Lab 6 CSE100 Report

Rafael Delwart

December 8 2023

1 Introduction

For the "Osmosis" game I used the BASYS3 board and a VGA monitor to create an interactive game where the player controls the movements of pixels based on the physical buttons on the BASYS3 board. The game's objective is to isolate red and blue molecules on different sides of a membrane within a time limit. The molecules start stationary and begin moving once the game starts. Players control the membrane's color with buttons, allowing molecules to cross only when the membrane is a specific color. The game design includes generating VGA control signals, manipulating RGB data for pixel representation, and implementing game logic in Verilog. This project challenged my skills in digital logic design, programming, and creative problem-solving in a real-world application.

2 New Modules

2.1 Pixel Address Module

The "PixelAddress" module generates pixel addresses for an active display region. It uses counters (hcount and vcount) for horizontal (hpos) and vertical (vpos) positioning. The horizontal limit (hlim) is set to 799, and the vertical limit (vlim) to 524, establishing the display boundaries. These counters increment with each clock cycle (clk). When hpos reaches hlim, hcount resets, and vcount increments, moving to the next line. This process repeats until vcount also reaches its limit, ensuring every pixel address within the specified dimensions is generated.

2.2 Border Module

The border module is designed to create a green border around a screen. It defines module borders with inputs H and V, representing horizontal and vertical positions, and outputs for vertical, horizontal, and general borders. The code uses these inputs to define four wire variables: TopBord, BotBord, LBord, and RBord, which represent the top, bottom, left, and right borders, respectively. These are defined using specific value ranges for H and V to create an 8-pixel wide border. The border output combines these four variables to signify the presence of any border, while vertborder and horzborder are specific to vertical and horizontal borders, this will be used later to detect the molecules for bouncing. This implementation effectively creates a green border around the screen with a width of 8 pixels.

2.3 Ball State Module

The "BallState" module is a state machine that controls the direction of a ball's movement. It has inputs like clk, frame, go, Vchange, and Hchange, and outputs such as downright1, upright2, upleft3, and downleft4, each representing a direction. The module uses a series of flip-flops (FDRE) to hold the state of the ball. State transitions are based on conditions like go, Vchange (vertical change), and Hchange (horizontal change). Each output corresponds to a specific direction, with conditions determining transitions between states (directions), enabling the ball to move in different directions based on inputs.

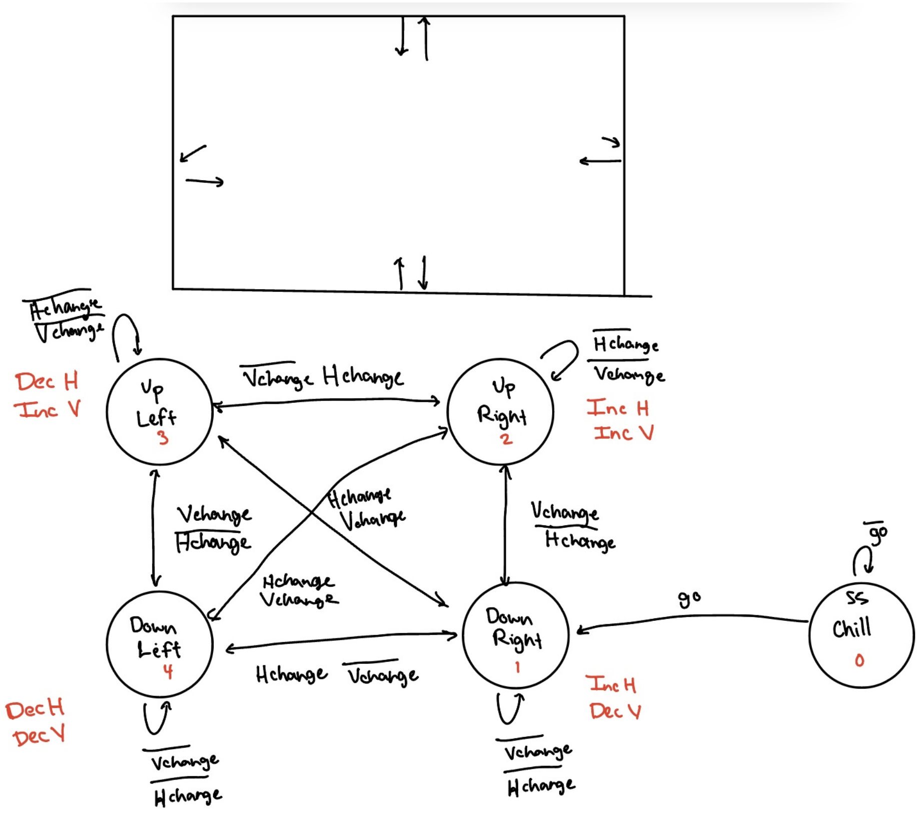


Figure 1: Diagram for the molecule state machine

```
NS[0] = (PS[0] & ~go);
NS[1] = (PS[0] & go) | (PS[1] & ~Vchange & ~Hchange) | (PS[2] & Vchange & ~Hchange) |
        (PS[4] & ~Vchange & Hchange) | (PS[3] & Vchange & Hchange);
NS[2] = (PS[2] & ~Vchange & ~Hchange) | (PS[1] & Vchange & ~Hchange) |
        (PS[3] & ~Vchange & Hchange) | (PS[4] & Vchange & Hchange);
NS[3] = (PS[3] & ~Vchange & ~Hchange) | (PS[4] & Vchange & ~Hchange) |
        (PS[2] & ~Vchange & Hchange) | (PS[1] & Vchange & Hchange);
NS[4] = (PS[4] & ~Vchange & ~Hchange) | (PS[3] & Vchange & ~Hchange) |
        (PS[1] & ~Vchange & Hchange) | (PS[2] & Vchange & Hchange);
down_right_1 = (PS[1] & ~Vchange & ~Hchange) | (PS[2] & Vchange & ~Hchange) |
               (PS[4] & ~Vchange & Hchange) | (PS[3] & Vchange & Hchange);
up_right_2 = (PS[2] & ~Vchange & ~Hchange) | (PS[1] & Vchange & ~Hchange) |
             (PS[3] & ~Vchange & Hchange) | (PS[4] & Vchange & Hchange);
up_left_3 = (PS[3] & ~Vchange & ~Hchange) | (PS[4] & Vchange & ~Hchange) |
            (PS[2] & ~Vchange & Hchange) | (PS[1] & Vchange & Hchange);
down_left_4 = (PS[4] & ~Vchange & ~Hchange) | (PS[3] & Vchange & ~Hchange) |
              (PS[1] & ~Vchange & Hchange) | (PS[2] & Vchange & Hchange);
intermed = PS[0];
```

2.4 Ball Module

The "Ball" module in my code controls the position of a ball on a screen. It takes inputs from the "BallState" module (directions like downright1, upright2, etc.) and uses these to manipulate two counters (ballhpos and ballvpos) that track the ball's horizontal and vertical positions (hposout and vposout). The module adjusts these counters based on the ball's current direction of movement, effectively changing its position on the screen. The Ball output signal is activated when the ball's position intersects with specific coordinates (H, V), indicating the ball's presence at that location.

