Lab 6: Turducken Counter

Gavin Chen

5/21/18

CE100 Spring 18

Description:

In this lab we created a simple state machine circuit to count turkeys as they wandered around. The lab consisted of 2 parts, a counter for the score and the time, and a state machine to keep track of the current action. Many of the modules were recycled from previous labs, and most of the work was done creating the top module as well as the 4 second timer and the state machine.

Methods:

This lab consisted of 5 parts, a counter to keep the score, a counter to keep the time, a state machine to control the overall game, a segment display converter, and a top module. We began our lab by designing the counter module.

Counter:

The counter module used in this lab was the same as in previous labs, wherein we implemented a 5 bit counter. For this lab we required a 4-bit counter, and in order to implement it we adjusted the logic for the 5 bit counter created in a previous lab, simply deleting the last flip flop and therefore the most significant bit. The counter was used twice, once in the score board for keeping track of turkey crossings, and once in a second counter to display how many seconds a sensor was being blocked.

Time counter:

For the time counter, we needed a counter that counted up until 4 seconds if a sensor was blocked and no change was occurring. Once the sensor reached 4 seconds it was made to flash to signify that no activity had occurred for 4 seconds. The time counter was simply implemented as a 4 bit counter, where up count events only incremented the count if the current count was less than 4. This was done by ANDing the up count input line with the inverse of the 3rd most significant bit. As long as the bit was low, i.e. the count had not reached 4, then the up count line behaved as expected. However once the number 4 was reached it blocked the up count line and made it unable to increase the count. The bit line corresponding to the binary value 4 was also output from the counter to signify that flashing should begin. Given this information, flashing was taken care of in the top module.

Score board:

The next thing the 4 bit counter was used for was to implement a score counter to keep track of turkey crossings. The scoreboard needed to count in both positive and negative, as a crossing in one direction corresponded to an increase in count and a crossing in the opposite direction a decrease. The challenge with this module was to display the negative number. We had already created a counter with the ability to roll under, however the roll under would now need to represent a negative number as opposed to a maximum positive number. The score counter took as input an increment/decrement input and output a 7 bit number as well as the sign of the number. We chose to implement 1's compliment to show the negative number, as this corresponded better to the segment display hardware. Given that the most significant bit represented the sign of the number, the sign output was simply the MSB. In order to convert the negative 1's compliment number into a positive representation, we derived the following identity. Given the 7 bit number A, the output was:

$$out = (7\{A[7]\} \otimes A) | (000000\,\hat{A})$$

Which was the XOR of the 7 bit extension of the topmost bit in A with itself, ORed with a bitmask of length 7, where the last bit is the AND of the entire number A. This essentially simulated 2's compliment conversion. By XORing with the top bit the bits were flipped only if the top bit was a 1, i.e. if the number was negative. ORing with the bitmask simulated addition of 1 and eliminated the duplicate 0 issue with 1's compliment numbers. The bit mask only had a 1 in the LSB in the case A was all 1's, i.e. when the number rolled over.

Segment display:

The segment display used in this lab was the same display as used in previous labs, with the slight modification of an added negative bit option. An added input was added to display a negative sign, wherein if the negative line was driven high it inactivated whatever output logic was being calculated in the segment display,

and only the central segment was activated, corresponding to a negative sign output. All other functionality of the segment display worked as normal.

State machine:

A state machine was implemented to control the behavior of the circuit and coordinate the output and component behavior. A diagram of the state machine is presented in the results section. It implemented one hot encoding to keep track of 5 states, and took in the 2 sensors as input and output 4 control lines: increment, decrement, reset counter, and show counter. The 5 states and their input logic are presented below:

State	Flip flop	Input logic
Idle	I	$\bar{L}\,\bar{R}$
Right	R	$\bar{L}\,R\,(I+Rtl+R)$
Right crossing	Rtl	$L\,(R+Rtl)$
Left	L	$L\,\bar{R}\,(I+L+Ltr)$
Left crossing	Ltr	$R\,(L+Ltr)$

Given this logic, the circuit transferred between the correct states at each given time, and output on the lines as a function of the current state and the input. The logic for the output lines were derived as such:

Output	Logic
Decrement	$Rtl\,\bar{L}\,\bar{R}$
Increment	$Ltr\,\bar{L}\,\bar{R}$
Reset counter	$Idle+RL+Rtl\,\bar{L}\,R+Ltr\,R+Ltr\,L\,\bar{R}$
Show counter	$\overline{Idle}$

Top:

Finally, given all the pieces a top module was created to wire them all together. In it the buttons corresponding to the sensors were input into the state machine as left and right sensors, and tied to leds 15 and 8 respectively. The output of the state machine was then connected to the time and score counters. The increment and decrement sensors were connected to the score counter, and the show and reset time were connected to the time counter. The output of the score was input into a 16 bit selector as the lowermost 8 bits. The time counter was also output onto the ring counter corresponding to the upper most 4 bits. The middle 4 bits of the selector were left blank to adjust for the negative sign.

Given the ring counter, the sign bit from the score counter was ANDed with the 2nd most significant bit in the ring counter and fed into the segment converter. Therefore, when there was a negative number and the ring counter was high on the bit for the middle segment, the display converter would output a negative sign number for display and nothing otherwise. The time counter output was ANDed with the show time output from the state machine so that it would only display if the show time option was high. Finally, the flashing of the timer was implemented using a quarter second timer. Given that we had a qsec timer, we created a half and full second signal by chaining together 4 flip flops in a ring and outputting a high for every full and half rotation. The half rotation corresponded to a half second and was XORed with the ring counter bit at the high bit anode to implement a flashing function. The full rotation output was fed into the counter so as to increment the time counter for every second.

Results:

Score counter:

The score counter consisted of 2 parts: the physical counter module and a converter to create a negative number display. The hardest part of the scoreboard was creating the logic to convert from a negative representation into a positive one. Several design iterations were considered before the final formula was created. It was noted that a quick conversion from 2's compliment could have been implemented in a separate module, however doing so would have necessitated an adder as well as a bit flipper, adding much more complexity to the score counter. In the end the formula was devised an bugs fixed such that the score counter was implemented.

