Aidan Smith

12/6/19

Section 01D

Lab 7 Report

Description:

This lab had us make a game using the VGA for the first time. We were to create a bird catching game, with birds, a net, and walls holding everything inside. This is all displayed on the monitor. It is then controlled by switches and buttons which change the time to catch the birds, the net's movement, the net's size, and starting/resetting the game. The game's goal is to catch all of the birds in the alloted time. To create this we needed to use more than just one state machine like we were used to.

Methods:

To create the lab we needed to use lots of components from previous labs, adders, muxes, counters, etc. Using these, we had to write logic for 3 main machines, the game's machine, the net's machine, and the bird's machines. For the birds we just needed to write one and then duplicate them when we were ready to make all 8 in the game. We had to start by creating the border and that involved working with the VGA for the first time. After this we added a bird and then later on a net. Then we could add the game knowledge and another bird to test the main functions. Normally we just used the hex display but now we used that with the VGA too. Thus we also needed our hex display logic in our lab. We used our components from previous labs to

help our machines in order to create the playable game. I put all of my VGA logic in my top level as it wasn't too horrendous.

Results:

Breakdown of top level

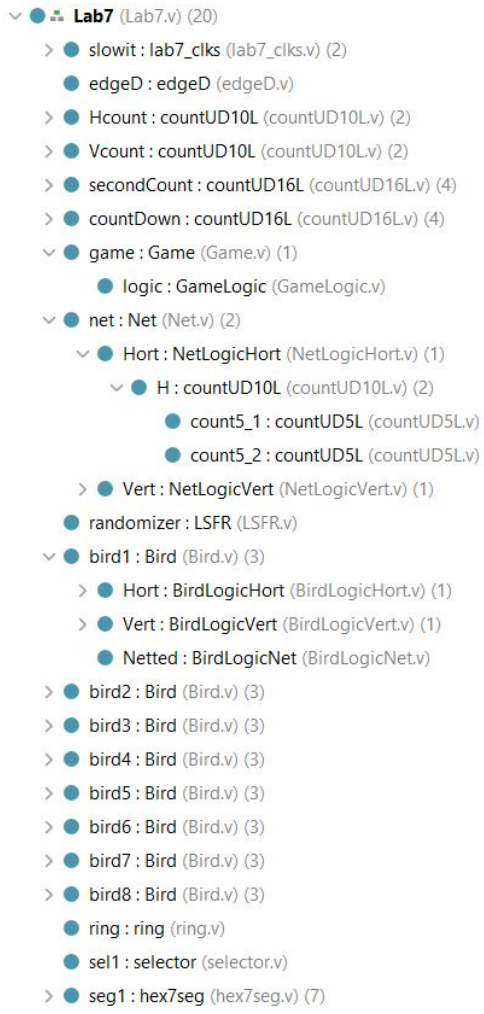
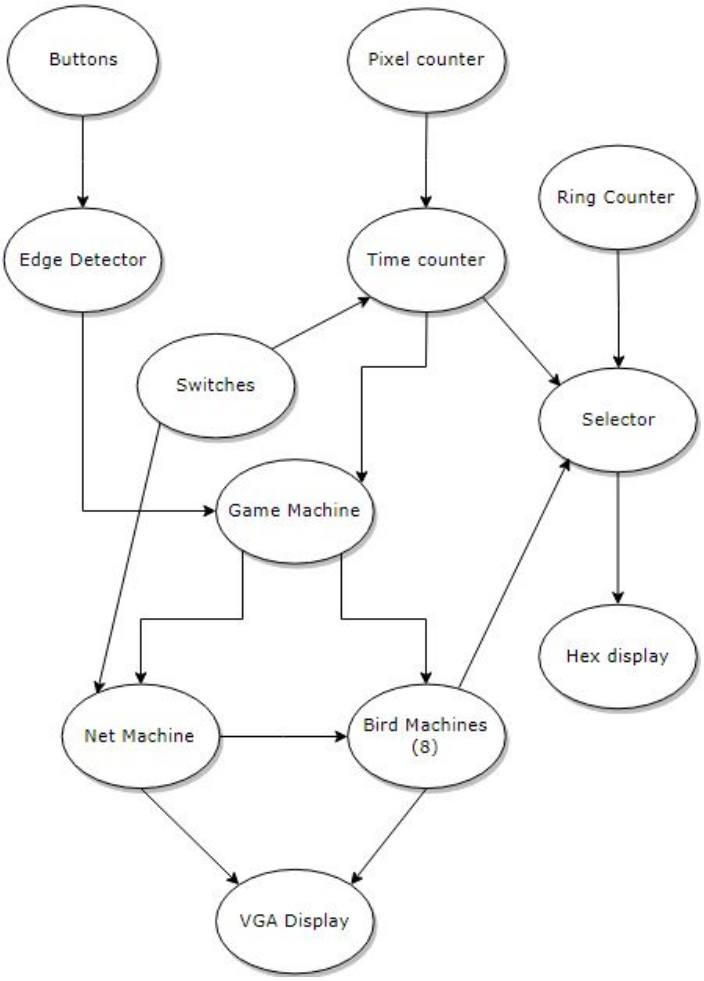


Diagram of top level



Discussion of top level:

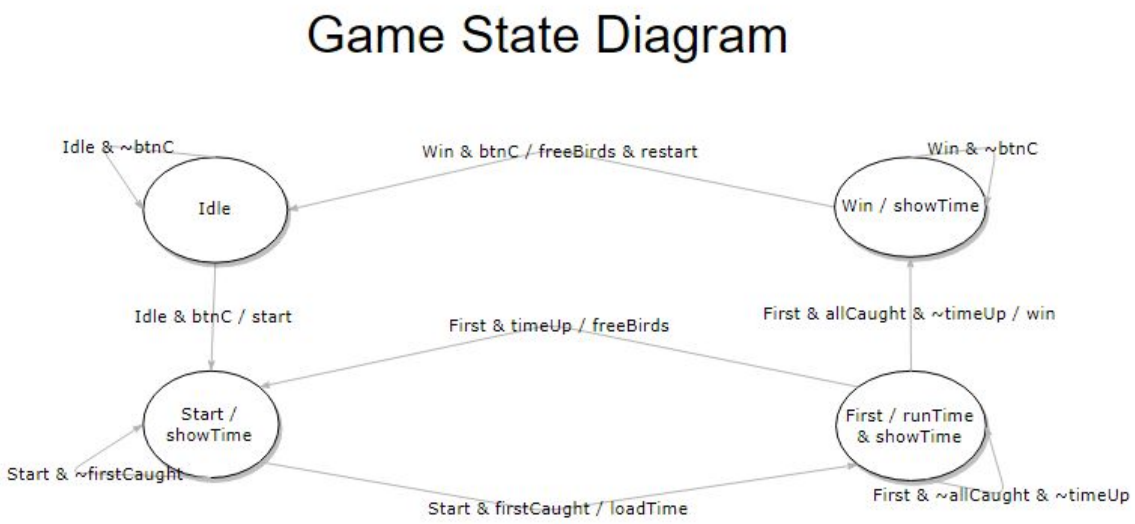
Buttons and switches are inputted to the module. The main button, btnC is edge detected, as it is used to start/restart the game after winning. From the win state, btnC must be pressed

twice to play again, and without the edge detector the game would go from the win state to the next game before we could release the button. The switches are used to control the net's size, the time allotted to catch all the birds, and to reset the entire program. The pixels are counted and displayed by the VGA logic at the end of my top module. The time counter counts every 60 frames and treats that as a second as that is the approximate refresh rate, 60Hz. The time counter counts down using the 60 frames defined as a second, its starting time from the switches. Much of the previously mentioned information is inputted to the game machine, which then controls what the birds and the net are doing. The net's size changes from the switches and its direction can be changed from the buttons. It's location, along with information from the game machine is used by the birds. All the birds and the net's locations are tracked and displayed by the VGA. The ring counter works as in previous labs, quickly cycling through the 4 things to be displayed on the hex display. The selector contains the time and the number of birds caught. This is then picked by the ring counter and displayed by the hex display logic.

Answers to general questions:

To test my design, I used the given simulation and created my own to test my bird. As I got further ahead without too many errors I used the VGA display to see what needed fixing. I chose the given inputs since they were defined by the lab manual. A corner case I considered was if a bird or the net hit the corner of the wall. This was not an issue however, as my machine logic was split into vertical and horizontal machines, so it would bounce off in both directions without any problems. The main issue I had at the start was when I converted my 16 bit counter to a 10 bit counter. I made a mistake when doing so, and really regretted it as it took an hour to find the

error, it had to do with the UTC as both 5 bit counters within the 10 bit counter went up every time it should have only gone up by 1.



The game machine controls the main stages of the game. It gives outputs that the bird state machines and the net state machine take in. It is the main state machine of the 3 types.

Idle: The initial state. For the game to start btnC must be pressed and then it will enter the Start state. All the birds and the net are still and in proper order.

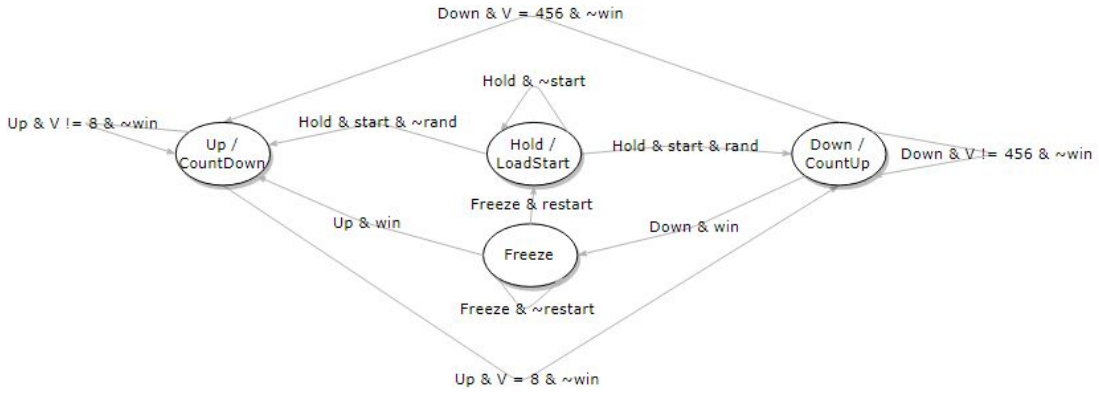
Start: The beginning of the game, everything starts moving and keeps doing so until a bird is caught.

First: The first bird has been caught, and now the timer is loaded from the switches and then begins counting down. If all the birds are caught it goes to the win state, but if time is up first, then it goes back to the start state.

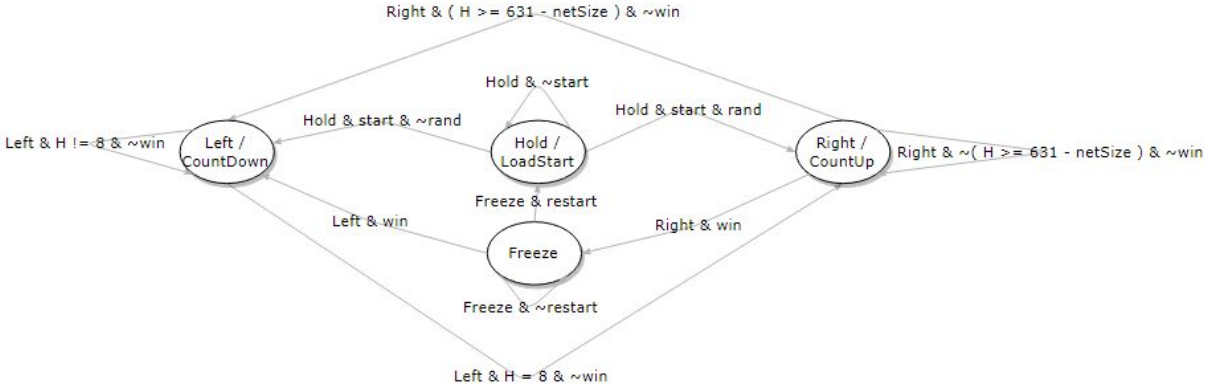
Win: All of the birds have been caught in the allotted time! The timer stops and the net stops moving, showing its victory pattern. Pressing btnC moves on from the victory screen and goes back to the idle state.

Net State Diagrams

Vertical Logic



Horizontal Logic



The net state machine handles the net's movement. It takes in the logic from the game state machine for how to behave. It also outputs its location and size to the bird state machines, and the VGA. The net state machine is the second in command of the three types of state machines as it takes orders from the game machine, but gives orders to the bird machines.

<u>Universal Net States:</u>

Hold: This is the initial state for both logics. This state is held during the game machine's idle state. When the state machine enters the start state, the net then begins moving, now in the Left or Right, and Up or Down state.

Freeze: This state is entered from one of the moving states (up, down, left, or right). It is entered when all of the birds have been caught and the net then freezes and displays its victory pattern.

Once btnC is pressed it leaves this state for the hold state, as the game has been taken back to the idle state.

Vertical Specific States:

Up: The up state is entered if the random output from the shift register causes it to be picked over the down state. The net goes up a single pixel for every frame. If every bird is caught the net then freezes, entering the freeze state. Otherwise if the net hits a wall it will change to the down state. This state can be forcefully entered by pressing btnU.

Down: The down state is entered if the random output from the shift register causes it to be picked over the up state. The net goes down a single pixel for every frame. If every bird is caught the net then freezes, entering the freeze state. Otherwise if the net hits a wall it will change to the up state. This state can be forcefully entered by pressing btnD.

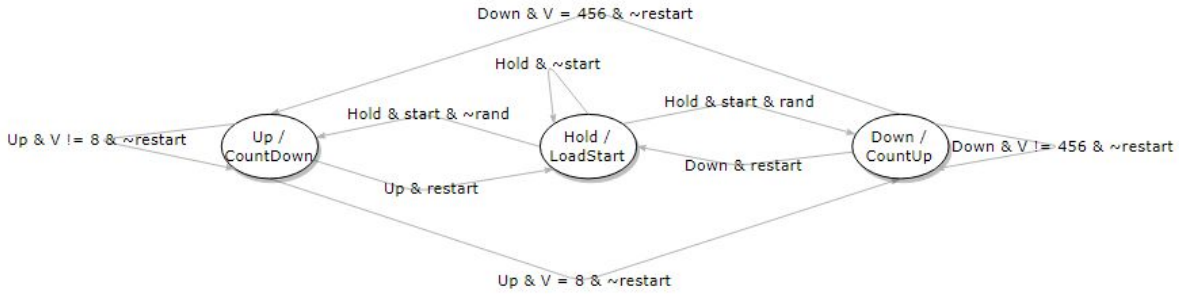
Horizontal Specific States:

Left: The left state is entered if the random output from the shift register causes it to be picked over the right state. The net goes left a single pixel for every frame. If every bird is caught the net then freezes, entering the freeze state. Otherwise if the net hits a wall it will change to the right state. This state can be forcefully entered by pressing btnL.