2.5 Game State Machine Module

The "TopState" module is a top-level state machine managing the main functionalities of a game. It transitions between various states Chill, Setup, Play Game, Win, and Lose based on inputs such as go, timeup8, and gametimeup. The module controls outputs like loadgametime, reset8timer, displaygametime, and others, to manage game events and displays. State changes are governed by specific conditions, enabling the game to respond dynamically to player interactions and game progress, effectively orchestrating the overall flow and rules of the game.

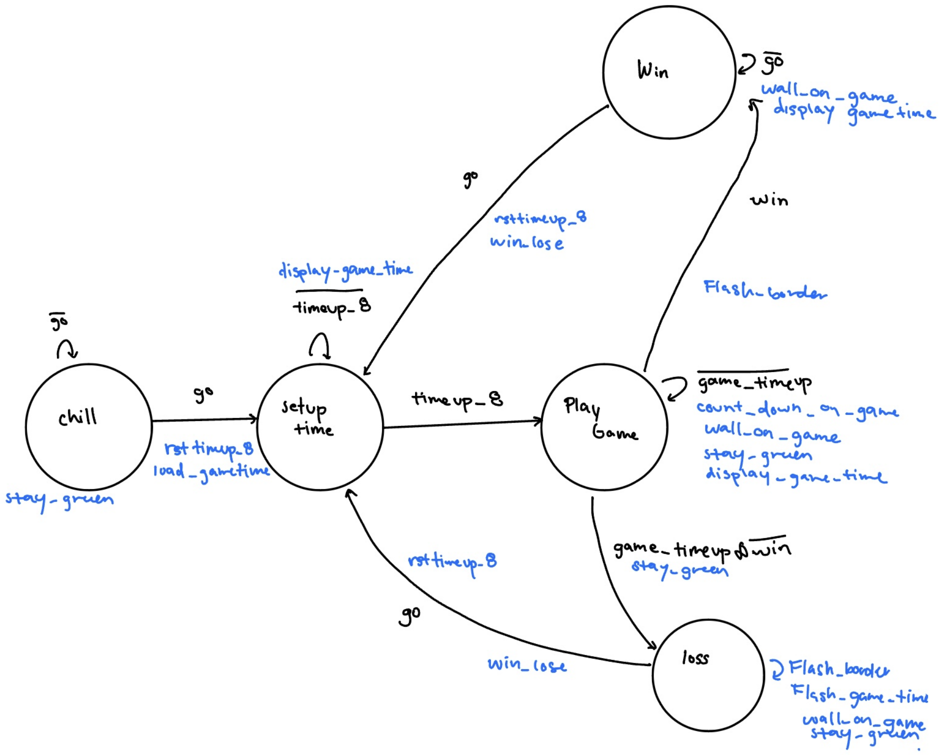


Figure 2: Diagram for the top game state machine

```
NS[0] = (PS[0] & ~go)
NS[1] = (PS[0] & go) | (PS[1] & ~timeup_8) | (PS[3] & go) | (PS[4] & go)
NS[2] = (PS[2] & ~game_timeup & ~win) | (PS[1] & timeup_8)
NS[3] = (PS[2] & win) | (PS[3] & ~go)
NS[4] = (PS[2] & game_timeup) | (PS[4] & ~go)
load_game_time = (PS[0] & go)
reset_8_timer = (PS[0] & go) | (PS[3] & go) | (PS[4] & go)
Display_game_time = PS[1] | PS[2] | PS[3] | PS[4]
Flash_border = PS[1] | PS[3]
Flash_game_time = PS[4]
wall_on_game = PS[2] | PS[3] | PS[4]
count_down_game = PS[2]
stay_green = PS[0] | PS[2] | PS[4]
win_lose = PS[3] | PS[4]
reset_ball_loc = (PS[3] & go) | (PS[4] & go)
```

2.6 Top Level Module

The "TopLevel" module serves as the central hub, integrating various components of a game. It includes inputs like buttons, switches, and outputs for VGA display and LEDs. Key elements like pixel addressing, ball movement, and game state management are interconnected. The module coordinates clock signals, processes player inputs, manages game states (win, lose, play), and controls the visual elements (borders, ball positions, etc.) on the VGA display. It effectively orchestrates the overall game logic, user interactions, and display management.

3 Timing and Testing Simulations

I tested the vsync and hsync using the given simulation file, I adjusted my code to find the right active region based on the error variable given in the simulation. In testing my "Osmosis" game design, I focused on thorough simulations. I chose to simulate btnU to start the game. Testing these inputs was crucial for ensuring the game responded accurately to player interactions. I paid special attention to corner cases, like molecules reaching the screen edges or rapid consecutive button presses, which could potentially disrupt game mechanics. However, to see these things happen in the simulation would be impossible due to the time it takes for a molecule to bounce and how long it takes to simulate even a second of real-time on Vivado. Instead, to test things that I couldn't see in the simulations, I hooked up certain output wires to the LEDs. During testing, I discovered a few issues. The counters for the ball location would not switch when it interacted with the wall I resolved this by adjusting timing constraints within the state machine module. Another challenge was getting the state machine to function correctly, again for this I outputted my current state from the state machine and wired that to the LEDs, this allowed me to see what state I was in, in real-time. Using the led technique I was able to observe that I was entering two states, this allowed me to quickly solve my problem, by adjusting the conditions that made me change states. As you can see in Figure 5 the worst negative stack is $29.433ns$ and from Figure 4 the clock period is $40ns$ thus the minimum clock period is calculated.

