Lab 7: Greatest Game in the World!

Gavin Chen

6/6/18

CE100 Spring 2018

Description:

This lab was the culmination of all the concepts we have been investigating throughout the year. In order to implement the game, we created state machines, counters, segment display logic and much more. Overall the lab was interesting in that we were able to implement a large project, as well as interface with the VGA port on the board to display on a monitor. All in all the lab was interesting and challenging.

Methods:

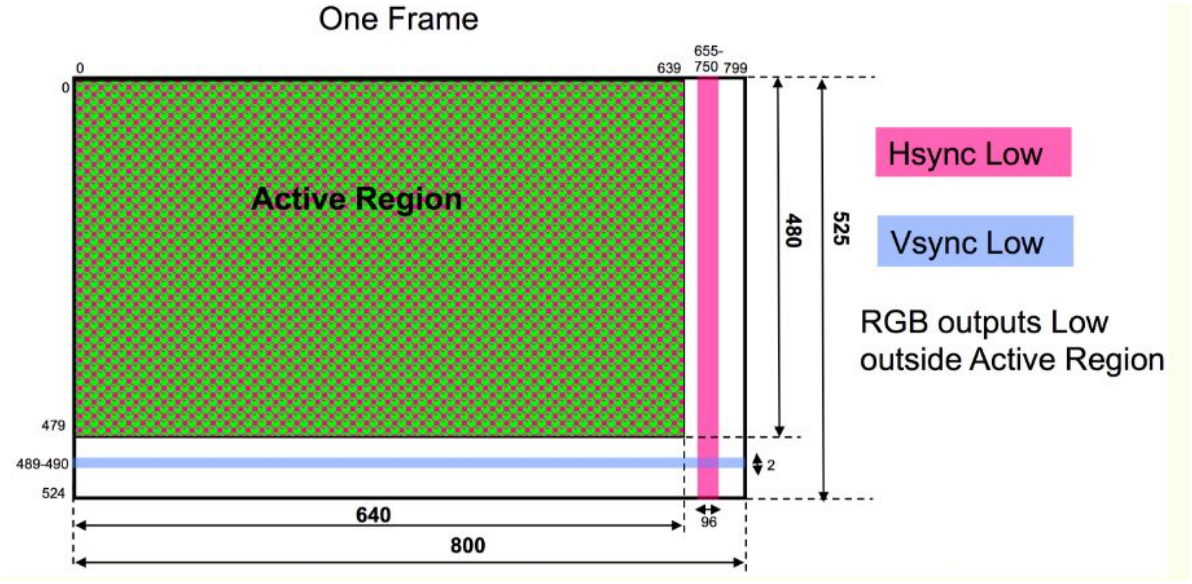
There were many parts to this lab, and designing was done incrementally and components were designed to be as versatile as possible.

Counter:

The counter was created using logic implemented in a previous lab. Since the dimension of the screen was determined to be 800 x 525, a 10 bit counter was needed to index all pixels on the screen. As such, we created a 5 bit counter using previous lab logic, and combined 2 instances together to create a 10 bit counter. The counter took in an increment, decrement, and load in value setting. It output a 10 bit bus that could cover the entire range of the screen, and was synchronized with the clock to move in step.

VGA pixel animator:

The VGA screen worked as a constant refresh rate, where the color of each individual pixel depended on the color output on the vga port at that time. To create an animator, it was necessary to implement a counter that could reach all bit values throughout the screen, and which incremented along with the refresh rate of the monitor. Since the lab clock was already synchronized with the refresh rate, we needed only to deal with the counter portion. A diagram of the screen is shown to the left. Two 10 bit counters were used, one for the horizontal position and one for the vertical. The horizontal counter reset when the count reached 799. It incremented with every clock cycle since the clock was synced to the refresh rate. The vertical counter reset upon reaching 524, signifying the end of the screen. The active region, Vsync, Hsync, and pixel range for a single VGA frame is shown below, courtesy of the CE100 Lab manual.

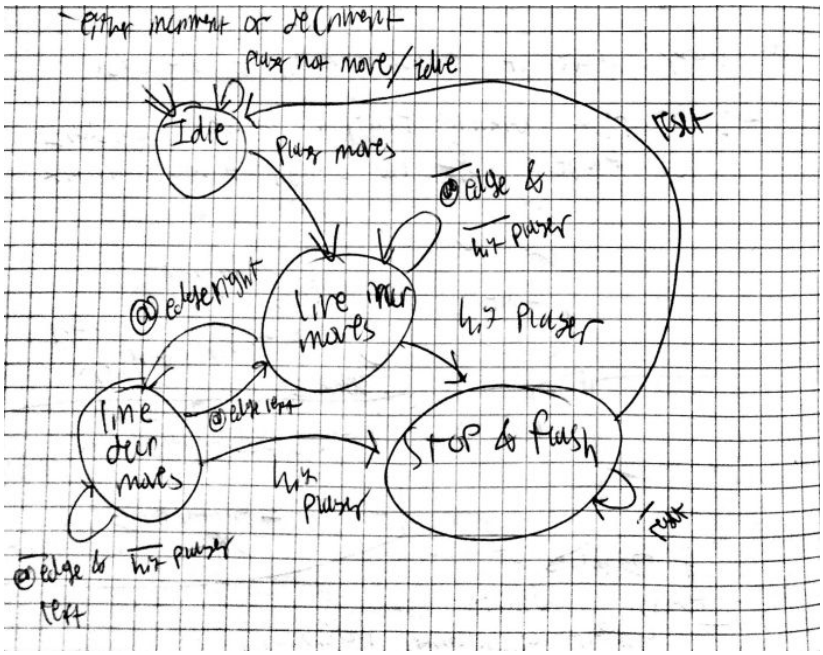


Player:

Generating the slug player on the screen was simply a matter of storing the position and animating the player for the correct area. Two 10-bit counters were implemented to keep track of the horizontal and vertical coordinates of the upper left corner of the player. To move left or right, the counter responsible for the horizontal position was decrimented or incremented respectively. The same was done for the vertical coordinates and up-down movement. A module was created to handle all of the player semantics. The counter was blocked from incrementing or decrementing if the player got within 10 pixels of the screen, representing the player coming to the border. A state machine was also created to halt and flash the player if it ever came in contact with a red obstacle. To accomplish this the bits representing the screen pixel position was passed in as input. If the screen position was within a 10 pixel range of the upper left hand corner coordinate, a full green output was sent. The red VGA output was also taken as input to the player module. If at any point the module indicated a green output while red was also high, that meant an intersection between the player and an obstacle. In this case the player movement was disabled and the player itself was made to flash.

Obstacle state machine:

Creating the obstacle was the most difficult part of the assignment. We began by creating a state machine that could implement the desired behaviour of the obstacle. There were 5 states: Idle where a solid red line was drawn, move left/up which was the first state transferred into upon player movement or if the obstacle bumped into the right/bottom, move right which was transferred into upon bumping into the upper or left hand border, and finally a stop and flash state which was entered into if the player hit the obstacle. A diagram of the state machine is shown to the right, and a table of the transition/output logic is included in the lab notebook section. It took as input the edge bounce event, a hit and move player event, and a reset.



Obstacle:

The obstacle was created around the state machine in the same method as the player module. Given the state machine, two 10-bit counters were added into a parent module that represented the upper left most point in the gap along the red line. The current position of the screen pixel was also passed in as input, same as with the player module. A start position of the

obstacle was also taken as input. An orientation bit was added to decide the orientation of the obstacle, either vertical or horizontal. If the screen bit was within 10 pixels either horizontally or vertically, and was within the correct horizontal or vertical border of the entire screen, a red signal was output to the VGA handler. When the obstacle was needed to move, it was decided that it would be easier to make an empty box that moved along the solid red line, rather than have the red line itself move. Therefore upon signal from the state machine the empty box length was calculated based on the switch input, and it began to oscillate back and forth. As with the player, the movement was restricted to one pixel difference per frame render. Also similar to the play module, a hit was detected as an overlap of the green VGA input with the individual red obstacle.

Timer and score module:

The timer and score module were combined together as it was seen as easier to handle rather than separately. Since both the timer and the score utilized the 7-segment display, they were bundled together in a module that also included a method for showing digits on the display. Showing digits on the display was done using modules created in previous labs including: a ring counter, selector, and hex to segment converter. The difficulty in displaying the correct information lay in deciding when to show the time versus the score, and how to increment the time counter. The time was kept track of using four 5-bit counters. The first and third counters corresponded to the second and minute counters. Both of these counters were created to roll over and reset upon reaching 10, while simultaneously sending out an up count signal. The second and fourth counters were made to roll over and output up count on a count of 6. The counters were wired to feed into each other, and a second input signal was used to increment the second counter which in turn powered subsequent counters. The score counters were similarly just individual 5-bit counters. All the outputs were piped into the selector to interface with the display.

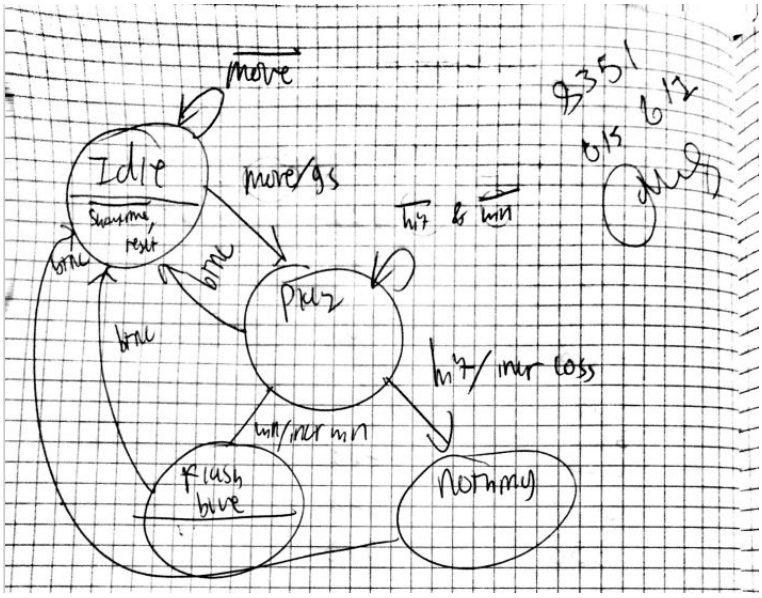
The only additional challenge was deciding when to show the score or the time. This decision was left up to the game state machine, and the module took in a single bit to dictate either showing the time or the score. The bit was used as a bit mask to suppress the corresponding output, therefore only one thing was output to the segment display at any given moment.

Border:

The border module was created as a simple interface to dictate when to output blue. The screen position was fed into the module, and a comparison was done to output high on the blue VGA output if the pixel was within 10 pixels of the border. A flash bit and half second bit were also input to be able to flash the border when the player won.

Game state machine:

The game state machine was the final piece in organizing all of the components. It consisted of 4 states. The beginning idle state which was remained in until the player moved, a play state which operated the game as normal, and 2 termination states. Either the player reached the corner in which case the state machine went into the win state and signal to



increment the score win counter and flash the blue border were sent out to the respective modules. Or the player hit a border in which case only the score loss counter was incremented. The win and hit event were both calculated by the respective player or obstacle module. In the event of either a signal was simply sent to the state machine to transition. Within all states, the ability existed to jump back into the idle state if the center button was pressed. In this case the idle state output a reset condition, and output a show score signal to the display module. A diagram of the state machine is shown above.

Top level module:

The top level module organized and tied in all the individual pieces of the lab to create one whole functioning game. It consisted of a single player module, 14 obstacle modules set to either horizontal or vertical, a game state machine, a screen animation counter, a border module, and a sync module to synchronize the VGA inputs. The start position of the obstacles were specified to space them out across the entirety of the screen, and the inputs and outputs of each module were connected together so that the necessary information could be passed around to each module. Finally the on board I/O was tied into each of the corresponding modules, with the buttons going to the player, and switches to the obstacles. Finally, the VGA ports were wired into the VGA animation counter module.

Results:

Counter:

The counter module was fairly straightforward to apply since we could reuse logic and modules from previous labs. The 10-bit counter was created out of 2 seperate 5-bit counters. Since the 10-bit counter would be used in most modules, it was tested first to ensure it worked as expected. After some error in tieing the 2 individual counter modules together, the 10-bit counter was able to function as expected.

VGA pixel animator:

The VGA pixel animator was essentially a specialized version of the 10-bit counter. Upon first creation there was some error in the Vsync and Hsync signals as they were outputting too early. However the error was found and corrected. Overall the VGA pixel animator worked

eventually and output the correct signals for given counter values. Since the overall board clock was set to the correct frequency to operate the VGA monitor, the counter incremented with each clock cycle as the increment event. Once again, since most of the other modules depended on the VGA pixel animator to essentially sync up logic calculations, thorough testing was done to ensure it worked as expected.

Player:

The player module had several issues with the state machine. Namely, the state machine was failing to recognize the intersection with an obstacle. The error turned out to be an issue with the input, and the other pieces involving animation and flashing were implemented easier. Since the player module depended on the outside top module, it became difficult to simulate the module when it was needed to interface with others in the top level. As such, a full board upload was done to test for bugs and to patch issues as opposed to the simulation.

Obstacle state machine:

In creating the obstacle state machine, error testing could be done somewhat simpler due to the fact it was nested within the obstacle module. There was some issue in the output and transition logic, but since the state machine was rather small and simple the errors were quickly found.

Obstacle:

The most difficult to catch errors occurred in the obstacle parent module. There were several errors in the rendering of an obstacle, as well as the movement of the gap within the line. The main error was in that the obstacle would reverse directions too soon, i.e. without first hitting the border. This was found to be due to the way in which an edge bounce event was calculated. It was necessary to include the orientation of the obstacle within the bounce logic, and upon doing so the error was resolved and the obstacle worked as promised. As was the case with the other modules, it was difficult to test the obstacle individually as its behaviour depended on much of the other modules output. Therefore when testing, simulations were done on the state machine, but testing the overall parent module was done through uploading the entire bitstream to the board and running a physical version.

Timer and score module:

The time and score module were easier to test as they encapsulated the entire display within a single module. Since the only interaction with the other modules was through the clock signal to generate a second signal, the entire module could be simulated much easier than the other modules. The main issue was in creating the time counters since the roll over and increment conditions changed with each counter. However through simulation the error was quickly accounted for and the module worked without issue.

Border:

The border module was the simplest animation module out of the three to write. It simply consisted of several lines to check for the blue condition and when to output the needed signal.

