**İhsan Doğramacı Bilkent University**

**Computer Engineering**



**CS223 Digital Design**

***DIGITAL HUNT***

Fuad AGHAZADA~21503691~Section2

08.05.2017

**Final Report**

***The Block Diagram***

***NewProject%20(6).pdf***

Picture 1. Main module

**Description of modules**

ClockDivider

* Divides the clock coming from the board to the appropriate frequencies depending on which modules will use them.
* Inputs: It uses the standard clock coming from Basys 3 FPGA Board.
* It uses a 27 bits and 11 bits counter and increments the each of them counter in each clock cycle, then it sets the 22th bit to gateClk1 (the speed for the first level), 21th bit to gateClk2 (the speed for the second level), 20th bit to gateClk3 (the speed for the third level) and again 20th bit to the cursorClk, which is defined as sensitivity of the crosshair. Finally, for VGA the second counter assigned 1th  bit to the vgaClk.

VGA\_Monitor

* Takes the clock coming from the ClockDivider, synchronizes the monitor horizontally and vertically and defines the x and y coordinates for the pixels.
* It uses vgaClk coming out of the ClockDivider.
* Horizontal synchronizer starts increase by 1 on every clock edge until the horizontal end (rightmost) of the screen. Then the vertical synchronizer begins the increment on the same way until the vertical end (bottom) of the monitor. Every increment is defined as a pixel. (for x coordinate increment of the hsync, for y coordinate increment of the vsync) Finally, the counts for horizontal and vertical synchronizers, and defined x and y pixels are assigned to the outputs of the module.

FlyingGates

* Creates the x and y coordinates and dimensions for the flying blocks of the game, selects randomly from one of the given 8 Y locations, changes the x coordinates, which makes the gates move horizontally on the screen, determines their direction, number of misses and lost state of the game.
* It uses gateClkOut, which is coming from a SpeedOfGates module, in which the speed of the gates is determined according to the difficulty level of the game. It is chosen from the 3 gate clocks coming from the ClockDivider.
* If the gate is not shot, the X coordinate of the gate will change from the leftmost (0) to rightmost (640) by incrementing the current x position of it, and the number of misses increases once, however, in case the gate is shot, the direction of the gate will change and the current x coordinate of the gate will be set to the rightmost (640) x coordinate and then it will decrease until it reaches the leftmost (0) of the screen, and it will continue like this until the number of misses reaches the maximum, which sets the game state to lost. In order to choose random y positions, a count variable is created, which counts until 8 and is set 0 again during the game. When the gate is shot or it comes to the edge of the screen, the count is selected, and according to the selected count the predefined y location is taken as a new y position.

Crosshair

* Defines x and y coordinates for the crosshair on the screen, which is able to aim the gates and shoot them using the center button on the board.
* It uses cursorClk which is generated in the ClockDivider.
* According to the inputs coming the buttons, the current x and y coordinates of the crosshair is able to change its positon to any location on the screen – if the player pushes up button the x position of the cursor will not change, but the y coordinate of it will be decremented so that the cursor goes up. If the crosshair reaches the edges of the screen (is checked by case statements), it appears on the symmetrically opposite side of the screen.

SpeedOfGates

* According to the current difficulty level, this module chooses the proper clock from 3 clock, which are coming from the Clock Divider.
* It uses standard clock coming from the board.
* Gate clocks are already divided to the proper frequencies in the ClockDivider module. It takes these clocks, difficulty level (coming from another module) and then checks by cases so that for every difficulty level there is a matching clock, which causes different speeds for every level of the game. Shortly, if difficultyLevel is 2’b00, it chooses the slow clock (gateClk1), if it is 2’b01, much faster clock (gateClk2) is selected, finally, for the max level the fastest clock is selected (gateClk3). After checking, it gives the selected clock as an input to the FlyingGates\* module.

GameState

* Most of the game logic is constructed in this module so that according to the obtained score and level, it is determined that you won the game or not.
* It uses the gateClkOut, which is generated in the SpeedOfGates module.
* On every clock edge, this module checks if you could shoot the flying gates, and according to this case, it increments your score. In order to pass the levels, the player should kill 5 gates for each of the levels, which means that the player should get 15 points in order to win the game. The module checks these conditions by proper if statements on every clock edge.

IsShot

* The role of this module is to check whether the player is successful on killing a gate.
* It uses cursorClk, which is divided in the ClockDivider.
* The way of checking is quite straightforward so that if the current coordinates of the crosshair is on the current position of the gate and the player has presses fire button, it is considered that the gate is shot and given as the output to this module, which is used as an input in other modules.

StartScreen

* Using the pixels coming from VGA\_Monitor module, this module draws the name of the game and a play button on the screen.
* It uses vgaClk, which is generated in the ClockDivider, in order to draw.
* To draw the letters on the screen, beginning positions and dimensions for every rectangle, in which the letters are drawn, are defined as constant parameters. Then in every rectangle using the x and y pixels coming from the VGA\_Monitor module, the letters are drawn. The drawn letters are kept in a variable, which is assigned to the output of the module, and sent as an input to the Color module, where it is colored with the proper colors.

EndScreen

* According to the game state, the player sees a proper smiley on the screen: if the player is won, there is a happy face on the screen, however, for the lost state a sad face appears on the screen.
* It uses vgaClk, which is generated in the ClockDivider.
* The smileys are drawn using the pixels coming from the VGA, and kept in a variable screen – assigned to the output of the module, which keeps the two different smileys.

Then according to the state of the game, Color module draws the proper image on the screen.

Background

* The flying gates and crosshair, which the player see during the game, are drawn in this module.
* It uses vgaClk, which is generated in the ClockDivider.
* According to the current position and direction of the flying gates it will draw the image of AND gate and according to the coordinates of the cursor, it will draw a crosshair on the screen and keeps it on a 2 bits’ variable – assigned to the output of the module. Then it is given to the Color module, in which the proper colors are generated for these objects.

Color

* In order to see the gates and crosshair on the screen, this module generates colors for them.
* It uses standard clock coming from the board.
* The module takes the inputs coming from the Background, StartScreen, EndScreen modules then generates colors for the objects, in case the video is on (vga synchronizers are working) and the start switch is on (until this it colors the start screen), which is for pausing the game. Then according to the game state, it also generates proper colors for the end screen by giving the proper values for the RGB colors in the proper case. If isGameOver, which is also an input of the module, is 1 it takes the screen that is output of the EndScreen,, checks that if it is 2’b00, and using red, green and blue sequences to colors it with red. If the player wins – isWon is 1, it will use the same output but this time checking whether if the screen output is 2’b01 and implementing the same coloring procedure. Similarly, to paint the gates and the crosshair it checks the data in the output of the Background module – if it is 2’b00, it paints the gate to the blue, otherwise it paints the parts of the crosshair; the position of the center is painted to red, and the other parts to white.

