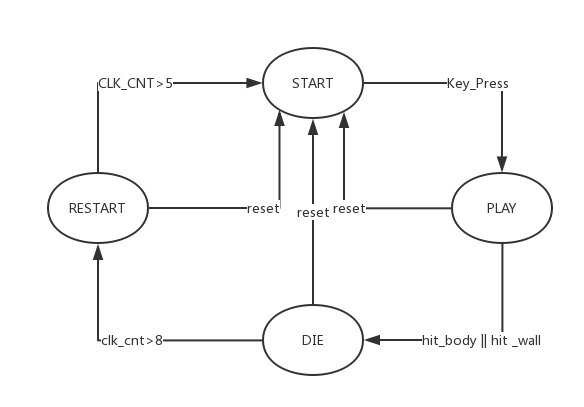
### 2.2.1 游戏的整体结构

游戏主要可以分为以下几个模块：

|  |
| --- |
| input clk,  input rst,  input key1\_press,  input key2\_press,  input key3\_press,  input key4\_press,    output reg[1:0] gameStatus,  input hit\_wall,  input hit\_body,  output reg dieFlash,  output reg restart |

Game\_Ctrl\_Unit——用于控制游戏的状态机，Game\_Ctrl\_Unit的模块接口如下所示：

游戏的状态机如下图所示:

图 1 贪吃蛇游戏状态机

在START状态下，屏幕上显示游戏地图和蛇的开始形状和开始位置，PLAY状态下游戏进行，DIE状态下屏幕闪烁，代表玩家失败。

一开始有任意键按下之后游戏即开始，在撞到墙或身体之后进入DIE状态。通过时钟计数可以实现DIE状态到RESTART状态再到START状态下的转换。在任意状态下按reset即可重新开始游戏。

2）Snake——用于控制蛇的移动及相关状态的变化，Snake模块的接口如下所示：

|  |
| --- |
| input clk,  input rst,    input left\_press,  input right\_press,  input up\_press,  input down\_press,  input dieFlash,  input addLengthh, //增加体长信号  input [1:0]gameStatus, //四种游戏状态  input [9:0] xPos,  input [9:0] yPos, //扫描坐标 单位:"像素点"  //pos的第四位表示一个格子内像素的坐标，高5为表示格坐标    output reg [1:0] snake, //用于表示当前扫描的部件，四种状态：00无 01头  //10身体 11墙  output [5:0] headX,  output [5:0] headY, //头部格坐标    output reg [6:0]bodyNum, //当前蛇的长度    output reg hitBody,  output reg hitWall |

（内容要

Snake模块下同样维护了一个状态机，这个状态机是用于控制贪吃蛇移动的方向，如下所示:

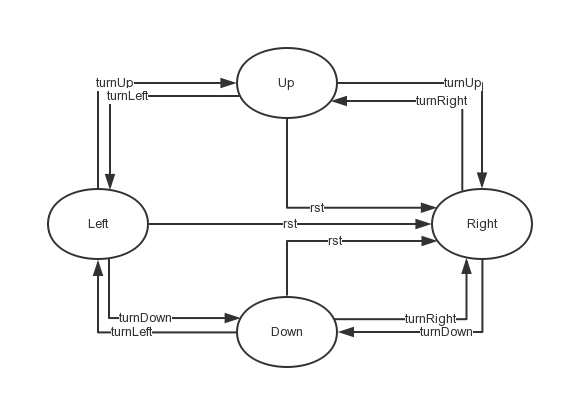


图 2 贪吃蛇移动方向状态机

3）Snake\_Eatting\_Food——用于控制食物位置的生成和蛇的吃食物的过程。其接口如下所示：

|  |
| --- |
| input clk,  input rst,  input [5:0] headX,  input [5:0] headY,  output reg [5:0] foodX,  output reg [5:0] foodY,  output reg addLengthh,  output reg [31:0] score |

input

4）VGA\_Control——根据输入的方块状态，输出VGA颜色，用于VGA驱动。其接口如下所示：

|  |
| --- |
| input [9:0] xPos,  input [9:0] yPos,  input [1:0] snake,  input [5:0] foodX,  input [5:0] foodY,  output reg [11:0] vgaData |

## 2.3 贪吃蛇硬件设计

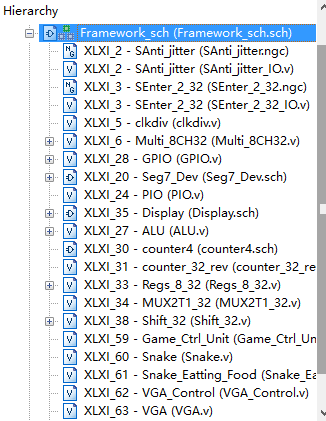


图 3 模块整体结构图

贪吃蛇的顶层模块继承自实验七的顶层模块，project中主要利用到了clkdiv（用于时钟分频）、SAnti\_jitter（按键去抖动模块）、Display（七段码显示模块），并且自己根据PPT的指导添加了VGA驱动模块。以下为主要思路讲解：