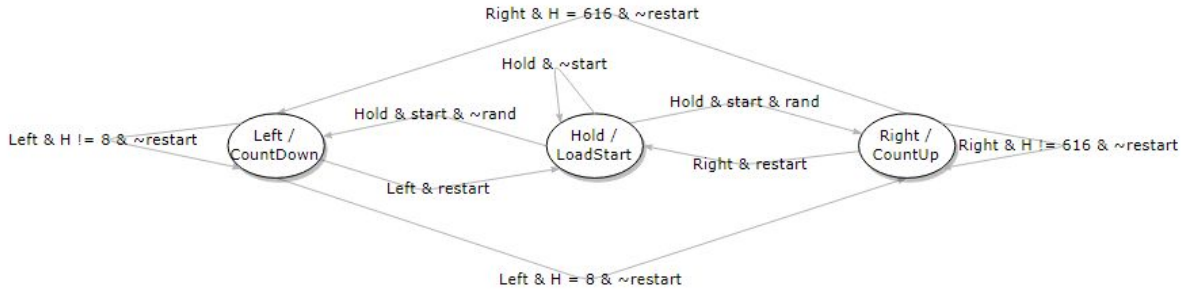
Right: The right state is entered if the random output from the shift register causes it to be picked over the left state. The net goes right a single pixel for every frame. If every bird is caught the net then freezes, entering the freeze state. Otherwise if the net hits a wall it will change to the left state. This state can be forcefully entered by pressing btnR.

Bird State Diagrams

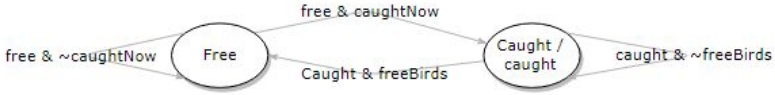
Vertical Logic



Horizontal Logic



Net Logic



The bird state machine handles the bird's movements. It takes in the logic from the game and net state machines for how to behave. It also outputs its location, for the VGA. The bird state machine is the lowest level of the state machines as it only takes orders from other state machines, except for tracking the number of birds caught, which is read by the game machine.

<u>Movement Specific State (for both Horizontal and Vertical machines):</u>

Hold: This is the initial state for both logics. This state is held during the game machine's idle state. When the state machine enters the start state, the net then begins moving, now in the Left or Right, and Up or Down state.

<u>Vertical Specific States:</u>

Up: The up state is entered if the random output from the shift register causes it to be picked over the down state. The bird goes up two pixels for every frame. If the bird hits a wall it will change to the down state.

Down: The down state is entered if the random output from the shift register causes it to be picked over the up state. The bird goes down two pixels for every frame. If the bird hits a wall it will change to the up state.

Horizontal Specific States:

Left: The left state is entered if the random output from the shift register causes it to be picked over the right state. The bird goes left two pixels for every frame. If the bird hits a wall it will change to the right state.

Right: The right state is entered if the random output from the shift register causes it to be picked over the left state. The bird goes right two pixels for every frame. If the bird hits a wall it will change to the left state.

Netted Specific States:

Free: The birds all start in this state. If the net touches the bird, the bird then enters the caught state. The bird is red in this state.

Caught: Once a bird is caught it enters this state. It changes from red to purple. If all the birds are caught then the birds stay caught until the game is restarted and goes back to the idle state. Likewise, if time runs out then the birds are immediately freed.

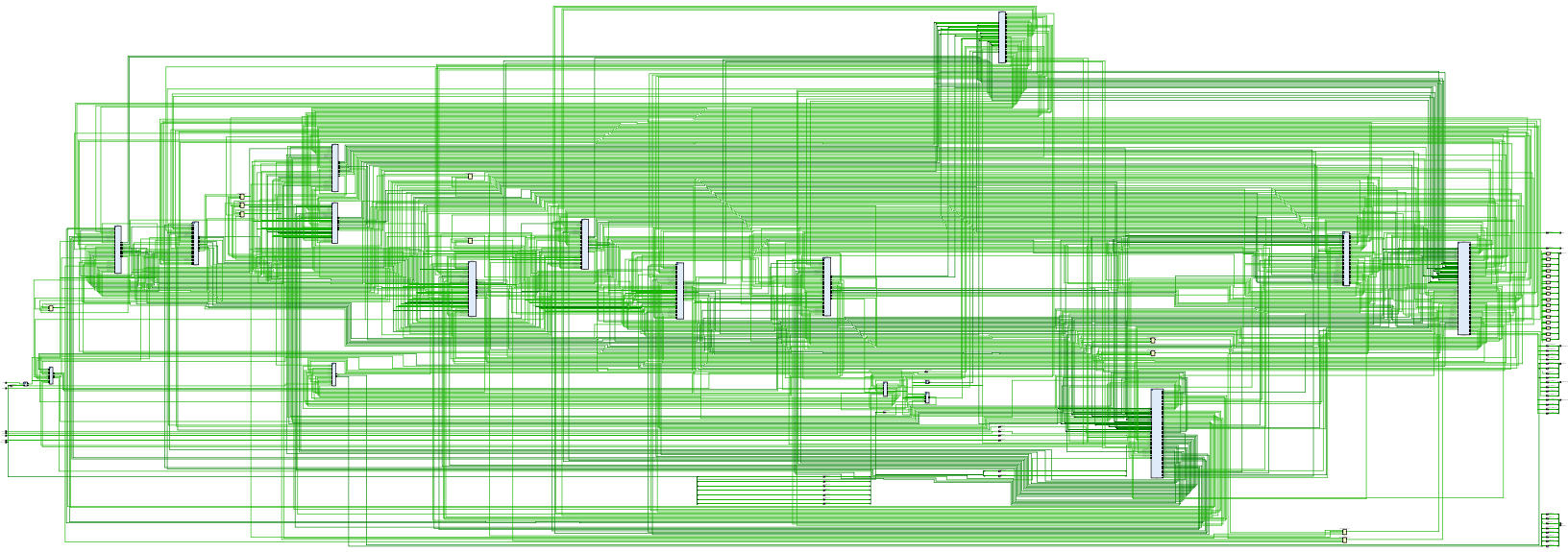
Clock period: 40ns (Frequency is 25MHz)

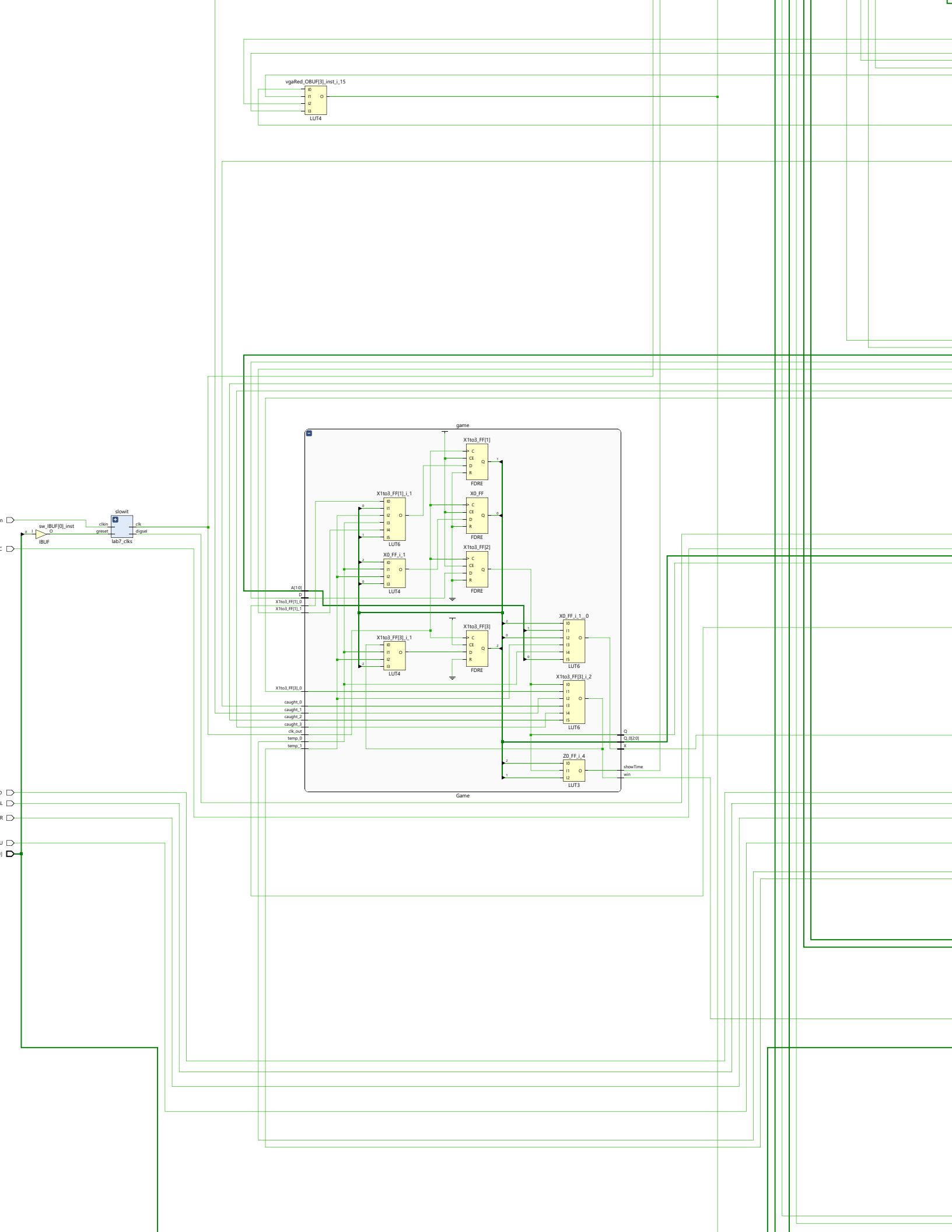
Worst negative slack: 28.915ns

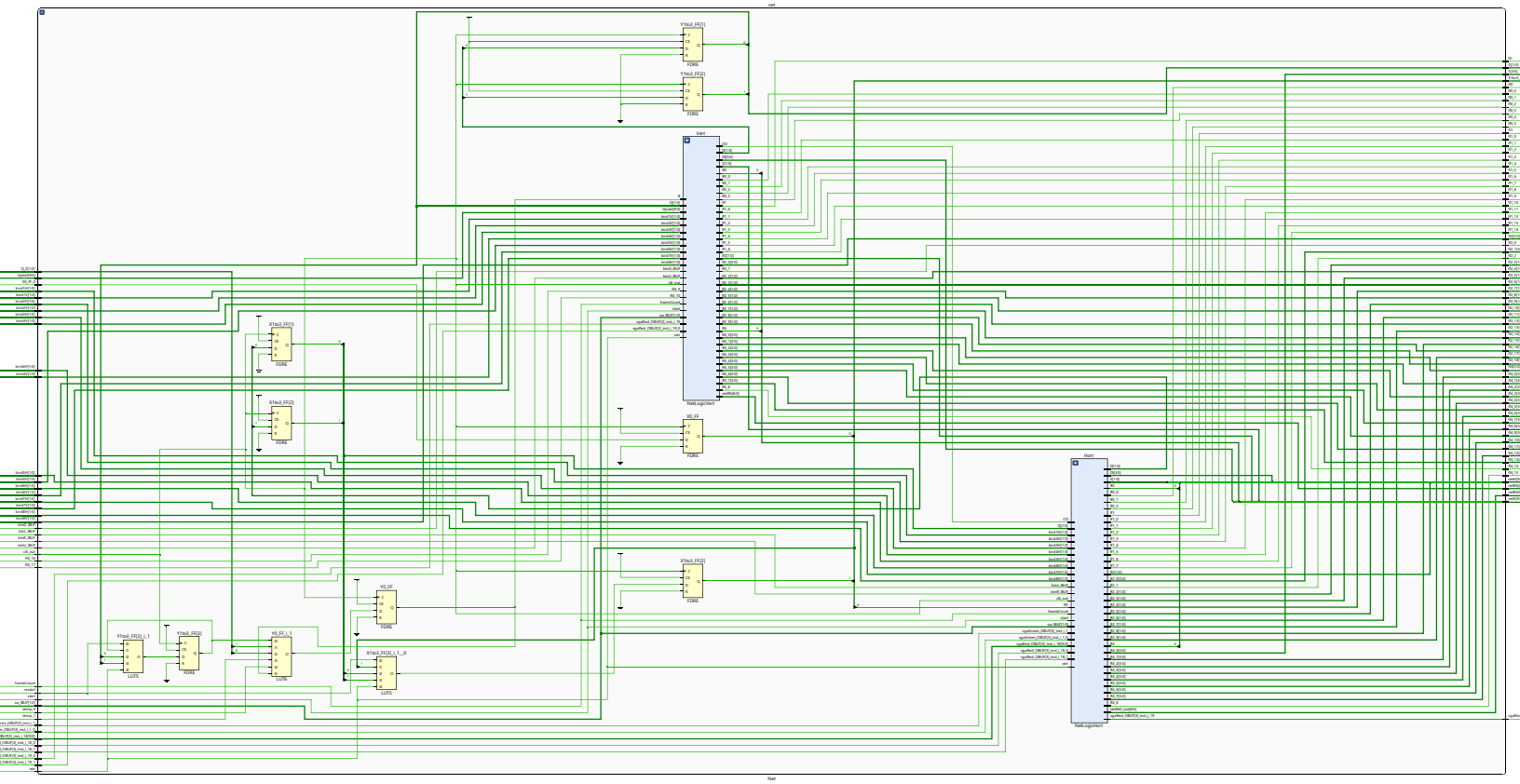
Maximum clock frequency: approximately **11.085ns**

