Terrence Ng

CSE 100L

Section C

2/25/2020

# Lab 6 Write Up

**Description:**

The goal of this lab is to create a turkey traffic simulator on the basys3 board that detects and records the number of times a turkey crosses from right to left minus the number of times it crosses from left to right. The turkey traffic simulator also displays if a turkey has been blocking the sensors for at least 15 seconds. The purpose of this lab is to allow the class to learn how to create their own state machines and designs from scratch.

**Methods:**

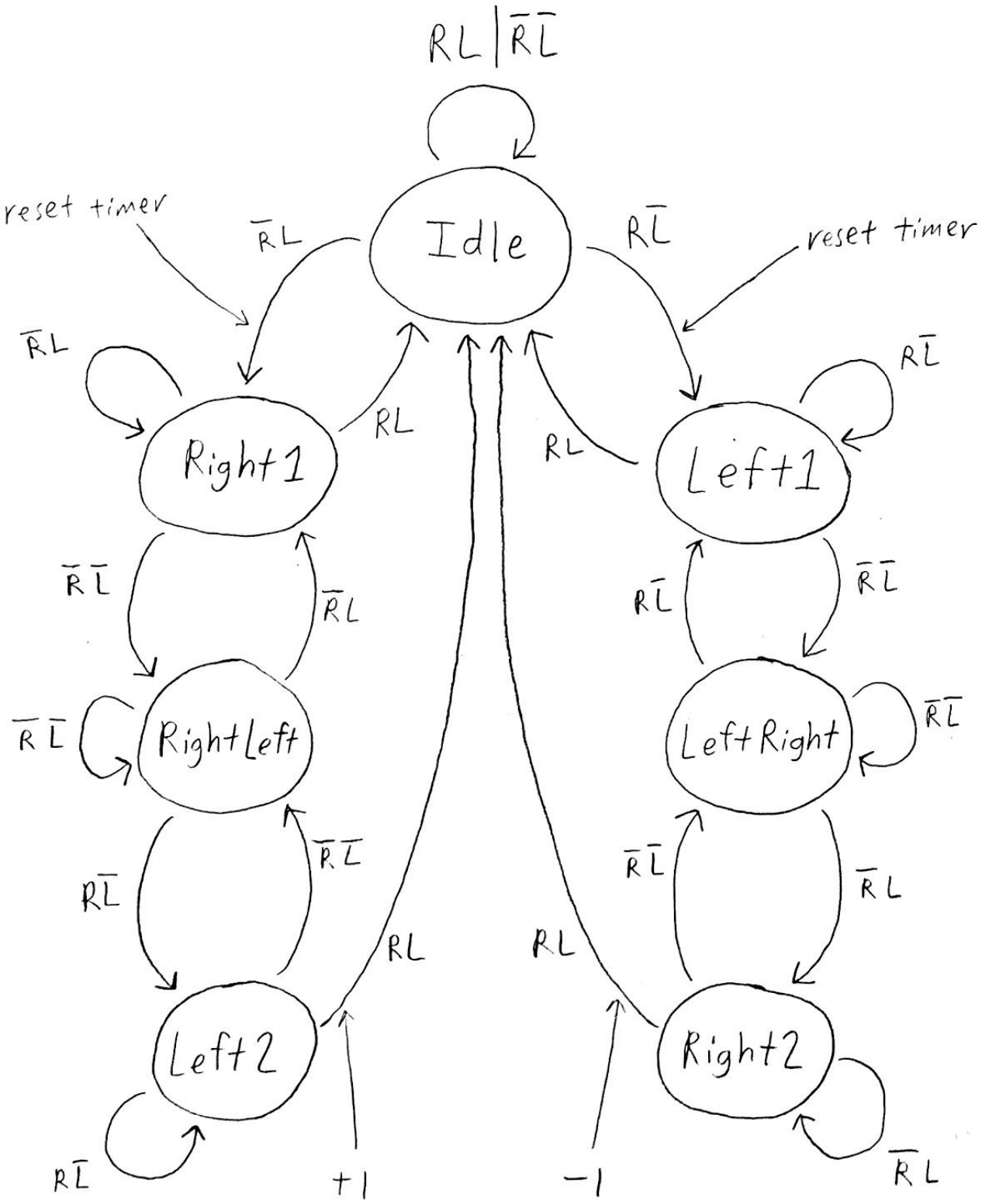
I began by first designing my state machine (Figure A) and the conditions for each state, making sure to account for all possible cases including the turkey crossing all the way in either direction as well as crossing halfway and turning back from either direction. When creating my state machine, I treated a high value as the sensor being unblocked and a low value as the sensor being blocked, this ended up reversing everything but I fixed it by inverting the button L and button R in my top level. Using my state machine, I created the logic needed for reaching each state as well as the outputs of the state machine. I used a Mealy Machine

I then made the turkey counter using two loadable up and down 4 bit counters from lab 4 (countUD4L). Next I created the time counter using the 4 bit up counter I created in lab 5 (Count_4b) and made the count enable low after the counter reached 15. I then made the two's complement module by using 8 full adders that I made in lab 2 to invert the input and add 1.

In my top level I used the ring counter and selector I created in lab 4 and the hex7seg I created in lab 2 to display values on the display. I displayed the time on anode 3 and made the time counter count every second. I displayed the turkey counters output on anode 0 and 1 if the value is positive, I displayed the two's complement of it if it is negative, using a m2_1x8 mux that I created in lab 3. I displayed the negative sign on anode 2 when the turkey counter's output is positive, by inverting the 7 segment display of 0.

**Results:**

**Figure A**



# State Machine

## IDLE

The machine is in state IDLE at the beginning when no sensors are blocked. The machine will remain in state IDLE if both sensors remain unblocked or both sensors are blocked, this is to account for pressing button L and button R at the same time in the beginning. The machine will be in state IDLE when in either state Right1 or Left1 and both sensors are unblocked, this is the case when the turkey has entered the cross path and left in the direction that it entered. The machine will also be in state IDLE when in either state Left2 or Right2 and both sensors are unblocked, this is the case when the turkey has performed a full cross in either direction. The turkey counter is decremented when entering state IDLE from state Right2 and is incremented when entering state IDLE from state Left2. The time counter is not displayed when in state IDLE and resets when leaving this state.

- next_IDLE = IDLE & (rightSen & leftSen) | IDLE & (~rightSen & ~leftSen) | Right1 & (rightSen & leftSen) | Left1 & (rightSen & leftSen) | Right2 & (rightSen & leftSen) | Left2 & (rightSen & leftSen)
- plus1 = Left2 & rightSen & leftSen
- minus1 = Right2 & rightSen & leftSen
- Show_time = ~IDLE
- Reset_timer = (IDLE & ~rightSen & leftSen) | (IDLE & rightSen & ~leftSen)

## Left1

The machine is in state Left1 when in state IDLE and only the left sensor is pressed, this is the case when the turkey enters on the left side. The machine will also be in state Left1 when in state LeftRight and only the left sensor is blocked, this is the case when the turkey has entered on the left side, reached the middle and walked back a little . The machine will remain in state Left1 if only the left sensor is blocked.

- next_Left1 = IDLE & (rightSen & ~leftSen) | Left1 & (rightSen & ~leftSen) | LeftRight & (rightSen & ~leftSen)

## LeftRight

The machine is in state LeftRight when in state Left1 and both sensors are blocked, this is the case when the turkey has entered on the left side and is currently in the middle. The machine will also be in state LeftRight when in state Right 2 and both sensors are blocked, this is the case when the turkey has entered on the left side, walked past the middle, and walked back to the middle. The machine will remain in state LeftRight if both sensors are blocked.

- next_LeftRight = Left1 & (~rightSen & ~leftSen) | LeftRight & (~rightSen & ~leftSen) | Right2 & (~rightSen & ~leftSen)

## Right2

The machine is in state Right2 when in state LeftRight and only the right sensor is blocked, this is the case when the turkey has entered on the left side and walked a little past the middle, not yet completing a full cross. The machine will remain in state Right2 if only the right sensor is blocked.

- next_Right2 = LeftRight & (~rightSen & leftSen) | Right2 & (~rightSen & leftSen)

Right1

The machine is in state Right1 when in state IDLE and only the right sensor is pressed, this is the case when the turkey enters on the right side. The machine will also be in state Right1 when in state RightLeft and only the right sensor is blocked, this is the case when the turkey has entered on the right side, reached the middle and walked back a little . The machine will remain in state Right1 if only the right sensor is blocked.

- next_Right1 = IDLE & (~rightSen & leftSen) | Right1 & (~rightSen & leftSen) | RightLeft & (~rightSen & leftSen)

RightLeft

The machine is in state RightLeft when in state Right1 and both sensors are blocked, this is the case when the turkey has entered on the right side and is currently in the middle. The machine will also be in state RightLeft when in state Left 2 and both sensors are blocked, this is the case when the turkey has entered on the right side, walked past the middle, and walked back to the middle. The machine will remain in state RightLeft if both sensors are blocked.

- next_RightLeft = Right1 & (~rightSen & ~leftSen) | RightLeft & (~rightSen & ~leftSen) | Left2 & (~rightSen & ~leftSen)

Left2

The machine is in state Left2 when in state RightLeft and only the right sensor is blocked, this is the case when the turkey has entered on the right side and walked a little past the middle, not yet completing a full cross. The machine will remain in state Left2 if only the left sensor is blocked.

- next_Left2 = RightLeft & (rightSen & ~leftSen) | Left2 & (rightSen & ~leftSen)

## Turkey Counter

The Turkey counter uses 2 loadable up and down 4 bit counters I created in lab 4 (countUD4L) to keep track of the sum of crosses (left to right right is -1 and right to left is +1). The second counter starts counting when the UTC of the first is high.

## Time Counter

The Time counter uses the 4 bit up counter that I created in lab 5 (Count_4b) to count up and I set the count enable to be low when all 4 bits of the output are high, thus stopping the time at 15.

## twos_comp

The twos_comp module uses 8 full adders that I created in lab 4 to perform 2's complement. To obtain the 2's complement of a number, you need to invert it and add 1. I did this by feeding the inverse of each bit of the input of twos_comp into a full adder and adding 1 to the first bit. The resulting output is the 2's complement of the input.

## CountUD4L

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

## Count_4b

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

## Full_adder

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

## Ring Counter

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

## Selector

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

## hex7seg

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

## m2_1x8

I used the same m2_1x8 that I made in lab 3.

## m4_1

I used the same m4_1 that I made in lab 3.

## Top Level

To create the top level (Figure B), I first wired the inverse of button L to the leftSen input of my state machine and the inverse of button R to the rightSen input. This is because I treated a high value as the sensor being unblocked when creating my state machine, inverting the input fixes the problems caused by this. I then wire the Reset_timer output of the state machine to the R input of the time counter so it can reset at the correct times. I also wired the plus1 and minus1 outputs of the state machine to the Up and Dw inputs of the turkey counter so that it knows when to increment or decrement.

I then use a 4 bit up counter I created in lab 5 (count_4b) to count every quarter second and output 1 when it counts to 4 (1 second) before resetting. The output of this 1 second counter is wired to the CE input of the time counter so that it increments every second. The 4 bit output of the time counter is then wired to bits 12 to 15 of the selector so that the time can be displayed.

Next I set the LD and Din inputs of the turkey counter to a 1 and 8 bit bus of 0's because they will not be used, I set the UTC and DTC outputs to dummy wires for the same reason. I then wire the 8 bit output of the turkey counter into the input of the twos_comp module to obtain the 2's complement of the counter. The 8 bit output of the turkey counter as well as the twos_comp module is wired to the inputs of a m2_8x1 mux with the most significant bit of the output of the turkey counter as the sel input. This outputs the correct turkey counter value depending on if the value is negative (negative if most significant bit is 1). The output of the m2_8x1 mux has its most significant bit set to 0 because it is only a 7 bit number. This value is then wired to bits 0 to 7 of the selector so that the turkey counter value can be displayed.

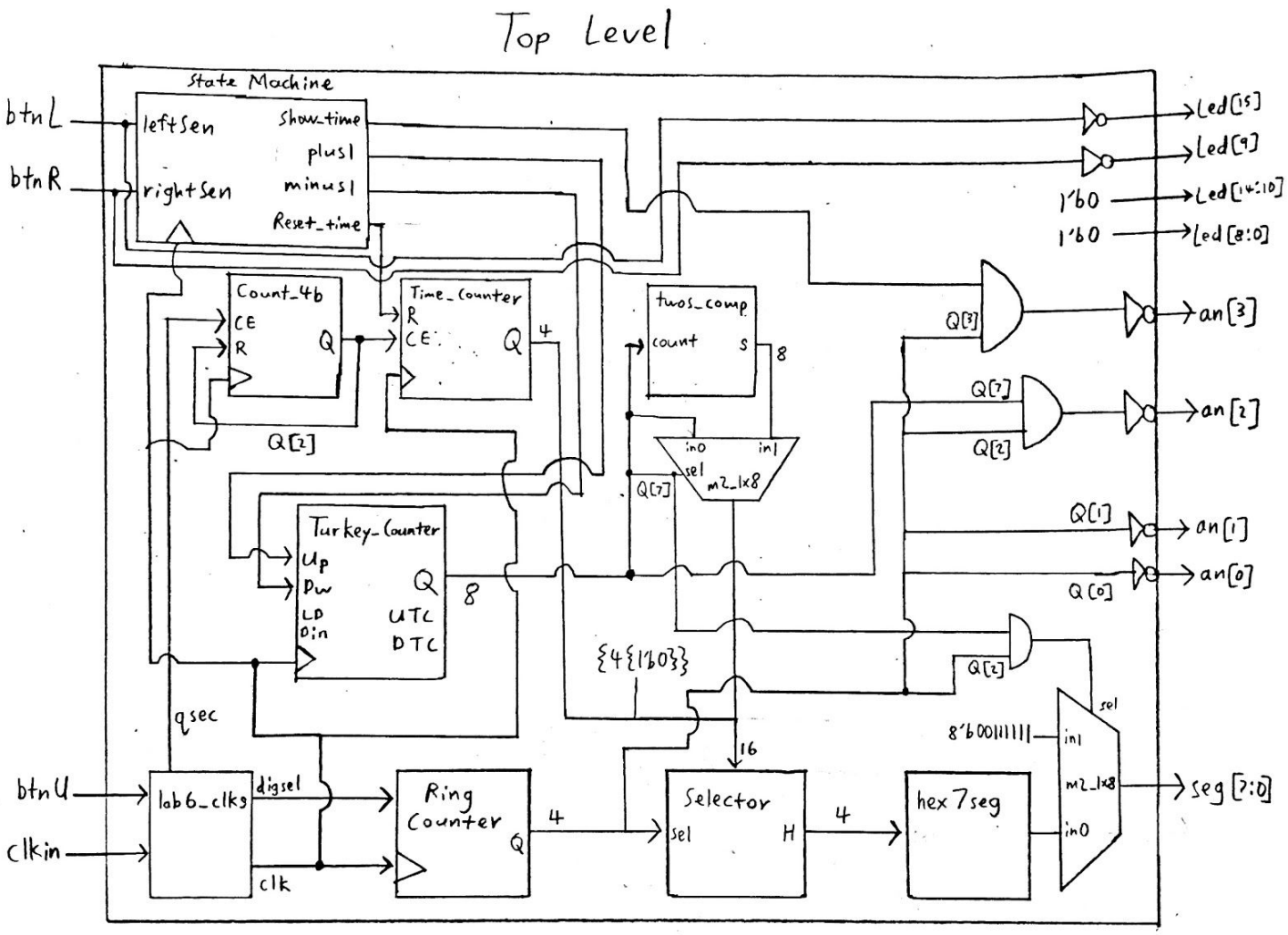
The ring counter takes in digsel and clk as inputs and outputs a 4 bit bus that determines which anode is currently lit up. The selector takes the outputs of the ring counter, time counter, and turkey counter and outputs a 4 bit bus to the hex7seg which displays the corresponding hexadecimal number.