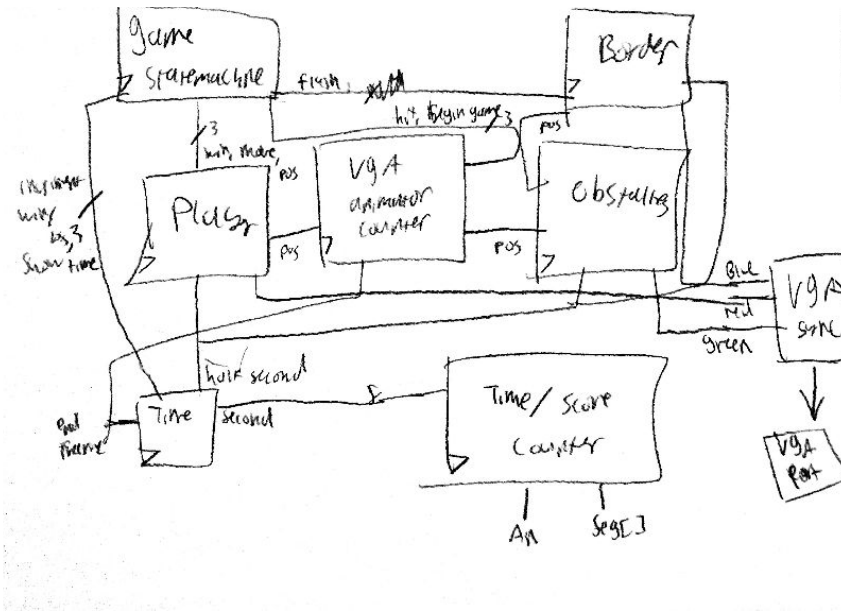
The module was the first to be implemented in the project, and served as the baseline to ensure that the animation and comparison logic worked as intended. Since the blue module was the first to be implemented, testing it in the top level was simpler than with the other modules. It was also faster as fewer components needed to be tested. The only issue faced with the blue module was in creating the flashing effect, however it was done eventually once the state machine was created.

Game state machine:

The game state machine was the same as the obstacle in that it was simpler to test given that the inputs worked correctly. There was some issue in transferring into the flash blue state upon winning the since the condition to check the player won was somewhat skewed. The game state machine itself worked as intended, as well as the output logic. The difficulty with it lay in the calculating the correct input, which was an issue within the individual modules more so than the game state machine.

Top level module:

The top level was implemented as the combination of all the individual pieces. Due to the complexity and amount of inputs that the top level managed, it was difficult to simulate the entire module since there were too many inputs to toggle and outputs to monitor. In the end bug finding was done by uploading the entire compiled bit schematic onto the board and testing the individual components one by one. It was a lengthy and tedious process, but worked in the end. A rough abstract schematic of the top module is presented below, with a sense of how the individual components were interconnected such that the desired information to control the state machines and the animation were passed around.



Timing summary:

The timing summary indicated that the maximum clock frequency that could drive the board without issue is given by the largest overall timing slack. The worst overall slack came from the setup phase, with a time of 18.11ns. This correspond to a period time of 55.22MHz. This was the highest clock frequency before timing errors would occur in the setup phase. As the on board clock was set to 25MHz, we were well below the limit and no timing errors occurred.

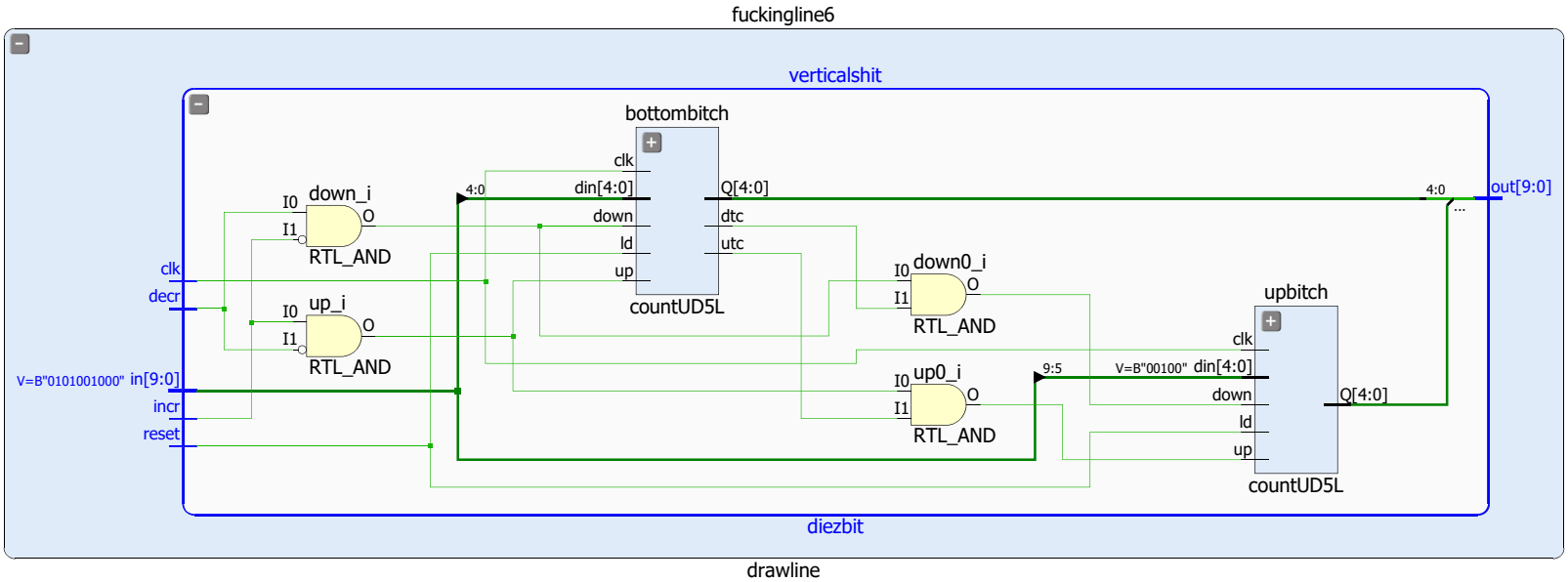
Conclusion:

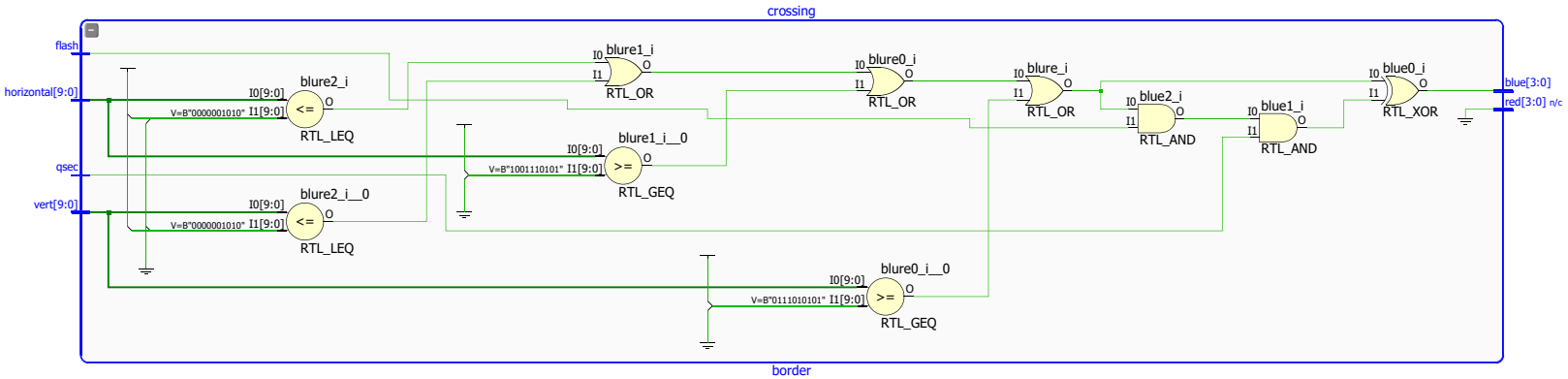
The lab was interesting and combined many concepts we have worked with throughout the quarter. The most interesting part was in organizing the VGA animation. Each module itself was not difficult to implement logically, perhaps with the exception of the obstacle module. However due to the number of components and the complexity of the entire project, organizing all the components into the final project proved to be the most difficult part. In the end the lab was fun, and we all had loads of fun guiding the slug to eventual victory over the red wormed devils!

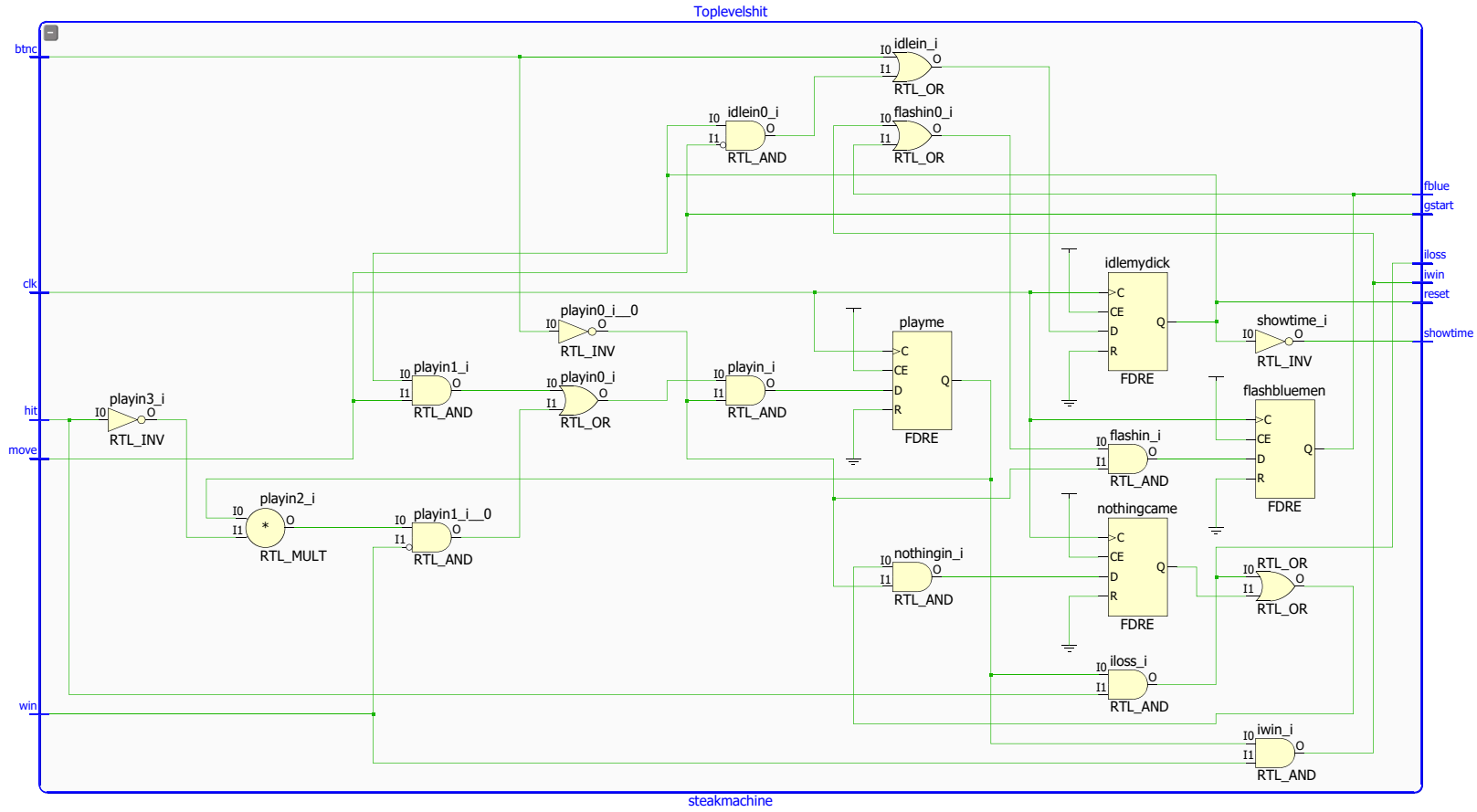
Bibliography

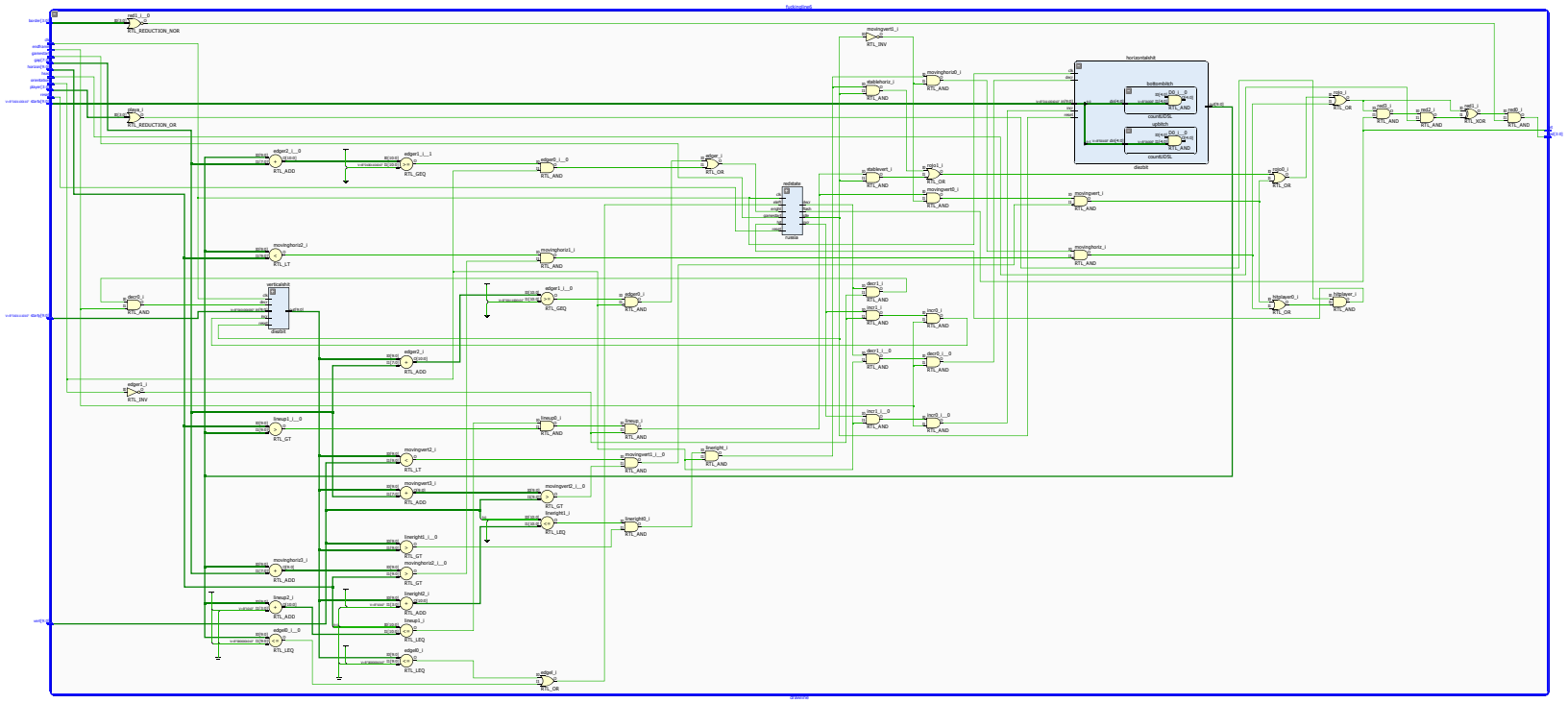
CE100 UCSC lab 7 Lab manual diagram for VGA output, Martine Schlag