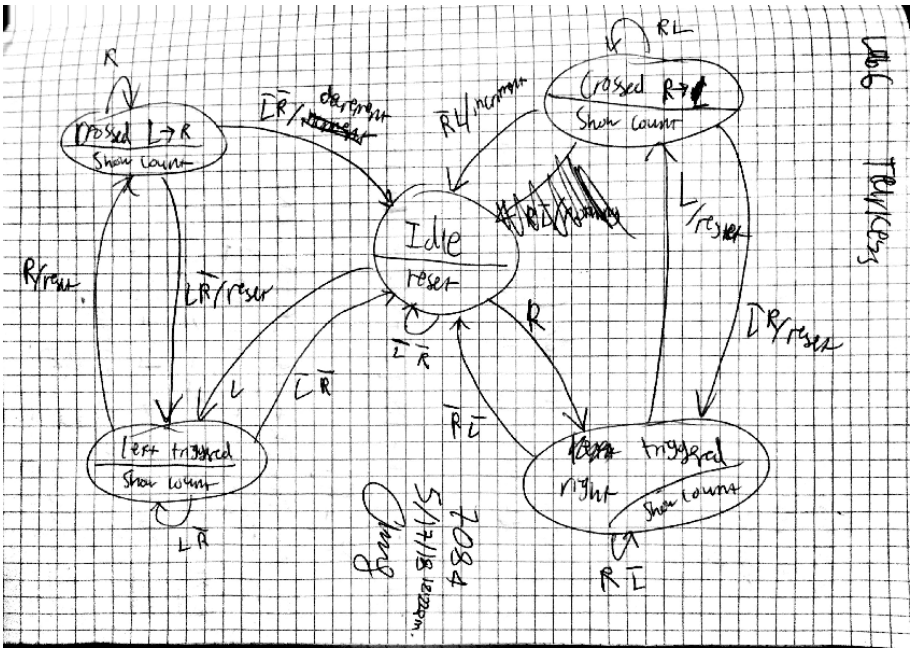
Time counter:

Creating the time counter was fairly straight forward, and the difficulty in implementing it was creating the second signal to control advancement, and implementing the flash signal. The time counter was essentially just an unmodified 4 bit counter that disabled increment at a certain range. There was some issue on the first iteration in stopping the counter at the correct value, but though bug fixing and simulation we were able to create workable logic.

Segment Display:

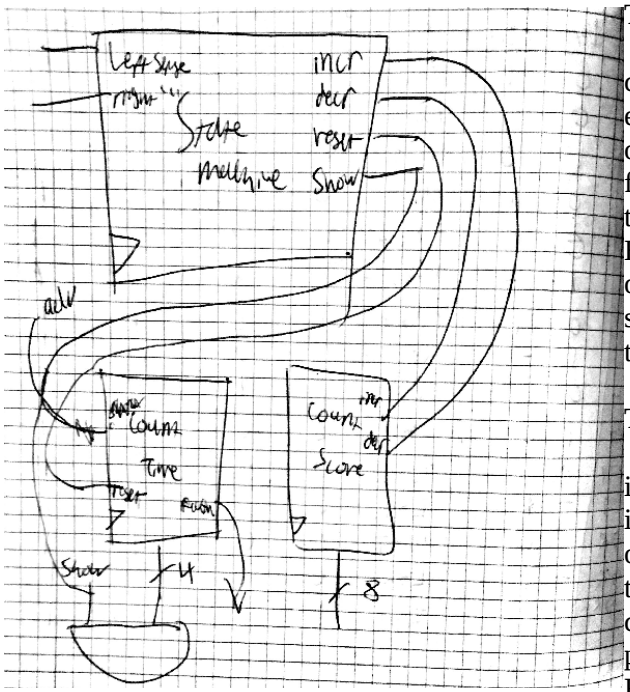
The main logic for the segment display was kept constant for this lab, with the only difference being a new negative sign function that could display a dash on command. There were multiple ways to display the negative sign for the lab, however it was decided that the simplest method was to do so in the segment display. There was an initial issue with the output sequence for the dash and the first iteration was inverted, outputting a zero instead of the correct format. However the output was fixed and the correct dash was shown on the display.

State machine:



A diagram for the state machine is shown adjacently. It began in the Idle state, where the reset counter pin was kept high and no increment, decrement, or show time pins were active. It remained in the Idle state until the right XOR left sensor was activated. If both were active or inactive at the same time the machine remained in the idle state. When the right button was pressed the machine went into the Right state to signify a right sensor trigger. In this state the time counter pin was active and the time was shown high. The machine stayed in this state while the right button was pressed and the left unpressed. Upon pressing the left button the machine transferred states into the Rtl state. In the transfer the reset count pin was set to reinitialize the 4 second counter, and the show count pin was again kept high. The machine remained in this state while both the left and right button were pressed. If the left button became inactive with the right active, this was taken as the turkey changing direction and the machine transferred states back into the Right state, awaiting another crossing. In the event the right button was unpressed while the left button was active this represented a full crossing though the sensors and an increment signal was sent out for one clock cycle as the machine transferred states from Rtl back into Idle again. This sequence of state transfers was repeated for the Left triggered portion. All the logic was kept essentially the same, with the only difference being the mirroring of the button pressing.

In this case, control was transferred from Idle on the event of a left button press. The same 2 stage intermediary was kept to determine a full left to right crossing, and in transferring control back to idle a decrement signal was sent out for one clock cycle.



Top:

The top module was implemented and tested once all component modules were created. It worked well after some error testing, and some logic was done to interpret the outputs of different modules to create the desired behavior, such as flashing the count or setting segments. The most difficult part of the top module was organizing all the inputs and outputs. However once the modules were connected together, it was only a matter of testing to get the final output working. A schematic of the connections between all the modules within the top module is shown to the left.

Testing and simulation:

Testing an simulation modules were created for each individual module to perform unit testing in accordance with incremental design. When testing the modules, edge cases and overall behavior was tested, however main focus was paid on the score display and the state machine. After all modules were created and verified, an overall test for the top module was performed. In it a full instance of a turkey crossing was done. First the turkey crossed all the way both left and right, then intermittent turkey crossings were done where the turkey made it half way through then changed minds and switched back. There were some glitches in the game, but full performance was eventually achieved.

Conclusion:

Overall the lab was interesting and we got a better feel for implementing a state machine and combinatorial logic. The lab was much easier having done the chicken game previously, and creating the state machine and components to interact with it were much more painless. If it were to be repeated the state machine could be improved by using regular flip flop coding for the states, as opposed to one hot encoding. Overall the lab was the best thing I’ve ever done in my whole life, and I look forward to counting turkeys for years to come!

Appendix

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 05/15/2018 06:28:21 AM
// Design Name:
// Module Name: countTIME
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

//4 second counter
//output flash at 4 seconds
//feed in seconds on adv pin

module countTIME(
    input adv,
    input clk,
    input reset,
    output flash,
    output [3:0] out
    );

    wire whenIsaySo;
    //incriment on step and when less than 4
    assign whenIsaySo = ~out[2]&adv;
    assign flash = out[2];

    countUD4L countshit (.clk(clk), .up(whenIsaySo), .down(1'b0), .ld(reset), .din(4'b0), .Q(o
ut));
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 05/15/2018 06:15:21 AM
// Design Name:
// Module Name: fuckingsteak
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


//fucking statemachine nutsack for the turkey bitches

module fuckingsteak(
    input clk, //fucking clock dipshit
    input btnl, //left sensor
    input btnr, //right sensor
    output incr, //incriment count
    output decr, //decriment count
    output reset, //reset 4 sec timer
    output show //show 4 sec timer
    );

    //output
    wire idle, right, left, rtl, ltr;

    //input fucker
    wire idlein, rightin, leftin, rtlin, ltrin;

    assign idlein = ~btnl&~btnr;
    assign rightin = ~btnl&btnr&(idle | right | rtl);
    assign rtlin = btnl&(right | rtl);
    assign leftin = btnl&~btnr&(idle | left | ltr);
    assign ltrin = btnr&(left | ltr);