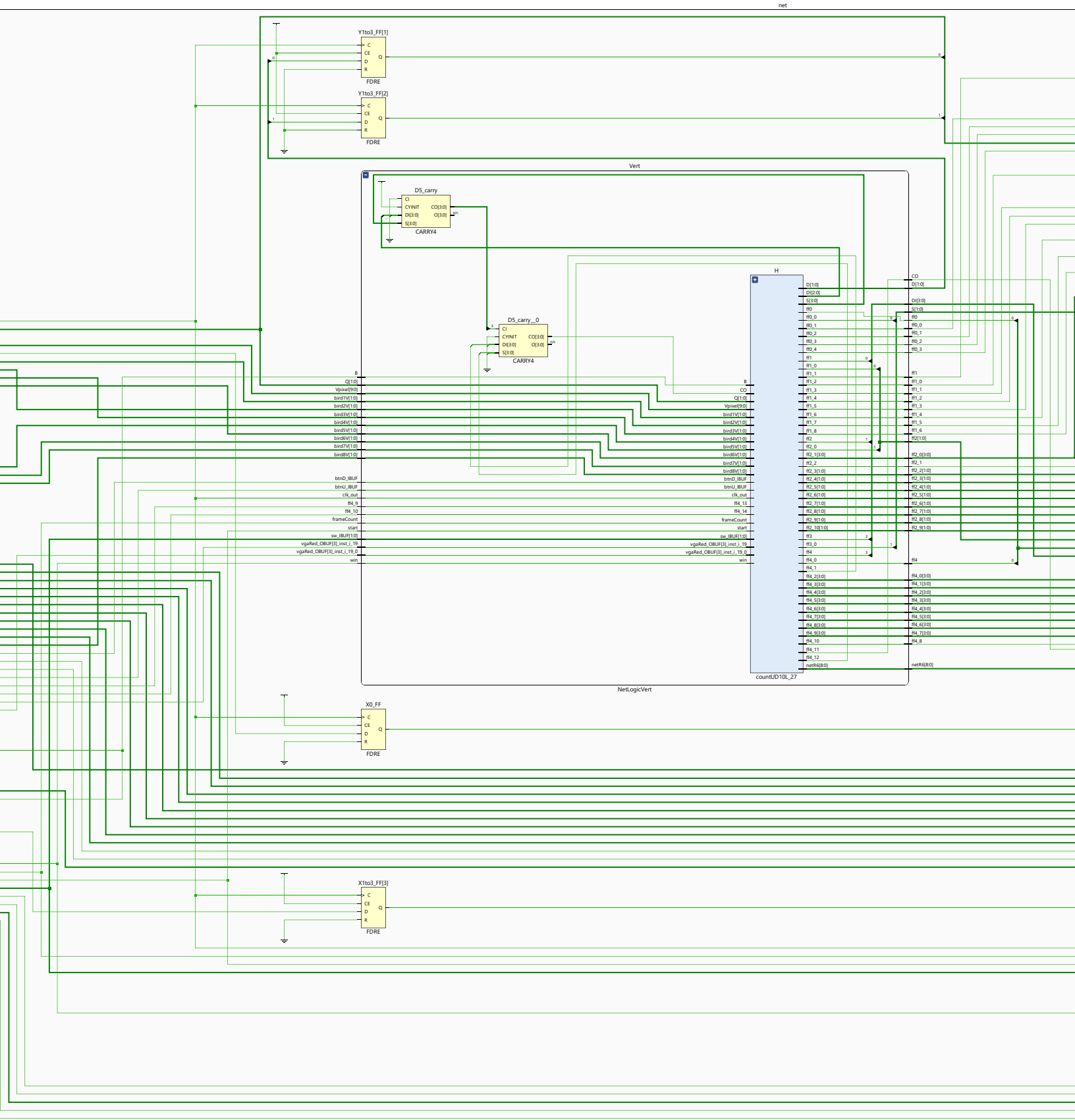
Conclusion:

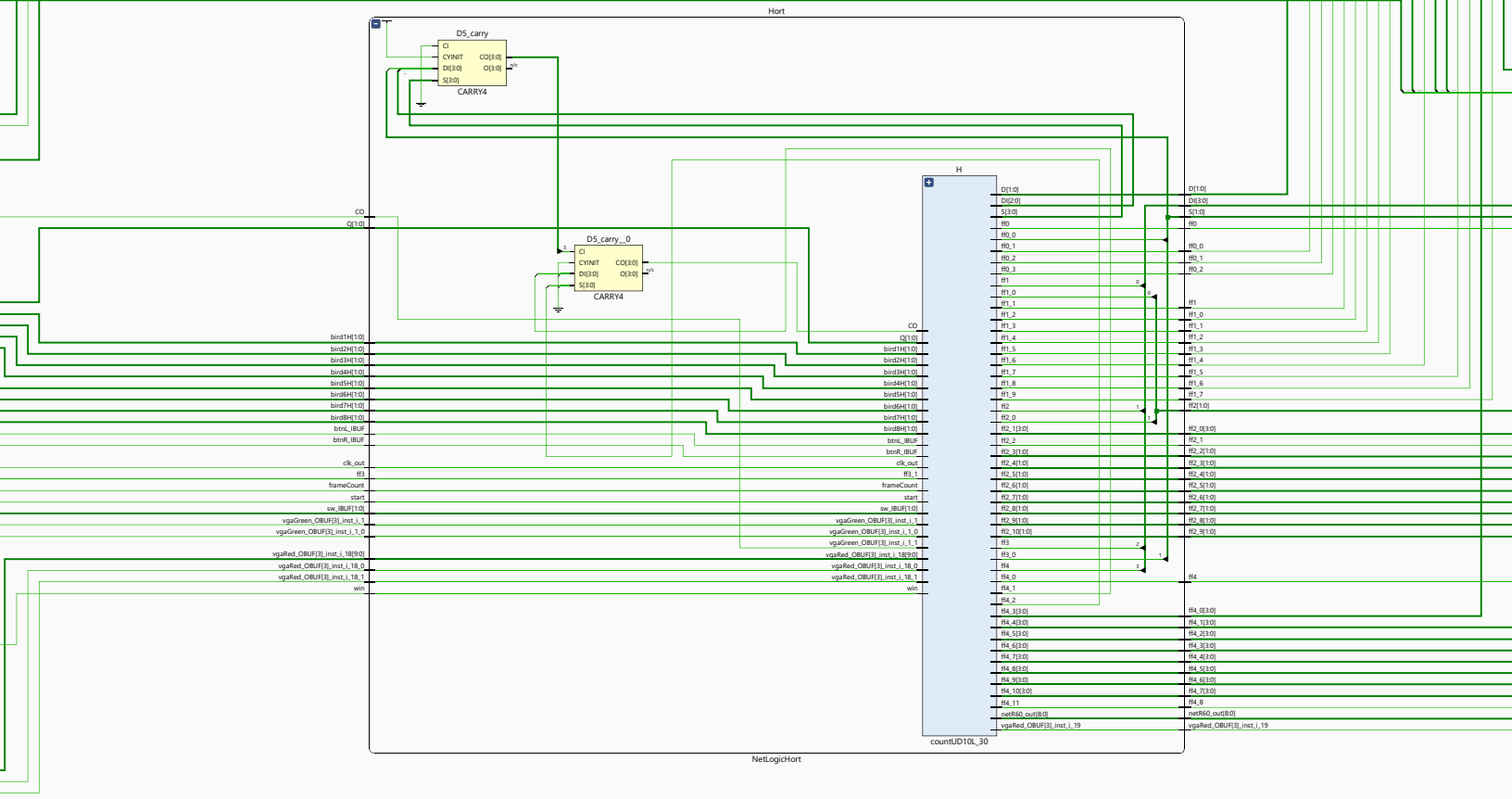
I learned how to use the VGA! This was super interesting to learn, even though they aren't practical any more. I had trouble making so many state machines and making them all work together, like having trouble with the net displaying correctly and counting all the caught birds correctly. If I did the lab again I would have kept my components from previous labs the exact same, and would have just used them as is to make this lab work. I would like to optimize my counters and such so that they do not carry more bits than needed, but as previously described this only hurt me. I would also probably put the VGA logic in its own module, but while writing the lab it was very nice having it on the top module and having to create its own module and have tons of inputs would be a pain.

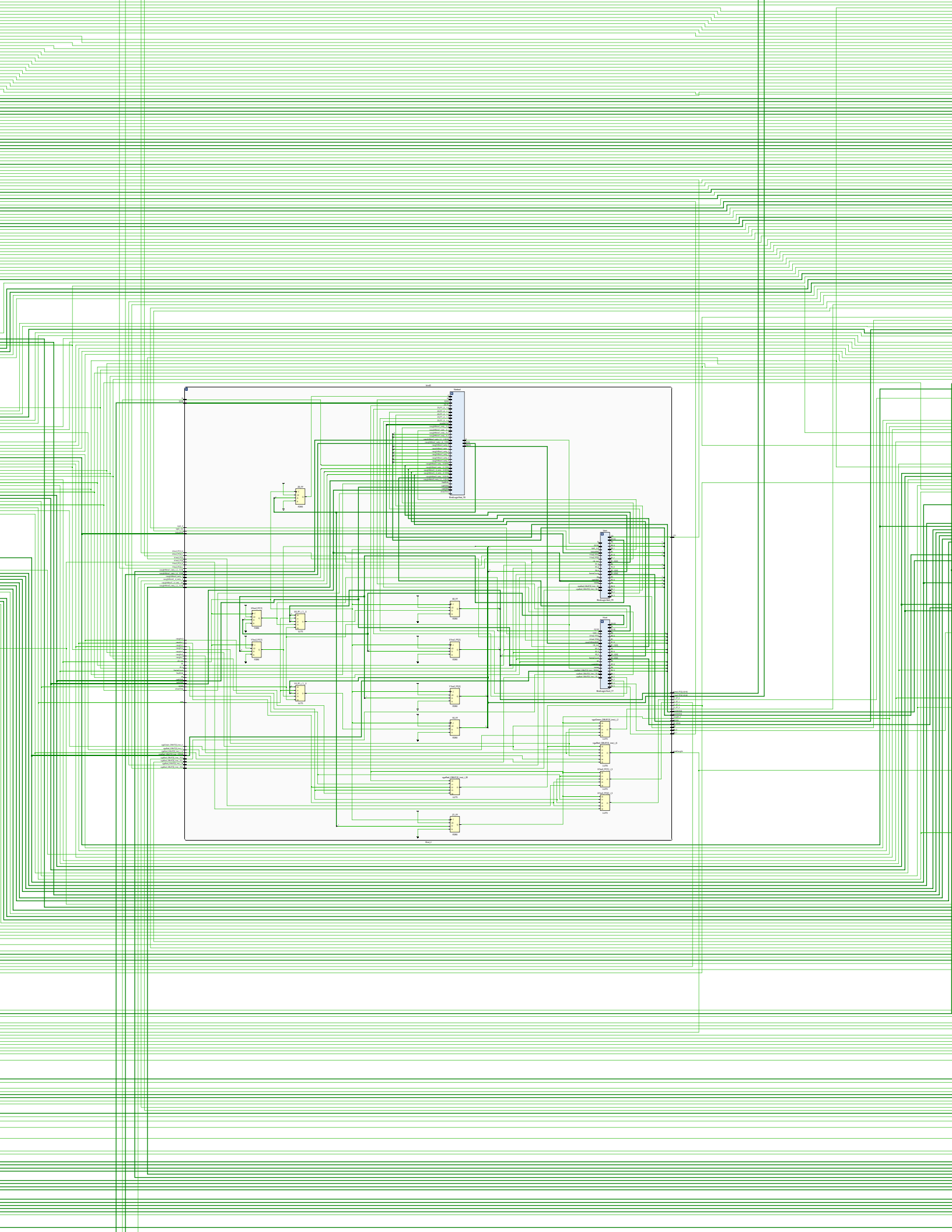


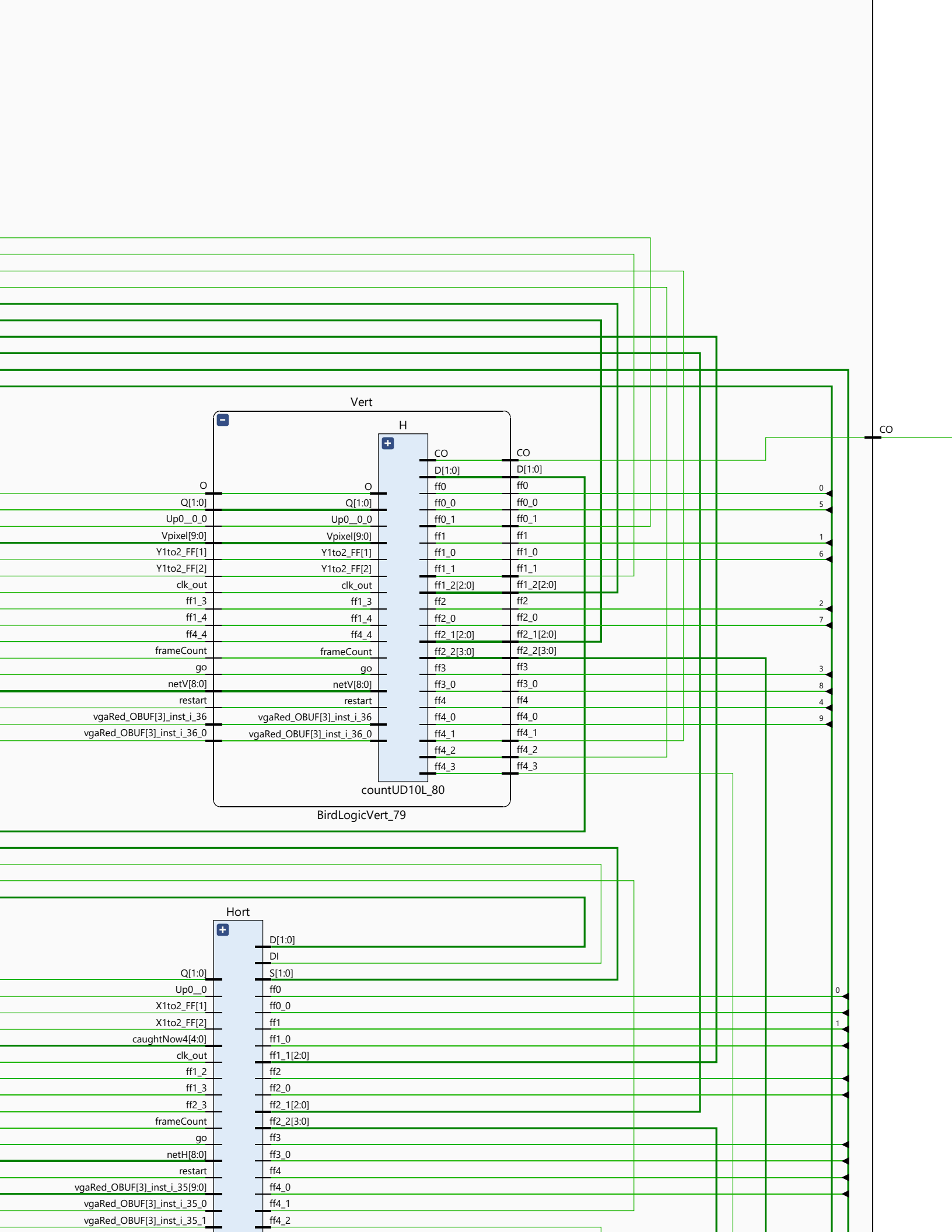

vgaRed_OBUF[3]_inst_i_15
LUT4
game
X1to3_FF[1]
FDRE
X1to3_FF[1]_i_1
LUT6
X0_FF
FDRE
X0_FF_i_1
LUT4
X1to3_FF[2]
FDRE
A[1:0]
X1to3_FF[1]_0
X1to3_FF[1]_1
X1to3_FF[3]
FDRE
X1to3_FF[3]_i_1
LUT4
X0_FF_i_1__0
LUT6
X1to3_FF[3]_i_2
LUT6
X1to3_FF[3]_0
caught_0
caught_1
caught_2
caught_3
clk_out
temp_0
temp_1
Q
Q_0[2:0]
X
Z0_FF_i_4
LUT3
showTime
win
Game
slowit
clkin
greset
clk
digsel
lab7_clks
sw_IBUF[0]_inst
IBUF

