**同济大学计算机系**

**数字逻辑课程综合实验报告**

****

**学 号 1652228**

**姓 名 王哲源**

**专 业 计算机**

**授课老师 张冬冬**

1. 实验内容

基于PS2、VGA、七段数码管及板子上的五个按键的倒计时打字游戏

下板之后，按下rst键，即板上五个键的中键可以重载游戏。当VGA屏上显示蓝灯时，表示游戏处于设置状态，此时可以按下上键对游戏时间进行增加，或者下键对游戏时间进行减少，设置的时间会在数码管上进行显示。

按下开始键，即左键，游戏开始，此使VGA上显示绿灯。需要通过键盘进行打字，有大小写区分，默认为小写，按下shift后再进行输入则为大写。绿色为已打部分，红色则为待打部分。同时数码管上会显示残余时间，当耗尽时，VGA上显示红灯，倒计时暂停，输入无效；当先于倒计时打完，则显示绿灯，倒计时暂停，显示白灯，输入无效。需要再按下rst才能重新开始。同时，在倒计时进行时按下暂停键，即右键，倒计时将会暂停，同时显示黄灯，需要按下开始键才会继续倒计时，或者按下rst键重新开始一局

1. 打字游戏数字系统总框图

**控制器**

VGA输出

VGA控制器

VGA子系统

PS2键盘编码

PS2控制器

输入子系统

数码管输出

数码管控制器

数码管子系统

时间计数器

计时子系统

寄存器组

文本控制器

储存子系统

输入子系统：读取键盘输入信号，并通过读取到的键值传递给控制器

储存子系统：储存文本信息

VGA子系统：根据储存子系统中的文本信息在屏幕上输出对应信息

计时子系统：进行倒计时残余时间计算

数码管子系统：根据剩余时间显示在七段数码管上

控制器：根据输入得到的信息对比寄存器中的文本信息，以及计时信息判断整个系统是否继续工作

1. 系统控制器设计

1

1

0

0

1

0

0

1

0

0

1

START

RST

TIME\_UP

TIME\_DOWN

TIME\_GO

GAME\_START

TIME\_OUT

TIME\_PAUSE

GAME\_PAUSE

TIME\_GO

CLOCK\_UP

CLOCK\_UP

RST

GAME\_WIN

WIN

LOSE

RST

RST

1

1

0

0

1

0

1

0

0

1

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| PS | | | | | | | | | | NS | | |
| A | B | C | RST | T\_U | T\_D | T\_G | T\_P | G\_W | G\_L | A | B | C |
| 0 | 0 | 0 | 1 | x | x | x | x | x | x | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 1 | x | x | x | x | x | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 1 | x | x | x | x | 0 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | x | x | x | 0 | 1 | 1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | x | x | x | 0 | 0 | 0 |
| 0 | 0 | 1 | x | x | x | x | x | x | x | 0 | 0 | 0 |
| 0 | 1 | 0 | x | x | x | x | x | x | x | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | x | x | x | x | x | x | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | x | x | x | x | 1 | x | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 | x | x | x | x | 0 | 1 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | x | x | x | 1 | 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 | x | x | x | 0 | 0 | 0 | 0 | 1 | 1 |
| 1 | 0 | 0 | 1 | x | x | x | x | x | x | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | x | x | 1 | x | x | x | 0 | 1 | 1 |
| 1 | 0 | 0 | 0 | x | x | 0 | x | x | x | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | x | x | x | x | x | x | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | x | x | x | x | x | x | 1 | 0 | 1 |
| 1 | 1 | 0 | 1 | x | x | x | x | x | x | 0 | 0 | 0 |
| 1 | 1 | 0 | 0 | x | x | x | x | x | x | 1 | 1 | 0 |

A(D) = nRST(nABCT\_P + nABCG\_W + nABCG\_L +AnBnC + AnBnCnT\_G + AnBC +ABnC)

B(D) = nRST(nAnBnCnT\_UT\_D + nAnBnCnT\_UnT\_DT\_G + nABCnG\_W + nABCnT\_PnG\_WnG\_L + AnBnCT\_G + ABC)

C(D) =nRST(nAnBnCT\_U + nAnBnCnT\_UnT\_DT\_G + nABCG\_W + nABCnT\_PnG\_WnG\_L + AnBnCT\_G + AnBC)

START = nAnBnC 游戏起始状态

CLOCK\_UP = nAnBC 设定时间增加

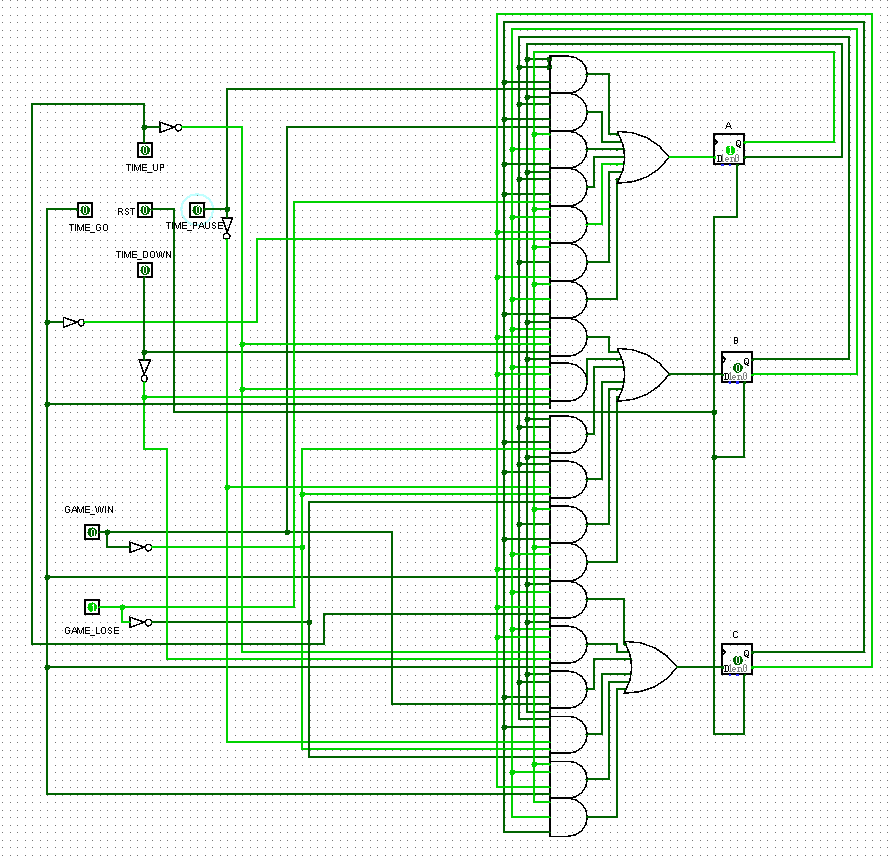
CLKCK\_DOWN =nABnC 设定时间减少

GAME\_START =nABC 游戏开始

GAME\_PAUSE =AnBnC 游戏暂停

WIN =AnBC 游戏胜利

LOSE =ABnC 游戏失败



1. 子系统模块建模

输入子系统：

PS2键盘编码

PS2控制器

clk、rst、

clk\_in、data\_in

alph

module PS2\_driver(

//--------系统信号--------

input clk,

input rst,

//--------PS/2信号--------

input data\_in,

input clk\_in,

output reg [5:0] alph

);

wire [15:0] ps2\_data;

reg [4:0] press\_shift;

initial

begin

alph <= 'd54; press\_shift <= 5'b00000;

end

PS2\_control PS2(.clk(clk), .rst(rst),

.data\_in(data\_in), .clk\_in(clk\_in),

.ps2\_data(ps2\_data));

always @(ps2\_data or rst)

begin

if(rst == 0)

begin

alph <= 'd54; press\_shift <= 5'b00000;

end

else

begin

case(ps2\_data[7:0])

