Lab4 实验报告 - 存储器与显示控制器

PB17000144 黄业琦

实验要求

控制画笔在800x600分辨率的显示器上随意涂画，画笔的颜色12位(红r绿g蓝b各4位)，绘画区域位于屏幕正中部，大小为256x256

- 画笔位置(x, y): x = y = 0 ~ 255, 复位时 (128, 128)
- 移动画笔(dir): 上/下/左/右按钮
- 画笔颜色(rgb): 12位开关设置
- 绘画状态(draw): 1-是, 0-否；处于绘画状态时，移动画笔同时绘制颜色，否则仅移动画笔

PCU：Paint Control Unit，绘画控制单元，修改VRAM中像素信息

- 通过12个拨动开关设置像素颜色 (rgb)
- 通过上/下/左/右(dir)按钮开关，移动画笔位置(x, y)
- 直角移动：单一按钮按下一次，x或y增加或减小1
- 对角移动：两按钮同时按下一次，x和y同时加或减1
- 连续移动：按钮按下超过t秒后，等效为s速率的连续点击，直至松开（调试时确定合适的t和s取值）

绘画 (draw=1) 时，依据 rgb 和 (x, y)，通过写端口(paddr, pdata, we) 修改VRAM像素信息

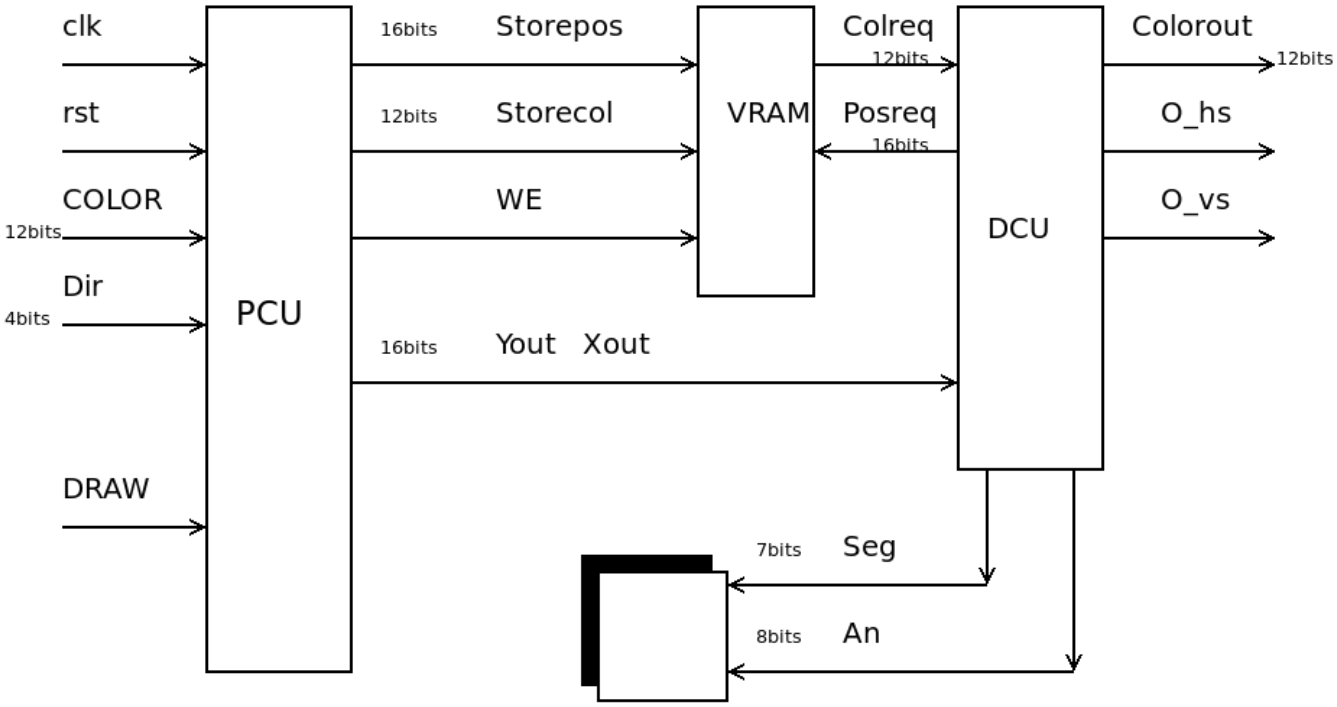
VRAM：视频存储器，存储256x256个像素的颜色信息，采用简单双端口存储器实现

- paddr, pdata, we: 地址、数据、写使能，用于绘画的同步写端口
- vaddr, vdata: 地址、数据，用于显示的异步读端口

实验环境

Linux下编程调试和仿真，使用IVerilog，GtkWave系列工具。 Windows下用于生成比特流文件，使用Vivado 2018.2，Verilog HDL。 所有下载均在Nexsy4-DDR实验板完成。

