Terrence Ng

CSE 100L

Section C

3/10/2020

Lab 7 Write Up

Description:

The goal of this lab is to create a driving game whose rainbow colored road is randomly generated. The road's width is changed using switches and is moved left or right using the L and R buttons. The player's goal is to stay on the road for as long as possible, his/her score will be constantly displayed during the game and flash at the end. The purpose of this lab is allow the class to use everything that they have learned throughout the quarter to build something completely out of their own design.

Methods:

I began by first creating my vertical counter (Vcounter) and horizontal counter (Hcounter), for detecting the Vsync and Hsync, I then tested them using the TestSync provided.

Vcounter

To create the Vcounter, I used 4, 4 bit up counters (count_4b). Each count_4b starts incrementing once the one before it is full (1111). In addition to the 16 bit output of the current counter value, the Vcounter also outputs a signal called Vsync that is low when the counter's value is in the range of 489 to 490 and high when the counter is outside of that range. The Vcounter is used to count every pixel on the vertical axis of the active region.

Hcounter

To create the Hcounter, I used 4, 4 bit up counters (count_4b). Each count_4b starts incrementing once the one before it is full (1111). In addition to the 16 bit output of the current counter value, the Hcounter also outputs a signal called Hsync that is low when the counter's value is in the range of 655 to 750 and high when the counter is outside of that range. The Hcounter is used to count every pixel on the horizontal axis of the active region.

Block

Next, I created my block module which is used to display blocks of color on the screen, which later became the car and road segments. To create the block module, I took in 9 inputs and 3 outputs. The inputs were: the 3 RGB values that I wanted the block to be, the x and y

coordinate of the current pixel, and the min and max range of the x and y coordinates that I wanted the block to appear in. The outputs were the RGB values that would be displayed in the region outlined by the min and max ranges of the x and y values. I first tested my block module by just having a solid static square on the screen. After successfully creating a static square, I slowly added more functionality to the game through the use of my block module in my top level, including making a block scroll top to bottom continuously, changing the width, and shifting left and right without going off the screen.

set_random

Once I had my road properly moving vertically, I worked on making my road randomly generated. To do this I created my set_random module which takes in the output of the LSFR, as an input, and outputs a 16 bit bus containing the random value to offset my road segments by. I used 16 flip flops to save an LSFR value and output the offset. The first 3 flip flops are set to 0, the 4th-6th flip flops are assigned the 1st-3rd bit of the LSFR output and the 7th-16th flip flops are assigned the 4th bit of the LSFR output. The reason for this is because the offsets are -56, -48, -40, -32, -24, -16, -8, 0, 8, 16, 24, 32, 40, 48, or 56 pixels from the center of the road segment just below it. All of these numbers contain zero in their first 3 bits, the only difference being their 4th to 6th bits (**Figure A**). The only difference between a negative and positive binary number is that a negative number is preceded with 1's and a positive number is preceded with 0's, sign extending the random offset with a random bit includes negatives in the random pool.

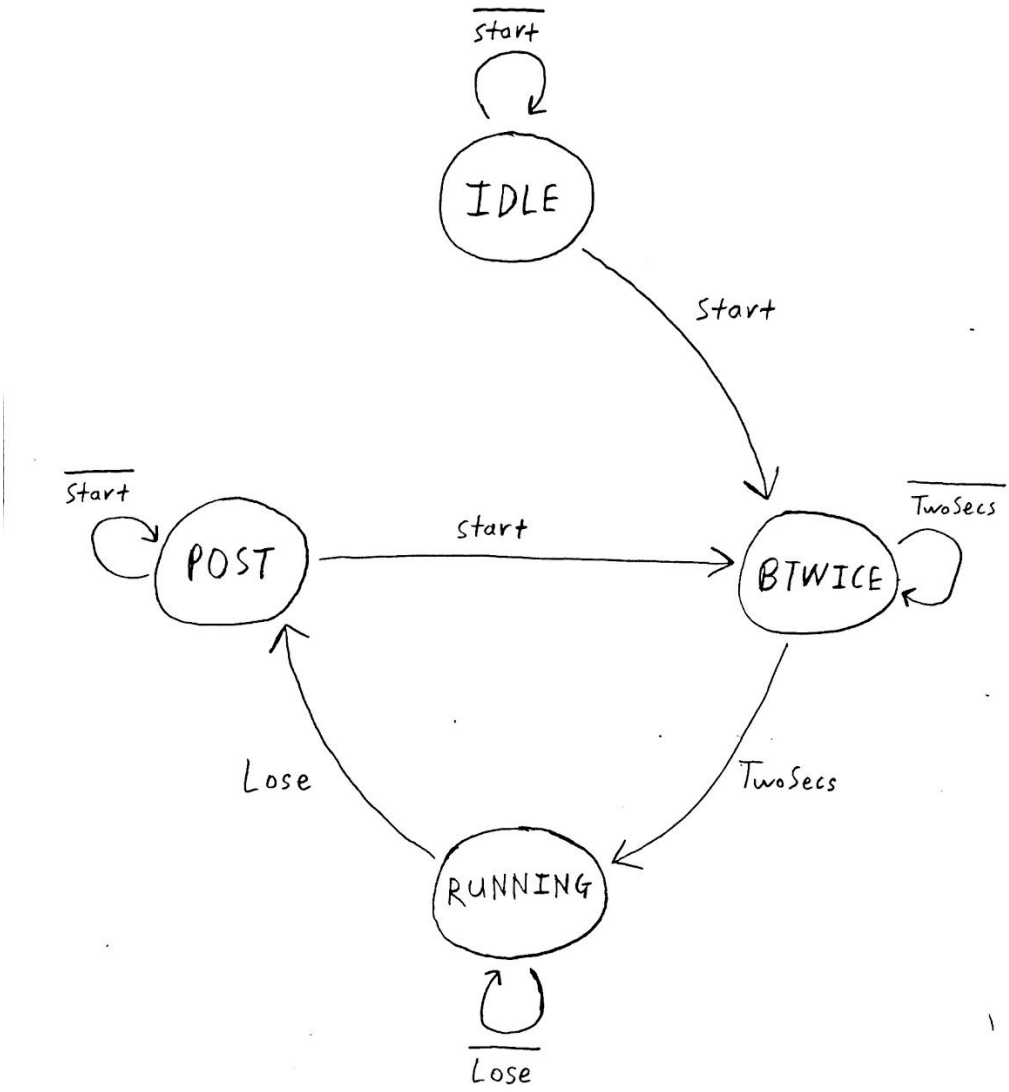
Figure A

Decimal	Binary					
0	0	0	0	0	0	0
8	0	0	1	0	0	0
16	0	1	0	0	0	0
24	0	1	1	0	0	0
32	1	0	0	0	0	0
40	1	0	1	0	0	0
48	1	1	0	0	0	0
56	1	1	1	0	0	0

State Machine

After I created my road, I made the state machine (**Figure B**) for my game as well as the conditions for each state. I used a Mealy machine. I then used my design to create the logic for entering each state as well as for the outputs of the state machine.

Figure B



IDLE

The machine is in state IDLE at the beginning when the player has not initially started the game yet. This is the only time the machine is in IDLE state and does not enter it after leaving this state. The game starts in this state, as well as after leaving state POST.

- next_IDLE = IDLE & ~start
- game_start = IDLE | POST & start

BTWICE

The machine is in state BTWICE after the player has started (from IDLE) or restarted (from POST) the game. The machine will remain in this state for 2 seconds and while in this state, the car will blink every quarter second.

- next_BTWICE = IDLE & start | POST & start | BTWICE & ~TwoSecs
- blink = BTWICE & ~TwoSecs

RUNNING

The machine is in state RUNNING after the player has started the game and the car has blinked for 2 seconds. The machine will remain in this state until the player has lost the game. If the player has not lost, the game is still running.

- next_RUNNING = BTWICE & TwoSecs | RUNNING & ~lose
- game_running = RUNNING & ~lose

POST

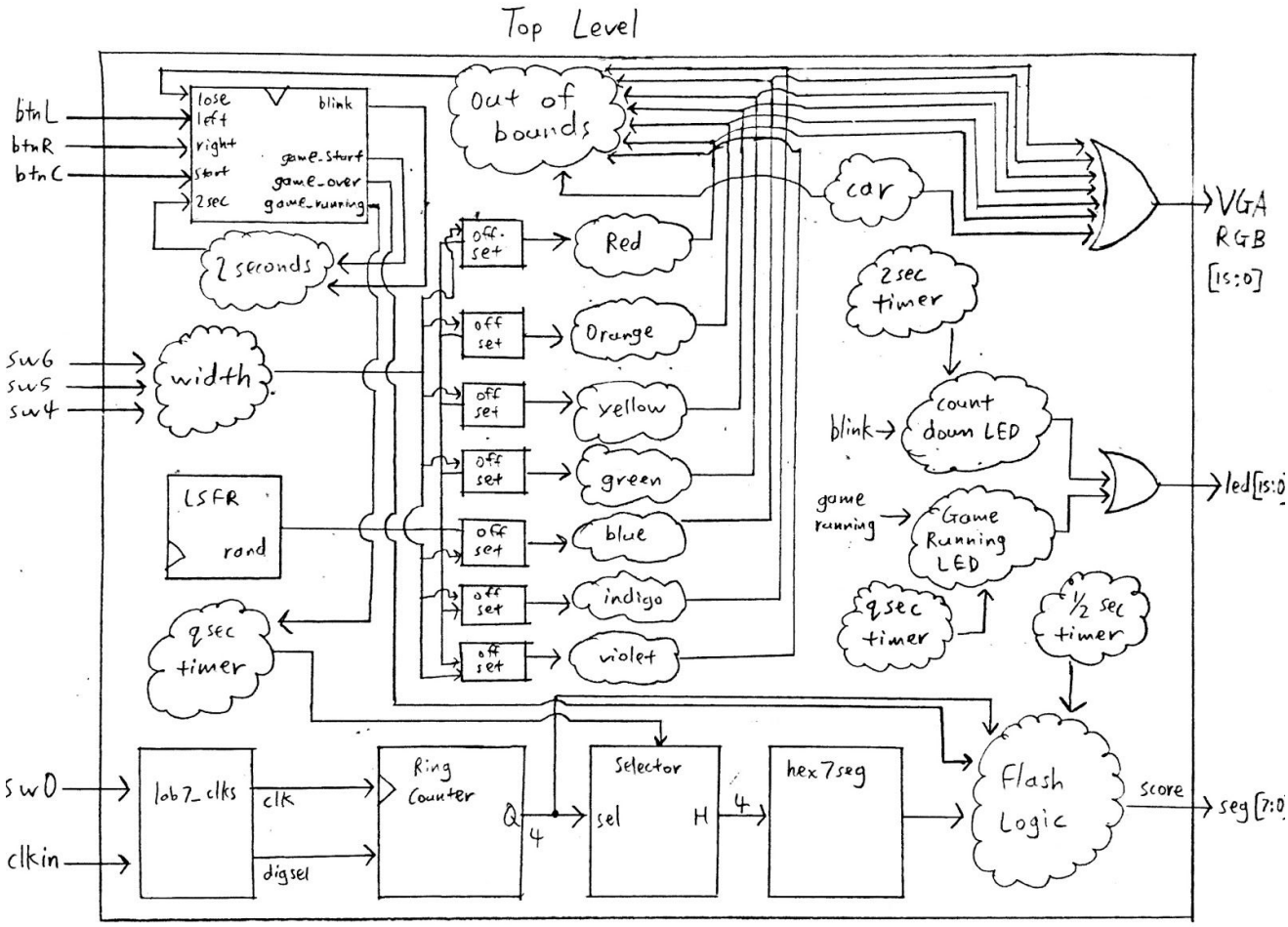
The machine is in state POST after the player has lost the game. The machine will remain in this state until the game is restarted by the player. If the player has not restarted the game, the game will remain over.

- next_POST = RUNNING & lose | POST & ~start
- game_over = POST & ~start

Top Level

To create the top level (**Figure C**), I first used 2 edge detectors to detect when the Vsync is high or low. This is because the Vsync is high and low during 1 frame, using this I can later make my blocks move 2 pixels per frame. Next I setup my Hcounter and Vcounter. I made my Hcounter count constantly and reset when it reaches 799. I made my Vcounter count every time my Hcounter reached 799 and reset when the Vcounter reaches 524 and the Hcounter reaches 799, this way the pixels are traversed left to right, one row at a time, filling out the entire active region with pixels all the way to the bottom right corner.

Figure C



State Machine

Next I set up my state machine and wired Button C to input start, TwoSecs to output TwoSecs, lose to output lose, blink to output blink, game_start to output game_start, game_running to output game_running, and game_over to output game_over. I

I then used a 16 bit counter (countUD16L) to count up every frame to 64 and reset back to 0 once it reaches 64 or the game has started (game_start). I then used a Time counter to count up every time the frame counter counts to 64 and reset when the game starts. The TwoSecs wire is high when the time counter has counted 2 seconds. This allows the car time to blink for 2 seconds before the game begins after the player presses start (button C).

Road

Next I create my road, first I make 2, 16 bit busses called i and width. The first 3 bits of i are controlled by switches 4 to 6, the rest are set to 0. Next I set width to 8 + 16 * i, this way the width of the road segment can range from 8 to 120 pixels depending on the switches.

I then make my road segments one at a time, basing each one's coordinates off the coordinates of the one before it. For example, to create my RED road segment. I use a 16 bit counter to count the Y coordinate of my RED road segment. It counts up every 2 frames while the game is running and is set to 0 when the game starts or its spawn point, in this case, 0 when the counter reaches 560. This way it will be in the right Y position at the start of the game as well as when it reaches the bottom and wraps around. The reason the counter counts past the active region is because 6 blocks perfectly cover the active region, so a 7th is needed to seamlessly wrap around the screen.

I then use a Set_Random to obtain an offset for the X position of the RED road segment. I then create the spawn point on the X position for the RED road segment. The spawn point is the position of the road segment below it, in this case ORANGE. If the RED road segment spawns in a place where any part of it is off the screen, the spawn point instead becomes the same as the spawn point of the road segment below it, in this case ORANGE. This way, the road is unable to spawn off the screen.

Next I use a 16 bit counter to count the X coordinate of my RED road segment. It counts up (right) 2 times per frame while the game is running and only button R is pressed and the road segment is not at the edge of the screen, it counts down (left) 2 times per frame while the game is running and only button L is pressed and the road segment is not at the edge of the screen. It loads in the center X coordinate of the screen when the game starts or the spawn point when the road segment above it, in this case VIOLET, reaches the bottom of the active region, meaning the RED road segment is now off the screen.

The outputs of the X and Y counters are then sent to the block module which also takes in the color, in this case RED and uses the bottom center of a road segment as the reference point. The X range is the width, so the Xmin value is the X position - width/2 and the Xmax value is the X position + width/2. The Y range is 80, the Ymax value is the Y position itself and the Ymin value is the Ymax value minus 80 if the Ymax value is greater than or equal to 80, otherwise it is 0. This is so that the road segment can scroll from the top smoothly.

- spawn = ((below_X + offset > width/16'd2) && (below_X + offset + width/16'd2 < 16'd639)) ? below_X + offset : below_X

I then repeat this 6 times, to create a total of 7 road segments, each a different color and each basing its coordinates off the coordinates of the road segment before it.

<u>Car</u>

After making my road, I make the car using a block module. Because the car never moves, I set the X and Y values and ranges to static values. I then create a wire called flash that is high when the game is running or high every second (based off of the 64 frame counter made earlier) when the game is over or blink is high, so that the car can blink.

- flash = ((blink | game_over) & (frame64[15:0] < 16'd32)) | (~blink & ~game_over)

To get the colors to show on the monitor for all the road segments as well as the car, every RGB block module output of every road segment including the car and flash is OR'd and assigned to the VGA outputs. This way, the colors can be displayed and the car can flash and won't interfere with the road colors. Another reason it is done this way is because you can't wire more than one thing to the same output.

- vgaRed[3] = red1[3] | red2[3] | red3[3] | red4[3] | red5[3] | red6[3] | red7[3] | (car_red[3] & flash)
- vgaRed[2] = red1[2] | red2[2] | red3[2] | red4[2] | red5[2] | red6[2] | red7[2] | (car_red[2] & flash)
- vgaRed[1] = red1[1] | red2[1] | red3[1] | red4[1] | red5[1] | red6[1] | red7[1] | (car_red[1] & flash)
- vgaRed[0] = red1[0] | red2[0] | red3[0] | red4[0] | red5[0] | red6[0] | red7[0] | (car_red[0] & flash)

- vgaGreen[3] = green1[3] | green2[3] | green3[3] | green4[3] | green5[3] | green6[3] | green7[3] | (car_green[3] & flash)
- vgaGreen[2] = green1[2] | green2[2] | green3[2] | green4[2] | green5[2] | green6[2] | green7[2] | (car_green[2] & flash)
- vgaGreen[1] = green1[1] | green2[1] | green3[1] | green4[1] | green5[1] | green6[1] | green7[1] | (car_green[1] & flash)
- vgaGreen[0] = green1[0] | green2[0] | green3[0] | green4[0] | green5[0] | green6[0] | green7[0] | (car_green[0] & flash)

- vgaBlue[3] = blue1[3] | blue2[3] | blue3[3] | blue4[3] | blue5[3] | blue6[3] | blue7[3] | (car_blue[3] & flash)
- vgaBlue[2] = blue1[2] | blue2[2] | blue3[2] | blue4[2] | blue5[2] | blue6[2] | blue7[2] | (car_blue[2] & flash)
- vgaBlue[1] = blue1[1] | blue2[1] | blue3[1] | blue4[1] | blue5[1] | blue6[1] | blue7[1] | (car_blue[1] & flash)

- vgaBlue[0] = blue1[0] | blue2[0] | blue3[0] | blue4[0] | blue5[0] | blue6[0] | blue7[0] | (car_blue[0] & flash)

Lose Condition

Next I create my lose condition. To do this, I create 4 equations to check if the car is within one of the edges of a Road segment. ANDing these 4 equations together, I get the condition if the car is touching one road segment. ORing 7 of these together, 1 for each road segment's X and Y coordinates, I get the condition if the car is touching at least 1 road segment. If the value of that condition is 0, that means the car is no longer on the road and therefore the player loses.

- (X < 16'd328 + width/16'd2)
- (X + width/16'd2 >16'd312)
- (Y < 16'd488)
- (Y > 16'd392)

Score

I then create the score and make it display on the hex7seg. I first use a 16 bit counter to count every frame while the game is running and reset to 0 when the game starts or the counter reaches 16. I then use a time counter that increments every time the 16 frame counter reaches 16 and resets at the start of the game, this way the time counter increments every quarter second.

Next I set up the ring counter, selector, and hex7seg. The ring counter takes in digsel and clk as inputs and outputs a 4 bit bus that determines which anode is currently lit up. The selector takes the outputs of the ring counter, and quarter second time counter and outputs a 4 bit bus to the hex7seg which displays the corresponding hexadecimal number.

I then use another 16 bit counter that counts every frame while the game is over and resets to 0 after reaching 64. To make the anodes flash at the end of the game, I assign each anode to its corresponding ring counter along with the game_over wire and the condition that the output of the 64 frame counter just created is less than 32, that way the anodes can flash every half second while the game is over.

- an[3] = ~(rings[3] & ((game_over & (scoreFlash[15:0] < 16'd32)) | ~game_over));
- an[2] = ~(rings[2] & ((game_over & (scoreFlash[15:0] < 16'd32)) | ~game_over));
- an[1] = ~(rings[1] & ((game_over & (scoreFlash[15:0] < 16'd32)) | ~game_over));
- an[0] = ~(rings[0] & ((game_over & (scoreFlash[15:0] < 16'd32)) | ~game_over));

LEDS

To create the LEDs, I first use 2 counters to make a 64 frame counter and a 128 frame counter.

To create the Countdown LED's during the BTWICE state, I set LED 15, 11, 7 and 3 to be high when the blink is high and a half second has passed since the LED before it has lit up. I made LED 15 high when blink is high and the 128 frame counter is greater than 0, LED 11 high when blink is high and the 128 frame counter is greater than 32, LED 7 high when blink is high and the 128 frame counter is greater than 64, and LED 3 high when blink is high and the 128 frame counter is greater than 96. This way the 4 LED's will light up every half second in sync with the 2 second blink period before the game starts.

To create the game running LED's during the running state. I set LED's 15, 11, 7, and 3 to be high when the game is running and the 64 frame counter is between 0 and 16. I set LED's 14, 10, 6, and 2 to be high when the game is running and the 64 frame counter is between 17 and 32. I set LED's 13, 9, 5, and 1 to be high when the game is running and the 64 frame counter is between 33 and 49. I set LED's 12, 8, 4, and 0 to be high when the game is running and the 64 frame counter is between 50 and 64. This way the LED's can have 4 lit up at a time and shift once to the right every quarter second.

I OR both of these conditions (for Countdown and game running LED's) together before assigning them to the LED's.

countUD16L

I used the same countUD16L that I made in lab 4.

countUD4L

I used the same countUD4L that I made in lab 4.

Count_4b

I used the same Count_4b that I made in Lab 5.

Edge_detector

I used the same Edge_detector that I made in Lab 4.

LSFR

I used the same LSFR that I made in Lab 5.

Time Counter

I used the same time counter that I made in Lab 5.

Ring Counter

I used the same ring counter that I made in Lab 4.

Selector

I used the same selector that I made in Lab 4.

hex7seg

I used the same hex7seg that I made in lab 2.

Results:

The result of my design was successful. After testing and debugging my code and logic, the game functioned as specified in the lab manual. I tested my design to make sure it worked by simulating it as well as some of it's inner modules such as the state machine and Vsync/Hsync. I also tested my design on the basys3 board by playing around with the buttons and switches and observing my project's behaviour on the monitor. The maximum clock frequency (MHz) at which my design will operate based on the Timing Report (**Figure D**) is $\frac{1}{Actual\ period - WNS} = \frac{1}{40ns - 24.261ns} = 0.0635$ns.

