基于FPGA的贪吃蛇游戏实现

李平治 @USTC

2020.12.25

游戏简介

贪吃蛇（Snake）是一个起源于1976年的街机游戏 Blockade。此类游戏在1990年代由于一些具有小型屏幕的移动电话的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。

在游戏中，玩家操控一条细长的直线（俗称蛇或虫），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。吃掉一些食物后会使蛇的移动速度逐渐加快，让游戏的难度渐渐变大。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。

实验环境

- Vivado 集成设计环境
- Nexys 4 FPGA开发板
- 分辨率为 1920x1080 的显示器

实现思路

游戏共有7 个模块，分别是顶层模块、时钟模块、蛇控制模块、苹果控制模块、游戏状态模块、墙生成模块、VGA显示模块。

- 顶层模块负责综合其他所有模块，并将它们与 FPGA 开发板接口相连。
- 时钟模块负责将板载的 100MHZ 时钟分频产生 25MHZ 和 2HZ 的时钟。25MHZ 时钟用于 VGA 显示，2HZ 时钟用于蛇的位置更新。
- 蛇控制模块负责控制蛇头与蛇身的位置，以及它们的显示。蛇头是主要的控制对象，蛇身随之移动。每过一个周期，蛇头朝着操控的方向移动一格，同时将蛇身体上的前一个位置赋给后一个方格点。对于显示部分，方法是将 VGA 当前扫描位置与蛇的坐标对比，若相同则输出显示信号。
- 苹果控制模块负责控制苹果的位置与显示。模块中维护了一个高频更新的随机数，当输入重置信号或蛇吃到苹果时，将当前随机数赋值给苹果坐标。显示苹果的方法与蛇相同，将苹果坐标与 VGA 当前扫描坐标相对比，相同则输出显示信号。
- 游戏状态模块负责控制游戏状态，pulse 或 dead 为 1 时暂停所有变量的更新。当蛇头碰到身体或者墙壁时，dead 置 1，需要通过板载按钮置 0 从而重新开始游戏； pulse 可以通过板载开关控制。
- 墙生成模块负责对比墙坐标与 VGA 当前扫描位置，若相同则输出墙生成信号。
- VGA 显示模块通过逐行扫描，实现在显示器上的图像显示。