SevSeg\_4digit

* The score and the number of misses are display on the 4 digit seven segment display on the board.
* It uses standard clock coming from the board.
* According to the coming input from the game, it shows the current score and the number of misses during the game so that the player is aware of the state of the game. This information about the game causes the proper led segments to be turned on.

DotMatrix

* The difficulty level is denoted on the 8x8 LED matrix with matching number of lines.
* It uses standard clock coming from the board.
* The clock is divided to the proper frequency inside the module. According to the pulses coming from the clock, the rows of the matrix is powered by the shifting of the data. According to the given difficult level, the number of rows increases by once, which are checked in a case statement.

**References**

We used 2 modules, which we found on the Internet for our project:

* *VGA\_Monitor* (we translated the Verilog code to the System Verilog) - <https://www.element14.com/community/thread/23394/l/draw-vga-color-bars-with-fpga-in-verilog?displayFullThread=true>

We made some few modifications on the code and then used it in our project.

* *DotMatrix* - <https://courses.cs.washington.edu/courses/cse369/16wi/labs/LED_Array_Tutorial.pdf>

We used the whole code with only a few modifications.

* We also used the *SevSeg\_4digit* module from our Lab assignment 5:

<https://www.unilica.com/index.php?v=course&s=home&cid=166&mv=11&aid=24277>

**Appendices**

*Code list*

-----*Main* – top module for the project

-----*ClockDivider* – located in the ClockDivider.sv and used in the main module with the name clkdvdr.

-----*VGA\_Monitor* – located in the VGA\_Monitor.sv and used in the main module with the name vga.

-----*FlyingGates* – located in the FlyingGates.sv and used in the main module with the name gates.

-----*Crosshair* – located in the Crosshair.sv and used in the main module with the name crs.

-----*SpeedOfGates* – located in the SpeedOfGates.sv and used in the main module with the name sog.

-----*GameState* – located in the GameState.sv and used in the main module with the name gs.

-----*IsShot* – located in the IsShot.sv and used in the main module with the name is.

-----*StartScreen* – located in the StartScreen.sv and used in the main module with the name ss.

-----*EndScreen* – located in the EndScreen.sv and used in the main module with the name es.

-----*Background* – located in the Background.sv and used in the main module with the name bg.

-----*Color* – located in the Color.sv and used in the main module with the name c.

-----*SevSeg\_4digit* – located in the SevSeg\_4digit.sv and used in the main module with the name svndgt.

-----*DotMatrix* – located in the DotMatrix.sv and used in the main module with the name dm.

* Data sheet for 8x8 LED Matrix - <http://oomlout.com/8X8M/8X8M-Guide.pdf>
* We did not uje any additional hardware from out of Bilkent, we used the VGA cable, Beti board and Basys3 FPGA board to implement out project.

**System Verilog code**

# Main.sv

`timescale 1ns / 1ps

module Main(

input logic clk, reset, right, left, up, down, fire, start,

output logic hsync, vsync, tempLed, dp,

output logic [3:0] red,

output logic [3:0] green,

output logic [3:0] blue,

output logic [6:0] leds,

output logic [3:0] anode,

output logic oe,

output logic SH\_CP,

output logic ST\_CP,

output logic reset2,

output logic DS,

output logic [7:0] rows

);

logic vidOn;

logic [9:0] pixelX;

logic [9:0] pixelY;

logic [9:0] width;

logic [9:0] height;

logic [9:0] X;

logic [9:0] Y;

logic dir;

logic [9:0] width\_cursor;

logic [9:0] height\_cursor;

logic [9:0] X\_cursor;

logic [9:0] Y\_cursor;

logic fired;

logic [1:0] background;

logic [1:0] screen; //endscreen

logic [1:0] screen2; //main screen

logic vgaClk;

logic gateClk1;

logic gateClk2;

logic gateClk3;

logic gateClkOut;

logic cursorClk;

logic [3:0] score;

logic [1:0] diffLevel;

logic [2:0] missed;

logic isGameOver;

logic isWon;

logic firstDigit;

logic [3:0] secondDigit;

always

begin

if(score < 10)

begin

firstDigit <= 0;

secondDigit <= score;

end

else

begin

firstDigit <= 1;

secondDigit <= score - 10;

end

end

always

tempLed <= fired;

ClockDivider clkdvdr(clk, reset, vgaClk, gateClk1, gateClk2, gateClk3, cursorClk);

SpeedOfGates sog(clk, gateClk1, gateClk2, gateClk3, diffLevel, gateClkOut);

VGA\_Monitor vga(vgaClk, reset, hsync, vsync, vidOn, pixelX, pixelY);

IsShot is(cursorClk, X, Y, height, width, X\_cursor, Y\_cursor, fire, fired);

GameState gs(gateClkOut, reset, fired, diffLevel, isWon, score);

SevSeg\_4digit svndgt(clk, diffLevel + 1, 7 - missed, firstDigit, secondDigit, leds[6], leds[5], leds[4], leds[3], leds[2], leds[1], leds[0], dp, anode);

DotMatrix dm(clk, diffLevel, oe, SH\_CP, ST\_CP, reset2, DS, rows);

Crosshair crs(cursorClk, reset, X\_cursor, Y\_cursor, right, left, up, down, X\_cursor, Y\_cursor);

FlyingGates gates(gateClkOut, reset, fired, isWon, start, X, Y, isGameOver, dir, missed, height, width, X, Y);

Background bg(vgaClk, dir, pixelX, pixelY, X, Y, height, width, X\_cursor, Y\_cursor, background);

EndScreen es(vgaClk, pixelX, pixelY, screen);

StartScreen ss(vgaClk, pixelX, pixelY, screen2);