模块设计



PCU输出正确的坐标位置

VRAM存储各个未知的颜色

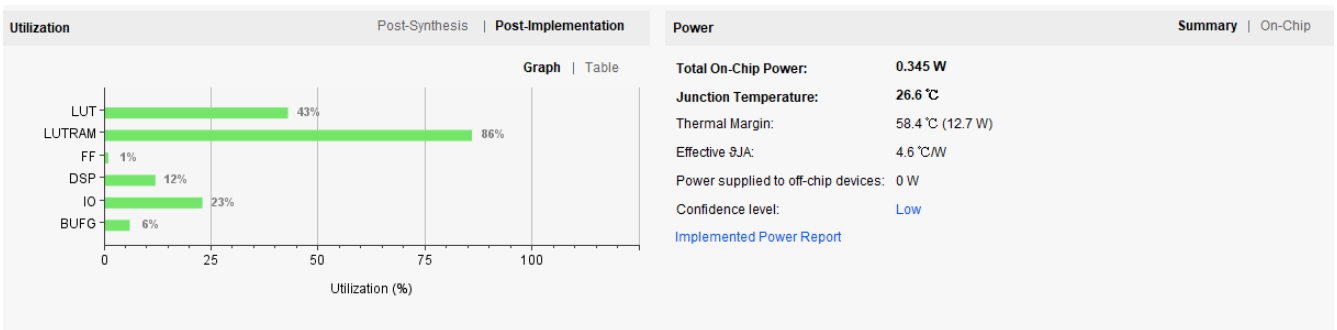
DCU负责处理输出颜色

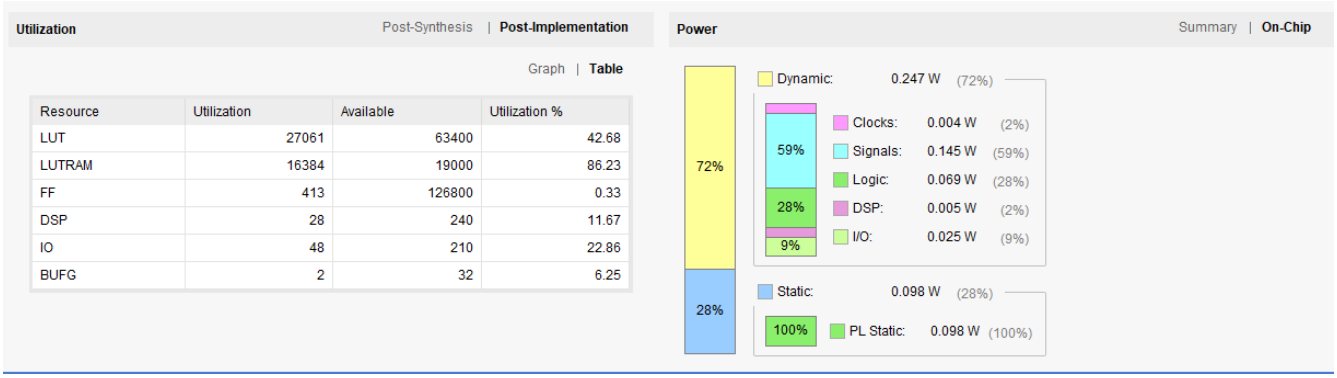
同时增加数码管，以方便找到笔的位置

增加功能：

- 数码管显示笔的坐标未知
- 长按按键更改了功能设计，不是单纯连续移动，如果落笔，画出虚线
- 十字坐标显示笔
- 斜角移动

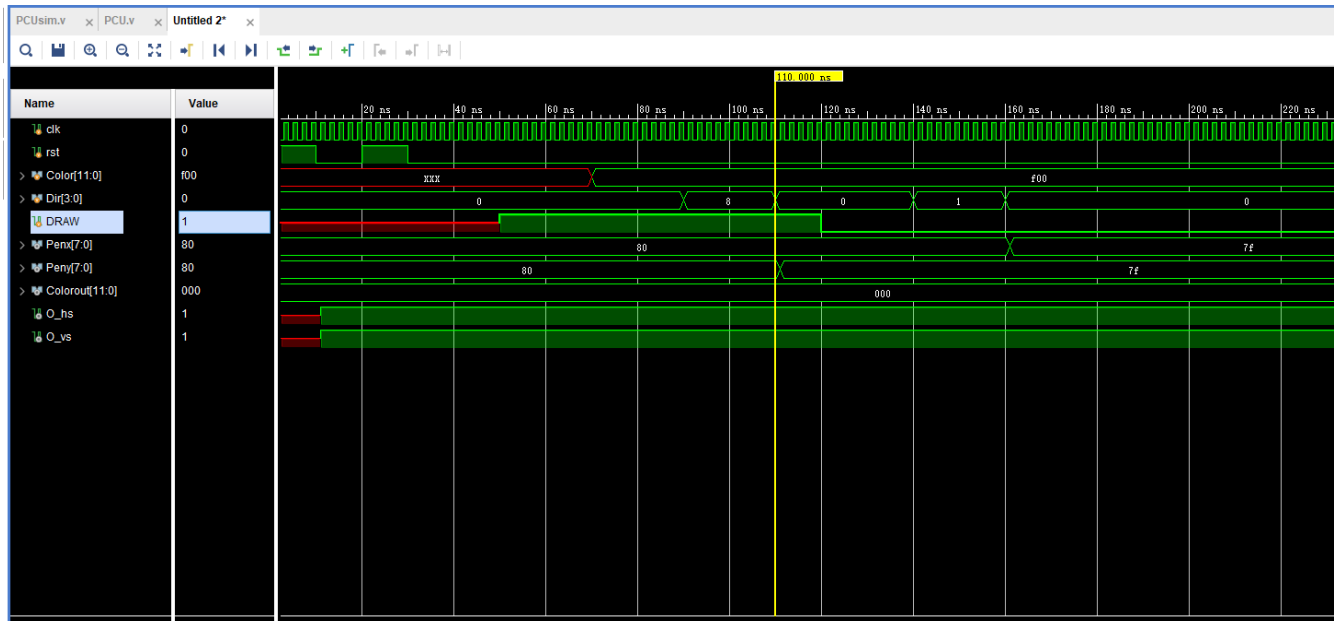
程序性能





程序模拟

模拟是割掉了DCU模块，将Color和Pen的坐标输出



实现方法

本次实验为了挑战自己，拒绝了使用状态机，采用纯逻辑的方法，对按键行为进行了周密的讨论：

- 某按键由未按到按
    - 更新计数器 = 0
    - 短按状态设为 1
    - 若其他按键有被按下
        - 斜角状态置为1
        - 获取斜角方位
- 某按键由按到继续按
    - 计数器+1
    - 若按下时间达到2秒
        - 每再按下1秒，长按计数器+3
        - 短按状态置为0
- 某按键由按到松开

    - 若斜角移动
        - 若还有其他键未松开，直接计数器置0，不移动
        - 若其他按键全部松开，斜角移动
    - 非斜角移动
        - 短按标记为1，短移动，否则不动
- 长按键计数器非0
    - 短移一次，计数器减一

*技巧：这里我将**PCU**和**DCU**的**clk**设置为不同，**PCU**的**clk**为**50Hz**，**DCU**的**clk**为**25Hz**，这样我们移动快，扫描慢，短移、斜移无影响，但是长移时会再某些周期无法涂上颜色，即达到了虚线的目的*