8'h12: // l\_shift

begin

press\_shift <= 'b11111; alph <= 'd55;

end

8'h1c: alph <= ('d26 & ~press\_shift) + 'd1; //A

8'h32: alph <= ('d26 & ~press\_shift) + 'd2; //B

8'h21: alph <= ('d26 & ~press\_shift) + 'd3; //C

8'h23: alph <= ('d26 & ~press\_shift) + 'd4; //D

8'h24: alph <= ('d26 & ~press\_shift) + 'd5; //E

8'h2b: alph <= ('d26 & ~press\_shift) + 'd6; //F

8'h34: alph <= ('d26 & ~press\_shift) + 'd7; //G

8'h33: alph <= ('d26 & ~press\_shift) + 'd8; //H

8'h43: alph <= ('d26 & ~press\_shift) + 'd9; //I

8'h3b: alph <= ('d26 & ~press\_shift) + 'd10; //J

8'h42: alph <= ('d26 & ~press\_shift) + 'd11; //K

8'h4b: alph <= ('d26 & ~press\_shift) + 'd12; //L

8'h3a: alph <= ('d26 & ~press\_shift) + 'd13; //M

8'h31: alph <= ('d26 & ~press\_shift) + 'd14; //N

8'h44: alph <= ('d26 & ~press\_shift) + 'd15; //O

8'h4d: alph <= ('d26 & ~press\_shift) + 'd16; //P

8'h15: alph <= ('d26 & ~press\_shift) + 'd17; //Q

8'h2d: alph <= ('d26 & ~press\_shift) + 'd18; //R

8'h1b: alph <= ('d26 & ~press\_shift) + 'd19; //S

8'h2c: alph <= ('d26 & ~press\_shift) + 'd20; //T

8'h3c: alph <= ('d26 & ~press\_shift) + 'd21; //U

8'h2a: alph <= ('d26 & ~press\_shift) + 'd22; //V

8'h1d: alph <= ('d26 & ~press\_shift) + 'd23; //W

8'h22: alph <= ('d26 & ~press\_shift) + 'd24; //X

8'h35: alph <= ('d26 & ~press\_shift) + 'd25; //Y

8'h1a: alph <= ('d26 & ~press\_shift) + 'd26; //Z

8'hF0:

begin

if(ps2\_data[15:8] == 8'h12)

press\_shift <= 5'b0000;

alph <= 'd53; //F0

end

default: ; //others

endcase

end

end

endmodule

module PS2\_control(

input clk,

input rst,

input data\_in,

input clk\_in,

output reg [15:0] ps2\_data

/\*

output oready,

output [4:0] oQ, //Test\_Port

output [9:0] obuffer

\*/

);

reg [4:0] Q; //有限状态机

reg [9:0] buffer;

reg [2:0] ps2\_clk\_sync;

reg [15:0] pre\_ps2\_data;

reg ready;

wire clk\_in\_neg;//clk\_in下降沿检测

//assign oready = ready; assign oQ = Q; assign obuffer = buffer;

initial

begin

ps2\_data <= 0; Q <= 0;

ps2\_clk\_sync <= 0; ready <= 0;

end

//--------clk\_in下降沿检测--------

always @(posedge clk or negedge rst)

begin

if(rst == 0)

ps2\_clk\_sync <= 0;

else

ps2\_clk\_sync <= {ps2\_clk\_sync[1:0], clk\_in};

end

assign clk\_in\_neg = ps2\_clk\_sync[2] & ~ps2\_clk\_sync[1];

//--------ps2键值获取--------

always @(posedge clk)

begin

if(rst == 0)

begin

buffer <= 'd0; Q <= 'd0; ps2\_data <= 8'b0;

end

else if(clk\_in\_neg == 1)

begin

case(Q)

'd0, 'd1, 'd2, 'd3, 'd4, 'd5, 'd6, 'd7, 'd8, 'd9: //读前10位

begin

buffer[Q] <= data\_in;

Q <= Q + 1;

end

'd10: //读第11位的的同时判断，否则有一个周期的延时

begin

if((buffer[0] == 0) //start bit

&& (data\_in == 1) // stop bit

&& (^buffer[9:1])) // odd parity

begin

pre\_ps2\_data[7:0] <= buffer[8:1]; //后八位清零啊！！

if(buffer[8:1] != 8'hF0)

ready <= 1;

end

if(buffer[8:1] == 8'hF0)

Q <= Q + 1;

else

Q <= 0;

end

'd11, 'd12, 'd13, 'd14, 'd15, 'd16, 'd17, 'd18, 'd19, 'd20: //断码过滤后11位

begin

buffer [Q - 'd11] <= data\_in;

Q <= Q + 1;

end

'd21:

begin

if((buffer[0] == 0) //start bit

&& (data\_in == 1) // stop bit

&& (^buffer[9:1])) // odd parity

begin

ps2\_data[15:8] <= buffer[8:1];

ready <= 1;

end

Q <= 0;

end

endcase

end

if(ready == 1)

begin

ps2\_data <= pre\_ps2\_data;

ready <= 0;

end

end

endmodule

控制子系统：

游戏控制器

alph、last\_time

、text[cur]

clock\_go、text、cur、Q

、clk、rst

`define N 200

`define M (6 \* `N - 1)

module GAME\_control(

//--------SYSTEM\_in--------

input clk,

input rst,

//--------PS/2\_in--------

input data\_in,

input clk\_in,

//--------PRESS\_in--------

input time\_up,

input time\_down,

input time\_go,

input time\_pause,

//--------VGA\_out--------

output [3:0] R,

output [3:0] G,

output [3:0] B,

output HS,

output VS,

//--------DIG\_out--------

output [7:0] DIG,

output [7:0] SEG

);

//--------PS2用参数--------

wire [5:0] alph;

reg [5:0] pre\_alph;

//--------文档用参数--------

integer i;

reg [`M:0] text;

reg [9:0] p;

//--------随机数用参数--------

reg [15:0] seed;

wire [`M:0] ranNum;

//--------游戏状态机--------

reg [2:0] Q;

//--------数码管用参数--------

reg [26:0] setting\_time;

reg clock\_go;

reg [1:0] time\_up\_press, time\_down\_press;

wire time\_up\_sync, time\_down\_sync;

wire [26:0] last\_time;

initial

begin

seed <= 'd1;

pre\_alph <= 'd0;

Q <= 'd0;

p <= 0;

setting\_time <= 'd1200; clock\_go <= 'd0;

time\_up\_press <= 'd0; time\_down\_press <= 'd0;

end

//--------伪随机序列获取--------

LFSR get\_rand(.rst(rst), .clk(clk), .seed(seed), .oNum(ranNum));

//--------PS2读取数据--------

PS2\_driver get\_key(.clk(clk), .rst(~rst), .data\_in(data\_in), .clk\_in(clk\_in),

.alph(alph));

//--------VGA输出--------

VGA\_driver output\_screen(.CLK(clk), .RST(rst), .text(text), .Q(Q), .cur(p),

.R(R), .G(G), .B(B), .HS(HS), .VS(VS));

//--------数码管倒计时显示--------

always @(posedge clk)

begin

time\_up\_press <= {time\_up\_press[0], time\_up};

time\_down\_press <= {time\_down\_press[0], time\_down};

end

assign time\_up\_sync = time\_up\_press[1] & ~time\_up\_press[0];

assign time\_down\_sync = time\_down\_press[1] & ~time\_down\_press[0];

DIG\_count\_down count\_down(.clk(clk), .rst(rst), .setting\_change(Q == 'd0),

.start\_time(setting\_time),

.clock\_go(clock\_go),

.SEG(SEG), .DIG(DIG), .last\_time(last\_time));

