中国科学技术大学计算机学院

《数字电路实验》报告

实验题目：贪吃蛇游戏

学生姓名：Ouedraogo Ezekiel B.

学生学号：PL19215001

完成日期：1/12/2021

计算机实验教学中心制

2020 年 10 月

【实验环境】

Vivado

FPGA 板子

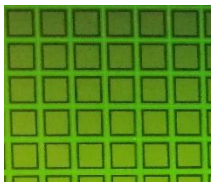
【实验过程】

首先设计游戏的背景

```
module cel_color(
    input VGA_clk,
    input displayArea,
    input [10:0] xCount, yCount,
    output reg cel, in_space, cel_border, out_space
    );
    parameter CEL_SIZE = 32;
    parameter OUT_SPACE = 1;
    parameter IN_SPACE = 2;
    parameter BORDER = 2;

    wire [10:0] q_x = xCount/(CEL_SIZE);
    wire [10:0] q_y = yCount/(CEL_SIZE);

    wire [7:0] cel_x = xCount - q_x*CEL_SIZE;
    wire [7:0] cel_y = yCount - q_y*CEL_SIZE;

    integer i = IN_SPACE + BORDER + OUT_SPACE;
    integer j = BORDER + OUT_SPACE;
    integer k = OUT_SPACE;

    always@(posedge VGA_clk)
    begin
        if(displayArea)
        begin
            cel <= (cel_x >= i) && (cel_y >= i) && (cel_x < CEL_SIZE -
i) && (cel_y < CEL_SIZE - i) ? 1 : 0;
            if(cel)
            begin
                in_space   <= 0;
                cel_border <= 0;
                out_space  <= 0;
            end
            else
            begin
```

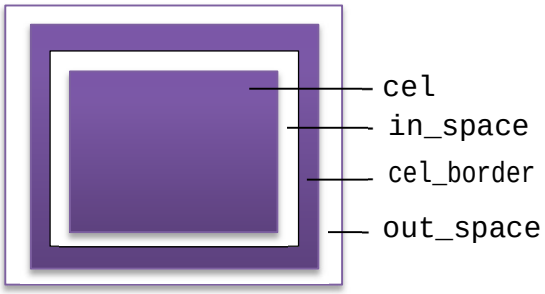
```
                in_space = (cel_x >= j) && (cel_y >= j) && (cel_x <
CEL_SIZE - j) && (cel_y < CEL_SIZE - j) ? 1 : 0;
                if(in_space)
                begin
                    cel_border <= 0;
                    out_space  <= 0;
                end
                else
                begin
                    cel_border = (cel_x >= k) && (cel_y >= k) && (cel_x
< CEL_SIZE - k) && (cel_y < CEL_SIZE - k) ? 1 : 0;
                    if(cel_border)
                        out_space  <= 0;
                    else
                        out_space = 1;
                end
            end
        end
    end
endmodule
```

结果



分析：输入信号为 VGA_clk，displayArea，xCount，yCount。其中 displayArea = 1 时，表示在显示区域。 xCount 和 yCount 表示位置。

看结果图可以发现背景时很多小正方形组成的。每个正方形含四部分，如下图:



到每个变量指的部分，其输出变量为 1 否则为 0。四部分都可以设置自己的颜色。所以用模块就可以显示背景，蛇，目标和墙， 只需

要到每个背景，蛇，目标或墙时候换颜色。

其次给目标设计一个输出随机位置的模块

```
module randomGrid(
    input VGA_clk,
    input [10:0] max_X, min_X,
    input [10:0] max_Y, min_Y,
    output reg [10:0] rand_X,
    output reg [10:0] rand_Y
    );
    always @(posedge VGA_clk)
    begin
        rand_X <= ((rand_X + 3) % (max_X-min_X)) + min_X;
        rand_Y <= ((rand_Y + 5) % max_Y-min_Y) + min_Y;
    end
endmodule
```