The 8 bit output of the hex7seg is then wired to the input of a m2_1x8 mux with 8'b00111111 being the other input. 8'b00111111 represents the negative sign in the hex7seg (active low). The sel input is bit 2 of the

ring counter output and the most significant bit of the turkey counter output. This mux displays a negative sign when the ring counter lights up anode 2 and the proper value otherwise.

Anode 0 and 1 are on indefinitely. Anode 2 is on when the turkey counter value is negative. Anode 3 is on when the showTime output of the state machine is high. Led 15 is on when button L isn't pressed and led 9 is on when button R isn't pressed.

**Figure B**



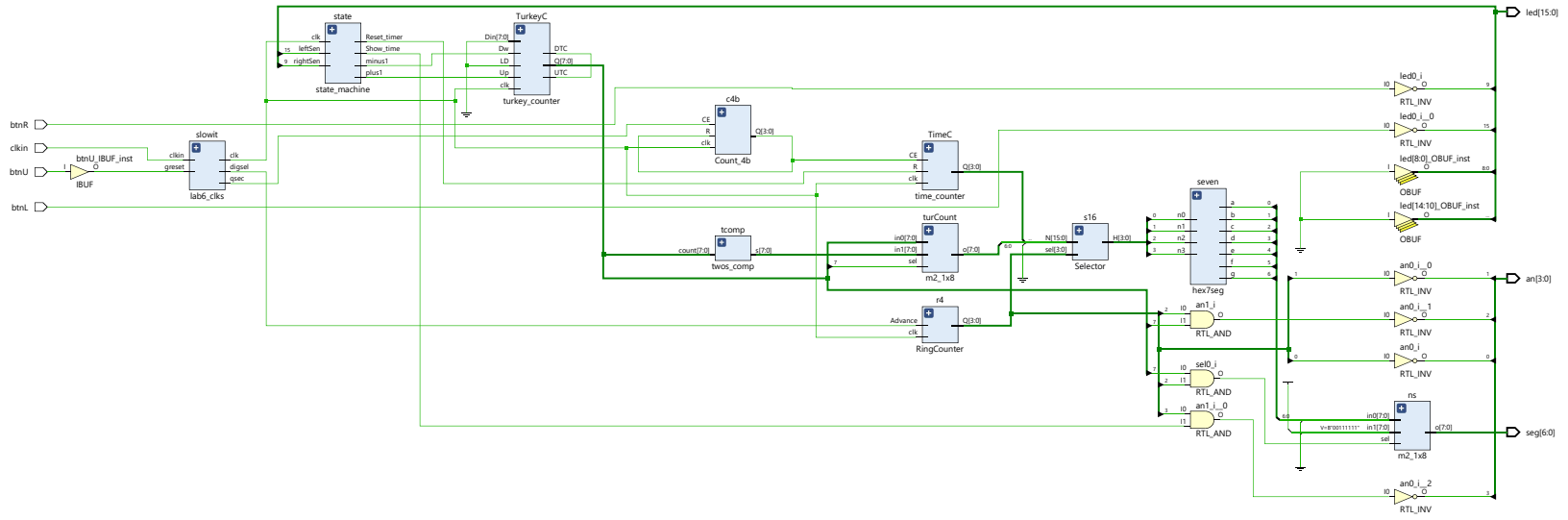

The result of my design was successful. After testing and debugging my code and logic, the board functioned as specified in the lab manual. I tested my design to make sure it worked by simulating it as well as some of it's inner modules such as the state machine. I also tested my design on the basys3 board by playing around with the buttons and switches.

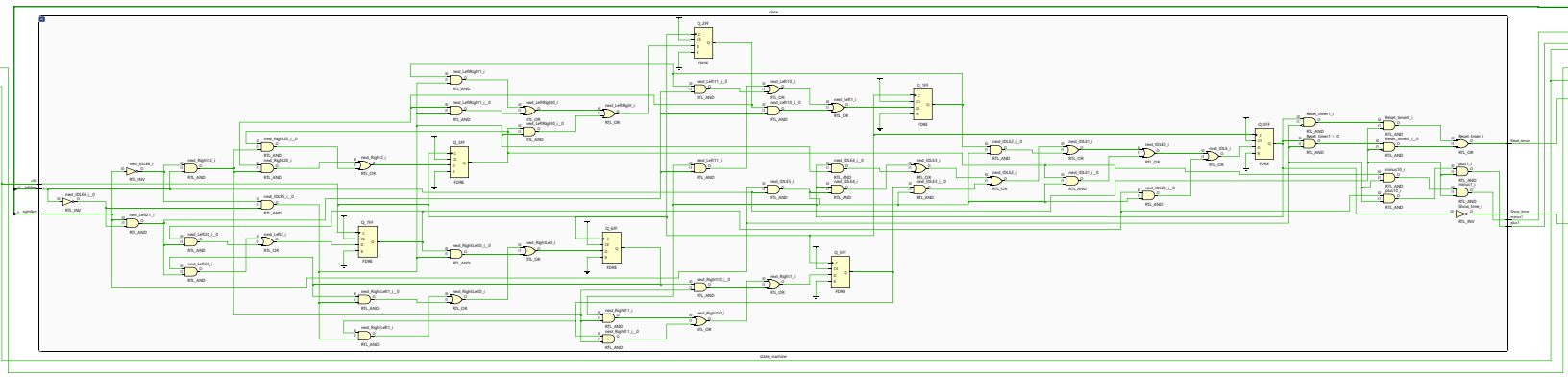
Some problems that I discovered during testing was that everything in my project was reversed and therefore wrong. The reason for this was because when creating my state machine, I treated a high value as a sensor being unblocked and a low value as a sensor being blocked. I realized that the buttons work opposite to

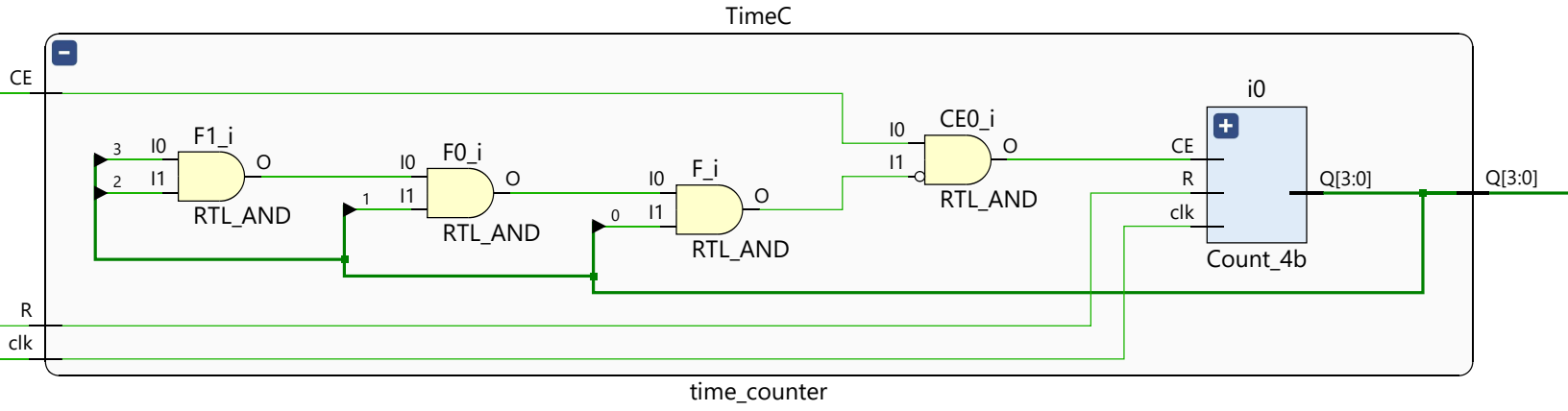
this so I fixed this problem in my top level by inverting  button L and R in my state machine's leftSen and rightSen inputs.  Another problem that I faced was that my time counter would start counting from values other than 0 sometimes, I realized that this was because I was resetting the time counter before entering the IDLE state when I should be resetting it after leaving the IDLE state.

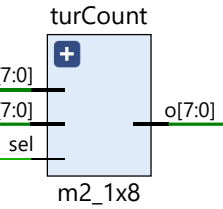
**Conclusion:**

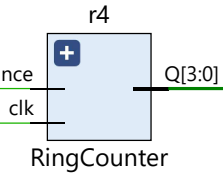
In conclusion, this lab was a good learning experience. I learned a lot especially about creating my own designs from scratch as there were no instructions on how to create the state machine or top level for this lab. I had difficulty creating the state machine because I didn't know how to account for all possible cases in my design at first. If I could do this again, I would definitely completely test my state machine and make sure it works fully for all cases before moving onto my other modules. There aren't any components that I would optimize.

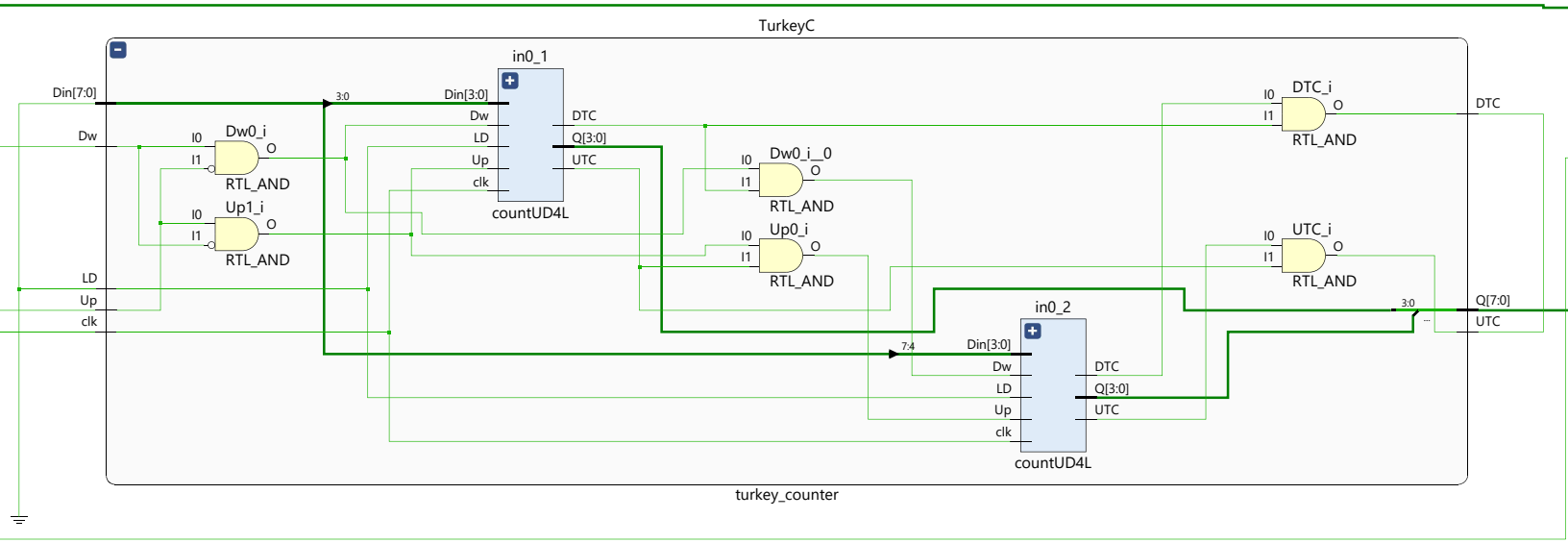

state
clk
leftSen
rightSen
Reset_timer
Show_time
minus1
plus1
state_machine
TurkeyC
Din[7:0]
Dw
LD
Up
clk
DTC
Q[7:0]
UTC
turkey_counter
c4b
CE
R
clk
Q[3:0]
Count_4b
TimeC
time_counter
btnR
clkin
btnU
btnL
btnU_IBUF_inst
IBUF
slowit
clkin
greset
clk
digsel
qsec
lab6_clks
tcomp
count[7:0]
s[7:0]
twos_comp
turCount
in0[7:0]
in1[7:0]
sel
o[7:0]
m2_1x8
r4
Advance
RingCounter
s16
N[15:0]
sel[3:0]
H[3:0]
Selector
seven
n0
n1
n2
n3
a
b
c
d
e
f
g
hex7seg
an1_i
RTL_AND
sel0_i
RTL_AND
an1_i_0
RTL_AND
led0_i
RTL_INV
led0_i_0
RTL_INV
led[8:0]_OBUF_inst
OBUF
led[14:10]_OBUF_inst
OBUF
an0_i_0
RTL_INV
an0_i_1
RTL_INV
an0_i
RTL_INV
ns
in0[7:0]
in1[7:0]
sel
o[7:0]
m2_1x8
an0_i_2
RTL_INV
led[15:0]
an[3:0]
seg[6:0]