代码：

1. PCU code

```
`timescale 1ns / 1ps

`define ColorBUS 11:0
`define DirBUS   3:0
`define PenBUS   7:0

(* use_dsp = "yes" *)
module PCU(
    input clk,
    input rst,
    input [`ColorBUS] Color,
    input [  `DirBUS] Dir,
    input DRAW,
    //output reg [`PenBUS] Storex,
   // output reg [`PenBUS] Storey,
    //output reg [`ColorBUS] Storecol,
    ///output reg we,
    //output reg [`PenBUS] Penx,
    //output reg [`PenBUS] Peny,
    output [`ColorBUS] Colorout,
    output O_hs,
    output O_vs,
    /*,
    output Dirdelay,
    output cu,
    output cl,
    output cr,
    output cd,
    output Xcor,
    output Ycor,
    output Xflag*/
    output reg [7:0] an,
    output reg [6:0] seg
    );
```

```
reg [`PenBUS] Xcor;
reg [`PenBUS] Ycor;
reg [`DirBUS] Dirdelay;

reg [`PenBUS] Storex;
reg [`PenBUS] Storey;
reg [`ColorBUS] Storecol;
reg we;
reg [`PenBUS] Penx;
reg [`PenBUS] Peny;

reg [3:0] Q [7:0];
reg [7:0] avail;


reg Xflag;
reg Shortmove;
reg [31:0] cu;
reg [31:0] cd;
reg [31:0] cl;
reg [31:0] cr;
reg [13:0] lumove;
reg [13:0] ldmove;
reg [13:0] llmove;
reg [13:0] lrmove;

reg [3:0] Movex;

always @(posedge clk or negedge rst)
begin
    if (rst)
    begin
        Xcor <= 8'd128;
        Ycor <= 8'd128;
        Dirdelay <= 0;
        Shortmove <= 1;
        Xflag <= 0;
        cu <= 0;
        cd <= 0;
        cr <= 0;
        cl <= 0;
        ldmove <= 0;
        lumove <= 0;
        llmove <= 0;
        lrmove <= 0;
    end
    else
    begin
        if (Dir[0] && !Dirdelay[0]) // up appear
        begin
```

```
            cu <= 0;
            Shortmove <= 1;
            if (cd + cl + cr > 0)
            begin
                Xflag <= 1;
                if (cr > 0) Movex[0] <= 1;
                else Movex[3] <= 1;
            end
        end

        if (Dir[0] && Dirdelay[0]) // up remain
        begin
            cu <= cu + 1;
            if (!Xflag && cu > 200000000)
            begin
                Shortmove <= 0;
                if ((cu - 200000000) % 100000000 == 0)
                lumove <= lumove + 3;
            end
        end

        if (!Dir[0] && Dirdelay[0]) // up end
        begin
            if (Xflag)
            begin
                if (cd + cl + cr == 0)
                begin
                    if (Movex[0])
                    begin
                        Xcor <= (Xcor == 0 ? 0 : Xcor - 1);
                        Ycor <= (Ycor == 255 ? 255 : Ycor + 1);
                    end
                    else
                    begin
                        Xcor <= (Xcor == 0 ? 0 : Xcor - 1);
                        Ycor <= (Ycor == 0 ? 0 : Ycor - 1);
                    end
                    cu <= 0;
                    Movex <= 0;
                    Xflag <= 0;
                end
                else
                    cu <= 0;
            end
            else
            begin
                cu <= 0;
                if (Shortmove)
                    Xcor <= (Xcor == 0 ? 0 : Xcor - 1);
            end
        end
```

```
if (lumove > 0)
begin
    lumove <= lumove - 1;
    Xcor <= (Xcor == 0 ? 0 : Xcor - 1);
end


if (Dir[1] && !Dirdelay[1]) // right appear
begin
    cr <= 0;
    Shortmove <= 1;
    if (cd + cl + cu > 0)
    begin
        Xflag <= 1;
        if (cu > 0) Movex[0] <= 1;
        else Movex[1] <= 1;
    end
end

if (Dir[1] && Dirdelay[1]) // right remain
begin
    cr <= cr + 1;
    if (!Xflag && cr > 200000000)
    begin
        Shortmove <= 0;
        if ((cr - 200000000) % 100000000 == 0)
        lrmove <= lrmove + 3;
    end
end

if (!Dir[1] && Dirdelay[1]) // right end
begin
    if (Xflag)
    begin
        if (cd + cl + cu == 0)
        begin
            if (Movex[0])
            begin
                Xcor <= (Xcor == 0 ? 0 : Xcor - 1);
                Ycor <= (Ycor == 255 ? 255 : Ycor + 1);
            end
            else
            begin
                Xcor <= (Xcor == 255 ? 255 : Xcor + 1);
                Ycor <= (Ycor == 255 ? 255 : Ycor + 1);
            end
            cr <= 0;
            Movex <= 0;
            Xflag <= 0;
        end
```

```
            else
                cr <= 0;
        end
        else
        begin
            cr <= 0;
            if (Shortmove)
                Ycor <= (Ycor == 255 ? 255 : Ycor + 1);
        end
    end

    if (lrmove > 0)
    begin
        lrmove <= lrmove - 1;
        Ycor <= (Ycor == 255 ? 255 : Ycor + 1);
    end


    if (Dir[2] && !Dirdelay[2]) // down appear
    begin
        cd <= 0;
        Shortmove <= 1;
        if (cr + cl + cu > 0)
        begin
            Xflag <= 1;
            if (cd > 0) Movex[1] <= 1;
            else Movex[2] <= 1;
        end
    end

    if (Dir[2] && Dirdelay[2]) // down remain
    begin
        cd <= cd + 1;
        if (!Xflag && cd > 200000000)
        begin
            Shortmove <= 0;
            if ((cd - 200000000) % 100000000 == 0)