工作量

> Verilog HDL源代码

```
module topModule(
    input CLK100MHZ,
    input [4:0]BTN,
    input pulse,
    output wire [4:0]LED,
    output [3:0]VGA_R,
    output [3:0]VGA_G,
    output [3:0]VGA_B,
    output VGA_HS,
```

```
    output VGA_VS
);
wire btnc,btnl,btnr,btnu,btnd;//center,left,right,down
wire CLK25MHZ;//25MHZ clock, for VGA
wire CLK2HZ clock, for GAME CONTROL
wire dead;
wire [8:0] xPos,yPos;
wire [8:0] apple_xPos,apple_yPos;
wire apple,snake,border,head;
wire [8:0] head_xPos, head_yPos;//snake's head position
wire intoBorder,intoBody;
wire appleRefresh;
assign LED[1]=intoBorder;
assign LED[2]=intoBody;
assign LED[3]=dead;
assign LED[4]=appleRefresh;
assign btnc=BTN[4];
assign btnl=BTN[1];
assign btnu=BTN[0];
assign btnd=BTN[3];
assign btnr=BTN[2];
clockInitialize clockInitialize(
    .CLK100MHZ(CLK100MHZ),
    .CLK25MHZ(CLK25MHZ),
    .CLK2HZ(CLK2HZ)
);
snakeMove snakeMove(
    .btnc(btnc),
    .clockDirect(CLK25MHZ),
    .clockMove(CLK2HZ),
    .btnr(btnr),
    .btnl(btnl),
    .btnu(btnu),
    .btnd(btnd),
    .LED(LED[0]),
    .intoBorder(intoBorder),
    .intoBody(intoBody),
    .xPos(xPos),
    .yPos(yPos),
```

```
    .apple_xPos(apple_xPos),
    .apple_yPos(apple_yPos),
    .dead(dead),
    .snake(snake),
    .head(head),
    .pulse(pulse),
    .appleRefresh(appleRefresh),
    .head_xPos(head_xPos),
    .head_yPos(head_yPos)
);
gameControl gameControl(
    .intoBody(intoBody),
    .intoBorder(intoBorder),
    .pulse(pulse),
    .btnc(btnc),
    .CLK25MHZ(CLK25MHZ),
    .dead(dead)
);
appleControl appleControl(
    .clock(CLK25MHZ),
    .xPos(xPos),
    .yPos(yPos),
    .btnc(btnc),
    .dead(dead),
    .appleRefresh(appleRefresh),
    .apple_xPos(apple_xPos),
    .apple_yPos(apple_yPos),
    .apple(apple)
);
wallControl wallControl(
    .xPos(xPos),
    .yPos(yPos),
    .border(border)
);
VGA VGA(
    .R_clk_25M(CLK25MHZ),
    .snake(snake),
    .apple(apple),
    .border(border),
```

```
    .xpos(xPos),
    .ypos(yPos),
    .O_red(VGA_R),
    .O_green(VGA_G),
    .O_blue(VGA_B),
    .O_hs(VGA_HS),
    .O_vs(VGA_VS),
    .head(head)
);
endmodule


module clockInitialize(
    input CLK100MHZ,
    output reg LED,
    output reg CLK25MHZ,
    output reg CLK2HZ
);
reg [24:0]count2HZ;
reg count25MHZ;
always@(posedge CLK100MHZ)//0.5HZ
begin
    if (count2HZ == 25'b10_1111_1010_1111_0000_1000_00) begin
        CLK2HZ <= ~CLK2HZ;
        count2HZ <= 25'b0;
    end
    else begin
        count2HZ <= count2HZ + 1'b1;
    end
end
always @(posedge CLK100MHZ)//25MHZ
begin
    count25MHZ <= count25MHZ + 1'b1;
    if(count25MHZ == 1'b0)
        CLK25MHZ <= ~CLK25MHZ;
end
endmodule

module snakeMove(
```

```
    input clockDirect,//CLK25MHZ
    input clockMove,//CLKtwoHZ
    input btnr,
    input btnl,
    input btnu,
    input btnd,
    input btnc,
    input [8:0]xPos,
    input [8:0]yPos,
    input [8:0]apple_xPos,
    input [8:0]apple_yPos,
    input dead,
    input pulse,
    output reg LED,
    output reg intoBorder,intoBody,
    output wire snake,
    output head,
    output reg appleRefresh,
    output reg[8:0] head_xPos,
    output reg[8:0] head_yPos
);
reg [1:0]direction;
parameter up=2'b00,down=2'b01,left=2'b10,right=2'b11;
reg [8:0] body_xPos [0:99];//snake's body position X, MAX_LENGTH=100
reg [8:0] body_yPos [0:99];//snake's body position Y, MAX_LENGTH=100
reg [7:0] length;
  always@(posedge clockDirect or posedge dead)////update direction