Color c(clk, reset, start, background, vidOn, isGameOver, isWon, screen, screen2, red, green, blue);

endmodule

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# ClockDivider.sv

`timescale 1ns / 1ps

module ClockDivider(

input logic clk, reset,

output vgaClk, gateClock1, gateClock2, gateClock3, cursorClock

);

logic [26:0] count;

logic [10:0] count2;

always\_ff @ (posedge clk, posedge reset)

begin

if(reset)

count <= 0;

else

count <= count + 1;

end

always\_ff @ (posedge clk, posedge reset)

begin

if(reset)

count2 <= 0;

else

count2 <= count2 + 1;

end

assign vgaClk = count2[1];

assign gateClock1 = count[22];

assign gateClock2 = count[21];

assign gateClock3 = count[20];

assign cursorClock = count[20];

endmodule

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# VGA\_Monitor.sv

`timescale 1ns / 1ps

module VGA\_Monitor (

input logic vgaclk, //25Mhz

input logic reset, //asynchronus reset

output logic hsync,

output logic vsync,

output logic vidOn, //Looks when the screen is active

output logic [9:0]pixelX, //x goes up to 640 so we need 10 bits because 2^9=512 and 2^10=1024

output logic [9:0]pixelY);

parameter maximumHorizontalPixels = 800; //MAximum pixels in a line

parameter hbp = 144; //End of the horizontal back porch time

parameter hfp = 784; //Start of horizontal front porch time

parameter hPulse = 96;

parameter maximumVerticalLines = 521; //Maximum lines in the screen

parameter vbp = 30; //End of the vertical back porch time

parameter vfp = 511; //Start of vertical front porch time

parameter vPulse = 2;

logic [9:0] hcnt;

logic [9:0] vcnt;

always @(posedge vgaclk, posedge reset)

begin

if (reset)

begin

hcnt <= 0;

vcnt <= 0;

end

else

begin

if (hcnt < maximumHorizontalPixels - 1)

hcnt <= hcnt + 1;

else

begin

hcnt <= 0;

if (vcnt < maximumVerticalLines - 1)

vcnt <= vcnt + 1;

else

vcnt <= 0;

end

end

end

assign hsync = (hcnt < hPulse) ? 0:1;

assign vsync = (vcnt < vPulse) ? 0:1;

always\_comb

begin

if (hcnt >= hbp && hcnt < hfp && vcnt >= vbp && vcnt < vfp)

begin

pixelX <= hcnt - hbp; // pixelX

pixelY <= vcnt - vbp; // pixelY

vidOn <= 1; // active dýsplay

end

else

begin

pixelX <= 0;

pixelY <= 0;

vidOn <= 0; //ýnactýve dýsplay

end

end

endmodule

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# FlyingGates.sv

`timescale 1ns / 1ps

module FlyingGates(

input logic clk, reset, fire, isWon, start,

input logic [9:0] currentX,

input logic [9:0] currentY,

output logic isGameOver, direction,

output logic [2:0] miss,

output logic [9:0] height,

output logic [9:0] width,

output logic [9:0] outX,

output logic [9:0] outY

);

//internal variables

parameter WIDTH = 10'b0000011110;

parameter HEIGHT = 10'b0000011110;

//beginning x position

parameter beginX1 = 20;

//assigning the dimensions of the logic blocks

assign height = HEIGHT;

assign width = WIDTH;

//direction of the flying gate

logic dir;

//missed shots

logic [2:0] missed;

//internal constants for y

parameter Y0 = 40;

parameter Y1 = 100;

parameter Y2 = 160;

parameter Y3 = 220;

parameter Y4 = 280;

parameter Y5 = 340;

parameter Y6 = 400;

parameter Y7 = 450;

logic [2:0] count;

logic [2:0] select;

//state regsiter

always\_ff@(posedge clk, posedge reset)

begin

if(reset)

begin

currentX <= beginX1; //reseting the positons to the beginning

select <= count;

missed <= 0;

end

else

begin

if(count < 3'b111)

count <= count + 1;

else

count <= 0;

if(start)

begin

if(missed < 3'b111 && ~isWon)

begin

isGameOver <= 0;

if(~fire)

case(dir)

1'b0:

begin

if(currentX < 640)

begin

currentX <= currentX + 10;

if(count < 3'b111)

count <= count + 1;

else

count <= 0;

end

else

begin

if(missed < 3'b111)

missed <= missed + 1;

dir <= 1'b1;

select <= count;

end

end

1'b1:

begin

if(currentX > 0)

begin

currentX <= currentX - 10;

if(count < 3'b111)

count <= count + 1;

else

count <= 0;

end

else

begin

if(missed < 3'b111)

missed <= missed + 1;

dir <= 1'b0;

select <= count;

end

end

endcase

else

begin

if(dir)

currentX <= 640;

else

currentX <= 0;

select <= count;

end

end

else if(isWon)

begin

currentX <= currentX;

currentY <= currentY;

end

else

begin

currentX <= currentX;

currentY <= currentY;

isGameOver <= 1'b1;

end

end

else

begin

currentX <= currentX;

currentY <= currentY; //reseting the positons to the beginning

missed <= missed;

end

end

end

always\_comb

case(select)

3'b000 : currentY <= Y5;

3'b001 : currentY <= Y7;

3'b010 : currentY <= Y2;

3'b011 : currentY <= Y1;

3'b100 : currentY <= Y4;

3'b101 : currentY <= Y0;

3'b110 : currentY <= Y6;

3'b111 : currentY <= Y3;

endcase

always\_comb

begin

miss <= missed;

direction <= dir;

end

//assigning outputs

assign outX = currentX;

assign outY = currentY;

endmodule

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Crosshair.sv

`timescale 1ns / 1ps

module Crosshair(

input logic clk, reset,

input logic [9:0] currentX,

input logic [9:0] currentY,

input logic right, left, up, down,

output logic [9:0] X,

output logic [9:0] Y

);

//internal variables

parameter beginX = 320;

parameter beginY = 240;

always\_ff@(posedge clk, posedge reset)

begin

if(reset)

begin

currentX <= beginX; //reseting the positons to the beginning

currentY <= beginY;

end

else

begin

if(right)

begin

if(currentX + 10 < 640)

begin //end of the monitor

currentX <= currentX + 10;

currentY <= currentY;

end

else

begin

begin

currentX <= 0;

currentY <= currentY;

end

end

end

else if(left)

begin

if(currentX - 10 > 0)

begin //end of the monitor

currentX <= currentX - 10;

currentY <= currentY;

end

else

begin

begin

currentX <= 640;

currentY <= currentY;

end

end

end

else if(up)

begin

if(currentY - 10 < 480)

begin //end of the monitor

currentY <= currentY - 10;

currentX <= currentX;

end

else

begin

begin

currentX <= currentX;

currentY <= 480;

end

end

end

else if(down)

begin

if(currentY + 10 > 0)

begin //end of the monitor

currentY <= currentY + 10;

currentX <= currentX;

end

else

begin

begin

currentX <= currentX;

currentY <= 0;

end

end

end

else

begin

currentX <= currentX;

currentY <= currentY;

end

end

end

//assigning outputs

assign X = currentX;

assign Y = currentY;

endmodule

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# SpeedOfGates.sv

`timescale 1ns / 1ps

module SpeedOfGates(

input logic clk, clk1, clk2, clk3,

input logic [1:0] diff,

output logic clock\_out

);

always\_ff @(posedge clk)

case(diff)