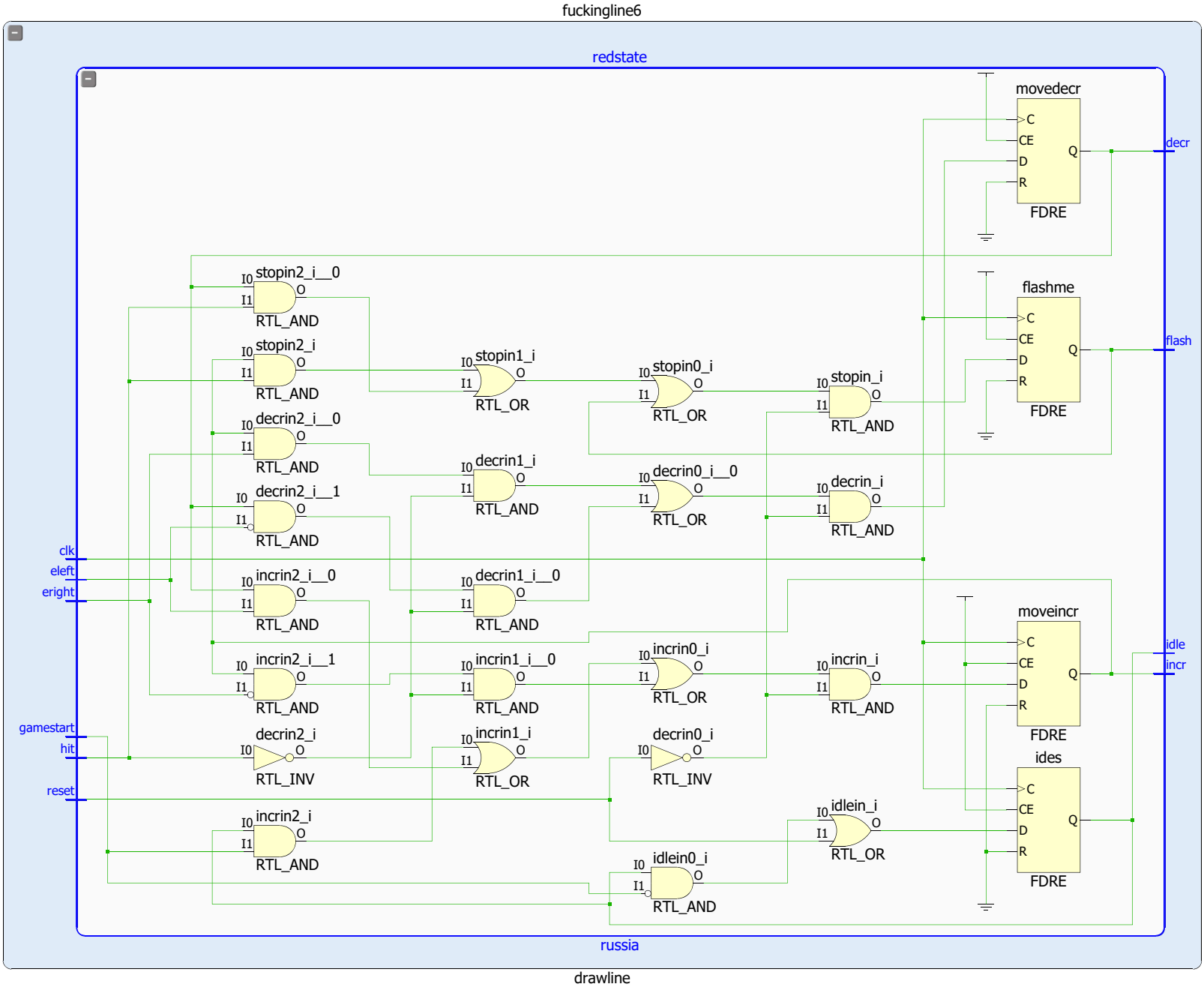
Appendix

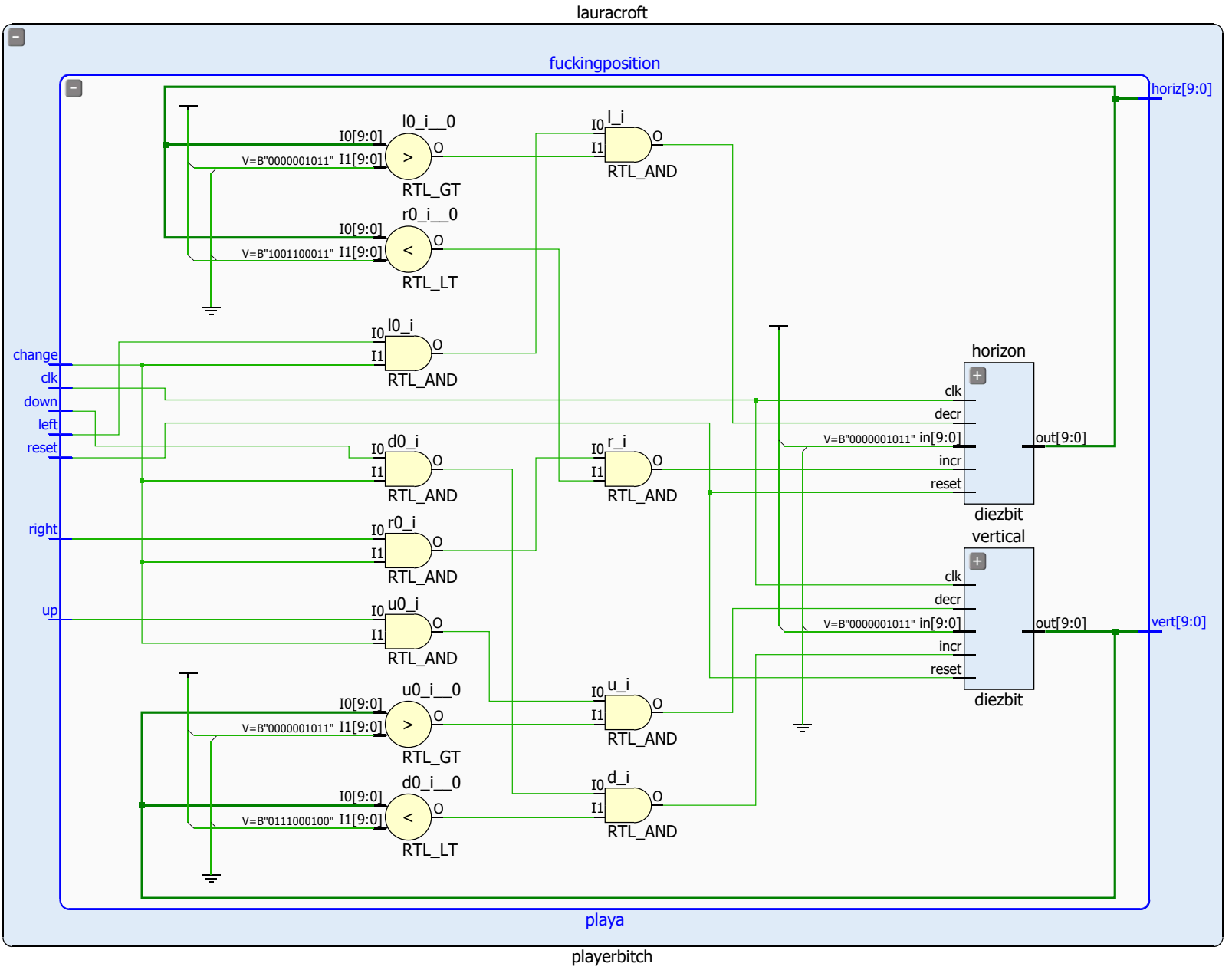

fuckingline6
verticalshit
bottombitch
clk
din[4:0]
down
ld
up
Q[4:0]
dtc
utc
countUD5L
4:0
down_i
I0
I1
O
RTL_AND
up_i
down0_i
up0_i
clk
decr
in[9:0]
V=B"0101001000"
incr
reset
9:5
upbitch
V=B"00100"
out[9:0]
diezbit
drawline

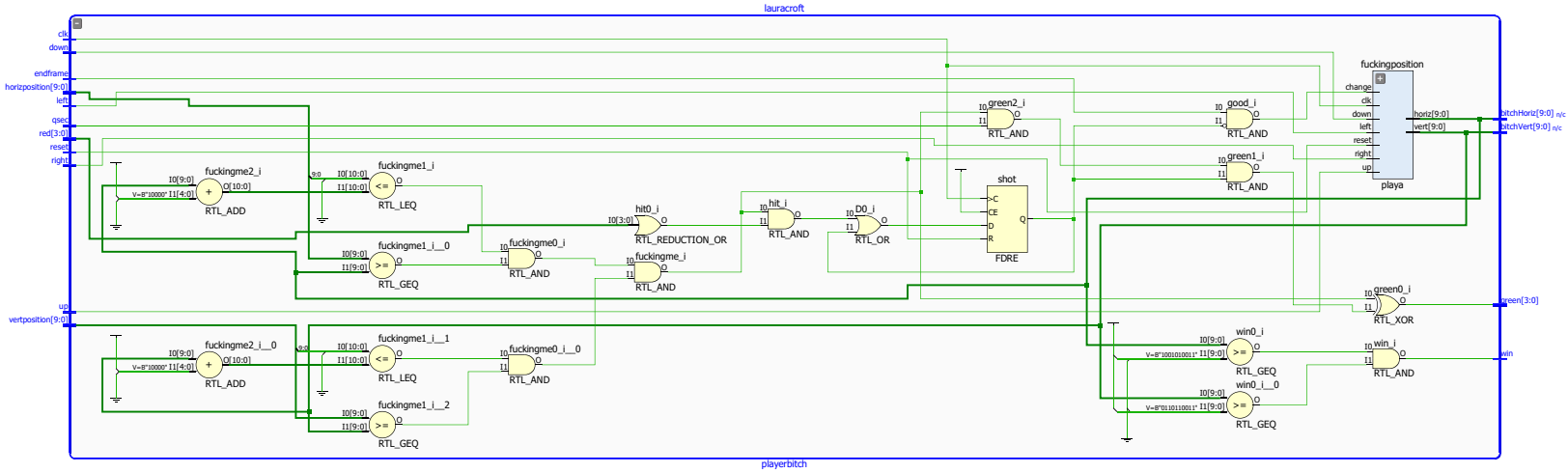

crossing
flash
horizontal[9:0]
qsec
vert[9:0]
blure2_i
I0[9:0]
V="0000001010"
I1[9:0]
<=
O
RTL_LEQ
blure2_i__0
blure1_i
I0
I1
RTL_OR
blure1_i__0
I0[9:0]
V="1001110101"
>=
blure0_i
blure_i
blue2_i
RTL_AND
blue1_i
blue0_i
RTL_XOR
blue[3:0]
red[3:0] n/c
blure0_i__0
V="0111010101"
RTL_GEQ
border

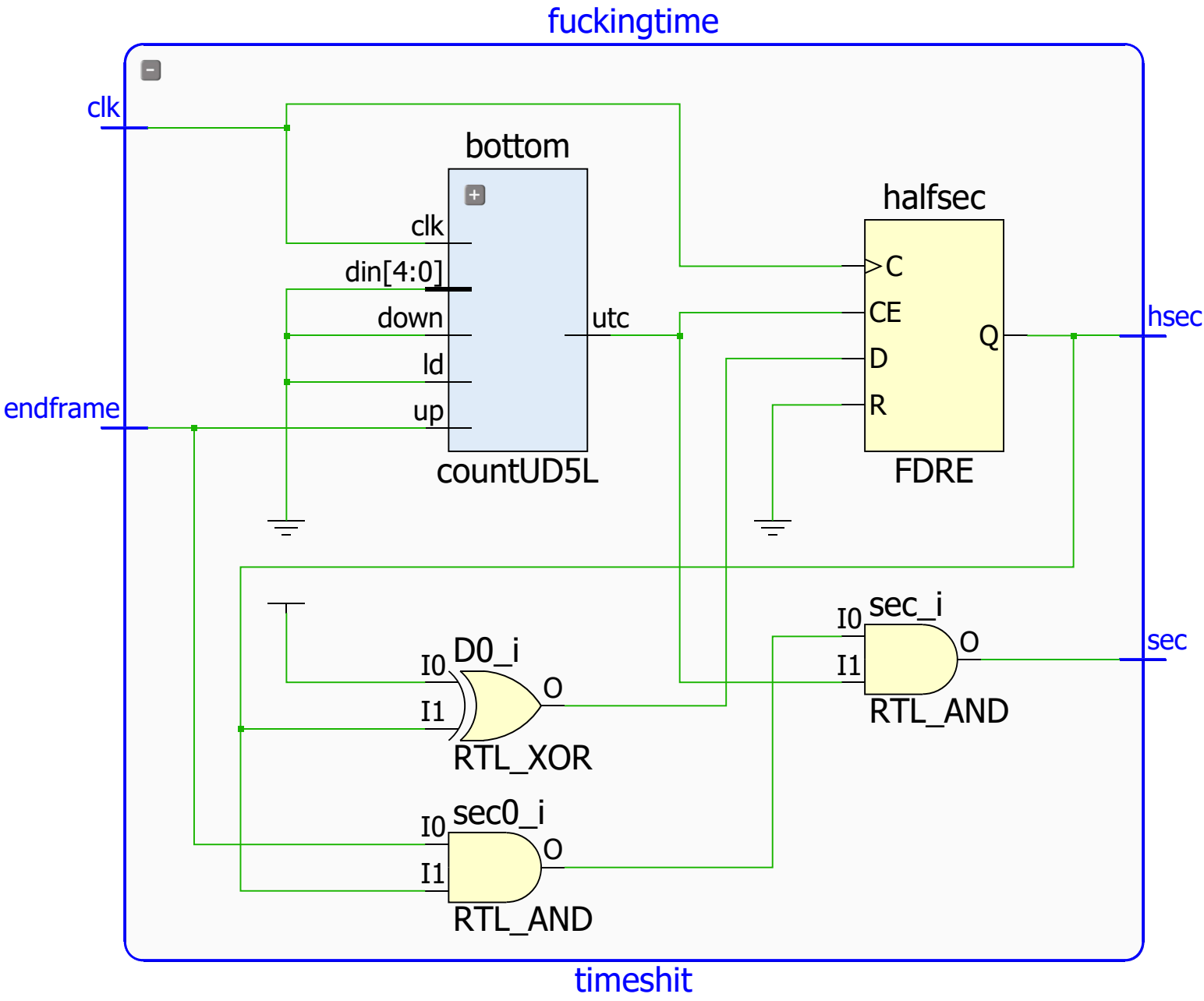

Toplevelshit
btnc
clk
hit
move
win
fblue
gstart
iloss
iwin
reset
showtime
idlein_i
RTL_OR
idlein0_i
RTL_AND
flashin0_i
RTL_OR
idlemydick
FDRE
showtime_i
RTL_INV
playin0_i__0
RTL_INV
playin1_i
RTL_AND
playin0_i
RTL_OR
playin_i
RTL_AND
playme
FDRE
flashbluemen
FDRE
playin3_i
RTL_INV
flashin_i
RTL_AND
playin2_i
RTL_MULT
playin1_i__0
RTL_AND
nothingcame
FDRE
nothingin_i
RTL_AND
RTL_OR
RTL_OR
iloss_i
RTL_AND
iwin_i
RTL_AND
steakmachine