在设计一个 VGA 控制模块

```
module VGA_gen(
    input   CLK, RESET, //clk=65MHzz
    output displayArea,
    output reg [10:0] xCount,yCount, //x,y pixel
    output reg VGA_HS,VGA_VS
    );
    parameter H_CNT = 11'd1343; //136+160+1024+24=1344-1
    parameter V_CNT = 11'd805; //6+29+768+3=806-1

    reg h_de, v_de; //data enable
    reg [10:0] h_cnt,v_cnt;

    always@(posedge CLK)
    begin
        if(RESET)
            h_cnt <= 11'd0;
        else if(h_cnt>=H_CNT)
            h_cnt <= 11'd0;
        else
            h_cnt <= h_cnt + 11'd1;
    end
    always@(posedge CLK)
    begin
```

```
    if(RESET)
        v_cnt <= 11'd0;
    else if(h_cnt==H_CNT)
    begin
        if(v_cnt>=V_CNT)
            v_cnt <= 11'd0;
        else
            v_cnt <= v_cnt + 11'd1;
    end
end
always@(posedge CLK)
begin
    if(RESET)
        h_de <= 1'b0;
    else if((h_cnt>=296)&&(h_cnt<=1319))
        h_de <= 1'b1;
    else
        h_de <= 1'b0;
end
always@(posedge CLK)
begin
    if(RESET)
        v_de <= 1'b0;
    else if((v_cnt>=35)&&(v_cnt<=802))
        v_de <= 1'b1;
    else
        v_de <= 1'b0;
end
always@(posedge CLK)
begin
    if(RESET)
        VGA_HS <= 1'b1;
    else if(h_cnt<=11'd135)
        VGA_HS <= 1'b0;
    else
        VGA_HS <= 1'b1;
end
always@(posedge CLK)
begin
    if(RESET)

        VGA_VS <= 1'b1;
    else if(v_cnt<=11'd5)
        VGA_VS <= 1'b0;
```

```
        else
            VGA_VS <= 1'b1;
    end
    always@(posedge CLK)
    begin
        if(h_de == 1'h0)
            xCount <= 11'h0;
        else
            xCount <= xCount + 11'h1;
    end
    always@(negedge h_de)
    begin
        if(v_de == 1'h0)
            yCount <= 11'h0;
        else
            yCount <= yCount + 11'h1;
    end
    assign displayArea = (v_de==1 && h_de==1);
endmodule
```

最后设计游戏的基本功能

```
module Play(
    input CLK, RESET,
    input UP, LEFT, DOWN, RIGHT,
    output reg [3:0] VGA_R, VGA_G, VGA_B,
    output VGA_HS, VGA_VS
    );

    parameter SCREEN_H = 1024;
    parameter SCREEN_V = 768;
    parameter UNIT = 32;
    parameter SNAKE_MAX_SIZE = 20;

    wire VGA_clk, locked; //65 MHz
    wire displayArea;
    wire [10:0] xCount, yCount; //x ,y pixel
    reg [3:0] direction;
    reg [10:0] snakeX[0:SNAKE_MAX_SIZE];
    reg [10:0] snakeY[0:SNAKE_MAX_SIZE];
    reg [SNAKE_MAX_SIZE:0] snakeBody;
    reg [7:0] snake_size;
    reg game_over;
    reg apple, border;
```

```
    reg [10:0] appleX = 64;
    reg [10:0] appleY = 64;
    wire [10:0]rand_X;
    wire [10:0]rand_Y;
    reg update;
    wire cel;
    wire in_space;
    wire cel_border;
    wire out_space;
    reg [27:0] count, max_count;

    // generate VGA clock (65 MHz) from input clock (100 MHz)
    clk_wiz_0 clk_wiz_0(
        .clk_in1 (CLK),
        .reset (RESET),
        .clk_out1 (VGA_clk),
        .locked (locked)
    );

    //VGA controller
    VGA_gen VGA_gen(
        .CLK(VGA_clk), .RESET(~locked),
        .displayArea(displayArea),
        .xCount(xCount),.yCount(yCount),
        .VGA_HS(VGA_HS),.VGA_VS(VGA_VS)
    );

    //get random number to place the apple
    randomGrid randomGrid(
        .VGA_clk(VGA_clk),
        .max_X(SCREEN_H-UNIT), .min_X(UNIT),
        .max_Y(SCREEN_V-UNIT), .min_Y(UNIT),
        .rand_X(rand_X),
        .rand_Y(rand_Y)
    );

    //color cels
    cel_color cel_color(
        .VGA_clk(VGA_clk),
        .displayArea(displayArea),
        .xCount(xCount), .yCount(yCount),
        .cel(cel), .in_space(in_space), .cel_border(cel_border), .out
_space(out_space)
        );
```