//--------PS2\_数码管测试--------

/\*wire [3:0] num0, num1;

assign num0 = alph % 10;

assign num1 = alph / 10;

assign last\_time = 1;

DIG\_driver DIG\_out(.clk(clk), .rst(~rst),

.num0(num0), .num1(num1), .num2(0), .num3(0),

.num4(4), .num5(5), .num6(6), .num7(7),

.DIG(DIG), .SEG(SEG));\*/

//--------状态机--------

always @(posedge clk)

begin

if(rst == 1)

begin

seed <= (text['d413 -:'d16] | 'd407);

text <= ranNum;

p <= 0; Q <= 'd0;

clock\_go <= 0; pre\_alph <= 'd0;

end

else

begin

case(Q)

'd0: // stop

begin

if(time\_up\_sync == 1 && setting\_time <= 'd99999989)

setting\_time <= setting\_time + 10;

else if(time\_down\_sync == 1 && setting\_time >= 'd10)

setting\_time <= setting\_time - 10;

else if(time\_go == 1)

begin

clock\_go <= 1; Q <= 'd1;

end

end

'd1: // start

begin

if(time\_pause == 1)

begin

clock\_go <= 0; Q <= 'd2;

end

if(last\_time == 'd0)

Q <= 'd4;

if(alph != pre\_alph && alph >= 'd1 && alph <= 'd52) // is\_alph

begin

pre\_alph <= alph;

if(alph != text[6 \* (p + 1) - 1 -:6])

;//Q <= 'd4;

else

begin

if(p == `N - 1)

Q <= 'd3;

p <= p + 'd1;

end

end

else if (alph == 'd53 || alph == 'd55) //F0 and lshift

pre\_alph <= 'd0;

end

'd2: // pause

begin

if(time\_go == 1)

begin

clock\_go <= 1; Q <= 'd1; pre\_alph <= 'd0;

end

end

'd3: // win

begin

clock\_go <= 0;

end

'd4: // lose

begin

clock\_go <= 0;

end

endcase

end

end

endmodule

VGA及储存子系统：

VGA输出

VGA控制器

clk、rst、Q

HS、VS、R、G、B

寄存器组

文本控制器

clk、rst、

text、cur

text[cur]

text

`define H\_FRONT 11'd40

`define H\_BACK 11'd88

`define H\_SYNC 11'd128

`define H\_DISP 11'd800

`define H\_TOTAL (`H\_FRONT + `H\_BACK + `H\_SYNC + `H\_DISP)

`define V\_FRONT 11'd1

`define V\_BACK 11'd23

`define V\_SYNC 11'd4

`define V\_DISP 11'd600

`define V\_TOTAL (`V\_FRONT + `V\_BACK + `V\_SYNC + `V\_DISP)

`define TITLE\_X1 'd50

`define TITLE\_Y1 'd10

`define TITLE\_X2 'd750

`define TITLE\_Y2 'd150

`define TITLE\_XL (`TITLE\_X2 - `TITLE\_X1)

`define TITLE\_YL (`TITLE\_Y2 - `TITLE\_Y1)

`define TEXT\_X1 'd88

`define TEXT\_Y1 'd170

`define TEXT\_X2 'd728

`define TEXT\_Y2 'd590

`define TEXT\_XL (`TEXT\_X2 - `TEXT\_X1)

`define TEXT\_YL (`TEXT\_Y2 - `TEXT\_Y1)

module VGA\_driver(

input CLK,

input RST,

input [1199:0] text,

input [2:0] Q,

input [9:0] cur,

output reg[3:0] R,

output reg[3:0] G,

output reg[3:0] B,

output HS,

output VS,

output reg otest

);

wire [10:0] Hcnt, Vcnt;

VGA\_control#(.H\_TOTAL(`H\_TOTAL), .V\_TOTAL(`V\_TOTAL)) get\_HV(.CLK(CLK), .RST(RST), .Hcnt(Hcnt), .Vcnt(Vcnt));

assign HS = (Hcnt < `H\_SYNC) ? 1'b0 : 1'b1; //VGA行同步信号

assign VS = (Vcnt < `V\_SYNC) ? 1'b0 : 1'b1; //VGA场信号同步

//---------------输出区域使能 800\*600---------------

wire [10:0] en\_H, en\_V; wire en;

assign en\_H = ((Hcnt >= `H\_SYNC + `H\_BACK) && (Hcnt < `H\_TOTAL - `H\_FRONT)) ? (Hcnt - `H\_SYNC - `H\_BACK + 1) : 11'b0;

assign en\_V = ((Vcnt >= `V\_SYNC + `V\_BACK) && (Vcnt < `V\_TOTAL - `V\_FRONT)) ? (Vcnt - `V\_SYNC - `V\_BACK + 1) : 11'b0;

assign en = ((en\_H > 11'b0) && (en\_V > 11'b0)) ? 1'b1 : 1'b0;

//---------------提示坐标确定---------------

wire [10:0] dTITLE\_H, dTITLE\_V;

assign dTITLE\_H = ((Hcnt >= `H\_SYNC + `H\_BACK + `TITLE\_X1) && (Hcnt < `H\_SYNC + `H\_BACK + `TITLE\_X2)) ?

(Hcnt - `H\_SYNC - `H\_BACK - `TITLE\_X1) : `TITLE\_XL;

assign dTITLE\_V = ((Vcnt >= `V\_SYNC + `V\_BACK + `TITLE\_Y1) && (Vcnt < `V\_SYNC + `V\_BACK + `TITLE\_Y2)) ?

(Vcnt - `V\_SYNC - `V\_BACK - `TITLE\_Y1) : `TITLE\_YL;

//---------------正文坐标确定---------------

wire [10:0] dTEXT\_H, dTEXT\_V;

assign dTEXT\_H = ((Hcnt >= `H\_SYNC + `H\_BACK + `TEXT\_X1) && (Hcnt < `H\_SYNC + `H\_BACK + `TEXT\_X2)) ?

(Hcnt - `H\_SYNC - `H\_BACK - `TEXT\_X1) : `TEXT\_XL;

assign dTEXT\_V = ((Vcnt >= `V\_SYNC + `V\_BACK + `TEXT\_Y1) && (Vcnt < `V\_SYNC + `V\_BACK + `TEXT\_Y2)) ?

(Vcnt - `V\_SYNC - `V\_BACK - `TEXT\_Y1) : `TEXT\_YL;

wire [2:0] dTH; wire[3:0] dTV;

assign dTH = ((Hcnt >= `H\_SYNC + `H\_BACK + `TEXT\_X1) && (Hcnt < `H\_SYNC + `H\_BACK + `TEXT\_X2)) ?

(((Hcnt - `H\_SYNC - `H\_BACK - `TEXT\_X1) & 'b1111) >> 'd1) : 'd0;

assign dTV = ((Vcnt >= `V\_SYNC + `V\_BACK + `TEXT\_Y1) && (Vcnt < `V\_SYNC + `V\_BACK + `TEXT\_Y2)) ?

(((Vcnt - `V\_SYNC - `V\_BACK - `TEXT\_Y1) & 'b11111) >> 'd1) : 'd0;

//---------------正文字符位置确定---------------

wire [9:0] dcnt;

assign dcnt = ((Hcnt >= `H\_SYNC + `H\_BACK + `TEXT\_X1) && (Hcnt < `H\_SYNC + `H\_BACK + `TEXT\_X2)

&& (Vcnt >= `V\_SYNC + `V\_BACK + `TEXT\_Y1) && (Vcnt < `V\_SYNC + `V\_BACK + `TEXT\_Y2)) ?