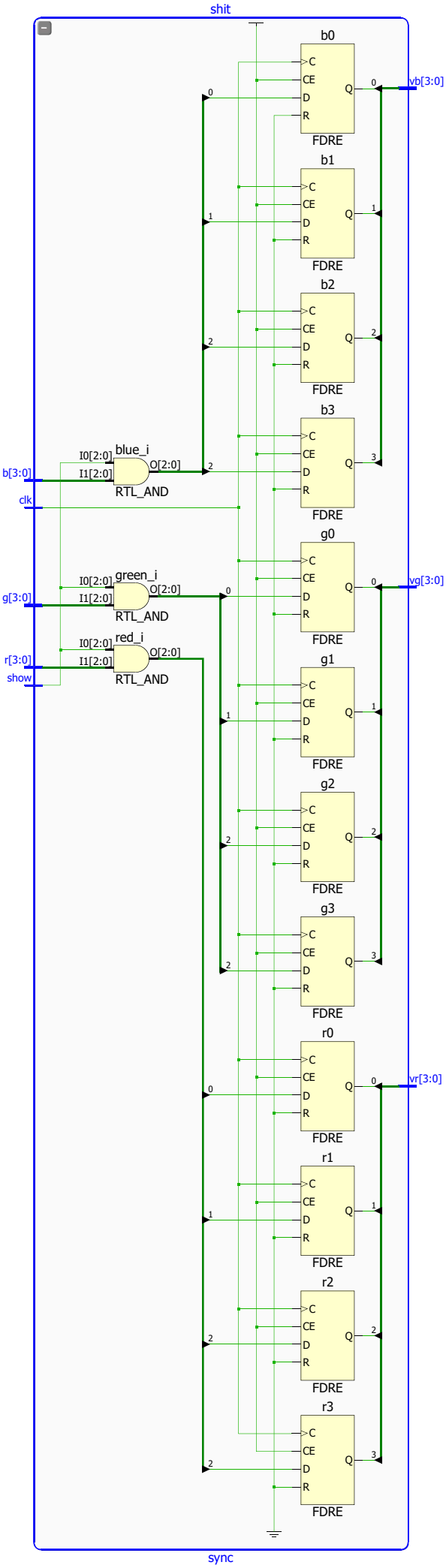


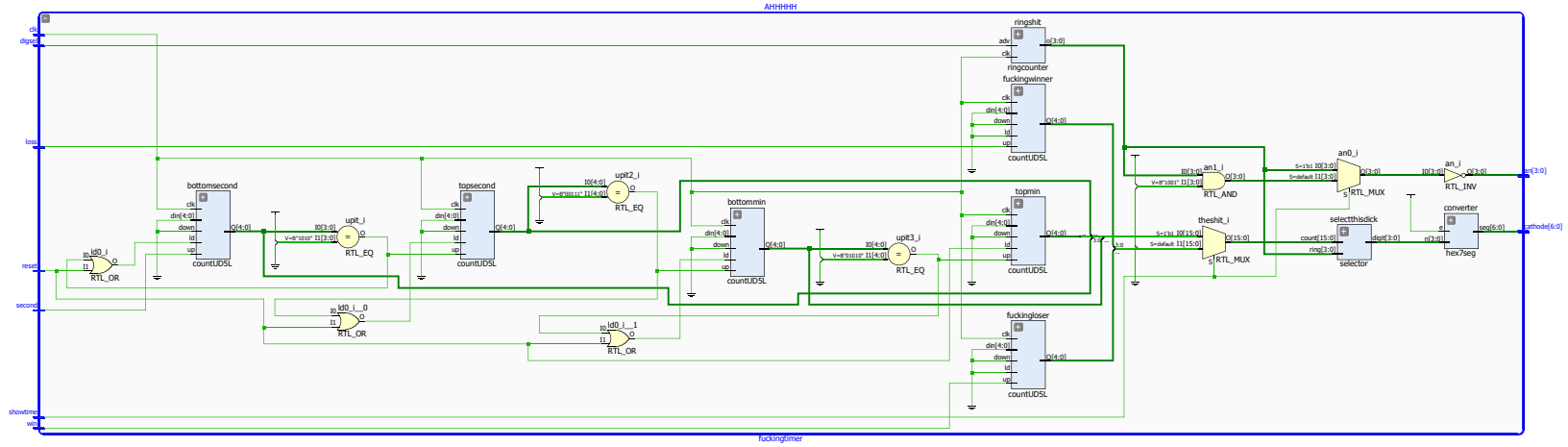

fuckingline6
redstate
movedecr
C
CE
D
R
Q
FDRE
decr
flashme
flash
moveincr
idle
incr
ides
stopin2_i__0
stopin2_i
decrin2_i__0
decrin2_i__1
incrin2_i__0
incrin2_i__1
decrin2_i
incrin2_i
RTL_AND
stopin1_i
decrin1_i
decrin1_i__0
incrin1_i__0
incrin1_i
RTL_OR
stopin0_i
decrin0_i__0
incrin0_i
decrin0_i
RTL_INV
stopin_i
decrin_i
incrin_i
idlein_i
idlein0_i
clk
eleft
eright
gamestart
hit
reset
russia
drawline

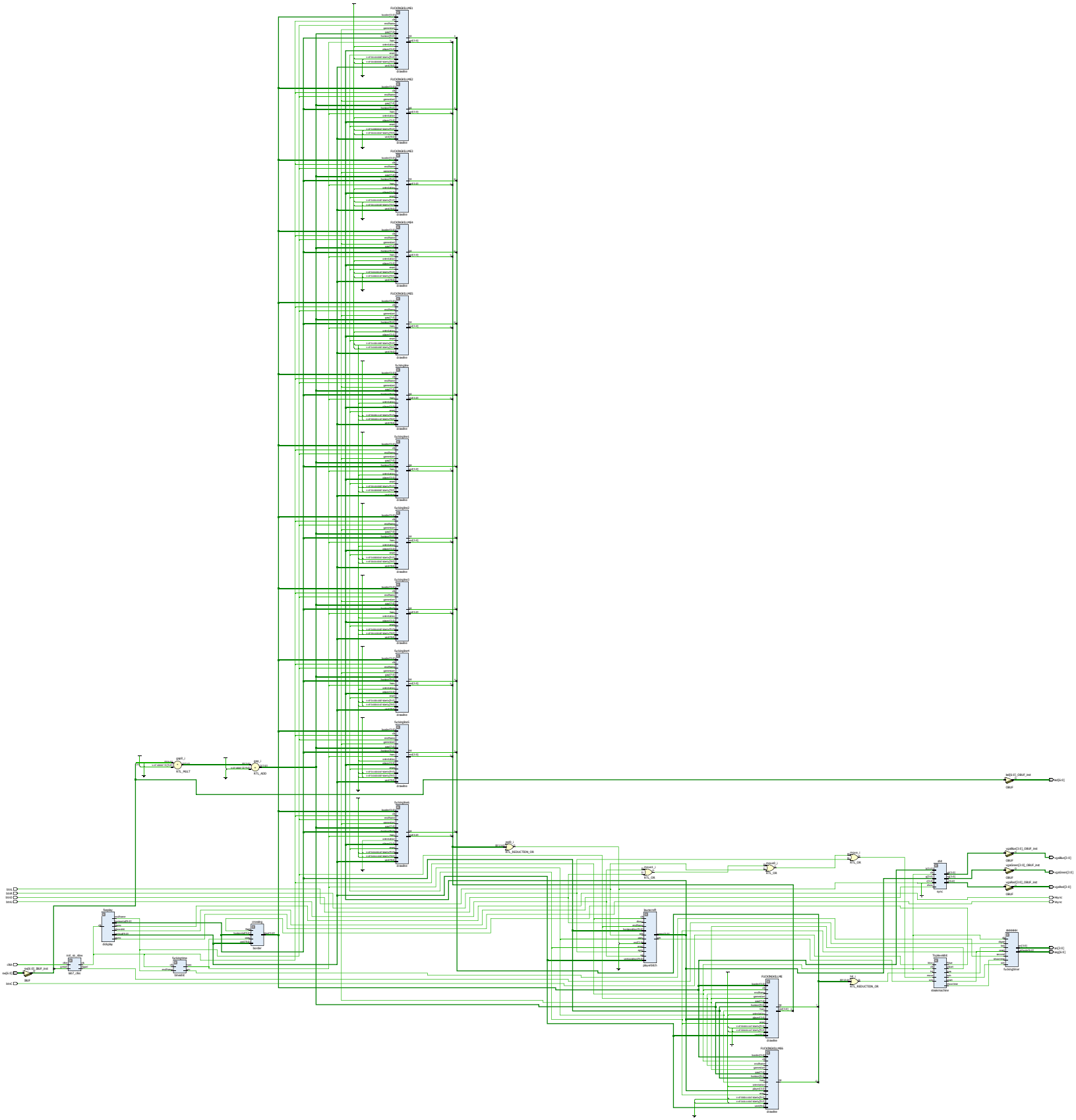

lauracroft
fuckingposition
playa
playerbitch
horiz[9:0]
vert[9:0]
change
clk
down
left
reset
right
up
l0_i__0
I0[9:0]
V=B"0000001011" I1[9:0]
>
RTL_GT
r0_i__0
V=B"1001100011" I1[9:0]
<
RTL_LT
u0_i__0
V=B"0000001011" I1[9:0]
>
RTL_GT
d0_i__0
V=B"0111000100" I1[9:0]
<
RTL_LT
l_i
l0_i
d0_i
r0_i
u0_i
r_i
u_i
d_i
I0
I1
O
RTL_AND
horizon
vertical
clk
decr
V=B"0000001011" in[9:0]
incr
reset
out[9:0]
diezbit

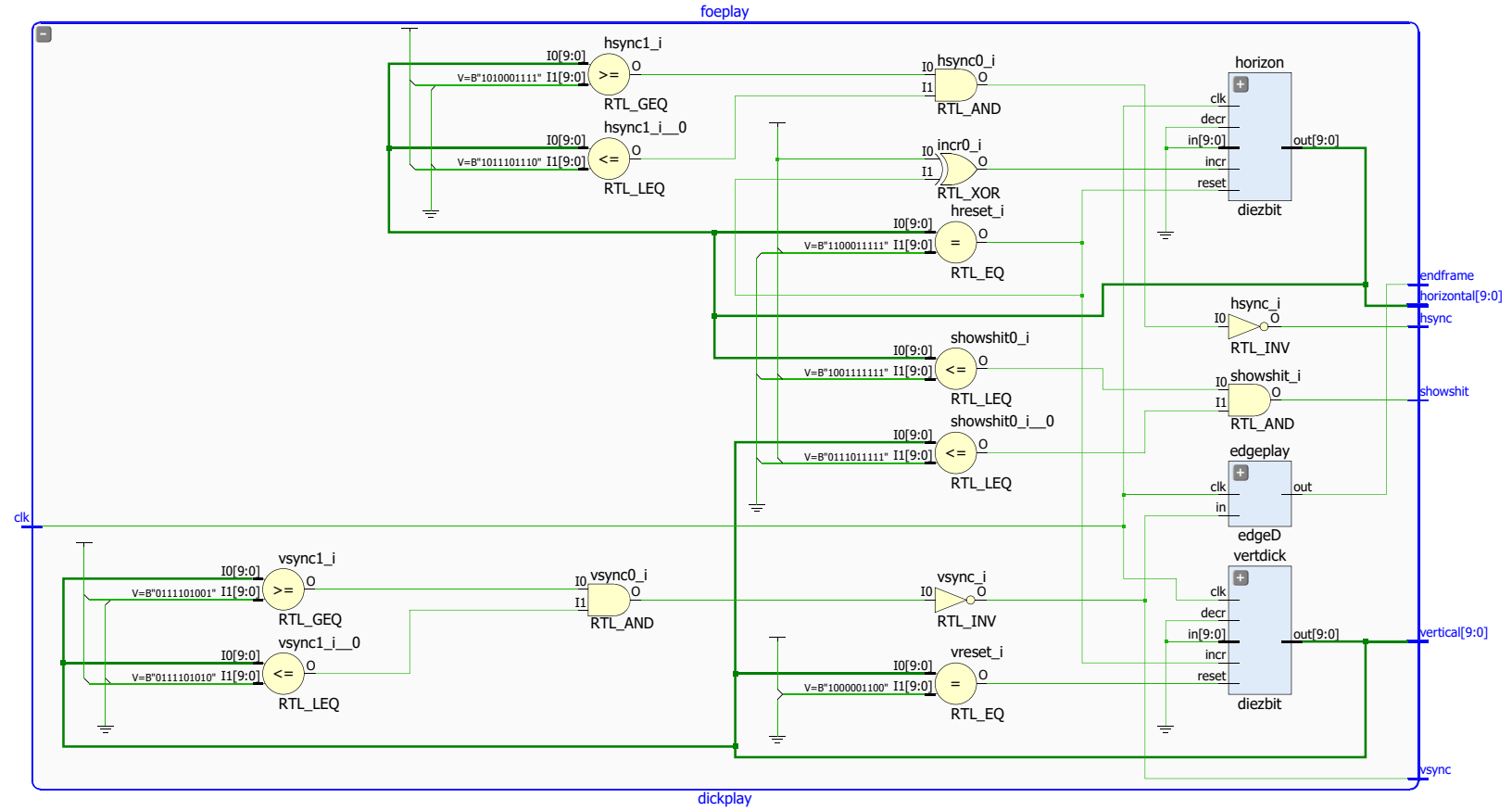

lauracroft
clk
down
endframe
horizposition[9:0]
left
qsec
red[3:0]
reset
right
up
vertposition[9:0]
fuckingme2_i
V=B"10000"
I1[4:0]
O[10:0]
RTL_ADD
fuckingme1_i
I0[10:0]
I1[10:0]
RTL_LEQ
fuckingme1_i_0
I0[9:0]
I1[9:0]
RTL_GEQ
fuckingme0_i
RTL_AND
hit0_i
I0[3:0]
RTL_REDUCTION_OR
fuckingme_i
hit_i
D0_i
RTL_OR
green2_i
shot
C
CE
D
R
Q
FDRE
good_i
green1_i
fuckingposition
change
clk
down
left
reset
right
up
horiz[9:0]
vert[9:0]
playa
green0_i
RTL_XOR
green[3:0]
win0_i
win0_i_0
win_i
win
fuckingme2_i_0
fuckingme1_i_1
fuckingme1_i_2
fuckingme0_i_0
playerbitch


fuckingtime
clk
bottom
clk
din[4:0]
down
ld
up
utc
countUD5L
halfsec
C
CE
D
R
Q
FDRE
hsec
endframe
D0_i
I0
I1
O
RTL_XOR
sec_i
I0
I1
O
RTL_AND
sec
sec0_i
I0
I1
O
RTL_AND
timeshit


shit
b0
C
CE
D
R
Q
FDRE
vb[3:0]
b1
b2
b3
blue_i
I0[2:0]
I1[2:0]
O[2:0]
RTL_AND
b[3:0]
clk
g0
green_i
g[3:0]
vg[3:0]
red_i
r[3:0]
show
g1
g2
g3
r0
vr[3:0]
r1
r2
r3
sync