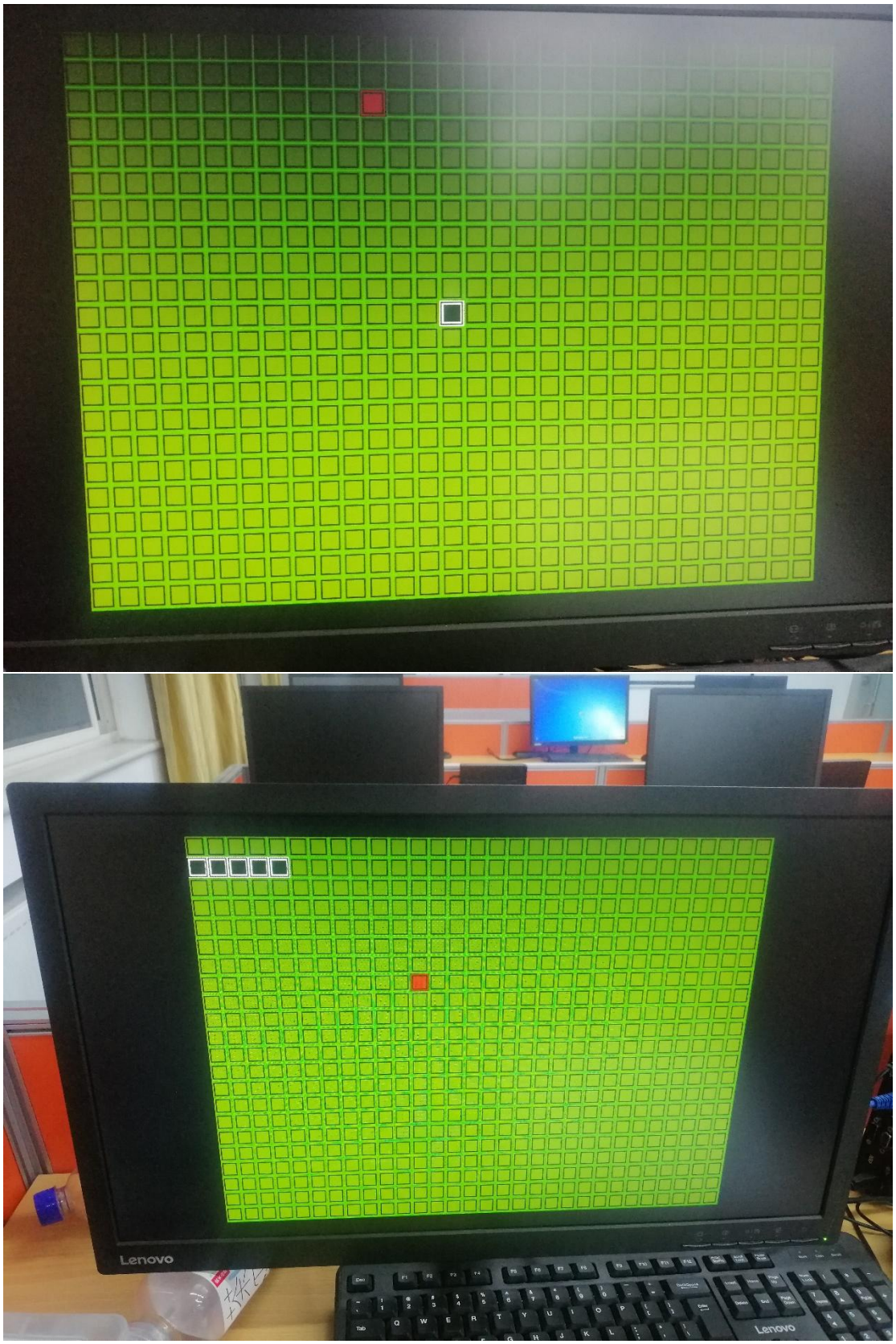
```
    integer i,j;

    always@(posedge VGA_clk or posedge RESET)
    begin
        if(RESET)
        begin
            // place the snake head at display center
            i = SCREEN_H/2 - UNIT/2;
            j = SCREEN_V/2 - UNIT/2;
            snakeX[0] <= (i - i%UNIT);
            snakeY[0] <= (j - j%UNIT);

            // place apple
            i = rand_X;
            j = rand_Y;
            appleX <= (i - i%UNIT);
            appleY <= (j - j%UNIT);

            for(i = 1; i < SNAKE_MAX_SIZE; i = i + 1)
            begin
                // place the invisible snake parts outside the scanning
area
                snakeX[i] <= SCREEN_H;
                snakeY[i] <= SCREEN_V;
            end
            max_count <= 28'd65_000_000;
            snake_size <= 1;
            game_over <= 0;
        end
        else if(~game_over)
        begin
            if(update)
            begin
                for(i = 1; i < snake_size; i = i + 1)
                begin
                    snakeX[i] <= snakeX[i - 1];
                    snakeY[i] <= snakeY[i - 1];
                end
                case(direction)
                    4'b1: snakeY[0] <= (snakeY[0] - UNIT);
                    4'b10: snakeX[0] <= (snakeX[0] - UNIT);
                    4'b100: snakeY[0] <= (snakeY[0] + UNIT);
                    4'b1000: snakeX[0] <= (snakeX[0] + UNIT);
```

```
                endcase
            end
            else
            begin
            // Detect if snake head hit the apple
                if (apple && snakeBody[0])
                begin
                    i = rand_X;
                    j = rand_Y;
                    appleX <= (i - i%UNIT);
                    appleY <= (j - j%UNIT);
                    if(snake_size < SNAKE_MAX_SIZE )
                        snake_size <= snake_size + 1;
                end
            // Detect if snake head hit border
                else if (border && snakeBody[0])
                    game_over <= 1'b1;
            // Detect if snake head hit the snake body
                else if (|snakeBody[SNAKE_MAX_SIZE - 1 :1] &&
snakeBody[0])
                    game_over <= 1'b1;
            end
        end
    end

    //check update
    always@(posedge VGA_clk or posedge RESET)
    begin
        if(RESET)
        begin
            count <= 28'd0;
            update <= 0;
        end
        else if(count >= max_count)
        begin
            count <= 28'd0;
            update <= 1;
        end
        else
        begin
            count <= count + 28'd1;
            update <= 0;
        end
    end
```

```
//color cel
always@(posedge VGA_clk)
begin