2'b00: begin

clock\_out <= clk1;

end

2'b01: begin

clock\_out <= clk2;

end

2'b10: begin

clock\_out <= clk3;

end

default : begin

clock\_out <= clk1;

end

endcase

endmodule

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# GameState.sv

`timescale 1ns / 1ps

module GameState(

input logic clk, reset,

input logic fired,

input logic [1:0] difficultyLevel,

output logic isWon,

output logic [3:0] score

);

logic [3:0] count;

logic [1:0] diffL;

logic [3:0] levelCount;

always\_ff @(posedge clk, posedge reset)

begin

if(reset)

begin

count <= 4'b0;

diffL <= 2'b00;

isWon <= 0;

levelCount <= 4'b0101;

end

else if(fired)

begin

isWon <= 0;

if(count < levelCount - 1)

begin

count <= count + 1;

end

else

begin

//count <= 3'b0;

if(difficultyLevel < 2'b10)

begin

count <= count + 1;

diffL <= diffL + 1;

levelCount <= levelCount + 4'b0101;

end

else

begin

count <= count + 1;

isWon <= 1'b1;

end

end

end

else

begin

count <= count;

end

end

assign score = {count};

assign difficultyLevel = {diffL};

endmodule

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# IsShot.sv

`timescale 1ns / 1ps

module IsShot(

input logic clk,

input logic [9:0] gate\_x,

input logic [9:0] gate\_y,

input logic [9:0] gate\_height,

input logic [9:0] gate\_width,

input logic [9:0] cursor\_x,

input logic [9:0] cursor\_y,

input logic fire,

output logic out

);

always\_ff @(posedge clk)

if(cursor\_x > gate\_x - gate\_width/2 && cursor\_x < gate\_x + gate\_width/2 && cursor\_y > gate\_y - (gate\_height/2 + 5) && cursor\_y < gate\_y + (gate\_height/2 + 5))

out <= fire;

else

out <= 1'b0;

endmodule

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# StartScreen.sv

`timescale 1ns / 1ps

module StartScreen(

input logic clk,

input logic [9:0] vga\_x,

input logic [9:0] vga\_y,

output logic [1:0] out

);

logic [1:0] screen;

parameter letter\_height = 100;

parameter letter\_width = 30;

parameter a = 10;

parameter gap = 10;

parameter beginPoint = 60;

parameter beginPoint1 = 80; //for D

parameter beginPoint2 = beginPoint1 + gap + letter\_width; //for I

parameter beginPoint3 = beginPoint2 + gap + (letter\_width/3); //for G

parameter beginPoint4 = beginPoint3 + gap + letter\_width; //for I

parameter beginPoint5 = beginPoint4 + gap + (letter\_width/3); //for T

parameter beginPoint6 = beginPoint5 + gap + letter\_width; //for A

parameter beginPoint7 = beginPoint6 + gap + letter\_width; //for L

parameter beginPoint8 = beginPoint7 + letter\_width + 50;

parameter beginPoint9 = beginPoint8 + gap + letter\_width;

parameter beginPoint10 = beginPoint9 + gap + letter\_width;

parameter beginPoint11 = beginPoint10 + gap + letter\_width;

//play sign

parameter beginPointP = 240;

parameter beginXpoint = 260;

parameter length = 80;

always\_ff @(posedge clk)

begin

if(vga\_x >= beginPoint1 && vga\_x <= beginPoint1 + a && vga\_y >= beginPoint && vga\_y <= beginPoint + letter\_height)

screen <= 2'b00;

else if(vga\_x >= beginPoint1 + a && vga\_x <= beginPoint1 + 2\*a && vga\_y >= beginPoint && vga\_y <= beginPoint + a)

screen <= 2'b00;

else if(vga\_x >= beginPoint1 + a && vga\_x <= beginPoint1 + 2\*a && vga\_y >= beginPoint + letter\_height - a && vga\_y <= beginPoint + letter\_height )

screen <= 2'b00;

else if(vga\_x >= beginPoint1 + 2\*a && vga\_x <= beginPoint1 + 3\*a && vga\_y >= beginPoint + a && vga\_y <= beginPoint + letter\_height - a)

screen <= 2'b00;

//

else if(vga\_x >= beginPoint2 && vga\_x <= beginPoint2 + a && vga\_y >= beginPoint && vga\_y <= beginPoint + letter\_height)

screen <= 2'b00;

//

else if(vga\_x >= beginPoint3 && vga\_x <= beginPoint3 + a && vga\_y >= beginPoint && vga\_y <= beginPoint + letter\_height)

screen <= 2'b00;

else if(vga\_x >= beginPoint3 + a && vga\_x <= beginPoint3 + 2\*a && vga\_y >= beginPoint && vga\_y <= beginPoint + a)

screen <= 2'b00;

else if(vga\_x >= beginPoint3 + a && vga\_x <= beginPoint3 + 2\*a && vga\_y >= beginPoint + 4\*a && vga\_y <= beginPoint + 5\*a)

screen <= 2'b00;

else if(vga\_x >= beginPoint3 + a && vga\_x <= beginPoint3 + 2\*a && vga\_y >= beginPoint + letter\_height - a && vga\_y <= beginPoint + letter\_height)

screen <= 2'b00;

else if(vga\_x >= beginPoint3 + 2\*a && vga\_x <= beginPoint3 + 3\*a && vga\_y >= beginPoint && vga\_y <= beginPoint + a)

screen <= 2'b00;

else if(vga\_x >= beginPoint3 + 2\*a && vga\_x <= beginPoint3 + 3\*a && vga\_y >= beginPoint + 4\*a && vga\_y <= beginPoint + letter\_height)

screen <= 2'b00;

//

else if(vga\_x >= beginPoint4 && vga\_x <= beginPoint4 + a && vga\_y >= beginPoint && vga\_y <= beginPoint + letter\_height)

screen <= 2'b00;

//

else if(vga\_x >= beginPoint5 && vga\_x <= beginPoint5 + a && vga\_y >= beginPoint && vga\_y <= beginPoint + a)

screen <= 2'b00;

else if(vga\_x >= beginPoint5 + a && vga\_x <= beginPoint5 + 2\*a && vga\_y >= beginPoint && vga\_y <= beginPoint + letter\_height)

screen <= 2'b00;

else if(vga\_x >= beginPoint5 + 2\*a && vga\_x <= beginPoint5 + 3\*a && vga\_y >= beginPoint && vga\_y <= beginPoint + a)

screen <= 2'b00;