Figure D

Clock Summary

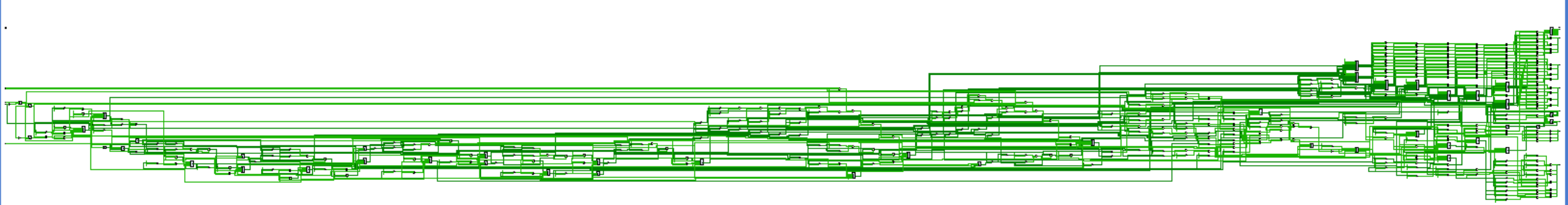
Name	Waveform	Period (ns)	Frequency (MHz)
sys_clk_pin	{0.000 5.000}	10.000	100.000
clk_out1_clk_wiz_0	{0.000 20.000}	40.000	25.000
clkfbout_clk_wiz_0	{0.000 5.000}	10.000	100.000

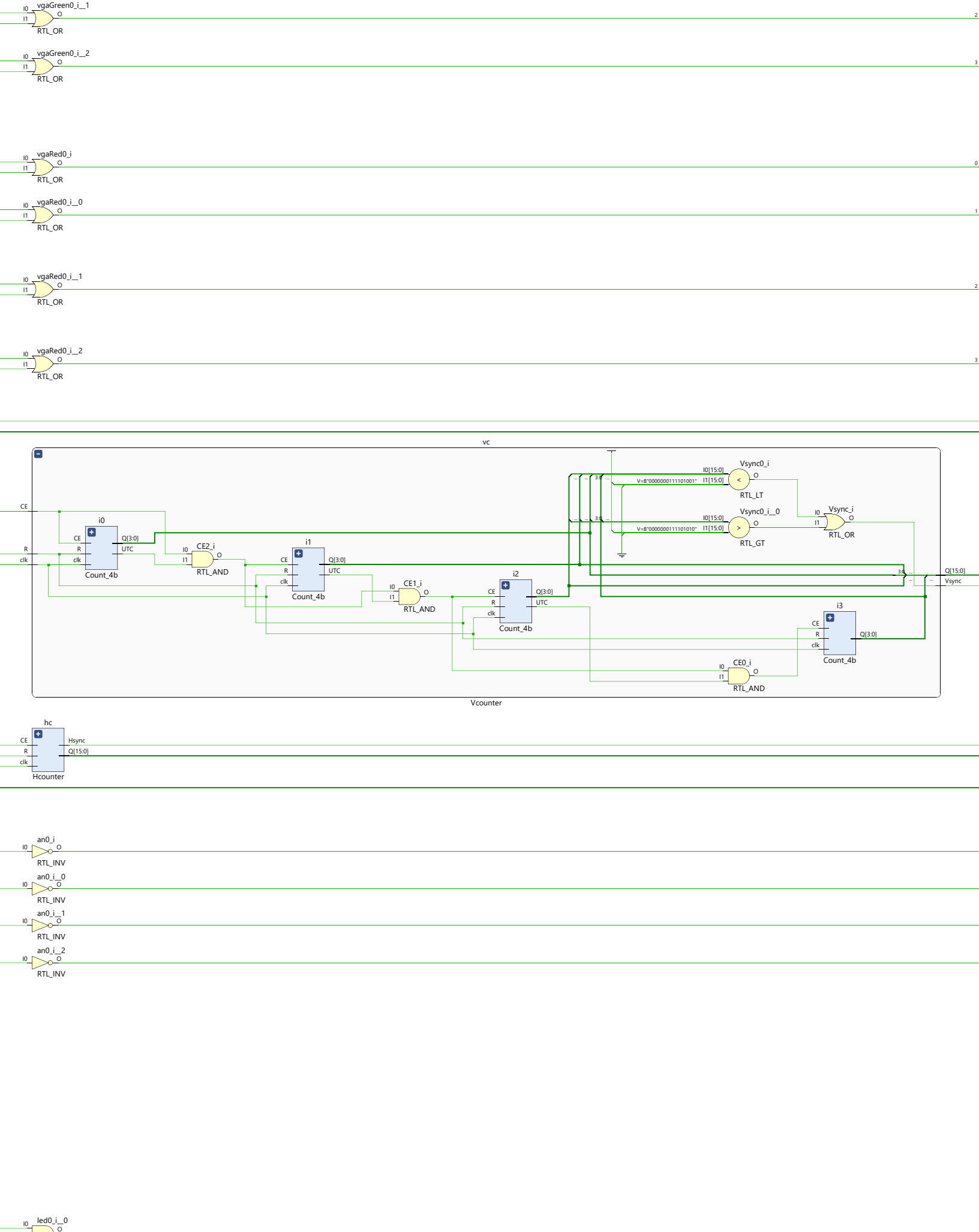
I tested my State machine by simulating it to make sure it went through or stayed in all the states depending on various conditions. I tried testing my top level by simulating it but I was told that simulating my top level just wouldn't be practical, so instead I debugged my top level partially by assigning my states to LEDs to make sure that I am getting to the right states in my top level. I tested my Vcounter and Hcounter using the provided TestSync simulation to make sure that my Vsync and Hsync were low in the correct rows/columns, they were. Vsync was low in 489-490 and Hsync was low in 655-750.

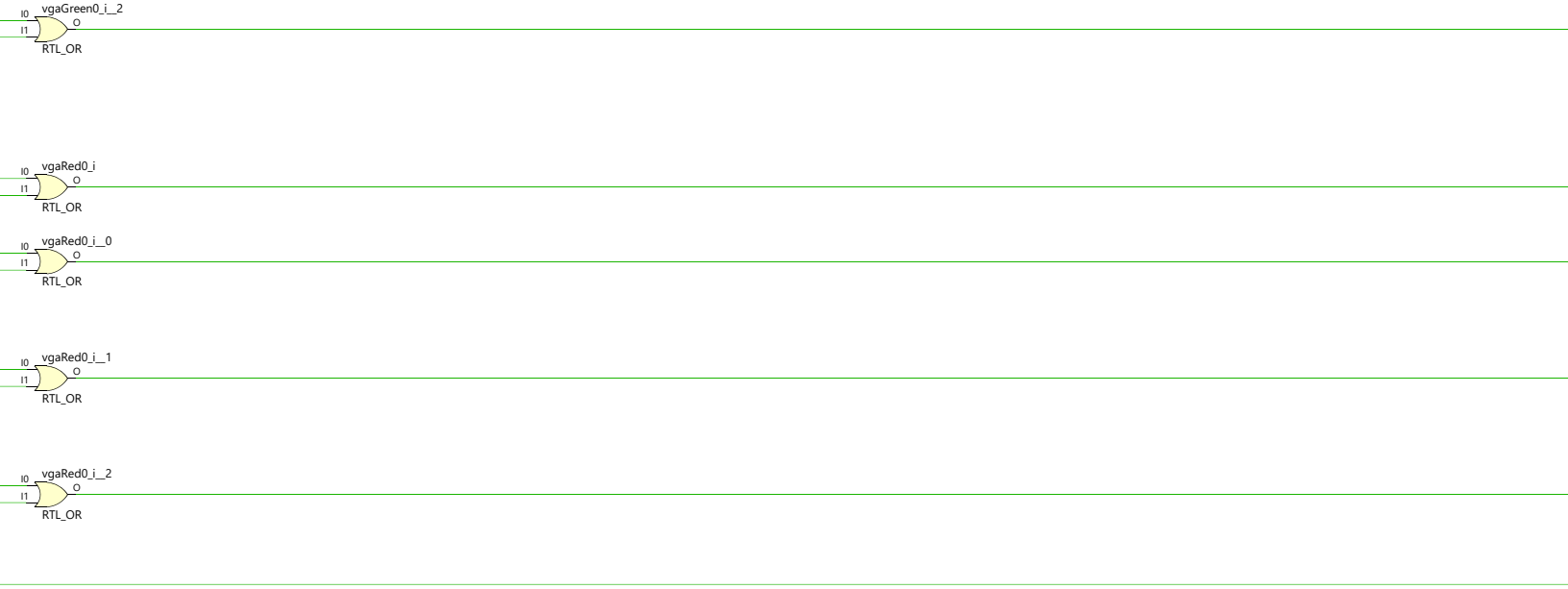
Some problems that I discovered during testing was that I would keep generating the same offset for each road segment, I resolved this problem by getting 7 offsets instead of just 1. I also could not scroll all 7 road segments properly without having 1 be static, I resolved this problem by using a ternary to set the road segments to a initialized value at the start of the game and a random point during the game.

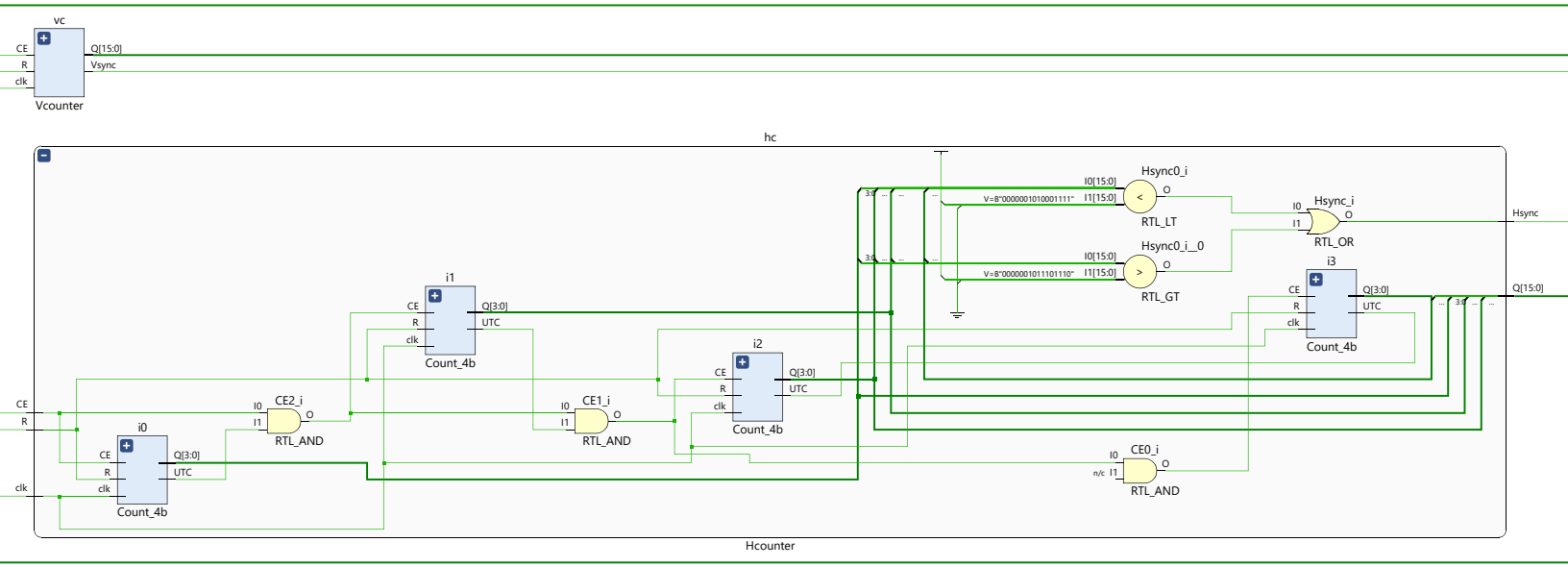
Conclusion:

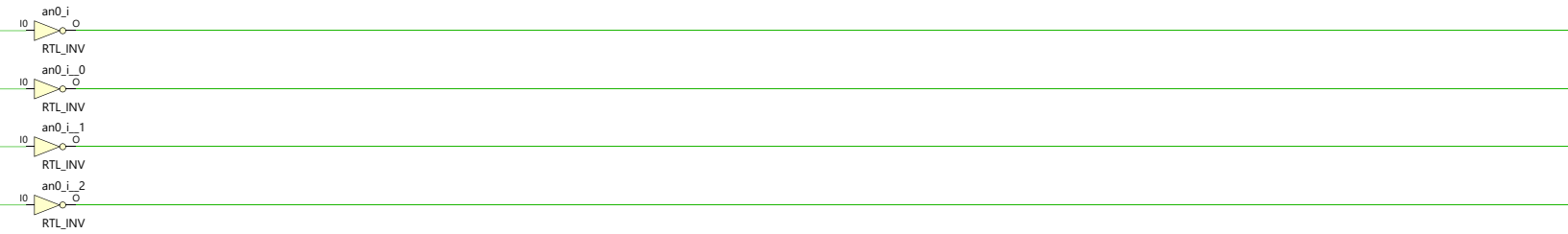
In conclusion, though this lab was incredibly difficult, it was very rewarding in the end. I learned a lot about creating my own designs from scratch as well as how to program the display of a monitor using the VGA. I had difficulty creating the road because it was my first time working with the VGA. If I could do this lab again, I would plan a lot more before attempting to build anything. There aren't any components that I would optimize, though my top level is incredibly large, I find it much easier to understand for a project of this scale because my logic can be clearly seen in one spot and isn't lost in the abstraction of modules.

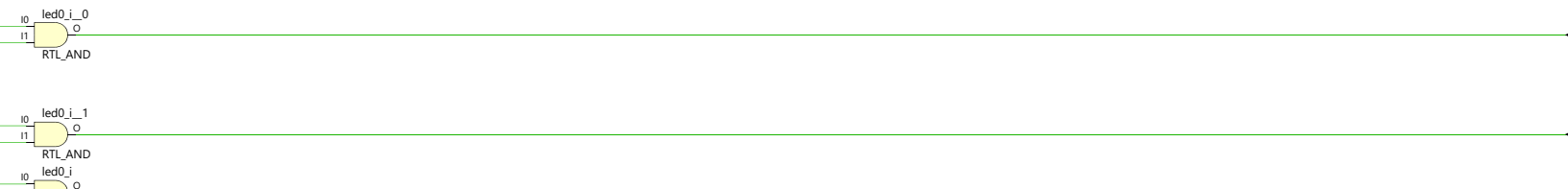


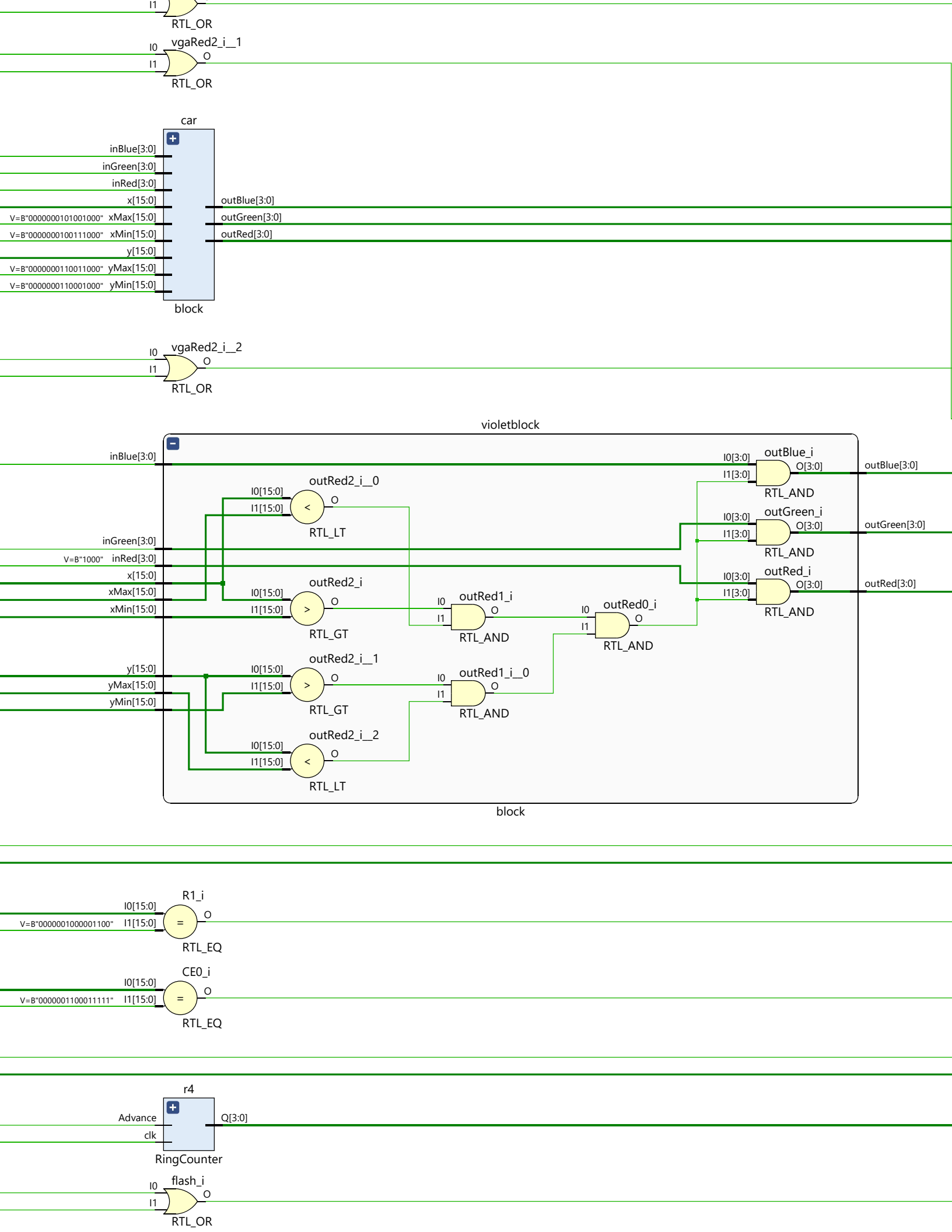

vgaGreen0_i__1
RTL_OR
vgaGreen0_i__2
RTL_OR
vgaRed0_i
RTL_OR
vgaRed0_i__0
RTL_OR
vgaRed0_i__1
RTL_OR
vgaRed0_i__2
RTL_OR
vc
CE
R
clk
i0
Count_4b
CE2_i
RTL_AND
i1
Count_4b
CE1_i
RTL_AND
i2
Count_4b
Vsync0_i
V=B"0000000111101001"
RTL_LT
Vsync0_i__0
V=B"0000000111101010"
RTL_GT
Vsync_i
RTL_OR
i3
Count_4b
CE0_i
RTL_AND
Q[15:0]
Vsync
Vcounter
hc
Hsync
Q[15:0]
Hcounter
an0_i
RTL_INV
an0_i__0
RTL_INV
an0_i__1
RTL_INV
an0_i__2
RTL_INV
led0_i__0

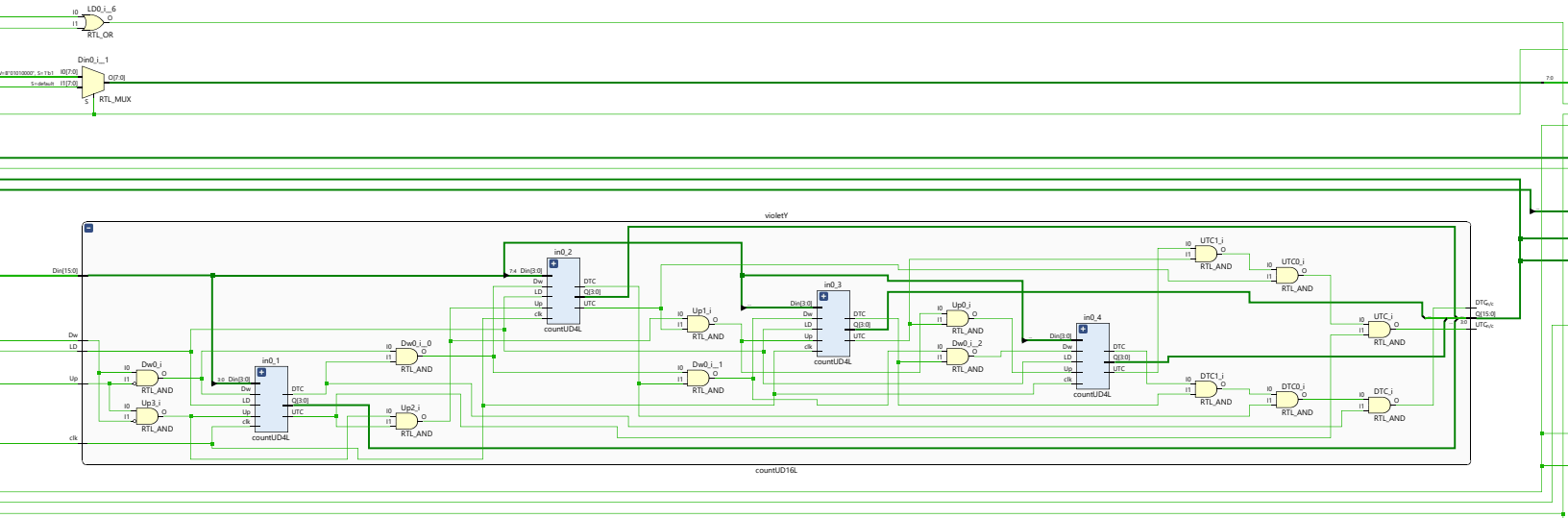

vgaGreen0_i__2
RTL_OR
vgaRed0_i
RTL_OR
vgaRed0_i__0
RTL_OR
vgaRed0_i__1
RTL_OR
vgaRed0_i__2
RTL_OR