(((Vcnt - `V\_SYNC - `V\_BACK - `TEXT\_Y1) >> 'd5) \* 40 + ((Hcnt - `H\_SYNC - `H\_BACK - `TEXT\_X1) >> 'd4)): 'b0;

//---------------提示部分判圆---------------

wire [4:0] fC;

judge\_circle#(

.X1(`H\_SYNC + `H\_BACK + 'd120),

.X2(`H\_SYNC + `H\_BACK + 'd260),

.X3(`H\_SYNC + `H\_BACK + 'd400),

.X4(`H\_SYNC + `H\_BACK + 'd540),

.X5(`H\_SYNC + `H\_BACK + 'd680),

.Y(`V\_SYNC + `V\_BACK + ((`TITLE\_Y1 + `TITLE\_Y2) >> 'b1))

)

in\_circle(.Hcnt(Hcnt), .Vcnt(Vcnt), .fC(fC));

//---------------正文部分取字符点阵---------------

wire Point;

VGA\_char print\_char(.dH(dTH), .dV(dTV),

.alph(text[6 \* (dcnt >> 'b1) + 5 -: 6]), .dcnt(dcnt),

.Point(Point));

//---------------RGB值确定---------------

always @(Hcnt or Vcnt)

begin

if(en > 1'b0)

begin

if(dTITLE\_H < `TITLE\_XL && dTITLE\_V < `TITLE\_YL ) //TITLE\_part

begin

case(Q)

'd0: // stop

if(fC[0] == 1)

begin

R <= 'd0; G <= 'd0; B <= 'd15; // blue

end

else

begin

R <= 'd0; G <= 'd0; B <= 'd0;

end

'd1: // start

if(fC[1] == 1)

begin

R <= 'd0; G <= 'd15; B <= 'd0; // green

end

else

begin

R <= 'd0; G <= 'd0; B <= 'd0;

end

'd2: // pause

if(fC[2] == 1)

begin

R <= 'd15; G <= 'd15; B <= 'd0; // yellow

end

else

begin

R <= 'd0; G <= 'd0; B <= 'd0;

end

'd3: // win

if(fC[3] == 1)

begin

R <= 'd15; G <= 'd15; B <= 'd15; // white

end

else

begin

R <= 'd0; G <= 'd0; B <= 'd0;

end

'd4: // lose

if(fC[4] == 1)

begin

R <= 'd15; G <= 'd0; B <= 'd15; // red

end

else

begin

R <= 'd0; G <= 'd0; B <= 'd0;

end

default: ;

endcase

end

else if (dTEXT\_H < `TEXT\_XL && dTEXT\_V < `TEXT\_YL) //TEXT\_part

begin

if(Point == 1)

begin

if((dcnt >> 1) < cur)

begin

R <= 'd0; G <= 'd15; B <= 'd0;

end

else

begin

R <= 'd15; G <= 'd0; B <= 'd0;

end

end

else

begin

R <= 'd0; G <= 'd0; B <= 'd0;

end

end

else

begin

R <= 'd0; G <= 'd0; B <= 'd0;

end

end

else

begin

R <= 'd2;

G <= 'd2; //消隐区设置RBG，亮度调整，越低越亮

B <= 'd2;

end

end

endmodule

module VGA\_control

#(

parameter H\_TOTAL = 1056,

parameter V\_TOTAL = 628

)

(

input CLK,

input RST,

output reg [10:0] Hcnt,

output reg [10:0] Vcnt

);

initial

begin

Hcnt <= 0;

Vcnt <= 0;

end

wire VGA\_CLK;

Divider#(.mod(2)) get\_VGA\_CLK(.I\_CLK(CLK), .RST(RST), .O\_CLK(VGA\_CLK));

//---------------行扫描---------------

always @(posedge VGA\_CLK or posedge RST)

begin

if(RST)

Hcnt <= 11'd0;

else

begin

if(Hcnt < H\_TOTAL - 1)

Hcnt <= Hcnt + 1;

else

Hcnt <= 11'd0;

end

end

//---------------列扫描---------------

always @(posedge VGA\_CLK or posedge RST)

begin

if(RST)

Vcnt <= 11'd0;

else if(Hcnt == H\_TOTAL - 1) //行结束

begin

if(Vcnt < V\_TOTAL - 1)

Vcnt <= Vcnt + 1;

else

Vcnt <= 11'd0;

end

end

endmodule

module judge\_circle

#(

parameter signed X1 = 120,

parameter signed X2 = 260,

parameter signed X3 = 400,

parameter signed X4 = 540,

parameter signed X5 = 680,

parameter signed Y = 80,

parameter signed R2 = 3600

)

(

input signed [10:0] Hcnt,

input signed [10:0] Vcnt,

output [4:0] fC

);

assign fC[0] = ((Hcnt - X1) \* (Hcnt - X1) + (Vcnt - Y) \* (Vcnt - Y)) <= R2;

assign fC[1] = ((Hcnt - X2) \* (Hcnt - X2) + (Vcnt - Y) \* (Vcnt - Y)) <= R2;

assign fC[2] = ((Hcnt - X3) \* (Hcnt - X3) + (Vcnt - Y) \* (Vcnt - Y)) <= R2;

assign fC[3] = ((Hcnt - X4) \* (Hcnt - X4) + (Vcnt - Y) \* (Vcnt - Y)) <= R2;

assign fC[4] = ((Hcnt - X5) \* (Hcnt - X5) + (Vcnt - Y) \* (Vcnt - Y)) <= R2;

endmodule

module VGA\_char(

input [2:0] dH,

input [3:0] dV,

input [5:0] alph,

input [9:0] dcnt,

output reg Point

);

reg [127:0] Char\_Matrix[53:0];

initial

begin

Char\_Matrix[0] = {8'h00,8'h00,8'h00,8'h00,8'h00,8'h00,8'h00,8'h00,8'h00,8'h00,8'h00,8'h00,8'h00,8'h00,8'h00,8'h00}; // space

Char\_Matrix[1] = {8'h00,8'h00,8'h08,8'h1C,8'h36,8'h63,8'h63,8'h63,8'h7F,8'h63,8'h63,8'h63,8'h00,8'h00,8'h00,8'h00}; // A

Char\_Matrix[2] = {8'h00,8'h00,8'h7E,8'h33,8'h33,8'h33,8'h3E,8'h33,8'h33,8'h33,8'h33,8'h7E,8'h00,8'h00,8'h00,8'h00}; // B

Char\_Matrix[3] = {8'h00,8'h00,8'h1E,8'h33,8'h61,8'h60,8'h60,8'h60,8'h60,8'h61,8'h33,8'h1E,8'h00,8'h00,8'h00,8'h00}; // C

Char\_Matrix[4] = {8'h00,8'h00,8'h7C,8'h36,8'h33,8'h33,8'h33,8'h33,8'h33,8'h33,8'h36,8'h7C,8'h00,8'h00,8'h00,8'h00}; // D

Char\_Matrix[5] = {8'h00,8'h00,8'h7F,8'h33,8'h31,8'h34,8'h3C,8'h34,8'h30,8'h31,8'h33,8'h7F,8'h00,8'h00,8'h00,8'h00}; // E

Char\_Matrix[6] = {8'h00,8'h00,8'h7F,8'h33,8'h31,8'h34,8'h3C,8'h34,8'h30,8'h30,8'h30,8'h78,8'h00,8'h00,8'h00,8'h00}; // F

Char\_Matrix[7] = {8'h00,8'h00,8'h1E,8'h33,8'h61,8'h60,8'h60,8'h6F,8'h63,8'h63,8'h37,8'h1D,8'h00,8'h00,8'h00,8'h00}; // G

Char\_Matrix[8] = {8'h00,8'h00,8'h63,8'h63,8'h63,8'h63,8'h7F,8'h63,8'h63,8'h63,8'h63,8'h63,8'h00,8'h00,8'h00,8'h00}; // H