    //fucking up the output bitch
    assign decr = rtl&~btnl&~btnr;
    assign incr = ltr&~btnl&~btnr;
    assign reset = idle | right&btnl | rtl&~btnl&btnr | left&btnr | ltr&btnl&~btnr;
    assign show = ~idle;

    FDRE #(.INIT(1'b1) ) idling (.C(clk), .R(0), .CE(~clk), .D(idlein), .Q(idle));
    FDRE #(.INIT(1'b0) ) TRIGGEREDr (.C(clk), .R(0), .CE(~clk), .D(rightin), .Q(right));
    FDRE #(.INIT(1'b0) ) TRIGGEREDl (.C(clk), .R(0), .CE(~clk), .D(leftin), .Q(left));
    FDRE #(.INIT(1'b0) ) tighttoleft (.C(clk), .R(0), .CE(~clk), .D(rtlin), .Q(rtl));
    FDRE #(.INIT(1'b0) ) lefttotight (.C(clk), .R(0), .CE(~clk), .D(ltrin), .Q(ltr));
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 04/26/2018 01:19:40 PM
// Design Name:
// Module Name: countUD4L
// 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 countUD4L(
    input clk, //clock
    input up, //up event
    input down, //down event
    input ld, //load in switch event
    input [3:0] din, //switch preset
    output utc, //shits all one
    output dtc, //shits all 0
    output [3:0] 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 [3:0] 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]);


    //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]);
    //assign final ooutput as choosing between up or down bus or set switches
    assign D = {4{up}}&Du | {4{down}}&Dd | {4{ld}}&din;

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

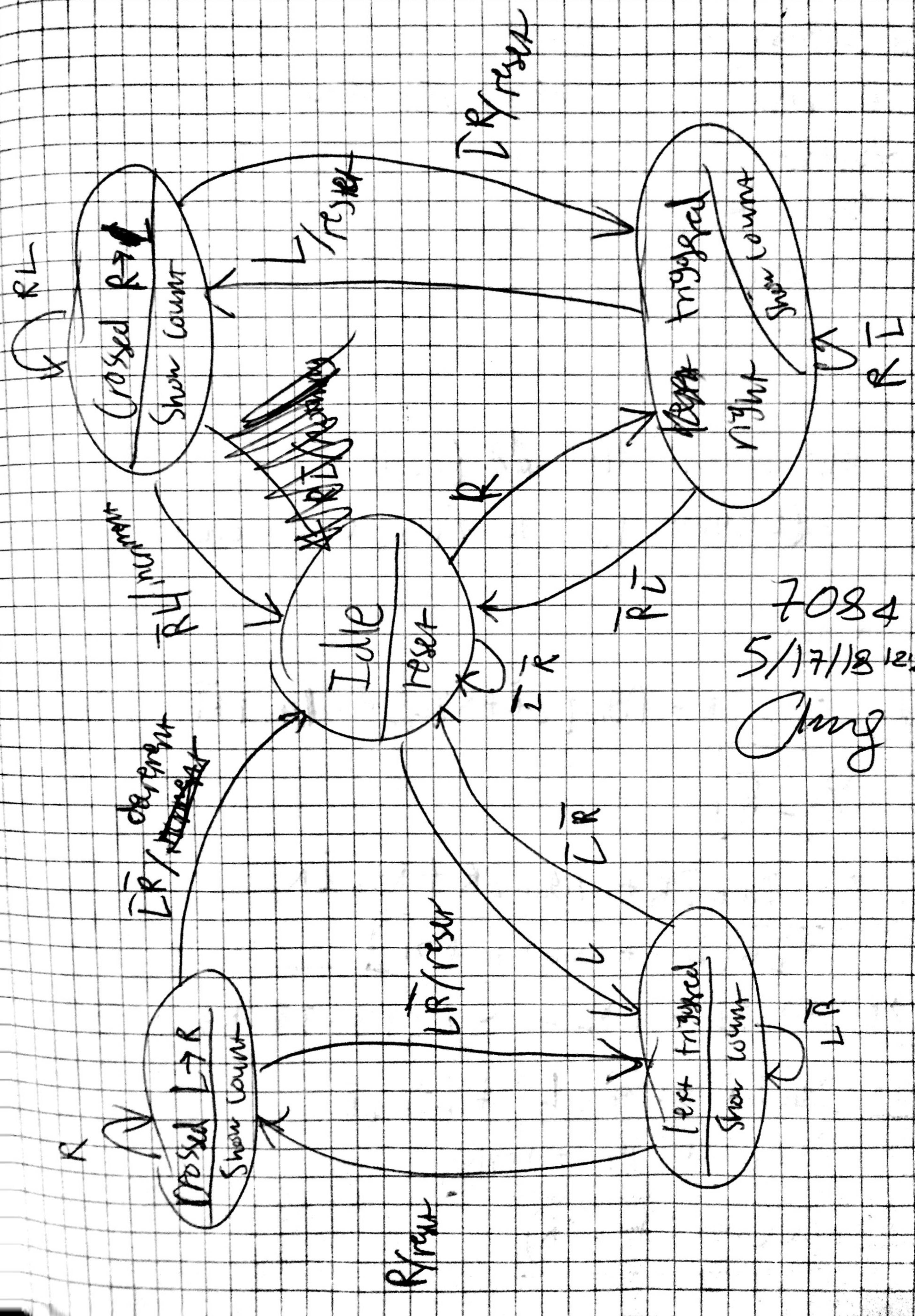
```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 05/15/2018 06:32:44 AM
// Design Name:
// Module Name: score
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


//counter of 8 bits
//goes up or down depending on incr or decr
//outputs a 7 bit number with corresponding negative or not

module score(
    input incr,
    input decr,
    input clk,
    output neg,
    output [7:0] out
    );

    //carry shit
    wire ass, taint, shit;

    //actual number
    wire [7:0] lewd, lower;

    //upper 4 bits
    countUD4L low (.clk(clk), .up(incr), .down(decr), .ld(1'b0), .din(4'b0),
        .utc(ass), .dtc(taint), .Q(lewd[3:0]));

    //lower 4 bits
    countUD4L upper (.clk(clk), .up(ass&incr), .down(taint&decr), .ld(1'b0), .din(4'b0), .Q(le
wd[7:4]));

    //if a neg number, bc in 2's compliment 1st bit will show some shit
    assign neg = lewd[7];

    //convert from 1's compliment
    assign shit = &lewd;

    //intermdtiatary to hold actual number
    assign lower = ({7{lewd[7]}} ^ lewd) | {6'b0,shit};

    //actual number
    assign out = {1'b0, lower[6:0]};



endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 05/15/2018 01:24:34 PM
// Design Name:
// Module Name: TIMEBITCH
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


//counter that incriments on adv

module TIMEBITCH(
    input clk,
    input adv,
    input reset,
    output halfsec,
    output second
    );

    wire q0, q1, d1, fuck, shit;

    assign d1 = ~q1&q0 | q1&~q0;

    //first bit of counter
    FDRE #(.INIT(1'b1) ) quart0 (.C(clk), .R(1'b0), .CE(adv), .D(~q0), .Q(q0));


    //fuckking second
    FDRE #(.INIT(1'b0) ) quart1 (.C(clk), .R(1'b0), .CE(adv), .D(d1), .Q(q1));

       //halfsec on 01 or 11
       //constant output for halfsec
     assign halfsecin = ~q1&q0 | q1&q0;

     //second on 11
     assign secondin = q1&q0;

    //state flop for half sec
    FDRE #(.INIT(1'b0) ) halfshell (.C(clk), .R(reset), .CE(~clk), .D(halfsecin|fuck), .Q(fuck
));

    //edge detector for sec
    FDRE #(.INIT(1'b0) ) sex (.C(clk), .R(1'b0), .CE(~clk), .D(secondin), .Q(shit));



    //only high for singel cock cycle
    assign second = ~shit&secondin;
    assign halfsec = shit;
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 05/15/2018 06:48:15 AM
// Design Name:
// Module Name: topdawg
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module topdawg(
    input clkin,
    input btnR,
    input btnD,
    input btnL,
    output [6:0] seg,
    output [3:0] an,
    output led1,
    output led2
    );

    wire clk, digsel, qsec, incr, decr, reset, show, neg, flash, leftb, rightb;
    //on for half sec and full second respectivly dipshit
    wire halfsec, second;
    wire [3:0] sec, ring, n;
    wire [7:0] dipshit;
    wire [15:0] display;

    assign display = {sec, 4'b0000, dipshit};

    assign led1 = btnR;
    assign led2 = btnL;
    //assign segment on shit
    assign an = ~{show & ring[3] & (1'b1^(flash & halfsec)), ring[2]&neg, ring[1:0]};


    //need to do logic for second and halfsecond
    TIMEBITCH foat (.clk(clk), .adv(qsec), .halfsec(halfsec), .second(second), .reset(reset));

    //cock
    lab6_clks slowit (.clkin(clkin), .greset(btnD), .clk(clk), .digsel(digsel), .qsec(qsec));

    //state machine
    fuckingsteak beefbitch (.clk(clk), .btnl(btnL), .btnr(btnR), .incr(incr),
         .decr(decr), .show(show));

    //holds score
    score fuckyou (.incr(incr), .decr(decr), .clk(clk), .neg(neg), .out(dipshit));

    //holds 4 second timer
    edgeD rightness (.clk(clk), .sh(btnR), .it(rightb));
```

```
    edgeD leftness (.clk(clk), .sh(btnL), .it(leftb));

    countTIME dracula (.adv(second), .clk(clk), .reset(leftb|rightb), .flash(flash), .out(sec)
);

    //ring
    ringcounter ringonit (.adv(digsel), .clk(clk), .o(ring));

    //selector
    selector select (.count(display), .ring(ring), .digit(n));

    //segment converter
    hex7seg lickmydick (.n(n), .neg(ring[2]&neg), .e(1'b1), .out(seg));

endmodule
```

