Arib Ahmed

Section C

3/9/2020

S. Beamer

Lab 7

Introduction:

Lab 7 of CSE100, known better as “Rainbow Road”, is the largest lab of the class. Using all knowledge learned from the entire class, it is the students duty to create a game using the Basys3 board and a monitor. Unlike other labs, students are instructed to create a game using a monitor with the Basys3’s VGA port. The game design and implementation is as follows: The student must create a game where a car is on a road. The car is static but the road moves with the buttons to simulate the car driving on the road. The left button moves the road left and the right button moves the button to the right. Initially, the road is not moving but the user presses the center button as a signal to make the road start moving. Once the road is moving, it scrolls infinitely down until the car is off the road. As one might think, there are a lot of nuances to take care off. First, once the center button is pressed, the car blinks for two seconds, then the road moves. Similarly, once the car is off the road, the car blinks until the user presses the button again. As I go through my methods, I will go extremely in depth of what my idea processes were in doing every single step.

Methods:

As there are so many different things to do in this lab, it was hard to think about which to do first. I felt like the best way to start this lab was just to get something to show up on the screen. To start this, I created a module named the VGA controller and went from there. Our games are played on a 640 by 480 pixel screen. Although this is where our game is shown on the screen, there are actually 800 by 525 pixels to account for in the entire screen. The best way to account for every pixel is to create a pixel counter. Since I wanted the decimal numbers to be at max 800, I thought the best counter would be a 10 bit counter. I used an old lab and created a 10 bit counter for the columns and rows of the screen. The counter would count every single column and as soon as it reached the 799th pixel of the column, it would increment a row and start counting again. This is how the entire project is built upon. I had a counter that freely incremented until it reached 799 and once it did, it would reset and go back to 0 and increment another counter by 1 only. Once that second counter goes all the way to 524, then the whole screen would be accounted for and that would reset both counters. Once the entire screen was accounted for, that was one "frame". We can use this frame as a time counter. Since the screen refreshed 60 times a frame, every time we had a frame signal appear 60 times, it would be one second. I preemptively created a counter that would go up every frame signal and once it reached 60 frames, it would send out a signal saying one second has passed. To finish this off, we also had signals Hsync and Vsync that needed to be accounted for. These were outputs that went to the Basys3 board in order to have correct signals on the screen. The Hsync region was when the row pixels were in between 489 and 490, and the Vsync region was when the column pixels were in between 655 and 750. Once it was in these regions, Hsync and Vsync must both be low.

The next step for me was to have something show up on screen. The Basys3 board had 3 more outputs that allowed us to do that. We had RGB values that turned on at specific pixels on the screen. If we wanted the screen to be white, we can set R, G, and B to 1111 to have a completely white screen. Now, the fact is that we need a specific region to be a specific color only. To do this, we need our pixel counter that we created beforehand. To create a 16 by 16 square, for example, we needed the RGB values to be all high when the pixel counter was in between a 16 by 16 square. All we had to do was set the RGB values to high whenever the pixels were less than or equal to the pixel row and greater than or equal to the pixel row plus whatever we wanted our width to be. We did the same for our pixel column and length and we would essentially have a module that created a rectangle wherever we desired. To finish this off, we needed an output that would go directly into each RGB value that would designate if it was high or low at a dependent rectangle. If we wanted a white rectangle, we would have the input RGB values to all be high so that our output RGB values will be high whenever the rectangle is supposed to appear.

Now that we have a module that created a rectangle at a specific point, the next step was to create 7 of these rectangles in order to fill up the screen and create a road. Since this lab is called rainbow road, it is important to follow ROYGBIV in order to simulate a rainbow. I had the following rectangles: red, orange, yellow, green, blue, indigo, and violet in that order. Each rectangle is 80 pixels long so that is what I set my length as. In order to simulate the rectangles falling on the y axis. I had a signal whenever the center button was pressed to start a counter for each rectangle's y axis. The counter was connected to the rectangles coordinates as it would increment (since the higher the pixel, the lower the place on screen) twice a frame. Every time it reached a frame and the pixel right under it, I would have it be high on those two pixels to create a double frame signal. That double frame signal would be connected to the counter that controlled the y axis of all my rectangles. I would have each rectangle's y axis to be reset once the top of each rectangle hit the pixel 481. Once it reached 481, it was off the game screen, meaning it needed to be reset. I set it to whenever the counter reached 481, it would reset to 0. This led to the second objective. Once the rectangles were reset back to the top of the screen, it would not gradually spawn in, it would glitch in. For example, if my red road would go all the way to the bottom, it would reset until the entire road was able to fit on screen. To do this, I had another counter for each road's width. Since each rectangles width was 80 pixels, I would have another counter specifically for each road's width. This is one of my proudest moments in this lab as I figured out something so unique that even the TA was surprised at my solution. I had a counter that went all the way to 80 every double frame and it would send a signal once it did to start the counter for the y axis to start incrementing. This way, each rectangle would gradually spawn in instead of spawning in all at once.

All the couple paragraphs just written took days upon days to figure out. I somehow finished it all just minutes before the first check off in order to get the first extra credit part. The next step was to finish the game. The next step was to create another counter for the x axis of each rectangle. Since the left and right button controlled the right and left direction of the car, I created another counter for the x axis. If the right button was pressed, it decremented the x axis to make all the rectangles move to the left (so it makes it look like the car is moving to the right), and vice versa with the left button. Now that I created a way to move the road, the next step was to do the road size.

The switches 4-6 controlled the size of the road. It could spam from 8 pixels all the way to 120. To do this, we use the formula (8 + (16 * sw[6:4])) in order to create the amount of pixels we want. I used the output of that formula was the size of the road I wanted.

The easiest part of this lab was to create a state machine for the game. I was honestly expecting to make 20 state machines for every single part of the lab, but funny enough it was a simple state machine with only 4 states. It starts off on an idle state where none of the roads are moving and it is waiting for an input from the center button. Once that is pressed, the road scrolls infinitely until the car is off the road. Once we find out how the car is off the road, it sends a signal to stop the scrolling. The state after that is the lose state where it stays there until another center button press, which then sends it back to the game state.

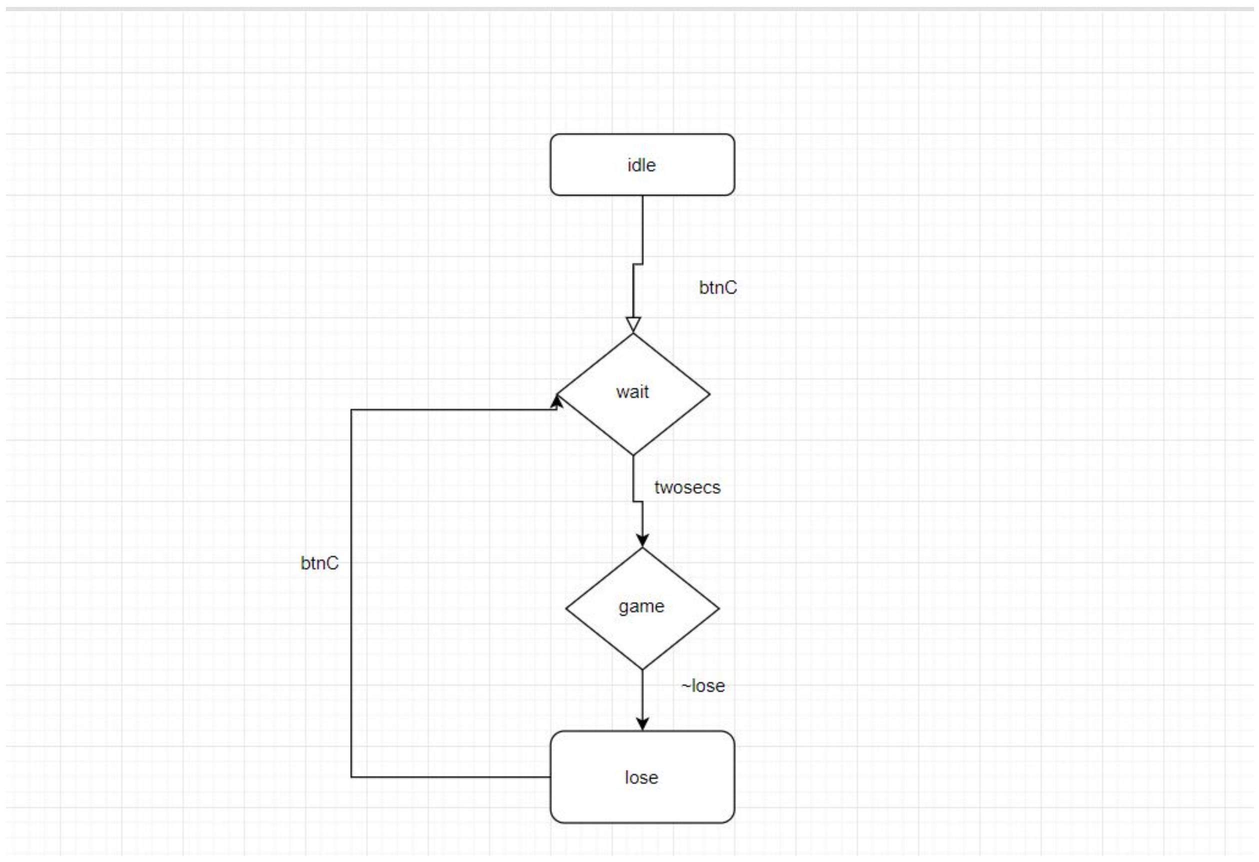
Before I get to the hardest parts of this lab that took me the most time, I want to talk about the board anode, and led implementation. The anodes in this lab correspond to the score. It only increments during the game state every 16 frames. I created another counter specifically for the score that went up every 16 frames during the game state. Once it hit the lose state, it would blink until it went back to the game state. The LEDs were a fun part to implement and I used another counter to simulate what we needed to do. First, the LEDs were used as a countdown once the center button is initially pressed for the two seconds. It would turn the 15th led, the 11th led, the 7th led, and the 3rd led every quarter second during the two second time frame. Once two seconds passed, it would shift to the right every quarter second during the game state. In the lose state, the LED’s will turn off. To implement this, I had a freerunning counter that went up to an arbitrary number. I simulated having a shift LED register by just turning on the LEDs at the specific time. It now looked like I had the LEDs shifting to the right, but in reality, it was just turning on a specific time to look like it was shifting. I just designated it to have each LED be on every quarter second with 3 other LEDs, once that quarter second was up, it would turn on the ones directly to the right for another quarter second.