            ldmove <= ldmove + 3;
        end
    end

    if (!Dir[2] && Dirdelay[2]) // down end
    begin
        if (Xflag)
        begin
            if (cr + cl + cu == 0)
            begin
                if (Movex[1])
                begin
                    Xcor <= (Xcor == 255 ? 255 : Xcor + 1);
                    Ycor <= (Ycor == 255 ? 255 : Ycor + 1);
```

```
                end
                else
                begin
                    Xcor <= (Xcor == 255 ? 255 : Xcor + 1);
                    Ycor <= (Ycor == 0 ? 0 : Ycor - 1);
                end
                cd <= 0;
                Movex <= 0;
                Xflag <= 0;
            end
            else
                cd <= 0;
        end
        else
        begin
            cd <= 0;
            if (Shortmove)
                Xcor <= (Xcor == 255 ? 255 : Xcor + 1);
        end
    end

    if (ldmove > 0)
    begin
        ldmove <= ldmove - 1;
        Xcor <= (Xcor == 255 ? 255 : Xcor + 1);
    end

    if (Dir[3] && !Dirdelay[3]) // left appear
    begin
        cl <= 0;
        Shortmove <= 1;
        if (cr + cd + cu > 0)
        begin
            Xflag <= 1;
            if (cd > 0) Movex[2] <= 1;
            else Movex[3] <= 1;
        end
    end

    if (Dir[3] && Dirdelay[3]) // left remain
    begin
        cl <= cl + 1;
        if (!Xflag && cl > 200000000)
        begin
            Shortmove <= 0;
            if ((cl - 200000000) % 100000000 == 0)
            llmove <= llmove + 3;
        end
    end

    if (!Dir[3] && Dirdelay[3]) // left end
```

```
            begin
                if (Xflag)
                begin
                    if (cr + cd + cu == 0)
                    begin
                        if (Movex[2])
                        begin
                            Xcor <= (Xcor == 255 ? 255 : Xcor + 1);
                            Ycor <= (Ycor == 0 ? 0 : Ycor - 1);
                        end
                        else
                        begin
                            Xcor <= (Xcor == 0 ? 0 : Xcor - 1);
                            Ycor <= (Ycor == 0 ? 0 : Ycor - 1);
                        end
                        cl <= 0;
                        Movex <= 0;
                        Xflag <= 0;
                    end
                    else
                        cl <= 0;
                end
                else
                begin
                    cl <= 0;
                    if (Shortmove)
                        Ycor <= (Ycor == 0 ? 0 : Ycor - 1);
                end
            end

            if (llmove > 0)
            begin
                llmove <= llmove - 1;
                Ycor <= (Ycor == 0 ? 0 : Ycor - 1);
            end

            Dirdelay <= Dir;
        end
    end

    always @*
    begin
        we   <= DRAW;
        Penx <= Xcor;
        Peny <= Ycor;
        Storecol <= Color;
        Storex <= Xcor;
        Storey <= Ycor;
    end

    reg [3:0] num;
```

```
wire [6:0] seg0[7:0];
reg [30:0] count;
reg clk1;

BCD27 B0(Q[0],seg0[0]);
BCD27 B1(Q[1],seg0[1]);
BCD27 B2(Q[2],seg0[2]);
BCD27 B3(Q[3],seg0[3]);
BCD27 B4(Q[4],seg0[4]);
BCD27 B5(Q[5],seg0[5]);
BCD27 B6(Q[6],seg0[6]);
BCD27 B7(Q[7],seg0[7]);

initial
begin
an<=8'b11111111;
clk1<=0;
num<=0;
count<=0;
end

always @*
begin
    avail <= 8'b01110111;

    Q[0] <= Ycor % 10;
    Q[1] <= Ycor / 10 % 10;
    Q[2] <= Ycor / 100;

    Q[4] <= Xcor % 10;
    Q[5] <= Xcor / 10 % 10;
    Q[6] <= Xcor / 100;
end

always@(posedge clk)
    if(count>=31'd19999)
    begin
        clk1<=~clk1;
        count=31'b0;
        num <= (num+1)%8;
    end
    else
    count=count+1;

always@*
case(num)
    0:begin
        an <=8'b11111110;
        seg<=seg0[0];
        if (!avail[0])
        seg<=7'b1111111;
```

```
        end
        1:begin
            an <=8'b11111101;
            seg<=seg0[1];
            if (!avail[1])
            seg<=7'b1111111;
        end
        2:begin
            an <=8'b11111011;
            seg<=seg0[2];
            if (!avail[2])
            seg<=7'b1111111;
        end
        3:begin
            an <=8'b11110111;
            seg<=seg0[3];
            if (!avail[3])
            seg<=7'b1111111;
        end
        4:begin
            an <=8'b11101111;
            seg<=seg0[4];
            if (!avail[4])
            seg<=7'b1111111;
        end
        5:begin
            an <=8'b11011111;
            seg<=seg0[5];
            if (!avail[5])
            seg<=7'b1111111;
        end
        6:begin
            an <=8'b10111111;
            seg<=seg0[6];
            if (!avail[6])
            seg<=7'b1111111;
        end
        7:begin
            an <=8'b01111111;
            seg<=seg0[7];
            if (!avail[7])
            seg<=7'b1111111;
        end
    endcase
    DCU  Screen (clk,rst,we,Penx,Peny,Storecol,O_hs,O_vs,Colorout);
endmodule
```