Lab6 Tenkesz



4 outputs:
- increment
- decrement
- reset counter
- show counter

2 inputs
left & right sense

- have counter that counts until it calculates how to push on its own

Inputs (00, 01, 10, 11):

Idle: Idle $\bar{L}\bar{R}$ + right $\bar{L}\bar{R}$ + R→L $\bar{L}\bar{R}$ + left $\bar{R}\bar{L}$ + L→R $\bar{L}\bar{R}$
↳ $\boxed{\bar{L}\bar{R}}$

right: Idle $\bar{L}R$ + right $\bar{L}R$ + R→L $\bar{L}R$
$\boxed{\bar{L}R(\text{Idle} + \text{right} + R\to L)}$

R→L: right L + R→L L $= \boxed{L(\text{right} + R\to L)}$

left: Idle $L\bar{R}$ + left $L\bar{R}$ + L→R $L\bar{R}$
↳ $\boxed{L\bar{R}(\text{Idle} + \text{left} + L\to R)}$

L→R: $\boxed{R(\text{left} + L\to R)}$

CS	Input left	Input right	NS	Output increment	Output decrement	Output reset	Output Show
Idle	0	0	Idle	0	0	1	0
Idle	0	1	right T	0	0	1	0
Idle	1	0	left T	0	0	1	0
right T	0	1	right T	0	0	0	1
right T	0	0	Idle	0	0	0	1
right T	1	–	R→L	0	0	1	1
R→L	0	1	right T	0	0	1	1
R→L	1	–	R→L	0	0	0	1
R→L	0	0	Idle	0	1	0	1
left T	1	0	left T	0	0	0	1
left T	0	0	Idle	0	0	0	1
left T	–	1	L→R	0	0	1	1
L→R	1	0	left T	0	0	1	1
L→R	–	1	L→R	0	0	0	1
L→R	0	0	Idle	1	0	0	1