begin
    if(dead)begin
        direction <= right;
    end
    else if (btnc) begin
        direction <= right;
    end
    else if(btnr)begin
        if(direction != left)
        direction <= right;
        else begin
            direction <= left;
```

```
        end
    end
    else if(btnl)begin
        if(direction != right)
        direction <= left;
        else begin
            direction <= right;
        end
    end
    else if(btnu)begin
        if(direction != down)
        direction <= up;
        else begin
            direction <= down;
        end
    end
    else if(btnd)begin
        if(direction != up)
        direction <= down;
        else begin
            direction <= up;
        end
    end
    else begin
        direction <= direction;
     end
end
always@(posedge clockDirect or posedge dead)///////update the length
begin
    if (pulse) begin
        length <= length;
        appleRefresh <= 0;
    end
    else if(btnc)begin
        appleRefresh <= 0;
        length <= 8'd1;
    end
    else if(dead) begin
        length <= 8'd1;
```

```
        appleRefresh <= 0;
    end
    else if(head_xPos==apple_xPos && head_yPos==apple_yPos)
    begin
        if(length < 8'd100)
        begin
            appleRefresh <= 1;
            length <= length+ 1  ;
        end
        else begin
            length <= length;
            appleRefresh <= 0;
        end
    end
    else begin
        appleRefresh <= 0;
    end
end
assign snake = (body_xPos[0] == xPos && body_yPos[0] == yPos && length > 8'd1) ||
(body_xPos[1] == xPos && body_yPos[1] == yPos && length > 8'd2) ||(body_xPos[2] ==
xPos && body_yPos[2] == yPos && length > 8'd3) ||(body_xPos[3] == xPos && body_yPos[3]
== yPos && length > 8'd4) ||......||(body_xPos[99] == xPos && body_yPos[99] == yPos &&
length > 8'd100);
assign head = (head_xPos == xPos && head_yPos == yPos);
always@(posedge clockMove or posedge dead)//update snake's postion
begin
    if(btnc | dead)begin
    head_xPos <= 9'd20;
    head_yPos <= 9'd20;
    body_xPos[99] <= 9'b0;
    body_yPos[99] <= 9'b0;
.........
    body_xPos[1] <= 9'b0;
    body_yPos[1] <= 9'b0;
    body_xPos[0] <= 9'b0;
    body_yPos[0] <= 9'b0;
    end
    else if(pulse == 0)begin
        body_xPos[99][8:0] <= body_xPos[98][8:0];
```

```
        body_yPos[99][8:0] <= body_yPos[98][8:0];
.........
        body_xPos[1][8:0] <= body_xPos[0][8:0];
        body_yPos[1][8:0] <= body_yPos[0][8:0];
        body_xPos[0][8:0] <= head_xPos[8:0];
        body_yPos[0][8:0] <= head_yPos[8:0];
        case (direction)
            up: head_yPos <= head_yPos - 9'd1;
            down: head_yPos <= head_yPos + 9'd1;
            right: head_xPos <= head_xPos + 9'd1;
            left: head_xPos <= head_xPos - 9'd1;
        endcase
    end
    else begin end
end
  always@(posedge clockDirect) //check if run into the BORDER or its BODY
begin
    if(btnc)begin
        intoBorder <= 0;
    end
    else if((head_xPos<9'd1) || (head_yPos<9'd1) || (head_xPos>9'd50) ||
(head_yPos>9'd50))begin
        intoBorder <= 1;
    end
    else begin
        intoBorder <= 0;
    end

    if (btnc) begin
        intoBody <= 0;
    end
    else if((head_xPos[8:0]==body_xPos[3][8:0] && head_yPos[8:0]==body_yPos[3][8:0] &&
length>4)||(head_xPos[8:0]==body_xPos[4][8:0] && head_yPos[8:0]==body_yPos[4][8:0] &&
length>5)||(head_xPos[8:0]==body_xPos[5][8:0] && head_yPos[8:0]==body_yPos[5][8:0] &&