Char\_Matrix[9] = {8'h00,8'h00,8'h3C,8'h18,8'h18,8'h18,8'h18,8'h18,8'h18,8'h18,8'h18,8'h3C,8'h00,8'h00,8'h00,8'h00}; // I

Char\_Matrix[10] = {8'h00,8'h00,8'h0F,8'h06,8'h06,8'h06,8'h06,8'h06,8'h06,8'h66,8'h66,8'h3C,8'h00,8'h00,8'h00,8'h00}; // J

Char\_Matrix[11] = {8'h00,8'h00,8'h73,8'h33,8'h36,8'h36,8'h3C,8'h36,8'h36,8'h33,8'h33,8'h73,8'h00,8'h00,8'h00,8'h00}; // K

Char\_Matrix[12] = {8'h00,8'h00,8'h78,8'h30,8'h30,8'h30,8'h30,8'h30,8'h30,8'h31,8'h33,8'h7F,8'h00,8'h00,8'h00,8'h00}; // L

Char\_Matrix[13] = {8'h00,8'h00,8'h63,8'h77,8'h7F,8'h6B,8'h63,8'h63,8'h63,8'h63,8'h63,8'h63,8'h00,8'h00,8'h00,8'h00}; // M

Char\_Matrix[14] = {8'h00,8'h00,8'h63,8'h63,8'h73,8'h7B,8'h7F,8'h6F,8'h67,8'h63,8'h63,8'h63,8'h00,8'h00,8'h00,8'h00}; // N

Char\_Matrix[15] = {8'h00,8'h00,8'h1C,8'h36,8'h63,8'h63,8'h63,8'h63,8'h63,8'h63,8'h36,8'h1C,8'h00,8'h00,8'h00,8'h00}; // O

Char\_Matrix[16] = {8'h00,8'h00,8'h7E,8'h33,8'h33,8'h33,8'h3E,8'h30,8'h30,8'h30,8'h30,8'h78,8'h00,8'h00,8'h00,8'h00}; // P

Char\_Matrix[17] = {8'h00,8'h00,8'h3E,8'h63,8'h63,8'h63,8'h63,8'h63,8'h63,8'h6B,8'h6F,8'h3E,8'h06,8'h07,8'h00,8'h00}; // Q

Char\_Matrix[18] = {8'h00,8'h00,8'h7E,8'h33,8'h33,8'h33,8'h3E,8'h36,8'h36,8'h33,8'h33,8'h73,8'h00,8'h00,8'h00,8'h00}; // R

Char\_Matrix[19] = {8'h00,8'h00,8'h3E,8'h63,8'h63,8'h30,8'h1C,8'h06,8'h03,8'h63,8'h63,8'h3E,8'h00,8'h00,8'h00,8'h00}; // S

Char\_Matrix[20] = {8'h00,8'h00,8'hFF,8'hDB,8'h99,8'h18,8'h18,8'h18,8'h18,8'h18,8'h18,8'h3C,8'h00,8'h00,8'h00,8'h00}; // T

Char\_Matrix[21] = {8'h00,8'h00,8'h63,8'h63,8'h63,8'h63,8'h63,8'h63,8'h63,8'h63,8'h63,8'h3E,8'h00,8'h00,8'h00,8'h00}; // U

Char\_Matrix[22] = {8'h00,8'h00,8'h63,8'h63,8'h63,8'h63,8'h63,8'h63,8'h63,8'h36,8'h1C,8'h08,8'h00,8'h00,8'h00,8'h00}; // V

Char\_Matrix[23] = {8'h00,8'h00,8'h63,8'h63,8'h63,8'h63,8'h63,8'h6B,8'h6B,8'h7F,8'h36,8'h36,8'h00,8'h00,8'h00,8'h00}; // W

Char\_Matrix[24] = {8'h00,8'h00,8'hC3,8'hC3,8'h66,8'h3C,8'h18,8'h18,8'h3C,8'h66,8'hC3,8'hC3,8'h00,8'h00,8'h00,8'h00}; // X

Char\_Matrix[25] = {8'h00,8'h00,8'hC3,8'hC3,8'hC3,8'h66,8'h3C,8'h18,8'h18,8'h18,8'h18,8'h3C,8'h00,8'h00,8'h00,8'h00}; // Y

Char\_Matrix[26] = {8'h00,8'h00,8'h7F,8'h63,8'h43,8'h06,8'h0C,8'h18,8'h30,8'h61,8'h63,8'h7F,8'h00,8'h00,8'h00,8'h00}; // Z

Char\_Matrix[27] = {8'h00,8'h00,8'h00,8'h00,8'h00,8'h3C,8'h46,8'h06,8'h3E,8'h66,8'h66,8'h3B,8'h00,8'h00,8'h00,8'h00}; // a

Char\_Matrix[28] = {8'h00,8'h00,8'h70,8'h30,8'h30,8'h3C,8'h36,8'h33,8'h33,8'h33,8'h33,8'h6E,8'h00,8'h00,8'h00,8'h00}; // b

Char\_Matrix[29] = {8'h00,8'h00,8'h00,8'h00,8'h00,8'h3E,8'h63,8'h60,8'h60,8'h60,8'h63,8'h3E,8'h00,8'h00,8'h00,8'h00}; // c

Char\_Matrix[30] = {8'h00,8'h00,8'h0E,8'h06,8'h06,8'h1E,8'h36,8'h66,8'h66,8'h66,8'h66,8'h3B,8'h00,8'h00,8'h00,8'h00}; // d

Char\_Matrix[31] = {8'h00,8'h00,8'h00,8'h00,8'h00,8'h3E,8'h63,8'h63,8'h7E,8'h60,8'h63,8'h3E,8'h00,8'h00,8'h00,8'h00}; // e

Char\_Matrix[32] = {8'h00,8'h00,8'h1C,8'h36,8'h32,8'h30,8'h7C,8'h30,8'h30,8'h30,8'h30,8'h78,8'h00,8'h00,8'h00,8'h00}; // f

Char\_Matrix[33] = {8'h00,8'h00,8'h00,8'h00,8'h00,8'h3B,8'h66,8'h66,8'h66,8'h66,8'h3E,8'h06,8'h66,8'h3C,8'h00,8'h00}; // g

Char\_Matrix[34] = {8'h00,8'h00,8'h70,8'h30,8'h30,8'h36,8'h3B,8'h33,8'h33,8'h33,8'h33,8'h73,8'h00,8'h00,8'h00,8'h00}; // h

Char\_Matrix[35] = {8'h00,8'h00,8'h0C,8'h0C,8'h00,8'h1C,8'h0C,8'h0C,8'h0C,8'h0C,8'h0C,8'h1E,8'h00,8'h00,8'h00,8'h00}; // i

Char\_Matrix[36] = {8'h00,8'h00,8'h06,8'h06,8'h00,8'h0E,8'h06,8'h06,8'h06,8'h06,8'h06,8'h66,8'h66,8'h3C,8'h00,8'h00}; // j

Char\_Matrix[37] = {8'h00,8'h00,8'h70,8'h30,8'h30,8'h33,8'h33,8'h36,8'h3C,8'h36,8'h33,8'h73,8'h00,8'h00,8'h00,8'h00}; // k

Char\_Matrix[38] = {8'h00,8'h00,8'h1C,8'h0C,8'h0C,8'h0C,8'h0C,8'h0C,8'h0C,8'h0C,8'h0C,8'h1E,8'h00,8'h00,8'h00,8'h00}; // l

Char\_Matrix[39] = {8'h00,8'h00,8'h00,8'h00,8'h00,8'h6E,8'h7F,8'h6B,8'h6B,8'h6B,8'h6B,8'h6B,8'h00,8'h00,8'h00,8'h00}; // m