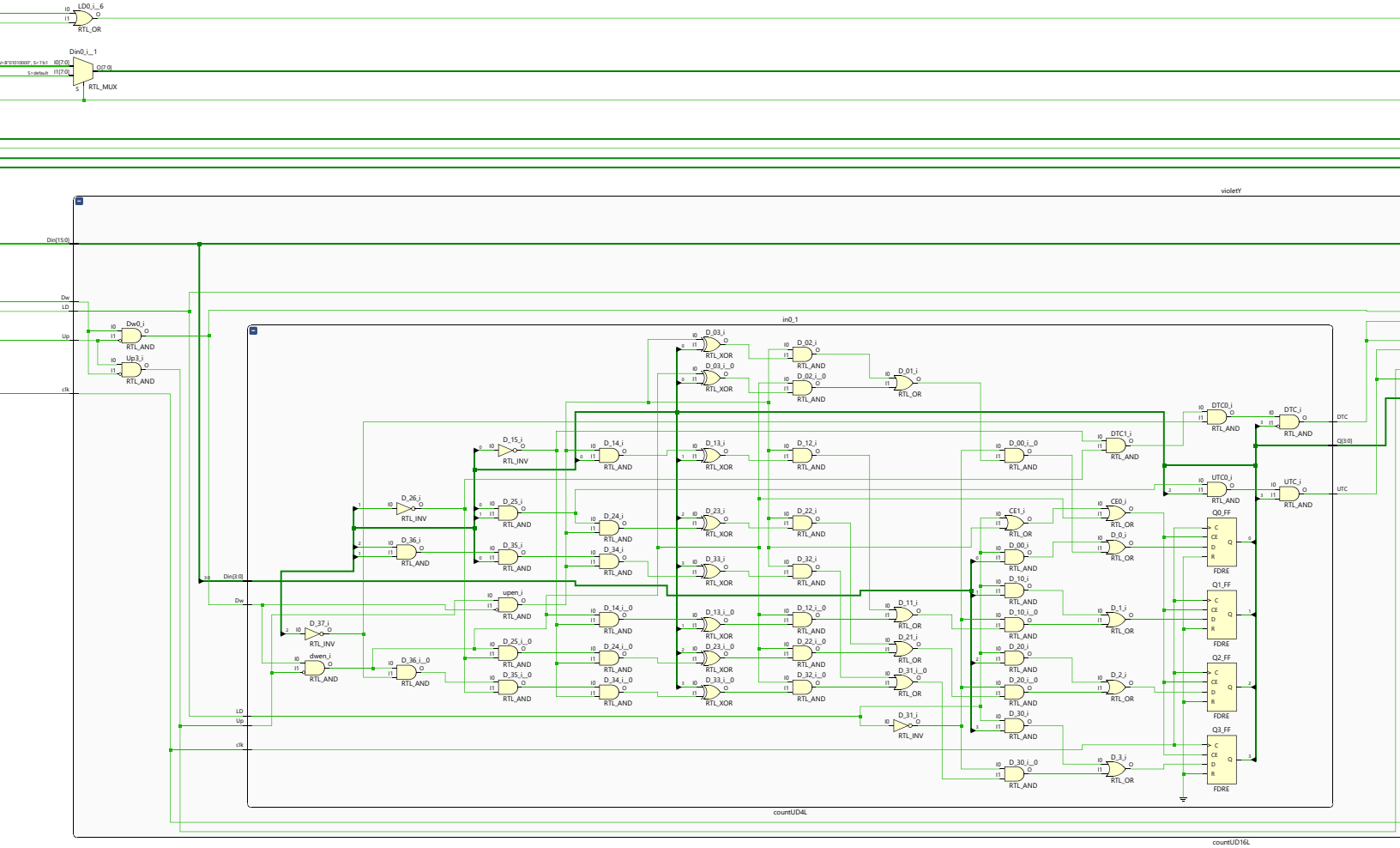

vc
Q[15:0]
Vsync
Vcounter
hc
Hsync0_i
RTL_LT
Hsync0_i__0
RTL_GT
Hsync_i
RTL_OR
Hsync
i3
Count_4b
Q[15:0]
i1
Count_4b
i2
Count_4b
i0
Count_4b
CE2_i
RTL_AND
CE1_i
RTL_AND
CE0_i
RTL_AND
Hcounter


an0_i
RTL_INV
an0_i__0
RTL_INV
an0_i__1
RTL_INV
an0_i__2
RTL_INV

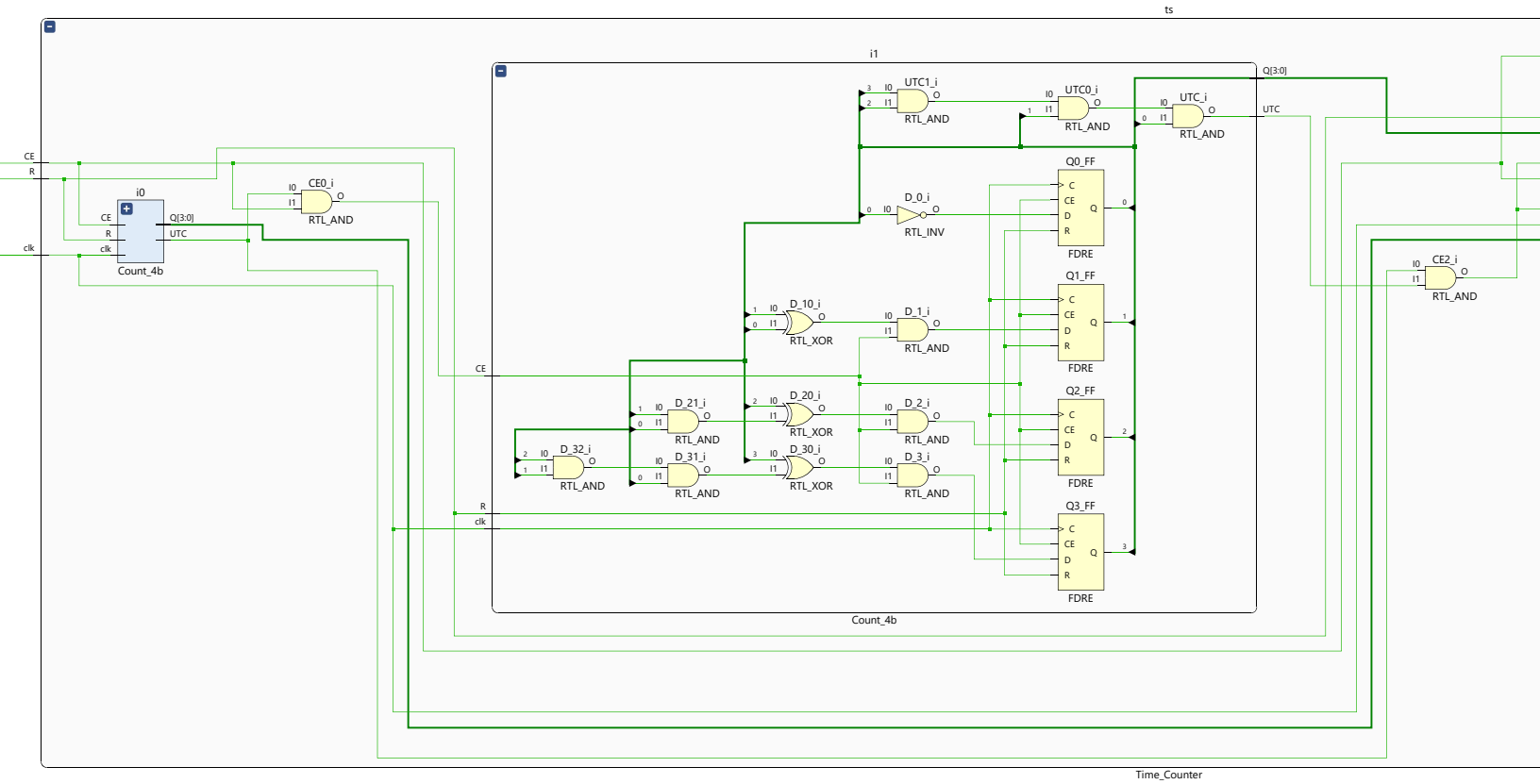

led0_i__0
RTL_AND
led0_i__1
RTL_AND
led0_i

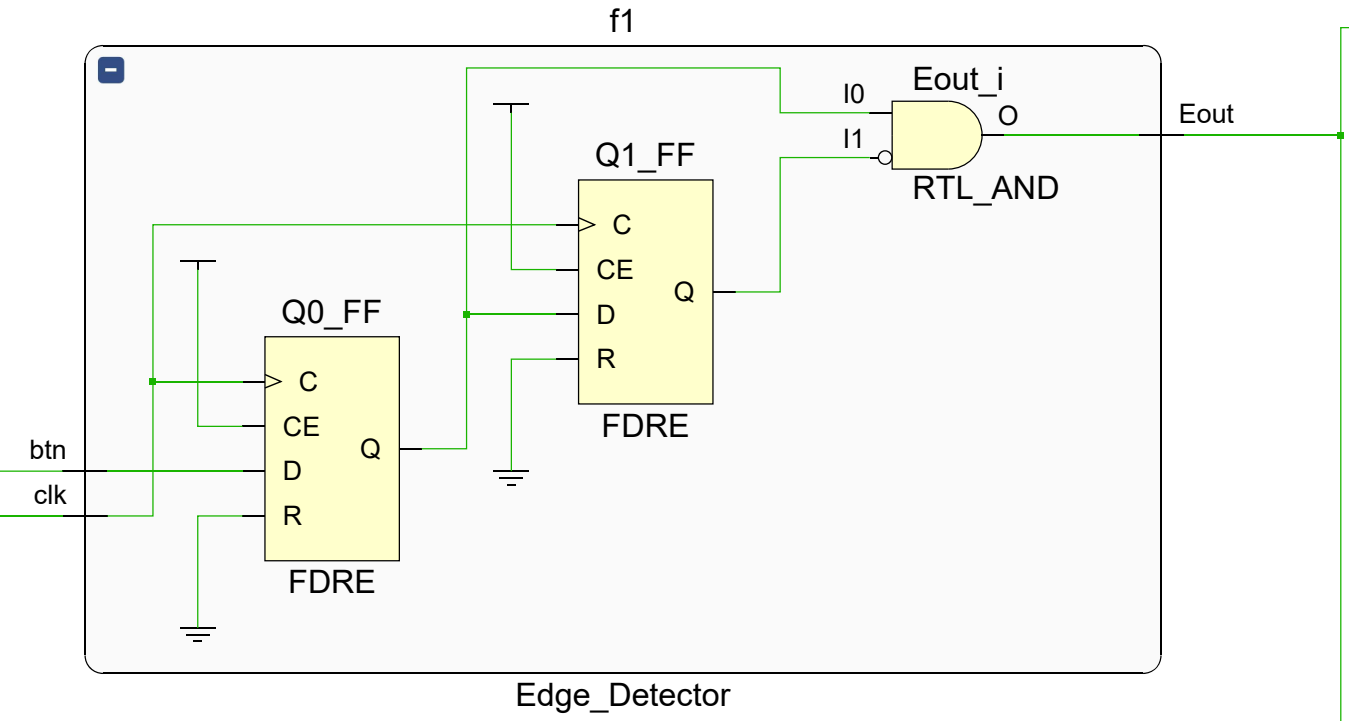

I1
RTL_OR
vgaRed2_i__1
I0
I1
O
RTL_OR
car
inBlue[3:0]
inGreen[3:0]
inRed[3:0]
x[15:0]
V=B"0000000101001000" xMax[15:0]
V=B"0000000100111000" xMin[15:0]
y[15:0]
V=B"0000000110011000" yMax[15:0]
V=B"0000000110001000" yMin[15:0]
outBlue[3:0]
outGreen[3:0]
outRed[3:0]
block
vgaRed2_i__2
I0
I1
O
RTL_OR
violetblock
inBlue[3:0]
inGreen[3:0]
V=B"1000" inRed[3:0]
x[15:0]
xMax[15:0]
xMin[15:0]
y[15:0]
yMax[15:0]
yMin[15:0]
outRed2_i__0
I0[15:0]
I1[15:0]
O
RTL_LT
outRed2_i
RTL_GT
outRed2_i__1
RTL_GT
outRed2_i__2
RTL_LT
outRed1_i
RTL_AND
outRed1_i__0
RTL_AND
outRed0_i
RTL_AND
outBlue_i
I0[3:0]
I1[3:0]
O[3:0]
RTL_AND
outGreen_i
RTL_AND
outRed_i
RTL_AND
outBlue[3:0]
outGreen[3:0]
outRed[3:0]
block
R1_i
V=B"0000001000001100"
RTL_EQ
CE0_i
V=B"0000001100011111"
RTL_EQ
r4
Advance
clk
Q[3:0]
RingCounter
flash_i
RTL_OR


LD0_i_6
RTL_OR
Din0_i_1
RTL_MUX
violetY
countUD16L
in0_1
in0_2
in0_3
in0_4
countUD4L
Dw0_i
Up3_i
Dw0_i_0
Up2_i
Up1_i
Dw0_i_1
Up0_i
Dw0_i_2
UTC1_i
UTC0_i
UTC_i
DTC1_i
DTC0_i
DTC_i
RTL_AND

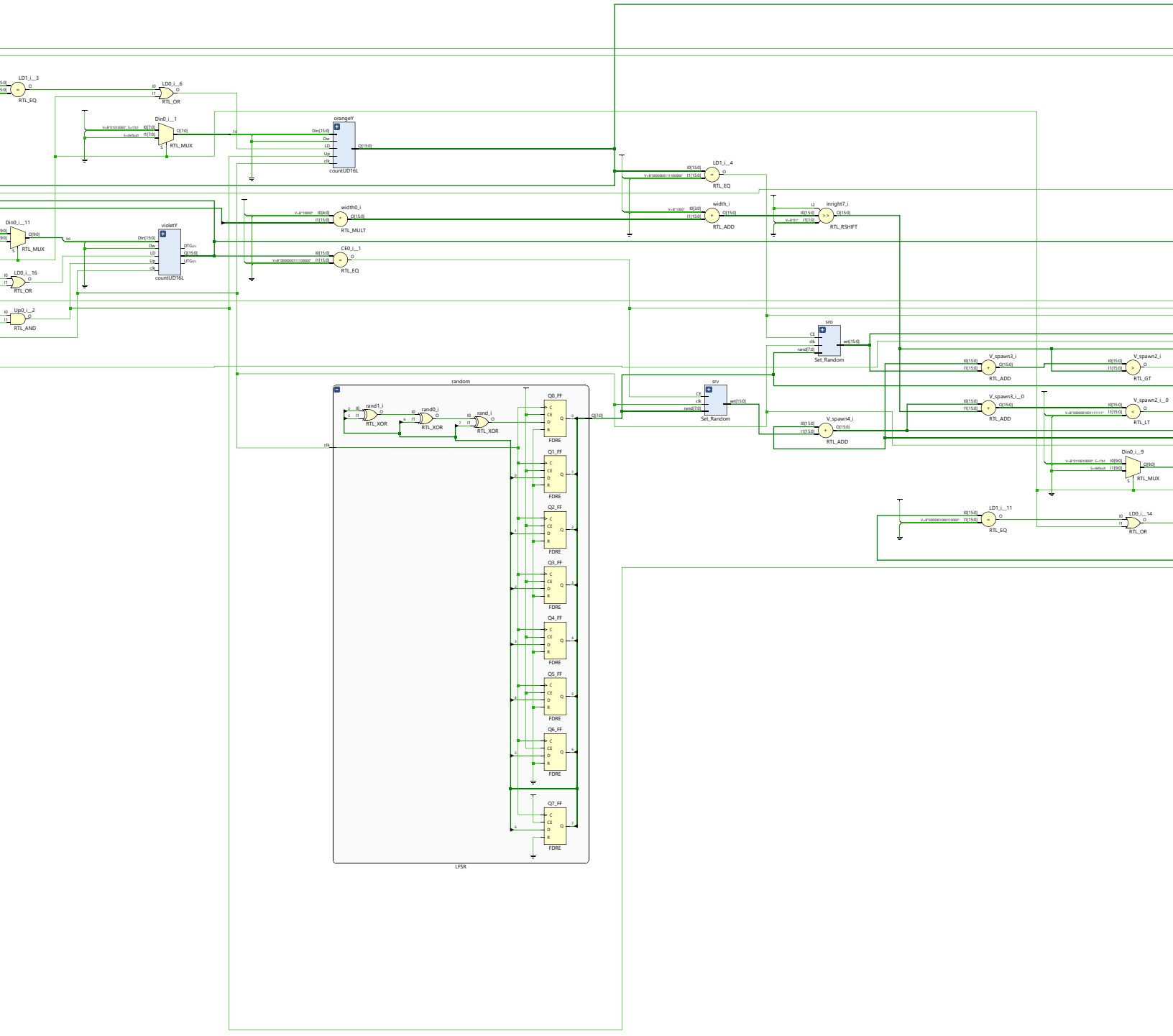

violetY
Din[15:0]
Dw
LD
Up
clk
Din[3:0]
in0_1
countUD4L
countUD16L
DTC
Q[3:0]
UTC
Q0_FF
Q1_FF
Q2_FF
Q3_FF
FDRE
RTL_AND
RTL_OR
RTL_XOR
RTL_INV
RTL_MUX

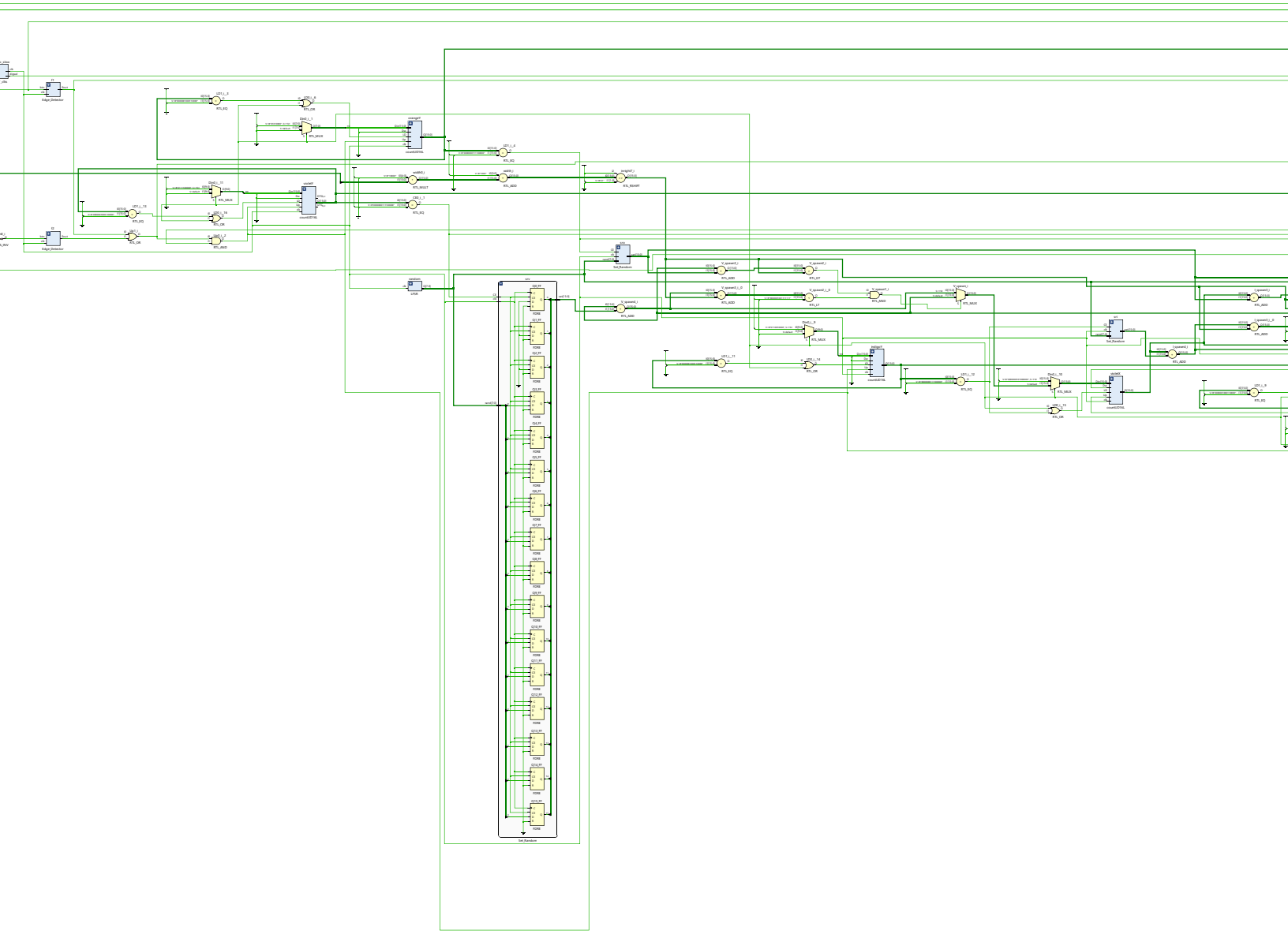

lose0_i
I0
I1
O
RTL_OR

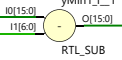

ts
i1
CE
R
clk
i0
Count_4b
CE0_i
RTL_AND
UTC1_i
UTC0_i
UTC_i
Q[3:0]
UTC
D_0_i
RTL_INV
D_10_i
RTL_XOR
D_1_i
D_21_i
D_20_i
D_2_i
D_32_i
D_31_i
D_30_i
D_3_i
Q0_FF
Q1_FF
Q2_FF
Q3_FF
FDRE
CE2_i
Count_4b
Time_Counter

