CE 100 Lab Report 7

Loop It

Armando Silva

Tutor: Dawn Hustig-Schultz

Lab Section: Tuesday, Thursday 2:00pm – 4:00pm

Date: March 11, 2016

Description:

The purpose of this lab was to get familiar with all the information taught throughout the quarter; this includes but is not constraint to flip-flops, state machines, edge detectors, Verilog, buses, 7-hex, selector, and the VGA controller. Through the “Loop It” game we built, it demonstrates our understanding and use of assembling an implemented design that could be verified and tested with the use of a VGA controller.

Part A- The VGA controller

In order to display our game, I had to connect the VGA monitor to communicate with the Basys2 Board that I worked with. In order to control the monitor, I generated two control signals, HS (horizontal synchronization signal), VS, and 8 RGB data signals for the 640 x 480 pixels of the screen. Using the Basys2 Reference Manual provided to us by the class, we determine when to set the H-Sync Low and V-Sync Low. Here we have to have specific regions to correspond with the pixels of the screen to control. For the active region, the pixel region where the monitor is displayed, we set that with dimensions 640 x 480. Then for the HS region, it is the 96th pixel in each row starting with the 656th pixel to 750th. As for the V-Sync, it is 490th and 491th row. When setting HS and VS, I used a simulator to simulate when HS and VS are high or low, they should correspond with our theoretical values. One can observe from **Figure 1-3**.