    if(displayArea)
    begin
        if(border)
        begin
            if(cel)             {VGA_R, VGA_G, VGA_B} <= 12'h3;
            else if(in_space)   {VGA_R, VGA_G, VGA_B} <= 12'h0;
            else if(cel_border) {VGA_R, VGA_G, VGA_B} <= 12'h3;
            else if(out_space)  {VGA_R, VGA_G, VGA_B} <= 12'h0;
            else                {VGA_R, VGA_G, VGA_B} <= 12'h0;
        end
        else if(|snakeBody)
        begin
            if(cel)             {VGA_R, VGA_G, VGA_B} <= 12'h0;
            else if(in_space)   {VGA_R, VGA_G, VGA_B} <= 12'hfff;
            else if(cel_border) {VGA_R, VGA_G, VGA_B} <= 12'h0;
            else if(out_space)  {VGA_R, VGA_G, VGA_B} <= 12'hfff;
            else                {VGA_R, VGA_G, VGA_B} <= 12'h0;
        end
        else if(apple)
        begin
            if(cel)             {VGA_R, VGA_G, VGA_B} <= 12'h700;
            else if(in_space)   {VGA_R, VGA_G, VGA_B} <= 12'h0;
            else if(cel_border) {VGA_R, VGA_G, VGA_B} <= 12'h700;
            else if(out_space)  {VGA_R, VGA_G, VGA_B} <= 12'h0;
            else                {VGA_R, VGA_G, VGA_B} <= 12'h0;
        end
        else
        begin
            if(cel)             {VGA_R, VGA_G, VGA_B} <= 12'h340;
            else if(in_space)   {VGA_R, VGA_G, VGA_B} <= 12'h3;
            else if(cel_border) {VGA_R, VGA_G, VGA_B} <= 12'h340;
            else if(out_space)  {VGA_R, VGA_G, VGA_B} <= 12'h70;
            else                {VGA_R, VGA_G, VGA_B} <= 12'h0;
        end
    end
end

//init direction
always@(posedge CLK)
begin
```

```
        if(RESET)
            direction  <= 4'b0;
        else if(UP)
            direction  <= 4'b1;
        else if(LEFT)
            direction  <= 4'b10;
        else if(DOWN)
            direction  <= 4'b100;
        else if(RIGHT)
            direction  <= 4'b1000;
        else
            direction <= direction;
    end

    // Detect if the VGA scanning is hitting the border
    always @(posedge VGA_clk)
    begin
        border <= ((xCount <= UNIT) || (xCount > SCREEN_H - UNIT) ||
(yCount <= UNIT) || (yCount > (SCREEN_V - UNIT)));
    end

    // Detect if the VGA scanning is hitting the apple
    always @(posedge VGA_clk)
    begin
        apple <= ((xCount >= appleX) & (yCount >= appleY) & (xCount <
appleX + UNIT) & (yCount < appleY + UNIT));
    end

    // Detect if the VGA scanning is hitting the snake head or snake body
    always@(posedge VGA_clk)
    begin
        for(i = 0; i < SNAKE_MAX_SIZE; i = i + 1)
            snakeBody[i] <= ((xCount >= snakeX[i]) & (yCount >= snakeY[i])
& (xCount < snakeX[i] + UNIT) & (yCount < snakeY[i] + UNIT));
    end

endmodule
```