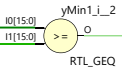

f1
Eout_i
I0
I1
O
Eout
RTL_AND
Q1_FF
C
CE
D
R
Q
FDRE
Q0_FF
C
CE
D
R
Q
FDRE
btn
clk
Edge_Detector

V=B"0000001

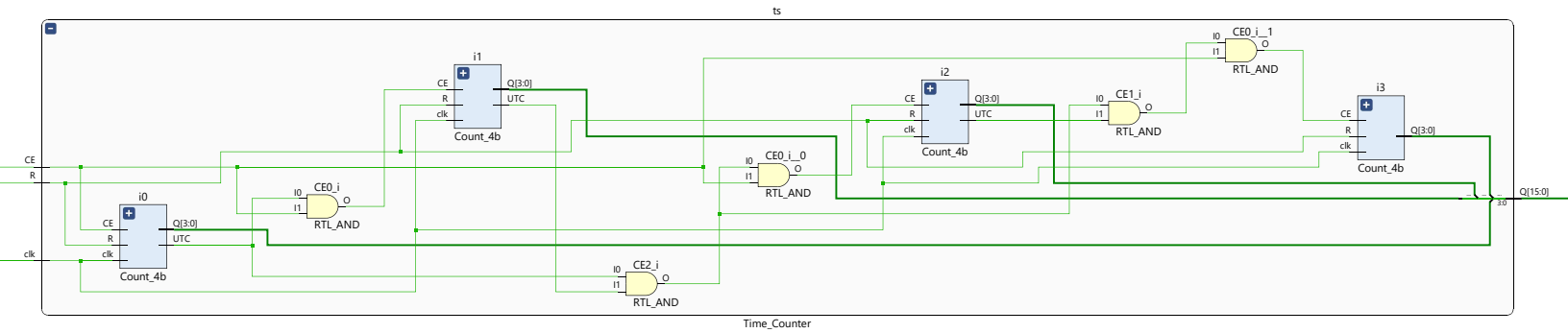

LD1_i__3
RTL_EQ
LD0_i__6
RTL_OR
Din0_i__1
RTL_MUX
orangeY
countUD16L
LD1_i__4
RTL_EQ
width_i
RTL_ADD
inright7_i
RTL_RSHIFT
width0_i
RTL_MULT
violetY
countUD16L
CE0_i__1
RTL_EQ
Din0_i__11
RTL_MUX
LD0_i__16
RTL_OR
Up0_i__2
RTL_AND
srd
Set_Random
V_spawn3_i
RTL_ADD
V_spawn2_i
RTL_GT
srv
Set_Random
V_spawn4_i
RTL_ADD
V_spawn3_i__0
RTL_ADD
V_spawn2_i__0
RTL_LT
Din0_i__9
RTL_MUX
LD1_i__11
RTL_EQ
LD0_i__14
RTL_OR
random
rand1_i
RTL_XOR
rand0_i
RTL_XOR
rand_i
RTL_XOR
Q0_FF
Q1_FF
Q2_FF
Q3_FF
Q4_FF
Q5_FF
Q6_FF
Q7_FF
FDRE
LFSR

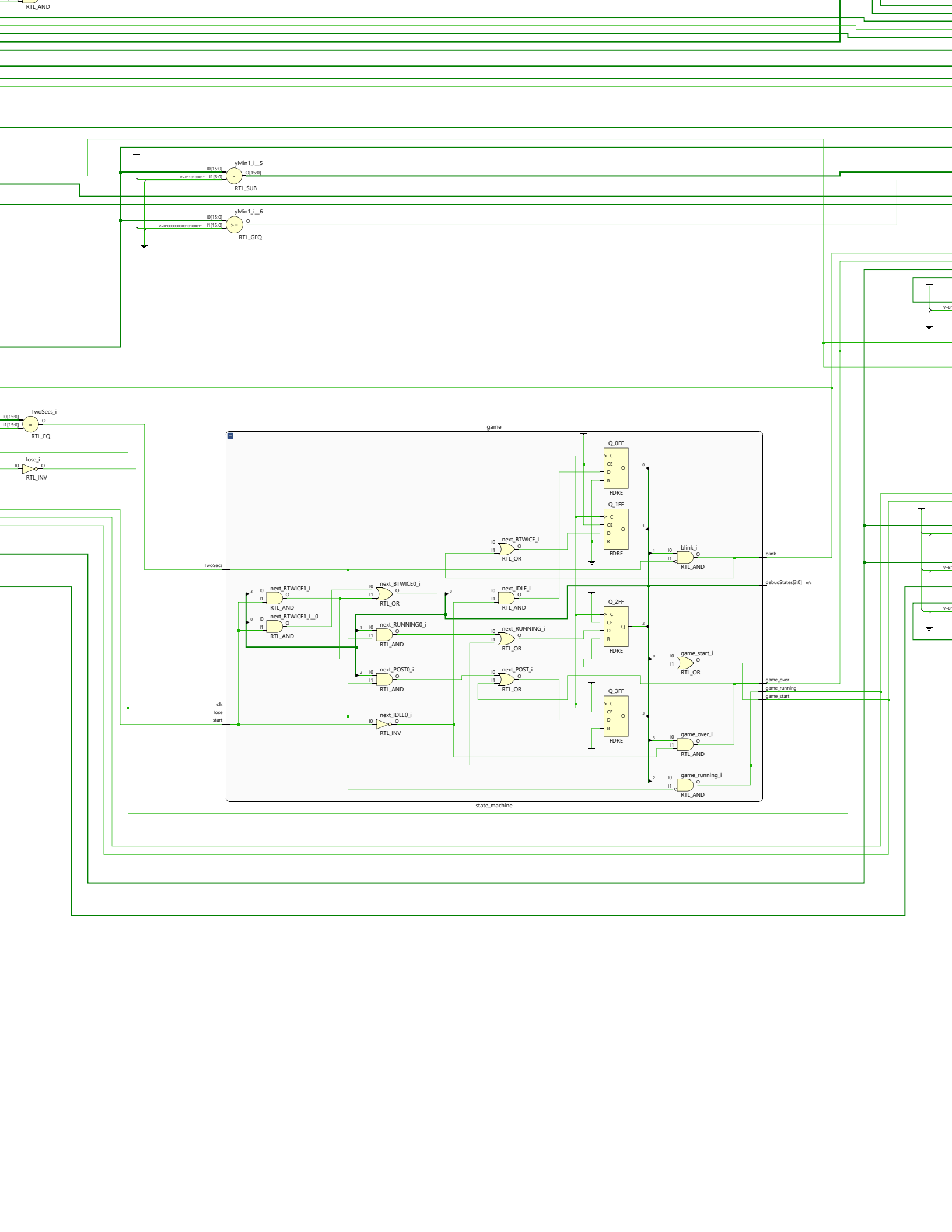


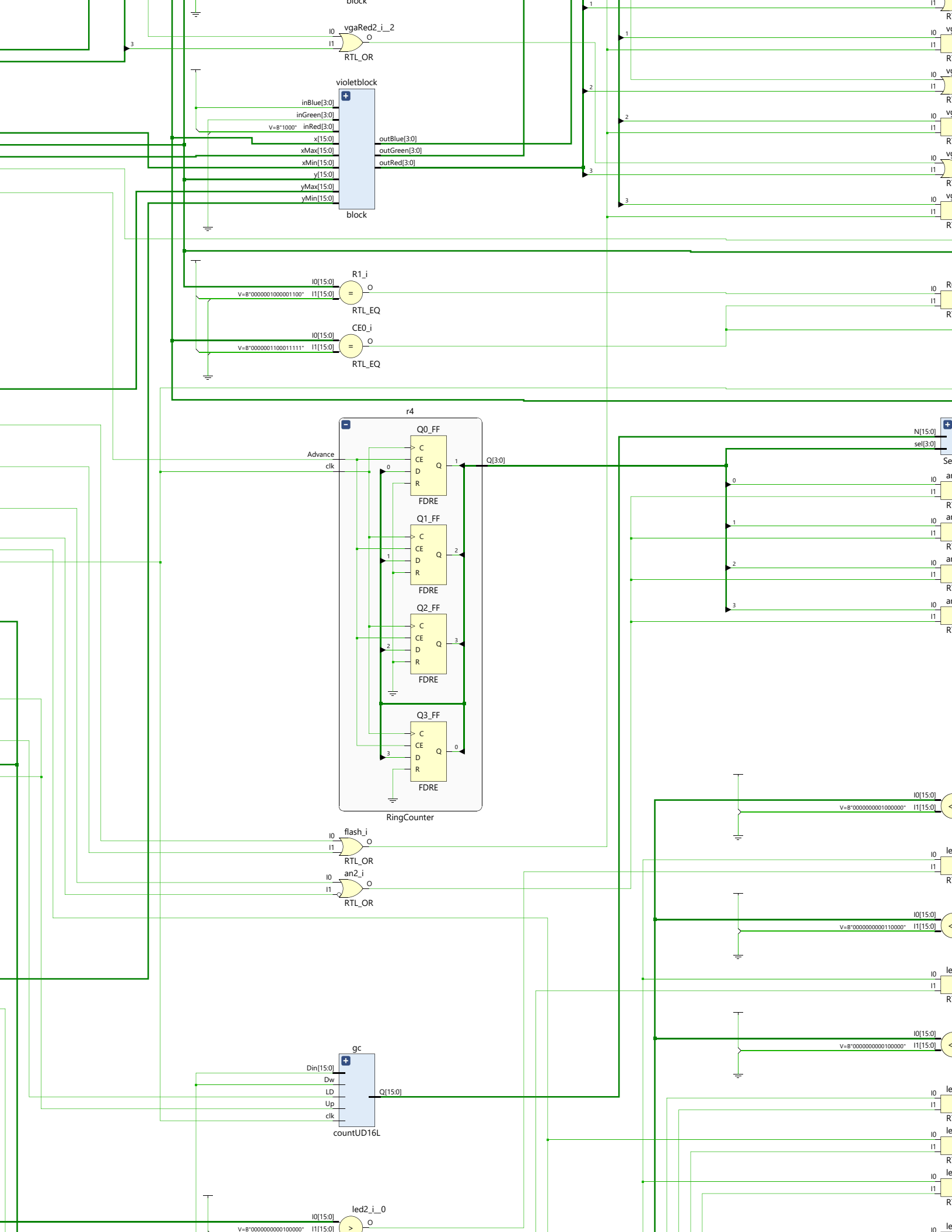

I0[15:0]
I1[6:0]
-
O[15:0]
RTL_SUB

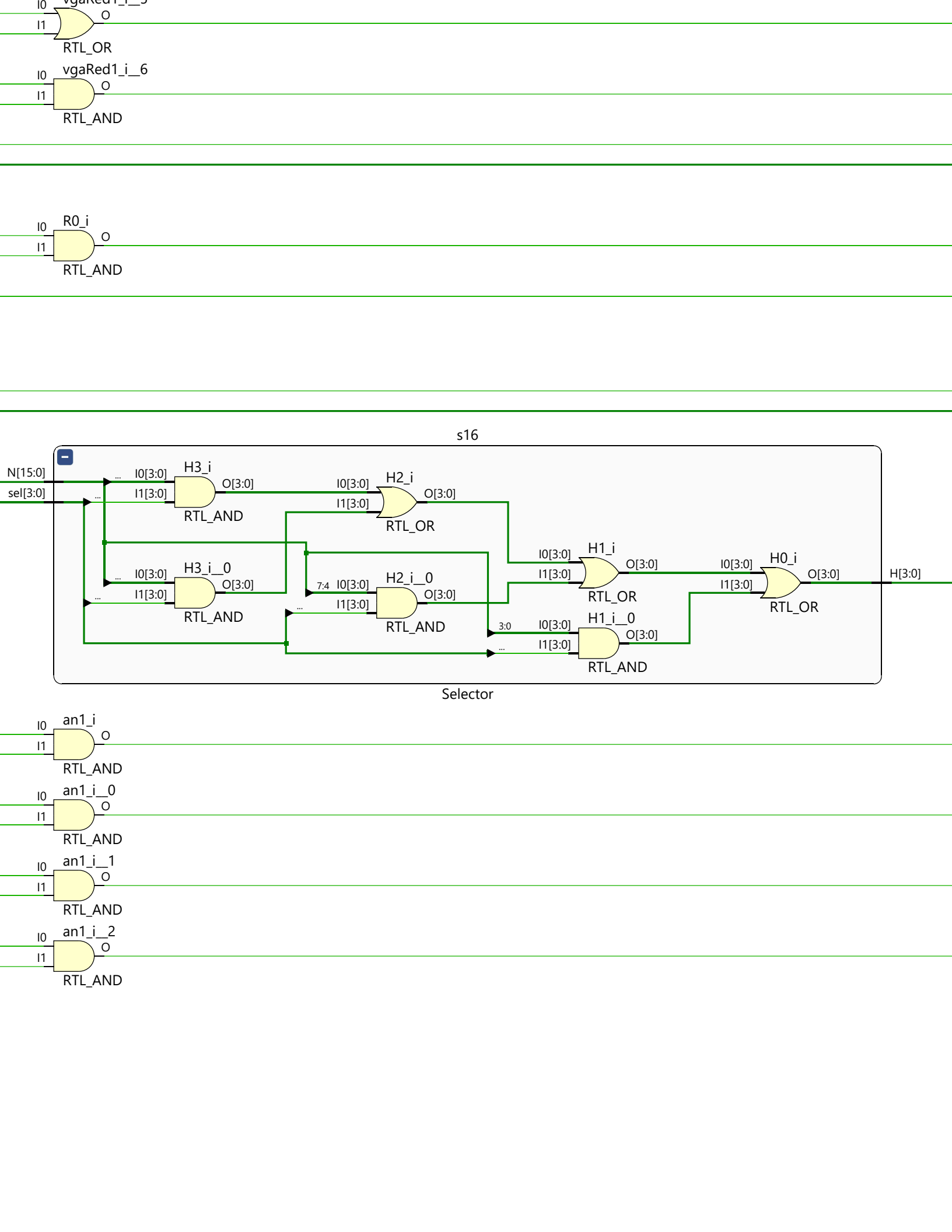

yMin1_i__2
I0[15:0]
I1[15:0]
>=
O
RTL_GEQ

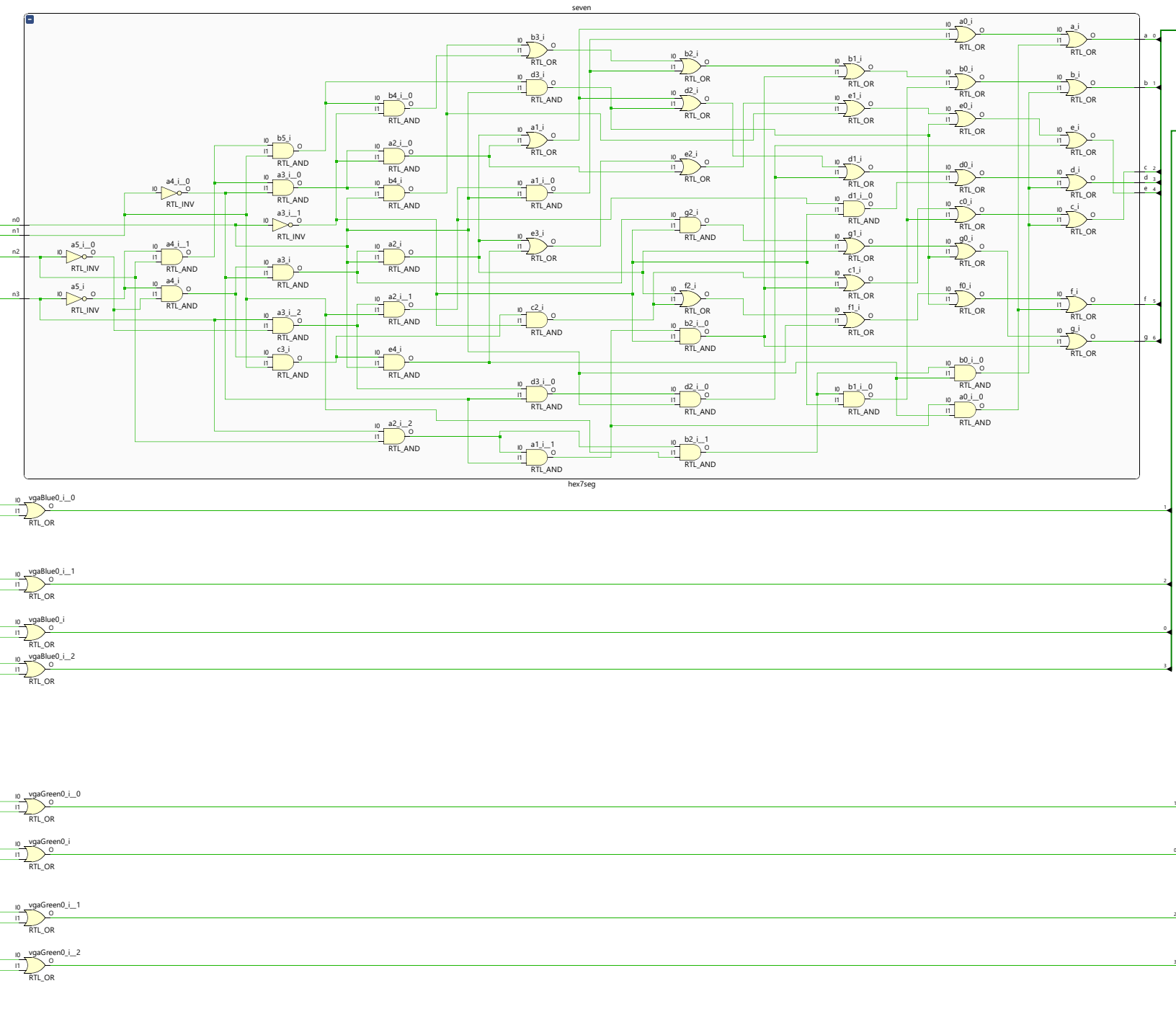

lose0_i
I0
I1
O
RTL_OR


ts
CE
R
clk
i0
Count_4b
Q[3:0]
UTC
CE0_i
RTL_AND
i1
CE2_i
CE0_i__0
i2
CE1_i
CE0_i__1
i3
Q[15:0]
Time_Counter


RTL_AND
yMin1_i__5
I0[15:0]
V="1010001"
I1[6:0]
O[15:0]
RTL_SUB
yMin1_i__6
I0[15:0]
V="0000000001010001"
I1[15:0]
O
RTL_GEQ
TwoSecs_i
I0[15:0]
I1[15:0]
RTL_EQ
lose_i
RTL_INV
game
Q_0FF
Q_1FF
Q_2FF
Q_3FF
FDRE
next_BTWICE_i
next_BTWICE0_i
next_BTWICE1_i
next_BTWICE1_i__0
next_IDLE_i
next_IDLE0_i
next_RUNNING_i
next_RUNNING0_i
next_POST_i
next_POST0_i
blink_i
game_start_i
game_over_i
game_running_i
RTL_AND
RTL_OR
RTL_INV
TwoSecs
clk
lose
start
blink
debugStates[3:0]
game_over
game_running
game_start
state_machine


block
vgaRed2_i__2
I0
I1
O
RTL_OR
violetblock
inBlue[3:0]
inGreen[3:0]
V=B"1000"
inRed[3:0]
x[15:0]
xMax[15:0]
xMin[15:0]
y[15:0]
yMax[15:0]
yMin[15:0]
outBlue[3:0]
outGreen[3:0]
outRed[3:0]
block
R1_i
I0[15:0]
V=B"0000001000001100"
I1[15:0]
=
RTL_EQ
CE0_i
V=B"0000001100011111"
r4
Advance
clk
Q0_FF
Q1_FF
Q2_FF
Q3_FF
C
CE
D
R
Q
FDRE
Q[3:0]
RingCounter
N[15:0]
sel[3:0]
flash_i
an2_i
V=B"0000000001000000"
V=B"0000000000110000"
V=B"0000000000100000"
gc
Din[15:0]
Dw
LD
Up
clk
Q[15:0]
countUD16L
led2_i__0
V=B"0000000000100000"


I0
I1
O
RTL_OR
vgaRed1_i__6
RTL_AND
R0_i
s16
N[15:0]
sel[3:0]
H3_i
I0[3:0]
I1[3:0]
O[3:0]
RTL_AND
H2_i
RTL_OR
H3_i__0
H2_i__0
7:4
H1_i
H1_i__0
3:0
H0_i
H[3:0]
Selector
an1_i
an1_i__0
an1_i__1
an1_i__2


seven
hex7seg
n0
n1
n2
n3
a
b
c
d
e
f
g
RTL_INV
RTL_AND
RTL_OR
vgaBlue0_i__0
vgaBlue0_i__1
vgaBlue0_i
vgaBlue0_i__2
vgaGreen0_i__0
vgaGreen0_i
vgaGreen0_i__1
vgaGreen0_i__2

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 02/25/2020 01:33:52 PM
// Design Name:
// Module Name: top_level
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module top_level(
// Inputs
   input btnC, //new game
   input btnD, //not used (only here for test sync)
   input btnU, //not used (only here for test sync)
   input btnL, //left
   input btnR, //right
   input [15:0] sw, //only using sw4, sw5, sw6
   input clkin,
// Outputs
   output [3:0] an,
   output dp, //not used (only here for test sync)
   output [6:0] seg,
   output [15:0] led,
   output Hsync,
   output [3:0] vgaBlue,
   output [3:0] vgaGreen,
   output [3:0] vgaRed,
   output Vsync
    );

    wire clk, digsel;
    wire [15:0] Hcount, Vcount;
    wire frame1, frame2; //high when one frame has passed
```