AHHHHH
clk
digits
load
reset
second
showtime
win
ringshit
ringcounter
fuckingwinner
countUD5L
bottomsecond
countUD5L
upit_i
RTL_EQ
topsecond
countUD5L
upit2_i
RTL_EQ
bottommin
countUD5L
upit3_i
RTL_EQ
topmin
countUD5L
fuckingloser
countUD5L
ld0_i
RTL_OR
ld0_i__0
RTL_OR
ld0_i__1
RTL_OR
an1_i
RTL_AND
an0_i
RTL_MUX
an_i
RTL_INV
theshit_i
RTL_MUX
selectthisdick
selector
converter
hex7seg
an[3:0]
cathode[6:0]
fuckingtimer




foeplay
hsync1_i
I0[9:0]
V=B"1010001111"
I1[9:0]
>=
O
RTL_GEQ
hsync1_i__0
V=B"1011101110"
<=
RTL_LEQ
hsync0_i
I0
I1
RTL_AND
incr0_i
RTL_XOR
hreset_i
V=B"1100011111"
=
RTL_EQ
horizon
clk
decr
in[9:0]
incr
reset
out[9:0]
diezbit
endframe
horizontal[9:0]
hsync
hsync_i
RTL_INV
showshit0_i
V=B"1001111111"
showshit0_i__0
V=B"0111011111"
showshit_i
showshit
edgeplay
out
edgeD
clk
vertdick
vsync1_i
V=B"0111101001"
vsync1_i__0
V=B"0111101010"
vsync0_i
vsync_i
vreset_i
V=B"1000001100"
vertical[9:0]
vsync
dickplay

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 05/22/2018 01:03:36 PM
// Design Name:
// Module Name: border
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

//takes in as input the current screen pixel location
//outputs a rgb bullshit thing if the current pixel is within the border tolerances

module border(
    input [9:0] horizontal,
    input flash,
    input qsec,
    input [9:0] vert,
    output [3:0] blue,
    output [3:0] red
    );
    wire blure;
    //blue is active in this fucking range bitch
    assign blure = horizontal <= 10'd10 | vert <= 10'd10 | horizontal >= 10'd629 | vert >= 10'
d469;
    assign blue = {4{blure ^ (blure & flash & qsec)}};
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 04/26/2018 01:19:40 PM
// Design Name:
// Module Name: countUD5L
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

//fucking flip flop counter
//input the clock cycle to synchinoize flip flops
//up and down inputs are events to change counter
//either count up or down dumbass
//ld is option to load in binary num specified by din
//press up to enable up count, down to enable down count, load to load in din number
//utc is high on all bits equal 1, ie count to next bit bitchass
//dtc is high on all 0, i.e. count down previous bie bitchass
//Q is 5 bit output

module countUD5L(
    input clk, //clock
    input up, //up event
    input down, //down event
    input ld, //load in switch event
    input [4:0] din, //switch preset
    output utc, //shits all one
    output dtc, //shits all 0
    output [4:0] Q //output shit
    );

    wire enable;
    //enable flip flop change when that shit wants to happen
   assign enable = up | down | ld;

   //high on all ones
   assign utc = up & (&Q);
    //high on all zeros and weed
    assign dtc = down & (~(|Q));

    //Q input logic
    //up count and down and overall count input respectively bitch
    wire [4:0] Du, Dd, D;

    //incrimenter logic
    //incriment lowest shit
    assign Du[0] = ~Q[0];
    //incriment 2nd bit shit
    assign Du[1] = Q[0]&~Q[1] | ~Q[0]&Q[1];
    //incriment 3rd shit
    assign Du[2] = Q[2]&(~(&Q[1:0])) | ~Q[2]&(&Q[1:0]);
    //incriment 4th shit
```

```
    assign Du[3] = Q[3]&(~(&Q[2:0])) | ~Q[3]&(&Q[2:0]);
    //incriment 5th shit
    assign Du[4] = Q[4]&(~(&Q[3:0])) | ~Q[4]&(&Q[3:0]);


    //decrimenter logic
    //down on 0 bitch
    assign  Dd[0] = ~Q[0];
    //down on 1 bitch
    assign Dd[1] = ~Q[1]&~Q[0] | Q[1]&Q[0];

    //one if i'm one and theres another one after me, i.e. a one to absorb the subtraction
    // or if were all zero in which case reset to all
    //down on 2 bitch
    assign Dd[2] = Q[2]&(|(Q[1:0])) | ~(|Q[2:0]);
    //down on 3 bitch
    assign Dd[3] = Q[3]&(|(Q[2:0])) | ~(|Q[3:0]);
    //down on 4 bitch
    assign Dd[4] = Q[4]&(|(Q[3:0])) | ~(|Q[4:0]);
    //assign final ooutput as choosing between up or down bus or set switches
    assign D = {5{up}}&Du | {5{down}}&Dd | {5{ld}}&din;

    FDRE #(.INIT(1'b0) ) q0 (.C(clk), .R(1'b0), .CE(enable), .D(D[0]), .Q(Q[0]));
    FDRE #(.INIT(1'b0) ) q1 (.C(clk), .R(1'b0), .CE(enable), .D(D[1]), .Q(Q[1]));
    FDRE #(.INIT(1'b0) ) q2 (.C(clk), .R(1'b0), .CE(enable), .D(D[2]), .Q(Q[2]));
    FDRE #(.INIT(1'b0) ) q3 (.C(clk), .R(1'b0), .CE(enable), .D(D[3]), .Q(Q[3]));
    FDRE #(.INIT(1'b0) ) q4 (.C(clk), .R(1'b0), .CE(enable), .D(D[4]), .Q(Q[4]));
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 05/22/2018 12:44:31 PM
// Design Name:
// Module Name: dickplay
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

//display module
//horizonatal and vertical counter to represent screen pos bitch
//hsync and vsync output high as specified
//endframe high for one clock cycle to show finished rendering one frame

module dickplay(
    input clk,
    output [9:0] horizontal,
    output [9:0] vertical,
    output hsync,
    output vsync,
    output endframe,
    output showshit
    );

    wire hreset, vreset;
    assign showshit = (horizontal <= 639 & vertical <= 479);

    //horizontal counter
    //hsync low between 655, 750
    //reset counter when reach edge at 799
    assign hsync = ~(horizontal >= 655 & horizontal <= 750);
    assign hreset = (horizontal == 799);
    diezbit horizon (.clk(clk), .incr(1'b1 ^ hreset), .decr(1'b0), .in(10'd0), .reset(hreset),
 .out(horizontal));

    //vertical counter
    //vsync low between 655, 750
    //reset counter when reach edge at 799
    assign vsync = ~(vertical >= 489 & vertical <= 490);
    assign vreset = (vertical == 524);
    diezbit vertdick (.clk(clk), .incr(hreset), .decr(1'b0), .in(10'd0), .reset(vreset), .out(
vertical));
    edgeD edgeplay (.clk(clk), .in(vsync), .out(endframe));

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 05/22/2018 12:33:39 PM
// Design Name:
// Module Name: diezbit
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

//10 bit counter
//resets to 0 on reset high

module diezbit(
    input clk,
    input reset,
    input incr,
    input decr,
    input [9:0] in,
    output [9:0] out
    );

    wire upit, downit, up, down;

    //only one input at a time
    assign up = incr & ~ decr;
    assign down = decr & ~ incr;

    countUD5L bottombitch (.clk(clk), .ld(reset), .din(in[4:0]), .down(down),
        .up(up), .utc(upit), .dtc(downit), .Q(out[4:0]));

   countUD5L upbitch (.clk(clk), .ld(reset), .din(in[9:5]), .down(down&downit),
                .up(up&upit), .Q(out[9:5]));
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 05/31/2018 07:29:13 PM
// Design Name:
// Module Name: drawline
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

//input gamestart signal from game state machine
//endframe shit
//orientation to draw type of line
//reset to

module drawline(
    input clk,
    input [7:0] gap, //variable gap length
    input hsec, //half second timer for flash
    input gamestart, //begin movement of lines
    input endframe, //only move on frame ending
    input orientation, // 0 for up down, 1 other shit
    input reset, //reset player and go to idle state
    input [9:0] horizon, //animation horizontal
    input [9:0] vert, //animation vertical
    input [9:0] startx, //beginning position
    input [9:0] starty, //beginning position fucking y
    input [3:0] player, //green line, is active when animation is currently in player area
    input [3:0] border, //same idea as player suppress shit when animation drawing border
    output [3:0] red, //output line
    output hit
    );

    //current point position of line
    wire [9:0] mex, mey;

    //state machine shit
    wire edger, edgel, hitplayer, idle, incr, decr;

    russia redstate (.clk(clk), .reset(reset), .gamestart(gamestart), .eright(edger), .eleft(e
dgel), .hit(hitplayer),
        .idle(idle), .incr(incr), .decr(decr), .flash(hit));


    //position shit
    diezbit horizontalshit (.clk(clk), .reset(idle), .in(startx), .incr(incr & orientation & e
ndframe),
        .decr(decr & orientation & endframe), .out(mex));

    diezbit verticalshit (.clk(clk), .reset(idle), .in(starty), .incr(incr & ~orientation & en
dframe),
        .decr(decr & ~orientation & endframe), .out(mey));
```

```
    //actual line itself
    //just a solid line for base drawing
    //line up is a vertical line, lineright is fucking left to right
    wire lineup, lineright;

    //true for when horizontal is within border width
    assign lineup = horizon <= mex + 10 & horizon > mex & ~orientation;

    //true for when vertical of animation within width of border
    assign lineright = vert <= mey+10 & vert > mey & orientation;


    // in idle
    wire stablevert, stablehoriz;
    //just a stable line
    assign stablehoriz = lineright & idle;
    //just a stable fucking line
    assign stablevert = lineup & idle;



    //compute when it hits the edge
    assign edgel = (mey <= 10) | (mex <= 10);
    assign edger = (mey + gap >= 453 & ~orientation) | (mex + gap  >= 618 & orientation);


    //game is in action draw red
    wire movingvert, movinghoriz;
    //insert gap moving up and down
    assign movingvert = lineup & ~idle & ~((mey < vert) & (mey +gap > vert));
    //insert gap moving left to right
    assign movinghoriz = lineright & ~idle & ~((mex < horizon) & (mex + gap > horizon ));

    wire playa;
    //high when animation line is on player
    assign playa = |player;
    //moving horiz only goes high when animation bit is within the line shit, not including ga
p
    //if player is high and moving vert is high, animation bit is trying to animate both green
 and red
    //ie player green and line red are in same position on screen animation bit
    //ergo it was fucking hit
    assign hitplayer = playa & (movingvert | movinghoriz);

    //red shit active when any of these fuckers is active
    wire rojo = stablehoriz | stablevert | movingvert | movinghoriz;

    //put all that shit together
    assign red = {4{(rojo ^ (rojo & hit & hsec)) & ~|border}};

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 05/31/2018 10:26:06 PM
// Design Name:
// Module Name: fuckingtimer
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


//entire display module
//takes care of time and shit
//just hook up to clk and signals
//output win/loss event for single clock cycle to change counter

module fuckingtimer(
    input clk,
    input digsel,
    input second,
    input reset, //reset timer
    input showtime, //high to show time, low to show score
    input win, //game won, incriment score
    input loss, //game loss incriment fucking loser
    output [3:0] an,
    output [6:0] cathode
    );


    //wires to hold time and score
    wire [15:0] timed, scored;

    //ring counter fucking ring and number to fucking show
    wire [3:0] ring, number;

    //the whole fucking time shit
    wire [15:0] theshit;



    assign theshit = (({16{showtime}} & timed) | {16{~showtime}} & scored);
    assign an = ~((ring & {4{showtime}}) | (ring & 4'b1001 & {4{~showtime}}));

    hex7seg converter (.n(number), .e(1'b1), .seg(cathode));

    ringcounter ringshit (.adv(digsel), .clk(clk), .o(ring));

    selector selectthisdick (.count(theshit), .ring(ring), .digit(number));


    ////////////////////////////////
    /////////// TIME AND SHIT ////////
```

```
    ////////////////////////////////

    wire [4:0] lowersec, highsec, lowmin, highmin;
    wire upit, upit2, upit3;


    //incriment every second
    countUD5L bottomsecond (.clk(clk), .up(second), .down(1'b0), .ld(upit | reset), .din(5'd0)
, .Q(lowersec));

    //incriment when gets to 9 and second hits
    assign upit = (lowersec[3:0] == 4'd10);
    countUD5L topsecond (.clk(clk), .up(upit), .down(1'b0), .ld(upit2 | reset), .din(5'd0), .Q
(highsec));

    //incriment when gets to minute and upit hits
    assign upit2 = (highsec == 5'd7);
    countUD5L bottommin (.clk(clk), .up(upit2), .down(1'b0), .ld(upit3 | reset), .din(5'd0), .
Q(lowmin));

   //you get the fucking idea
    assign upit3 = (lowmin == 5'd10);
    countUD5L topmin (.clk(clk), .up(upit3), .down(1'b0), .ld(1'b0 | reset), .din(5'd0), .Q(hi
ghmin));

    assign timed = {highmin[3:0], lowmin[3:0], highsec[3:0], lowersec[3:0]};


    ////////////////////////////////////
    ////////////// SCORE AND SHIT/////////
    ////////////////////////////////////

    wire [3:0] fuckingwin, fuckingloss;

    wire [15:0] scored;

    countUD5L fuckingloser (.clk(clk), .up(win), .down(1'b0), .ld(1'b0), .din(5'd0), .Q(fuckin
gloss));
    countUD5L fuckingwinner (.clk(clk), .up(loss), .down(1'b0), .ld(1'b0), .din(5'd0), .Q(fuck
ingwin));

    assign scored = {fuckingloss[3:0], 8'd0, fuckingwin[3:0]};

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 05/29/2018 12:14:15 PM
// Design Name:
// Module Name: playa
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


//takes care of player position
//only move player on specific inputs and current state

module playa(
    input clk,
    input up,
    input down,
    input left,
    input right,
    input change,
    input reset,
    output [9:0] horiz,
    output [9:0] vert
    );

    //takes care of horizontal movement
    wire l, r;
    //dictate when moving left is possible
    assign r = right & change & horiz < 611;
    assign l = left & change & horiz > 11;

    diezbit horizon (.clk(clk), .reset(reset), .in(10'd11), .incr(r), .decr(l), .out(horiz));


