VGA 实验报告

517021911052 周佳懿

一、 实验目的

1. 利用 Verilog 语言的描述方法进行设计完成 VGA 显示的系统设计
2. 自行选择 VGA 显示模式
3. 通过 FPGA 产生彩条图形，在 VGA 显示器上显示
4. 通过 Verilog 语言完成一个贪吃蛇游戏，并且使用 VGA 显示
5. 感受并行编程编写游戏和面向对象编程的区别

二、 设计原理

1、 VGA 简要介绍

显示绘图阵列（video graphic array，VGA）接口是 LCD 液晶显示设备的标准接口，大多应用在显示器与显卡之间，同时还可以用在等离子电视输入图像的模数转换上。VGA 显示输出 RGB 三原色信号，RGB 色彩模式是工业界的一种颜色标准，是通过对红(R)、绿(G)、蓝(B)三个颜色通道的变化以及它们相互之间的叠加来得到各式各样的颜色，目前在图像显示领域中应用非常广泛。

2、 VGA 的显示特点

1) 扫描格式繁多，分辨率从 320×200 一直延伸到 1280×102，行频 15.8~70Hz，场频 50~100Hz。常见的行频有 31.4Hz，37.8Hz，57.9Hz，62.5Hz 等，常见场频有 50Hz，60Hz，70Hz，100Hz，16700K 之分。
2) VGA 接口为显示器提供两类信号，一类是数据信号，一类是控制信号。数据信号包括红（Red）、绿（Green）、蓝（Blue）信号，简称 RGB 信号，控制信号包括水平同步信号和垂直同步信号。输出不同分辨率时，水平同步信号和垂直同步信号的频率也不相同。

3、 VGA 显示原理

常见的彩色显示器一般由 CRT(阴极射线管)构成，彩色是由 R(红)、G(绿)、B(蓝)三种基色组成。显示是采用逐行扫描的方式，阴极射线枪发出的电子束打在涂有荧光粉的荧光屏上，产生 RGB 三色基，最后合成一个彩色图像。从荧幕的左上方开始自左向右扫描，每扫完一行图像电子束回到下一行的最左端，每行结束后电子枪回扫的过程中进行消隐。然后重新开始行扫描，消隐，直到扫到荧幕的右下方，电子束回到荧幕的左上方重新开始新的图像扫描，并且在回到荧幕左上方的过程中进行消隐。在消隐过程中不发射电子束。每一行扫描结束时，用 HS(行同步)信号进行同步；扫描完所有的行后用 VS (场同步)信号进行同步。它的行场扫描时序示意图如图 1 所示。

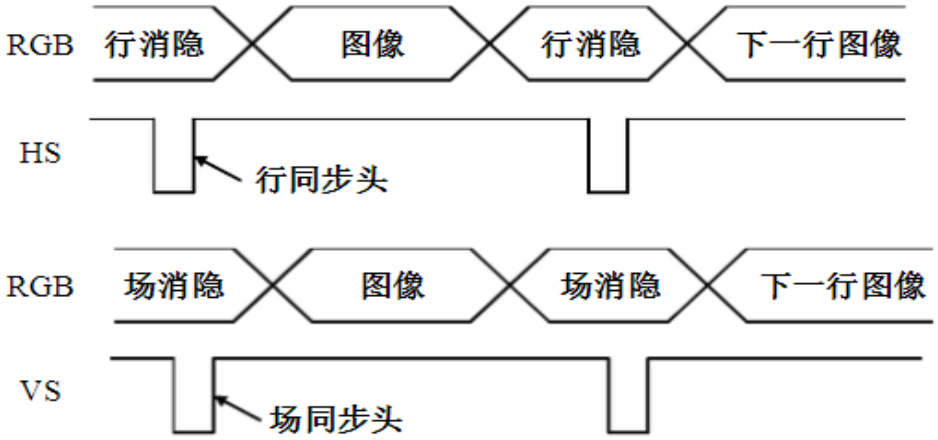


图 1　行场扫描时序图

4、时序分析

通过对 VGA 显示基本工作原理的分析可知，要实现 VGA 显示就要解决数据来源、数据存储、时序实现等问题，其中关键还是如何实现 VGA 时序。基于像素时钟，VGA 时序控制器必须产生 HS 和 VS 时序信号。像素时钟定义了用于显示一个像素信息的时间，VS 信号定义了显示的刷新频率，通常刷新频率在 50Hz 到 120Hz 之间。给定刷新频率后即定义了水平扫描频率即 HS。