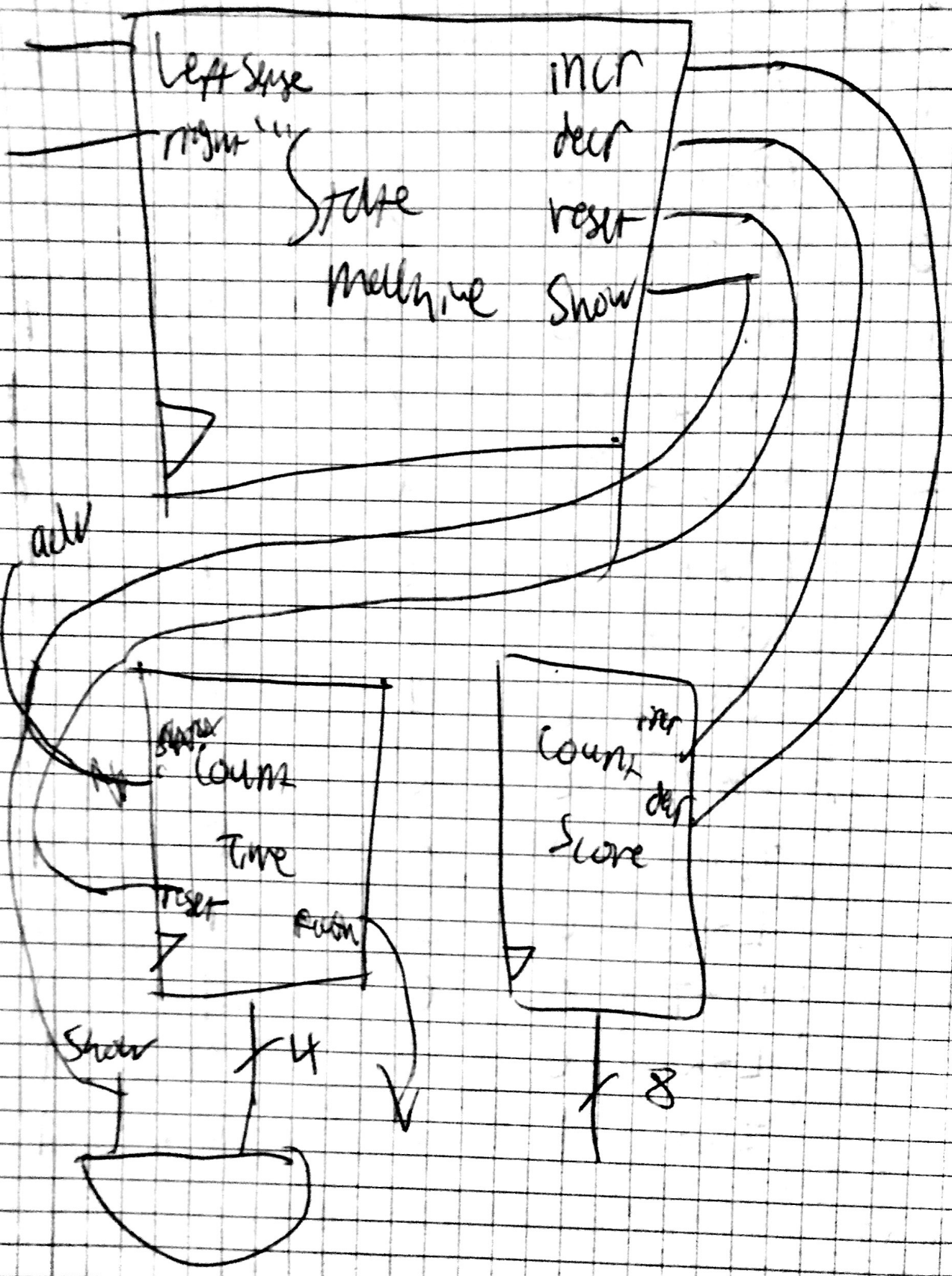
Output logic

decrement = $R \to L \bar{L} \bar{R}$

increment = $L \to R \bar{L} \bar{R}$

reset = $Idle + Right\ T\ L + R \to L \bar{L} R + left\ T\ R + L \to R\ L \bar{R}$

Show = $\overline{Idle}$


left
right
State
machine
incr
decr
reset
show
adv
start
count
time
reset
run
show
4
count
incr
decr
score
8

Score.v

Count time:

input = advance   output = ~~[illegible]~~ 4 bit #
reset
clk

~ ~~stop~~ @ 4 & output flush

Count score =

input = incr   output
decr   8 bit #
clk

= flip using ~~signed magnitude~~ 1's compliment
~ xor bits w/ MSB

111111 = 0

or [illegible] w/ LSB → fixe adding for 11111 edge case

This shit
~~[illegible]~~ = ({7{lend[7]}} ^ lend) ~~[illegible]~~ or (~~[illegible]~~ {6'b0, shit})
performs 2's compliment on MSB = high

shit = & {lend}   ~~[illegible]~~ | ~~lend~~

- make register option for hex 7 seg ✓
- hook up selector, ring ~~display~~ seg

- state

score | count

display

output reg ip

- ring on [2] & reg = high

& ~~display~~ search state

in hex 7 seg

d3p

{ used, ~~reg~~, count } = display => input into selector

blank

need second counter for 4 sec timeout

- & hard see for flash

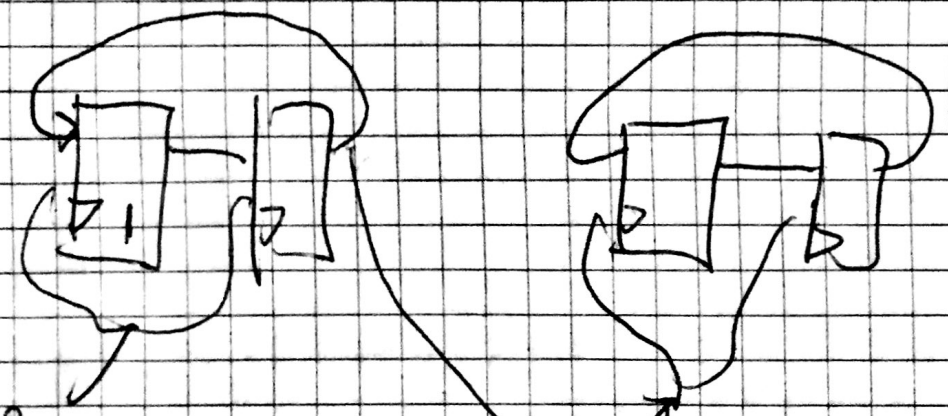
An[0:1] ~~actual on~~ = ring[0:1]

An[2] = one on for reg => ring[2] & reg

An[3] = Count/flash = Show & ring[3] & [(⊕ flash & q sec]