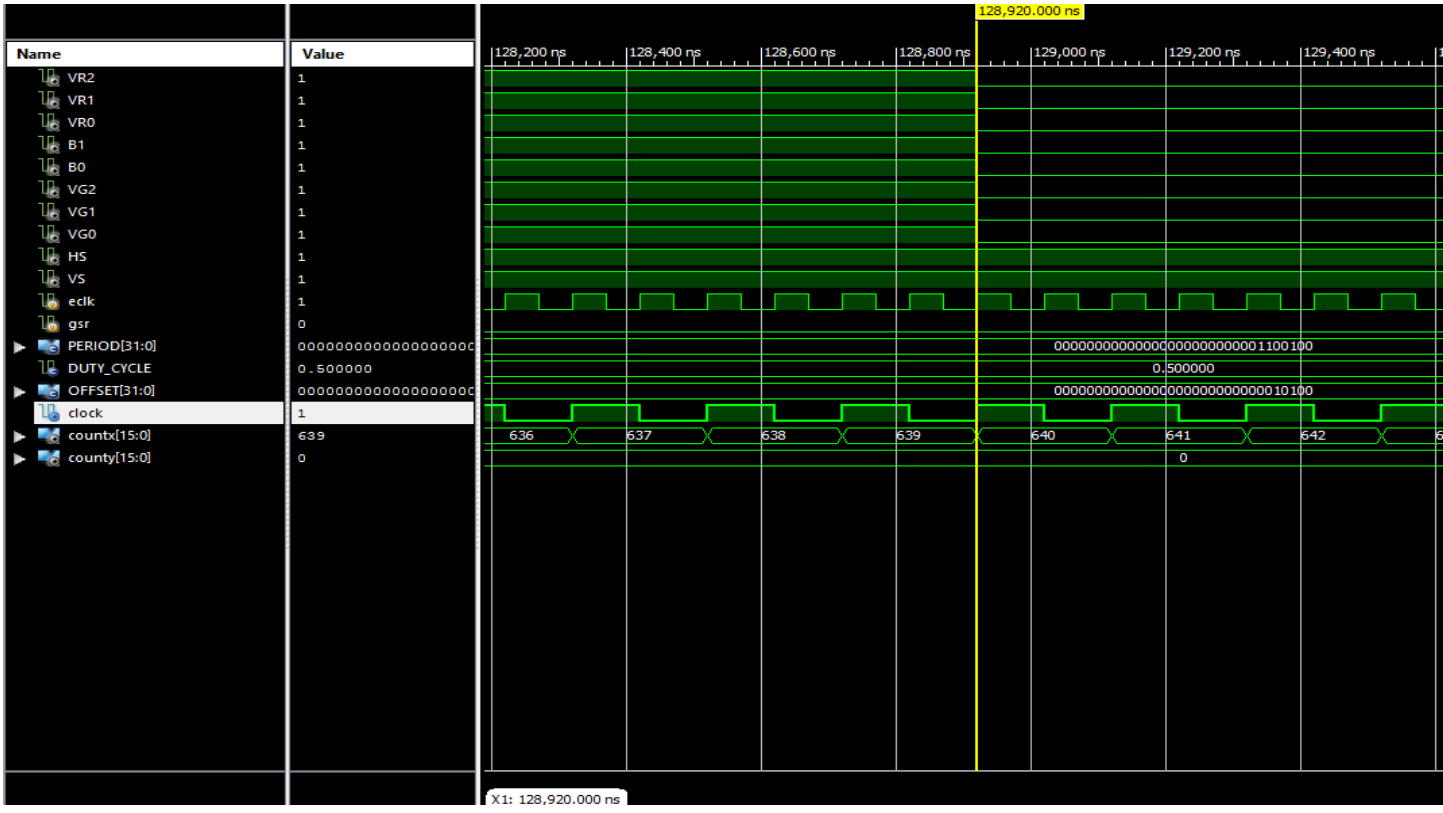

Figure 1

Active region, at pixel 640, the output go low

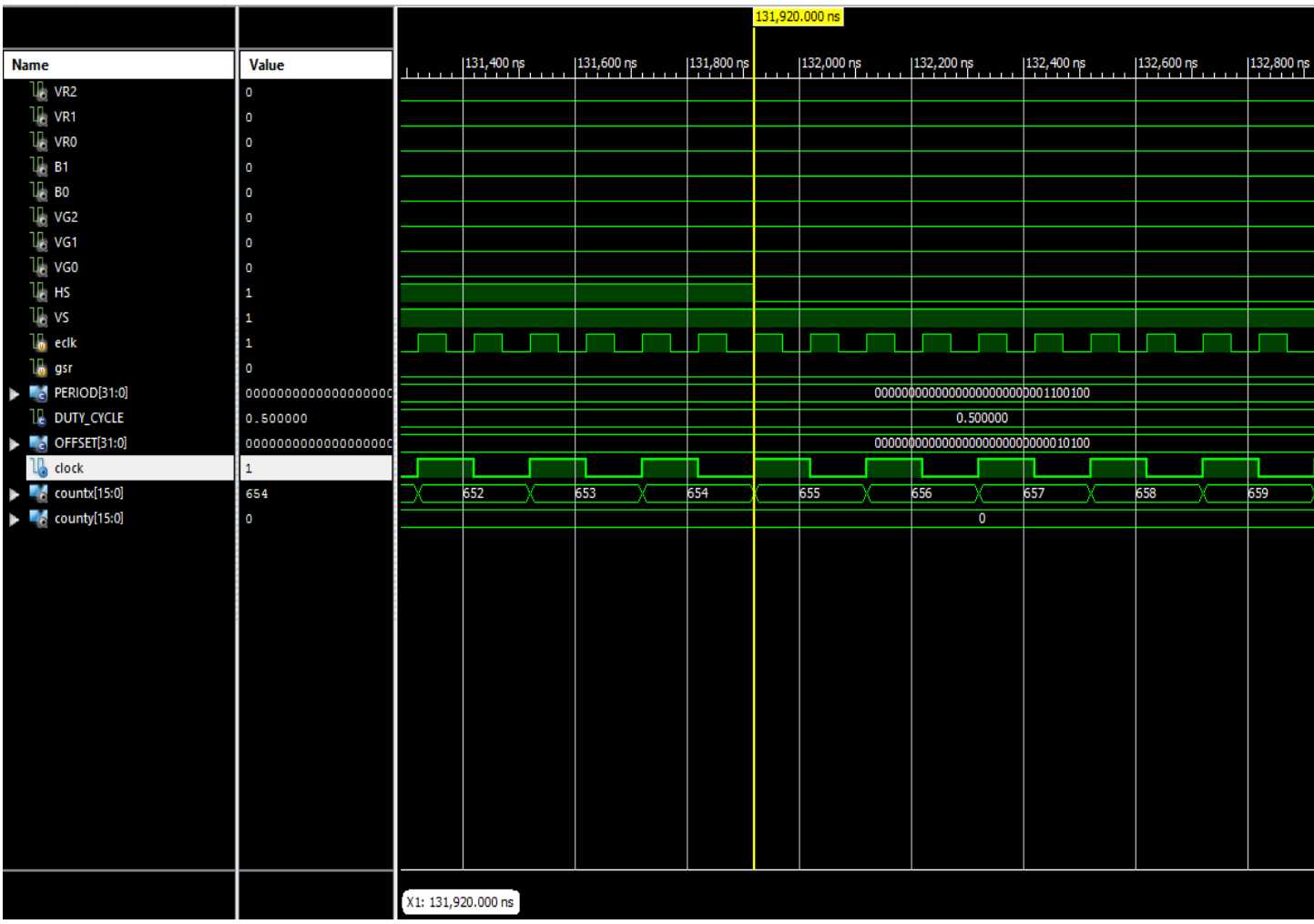

Figure 2

Hsync goes low at pixel 655

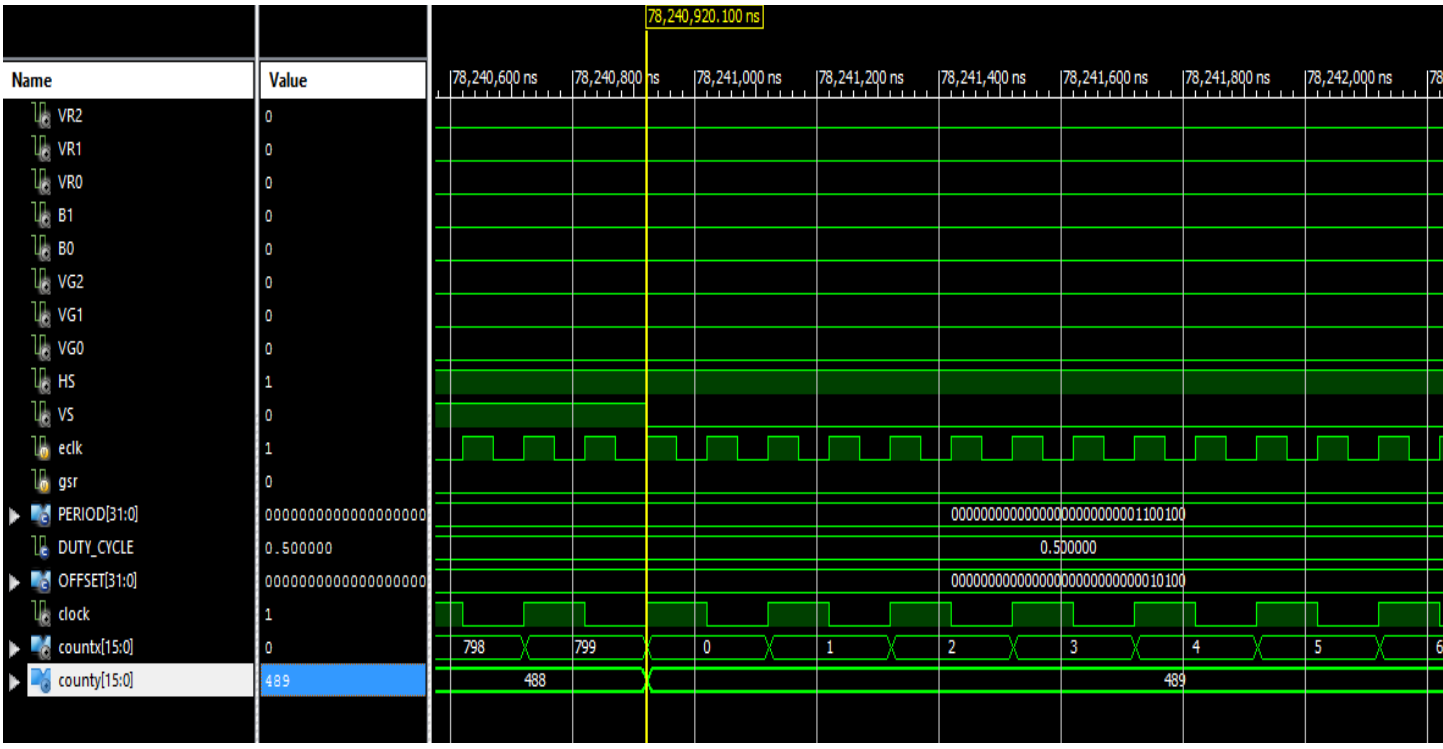

Figure 3

At y-axis pixel 489, Vsync goes low

Part B- Setting Region

This part was discussed a little by the VGA controller; however, next I was to come up with the design for a moving carpet and ball. Here, I used Verilog to meet the game requirements. I used two counters to input into this symbol, an X-axis counter for the x-axis pixels and a Y-axis counter for the y-axis of the active region monitor. These are then my inputs to my symbol "Setting Region" to coordinate my positioning of my background of the game, positioning of the ball, paddle, and the carpet.

Next was to move the carpet, this was simple math where I was to move the carpet 32 pixels to the left and hard coded this. I used the another counter, moving counter and had a Boolean expression that when that reaches the 6^{th} bit, the counter is reset. However, until it has been reset, every frame it subtracts the hard coded portion of the carpet by 1.

For the paddle, I used a similar approach as my carpet by moving it to the left, however to move it to the right when the push button 0 is pressed I used a CC16CLED and used two's compliment in order to move it to the right. If I did not include two's compliment, then when the subtraction of the paddle position became negative, the position would be "100….1", which should translate to minus 1; which is why I needed two's complement. I had a condition when the paddle reaches the left side or right side of the carpet, then I stop the counter to decrement or increase the paddle. This then sends the signal to my Paddle Y coordination to start counting and decrement the paddle, which then the paddle "falls".

This symbol contains one of two heart and souls of my program, which is why not only does "Setting Region" hold paddle, carpet, and active region's coordination's, but also the ball positioning.

Here with the ball I display it at the top of the background frame the first time around. I used a counter to move the ball from right to left just as the paddle and the carpet's color. For Y coordination, it was a little tricky, I used a trigger edge flip flop to determine if the ball hit the carpet and/or top of the background. When one of these cases happens, then the ball either increments or decrements depending on the scenario. I used signal "stoopball" to detect when the ball hits the carpet or top of the background. This will trigger the flip flop count enable high, and I have the D input to be top ball which is only if the ball is at the top of the background. If it is high then we start decrementing the position of the ball, else we increment knowing it is at the carpet.

Not only did we have to move the ball, we had to make the second and on iteration random generated. At first my thought was to create an LFSR, which I have in my design; however, when I implemented this, it bounced on the carpet and incremented heading towards the top. Then it got close to the top, but with my algebra, it reached top – random number, and did not hit the top of the background, it was a bit more sophisticated then what I thought. So what I did was go with a counter. My counter is CC8CLED and starts with the 8-bit at 0. My count enable was runLFSR, which is a signal that goes high when the ball is at idle state or when pb2 is pressed, thus when the ball is off-screen it is generating randomly, with the frames so fast it stops at "random" number and then displays the ball. Then I reset the counter when the ball hits the top of the background so that the y positioning of the ball is now only ball + county, this eliminated the problem I had with the LFSR, where the random number affected the y-axis when the ball hit the top of the background.

The last part of my "Setting Region" was detecting the collision of the ball and paddle. Here I had a signal, touch which is ball & paddle. This signal goes high; however, it passes it into my edge detector because this signal is asynchronous the signal does not stay high. Therefore, I have two edge detectors, one for paddle hit and another for incrementing the score if the ball passes through the paddle.

The heart for movement was here in "Setting Region", inside computation symbol I had helper symbols that adjusted the display of my objects in the game; however, I needed to know the states of my game, which is why in the next part I will describe my "state machine" in my vgacontroller.sch.

Part C – Top Level Schematic

In this schematic, vgacontroller.sch, is my top level design that has the symbol computation which was described in the previous part. This has the outputs to the ball, carpet, paddle, when the paddle and ball touch, and when to increment the score. Now with all of this, I create a statemachine to keep track of where I am in the "Loop It" game.

As I start the game, I have a state called herestart, I use this to determine the beginning of the game, where I have all the objects still. Once push button 2 is pressed, I go to state check, here I keep looping until the paddle falls, ball touches the paddle. In check state, it is possible that the user scores and increment is high, thus on the arc I have if increment is high at this point then I increment my score. Then say the paddle touches the ball, then here I go to state waiting. In waiting, the game requirement was to flash on and off for four seconds, thus is why I have this state. Then say we did meet the four seconds, we go to idle; however, we could also go idle if we fell off the carpet. Thus here in idle we

either go back to playing the game and go to check or if we have no more lives we go to ending state. We discussed check already, so if we go to ending state then here we show the score and stop everything. Until push button 1 is pressed we stay here, until push button 1 is pressed then we reset our game.

Now that I discussed my states, I use these signals to send out to the other symbols across VGA controller. The CB8CLED is the score for the game. The CB4CLED is the number of lives a person has, we start with 3 and decrement to 0. The four seconds is in my computation, which sends out the signal when four seconds has passed. I used my selector, ring counter, edge detector, and 7-segment display from previous lab. As for the clock, it was provided to us by Schlag.

Part D- The End, Max Frequency

After finishing the design and properly demonstrating my working Loop It game in my top level schematic, I temporarily remove the BUFGMUX and connected the output of the IBUFG with betterclk directly to clkcntrl4. Then I selected “Timing Analyzer” from the tools menu and clicking on the post-place route to obtain the static timing report for my design. For betterclk I got 12.729 nanoseconds, 1/12.729 nanoseconds = 7.85*10^7 Hz. As for recording the maximum delays, they are in the Timing Report that I will submit along with.

Conclusion

This lab proved to be very tedious, which tested my strength mentally. I spent 24 hours straight in lab and by finishing this, it was one of the best compliments I have made so far.

Along with that, I learned how to use all the design we were taught from class to build a game. I also understood how a monitor is controlled by a Basys2 board. I used state machines, flip flops, edge trigger flip flops, counters, and much more to help my schematic work. I used a lot of debugging methods such as sending outputs to RGB colors, which is why some outputs go nowhere. This makes my design one that is not the simplest, if I would go back and clean the design I would; however, given that other stuff is due, I keep it as is.

Appendix:

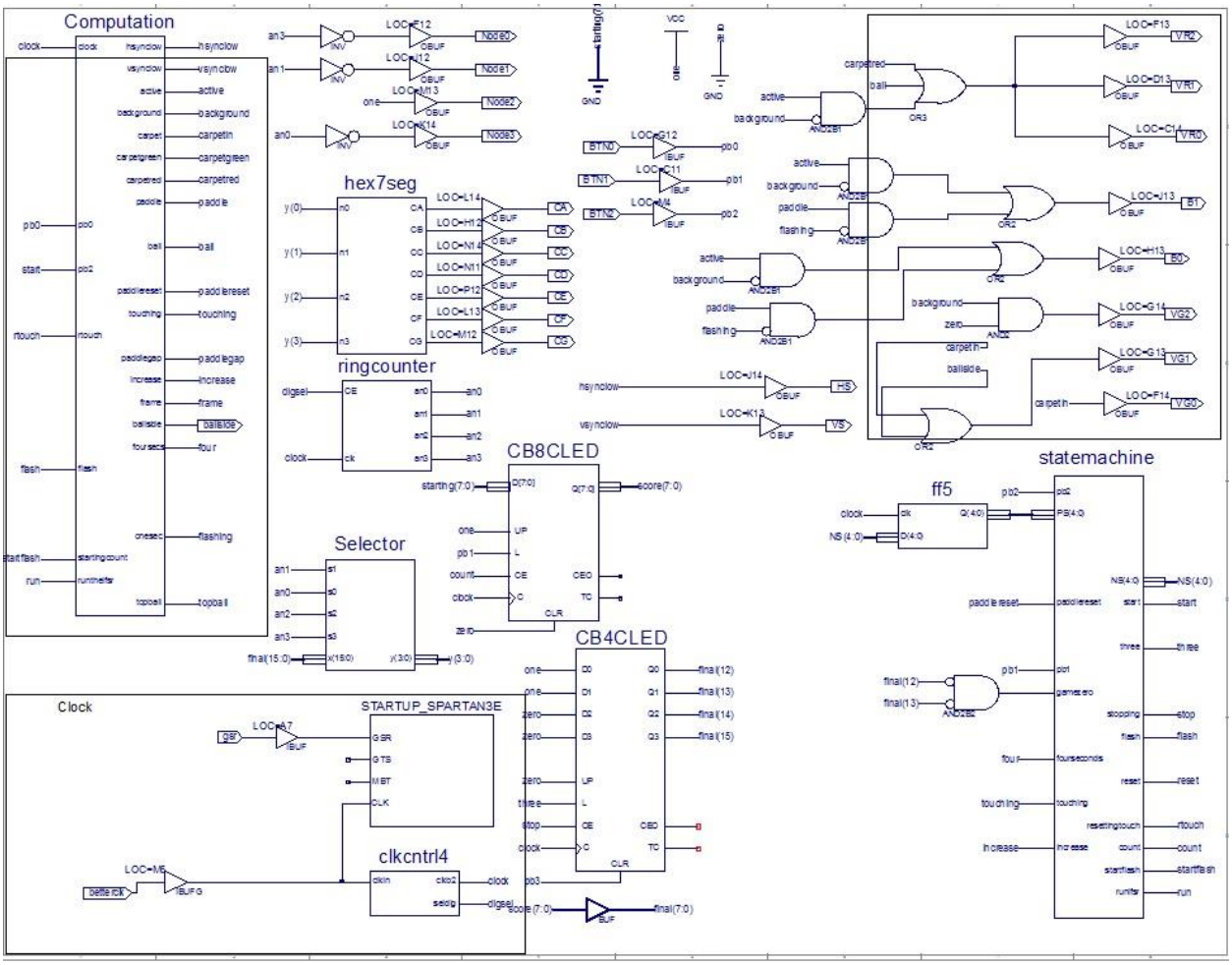

Figure 1

Top Schematic, VGA controller

```
13  //
14  // Dependencies:
15  //
16  // Revision:
17  // Revision 0.01 - File Created
18  // Additional Comments:
19  //
20  //////////////////////////////////////////////////////////////////////////////////
21  module hex7seg(
22      input n0,
23      input n1,
24      input n2,
25      input n3,
26      output CA,
27      output CB,
28      output CC,
29      output CD,
30      output CE,
31      output CF,
32      output CG
33      );
34
35
36
37      //sop for CA
38
39      assign CA = ((~n3&~n2&~n1&n0) | (~n3&n2&~n1&~n0) | (n3&~n2&n1&n0) | (n3&n2&~n1&n0));
40
41      //sop for CB
42
43      assign CB = ((~n3&n2&~n1&n0) | (~n3&n2&n1&~n0) | (n3&~n2&n1&n0) | (n3&n2&~n1&~n0) | (n3&n2&n1&~n0) | (n3&n2&n1&n0));
44
45      //sop for CC
46
47      assign CC = ((~n3&~n2&n1&~n0) | (n3&n2&~n1&~n0) | (n3&n2&n1&~n0) | (n3&n2&n1&n0));
48
49      //sop for CD
50
51      assign CD = ((~n3&~n2&~n1&n0) | (~n3&n2&~n1&~n0) | (~n3&n2&n1&n0) | (n3&~n2&n1&~n0) | (n3&n2&n1&n0));
52
53      //sop for CE
54
55      assign CE = ((~n3&~n2&~n1&n0) | (~n3&~n2&n1&n0) | (~n3&n2&~n1&~n0) | (~n3&n2&~n1&n0) | (~n3&n2&n1&n0) | (n3&~n2&~n1&n0));
56
57      //sop for CF
58
59      assign CF = ((~n3&~n2&~n1&n0) | (~n3&~n2&n1&~n0) | (~n3&~n2&n1&n0) | (~n3&n2&n1&n0) | (n3&n2&~n1&n0));
60
61      //sop for CG
62
63      assign CG = ((~n3&~n2&~n1&~n0) | (~n3&~n2&~n1&n0) | (~n3&n2&n1&n0) | (n3&n2&~n1&~n0));
64
65  endmodule
```

Figure 2

Hex7seg from VGA schematic

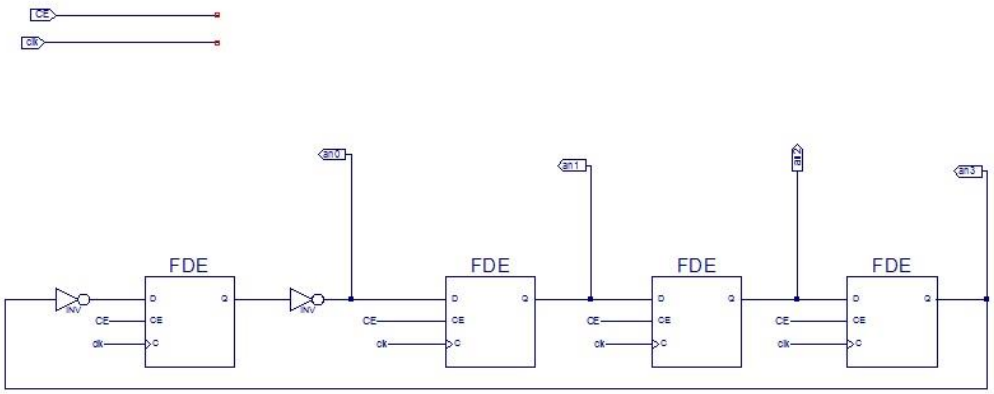

Figure 3

Ring Counter from VGA schematic

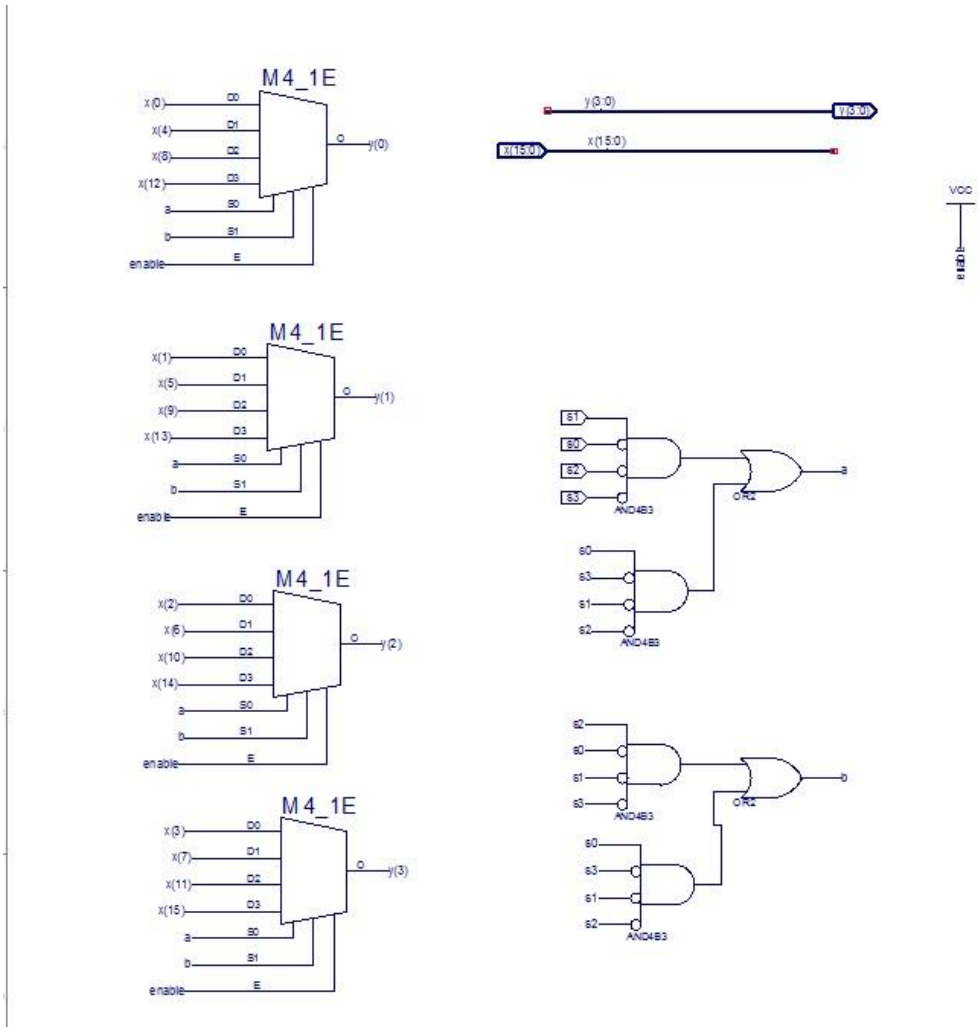

Figure 4

Selector from the VGA schematic

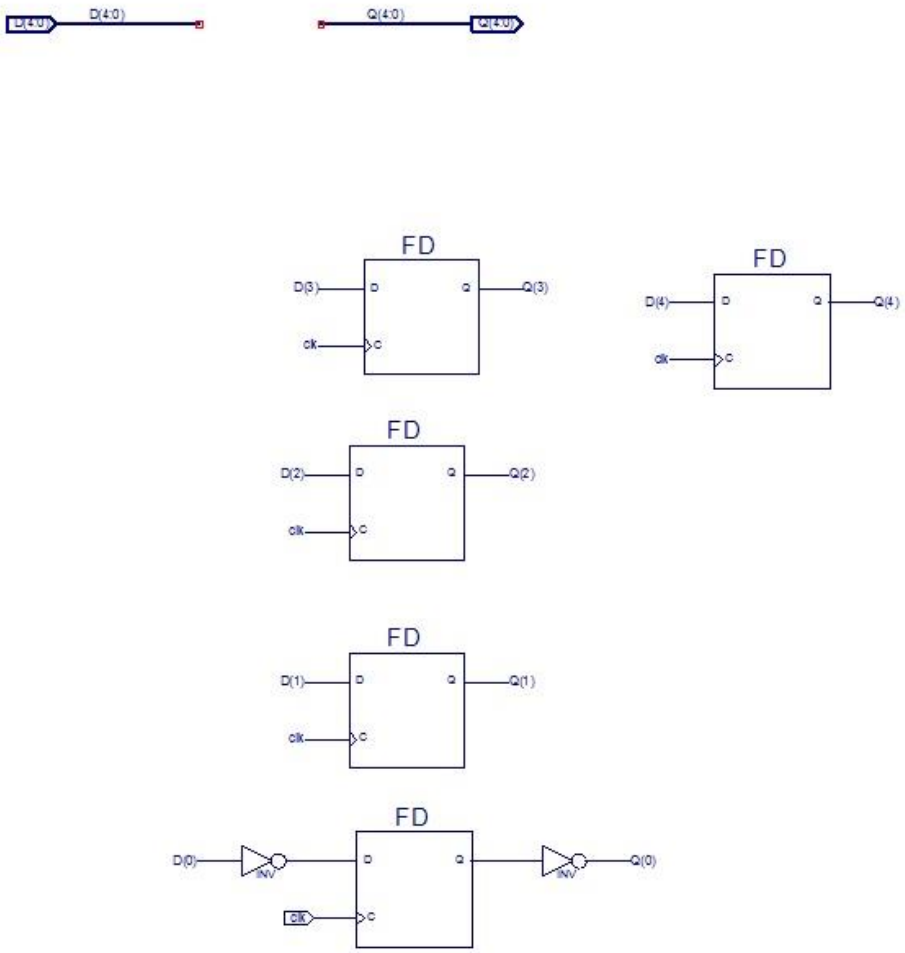

Figure 5

FF5 flip flop for the statemachine for top level

```
21  module statemachine(
22      input paddlereset,
23      input pb2,
24      input pb1,
25      input increase,
26      input fourseconds,
27      input touching,
28      input gamezero,
29      input [4:0] PS,
30      output [4:0] NS,
31      output start,
32      output reset,
33      output stopping,
34      output resettingtouch,
35      output three,
36      output count,
37      output startflash,
38      output runlfsr,
39      output flash
40      );
41      wire IDLE, check, herestart, ending, waiting;
42      wire Next_IDLE, Next_check, Next_herestart, Next_ending, Next_waiting;
43
44     assign herestart = PS[0];
45     assign check = PS[1];
46     assign IDLE = PS[2];
47     assign ending = PS[3];
48     assign waiting = PS[4];
49
50
51     assign NS[0] = Next_herestart;
52     assign NS[1] = Next_check;
53     assign NS[2] = Next_IDLE;
54     assign NS[3] = Next_ending;
55     assign NS[4] = Next_waiting;
56
57     assign Next_herestart = (herestart&~pb2) | (ending&pb1);
58     assign Next_check = (IDLE&pb2&~gamezero) | (check&~paddlereset&~touching) | (herestart&pb2);
59     assign Next_IDLE = (IDLE&~pb2)|(waiting&fourseconds)| (check&paddlereset);
60     assign Next_ending = (IDLE&gamezero) | (ending&~pb1);
61     assign Next_waiting =(waiting&~fourseconds) | (check&touching);
62
63     assign start =(check&~paddlereset);
64     assign stopping = (check&paddlereset) | (check&touching);
65     assign reset = (waiting&fourseconds);
66     assign resettingtouch = (IDLE&~pb2);
67     assign three = (herestart&~pb2);
68     assign flash = (stopping) || (waiting&fourseconds);
69     assign count = (check&increase);
70     assign startflash = (waiting&~fourseconds);
71     assign runlfsr = (herestart&pb2) | (IDLE&pb2);
72
73  endmodule
74
```