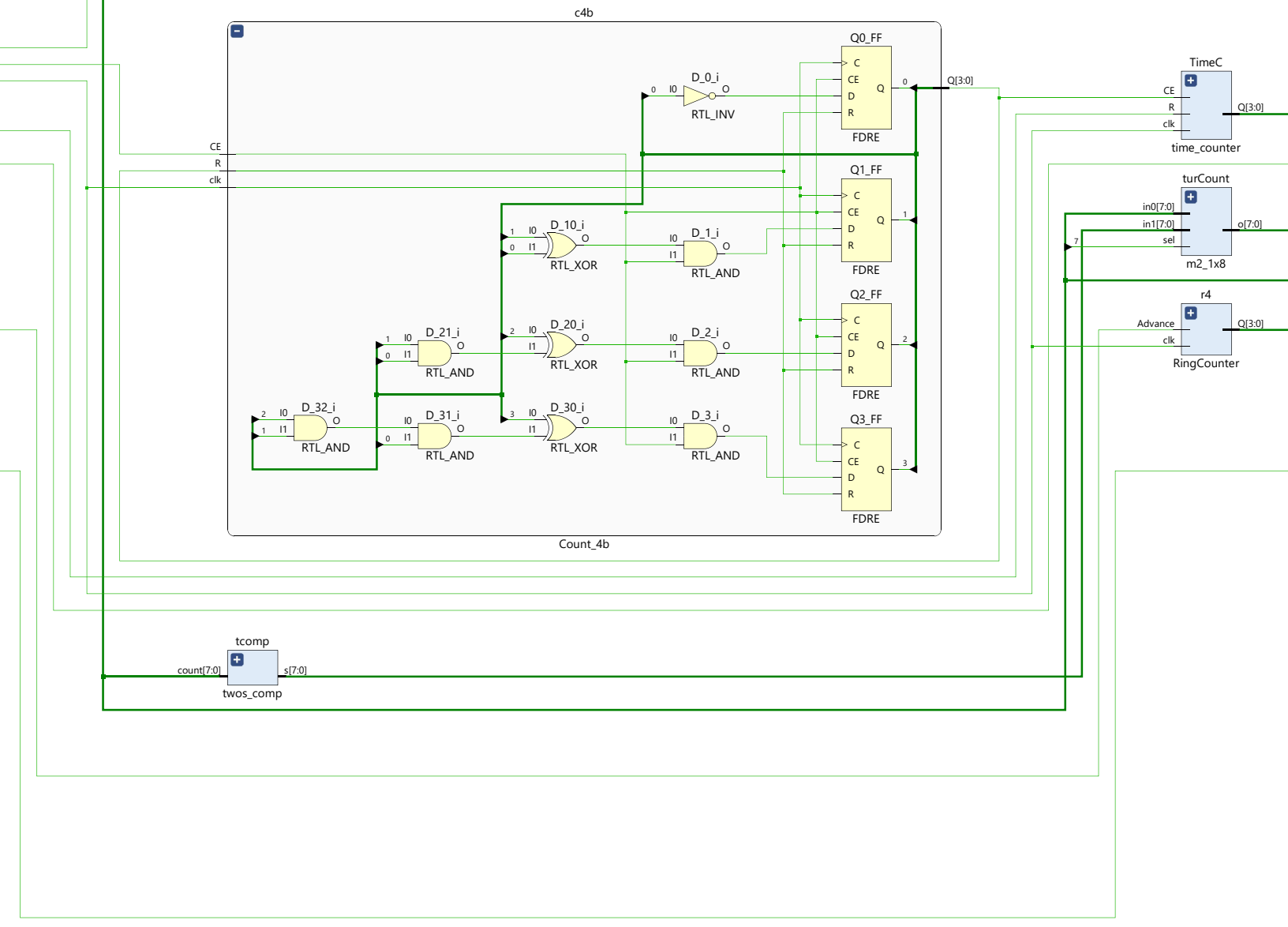

state
state_machine

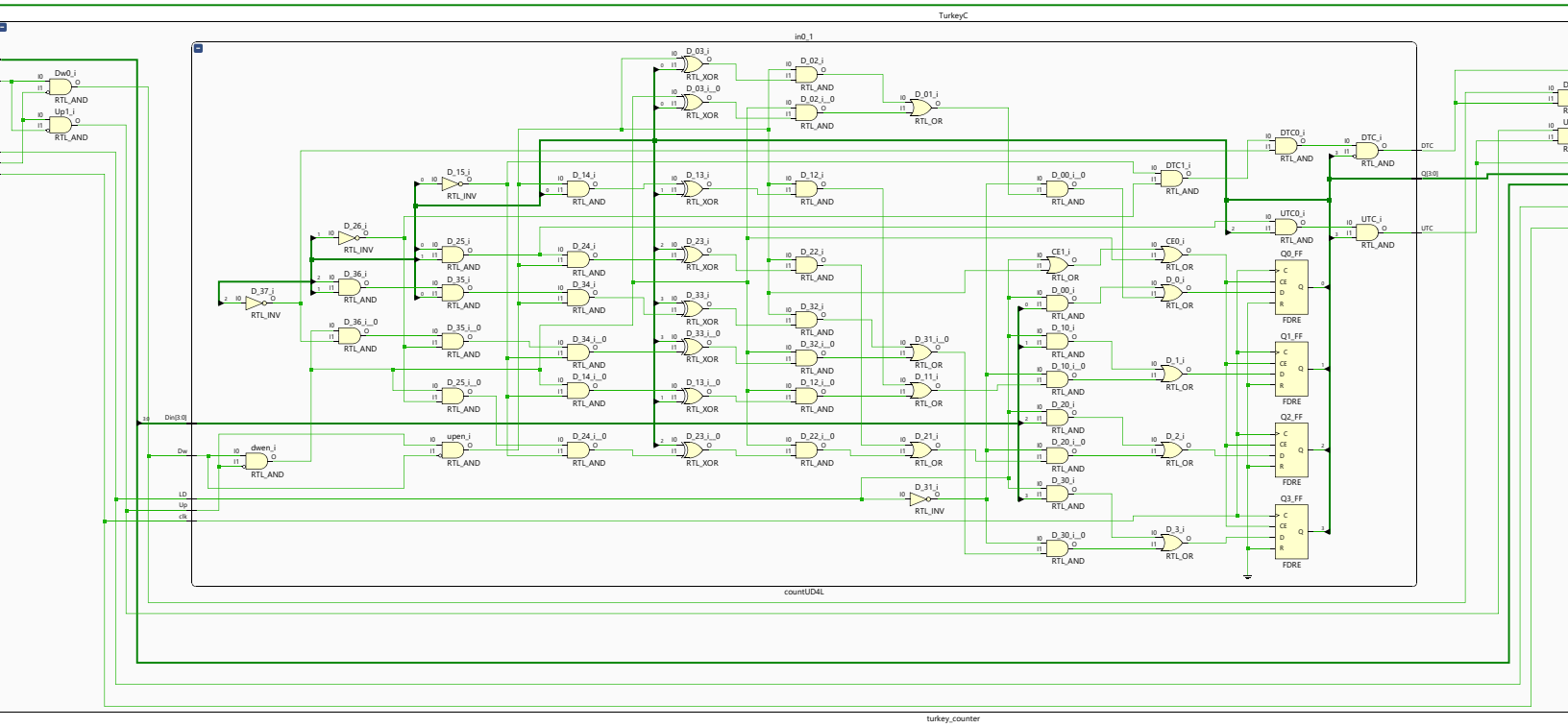

TimeC
CE
R
clk
F1_i
F0_i
F_i
CE0_i
i0
I0
I1
O
RTL_AND
CE
R
clk
Q[3:0]
Count_4b
time_counter

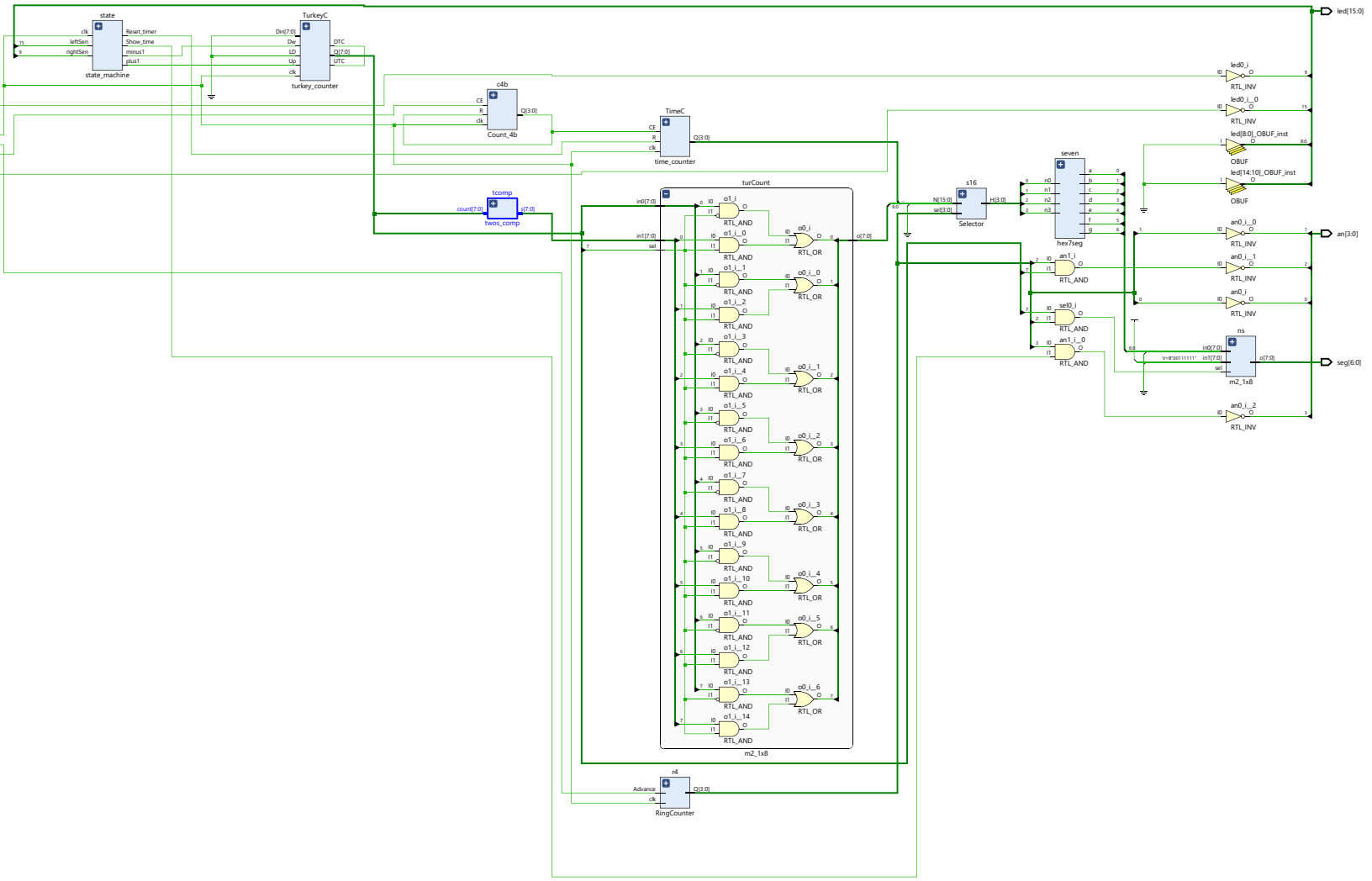

turCount
7:0]
7:0]
sel
o[7:0]
m2_1x8


r4
nce
clk
Q[3:0]
RingCounter


TurkeyC
Din[7:0]
Dw
LD
Up
clk
3:0
Dw0_i
Up1_i
RTL_AND
in0_1
Din[3:0]
Dw
LD
Up
clk
DTC
Q[3:0]
UTC
countUD4L
Dw0_i_0
Up0_i
DTC_i
UTC_i
7:4
in0_2
Q[7:0]
turkey_counter

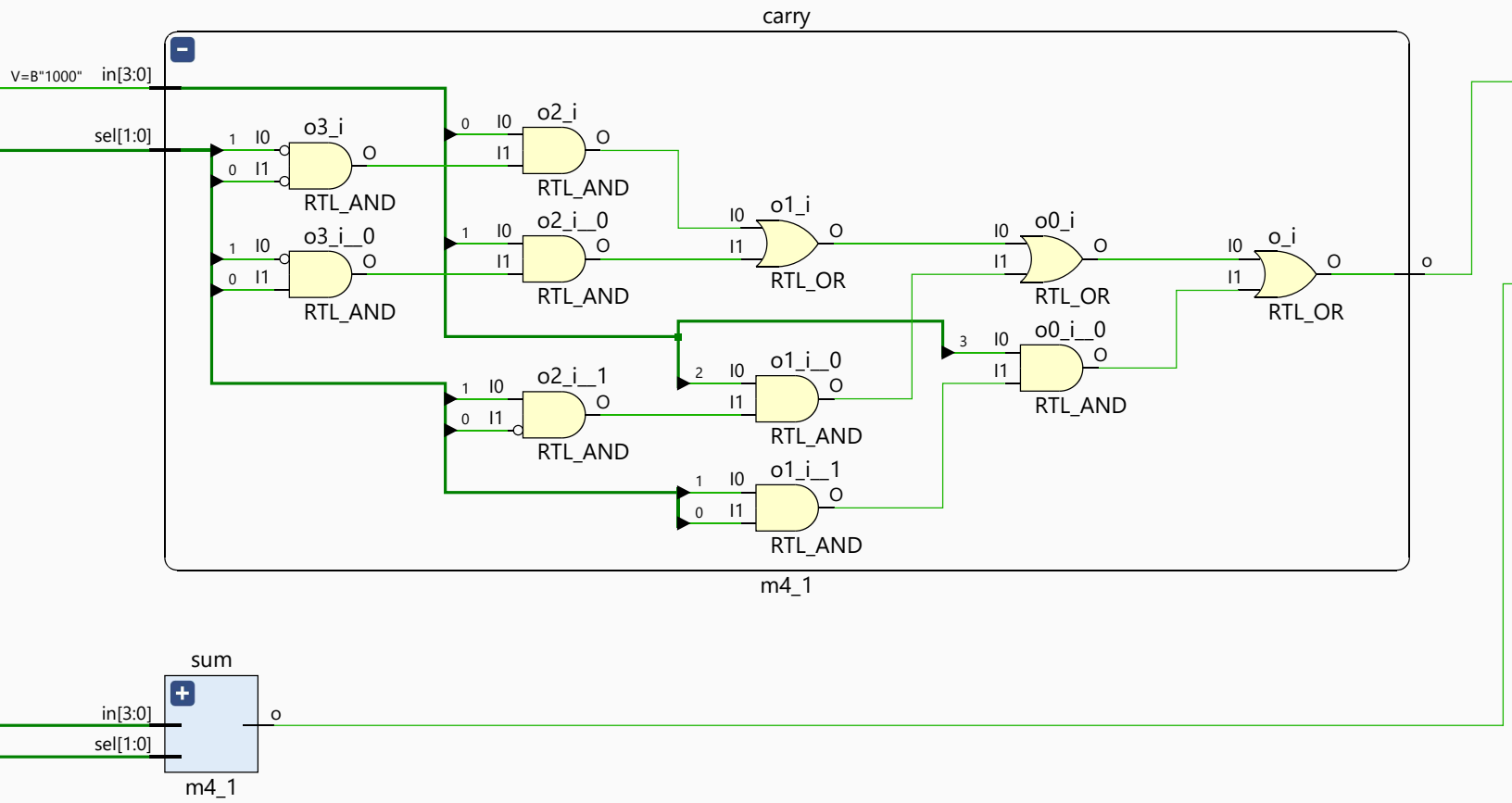

c4b
Q0_FF
D_0_i
RTL_INV
FDRE
Q[3:0]
CE
R
clk
Q1_FF
D_10_i
RTL_XOR
D_1_i
RTL_AND
FDRE
Q2_FF
D_21_i
RTL_AND
D_20_i
RTL_XOR
D_2_i
RTL_AND
FDRE
D_32_i
RTL_AND
D_31_i
RTL_AND
D_30_i
RTL_XOR
D_3_i
RTL_AND
Q3_FF
FDRE
Count_4b
TimeC
time_counter
turCount
m2_1x8
r4
RingCounter
Advance
tcomp
count[7:0]
s[7:0]
twos_comp