2. DCU code

```
```

```
`timescale 1ns / 1ps

`define ColorBUS 11:0
`define DirBUS   3:0
`define PenBUS   7:0

module DCU
(board_clk,reset,we,Penx,Peny,colorst,vga_h_sync,vga_v_sync,color);
input board_clk;
input reset;
input Penx;
input we;
input Peny;
input colorst;
output vga_h_sync, vga_v_sync;
output color;

wire [`ColorBUS] colorst;
reg [`ColorBUS] Color;
reg [19:0] CounterX;
reg [19:0] CounterY;
wire we;
wire [7:0] Xreq;
wire [7:0] Yreq;
reg vga_HS, vga_VS;
wire [7:0] Penx;
wire [7:0] Peny;
reg inDisplayArea;
reg [`ColorBUS] color;
wire [`ColorBUS] Colout;

    reg [27:0]  DIV_CLK;
    always @ (posedge board_clk, posedge reset)
    begin : CLOCK_DIVIDER
      if (reset)
            begin
                DIV_CLK <= 0;
            end
      else
            DIV_CLK <= DIV_CLK + 1'b1;
    end
wire clk;
assign clk = DIV_CLK[0];

always @(posedge clk, posedge reset)
begin
   if(reset)
      CounterX <= 0;
   else if(CounterX==19'd1056)
       CounterX <= 0;
   else
```

```
        CounterX <= CounterX +1;
end
always @(posedge clk, posedge reset)
begin
   if(reset)
      CounterY<=0;
   else if(CounterY==19'd628)
      CounterY<=0;
   else if(CounterX==19'd1056)
      CounterY <= CounterY + 1;
end
always @(posedge clk)
begin
    vga_HS <= (CounterX>855 && CounterX<976);
    vga_VS <= (CounterY>636 && CounterY<643);
end


always @(posedge clk)
   if(reset)
      inDisplayArea<=0;
   else
       inDisplayArea <= (CounterX<800) && (CounterY<600);

assign vga_h_sync = ~vga_HS;
assign vga_v_sync = ~vga_VS;

assign Yreq = CounterY - 172 ;
assign Xreq = CounterX - 272 ;
VRAM Ld (Penx,Peny,colorst,we,clk,Xreq,Yreq,Colout);
always @*
begin
if (inDisplayArea)
begin
    if (CounterX >= 272 && CounterX <= 272+255 && CounterY>=172 &&
CounterY<=172+255)
    begin
        if ((Xreq == Penx && (Yreq > Peny && Yreq-Peny <=3 || Peny > Yreq
&& Peny-Yreq <=3)) ||
            (Yreq == Peny && (Xreq > Penx && Xreq-Penx <=3 || Penx > Xreq
&& Penx-Xreq <=3)))
            color <= 0;
        else
            color <= ~Colout;
    end
    else
    begin
        color <= 0;
    end
end
else
```

```
color = 0;
end

endmodule
```