$$40ns - 29.433ns = 10.567ns$$

To find the maximum clock frequency we take the reciprocal

$$\frac{1}{10.567ns} = 94.6Mhz$$

Thus the maximum frequency is $94.6Mhz$

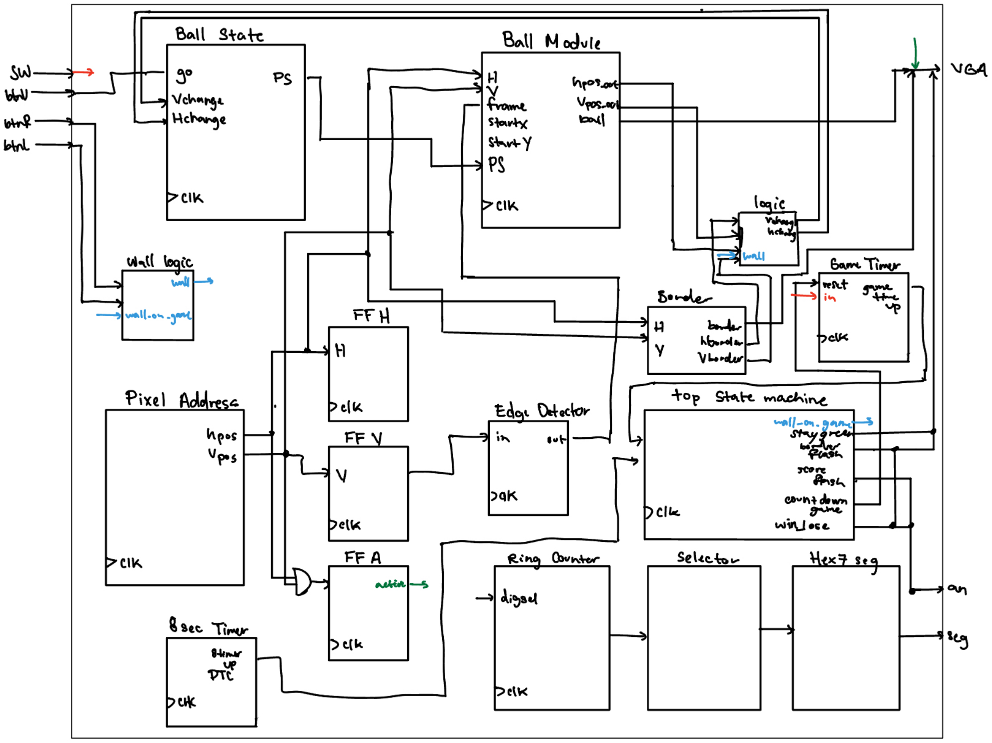
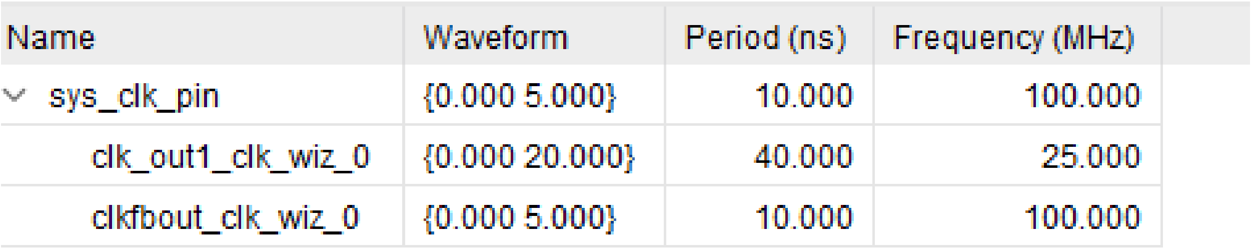


Figure 3: Wiring diagram for the top level

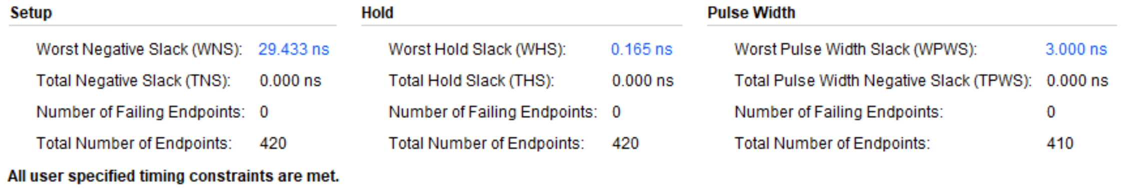
Appendix

Design Summaries



Name	Waveform	Period (ns)	Frequency (MHz)
sys_clk_pin	{0.000 5.000}	10.000	100.000
clk_out1_clk_wiz_0	{0.000 20.000}	40.000	25.000
clkfbout_clk_wiz_0	{0.000 5.000}	10.000	100.000

Figure 4: Clock Summary



Setup	
Worst Negative Slack (WNS):	29.433 ns
Total Negative Slack (TNS):	0.000 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	420

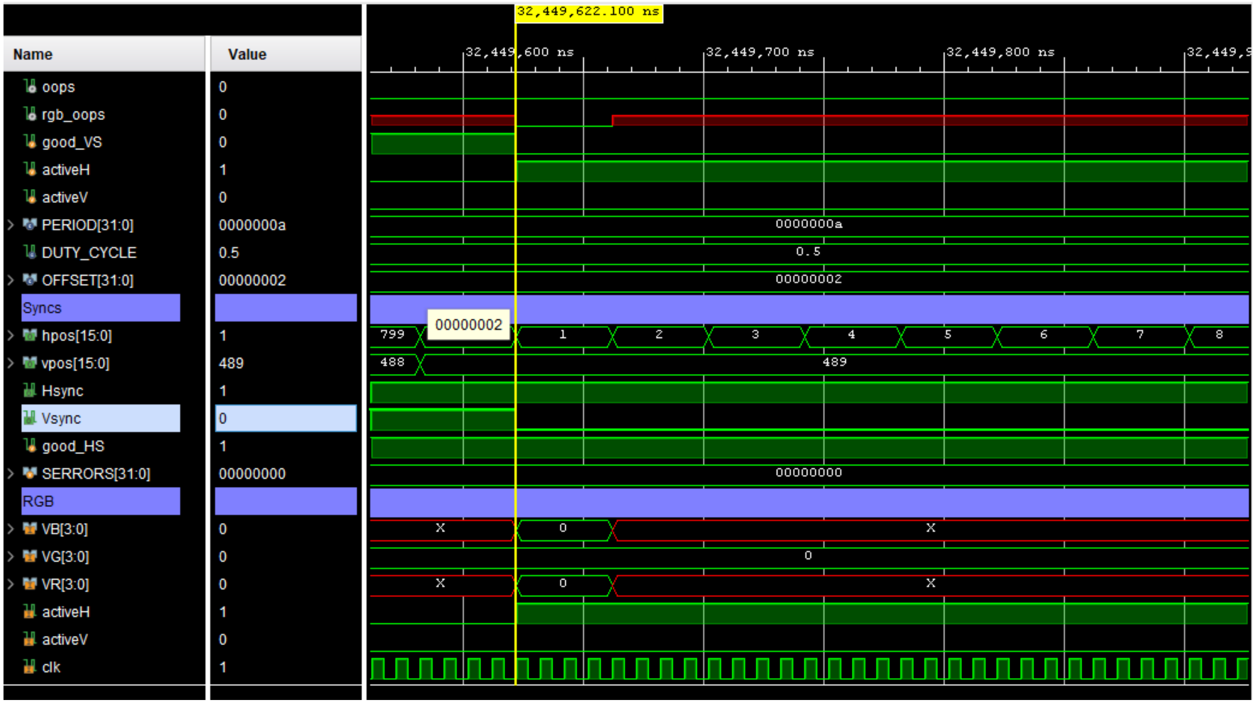
Hold	
Worst Hold Slack (WHS):	0.165 ns
Total Hold Slack (THS):	0.000 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	420

Pulse Width	
Worst Pulse Width Slack (WPWS):	3.000 ns
Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	410

All user specified timing constraints are met.