Figure 6

Statemachine for the top level

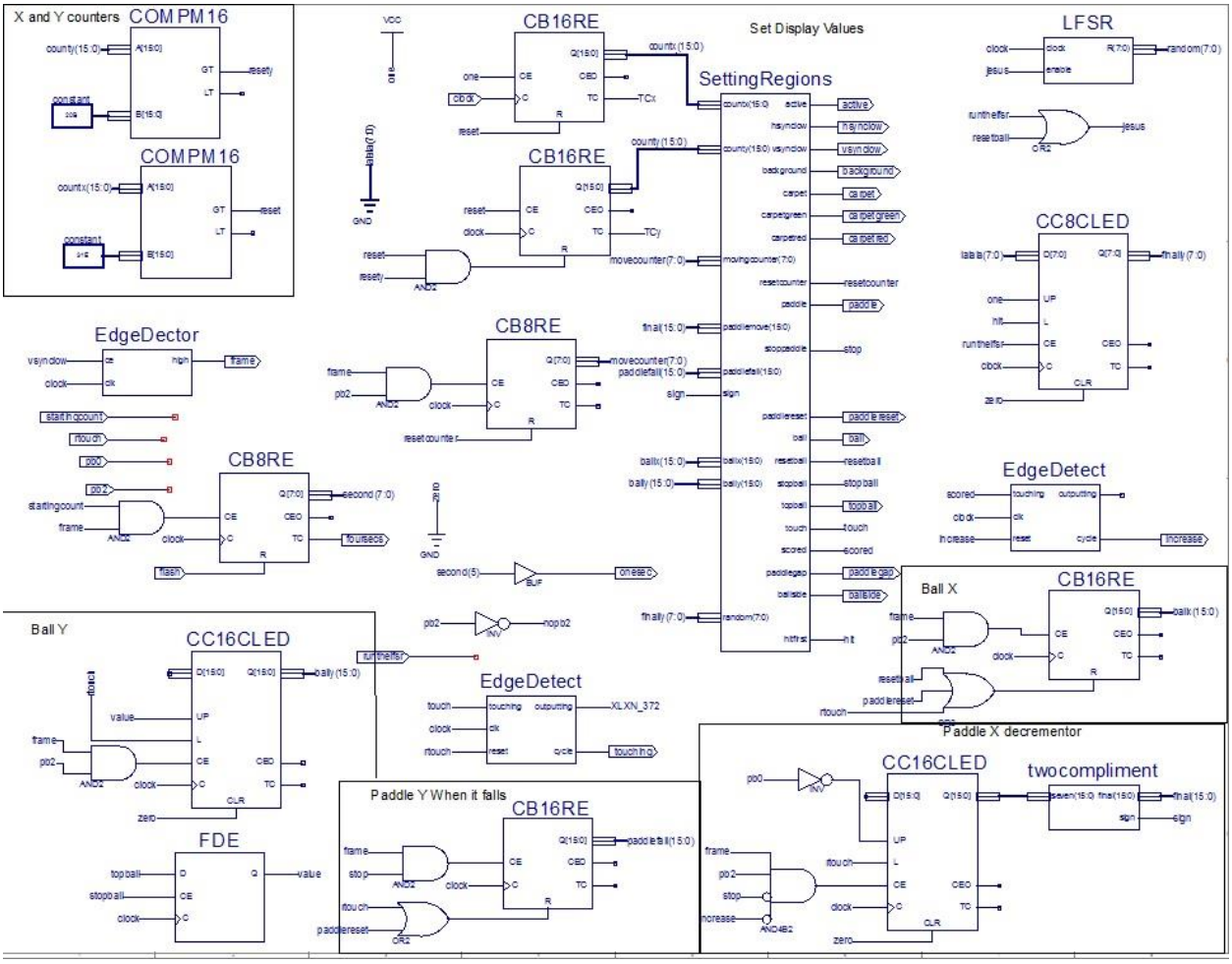

Figure 7

Computation symbol

```
21  module SettingRegions(
22     input [15:0] countx,
23     input [15:0] county,
24     input [7:0] movingcounter,
25     input [15:0] paddlefall,
26     input [15:0] paddlemove,
27     input [15:0] ballx,
28     input [15:0] bally,
29     input [7:0] random,
30     input sign,
31     output active,
32     output hsynclow,
33     output vsynclow,
34     output background,
35     output carpet,
36     output carpetgreen,
37     output carpetred,
38     output resetcounter,
39     output paddle,
40     output paddlereset,
41     output ball,
42     output stopball,
43     output topball,
44     output resetball,
45     output stoppaddle,
46     output scored,
47     output paddlegap,
48     output ballside,
49     output hitfirst,
50     output touch
51     );
52
53     assign active = (countx < 10'd640 & county < 10'd480);
54     assign hsynclow = ~(countx > 10'd654 & countx < 10'd751 & county < 10'd525);
55     assign vsynclow = ~((countx >= 10'd0 & countx < 10'd800) & (county > 10'd488 & county < 10'd491));
56     assign background = (countx > 10'd7 & countx < 10'd631 & county > 10'd225 & county < 10'd472);
57     assign carpety = ((county > 399 )& (county < 416));
58
59     //This is for the carpet
60     assign carpet = (countx > 56)  & (countx < 584) & carpety;
61     assign carpetgreen = (countx[5] == 1) & carpet;
62     //This is for the moving orange part
63     assign carpetred = ( ((countx > (56 - movingcounter)) & (countx < (88 - movingcounter )) ) || ((countx > (120 - movingcounter)) & (countx < (152-movingcounter))) ||
64                          ((countx > (184 - movingcounter)) & (countx < (216- movingcounter))) ||
65                          ((countx > (248- movingcounter)) & (countx <  (280- movingcounter))) || ((countx > (312- movingcounter)) & (countx < (344- movingcounter))) ||
66                          ((countx > (376- movingcounter)) & (countx < (408 - movingcounter))) ||
67                          ((countx > (440 - movingcounter)) & (countx < (472- movingcounter))) || ((countx > (504 - movingcounter)) & (countx < (536- movingcounter))) ||
68                          ((countx > (568- movingcounter)) & (countx < (600- movingcounter)))) & carpet;
69     //resseting counter of moving orange part
70     assign resetcounter = (movingcounter[6]);
71
72     //This is for the paddle
73     assign paddle = ( (((~sign) & (countx > (264 - paddlemove) & countx < (280 - paddlemove))) || (((sign) & (countx > (264 + paddlemove) & countx < (280 + paddlemove))))) &
74                  (((county > (336 + (2*paddlefall))) & county < (352 + (2*paddlefall))) || (county > (384 + (2*paddlefall)) & county < (400 + (2*paddlefall))))));
```

```
72    //This is for the paddle
73    assign paddle = ( (((~sign) & (countx > (264 - paddlemove) & countx < (280 - paddlemove))) || (((sign) & (countx > (264 + paddlemove) & countx < (280 + paddlemove))))) &
74                  (((county > (336 + (2*paddlefall))) & county < (352 + (2*paddlefall))) || (county > (384 + (2*paddlefall)) & county < (400 + (2*paddlefall)))));
75
76    assign paddlex = (((~sign) & (countx === (264 - paddlemove))) || (((sign) & (countx === (264 + paddlemove)))));
77
78    assign paddlegap = paddlex & ~((((county < (352)) || (county > (384)))));
79
80    //This is to stop the paddle x-axis when falling off carpet
81    assign stoppaddle = (((~sign) & (264 - paddlemove) < 40)   || ( (sign) & (280 + paddlemove) > 600));
82    //This is to reset the paddle position after falling
83    assign paddlereset = (paddlefall == 68);
84
85    //This is position of the ball
86    assign ball = (((countx > (629- ballx)) & (countx <= (639-ballx))) & (county > (225 + bally+random) & county < (234 + bally+random)));
87    assign ballside = (((countx > (621- ballx)) & (countx <= (622-ballx))) & (county > (225 + bally+random) & county < (227 + bally+random)));
88
89    assign hitfirst = (235 + bally + random > 400);
90    //This is to reset position of the ball
91    assign resetball = (ballx === 622);
92    //This is to stop the y-axis when it hits the carpet or top of background
93    assign stopball = (235 + bally + random > 400) || ((225+ bally) == 225);
94    //This is to detect when the ball is at the top of background
95    assign topball = (225+bally === 225);
96
97    //This is to detect when the ball is touching with the paddle
98    assign touch = (ball&paddle);
99
100   //This detects when the ball is passed through the paddle
101   assign scored = (ballside&paddlegap);
102 endmodule
103
```