//

else if(vga\_x >= beginPoint6 && vga\_x <= beginPoint6 + a && vga\_y >= beginPoint && vga\_y <= beginPoint + letter\_height)

screen <= 2'b00;

else if(vga\_x >= beginPoint6 + a && vga\_x <= beginPoint6 + 2\*a && vga\_y >= beginPoint && vga\_y <= beginPoint + a)

screen <= 2'b00;

else if(vga\_x >= beginPoint6 + a && vga\_x <= beginPoint6 + 2\*a && vga\_y >= beginPoint + 4\*a && vga\_y <= beginPoint + 5\*a)

screen <= 2'b00;

else if(vga\_x >= beginPoint6 + 2\*a && vga\_x <= beginPoint6 + 3\*a && vga\_y >= beginPoint && vga\_y <= beginPoint + letter\_height)

screen <= 2'b00;

//

else if(vga\_x >= beginPoint7 && vga\_x <= beginPoint7 + a && vga\_y >= beginPoint && vga\_y <= beginPoint + letter\_height)

screen <= 2'b00;

else if(vga\_x >= beginPoint7 + a && vga\_x <= beginPoint7 + 3\*a && vga\_y >= beginPoint + letter\_height - a && vga\_y <= beginPoint + letter\_height)

screen <= 2'b00;

//

//

//

else if(vga\_x >= beginPoint8 && vga\_x <= beginPoint8 + a && vga\_y >= beginPoint && vga\_y <= beginPoint + letter\_height)

screen <= 2'b00;

else if(vga\_x >= beginPoint8 + a && vga\_x <= beginPoint8 + 2\*a && vga\_y >= beginPoint + 4\*a && vga\_y <= beginPoint + 5\*a)

screen <= 2'b00;

else if(vga\_x >= beginPoint8 + 2\*a && vga\_x <= beginPoint8 + 3\*a && vga\_y >= beginPoint && vga\_y <= beginPoint + letter\_height)

screen <= 2'b00;

//

else if(vga\_x >= beginPoint9 && vga\_x <= beginPoint9 + a && vga\_y >= beginPoint && vga\_y <= beginPoint + letter\_height)

screen <= 2'b00;

else if(vga\_x >= beginPoint9 + a && vga\_x <= beginPoint9 + 2\*a && vga\_y >= beginPoint + letter\_height - a && vga\_y <= beginPoint + letter\_height)

screen <= 2'b00;

else if(vga\_x >= beginPoint9 + 2\*a && vga\_x <= beginPoint9 + 3\*a && vga\_y >= beginPoint && vga\_y <= beginPoint + letter\_height)

screen <= 2'b00;

//

else if(vga\_x >= beginPoint10 && vga\_x <= beginPoint10 + a && vga\_y >= beginPoint && vga\_y <= beginPoint + letter\_height)

screen <= 2'b00;

else if(vga\_x >= beginPoint10 + a && vga\_x <= beginPoint10 + 15 && vga\_y >= beginPoint + 4\*a && vga\_y <= beginPoint + 5\*a)

screen <= 2'b00;

else if(vga\_x >= beginPoint10 + 15 && vga\_x <= beginPoint10 + 2\*a && vga\_y >= beginPoint + 5\*a && vga\_y <= beginPoint + 6\*a)

screen <= 2'b00;

else if(vga\_x >= beginPoint10 + 2\*a && vga\_x <= beginPoint10 + 3\*a && vga\_y >= beginPoint && vga\_y <= beginPoint + letter\_height)

screen <= 2'b00;

//

else if(vga\_x >= beginPoint11 && vga\_x <= beginPoint11 + a && vga\_y >= beginPoint && vga\_y <= beginPoint + a)

screen <= 2'b00;

else if(vga\_x >= beginPoint11 + a && vga\_x <= beginPoint11 + 2\*a && vga\_y >= beginPoint && vga\_y <= beginPoint + letter\_height)

screen <= 2'b00;

else if(vga\_x >= beginPoint11 + 2\*a && vga\_x <= beginPoint11 + 3\*a && vga\_y >= beginPoint && vga\_y <= beginPoint + a)

screen <= 2'b00;

//

else if(vga\_x >= beginXpoint && vga\_x <= beginXpoint + 10 && vga\_y >= beginPointP && vga\_y <= beginPointP + length)

screen <= 2'b01;

else if(vga\_x >= beginXpoint + 10 && vga\_x <= beginXpoint + 20 && vga\_y >= beginPointP + 5 && vga\_y <= beginPointP + length - 5)

screen <= 2'b01;

else if(vga\_x >= beginXpoint + 20 && vga\_x <= beginXpoint + 30 && vga\_y >= beginPointP + 10 && vga\_y <= beginPointP + length - 10)

screen <= 2'b01;

else if(vga\_x >= beginXpoint + 30 && vga\_x <= beginXpoint + 40 && vga\_y >= beginPointP + 15 && vga\_y <= beginPointP + length - 15)

screen <= 2'b01;

else if(vga\_x >= beginXpoint + 40 && vga\_x <= beginXpoint + 50 && vga\_y >= beginPointP + 20 && vga\_y <= beginPointP + length - 20)

screen <= 2'b01;

else if(vga\_x >= beginXpoint + 50 && vga\_x <= beginXpoint + 60 && vga\_y >= beginPointP + 25 && vga\_y <= beginPointP + length - 25)

screen <= 2'b01;

else if(vga\_x >= beginXpoint + 60 && vga\_x <= beginXpoint + 70 && vga\_y >= beginPointP + 30 && vga\_y <= beginPointP + length - 30)

screen <= 2'b01;

else if(vga\_x >= beginXpoint + 70 && vga\_x <= beginXpoint + 80 && vga\_y >= beginPointP + 35 && vga\_y <= beginPointP + length - 35)

screen <= 2'b01;

else

screen <= 2'b11;

end

assign out = {screen};

endmodule

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# EndScreen.sv

`timescale 1ns / 1ps

module EndScreen(

input logic clk,

input logic [9:0] vga\_x,

input logic [9:0] vga\_y,

output logic [1:0] out

);

logic [1:0] screen;

always\_ff @(posedge clk)

begin

if(vga\_x > 260 && vga\_x < 300 && vga\_y > 80 && vga\_y < 120) //

screen <= 2'b01;

else if(vga\_x > 360 && vga\_x < 400 && vga\_y > 80 && vga\_y < 120) //

screen <= 2'b01;

else if(vga\_x > 220 && vga\_x < 260 && vga\_y > 280 && vga\_y < 320)

screen <= 2'b00;

else if(vga\_x > 260 && vga\_x < 400 && vga\_y > 240 && vga\_y < 280) //

screen <= 2'b01;

else if(vga\_x > 400 && vga\_x < 440 && vga\_y > 280 && vga\_y < 320)

screen <= 2'b00;

else if(vga\_x > 220 && vga\_x < 260 && vga\_y > 200 && vga\_y < 240)