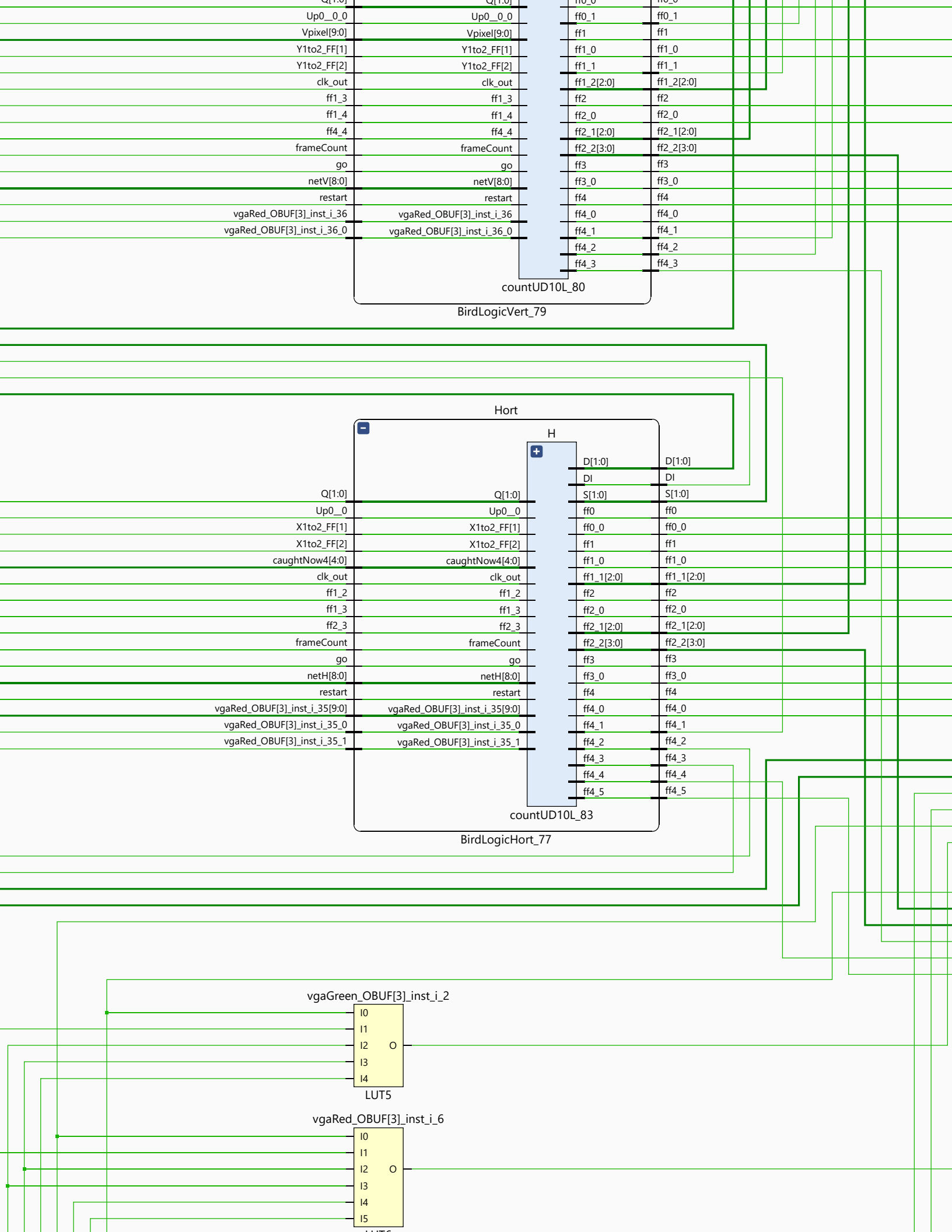

net
Vsync
Hsync
Net

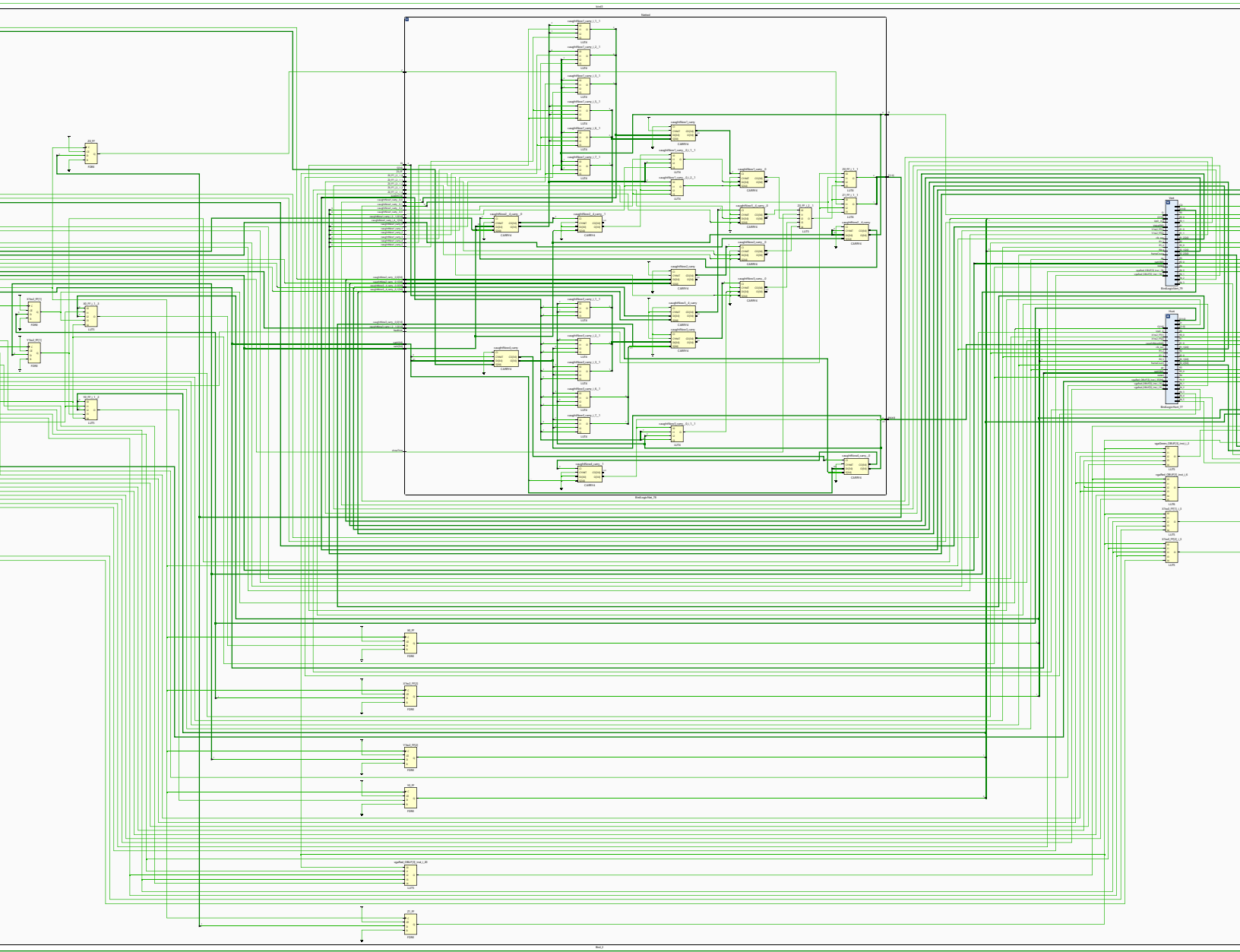

net
Y1to3_FF[1]
FDRE
Y1to3_FF[2]
FDRE
Vert
D5_carry
CARRY4
D5_carry__0
CARRY4
H
countUD10L_27
NetLogicVert
X0_FF
FDRE
X1to3_FF[3]
FDRE


Hort
D5_carry
CI
CYINIT
DI[3:0]
S[3:0]
CO[3:0]
O[3:0]
CARRY4
D5_carry__0
CARRY4
H
countUD10L_30
NetLogicHort
CO
O[1:0]
bird1H[1:0]
bird2H[1:0]
bird3H[1:0]
bird4H[1:0]
bird5H[1:0]
bird6H[1:0]
bird7H[1:0]
bird8H[1:0]
btnL_IBUF
btnR_IBUF
clk_out
ff3
frameCount
start
sw_IBUF[1:0]
vgaGreen_OBUF[3]_inst_i_1
vgaGreen_OBUF[3]_inst_i_1_0
vgaRed_OBUF[3]_inst_i_18[9:0]
vgaRed_OBUF[3]_inst_i_18_0
vgaRed_OBUF[3]_inst_i_18_1
win
D[1:0]
DI[2:0]
S[3:0]
ff0
ff0_0
ff0_1
ff0_2
ff0_3
ff1
ff1_0
ff1_1
ff1_2
ff1_3
ff1_4
ff1_5
ff1_6
ff1_7
ff1_8
ff1_9
ff2
ff2_0
ff2_1[3:0]
ff2_2
ff2_3[1:0]
ff2_4[1:0]
ff2_5[1:0]
ff2_6[1:0]
ff2_7[1:0]
ff2_8[1:0]
ff2_9[1:0]
ff2_10[1:0]
ff3
ff3_0
ff4
ff4_0
ff4_1
ff4_2
ff4_3[3:0]
ff4_4[3:0]
ff4_5[3:0]
ff4_6[3:0]
ff4_7[3:0]
ff4_8[3:0]
ff4_9[3:0]
ff4_10[3:0]
ff4_11
netR60_out[8:0]
vgaRed_OBUF[3]_inst_i_19
DI[3:0]
S[1:0]
ff0
ff0_0
ff0_1
ff0_2
ff1
ff1_0
ff1_1
ff1_2
ff1_3
ff1_4
ff1_5
ff1_6
ff1_7
ff2[1:0]
ff2_0[3:0]
ff2_1
ff2_2[1:0]
ff2_3[1:0]
ff2_4[1:0]
ff2_5[1:0]
ff2_6[1:0]
ff2_7[1:0]
ff2_8[1:0]
ff2_9[1:0]
ff4
ff4_0[3:0]
ff4_1[3:0]
ff4_2[3:0]
ff4_3[3:0]
ff4_4[3:0]
ff4_5[3:0]
ff4_6[3:0]
ff4_7[3:0]
ff4_8
netR60_out[8:0]
vgaRed_OBUF[3]_inst_i_19




Vert
H
CO
D[1:0]
O
Q[1:0]
Up0__0_0
Vpixel[9:0]
Y1to2_FF[1]
Y1to2_FF[2]
clk_out
ff1_3
ff1_4
ff4_4
frameCount
go
netV[8:0]
restart
vgaRed_OBUF[3]_inst_i_36
vgaRed_OBUF[3]_inst_i_36_0
ff0
ff0_0
ff0_1
ff1
ff1_0
ff1_1
ff1_2[2:0]
ff2
ff2_0
ff2_1[2:0]
ff2_2[3:0]
ff3
ff3_0
ff4
ff4_0
ff4_1
ff4_2
ff4_3
countUD10L_80
BirdLogicVert_79
CO
0
5
1
6
2
7
3
8
4
9
Hort
D[1:0]
DI
S[1:0]
Q[1:0]
Up0__0
X1to2_FF[1]
X1to2_FF[2]
caughtNow4[4:0]
clk_out
ff1_2
ff1_3
ff2_3
frameCount
go
netH[8:0]
restart
vgaRed_OBUF[3]_inst_i_35[9:0]
vgaRed_OBUF[3]_inst_i_35_0
vgaRed_OBUF[3]_inst_i_35_1
ff0
ff0_0
ff1
ff1_0
ff1_1[2:0]
ff2
ff2_0
ff2_1[2:0]
ff2_2[3:0]
ff3
ff3_0
ff4
ff4_0
ff4_1
ff4_2
0
1