### 2.3.1时钟分频模块

时钟分频模块是一个32位时钟计数分频器，可输出2-232分频信号，这种方式延时比较高，但是本实验对于时钟的要求不高，所以这种方法也是可行的。

代码如下所示：

|  |
| --- |
| module clkdiv(input clk,  input rst,  output reg[31:0]clkdiv  );  always @ (posedge clk or posedge rst)begin  if(rst) clkdiv<=0;  else clkdiv<=clkdiv+1'b1;  end  endmodule |

P

R

### 2.3.2按键去抖动模块

按键在按下或者放开的时候，存在机械振动，这种抖动时间大概10~20ms左右。所以我们需要通过延时的方法来避开机械抖动。

在本工程中，按键去抖动辅助模块调用了IP核。其输入输出如下所示：

☉输入：

◆clk=主板时钟

◆RSTN复位输入

◆K\_COL(3:0)输入按钮

◆K\_ROW(4:0) = 00000

◆SW(15:0)输入开关

☉输出：

◆rst：长按RSTN复位输出CR：短按RSTN输出

◆button\_out(3:0)：K\_COL (3:0)去抖动输出

◆button\_pulse(3:0)：K\_COL (3:0)去抖动脉冲输出

◆SW\_OK(15:0)：SW(15:0)去抖动输出

### 2.3.3七段码显示模块

8位七段码显示模块是由8个七段码显示模块Hex2Seg以及P2S模块构成的。Display的结构如下所示：

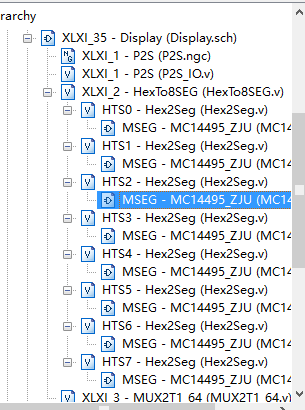


图 4 Display模块结构图

其中七段显示模块Hex2Seg中包括MC14495\_ZJU模块，其利用真值表，绘制相应的卡诺图对表达式进行化简，最后利用化简后的表达式进行原理图的绘制，得到七段码的显示。其原理图如下：

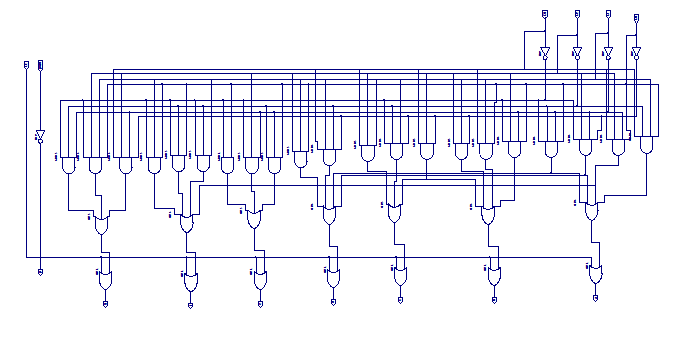


图 5 MC14495\_ZJU模块原理图

Hex2Seg还包括利用LE使能端增加当前七段码的闪烁，其逻辑代码如下所示：

|  |
| --- |
| module Hex2Seg(input [3:0] Hex,  input LE,  input point,  input flash,  output [7:0] Segment  );  wire en = LE & flash;  MC14495\_ZJU MSEG(.D3(Hex[3]),.D2(Hex[2]),.D1(Hex[1]),.D0(Hex[0]),.LE(en),.point(point),  .a(a),.b(b),.c(c),.d(d),.e(e),.f(f),.g(g),.p(p));  assign Segment = {a,b,c,d,e,f,g,p};  endmodule |