Char\_Matrix[40] = {8'h00,8'h00,8'h00,8'h00,8'h00,8'h6E,8'h33,8'h33,8'h33,8'h33,8'h33,8'h33,8'h00,8'h00,8'h00,8'h00}; // n

Char\_Matrix[41] = {8'h00,8'h00,8'h00,8'h00,8'h00,8'h3E,8'h63,8'h63,8'h63,8'h63,8'h63,8'h3E,8'h00,8'h00,8'h00,8'h00}; // o

Char\_Matrix[42] = {8'h00,8'h00,8'h00,8'h00,8'h00,8'h6E,8'h33,8'h33,8'h33,8'h33,8'h3E,8'h30,8'h30,8'h78,8'h00,8'h00}; // p

Char\_Matrix[43] = {8'h00,8'h00,8'h00,8'h00,8'h00,8'h3B,8'h66,8'h66,8'h66,8'h66,8'h3E,8'h06,8'h06,8'h0F,8'h00,8'h00}; // q

Char\_Matrix[44] = {8'h00,8'h00,8'h00,8'h00,8'h00,8'h6E,8'h3B,8'h33,8'h30,8'h30,8'h30,8'h78,8'h00,8'h00,8'h00,8'h00}; // r

Char\_Matrix[45] = {8'h00,8'h00,8'h00,8'h00,8'h00,8'h3E,8'h63,8'h38,8'h0E,8'h03,8'h63,8'h3E,8'h00,8'h00,8'h00,8'h00}; // s

Char\_Matrix[46] = {8'h00,8'h00,8'h08,8'h18,8'h18,8'h7E,8'h18,8'h18,8'h18,8'h18,8'h1B,8'h0E,8'h00,8'h00,8'h00,8'h00}; // t

Char\_Matrix[47] = {8'h00,8'h00,8'h00,8'h00,8'h00,8'h66,8'h66,8'h66,8'h66,8'h66,8'h66,8'h3B,8'h00,8'h00,8'h00,8'h00}; // u

Char\_Matrix[48] = {8'h00,8'h00,8'h00,8'h00,8'h00,8'h63,8'h63,8'h36,8'h36,8'h1C,8'h1C,8'h08,8'h00,8'h00,8'h00,8'h00}; // v

Char\_Matrix[49] = {8'h00,8'h00,8'h00,8'h00,8'h00,8'h63,8'h63,8'h63,8'h6B,8'h6B,8'h7F,8'h36,8'h00,8'h00,8'h00,8'h00}; // w

Char\_Matrix[50] = {8'h00,8'h00,8'h00,8'h00,8'h00,8'h63,8'h36,8'h1C,8'h1C,8'h1C,8'h36,8'h63,8'h00,8'h00,8'h00,8'h00}; // x

Char\_Matrix[51] = {8'h00,8'h00,8'h00,8'h00,8'h00,8'h63,8'h63,8'h63,8'h63,8'h63,8'h3F,8'h03,8'h06,8'h3C,8'h00,8'h00}; // y

Char\_Matrix[52] = {8'h00,8'h00,8'h00,8'h00,8'h00,8'h7F,8'h66,8'h0C,8'h18,8'h30,8'h63,8'h7F,8'h00,8'h00,8'h00,8'h00}; // z

Char\_Matrix[53] = {8'h00,8'h00,8'h00,8'h00,8'h00,8'h00,8'h00,8'h00,8'h00,8'h00,8'h00,8'h00,8'h00,8'h00,8'h00,8'h00}; // FO

end

wire [127:0] Matrix;

assign Matrix = Char\_Matrix[alph];

always @(\*)

begin

if((dcnt & 1) == 'b0)

Point <= Matrix[127 - (8\*dV + dH)]; //Char\_Matrix里是从高位往低位存的

else

Point <= 'b0;

end

endmodule

计时及数码管子系统：

数码管输出

数码管控制器

clk、rst、Q

SEG、DIG、last\_time

module DIG\_count\_down(

input wire [26:0] start\_time,

input rst,

input setting\_change,

input clk,

input clock\_go,

output [7:0] SEG,

output [7:0] DIG,

output [29:0] last\_time

);

reg [26:0] count\_time;

initial

begin

count\_time <= start\_time;

end

assign last\_time = count\_time;

wire [3:0] num[7:0];

assign num[0] = count\_time % 'd10;

assign num[1] = count\_time / 'd10 % 'd10;

assign num[2] = count\_time / 'd100 % 'd10;

assign num[3] = count\_time / 'd1000 % 'd10;

assign num[4] = count\_time / 'd10000 % 'd10;

assign num[5] = count\_time / 'd100000 % 'd10;

assign num[6] = count\_time / 'd1000000 % 'd10;

assign num[7] = count\_time / 'd10000000;

wire CLK\_count;

Divider #(10000000) per\_sec(.I\_CLK(clk), .RST(rst), .O\_CLK(CLK\_count));

DIG\_driver show\_dig(.rst(~rst), .clk(clk),

.num0(num[0]), .num1(num[1]), .num2(num[2]), .num3(num[3]),

.num4(num[4]), .num5(num[5]), .num6(num[6]), .num7(num[7]),

.SEG(SEG), .DIG(DIG));

always @(posedge CLK\_count or posedge rst)

begin

if(rst == 1 || setting\_change == 1)

count\_time <= start\_time;

else if(clock\_go == 1 && count\_time != 0)

count\_time <= count\_time - 1;

end

endmodule

module DIG\_control(

input clk,

input rst,

output reg [2:0] DIG\_ID

);

initial

begin

DIG\_ID <= 0;

end

always @(posedge clk or negedge rst)

begin

if(!rst)

DIG\_ID <= 1'b0;

else

begin

if(DIG\_ID == 'd7)

DIG\_ID <= 1'b0;

else

DIG\_ID <= DIG\_ID + 1'b1;

end

end

endmodule

module DIG\_decoder\_3\_8(

input [2:0] iC,

input rst,

output reg [7:0] oC

);

always @(iC or rst)

begin

if(!rst)

oC <= 8'b0000\_0000;

else

begin

case(iC)

'd0: oC <= 8'b1111\_1110;

'd1: oC <= 8'b1111\_1101;

'd2: oC <= 8'b1111\_1011;

'd3: oC <= 8'b1111\_0111;

'd4: oC <= 8'b1110\_1111;

'd5: oC <= 8'b1101\_1111;

'd6: oC <= 8'b1011\_1111;

'd7: oC <= 8'b0111\_1111;

default: oC <= 8'b0000\_0000;

endcase

end

end

endmodule

module NUM\_selector\_8(

input [2:0] iC,

input [3:0] num0,

input [3:0] num1,

input [3:0] num2,

input [3:0] num3,

input [3:0] num4,

input [3:0] num5,

input [3:0] num6,

input [3:0] num7,

output reg [3:0] num

);

always @(iC)

case(iC)

'd0: num <= num0;

'd1: num <= num1;

'd2: num <= num2;

'd3: num <= num3;

'd4: num <= num4;

'd5: num <= num5;

'd6: num <= num6;

'd7: num <= num7;

default: num <= 4'bzzzz;

endcase

endmodule

module SEG\_decoder\_4\_8(

input [7:0] dig,

input rst,

input [3:0] num,

output reg [7:0] seg

);

always @(dig or rst)

begin

if(!rst)

seg <= 8'b0000\_0000;

else

begin

casez(num)

'd0: seg[6:0] <= 8'b100\_0000;

'd1: seg[6:0] <= 8'b111\_1001;

'd2: seg[6:0] <= 8'b010\_0100;

'd3: seg[6:0] <= 8'b011\_0000;

'd4: seg[6:0] <= 8'b001\_1001;