Figure 5: Design Timing Summary

Waveforms



Name	Value
oops	0
rgb_oops	0
good_VS	0
activeH	1
activeV	0
PERIOD[31:0]	0000000a
DUTY_CYCLE	0.5
OFFSET[31:0]	00000002
Syncs	
hpos[15:0]	1
vpos[15:0]	489
Hsync	1
Vsync	0
good_HS	1
SERRORS[31:0]	00000000
RGB	
VB[3:0]	0
VG[3:0]	0
VR[3:0]	0
activeH	1
activeV	0
clk	1

Figure 6: Vsync Waveform Low Zoom

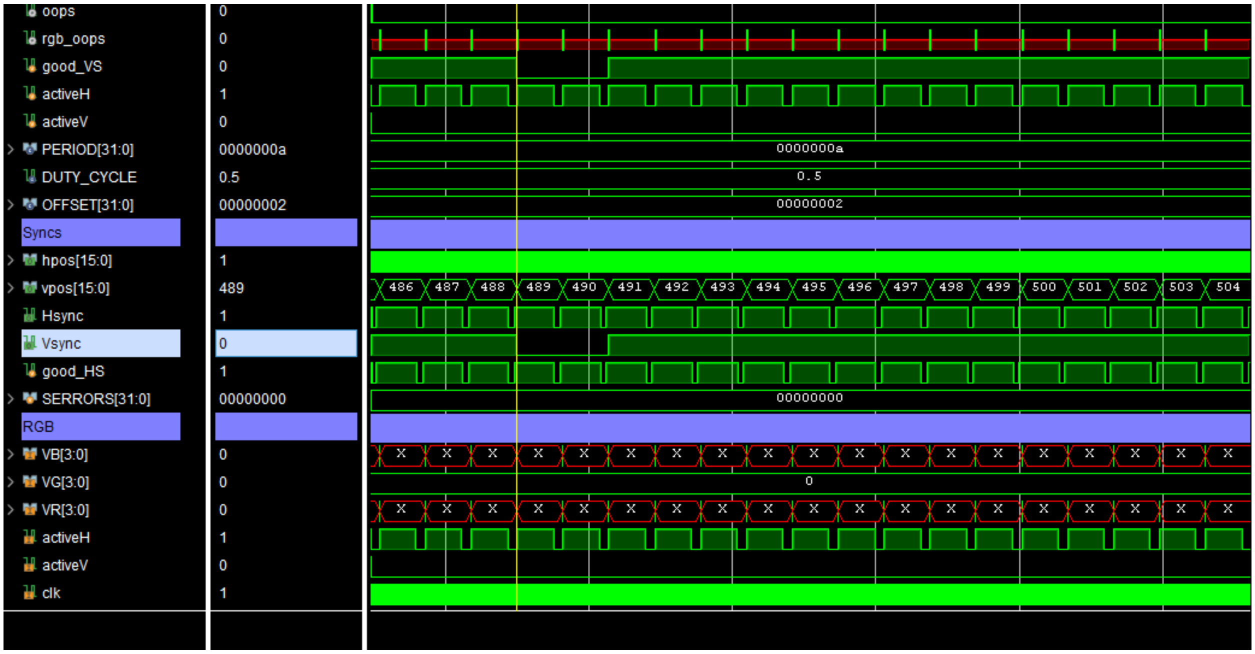


Figure 7: Vsync Waveform

Verilog Code

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 11/16/2023 01:00:02 PM
// Design Name:
// Module Name: PixelAddress
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module PixelAddress(
    input clk,
    output [15:0]hpos,
    output [15:0]vpos
    );
    wire [15:0]hlim;
    wire [15:0]vlim;
    wire [15:0]reset_val;
    assign hlim = 16'd799;
    assign vlim = 16'd524;
    assign reset_val = 16'd0;
    counterUD16L hcount(.clk(clk), .Din(reset_val), .Dw(1'b0), .LD(hpos == hlim),
.Up(1'b1) , .Q(hpos));
    counterUD16L vcount(.clk(clk), .Din(reset_val), .Dw(1'b0), .LD((hpos ==
hlim)&(vpos==vlim)), .Up(hpos == hlim), .Q(vpos));
endmodule
```

```verilog
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: 
// Engineer: 
// 
// Create Date: 11/17/2023 11:02:30 AM
// Design Name: 
// Module Name: borders
// Project Name: 
// Target Devices: 
// Tool Versions: 
// Description: 
// 
// Dependencies: 
// 
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
// 
//////////////////////////////////////////////////////////////////////////////////


module borders(
    input [15:0] H,
    input [15:0] V,
    output vert_border,
    output horz_border,
    output border
    );
    wire TopBord, BotBord, LBord, RBord;
    assign TopBord = (H >= 16'd8) & (H <= 16'd631) & (V >= 16'd0) & (V <= 16'd7);
    assign BotBord = (H >= 16'd8) & (H <= 16'd631) & (V >= 16'd471) & (V <= 16'd479);
    assign LBord = (H >= 16'd0) & (H <= 16'd8) & (V >= 16'd0) & (V <= 16'd479);
    assign RBord = (H >= 16'd632) & (H <= 16'd639) & (V >= 16'd0) & (V <= 16'd479);
    assign border = TopBord | BotBord | LBord | RBord;
    assign vert_border = TopBord | BotBord;
    assign horz_border = LBord | RBord;

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 11/20/2023 01:06:46 PM
// Design Name:
// Module Name: BallState
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module BallState(
    input clk,
    input frame,
    input go,
    input Vchange,
    input Hchange,
    output down_right_1,
    output up_right_2,
    output up_left_3,
    output down_left_4,
    output [4:0]PS,
    output intermed
    );
    wire[4:0] NS, PS;
    FDRE #(.INIT(1'b1)) Q0s(.C(clk), .R(1'b0), .CE(1'b1), .D(NS[0]), .Q(PS[0]));
//Chill 0
    FDRE #(.INIT(1'b0)) Q1s(.C(clk), .R(1'b0), .CE(1'b1), .D(NS[1]), .Q(PS[1]));
//Down Right 1
    FDRE #(.INIT(1'b0)) Q2s(.C(clk), .R(1'b0), .CE(1'b1), .D(NS[2]), .Q(PS[2]));
//Up Right 2
    FDRE #(.INIT(1'b0)) Q3s(.C(clk), .R(1'b0), .CE(1'b1), .D(NS[3]), .Q(PS[3]));
//Up Left 3
    FDRE #(.INIT(1'b0)) Q4s(.C(clk), .R(1'b0), .CE(1'b1), .D(NS[4]), .Q(PS[4]));
//Down Left 4
```

```
    assign NS[0] = (PS[0] & ~go);
//    assign NS[1] = (PS[0] & go) | (PS[1] & ~Vchange & ~Hchange);
    assign NS[1] = (PS[0] & go) | (PS[1] & ~Vchange & ~Hchange) | (PS[2] & Vchange &
~Hchange) | (PS[4] & ~Vchange & Hchange) | (PS[3] & Vchange & Hchange);
//    assign NS[2] = (PS[2] & ~Vchange & ~Hchange) | (PS[1] & Vchange & ~Hchange) |
(PS[3] & ~Vchange & Hchange) | (PS[4] & Vchange & Hchange);
    assign NS[2] = (PS[2] & ~Vchange & ~Hchange) | (PS[1] & Vchange & ~Hchange) |
(PS[3] & ~Vchange & Hchange) | (PS[4] & Vchange & Hchange);