    //takes care of up down movememnt
    wire u, d;
    //dictate when moving up good
    assign d = down & change & vert < 452;
    assign u = up & change & vert > 11;

    diezbit vertical (.clk(clk), .reset(reset), .in(10'd11), .incr(d), .decr(u), .out(vert));



endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 05/31/2018 12:21:03 AM
// Design Name:
// Module Name: playerbitch
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


//takes care of player
//reset to begining on reset input
//input the buttons, endframe rendering, and current animation pixel position
//also input the current red output and a qsec input that is high for a quarter second
//output the state of the green player and its upper left coordinates

module playerbitch(
    input clk,
    input qsec,
    input up,
    input down,
    input reset,
    input left,
    input right,
    input endframe,
    input [9:0] horizposition,
    input [9:0] vertposition,
    input [3:0] red,
    output [3:0] green,
    output [9:0] bitchHoriz,
    output [9:0] bitchVert,
    output win
    );

    //let player move or not
    wire good, hit, dead, fuckingme;

    //player upper left corner coordinate
    playa fuckingposition (.clk(clk), .up(up), .down(down), .left(left), .right(right),
        .change(good), .reset(reset), .horiz(bitchHoriz), .vert(bitchVert));


   //animation bit of frame actually in player coordinates
   assign fuckingme = (horizposition <= bitchHoriz + 16 & horizposition >= bitchHoriz) &
                   (vertposition <= bitchVert + 16 & vertposition >= bitchVert);

    //player is hit by red shit
    assign hit = fuckingme & |red;

    //keep information that player hit a block and got fucked
    FDRE #(.INIT(1'b0) ) shot (.C(clk), .R(reset), .CE(1'b1), .D(hit|dead), .Q(dead));
```

```
    //can move on endframe signal and player isnt dead
    assign good = endframe & ~dead;

    //green is high if annimation bit is within player coordinates
    assign green = {4{fuckingme ^ (fuckingme & qsec & dead)}};

    assign win = bitchHoriz >= 10'd595 & bitchVert >= 10'd435;

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 05/31/2018 07:13:41 PM
// Design Name:
// Module Name: russia
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

//moore state machine for line movement shit
//all outputs are held high for duration of state

module russia(
    input clk,
    input reset,
    input gamestart,
    input eright,
    input eleft,
    input hit,
    output idle,
    output incr,
    output decr,
    output flash
    );


    //inputs
    wire idlein, incrin, decrin, stopin;


    //idle state
    assign idlein = idle&~gamestart | reset;
    FDRE #(.INIT(1'b1) ) ides (.C(clk), .R(1'b0), .CE(1'b1), .D(idlein), .Q(idle));

    //incriment shit
    assign incrin = (idle&gamestart | decr&eleft | incr&~eright&~hit) & ~reset;
    FDRE #(.INIT(1'b0) ) moveincr (.C(clk), .R(1'b0), .CE(1'b1), .D(incrin), .Q(incr));

    //decriment shit
    assign decrin = (incr&eright&~hit | decr&~eleft&~hit) & ~reset;
    FDRE #(.INIT(1'b0) ) movedecr (.C(clk), .R(1'b0), .CE(1'b1), .D(decrin), .Q(decr));

    assign stopin = (incr&hit | decr&hit | flash) & ~reset;
    FDRE #(.INIT(1'b0) ) flashme (.C(clk), .R(1'b0), .CE(1'b1), .D(stopin), .Q(flash));

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 06/01/2018 12:56:44 AM
// Design Name:
// Module Name: steakmachine
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module steakmachine(
    input clk,
    input hit,
    input move,
    input win,
    input btnc,
    output gstart,
    output showtime,
    output reset,
    output iloss,
    output iwin,
    output fblue
    );

    //output lines
    wire idle, play, nothing;

    //input logic
    wire idlein, playin, nothingin, flashin;

    //input states
    assign idlein = btnc | idle&~move;
    assign playin = (idle&move | play*~hit&~win)&~btnc;
    assign nothingin = (play&hit | nothing)&~btnc;
    assign flashin = (play&win | fblue)&~btnc;

    //output lines
    assign gstart = move;
    assign showtime = ~idle;
    assign reset = idle;
    assign iloss = play&hit;
    assign iwin = play&win;

    FDRE #(.INIT(1'b1) ) idlemydick (.C(clk), .R(1'b0), .CE(1'b1), .D(idlein), .Q(idle));

    FDRE #(.INIT(1'b0) ) playme (.C(clk), .R(1'b0), .CE(1'b1), .D(playin), .Q(play));

    FDRE #(.INIT(1'b0) ) nothingcame (.C(clk), .R(1'b0), .CE(1'b1), .D(nothingin), .Q(nothing)
);

    FDRE #(.INIT(1'b0) ) flashbluemen (.C(clk), .R(1'b0), .CE(1'b1), .D(flashin), .Q(fblue));
```

```
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 05/22/2018 01:11:47 PM
// Design Name:
// Module Name: sync
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

//synchronizes all input for display shit

module sync(
    input [3:0] r,
    input [3:0] g,
    input [3:0] b,
    input show,
    output [3:0] vr,
    output [3:0] vg,
    output [3:0] vb,
    input clk
    );

    wire [3:0] red, green, blue;

    //show shit filter
    assign red = {4{show}} & r;
    assign green = {4{show}} & g;
    assign blue = {4{show}} & b;


    //sync red shit
    FDRE #(.INIT(1'b0) ) r0 (.C(clk), .R(1'b0), .CE(1'b1), .D(red[0]), .Q(vr[0]));
    FDRE #(.INIT(1'b0) ) r1 (.C(clk), .R(1'b0), .CE(1'b1), .D(red[1]), .Q(vr[1]));
    FDRE #(.INIT(1'b0) ) r2 (.C(clk), .R(1'b0), .CE(1'b1), .D(red[2]), .Q(vr[2]));
    FDRE #(.INIT(1'b0) ) r3 (.C(clk), .R(1'b0), .CE(1'b1), .D(red[2]), .Q(vr[3]));


    //sync green shit
    FDRE #(.INIT(1'b0) ) g0 (.C(clk), .R(1'b0), .CE(1'b1), .D(green[0]), .Q(vg[0]));
    FDRE #(.INIT(1'b0) ) g1 (.C(clk), .R(1'b0), .CE(1'b1), .D(green[1]), .Q(vg[1]));
    FDRE #(.INIT(1'b0) ) g2 (.C(clk), .R(1'b0), .CE(1'b1), .D(green[2]), .Q(vg[2]));
    FDRE #(.INIT(1'b0) ) g3 (.C(clk), .R(1'b0), .CE(1'b1), .D(green[2]), .Q(vg[3]));


    //sync blue shit
    FDRE #(.INIT(1'b0) ) b0 (.C(clk), .R(1'b0), .CE(1'b1), .D(blue[0]), .Q(vb[0]));
    FDRE #(.INIT(1'b0) ) b1 (.C(clk), .R(1'b0), .CE(1'b1), .D(blue[1]), .Q(vb[1]));
    FDRE #(.INIT(1'b0) ) b2 (.C(clk), .R(1'b0), .CE(1'b1), .D(blue[2]), .Q(vb[2]));
    FDRE #(.INIT(1'b0) ) b3 (.C(clk), .R(1'b0), .CE(1'b1), .D(blue[2]), .Q(vb[3]));
```

```
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 05/31/2018 10:56:55 AM
// Design Name:
// Module Name: timeshit
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


//endframe is from vga counter
//60hz refresh rate = counter for 60 end frames
//hsec is high/low for a full half second
//sec is high for only single clock cycle only every second

module timeshit(
    input endframe,
    input clk,
    output hsec,
    output sec
    );

    //high for one clock cycle once frame buffer = 32
    //ie high for one clock cycle every half second
    wire tick, hsec;
    countUD5L bottom (.clk(clk), .up(endframe), .down(1'b0), .ld(1'b0),
        .din(5'b00000), .utc(tick));

    //one extra bit that will toggle itself only on half second
    //FDRE #(.INIT(1'b0) ) halfsec (.C(clk), .R(1'b0), .CE(tick), .D(1'b1 ^ hsec), .Q(hsec));
    FDRE #(.INIT(1'b0) ) halfsec (.C(clk), .R(1'b0), .CE(tick), .D(1'b1 ^ hsec), .Q(hsec));


    //sec is high if hsec is rolling over
    assign sec = endframe & hsec & tick;

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 05/22/2018 12:56:48 PM
// Design Name:
// Module Name: topoffmydick
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


 module topoffmydick(
    input clkin,
    input btnC,
    input btnU,
    input btnL,
    input btnR,
    input btnD,
    input [6:0] sw,
    output [6:0] seg,
    output [3:0] an,
    output [3:0] vgaRed,
    output [3:0] vgaBlue,
    output [3:0] vgaGreen,
    output [6:0] led,
    output Hsync,
    output Vsync
    );

    //create qsec timer
    wire hsec, reset, sec;

    //gap width of lines
    wire [7:0] gap;

    //fucking statemachine shit
    wire showtime, reset, move, gs, hit, win, loss, incrwin, flashblue;

    //animation bits
    wire Horizsync, VertSync, endframe, showshit;

    //vga shit
    wire [3:0] red, green, blue;

    //screen position shit
    wire [9:0] horizon, vert;


    ////////////////////////////////
    //////// OVERHEAD //////////////
    ////////////////////////////////
```

```
    //reset game to initial cond if btnC is pressed
    assign led = sw;
    assign gap = {4'd0, sw[6:4]} * 7'd32 + 7'd16;
    //clock module
    wire clk, digsel;
    lab7_clks not_so_slow (.clkin(clkin), .greset(sw[0]), .clk(clk), .digsel(digsel));

    //syncs pixel showing
    sync shit (.clk(clk), .r(red), .g(green), .b(blue), .vr(vgaRed), .vb(vgaBlue), .vg(vgaGree
n), .show(showshit));

    timeshit fuckingtime (.clk(clk), .endframe(endframe), .hsec(hsec), .sec(sec));


    ////////////////////////////
    //////// STATE MACHINE //////
    ////////////////////////////
    assign move = btnU | btnD | btnL | btnR;

    steakmachine Toplevelshit (.clk(clk), .hit(hit), .move(move), .win(win), .btnc(btnC),
        .gstart(gs), .showtime(showtime), .reset(reset), .iloss(loss), .iwin(incrwin), .fblue(
flashblue));



    ////////////////////////////
    //////// SCREEN ANIMATION////
    ////////////////////////////


    //gives pixel position
    dickplay foeplay (.clk(clk), .horizontal(horizon), .vertical(vert), .hsync(Hsync),
        .vsync(Vsync), .showshit(showshit), .endframe(endframe));



    ////////////////////////////////
    ////// BORDER INFORMATION ///////
    ////////////////////////////////

    //show blue border
    //drive correct lines high on specific inputs
    border crossing (.horizontal(horizon), .qsec(hsec), .flash(flashblue), .vert(vert), .blue(
blue));


    ////////////////////////////////
    ////// TIMER SHIT //////////////
    ////////////////////////////////

    fuckingtimer AHHHHH (.clk(clk), .digsel(digsel), .second(sec), .reset(reset), .showtime(sh
owtime),
        .win(incrwin), .loss(loss), .an(an), .cathode(seg));

    ////////////////////////////////
    ////// BARRIER SHIT //////////////
    ////////////////////////////////

    wire [13:0] redoctober, gethit;

    assign red = {3{|redoctober}};
    //for the state machine
    assign hit = |gethit;
```

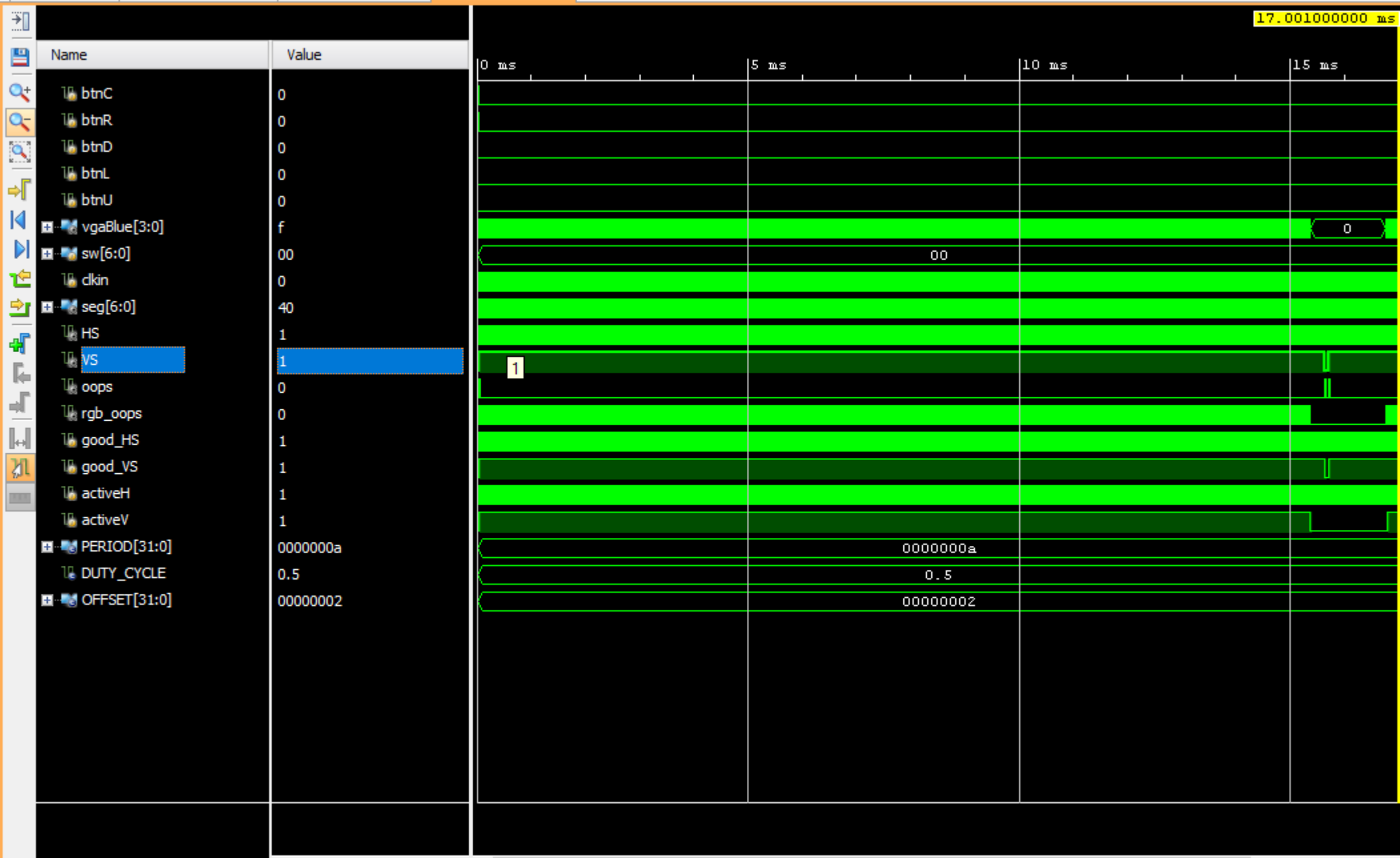
```
    //vertical