TurkeyC
in0_1
Dw0_i
Up1_i
D_03_i
D_03_i_0
D_02_i
D_02_i_0
D_01_i
D_15_i
D_14_i
D_13_i
D_12_i
D_00_i_0
DTC0_i
DTC_i
DTC
UTC0_i
UTC_i
UTC
Q[3:0]
D_26_i
D_25_i
D_24_i
D_23_i
D_22_i
CE1_i
CE0_i
Q0_FF
D_36_i
D_35_i
D_34_i
D_33_i
D_32_i
D_00_i
D_0_i
D_37_i
D_36_i_0
D_35_i_0
D_34_i_0
D_33_i_0
D_32_i_0
D_31_i_0
D_10_i
D_1_i
Q1_FF
D_25_i_0
D_14_i_0
D_13_i_0
D_12_i_0
D_11_i
D_10_i_0
Q2_FF
Din[3:0]
Dw
dwen_i
upen_i
D_24_i_0
D_23_i_0
D_22_i_0
D_21_i
D_20_i
D_2_i
D_20_i_0
D_31_i
D_30_i
Q3_FF
LD
Up
clk
D_30_i_0
D_3_i
RTL_AND
RTL_OR
RTL_XOR
RTL_INV
FDRE
countUD4L
turkey_counter


led[15:0]
state
clk
ns
leftSen
rightSen
Reset_timer
Show_time
minus1
plus1
state_machine
TurkeyC
Din[7:0]
Dw
LD
Up
clk
DTC
Q[7:0]
UTC
turkey_counter
c4b
CE
R
clk
Q[3:0]
Count_4b
TimeC
time_counter
tcomp
count[7:0]
twos_comp
turCount
in0[7:0]
in1[7:0]
sel
o[7:0]
RTL_AND
RTL_OR
m2_1x8
s16
N[15:0]
sel[3:0]
H[3:0]
Selector
seven
hex7seg
led0_i
led0_i_0
led[8:0]_OBUF_inst
led[14:10]_OBUF_inst
OBUF
RTL_INV
an0_i_0
an0_i_1
an0_i
an1_i
sel0_i
an1_i_0
an0_i_2
an[3:0]
ns
V="8'b01111111"
seg[6:0]
r4
Advance
RingCounter