VGA 的标准参考显示时序如图 2 所示：

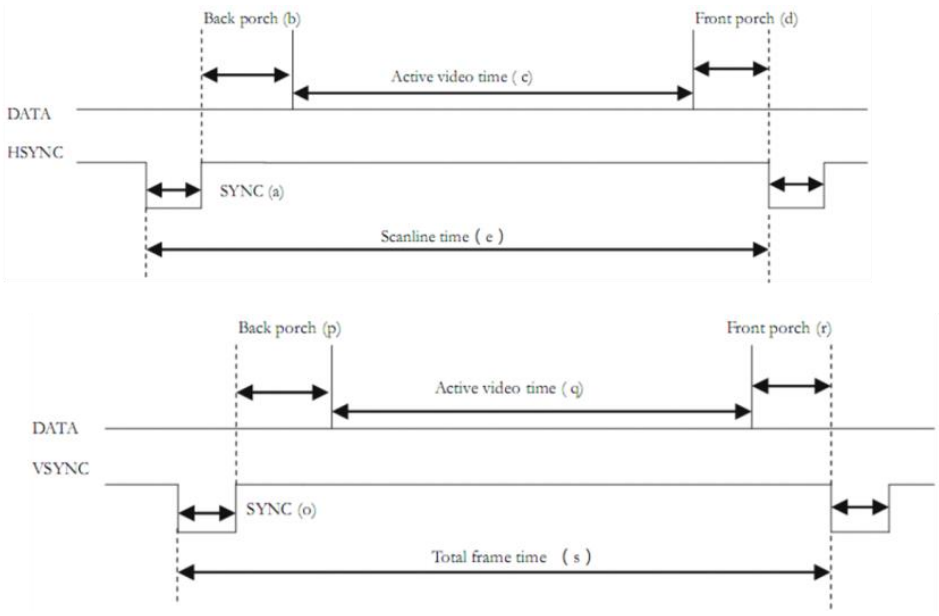


图 2　VGA 时序图

行时序和帧时序都需要产生同步脉冲(Sync)、显示后沿(Back porch)、显示时序段(Display interval)和显示前沿(Front porch)四个部分。其中场频定义了显示的刷新频率，指定场频后所要扫描的行数指定了水平回扫频率即行频。

几种常用模式的时序参数如表 1 所示：

表 1　VGA 时序参考表

常见刷新率时序表

显示模式	时钟（MHz）	水平时序（像素数）					垂直时序（行数）				
		a	b	c	d	e	o	p	q	r	s
640×480@60	25.175	96	48	640	16	800	2	33	480	10	525
640×480@75	31.5	64	120	640	16	840	3	16	480	1	500
800×600@60	40.0	128	88	800	40	1056	4	23	600	1	628
800×600@75	49.5	80	160	800	16	1056	3	21	600	1	625
1024×768@60	65	136	160	1024	24	1344	6	29	768	3	806
1024×758@75	78.8	176	176	1024	16	1312	3	28	768	3	806
1280×1024@60	108.0	112	248	1280	48	1688	3	38	1024	1	1066
1280×800@60	83.46	136	200	1280	64	1680	3	24	800	1	828
1440×900@60	106.47	152	232	1440	80	1904	3	28	900	1	932

根据 VGA 的频率图在此选择时钟为 40Hz，水平时序则对应着 H_SYNC 的变化，垂直时序则对应 V_SYNC 的变化。

三、 贪吃蛇功能描述

1) 玩家可以操控蛇上下左右移动

2) 蛇和食物触碰就表示蛇吃掉了食物

3) 食物被吃掉后会在另外的位置重新生成

4) 蛇吃了一个食物后身体会变长一节

5) 记分板（LED 数码管）会显示吃了多少个食物

四、 主要模块设计思想

1. 输出信号的设计

对于普通的 VGA 显示器，其引出线共含五个信号： ●R、G、B：三基色信号。●HS：行同步信号。 ●VS：场同步信号。VGA 工业标准要求的频率： 产生时钟频率 25.175 MHz（像素输出的频率） 行频 31469 Hz 场频 59.94 Hz（每秒图像刷新频率）

所以我们 simple_vga_game 模块输出信号就是 VGA_R, VGA_G, VGA_B（三基色信号），以及 HS（行同步信号），VS（场同步信号），VGA_CLK（像素输出的频率）。

hex1 ， hex0 是 2 个 LED 数码管的控制信号，用于显示蛇吃掉多少个食物，是游戏的计分板功能。

2. 输入信号的设计

reset 信号。对应板子上的第四个按键，按键复位，可以对 VGA 的显示进行重置。
clk 信号：50mhz 板载时钟。
left 信号，对应板子上的第一个按键，用于控制蛇的移动方向。