    assign NS[3] = (PS[3] & ~Vchange & ~Hchange) | (PS[4] & Vchange & ~Hchange) |
(PS[2] & ~Vchange & Hchange) | (PS[1] & Vchange & Hchange);
    assign NS[4] = (PS[4] & ~Vchange & ~Hchange) | (PS[3] & Vchange & ~Hchange) |
(PS[1] & ~Vchange & Hchange) | (PS[2] & Vchange & Hchange);
    assign down_right_1 = (PS[1] & ~Vchange & ~Hchange) | (PS[2] & Vchange &
~Hchange) | (PS[4] & ~Vchange & Hchange) | (PS[3] & Vchange & Hchange);
    assign up_right_2 = (PS[2] & ~Vchange & ~Hchange) | (PS[1] & Vchange & ~Hchange)
| (PS[3] & ~Vchange & Hchange) | (PS[4] & Vchange & Hchange);
    assign up_left_3 = (PS[3] & ~Vchange & ~Hchange) | (PS[4] & Vchange & ~Hchange)
| (PS[2] & ~Vchange & Hchange) | (PS[1] & Vchange & Hchange);
    assign down_left_4 = (PS[4] & ~Vchange & ~Hchange) | (PS[3] & Vchange &
~Hchange) | (PS[1] & ~Vchange & Hchange) | (PS[2] & Vchange & Hchange);
    assign intermed = PS[0];


endmodule
```

```
module Ball(
    input frame,
    input clk,
    input [15:0] H,
    input [15:0] V,
    input [15:0] starting_pix_x,
    input [15:0] starting_pix_y,
    input down_right_1,
    input up_right_2,
    input up_left_3,
    input down_left_4,
    input intermed,
    output [15:0] hpos_out,
    output [15:0] vpos_out,
    output Ball
    );
    wire up_x, up_y, dw_x, dw_y;
    assign up_x = ((down_right_1 | up_right_2) & frame);
    assign up_y = ((down_right_1 | down_left_4) & frame);
    assign dw_x = ((down_left_4 | up_left_3) & frame);
    assign dw_y = ((up_left_3 | up_right_2) & frame);
    counterUD16L ball_hpos(.clk(clk), .Din(starting_pix_x), .Dw(dw_x),
.LD(intermed), .Up(up_x), .Q(hpos_out));
    counterUD16L ball_vpos(.clk(clk), .Din(starting_pix_y), .Dw(dw_y),
.LD(intermed), .Up(up_y), .Q(vpos_out));
    assign Ball = ((hpos_out <= H) & (H <= (hpos_out + 16'd15))) & ((vpos_out <= V)
& (V <= (vpos_out + 16'd15)));



endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: 
// Engineer: 
// 
// Create Date: 11/25/2023 06:14:19 PM
// Design Name: 
// Module Name: TopState
// Project Name: 
// Target Devices: 
// Tool Versions: 
// Description: 
// 
// Dependencies: 
// 
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
// 
//////////////////////////////////////////////////////////////////////////////////


module TopState(
    input go,
    input clk,
    input timeup_8,
    input game_timeup,
    input win,
    output load_game_time,
    output reset_8_timer,
    output Display_game_time,
    output Flash_border,
    output Flash_game_time,
    output [4:0]led,
    output count_down_game,
    output stay_green,
    output win_lose,
    output reset_ball_loc,
    output wall_on_game
    );
    wire[4:0] NS, PS;
    FDRE #(.INIT(1'b1)) Q0s(.C(clk), .R(1'b0), .CE(1'b1), .D(NS[0]), .Q(PS[0]));
//Chill 0
    FDRE #(.INIT(1'b0)) Q1s(.C(clk), .R(1'b0), .CE(1'b1), .D(NS[1]), .Q(PS[1]));
//Setup 1
    FDRE #(.INIT(1'b0)) Q2s(.C(clk), .R(1'b0), .CE(1'b1), .D(NS[2]), .Q(PS[2]));
```

```
//Play Game 2
    FDRE #(.INIT(1'b0)) Q3s(.C(clk), .R(1'b0), .CE(1'b1), .D(NS[3]), .Q(PS[3]));
//Win 3
    FDRE #(.INIT(1'b0)) Q4s(.C(clk), .R(1'b0), .CE(1'b1), .D(NS[4]), .Q(PS[4]));
//Lose 4
    assign led = PS;
    assign NS[0] = (PS[0] & ~go);
    assign NS[1] = (PS[0] & go) | (PS[1] & ~timeup_8) | (PS[3] & go) | (PS[4] & go);
    assign NS[2] = (PS[2] & ~game_timeup & ~win) | (PS[1] & timeup_8);
    assign NS[3] = (PS[2] & win) | (PS[3] & ~go);
    assign NS[4] = (PS[2] & game_timeup) | (PS[4] & ~go);
    assign load_game_time = (PS[0] & go);
    assign reset_8_timer = (PS[0] & go) | (PS[3] & go) | (PS[4] & go);
    assign Display_game_time = PS[1] | PS[2] | PS[3] | PS[4];
    assign Flash_border  = PS[1] | PS[3];
    assign Flash_game_time = PS[4];
    assign wall_on_game = PS[2] | PS[3] | PS[4];
    assign count_down_game = PS[2];
    assign stay_green = PS[0] | PS[2] | PS[4];
    assign win_lose = PS[3] | PS[4];
    assign reset_ball_loc = (PS[3] & go) | (PS[4] & go);

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 11/16/2023 12:58:14 PM
// Design Name:
// Module Name: TopLevel
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module TopLevel(
    input btnU,
    input btnD,
    input btnR,
    input btnC,
    input btnL,
    input [15:0] sw,
    input clkin,
    input dp,
    output Hsync,
    output Vsync,
    output [3:0]vgaRed,
    output [3:0]vgaGreen,
    output [3:0]vgaBlue,
    output [15:0] led,
    output [3:0] an,
    output [6:0] seg
    );
    wire [15:0] hpos, vpos;
    wire clk, digsel;
    //clock
    labVGA_clks clock(.clkin(clkin), .greset(btnC), .clk(clk), .digsel(digsel));
    //current pixel location
    PixelAddress pixelAdd(.clk(clk), .hpos(hpos), .vpos(vpos));
    wire active_region;
    //syncs and active region
    FDRE #(.INIT(1'b0)) FF_A (.C(clk), .CE(1'b1), .D((hpos&&vpos)), .Q(active_region)
    FDRE #(.INIT(1'b1)) FF_H (.C(clk), .CE(1'b1), .D(~((hpos >= 16'd655) & (hpos <=
16'd750))), .Q(Hsync));
    FDRE #(.INIT(1'b1)) FF_V (.C(clk), .CE(1'b1), .D(~((vpos >= 16'd489) & (vpos <=
16'd490))), .Q(Vsync));

    //green border
    wire border, horz_border, vert_border;
    borders Bs(.H(hpos), .V(vpos), .border(border), .horz_border(horz_border),
.vert_border(vert_border));