'd5: seg[6:0] <= 8'b001\_0010;

'd6: seg[6:0] <= 8'b000\_0010;

'd7: seg[6:0] <= 8'b111\_1000;

'd8: seg[6:0] <= 8'b000\_0000;

'd9: seg[6:0] <= 8'b001\_0000;

default: seg <= 8'b0000\_0000;

endcase

if(dig == 8'b1111\_1101)

seg[7] <= 1'b0;

else

seg[7] <= 1'b1;

//最高位为小数点

end

end

endmodule

1. 测试模块建模

倒计时模块直接下板就能测试



PS2模块测试：

module PS2\_tb;

/\* parameter \*/

parameter [31:0] clock\_period = 10;

//--------ps2\_keyboard interface signals--------

reg clk;

wire kbd\_clk, kbd\_data;

wire [7:0] DIG, SEG;

wire [7:0]ps2\_data;

wire [2:0] oQ;

reg [1199:0] text;

initial

begin

text[5:0] <= 'd1;

text[11:6] <= 'd2;

end

GAME\_control uut(.rst(0), .itext(text), .clk(clk), .data\_in(kbd\_data), .clk\_in(bkd\_clk),

.time\_go('d1), .oQ(oQ));

PS2\_keyboard\_model model(.ps2\_clk(kbd\_clk), .ps2\_data(kbd\_data));

//--------clock driver--------

initial

begin

clk = 0;

forever #(clock\_period/2) clk = ~clk;

end

initial

begin

#40

model.kbd\_sendcode(8'h1C); // press 'A'

model.kbd\_sendcode(8'hF0); // break code

model.kbd\_sendcode(8'h1C); // release 'A'

model.kbd\_sendcode(8'h1B); // press 'S'

#20 model.kbd\_sendcode(8'h1B); // keep pressing 'S'

#20 model.kbd\_sendcode(8'h1B); // keep pressing 'S'

model.kbd\_sendcode(8'hF0); // break code

model.kbd\_sendcode(8'h1B); // release 'S'

#20;

$stop;

end

endmodule

module PS2\_keyboard\_model(

output reg ps2\_clk,

output reg ps2\_data

);

parameter [31:0] kbd\_clk\_period = 60;

initial ps2\_clk = 1'b1;

task kbd\_sendcode;

input [7:0] code; // key to be sent

integer i;

reg[10:0] send\_buffer;

begin

send\_buffer[0] = 1'b0; // start bit

send\_buffer[8:1] = code; // code bits

send\_buffer[9] = ~(^code); // odd parity bit

send\_buffer[10] = 1'b1; // stop bit

i = 0;

while( i < 11)

begin

// set kbd\_data

ps2\_data = send\_buffer[i];

#(kbd\_clk\_period/2) ps2\_clk = 1'b0;

#(kbd\_clk\_period/2) ps2\_clk = 1'b1;

i = i + 1;

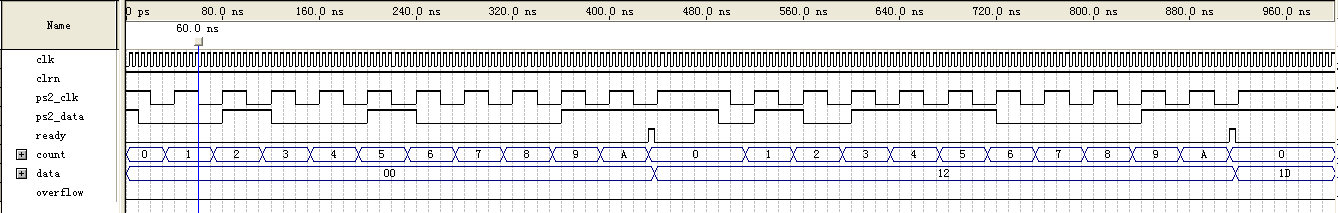
end

end

endtask

endmodule

所得仿真波形图如下：



VGA利用IP核实现测试，在VGA中添加IP核并链接本地coe文件

部分代码如下：

wire [17:0]addr;

wire [15:0]data;

assign addr = (en > 1'b0) ? ((en\_V - 1) / 2 \* 400 + (en\_H - 1) / 2) : {18{1'bz}};

blk\_mem\_gen\_0 IP\_ram(.clka(VGA\_CLK), .addra(addr), .douta(data), .ena(1));

//---------------RBG数值确定---------------

always @(Hcnt or Vcnt)

begin

if(en > 1'b0)

begin

R <= (out\_en > 1'b0) ? data[15:12] : 11'd0;

G <= (out\_en > 1'b0) ? data[10:7] : 11'd0;

B <= (out\_en > 1'b0) ? data[4:1] : 11'd0;

/\*if(Vcnt<100)begin

R = 0;

G = 0;

B = 15;

end

else if(Vcnt<200)begin

R = 0;

G = 15;

B = 0;

end

else begin

R = 15;

G = 0;

B = 0;

end\*/

end

else

begin

R = 11'd2;

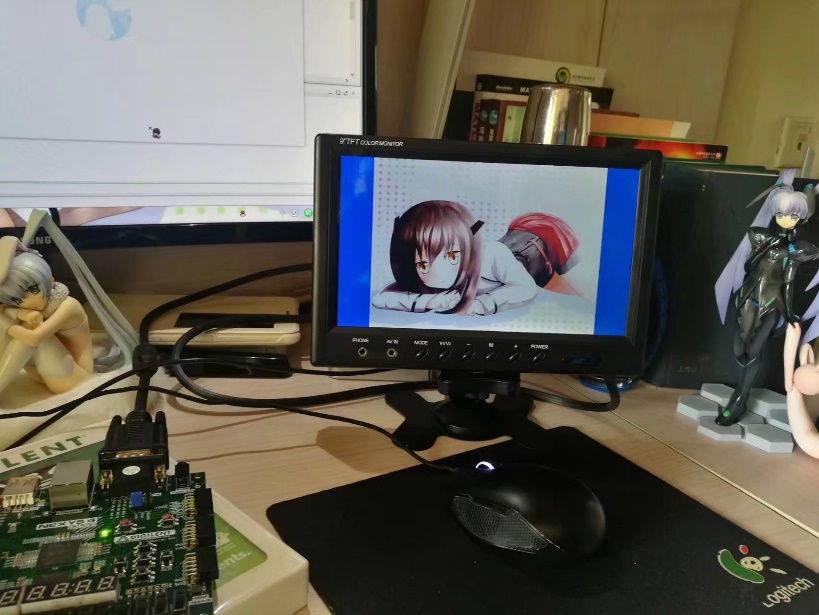
G = 11'd2; //消隐区设置RBG，如果不管好像默认全15会黑屏(?)