Up0__0_0
Vpixel[9:0]
Y1to2_FF[1]
Y1to2_FF[2]
clk_out
ff1_3
ff1_4
ff4_4
frameCount
go
netV[8:0]
restart
vgaRed_OBUF[3]_inst_i_36
vgaRed_OBUF[3]_inst_i_36_0
ff0_1
ff1
ff1_0
ff1_1
ff1_2[2:0]
ff2
ff2_0
ff2_1[2:0]
ff2_2[3:0]
ff3
ff3_0
ff4
ff4_0
ff4_1
ff4_2
ff4_3
countUD10L_80
BirdLogicVert_79
Hort
H
Q[1:0]
Up0__0
X1to2_FF[1]
X1to2_FF[2]
caughtNow4[4:0]
clk_out
ff1_2
ff1_3
ff2_3
frameCount
go
netH[8:0]
restart
vgaRed_OBUF[3]_inst_i_35[9:0]
vgaRed_OBUF[3]_inst_i_35_0
vgaRed_OBUF[3]_inst_i_35_1
D[1:0]
DI
S[1:0]
ff0
ff0_0
ff1
ff1_0
ff1_1[2:0]
ff2
ff2_0
ff2_1[2:0]
ff2_2[3:0]
ff3
ff3_0
ff4
ff4_0
ff4_1
ff4_2
ff4_3
ff4_4
ff4_5
countUD10L_83
BirdLogicHort_77
vgaGreen_OBUF[3]_inst_i_2
I0
I1
I2
I3
I4
O
LUT5
vgaRed_OBUF[3]_inst_i_6
I5



```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 11/12/2019 01:48:39 PM
// Design Name:
// Module Name: Lab6
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module Lab7(
    input clkin,
    input btnL, // Left
    input btnR, // Right
    input btnU, // Up
    input btnD, // Down
    input BTNC, // game start (diff name since it must
```

```
be edge detected!)
    input [15:0] sw, // time to catch birds

    output [15:0] led,
    output [3:0] an,
    output dp,
    output [6:0] seg,

    output Hsync,
    output Vsync,
    output [3:0] vgaRed,
    output [3:0] vgaBlue,
    output [3:0] vgaGreen
    );

    assign dp = 1'b1;

    wire left, right;

    wire clk, digsel;

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

    wire btnC; // edge detected

    edgeD edgeD(.clk(clk), .btn(BTNC), .o(btnC) );

    FDRE #(.INIT(1'b0) ) ff1 ( .C(clk), .R(1'b0),
.CE(1'b1), .D(btnL), .Q(left) ); // synchronize the
buttons
```

```
    FDRE #(.INIT(1'b0) ) ff2 ( .C(clk), .R(1'b0),
.CE(1'b1), .D(btnR), .Q(right) );

    wire [9:0] Hpixel, Vpixel;

    countUD10L Hcount( .clk(clk), .Up(1'b1),
.Dw(1'b0), .LD(Hpixel == 10'd799),
.Din(10'b0000000000), .Q(Hpixel) );
    countUD10L Vcount( .clk(clk), .Up(Hpixel ==
10'd799), .Dw(1'b0), .LD((Hpixel == 10'd799) & (Vpixel
== 10'd524)), .Din(10'b0000000000), .Q(Vpixel) );

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

    wire active = (Hpixel <= 10'd639) & (Vpixel <=
10'd479); // defines active region

    wire frameCount = (Hpixel == 10'd799) & (Vpixel ==
10'd524); // gives a 1 once per frame
    wire extraCount = (Hpixel == 10'd798) & (Vpixel ==
10'd523);

    wire go = frameCount | extraCount; // causes a
count twice per frame

    wire [7:0] useless1, useless2, frames, timer; //
frames counts frames, timer counts down from set time
```

```
    wire timeUp, showTime, startTime, loadTime,
freeBirds, win, start, runTime, restart;

    countUD16L secondCount( .clk(clk),
.Up(frameCount), .Dw(1'b0), .LD(frames == 16'd60),
.Din( 16'h0000 ), .Q( {useless1, frames} ) );

    countUD16L countDown( .clk(clk), .Up(1'b0), .Dw(
(frames == 8'd60 ) & runTime), .LD(loadTime | restart
), .Din( { 8'b00000000, sw[11:4] } & {16{loadTime}} ),
.Q( {useless2, timer} ) );

    assign timeUp = runTime & ( timer == 8'h00 );

    wire [7:0] caught; // caught = # birds caught

    Game game(.clk(clk), .btnC(btnC), .caught(caught),
.win(win), .timeUp(timeUp), .showTime(showTime),
.runTime(runTime), .start(start), .loadTime(loadTime),
.restart(restart), .startTime(startTime),
.freeBirds(freeBirds));

    wire [9:0] netH, netV, netSize; // netSize same
size to avoid math errors
    assign netSize = { 2'b00, sw[15], sw[14],
6'b011111}; // normally 32 long, 1 + 31. increased by
switches

    Net net( .clk(clk), .start(start),
.loadTime(loadTime), .win(win), .restart(restart),
.btnL(btnL), .btnR(btnR), .btnU(btnU), .btnD(btnD),
```

```
.netSize(netSize), .go(frameCount), .startH(10'd310 -
(netSize/10'd2) ), .startV(10'd255 - (netSize/10'd2)
), .nextH(netH), .nextV(netV) /*.netLook(netLook)*/ );

    wire [7:0] rand;

    LSFR randomizer(.clk(clk), .rnd(rand));

    wire [15:0] longRand = { ~rand, rand}; // every
bird gets a random bit for up/down and all cannot be
the same

    wire [9:0] bird1H, bird1V, bird2H, bird2V, bird3H,
bird3V, bird4H, bird4V, bird5H, bird5V, bird6H,
bird6V, bird7H, bird7V, bird8H, bird8V;

    Bird bird1( .clk(clk), .showTime(showTime),
.start(start), .go(go), .restart(restart),
.freeBirds(freeBirds), .rand(longRand[1:0]),
.netH(netH), .netV(netV), .netSize(netSize),
.startH(10'd100), .startV(10'd75), .nextH(bird1H),
.nextV(bird1V), .caught(caught[0]) );
    Bird bird2( .clk(clk), .showTime(showTime),
.start(start), .go(go), .restart(restart),
.freeBirds(freeBirds), .rand(longRand[3:2]),
.netH(netH), .netV(netV), .netSize(netSize),
.startH(10'd300), .startV(10'd75), .nextH(bird2H),
.nextV(bird2V), .caught(caught[1]) );
    Bird bird3( .clk(clk), .showTime(showTime),
.start(start), .go(go), .restart(restart),
.freeBirds(freeBirds), .rand(longRand[5:4]),
```

```
.netH(netH), .netV(netV), .netSize(netSize),
.startH(10'd500), .startV(10'd75), .nextH(bird3H),
.nextV(bird3V), .caught(caught[2]) );
    Bird bird4( .clk(clk), .showTime(showTime),
.start(start), .go(go), .restart(restart),
.freeBirds(freeBirds), .rand(longRand[7:6]),
.netH(netH), .netV(netV), .netSize(netSize),
.startH(10'd100), .startV(10'd250), .nextH(bird4H),
.nextV(bird4V), .caught(caught[3]) );
    Bird bird5( .clk(clk), .showTime(showTime),
.start(start), .go(go), .restart(restart),
.freeBirds(freeBirds), .rand(longRand[9:8]),
.netH(netH), .netV(netV), .netSize(netSize),
.startH(10'd500), .startV(10'd250), .nextH(bird5H),
.nextV(bird5V), .caught(caught[4]) );
    Bird bird6( .clk(clk), .showTime(showTime),
.start(start), .go(go), .restart(restart),
.freeBirds(freeBirds), .rand(longRand[11:10]),
.netH(netH), .netV(netV), .netSize(netSize),
.startH(10'd100), .startV(10'd400), .nextH(bird6H),
.nextV(bird6V), .caught(caught[5]) );
    Bird bird7( .clk(clk), .showTime(showTime),
.start(start), .go(go), .restart(restart),
.freeBirds(freeBirds), .rand(longRand[13:12]),
.netH(netH), .netV(netV), .netSize(netSize),
.startH(10'd300), .startV(10'd400), .nextH(bird7H),
.nextV(bird7V), .caught(caught[6]) );
    Bird bird8( .clk(clk), .showTime(showTime),
.start(start), .go(go), .restart(restart),
.freeBirds(freeBirds), .rand(longRand[15:14]),
.netH(netH), .netV(netV), .netSize(netSize),
```

```
.startH(10'd500), .startV(10'd400), .nextH(bird8H),
.nextV(bird8V), .caught(caught[7]) );

    wire [3:0] wall, netG, netB, netR, bird1R, bird1B,
bird2R, bird2B, bird3R, bird3B, bird4R, bird4B;
    wire [3:0] bird5R, bird5B, bird6R, bird6B, bird7R,
bird7B, bird8R, bird8B;

    assign wall = {4{ ( ( ( (Hpixel <= 10'd7) | (
(Hpixel >= 10'd632) & (Hpixel <= 10'd639) ) ) | (
(Vpixel <= 10'd7) | ( (Vpixel >= 10'd472) & (Vpixel <=
10'd479) ) ) ) & active)}};
    assign netG = {4{ ( ( (Hpixel >= netH + 10'd7) & (
Hpixel <= ( netH + netSize - 10'd7) ) ) & ( ( Vpixel
>= netV + 10'd7) & ( Vpixel <= ( netV + netSize -
10'd7) ) ) ) & active & |caught }};
    assign netB = {4{ ( ( (Hpixel >= netH) & ( Hpixel
<= ( netH + netSize ) ) ) & ( ( Vpixel >= netV ) & (
Vpixel <= ( netV + netSize ) ) ) ) & active }};
    assign netR = {4{ ( ( (Hpixel >= netH + 10'd7) & (
Hpixel <= ( netH + netSize - 10'd7) ) ) & ( ( Vpixel
>= netV + 10'd7) & ( Vpixel <= ( netV + netSize -
10'd7) ) ) ) & active & &caught }};
    assign bird1R = {4{ ( ( (Hpixel >= bird1H) & (
Hpixel <= ( bird1H + 10'd15 ) ) ) & ( ( Vpixel >=
bird1V ) & ( Vpixel <= ( bird1V + 10'd15 ) ) ) ) &
active }};
    assign bird1B = bird1R & {4{ caught[0] }};
    assign bird2R = {4{ ( ( (Hpixel >= bird2H) & (
Hpixel <= ( bird2H + 10'd15 ) ) ) & ( ( Vpixel >=
bird2V ) & ( Vpixel <= ( bird2V + 10'd15 ) ) ) ) &
```

```
active }};
    assign bird2B = bird2R & {4{ caught[1] }};
    assign bird3R = {4{ ( ( (Hpixel >= bird3H) & (
Hpixel <= ( bird3H + 10'd15 ) ) ) & ( ( Vpixel >=
bird3V ) & ( Vpixel <= ( bird3V + 10'd15 ) ) ) ) &
active }};
    assign bird3B = bird3R & {4{ caught[2] }};
    assign bird4R = {4{ ( ( (Hpixel >= bird4H) & (
Hpixel <= ( bird4H + 10'd15 ) ) ) & ( ( Vpixel >=
bird4V ) & ( Vpixel <= ( bird4V + 10'd15 ) ) ) ) &
active }};
    assign bird4B = bird4R & {4{ caught[3] }};
    assign bird5R = {4{ ( ( (Hpixel >= bird5H) & (
Hpixel <= ( bird5H + 10'd15 ) ) ) & ( ( Vpixel >=
bird5V ) & ( Vpixel <= ( bird5V + 10'd15 ) ) ) ) &
active }};
    assign bird5B = bird5R & {4{ caught[4] }};
    assign bird6R = {4{ ( ( (Hpixel >= bird6H) & (
Hpixel <= ( bird6H + 10'd15 ) ) ) & ( ( Vpixel >=
bird6V ) & ( Vpixel <= ( bird6V + 10'd15 ) ) ) ) &
active }};
    assign bird6B = bird6R & {4{ caught[5] }};
    assign bird7R = {4{ ( ( (Hpixel >= bird7H) & (
Hpixel <= ( bird7H + 10'd15 ) ) ) & ( ( Vpixel >=
bird7V ) & ( Vpixel <= ( bird7V + 10'd15 ) ) ) ) &
active }};
    assign bird7B = bird7R & {4{ caught[6] }};
    assign bird8R = {4{ ( ( (Hpixel >= bird8H) & (
Hpixel <= ( bird8H + 10'd15 ) ) ) & ( ( Vpixel >=
bird8V ) & ( Vpixel <= ( bird8V + 10'd15 ) ) ) ) &
active }};
```

```
    assign bird8B = bird8R & {4{ caught[7] }};

    assign vgaGreen = netG | wall;
    assign vgaBlue = netB | bird1B | bird2B | bird3B |
bird4B | bird5B | bird6B | bird7B | bird8B;
    assign vgaRed = netR | bird1R| bird2R | bird3R |
bird4R | bird5R | bird6R | bird7R | bird8R;

    wire [15:0] Q; // Q is the bits that are going to
be used to select and then show
    wire [3:0] sel, n; // sel is selector 4 bits, n is
chosen 4 bits of Q

    wire [3:0] totalCaught = 4'h0 + caught[7] +
caught[6] + caught[5] + caught[4] + caught[3] +
caught[2] + caught[1] + caught[0];

    assign Q = { totalCaught, 4'h0, timer };

    ring ring( .digsel(digsel), .clk(clk), .out(sel) );

    selector sel1( .sel(sel), .N(Q), .H(n) );

    assign an[3] = ~sel[3]; // birds caught
    assign an[2] = 1'b1; // not needed
    assign an[1:0] = ~sel[1:0]; // time

    hex7seg seg1( .n(n), .e(1'b1), .seg(seg) );

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 11/19/2019 04:25:41 PM
// Design Name:
// Module Name: Bird
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module Game(
    input clk,
    input btnC,
    input [7:0] caught,
    input timeUp,

    output win,
```

```
    output start,
    output showTime,
    output loadTime,
    output startTime,
    output freeBirds,
    output restart,
    output runTime
    );

    wire [3:0] Q, D; // current/next state of game
logic

    GameLogic logic( .clk(clk), .btnC(btnC),
.caught(caught), .win(win), .Q(Q), .timeUp(timeUp),
.showTime(showTime), .runTime(runTime), .start(start),
.loadTime(loadTime), .restart(restart),
.startTime(startTime), .freeBirds(freeBirds), .D(D) );

    FDRE #(.INIT(1'b1)) X0_FF (.C(clk), .CE(1'b1),
.D(D[0]), .Q(Q[0])); // greset gives 0001 (start in
idle state)
    FDRE #(.INIT(1'b0)) X1to3_FF[3:1] (.C({3{clk}}),
.CE({3{1'b1}}), .D(D[3:1]), .Q(Q[3:1]));

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 11/05/2019 04:57:33 PM
// Design Name:
// Module Name: StateLogic
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module GameLogic(
    input clk,
    input btnC,
    input [3:0] Q,
    input [7:0] caught,
    input timeUp,
```

```
    output win,
    output start,
    output showTime,
    output loadTime,
    output startTime,
    output freeBirds,
    output restart,
    output runTime,
    output [3:0] D
    );

    wire firstCaught = |caught;
    wire allCaught = &caught;

    // NEXT STATE
    assign D[3] = ( Q[2] & allCaught ) | ( Q[3] &
~btnC ); // win
    assign D[2] = ( Q[1] & firstCaught ) | ( Q[2] &
~timeUp & ~allCaught ); // first
    assign D[1] = ( Q[0] & btnC ) | ( Q[1] &
~firstCaught ) | ( Q[2] & timeUp ); // start
    assign D[0] = ( Q[0] & ~btnC ) | ( Q[3] & btnC );
// idle

    // OUTPUTS
    assign win = ( Q[2] & allCaught ); // doubles as
our output to stop time, freezes net
    assign showTime = |Q[3:1]; // shows time on disp
    assign runTime = Q[2]; // count down
    assign loadTime = ( Q[1] & firstCaught ); // reads
in the time from switches
```

```
    assign start = ( Q[0] & btnC ); // everything
starts moving
    assign freeBirds = ( Q[2] & timeUp ) | ( Q[3] &
btnC ); // when birds must be uncaught, doubles as
restartTime
    assign restart = ( Q[3] & btnC ); // moves
everything back to start and freezes

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
////////////////////////////
// Company:
// Engineer:
//
// Create Date: 11/19/2019 04:25:41 PM
// Design Name:
// Module Name: Bird
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
////////////////////////////


module Net(
    input clk,
    input go,
    input start,
    input btnL,
    input btnR,
    input btnU,
```

```
    input btnD,
    input [9:0] netSize,
    input [9:0] startH,
    input [9:0] startV,
    input win,
    input restart,
    input loadTime,

    output [9:0] nextH,
    output [9:0] nextV
    );

    wire [3:0] A, B; // current states of H/V
    wire [3:0] X, Y; // next states of H/V

    NetLogicHort Hort( .clk(clk), .go(go),
.netSize(netSize), .restart(restart), .btnL(btnL),
.btnR(btnR), .start(start), .win(win), .Q(A),
.startH(startH), .nextH(nextH), .D(X) );
    NetLogicVert Vert( .clk(clk), .go(go),
.netSize(netSize), .restart(restart), .btnU(btnU),
.btnD(btnD), .Q(B), .start(start), .win(win),
.startV(startV), .nextV(nextV), .D(Y) );

    FDRE #(.INIT(1'b1)) X0_FF (.C(clk), .CE(1'b1),
.D(X[0]), .Q(A[0])); // greset gives 001 (start in
hold state)
    FDRE #(.INIT(1'b0)) X1to3_FF[3:1] (.C({3{clk}}),
.CE({3{1'b1}}), .D(X[3:1]), .Q(A[3:1]));

    FDRE #(.INIT(1'b1)) Y0_FF (.C(clk), .CE(1'b1),
```

```
.D(Y[0]), .Q(B[0])); // greset gives 001 (start in
hold state)
    FDRE #(.INIT(1'b0)) Y1to3_FF[3:1] (.C({3{clk}}),
.CE({3{1'b1}}), .D(Y[3:1]), .Q(B[3:1]));

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 11/05/2019 04:57:33 PM
// Design Name:
// Module Name: StateLogic
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module NetLogicVert(
    input clk,
    input go,
    input start,
    input btnU,
    input btnD,
    input [9:0] netSize,
```

```
    input [9:0] startV,
    input win,
    input [3:0] Q,
    input restart,

    output [9:0] nextV,
    output [3:0] D
    );

    wire [9:0] tempV;

    // NEXT STATE
    assign D[3] = ( |Q[2:1] & win ) | ( Q[3] &
~restart ); // freeze in place
    assign D[2] = ( Q[1] & ( tempV == 10'd8 | btnD ) &
~win ) | ( Q[2] & ~( tempV >= ( 10'd471 - netSize) ) &
~win & ~btnU ); // down
    assign D[1] = ( Q[2] & ( ( tempV >= ( 10'd471 -
netSize) ) | btnU ) & ~win ) | ( Q[1] & ~( tempV ==
10'd8 ) & ~win & ~btnD ) | ( Q[0] & start ); // up
    assign D[0] = ( Q[0] & ~start) | ( Q[3] &
restart); // hold, start position


    // OUTPUTS
    countUD10L H( .clk(clk), .Up(Q[2] & go), .Dw(Q[1]
& go), .LD(Q[0]), .Din(startV), .Q(tempV) );
    assign nextV = tempV;

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 11/05/2019 04:57:33 PM
// Design Name:
// Module Name: StateLogic
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module BirdLogicHort(
    input clk,
    input go,
    input start,
    input rand,
    input [9:0] startH,
    input [2:0] Q,
```

```
    input restart,

    output [9:0] nextH,
    output [2:0] D
    );

    wire [9:0] tempH;

    // NEXT STATE
    assign D[2] = ( Q[1] & ( tempH == 10'd8 ) &
~restart) | ( Q[2] & ~( tempH == 10'd616 ) & ~restart)
| ( Q[0] & start & rand ); // right
    assign D[1] = ( Q[2] & ( tempH == 10'd616 ) &
~restart) | ( Q[1] & ~( tempH == 10'd8 ) & ~restart )
| ( Q[0] & start & ~rand ); // left
    assign D[0] = ( Q[0] & ~start) | ( |Q[2:1] &
restart ); // hold

    // OUTPUTS
    countUD10L H( .clk(clk), .Up(Q[2] & go), .Dw(Q[1]
& go), .LD(Q[0]), .Din(startH), .Q(tempH) );
    assign nextH = tempH;

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
////////////////////////////
// Company:
// Engineer:
//
// Create Date: 11/19/2019 04:25:41 PM
// Design Name:
// Module Name: Bird
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
////////////////////////////


module Bird(
    input clk,
    input go,
    input [1:0] rand,
    input start,
    input [9:0] startH,
    input [9:0] startV,
```

```
    input [9:0] netH, // net location
    input [9:0] netV,
    input [9:0] netSize, // net size
    input restart,
    input freeBirds,
    input showTime,

    output caught, // track if bird caught, if so add
blue to disp to get purp
    output [9:0] nextH,
    output [9:0] nextV
    );

    wire [2:0] A, B; // current states of H/V
    wire [2:0] X, Y; // next states of H/V
    wire [1:0] C; // current states of netted
    wire [1:0] Z; // next states of netted

    BirdLogicHort Hort( .clk(clk), .go(go),
.rand(rand[1]), .restart(restart), .start(start),
.Q(A), .startH(startH), .nextH(nextH), .D(X) );
    BirdLogicVert Vert( .clk(clk), .go(go),
.rand(rand[0]), .restart(restart), .start(start),
.Q(B), .startV(startV), .nextV(nextV), .D(Y) );
    BirdLogicNet Netted( .clk(clk),
.showTime(showTime), .freeBirds(freeBirds),
.netH(netH), .netV(netV), .netSize(netSize),
/*.timeUp(timeUp),*/ .Q(C), .nextH(nextH),
.nextV(nextV), .caught(caught), .D(Z) );

    FDRE #(.INIT(1'b1)) X0_FF (.C(clk), .CE(1'b1),
```

```
.D(X[0]), .Q(A[0])); // greset gives 001 (start in
hold state)
    FDRE #(.INIT(1'b0)) X1to2_FF[2:1] (.C({2{clk}}),
.CE({2{1'b1}}), .D(X[2:1]), .Q(A[2:1]));

    FDRE #(.INIT(1'b1)) Y0_FF (.C(clk), .CE(1'b1),
.D(Y[0]), .Q(B[0])); // greset gives 001 (start in
hold state)
    FDRE #(.INIT(1'b0)) Y1to2_FF[2:1] (.C({2{clk}}),
.CE({2{1'b1}}), .D(Y[2:1]), .Q(B[2:1]));

    FDRE #(.INIT(1'b1)) Z0_FF (.C(clk), .CE(1'b1),
.D(Z[0]), .Q(C[0])); // greset gives 01 (start in free
state)
    FDRE #(.INIT(1'b0)) Z1_FF (.C(clk), .CE(1'b1),
.D(Z[1]), .Q(C[1]));

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 11/05/2019 04:57:33 PM
// Design Name:
// Module Name: StateLogic
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module BirdLogicVert(
    input clk,
    input go,
    input rand,
    input start,
    input [9:0] startV,
    input restart,
```

```
    input [2:0] Q,

    output [9:0] nextV,
    output [2:0] D
    );

    wire [9:0] tempV;

    // NEXT STATE
    assign D[2] = ( Q[1] & ( tempV == 10'd8 ) &
~restart) | ( Q[2] & ~( tempV == 10'd456) & ~restart)
| ( Q[0] & start & rand); // down
    assign D[1] = ( Q[2] & ( tempV == 10'd456 ) &
~restart ) | ( Q[1] & ~( tempV == 10'd8 ) & ~restart )
| ( Q[0] & start & ~rand ); // up
    assign D[0] = ( Q[0] & ~start) | ( |Q[2:1] &
restart); // hold

    // OUTPUTS
    countUD10L H( .clk(clk), .Up(Q[2] & go), .Dw(Q[1]
& go), .LD(Q[0]), .Din(startV), .Q(tempV) );
    assign nextV = tempV;

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
////////////////////////////
// Company:
// Engineer:
//
// Create Date: 11/05/2019 04:57:33 PM
// Design Name:
// Module Name: StateLogic
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
////////////////////////////


module NetLogicHort(
    input clk,
    input go,
    input start,
    input win,
    input btnL,
    input btnR,
```

```
    input [9:0] netSize,
    input [9:0] startH,
    input [3:0] Q,
    input restart,

    output [9:0] nextH,
    output [3:0] D
    );

    wire [9:0] tempH;

    // NEXT STATE
    assign D[3] = ( |Q[2:1] & win ) | ( Q[3] &
~restart ); // freeze in place
    assign D[2] = ( Q[1] & ( tempH == 10'd8 | btnR ) &
~win) | ( Q[2] & ~( tempH >= ( 10'd631 - netSize) ) &
~win & ~btnL ); // right
    assign D[1] = ( Q[2] & ( ( tempH >= ( 10'd631 -
netSize) ) | btnL ) & ~win) | ( Q[1] & ~( tempH ==
10'd8 ) & ~win & ~btnR ) | ( Q[0] & start ); // left
    assign D[0] = ( Q[0] & ~start) | ( Q[3] & restart
); // hold



    // OUTPUTS
    countUD10L H( .clk(clk), .Up(Q[2] & go), .Dw(Q[1]
& go), .LD(Q[0]), .Din(startH), .Q(tempH) );
    assign nextH = tempH;

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
////////////////////////////
// Company:
// Engineer:
//
// Create Date: 11/05/2019 04:57:33 PM
// Design Name:
// Module Name: StateLogic
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
////////////////////////////


module BirdLogicNet(
    input clk,
    input [9:0] netH,
    input [9:0] netV,
    input [9:0] netSize,
    input freeBirds,
    input [1:0] Q,
```

```
    input [9:0] nextH,
    input [9:0] nextV,
    input showTime,

    output caught,
    output [1:0] D
    );

    wire caughtNow = ( ( nextH + 10'd15 ) >= netH ) &
( nextH <= ( netH + netSize ) ) & ( ( nextV + 10'd15 )
>= netV ) & ( nextV <= ( netV + netSize ) ) & showTime;

    // NEXT STATE
    assign D[1] = (Q[0] & caughtNow) | ( Q[1] & ~(
freeBirds ) ); // caught
    assign D[0] = ( Q[0] & ~caughtNow ) | ( Q[1] & (
freeBirds ) ); // free

    // OUTPUTS
    assign caught = Q[1];

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 10/29/2019 02:22:28 PM
// Design Name:
// Module Name: edge
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


 module edgeD(
    input clk,
    input btn,

    output o
    );
```

```
    wire [1:0] temp;

    FDRE #(.INIT(1'b0) ) ff1 (.C(clk), .R(1'b0),
.CE(1'b1), .D(btn), .Q(temp[0]));
    FDRE #(.INIT(1'b0) ) ff2 (.C(clk), .R(1'b0),
.CE(1'b1), .D(temp[0]), .Q(temp[1]));

    assign o = ~temp[1] & temp[0];

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 11/05/2019 11:13:11 AM
// Design Name:
// Module Name: LSFR
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module LSFR(
    input clk,

    output [7:0] rnd
    );

    wire start;
```

```
    assign start = rnd[0] ^ rnd[5] ^ rnd[6] ^ rnd[7];

    FDRE #(.INIT(1'b0) ) ff0 (.C(clk), .R(1'b0),
.CE(1'b1), .D(start), .Q(rnd[0]));
    FDRE #(.INIT(1'b0) ) ff1 (.C(clk), .R(1'b0),
.CE(1'b1), .D(rnd[0]), .Q(rnd[1]));
    FDRE #(.INIT(1'b0) ) ff2 (.C(clk), .R(1'b0),
.CE(1'b1), .D(rnd[1]), .Q(rnd[2]));
    FDRE #(.INIT(1'b0) ) ff3 (.C(clk), .R(1'b0),
.CE(1'b1), .D(rnd[2]), .Q(rnd[3]));
    FDRE #(.INIT(1'b0) ) ff4 (.C(clk), .R(1'b0),
.CE(1'b1), .D(rnd[3]), .Q(rnd[4]));
    FDRE #(.INIT(1'b0) ) ff5 (.C(clk), .R(1'b0),
.CE(1'b1), .D(rnd[4]), .Q(rnd[5]));
    FDRE #(.INIT(1'b0) ) ff6 (.C(clk), .R(1'b0),
.CE(1'b1), .D(rnd[5]), .Q(rnd[6]));
    FDRE #(.INIT(1'b1) ) ff7 (.C(clk), .R(1'b0),
.CE(1'b1), .D(rnd[6]), .Q(rnd[7])); // initially 1

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 10/24/2019 04:06:57 PM
// Design Name:
// Module Name: ring
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module ring(
    input digsel,
    input clk,

    output [3:0] out
    );
```

```
    wire [3:0] rcount;

    FDRE #(.INIT(1'b1) ) ff1 (.C(clk), .R(1'b0),
.CE(digsel), .D(rcount[3]), .Q(rcount[0])); //
initalizied to 1
    FDRE #(.INIT(1'b0) ) ff2 (.C(clk), .R(1'b0),
.CE(digsel), .D(rcount[0]), .Q(rcount[1]));
    FDRE #(.INIT(1'b0) ) ff3 (.C(clk), .R(1'b0),
.CE(digsel), .D(rcount[1]), .Q(rcount[2]));
    FDRE #(.INIT(1'b0) ) ff4 (.C(clk), .R(1'b0),
.CE(digsel), .D(rcount[2]), .Q(rcount[3]));

    assign out = rcount;

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 10/24/2019 03:40:36 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
    );
    //H is x[15:12] when sel=(1000)
```

```
    //H is x[11:8] when sel=(0100)
    //H is x[7:4] when sel=(0010)
    //H is x[3:0] when sel=(0001)

    wire fail;
    assign fail = (sel[0] & (|sel[3:1])) | (sel[1] &
(|sel[3:2])) | (sel[2] & sel[3]); // more than one is
on

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

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 10/16/2019 06:41:25 PM
// Design Name:
// Module Name: hex7seg
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module hex7seg(
    input [3:0] n,
    input e, // enable

    output [6:0] seg
    );
```

```
    m8_1e mux8A( .in({1'b0, n[0], n[0], 1'b0, 1'b0,
~n[0], 1'b0, n[0]}), .sel({n[3], n[2], n[1]}), .e(e),
.o(seg[0])); // A
    m8_1e mux8B( .in({1'b1, ~n[0], n[0], 1'b0, ~n[0],
n[0], 1'b0, 1'b0}), .sel({n[3], n[2], n[1]}), .e(e),
.o(seg[1])); // B
    m8_1e mux8C( .in({1'b1, ~n[0], 1'b0, 1'b0, 1'b0,
1'b0, ~n[0], 1'b0}), .sel({n[3], n[2], n[1]}), .e(e),
.o(seg[2])); // C
    m8_1e mux8D( .in({n[0], 1'b0, ~n[0], n[0], n[0],
~n[0], 1'b0, n[0]}), .sel({n[3], n[2], n[1]}), .e(e),
.o(seg[3])); // D
    m8_1e mux8E( .in({1'b0, 1'b0, 1'b0, n[0], n[0],
1'b1, n[0], n[0]}), .sel({n[3], n[2], n[1]}), .e(e),
.o(seg[4])); // E
    m8_1e mux8F( .in({1'b0, n[0], 1'b0, 1'b0, n[0],
1'b0, 1'b1, n[0]}), .sel({n[3], n[2], n[1]}), .e(e),
.o(seg[5])); // F
    m8_1e mux8G( .in({1'b0, ~n[0], 1'b0, 1'b0, n[0],
1'b0, 1'b0, 1'b1}), .sel({n[3], n[2], n[1]}), .e(e),
.o(seg[6])); // G

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 10/24/2019 03:05:17 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,
    input Dw,
    input LD,
    input [15:0] Din, // only for load enabled
```

```
    output [15:0] Q,
    output UTC,
    output DTC
    );

    wire [3:0] outUTC, outDTC;
    wire [15:0] tempQ;

    countUD3L count3_0(.clk(clk), .Up(Up), .Dw(Dw),
.LD(LD), .Din(Din[2:0]), .Q(tempQ[2:0]),
.UTC(outUTC[0]), .DTC(outDTC[0]));
    countUD5L count5_1(.clk(clk),
.Up(outUTC[0]&Up&~LD), .Dw(outDTC[0]&Dw), .LD(LD),
.Din(Din[7:3]), .Q(tempQ[7:3]), .UTC(outUTC[1]),
.DTC(outDTC[1]));
    countUD5L count5_2(.clk(clk),
.Up((&outUTC[1:0])&Up&~LD),
.Dw((&outDTC[1:0])&Dw&~LD), .LD(LD), .Din(Din[12:8]),
.Q(tempQ[12:8]), .UTC(outUTC[2]), .DTC(outDTC[2]));
    countUD3L count3_3(.clk(clk),
.Up((&outUTC[2:0])&Up&~LD),
.Dw((&outDTC[2:0])&Dw&~LD), .LD(LD), .Din(Din[15:13]),
.Q(tempQ[15:13]), .UTC(outUTC[3]), .DTC(outDTC[3]));

    assign UTC = &outUTC;
    assign DTC = &outDTC;
    assign Q = tempQ;

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 10/24/2019 03:05:17 PM
// Design Name:
// Module Name: countUD16L
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module countUD10L(
    input clk,
    input Up,
    input Dw,
    input LD,
    input [9:0] Din, // only for load enabled
```

```
    output [9:0] Q
    );

    wire [1:0] outUTC, outDTC;
    wire [9:0] tempQ;

    countUD5L count5_1(.clk(clk), .Up(Up), .Dw(Dw),
.LD(LD), .Din(Din[4:0]), .Q(tempQ[4:0]),
.UTC(outUTC[0]), .DTC(outDTC[0]));
    countUD5L count5_2(.clk(clk),
.Up(outUTC[0]&Up&~LD), .Dw(outDTC[0]&Dw), .LD(LD),
.Din(Din[9:5]), .Q(tempQ[9:5]), .UTC(outUTC[1]),
.DTC(outDTC[1]));

    assign Q = tempQ;

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 10/22/2019 03:45:06 PM
// Design Name:
// Module Name: countUD5L
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module countUD5L(
    input clk,
    input Up,
    input Dw,
    input LD,
    input [4:0] Din, // only for load enabled
```

```
    output [4:0] Q,
    output UTC,
    output DTC
    );

    wire enable;
    assign enable = Up | Dw | LD; // if all on then
load overrides. up/dw cancel out
    // ~enable = ~Up & ~Dw & ~LD;

    wire [4:0] D;

    FDRE #(.INIT(1'b0) ) ff0 (.C(clk), .R(1'b0),
.CE(enable), .D(D[0]), .Q(Q[0]));
    FDRE #(.INIT(1'b0) ) ff1 (.C(clk), .R(1'b0),
.CE(enable), .D(D[1]), .Q(Q[1]));
    FDRE #(.INIT(1'b0) ) ff2 (.C(clk), .R(1'b0),
.CE(enable), .D(D[2]), .Q(Q[2]));
    FDRE #(.INIT(1'b0) ) ff3 (.C(clk), .R(1'b0),
.CE(enable), .D(D[3]), .Q(Q[3]));
    FDRE #(.INIT(1'b0) ) ff4 (.C(clk), .R(1'b0),
.CE(enable), .D(D[4]), .Q(Q[4]));

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

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

    assign UTC = &Q;
    assign DTC = &(~Q);

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 10/22/2019 03:44:43 PM
// Design Name:
// Module Name: countUD3L
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module countUD3L(
    input clk,
    input Up,
    input Dw,
    input LD,
    input [2:0] Din, // only for load enabled
```

```
    output [2:0] Q,
    output UTC,
    output DTC
    );

    wire enable;
    assign enable = Up | Dw | LD; // if all on then
load overrides. up/dw cancel out
    // ~enable = ~Up & ~Dw & ~LD;

    wire [2:0] D;

    FDRE #(.INIT(1'b0) ) ff1 (.C(clk), .R(1'b0),
.CE(enable), .D(D[0]), .Q(Q[0]));
    FDRE #(.INIT(1'b0) ) ff2 (.C(clk), .R(1'b0),
.CE(enable), .D(D[1]), .Q(Q[1]));
    FDRE #(.INIT(1'b0) ) ff3 (.C(clk), .R(1'b0),
.CE(enable), .D(D[2]), .Q(Q[2]));

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

    assign UTC = &Q;
    assign DTC = &(~Q);
```

```
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 10/15/2019 03:33:43 PM
// Design Name:
// Module Name: m8_1e
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module m8_1e(
    input [7:0] in,
    input [2:0] sel,
    input e, // enable

    output o
    );
```