length>6)||......||(head_xPos[8:0]==body_xPos[99][8:0] &&
head_yPos[8:0]==body_yPos[99][8:0] && length>100))
    begin
        intoBody <= 1;
    end
```

```
    else begin
        intoBody <= 0;
    end
end
endmodule

module gameControl(
    input intoBody,
    input intoBorder,
    input pulse,
    input CLK25MHZ,
    input btnc,
    output reg dead
);
always@(posedge CLK25MHZ)
begin
    if((pulse == 0) && (dead == 0))begin
        if(intoBody)
        begin
            dead <= 1;
        end
        else if (intoBorder) begin
            dead <= 1;
        end
    end
    else if(btnc) begin
      dead <= 0;
    end
end
endmodule


module appleControl(
    input clock,//for random number initializing, 25MHZ
    input [8:0]xPos,
    input [8:0]yPos,
    input dead,
    input appleRefresh,
    input btnc,
```

```
    output reg [8:0] apple_xPos,
    output reg [8:0] apple_yPos,
    output wire apple
);
reg [18:0]rand_num;
always@(posedge clock or posedge btnc)
begin
    if(btnc)
        rand_num <= 19'b000_0110_1100_0000_1101;   //seed
    else
        begin
            rand_num[0] <= 1;
            rand_num[1] <= rand_num[0];
            rand_num[2] <= rand_num[1];
            rand_num[3] <= rand_num[2];
            rand_num[4] <= rand_num[3] ^~rand_num[14];
            rand_num[5] <= 0;
            rand_num[6] <= 0;
            rand_num[7] <= 0;
      rand_num[8] <= 0;
      rand_num[9] <= 0;
      rand_num[10] <= 1;
      rand_num[11] <= rand_num[10];
      rand_num[12] <= rand_num[11]^~rand_num[14];
      rand_num[13] <= rand_num[12]^~rand_num[14];
      rand_num[14] <= rand_num[13]^~rand_num[12];
      rand_num[15] <= 0;
            rand_num[16] <= 0;
      rand_num[17] <= 0;
      rand_num[18] <= 0;
        end
end
  always@(posedge appleRefresh or posedge btnc)//update APPLE's position
begin
    if(appleRefresh)begin
        apple_yPos[8:0] <= rand_num[8:0];
        apple_xPos[8:0] <= rand_num[18:10];
    end
    else if (btnc) begin
```

```
        apple_yPos[8:0] <= rand_num[8:0];
        apple_xPos[8:0] <= rand_num[18:10];
    end
    else begin
        apple_xPos[8:0] <= apple_xPos[8:0];
        apple_yPos[8:0] <= apple_yPos[8:0];
    end
end
assign apple = (apple_xPos[8:0]==xPos[8:0]) && (apple_yPos[8:0]==yPos[8:0]) &&
(dead==0);
endmodule

module wallControl(
    input [8:0]xPos,
    input [8:0]yPos,
    output wire border
);
assign border = (xPos==9'd0 && yPos<9'd52) || (xPos==9'd51 && yPos<9'd52) ||
(yPos==9'd0 && xPos<9'd52) || (yPos==9'd51 && xPos<9'd52);
endmodule

module VGA(
    input                   R_clk_25M    , // clock with 25MHz
    input                   snake,
    input                   apple,
    input                   border,
    input                   head,
    output   wire  [8:0]    xpos,//current scanning position X
    output   wire  [8:0]    ypos,//current scanning position Y
    output   reg   [3:0]    O_red    , // VGA red
    output   reg   [3:0]    O_green  , // VGA green
    output   reg   [3:0]    O_blue   , // VGA blue
    output                  O_hs     , // VGA行同步信号
    output                  O_vs       // VGA场同步信号
);
parameter I_rst_n = 0;
// 分辨率为640*480时行时序各个参数定义
parameter       C_H_SYNC_PULSE      =   96  ,
                C_H_BACK_PORCH      =   48  ,
```

```
                C_H_ACTIVE_TIME     =   640 ,
                C_H_FRONT_PORCH     =   16  ,
                C_H_LINE_PERIOD     =   800 ;