B = 11'd2;

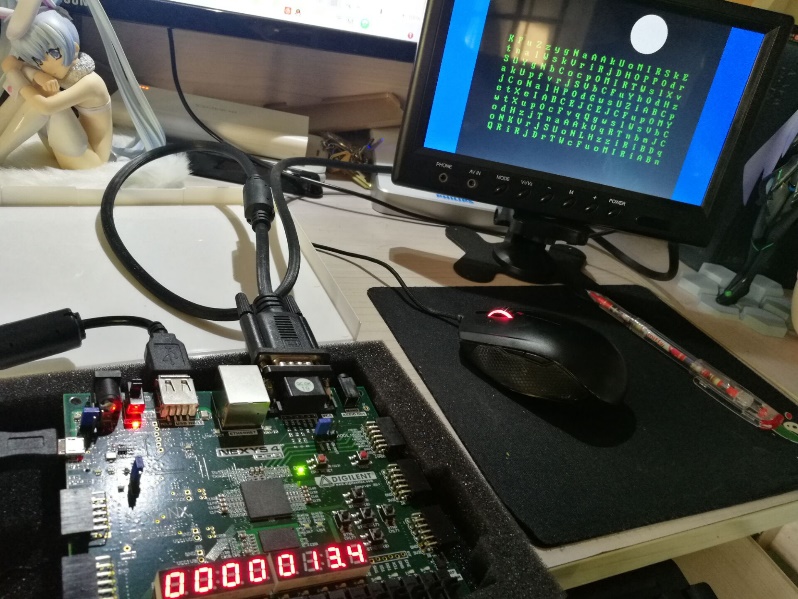
end

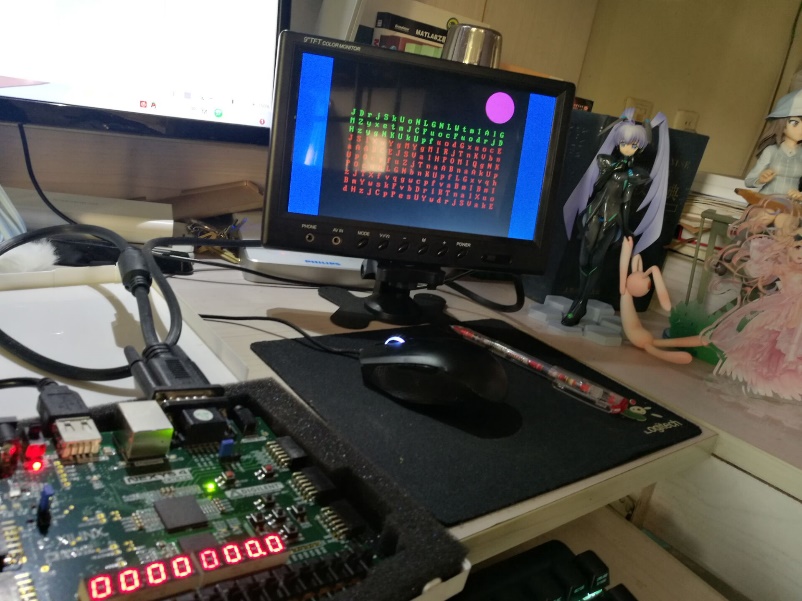
end

实际测试结果

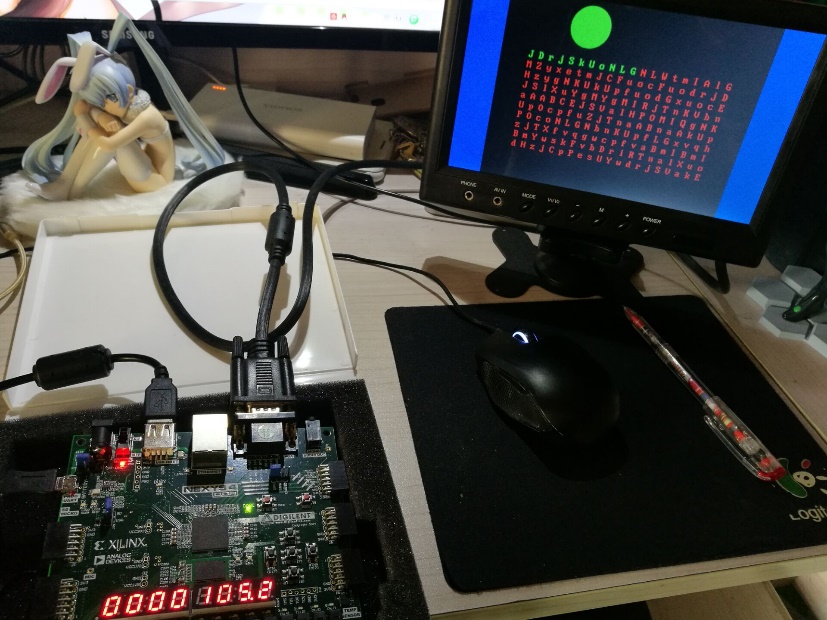


1. 实验结果

游戏胜利

游戏失败（时间结束）

暂停倒计时

倒计时开始

设置状态（可以调整时间）

1. 结论

通过上述的程序编写，实现了通过PS2进行键盘输入，通过七段数码管进行倒计时输出，通过板上按钮控制时间、开始、暂停并且通过VGA显示提示及字符的实验目的。

其中，键盘输入支持了26个英文字母，包括大小写一共52个数据，包括shift一共27个键的输入

七段数码管实现了每组数码管显示不同信息，并反映游戏倒计时的实际功能

VGA实现了定点输出不同颜色的提示圆，以及输出点阵英文字符的功能

三者共同组成了本次综合实验的实际功能，并通过时序逻辑将三者的时序统一在了一起，实现共同输出，共同反馈

1. 心得体会及建议

心得体会：

1. 模块分块测试

本次作业我编写时采用了分块编写的方法。首先是单独的VGA编写，编写好时序和行、场扫描后利用板子内自带ram的IP核实现了图片的下板。

之后则是对于PS2的编写，通过单独编写PS2，利用test\_bench测试获得了波形图。

接下来是对板上七段数码管的编写，通过在之前的七段数码管显示作业上的修改，首先实现了对于单组数码管显示不同信息的实现，进而加入分频器，使数码管以高频进行明暗切换，实现不同数码管显示不同信息的功能。

在实现以上功能之后，联合PS2和数码管的代码，实现在数码管上输出PS2键值的功能，最后将时序连入VGA，添加入游戏状态机，实现了游戏的整体编写。

通过实际操作可以体会到，将不同模块分不同组进行编写，独立测试其功能后再将所有模块合并起来，可以实现程序编写的高效化

1. 提前规划程序结构

这次实验编写时，由于前期没有预先的规划，导致在VGA编写时对于控制器和程序输入输出接口规划不够合理，在后期合并其他模块时修改较大，导致最后程序的bug也主要出现在VGA部分。

因此在程序设计时需要对于整个程序要进行有序的规划，对于各模块所需接口、各功能由几个模块组成要有合理、有序的规划，可以有效避免后期合并时造成的不必要的模块间问题的冲突

1. 善于利用各部件进行测试

这次测试PS2时采用了数码管的测试方法，但实际上联合两者时遇到的问题依旧不少，调了大概有5小时左右。实际上板子上的LED灯可以实现更简便的测试，而所幸这次实验也利用到了数码管，使得这些时间没有白费。对于测试的方法还应多加考虑

1. 注意各模块间的时序统一

这次的各个模块都用到了时序逻辑，分开编写时问题不大，但合并编写时就出现了时序冲突的矛盾，一开始甚至没有发现导致在查找错误上花费了一定的时间。

因此对于各模块间的磨合和时序统一这一点还需要多加注意

1. 关于随机数

由于$random无法综合的缘故，在硬件中实现随机数一般采用LFSR线性反馈移位寄存器来实现伪随机数的生成，其生成的循环节长度与D触发器个数挂钩，一开始仅采用了网上较多的8位导致生成了较多的相同字符，后通过查找LFSR\_TABLE编写了16位LFSR，得到了较为良好的随机数序列，对于实际中计算机硬件内部随机数的生成有了更好的理解

建议：

1. 在平时课程上增加模块使用的教学

对于一些基础模块，例如VGA、PS2，课上可以添加相应的课时介绍其工作原理，例如时序的变化，以及使用时需要注意的部分，也可以在课后作业中进行一定的练手，而不至于将所有的工作量都堆至期末，使得时间略微有点吃紧

1. 增加时序逻辑相应的课时

这次编写实验时着实体会到了时序逻辑的双面性。它在某些操作时显得十分便利，但也会导致部件编写时十分容易出现端口间的冲突。因此希望能增加时序逻辑在实际工程编写中的教学课程，着重强调其在实际应用中所带来的各种问题以及避免的方法