P2S的原理是利用双向移位寄存器来进行串行输出适应，在本工程中我们采用了调用IP核的方式来进行实现。

Display模块的原理图如下所示：

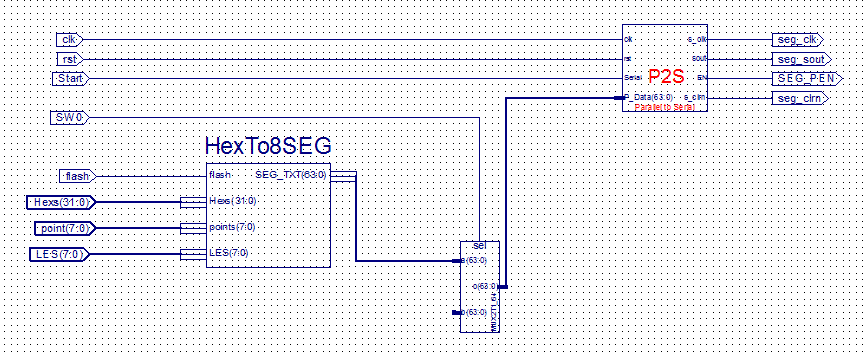


图 6 Display模块原理图

### 2.3.3 VGA驱动模块

Project工程主要新增了VGA驱动的硬件描述代码，所以在实验报告中讲解一下实现的思路：

VGA的hsync和vsync的控制时序如下：

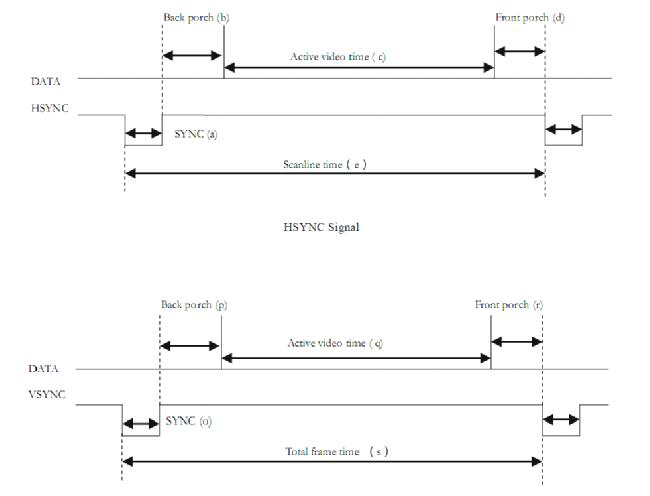


图 7 VGA控制时序

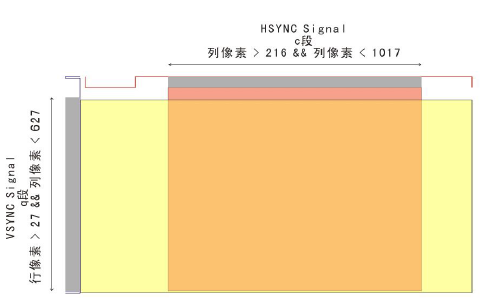


图 8 VGA有效区域

也就是说，VGA只有在HSYNC signal的c段和VSYNC Singal的q段，数据输入才有效。也就是说，显示图片发生在交叉的部分，也就是“有效区域”。

在代码中，在前两个always模块中，我们分别进行了行扫描和列扫描。在第三个always模块中，我们对于之前的扫描计数进行了判断，生成了同步信号。然后在第四个模块当中，利用行列地址、同步信号和输入的显示信号，我们进行输出信号的赋值。

代码如下所示：

|  |
| --- |
| module VGA(  input clk,  input rst,  input [11:0] Din, //bbbb\_gggg\_rrrr\_ pixel    output reg[9:0] row,  output reg[9:0] col,    output reg rdn, //read pixel RAM (active\_low)    output reg [3:0]R,G,B,  output reg HS,VS  );  //h\_count : VGA horizotal counter(0-799) : pixels  reg [9:0] h\_count;  always @ (posedge clk)  begin  if (rst) h\_count <= 10'h0;  else if (h\_count == 10'd799)  h\_count <= 10'h0;  else h\_count <= h\_count+10'h1;  end  //v\_count : VGA vertical counter (0-524) : lines  reg[9:0] v\_count;  always @ (posedge clk or posedge rst)  begin  if (rst) v\_count <= 10'h0;  else if (h\_count == 10'd799) begin  if (v\_count == 10'd524)  v\_count <= 10'h0;  else v\_count <= v\_count+10'h1;  end  end    //singals, will latched for outpus  wire [9:0] row\_addr = v\_count - 10'd35;  wire [9:0] col\_addr = h\_count - 10'd143;  wire h\_sync = (h\_count > 10'd95); //96->799  wire v\_sync = (v\_count > 10'd1); //2->524  wire read = (h\_count > 10'd142) && //143->782  (h\_count < 10'd783) && // 640 pixels  (v\_count > 10'd34) && //35->514  (v\_count < 10'd515); // 480 lines  //vga signals  always @ (posedge clk)  begin  row <= row\_addr; //pixel ram row address  col <= col\_addr; //pixel ram col address  rdn <= ~read; //read pixel (active low)  HS <= h\_sync; //horizontal sychronization  VS <= v\_sync; //vertical sychronization  R <= rdn ? 4'h0: Din[3:0]; //3-bit red  G <= rdn ? 4'h0: Din[7:4]; //3-bit green  B <= rdn ? 4'h0: Din[11:8]; //3-bit blue  end  endmodule |

# 第3章 贪吃蛇设计实现

## 3.1 实现方法

1)Snake模块

首先需要介绍一下像素与格子的概念。由于VGA屏幕的扫描模式是640\*480，如果我们直接采用像素点来表示游戏的各个部件，这无疑是非常复杂并且不好处理的。所以我们采用的是“降低”分辨率的方式，即用16\*16的像素范围为一个”格”，用格作为扫描单位扫描整个屏幕。示意图如下所示：

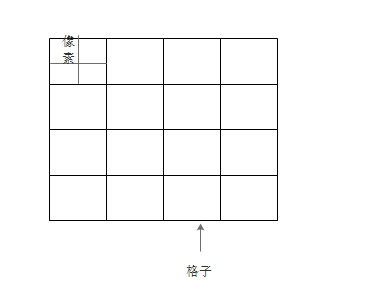


图 9 贪吃蛇降低分辨率示意图

通过上述方案，我们定义pos数组如下，pos的低4位用于表示一个格子内像素的坐标，高5位表示格坐标。

屏幕上的格子有四种状态，分别是NONE,WALL,BODY和HEAD。分别代表可运动区域、墙、身体和头。其中墙是地图的边界：

if (xPos[9:4]==0 || xPos[9:4]==MAXX-1 ||

yPos[9:4]==0 || yPos[9:4]==MAXY-1)

begin

snake=WALL;