The last two things I had to do for this lab was a doozy: having a random portion of the road be shifted everytime it respawned and have a lose signal that would detect when the car was off the road. The first thing I decided to do was the randomness factor. In lab 5, it was required for all students to create a random number generator using an LFSR. Unfortunately for me, I did not create the required LFSR as it did not work for me. To make a random number generator, I had a free running counter that determined the amount of pixels that the road shifted by. I had a value that saved the amount of pixels shifted by determining if the counter was in a specific frame. If it was in between 0 and 100, it would save the value 8. I did this for each value it wanted me to do. For positive shifts, I did even numbers, and for negative shifts I did odd numbers. Since all odd numbers have the last bit as 1, I used that to determine if I wanted to subtract or add the shift. Once the roads hit the signal to respawn, it saves the shift counter wherever it was to determine how much to shift by. It then sets its x axis by the block under it and subtracts or adds the shift.

The last segment is to determine whether or not the car is off the road. To do this, I researched simple geometry. I thought to myself, how do I find out if two rectangles are intersecting. I did this using python, a high level language, and then I switched it over to verilog. To do this, we can check the coordinates of each rectangle and see if it is constantly intersecting with the car. As long as one segment is intersecting with the car, then we know that the car is still on the road. If the car is off the road, then none of the roads are intersecting. To find this out, we check each of the rectangles coordinates to see if the car is between the coordinates, if it is not, then we can assume that the car is off the road.

Results:

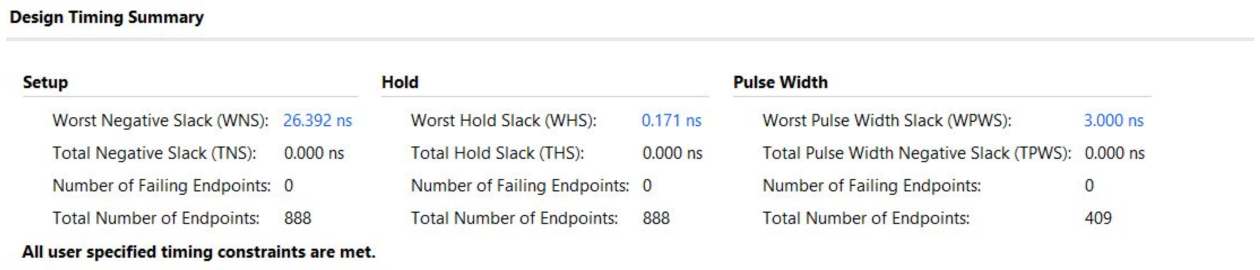
State Machine:



(As simple as it is, yes, this is the only state machine in the entire project)

Testing in this lab was incredibly tedious and it is the reason why this lab takes so long. Since we cannot use the simulation to test the screen output, we had to generate a bitstream everytime we wanted to test something. This was actually most of the lab since I had to wait 10-20 minutes every time I changed one line of code to see if it did anything.

Timing Report (Also in Appendix):



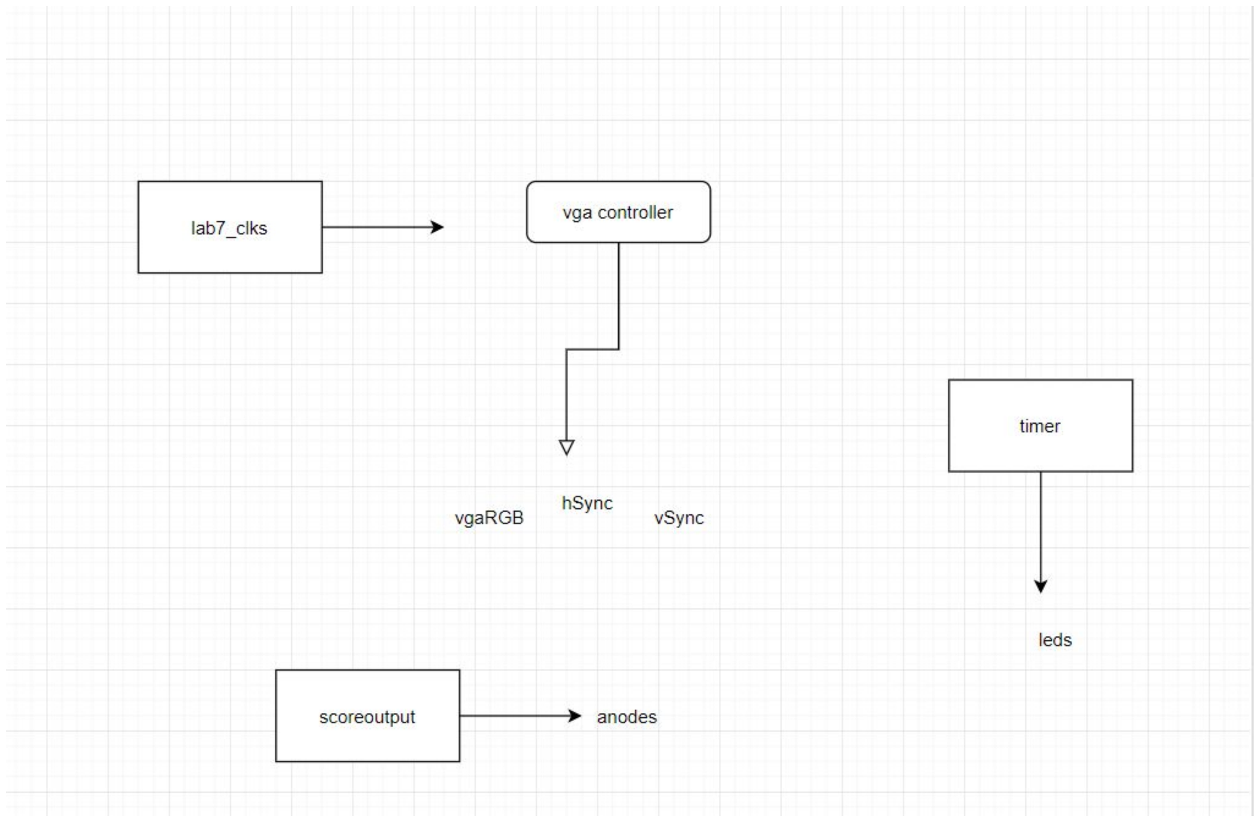
Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	26.392 ns	Worst Hold Slack (WHS):	0.171 ns	Worst Pulse Width Slack (WPWS):	3.000 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	888	Total Number of Endpoints:	888	Total Number of Endpoints:	409

All user specified timing constraints are met.

It looks like the worst negative slack was 26.392 ns. I submitted the pdf in the appendix.

Top Level Diagram:



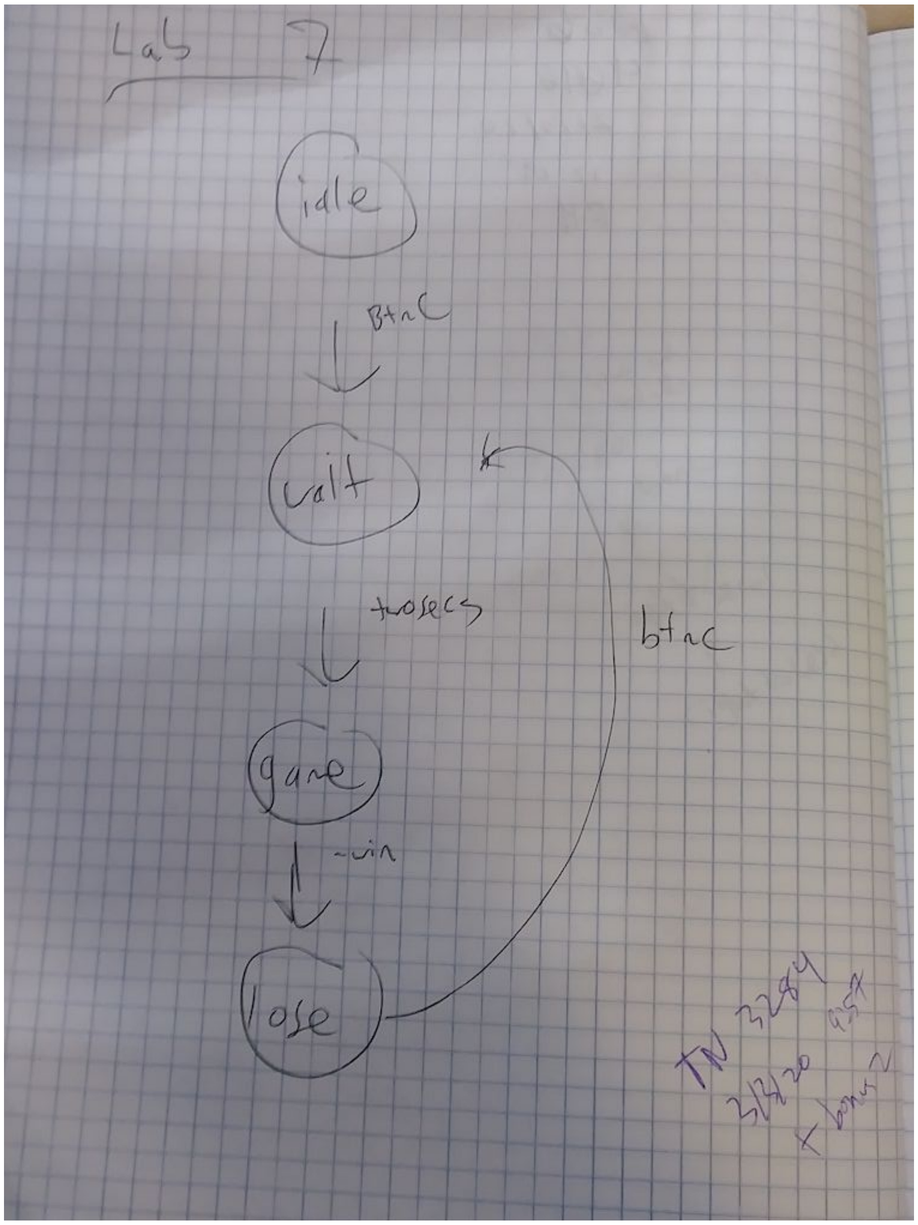
I tried to keep it as simple as possible as I did not really want to go in depth with this top level as it would have been way too insane. Basically, the top level has two modules, lab7_clks and vga controller. I put all the logic inside vga controller to make my top level cleaner. Tha vga controller sends outputs to hsync, vsync, and the vgaRGB outputs. It also has outputs that controls the anodes and LEDS.

Conclusion:

Once in my life, I started something extremely early and finished on time. Even better, I got both extra credit points and received 40% on late lab assignments. The best part of it all, I was the first to get checked off in the entire class. This was the best feeling to me since I have not been doing so well in the class earlier on but it made me so happy to see myself picking it up at the last half. All those late nights banging my head against the wall all seemed worth it once I had a whole week to relax and do nothing. I also tried my best to help others finish this lab, because I wanted to see everyone succeed like I did.