Figure 8

Setting Region

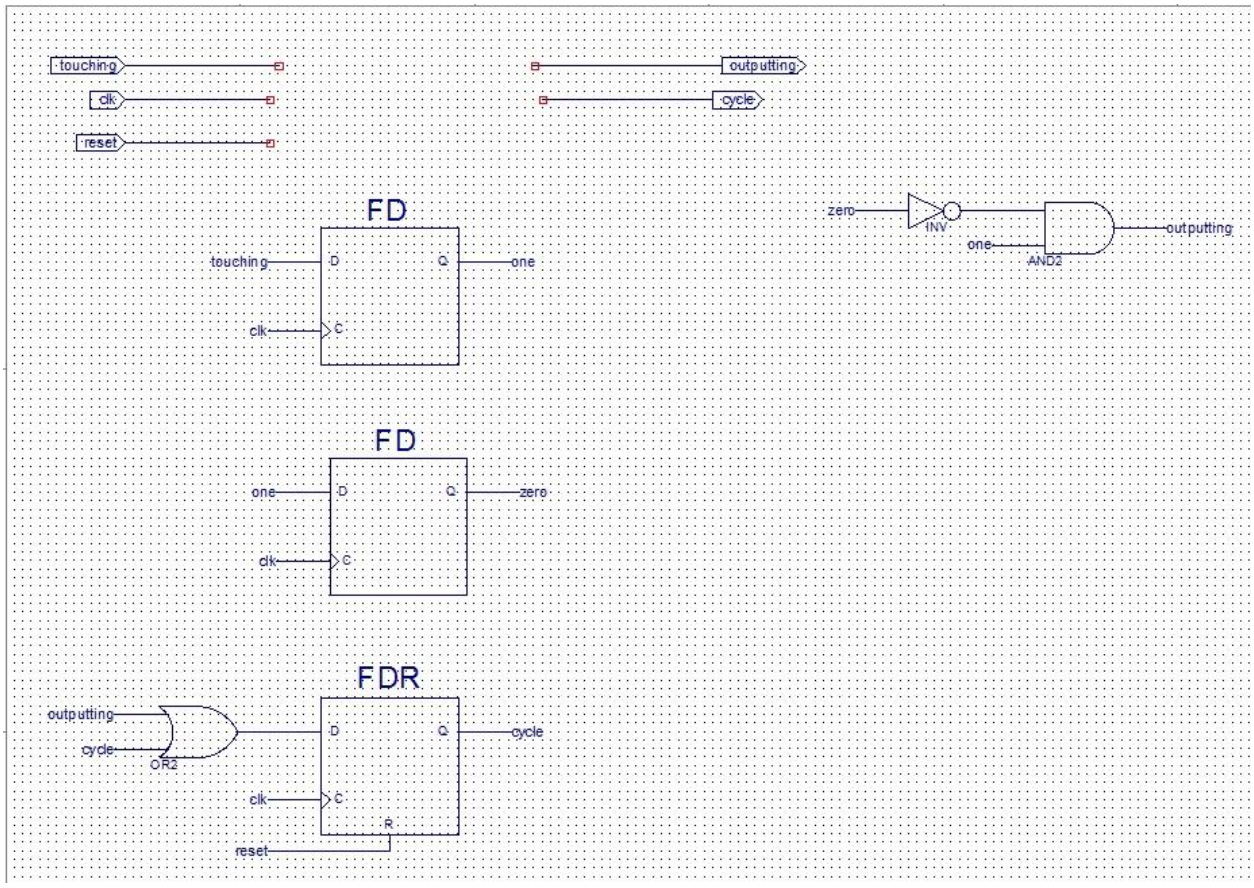

Figure 9

Edge Detector for when ball scores a point

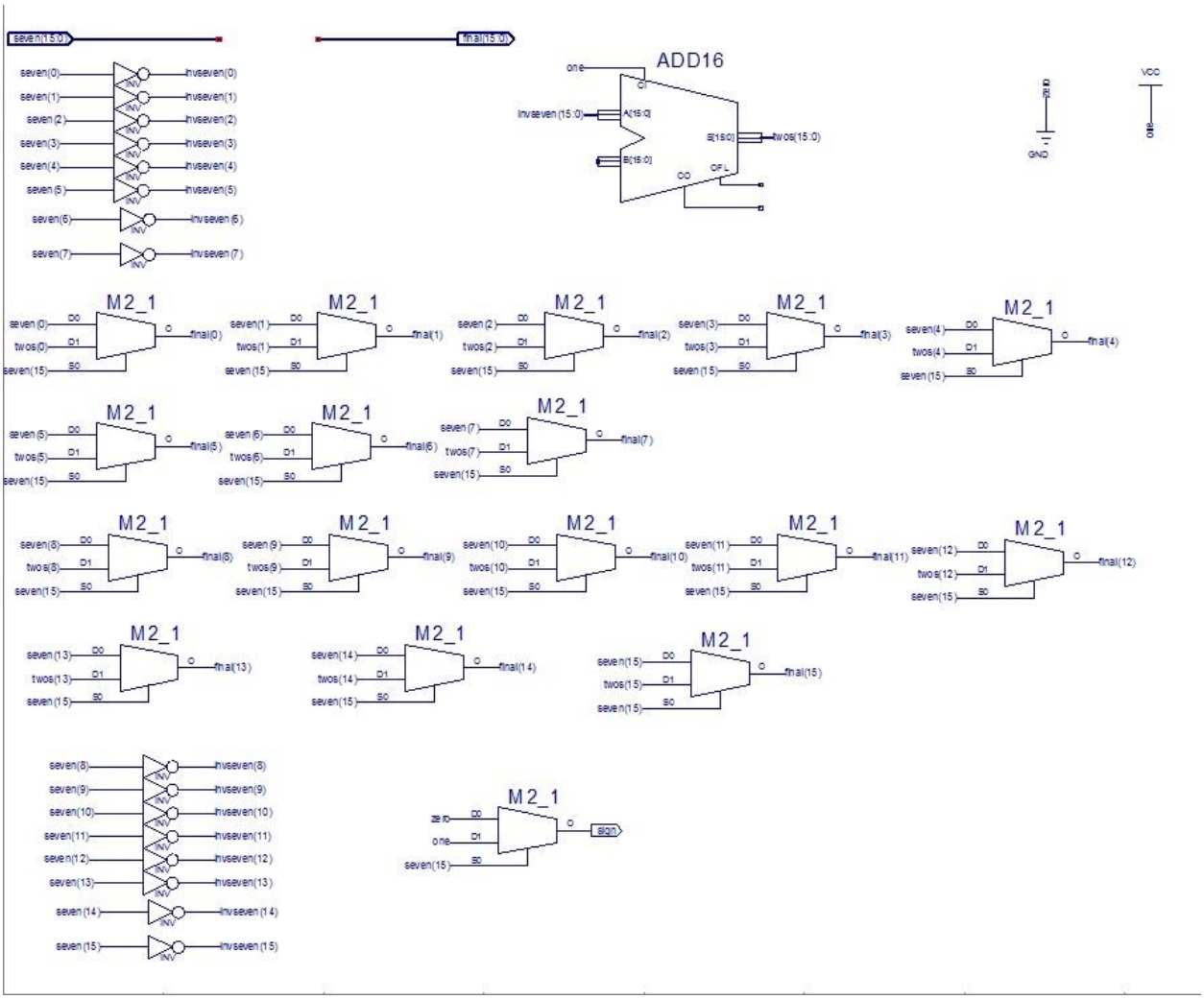

Figure 10

Two compliment when push button 0 is pressed and paddle passes half of carpet