对于蛇，我们通过三组变量进行描述，分别是head,cube[15],valid[15],标记蛇的头的位置,蛇的身体的每一点位置，蛇身体的每一位是否存在。示意图如下所示：

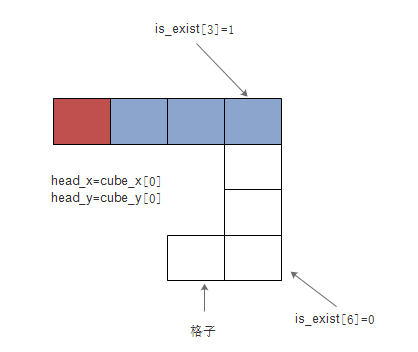


图 10 贪吃蛇示意图

我们利用时钟分频Div(24)作为时钟输入信号，大概使贪吃蛇每秒移动3次。身体移动的算法非常直观，即本长度位移动的下个格坐标为上一个长度的当前格坐标，相关代码如下所示：

|  |
| --- |
| bodyX[1]<=bodyX[0];  bodyY[1]<=bodyY[0];    bodyX[2]<=bodyX[1];  bodyY[2]<=bodyY[1];    ………… |

* 1. **实现过**

碰到身体判断为头的坐标=身体任意一位的坐标。

|  |
| --- |
| if ((bodyY[0]==bodyY[1] && bodyX[0]==bodyX[1]&& valid[1]==1) ||  (bodyY[0]==bodyY[2] && bodyX[0]==bodyX[2]&& valid[2]==1) ||  (bodyY[0]==bodyY[3] && bodyX[0]==bodyX[3]&& valid[3]==1) ||  (bodyY[0]==bodyY[4] && bodyX[0]==bodyX[4]&& valid[4]==1) ||  ………… |

根据头部位置信息和按键信息决定head的移动方向，即维护一个当前运动方向，根据当前运动方向和按键信息确定出下一个时钟周期内的运动方向，在下一个时钟上升沿来到的时候，刷新当前运动方向：

|  |
| --- |
| case (direct)  UP:  begin  if (turnLeft)  direct\_next=LEFT;  else if (turnRight)  direct\_next=RIGHT;  else  direct\_next=UP;  end  ………… |
| always @ (posedge clk or posedge rst)  if (rst)  direct\_r <= RIGHT;  else  direct\_r <= direct\_next; |

根据当前运动方向和当前的位置判断是否撞墙，代码如下所示：

|  |
| --- |
| if ( (direct == UP && bodyY[0] == 1)  || (direct == DOWN && bodyY[0] == MAXY-2)  || (direct == LEFT && bodyX[0] == 1)  || (direct == RIGHT && bodyX[0] == MAXX-2))  hit\_wall<=1; |

详细的实现 同时，在head坐标和apple坐标重合的时候，收到了addLength信号，此时 我们需要做的是使得相应位valid[bodyNum]<=1；并且发出addLengthState<=1信号 用于后续的判断。

|  |
| --- |
| case (addLengthState)  0:begin  if (addLength)  begin  bodyNum<=bodyNum+1;  valid[bodyNum]<=1;  addLengthState<=1; //”吃下“信号  end  end  1:begin  if (!addLength)  addLengthState<=0;  end  endcase |

最后，在Snake模块中，我们可以对地图进行一遍扫描，输出一个信号代表当前扫描到的位置是那种类型的点。在VGA\_Control模块中我们利用这个信号在屏幕上显示不同的颜色就可以达到我们所要的效果。其中dieFlash是在死亡状态下发出的方波信号，利用这个信号，可以使得屏幕出现闪烁的效果。

|  |
| --- |
| always@(xPos or yPos)  begin  if (xPos>=0 && xPos<SCRX && yPos>=0 && yPos<SCRY)  begin  if (xPos[9:4]==0 || xPos[9:4]==MAXX-1 ||  yPos[9:4]==0 || yPos[9:4]==MAXY-1)  begin  snake=WALL;  end else if (xPos[9:4]==bodyX[0] &&  yPos[9:4]==bodyY[0] && valid[0] == 1)  begin  snake=(dieFlash==1)?HEAD:NONE; //扫描头  end else  begin  flag=1;  for (i=1;i<16;i=i+1)  begin  if (xPos[9:4]==bodyX[i] && yPos[9:4]==bodyY[i]  && valid[i]==1)  begin  flag=0;  end  end  if (flag==1)  snake=(dieFlash==1)?BODY:NONE;  else  snake=NONE;  end  end  end |