3. 显示不同部件的三原色参数

1. 蛇的 RGB 三原色信号

```
VGA_R = 8'd210;
VGA_G = 8'd80;
VGA_B = 8'd80;
```

2. 背景的 RGB 三原色信号

```
VGA_R = 8'd135;
VGA_G = 8'd206;
VGA_B = 8'd250;
```

3. 食物的 RGB 三原色信号

```
VGA_R = 8'd221;
VGA_G = 8'd169;
VGA_B = 8'd105;
```

4. Vga_display 模块

1) 频率的选择

根据 VGA 的频率图在此选择时钟为 25MHz，显示模式是 640*480。
时钟频率的计算：以 640x480@59.94Hz(60Hz)为例，每场对应 525 个行周期(525=10+2+480+33),其中 480 为显示行。每场有场同步信号,该脉冲宽度为 2 个行周期的负脉冲，每显示行包括 800 点时钟,其中 640 点为有效显示区,每一行有一个行同步信号，该脉冲宽度为 96 个点时钟。由此可知：行频为 525*59.94=31469Hz,需要点时钟频率：525*800*59.94 约 25MHz。

2) 扫描过程的

接收 clk, reset 作为输入信号，
输出 Hcnt、Vcnt、HS、VS、VGA_BLANK_N、VGA_CLK 这几个信号。
正向扫描过程由 Hcnt = Hcnt + 1;实现
当一行扫描完毕，行同步 Vcnt = Vcnt + 1;同时将 Hcnt 置零。Hcnt = 0;
当扫描完 480 行后，产生场同步是扫描线回到 CRT 的第一行第一列，也就是 Vcnt = 0; Hcnt = 0;

5. 游戏实现

1. 显示不同组件时候的 if 语句的先后顺序

对于蛇的头、蛇的身体、食物和背景，需要判定该像素（由 Hcnt、Vcnt 变量表示像素坐标）是在哪个部件的位置上，然后对 VGA_R、VGA_G、VGA_B 赋相应的值。
由于蛇的身体的坐标存储在寄存器数组中，需要用 for 循环遍历，这就导致在一个 if 条件中没有办法直接判定该像素点是否是蛇身体的坐标，但是我们又需要让蛇的头、蛇的身体、食物以外的所有像素点绘制成背景的颜色。
那么我们写如下控制流来保证蛇身体和背景能一起正常显示。
以下是为了说明控制流逻辑的伪代码

```
if (像素坐标是蛇的头的坐标)
        绘制蛇的头
else (像素坐标是食物的坐标)
        绘制食物
else
{
    先把像素赋值成背景的颜色信号
    循环遍历蛇身体的数组，如果该像素坐标是蛇的身体，那么就把像素赋值成蛇身体的颜色信号
}
```

2. 蛇的移动的功能的实现

为了使蛇移动的像素数合理，选择 5hz 的信号作为重新计算蛇的坐标的信号。这样既不会移动得太快，画面连贯性也比较好。

显示蛇移动的主要思想就是：

1) 5hz 的信号一个周期，把蛇的头的坐标赋给蛇身体数组[0]
2) 利用 for 循环，从蛇身体数组[1]开始直到蛇身体的长度的最后一个数组元素，把它前一个数组的值赋给它。
3) 更新蛇的头的坐标的值

更新蛇的头的坐标的值的主要思想就是：

1) 5hz 的信号一个周期，由 move_direction 得到移动的方向，然后得到该方向移动了的值。
2) 对各个方向的坐标值进行更新。
如下

```
head_left <= head_left_init - left_total+ right_total;
head_right <= head_right_init - left_total + right_total;
head_top <= head_top_init + up_total - down_total ;
head_bottom <= head_bottom_init + up_total - down_total ;
```