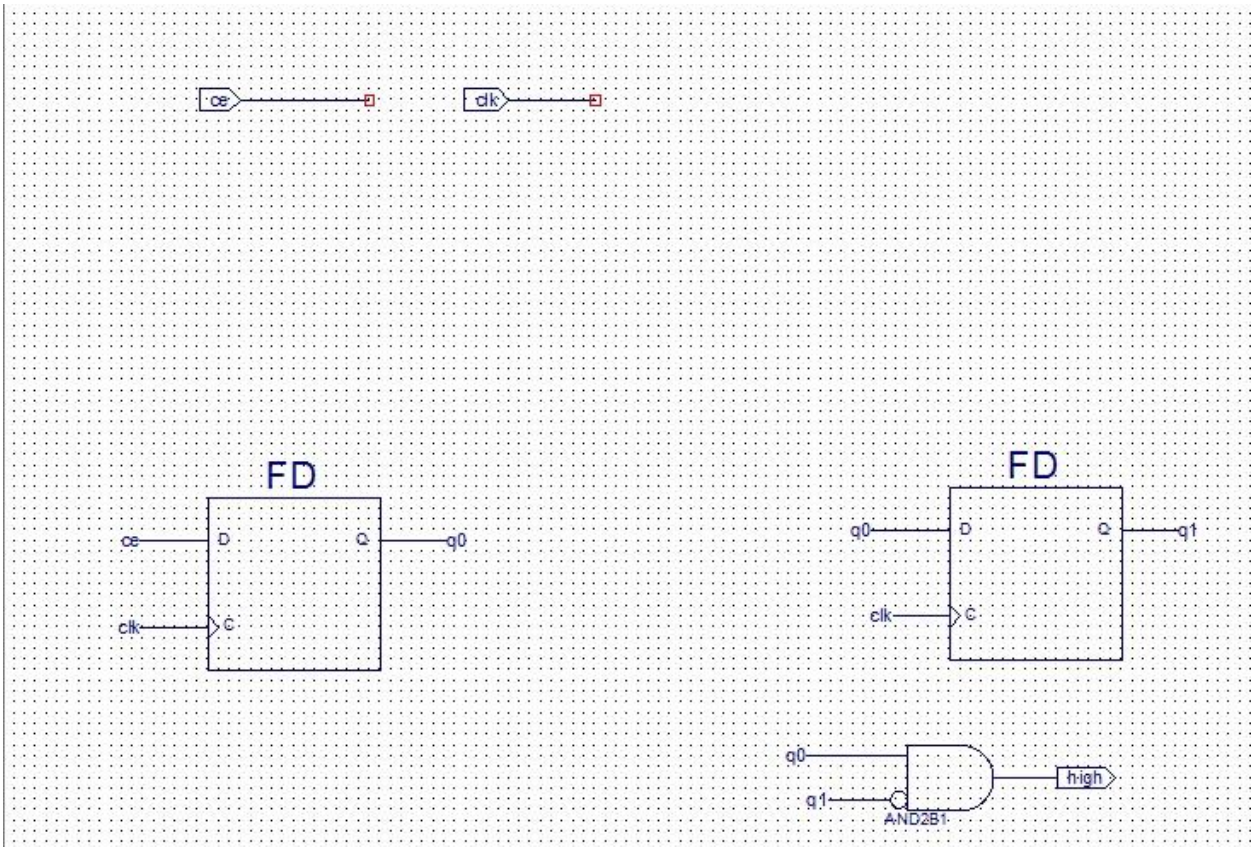

Figure 11

Edge detector for the Vsync

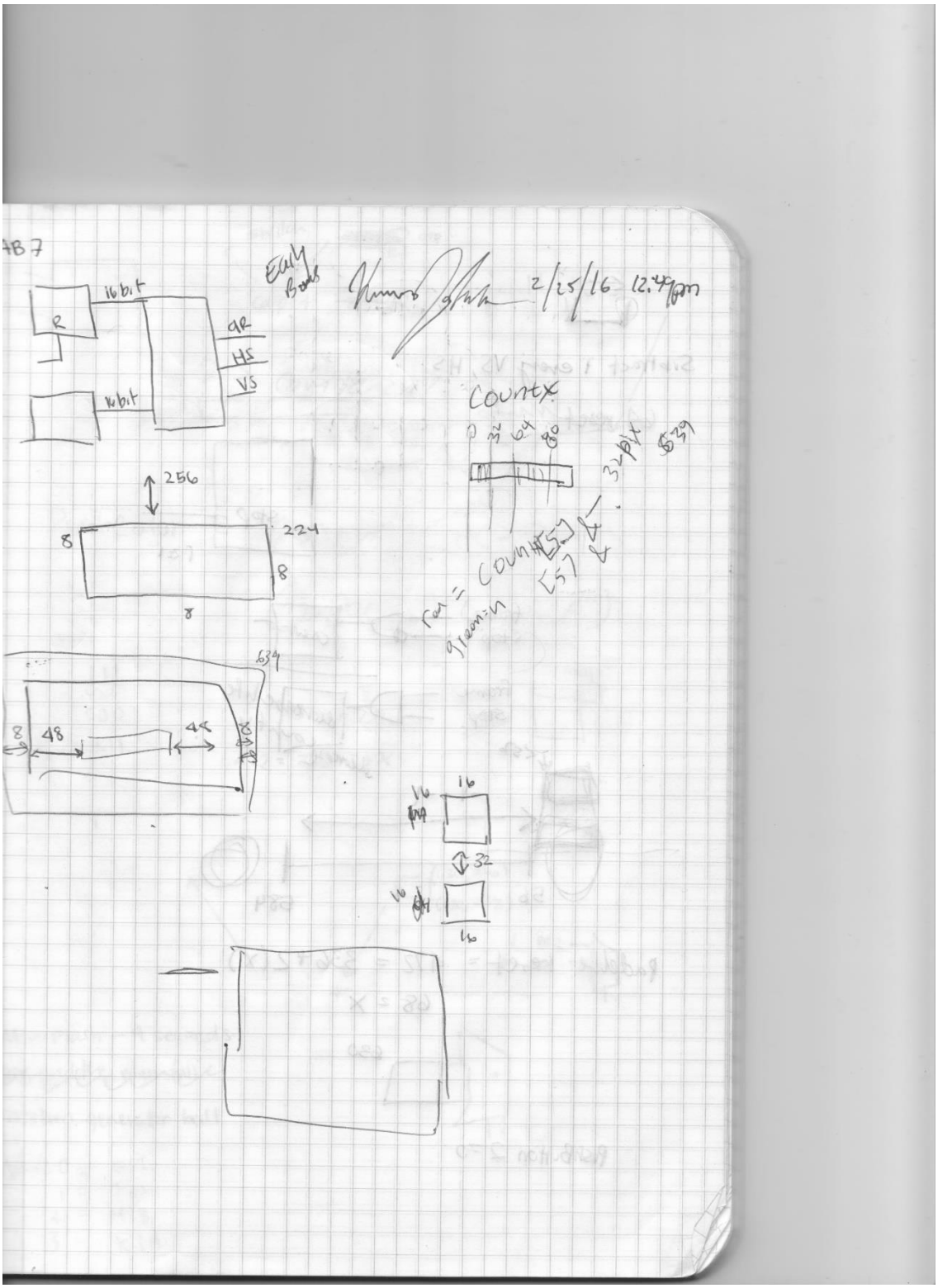

Figure 12

First page of my lab notebook

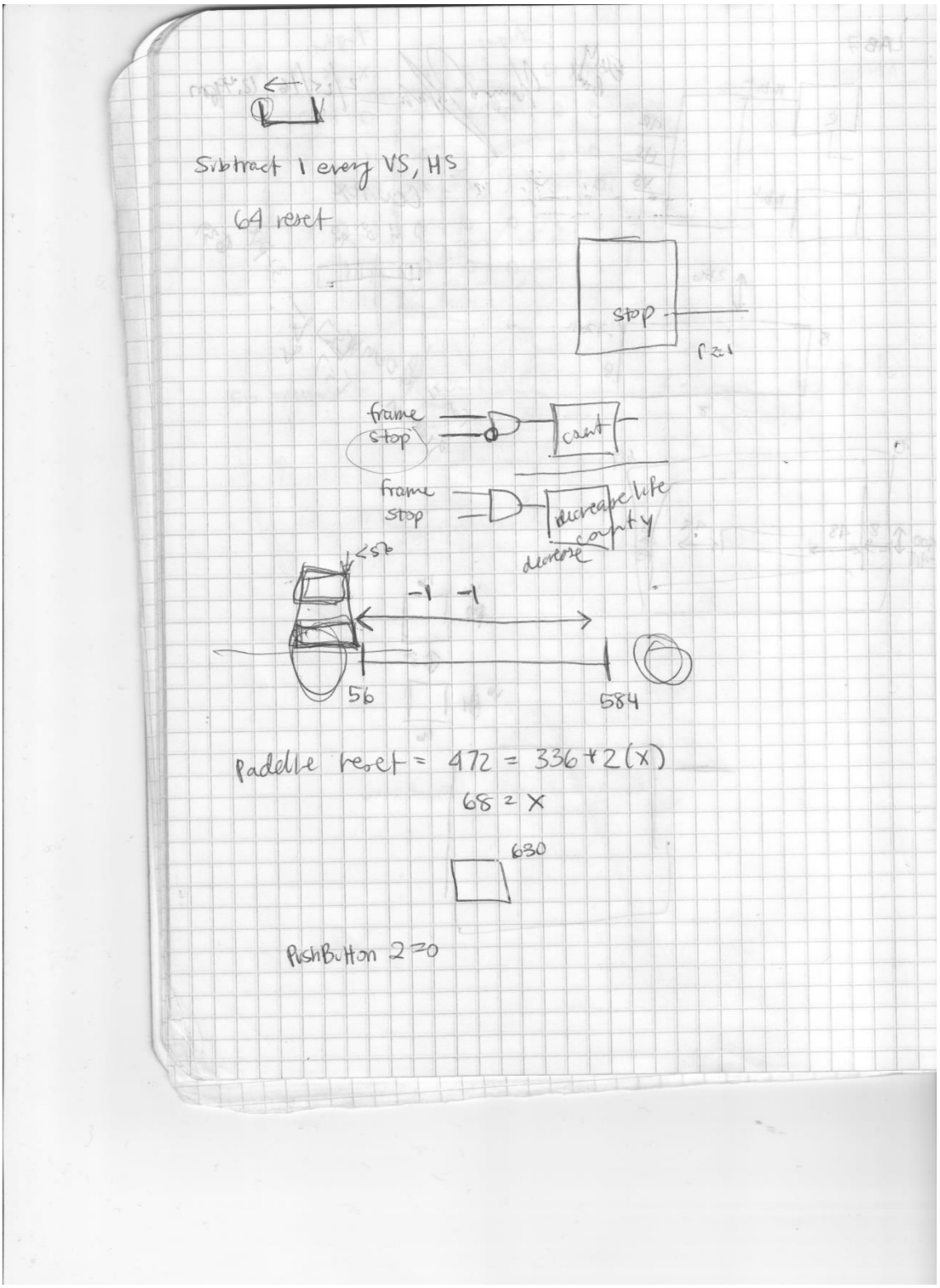

Figure 13

Second page of my lab notebook

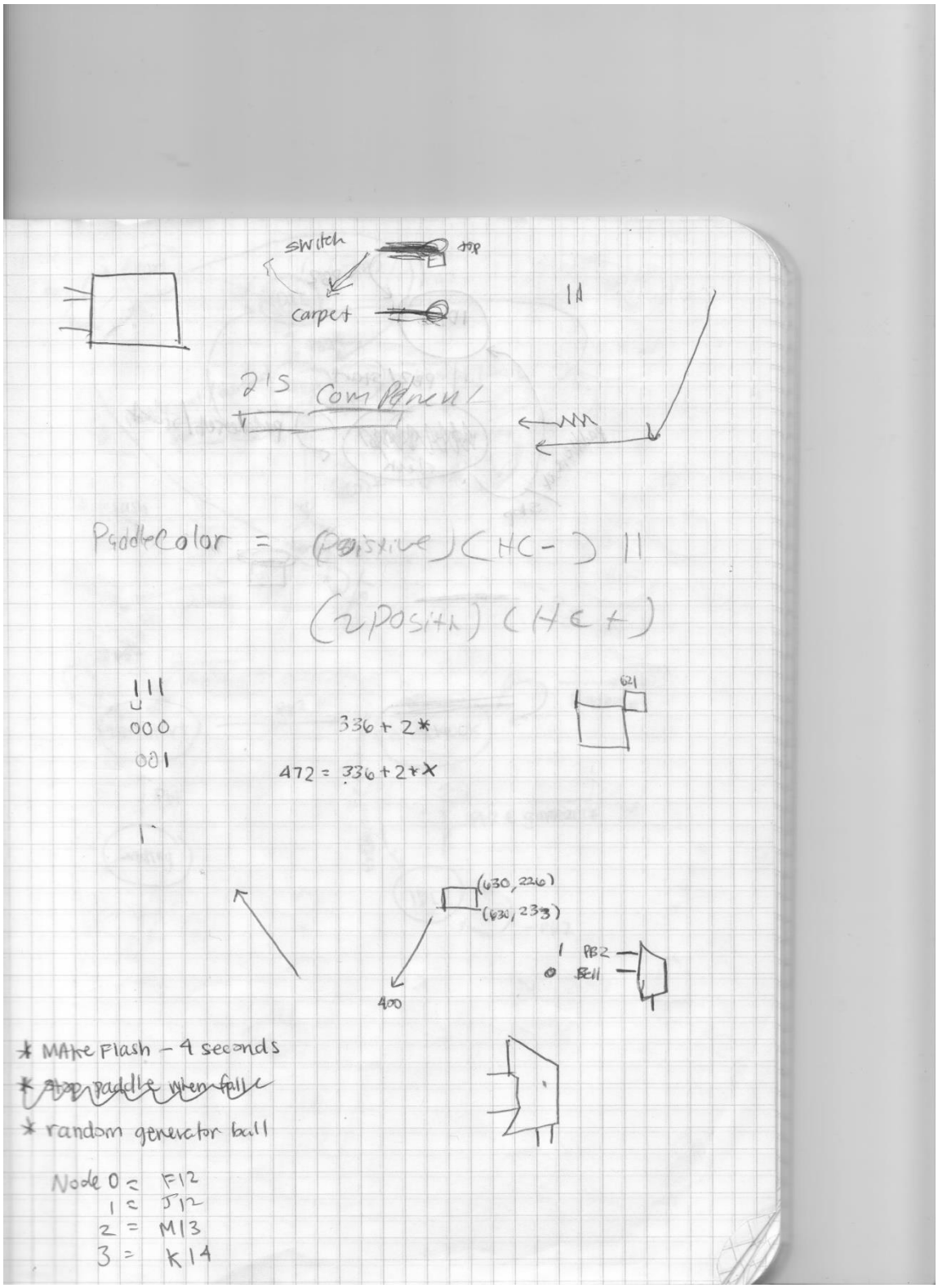