2）Snake\_Eatting\_Apple模块

首先我利用了一个简单的方法产生随机数，利用Div（1）分频信号作为时钟，在每一次时钟上升沿的时候，对于随机数加927。

如果当前食物位置等于蛇头位置，那么代表蛇吃到了食物，那么计分+1，重新放置食物。这个时候利用上述随机数，我们再随机生成x和y，作为下一个食物的位置。

## 3.2 实现过程

### 3.2.1 Game\_Ctrl\_Unit模块

输入信号为外部输入clk（时钟信号）, rst（重新开始游戏信号）, key1\_press, key2\_press, key3\_press, key4\_press,（分别代表游戏的四个按键）。在按下四个按键的其中任意一个的时候可以从游戏的Start状态转换到Play状态。

输入信号同时包括hit\_wall、hit\_body， 这两种信号可以使得游戏状态从PLAY状态转换到DIE状态，代表贪吃蛇的死亡。

输出信号有2位的gameStatus（代表游戏的状态机，其他模块要用到）以及dieFlash用于控制蛇死亡时候的屏幕闪烁。

其逻辑图如下所示：

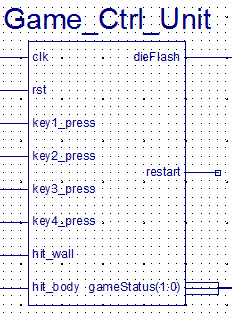


图 11 Game\_Ctrl\_Unit模块逻辑图

**该模块需要进行仿真，测试游戏状态机的转换情况。**

### 3.2.2 Snake模块

Snake模块的输入信号包括外部输入clk, rst, left\_press, right\_press, up\_press, down\_press，分别代表时钟信号，重新开始游戏信号和四个方向的移动信号（通过button经过去抖动模块之后输入）。其他输入信号包括dieFlash, addLength（蛇吃到食物之后得到的增加体长信号）, gameStatus（游戏状态机信号）， xPos，yPos（VGA当前扫描到的像素点，需要在本模块判断这个像素点属于哪种类型，如蛇头，蛇深或者墙壁）。

输出信号包括蛇的头部坐标（需要送到其他模块，用于判断蛇当前有没有吃到食物）和hit\_wall、hit\_body两个死亡信号。

其逻辑图如下所示：

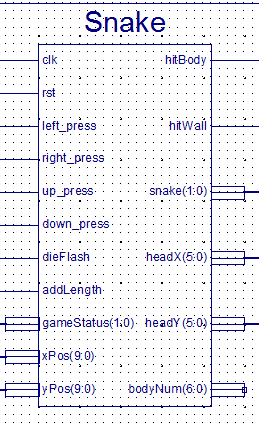


图 12 Snake模块逻辑图

**该模块需要进行仿真，用来判断蛇运动的方向是否达到预期，以及蛇是否能够按照预期实现撞墙和撞到身体死亡。**

### 3.2.3 Snake\_Eatting\_Food模块

该模块的外部输入信号包括clk, rst。其他输入信号有 headX, headY（从Snake模块获得，代表头部位置）。

输出信号有foodX, foodY（代表食物位置，用于VGA显示）、addLength（之前提到的增加体长信号，送入Snake模块）、score（得分情况，送到七段码显示的通道七）

其逻辑图如下所示：



图 13 Snake\_Eatting\_Food模块逻辑图

**该模块需要进行仿真，用于判断蛇能否按照预期实现吃到食物之后发出addLength信号。**

### 3.2.4 VGA\_Control模块

输入信号包括当前VGA扫描到的行列位置xPos、yPos和由Snake模块产生的snake 信号（部分判断xPos、yPos的类型）。这个模块最终得出当前像素点的颜色信息，用12位RGB输出。

其逻辑图如下所示：

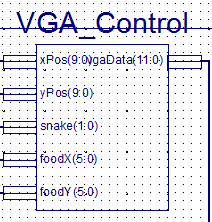


图 14 VGA\_Control模块逻辑图

## 3.2.5 顶层模块设计

顶层模块根据之前所设计的接口将各个模块连接在一起，同时接入系统所需要的IO接口，顶层模块原理图如下所示：

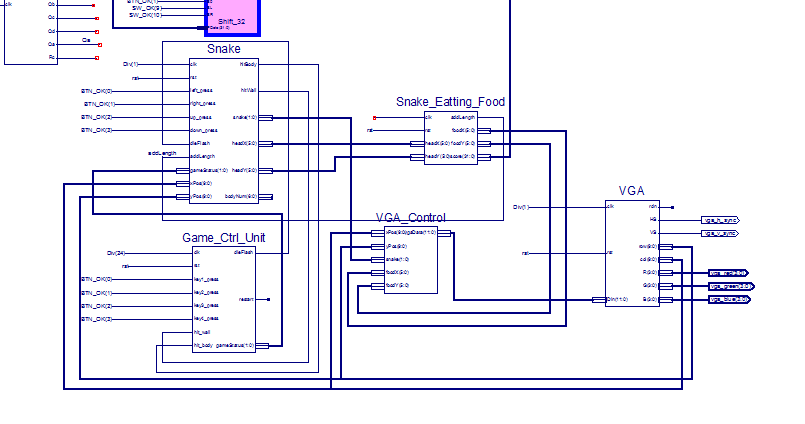


图 15 顶层模块原理图

## 3.3仿真与调试

### 实现中出现的问题

调试过程中遇到了许多问题，许多都是比较低级的比如变量打错之类的问题。在这里提一个看上去不是那么低级的问题。

由于我对于贪吃蛇的食物生成利用到了基于时钟信号的随机数，所以在Snake模块和Snake\_Eatting\_Food模块中我采用了两个不同的时钟计数器，这将会导致当Snake\_Eatting\_Food模块发出AddLength信号的时候并不一定是蛇刚好吃到食物的时候，正确的情况应该是蛇第一次收到为1的AddLength信号的时候。我一开始并没有注意到这个问题。