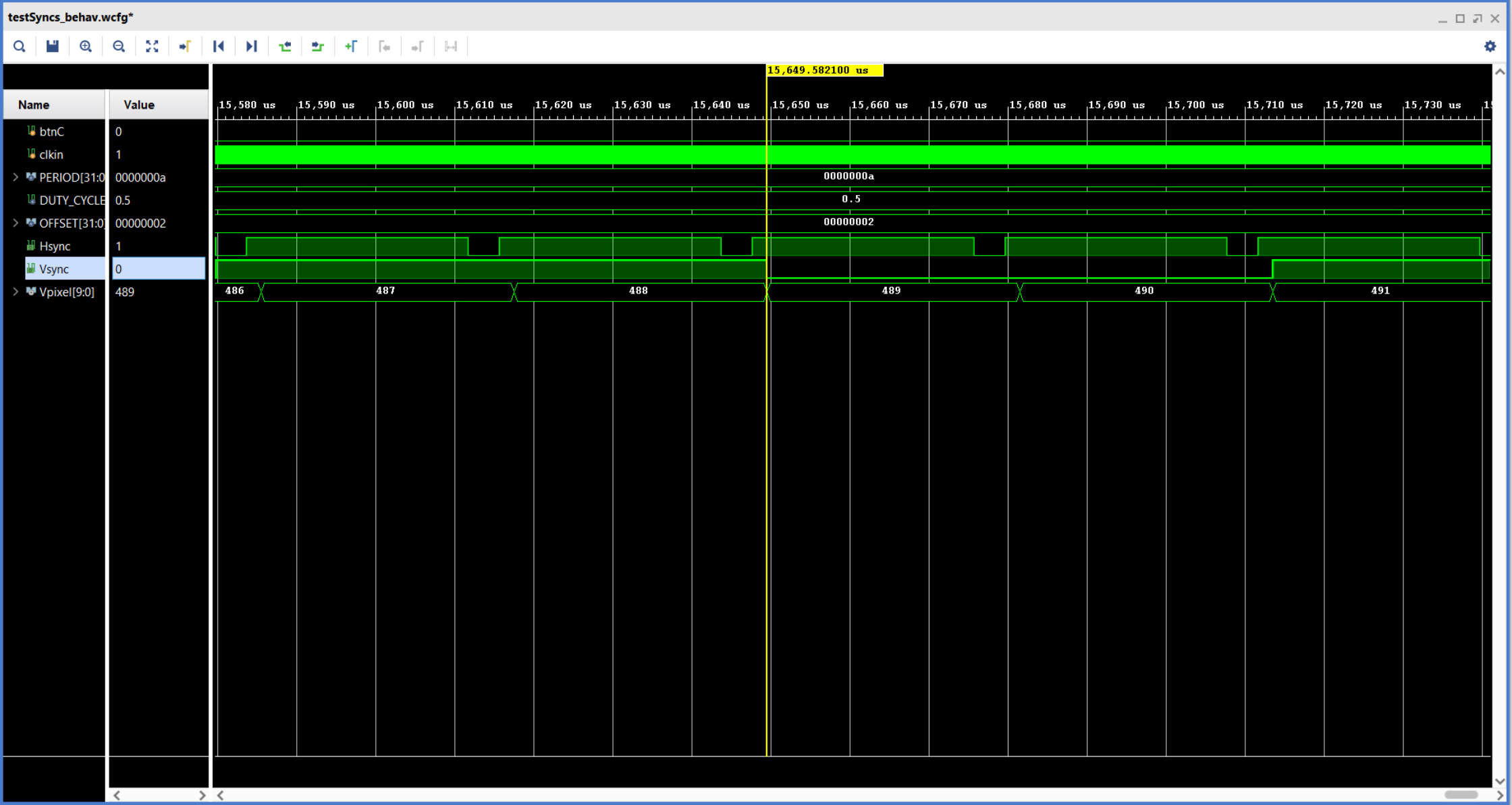
```
    wire [7:0] temp;

    assign temp[0] = in[0] & ~sel[2] & ~sel[1] &
~sel[0]; // 000
    assign temp[1] = in[1] & ~sel[2] & ~sel[1] &
sel[0];  // 001
    assign temp[2] = in[2] & ~sel[2] & sel[1] &
~sel[0];  // 010
    assign temp[3] = in[3] & ~sel[2] & sel[1] &
sel[0];   // 011
    assign temp[4] = in[4] & sel[2] & ~sel[1] &
~sel[0];  // 100
    assign temp[5] = in[5] & sel[2] & ~sel[1] &
sel[0];   // 101
    assign temp[6] = in[6] & sel[2] & sel[1] &
~sel[0];   // 110
    assign temp[7] = in[7] & sel[2] & sel[1] & sel[0];
   // 111

    assign o = (|temp) & e;

endmodule
```