```
    wire [15:0] frame16, frame64, frame128, scoreFlash, ledframe; //counts every 16
frames (quarter sec), 64 frames (1 sec), 128 frames (2 secs), score flash, led shift
    wire [3:0] rings; //output of ring counter
    wire [3:0] sel; //output of selector
    wire [15:0] bits; //input of selector to show on display
    wire [15:0] sec_count; //holds result of second counter (time counter for seconds
    wire TwoSecs, lose, blink, game_start, game_running, game_over; //HAVE NOT
ASSIGNED VALUES TO THESE YET

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

    Edge_Detector f1 (.clk(clk), .btn(Vsync), .Eout(frame1)); //detects each frame
    Edge_Detector f2 (.clk(clk), .btn(~Vsync), .Eout(frame2)); //detects each frame
(for counting twice per frame)

    Hcounter hc (.clk(clk), .CE(1'b1), .R(Hcount == 16'd799), .Q(Hcount),
.Hsync(Hsync));
    Vcounter vc (.clk(clk), .CE(Hcount == 16'd799), .R(Vcount == 16'd524 & Hcount ==
16'd799), .Q(Vcount), .Vsync(Vsync)); //goes to bottom right corner

    //================================================STATE
MACHINE=============================================
    wire [3:0] debugStates;
    state_machine game (.clk(clk), .start(btnC), .TwoSecs(TwoSecs), .lose(lose),
.blink(blink), .game_start(game_start), .game_running(game_running),
.game_over(game_over)
                        ,.debugStates(debugStates));

    // Debugging
    //assign led[3:0] = debugStates;
    //assign led[15:4] = 12'd0;

    countUD16L f64 (.clk(clk), .Up(frame1), .Dw(1'b0), .LD ((frame64[15:0] ==
16'd64) | game_start), .Din(16'b0), .Q(frame64[15:0])); //64 frame counter
    Time_Counter ts (.clk(clk), .CE( (frame64[15:0] == 16'd64) & blink),
.R(game_start), .Q(sec_count[15:0])); //second counter

    assign TwoSecs = (sec_count[15:0] == 16'd2); //determines if 2 secs has been pass

    //================================================BLOCK
STUFF==============================================
    wire [15:0] red_X, red_Y, orange_X, orange_Y, yellow_X, yellow_Y, green_X,
green_Y, blue_X, blue_Y, indigo_X, indigo_Y, violet_X, violet_Y; //coordinate
    wire [3:0] red1, red2, red3, red4, red5, red6, red7, car_red;
    wire [3:0] green1, green2, green3, green4, green5, green6, green7, car_green;
    wire [3:0] blue1, blue2, blue3, blue4, blue5, blue6, blue7, car_blue;
```

```
    wire [7:0] rand;
    wire [15:0] R_offset, O_offset, Y_offset, G_offset, B_offset, I_offset, V_offset;
    wire [15:0] i, width;
    wire [15:0] R_spawn, O_spawn, Y_spawn, G_spawn, B_spawn, I_spawn, V_spawn;
    //wire block_respawn;
    wire [15:0] carX;
    wire flash; //controls car blink
    assign i[2] = sw[6];
    assign i[1] = sw[5];
    assign i[0] = sw[4];
    assign width = 16'd8 + 16'd16 * i;
    wire in_left, in_right;
    assign inleft = (red_X - width/16'd2 > 16'd0) & (orange_X - width/16'd2 > 16'd0)
& (yellow_X - width/16'd2 > 16'd0) & (green_X - width/16'd2 > 16'd0) & (blue_X -
width/16'd2 > 16'd0) & (indigo_X - width/16'd2 > 16'd0) & (violet_X - width/16'd2 >
16'd0);
    assign inright = (red_X + width/16'd2 < 16'd639) & (orange_X + width/16'd2 <
16'd639) & (yellow_X + width/16'd2 < 16'd639) & (green_X + width/16'd2 < 16'd639) &
(blue_X + width/16'd2 < 16'd639) & (indigo_X + width/16'd2 < 16'd639) & (violet_X +
width/16'd2 < 16'd639);

    LFSR random (.clk(clk), .Q(rand[7:0]));
    //assign block_respawn = (red_Y == 16'd559) | (orange_Y == 16'd559) | (yellow_Y
== 16'd559) | (green_Y == 16'd559) | (blue_Y == 16'd559) | (indigo_Y == 16'd559) |
(violet_Y == 16'd559);
    //Set_Random sr(.clk(clk), .CE(block_respawn), .rand(rand[7:0]),
.set(offset[15:0]));

    //==========RED==========
    //Ycounter yc (.clk(clk), .CE(frame1 | frame2), .R(Ycount == 12'd559),
.Q(Ycount)); //Y coordinate counter
    countUD16L redY (.clk(clk), .Up( (frame1 | frame2) & game_running ), .Dw(1'b0),
.LD(red_Y == 16'd560 | game_start), .Din(game_start ? 16'd0 : 16'd0), .Q(red_Y),
.UTC(), .DTC());//Y coordinate counter
    Set_Random srr(.clk(clk), .CE(red_Y == 16'd480), .rand(rand[7:0]),
.set(R_offset[15:0])); //gets offset amount
    assign R_spawn = ( (orange_X + R_offset > width/16'd2) && (orange_X + R_offset +
width/16'd2 < 16'd639)) ? orange_X  + R_offset : orange_X; //determines new spawn
point (doesn't let go past screen)
    countUD16L redX (.clk(clk), .Up( btnR & ~btnL & inright & (frame1 | frame2) &
game_running ), .Dw( btnL & ~btnR & inleft & (frame1 | frame2)  & game_running ),
.LD(game_start | violet_Y == 16'd480),
                    .Din(game_start ? 16'd320 : R_spawn), .Q(red_X), .UTC(),
.DTC());//X coordinate counter
    block redblock (.inRed(4'hf), .inGreen(4'h0), .inBlue(4'h0), .x(Hcount),
.xMin(red_X - width/16'd2), .xMax(red_X + width/16'd2),
```

```
                .y(Vcount), .yMin( (red_Y - 16'd81) & {16{(red_Y >= 16'd81)}} ),
.yMax(red_Y), .outRed(red1), .outGreen(green1), .outBlue(blue1) );

    //==========ORANGE==========
    countUD16L orangeY (.clk(clk), .Up( (frame1 | frame2) & game_running ),
.Dw(1'b0), .LD(orange_Y == 16'd560 | game_start), .Din(game_start ? 16'd80 : 16'd0),
.Q(orange_Y), .UTC(), .DTC());//Y coordinate counter
    Set_Random sro(.clk(clk), .CE(orange_Y == 16'd480), .rand(rand[7:0]),
.set(O_offset[15:0])); //gets offset amount
    assign O_spawn = ( (yellow_X + O_offset > width/16'd2) && (yellow_X + O_offset +
width/16'd2 < 16'd639)) ? yellow_X  + O_offset : yellow_X; //determines new spawn
point (doesn't let go past screen)
    countUD16L orangeX (.clk(clk), .Up( btnR & ~btnL & inright & (frame1 | frame2) &
game_running ), .Dw( btnL & ~btnR & inleft & (frame1 | frame2)  & game_running ),
.LD(game_start | red_Y == 16'd480),
                        .Din(game_start ? 16'd320 : O_spawn), .Q(orange_X), .UTC(),
.DTC());//X coordinate counter
    block orangeblock (.inRed(4'hf), .inGreen(4'h7), .inBlue(4'h0), .x(Hcount),
.xMin(orange_X - width/16'd2), .xMax(orange_X + width/16'd2),
                .y(Vcount), .yMin( (orange_Y - 16'd81) & {16{(orange_Y >= 16'd81)}}
), .yMax(orange_Y), .outRed(red2), .outGreen(green2), .outBlue(blue2) );

    //==========YELLOW==========
    countUD16L yellowY (.clk(clk), .Up( (frame1 | frame2) & game_running ),
.Dw(1'b0), .LD(yellow_Y == 16'd560 | game_start), .Din(game_start ? 16'd160 :
16'd0), .Q(yellow_Y), .UTC(), .DTC());//Y coordinate counter
    Set_Random sry(.clk(clk), .CE(yellow_Y == 16'd480), .rand(rand[7:0]),
.set(Y_offset[15:0]));  //gets offset amount
    assign Y_spawn = ( (green_X + O_offset > width/16'd2) && (green_X + Y_offset +
width/16'd2 < 16'd639)) ? green_X  + Y_offset : green_X; //determines new spawn
point (doesn't let go past screen)
    countUD16L yellowX (.clk(clk), .Up( btnR & ~btnL & inright & (frame1 | frame2) &
game_running ), .Dw( btnL & ~btnR & inleft & (frame1 | frame2)  & game_running ),
.LD(game_start | orange_Y == 16'd480),
                        .Din(game_start ? 16'd320 : Y_spawn), .Q(yellow_X), .UTC(),
.DTC());//X coordinate counter
    block yellowblock (.inRed(4'hf), .inGreen(4'hf), .inBlue(4'h0), .x(Hcount),
.xMin(yellow_X - width/16'd2), .xMax(yellow_X + width/16'd2),
                .y(Vcount), .yMin( (yellow_Y - 16'd81) & {16{(yellow_Y >= 16'd81)}}
), .yMax(yellow_Y), .outRed(red3), .outGreen(green3), .outBlue(blue3) );

    //==========GREEN==========
    countUD16L greenY (.clk(clk), .Up( (frame1 | frame2) & game_running ),
.Dw(1'b0), .LD(green_Y == 16'd560 | game_start), .Din(game_start ? 16'd240 : 16'd0),
.Q(green_Y), .UTC(), .DTC());//Y coordinate counter
    Set_Random srg(.clk(clk), .CE(green_Y == 16'd480), .rand(rand[7:0]),
```

```
.set(G_offset[15:0])); //gets offset amount
    assign G_spawn = ( (blue_X + O_offset > width/16'd2) && (blue_X + G_offset +
width/16'd2 < 16'd639)) ? blue_X  + G_offset : blue_X; //determines new spawn point
(doesn't let go past screen)
    countUD16L greenX (.clk(clk), .Up( btnR & ~btnL & inright & (frame1 | frame2) &
game_running ), .Dw( btnL & ~btnR & inleft & (frame1 | frame2)  & game_running ),
.LD(game_start | yellow_Y == 16'd480),
                        .Din(game_start ? 16'd320 : G_spawn), .Q(green_X), .UTC(),
.DTC());//X coordinate counter
    block greenblock (.inRed(4'h0), .inGreen(4'hf), .inBlue(4'h0), .x(Hcount),
.xMin(green_X - width/16'd2), .xMax(green_X + width/16'd2),
                .y(Vcount), .yMin( (green_Y - 16'd81) & {16{(green_Y >= 16'd81)}} ),
.yMax(green_Y), .outRed(red4), .outGreen(green4), .outBlue(blue4) );

    //==========BLUE==========
    countUD16L blueY (.clk(clk), .Up( (frame1 | frame2) & game_running ), .Dw(1'b0),
.LD(blue_Y == 16'd560 | game_start), .Din(game_start ? 16'd320 : 16'd0), .Q(blue_Y),
.UTC(), .DTC());//Y coordinate counter
    Set_Random srb(.clk(clk), .CE(blue_Y == 16'd480), .rand(rand[7:0]),
.set(B_offset[15:0]));  //gets offset amount
    assign B_spawn = ( (indigo_X + O_offset > width/16'd2) && (indigo_X + B_offset +
width/16'd2 < 16'd639)) ? indigo_X  + B_offset : indigo_X; //determines new spawn
point (doesn't let go past screen)
    countUD16L blueX (.clk(clk), .Up( btnR & ~btnL & inright & (frame1 | frame2) &
game_running ), .Dw( btnL & ~btnR & inleft & (frame1 | frame2)  & game_running ),
.LD(game_start | green_Y == 16'd480),
                        .Din(game_start ? 16'd320 : B_spawn), .Q(blue_X), .UTC(),
.DTC());//X coordinate counter
    block blueblock (.inRed(4'h0), .inGreen(4'h0), .inBlue(4'hf), .x(Hcount),
.xMin(blue_X - width/16'd2), .xMax(blue_X + width/16'd2),
                .y(Vcount), .yMin( (blue_Y - 16'd81) & {16{(blue_Y >= 16'd81)}} ),
.yMax(blue_Y), .outRed(red5), .outGreen(green5), .outBlue(blue5) );

    //==========INDIGO==========
    countUD16L indigoY (.clk(clk), .Up( (frame1 | frame2) & game_running ),
.Dw(1'b0), .LD(indigo_Y == 16'd560 | game_start), .Din(game_start ? 16'd400 :
16'd0), .Q(indigo_Y), .UTC(), .DTC());//Y coordinate counter
    Set_Random sri(.clk(clk), .CE(indigo_Y == 16'd480), .rand(rand[7:0]),
.set(I_offset[15:0]));  //gets offset amount
    assign I_spawn = ( (violet_X + O_offset > width/16'd2) && (violet_X + I_offset +
width/16'd2 < 16'd639)) ? violet_X  + I_offset : violet_X; //determines new spawn
point (doesn't let go past screen)
    countUD16L indigoX (.clk(clk), .Up( btnR & ~btnL & inright & (frame1 | frame2) &
game_running ), .Dw( btnL & ~btnR & inleft & (frame1 | frame2)  & game_running ),
.LD(game_start | blue_Y == 16'd480),
                        .Din(game_start ? 16'd320 : I_spawn), .Q(indigo_X), .UTC(),
```

```
.DTC());//X coordinate counter
    block indigoblock (.inRed(4'h2), .inGreen(4'h2), .inBlue(4'h5), .x(Hcount),
.xMin(indigo_X - width/16'd2), .xMax(indigo_X + width/16'd2),
                .y(Vcount), .yMin( (indigo_Y - 16'd81) & {16{(indigo_Y >= 16'd81)}}
), .yMax(indigo_Y), .outRed(red6), .outGreen(green6), .outBlue(blue6) );

    //==========VIOLET==========
    countUD16L violetY (.clk(clk), .Up( (frame1 | frame2) & game_running ),
.Dw(1'b0), .LD(violet_Y == 16'd560 | game_start), .Din(game_start ? 16'd480 :
16'd0), .Q(violet_Y), .UTC(), .DTC());//Y coordinate counter
    Set_Random srv(.clk(clk), .CE(violet_Y == 16'd480), .rand(rand[7:0]),
.set(V_offset[15:0]));  //gets offset amount
    assign V_spawn = ( (red_X + O_offset > width/16'd2) && (red_X + V_offset +
width/16'd2 < 16'd639)) ? red_X  + V_offset : red_X; //determines new spawn point
(doesn't let go past screen)
    countUD16L violetX (.clk(clk), .Up( btnR & ~btnL & inright & (frame1 | frame2)
& game_running ), .Dw( btnL & ~btnR & inleft & (frame1 | frame2)  & game_running ),
.LD(game_start | indigo_Y == 16'd480),
                        .Din(game_start ? 16'd320 : V_spawn), .Q(violet_X), .UTC(),
.DTC());//X coordinate counter
    block violetblock (.inRed(4'h8), .inGreen(4'h0), .inBlue(4'hf), .x(Hcount),
.xMin(violet_X - width/16'd2), .xMax(violet_X + width/16'd2),
                .y(Vcount), .yMin( (violet_Y - 16'd81) & {16{(violet_Y >= 16'd81)}}
), .yMax(violet_Y), .outRed(red7), .outGreen(green7), .outBlue(blue7) );



    //==========CAR==========
    block car (.inRed(4'hf), .inGreen(4'hf), .inBlue(4'hf), .x(Hcount),
.xMin(16'd312), .xMax(16'd328),
                .y(Vcount), .yMin(16'd392), .yMax(16'd408), .outRed(car_red),
.outGreen(car_green), .outBlue(car_blue) );

    assign flash = ( (blink | game_over) & (frame64[15:0] < 16'd32) ) | (~blink &
~game_over); //controls car blink (blink every second  (on/off))

    assign vgaRed[3] = red1[3] | red2[3] | red3[3] | red4[3] | red5[3] | red6[3] |
red7[3] | (car_red[3] & flash);
    assign vgaRed[2] = red1[2] | red2[2] | red3[2] | red4[2] | red5[2] | red6[2] |
red7[2] | (car_red[2] & flash);
    assign vgaRed[1] = red1[1] | red2[1] | red3[1] | red4[1] | red5[1] | red6[1] |
red7[1] | (car_red[1] & flash);
    assign vgaRed[0] = red1[0] | red2[0] | red3[0] | red4[0] | red5[0] | red6[0] |
red7[0] | (car_red[0] & flash);

    assign vgaGreen[3] = green1[3] | green2[3] | green3[3] | green4[3] | green5[3] |
green6[3] | green7[3] | (car_green[3] & flash);
```

```
    assign vgaGreen[2] = green1[2] | green2[2] | green3[2] | green4[2] | green5[2] |
green6[2] | green7[2] | (car_green[2] & flash);
    assign vgaGreen[1] = green1[1] | green2[1] | green3[1] | green4[1] | green5[1] |
green6[1] | green7[1] | (car_green[1] & flash);
    assign vgaGreen[0] = green1[0] | green2[0] | green3[0] | green4[0] | green5[0] |
green6[0] | green7[0] | (car_green[0] & flash);

    assign vgaBlue[3] = blue1[3] | blue2[3] | blue3[3] | blue4[3] | blue5[3] |
blue6[3] | blue7[3] | (car_blue[3] & flash);
    assign vgaBlue[2] = blue1[2] | blue2[2] | blue3[2] | blue4[2] | blue5[2] |
blue6[2] | blue7[2] | (car_blue[2] & flash);
    assign vgaBlue[1] = blue1[1] | blue2[1] | blue3[1] | blue4[1] | blue5[1] |
blue6[1] | blue7[1] | (car_blue[1] & flash);
    assign vgaBlue[0] = blue1[0] | blue2[0] | blue3[0] | blue4[0] | blue5[0] |
blue6[0] | blue7[0] | (car_blue[0] & flash);

    //==========OUT OF BOUNDS==========(checks if inside left, right, top, and bot
bound)
    wire [6:0] inside; //if inside road segments

    assign inside[0] = (red_X < 16'd328 + width/16'd2)      & (red_X + width/16'd2 >
16'd312)        &    (red_Y <    16'd488)    & (red_Y > 16'd392);    //red
    assign inside[1] = (orange_X < 16'd328 + width/16'd2)   & (orange_X +
width/16'd2 > 16'd312)     &    (orange_Y < 16'd488) & (orange_Y > 16'd392); //orange
    assign inside[2] = (yellow_X < 16'd328 + width/16'd2)   & (yellow_X +
width/16'd2 > 16'd312)     &    (yellow_Y < 16'd488) & (yellow_Y > 16'd392); //yellow
    assign inside[3] = (green_X < 16'd328 + width/16'd2)    & (green_X + width/16'd2
> 16'd312)      &    (green_Y <  16'd488)  & (green_Y > 16'd392);  //green
    assign inside[4] = (blue_X < 16'd328 + width/16'd2)     & (blue_X + width/16'd2
> 16'd312)       &    (blue_Y <   16'd488)   & (blue_Y > 16'd392);   //blue
    assign inside[5] = (indigo_X < 16'd328 + width/16'd2)   & (indigo_X +
width/16'd2 > 16'd312)     &    (indigo_Y < 16'd488) & (indigo_Y > 16'd392); //indigo
    assign inside[6] = (violet_X < 16'd328 + width/16'd2)   & (violet_X +
width/16'd2 > 16'd312)     &    (violet_Y < 16'd488) & (violet_Y > 16'd392); //violet

    assign lose =  ~( inside[0] | inside[1] | inside[2] | inside[3] | inside[4] |
inside[5] | inside[6] ); //not (inside at least one block)

    //================================================BOARD DISPLAY
STUFF================================================

    //==========SCORE==========
    countUD16L f16 (.clk(clk), .Up(frame1 & game_running), .Dw(1'b0), .LD
((frame16[15:0] == 16'd16) | game_start), .Din(16'b0), .Q(frame16[15:0])); //16
frame counter
    //Time_Counter tc (.clk(clk), .CE(frame16[15:0] == 16'd16), .R(start_game),
```

```
.Q(bits[15:0])); //game counter (counts every quarter second)
    countUD16L gc (.clk(clk), .Up(frame16[15:0] == 16'd16), .Dw(1'b0),
.LD(game_start), .Din(16'b0), .Q(bits[15:0]) );

    RingCounter r4 (.clk(clk), .Advance(digsel), .Q(rings[3:0]));
    Selector s16 (.sel(rings[3:0]) , .N(bits[15:0]), .H(sel[3:0]));
    hex7seg seven (.n3(sel[3]), .n2(sel[2]), .n1(sel[1]), .n0(sel[0]), .a(seg[0]),
.b(seg[1]), .c(seg[2]), .d(seg[3]),.e(seg[4]), .f(seg[5]), .g(seg[6]));

    countUD16L sf (.clk(clk), .Up(frame1 & game_over), .Dw(1'b0), .LD
(scoreFlash[15:0] == 16'd64), .Din(16'b0), .Q(scoreFlash[15:0])); //64 frame counter
for making score flash
    assign an[3] = ~(rings[3] & ( (game_over & (scoreFlash[15:0] < 16'd32)) |
~game_over) );
    assign an[2] = ~(rings[2] & ( (game_over & (scoreFlash[15:0] < 16'd32)) |
~game_over) );
    assign an[1] = ~(rings[1] & ( (game_over & (scoreFlash[15:0] < 16'd32)) |
~game_over) );
    assign an[0] = ~(rings[0] & ( (game_over & (scoreFlash[15:0] < 16'd32)) |
~game_over) );