// 分辨率为640*480时场时序各个参数定义
parameter       C_V_SYNC_PULSE      =   2   ,
                C_V_BACK_PORCH      =   33  ,
                C_V_ACTIVE_TIME     =   480 ,
                C_V_FRONT_PORCH     =   10  ,
                C_V_FRAME_PERIOD    =   525 ;

parameter       C_COLOR_BAR_WIDTH   =   C_H_ACTIVE_TIME / 8  ;
reg [11:0]      R_h_cnt         ; // 行时序计数器
reg [11:0]      R_v_cnt         ; // 列时序计数器
wire [11:0]     R_h_cnt_pos     ;
wire [11:0]     R_v_cnt_pos     ;
wire            W_active_flag   ; // 激活标志，当这个信号为1时RGB的数据可以显示在屏幕上
//////////////////////////////////////////////////////////////////
// 功能：产生行时序
//////////////////////////////////////////////////////////////////
always @(posedge R_clk_25M )
begin
    if(I_rst_n)
        R_h_cnt <=  12'd0   ;
    else if(R_h_cnt == C_H_LINE_PERIOD - 1'b1)
        R_h_cnt <=  12'd0   ;
    else
        R_h_cnt <=  R_h_cnt + 1'b1  ;
end
assign O_hs =   (R_h_cnt < C_H_SYNC_PULSE) ? 1'b0 : 1'b1    ;
//////////////////////////////////////////////////////////////////
//////////////////////////////////////////////////////////////////
// 功能：产生场时序
//////////////////////////////////////////////////////////////////
always @(posedge R_clk_25M )
begin
    if(I_rst_n)
        R_v_cnt <=  12'd0   ;
    else if(R_v_cnt == C_V_FRAME_PERIOD - 1'b1)
        R_v_cnt <=  12'd0   ;
```

```
    else if(R_h_cnt == C_H_LINE_PERIOD - 1'b1)
        R_v_cnt <=  R_v_cnt + 1'b1  ;
    else
        R_v_cnt <=  R_v_cnt ;
end
assign O_vs =   (R_v_cnt < C_V_SYNC_PULSE) ? 1'b0 : 1'b1    ;
assign W_active_flag =  (R_h_cnt >= (C_H_SYNC_PULSE + C_H_BACK_PORCH
 ))  &&
                        (R_h_cnt <= (C_H_SYNC_PULSE + C_H_BACK_PORCH +
C_H_ACTIVE_TIME))  &&
                        (R_v_cnt >= (C_V_SYNC_PULSE + C_V_BACK_PORCH
 ))  &&
                        (R_v_cnt <= (C_V_SYNC_PULSE + C_V_BACK_PORCH +
C_V_ACTIVE_TIME))  ;
assign R_h_cnt_pos = (R_h_cnt - C_H_SYNC_PULSE - C_H_BACK_PORCH);
assign R_v_cnt_pos = (R_v_cnt - C_V_SYNC_PULSE - C_V_BACK_PORCH);
assign xpos[8:0] = R_h_cnt_pos[11:3];
assign ypos[8:0] = R_v_cnt_pos[11:3];
always @(posedge R_clk_25M)
begin
    if(I_rst_n)
        begin
            O_red   <=  4'b0000    ;
            O_green <=  4'b0000   ;
            O_blue  <=  4'b0000    ;
        end
    else if(W_active_flag)
        begin
            if(apple)//red apple
              begin
                O_red   <=  4'b1111    ;
                O_green <=  4'b0000   ;
                O_blue  <=  4'b0000    ;
              end
            else if(snake)//green snake
              begin
                O_red   <=  4'b0000    ;
                O_green <=  4'b1100   ;
                O_blue  <=  4'b0010    ;
```

```
            end
        else if(head)
            begin
                O_red   <=  4'b0110    ;
                O_green <=  4'b1011    ;
                O_blue  <=  4'b1110    ;
            end
        else if(border)//white border
            begin
                O_red   <=  4'b1111    ;
                O_green <=  4'b1111    ;
                O_blue  <=  4'b1111    ;
            end
        else begin//black background
                O_red   <=  4'b0000    ;
                O_green <=  4'b0000    ;
                O_blue  <=  4'b0000    ;
        end
    end
else
    begin
        O_red   <=  4'b0000    ;
        O_green <=  4'b0000    ;
        O_blue  <=  4'b0000    ;
    end
end
endmodule
```