screen <= 2'b10;

else if(vga\_x > 400 && vga\_x < 440 && vga\_y > 200 && vga\_y < 240)

screen <= 2'b10;

else

screen <= 2'b11;

end

assign out = {screen};

endmodule

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Background.sv

`timescale 1ns / 1ps

module Background(

input logic clk, dir,

input logic [9:0] vga\_x,

input logic [9:0] vga\_y,

input logic [9:0] gate\_x,

input logic [9:0] gate\_y,

input logic [9:0] gate\_height,

input logic [9:0] gate\_width,

input logic [9:0] cursor\_x,

input logic [9:0] cursor\_y,

output logic [1:0] out

);

logic [1:0] background;

always\_ff @ (posedge clk)

begin

if(vga\_x >= gate\_x - gate\_width/2 && vga\_x <= gate\_x + gate\_width/2 && vga\_y >= gate\_y - gate\_height/2 && vga\_y <= gate\_y + gate\_height/2 )

background <= 2'b00;

else if(vga\_x >= cursor\_x - 1 && vga\_y >= cursor\_y - 1 && vga\_y <= cursor\_y + 1 && vga\_x <= cursor\_x + 1 )

background <= 2'b01;

else if(vga\_x == cursor\_x && vga\_y <= cursor\_y + 10 && vga\_y >= cursor\_y - 10)

background <= 2'b10;

else if(vga\_y == cursor\_y && vga\_x <= cursor\_x + 10 && vga\_x >= cursor\_x - 10)

background <= 2'b10;

else

background <= 2'b11;

case(dir)

1'b1:

begin

if(vga\_x >= gate\_x + gate\_width/2 && vga\_x <= gate\_x + 30 && vga\_y <= gate\_y + 10 && vga\_y >= gate\_y + 9)

background <= 2'b00;

if(vga\_x >= gate\_x + gate\_width/2 && vga\_x <= gate\_x + 30 && vga\_y >= gate\_y - 10 && vga\_y <= gate\_y - 9)

background <= 2'b00;

if(vga\_x <= gate\_x + gate\_width/2 && vga\_x >= gate\_x && vga\_y <= gate\_y + gate\_height/2 + 5 && vga\_y >= gate\_y - gate\_height/2 - 5)

background <= 2'b00;

if(vga\_x <= gate\_x && vga\_x >= gate\_x - gate\_width/2 + 4 && vga\_y <= gate\_y + gate\_height/2 + 4 && vga\_y >= gate\_y - gate\_height/2 - 4)

background <= 2'b00;

if(vga\_x <= gate\_x - 1 && vga\_x >= gate\_x - gate\_width/2 + 3 && vga\_y <= gate\_y + gate\_height/2 + 3 && vga\_y >= gate\_y - gate\_height/2 - 3)

background <= 2'b00;

if(vga\_x <= gate\_x - 2 && vga\_x >= gate\_x - gate\_width/2 + 2 && vga\_y <= gate\_y + gate\_height/2 + 2 && vga\_y >= gate\_y - gate\_height/2 - 2)

background <= 2'b00;

if(vga\_x <= gate\_x - 3 && vga\_x >= gate\_x - gate\_width/2 + 1 && vga\_y <= gate\_y + gate\_height/2 + 1 && vga\_y >= gate\_y - gate\_height/2 - 1)

background <= 2'b00;

if(vga\_x <= gate\_x - 4 && vga\_x >= gate\_x - gate\_width/2 && vga\_y <= gate\_y + gate\_height/2 && vga\_y >= gate\_y - gate\_height/2 )

background <= 2'b00;

end

1'b0:

begin

if(vga\_x <= gate\_x - gate\_width/2 && vga\_x >= gate\_x - 30 && vga\_y <= gate\_y + 10 && vga\_y >= gate\_y + 9)

background <= 2'b00;

if(vga\_x <= gate\_x - gate\_width/2 && vga\_x >= gate\_x - 30 && vga\_y >= gate\_y - 10 && vga\_y <= gate\_y - 9)

background <= 2'b00;

if(vga\_x >= gate\_x - gate\_width/2 && vga\_x <= gate\_x && vga\_y <= gate\_y + gate\_height/2 + 5 && vga\_y >= gate\_y - gate\_height/2 - 5)

background <= 2'b00;

if(vga\_x >= gate\_x && vga\_x <= gate\_x + gate\_width/2 - 4 && vga\_y <= gate\_y + gate\_height/2 + 4 && vga\_y >= gate\_y - gate\_height/2 - 4)

background <= 2'b00;

if(vga\_x >= gate\_x + 1 && vga\_x <= gate\_x + gate\_width/2 - 3 && vga\_y <= gate\_y + gate\_height/2 + 3 && vga\_y >= gate\_y - gate\_height/2 - 3)

background <= 2'b00;

if(vga\_x >= gate\_x + 2 && vga\_x <= gate\_x + gate\_width/2 - 2 && vga\_y <= gate\_y + gate\_height/2 + 2 && vga\_y >= gate\_y - gate\_height/2 - 2)

background <= 2'b00;

if(vga\_x >= gate\_x + 3 && vga\_x <= gate\_x + gate\_width/2 - 1 && vga\_y <= gate\_y + gate\_height/2 + 1 && vga\_y >= gate\_y - gate\_height/2 - 1)

background <= 2'b00;

if(vga\_x >= gate\_x + 4 && vga\_x <= gate\_x + gate\_width/2 && vga\_y <= gate\_y + gate\_height/2 && vga\_y >= gate\_y - gate\_height/2 )

background <= 2'b00;

end

endcase

end

assign out = background;

endmodule

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Color.sv

`timescale 1ns / 1ps

module Color(

input logic clk, reset, start,

input logic [1:0] background,

input logic vidOn, isGameOver, isWon,

input logic [1:0] screen,

input logic [1:0] screen2,

output logic [3:0] red,

output logic [3:0] green,

output logic [3:0] blue

);

always\_ff @(posedge clk)

begin

if(vidOn)

if(start)

begin

if(~isGameOver && ~isWon)

case(background)

2'b00:

begin

red <= 4'b0000;

green <= 4'b0000;

blue <= 4'b1111;

end

2'b01:

begin

red <= 4'b1111;

green <= 4'b0000;

blue <= 4'b0000;

end

2'b10:

begin

red <= 4'b1111;

green <= 4'b1111;

blue <= 4'b1111;

end

2'b11:

begin

red <= 4'b0000;

green <= 4'b0000;

blue <= 4'b0000;

end

endcase

else if(isWon)

begin

case (screen)