    //==========LEDS==========
    countUD16L f128 (.clk(clk), .Up(frame1 & blink), .Dw(1'b0), .LD ((frame128[15:0]
== 16'd128) | game_start), .Din(16'b0), .Q(frame128[15:0])); //64 frame counter for
led countdown
    countUD16L ls64 (.clk(clk), .Up(frame1), .Dw(1'b0), .LD ((ledframe[15:0] ==
16'd64) | game_start), .Din(16'b0), .Q(ledframe[15:0])); //64 frame counter for led
shifter

    assign led[15] = (game_running & (ledframe[15:0] >= 16'd0) & (ledframe[15:0] <=
16'd16) ) | (blink & (frame128[15:0] > 16'd0));
    assign led[14] = (game_running & (ledframe[15:0] > 16'd16) & (ledframe[15:0] <=
16'd32) );
    assign led[13] = (game_running & (ledframe[15:0] > 16'd32) & (ledframe[15:0] <=
16'd48) );
    assign led[12] = (game_running & (ledframe[15:0] > 16'd48) & (ledframe[15:0] <=
16'd64) );
    assign led[11] = (game_running & (ledframe[15:0] >= 16'd0) & (ledframe[15:0] <=
16'd16) ) | (blink & (frame128[15:0] > 16'd32));
    assign led[10] = (game_running & (ledframe[15:0] > 16'd16) & (ledframe[15:0] <=
16'd32) );
    assign led[9] = (game_running & (ledframe[15:0] > 16'd32) & (ledframe[15:0] <=
16'd48) );
    assign led[8] = (game_running & (ledframe[15:0] > 16'd48) & (ledframe[15:0] <=
16'd64) );
    assign led[7] = (game_running & (ledframe[15:0] >= 16'd0) & (ledframe[15:0] <=
```

```
16'd16) ) | (blink & (frame128[15:0] > 16'd64));
    assign led[6] = (game_running & (ledframe[15:0] > 16'd16) & (ledframe[15:0] <=
16'd32) );
    assign led[5] = (game_running & (ledframe[15:0] > 16'd32) & (ledframe[15:0] <=
16'd48) );
    assign led[4] = (game_running & (ledframe[15:0] > 16'd48) & (ledframe[15:0] <=
16'd64) );
    assign led[3] = (game_running & (ledframe[15:0] >= 16'd0) & (ledframe[15:0] <=
16'd16) ) | (blink & (frame128[15:0] > 16'd96));
    assign led[2] = (game_running & (ledframe[15:0] > 16'd16) & (ledframe[15:0] <=
16'd32) );
    assign led[1] = (game_running & (ledframe[15:0] > 16'd32) & (ledframe[15:0] <=
16'd48) );
    assign led[0] = (game_running & (ledframe[15:0] > 16'd48) & (ledframe[15:0] <=
16'd64) );

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 02/25/2020 01:59:21 PM
// Design Name:
// Module Name: Vsync
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module Vcounter(
        input clk,
        input CE,
        input R,
        output [15:0] Q,
        output Vsync
    );

    wire utc1, utc2, utc3, dummy;

    Count_4b i0 (.clk(clk), .CE(CE), .R(R), .Q(Q[3:0]), .UTC(utc1));
    Count_4b i1 (.clk(clk), .CE(CE & utc1), .R(R), .Q(Q[7:4]), .UTC(utc2));
    Count_4b i2 (.clk(clk), .CE(CE & utc1 & utc2), .R(R), .Q(Q[11:8]), .UTC(utc3));
    Count_4b i3 (.clk(clk), .CE(CE & utc1 & utc2 & utc3), .R(R), .Q(Q[15:12]),
.UTC(dummy));

    assign Vsync = (Q[15:0] < 16'd489) | (Q[15:0] > 16'd490);

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 02/25/2020 01:59:07 PM
// Design Name:
// Module Name: Hsync
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module Hcounter(
        input clk,
        input CE,
        input R,
        output [15:0] Q,
        output Hsync
    );

    wire utc1, utc2, utc3, dummy;

    Count_4b i0 (.clk(clk), .CE(CE), .R(R), .Q(Q[3:0]), .UTC(utc1));
    Count_4b i1 (.clk(clk), .CE(CE & utc1), .R(R), .Q(Q[7:4]), .UTC(utc2));
    Count_4b i2 (.clk(clk), .CE(CE & utc1 & utc2), .R(R), .Q(Q[11:8]), .UTC(dummy));
    Count_4b i3 (.clk(clk), .CE(CE & utc1 & utc2 & utc3), .R(R), .Q(Q[15:12]),
.UTC(dummy));

    assign Hsync = (Q[15:0] < 16'd655) | (Q[15:0] > 16'd750);

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 02/27/2020 01:42:50 PM
// Design Name:
// Module Name: block
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module block(
        input [3:0] inRed,
        input [3:0] inGreen,
        input [3:0] inBlue,

        input [15:0] x, //bottom right
        input [15:0] xMin,
        input [15:0] xMax,

        input [15:0] y, //bottom right
        input [15:0] yMin,
        input [15:0] yMax,
```

```
        output [3:0] outRed,
        output [3:0] outGreen,
        output [3:0] outBlue
    );

    assign outRed = inRed & {4{(x > xMin & x < xMax) & (y > yMin &
y < yMax)}};
    assign outGreen = inGreen & {4{(x > xMin & x < xMax) & (y >
yMin & y < yMax)}};
    assign outBlue = inBlue & {4{(x > xMin & x < xMax) & (y > yMin
& y < yMax)}};

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 03/03/2020 04:22:29 PM
// Design Name:
// Module Name: countUD16L
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module countUD16L(
        input clk,
        input Up, //count enable (increment)
        input Dw, //not count enable (decrement)
        input LD,
        input [15:0] Din,
        output [15:0] Q,
        output UTC,
        output DTC
    );
    wire utc[3:0];
    wire dtc[3:0];
```

```
    countUD4L in0_1 (.clk(clk), .Up(Up & ~Dw), .Dw(Dw & ~Up),
.LD(LD), .Din(Din[3:0]), .Q(Q[3:0]), .UTC(utc[0]), .DTC(dtc[0]));
    countUD4L in0_2 (.clk(clk), .Up(Up & ~Dw & utc[0]), .Dw(Dw &
~Up & dtc[0]), .LD(LD), .Din(Din[7:4]), .Q(Q[7:4]), .UTC(utc[1]),
.DTC(dtc[1]));
    countUD4L in0_3 (.clk(clk), .Up(Up & ~Dw & utc[0] & utc[1]),
.Dw(Dw & ~Up & dtc[0] & dtc[1]), .LD(LD), .Din(Din[11:8]),
.Q(Q[11:8]), .UTC(utc[2]), .DTC(dtc[2]));
    countUD4L in0_4 (.clk(clk), .Up(Up & ~Dw & utc[0] & utc[1] &
utc[2]), .Dw(Dw & ~Up & dtc[0] & dtc[1] & dtc[2]), .LD(LD),
.Din(Din[15:12]), .Q(Q[15:12]), .UTC(utc[3]), .DTC(dtc[3]));
    assign UTC = utc[3] & utc[2] & utc[1] & utc[0];
    assign DTC = dtc[3] & dtc[2] & dtc[1] & dtc[0];
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 03/03/2020 04:19:54 PM
// Design Name:
// Module Name: countUD4L
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module countUD4L(
        input clk,
        input Up, //count enable (increment)
        input Dw, //not count enable (decrement)
        input LD,
        input [3:0] Din,
        output [3:0] Q,
        output UTC,
        output DTC
    );
    wire [3:0] D;
    wire upen, dwen;
```

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

    FDRE #(.INIT(1'b0)) Q0_FF (.C(clk), .CE(LD | upen | dwen),
.D(D[0]), .Q(Q[0]));
    FDRE #(.INIT(1'b0)) Q1_FF (.C(clk), .CE(LD | upen | dwen),
.D(D[1]), .Q(Q[1]));
    FDRE #(.INIT(1'b0)) Q2_FF (.C(clk), .CE(LD | upen | dwen),
.D(D[2]), .Q(Q[2]));
    FDRE #(.INIT(1'b0)) Q3_FF (.C(clk), .CE(LD | upen | dwen),
.D(D[3]), .Q(Q[3]));
    assign UTC = (Q[0] & Q[1] & Q[2] & Q[3]);
    assign DTC = ~Q[0] & ~Q[1] & ~Q[2] & ~Q[3];

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 02/25/2020 02:25:10 PM
// Design Name:
// Module Name: Count_4b
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module Count_4b(
        input clk,
        input CE, //count enable
        input R, //Reset
        output [3:0] Q,
        output UTC
    );

    wire [3:0] D;

    assign D[0] = ~Q[0];
    assign D[1] = (Q[1] ^ Q[0]) & CE;
    assign D[2] = (Q[2] ^ (Q[1] & Q[0])) & CE;
    assign D[3] = (Q[3] ^ (Q[2] & Q[1] & Q[0])) & CE;

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

    assign UTC = Q[3] & Q[2] & Q[1] & Q[0];

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 02/27/2020 04:34:42 PM
// Design Name:
// Module Name: Edge_Detector
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module Edge_Detector(
        input clk,
        input btn,
        output Eout
    );

    wire [1:0] Q;

    FDRE #(.INIT(1'b0)) Q0_FF (.C(clk), .CE(1'b1), .D(btn), .Q(Q[0]));
    FDRE #(.INIT(1'b0)) Q1_FF (.C(clk), .CE(1'b1), .D(Q[0]), .Q(Q[1])); //takes in
output of last flip flop

    assign Eout = Q[0] & ~Q[1];

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 03/03/2020 01:29:34 AM
// Design Name:
// Module Name: LSFR
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module LFSR(
        input clk,
        output [7:0] Q
    );

    wire rand;

    assign rand = Q[0] ^ Q[5] ^ Q[6] ^ Q[7];

    FDRE #(.INIT(1'b1)) Q0_FF (.C(clk), .CE(1'b1), .D(rand),
.Q(Q[0])); //xor some bits for 1st bit
    FDRE #(.INIT(1'b0)) Q1_FF (.C(clk), .CE(1'b1), .D(Q[0]),
.Q(Q[1])); //shift rand val down
```

```
    FDRE #(.INIT(1'b0)) Q2_FF (.C(clk), .CE(1'b1), .D(Q[1]),
.Q(Q[2]));
    FDRE #(.INIT(1'b0)) Q3_FF (.C(clk), .CE(1'b1), .D(Q[2]),
.Q(Q[3]));
    FDRE #(.INIT(1'b0)) Q4_FF (.C(clk), .CE(1'b1), .D(Q[3]),
.Q(Q[4]));
    FDRE #(.INIT(1'b0)) Q5_FF (.C(clk), .CE(1'b1), .D(Q[4]),
.Q(Q[5]));
    FDRE #(.INIT(1'b0)) Q6_FF (.C(clk), .CE(1'b1), .D(Q[5]),
.Q(Q[6]));
    FDRE #(.INIT(1'b0)) Q7_FF (.C(clk), .CE(1'b1), .D(Q[6]),
.Q(Q[7]));

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 03/03/2020 01:31:50 AM
// Design Name:
// Module Name: Set_Random
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module Set_Random(
        input clk,
        input CE,
        input [7:0] rand,
        output [15:0] set
    );

    //wire [15:0] set; //pos
    //wire [15:0] neg;
    //wire sign;

    FDRE #(.INIT(1'b0)) Q0_FF (.C(clk), .CE(CE), .D(1'b0),
.Q(set[0]));
```

```
    FDRE #(.INIT(1'b0)) Q1_FF (.C(clk), .CE(CE), .D(1'b0),
.Q(set[1]));
    FDRE #(.INIT(1'b0)) Q2_FF (.C(clk), .CE(CE), .D(1'b0),
.Q(set[2]));
    FDRE #(.INIT(1'b0)) Q3_FF (.C(clk), .CE(CE), .D(rand[0]),
.Q(set[3])); //4th bit
    FDRE #(.INIT(1'b0)) Q4_FF (.C(clk), .CE(CE), .D(rand[1]),
.Q(set[4])); //5th bit
    FDRE #(.INIT(1'b0)) Q5_FF (.C(clk), .CE(CE), .D(rand[2]),
.Q(set[5])); //6th bit
    FDRE #(.INIT(1'b0)) Q6_FF (.C(clk), .CE(CE), .D(rand[3]),
.Q(set[6])); //sign bit
    FDRE #(.INIT(1'b0)) Q7_FF (.C(clk), .CE(CE), .D(rand[3]),
.Q(set[7]));
    FDRE #(.INIT(1'b0)) Q8_FF (.C(clk), .CE(CE), .D(rand[3]),
.Q(set[8]));
    FDRE #(.INIT(1'b0)) Q9_FF (.C(clk), .CE(CE), .D(rand[3]),
.Q(set[9]));
    FDRE #(.INIT(1'b0)) Q10_FF (.C(clk), .CE(CE), .D(rand[3]),
.Q(set[10]));
    FDRE #(.INIT(1'b0)) Q11_FF (.C(clk), .CE(CE), .D(rand[3]),
.Q(set[11]));
    FDRE #(.INIT(1'b0)) Q12_FF (.C(clk), .CE(CE), .D(rand[3]),
.Q(set[12]));
    FDRE #(.INIT(1'b0)) Q13_FF (.C(clk), .CE(CE), .D(rand[3]),
.Q(set[13]));
    FDRE #(.INIT(1'b0)) Q14_FF (.C(clk), .CE(CE), .D(rand[3]),
.Q(set[14]));
    FDRE #(.INIT(1'b0)) Q15_FF (.C(clk), .CE(CE), .D(rand[3]),
.Q(set[15]));

    //twos_comp conv (.count(set[6:0]), .s(neg[6:0]));

    //assign set[15:7] = {9{1'b0}}; //sign extend positive
    //assign neg[15:7] = {9{1'b1}}; //sign extend negative

    //m2_1x16 choose (.in0(set[15:0]), .in1(neg[15:0]),
```

```
.sel(set[6]), .o(final[15:0]));

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 03/03/2020 09:40:26 PM
// Design Name:
// Module Name: time_counter
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module Time_Counter(
        input clk,
        input CE, //count enable
        input R, //Reset
        output [15:0] Q
    );

    wire utc1, utc2, utc3, utc4;

    Count_4b i0 (.clk(clk), .CE(CE), .R(R), .Q(Q[3:0]),
.UTC(utc1));
    Count_4b i1 (.clk(clk), .CE(utc1 & CE), .R(R), .Q(Q[7:4]),
.UTC(utc2));
```

```
    Count_4b i2 (.clk(clk), .CE(utc1 & utc2 & CE), .R(R),
.Q(Q[11:8]), .UTC(utc3));
    Count_4b i3 (.clk(clk), .CE(utc1 & utc2 & utc3 & CE), .R(R),
.Q(Q[15:12]), .UTC(utc4));
    //utc for counters before current need to be 1 first

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 03/03/2020 12:15:09 AM
// Design Name:
// Module Name: state_machine
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module state_machine(
        input clk, //frame
        input start, //button C
        input TwoSecs, //2 seconds passing
        input lose, //out of bounds
        output blink, //car blinks
        output game_start, //game has started (start of game)
        output game_running, //game is still running (increment
score and move LEDS)
        output game_over, //game is over, flash score, LEDS OFF
        output [3:0] debugStates
    );
```

```
    wire IDLE, BTWICE, RUNNING, POST; //current state
    wire next_IDLE, next_BTWICE, next_RUNNING, next_POST; //next
state

    assign next_IDLE = IDLE & ~start;
    assign next_BTWICE = IDLE & start | POST & start | BTWICE &
~TwoSecs;
    assign next_RUNNING = BTWICE & TwoSecs | RUNNING & ~lose;
    assign next_POST = RUNNING & lose | POST & ~start;

    FDRE #(.INIT(1'b1)) Q_0FF (.C(clk), .CE(1'b1), .D(next_IDLE),
.Q(IDLE) );
    FDRE #(.INIT(1'b0)) Q_1FF (.C(clk), .CE(1'b1), .D(next_BTWICE),
.Q(BTWICE) );
    FDRE #(.INIT(1'b0)) Q_2FF (.C(clk), .CE(1'b1),
.D(next_RUNNING), .Q(RUNNING) );
    FDRE #(.INIT(1'b0)) Q_3FF (.C(clk), .CE(1'b1), .D(next_POST),
.Q(POST) );

    assign blink = BTWICE & ~TwoSecs;
    assign game_start = IDLE | POST & start;
    assign game_running = RUNNING & ~lose;
    assign game_over = POST & ~start;

    assign debugStates = {POST, RUNNING, BTWICE, IDLE};

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 03/02/2020 11:48:55 PM
// Design Name:
// Module Name: RingCounter
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module RingCounter(
        input clk,
        input Advance,
        output [3:0] Q
    );


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

```
    FDRE #(.INIT(1'b0)) Q3_FF (.C(clk), .CE(Advance), .D(Q[3]),
.Q(Q[0]));

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 03/02/2020 11:47:25 PM
// Design Name:
// Module Name: Selector
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module Selector(
        input [3:0] sel,
        input [15:0] N,
        output [3:0] H
    );

    assign H[3:0] = (N[15:12] & {4{sel[3]}}) | (N[11:8] &
{4{sel[2]}}) | (N[7:4] & {4{sel[1]}}) | (N[3:0] & {4{sel[0]}});

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 03/02/2020 11:48:13 PM
// Design Name:
// Module Name: hex7seg
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module hex7seg ( //from lab 2
        input n3,
        input n2,
        input n1,
        input n0,
        output a,
        output b,
        output c,
        output d,
        output e,
        output f,
        output g
    );
```

```
    assign a = (~n3 & ~n2 & ~n1 & n0) | (~n3 & n2 & ~n1 & ~n0) |
(n3 & ~n2 & n1 & n0) | (n3 & n2 & ~n1 & n0);
    assign b = (~n3 & n2 & ~n1 & n0) | (~n3 & n2 & n1 & ~n0) | (n3
& ~n2 & n1 & n0) | (n3 & n2 & ~n1 & ~n0) | (n3 & n2 & n1 & ~n0) |
(n3 & n2 & n1 & n0);
    assign c = (~n3 & ~n2 & n1 & ~n0) | (n3 & n2 & ~n1 & ~n0) | (n3
& n2 & n1 & ~n0) | (n3 & n2 & n1 & n0);
    assign d = (~n3 & ~n2 & ~n1 & n0) | (~n3 & n2 & ~n1 & ~n0) |
(~n3 & n2 & n1 & n0) | (n3 & ~n2 & ~n1 & n0) | (n3 & ~n2 & n1 &
~n0) | (n3 & n2 & n1 & n0);
    assign e = (~n3 & ~n2 & ~n1 & n0) | (~n3 & ~n2 & n1 & n0) |
(~n3 & n2 & ~n1 & ~n0) | (~n3 & n2 & ~n1 & n0) | (~n3 & n2 & n1 &
n0) | (n3 & ~n2 & ~n1 & n0);
    assign f = (~n3 & ~n2 & ~n1 & n0) | (~n3 & ~n2 & n1 & ~n0) |
(~n3 & ~n2 & n1 & n0) | (~n3 & n2 & n1 & n0) | (n3 & n2 & ~n1 &
n0);
    assign g = (~n3 & ~n2 & ~n1 & ~n0) | (~n3 & ~n2 & ~n1 & n0) |
(~n3 & n2 & n1 & n0) | (n3 & n2 & ~n1 & ~n0);

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 03/05/2020 03:06:11 PM
// Design Name:
// Module Name: top_sim
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module top_sim();
// Inputs
   reg btnC,btnD,btnU,btnL,btnR;
   reg [15:0] sw;
   reg clkin;

// Output
   wire [3:0] an;
   wire dp;
   wire [6:0] seg;
   wire [15:0] led;
   wire HS;
   wire [3:0] vgaBlue;
   wire [3:0] vgaGreen;
   wire [3:0] vgaRed;
   wire VS;

top_level UUT(
        .btnU(btnU),
        .btnD(btnD),
        .btnC(btnC),
        .btnR(btnR),
        .btnL(btnL),
        .clkin(clkin),
```

```
        .seg(seg),
        .dp(dp),
        .an(an),
        .vgaBlue(vgaBlue),
        .vgaRed(vgaRed),
        .vgaGreen(vgaGreen),
        .Vsync(VS),
        .Hsync(HS),
        .sw(sw),
        .led(led)
);

    parameter PERIOD = 10;
    parameter real DUTY_CYCLE = 0.5;
    parameter OFFSET = 2;
    initial // Clock process for clkin
    begin
        #OFFSET
            clkin = 1'b1;
        forever
        begin
            #(PERIOD-(PERIOD*DUTY_CYCLE)) clkin = ~clkin;
        end
    end

    initial
    begin
    //current time is 0ns
    //-------------------- all 0 (IDLE)
    btnC = 1'b0;
    btnR = 1'b0;
    btnL = 1'b0;
    sw = 16'd0;

    #1000; //current time is 1000ns
    //-------------------- all 0 (IDLE)
    btnC = 1'b0;
    btnR = 1'b0;
    btnL = 1'b0;

    #100; //current time is 1100ns
    //-------------------- start game (BTWICE)
    btnC = 1'b1;
    btnR = 1'b0;
    btnL = 1'b0;
```

```
    #100; //current time is 1200ns
    //-------------------- not two seconds yet (BTWICE)
    btnC = 1'b0;
    btnR = 1'b0;
    btnL = 1'b0;

    #100; //current time is 1300ns
    //-------------------- blinked for 2 seconds (RUNNING)
    btnC = 1'b0;
    btnR = 1'b0;
    btnL = 1'b0;

    #100; //current time is 1400ns
    //-------------------- have not lost yet (RUNNING)
    btnC = 1'b0;
    btnR = 1'b0;
    btnL = 1'b0;

    #100; //current time is 1500ns
    //-------------------- out of bounds (POST)
    btnC = 1'b0;
    btnR = 1'b0;
    btnL = 1'b0;