testSyncs_behav.wcfg*



Timing

General Information
Timer Settings
Design Timing Summary
Clock Summary (3)
Check Timing (39)
Intra-Clock Paths
Inter-Clock Paths
Other Path Groups
User Ignored Paths
Unconstrained Paths

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	28.915 ns	Worst Hold Slack (WHS):	0.090 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:	548	Total Number of Endpoints:	548	Total Number of Endpoints:	328

All user specified timing constraints are met.

Timing Summary - impl_1 (saved) | **Timing Summary - timing_1**

Timing

- General Information
- Timer Settings
- Design Timing Summary
- Clock Summary (3)
- Check Timing (39)
- Intra-Clock Paths
- Inter-Clock Paths
- Other Path Groups
- User Ignored Paths
- Unconstrained Paths

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

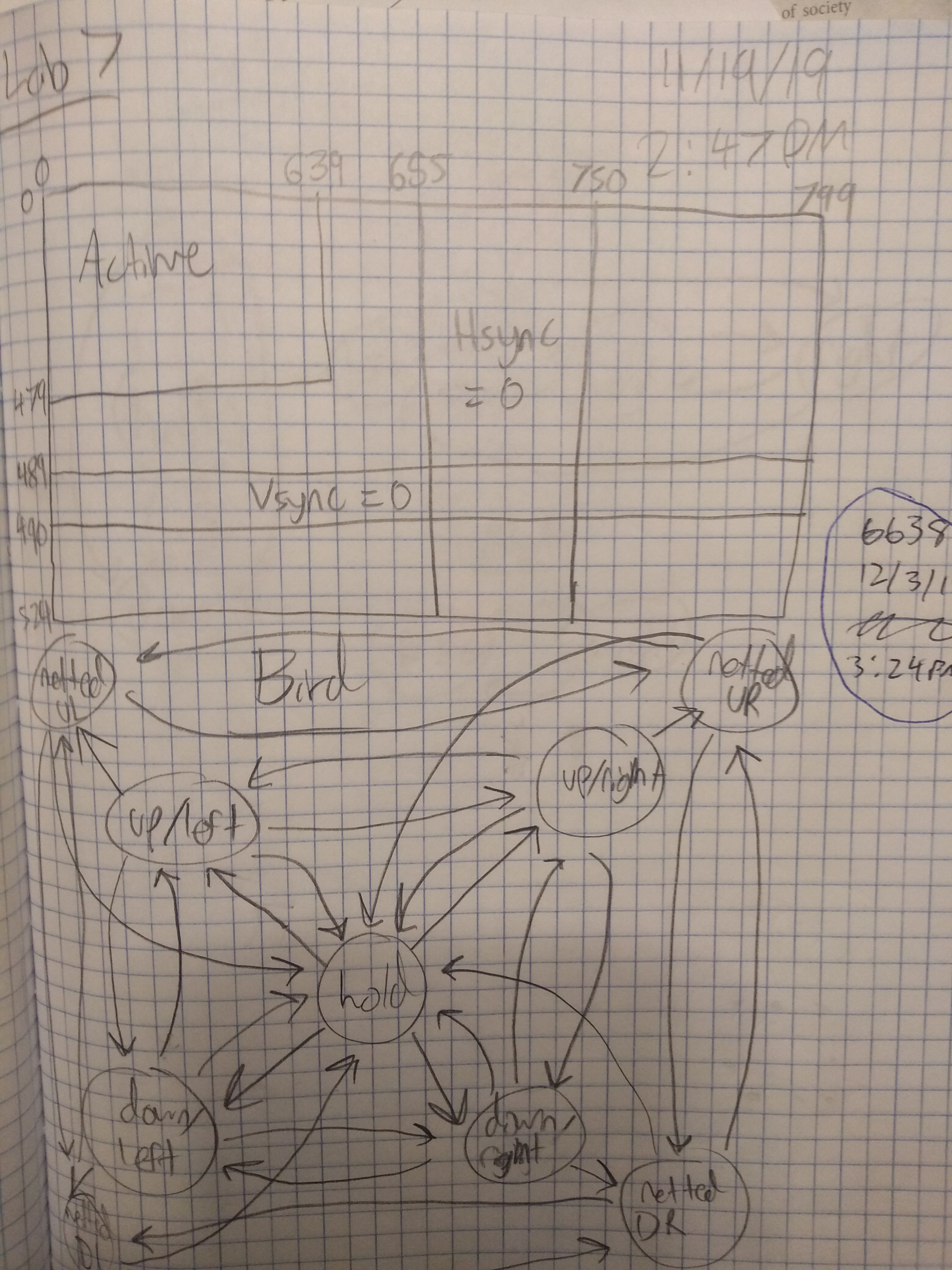
Timing Summary - impl_1 (saved) | **Timing Summary - timing_1**