    drawline fuckingline (.clk(clk), .hsec(hsec), .gamestart(gs), .gap(gap), .endframe(endfram
e),
        .orientation(1'b0), .reset(reset), .horizon(horizon), .vert(vert), .startx(10'd158),
        .starty(10'd78), .player(green), .border(blue), .red(redoctober[0]), .hit(gethit[0]));

    drawline fuckingline1 (.clk(clk), .hsec(hsec), .gamestart(gs), .gap(gap), .endframe(endfra
me),
        .orientation(1'b0), .reset(reset), .horizon(horizon), .vert(vert), .startx(10'd208),
        .starty(10'd128), .player(green), .border(blue), .red(redoctober[1]), .hit(gethit[1]))
;

    drawline fuckingline2 (.clk(clk), .hsec(hsec), .gamestart(gs), .gap(gap), .endframe(endfra
me),
        .orientation(1'b0), .reset(reset), .horizon(horizon), .vert(vert), .startx(10'd258),
        .starty(10'd178), .player(green), .border(blue), .red(redoctober[2]), .hit(gethit[2]))
;


    drawline fuckingline3 (.clk(clk), .hsec(hsec), .gamestart(gs), .gap(gap), .endframe(endfra
me),
        .orientation(1'b0), .reset(reset), .horizon(horizon), .vert(vert), .startx(10'd308),
        .starty(10'd228), .player(green), .border(blue), .red(redoctober[4]), .hit(gethit[3]))
;


    drawline fuckingline4 (.clk(clk), .hsec(hsec), .gamestart(gs), .gap(gap), .endframe(endfra
me),
        .orientation(1'b0), .reset(reset), .horizon(horizon), .vert(vert), .startx(10'd408),
        .starty(10'd278), .player(green), .border(blue), .red(redoctober[5]), .hit(gethit[4]))
;


    drawline fuckingline5 (.clk(clk), .hsec(hsec), .gamestart(gs), .gap(gap), .endframe(endfra
me),
        .orientation(1'b0), .reset(reset), .horizon(horizon), .vert(vert), .startx(10'd458),
        .starty(10'd328), .player(green), .border(blue), .red(redoctober[6]), .hit(gethit[5]))
;


    drawline fuckingline6 (.clk(clk), .hsec(hsec), .gamestart(gs), .gap(gap), .endframe(endfra
me),
        .orientation(1'b0), .reset(reset), .horizon(horizon), .vert(vert), .startx(10'd358),
        .starty(10'd378), .player(green), .border(blue), .red(redoctober[7]), .hit(gethit[6]))
;



      //horizontal

      drawline FUCKINGKILLME (.clk(clk), .hsec(hsec), .gamestart(gs), .gap(gap), .endframe(end
frame),
          .orientation(1'b1), .reset(reset), .horizon(horizon), .vert(vert), .startx(10'd158),
          .starty(10'd78), .player(green), .border(blue), .red(redoctober[8]), .hit(gethit[7])
);

      drawline FUCKINGKILLME1 (.clk(clk), .hsec(hsec), .gamestart(gs), .gap(gap), .endframe(en
dframe),
          .orientation(1'b1), .reset(reset), .horizon(horizon), .vert(vert), .startx(10'd208),
          .starty(10'd128), .player(green), .border(blue), .red(redoctober[9]), .hit(gethit[8]
));
```

```
      drawline FUCKINGKILLME2 (.clk(clk), .hsec(hsec), .gamestart(gs), .gap(gap), .endframe(en
dframe),
          .orientation(1'b1), .reset(reset), .horizon(horizon), .vert(vert), .startx(10'd258),
          .starty(10'd178), .player(green), .border(blue), .red(redoctober[10]), .hit(gethit[9
]));


      drawline FUCKINGKILLME3 (.clk(clk), .hsec(hsec), .gamestart(gs), .gap(gap), .endframe(en
dframe),
          .orientation(1'b1), .reset(reset), .horizon(horizon), .vert(vert), .startx(10'd308),
          .starty(10'd228), .player(green), .border(blue), .red(redoctober[11]), .hit(gethit[1
0]));


      drawline FUCKINGKILLME4 (.clk(clk), .hsec(hsec), .gamestart(gs), .gap(gap), .endframe(en
dframe),
          .orientation(1'b1), .reset(reset), .horizon(horizon), .vert(vert), .startx(10'd358),
          .starty(10'd278), .player(green), .border(blue), .red(redoctober[12]), .hit(gethit[1
1]));


      drawline FUCKINGKILLME5 (.clk(clk), .hsec(hsec), .gamestart(gs), .gap(gap), .endframe(en
dframe),
          .orientation(1'b1), .reset(reset), .horizon(horizon), .vert(vert), .startx(10'd408),
          .starty(10'd328), .player(green), .border(blue), .red(redoctober[13]), .hit(gethit[1
2]));


      drawline FUCKINGKILLME6 (.clk(clk), .hsec(hsec), .gamestart(gs), .gap(gap), .endframe(en
dframe),
          .orientation(1'b1), .reset(reset), .horizon(horizon), .vert(vert), .startx(10'd58),
          .starty(10'd378), .player(green), .border(blue), .red(redoctober[14]), .hit(gethit[1
3]));


    ////////////////////////////////
    ////// PLAYER INFORMATION ///////
    ////////////////////////////////

    //player position
    wire [9:0] playerX, playerY;

    playerbitch lauracroft (.clk(clk), .qsec(hsec), .up(btnU), .down(btnD), .left(btnL), .righ
t(btnR),
        .reset(reset), .endframe(endframe), .horizposition(horizon), .vertposition(vert), .red
(red),
        .green(green), .bitchHoriz(playerX), .bitchVert(playerY), .win(win));