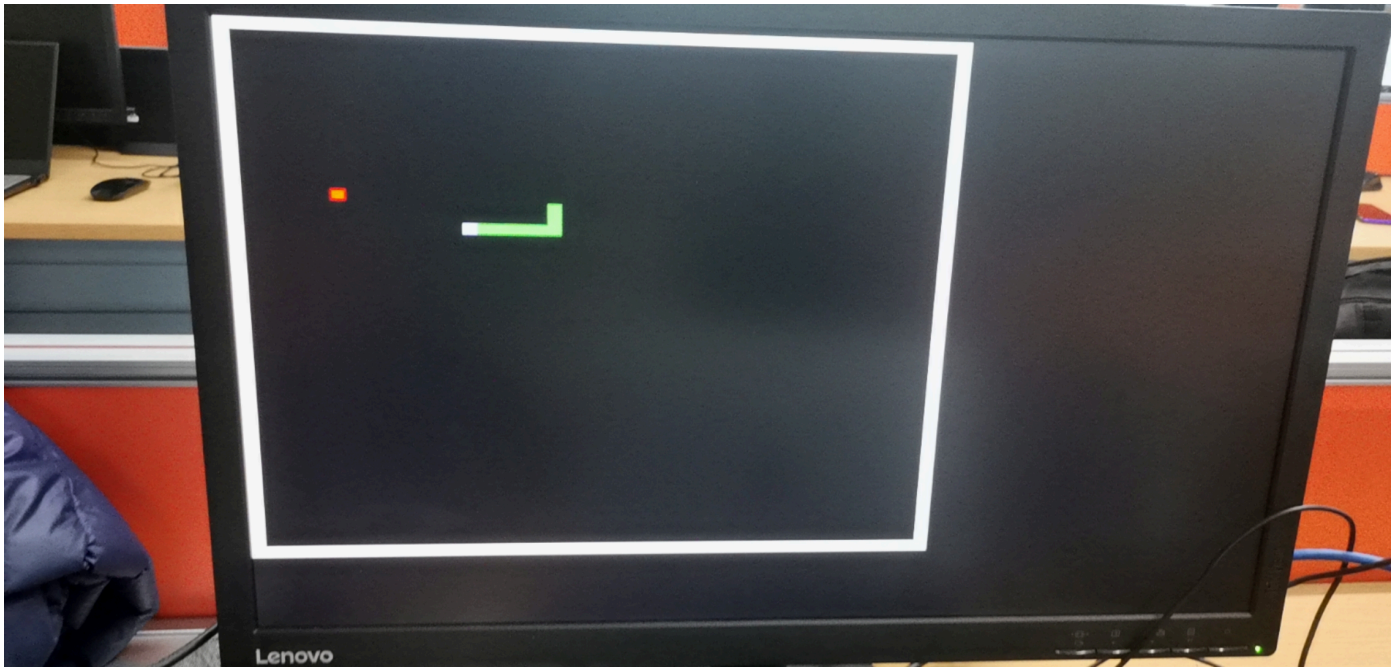
> 约束文件

```
## This file is a general .xdc for the Nexys4 DDR Rev. C
## To use it in a project:
## - uncomment the lines corresponding to used pins
## - rename the used ports (in each line, after get_ports) according to the top level signal names in the project


## Clock signal
```

```
set_property -dict { PACKAGE_PIN E3    IOSTANDARD LVCMOS33 } [get_ports { CLK100MHZ }];
#IO_L12P_T1_MRCC_35 Sch=clk100mhz
#create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports {CLK}];


##Switches
set_property -dict { PACKAGE_PIN J15   IOSTANDARD LVCMOS33 } [get_ports { pulse }];
#IO_L24N_T3_RS0_15 Sch=sw[0]


## LEDs
set_property -dict { PACKAGE_PIN H17   IOSTANDARD LVCMOS33 } [get_ports { LED[0] }];
#IO_L18P_T2_A24_15 Sch=led[0]
set_property -dict { PACKAGE_PIN K15   IOSTANDARD LVCMOS33 } [get_ports { LED[1] }];
#IO_L24P_T3_RS1_15 Sch=led[1]
set_property -dict { PACKAGE_PIN J13   IOSTANDARD LVCMOS33 } [get_ports { LED[2] }];
#IO_L17N_T2_A25_15 Sch=led[2]
set_property -dict { PACKAGE_PIN N14   IOSTANDARD LVCMOS33 } [get_ports { LED[3] }];
#IO_L8P_T1_D11_14 Sch=led[3]
set_property -dict { PACKAGE_PIN R18   IOSTANDARD LVCMOS33 } [get_ports { LED[4] }];
#IO_L7P_T1_D09_14 Sch=led[4]


##Buttons
set_property -dict { PACKAGE_PIN N17   IOSTANDARD LVCMOS33 } [get_ports { BTN[4] }];
#IO_L9P_T1_DQS_14 Sch=btnc
set_property -dict { PACKAGE_PIN M18   IOSTANDARD LVCMOS33 } [get_ports { BTN[0] }];
#IO_L4N_T0_D05_14 Sch=btnu
set_property -dict { PACKAGE_PIN P17   IOSTANDARD LVCMOS33 } [get_ports { BTN[1] }];
#IO_L12P_T1_MRCC_14 Sch=btnl
set_property -dict { PACKAGE_PIN M17   IOSTANDARD LVCMOS33 } [get_ports { BTN[2] }];
#IO_L10N_T1_D15_14 Sch=btnr
set_property -dict { PACKAGE_PIN P18   IOSTANDARD LVCMOS33 } [get_ports { BTN[3] }];
#IO_L9N_T1_DQS_D13_14 Sch=btnd