    //frame clock for moving the ball
    wire frame;
    assign frame = ((hpos == 16'd640) & (vpos == 16'd480));

    wire load_game_time, wall_on_game, win_lose, reset_ball_loc;
    //wire controlled by top sm for wehter or not wall is on
    wire wall_on, wall_off;
    wire red_wall_on;
    wire Display_game_time;
    assign red_wall_on = btnL & ~win_lose;
    //first red ball
    wire red_ball_1;
    wire [15:0] starting_pix_x_red_1, starting_pix_y_red_1;
    assign starting_pix_x_red_1 = 16'd230;
    assign starting_pix_y_red_1 = 16'd230;
    wire Hchange_red_1 , Vchange_red_1;
    wire down_right_red_1, down_left_red_1, up_right_red_1, up_left_red_1,
intermed_red_1;
    wire [15:0] hpos_red_1, vpos_red_1;
    wire [4:0] PS_r_1;
    BallState redball_1_state(.clk(clk), .go(load_game_time), .frame(frame),
.Hchange(Hchange_red_1), .Vchange(Vchange_red_1),
     .down_right_1(down_right_red_1),
     .up_right_2(up_right_red_1),
     .up_left_3(up_left_red_1),
     .down_left_4(down_left_red_1),
     .intermed(intermed_red_1),
     .PS(PS_r_1));
    Ball redball_1(.clk(clk), .frame(frame), .H(hpos), .V(vpos),
     .down_right_1(down_right_red_1),
     .up_right_2(up_right_red_1),
     .up_left_3(up_left_red_1),
     .down_left_4(down_left_red_1),
     .intermed(reset_ball_loc | intermed_red_1),
     .starting_pix_x(starting_pix_x_red_1), .starting_pix_y(starting_pix_y_red_1),
.hpos_out(hpos_red_1), .vpos_out(vpos_red_1), .Ball(red_ball_1));
    assign Hchange_red_1 = ((hpos_red_1 == 16'd8) | (hpos_red_1 == 16'd615) |
((hpos_red_1 == 16'd300) & ((red_wall_on | wall_on) & ~wall_off & wall_on_game &
(PS_r_1[1] | PS_r_1[2]))) | ((hpos_red_1 == 16'd323)& ((red_wall_on | wall_on) &
wall_on_game & ~wall_off & (PS_r_1[3] | PS_r_1[4]) ))) & frame;
    assign Vchange_red_1 = ((vpos_red_1 == 16'd8) | (vpos_red_1 == 16'd455)) & frame
    wire red_1_left, red_1_right;
    assign red_1_left = (hpos_red_1 <= 16'd284);
    assign red_1_right = (hpos_red_1 >= 16'd323);

    //second red ball
    wire red_ball_2;
    wire [15:0] starting_pix_x_red_2, starting_pix_y_red_2;
    assign starting_pix_x_red_2 = 16'd400;
    assign starting_pix_y_red_2 = 16'd420;
    wire Hchange_red_2 , Vchange_red_2;
    wire down_right_red_2, down_left_red_2, up_right_red_2, up_left_red_2,
intermed_red_2;
    wire [15:0] hpos_red_2, vpos_red_2;
    wire [4:0] PS_r_2;
    BallState redball_2_state(.clk(clk), .go(load_game_time), .frame(frame),
.Hchange(Hchange_red_2), .Vchange(Vchange_red_2),
     .down_right_1(down_right_red_2),
     .up_right_2(up_right_red_2),
     .up_left_3(up_left_red_2),
     .down_left_4(down_left_red_2),
     .intermed(intermed_red_2),
     .PS(PS_r_2));
    Ball redball_2(.clk(clk), .frame(frame), .H(hpos), .V(vpos),
     .down_right_1(down_right_red_2),
     .up_right_2(up_right_red_2),
     .up_left_3(up_left_red_2),
     .down_left_4(down_left_red_2),
     .intermed(reset_ball_loc | intermed_red_2),
     .starting_pix_x(starting_pix_x_red_2), .starting_pix_y(starting_pix_y_red_2),
.hpos_out(hpos_red_2), .vpos_out(vpos_red_2), .Ball(red_ball_2));
    assign Hchange_red_2 = ((hpos_red_2 == 16'd8) | (hpos_red_2 == 16'd615) |
((hpos_red_2 == 16'd300) & ((red_wall_on | wall_on) & ~wall_off & wall_on_game &
(PS_r_2[1] | PS_r_2[2]))) | ((hpos_red_2 == 16'd323)& ((red_wall_on | wall_on) &
wall_on_game & ~wall_off & (PS_r_2[3] | PS_r_2[4]) ))) & frame;
    assign Vchange_red_2 = ((vpos_red_2 == 16'd8) | (vpos_red_2 == 16'd455)) & frame;
    wire red_2_left, red_2_right;
    assign red_2_left = (hpos_red_2 <= 16'd284);
    assign red_2_right = (hpos_red_2 >= 16'd323);
    //third red ball
    wire red_ball_3;
    wire [15:0] starting_pix_x_red_3, starting_pix_y_red_3;
    assign starting_pix_x_red_3 = 16'd150;
    assign starting_pix_y_red_3 = 16'd100;
    wire Hchange_red_3 , Vchange_red_3;
    wire down_right_red_3, down_left_red_3, up_right_red_3, up_left_red_3,
intermed_red_3;
    wire [15:0] hpos_red_3, vpos_red_3;
    wire [4:0] PS_r_3;
    BallState redball_3_state(.clk(clk), .go(load_game_time), .frame(frame),
.Hchange(Hchange_red_3), .Vchange(Vchange_red_3),
     .down_right_1(down_right_red_3),
     .up_right_2(up_right_red_3),
     .up_left_3(up_left_red_3),
     .down_left_4(down_left_red_3),
     .intermed(intermed_red_3),
     .PS(PS_r_3));
    Ball redball_3(.clk(clk), .frame(frame), .H(hpos), .V(vpos),
     .down_right_1(down_right_red_3),
     .up_right_2(up_right_red_3),
     .up_left_3(up_left_red_3),
     .down_left_4(down_left_red_3),
     .intermed(reset_ball_loc | intermed_red_3),
     .starting_pix_x(starting_pix_x_red_3), .starting_pix_y(starting_pix_y_red_3),
.hpos_out(hpos_red_3), .vpos_out(vpos_red_3), .Ball(red_ball_3));
    assign Hchange_red_3 = ((hpos_red_3 == 16'd8) | (hpos_red_3 == 16'd615) |
((hpos_red_3 == 16'd300) & ((red_wall_on | wall_on) & ~wall_off & wall_on_game &
(PS_r_3[1] | PS_r_3[2]))) | ((hpos_red_3 == 16'd323)& ((red_wall_on | wall_on) &
wall_on_game & ~wall_off & (PS_r_3[3] | PS_r_3[4]) ))) & frame;
    assign Vchange_red_3 = ((vpos_red_3 == 16'd8) | (vpos_red_3 == 16'd455)) & frame;
    wire red_3_left, red_3_right;
    assign red_3_left = (hpos_red_3 <= 16'd284);
    assign red_3_right = (hpos_red_3 >= 16'd323);