Appendix:

Notebook:


Lab 7
idle
BtnC
wait
twosecs
game
~win
lose
btnC
TN 3284
3/3/20 957
+ bonus 2

Code and Schematic (I could not even most of the schematic/timing diagram without vivado crashing):

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////
///////////////
// Company:
// Engineer:
//
// Create Date: 02/26/2020 06:11:25 PM
// Design Name:
// Module Name: rectangle
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////
///////////////


module rectangle(

    input [9:0] startingx,
    input [9:0] startingy,

    input [9:0] currentx,
    input [9:0] currenty,

    input [9:0] width,
    input [9:0] length,

    input [3:0] inputBlue,
    input [3:0] inputRed,
    input [3:0] inputGreen,
    output [3:0] outputRed,
    output [3:0] outputGreen,
    output [3:0] outputBlue,
    output flag
    );

    wire flag1, flag2;

    assign flag1 = ((currentx >= startingx) & (currentx <=
(startingx + width)));
    assign flag2= ((currenty >= startingy) & (currenty <=
(startingy + length)));

    assign flag = (flag1 & flag2);



    assign outputRed = ({4{flag}} & inputRed);
    assign outputBlue = ({4{flag}} & inputBlue);
    assign outputGreen = ({4{flag}} & inputGreen);









endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////
///////////////
// Company:
// Engineer:
//
// Create Date: 01/30/2020 01:20:25 PM
// Design Name:
// Module Name: counter
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////
///////////////


module tenbitcounter(
    input clk,
    input CE,
    input reset,
    input Up,
    input Dw,
    input LD,
    input [9:0] Din,
    output [9:0] Q,
    output twosecs,
    output sixframe,
    output quartersecond4,
    output quartersecond1,
    output quartersecond2,
    output quartersecond3


    );

 wire [9:0] D;


 assign D[0] = (Up & ~Dw & ~LD & (Q[0] ^ 1))

             | (Dw & ~Up & ~LD & (Q[0] ^ 1))

             | (LD & Din[0])

             | (~Up & ~Dw & ~LD & Q[0]);



 assign D[1] = (Up & ~Dw & ~LD & (Q[1] ^ Q[0]))

             | (Dw & ~Up & ~LD & (Q[1] ^ ~Q[0]))

             | (LD & Din[1])

             | (~Up & ~Dw & ~LD & Q[1]);



 assign D[2] = (Up & ~Dw & ~LD & (Q[2] ^ (Q[1] & Q[0])))

             | (Dw & ~Up & ~LD & (Q[2] ^ (~Q[1] & ~Q[0])))

             | (LD & Din[2])

             | (~Up & ~Dw & ~LD & Q[2]);
```

```
 assign D[3] = (Up & ~Dw & ~LD & (Q[3] ^ (Q[2] & Q[1] & Q[0])))

             | (Dw & ~Up & ~LD & (Q[3] ^ (~Q[2] & ~Q[1] & ~Q[0])))

             | (LD & Din[3])

             | (~Up & ~Dw & ~LD & Q[3]);

assign D[4] = (Up & ~Dw & ~LD & (Q[4] ^ (Q[3] & Q[2] & Q[1] &
Q[0])))
             | (Dw & ~Up & ~LD & (Q[4] ^ (~Q[3] & ~Q[2] & ~Q[1] &
~Q[0])))
             | (LD & Din[4])

             | (~Up & ~Dw & ~LD & Q[4]);

assign D[5] = (Up & ~Dw & ~LD & (Q[5] ^ (Q[4] & Q[3] & Q[2] & Q[1]
& Q[0])))
             | (Dw & ~Up & ~LD & (Q[5] ^ (~Q[4] & ~Q[3] & ~Q[2] &
~Q[1] & ~Q[0])))
             | (LD & Din[5])

             | (~Up & ~Dw & ~LD & Q[5]);

 assign D[6] = (Up & ~Dw & ~LD & (Q[6] ^ (Q[5] & Q[4] & Q[3] & Q[2]
& Q[1] & Q[0])))
             | (Dw & ~Up & ~LD & (Q[6] ^ (~Q[5] & ~Q[4] & ~Q[3] &
~Q[2] & ~Q[1] & ~Q[0])))
             | (LD & Din[6])

             | (~Up & ~Dw & ~LD & Q[6]);

 assign D[7] = (Up & ~Dw & ~LD & (Q[7] ^ (Q[6] & Q[5] & Q[4] & Q[3]
& Q[2] & Q[1] & Q[0])))
             | (Dw & ~Up & ~LD & (Q[7] ^ (~Q[6] & ~Q[5] & ~Q[4] &
~Q[3] & ~Q[2] & ~Q[1] & ~Q[0])))
             | (LD & Din[7])

             | (~Up & ~Dw & ~LD & Q[7]);



 assign D[8] = (Up & ~Dw & ~LD & (Q[8] ^ (Q[7] & Q[6] & Q[5] & Q[4]
& Q[3] & Q[2] & Q[1] & Q[0])))
             | (Dw & ~Up & ~LD & (Q[8] ^ (~Q[7] & ~Q[6] & ~Q[5] &
~Q[4] & ~Q[3] & ~Q[2] & ~Q[1] & ~Q[0])))
             | (LD & Din[8])

             | (~Up & ~Dw & ~LD & Q[8]);



 assign D[9] = (Up & ~Dw & ~LD & (Q[9] ^ (Q[8] & Q[7] & Q[6] & Q[5]
& Q[4] & Q[3] & Q[2] & Q[1] & Q[0])))
             | (Dw & ~Up & ~LD & (Q[9] ^ (~Q[8]& ~Q[7] & ~Q[6] &
~Q[5] & ~Q[4] & ~Q[3] & ~Q[2] & ~Q[1] & ~Q[0])))
             | (LD & Din[9])

             | (~Up & ~Dw & ~LD & Q[9]);




 FDRE #(.INIT(1'b0)) countUD4L0 (.C(clk), .CE(CE), .R(reset),
.D(D[0]), .Q(Q[0]));
 FDRE #(.INIT(1'b0)) countUD4L1 (.C(clk), .CE(CE), .R(reset),
.D(D[1]), .Q(Q[1]));
 FDRE #(.INIT(1'b0)) countUD4L2 (.C(clk), .CE(CE), .R(reset),
.D(D[2]), .Q(Q[2]));
 FDRE #(.INIT(1'b0)) countUD4L3 (.C(clk), .CE(CE), .R(reset),
.D(D[3]), .Q(Q[3]));
 FDRE #(.INIT(1'b0)) countUD4L4 (.C(clk), .CE(CE), .R(reset),
.D(D[4]), .Q(Q[4]));
 FDRE #(.INIT(1'b0)) countUD4L5 (.C(clk), .CE(CE), .R(reset),
.D(D[5]), .Q(Q[5]));
```

```
 FDRE #(.INIT(1'b0)) countUD4L6 (.C(clk), .CE(CE), .R(reset),
.D(D[6]), .Q(Q[6]));
 FDRE #(.INIT(1'b0)) countUD4L7 (.C(clk), .CE(CE), .R(reset),
.D(D[7]), .Q(Q[7]));
 FDRE #(.INIT(1'b0)) countUD4L8 (.C(clk), .CE(CE), .R(reset),
.D(D[8]), .Q(Q[8]));
 FDRE #(.INIT(1'b0)) countUD4L9 (.C(clk), .CE(CE), .R(reset),
.D(D[9]), .Q(Q[9]));


 assign twosecs = (Q == 10'd120);
 assign quartersecond1 = (Q>10'd30);
 assign quartersecond2 = (Q>10'd60);
 assign quartersecond3 = (Q>10'd90);
 assign quartersecond4 = (Q >= 10'd120);
 assign sixframe = (Q == 10'd16);

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////
////////////////
// Company:
// Engineer:
//
// Create Date: 02/26/2020 11:06:13 AM
// Design Name:
// Module Name: toplevel
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////
////////////////


module toplevel(

    input clkin,
    input btnC,
    input btnD,
    input btnR,
    input btnL,
    input btnU,
    input [15:0] sw,
    output Hsync,
    output Vsync,
    output [3:0] vgaRed,
    output [3:0] vgaBlue,
    output [3:0] vgaGreen,
    output dp,
    output [6:0] seg,
    output [15:0] led,
    output [3:0] an

    );

    wire clk, digsel;

    lab7_clks not_so_slow (.clkin(clkin), .greset(sw[0]),
.clk(clk), .digsel(digsel));


    vga_controller vga(.Hsync(Hsync), .Vsync(Vsync), .clk(clk),
.vgaRed(vgaRed),
    .vgaBlue(vgaBlue), .vgaGreen(vgaGreen), .sw(sw), .dp(dp),
.led(led),
    .an(an), .seg(seg), .btnC(btnC), .btnR(btnR), .btnL(btnL),
.btnU(btnU),
    .btnD(btnD), .digsel(digsel));




endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////
////////////////
// Company:
// Engineer:
//
// Create Date: 02/26/2020 10:19:11 AM
// Design Name:
// Module Name: vga_controller
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////
////////////////


module vga_controller(

    input clk,
    input btnC,
    input btnD,
    input btnR,
    input btnL,
    input btnU,
    input digsel,
    input [15:0] sw,
    output Hsync,
    output Vsync,
    output [3:0] vgaRed,
    output [3:0] vgaBlue,
    output [3:0] vgaGreen,
    output dp,
    output [6:0] seg,
    output oops,
    output rgb_oops,
    output [15:0] led,
    output [3:0] an

    );

    wire t1, t2;
    wire [9:0] out1, out2;
    wire uptick;
    wire activeregion;
    wire [3:0] ringo;
    wire [3:0] seloutput;
    wire [15:0] Ninput;
    wire synchbtnC;
    wire synchbtnL;
    wire synchbtnD;
    wire synchbtnU;
    wire synchbtnR;

    wire carflag, redflag, orangeflag, yellowflag, greenflag,
blueflag, indigoflag,violetflag;
    wire win;