    #100; //current time is 1600ns
    //-------------------- not started new game yet (POST)
    btnC = 1'b0;
    btnR = 1'b0;
    btnL = 1'b0;

    #100; //current time is 1700ns
    //-------------------- started new game (BTWICE)
    btnC = 1'b1;
    btnR = 1'b0;
    btnL = 1'b0;

    #100; //current time is 1800ns
    end

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 03/03/2020 08:47:52 PM
// Design Name:
// Module Name: state_sim
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module state_sim();
        reg clk; //frame
        reg start; //button C
        reg TwoSecs; //2 seconds passing
        reg lose; //out of bounds
        wire blink; //car blinks
        wire game_running; //game is still running (increment score and move LEDS)
        wire game_over; //game is over, flash score, LEDS OFF

state_machine UUT(
        .clk(clk),
        .start(start),
        .TwoSecs(TwoSecs),
        .lose(lose),
        .blink(blink),
        .game_running(game_running),
        .game_over(game_over)
);

    parameter PERIOD = 10;
    parameter real DUTY_CYCLE = 0.5;
    parameter OFFSET = 2;
    initial // Clock process for clkin
    begin
```

```
    #OFFSET
        clk = 1'b1;
    forever
    begin
        #(PERIOD-(PERIOD*DUTY_CYCLE)) clk = ~clk;
    end
end

initial
begin
//current time is 0ns
//-------------------- all 0 (IDLE)
start = 1'b0;
TwoSecs = 1'b0;
lose = 1'b0;

#1000; //current time is 1000ns
//-------------------- all 0 (IDLE)
start = 1'b0;
TwoSecs = 1'b0;
lose = 1'b0;

#100; //current time is 1100ns
//-------------------- start game (BTWICE)
start = 1'b1;
TwoSecs = 1'b0;
lose = 1'b0;

#100; //current time is 1200ns
//-------------------- not two seconds yet (BTWICE)
start = 1'b0;
TwoSecs = 1'b0;
lose = 1'b0;

#100; //current time is 1300ns
//-------------------- blinked for 2 seconds (RUNNING)
start = 1'b0;
TwoSecs = 1'b1;
lose = 1'b0;

#100; //current time is 1400ns
//-------------------- have not lost yet (RUNNING)
start = 1'b0;
TwoSecs = 1'b0;
lose = 1'b0;
```

```
    #100; //current time is 1500ns
    //-------------------- out of bounds (POST)
    start = 1'b0;
    TwoSecs = 1'b0;
    lose = 1'b1;

    #100; //current time is 1600ns
    //-------------------- not started new game yet (POST)
    start = 1'b0;
    TwoSecs = 1'b0;
    lose = 1'b0;

    #100; //current time is 1700ns
    //-------------------- started new game (BTWICE)
    start = 1'b1;
    TwoSecs = 1'b0;
    lose = 1'b0;

    #100; //current time is 1800ns
    end

endmodule
```

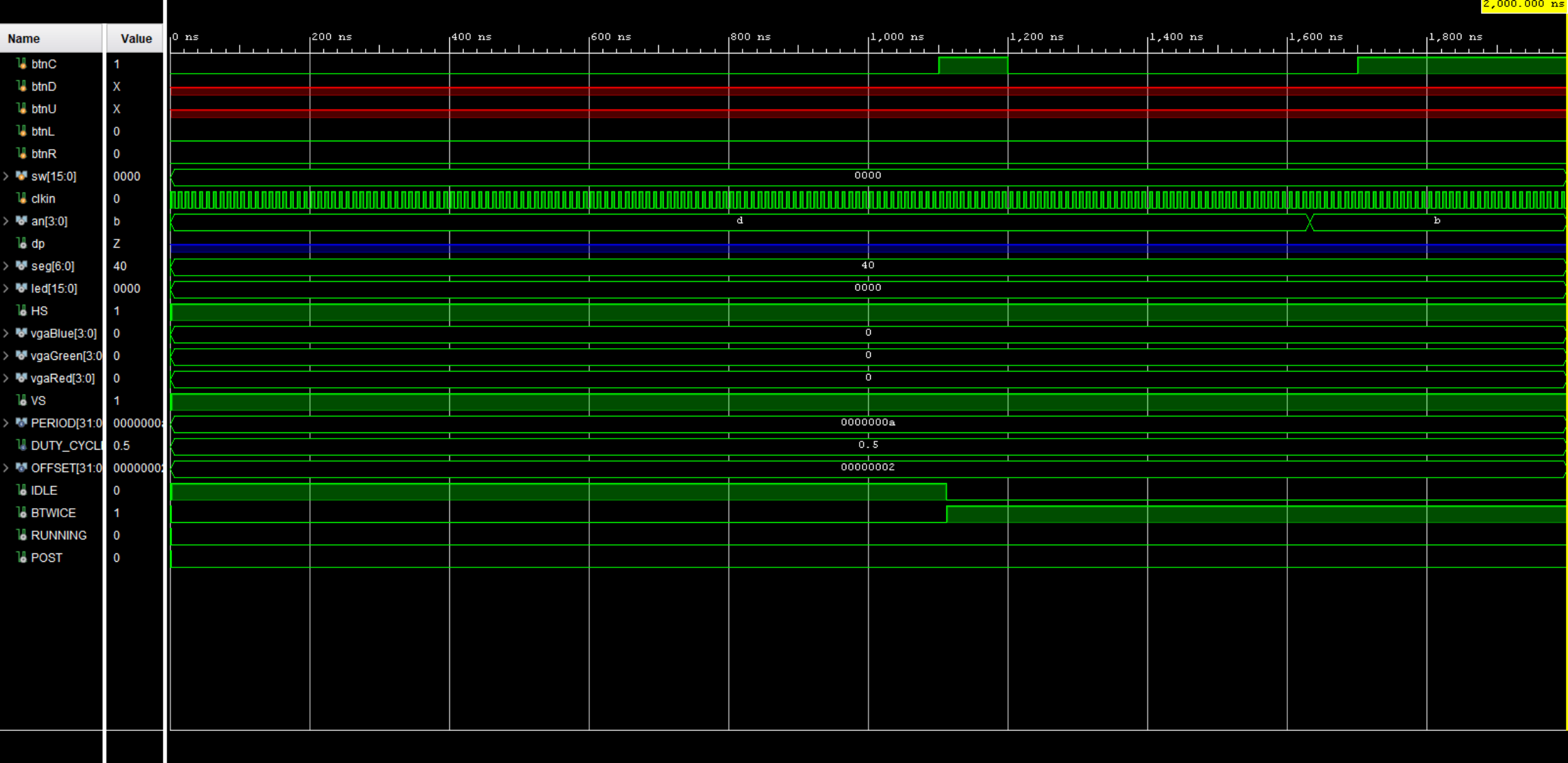
```
// Verilog test fixture created to test sync signals
//  Instructions:
// 1. Add to your project.
// 2. Adjust instantiation of UUT (line 43).
// 3. Comment out lines 107 and 132 if your outputs are not run through FFs.
// 4. Run Simulator.
// 5. Add SERRORS and RGBERRORS to the waveform viewer (under GUT and RUT)
// 6. Reset simulation time to 0
// 7. Set step size to 17ms
// 8. Simulate one step (may take ?? seconds, or more if you have more components.
// 9. If SERRORs or RGBERRORS is not 0 find where they changed: that's a problem.
// check for transitions on "oops" and rgb_oops" to find out where
//10. Otherwise simulate one more step and check them again.
//
//   Martine  Feb 22 2019

`timescale 1ns / 1ps

module testSyncs();

// Inputs
   reg btnC,btnD,btnU,btnL,btnR;
   reg [15:0] sw;
   reg clkin;

// Output
   wire [3:0] an;
   wire dp;
   wire [6:0] seg;
   wire [15:0] led;
   wire HS;
   wire [3:0] vgaBlue;
   wire [3:0] vgaGreen;
   wire [3:0] vgaRed;
   wire VS;
   wire oops;
   wire rgb_oops;


// You may need to replace the instantiation below
// to match your top level module and its port names.

// Instantiate the UUT
   top_level UUT (
        .btnU(btnU),
        .btnD(btnD),
```

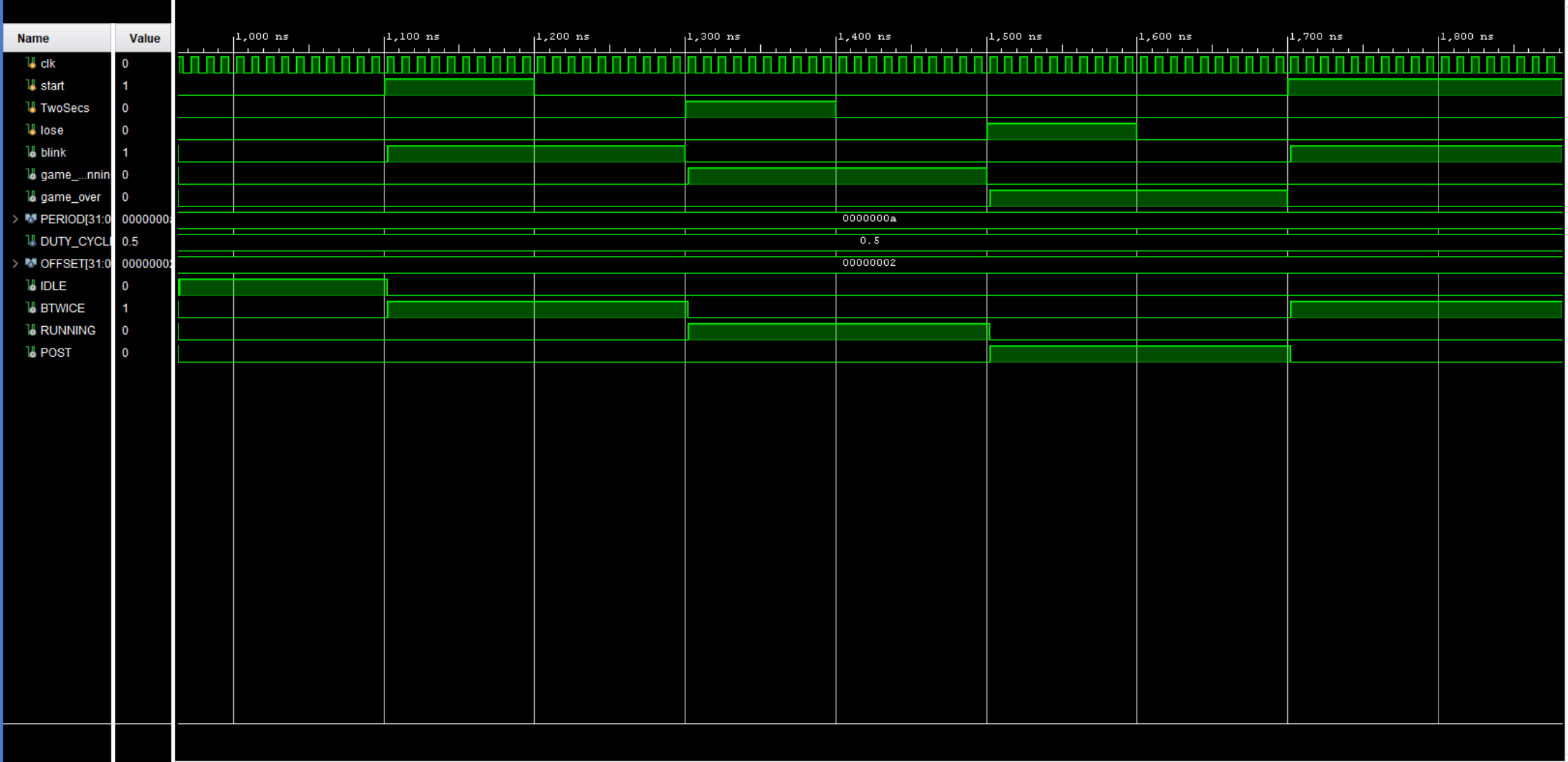
```
      .btnC(btnC),
      .btnR(btnR),
      .btnL(btnL),
      .clkin(clkin),
      .seg(seg),
      .dp(dp),
      .an(an),
      .vgaBlue(vgaBlue),
      .vgaRed(vgaRed),
      .vgaGreen(vgaGreen),
      .Vsync(VS),
      .Hsync(HS),
      .sw(sw),
      .led(led)
  );

   reg good_HS;
   reg good_VS;
   reg activeH;
   reg activeV;

// Clock parameters
  parameter PERIOD = 10;
  parameter real DUTY_CYCLE = 0.5;
  parameter OFFSET = 2;

   initial
   begin
       clkin = 1'b0;
      #OFFSET
       clkin = 1'b1;
     forever
        begin
           #(PERIOD-(PERIOD*DUTY_CYCLE)) clkin = ~clkin;
        end
   end

   initial // the only input needed is a GSR pulse on sw[0]
     begin // if you have others, set them to 0 or 1
      sw = 16'b0;
      sw[0] = 1'b0;
      btnU = 1'b0;
      btnC = 1'b0;
      btnD = 1'b0;
      #600 sw[0] = 1'b1;
      #80 sw[0] = 1'b0;
```

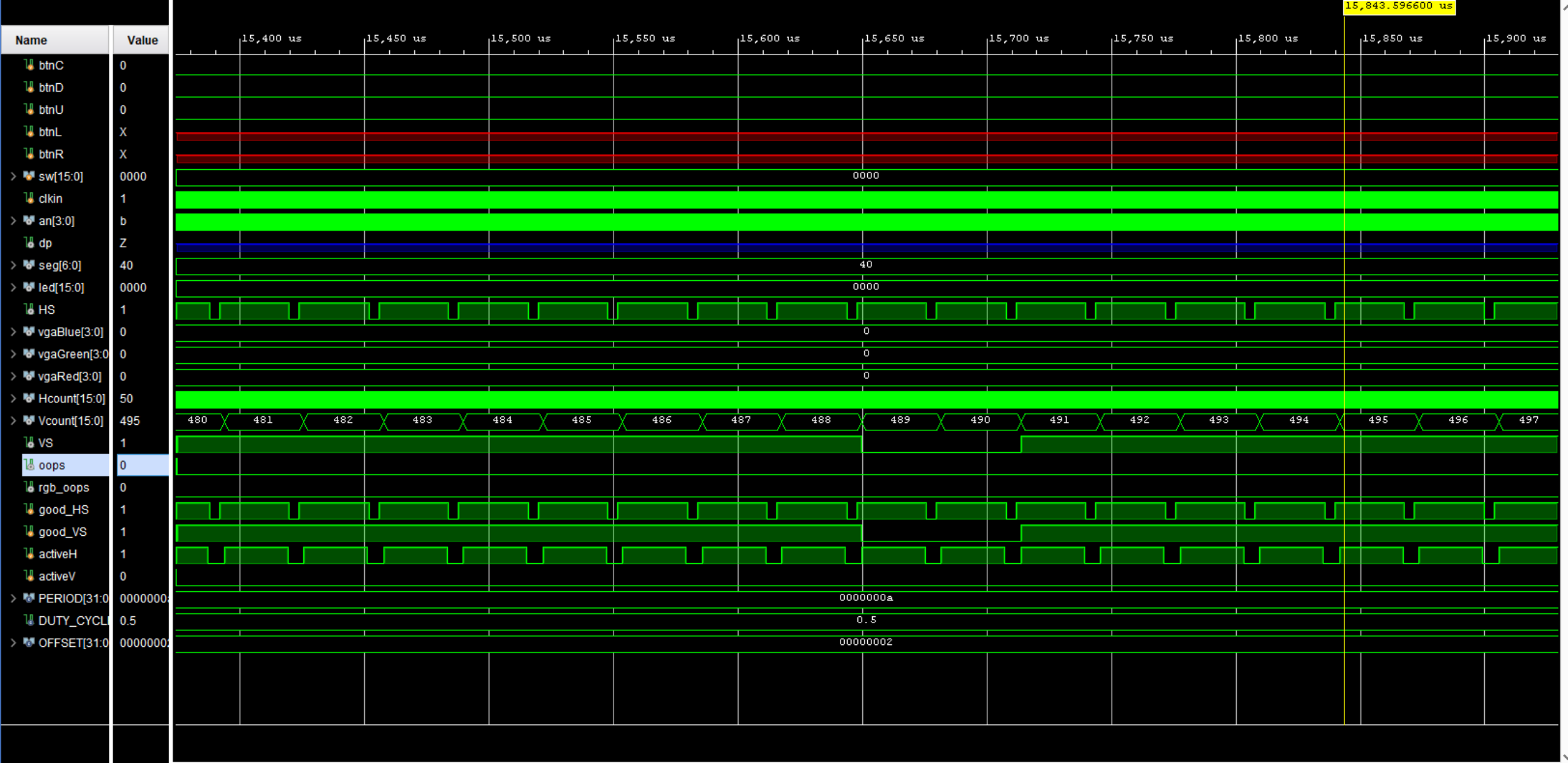
```
   end

// process to generate correct values for HS and activeH
initial
   begin
     #OFFSET;
     #0.1;

     activeH = 1'b1;
     good_HS = 1'b1;
     #880;
     #700; // to get past gsr pulse and clock delay
      // comment out the line below if your HSync is not passed through a
      // D FF before leaving the FPGA
     //#40;  // one more clock for FF delay output
     forever
         begin
           activeH = 1'b1;
          #(640*40);
           activeH = 1'b0;
         #(15*40);
         good_HS = 1'b0;
         #((96)*40);
         good_HS = 1'b1;
         #(49*40);
        end
  end

//correct values for VS and activeV
initial
   begin
     #OFFSET;
     #0.1;
     activeV = 1'b1;
     good_VS = 1'b1;
     #880;
     #700; // to get past gsr and DCM clock delay
    // comment out the line below if your VSync is not passed through a
      // D FF before leaving the FPGA
     //#40;  // one more clock for FF delay output
     //btnL=1;
     //#200
     //btnL=0;
      forever
         begin
         activeV = 1'b1;
```

```
            #(480*800*40);
            activeV=1'b0;
              #(9*800*40);
             good_VS = 1'b0;
             #(2*800*40);
             good_VS = 1'b1;
             #(34*800*40);
           end
     end



// Instantiate the module comparing good and actual sync signals
   check_the_sync_signals GUT (
        .myHS(HS),
        .myVS(VS),
        .correct_HS(good_HS),
        .correct_VS(good_VS),
        .clk(clkin),
        .btnR(sw[0]),
        .sync_error(oops)
   );

// Instantiate the module comparing checking RGB signals are
// low outside the active region
   check_the_rgb_signals RUT (.VB(vgaBlue), .VG(vgaGreen), .VR(vgaRed),
       .activeH(activeH), .activeV(activeV),
        .clk(clkin),
          .btnR(sw[0]),
         .rgb_error(rgb_oops)
   );

endmodule

module check_the_sync_signals(
 input myHS,
 input myVS,
 input correct_HS,
 input correct_VS,
 input clk,
 input btnR,
 output sync_error);

   // sync_error is high when actual and expected sync signals differ
  assign sync_error = (myHS^correct_HS)|(myVS^correct_VS);
```

```
   // SERRORS is incremented when sync_error is high at the rising edge of betterclk
  integer SERRORS;

   // since betterclk is divided by 2 in clkcntrl4 the error count
    // will be double (each error will be counted twice)
  always @(posedge clk)
    begin
        SERRORS  <= SERRORS + sync_error;   // non-blocking assignment
     end

     // reset SERRORS on rising edge of gsr
  always @(posedge btnR)
    begin
        SERRORS <= 32'b0;   //  non-blocking assignment
     end

endmodule

module check_the_rgb_signals(VB, VG, VR,
                      activeH, activeV, clk, btnR, rgb_error);

 input [3:0] VB, VG, VR;
 input activeH, activeV;
 input clk, btnR;
 output rgb_error;

   // rgb_error is high when any RGB output is high outside the active region
 assign rgb_error = ((|VR)|(|VB)|(|VG))&~(activeH*activeV);

   // RGBERRORS is incremented when rgb_error is high at the rising edge of betterclk
  integer RGBERRORS;

   // since betterclk is divided by 2 in clkcntrl4 the error count
    // will be double (each error will be counted twice)
  always @(posedge clk)
    begin
        RGBERRORS  <= RGBERRORS + rgb_error;   // non-blocking assignment
     end