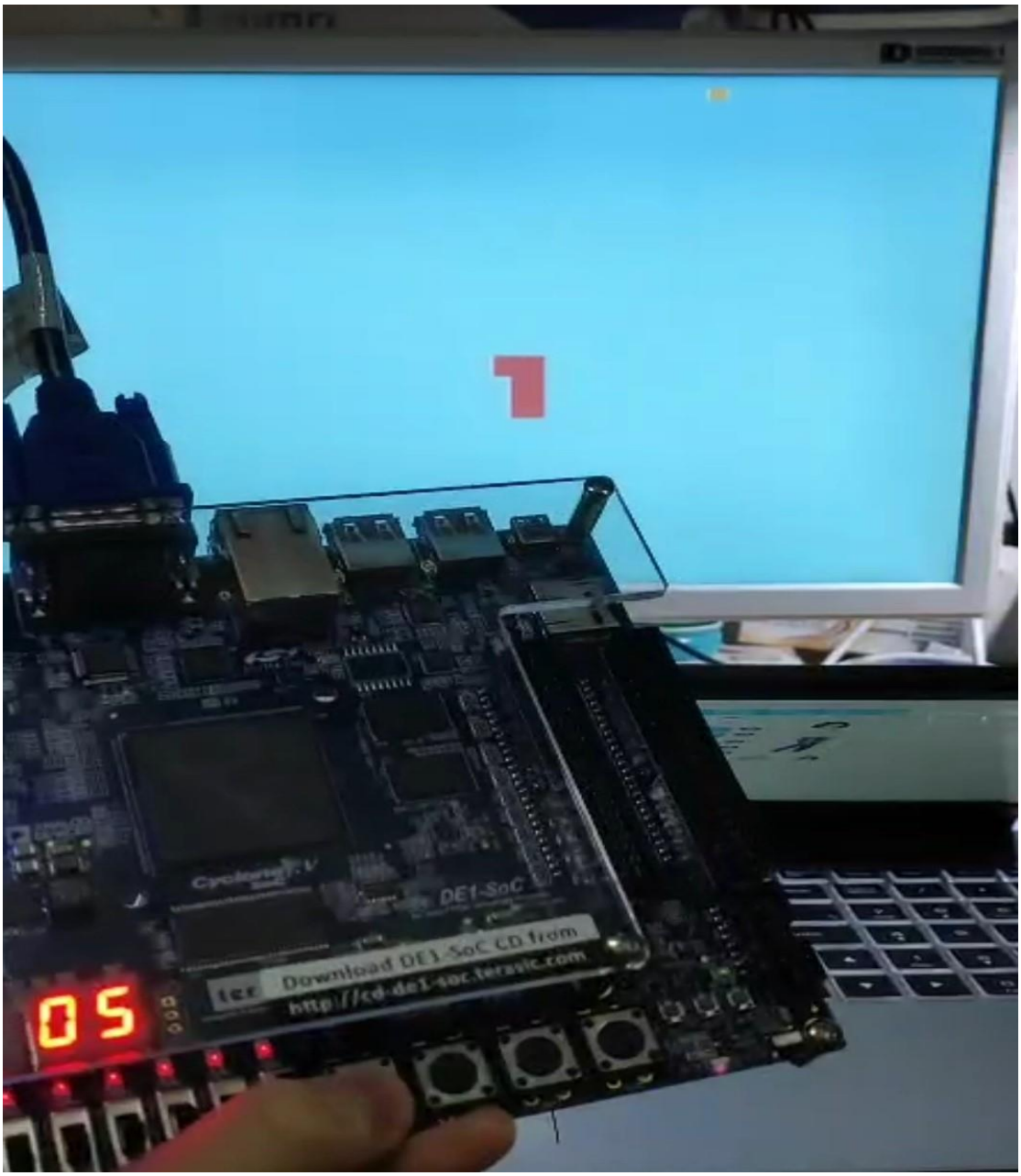
3. 吃食物后身体变长功能的实现

5hz 的信号一个周期检测蛇和食物有没有相撞，检测到蛇和食物的坐标有重叠区域的时候，就更新 body_num 的值。
如下
body_num = body_num+1;
然后在下一个周期（也是 5hz 的时钟）的时候，因为 body_num 的值加一，就会在 for 循环的时候多循环一次，多给蛇的身体数组赋值一次，也就在 clk_25M 的时钟周期上升沿时，重新绘制身体的时候，多画一格身体。

4. 食物被吃掉后会在另外的位置重新生成

5hz 的信号一个周期检测蛇和食物有没有相撞，检测到蛇和食物的坐标有重叠区域的时候，就改变 food_left_init、food_right_init、food_top_init、food_bottom_init 的值。
然后在下一个计算食物坐标的周期（也是 5hz 的时钟）的时候，就会给 food_left、food_right、food_top、food_bottom 赋值为新的值，从而在 clk_25M 的时钟周期上升沿时，重新绘制食物。