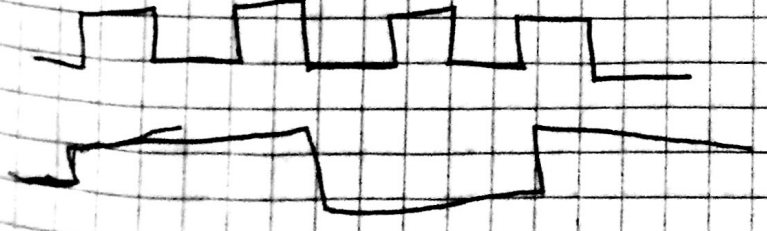
Hour second

Sec

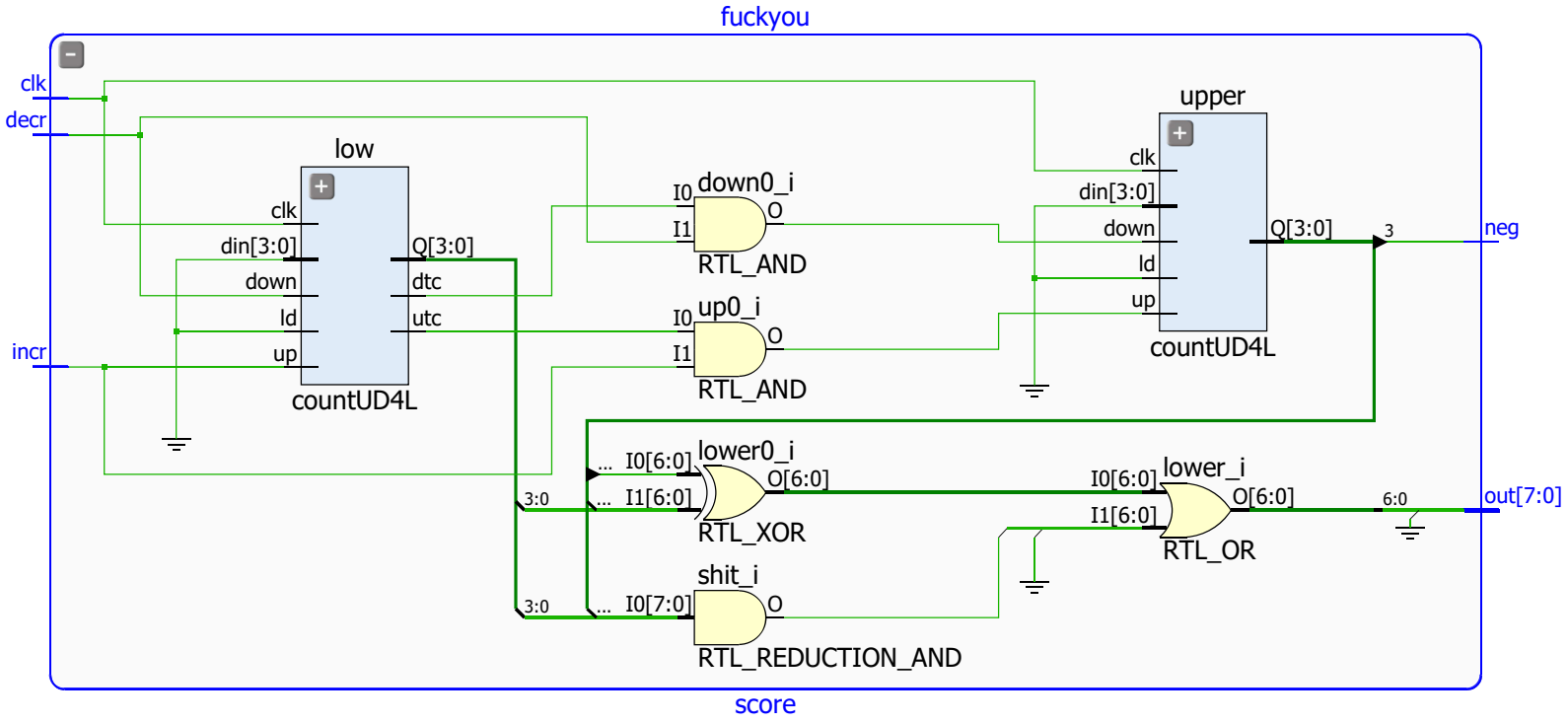


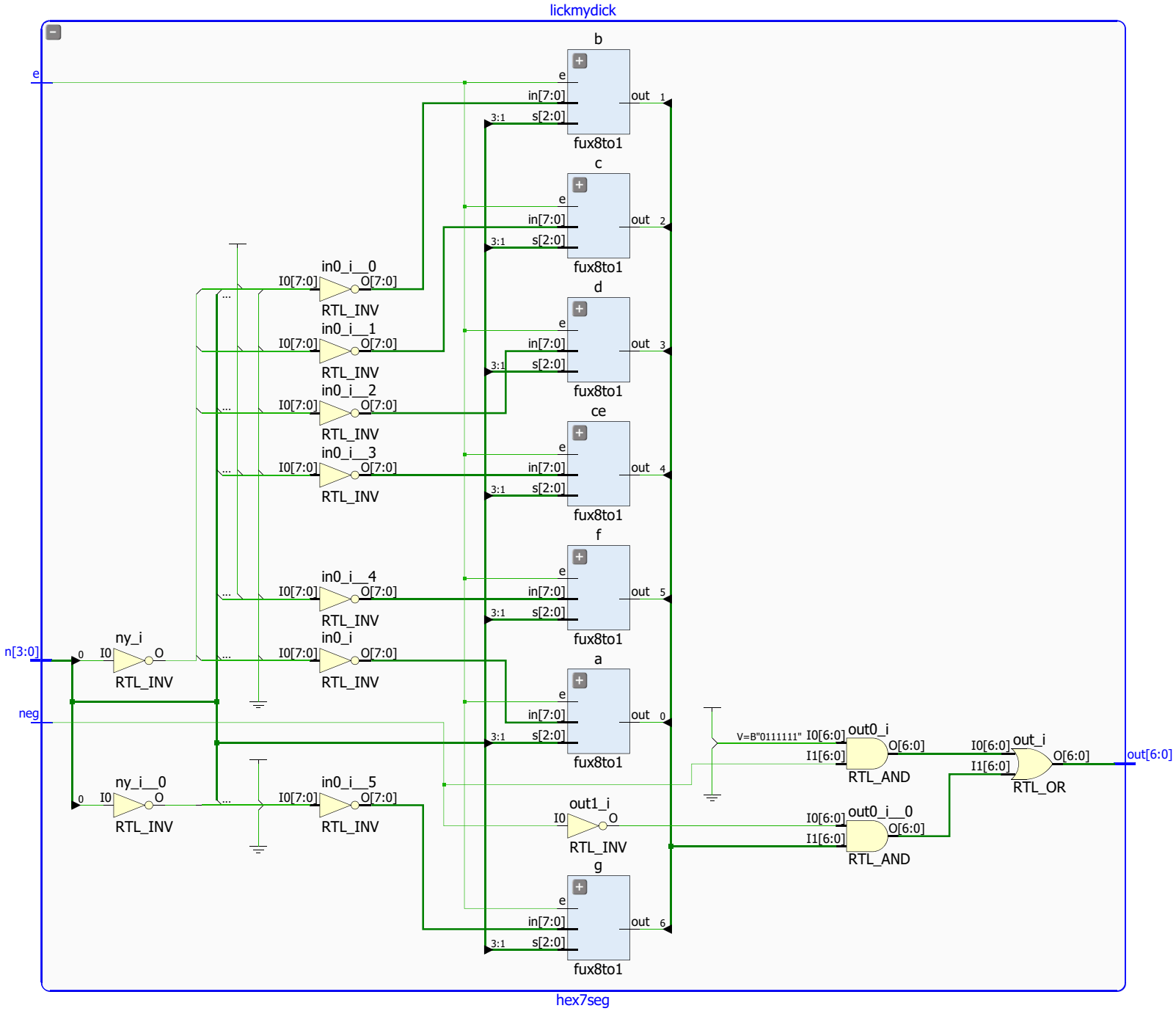
Change on q/sec

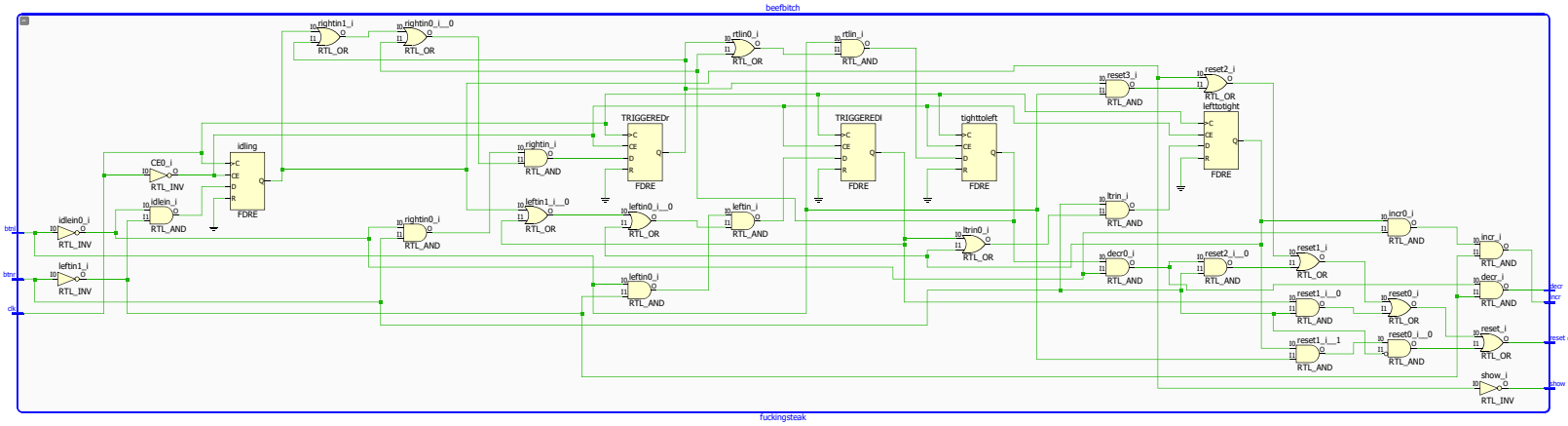
Change on 1/2 sec

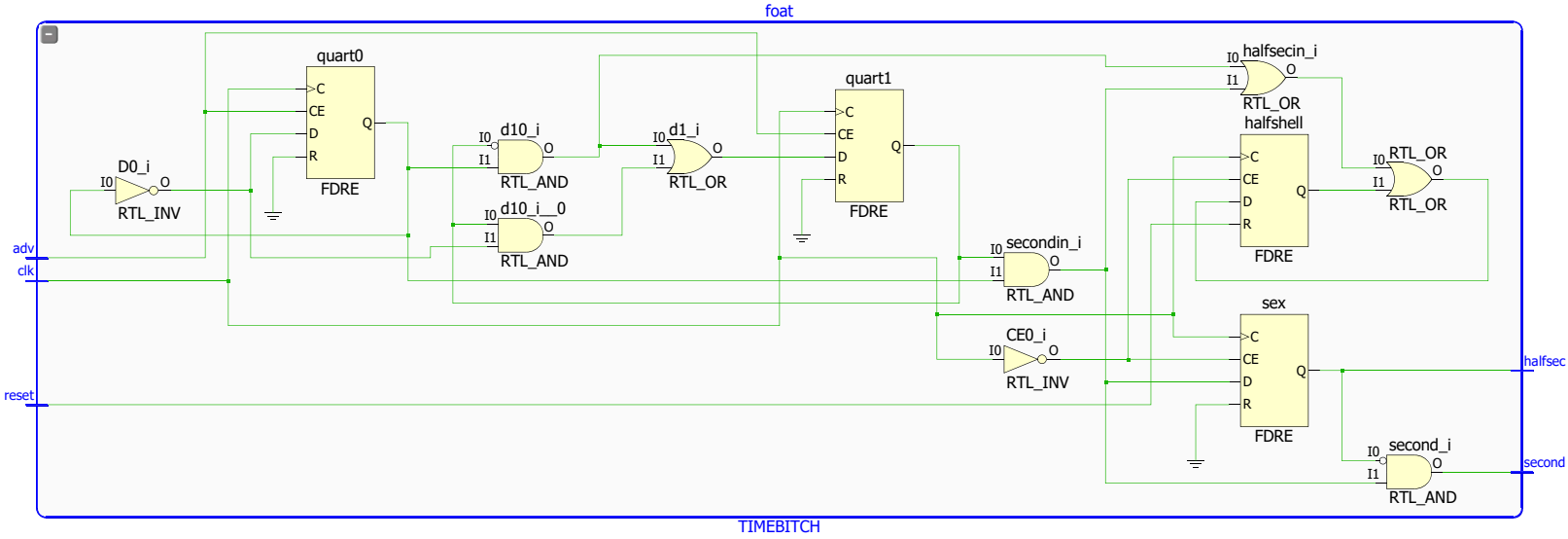


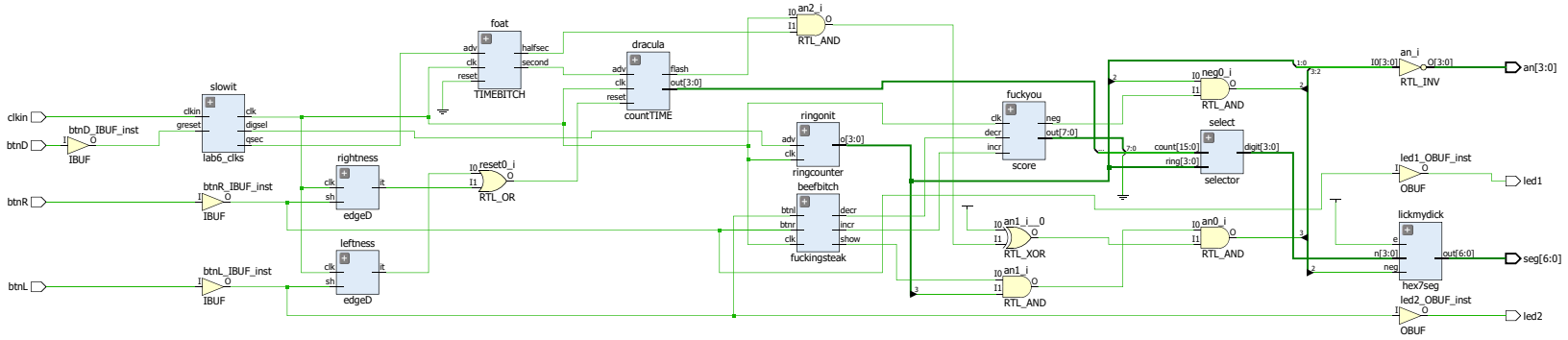
q sec

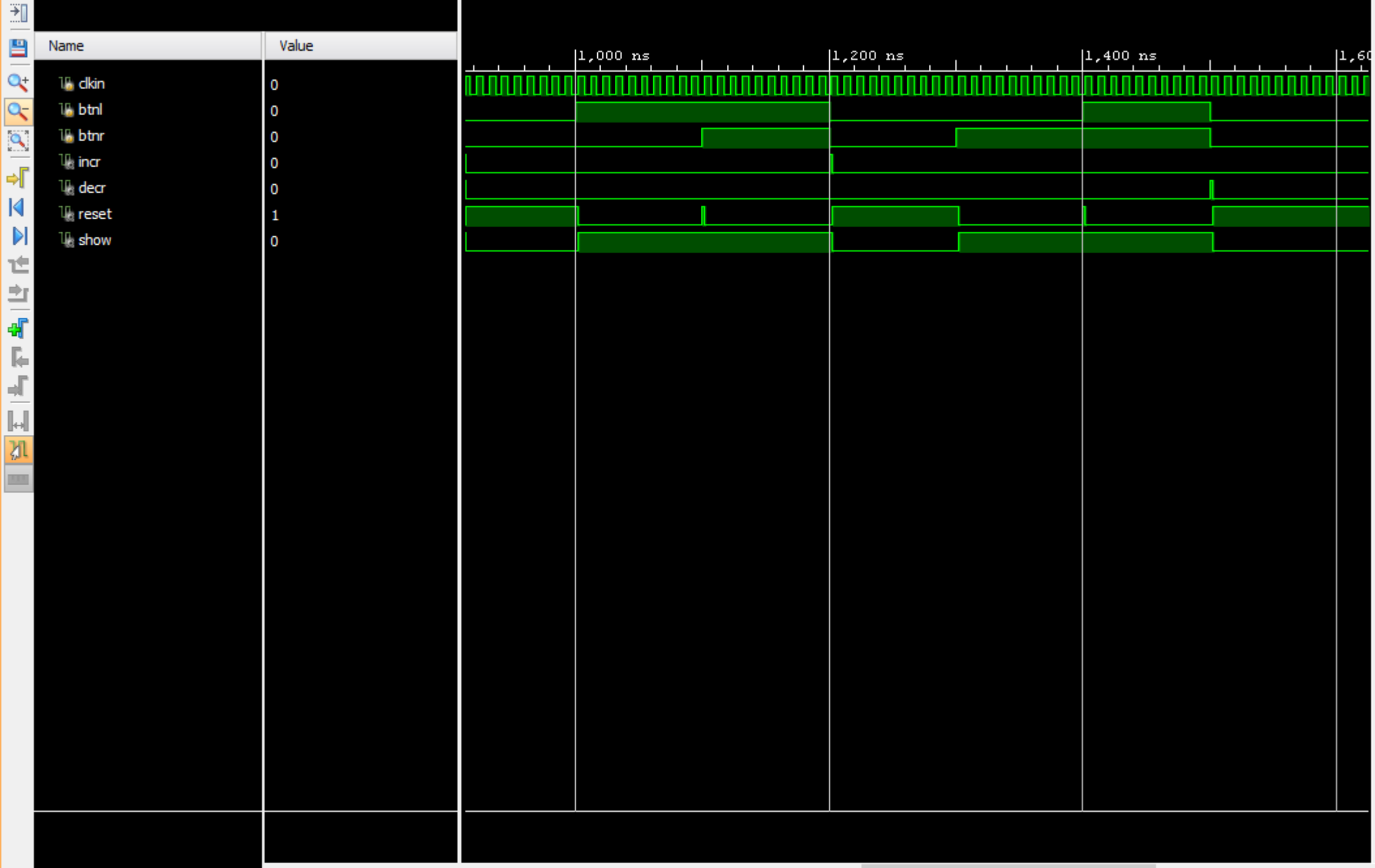

fuckyou
clk
decr
incr
low
clk
din[3:0]
down
ld
up
Q[3:0]
dtc
utc
countUD4L
down0_i
I0
I1
O
RTL_AND
up0_i
I0
I1
O
RTL_AND
upper
clk
din[3:0]
down
ld
up
Q[3:0]
countUD4L
3
neg
lower0_i
I0[6:0]
I1[6:0]
O[6:0]
RTL_XOR
3:0
3:0
shit_i
I0[7:0]
O
RTL_REDUCTION_AND
lower_i
I0[6:0]
I1[6:0]
O[6:0]
RTL_OR
6:0
out[7:0]
score


lickmydick
b
c
d
ce
f
a
g
fux8to1
e
in[7:0]
s[2:0]
3:1
out
in0_i__0
in0_i__1
in0_i__2
in0_i__3
in0_i__4
in0_i__5
in0_i
ny_i
ny_i__0
out1_i
RTL_INV
I0[7:0]
O[7:0]
I0
O
n[3:0]
neg
out0_i
out0_i__0
RTL_AND