结果



XDC 文件

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

#set_property -dict { PACKAGE_PIN C12   IOSTANDARD LVCMOS33 } [get_ports { CPU_RESETN }]; #IO_L3P_T0_DQS_AD1P_15 Sch=cpu_resetn

set_property -dict { PACKAGE_PIN N17   IOSTANDARD LVCMOS33 } [get_ports { RESET }]; #IO_L9P_T1_DQS_14 Sch=btnc
set_property -dict { PACKAGE_PIN M18   IOSTANDARD LVCMOS33 } [get_ports { UP }]; #IO_L4N_T0_D05_14 Sch=btnu
set_property -dict { PACKAGE_PIN P17   IOSTANDARD LVCMOS33 } [get_ports { LEFT }]; #IO_L12P_T1_MRCC_14 Sch=btnl
set_property -dict { PACKAGE_PIN P18   IOSTANDARD LVCMOS33 } [get_ports { DOWN }]; #IO_L9N_T1_DQS_D13_14 Sch=btnd
set_property -dict { PACKAGE_PIN M17   IOSTANDARD LVCMOS33 } [get_ports { RIGHT }]; #IO_L10N_T1_D15_14 Sch=btnr

##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
```

FPGA 的上下左右按键用来控制蛇的行动。

【总结与思考】

该实验花的大部分时间在了设计完美的图像，所以有些的功能只是做出了最基础的比如蛇吃一个目标后增长，撞到墙游戏暂停，等。