五、 实验效果



六、 代码

```
module simple_vga_game (
    input reset,
    input left,
    input clk,
    output HS,
    output VS,
    output reg[7:0] VGA_R,
    output reg[7:0] VGA_G,
    output reg[7:0] VGA_B,
    output VGA_BLANK_N,
    output VGA_CLK,
```

```
    output [6:0] hex0,
    output [6:0] hex1,
    output [6:0] hex4,
    output [6:0] hex5
);
    //Hcnt, Vcnt for vga
    wire[9:0] Hcnt;
    wire[9:0] Vcnt;

    //head(left, top, right, bottom)
    reg[9:0]  head_left_init,  head_right_init,  head_top_init,  head_bottom_init;

    reg[9:0] head_left, head_right, head_top, head_bottom;

    reg[9:0] body_left[20], body_right[20], body_top[20], body_bottom[20];

    reg[9:0] food_left,food_right,food_top,food_bottom;
    reg[9:0] food_left_init, food_right_init, food_top_init, food_bottom_init;

    initial
    begin
        //head
        head_left_init <= 10'd260;
        head_right_init <= 10'd280;
        head_top_init <= 10'd230;
        head_bottom_init <= 10'd250;


        //food
        food_left_init <= 10'd220;
        food_right_init <= 10'd230;
        food_top_init <= 10'd90;
        food_bottom_init <= 10'd100;
    end

    reg clk_25M;
    //generate a half frequency clock of 25MHz
    always@(posedge(clk))
    begin
        clk_25M <= ~clk_25M;
    end

    //generate 200ms clock
    reg[31:0] counter_200ms;
```

```
reg clk_200ms;
parameter COUNT_200ms = 4999999;
always@(posedge(clk))
begin
    if (counter_200ms == COUNT_200ms)
    begin
        counter_200ms = 0;
        clk_200ms = ~clk_200ms;
    end
    else
    begin
        counter_200ms = counter_200ms + 1;
    end
end

//generate 500ms clock
reg[31:0] counter_500ms;
reg clk_500ms;
parameter COUNT_500ms = 12999999;
always@(posedge(clk))
begin
    if (counter_500ms == COUNT_500ms)
    begin
        counter_500ms = 0;
        clk_500ms = ~clk_500ms;
    end
    else
    begin
        counter_500ms = counter_500ms + 1;
    end
end

reg[2:0] move_direction;

//move left
parameter left_distance = 10;
reg[9:0] left_total;
parameter right_distance = 10;
reg[9:0] right_total;
parameter down_distance = 10;
reg[9:0] down_total;
parameter up_distance = 10;
reg[9:0] up_total;
```

```
always@(posedge(left) )
begin
        move_direction    = move_direction +1;
        if(move_direction == 4)
            move_direction    = 0;
end


always@(posedge(clk_200ms))
   begin
        if(move_direction == 0)
            left_total = left_total + left_distance ;
        if(move_direction == 1)
            right_total = right_total + right_distance ;
        if(move_direction == 2)
            up_total = up_total + up_distance    ;
        if(move_direction == 3)
            down_total = down_total + down_distance    ;
     end

//refresh head pos

reg [8:0] temp1;   //for 循环变量
always@(posedge(clk_200ms))
        begin
         //body
        for( temp1= 0 ; temp1 + 1< body_num ; temp1=temp1+1'b1 )
                //循环次数固定
            begin
                body_left[temp1+1'b1] <= body_left[temp1];
                body_right[temp1+1'b1] <= body_right[temp1] ;
                body_bottom[temp1+1'b1] <= body_bottom[temp1];
                body_top[temp1+1'b1] <= body_top[temp1] ;
            end

        if(body_num >=1)
            begin
                body_left[0]    <=    head_left;
                body_right[0] <=    head_right;
                body_top[0]    <=    head_top;
                body_bottom[0]    <=    head_bottom;
            end
```

```
                //head
                head_left <= head_left_init - left_total+ right_total;
                head_right <= head_right_init - left_total + right_total;
                head_top <= head_top_init + up_total - down_total    ;
                head_bottom <= head_bottom_init    + up_total - down_total ;


                //food
                food_left <= food_left_init ;
                food_right <= food_right_init ;
                food_top <= food_top_init;
                food_bottom <= food_bottom_init;



                end

    reg[31:0] score;
    reg [3:0] body_num;

    //detect collision
    always@(posedge(clk_200ms))
                begin
                if(head_left <= food_left && head_top <= food_top    &&
head_right>=food_right && head_bottom >= food_bottom)
                    begin
                        score = score + 1;

                    //food relocation
                        food_left_init <= (food_left_init + 10'd180) % 10'd640;