    //fourth red ball
    wire red_ball_4;
    wire [15:0] starting_pix_x_red_4, starting_pix_y_red_4;
    assign starting_pix_x_red_4 = 16'd500;
    assign starting_pix_y_red_4 = 16'd420;
    wire Hchange_red_4 , Vchange_red_4;
    wire down_right_red_4, down_left_red_4, up_right_red_4, up_left_red_4,
intermed_red_4;
    wire [15:0] hpos_red_4, vpos_red_4;
    wire [4:0] PS_r_4;
    BallState redball_4_state(.clk(clk), .go(load_game_time), .frame(frame),
.Hchange(Hchange_red_4), .Vchange(Vchange_red_4),
     .down_right_1(down_right_red_4),
     .up_right_2(up_right_red_4),
     .up_left_3(up_left_red_4),
     .down_left_4(down_left_red_4),
     .intermed(intermed_red_4),
     .PS(PS_r_4));
    Ball redball_4(.clk(clk), .frame(frame), .H(hpos), .V(vpos),
     .down_right_1(down_right_red_4),
     .up_right_2(up_right_red_4),
     .up_left_3(up_left_red_4),
     .down_left_4(down_left_red_4),
     .intermed(reset_ball_loc | intermed_red_4),
     .starting_pix_x(starting_pix_x_red_4), .starting_pix_y(starting_pix_y_red_4),
.hpos_out(hpos_red_4), .vpos_out(vpos_red_4), .Ball(red_ball_4));
    assign Hchange_red_4 = ((hpos_red_4 == 16'd8) | (hpos_red_4 == 16'd615) |
((hpos_red_4 == 16'd300) & ((red_wall_on | wall_on) & ~wall_off & wall_on_game &
(PS_r_4[1] | PS_r_4[2]))) | ((hpos_red_4 == 16'd323)& ((red_wall_on | wall_on) &
wall_on_game & ~wall_off & (PS_r_4[3] | PS_r_4[4]) ))) & frame;
    assign Vchange_red_4 = ((vpos_red_4 == 16'd8) | (vpos_red_4 == 16'd455)) & frame;
    wire red_4_left, red_4_right;
    assign red_4_left = (hpos_red_4 <= 16'd284);
    assign red_4_right = (hpos_red_4 >= 16'd323);

    //first blue ball
    wire blue_wall_on;
    assign blue_wall_on = btnR & ~win_lose;
    wire blue_ball_1;
    wire [15:0] starting_pix_x_blue_1, starting_pix_y_blue_1;
    assign starting_pix_x_blue_1 = 16'd400;
    assign starting_pix_y_blue_1 = 16'd329;
    wire Hchange_blue_1 , Vchange_blue_1;
    wire down_right_blue_1, down_left_blue_1, up_right_blue_1, up_left_blue_1,
intermed_blue_1;
    wire [15:0] hpos_blue_1, vpos_blue_1;
    wire [4:0] PS_b_1;

    BallState blueball_1_state(.clk(clk), .go(load_game_time), .frame(frame),
.Hchange(Hchange_blue_1), .Vchange(Vchange_blue_1),
     .down_right_1(down_right_blue_1),
     .up_right_2(up_right_blue_1),
     .up_left_3(up_left_blue_1),
     .down_left_4(down_left_blue_1),
     .intermed(intermed_blue_1),
     .PS(PS_b_1));
    Ball blueball_1(.clk(clk), .frame(frame), .H(hpos), .V(vpos),
     .down_right_1(down_right_blue_1),
     .up_right_2(up_right_blue_1),
     .up_left_3(up_left_blue_1),
     .down_left_4(down_left_blue_1),
     .intermed(reset_ball_loc | intermed_blue_1),
     .starting_pix_x(starting_pix_x_blue_1), .starting_pix_y(starting_pix_y_blue_1),
.hpos_out(hpos_blue_1), .vpos_out(vpos_blue_1), .Ball(blue_ball_1));
    assign Hchange_blue_1 = ((hpos_blue_1 == 16'd8) | (hpos_blue_1 == 16'd615) |
((hpos_blue_1 == 16'd300) & ((blue_wall_on | wall_on) & ~wall_off  & wall_on_game &
(PS_b_1[1] | PS_b_1[2]))) | ((hpos_blue_1 == 16'd323) & ((blue_wall_on | wall_on)  &
wall_on_game & ~wall_off & & (PS_b_1[3] | PS_b_1[4])))) & frame;
    assign Vchange_blue_1 = ((vpos_blue_1 == 16'd8) | (vpos_blue_1 == 16'd455)) &
frame;
    wire blue_1_left, blue_1_right;
    assign blue_1_left = (hpos_blue_1 <= 16'd284);
    assign blue_1_right = (hpos_blue_1 >= 16'd323);

    //blue ball 2
      //first blue ball

    wire blue_ball_2;
    wire [15:0] starting_pix_x_blue_2, starting_pix_y_blue_2;
    assign starting_pix_x_blue_2 = 16'd200;
    assign starting_pix_y_blue_2 = 16'd200;
    wire Hchange_blue_2 , Vchange_blue_2;
    wire down_right_blue_2, down_left_blue_2, up_right_blue_2, up_left_blue_2,
intermed_blue_2;
    wire [15:0] hpos_blue_2, vpos_blue_2;
    wire [4:0] PS_b_2;
    BallState blueball_2_state(.clk(clk), .go(load_game_time), .frame(frame),
.Hchange(Hchange_blue_2), .Vchange(Vchange_blue_2),
     .down_right_1(down_right_blue_2),
     .up_right_2(up_right_blue_2),
     .up_left_3(up_left_blue_2),
     .down_left_4(down_left_blue_2),
     .intermed(intermed_blue_2),
     .PS(PS_b_2));
    Ball blueball_2(.clk(clk), .frame(frame), .H(hpos), .V(vpos),
     .down_right_1(down_right_blue_2),
     .up_right_2(up_right_blue_2),
     .up_left_3(up_left_blue_2),
     .down_left_4(down_left_blue_2),
     .intermed(reset_ball_loc | intermed_blue_2),
     .starting_pix_x(starting_pix_x_blue_2), .starting_pix_y(starting_pix_y_blue_2),
.hpos_out(hpos_blue_2), .vpos_out(vpos_blue_2), .Ball(blue_ball_2));
    assign Hchange_blue_2 = ((hpos_blue_2 == 16'd8) | (hpos_blue_2 == 16'd615) |
((hpos_blue_2 == 16'd300) & ((blue_wall_on | wall_on) & ~wall_off  & wall_on_game &
(PS_b_2[1] | PS_b_2[2]))) | ((hpos_blue_2 == 16'd323) & ((blue_wall_on | wall_on) &
wall_on_game & ~wall_off & (PS_b_2[3] | PS_b_2[4])))) & frame;
    assign Vchange_blue_2 = ((vpos_blue_2 == 16'd8) | (vpos_blue_2 == 16'd455)) &
frame;
    wire blue_2_left, blue_2_right;
    assign blue_2_left = (hpos_blue_2 <= 16'd284);
    assign blue_2_right = (hpos_blue_2 >= 16'd323);