最后的解决方法是在Snake模块中引入一个新的变量addLengthState变量，当蛇接收到为1的AddLength信号的时候，需要判断addLengthState是否为0，只有为0的时候才代表蛇吃到了食物。

### 最终的仿真结果

2.3中介绍的硬件模块由于之前的实验中已经进行了仿真测试，并且已经经过多次的物理测试，所以不再进行仿真测试与分析。

最终的仿真结果如下：

### 3.3.1 Game\_Control\_Unit模块

该模块中主要实现的是游戏状态机的实现，相对来说比较简单，我利用仿真的方法模拟了按键的输入以及游戏结束信号的输入。仿真代码如下所示：

|  |
| --- |
| fork  forever #20 clk <= ~clk;  begin  #20;  rst=0;  key1\_press=1;  #30;  key1\_press=0;  #40;    hit\_body=1;  #30;  hit\_body=0;  #700;    key2\_press=1;  #30;  key2\_press=0;  #40;    hit\_wall=1;  #30;  hit\_wall=0;  end  join |

仿真结果图如下所示:

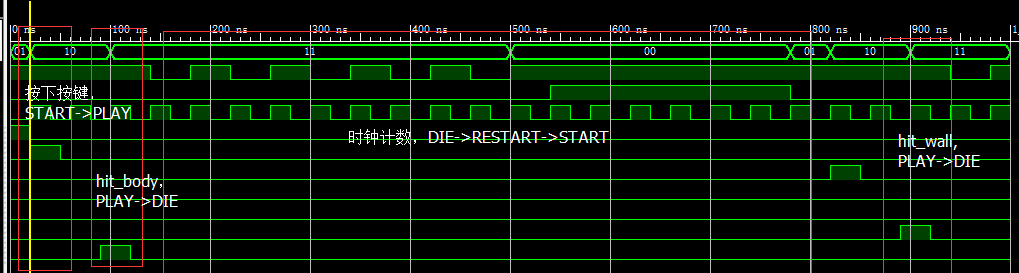


图 16 Game\_Control\_Unit仿真结果

### 3.3.2 Snake模块

Snake模块是工程中最核心的模块，我们首先需要模拟的是蛇头的移动（根据输入的按键信号改变方向），仿真代码如下所示：

|  |
| --- |
| fork  forever #20 clk <= ~clk;  begin  #20;  rst=0;    right\_press=1;  #30;  right\_press=0;  #30;    up\_press=1;  #30;  up\_press=0;  #30;  end  join |

仿真结果图如下所示，可以看到，根据输入信号，贪吃蛇的移动方向发生了改变。同时，当贪吃蛇到达墙的边界的时候，根据程序判断，产生了hitWall的输出信号。

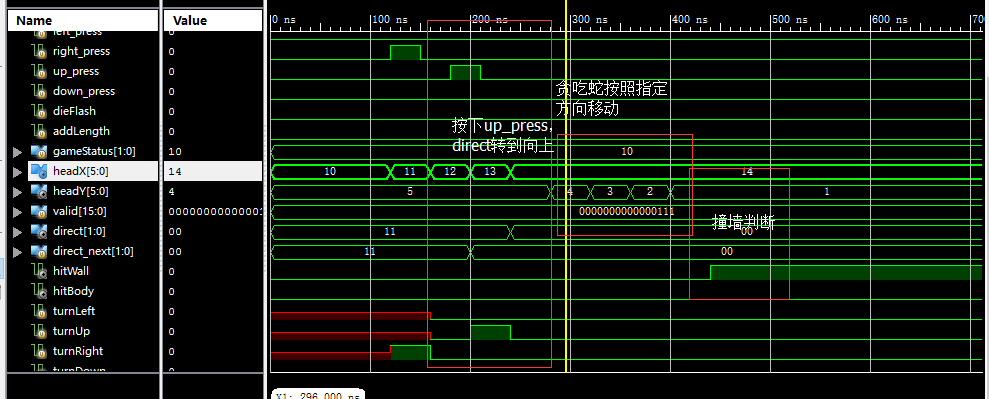


图 17 Snake仿真结果1

接下来需要模拟的是当我们从Snake\_Eatting\_Food模块中获取到一个addLength信号的时候，程序是否能够正确对贪吃蛇进行修改。从以下的仿真结果来看，程序可以正确实现这一过程，bodyNum+1，同时valid从111变为1111.