    FDRE #(.INIT(1'b0)) sync1 (.C(clk), .CE(1'b1), .D(btnL),
.Q(synchbtnL));
    FDRE #(.INIT(1'b0)) sync2 (.C(clk), .CE(1'b1), .D(btnR),
.Q(synchbtnR));
    FDRE #(.INIT(1'b0)) sync3 (.C(clk), .CE(1'b1), .D(btnU),
.Q(synchbtnU));
    FDRE #(.INIT(1'b0)) sync4 (.C(clk), .CE(1'b1), .D(btnC),
.Q(synchbtnC));
    FDRE #(.INIT(1'b0)) sync5 (.C(clk), .CE(1'b1), .D(btnD),
.Q(synchbtnD));



    tenbitcounter colcount(.Up(1'b1), .Dw(1'b0), .LD(1'b0),
.Din(10'b0),
    .clk(clk), .CE(1'b1), .Q(out1), .reset(t1));

    tenbitcounter rowcount(.Up(1'b1), .Dw(1'b0), .LD(1'b0),
.Din(10'b0),
    .clk(clk), .CE(t1), .Q(out2), .reset(t2));

    assign Hsync = ~((out1 >= 10'd655) & (out1 <= 10'd750));
    assign Vsync = ~((out2 >= 10'd489) & (out2 <= 10'd490));

    assign t1 = (out1 == 10'd799);
    assign t2 = ((out1 == 10'd799) & (out2 == 10'd524));

    wire frame, twoframe;

    assign frame = ((out1 == 10'd799) & (out2 == 10'd524));
    assign twoframe = ((out1 == 10'd799) & (out2 == 10'd524))
|((out1 == 10'd798) & (out2 == 10'd524)) ;
    assign activeregion = ({out1 >= 10'd0 & out1 <= 10'd639} &
{out2 >= 10'd0 & out2 <= 10'd479});





    wire road1, road2, road3, road4, road5, road6, road7;

    wire [9:0] roadsize, multiplyroad;


    assign multiplyroad = (10'd16 * sw[6:4]);
    assign roadsize = (10'd8 + multiplyroad);
```

```
//    rectangle roadd1(.clk(clk), .oghori1(10'd260),
.ogveri1(10'd0), .oghori2(10'd380), .ogveri2(10'd0), .row(out2),
//     .column(out1), .frame(frame), .horipos1(horipos1),
.veripos1(veripos1), .horipos2(horipos2), .veripos2(veripos2),
//     .reset(veripos1 == 10'd479));

//    rectangle roadd2(.clk(clk), .oghori1(10'd260),
.ogveri1(10'd0), .oghori2(10'd380), .ogveri2(10'd0), .row(out2),
//     .column(out1), .frame(frame), .horipos1(horipos3),
.veripos1(veripos3), .horipos2(horipos4), .veripos2(veripos4),
//     .reset((veripos3+10'd81) == 10'd479));

//     rectangle roadd3(.clk(clk), .oghori1(10'd260),
.ogveri1(10'd0), .oghori2(10'd380), .ogveri2(10'd0), .row(out2),
//     .column(out1), .frame(frame), .horipos1(horipos5),
.veripos1(veripos5), .horipos2(horipos6), .veripos2(veripos6),
//     .reset((veripos5+10'd162) == 10'd479));

//      rectangle roadd4(.clk(clk), .oghori1(10'd260),
.ogveri1(10'd0), .oghori2(10'd380), .ogveri2(10'd0), .row(out2),
//     .column(out1), .frame(frame), .horipos1(horipos7),
.veripos1(veripos7), .horipos2(horipos8), .veripos2(veripos8),
//     .reset((veripos7+10'd243) == 10'd479));

//      rectangle roadd5(.clk(clk), .oghori1(10'd260),
.ogveri1(10'd0), .oghori2(10'd380), .ogveri2(10'd0), .row(out2),
//     .column(out1), .frame(frame), .horipos1(horipos9),
.veripos1(veripos9), .horipos2(horipos10), .veripos2(veripos10),
//     .reset((veripos9+10'd324) == 10'd479));

//      rectangle roadd6(.clk(clk), .oghori1(10'd260),
.ogveri1(10'd0), .oghori2(10'd380), .ogveri2(10'd0), .row(out2),
//     .column(out1), .frame(frame), .horipos1(horipos11),
.veripos1(veripos11), .horipos2(horipos12), .veripos2(veripos12),
//     .reset((veripos11+10'd404) == 10'd479));
//      rectangle roadd7(.clk(clk), .oghori1(10'd260),
.ogveri1(10'd484), .oghori2(10'd380), .ogveri2(10'd484),
.row(out2),
//     .column(out1), .frame(frame), .horipos1(horipos13),
.veripos1(veripos13), .horipos2(horipos14), .veripos2(veripos14),
//     .reset((veripos13) == 10'd479));

//
/////////////////////////////////////////////////////////////////
/////////

//    assign road1 = ((out1 >= 10'd260) & (out1 <= 10'd380)
//    & (out2 >= veripos1) & (out2 <= (veripos2+10'd80)));

//    assign road2 = ((out1 >= 10'd260) & (out1 <= 10'd380)
//       & (out2 >= (veripos3+10'd81)) & (out2 <=
(veripos3+10'd161)));

//    assign road3 = ((out1 >= 10'd260) & (out1 <= 10'd380)
//    & (out2 >= (veripos5+10'd162)) & (out2 <=
(veripos5+10'd242)));

//    assign road4 = ((out1 >= 10'd260) & (out1 <= 10'd380)
//    & (out2 >= (veripos7+10'd243)) & (out2 <=
(veripos7+10'd323)));

//    assign road5 = ((out1 >= 10'd260) & (out1 <= 10'd380)
//    & (out2 >= (veripos9+10'd324)) & (out2 <=
(veripos9+10'd403)));

//    assign road6 = ((out1 >= 10'd260) & (out1 <= 10'd380)
//    & (out2 >= (veripos11+10'd404)) & (out2 <=
(veripos11+10'd484)));

//    assign road7 = ((out1 >= 10'd260) & (out1 <= 10'd380)
//    & (out2 >= (veripos13)) & (out2 <= (veripos13+10'd80)));
//    assign vgaRed = ~(~{4{car}} & ~{4{road1}} & ~{4{road4}} &
~{4{road5}} & ~{1{road6}} & ~{4{road7}});
//    assign vgaBlue= ~(~{4{car}} & ~{4{road2}} & ~{2{road6}} &
~{2{road7}});
//    assign vgaGreen = ~(~{4{car}} & ~{4{road3}} & ~{2{road4}} &
~{4{road5}} & ~{4{road7}});



//    rectangle violetroad(.clk(clk), .oghori1(10'd260),
.ogveri1(10'd0), .oghori2(10'd380), .ogveri2(10'd80), .row(out2),
//     .column(out1), .frame(frame), .horipos1(horipos13),
.veripos1(veripos13), .horipos2(horipos14), .veripos2(veripos14),
//     .reset((veripos13-10'd80) == 10'd479));

//    assign road7 = ((out1 >= 10'd260) & (out1 <= 10'd380)
//    & (out2 >= (veripos13-10'd80)) & (out2 <= (veripos13)));

    //0 until it finishes 80 pixels, then it starts counting up one


//    rectangle indigoroad(.clk(clk), .oghori1(10'd260),
.ogveri1(10'd0), .oghori2(10'd380), .ogveri2(10'd0), .row(out2),
//     .column(out1), .frame(frame), .horipos1(horipos11),
.veripos1(veripos11), .horipos2(horipos12), .veripos2(veripos12),
//     .reset((veripos11) == 10'd479));



//    assign road6 = ((out1 >= 10'd260) & (out1 <= 10'd380)
//    & (out2 >= (veripos13-veripos11)) & (out2 <=
(veripos12+10'd80)));

//    assign car = ((out1 >= 10'd312) & (out1 <= 10'd328)
//    & (out2 >= 10'd392) & (out2 <= 10'd408));

 wire[9:0] timeoutput;
 wire resettimer;
 wire twosecs;
 wire notmatching;
 wire quartersecond1, quartersecond2, quartersecond3,
quartersecond4;

 tenbitcounter timecounter(.clk(clk), .Up(frame), .Dw(1'b0),
.LD(1'b0), .CE(1'b1), .reset(resettimer), .Q(timeoutput),
 .twosecs(twosecs), .quartersecond1(quartersecond1),
.quartersecond2(quartersecond2), .quartersecond3(quartersecond3),
.quartersecond4(quartersecond4));

 wire [9:0] timeoutput2;



 wire [3:0] PS;
 wire [3:0] NS;

 wire idle, game, waitpls, lose;

 assign NS[0] = (PS[0] & ~synchbtnC);
 assign NS[1] = (PS[0] & synchbtnC | (PS[3] & synchbtnC) | (PS[1] &
~twosecs));
 assign NS[2] = (PS[1] & twosecs | (PS[2] & win));
 assign NS[3] = (PS[2] & ~win | (PS[3] & ~synchbtnC));

 assign idle = PS[0];
 assign waitpls = PS[1];
 assign game = PS[2];
 assign resettimer = PS[0] | PS[2] | PS[3];
 assign lose = PS[3];




 FDRE #(.INIT(1'b1)) Q0_FF (.C(clk), .CE(1'b1), .D(NS[0]),
.Q(PS[0]));
 FDRE #(.INIT(1'b0)) Q1_FF (.C(clk), .CE(1'b1), .D(NS[1]),
.Q(PS[1]));
 FDRE #(.INIT(1'b0)) Q2_FF (.C(clk), .CE(1'b1), .D(NS[2]),
.Q(PS[2]));
 FDRE #(.INIT(1'b0)) Q3_FF (.C(clk), .CE(1'b1), .D(NS[3]),
.Q(PS[3]));



  wire [3:0] carR, carB, carG;

  rectangle car(.startingx(cary), .startingy(carx),
.currentx(out2), .currenty(out1), .width(10'd16), .length(10'd16),
 .inputRed(4'b1111), .inputGreen(4'b1111), .inputBlue(4'b1111),
.outputRed(carR), .outputBlue(carB), .outputGreen(carG),
.flag(carflag));



  wire [3:0] redRoadR, redRoadB, redRoadG;
  wire [9:0] redroady, orroady, yelroady, violetroady, blueroady,
greenroady, indigoroady;
  wire [9:0] redlength;
  wire[9:0] redLR, orangeLR, yellowLR, greenLR, blueLR, indigoLR,
violetLR;
  wire[9:0] rngoutRed, rngoutOra, rngoutYel, rngoutGreen,
rngoutBlue, rngoutIndie, rngoutVio;
  wire [9:0] redshift, orangeshift, yellowshift, greenshift,
blueshift, indigoshift, violetshift;

 wire [3:0] orangeRoadR, orangeRoadB, orangeRoadG;
 wire [3:0] yellowRoadR, yellowRoadB, yellowRoadG;
 wire [3:0] greenRoadR, greenRoadB, greenRoadG;
 wire [3:0] blueRoadR,blueRoadB, blueRoadG;
 wire [3:0] indigoRoadR, indigoRoadB, indigoRoadG;
 wire [3:0] violetRoadR, violetRoadB, violetRoadG;
```