2'b10:

begin

red <= 4'b0000;

green <= 4'b1111;

blue <= 4'b0000;

end

2'b01:

begin

red <= 4'b0000;

green <= 4'b1111;

blue <= 4'b0000;

end

2'b11:

begin

red <= 4'b0000;

green <= 4'b0000;

blue <= 4'b0000;

end

endcase

end

else if(isGameOver)

begin

case (screen)

2'b00:

begin

red <= 4'b1111;

green <= 4'b0000;

blue <= 4'b0000;

end

2'b01:

begin

red <= 4'b1111;

green <= 4'b0000;

blue <= 4'b0000;

end

2'b11:

begin

red <= 4'b0000;

green <= 4'b0000;

blue <= 4'b0000;

end

endcase

end

end

else

begin

case(screen2)

2'b00:

begin

red <= 4'b1000;

green <= 4'b0000;

blue <= 4'b0000;

end

2'b01:

begin

red <= 4'b0000;

green <= 4'b1000;

blue <= 4'b0000;

end

2'b11:

begin

red <= 4'b0000;

green <= 4'b0000;

blue <= 4'b0000;

end

endcase

end

else

begin

red <= 4'b0000;

green <= 4'b0000;

blue <= 4'b0000;

end

end

endmodule

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# SevSeg\_4digit.sv

`timescale 1ns / 1ps

module SevSeg\_4digit(

input clk,

input [3:0] in0, in1, in2, in3, // 4 values for 4 digits (decimal value)

output a, b, c, d, e, f, g, dp, //individual LED output for the 7-segment along with the digital point

output [3:0] an // anode: 4-bit enable signal (active low)

);

// divide system clock (100Mhz for Basys3) by 2^N using a counter, which allows us to multiplex at lower speed

localparam N = 18;

logic [N-1:0] count = {N{1'b0}}; //initial value

always@ (posedge clk)

count <= count + 1;

logic [3:0]digit\_val; // 7-bit register to hold the current data on output

logic [3:0]digit\_en; //register for enable vector

always\_comb

begin

digit\_en = 4'b1111; //default

digit\_val = in0; //default

case(count[N-1:N-2]) //using only the 2 MSB's of the counter

2'b00 : //select first 7Seg.

begin

digit\_val = in0;

digit\_en = 4'b1110;

end

2'b01: //select second 7Seg.

begin

digit\_val = in1;

digit\_en = 4'b1101;

end

2'b10: //select third 7Seg.

begin

digit\_val = in2;

digit\_en = 4'b1011;

end

2'b11: //select forth 7Seg.

begin

digit\_val = in3;

digit\_en = 4'b0111;

end

endcase

end

//Convert digit number to LED vector. LEDs are active low.

logic [6:0] sseg\_LEDs;

always\_comb

begin

sseg\_LEDs = 7'b1111111; //default

case(digit\_val)

4'd0 : sseg\_LEDs = 7'b1000000; //to display 0

4'd1 : sseg\_LEDs = 7'b1111001; //to display 1

4'd2 : sseg\_LEDs = 7'b0100100; //to display 2

4'd3 : sseg\_LEDs = 7'b0110000; //to display 3

4'd4 : sseg\_LEDs = 7'b0011001; //to display 4

4'd5 : sseg\_LEDs = 7'b0010010; //to display 5

4'd6 : sseg\_LEDs = 7'b0000010; //to display 6

4'd7 : sseg\_LEDs = 7'b1111000; //to display 7

4'd8 : sseg\_LEDs = 7'b0000000; //to display 8

4'd9 : sseg\_LEDs = 7'b0010000; //to display 9

default : sseg\_LEDs = 7'b0111111; //dash

endcase

end

assign an = digit\_en;

assign {g, f, e, d, c, b, a} = sseg\_LEDs;

assign dp = 1'b1; //turn dp off

endmodule

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# DotMatrix.sv

module DotMatrix(

input logic clk,

input logic [1:0] a,

output logic oe, //output enable

output logic SH\_CP, // shift register clk pulse

output logic ST\_CP, // store register clk pulse

output logic reset, // reset for the shift register

output logic DS, // digital signal

output logic [7:0] rows

);

logic [24:1] message;

logic [7:0] red;

initial begin

red = 8'hFF;

rows = 8'b00000000;

end

assign message[24:17] = red;

logic f;

logic e;

logic [1:0] counter;

logic [8:0] i = 1; // --data signalin seri olarak iletilmesini kontrol eder.

logic [2:0] a = 0;

logic [9:0] d = 0;

always@(posedge clk)

begin

counter = counter+1;

f<= counter[1]; // clk signal for the shift register

e<= ~f;

end

//------------------------------------------------------------

always@( posedge e)

begin

i = i+9'b000000001;

end

always@(\*)

begin

if (i < 9'b000000100)

reset<=0;

else

reset<=1;

if (i>9'b000000011 && i<9'b000011100)

DS<=message[i-9'b000000011];

else

DS<=0;

if (i<9'b000011100)

begin

SH\_CP<=f;

ST\_CP<=e;

end

else

begin

SH\_CP<=0;

ST\_CP<=1;

end

end

always @(posedge f)

begin

if (i>9'b000011100 && i<9'b110011101)

oe<=0;

else

oe<=1;

end

always @(posedge f)

begin

if (i==(9'b000011100 + 9'b110011101)/2)

begin

a = a;

end

end

always @(posedge f)

begin

if (i==9'b110011110)

if (a==7)

d=d+1;

end

logic[26:0] count=0;

logic clk\_en;

always@(posedge clk)

begin

count<=count+1;

if(count==27'd50000000)

begin

clk\_en=1;

count<=27'd0;

end

else

clk\_en=0;

end

always@(a)

begin

if(clk\_en)

begin

case(a)

2'b00:

begin

rows<=8'b00010000;

red<=8'b11011111;

end

2'b01:

begin

rows<=8'b00011000;

red<=8'b11011111;

end

2'b10:

begin

rows<=8'b00111000;

red<=8'b11011111;

end

endcase

end

end

endmodule

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

# Constraints

# Clock signal

set\_property PACKAGE\_PIN W5 [get\_ports clk]

  set\_property IOSTANDARD LVCMOS33 [get\_ports clk]

# create\_clock -add -name sys\_clk\_pin -period 10.00 -waveform {0 5} [get\_ports clk]

# Switches

set\_property PACKAGE\_PIN V17 [get\_ports reset]

  set\_property IOSTANDARD LVCMOS33 [get\_ports reset]

set\_property PACKAGE\_PIN V16 [get\_ports start]

      set\_property IOSTANDARD LVCMOS33 [get\_ports start]

# LEDs

set\_property PACKAGE\_PIN U16 [get\_ports tempLed]

  set\_property IOSTANDARD LVCMOS33 [get\_ports tempLed]