V=B"0111111"
I0[6:0]
I1[6:0]
O[6:0]
out_i
RTL_OR
out[6:0]
hex7seg


beefbitch
rightin1_i
RTL_OR
rightin0_i__0
RTL_OR
rtlin0_i
RTL_OR
rtlin_i
RTL_AND
reset3_i
RTL_AND
reset2_i
RTL_OR
lefttotight
FDRE
TRIGGEREDr
FDRE
TRIGGEREDl
FDRE
tighttoleft
FDRE
idling
FDRE
CE0_i
RTL_INV
idlein_i
RTL_AND
rightin_i
RTL_AND
leftin1_i__0
RTL_OR
leftin0_i__0
RTL_OR
leftin_i
RTL_AND
ltrin_i
RTL_AND
ltrin0_i
RTL_OR
incr0_i
RTL_AND
incr_i
RTL_AND
idlein0_i
RTL_INV
rightin0_i
RTL_AND
leftin1_i
RTL_INV
leftin0_i
RTL_AND
decr0_i
RTL_AND
reset2_i__0
RTL_AND
reset1_i
RTL_OR
decr_i
RTL_AND
reset1_i__0
RTL_AND
reset0_i
RTL_OR
reset1_i__1
RTL_AND
reset0_i__0
RTL_AND
reset_i
RTL_OR
show_i
RTL_INV
btnl
btnr
clk
decr
incr
reset
show
fuckingsteak


foat
quart0
C
CE
D
R
Q
FDRE
D0_i
I0
O
RTL_INV
d10_i
I0
I1
O
RTL_AND
d1_i
RTL_OR
d10_i__0
RTL_AND
quart1
FDRE
halfsecin_i
RTL_OR
halfshell
FDRE
RTL_OR
RTL_OR