##VGA Connector
set_property -dict { PACKAGE_PIN A3    IOSTANDARD LVCMOS33 } [get_ports { VGA_R[0] }];
#IO_L8N_T1_AD14N_35 Sch=vga_r[0]
```

```
set_property -dict { PACKAGE_PIN B4    IOSTANDARD LVCMOS33 } [get_ports { VGA_R[1] }];
#IO_L7N_T1_AD6N_35 Sch=vga_r[1]
set_property -dict { PACKAGE_PIN C5    IOSTANDARD LVCMOS33 } [get_ports { VGA_R[2] }];
#IO_L1N_T0_AD4N_35 Sch=vga_r[2]
set_property -dict { PACKAGE_PIN A4    IOSTANDARD LVCMOS33 } [get_ports { VGA_R[3] }];
#IO_L8P_T1_AD14P_35 Sch=vga_r[3]
set_property -dict { PACKAGE_PIN C6    IOSTANDARD LVCMOS33 } [get_ports { VGA_G[0] }];
#IO_L1P_T0_AD4P_35 Sch=vga_g[0]
set_property -dict { PACKAGE_PIN A5    IOSTANDARD LVCMOS33 } [get_ports { VGA_G[1] }];
#IO_L3N_T0_DQS_AD5N_35 Sch=vga_g[1]
set_property -dict { PACKAGE_PIN B6    IOSTANDARD LVCMOS33 } [get_ports { VGA_G[2] }];
#IO_L2N_T0_AD12N_35 Sch=vga_g[2]
set_property -dict { PACKAGE_PIN A6    IOSTANDARD LVCMOS33 } [get_ports { VGA_G[3] }];
#IO_L3P_T0_DQS_AD5P_35 Sch=vga_g[3]
set_property -dict { PACKAGE_PIN B7    IOSTANDARD LVCMOS33 } [get_ports { VGA_B[0] }];
#IO_L2P_T0_AD12P_35 Sch=vga_b[0]
set_property -dict { PACKAGE_PIN C7    IOSTANDARD LVCMOS33 } [get_ports { VGA_B[1] }];
#IO_L4N_T0_35 Sch=vga_b[1]
set_property -dict { PACKAGE_PIN D7    IOSTANDARD LVCMOS33 } [get_ports { VGA_B[2] }];
#IO_L6N_T0_VREF_35 Sch=vga_b[2]
set_property -dict { PACKAGE_PIN D8    IOSTANDARD LVCMOS33 } [get_ports { VGA_B[3] }];
#IO_L4P_T0_35 Sch=vga_b[3]
set_property -dict { PACKAGE_PIN B11   IOSTANDARD LVCMOS33 } [get_ports { VGA_HS }];
#IO_L4P_T0_15 Sch=vga_hs
set_property -dict { PACKAGE_PIN B12   IOSTANDARD LVCMOS33 } [get_ports { VGA_VS }];
#IO_L3N_T0_DQS_AD1N_15 Sch=vga_vs
```

游戏展示



附件

- 【snake.v】Verilog HDL 源代码文件
- 【snake.xdc】Nexys 4 管脚约束文件
- 【snake.bit】Vivado 生成的可烧写文件