tcomp

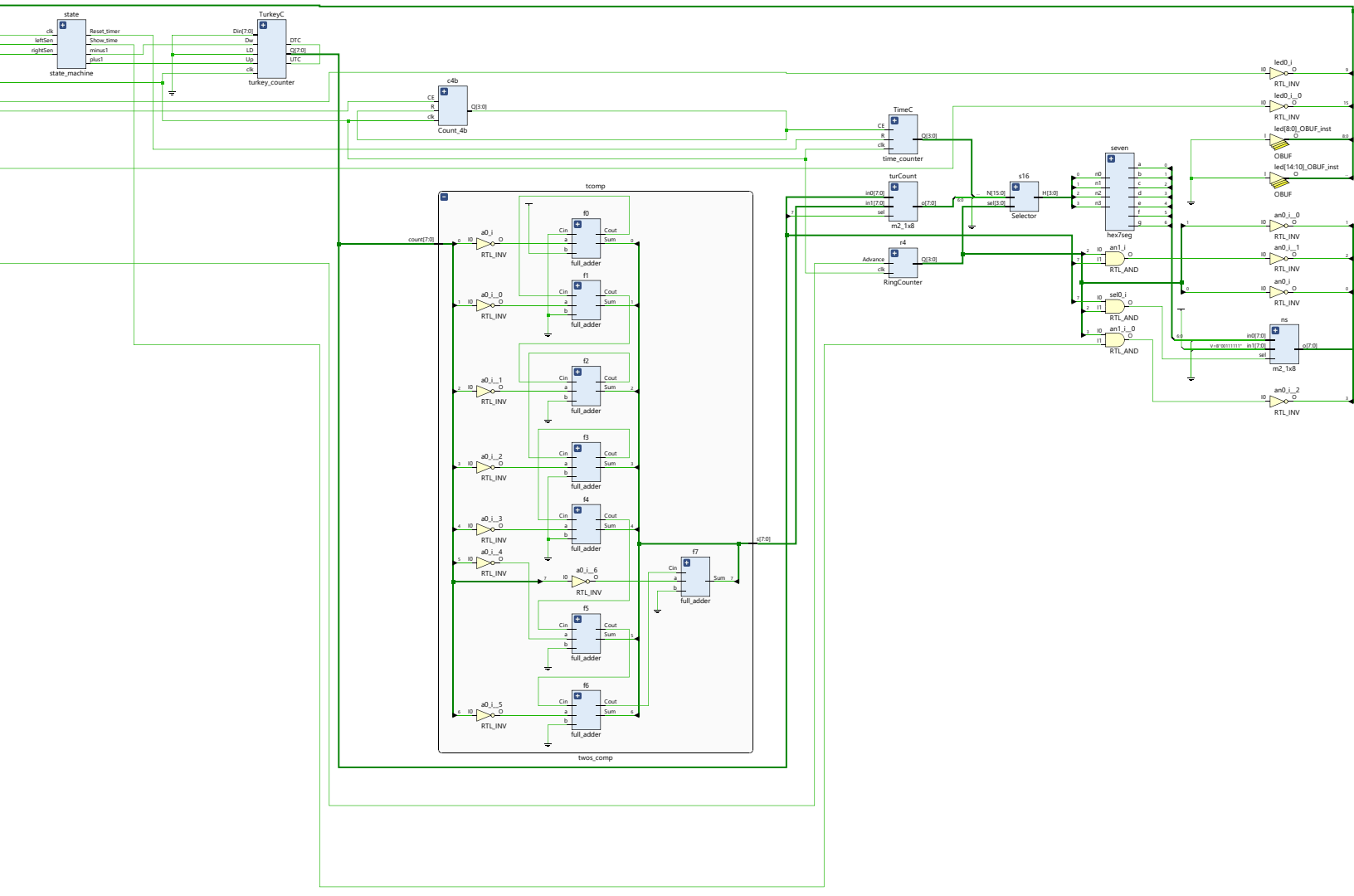
f0

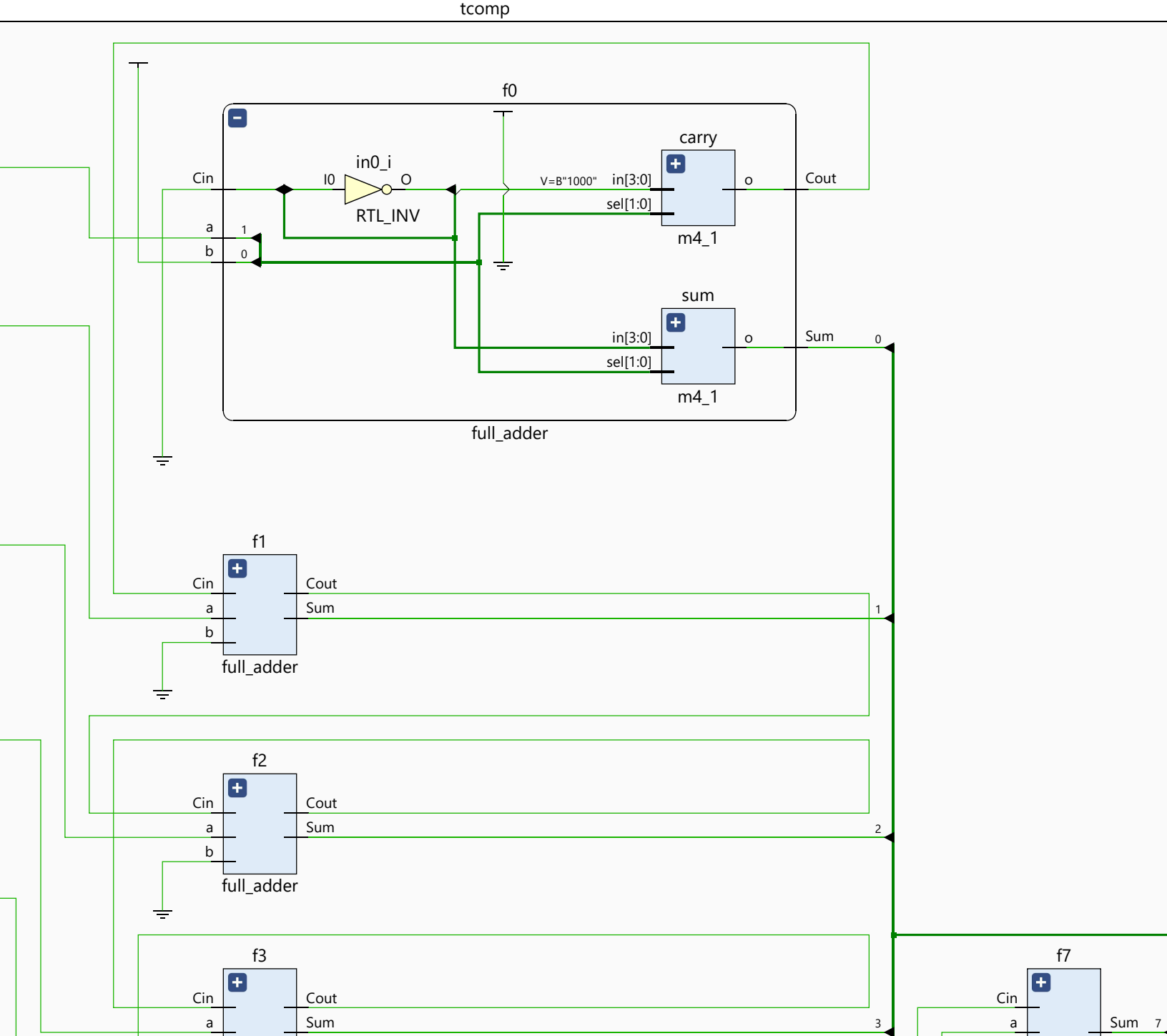


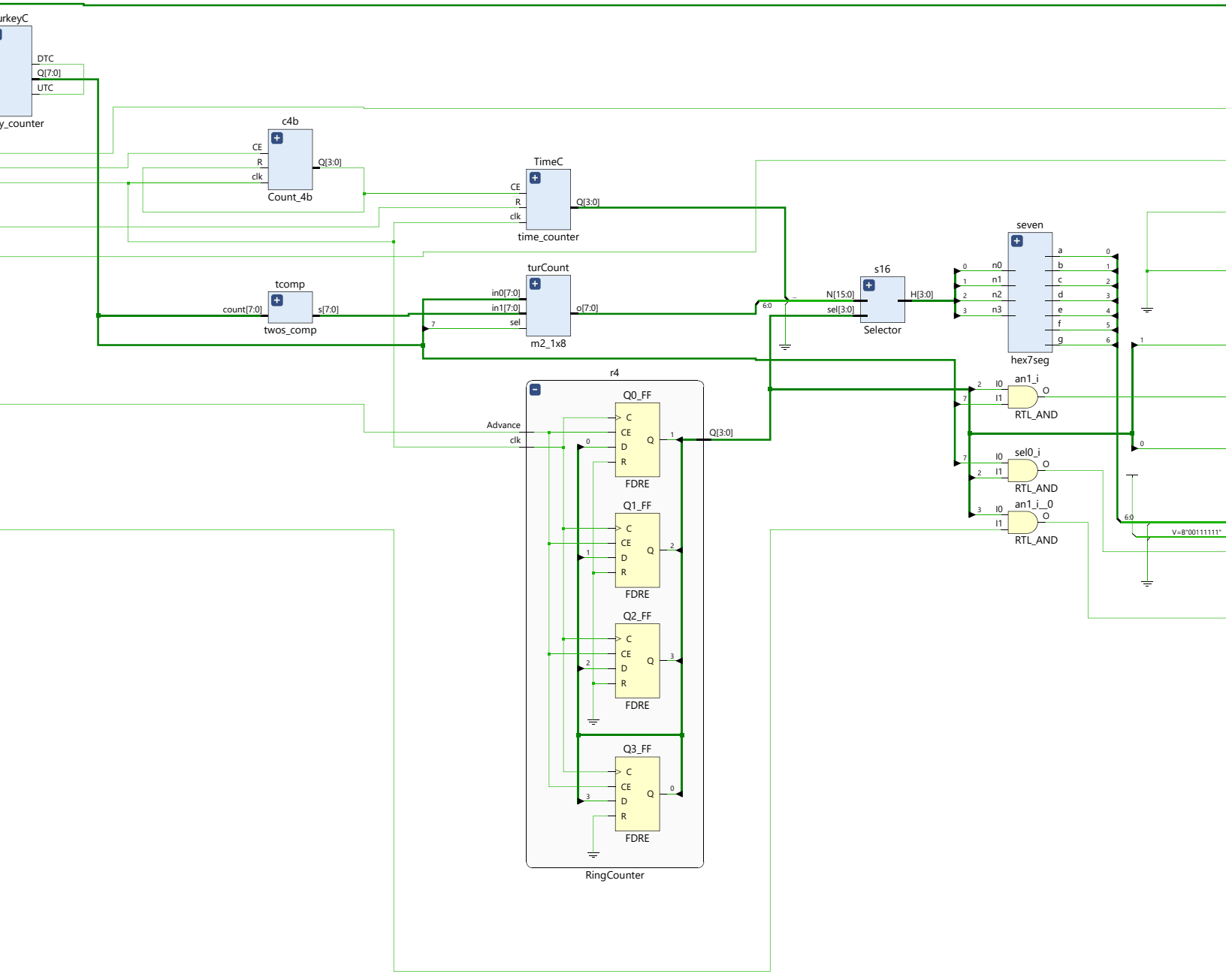


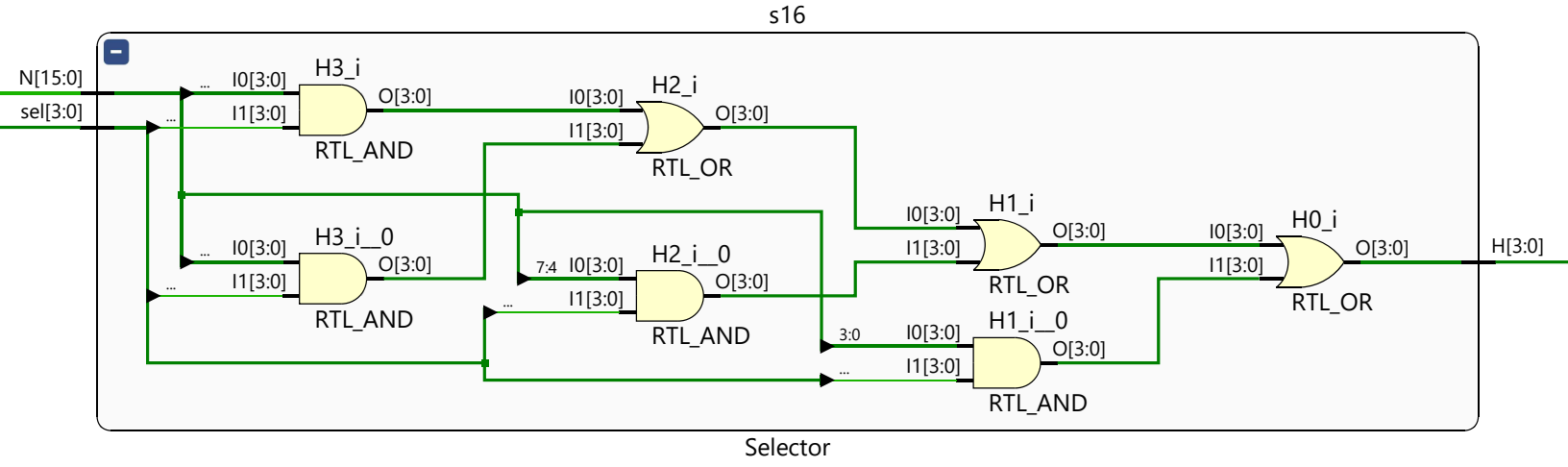


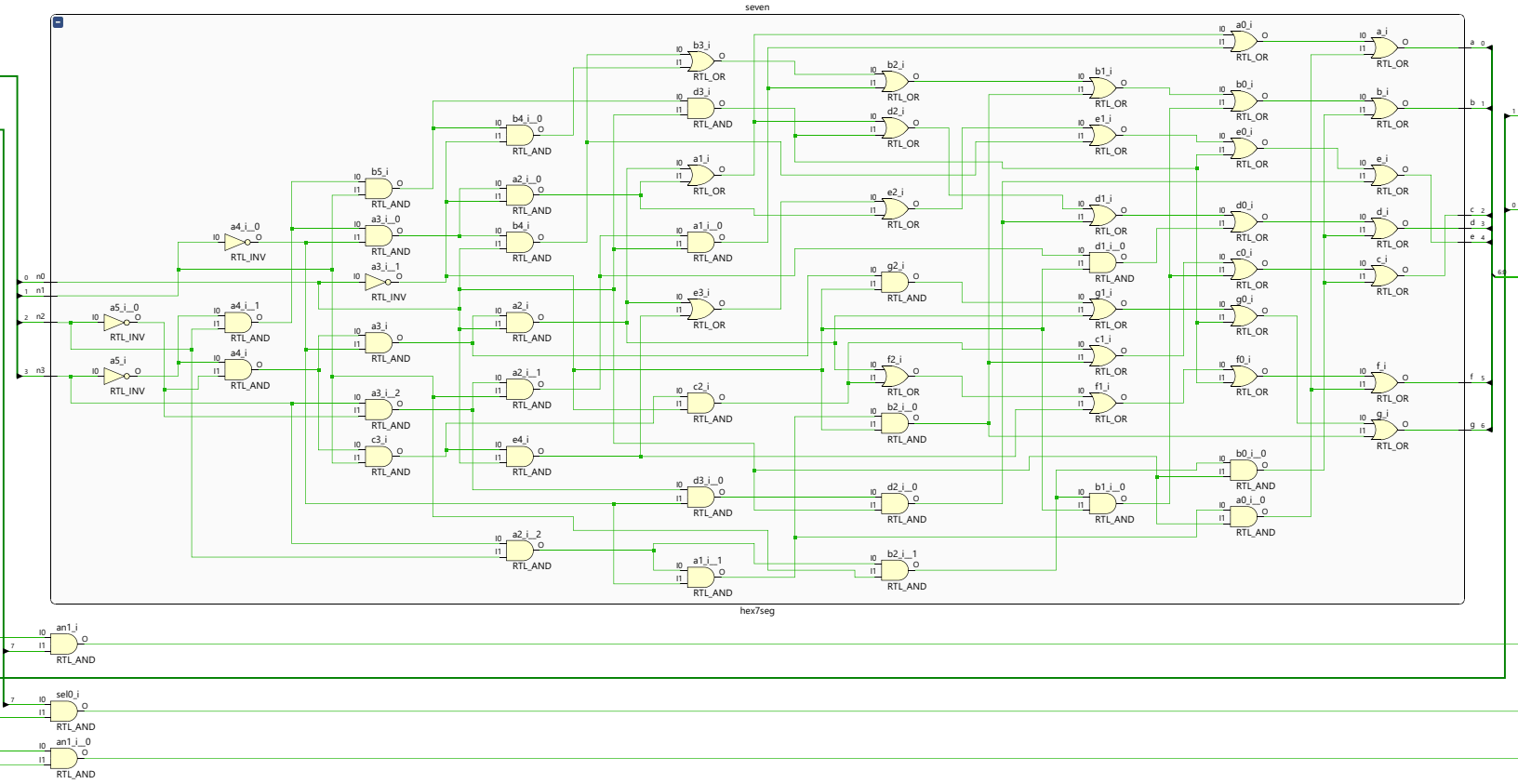
full_adder


state
clk
leftSen
rightSen
Reset_timer
Show_time
minus1
plus1
state_machine
TurkeyC
Din[7:0]
Dw
LD
Up
clk
DTC
O[7:0]
UTC
turkey_counter
c4b
CE
R
clk
O[3:0]
Count_4b
TimeC
CE
R
clk
O[3:0]
time_counter
turCount
in0[7:0]
in1[7:0]
sel
o[7:0]
m2_1x8
r4
Advance
clk
O[3:0]
RingCounter
s16
N[15:0]
sel[3:0]
H[3:0]
Selector
seven
n0
n1
n2
n3
a
b
c
d
e
f
g
hex7seg
led0_i
led0_i_0
led[8:0]_OBUF_inst
led[14:10]_OBUF_inst
OBUF
an0_i_0
an1_i
an0_i_1
an0_i
sel0_i
an1_i_0
an0_i_2
RTL_INV
RTL_AND
ns
in0[7:0]
in1[7:0]
sel
o[7:0]
m2_1x8
tcomp
count[7:0]
a0_i
a0_i_0
a0_i_1
a0_i_2
a0_i_3
a0_i_4
a0_i_6
a0_i_5
f0
f1
f2
f3
f4
f5
f6
f7
Cin
a
b
Cout
Sum
full_adder
o[7:0]
twos_comp


tcomp
f0
carry
in0_i
Cin
I0
O
V=B"1000"
in[3:0]
o
Cout
sel[1:0]
m4_1
a
1
b
0
sum
in[3:0]
o
Sum
0
sel[1:0]
m4_1
full_adder
f1
Cin
Cout
a
Sum
1
b
full_adder
f2
Cin
Cout
a
Sum
2
b
full_adder
f3
Cin
Cout
a
Sum
3
f7
Cin
a
Sum
7


rkeyC
DTC
Q[7:0]
UTC
y_counter
c4b
CE
R
clk
Q[3:0]
Count_4b
TimeC
CE
R
clk
Q[3:0]
time_counter
tcomp
count[7:0]
s[7:0]
twos_comp
turCount
in0[7:0]
in1[7:0]
sel
o[7:0]
m2_1x8
s16
N[15:0]
sel[3:0]
H[3:0]
Selector
seven
n0
n1
n2
n3
a
b
c
d
e
f
g
hex7seg
r4
Advance
clk
Q0_FF
Q1_FF
Q2_FF
Q3_FF
C
CE
D
R
Q
FDRE
Q[3:0]
RingCounter
an1_i
sel0_i
an1_i_0
I0
I1
O
RTL_AND
V=B"00111111"


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


seven
hex7seg
RTL_AND
RTL_OR
RTL_INV

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 02/16/2020 09:24:04 PM
// Design Name:
// Module Name: Top_level
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module Top_level(
    input clkin,
    input btnL, //left sensor
    input btnR, //right sensor
    input btnU, //global reset
    output [15:0] led,
    output [3:0] an,
    output [6:0] seg

    );

    wire clk, digsel, qsec; //quarter second
    wire [3:0] rings; //output of ring counter
```

```
    wire [3:0] sel; //output of selector
    wire [15:0] bits; //15 to 12 = time, 11 to 8 = neg, 7 to 0 =
turkey count
    wire showTime, showNeg, resetTimer, plus, minus, dummy;
    wire [7:0] turkeyCount; //current turkey count
    wire [7:0] turkeyInitial; //output of turkey count
    wire [7:0] turkeyComp; //twos complement of turkey count
    wire [3:0] osecTime; //used to keep track of every 1 second
interval
    wire [6:0] segm; //used to determine segment value

    lab6_clks slowit (.clkin(clkin), .greset(btnU), .clk(clk),
.digsel(digsel), .qsec(qsec));

    state_machine state (.clk(clk), .leftSen(~btnL),
.rightSen(~btnR), .Show_time(showTime), .Reset_timer(resetTimer),
.plus1(plus), .minus1(minus) );

    Count_4b c4b (.clk(clk), .CE(qsec), .R(osecTime[2]),
.Q(osecTime[3:0])); //one second counter
    time_counter TimeC (.clk(clk), .CE(osecTime[2]),
.R(resetTimer), .Q(bits[15:12]) );

    turkey_counter TurkeyC (.clk(clk), .Up(plus), .Dw(minus),
.LD(1'b0), .Din({8{1'b0}}), .Q(turkeyInitial), .UTC(dummy),
.DTC(dummy) ); //load isnt used
    twos_comp tcomp (.count(turkeyInitial[7:0]),
.s(turkeyComp[7:0])); //gets twos complement to use if negative
(2's comp of FFFF is 0001)
    m2_1x8 turCount (.in0(turkeyInitial[7:0]),
.in1(turkeyComp[7:0]), .sel(turkeyInitial[7]), .o(turkeyCount[7:0])
); //determines if negative

    assign bits[7] = 1'b0; //turkey counter is a 7 bit number so
8th bit is 0
    assign bits[6:0] = turkeyCount[6:0];
    assign showNeg = turkeyInitial[7];
```

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

    assign bits[11:8] = {4{1'b0}}; //inverse of 0 is negative (not
used, just placeholder)
    wire dummy_ns;
    m2_1x8 ns (.in0({1'b0,segm[6:0]}), .in1(8'b00111111),
.sel(turkeyInitial[7] & rings[2]), .o({dummy_ns,seg[6:0]}) ); //set
segments to negative sign if most significant bit is 1

    assign an[3] = ~(rings[3] & showTime); //time
    assign an[2] = ~(rings[2] & showNeg); //negative sign
(inverting 0 in hex7seg produces negative sign
    assign an[1] = ~rings[1]; //always on (active low)
    assign an[0] = ~rings[0]; //always on (active low)

    assign led[14:10] = {5{1'b0}}; //always off
    assign led[8:0] = {9{1'b0}}; //always off
    assign led[15] = ~btnL; //on when button L isnt pressed
    assign led[9] = ~btnR; //on when button R isnt pressed

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 02/18/2020 02:42:59 PM
// Design Name:
// Module Name: state_machine
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module state_machine(
        input clk,
        input leftSen, //starts unblocked
        input rightSen, //starts unblocked
        output Show_time, //anode 3
        output Reset_timer,
        output plus1, //right to left crossing (+1)
        output minus1 //left to right crossing (-1)
    );

    wire IDLE; //Q
    wire Left1, LeftRight, Right2; //Q
    wire Right1, RightLeft, Left2; //Q
```

```
    wire next_IDLE; //D
    wire next_Left1, next_LeftRight, next_Right2; //D
    wire next_Right1, next_RightLeft, next_Left2; //D

    assign next_IDLE = IDLE & (rightSen & leftSen) | IDLE &
(~rightSen & ~leftSen) | //accounting for both sensor @ same time)
                       Right1 & (rightSen & leftSen) | Left1 &
(rightSen & leftSen) | //first presses
                       Right2 & (rightSen & leftSen) | Left2 &
(rightSen & leftSen); //finished full crosses
    //Left Side
    assign next_Left1 = IDLE & (rightSen & ~leftSen) | Left1 &
(rightSen & ~leftSen) | LeftRight & (rightSen & ~leftSen); //enter
on left side
    assign next_LeftRight = Left1 & (~rightSen & ~leftSen) |
LeftRight & (~rightSen & ~leftSen) | Right2 & (~rightSen &
~leftSen); //moves left to right blocking both sensors
    assign next_Right2 = LeftRight & (~rightSen & leftSen) | Right2
& (~rightSen & leftSen); //moves left to right leaving left

    //Right Side
    assign next_Right1 = IDLE & (~rightSen & leftSen) | Right1 &
(~rightSen & leftSen) | RightLeft & (~rightSen & leftSen); //enter
on righ side
    assign next_RightLeft = Right1 & (~rightSen & ~leftSen) |
RightLeft & (~rightSen & ~leftSen) | Left2 & (~rightSen &
~leftSen); //moves right to left blocking both sensors
    assign next_Left2 = RightLeft & (rightSen & ~leftSen) | Left2 &
(rightSen & ~leftSen); //moves right to left leaving right

    FDRE #(.INIT(1'b1)) Q_0FF (.C(clk), .CE(1'b1), .D(next_IDLE),
.Q(IDLE) );
    FDRE #(.INIT(1'b0)) Q_1FF (.C(clk), .CE(1'b1), .D(next_Left1),
.Q(Left1) );
    FDRE #(.INIT(1'b0)) Q_2FF (.C(clk), .CE(1'b1),
.D(next_LeftRight), .Q(LeftRight) );
```

```
    FDRE #(.INIT(1'b0)) Q_3FF (.C(clk), .CE(1'b1), .D(next_Right2),
.Q(Right2) );
    FDRE #(.INIT(1'b0)) Q_5FF (.C(clk), .CE(1'b1), .D(next_Right1),
.Q(Right1) );
    FDRE #(.INIT(1'b0)) Q_6FF (.C(clk), .CE(1'b1),
.D(next_RightLeft), .Q(RightLeft) );
    FDRE #(.INIT(1'b0)) Q_7FF (.C(clk), .CE(1'b1), .D(next_Left2),
.Q(Left2) );

    assign Show_time = ~IDLE; //anode 3
    assign Reset_timer = (IDLE & ~rightSen & leftSen) | (IDLE &
rightSen & ~leftSen);
    assign plus1 = Left2 & rightSen & leftSen; //right to left
crossing (+1)
    assign minus1 = Right2 & rightSen & leftSen; //left to right
crossing (-1)

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 02/18/2020 02:50:36 PM
// Design Name:
// Module Name: time_counter
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


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

    wire F; //full time countr (15 seconds)

    assign F = Q[3] & Q[2] & Q[1] & Q[0] & 1'b1; //1111 = 15 = F

    Count_4b i0 (.clk(clk), .CE(CE & ~F), .R(R), .Q(Q[3:0]));
//stops counting past F
```

```
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 02/19/2020 09:57:34 PM
// Design Name:
// Module Name: turkey_counter
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module turkey_counter(
        input clk,
        input Up, //count enable (increment)
        input Dw, //not count enable (decrement)
        input LD,
        input [7:0] Din, //8 bit counter
        output [7:0] Q,
        output UTC,
        output DTC
    );

    wire utc[1:0];
    wire dtc[1:0];
```

```
    countUD4L in0_1 (.clk(clk), .Up(Up & ~Dw), .Dw(Dw & ~Up),
.LD(LD), .Din(Din[3:0]), .Q(Q[3:0]), .UTC(utc[0]), .DTC(dtc[0]));
    countUD4L in0_2 (.clk(clk), .Up(Up & ~Dw & utc[0]), .Dw(Dw &
~Up & dtc[0]), .LD(LD), .Din(Din[7:4]), .Q(Q[7:4]), .UTC(utc[1]),
.DTC(dtc[1]));

    assign UTC = utc[1] & utc[0];
    assign DTC = dtc[1] & dtc[0];

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 02/20/2020 02:25:04 AM
// Design Name:
// Module Name: countUD4L
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


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

    wire [3:0] D;
    wire upen, dwen;
```