3. VRAM code

```
`timescale 1ns / 1ps

`define ColorBUS 11:0
`define DirBUS   3:0
`define PenBUS   7:0

module VRAM(
        input [`PenBUS] Xcor,
        input [`PenBUS] Ycor,
        input [`ColorBUS] Color,
        input we,
        input clk,
        input [`PenBUS] Xreq,
        input [`PenBUS] Yreq,
        output [`ColorBUS] Colout
    );

    wire [15:0] address;
    assign address = {Xcor,Ycor};
    wire [15:0] reqadd;
    assign reqadd = {Xreq,Yreq};
    dist_mem_gen_0 St
(.a(address),.d(Color),.dpra(reqadd),.clk(clk),.we(we),.dpo(Colout));
endmodule
```

4. 数码管处理

```
`timescale 1ns / 1ps

module BCD27(
    input [3:0]m,
    output [6:0] out
    );
    reg[6:0]seg;
    assign out = seg;
    always@(m)
    case(m)
                4'b0000:seg=7'b1000000;
                4'b0001:seg=7'b1111001;
                4'b0010:seg=7'b0100100;
```

```
                4'b0011:seg=7'b0110000;
                4'b0100:seg=7'b0011001;
                4'b0101:seg=7'b0010010;
                4'b0110:seg=7'b0000010;
                4'b0111:seg=7'b1111000;
                4'b1000:seg=7'b0000000;
                4'b1001:seg=7'b0010000;
                default:seg=7'b1111111;
                endcase
endmodule
```