    //third blue ball
    wire blue_ball_3;
    wire [15:0] starting_pix_x_blue_3, starting_pix_y_blue_3;
    assign starting_pix_x_blue_3 = 16'd50;
    assign starting_pix_y_blue_3 = 16'd200;
    wire Hchange_blue_3 , Vchange_blue_3;
    wire down_right_blue_3, down_left_blue_3, up_right_blue_3, up_left_blue_3,
intermed_blue_3;
    wire [15:0] hpos_blue_3, vpos_blue_3;
    wire [4:0] PS_b_3;
    BallState blueball_3_state(.clk(clk), .go(load_game_time), .frame(frame),
.Hchange(Hchange_blue_3), .Vchange(Vchange_blue_3),
     .down_right_1(down_right_blue_3),
     .up_right_2(up_right_blue_3),
     .up_left_3(up_left_blue_3),
     .down_left_4(down_left_blue_3),
     .intermed(intermed_blue_3),
     .PS(PS_b_3));
    Ball blueball_3(.clk(clk), .frame(frame), .H(hpos), .V(vpos),
     .down_right_1(down_right_blue_3),
     .up_right_2(up_right_blue_3),
     .up_left_3(up_left_blue_3),
     .down_left_4(down_left_blue_3),
     .intermed(reset_ball_loc | intermed_blue_3),
     .starting_pix_x(starting_pix_x_blue_3), .starting_pix_y(starting_pix_y_blue_3),
.hpos_out(hpos_blue_3), .vpos_out(vpos_blue_3), .Ball(blue_ball_3));
    assign Hchange_blue_3 = ((hpos_blue_3 == 16'd8) | (hpos_blue_3 == 16'd615) |
((hpos_blue_3 == 16'd300) & ((blue_wall_on | wall_on) & ~wall_off  & wall_on_game &
(PS_b_3[1] | PS_b_3[2]))) | ((hpos_blue_3 == 16'd323) & ((blue_wall_on | wall_on) &
wall_on_game & ~wall_off & (PS_b_3[3] | PS_b_3[4])))) & frame;
    assign Vchange_blue_3 = ((vpos_blue_3 == 16'd8) | (vpos_blue_3 == 16'd455)) &
frame;
    wire blue_3_left, blue_3_right;
    assign blue_3_left = (hpos_blue_3 <= 16'd284);
    assign blue_3_right = (hpos_blue_3 >= 16'd323);
    //fourth blue ball
        //third blue ball
    wire blue_ball_4;
    wire [15:0] starting_pix_x_blue_4, starting_pix_y_blue_4;
    assign starting_pix_x_blue_4 = 16'd50;
    assign starting_pix_y_blue_4 = 16'd100;
    wire Hchange_blue_4 , Vchange_blue_4;
    wire down_right_blue_4, down_left_blue_4, up_right_blue_4, up_left_blue_4,
intermed_blue_4;
    wire [15:0] hpos_blue_4, vpos_blue_4;
    wire [4:0] PS_b_4;
    BallState blueball_4_state(.clk(clk), .go(load_game_time), .frame(frame),
.Hchange(Hchange_blue_4), .Vchange(Vchange_blue_4),
     .down_right_1(down_right_blue_4),
     .up_right_2(up_right_blue_4),
     .up_left_3(up_left_blue_4),
     .down_left_4(down_left_blue_4),
     .intermed(intermed_blue_4),
     .PS(PS_b_4));
    Ball blueball_4(.clk(clk), .frame(frame), .H(hpos), .V(vpos),
     .down_right_1(down_right_blue_4),
     .up_right_2(up_right_blue_4),
     .up_left_3(up_left_blue_4),
     .down_left_4(down_left_blue_4),
     .intermed(reset_ball_loc | intermed_blue_4),
     .starting_pix_x(starting_pix_x_blue_4), .starting_pix_y(starting_pix_y_blue_4),
.hpos_out(hpos_blue_4), .vpos_out(vpos_blue_4), .Ball(blue_ball_4));
    assign Hchange_blue_4 = ((hpos_blue_4 == 16'd8) | (hpos_blue_4 == 16'd615) |
((hpos_blue_4 == 16'd300) & ((blue_wall_on | wall_on) & ~wall_off  & wall_on_game &
(PS_b_4[1] | PS_b_4[2]))) | ((hpos_blue_4 == 16'd323) & ((blue_wall_on | wall_on) &
wall_on_game & ~wall_off & (PS_b_4[3] | PS_b_4[4])))) & frame;
    assign Vchange_blue_4 = ((vpos_blue_4 == 16'd8) | (vpos_blue_4 == 16'd455)) &
frame;
    wire blue_4_left, blue_4_right;
    assign blue_4_left = (hpos_blue_4 <= 16'd284);
    assign blue_4_right = (hpos_blue_4 >= 16'd323);


    // Magenta middle wall
    assign wall_on = ~red_wall_on & ~blue_wall_on;
    assign wall_off = red_wall_on & blue_wall_on;
    wire wall;
    wall mdwall(.H(hpos), .V(vpos), .wall(wall));
    // timers and Top State
    wire timeup_8, game_timeup;
    wire reset_8_timer;
    counterUD16L timer8_sec(.clk(clk), .Din(16'd480), .Dw(frame),
.LD(reset_8_timer), .DTC(timeup_8));
    wire [15:0]time_in;
    assign time_in = {8'b0, sw[15:8]};
    wire [15:0]sec_1;
    counterUD16L second(.clk(clk), .Din(16'b0), .Up(frame), .LD((sec_1 == 16'd60)),
.Q(sec_1));
    wire [15:0]game_time;
    wire count_down_game;
    counterUD16L gametime(.clk(clk), .Din(time_in), .Dw((sec_1 == 16'd60) &
count_down_game), .LD(load_game_time), .DTC(game_timeup), .Q(game_time));
    wire win;
    assign win = (red_1_left & red_2_left & red_3_left & red_4_left & blue_1_right &
blue_2_right & blue_3_right & blue_4_right) | (red_1_right & red_2_right &
red_3_right & red_4_right & blue_1_left & blue_2_left & blue_3_left & blue_4_left);
    wire Flash_border, Flash_game_time, stay_green;
    TopState gamestate(.go(btnU), .clk(clk),
        .timeup_8(timeup_8),
        .game_timeup(game_timeup),
        .win(win),
        .load_game_time(load_game_time),
        .reset_8_timer(reset_8_timer),
        .Display_game_time(Display_game_time),
        .Flash_border (Flash_border),
        .Flash_game_time(Flash_game_time),
        .wall_on_game(wall_on_game),
        .count_down_game(count_down_game),
        .stay_green(stay_green),
        .win_lose(win_lose),
        .reset_ball_loc(reset_ball_loc),
        .led(led));


    // assign vgas
    wire [3:0]active;
    assign active = {4{active_region}} & 4'b1111;
    assign vgaRed = active & ({4{red_ball_1}} | {4{red_ball_2}} | {4{red_ball_3}}
|{4{red_ball_4}} | ({4{wall}} & {4{wall_on_game}}  &  ~{4{blue_wall_on}}));
    assign vgaGreen = active & ({4{border}}  &  (({4{Flash_border}} & {4{sec_1[4]}})
 |   {4{stay_green}}));
    assign vgaBlue = active & ({4{blue_ball_1}} | {4{blue_ball_2}} |
{4{blue_ball_3}} | {4{blue_ball_4}} | ({4{wall}} & {4{wall_on_game}} &
~{4{red_wall_on}}));
```

```
    // anodes
    wire [3:0]ring;
    Ring_Counter An_Sel(.clk(clk), .advance(digsel), .out(ring));
    wire [3:0]bit4_out;
    wire [15:0]sel_Inp;
    assign sel_Inp = {game_time[15:8], game_time[7:0], 4'b0, 4'b0};
    Selector select(.N(sel_Inp), .sel(ring), .H(bit4_out));
    hex7seg hex7(.n(bit4_out), .seg(seg));
    assign an[3] = ~ring[3] | (Flash_game_time & sec_1[4]);
    assign an[2] = ~ring[2] | (Flash_game_time & sec_1[4]);
    assign an[1] = ~ring[1];
    assign an[0] = ~ring[0];
    assign dp = 1'b1;



endmodule
```