##Buttons

 set\_property PACKAGE\_PIN U18 [get\_ports fire]

      set\_property IOSTANDARD LVCMOS33 [get\_ports fire]

 set\_property PACKAGE\_PIN T18 [get\_ports up]

      set\_property IOSTANDARD LVCMOS33 [get\_ports up]

 set\_property PACKAGE\_PIN W19 [get\_ports left]

      set\_property IOSTANDARD LVCMOS33 [get\_ports left]

 set\_property PACKAGE\_PIN T17 [get\_ports right]

      set\_property IOSTANDARD LVCMOS33 [get\_ports right]

 set\_property PACKAGE\_PIN U17 [get\_ports down]

      set\_property IOSTANDARD LVCMOS33 [get\_ports down]

##VGA Connector

set\_property PACKAGE\_PIN G19 [get\_ports {red[0]}]

  set\_property IOSTANDARD LVCMOS33 [get\_ports {red[0]}]

set\_property PACKAGE\_PIN H19 [get\_ports {red[1]}]

  set\_property IOSTANDARD LVCMOS33 [get\_ports {red[1]}]

set\_property PACKAGE\_PIN J19 [get\_ports {red[2]}]

  set\_property IOSTANDARD LVCMOS33 [get\_ports {red[2]}]

set\_property PACKAGE\_PIN N19 [get\_ports {red[3]}]

  set\_property IOSTANDARD LVCMOS33 [get\_ports {red[3]}]

set\_property PACKAGE\_PIN N18 [get\_ports {blue[0]}]

  set\_property IOSTANDARD LVCMOS33 [get\_ports {blue[0]}]

set\_property PACKAGE\_PIN L18 [get\_ports {blue[1]}]

    set\_property IOSTANDARD LVCMOS33 [get\_ports {blue[1]}]

set\_property PACKAGE\_PIN K18 [get\_ports {blue[2]}]

  set\_property IOSTANDARD LVCMOS33 [get\_ports {blue[2]}]

set\_property PACKAGE\_PIN J18 [get\_ports {blue[3]}]

  set\_property IOSTANDARD LVCMOS33 [get\_ports {blue[3]}]

set\_property PACKAGE\_PIN J17 [get\_ports {green[0]}]

  set\_property IOSTANDARD LVCMOS33 [get\_ports {green[0]}]

set\_property PACKAGE\_PIN H17 [get\_ports {green[1]}]

  set\_property IOSTANDARD LVCMOS33 [get\_ports {green[1]}]

set\_property PACKAGE\_PIN G17 [get\_ports {green[2]}]

  set\_property IOSTANDARD LVCMOS33 [get\_ports {green[2]}]

set\_property PACKAGE\_PIN D17 [get\_ports {green[3]}]

  set\_property IOSTANDARD LVCMOS33 [get\_ports {green[3]}]

set\_property PACKAGE\_PIN P19 [get\_ports hsync]

  set\_property IOSTANDARD LVCMOS33 [get\_ports hsync]

set\_property PACKAGE\_PIN R19 [get\_ports vsync]

  set\_property IOSTANDARD LVCMOS33 [get\_ports vsync]

#7 ledsment display

set\_property PACKAGE\_PIN W7 [get\_ports leds[6]]

  set\_property IOSTANDARD LVCMOS33 [get\_ports leds[6]]

set\_property PACKAGE\_PIN W6 [get\_ports leds[5]]

  set\_property IOSTANDARD LVCMOS33 [get\_ports leds[5]]

set\_property PACKAGE\_PIN U8 [get\_ports leds[4]]

  set\_property IOSTANDARD LVCMOS33 [get\_ports leds[4]]

set\_property PACKAGE\_PIN V8 [get\_ports {leds[3]}]

  set\_property IOSTANDARD LVCMOS33 [get\_ports leds[3]]

set\_property PACKAGE\_PIN U5 [get\_ports leds[2]]

  set\_property IOSTANDARD LVCMOS33 [get\_ports leds[2]]

set\_property PACKAGE\_PIN V5 [get\_ports leds[1]]

  set\_property IOSTANDARD LVCMOS33 [get\_ports leds[1]]

set\_property PACKAGE\_PIN U7 [get\_ports leds[0]]

  set\_property IOSTANDARD LVCMOS33 [get\_ports leds[0]]

set\_property PACKAGE\_PIN V7 [get\_ports dp]

  set\_property IOSTANDARD LVCMOS33 [get\_ports dp]

set\_property PACKAGE\_PIN U2 [get\_ports anode[3]]

  set\_property IOSTANDARD LVCMOS33 [get\_ports anode[3]]

set\_property PACKAGE\_PIN U4 [get\_ports anode[2]]

  set\_property IOSTANDARD LVCMOS33 [get\_ports anode[2]]

set\_property PACKAGE\_PIN V4 [get\_ports anode[1]]

  set\_property IOSTANDARD LVCMOS33 [get\_ports anode[1]]

set\_property PACKAGE\_PIN W4 [get\_ports anode[0]]

  set\_property IOSTANDARD LVCMOS33 [get\_ports anode[0]]

#Dot matrix

set\_property PACKAGE\_PIN K17 [get\_ports {DS}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {DS}]

##Sch name = JC2

set\_property PACKAGE\_PIN M18 [get\_ports oe]

set\_property IOSTANDARD LVCMOS33 [get\_ports oe]

##Sch name = JC3

set\_property PACKAGE\_PIN N17 [get\_ports {ST\_CP}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {ST\_CP}]

##Sch name = JC4

set\_property PACKAGE\_PIN P18 [get\_ports {SH\_CP}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {SH\_CP}]

##Sch name = JC7

set\_property PACKAGE\_PIN L17 [get\_ports {reset2}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {reset2}]

##Sch name = JC8

set\_property PACKAGE\_PIN A14 [get\_ports {rows[0]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {rows[0]}]

##Sch name = JB2

set\_property PACKAGE\_PIN A16 [get\_ports {rows[1]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {rows[1]}]

##Sch name = JB3

set\_property PACKAGE\_PIN B15 [get\_ports {rows[2]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {rows[2]}]

set\_property PACKAGE\_PIN B16 [get\_ports {rows[3]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {rows[3]}]

##Sch name = JB7

set\_property PACKAGE\_PIN A15 [get\_ports {rows[4]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {rows[4]}]

##Sch name = JB8

set\_property PACKAGE\_PIN A17 [get\_ports {rows[5]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {rows[5]}]

##Sch name = JB9

set\_property PACKAGE\_PIN C15 [get\_ports {rows[6]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {rows[6]}]

##Sch name = JB10

set\_property PACKAGE\_PIN C16 [get\_ports {rows[7]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {rows[7]}]