endmodule
```

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	18.108 ns	Worst Hold Slack (WHS):	0.124 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:	638	Total Number of Endpoints:	638	Total Number of Endpoints:	365

All user specified timing constraints are met.


Name
Value
btnC
0
btnR
0
btnD
0
btnL
0
btnU
0
vgaBlue[3:0]
f
sw[6:0]
00
clkin
0
seg[6:0]
40
HS
1
VS
1
oops
0
rgb_oops
0
good_HS
1
good_VS
1
activeH
1
activeV
1
PERIOD[31:0]
0000000a
DUTY_CYCLE
0.5
OFFSET[31:0]
00000002
17.001000000 ms
0 ms
5 ms
10 ms
15 ms
0
00
1
0000000a
0.5
00000002

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

Lab 7

Playing Some Game Shit on VGA

- All signals need to be synced
  - use the flip flops
  - pass output through flip flops synchronized enable
  - edge detect vsync to keep track of vsync

Maze run game

VGA display method:

2 counters
- horizontal 0-800
- vertical 0-525

active range 0-640, 0-480

off counters

counters correspond to pixel location

horizontal counter resets, set to zero, during hsync low

range 655-750

- vertical counter resets to 0 during vsync low

range 489-490

Counter module

10 bits wide for horizontal - counts to 0 to 799
- reset @ 800

10 bits for vertical
- reset when @ 525
- increment when horizontal @ 800

10 bit counter module
- reset option

horizontal count:
- show bit in range 0-639
- hsync @ 655-750
- reset @ 799
- increment on clock

vertical count
- show bit in range 0-479
- hsync @ 489-491
- reset @ 524
- increment on horizontal @ end

Player.

- Single counter to hold position of upper left corner
- increment horizontal & vertical counter on button input

- only change on endframe input goes high

- move left if
  left button & change high & current horizontal > 10

- move right if
  right button & change high & current horizontal < 630

- move up if
  up button & change & vertical > 10

- move down if
  down & change & vert < 470

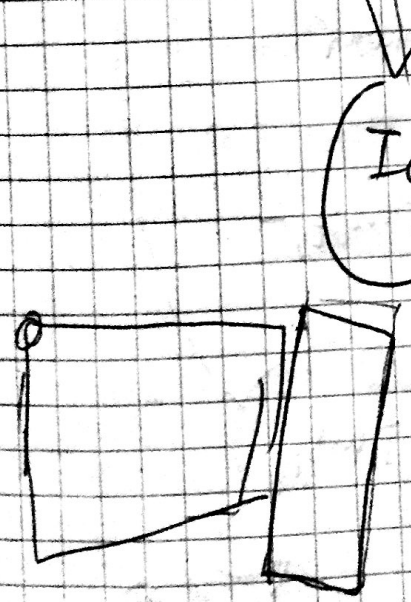
adjust for player size

Player top:

input
- draw square 16 to right & below player position
- take in button inputs
  - up down left right
- take in red bgu output

output
- output green line & player position

if player more if endframe hi yh & no red
- If current drawing pixel is between boundaries OR player & red is high → marks a hit



if current animation position is within player

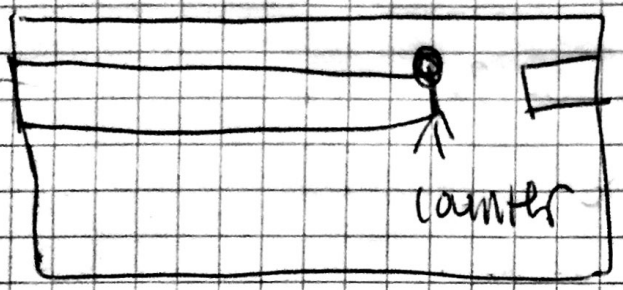
hori2 < animation < hi yhori2 &
hi yhright < animation < hi yhright

! 2 pairs of info
- Player coordinate
- current animation position

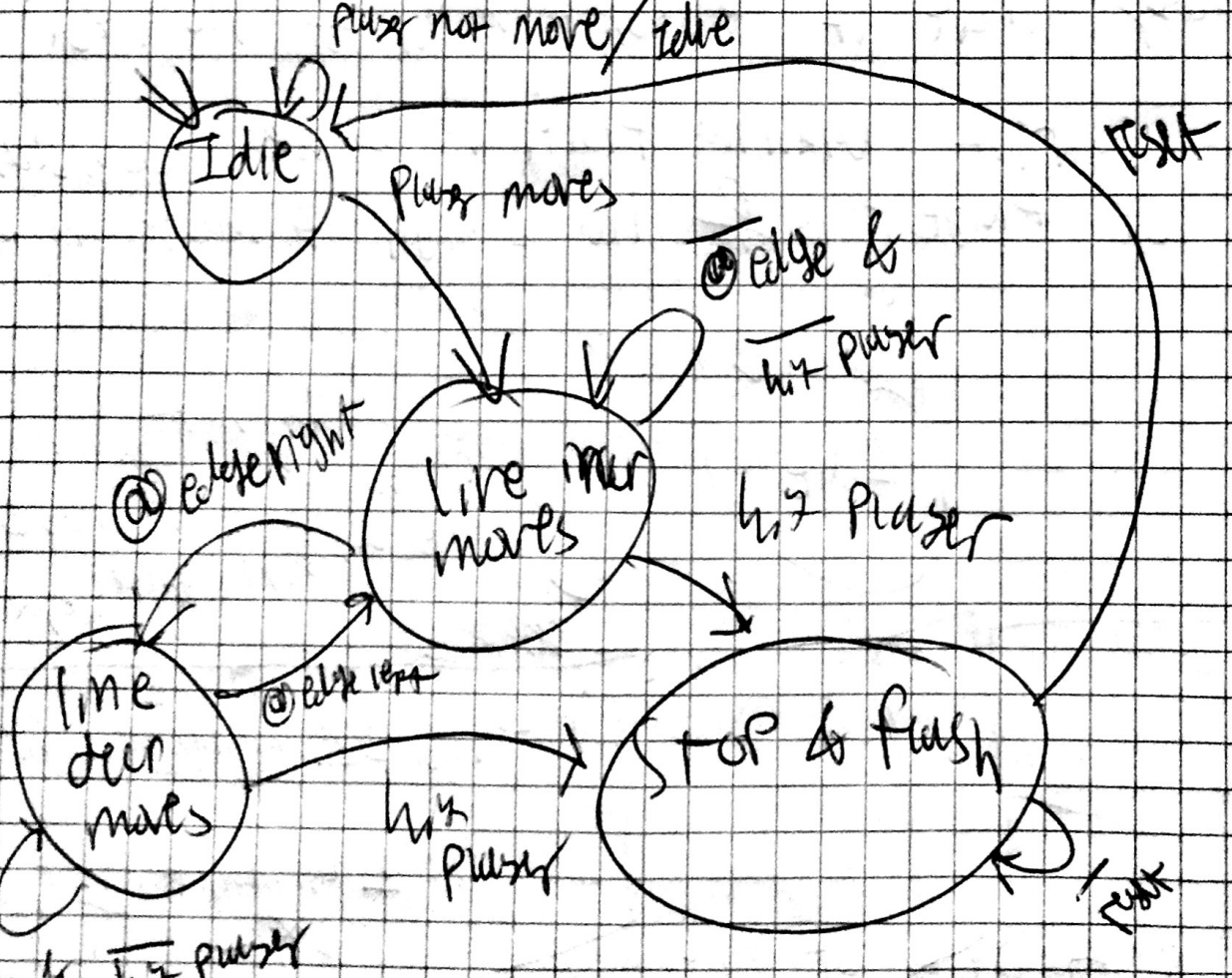
make 9sec timer

Obstacles

- counter to keep track of left side gap bounds



- either increment or decrement



PS	NS	input gs	input @edge left	input @edge right	input hit	output reset mode	output incr	output decr	output flash
Idle	Idle	0	—	0	—	—	1	0	0	0
Idle	incr move	1	—	0	—	—	0	1	0	0
incr move	decr incr move	—	—	0	0	0	0	1	0	0
incr move	decr move	—	+	0	1	0	0	0	0	0
Incr move	Sbf	—	-	0	—	1	0	0	0	1
decr	decr	—	0	0	—	0	0	0	1	0
decr	Sbf	—	—	0	—	1	0	0	0	1
decr	decr incr	—	1	0	—	0	0	0	0	0
Sbf	Sbf	—	—	0	—	—	0	0	0	0
Sbf	Idle	—	—	1	—	—	1	0	0	0

Idle in = Idle & $\overline{gs}$ + reset

incr in = (Idle & gs + decr & $\overline{edge_{left}}$) $\overline{reset}$ + incr & $\overline{edge_{right}}$ & $\overline{hit}$

decr in = (incr $edge_{right}$ $\overline{hit}$ + decr $\overline{edge_{left}}$ $\overline{hit}$) $\overline{reset}$

Sbf = (Incr hit + decr · hit + Sbf) $\overline{reset}$

Inputs

game start @ edge left @ edge right hit pass

Line Shift

Input
- Start pos (x, y)
- Game start
- Player pos (x, y)

Output
- red

- Counter for x
- Counter for y

- Collision when Player hits

- given current pos
- Show red
  - 5 below & all the way to left & right
    - blue horiz
- To change blue left horiz
  - after show red when animation
    - switches vert
    - switches horiz

- reset
- vert
- 1 for vertical
- 0 for horizontal

Borders of game

10 < horiz < 614

10 < vert < 453

Idle

animation [redacted] < mey + 5 & animation vert > mey

vert

Stable & Idle [redacted] & orientation

↑ Constant time

↑ draw const line

↑ make it horizontal or vertical

- draw stable line ✓
- draw moving line w/ gap ✓
- compute edge hit ✓
- compute player hit ✓

Compute edge h,7

topOut	bottomOut
mag >= 10	mag == 1e53
leftOut	rightOut
mag >= 10	mag == 614

edge left

↓

topOut & orientation

or

leftOut & ~~orientation~~

edge right

↓

bottomOut & orientation

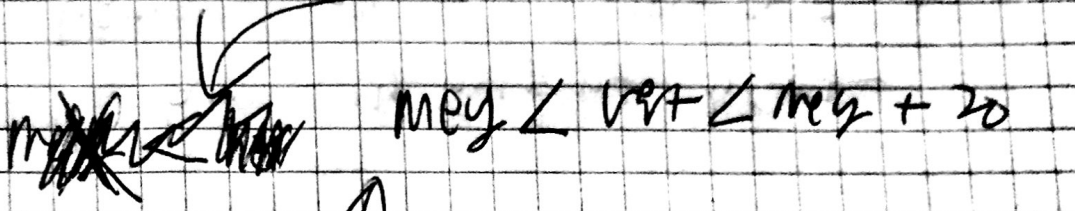
or

rightOut & ~~orientation~~

moving gap

have gap move along stable line

vert gap = if vertical animation within range



make gap

horz gap = if horz and within range

mex < horz < mex + 30

Compare player hit

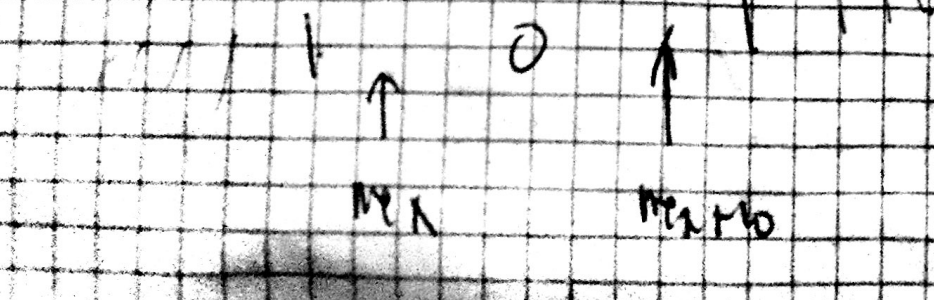
- player green line

- green line is only high when animation box is on player
  - if green line & me line is high
    - player fucking hit me

Show red

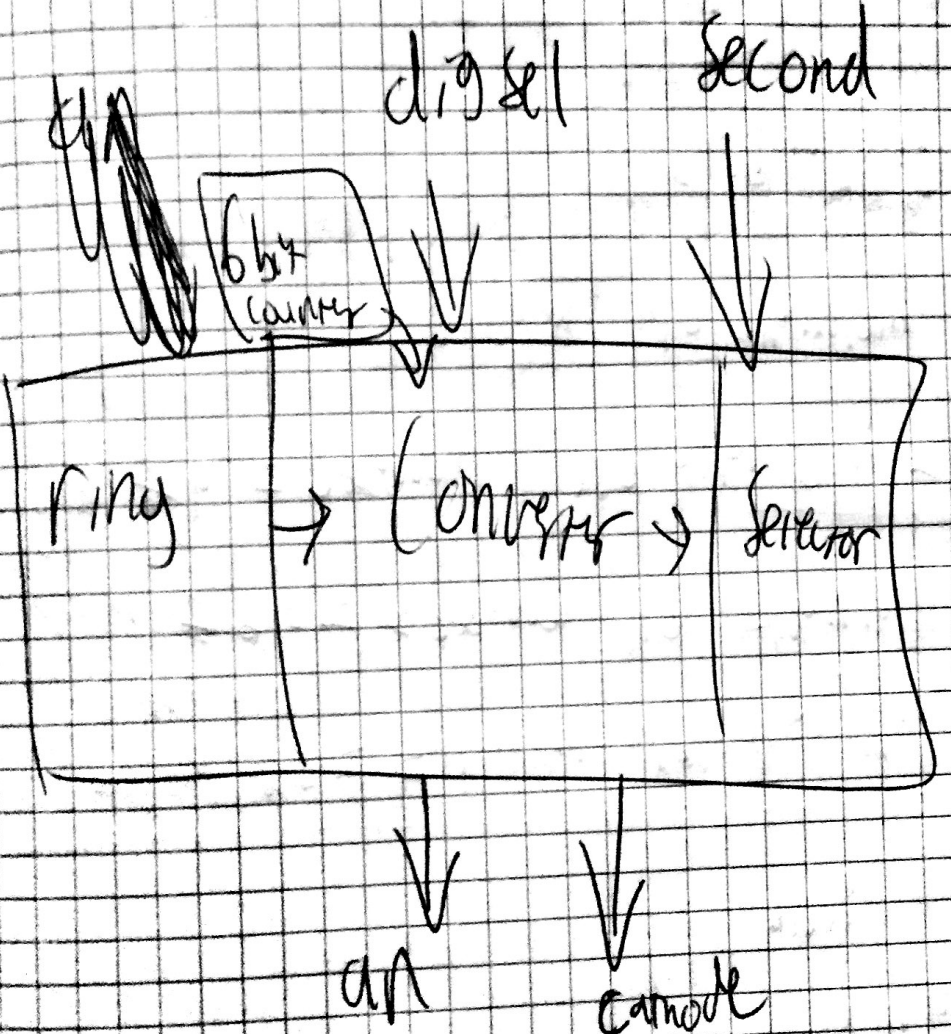
red = (Stube (x,y) or manny (x,y)) & (push & (half sec))

output $sw[6:4] x 32 + 6

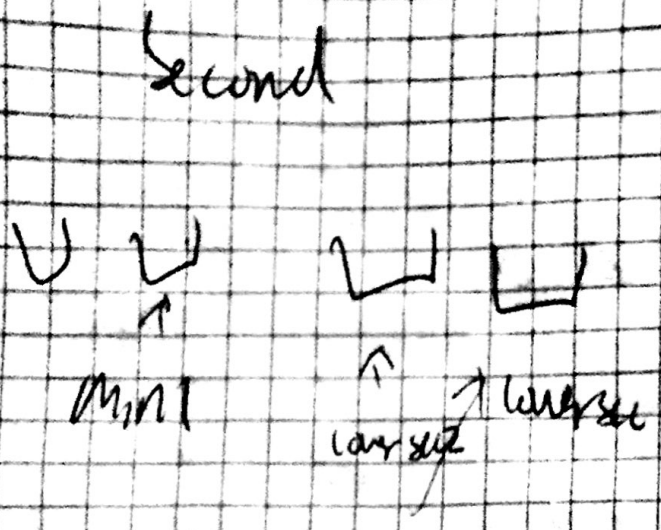


Timer

- 6 bit counter
- increments every second



my short memo



all 4 bits

lowsec increment on second

lowsec2 increment on lowsec = 9 & second

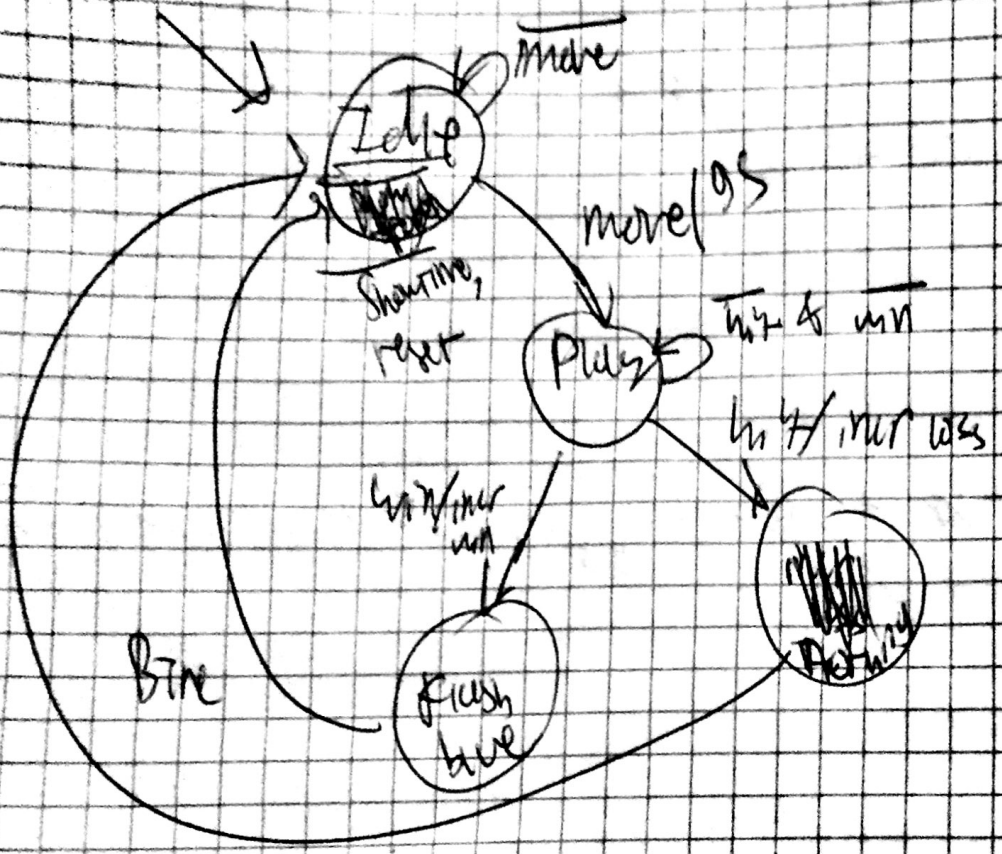
min1 = increment on lowsec2 == 6 & inc

min2 = increment on min1 == 6 & inc

Score

4 bits

Game State machine



BTNC

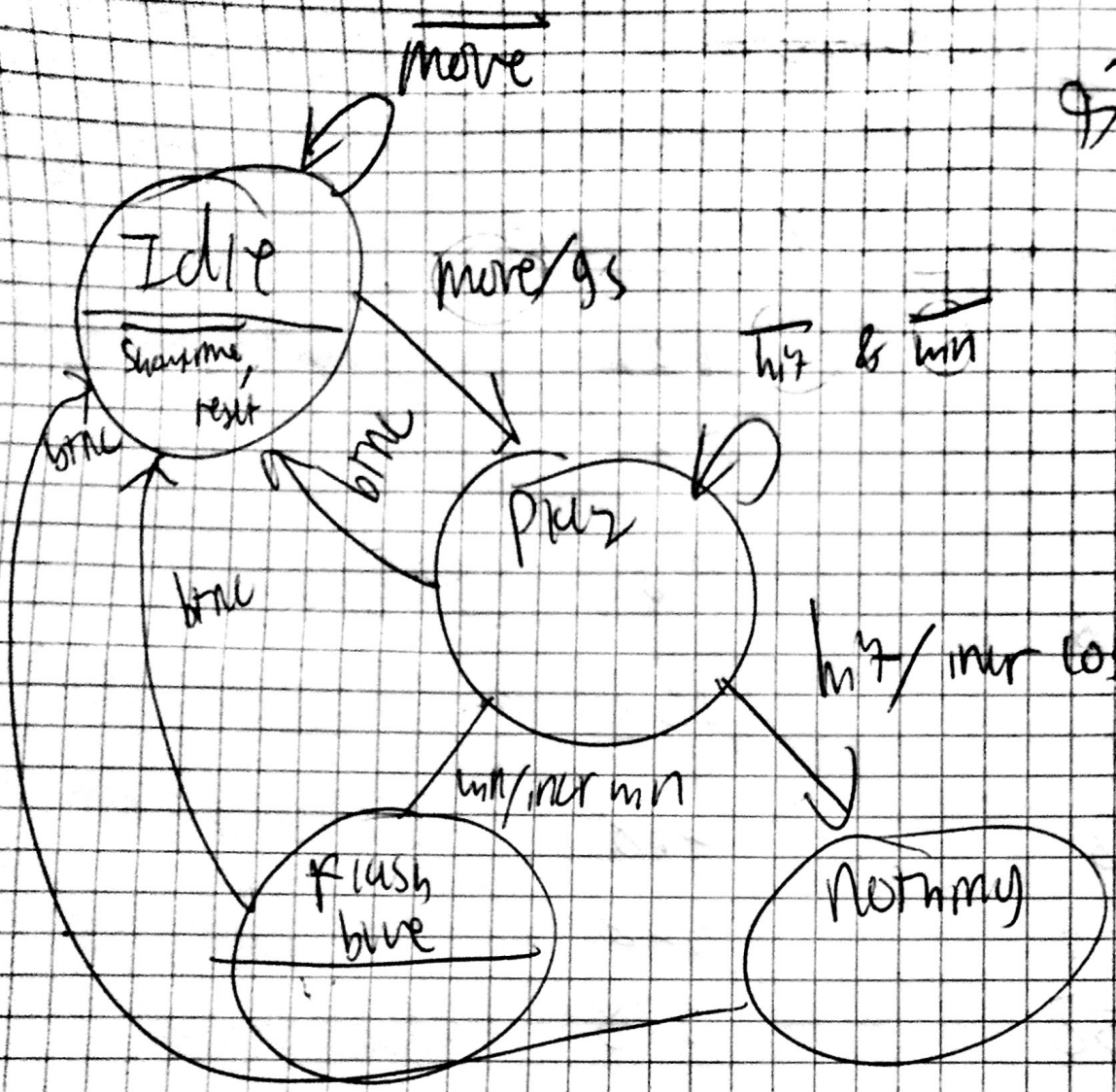
PS	input: move	hit BTNC win	NS	output: shooting reset init
Idle				

Vertical Coordinates

X Coordinate	Y Coordinate
158	78
208	128
258	178
308	228
408	278
458	328
358	378

Horizontal

Y cord	
78	158
128	208
178	258
228	308
278	408
328	458
378	358



hit ✓
move ✓
inv ✓
bmt

inv = bizenhoriz >= 595 & bizenvert >= 435

PS	input: hit	move	run	bmc	NS	output: gstart	showtime	reset	iloss	irun	fblue
Idle	—	0	—	—	Idle	0	0	1	0	0	0
Idle	—	1	—	—	Play	1	1	0	0	0	0
Play	0	—	0	0	Play	0	1	0	0	0	0
Play	1	—	0	—	Nothing	0	1	0	1	0	0
Play	0	—	1	0	Flash blue	0	1	0	0	0	1
Flash	—	—	—	0	Idle	0	0	0	0	0	1
Nothing	—	—	—	0	Idle	0	0	0	0	0	0

4 FF's

Idle = bmc + Idle $\overline{move}$

Play = (Idle move + Play $\overline{hit}$ $\overline{run}$) $\overline{bmc}$

Nothing = (Play hit) $\overline{bmc}$ + Nothing

Flash = (Play run) $\overline{bmc}$ + Flash

OUT

gstart = Idle move

Showtime = $\overline{Idle}$

reset = Idle

iloss = Play hit

irun = Play run

fblue = Flash