Figure 14

3rd Page of my notebook

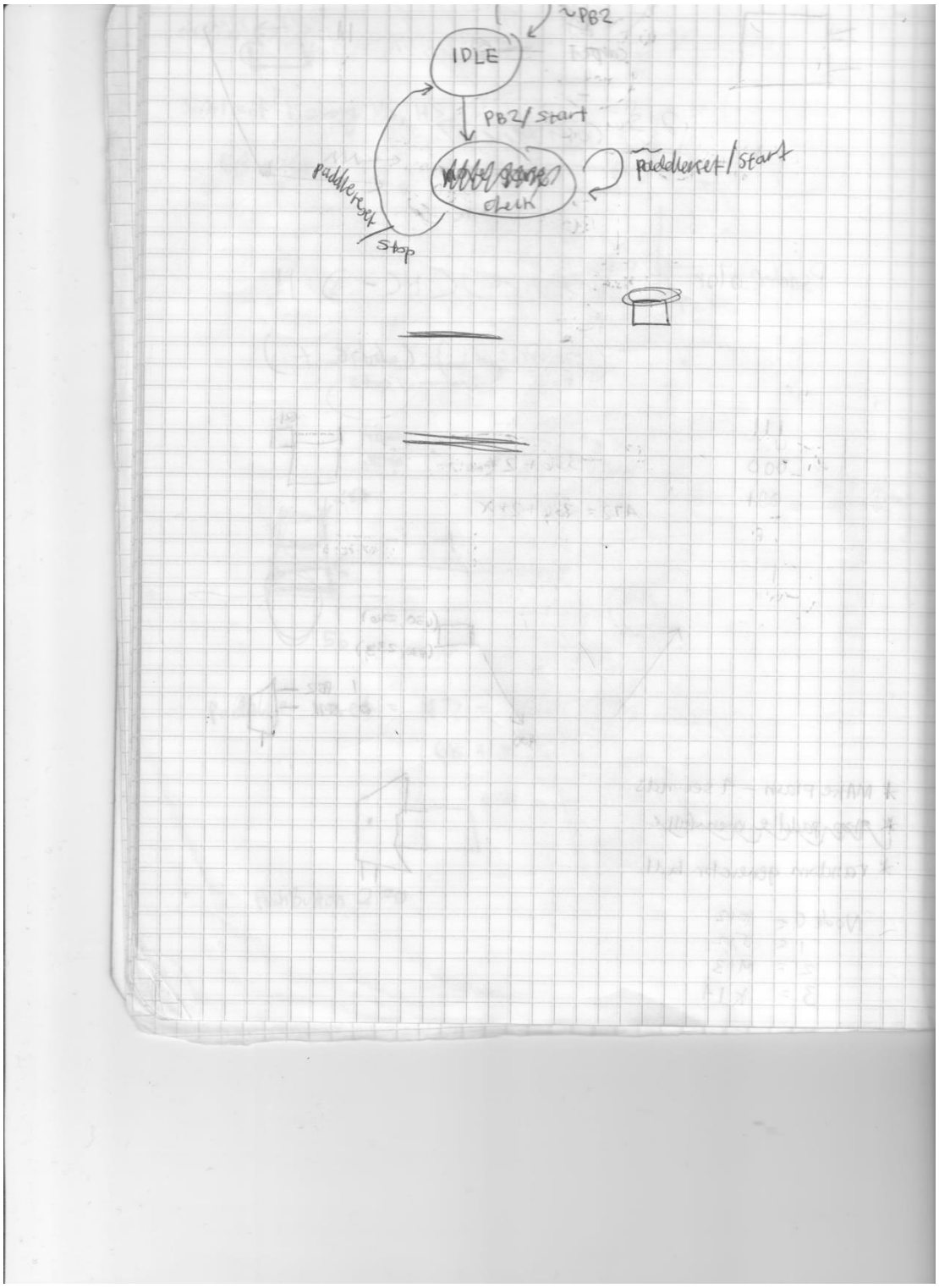

Figure 15

4th page of my notebook

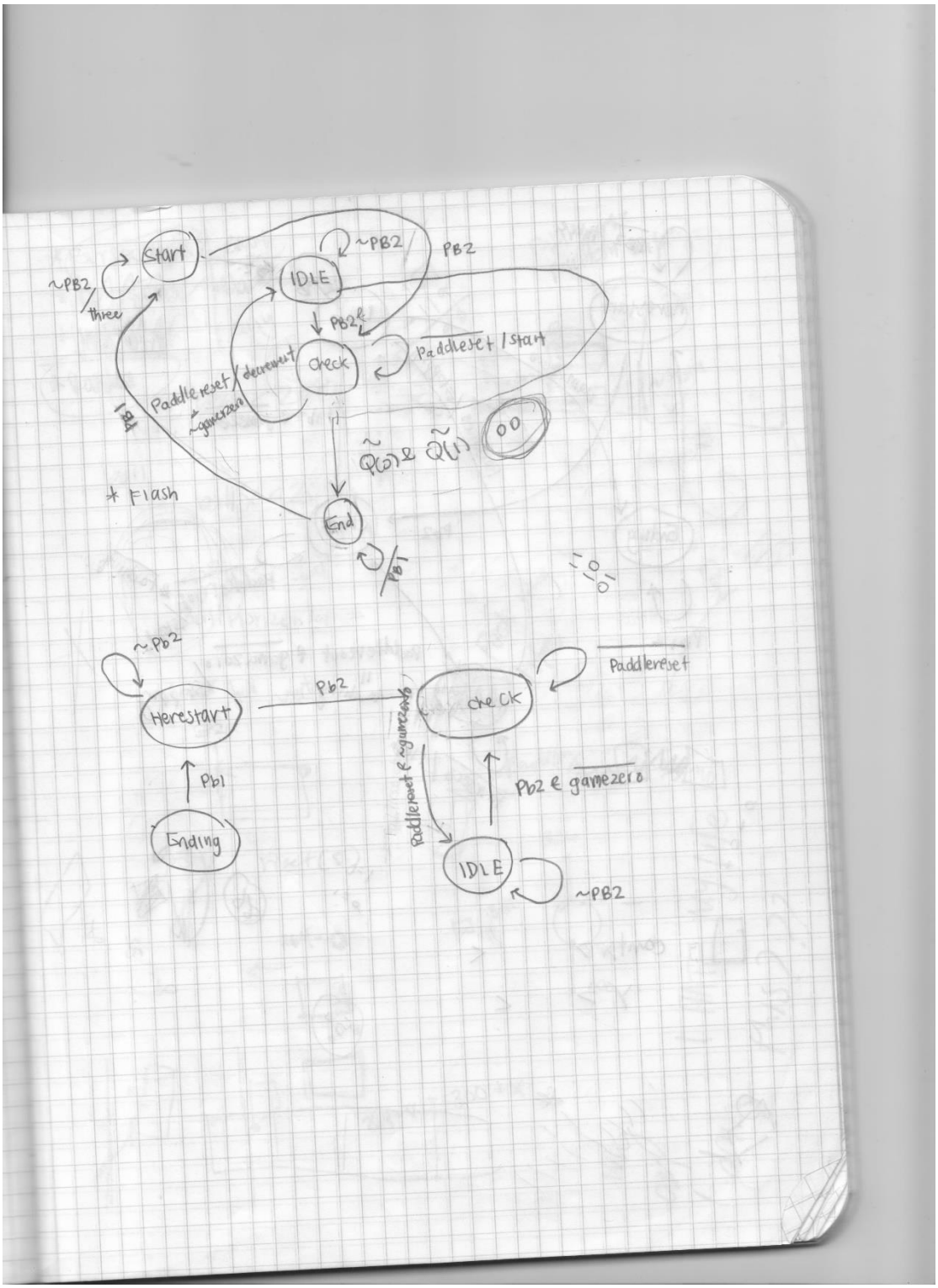

Figure 16

5th Page of my notebook

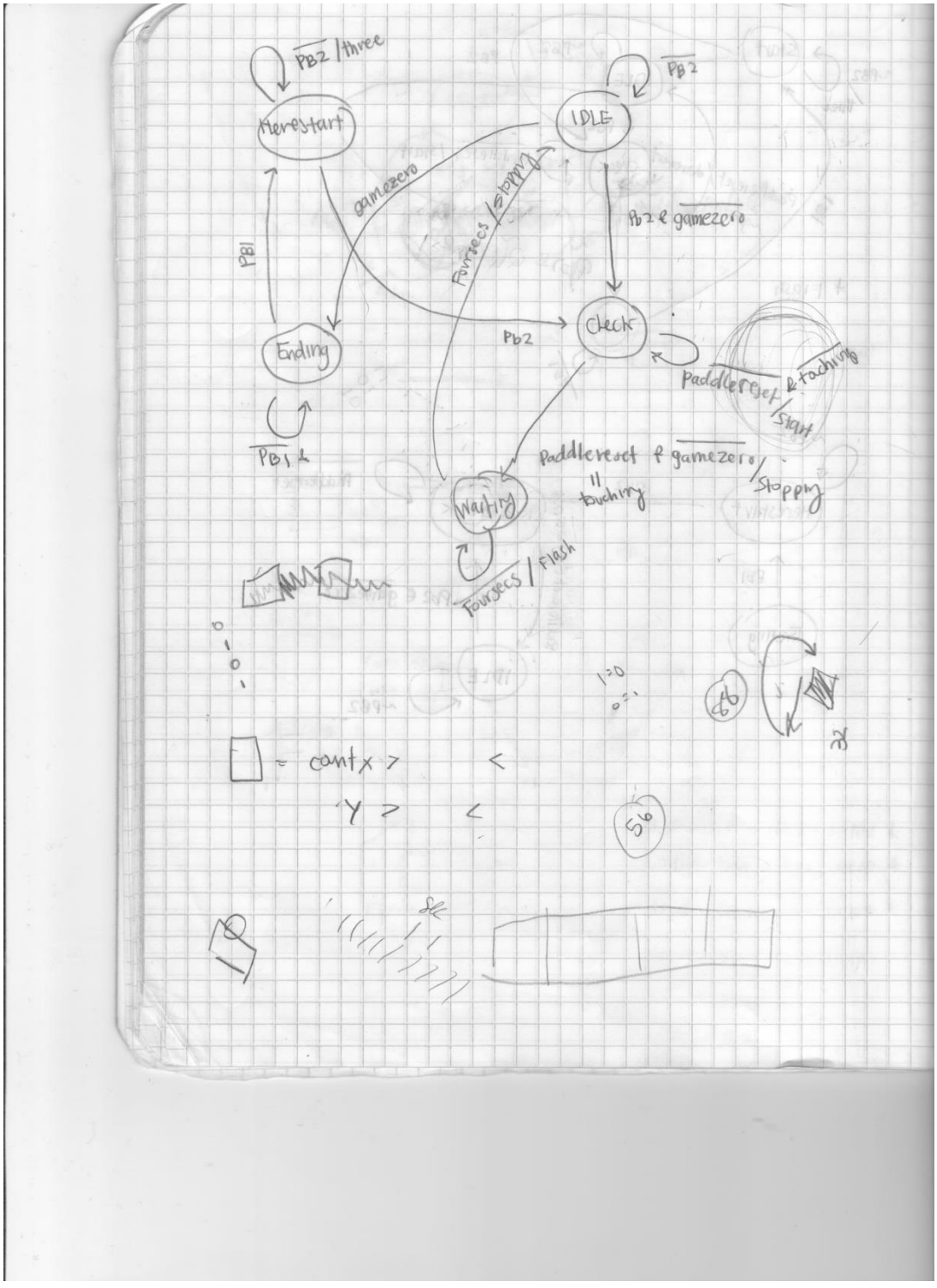

Figure 17

6th page of notebook