     // reset SERRORS on rising edge of gsr
  always @(posedge btnR)
    begin
        RGBERRORS <= 32'b0;   //  non-blocking assignment
     end

endmodule
```


Name
Value
0 ns
200 ns
400 ns
600 ns
800 ns
1,000 ns
1,200 ns
1,400 ns
1,600 ns
1,800 ns
2,000.000 ns
btnC
1
btnD
X
btnU
X
btnL
0
btnR
0
sw[15:0]
0000
0000
clkin
0
an[3:0]
b
d
b
dp
Z
seg[6:0]
40
40
led[15:0]
0000
0000
HS
1
vgaBlue[3:0]
0
0
vgaGreen[3:0
0
0
vgaRed[3:0]
0
0
VS
1
PERIOD[31:0
0000000a
0000000a
DUTY_CYCLI
0.5
0.5
OFFSET[31:0
00000002
00000002
IDLE
0
BTWICE
1
RUNNING
0
POST
0


Name
Value
clk
0
start
1
TwoSecs
0
lose
0
blink
1
game_...nnin
0
game_over
0
PERIOD[31:0
0000000
DUTY_CYCL
0.5
OFFSET[31:0
0000000
IDLE
0
BTWICE
1
RUNNING
0
POST
0
1,000 ns
1,100 ns
1,200 ns
1,300 ns
1,400 ns
1,500 ns
1,600 ns
1,700 ns
1,800 ns
0000000a
0.5
00000002


Name
Value
btnC 0
btnD 0
btnU 0
btnL X
btnR X
sw[15:0] 0000
clkin 1
an[3:0] b
dp Z
seg[6:0] 40
led[15:0] 0000
HS 1
vgaBlue[3:0] 0
vgaGreen[3:0] 0
vgaRed[3:0] 0
Hcount[15:0] 50
Vcount[15:0] 495
VS 1
oops 0
rgb_oops 0
good_HS 1
good_VS 1
activeH 1
activeV 0
PERIOD[31:0] 0000000
DUTY_CYCL 0.5
OFFSET[31:0] 0000000
15,400 us
15,450 us
15,500 us
15,550 us
15,600 us
15,650 us
15,700 us
15,750 us
15,800 us
15,850 us
15,900 us
15,843.596600 us
0000
40
0000
0
0
0
480 481 482 483 484 485 486 487 488 489 490 491 492 493 494 495 496 497
0000000a
0.5
00000002

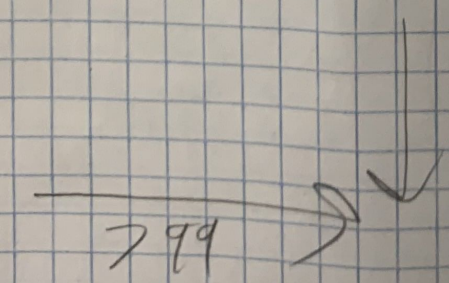
Lab 7

(Tue)
2/25/2020
1:30 pm

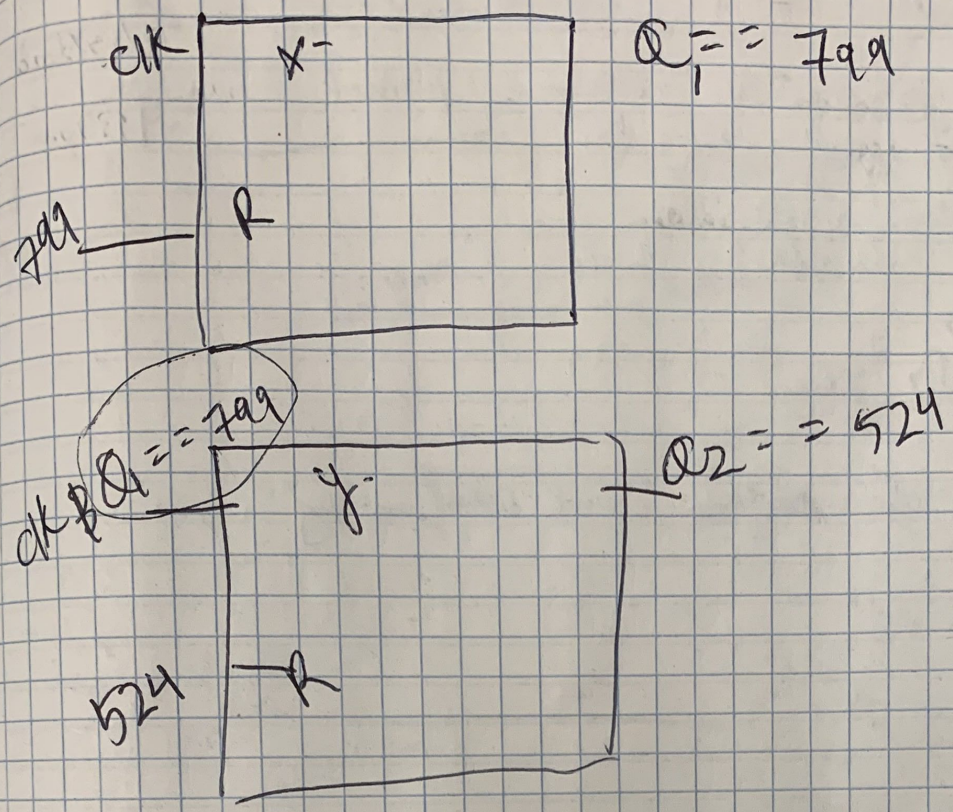
- maintain 2 counters
- output a signal when reaches certain values (regions)
- counts by clock
- hsync and vsync are counters that reset at certain values

H counter
- counts from 0 to 799
- Hsync signal is low from 656 to 750 = 10'b1010010000 to 10'b1011101110
- resets at 799 = 10'b1100011111

V counter
- counts from 0 to 524
- Vsync signal is low from 490 to 491 = 10'b0111101010 to 10'b0111101011
- resets at 524 = 10'b1000001100



Lab 7



- moves 1 pixel at a time left to right (then next row)

- make module for 1 block

(Thur)

2/27/2020

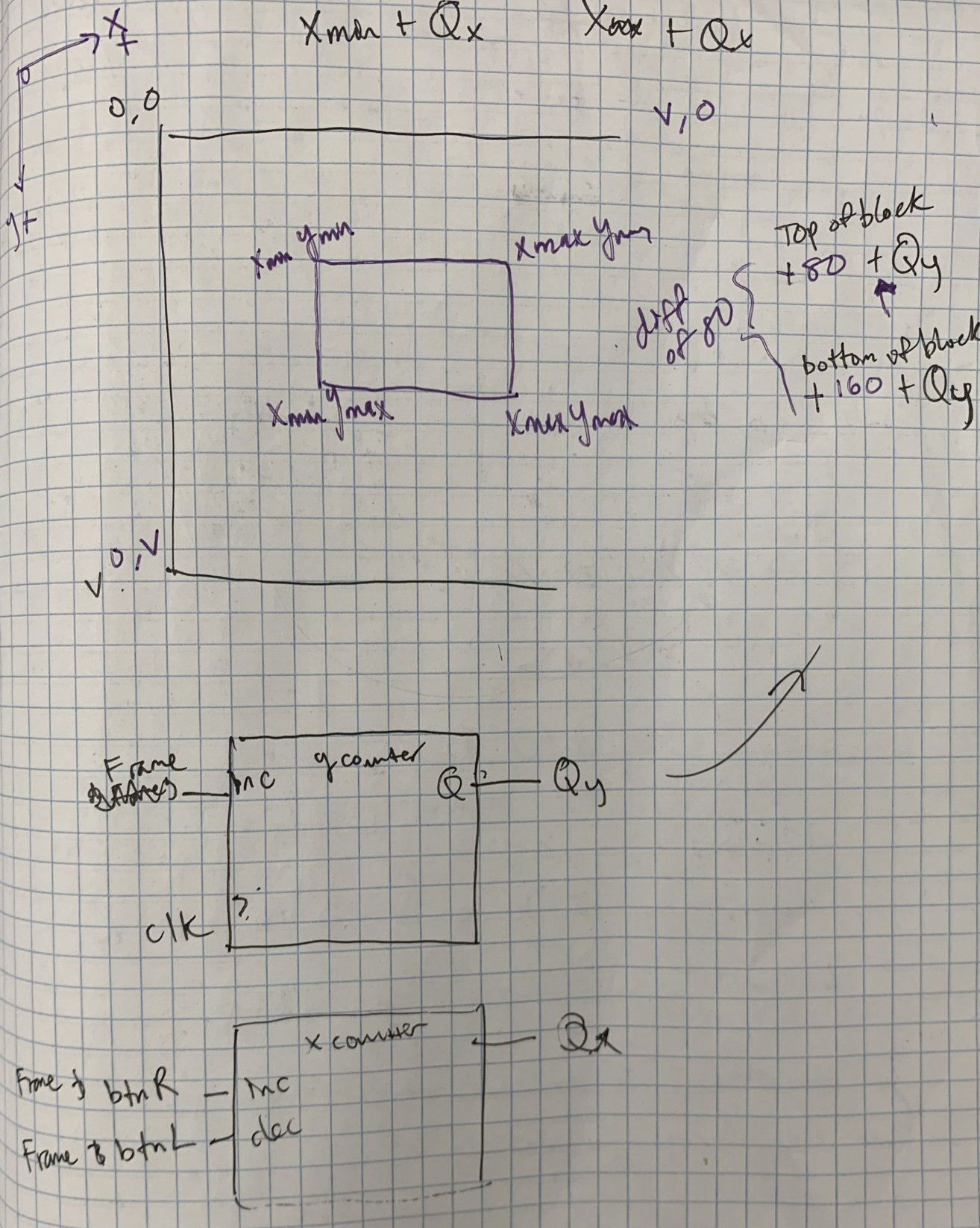
1:30pm

- right bottom pixel is reference point
  - (so it's not neg when coming from top

- counting up to 60 frames/sec is 1 sec

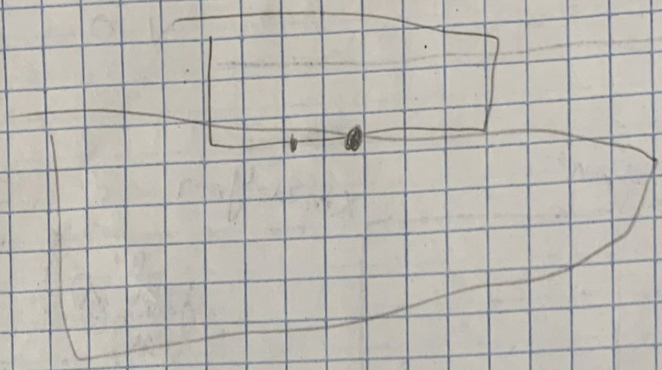
- 2 pixels per frame

- make edge detector to count frames using vsync


x+
y+
0,0
V,0
0,V
V,V
xmin ymin
xmax ymin
xmin ymax
xmax ymax
Top of block
+80 + Qy
diff of 80
bottom of block
+160 + Qy
Frame
inc
y counter
Q
Qy
clk
x counter
Qx
Frame & btnR
Frame & btnL
inc
dec

Lab 7

3/2/2020



v=0

v_counter ≥ v

v - 10'd79 ≤ v_counter ≤ v + 10'd79

Pixels

Segment Coords

h_coord

h_counter_out ≥ h_coord - width + 1 &

h_counter_out ≤ h_coord + width &

v_counter ≥ v_coord - 10'd79 &

v_counter ≤ v_coord;

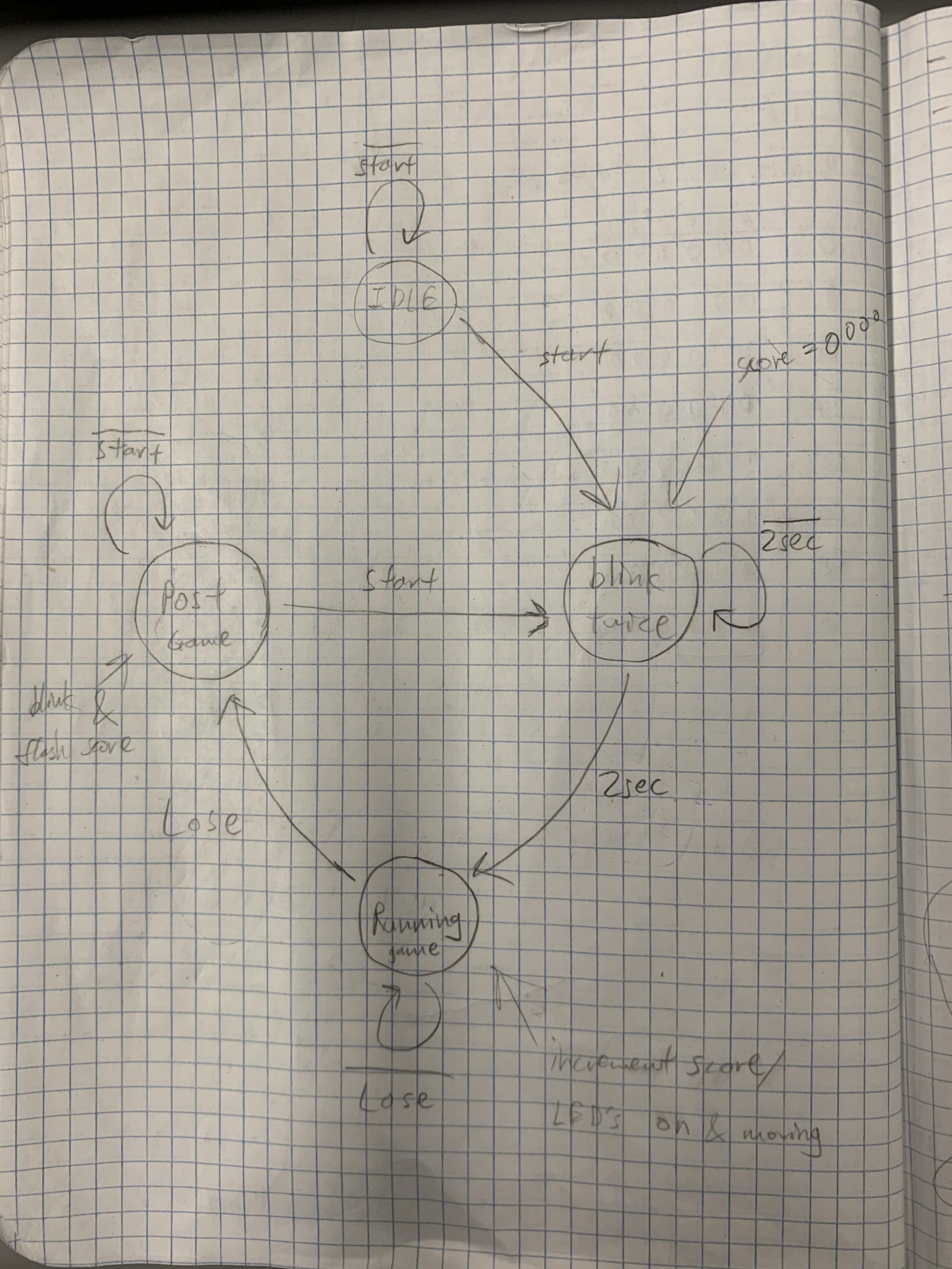
v_counter + 10'd79 ≥ v_coord

(bot ≤ y + 80) & (y ≤ bot)

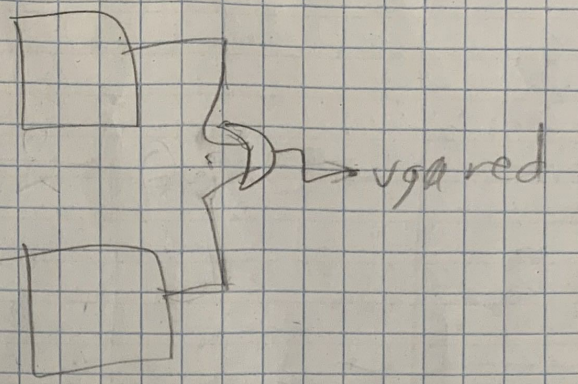
LSFR

0, 8, 16, 24, 32, 40, 48, 56

	6	5	4	3	2	1
0	0	0	0	0	0	0
8	0	0	1	0	0	0
16	0	1	0	0	0	0
24	0	1	1	0	0	0
32	1	0	0	0	0	0
40	1	0	1	0	0	0
48	1	1	0	0	0	0
56	1	1	1	0	0	0


start
IDLE
start
score = 0000
start
2sec
Post Game
start
blink twice
blink & flash score
2sec
Lose
Running game
Lose
increment score/ LED's on & moving

- car blinks every 1 second
- use ring counter to move leds



- work on state machine so you can get game start and turn when to load in initial starting point for blocks so you can make blocks

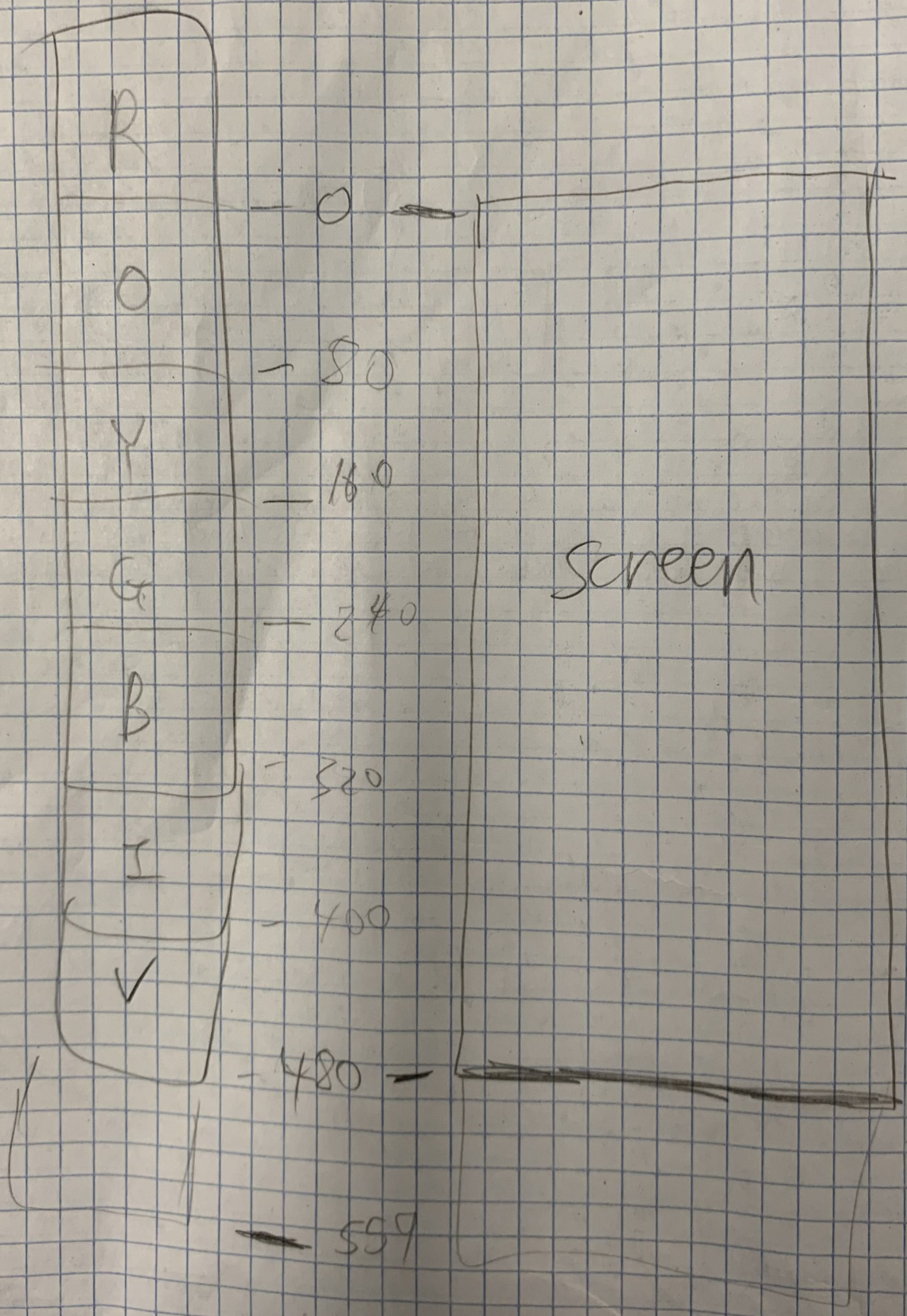
120-8 = 112 ÷ 16 = 7

- set speed to frame for CE (car)
- set LSFR CE signal to block reset signal
- should not have black line between (off by 1 error)

- buttons control road, not car

(06)

3/5/2020



~~fix~~ load values for random
(width or width/2
probably causing problems rn)

- make respawn base on block below NOT above

-8 = 1 1 1 1 1 0 0 0

0 0 0 0 1 1 1
\+ 1

8 0 0 0 1 0 0 0

Lab7

3/6/2020
@ 10am

off road detection

- off road if car x & y are outside road seg
- AND > of these

||(|)

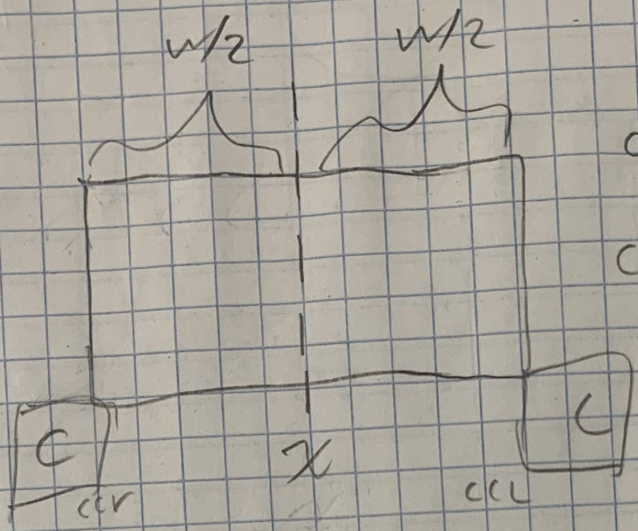
To Do

- ✓ blink
- ✓ die
- ✓ flash (score)
- ✓ countdown start LED
- ✓ running LED

J.W. 4:55pm
checkoff 3/9 3:01 p.m.
3/7
Canvas #7008
(20% deduction)

Lab7

3/8/2020
10:00pm



ccL = corner column left

ccR = corner column right

ccL = 312
ccR = 328

$x - w/2 < ccR$

$x < ccR + w/2$

⇓

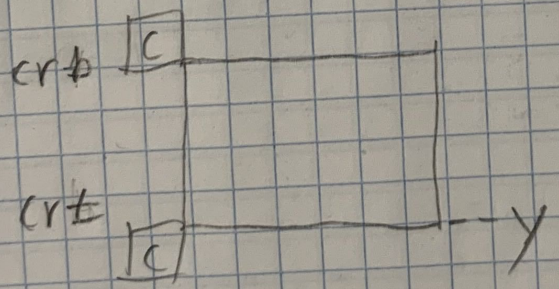
inside left bound

$x > ccL - w/2$

$x + w/2 > ccL$

⇓

inside right bound



crb = corner row top

crt = corner row bottom

crb = 408

crt = 392

$y - 80 < ccb$

$y < ccb + 80$

⇓

inside top bound

$y > crt$

⇓

inside bottom bound