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

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

    assign UTC = (Q[0] & Q[1] & Q[2] & Q[3]);
    assign DTC = ~Q[0] & ~Q[1] & ~Q[2] & ~Q[3];

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 02/18/2020 02:53:38 PM
// Design Name:
// Module Name: Count_4b
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


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

    wire [3:0] D;

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

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

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 02/20/2020 04:04:09 AM
// Design Name:
// Module Name: m2_1x8
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module m2_1x8 (
        input [7:0] in0,
        input [7:0] in1,
        input sel,
        output [7:0] o
    );

    assign o[0] = (in0[0] & ~sel) | (in1[0] & sel);
    assign o[1] = (in0[1] & ~sel) | (in1[1] & sel);
    assign o[2] = (in0[2] & ~sel) | (in1[2] & sel);
    assign o[3] = (in0[3] & ~sel) | (in1[3] & sel);
    assign o[4] = (in0[4] & ~sel) | (in1[4] & sel);
    assign o[5] = (in0[5] & ~sel) | (in1[5] & sel);
```

```
    assign o[6] = (in0[6] & ~sel) | (in1[6] & sel);
    assign o[7] = (in0[7] & ~sel) | (in1[7] & sel);

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 02/19/2020 10:18:29 PM
// Design Name:
// Module Name: m4_1
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module m4_1 (
    input [3:0] in,
    input [1:0] sel,
    output o
    );
assign o = (in[0] & (~sel[1] & ~sel[0])) | //when these two are 0,
whole thing is 1 (thats our condition)
    (in[1] & (~sel[1] & sel[0])) |
    (in[2] & (sel[1] & ~sel[0])) |
    (in[3] & (sel[1] & sel[0]));
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 02/20/2020 02:28:22 AM
// Design Name:
// Module Name: twos_comp
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module twos_comp(
        input [7:0] count, //current counter value
        output [7:0] s //sum (8th bit determines pos/neg)
    );
    wire [7:0] w;
    wire dummy;

    //get 2's complement by inverting and adding 1
    full_adder f0 (.a(~count[0]), .b(1'b1), .Cin(1'b0),
.Cout(w[0]), .Sum(s[0]));
    full_adder f1 (.a(~count[1]), .b(1'b0), .Cin(w[0]),
.Cout(w[1]), .Sum(s[1]));
    full_adder f2 (.a(~count[2]), .b(1'b0), .Cin(w[1]),
```

```
.Cout(w[2]), .Sum(s[2]));
    full_adder f3 (.a(~count[3]), .b(1'b0), .Cin(w[2]),
.Cout(w[3]), .Sum(s[3]));
    full_adder f4 (.a(~count[4]), .b(1'b0), .Cin(w[3]),
.Cout(w[4]), .Sum(s[4]));
    full_adder f5 (.a(~count[5]), .b(1'b0), .Cin(w[4]),
.Cout(w[5]), .Sum(s[5]));
    full_adder f6 (.a(~count[6]), .b(1'b0), .Cin(w[5]),
.Cout(w[6]), .Sum(s[6]));
    full_adder f7 (.a(~count[7]), .b(1'b0), .Cin(w[6]),
.Cout(dummy), .Sum(s[7]));
    //ignore last cout because can't support 9 bit output

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 02/19/2020 10:16:20 PM
// Design Name:
// Module Name: full_adder
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module full_adder (
    input a,
    input b,
    input Cin,
    output Cout,
    output Sum
    );
    m4_1 sum (.in({Cin, ~Cin, ~Cin, Cin}), .sel({a, b}), .o(Sum));
    m4_1 carry (.in({1'b1, Cin, Cin, 1'b0}), .sel({a, b}),
.o(Cout));
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 02/19/2020 11:11:14 PM
// Design Name:
// Module Name: RingCounter
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


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


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

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

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 02/19/2020 11:10:23 PM
// Design Name:
// Module Name: Selector
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


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

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

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 02/19/2020 11:09:32 PM
// Design Name:
// Module Name: hex7seg
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


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

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

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 02/20/2020 11:52:43 AM
// Design Name:
// Module Name: top_sim
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module top_sim();
        reg clkin;
        reg btnL; //left sensor
        reg btnR; //right sensor
        reg btnU; //global reset
        wire [15:0] led;
        wire [3:0] an;
        wire [6:0] seg;

Top_level UUT(
        .clkin(clkin),
        .btnL(btnL),
        .btnR(btnR),
```

```
        .btnU(btnU),
        .led(led),
        .an(an),
        .seg(seg)
);

    parameter PERIOD = 10;
    parameter real DUTY_CYCLE = 0.5;
    parameter OFFSET = 2;

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

    initial
    begin
        //current time is 0ns
        //-------------------- both sensors blocked (IDLE)
        btnR =1'b0; //unblocked
        btnL = 1'b0; //unblocked
        btnU = 1'b0;

        #1000; //current time is 1000ns
        //-------------------- both sensors blocked (IDLE)
        btnR =1'b0; //unblocked
        btnL = 1'b0; //unblocked
        btnU = 1'b0;

        #100; //current time is 1100ns
        //-------------------- both sensors unblocked (IDLE)
```

```
        btnR =1'b1; //blocked
        btnL = 1'b1; //blocked

        //============================================== LEFT ENTER
RETURN LEFTS
============================================================

        #100; //current time is 1200ns
        //-------------------- enters left (Left1) 0
        btnR =1'b0; //unblocked
        btnL = 1'b1; //blocked

        #100; //current time is 1300ns
        //-------------------- leaves left (IDLE) 0
        btnR =1'b0; //unblocked
        btnL = 1'b0; //unblocked

        #100; //current time is 1400ns
        //-------------------- enters left (Left1) 0
        btnR =1'b0; //unblocked
        btnL = 1'b1; //blocked

        #100; //current time is 1500ns
        //-------------------- in middle (LeftRight) 0
        btnR =1'b1; //blocked
        btnL = 1'b1; //blocked

        #100; //current time is 1600ns
        //-------------------- leaves right (Left1) 0
        btnR =1'b0; //unblocked
        btnL = 1'b1; //blocked

        #100; //current time is 1700ns
        //-------------------- leaves left(IDLE) 0
        btnR =1'b0; //unblocked
        btnL = 1'b0; //unblocked
```

```
        #100; //current time is 1800ns
        //-------------------- enters left (Left1)0
        btnR =1'b0; //unblocked
        btnL = 1'b1; //blocked

        #100; //current time is 1900ns
        //-------------------- in middle (LeftRight) 0
        btnR =1'b1; //blocked
        btnL = 1'b1; //blocked

        #100; //current time is 2000ns
        //-------------------- leave left (Right2) 0
        btnR =1'b1; //blocked
        btnL = 1'b0; //unblocked

        #100; //current time is 2100ns
        //-------------------- in middle (LeftRight) 0
        btnR =1'b1; //blocked
        btnL = 1'b1; //blocked

        #100; //current time is 2200ns
        //-------------------- leave right (Left1) 0
        btnR =1'b0; //unblocked
        btnL = 1'b1; //blocked

        #100; //current time is 2300ns
        //-------------------- leave left (IDLE) 0
        btnR = 1'b0; //unblocked
        btnL = 1'b0; //unblocked

        //============================================== RIGHT ENTER
RETURN RIGHTS
============================================================
        #100; //current time is 2400ns
        //-------------------- enters right (Right1) 0
        btnR =1'b1; //blocked
        btnL = 1'b0; //unblocked
```

```
#100; //current time is 2500ns
//-------------------- leaves right (IDLE) 0
btnR =1'b0; //unblocked
btnL = 1'b0; //unblocked

#100; //current time is 2600ns
//-------------------- enters right (Right1) 0
btnR =1'b1; //blocked
btnL = 1'b0; //unblocked

#100; //current time is 2700ns
//-------------------- in middle (RightLeft) 0
btnR =1'b1; //blocked
btnL = 1'b1; //blocked

#100; //current time is 2800ns
//-------------------- leaves left (Right1) 0
btnR =1'b1; //blocked
btnL = 1'b0; //unblocked

#100; //current time is 2900ns
//-------------------- leaves right(IDLE) 0
btnR =1'b0; //unblocked
btnL = 1'b0; //unblocked

#100; //current time is 3000ns
//-------------------- enters right (Right1)0
btnR =1'b1; //blocked
btnL = 1'b0; //unblocked

#100; //current time is 3100ns
//-------------------- in middle (RightLeft) 0
btnR =1'b1; //blocked
btnL = 1'b1; //blocked

#100; //current time is 3200ns
```

```
        //-------------------- leave right (Left2) 0
        btnR =1'b0; //unblocked
        btnL = 1'b1; //blocked

        #100; //current time is 3300ns
        //-------------------- in middle (RightLeft) 0
        btnR =1'b1; //blocked
        btnL = 1'b1; //blocked

        #100; //current time is 3400ns
        //-------------------- leave left (Right1) 0
        btnR =1'b1; //blocked
        btnL = 1'b0; //unblocked

        #100; //current time is 3500ns
        //-------------------- leave right (IDLE) 0
        btnR =1'b0; //unblocked
        btnL = 1'b0; //unblocked

        //============================================== RIGHT TO
LEFT CROSSES
============================================================

        #100; //current time is 3600ns
        //-------------------- enter right (Right1) 0
        btnR = 1'b1; //blocked
        btnL = 1'b0; //unblocked

        #100; //current time is 3700ns
        //-------------------- in middle (RightLeft) 0
        btnR =1'b1; //blocked
        btnL = 1'b1; //blocked

        #100; //current time is 3800ns
        //-------------------- leave right (Left2) 0
        btnR = 1'b0; //unblocked
        btnL = 1'b1; //blocked
```

```
        #100; //current time is 3900ns
        //-------------------- leave left (IDLE) 1
        btnR = 1'b0; //unblocked
        btnL = 1'b0; //unblocked

        #100; //current time is 4000ns
        //-------------------- enter right (Right1) 1
        btnR = 1'b1; //blocked
        btnL = 1'b0; //unblocked

        #100; //current time is 4100ns
        //-------------------- in middle (RightLeft) 1
        btnR =1'b1; //blocked
        btnL = 1'b1; //blocked

        #100; //current time is 4200ns
        //-------------------- leave right (Left2) 1
        btnR = 1'b0; //unblocked
        btnL = 1'b1; //blocked

        #100; //current time is 4300ns
        //-------------------- leave left (IDLE) 2
        btnR = 1'b0; //unblocked
        btnL = 1'b0; //unblocked

        //============================================== LEFT TO
RIGHT CROSSES
============================================================

        #100; //current time is 4400ns
        //-------------------- enter left (Left1) 2
        btnR = 1'b0; //unblocked
        btnL = 1'b1; //blocked

        #100; //current time is 4500ns
        //-------------------- in middle (LeftRight) 2
```

```
btnR =1'b1; //blocked
btnL = 1'b1; //blocked

#100; //current time is 4600ns
//-------------------- leave left (Right2) 2
btnR = 1'b1; //blocked
btnL = 1'b0; //unblocked

#100; //current time is 4700ns
//-------------------- leave left (IDLE) 1
btnR = 1'b0; //unblocked
btnL = 1'b0; //unblocked

#100; //current time is 4800ns
//-------------------- enter left (Left1) 1
btnR = 1'b0; //unblocked
btnL = 1'b1; //blocked

#100; //current time is 4900ns
//-------------------- in middle (LeftRight) 1
btnR =1'b1; //blocked
btnL = 1'b1; //blocked

#100; //current time is 5000ns
//-------------------- leave left (Right2) 1
btnR = 1'b1; //blocked
btnL = 1'b0; //unblocked

#100; //current time is 5100ns
//-------------------- leave left (IDLE) 0
btnR = 1'b0; //unblocked
btnL = 1'b0; //unblocked