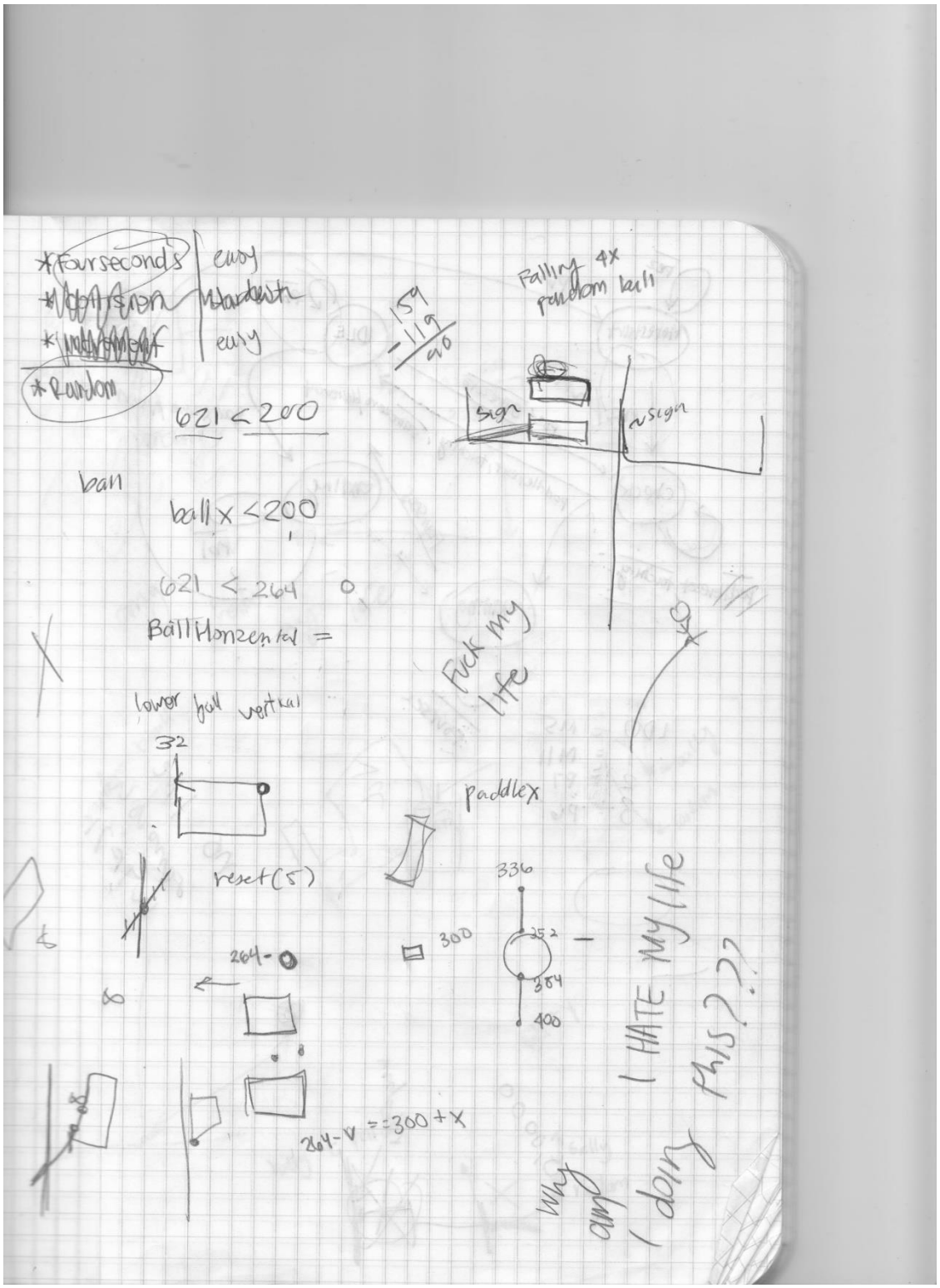

Figure 18

7th page of my notebook

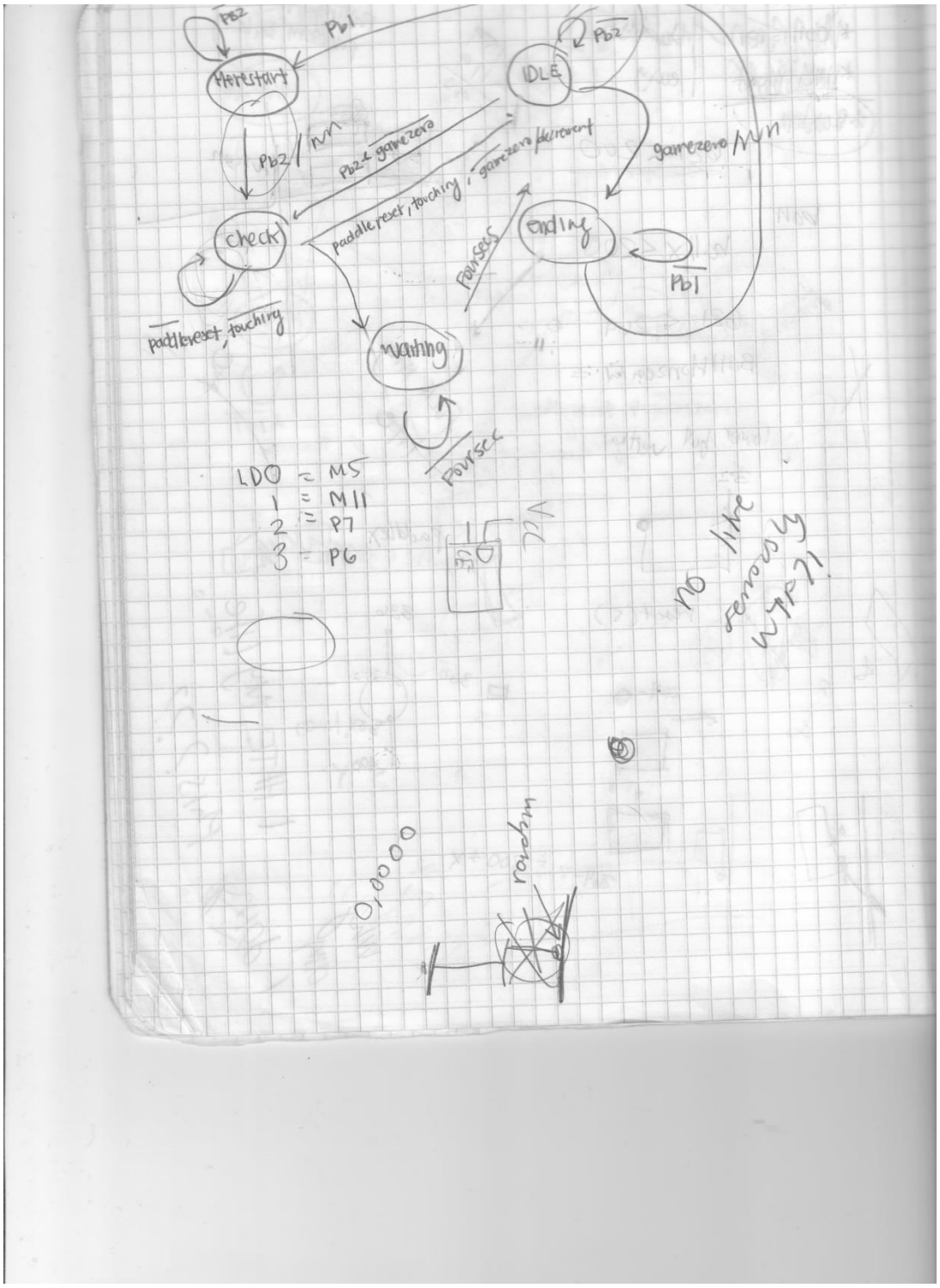

Figure 19

8th Page of my notebook

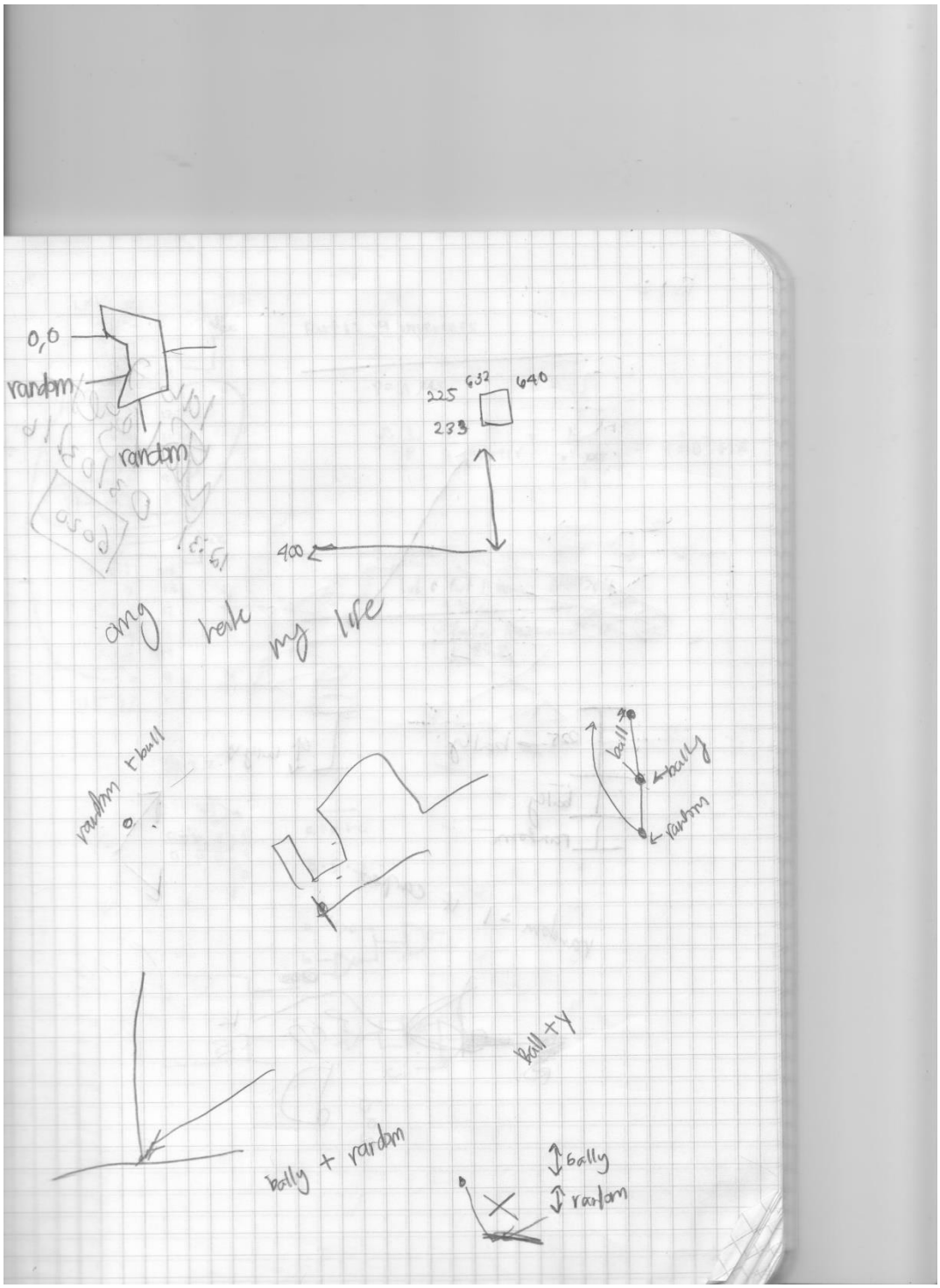

Figure 20

10th page of my notebook

Figure 21

11th page of my notebook