```
  wire [9:0] oralength, oraLR, yelLR, blueLR, greenLR, violetLR,
indigoLR, rngout;

 tenbitcounter RNG(.clk(clk), .Up(twoframe), .Dw(1'b0), .CE(1'b1),
.reset(1'b0), .LD(1'b0), .Q(rngout));

 tenbitcounter redRNG(.clk(clk), .Up(1'b0), .Dw(1'b0), .CE(1'b1),
.reset(1'b0),
                        .LD(redroady == 10'd481), .Din(rngout),
.Q(rngoutRed));

 tenbitcounter oraRNG(.clk(clk), .Up(1'b0), .Dw(1'b0), .CE(1'b1),
.reset(1'b0),
                        .LD(orroady == 10'd481), .Din(rngout),
.Q(rngoutOra));

 tenbitcounter yelRNG(.clk(clk), .Up(1'b0), .Dw(1'b0), .CE(1'b1),
.reset(1'b0),
                        .LD(yelroady == 10'd481), .Din(rngout),
.Q(rngoutYel));

 tenbitcounter greenRNG(.clk(clk), .Up(1'b0), .Dw(1'b0), .CE(1'b1),
.reset(1'b0),
                        .LD(greenroady == 10'd481), .Din(rngout),
.Q(rngoutGreen));

 tenbitcounter vlueRNG(.clk(clk), .Up(1'b0), .Dw(1'b0), .CE(1'b1),
.reset(1'b0),
                        .LD(blueroady == 10'd481), .Din(rngout),
.Q(rngoutBlue));

 tenbitcounter indigoRNG(.clk(clk), .Up(1'b0), .Dw(1'b0),
.CE(1'b1), .reset(1'b0),
                        .LD(indigoroady == 10'd481), .Din(rngout),
.Q(rngoutIndie));

 tenbitcounter violetRNG(.clk(clk), .Up(1'b0), .Dw(1'b0),
.CE(1'b1), .reset(1'b0),
                        .LD(violetroady == 10'd481), .Din(rngout),
.Q(rngoutVio));

 assign redshift = (({10{((rngoutRed >=10'd0) & (rngoutRed <=
10'd73))}} & 10'd8)
 | ({10{((rngoutRed >=10'd147) & (rngoutRed <= 10'd219))}} &
10'd16)
 | ({10{((rngoutRed >=10'd293) & (rngoutRed <= 10'd365))}} &
10'd24)
 | ({10{((rngoutRed >=10'd439) & (rngoutRed <= 10'd511))}} &
10'd32)
 | ({10{((rngoutRed >=10'd585) & (rngoutRed <= 10'd657))}} &
10'd40)
 | ({10{((rngoutRed >=10'd731) & (rngoutRed <= 10'd803))}} &
10'd48)
 | ({10{((rngoutRed >=10'd877) & (rngoutRed <= 10'd949))}} &
10'd56)
 //negative
 | ({10{((rngoutRed >=10'd73) & (rngoutRed <= 10'd146))}} & 10'd15)
 | ({10{((rngoutRed >=10'd220) & (rngoutRed <= 10'd292))}} &
10'd27)
 | ({10{((rngoutRed >=10'd366) & (rngoutRed <= 10'd438))}} &
10'd23)
 | ({10{((rngoutRed >=10'd512) & (rngoutRed <= 10'd584))}} &
10'd17)
 | ({10{((rngoutRed >=10'd658) & (rngoutRed <= 10'd730))}} &
10'd13)
 | ({10{((rngoutRed >=10'd804) & (rngoutRed <= 10'd876))}} & 10'd7)
 | ({10{((rngoutRed >=10'd950) & (rngoutRed <= 10'd1023))}} &
10'd5)

 );

  assign orangeshift = (({10{((rngoutOra >=10'd0) & (rngoutOra <=
10'd73))}} & 10'd56)
 | ({10{((rngoutOra >=10'd147) & (rngoutOra <= 10'd219))}} &
10'd48)
 | ({10{((rngoutOra >=10'd293) & (rngoutOra <= 10'd365))}} &
10'd40)
 | ({10{((rngoutOra >=10'd439) & (rngoutOra <= 10'd511))}} &
10'd32)
 | ({10{((rngoutOra >=10'd585) & (rngoutOra <= 10'd657))}} &
10'd24)
 | ({10{((rngoutOra >=10'd731) & (rngoutOra <= 10'd803))}} &
10'd17)
 | ({10{((rngoutOra >=10'd877) & (rngoutOra <= 10'd949))}} & 10'd8)
 //negative
 | ({10{((rngoutOra >=10'd73) & (rngoutOra <= 10'd146))}} & 10'd9)
 | ({10{((rngoutOra >=10'd220) & (rngoutOra <= 10'd292))}} &
10'd27)
 | ({10{((rngoutOra >=10'd366) & (rngoutOra <= 10'd438))}} &
10'd23)
 | ({10{((rngoutOra >=10'd512) & (rngoutOra <= 10'd584))}} &
10'd17)
 | ({10{((rngoutOra >=10'd658) & (rngoutOra <= 10'd730))}} &
10'd13)
 | ({10{((rngoutOra >=10'd804) & (rngoutOra <= 10'd876))}} &
10'd17)
 | ({10{((rngoutOra >=10'd950) & (rngoutOra <= 10'd1023))}} &
10'd5)

 );

  assign yellowshift = (({10{((rngoutYel >=10'd0) & (rngoutYel <=
10'd73))}} & 10'd5)
 | ({10{((rngoutYel >=10'd147) & (rngoutYel <= 10'd219))}} &
10'd13)
 | ({10{((rngoutYel >=10'd293) & (rngoutYel <= 10'd365))}} &
10'd17)
 | ({10{((rngoutYel >=10'd439) & (rngoutYel <= 10'd511))}} &
10'd23)
 | ({10{((rngoutYel >=10'd585) & (rngoutYel <= 10'd657))}} &
10'd27)
```

```
 | ({10{((rngoutYel >=10'd731) & (rngoutYel <= 10'd803))}} & 10'd7)
 | ({10{((rngoutYel >=10'd877) & (rngoutYel <= 10'd949))}} & 10'd9)
 //negative
 | ({10{((rngoutYel >=10'd73) & (rngoutYel <= 10'd146))}} & 10'd8)
 | ({10{((rngoutYel >=10'd220) & (rngoutYel <= 10'd292))}} &
10'd16)
 | ({10{((rngoutYel >=10'd366) & (rngoutYel <= 10'd438))}} &
10'd25)
 | ({10{((rngoutYel >=10'd512) & (rngoutYel <= 10'd584))}} & 10'd9)
 | ({10{((rngoutYel >=10'd658) & (rngoutYel <= 10'd730))}} &
10'd36)
 | ({10{((rngoutYel >=10'd804) & (rngoutYel <= 10'd876))}} &
10'd48)
 | ({10{((rngoutYel >=10'd950) & (rngoutYel <= 10'd1023))}} &
10'd56)

 );

  assign greenshift = (({10{((rngoutGreen >=10'd0) & (rngoutGreen
<= 10'd73))}} & 10'd8)
 | ({10{((rngoutGreen >=10'd147) & (rngoutGreen <= 10'd219))}} &
10'd5)
 | ({10{((rngoutGreen >=10'd293) & (rngoutGreen <= 10'd365))}} &
10'd16)
 | ({10{((rngoutGreen >=10'd439) & (rngoutGreen <= 10'd511))}} &
10'd7)
 | ({10{((rngoutGreen >=10'd585) & (rngoutGreen <= 10'd657))}} &
10'd24)
 | ({10{((rngoutGreen >=10'd731) & (rngoutGreen <= 10'd803))}} &
10'd13)
 | ({10{((rngoutGreen >=10'd877) & (rngoutGreen <= 10'd949))}} &
10'd32)
 //negative
 | ({10{((rngoutGreen >=10'd73) & (rngoutGreen <= 10'd146))}} &
10'd17)
 | ({10{((rngoutGreen >=10'd220) & (rngoutGreen <= 10'd292))}} &
10'd40)
```

```
 | ({10{((rngoutGreen >=10'd366) & (rngoutGreen <= 10'd438))}} &
10'd23)
 | ({10{((rngoutGreen >=10'd512) & (rngoutGreen <= 10'd584))}} &
10'd48)
 | ({10{((rngoutGreen >=10'd658) & (rngoutGreen <= 10'd730))}} &
10'd27)
 | ({10{((rngoutGreen >=10'd804) & (rngoutGreen <= 10'd876))}} &
10'd56)
 | ({10{((rngoutGreen >=10'd950) & (rngoutGreen <= 10'd1023))}} &
10'd9)

 );

  assign blueshift = (({10{((rngoutBlue >=10'd0) & (rngoutBlue <=
10'd73))}} & 10'd8)
 | ({10{((rngoutBlue >=10'd147) & (rngoutBlue <= 10'd219))}} &
10'd16)
 | ({10{((rngoutBlue >=10'd293) & (rngoutBlue <= 10'd365))}} &
10'd24)
 | ({10{((rngoutBlue >=10'd439) & (rngoutBlue <= 10'd511))}} &
10'd32)
 | ({10{((rngoutBlue >=10'd585) & (rngoutBlue <= 10'd657))}} &
10'd40)
 | ({10{((rngoutBlue >=10'd731) & (rngoutBlue <= 10'd803))}} &
10'd48)
 | ({10{((rngoutBlue >=10'd877) & (rngoutBlue <= 10'd949))}} &
10'd56)
 //negative
 | ({10{((rngoutBlue >=10'd73) & (rngoutBlue <= 10'd146))}} &
10'd9)
 | ({10{((rngoutBlue >=10'd220) & (rngoutBlue <= 10'd292))}} &
10'd27)
 | ({10{((rngoutBlue >=10'd366) & (rngoutBlue <= 10'd438))}} &
10'd23)
 | ({10{((rngoutBlue >=10'd512) & (rngoutBlue <= 10'd584))}} &
10'd17)
 | ({10{((rngoutBlue >=10'd658) & (rngoutBlue <= 10'd730))}} &
10'd13)
 | ({10{((rngoutBlue >=10'd804) & (rngoutBlue <= 10'd876))}} &
10'd7)
 | ({10{((rngoutBlue >=10'd950) & (rngoutBlue <= 10'd1023))}} &
10'd5)