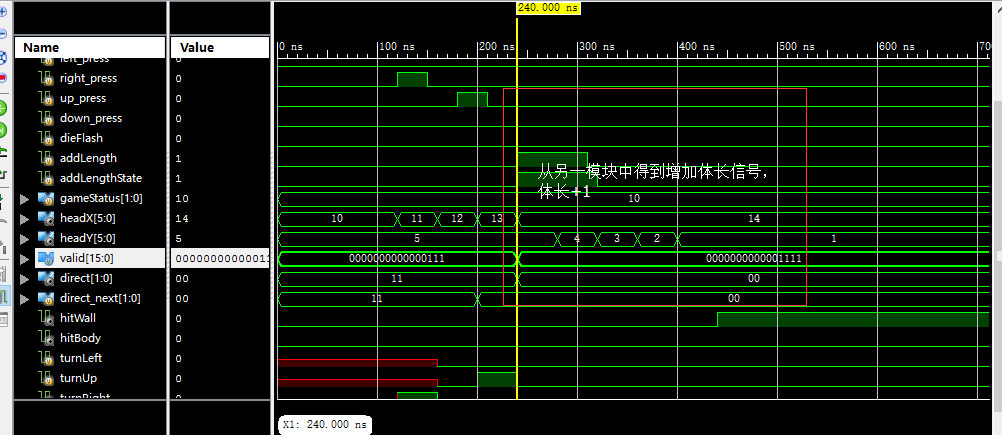


图 18 Snake仿真结果2

最后需要模拟一下贪吃蛇撞到自己身体的情况，这在贪吃蛇体长比较短的时候是很难发生的，所以我们需要首先增加贪吃蛇的体长，然后构造出贪吃蛇运行成环的情况（贪吃蛇依次以右下左上的方向运动）。仿真代码如下所示:

|  |
| --- |
| fork  forever #20 clk <= ~clk;  begin  #20;  rst=0;    right\_press=1;  #30;  right\_press=0;  #40;    for (i=0;i<6;i=i+1)  begin  addLength=1;  #40;  addLength=0;  #40;  end    down\_press=1;  #40;  down\_press=0;  #40;    left\_press=1;  #40;  left\_press=0;  #40;    up\_press=1;  #40;  up\_press=0;  #40;  end  join  end |

仿真结果如下所示，根据仿真结果，我们可以看到当头和身体的任意一点坐标相等的时候，hitBody信号就会输出1.

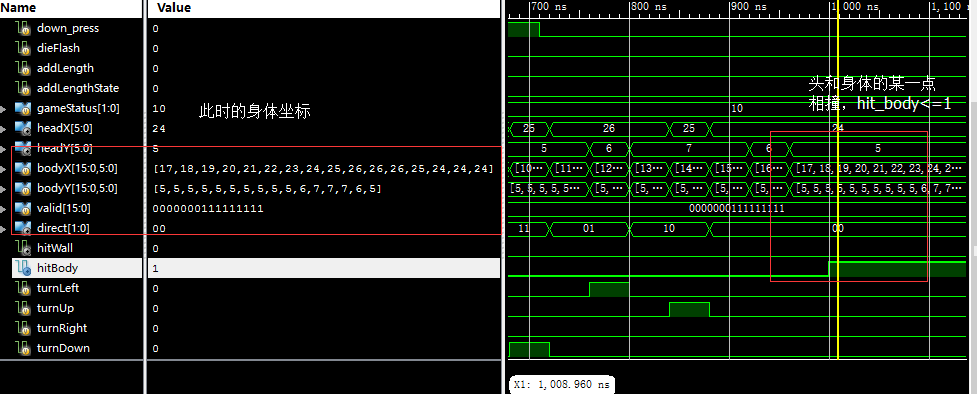


图 19 Snake仿真结果3

### 3.3.3 Snake\_Eatting\_Food模块

这个模块相对来说也是比较简单，需要仿真的内容就是当蛇头的位置和食物的位置相等的时候，addLength信号被置为1。同时更新食物的位置。根据以下仿真结果，我们可以发现工程正确实现所需的功能。

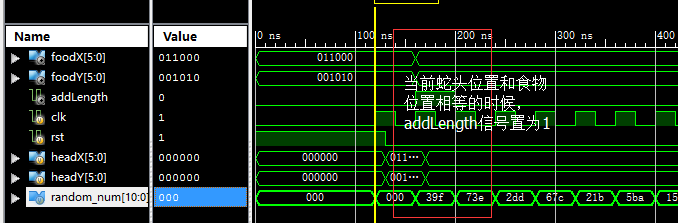


图 20 Snake\_Eating\_Food仿真结果

# 第4章 系统测试验证与结果分析

## 4.1 功能测试

上板接入VGA之后，可以显示初始游戏界面。当按下任意键之后，游戏开始，蛇开始移动。当蛇头和食物的位置相同的时候，蛇的长度加1，生成新的食物，同时七段码显示的得分也加1。可以通过BTN[0]-BTN[3]控制蛇的移动方向。按下reset键之后游戏可以重新开始。功能可以正常实现。