#100; //current time is 5200ns
//-------------------- enter left (Left1) 0
btnR = 1'b0; //unblocked
btnL = 1'b1; //blocked
```

```
#100; //current time is 5300ns
//-------------------- in middle (LeftRight) 0
btnR =1'b1; //blocked
btnL = 1'b1; //blocked

#100; //current time is 5400ns
//-------------------- leave left (Right2) 0
btnR = 1'b1; //blocked
btnL = 1'b0; //unblocked

#100; //current time is 5500ns
//-------------------- leave left (IDLE) -1
btnR = 1'b0; //unblocked
btnL = 1'b0; //unblocked

#100; //current time is 5600ns
//-------------------- enter left (Left1) -1
btnR = 1'b0; //unblocked
btnL = 1'b1; //blocked

#100; //current time is 5700ns
//-------------------- in middle (LeftRight) -1
btnR =1'b1; //blocked
btnL = 1'b1; //blocked

#100; //current time is 5800ns
//-------------------- leave left (Right2) -1
btnR = 1'b1; //blocked
btnL = 1'b0; //unblocked

#100; //current time is 5900ns
//-------------------- leave left (IDLE) -2
btnR = 1'b0; //unblocked
btnL = 1'b0; //unblocked

#100; //current time is 6000ns
```

```
    end

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 02/19/2020 07:37:34 PM
// Design Name:
// Module Name: state_sim
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module state_sim();
        reg clk;
        reg leftSen; //starts unblocked
        reg rightSen; //starts unblocked
        wire Show_time; //anode 3
        wire Reset_timer;
        wire plus1; //right to left crossing (+1)
        wire minus1; //left to right crossing (-1)

state_machine UUT(
        .clk(clk),
        .leftSen(leftSen),
        .rightSen(rightSen),
```

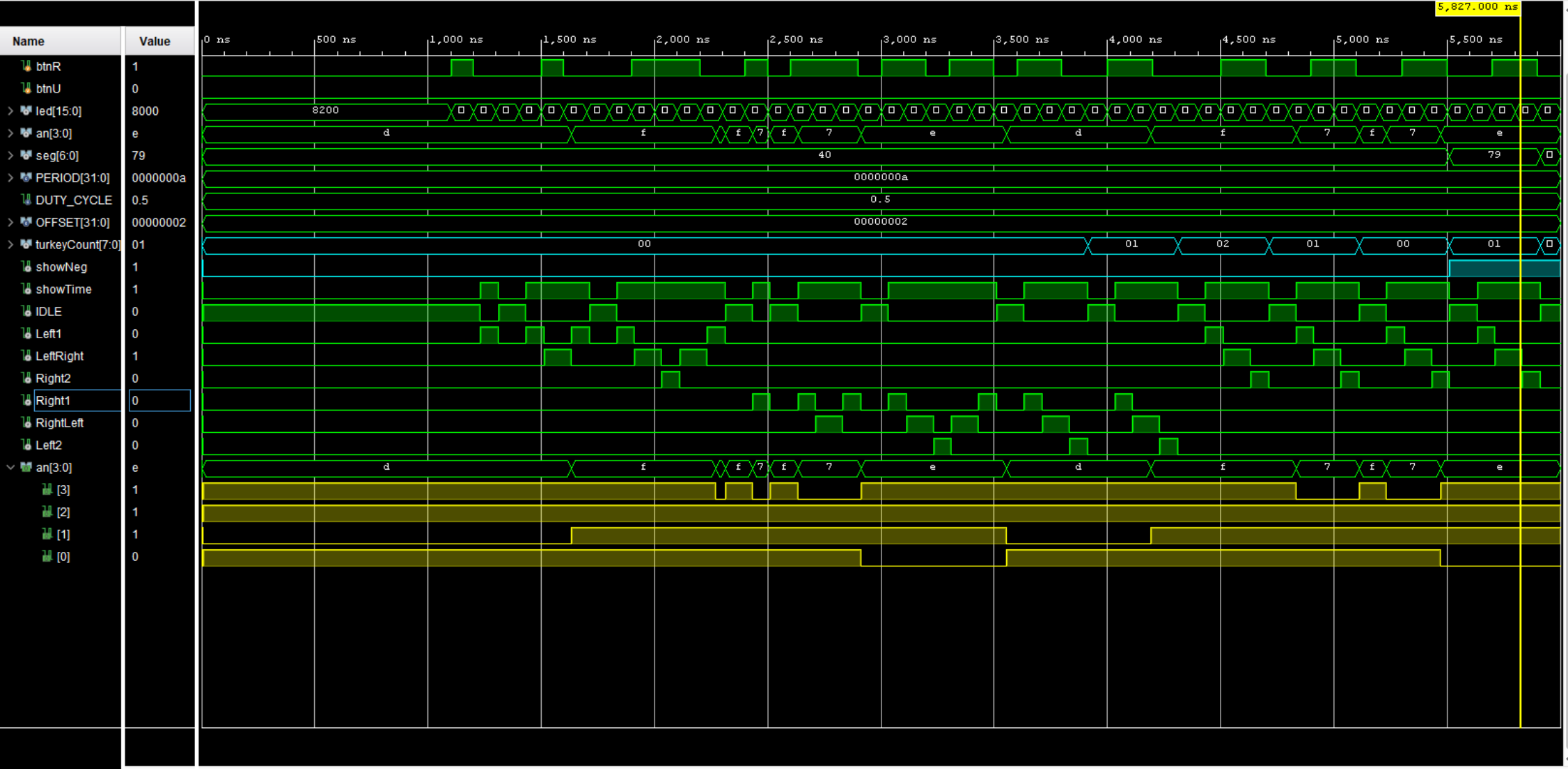
```
        .Show_time(Show_time),
        .Reset_timer(Reset_timer),
        .plus1(plus1),
        .minus1(minus1)
);

    parameter PERIOD = 10;
    parameter real DUTY_CYCLE = 0.5;
    parameter OFFSET = 2;

    initial    // Clock process for clkin
    begin
        #OFFSET
          clk = 1'b1;
        forever
        begin
            #(PERIOD-(PERIOD*DUTY_CYCLE)) clk = ~clk;
        end
    end

    initial
    begin
        //current time is 0ns
        //-------------------- both sensors blocked (IDLE)
        rightSen = 1'b0; //blocked
        leftSen = 1'b0; //blocked

        #1000; //current time is 1000ns
        //-------------------- both sensors blocked (IDLE)
        rightSen = 1'b0; //blocked
        leftSen = 1'b0; //blocked

        #100; //current time is 1100ns
        //-------------------- both sensors unblocked (IDLE)
        rightSen = 1'b1; //unblocked
        leftSen = 1'b1; //unblocked
```

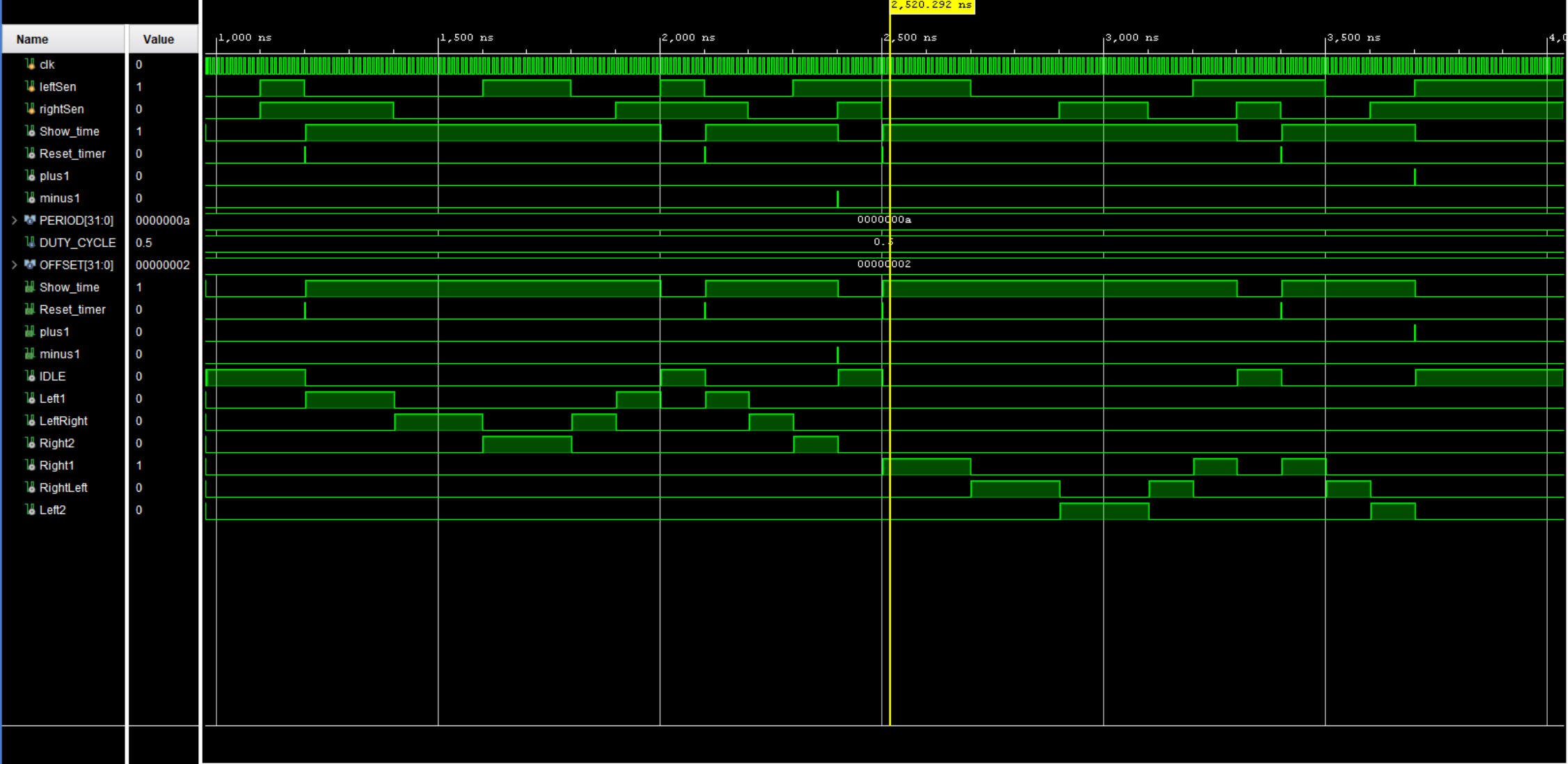
```
        //=======================================LEFT
ENTER=========================================

        #100; //current time is 1200ns
        //--------------------- left sensor blocked (Left1)
        rightSen = 1'b1; //unblocked
        leftSen = 1'b0; //blocked

        #100; //current time is 1300ns
        //--------------------- left sensor blocked (Left1)
        rightSen = 1'b1; //unblocked
        leftSen = 1'b0; //blocked

        #100; //current time is 1400ns
        //--------------------- both sensors blocked (LeftRight)
        rightSen = 1'b0; //blocked
        leftSen = 1'b0; //blocked

        #100; //current time is 1500ns
        //--------------------- both sensors blocked (LeftRight)
        rightSen = 1'b0; //blocked
        leftSen = 1'b0; //blocked

        #100; //current time is 1600ns
        //--------------------- left sensor unblocked (Right2)
        rightSen = 1'b0; //blocked
        leftSen = 1'b1; //unblocked

        #100; //current time is 1700ns
        //--------------------- left sensor unblocked (Right2)
        rightSen = 1'b0; //blocked
        leftSen = 1'b1; //unblocked

        #100; //current time is 1800ns
        //--------------------- both sensors blocked (LeftRight)
        rightSen = 1'b0; //blocked
        leftSen = 1'b0; //blocked
```

```
        #100; //current time is 1900ns
        //-------------------- right sensor unblocked (Left1)
        rightSen = 1'b1; //unblocked
        leftSen = 1'b0; //blocked

        #100; //current time is 2000ns
        //-------------------- both sensors unblocked (IDLE)
        rightSen = 1'b1; //unblocked
        leftSen = 1'b1; //unblocked

        #100; //current time is 2100ns
        //-------------------- left sensor blocked (Left1)
        rightSen = 1'b1; //unblocked
        leftSen = 1'b0; //blocked

        #100; //current time is 2200ns
        //-------------------- both sensors blocked (LeftRight)
        rightSen = 1'b0; //blocked
        leftSen = 1'b0; //blocked

        #100; //current time is 2300ns
        //-------------------- left sensor unblocked (Right2)
        rightSen = 1'b0; //blocked
        leftSen = 1'b1; //unblocked

        #100; //current time is 2400ns
        //-------------------- both sensors unblocked (IDLE)
        rightSen = 1'b1; //unblocked
        leftSen = 1'b1; //unblocked

        //==================================RIGHT
ENTER====================================

        #100; //current time is 2500ns
        //-------------------- left sensor blocked (Right1)
        rightSen = 1'b0; //blocked
```

```
leftSen = 1'b1; //unblocked

#100; //current time is 2600ns
//-------------------- left sensor blocked (Right1)
rightSen = 1'b0; //blocked
leftSen = 1'b1; //unblocked

#100; //current time is 2700ns
//-------------------- both sensors blocked (RightLeft)
rightSen = 1'b0; //blocked
leftSen = 1'b0; //blocked

#100; //current time is 2800ns
//-------------------- both sensors blocked (RightLeft)
rightSen = 1'b0; //blocked
leftSen = 1'b0; //blocked

#100; //current time is 2900ns
//-------------------- left sensor unblocked (Left2)
rightSen = 1'b1; //unblocked
leftSen = 1'b0; //blocked

#100; //current time is 3000ns
//-------------------- left sensor unblocked (Left2)
rightSen = 1'b1; //unblocked
leftSen = 1'b0; //blocked

#100; //current time is 3100ns
//-------------------- both sensors blocked (RightLeft)
rightSen = 1'b0; //blocked
leftSen = 1'b0; //blocked

#100; //current time is 3200ns
//-------------------- right sensor unblocked (Right1)
rightSen = 1'b0; //blocked
leftSen = 1'b1; //unblocked
```

```
        #100; //current time is 33000ns
        //-------------------- both sensors unblocked (IDLE)
        rightSen = 1'b1; //unblocked
        leftSen = 1'b1; //unblocked

        #100; //current time is 3400ns
        //-------------------- left sensor blocked (Right1)
        rightSen = 1'b0; //blocked
        leftSen = 1'b1; //unblocked

        #100; //current time is 3500ns
        //-------------------- both sensors blocked (RightLeft)
        rightSen = 1'b0; //blocked
        leftSen = 1'b0; //blocked

        #100; //current time is 3600ns
        //-------------------- left sensor unblocked (Left2)
        rightSen = 1'b1; //unblocked
        leftSen = 1'b0; //blocked