of society

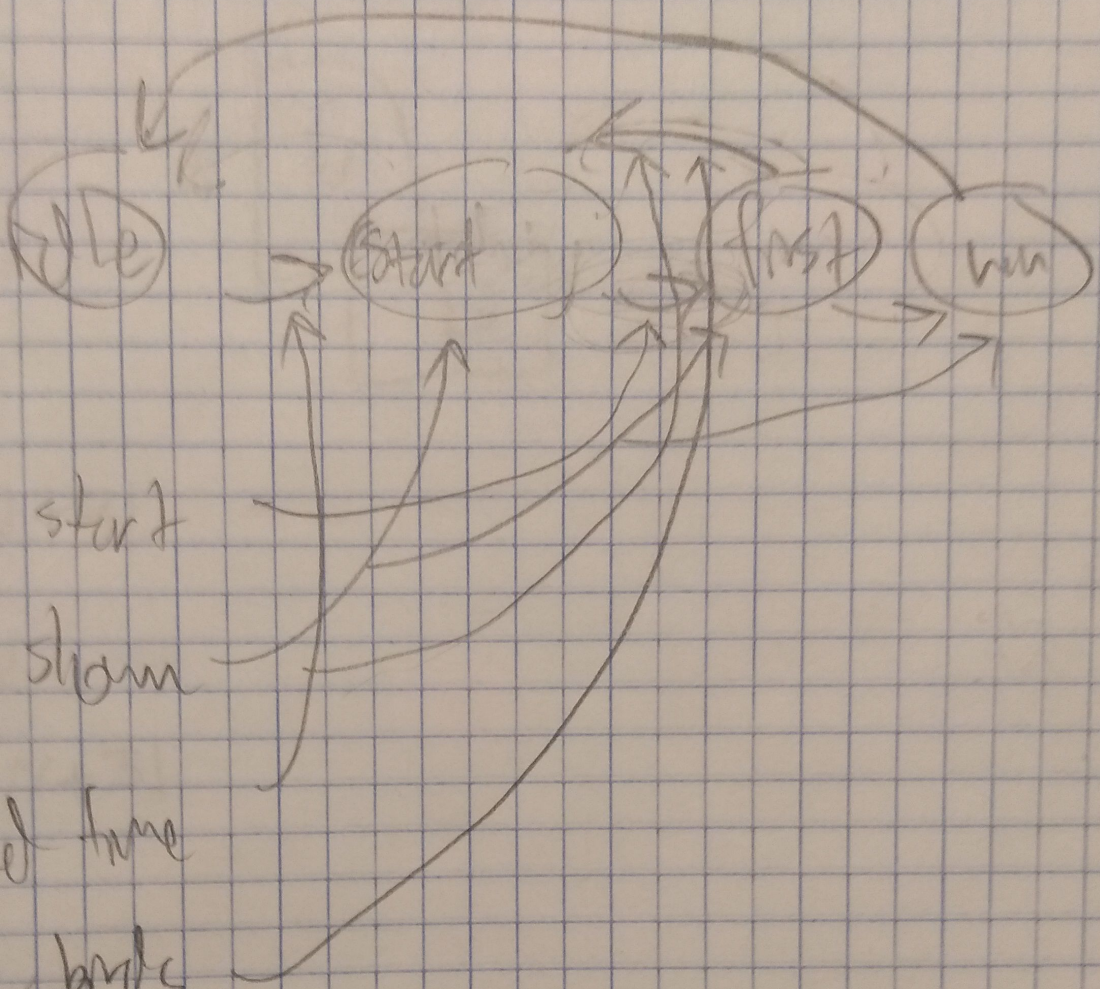
Lab 7

11/19/19

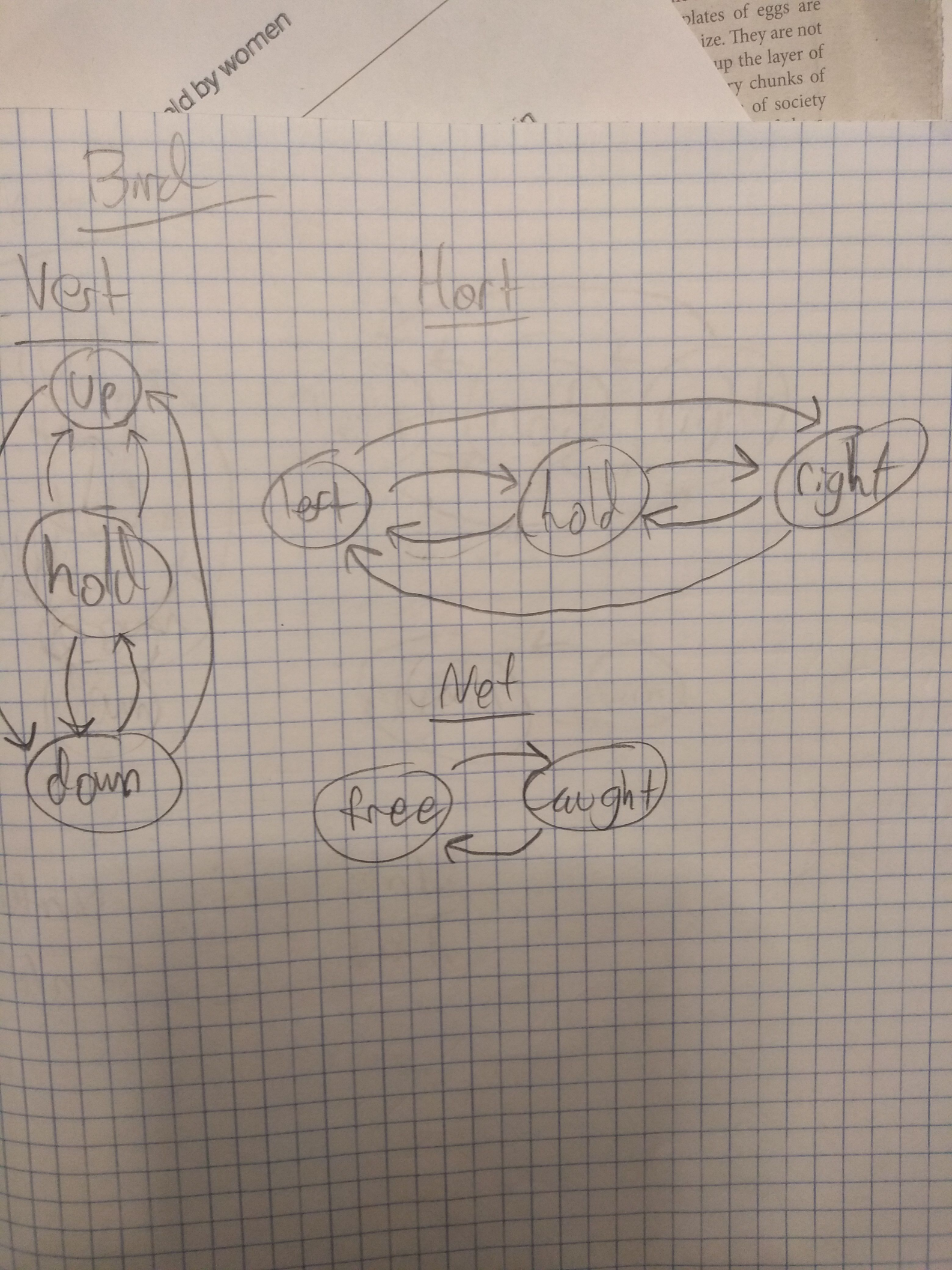
2:47 PM



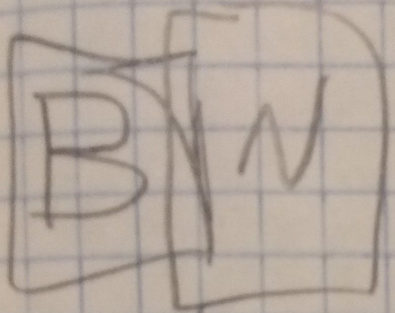
Scene

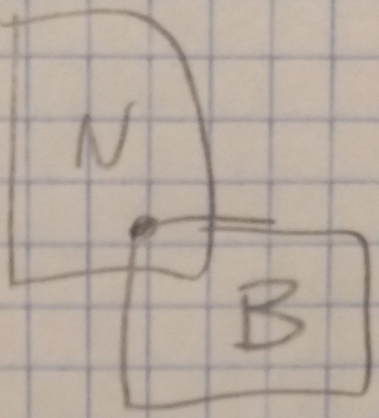


State	Time
idle	doesnt matter, not shown
start	show zero, maybe use selector?
finish	counting down
run	same time as exact finish


Bird
Vest
up
hold
down
Hart
left
hold
right
Net
free
caught

B and N not overlap



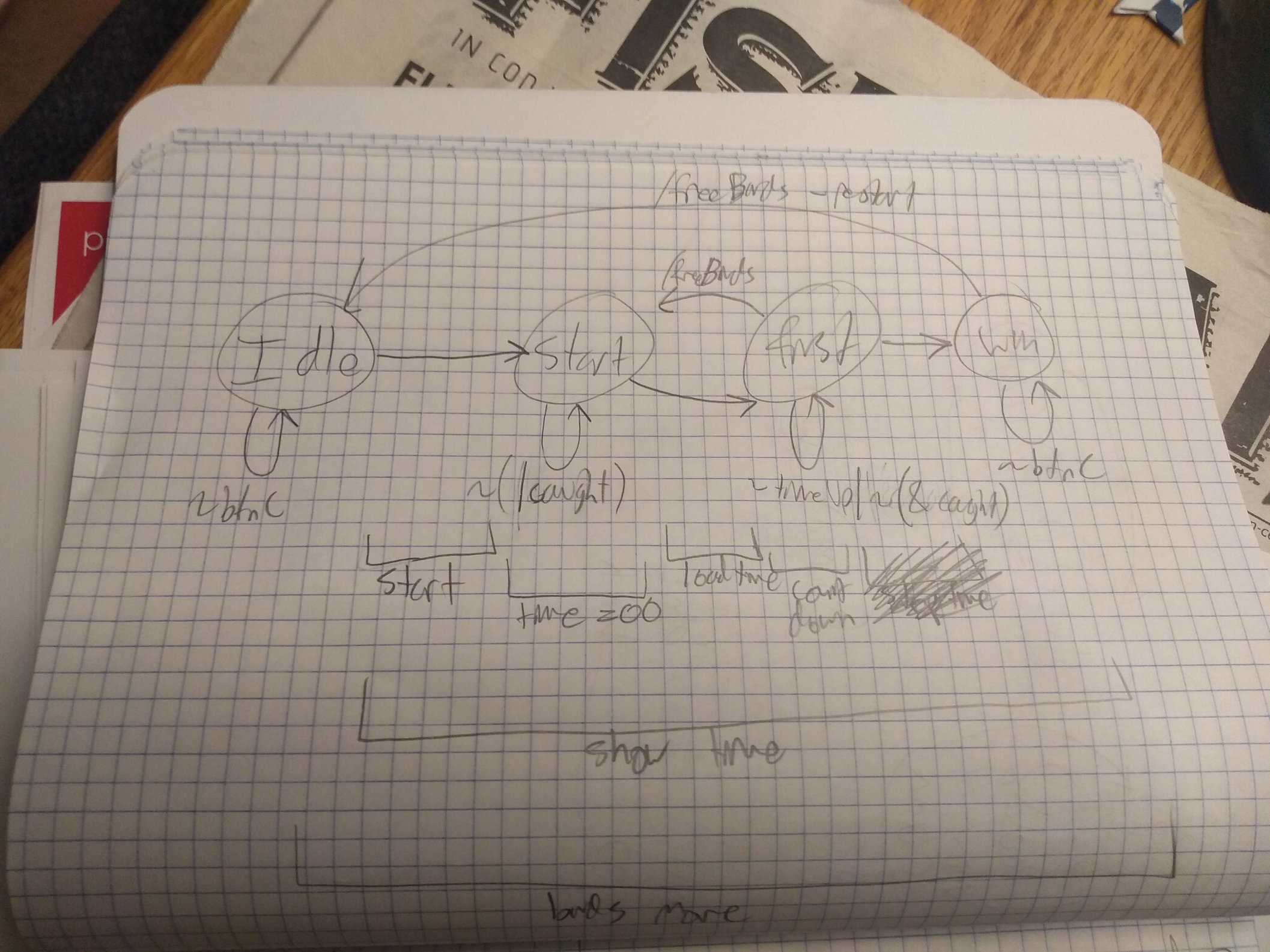


$B + 15 \geq N$

$B \leq N + 31$

$B + 15 \geq N$

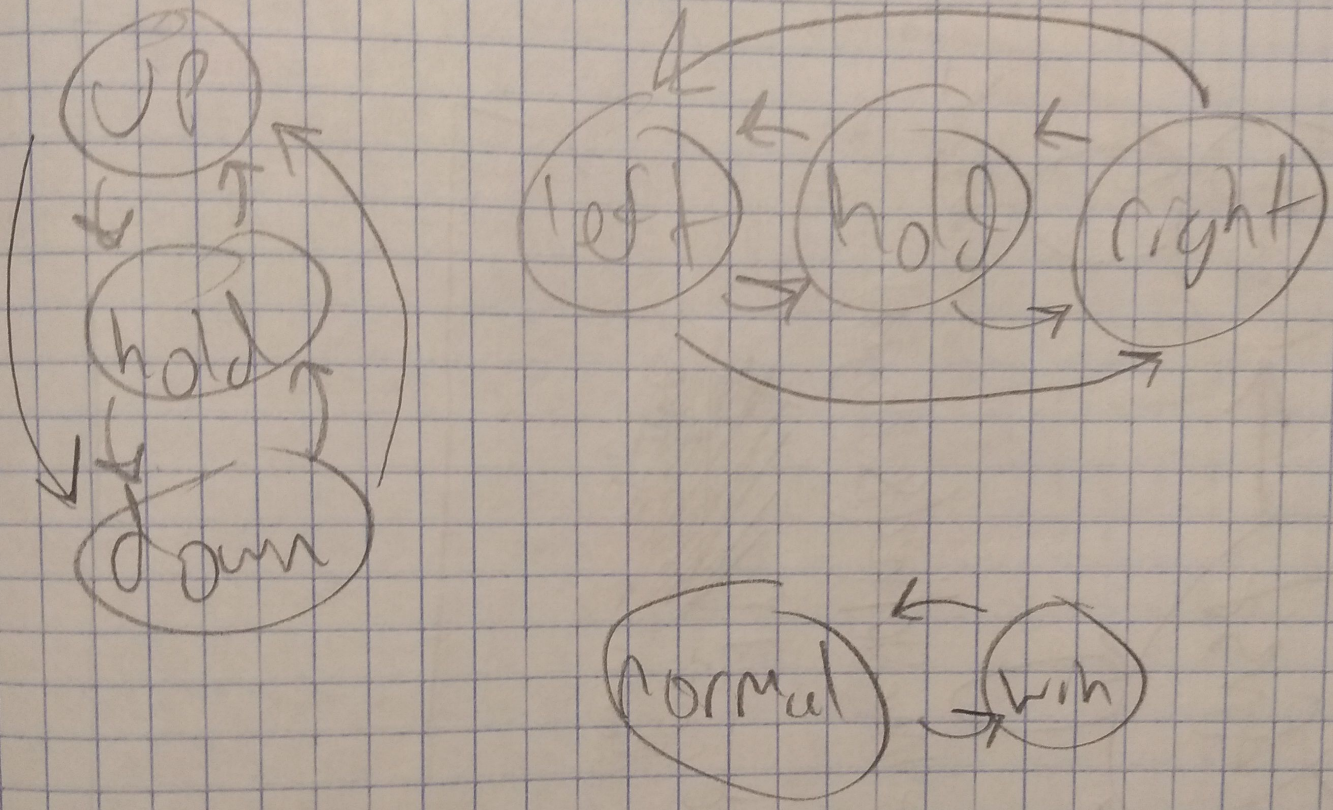
$B \leq N + 31$


freeBirds = restart
freeBirds
Idle
start
first
win
~btnC
~(|caught)
~timeUp | ~(&caught)
~btnC
start
time=00
load time
count down
show time
birds more

'd by women

The slices of orang
plates of eggs are
ize. They are not
up the layer of
y chunks of
of society

Net



btnU
btnL
btnR
btnD