## 4.2 技术参数测试

控制蛇撞墙，蛇会死亡。控制蛇撞到自己的身体，蛇也会死亡。

## 4.3 结果分析

最终游戏可以正常进行，游戏的结束判断正常，操作正常，可玩性良好。

## 4.4 系统演示与操作说明

上板接入VGA之后，可以显示初始游戏界面。当按下任意键之后，游戏开始，蛇开始移动。当蛇头和食物的位置相同的时候，蛇的长度加1，生成新的食物，同时七段码显示的得分也加1。可以通过BTN[0]-BTN[3]控制蛇的移动方向。按下reset键之后游戏可以重新开始。

系统使用说明：

BTN[0]——向左

BTN[1]——向右

BTN[2]——向上

BTN[3]——向下

Reset——重新开始游戏



图 21 游戏控制

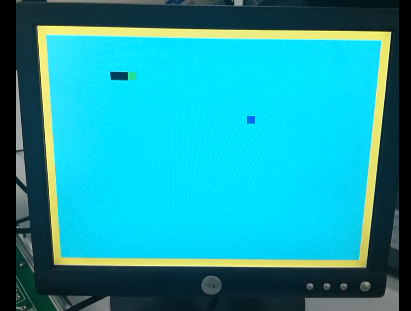
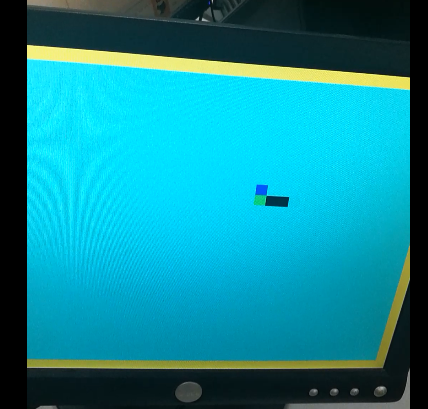


图 22 游戏开始



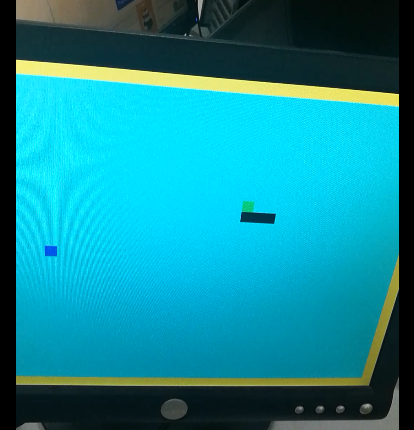
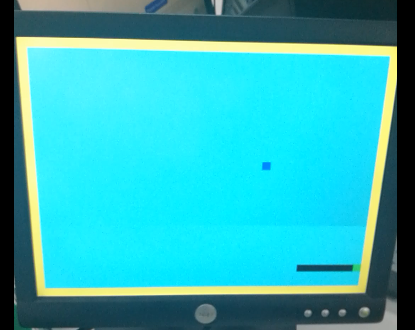


图 23 吃到食物



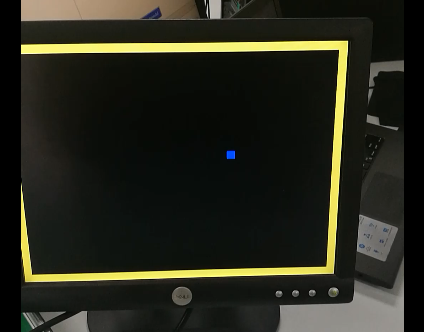


图 24 游戏结束

# 第5章 结论与展望

一开始听到老师让我们单人做一个project的时候，我是崩溃的，而且老师介绍了往年的一些优秀作品，心想自己肯定完成不了这样的游戏。之后通过一次次实验的积累，以及自己自学的一些知识，并参考了一些程序，我着手开始写自己的project。第一次见到vga上显示出了图形的时候我的内心的是很兴奋的，在此之后我继续进行设计，并且排除了一系列的问题，最终完成了一个比较基础的贪吃蛇游戏。

这次project让我收获了很多东西，他让我对Verilog语言更加熟悉，了解了许多Verilog的特性。比如关于寄存器和存储器的利用，在开始阅读project要求的时候，我不是非常理解二者的意义，我一度认为需要像之前实验那样，从最底层开始用门级语言设计。后来通过阅读Verilog教程，我了解到Verilog语言对寄存器和存储器进行了建模。对于寄存器，verilog提供了5种寄存器类型供程序员使用。而存储器就是一个寄存器数组。这样的话，我们就不在需要从底层设计寄存器和存储器。

当然，不得不承认，我这个project是一个相当简陋的贪吃蛇游戏，从功能上，他只实现了最基本的功能，从界面交互上，他也不是非常好看。当然，作为一个外专业的大三学生，本学期在同时修读本专业课程、计科的大三专业课，时间上确实只能够支持我完成一个最简单的游戏。如果未来有机会的话，我希望能够在地图上加上障碍、增加难度设置（贪吃蛇移动变速）和游戏的音响设置，把这个游戏做到和使用高级语言写出来的游戏差不多。当然，希望我这个简陋的游戏大家能够喜欢。