secondin_i
RTL_AND
CE0_i
RTL_INV
sex
FDRE
second_i
RTL_AND
adv
clk
reset
halfsec
second
TIMEBITCH


clkin
btnD
btnR
btnL
btnD_IBUF_inst
IBUF
slowit
clkin
greset
clk
dlgsel
qsec
lab6_clks
btnR_IBUF_inst
IBUF
btnL_IBUF_inst
IBUF
rightness
clk
sh
it
edgeD
leftness
clk
sh
it
edgeD
foat
adv
clk
reset
halfsec
second
TIMEBITCH
reset0_i
I0
I1
O
RTL_OR
dracula
adv
clk
reset
flash
out[3:0]
countTIME
an2_i
I0
I1
O
RTL_AND
ringonit
adv
clk
o[3:0]
ringcounter
beefbitch
btnl
btnr
clk
decr
incr
show
fuckingsteak
fuckyou
clk
decr
incr
neg
out[7:0]
score
an1_i_0
RTL_XOR
an1_i
RTL_AND
neg0_i
RTL_AND
an0_i
RTL_AND
select
count[15:0]
ring[3:0]
digit[3:0]
selector
an_i
I0[3:0]
O[3:0]
RTL_INV
an[3:0]
led1_OBUF_inst
OBUF
led1
lickmydick
e
n[3:0]
neg
out[6:0]
hex7seg
seg[6:0]
led2_OBUF_inst
OBUF
led2


Name
Value
clkin
0
btnl
0
btnr
0
incr
0
decr
0
reset
1
show
0
1,000 ns
1,200 ns
1,400 ns