5. 约束引脚文件

```
## 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 {
clk }]; #IO_L12P_T1_MRCC_35 Sch=clk
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 {
Color[0] }]; #IO_L24N_T3_RS0_15 Sch=sw[0]
set_property -dict { PACKAGE_PIN L16   IOSTANDARD LVCMOS33 } [get_ports {
Color[1] }]; #IO_L3N_T0_DQS_EMCCLK_14 Sch=sw[1]
set_property -dict { PACKAGE_PIN M13   IOSTANDARD LVCMOS33 } [get_ports {
Color[2] }]; #IO_L6N_T0_D08_VREF_14 Sch=sw[2]
set_property -dict { PACKAGE_PIN R15   IOSTANDARD LVCMOS33 } [get_ports {
Color[3] }]; #IO_L13N_T2_MRCC_14 Sch=sw[3]
set_property -dict { PACKAGE_PIN R17   IOSTANDARD LVCMOS33 } [get_ports {
Color[4] }]; #IO_L12N_T1_MRCC_14 Sch=sw[4]
set_property -dict { PACKAGE_PIN T18   IOSTANDARD LVCMOS33 } [get_ports {
Color[5] }]; #IO_L7N_T1_D10_14 Sch=sw[5]
set_property -dict { PACKAGE_PIN U18   IOSTANDARD LVCMOS33 } [get_ports {
Color[6] }]; #IO_L17N_T2_A13_D29_14 Sch=sw[6]
set_property -dict { PACKAGE_PIN R13   IOSTANDARD LVCMOS33 } [get_ports {
Color[7] }]; #IO_L5N_T0_D07_14 Sch=sw[7]
set_property -dict { PACKAGE_PIN T8    IOSTANDARD LVCMOS18 } [get_ports {
Color[8] }]; #IO_L24N_T3_34 Sch=sw[8]
set_property -dict { PACKAGE_PIN U8    IOSTANDARD LVCMOS18 } [get_ports {
Color[9] }]; #IO_25_34 Sch=sw[9]
set_property -dict { PACKAGE_PIN R16   IOSTANDARD LVCMOS33 } [get_ports {
Color[10] }]; #IO_L15P_T2_DQS_RDWR_B_14 Sch=sw[10]
```

```
set_property -dict { PACKAGE_PIN T13   IOSTANDARD LVCMOS33 } [get_ports {
Color[11] }]; #IO_L23P_T3_A03_D19_14 Sch=sw[11]
#set_property -dict { PACKAGE_PIN H6    IOSTANDARD LVCMOS33 } [get_ports {
Color[3] }]; #IO_L24P_T3_35 Sch=sw[12]
#set_property -dict { PACKAGE_PIN U12   IOSTANDARD LVCMOS33 } [get_ports {
Color }]; #IO_L20P_T3_A08_D24_14 Sch=sw[13]
#set_property -dict { PACKAGE_PIN U11   IOSTANDARD LVCMOS33 } [get_ports {
rst }]; #IO_L19N_T3_A09_D25_VREF_14 Sch=sw[14]
set_property -dict { PACKAGE_PIN V10   IOSTANDARD LVCMOS33 } [get_ports {
DRAW }]; #IO_L21P_T3_DQS_14 Sch=sw[15]


##7 segment display

set_property -dict { PACKAGE_PIN T10   IOSTANDARD LVCMOS33 } [get_ports {
seg[0] }]; #IO_L24N_T3_A00_D16_14 Sch=ca
set_property -dict { PACKAGE_PIN R10   IOSTANDARD LVCMOS33 } [get_ports {
seg[1] }]; #IO_25_14 Sch=cb
set_property -dict { PACKAGE_PIN K16   IOSTANDARD LVCMOS33 } [get_ports {
seg[2] }]; #IO_25_15 Sch=cc
set_property -dict { PACKAGE_PIN K13   IOSTANDARD LVCMOS33 } [get_ports {
seg[3] }]; #IO_L17P_T2_A26_15 Sch=cd
set_property -dict { PACKAGE_PIN P15   IOSTANDARD LVCMOS33 } [get_ports {
seg[4] }]; #IO_L13P_T2_MRCC_14 Sch=ce
set_property -dict { PACKAGE_PIN T11   IOSTANDARD LVCMOS33 } [get_ports {
seg[5] }]; #IO_L19P_T3_A10_D26_14 Sch=cf
set_property -dict { PACKAGE_PIN L18   IOSTANDARD LVCMOS33 } [get_ports {
seg[6] }]; #IO_L4P_T0_D04_14 Sch=cg

#set_property -dict { PACKAGE_PIN H15   IOSTANDARD LVCMOS33 } [get_ports {
dp }]; #IO_L19N_T3_A21_VREF_15 Sch=dp

set_property -dict { PACKAGE_PIN J17   IOSTANDARD LVCMOS33 } [get_ports {
an[0] }]; #IO_L23P_T3_FOE_B_15 Sch=an[0]
set_property -dict { PACKAGE_PIN J18   IOSTANDARD LVCMOS33 } [get_ports {
an[1] }]; #IO_L23N_T3_FWE_B_15 Sch=an[1]
set_property -dict { PACKAGE_PIN T9    IOSTANDARD LVCMOS33 } [get_ports {
an[2] }]; #IO_L24P_T3_A01_D17_14 Sch=an[2]
set_property -dict { PACKAGE_PIN J14   IOSTANDARD LVCMOS33 } [get_ports {
an[3] }]; #IO_L19P_T3_A22_15 Sch=an[3]
set_property -dict { PACKAGE_PIN P14   IOSTANDARD LVCMOS33 } [get_ports {
an[4] }]; #IO_L8N_T1_D12_14 Sch=an[4]
set_property -dict { PACKAGE_PIN T14   IOSTANDARD LVCMOS33 } [get_ports {
an[5] }]; #IO_L14P_T2_SRCC_14 Sch=an[5]
set_property -dict { PACKAGE_PIN K2    IOSTANDARD LVCMOS33 } [get_ports {
an[6] }]; #IO_L23P_T3_35 Sch=an[6]
set_property -dict { PACKAGE_PIN U13   IOSTANDARD LVCMOS33 } [get_ports {
an[7] }]; #IO_L23N_T3_A02_D18_14 Sch=an[7]


##Buttons
```