 );

  assign indigoshift = (({10{((rngoutIndie >=10'd0) & (rngoutIndie
<= 10'd73))}} & 10'd8)
 | ({10{((rngoutIndie >=10'd147) & (rngoutIndie <= 10'd219))}} &
10'd16)
 | ({10{((rngoutIndie >=10'd293) & (rngoutIndie <= 10'd365))}} &
10'd24)
 | ({10{((rngoutIndie >=10'd439) & (rngoutIndie <= 10'd511))}} &
10'd32)
 | ({10{((rngoutIndie >=10'd585) & (rngoutIndie <= 10'd657))}} &
10'd40)
 | ({10{((rngoutIndie >=10'd731) & (rngoutIndie <= 10'd803))}} &
10'd48)
 | ({10{((rngoutIndie >=10'd877) & (rngoutIndie <= 10'd949))}} &
10'd56)
 //negative
 | ({10{((rngoutIndie >=10'd73) & (rngoutIndie <= 10'd146))}} &
10'd9)
 | ({10{((rngoutIndie >=10'd220) & (rngoutIndie <= 10'd292))}} &
10'd27)
 | ({10{((rngoutIndie >=10'd366) & (rngoutIndie <= 10'd438))}} &
10'd23)
 | ({10{((rngoutIndie >=10'd512) & (rngoutIndie <= 10'd584))}} &
10'd17)
 | ({10{((rngoutIndie >=10'd658) & (rngoutIndie <= 10'd730))}} &
10'd13)
 | ({10{((rngoutIndie >=10'd804) & (rngoutIndie <= 10'd876))}} &
10'd7)
 | ({10{((rngoutIndie >=10'd950) & (rngoutIndie <= 10'd1023))}} &
10'd5)
```

```
 );

  assign violetshift = (({10{((rngoutVio >=10'd0) & (rngoutVio <=
10'd73))}} & 10'd56)
 | ({10{((rngoutVio >=10'd147) & (rngoutVio <= 10'd219))}} &
10'd48)
 | ({10{((rngoutVio >=10'd293) & (rngoutVio <= 10'd365))}} &
10'd40)
 | ({10{((rngoutVio >=10'd439) & (rngoutVio <= 10'd511))}} &
10'd32)
 | ({10{((rngoutVio >=10'd585) & (rngoutVio <= 10'd657))}} &
10'd24)
 | ({10{((rngoutVio >=10'd731) & (rngoutVio <= 10'd803))}} &
10'd16)
 | ({10{((rngoutVio >=10'd877) & (rngoutVio <= 10'd949))}} & 10'd8)
 //negative
 | ({10{((rngoutVio >=10'd73) & (rngoutVio <= 10'd146))}} & 10'd5)
 | ({10{((rngoutVio >=10'd220) & (rngoutVio <= 10'd292))}} & 10'd7)
 | ({10{((rngoutVio >=10'd366) & (rngoutVio <= 10'd438))}} &
10'd13)
 | ({10{((rngoutVio >=10'd512) & (rngoutVio <= 10'd584))}} &
10'd17)
 | ({10{((rngoutVio >=10'd658) & (rngoutVio <= 10'd730))}} &
10'd23)
 | ({10{((rngoutVio >=10'd804) & (rngoutVio <= 10'd876))}} &
10'd27)
 | ({10{((rngoutVio >=10'd950) & (rngoutVio <= 10'd1023))}} &
10'd9)

 );



 tenbitcounter red3(.clk(clk), .Up(twoframe & ~synchbtnR &
synchbtnL), .Dw(twoframe & ~synchbtnL & synchbtnR), .LD(idle |
waitpls), .CE(~lose), .Din(10'd260), .reset(1'b0), .Q(redLR));
```

```
 tenbitcounter red2(.clk(clk), .Up(twoframe & (redlength <
10'd80)), .Dw(1'b0), .LD(waitpls), .Din(10'd80), .CE(~lose),
.reset(redroady == 10'd481), .Q(redlength));
 tenbitcounter red1(.clk(clk), .Up(twoframe & (redlength >=
10'd80)), .Dw(1'b0), .CE(~lose), .Din(10'd0), .reset(redroady ==
10'd481), .LD(idle | waitpls), .Q(redroady));



  wire [9:0] ASRED, ASYELLOW, ASORANGE, ASGREEN, ASBLUE, ASINDIGO,
ASVIOLET;

//  assign ASRED = ({10{~redshift[0]}} & (redLR + redshift)) |
({10{redshift[0]}} & (redLR - redshift));
//  assign ASORANGE = ({10{~orangeshift[0]}} & (oraLR +
orangeshift)) | ({10{orangeshift[0]}} & (oraLR - orangeshift));
//  assign ASYELLOW = ({10{~yellowshift[0]}} & (yelLR +
yellowshift)) | ({10{yellowshift[0]}} & (yelLR - yellowshift));
//  assign ASGREEN = ({10{~greenshift[0]}} & (greenLR +
greenshift)) | ({10{greenshift[0]}} & (greenLR - greenshift));
//  assign ASBLUE = ({10{~blueshift[0]}} & (blueLR + blueshift)) |
({10{blueshift[0]}} & (blueLR - blueshift));
//  assign ASINDIGO = ({10{~indigoshift[0]}} & (indigoLR +
indigoshift)) | ({10{indigoshift[0]}} & (indigoLR - indigoshift));
//  assign ASVIOLET = ({10{~violetshift[0]}} & (violetLR +
violetshift)) | ({10{violetshift[0]}} & (violetLR - violetshift));

  assign ASRED = ({10{~redshift[0]}} & (oraLR  + redshift)) |
({10{redshift[0]}} & (oraLR - redshift));
  assign ASORANGE = ({10{~orangeshift[0]}} & (yelLR + orangeshift))
| ({10{orangeshift[0]}} & (yelLR - orangeshift));
  assign ASYELLOW = ({10{~yellowshift[0]}} & (greenLR +
yellowshift)) | ({10{yellowshift[0]}} & (greenLR - yellowshift));
  assign ASGREEN = ({10{~greenshift[0]}} & (blueLR + greenshift)) |
({10{greenshift[0]}} & (blueLR - greenshift));
  assign ASBLUE = ({10{~blueshift[0]}} & (indigoLR + blueshift)) |
({10{blueshift[0]}} & (indigoLR - blueshift));
  assign ASINDIGO = ({10{~indigoshift[0]}} & (violetLR +
indigoshift)) | ({10{indigoshift[0]}} & (violetLR - indigoshift));
  assign ASVIOLET = ({10{~violetshift[0]}} & (redLR + violetshift))
| ({10{violetshift[0]}} & (redLR - violetshift));



 rectangle redroad(.startingx(redroady), .startingy(ASRED),
.currentx(out2), .currenty(out1), .width(redlength),
.length(roadsize),
 .inputRed(4'b1111), .inputGreen(4'b0000), .inputBlue(4'b0000),
.outputRed(redRoadR), .outputBlue(redRoadB),
.outputGreen(redRoadG), .flag(redflag));



 tenbitcounter ora3(.clk(clk), .Up(twoframe & ~synchbtnR &
synchbtnL), .Dw(twoframe & ~synchbtnL & synchbtnR), .LD(idle |
waitpls), .CE(~lose), .Din(10'd260), .reset(1'b0), .Q(oraLR));

 tenbitcounter ora2(.clk(clk), .Up(twoframe & (oralength <
10'd80)), .Dw(1'b0), .LD(waitpls), .Din(10'd80), .CE(~lose),
.reset(orroady == 10'd481), .Q(oralength));

 tenbitcounter ora1(.clk(clk), .Up(twoframe & (oralength >=
10'd80)), .Dw(1'b0), .CE(~lose), .Din(10'd80), .reset(orroady ==
10'd481), .LD(idle | waitpls), .Q(orroady));


 rectangle orangeroad(.startingx(orroady), .startingy(ASORANGE),
.currentx(out2), .currenty(out1), .width(oralength),
.length(roadsize),
 .inputRed(4'b1111), .inputGreen(4'b0111), .inputBlue(4'b0000),
.outputRed(orangeRoadR), .outputBlue(orangeRoadB),
.outputGreen(orangeRoadG), .flag(orangeflag));

  wire [9:0]  yellength;
 tenbitcounter yel3(.clk(clk), .Up(twoframe & ~synchbtnR &
synchbtnL), .Dw(twoframe & ~synchbtnL & synchbtnR), .LD(idle |
waitpls), .CE(~lose), .Din(10'd260), .reset(1'b0), .Q(yelLR));

 tenbitcounter yel2(.clk(clk), .Up(twoframe & (yellength <
10'd80)), .Dw(1'b0), .LD(waitpls), .Din(10'd80), .CE(~lose),
.reset(yelroady == 10'd481), .Q(yellength));

 tenbitcounter yel1(.clk(clk), .Up(twoframe & (yellength >=
10'd80)), .Dw(1'b0), .CE(~lose), .Din(10'd160), .reset(yelroady ==
10'd481), .LD(idle | waitpls), .Q(yelroady));



  rectangle yellowroad(.startingx(yelroady), .startingy(ASYELLOW),
.currentx(out2), .currenty(out1), .width(yellength),
.length(roadsize),
 .inputRed(4'b1111), .inputGreen(4'b1111), .inputBlue(4'b0000),
.outputRed(yellowRoadR), .outputBlue(yellowRoadB),
.outputGreen(yellowRoadG), .flag(yellowflag));

wire [9:0] greenlength;

tenbitcounter green3(.clk(clk), .Up(twoframe & ~synchbtnR &
synchbtnL), .Dw(twoframe & ~synchbtnL & synchbtnR), .LD(idle |
waitpls), .CE(~lose), .Din(10'd260), .reset(1'b0), .Q(greenLR));

tenbitcounter greea2(.clk(clk), .Up(twoframe & (greenlength <
10'd80)), .Dw(1'b0), .LD(waitpls), .Din(10'd80), .CE(~lose),
.reset(greenroady == 10'd481), .Q(greenlength));

 tenbitcounter ogree1(.clk(clk), .Up(twoframe & (greenlength >=
10'd80)), .Dw(1'b0), .CE(~lose), .Din(10'd240), .reset(greenroady
== 10'd481), .LD(idle | waitpls), .Q(greenroady));
```

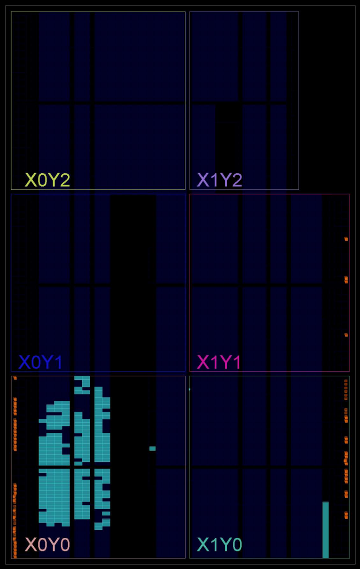
```
 rectangle greenroad(.startingx(greenroady), .startingy(ASGREEN),
.currentx(out2), .currenty(out1), .width(greenlength),
.length(roadsize),
 .inputRed(4'b0000), .inputGreen(4'b1111), .inputBlue(4'b0000),
.outputRed(greenRoadR), .outputBlue(greenRoadB),
.outputGreen(greenRoadG), .flag(greenflag));