                        food_right_init <= (food_right_init + 10'd180) % 10'd640;
                        food_top_init <= (food_top_init + 10'd80) % 10'd480;
                        food_bottom_init  <=  (food_bottom_init  +  10'd80)  %
10'd480;

                        //add body
                        body_num = body_num+1;
                    end
                end

    vga_display screen(
    .clk(clk_25M),//50MHZ
    .reset(reset),
    .Hcnt(Hcnt),
    .Vcnt(Vcnt),
```

```
.hs(HS),
.vs(VS),
.blank(VGA_BLANK_N),
.vga_clk(VGA_CLK)
);

out_port_seg historyScoreboard(
.in(history_score),
.out1(hex5),
.out0(hex4)
);

out_port_seg scoreboard(
.in(score),
.out1(hex1),
.out0(hex0)
);

//game over
integer k;
reg finish;


reg [8:0] temp2;   //for 循环变量
always@(posedge clk_25M)
begin
    if (finish == 0)
    begin
        //head
        if (Hcnt >= head_left && Hcnt < head_right
            && Vcnt >= head_top && Vcnt < head_bottom)
        begin
            VGA_R = 8'd210;
            VGA_G = 8'd80;
            VGA_B = 8'd80;
        end
        //food
        else if (Hcnt >= food_left && Hcnt < food_right
            && Vcnt >= food_top && Vcnt < food_bottom)
            begin
                VGA_R = 8'd221;
                VGA_G = 8'd169;
                VGA_B = 8'd105;
            end
```

```
                else
                    begin

                        //sky
                            VGA_R = 8'd135;
                            VGA_G = 8'd206;
                            VGA_B = 8'd250;

                        //body
                            if(body_num >=1)
                                begin
                                    for( temp2 = 0 ; temp2< body_num ;
temp2=temp2+1'b1 )
                                        begin
                                            if( Hcnt >= body_left[temp2] &&
Hcnt <   body_right[temp2]
                                                && Vcnt >= body_top[temp2]
&& Vcnt <   body_bottom[temp2])
                                                    begin
                                                    VGA_R = 8'd210;
                                                    VGA_G = 8'd80;
                                                    VGA_B = 8'd80;
                                                    end
                                        end
                                end
                    end
            end
            //game over
            else
            begin
                VGA_R = 8'd255;
                VGA_G = 8'd128;
                VGA_B = 8'd128;
            end
        end

endmodule


module vga_display(
        input clk,
        input reset,
        output reg[9:0] Hcnt,
        output reg[9:0] Vcnt,
```

```
    output hs,
    output vs,
    output blank,
    output vga_clk
);
    assign vga_clk = clk;
    assign vs = ~(Vcnt >= visible_area_v + front_porch_v && Vcnt < visible_area_v + front_porch_v + sync_pulse_v);
    assign hs = ~(Hcnt >= visible_area_h + front_porch_h && Hcnt < visible_area_h + front_porch_h + sync_pulse_h);
    assign blank = Vcnt < visible_area_v && Hcnt < visible_area_h;

     //parameter definition
    parameter integer visible_area_h = 640;
    parameter integer front_porch_h = 16;
    parameter integer sync_pulse_h = 96;
    parameter integer back_porch_h = 48;
    parameter integer whole_line_h = 800;

    parameter integer visible_area_v = 480;
    parameter integer front_porch_v = 10;
    parameter integer sync_pulse_v = 2;
    parameter integer back_porch_v = 33;
    parameter integer whole_line_v = 525;

    /*generate the hs && vs timing*/
    always@(posedge(clk))
    begin
        if (!reset)
        begin
            Hcnt = 0;
            Vcnt = 0;
        end
        /*conditions of reseting Hcnter && Vcnter*/
        else
        begin
            Hcnt = Hcnt + 1;
            if( Hcnt == whole_line_h)
            begin
                Hcnt = 0;
                Vcnt = Vcnt + 1;
                if( Vcnt == whole_line_v)
                    Vcnt = 0;
            end
```

```
            end
        end
endmodule

module out_port_seg(in,out1,out0);
        input [31:0] in;
        output [6:0] out1,out0;

        reg [3:0] num5,num4,num3,num2,num1,num0;

        sevenseg display_1( num1, out1 );
        sevenseg display_0( num0, out0 );

        always @ (in)
        begin
            num1 = ( in / 10 ) % 10;
            num0 = in % 10;
        end

endmodule
```