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

##VGA Connector

set_property -dict { PACKAGE_PIN A3    IOSTANDARD LVCMOS33 } [get_ports {
Colorout[0] }]; #IO_L8N_T1_AD14N_35 Sch=vga_r[0]
set_property -dict { PACKAGE_PIN B4    IOSTANDARD LVCMOS33 } [get_ports {
Colorout[1] }]; #IO_L7N_T1_AD6N_35 Sch=vga_r[1]
set_property -dict { PACKAGE_PIN C5    IOSTANDARD LVCMOS33 } [get_ports {
Colorout[2] }]; #IO_L1N_T0_AD4N_35 Sch=vga_r[2]
set_property -dict { PACKAGE_PIN A4    IOSTANDARD LVCMOS33 } [get_ports {
Colorout[3] }]; #IO_L8P_T1_AD14P_35 Sch=vga_r[3]

set_property -dict { PACKAGE_PIN C6    IOSTANDARD LVCMOS33 } [get_ports {
Colorout[4] }]; #IO_L1P_T0_AD4P_35 Sch=vga_g[0]
set_property -dict { PACKAGE_PIN A5    IOSTANDARD LVCMOS33 } [get_ports {
Colorout[5] }]; #IO_L3N_T0_DQS_AD5N_35 Sch=vga_g[1]
set_property -dict { PACKAGE_PIN B6    IOSTANDARD LVCMOS33 } [get_ports {
Colorout[6] }]; #IO_L2N_T0_AD12N_35 Sch=vga_g[2]
set_property -dict { PACKAGE_PIN A6    IOSTANDARD LVCMOS33 } [get_ports {
Colorout[7] }]; #IO_L3P_T0_DQS_AD5P_35 Sch=vga_g[3]

set_property -dict { PACKAGE_PIN B7    IOSTANDARD LVCMOS33 } [get_ports {
Colorout[8] }]; #IO_L2P_T0_AD12P_35 Sch=vga_b[0]
set_property -dict { PACKAGE_PIN C7    IOSTANDARD LVCMOS33 } [get_ports {
Colorout[9] }]; #IO_L4N_T0_35 Sch=vga_b[1]
set_property -dict { PACKAGE_PIN D7    IOSTANDARD LVCMOS33 } [get_ports {
Colorout[10] }]; #IO_L6N_T0_VREF_35 Sch=vga_b[2]
set_property -dict { PACKAGE_PIN D8    IOSTANDARD LVCMOS33 } [get_ports {
Colorout[11] }]; #IO_L4P_T0_35 Sch=vga_b[3]

set_property -dict { PACKAGE_PIN B11   IOSTANDARD LVCMOS33 } [get_ports {
O_hs }]; #IO_L4P_T0_15 Sch=vga_hs
set_property -dict { PACKAGE_PIN B12   IOSTANDARD LVCMOS33 } [get_ports {
O_vs }]; #IO_L3N_T0_DQS_AD1N_15 Sch=vga_vs
```

6. Sim_tb

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
//////
// Company:
// Engineer:
//
// Create Date: 2019/04/14 20:51:32
// Design Name:
// Module Name: PCUsim
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
//////


module PCUsim;

    reg clk;
    reg rst;
    reg [`ColorBUS] Color;
    reg [  `DirBUS] Dir;
    reg DRAW;
    wire [`PenBUS] Storex;
    wire [`PenBUS] Storey;
    wire [`ColorBUS] Storecol;
    wire we;
    wire [`PenBUS] Penx;
    wire [`PenBUS] Peny;
    wire [`PenBUS] Xcor;
    wire [`PenBUS] Ycor;
    wire [`DirBUS] Dirdelay;
    wire [31:0] cu;
    wire [31:0] cl;
    wire [31:0] cr;
    wire [31:0] cd;
    wire Xflag;

    wire [`ColorBUS] Colorout;
    wire  O_hs;
    wire  O_vs;

    initial clk<=0;
```

```
    always #1 clk<=~clk;

    initial
    begin
      rst = 1; Dir = 0;
      #10 rst = 0;
      #10 rst = 1;
      #10 rst = 0;
      #20 DRAW = 1;
      #20 Color = 12'b1111_0000_0000;
      #20 Dir = 4'b_1000;
      #20 Dir = 4'b_0000;
      #10 DRAW = 0;
      #20 Dir = 4'b_0001;
      #20 Dir = 4'b_0000;
    end
    PCU DUT
(clk,rst,Color,Dir,DRAW,/*Storex,Storey,Storecol,we,*/Penx,Peny,/*Dirdelay,
cu,cl,cr,cd,Xcor,Ycor,Xflag*/Colorout,O_hs,O_vs);
endmodule
```

去掉注释，可以得到更多的模拟信号信息。

总结

本次实验进展十分顺利，因为上个学期做了基于VGA的膜蛤游戏，所以写起来还算熟练。

但是这次完全怪自己没有常识使用状态机，如果使用状态机，那么代码可以大大简化，行数也可以大大缩减。这次所有实验只有写报告部分再Linux下进行，其余状态均在Vivado下进行。