wire [9:0] bluelength;



tenbitcounter blue3(.clk(clk), .Up(twoframe & ~synchbtnR &
synchbtnL), .Dw(twoframe & ~synchbtnL & synchbtnR), .LD(idle |
waitpls), .CE(~lose), .Din(10'd260), .reset(1'b0), .Q(blueLR));

tenbitcounter bluea2(.clk(clk), .Up(twoframe & (bluelength <
10'd80)), .Dw(1'b0), .LD(waitpls), .Din(10'd80), .CE(~lose),
.reset(blueroady == 10'd481), .Q(bluelength));

 tenbitcounter bluea1(.clk(clk), .Up(twoframe & (bluelength >=
10'd80)), .Dw(1'b0), .CE(~lose), .Din(10'd320), .reset(blueroady ==
10'd481), .LD(idle | waitpls), .Q(blueroady));




rectangle blueroad(.startingx(blueroady), .startingy(ASBLUE),
.currentx(out2), .currenty(out1), .width(bluelength),
.length(roadsize),
 .inputRed(4'b0000), .inputGreen(4'b0000), .inputBlue(4'b1111),
.outputRed(blueRoadR), .outputBlue(blueRoadB),
.outputGreen(blueRoadG), .flag(blueflag));

wire [9:0] indigolength;

tenbitcounter indig3(.clk(clk), .Up(twoframe & ~synchbtnR &
synchbtnL), .Dw(twoframe & ~synchbtnL & synchbtnR), .LD(idle |
waitpls), .CE(~lose), .Din(10'd260), .reset(1'b0), .Q(indigoLR));
```

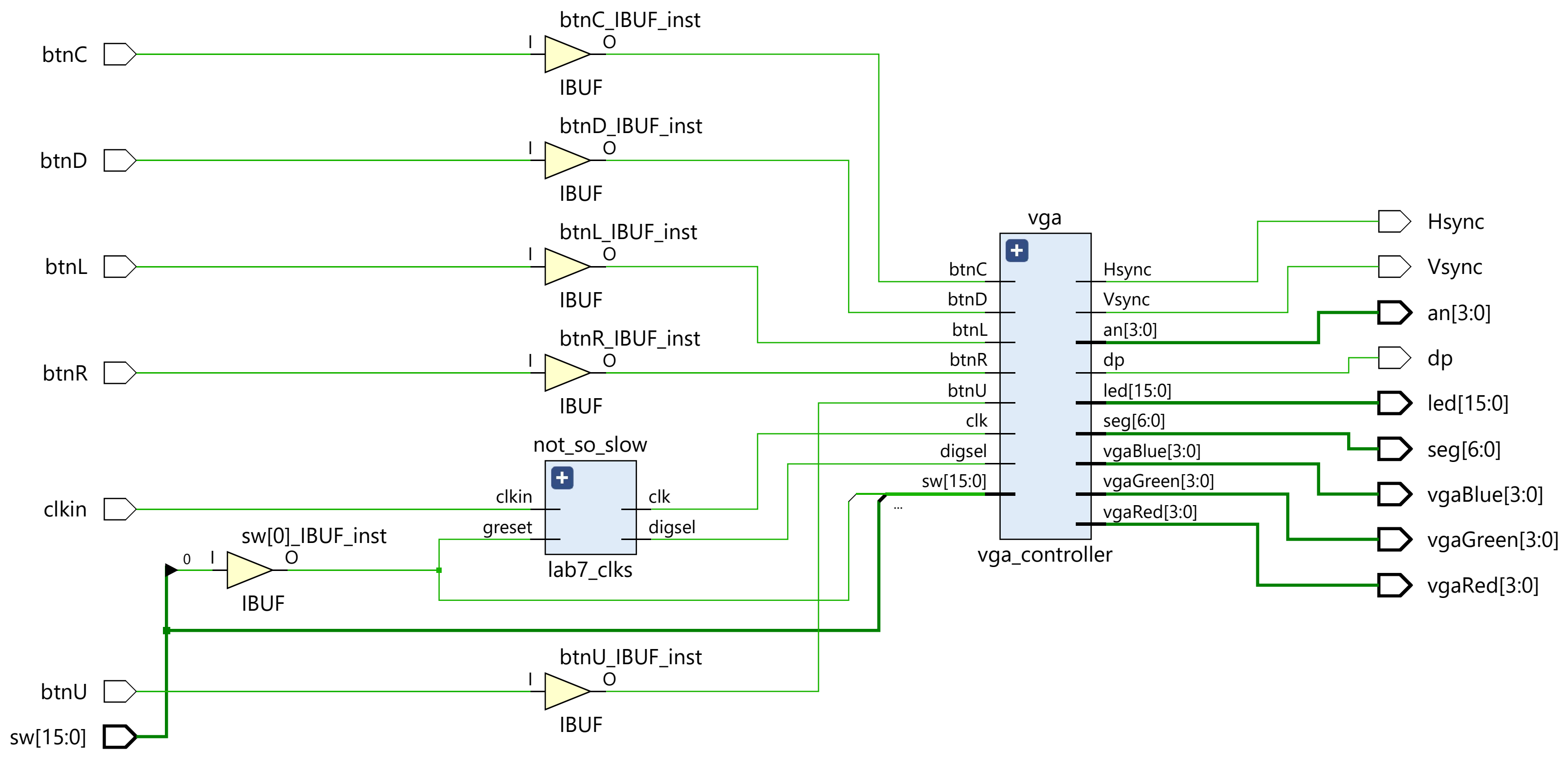
```
tenbitcounter indig2(.clk(clk), .Up(twoframe & (indigolength <
10'd80)), .Dw(1'b0), .LD(waitpls), .Din(10'd80), .CE(~lose),
.reset(indigoroady == 10'd481), .Q(indigolength));

 tenbitcounter ondie1(.clk(clk), .Up(twoframe & (indigolength >=
10'd80)), .Dw(1'b0), .CE(~lose), .Din(10'd400), .reset(indigoroady
== 10'd481), .LD(idle | waitpls), .Q(indigoroady));




    rectangle indigoroad(.startingx(indigoroady),
.startingy(ASINDIGO), .currentx(out2), .currenty(out1),
.width(indigolength), .length(roadsize),
 .inputRed(4'b0010), .inputGreen(4'b0010), .inputBlue(4'b0101),
.outputRed(indigoRoadR), .outputBlue(indigoRoadB),
.outputGreen(indigoRoadG), .flag(indigoflag));

wire [9:0] violetlength;

tenbitcounter violet3(.clk(clk), .Up(twoframe & ~synchbtnR &
synchbtnL), .Dw(twoframe & ~synchbtnL & synchbtnR), .LD(idle |
waitpls), .CE(~lose), .Din(10'd260), .reset(1'b0), .Q(violetLR));

tenbitcounter vioelt2(.clk(clk), .Up(twoframe & (violetlength <
10'd80)), .Dw(1'b0), .LD(waitpls), .Din(10'd80), .CE(~lose),
.reset(violetroady == 10'd481), .Q(violetlength));

 tenbitcounter violet1(.clk(clk), .Up(twoframe & (violetlength >=
10'd80)), .Dw(1'b0), .CE(~lose), .Din(10'd480), .reset(violetroady
== 10'd481), .LD(idle | waitpls), .Q(violetroady));

rectangle violetroad(.startingx(violetroady), .startingy(ASVIOLET),
.currentx(out2), .currenty(out1), .width(violetlength+10'd1),
.length(roadsize),
 .inputRed(4'b1000), .inputGreen(4'b0000), .inputBlue(4'b1111),
.outputRed(violetRoadR), .outputBlue(violetRoadB),
.outputGreen(violetRoadG), .flag(violetflag));
```

```
//done with rectangles

   //l1 = top left of first rectangle
   //l2 =top left of second rectangle
   //r1 = bottom right of first rectangle
   //r2= bottomright of second rectangle

   //l1.x = AS
   //l1.y = roady
   //l2.x = 400
   //l2.y = 320
   //r1.x = AS + roadsize
   //r1.y = roady + 80
   //r2.x = 416
   //r2.y = 336

   wire [9:0] carx, cary, carxx, caryy;

   assign carx = 10'd320;
   assign cary = 10'd400;


   //

   wire bool1,bool2;
   wire bool3, bool4;
   wire bool5, bool6;
   wire bool7, bool8;
   wire bool9, bool10;
   wire bool11, bool12;
   wire bool13, bool14;

   //width is fatness
   //height is tallness
   wire [9:0] FirstRecX, FirstRecY, RedRoadX, RedRoadY,
FirstRecWidth, FirstRecHeight, RoadWidth, RedRoadHeight;
   wire [9:0] OrangeRoadX, OrangeRoadY, OrangeRoadHeight;
   wire [9:0] YellowRoadX, YellowRoadY, YellowRoadHeight;
   wire [9:0] GreenRoadX, GreenRoadY, GreenRoadHeight;
   wire [9:0] BlueRoadX, BlueRoadY, BlueRoadHeight;
   wire [9:0] IndigoRoadX, IndigoRoadY, IndigoRoadHeight;
   wire [9:0] VioletRoadX, VioletRoadY, VioletRoadHeight;

   assign FirstRecX = cary;
   assign FirstRecY = carx;
   assign FirstRecWidth = 10'd16;
   assign FirstRecHeight = 10'd16;

   assign RedRoadX = redroady;
   assign RedRoadY = ASRED;
   assign RoadWidth = roadsize;
   assign RedRoadHeight = redlength;

   assign OrangeRoadX = orroady;
   assign OrangeRoadY = ASORANGE;
   assign OrangeRoadHeight = oralength;

   assign YellowRoadX = yelroady;
   assign YellowRoadY = ASYELLOW;
   assign YellowRoadHeight = yellength;

   assign GreenRoadX = greenroady;
   assign GreenRoadY = ASGREEN;
   assign GreenRoadHeight = greenlength;

   assign BlueRoadX = blueroady;
   assign BlueRoadY = ASBLUE;
   assign BlueRoadHeight = bluelength;

   assign IndigoRoadX = indigoroady;
   assign IndigoRoadY = ASINDIGO;
   assign IndigoRoadHeight = indigolength;

   assign VioletRoadX = violetroady;
   assign VioletRoadY = ASVIOLET;
   assign VioletRoadHeight = violetlength;



   assign bool1 = (((FirstRecX >= RedRoadX) & (FirstRecX <=
(RedRoadX + RedRoadHeight)))
    | ((RedRoadX >= FirstRecX) & (RedRoadX <= (FirstRecX +
FirstRecWidth))));

   assign bool2 = (((FirstRecY >= RedRoadY) & (FirstRecY <=
(RedRoadY + RoadWidth)))
   | ((RedRoadY >= FirstRecY) & (RedRoadY <= (FirstRecY +
FirstRecHeight))));