        #100; //current time is 3700ns
        //-------------------- both sensors unblocked (IDLE)
        rightSen = 1'b1; //unblocked
        leftSen = 1'b1; //unblocked

        #100; //current time is 3800ns

    end

endmodule
```


Name
Value
btnR
1
btnU
0
led[15:0]
8200
an[3:0]
e
seg[6:0]
79
PERIOD[31:0]
0000000a
DUTY_CYCLE
0.5
OFFSET[31:0]
00000002
turkeyCount[7:0]
01
showNeg
1
showTime
1
IDLE
0
Left1
0
LeftRight
1
Right2
0
Right1
0
RightLeft
0
Left2
0
an[3:0]
e
[3]
1
[2]
1
[1]
1
[0]
0
0 ns
500 ns
1,000 ns
1,500 ns
2,000 ns
2,500 ns
3,000 ns
3,500 ns
4,000 ns
4,500 ns
5,000 ns
5,500 ns
5,827.000 ns
40
79
0000000a
0.5
00000002
00
01
02
01
00
01


Name
Value
clk 0
leftSen 1
rightSen 0
Show_time 1
Reset_timer 0
plus1 0
minus1 0
PERIOD[31:0] 0000000a
DUTY_CYCLE 0.5
OFFSET[31:0] 00000002
Show_time 1
Reset_timer 0
plus1 0
minus1 0
IDLE 0
Left1 0
LeftRight 0
Right2 0
Right1 1
RightLeft 0
Left2 0
2,520.292 ns
1,000 ns
1,500 ns
2,000 ns
2,500 ns
3,000 ns
3,500 ns
0000000a
0.5
00000002

2/16/2020
(Sun)
9:00pm

# Lab 6

## What it does

- while both sensors are not unblocked, AN3 displays and counts seconds up to 15
  - stays at 15 if past 15 seconds
  - blank (not 0) if both sensors unblocked
- LED 15 on if left sensor unblocked
- LED 9 on if right sensor unblocked
- AN1 and AN0 display number of right to left crossings minus left to right crossings
  - AN1 is 0 (not blank) if difference is a 1 digit number
- AN2 displays a dash (-), negative sign, if difference is negative, blank otherwise
- AN1 and AN0 display 0 and AN2 and AN3 are blank on IDLE
- btn U resets difference
- btn L blocks left sensor, btn R blocks right sensor
- only detects full pass throughs (both sensors blocked at least once)
- probably mealy machine
- AN1 and AN0 go to at most 7F (127)

## Board stuff

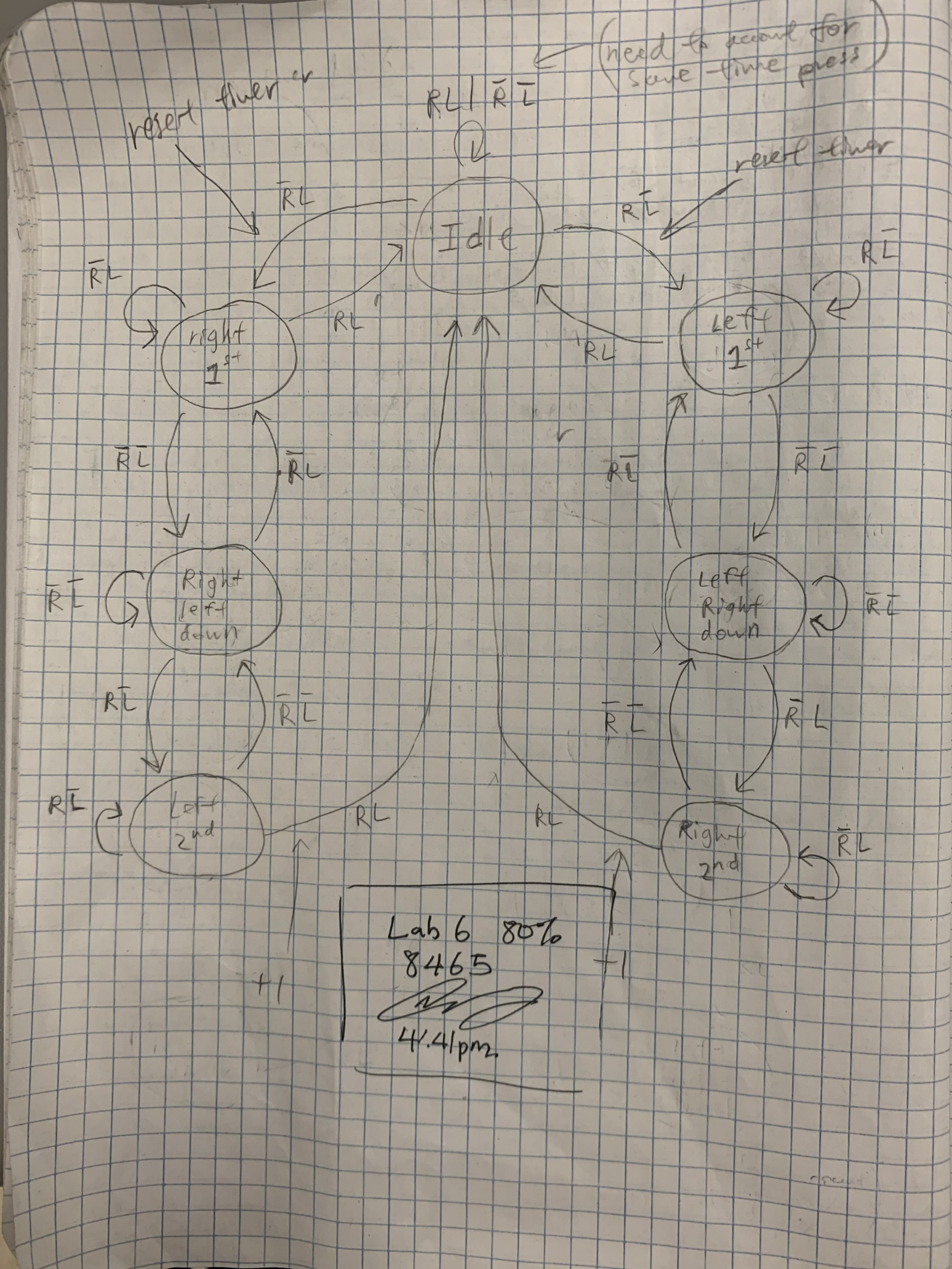
- btn R (right)
- btn L (left)
- btn U (reset)

## What I can Reuse

- edge detector
- counter

## Game Plan

- detect last 2 btn presses to determine direction
- need to account for triggering sensors and leaving the way they came
- include extra output/function so you don't get same problem from lab 5


need to account for same-time press
reset timer
reset timer
Idle
right 1st
Left 1st
Right left down
Left Right down
Left 2nd
Right 2nd
Lab 6 80%
8465
4.41pm
+1
-1

R = right sensor unblocked

L = left sensor unblocked

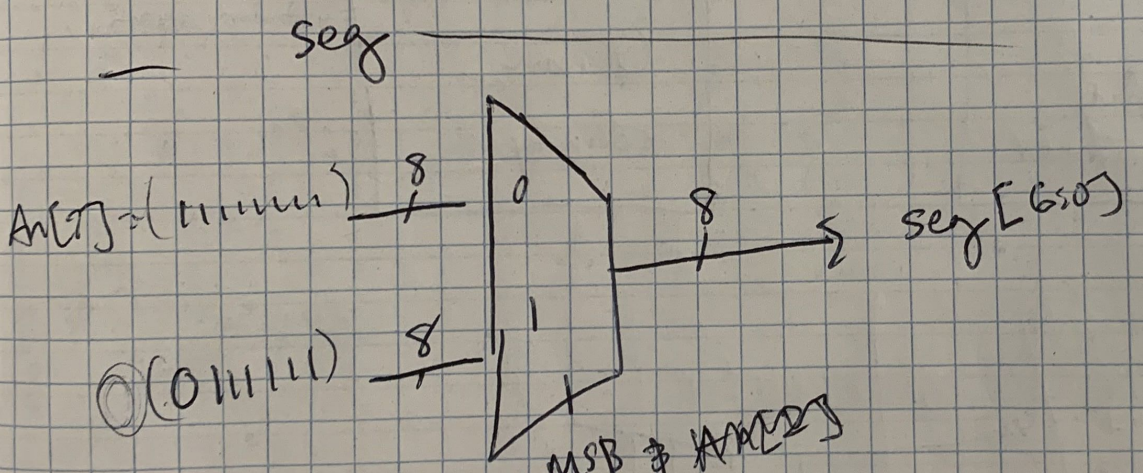
input
- left sensor
- right sensor

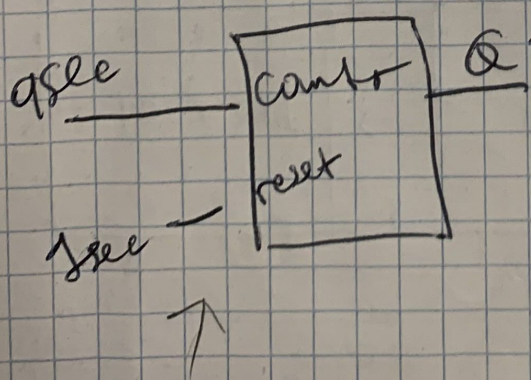
output
- show time
- reset timer
- increment up
- increment down

showtime = ! Idle & ! right left finish & ! left right finish

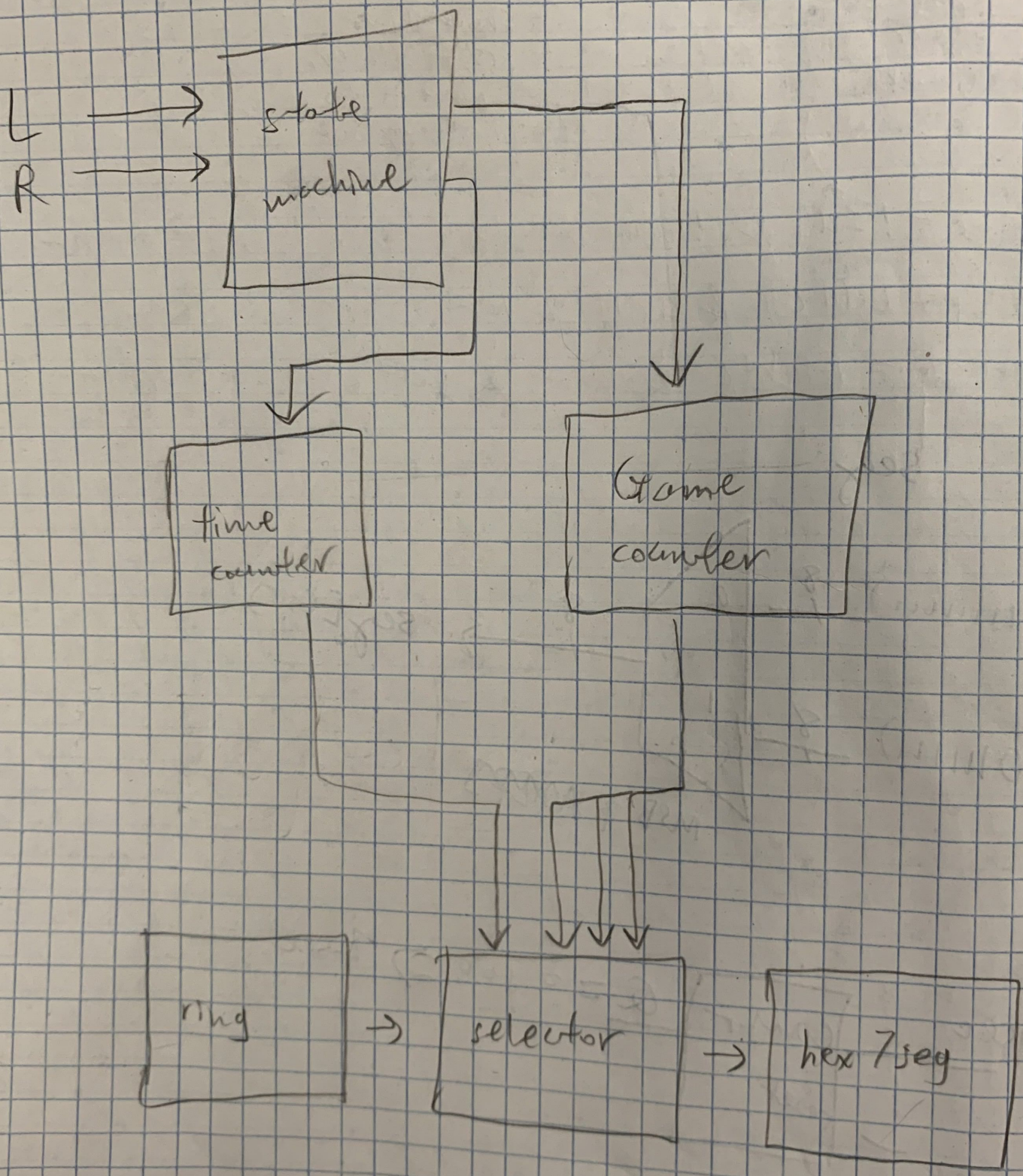
increment up = Left2 & right & left

increment down = Right2 & right & left




1 second counter, controls CE for time counter


L
R
state machine
time counter
frame counter
ring
selector
hex 7seg

## Turkey Counter

- 2's complement = inverse +1
- can use incrementer from lab 3
  - probably need to reverse a bunch of stuff

$-8 + 1 = -7$

| 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| + 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 | 0 | 0 | 1 |

pos/neg decider ↑

$8 = 0\ 0\ 0\ 0\ 1\ 0\ 0\ 0$

$1\ 1\ 1\ 1\ 0\ 1\ 1\ 1$

$+ \quad\quad\quad\quad\quad\quad\quad 1$

$-8 = 1\ 1\ 1\ 1\ 1\ 0\ 0\ 0$

comp = 1 if minus = 1 & plus = 0

comp = 0 if minus = 0 & plus = 1

| m | p | Comp |
|---|---|---|
| 0 | 0 | |
| 0 | 1 | 0 |
| 1 | 0 | 1 |
| 1 | 1 | |

$comp = m \,\&\, \sim p$

- invert zero to get negative sign

F to -1

$1\ 1\ 1\ 1\ 1\ 1\ 1\ 1 = F = -1$

$0\ 0\ 0\ 0\ 0\ 0\ 0\ 0$

$+ \quad\quad\quad\quad\quad\quad\quad 1$

$0\ 0\ 0\ 0\ 0\ 0\ 0\ 1 = 1$