   //red

   assign bool3 = (((FirstRecX >= OrangeRoadX) & (FirstRecX <=
(OrangeRoadX + OrangeRoadHeight)))
    | ((OrangeRoadX >= FirstRecX) & (OrangeRoadX <= (FirstRecX +
FirstRecWidth))));

   assign bool4 = (((FirstRecY >= OrangeRoadY) & (FirstRecY <=
(OrangeRoadY + RoadWidth)))
   | ((OrangeRoadY >= FirstRecY) & (OrangeRoadY <= (FirstRecY +
FirstRecHeight))));



   //orange

   assign bool5 = (((FirstRecX >= YellowRoadX) & (FirstRecX <=
(YellowRoadX + YellowRoadHeight)))
    | ((YellowRoadX >= FirstRecX) & (YellowRoadX <= (FirstRecX +
FirstRecWidth))));
```

```
   assign bool6 = (((FirstRecY >= YellowRoadY) & (FirstRecY <=
(YellowRoadY + RoadWidth)))
   | ((YellowRoadY >= FirstRecY) & (YellowRoadY <= (FirstRecY +
FirstRecHeight))));

   //yellow

   assign bool7 = (((FirstRecX >= GreenRoadX) & (FirstRecX <=
(GreenRoadX + GreenRoadHeight)))
    | ((GreenRoadX >= FirstRecX) & (GreenRoadX <= (FirstRecX +
FirstRecWidth))));

   assign bool8 = (((FirstRecY >= GreenRoadY) & (FirstRecY <=
(GreenRoadY + RoadWidth)))
   | ((GreenRoadY >= FirstRecY) & (GreenRoadY <= (FirstRecY +
FirstRecHeight))));

   //green

   assign bool9 = (((FirstRecX >= BlueRoadX) & (FirstRecX <=
(BlueRoadX + BlueRoadHeight)))
    | ((BlueRoadX >= FirstRecX) & (BlueRoadX <= (FirstRecX +
FirstRecWidth))));

   assign bool10 = (((FirstRecY >= BlueRoadY) & (FirstRecY <=
(BlueRoadY + RoadWidth)))
   | ((BlueRoadY >= FirstRecY) & (BlueRoadY <= (FirstRecY +
FirstRecHeight))));

   //blue

   assign bool11 = (((FirstRecX >= IndigoRoadX) & (FirstRecX <=
(IndigoRoadX + IndigoRoadHeight)))
    | ((IndigoRoadX >= FirstRecX) & (IndigoRoadX <= (FirstRecX +
FirstRecWidth))));

   assign bool12 = (((FirstRecY >= IndigoRoadY) & (FirstRecY <=
(IndigoRoadY + RoadWidth)))
   | ((IndigoRoadY >= FirstRecY) & (IndigoRoadY <= (FirstRecY +
FirstRecHeight))));

   //indigo

   assign bool13 = (((FirstRecX >= VioletRoadX) & (FirstRecX <=
(VioletRoadX + VioletRoadHeight)))
    | ((VioletRoadX >= FirstRecX) & (VioletRoadX <= (FirstRecX +
FirstRecWidth))));

   assign bool14 = (((FirstRecY >= VioletRoadY) & (FirstRecY <=
(VioletRoadY + RoadWidth)))
   | ((VioletRoadY >= FirstRecY) & (VioletRoadY <= (FirstRecY +
FirstRecHeight))));

   //violet



   assign win = (bool1 & bool2) | (bool3 & bool4) | (bool5 & bool6)
    | (bool7 & bool8) | (bool9 & bool10) | (bool11 & bool12) |
(bool13 & bool14);



   wire [4:0] waitcar, waitcar2;

   assign waitcar = ~{4{(((timeoutput > 10'd1) & (timeoutput <
10'd31)) | ((timeoutput2 > 10'd1) & (timeoutput2 < 10'd31)) |
((timeoutput > 10'd61) & (timeoutput < 10'd91)))}};
   assign waitcar2 = ~{4{((timeoutput2 > 10'd1) & (timeoutput2 <
10'd31))}};

   assign vgaRed = (carR & (waitcar  )) | redRoadR | orangeRoadR |
yellowRoadR | greenRoadR | blueRoadR | indigoRoadR | violetRoadR;
   assign vgaBlue = (carB & (waitcar)) |redRoadB | orangeRoadB |
yellowRoadB |  greenRoadB | blueRoadB | indigoRoadB | violetRoadB;
   assign vgaGreen = (carG & (waitcar)) | redRoadG | orangeRoadG |
yellowRoadG |  greenRoadG | blueRoadG | indigoRoadG | violetRoadG;

    wire [9:0] ledout, ledout2;
    wire [15:0] scoreoutput;

//    tenbitcounter scorecounter(.clk(clk), .Up(1'b1), .Dw(1'b0),
.CE(frame & game), .reset(scorecount == 10'd17), .Q(scorecount));
//    tenbitcounter score(.clk(clk), .Up(1'b1), .Dw(1'b0),
.CE(scorecount == 10'd16), .LD(1'b0), .reset(idle | waitpls),
.Din(1'b0), .Q(scoreoutput1));

//    tenbitcounter bribby(.clk(clk), .Up(twoframe), .Dw(1'b0),
.CE(1'b1), .reset(1'b0), .Q(scoreoutput1));
//    assign Ninput[15:0] = scoreoutput;

//    wire tt;
//    tenbitcounter framecounter(.Up(game), .Dw(1'b0), .LD(1'b0),
.Din(10'b0),
//    .clk(clk), .CE(1'b1), .Q(frameoutput), .reset(tt));

//    tenbitcounter scorecounter(.Up(1'b1), .Dw(1'b0), .LD(1'b0),
.Din(10'b0),
//    .clk(clk), .CE(tt), .Q(scoreoutput), .reset(1'b0));

//    assign tt = (frameoutput == 10'd17);

    wire sixframe;

    tenbitcounter whatever(.clk(clk), .Up(frame), .Dw(1'b0),
.LD(1'b0), .CE(~lose), .reset(sixframe), .sixframe(sixframe));

    sixteenbitcounter bribby(.clk(clk), .Up(sixframe), .CE(~lose),
.reset(idle | waitpls), .Q(scoreoutput));

    tenbitcounter ledshift(.clk(clk), .Up(frame), .Dw(1'b0),
.LD(1'b0), .CE(1'b1), .reset(idle| lose | waitpls | (ledout ==
10'd60)), .Q(ledout));
```

```
    tenbitcounter ledshift2(.clk(clk), .Up(frame), .Dw(1'b0),
.LD(1'b0), .CE(1'b1), .reset(idle| lose |(ledout2 == 10'd120)),
.Q(ledout2));

    assign Ninput[15:0] = scoreoutput;



    assign led[15] = (((ledout >=10'd0) & (ledout <= 10'd15)) &
game) | (waitpls & (((ledout2 >=10'd0) & (ledout2 <= 10'd120))));
    assign led[11] = (((ledout >=10'd0) & (ledout <= 10'd15)) &
game)| (waitpls & (((ledout2 >=10'd30) & (ledout2 <= 10'd120))));
    assign led[7] = (((ledout >=10'd0) & (ledout <= 10'd15)) &
game)| (waitpls & (((ledout2 >=10'd60) & (ledout2 <= 10'd120))));
    assign led[3] = (((ledout >=10'd0) & (ledout <= 10'd15)) &
game)| (waitpls & (((ledout2 >=10'd90) & (ledout2 <= 10'd120))));

    assign led[14] = (((ledout >=10'd16) & (ledout <= 10'd30)) &
game);
    assign led[10] = (((ledout >=10'd16) & (ledout <= 10'd30)) &
game);
    assign led[6] = (((ledout >=10'd16) & (ledout <= 10'd30)) &
game);
    assign led[2] = (((ledout >=10'd16) & (ledout <= 10'd30)) &
game);

    assign led[13] = (((ledout >=10'd31) & (ledout <= 10'd45)) &
game) ;
    assign led[9] = (((ledout >=10'd31) & (ledout <= 10'd45)) &
game);
    assign led[5] = (((ledout >=10'd31) & (ledout <= 10'd45)) &
game);
    assign led[1] = (((ledout >=10'd31) & (ledout <= 10'd45)) &
game);

    assign led[12] = (((ledout >=10'd46) & (ledout <= 10'd60)) &
game) ;
    assign led[8] = (((ledout >=10'd46) & (ledout <= 10'd60)) &
game);
    assign led[4] = (((ledout >=10'd46) & (ledout <= 10'd60)) &
game);
    assign led[0] = (((ledout >=10'd46) & (ledout <= 10'd60)) &
game);

    tenbitcounter timecounterlose(.clk(clk), .Up(frame & lose),
.Dw(1'b0), .LD(1'b0), .CE(1'b1),
 .reset(timeoutput2 == 10'd60 | ~lose), .Q(timeoutput2));



    ringcounter r1(.advance(digsel), .clk(clk), .o(ringo));

    selector s1(.sel(ringo), .N(Ninput), .H(seloutput));

    hex7seg sevenseg(.n0(seloutput[0]), .n1(seloutput[1]),
.n2(seloutput[2]) ,.n3(seloutput[3]), .segment(seg));

    assign an[3] = ~(ringo[3] & ~((timeoutput2 > 10'd0) &
(timeoutput2 < 10'd30)));
    assign an[2] = ~(ringo[2] & ~((timeoutput2 > 10'd0) &
(timeoutput2 < 10'd30)));
    assign an[1] = ~(ringo[1] & ~((timeoutput2 > 10'd0) &
(timeoutput2 < 10'd30)));
    assign an[0] = ~(ringo[0] & ~((timeoutput2 > 10'd0) &
(timeoutput2 < 10'd30)));

endmodule
```


X0Y2
X1Y2
X0Y1
X1Y1
X0Y0
X1Y0


btnC
btnD
btnL
btnR
clkin
btnU
sw[15:0]
btnC_IBUF_inst
I
O
IBUF
btnD_IBUF_inst
btnL_IBUF_inst
btnR_IBUF_inst
not_so_slow
clkin
greset
clk
digsel
lab7_clks
sw[0]_IBUF_inst
0
btnU_IBUF_inst
vga
btnC
btnD
btnL
btnR
btnU
clk
digsel
sw[15:0]
Hsync
Vsync
an[3:0]
dp
led[15:0]
seg[6:0]
vgaBlue[3:0]
vgaGreen[3:0]
vgaRed[3:0]
vga_controller
Hsync
Vsync
an[3:0]
dp
led[15:0]
seg[6:0]
vgaBlue[3:0]
vgaGreen[3:0]
vgaRed[3:0]