CSE100 Lab Report 6

Turkey counting with state machines

Student Name: Artyom Martirosyan

Lab Sec: 1B

Date: 5/23/2020

Description:

For this assignment the students will be creating sensor logic in order to solve an important issue at our UCSC: recently the school has been having trouble keeping count of how many turkeys are populating the campus therefore they want for the student to implement two sensors(btnL and btnR) in order to keep track of the current turkey population. In order to do this the student is tasked with creating a state machine which will count up if the turkey enters from the right sensor and leaves from the left and counts down if the turkey goes from the left sensor to the site, but it will not count if the turkey exits the same way it enters meaning that if the turkey goes in from right and out right of in from left and leaves from the left sensor it will not count. Another thing to note is that these turkeys are rather large meaning that it is not possible for a turkey to be in between the sensors as the distance between the two sensors is rather small. The student must also implement a counter which will count the number of seconds the turkey is interacting with the sensor(from 1 to 15). In order to do this the student will also need to implement a counter as well as an eight bit adder/subtractor to keep track of the number of turkeys inside of the campus. The student is also tasked with displaying the number properly meaning that if there are alot of turkeys leaving the campus the students display should also display negative eight bit numbers. The student will also need to implement a seven segment converter in order to display the numbers properly on the analogue display as well as a selector and ring counter in order to show which value is being displayed. This is due to the fact that the display can only display one value at a time in each analogue area meaning that the only way to make it seem as if the display is constant the user will have to alternate the values being displayed at a very fast rate in order for the alternating values to seem transparent.

Methods:

T Flip-Flop:

Since this module was developed in a previous lab(lab 4) below will be the same instructions provided from that manual:

If the student reads through the textbook they will hopefully uncover a schematic for a functioning up down counter and will notice that the counter relies on t flip-flops. Once releasing this they can also read the next section and find the schematic for the t flip-flop as shown below:

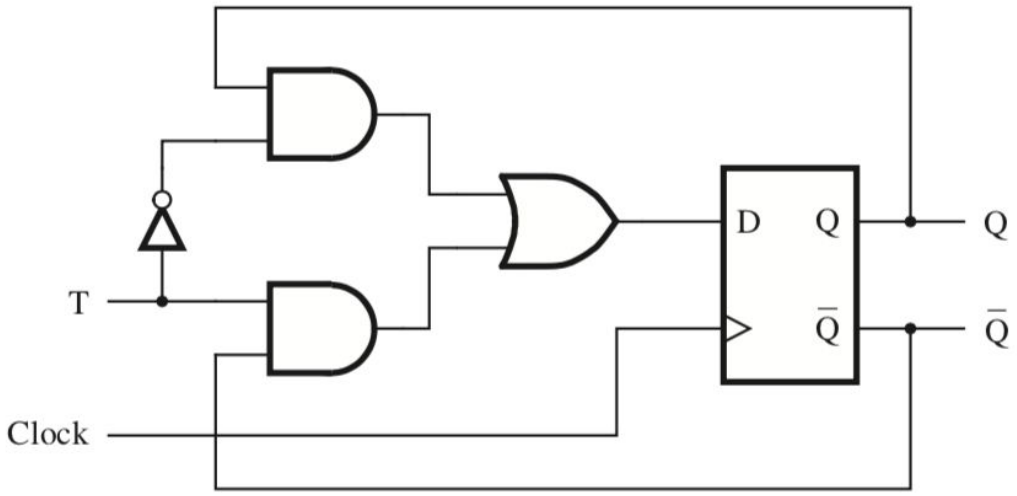


*Figure 1 is a schematic of a T flip-flop provided by the textbook

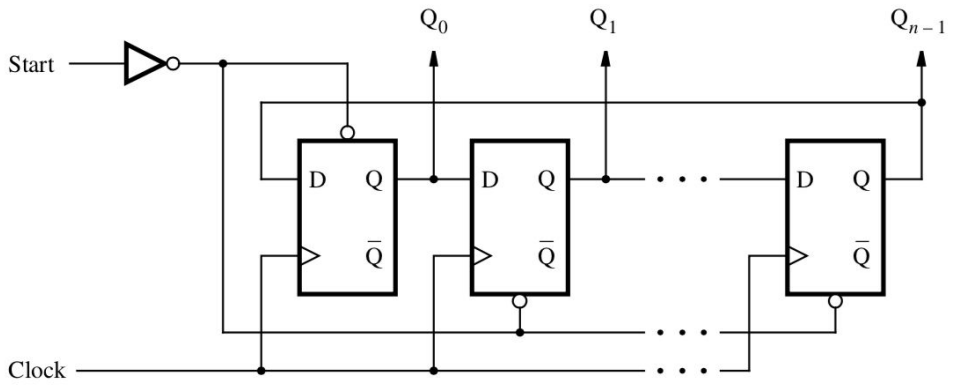
From this schematic the student should be able to create a T-flip flop

Selector:

Again this module was taken from lab 4 therefore here is the previous explanation: The selector can be looked at as a four to four multiplexor as it will be receiving a four bit value from the ring counter(where one of the four bits will be one and the rest zero). And it will simply select 4 bits according to the selector, for example if I was to receive 1000 meaning that the most significant bit of the selector is one i would take bits 15:12 from the 16 bit number given to me(the most significant four bits). Again please look in the section below for a schematic.

Ring counter:

For the ring counter the student can take the same ring counter logic used in the previous assignment(lab 4) the student will be reluctant to find a schematic for an n-bit ring counter in which if the student wishes to add more bits they will simply have to add an additional flip flop as seen below:



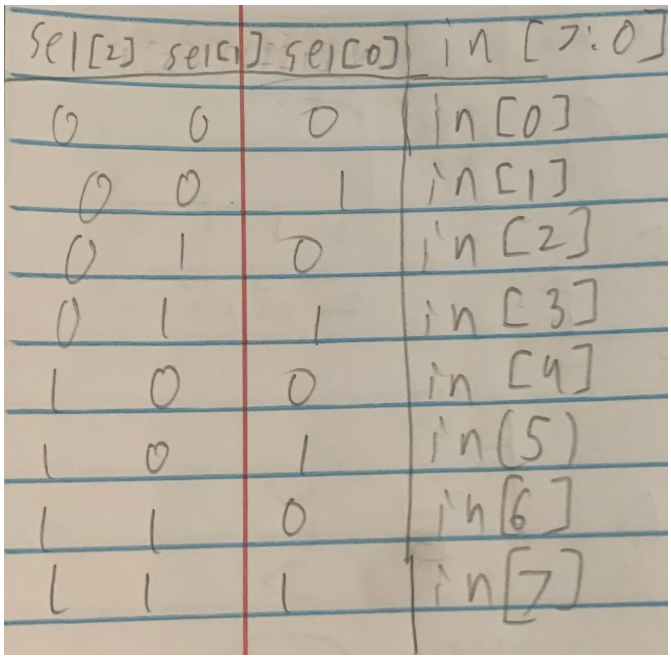
*The figure above(figure 2) is a schematic of an n-bit ring counter provided from the textbook.

8 to 1 Mux(m8_1e):

Note: the section below is a snippet from the lab 3 writeup since the multiplexor is implemented from the hex 7 segment display which is also borrowed from lab 3. This will be added to the manual with the assumption that the reader does not have access to previous lab manuals

For this module the student must start off by writing out the truth table and deriving a boolean expression for the multiplexer:

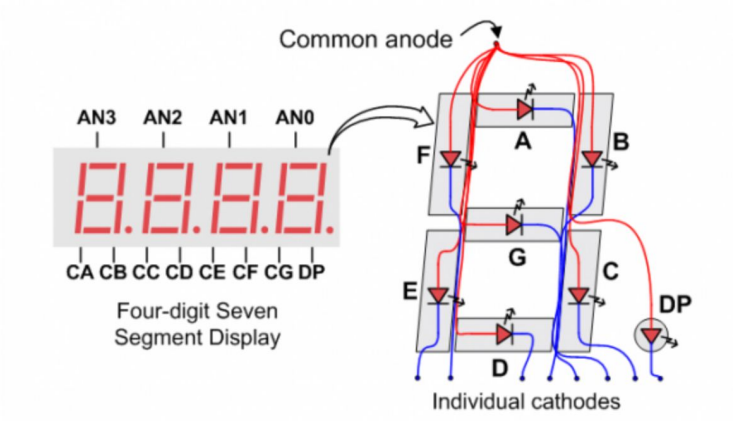
*The figure below(fig 3) is a truth table for m8_le taken from lab manual 3



From this the student is able to derive the boolean expression(note: actual schematic in results):
Output = e & ((~sel[2]&~sel[1]&~sel[0] & in[0]) |
(~sel[2]&~sel[1]&sel[0]& in[1]) |
(~sel[2]&sel[1]&~sel[0]& in[2]) |
(~sel[2]&sel[1]&sel[0]& in[3]) |
(sel[2]&~sel[1]&~sel[0]& in[4]) |
(sel[2]&~sel[1]&sel[0]& in[5]) |

Seven segment display:

Again rather than repeating the same instruction from the previous lab manual(lab 3) here is the snipped from the lab 3 writeup(will be sourced below). Before we can design the seven segment display we must first understand the logic behind the display. For starters the display is an active high component meaning that the led/analogue will light up when it is set to zero and be turned off when it is set to one. Also each line in the individual analogues are depicted using 7 lines meaning that you will need to set the appropriate lines depending on the value you want displayed.



* figure 4 above is a schematic of the displays on the basys 3 circuit board taken from the reference manual

Since we are required to use multiplexors for the seven segment display we will start off by creating a truth table for one component of the display:

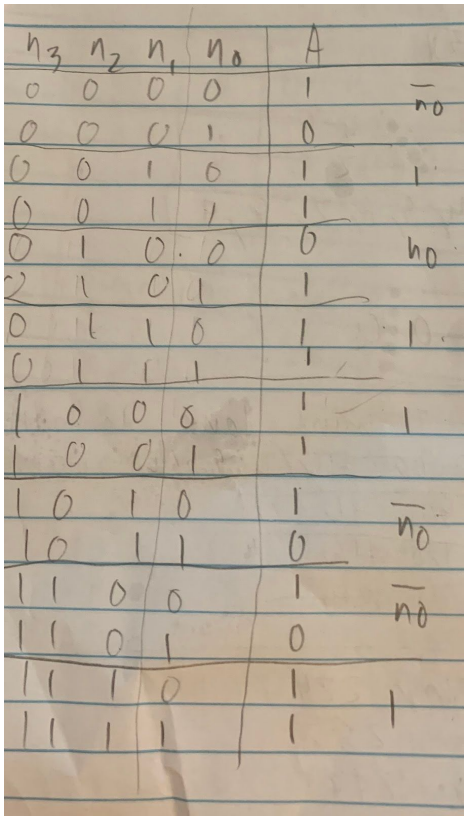


*Figure 5 to the left is a truth table of what A from the display above will represent according to the 4 bit vector it is provided.

As seen to the left we will be using n[3:1] as the switches for the mux. We will also use {1,~n[0], ~n[0], 1, 1, n[0], 1, ~n[0]} as the either bit input into the 8-1 multiplexor. The module will also take in an enable making sure that the wrong value isn't being displayed on one of the two analogues.

Four bit up/down counter(countUD4L):

For the four bit counter the student can simply refer to lab manual 4(however in the case that the student does not have access to this here is the module from manual 4, this will be used to keep score for each player)The student will be creating a four bit up/down counter with a load capability meaning that the student must have a load function attached to each flip flop. Fortunately, the textbook has a schematic of a three-bit up down counter without load capabilities as seen below:

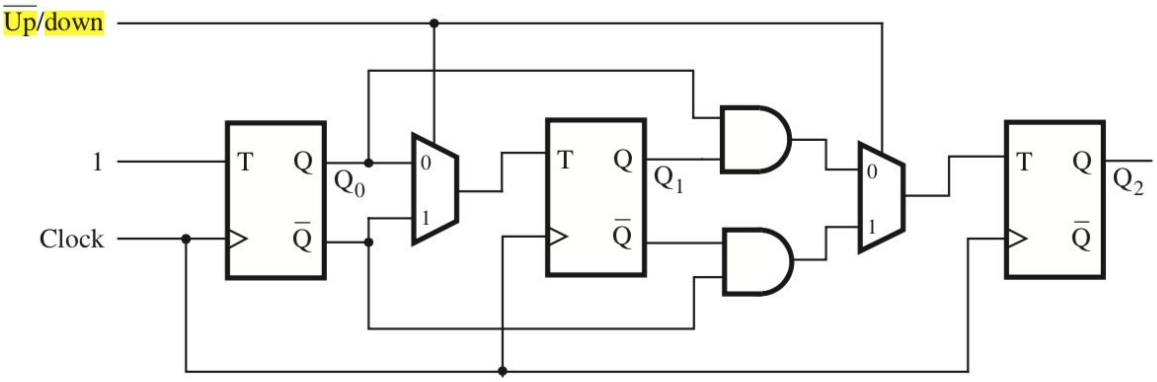
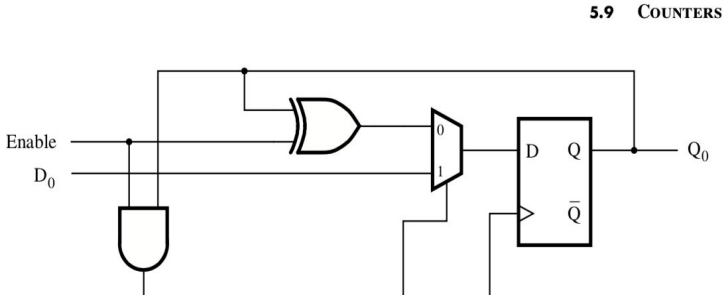


*Figure 6 above is an up/down 3-bit counter without load capabilities

With this data we can simply add an extra t-flip flop which would require another two to one multiplexor which would hold the logic and of all of the results from the flip flops(Q for upward and ~Q for downward counting). In order to add loading we can simply add an additional multiplexor which would decide if a load is being initialized(not in order to load a value you will need to do an xor with the result from the T flip-flop and the bit being loaded). Warning before you implement this keep in mind that you will be requiring a t-flip flop therefore you must create an additional module for the t flip flop and implement the schematic above. Below is an example of a parallel load on a counter(input logic should still be the same with an up/down counter as you simply need a multiplexor before the T inputs). For UTC and DTC simply and all of the ~Qs for DTC and all of the Qs for UTC.



The image to the left(figure 7) is a design for a load. As seen to the left you will need to xor the output value of the flip flip with the load bit before running it into a secondary multiplexor.

eight bit up/down counter(countUD4L):

For the eight bit counter I simply connected two 4 bit adders, but the condition for counting for the second adder is either utc and count up or dtc and countdown.

16 bit up/down counter(countUD26L):

(again module below is taken from the lab 4 manual this time it will be used as the countdown clock)For the 16 bit up/down counter we will simply be implementing the design provided to us from the image below:

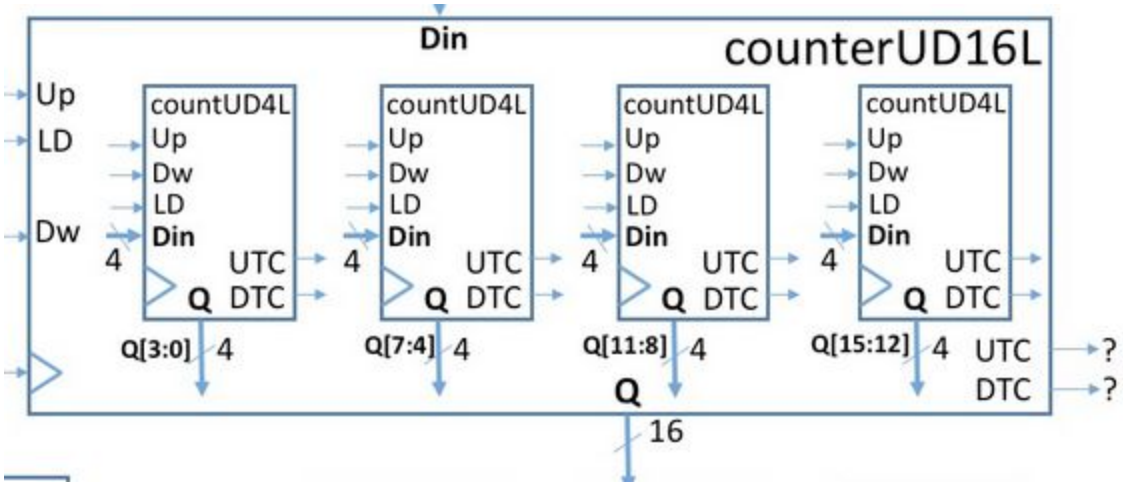


*Figure 8 above is a basic schematic of the 16 bit up/down counter taken from the lab four manual

As seen above we will simply be attaching several 4 bit counters to each other as well as loading in the appropriate four bits into each counter. Note: there are a few extra logical aspects not mentioned in the schematic above which will be mentioned in detail below. After the results from the four bit counters we can simply combine all 16 bits as the output as well as having the UTC be a logical and of the four UTCs from the four counters. The logic for the DTC is similar to the logic for UTC, but with the DTCs from the four counters instead.

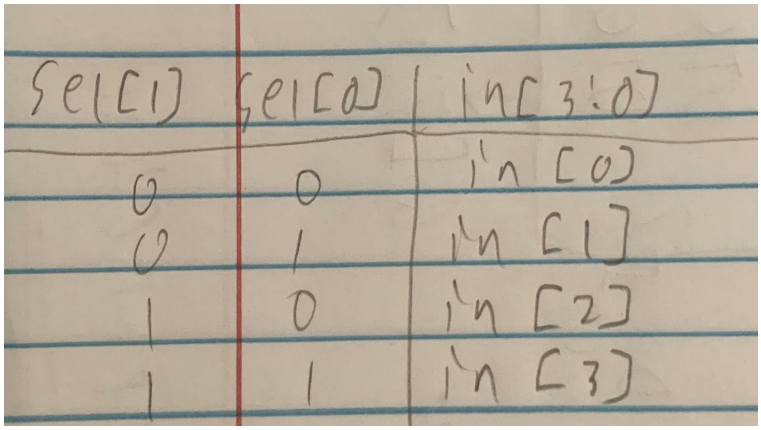
2 to 8 Mux(m2_8):

(taken from lab 3)For this Module we start off by reading the inputs and outputs and realise that if the selector is 1 then i'd send the first input and if its zero id send the second input. Also notice that the inputs were two eight bit values as well as one selector and the output was supposed to be an eight bit value. From this information we

derived the logic: output = {8{sel}} & input1 | ~{8{sel}} & input2. The reason that we need to multiply the selector by eight is so that we can simply do a logical and as the result would either be input 1 anded with eight ones or either zeros depending on the selector(Note since there is no truth table both schematics will be below).

4 to 1 Mux(m8_1):

(taken from lab 3 manual)For this module I started off with a truth table as seen below:



From this truth table we are able to derive the following logic:
Output = e & ((~sel[1] & ~sel[0] & in[0]) | (~sel[1] & sel[0] & in[1]) | (sel[1] & ~sel[0]& in[2]) | (sel[1] & sel[0] & in[3])).

*figure 9 is a truth table for a 4 to 1 multiplexor.
You will also have to add an enable since the expected inputs are in[3:0], sel[1:0], and e
And the expected output is o.

Full adder:

For this module the student is required to use several 4-1 multiplexers in order to create a full adder. To start they will create another table and once the table is created the student will notice that there is a relation between the sum and the carryout as seen below(this module is taken from lab 3):

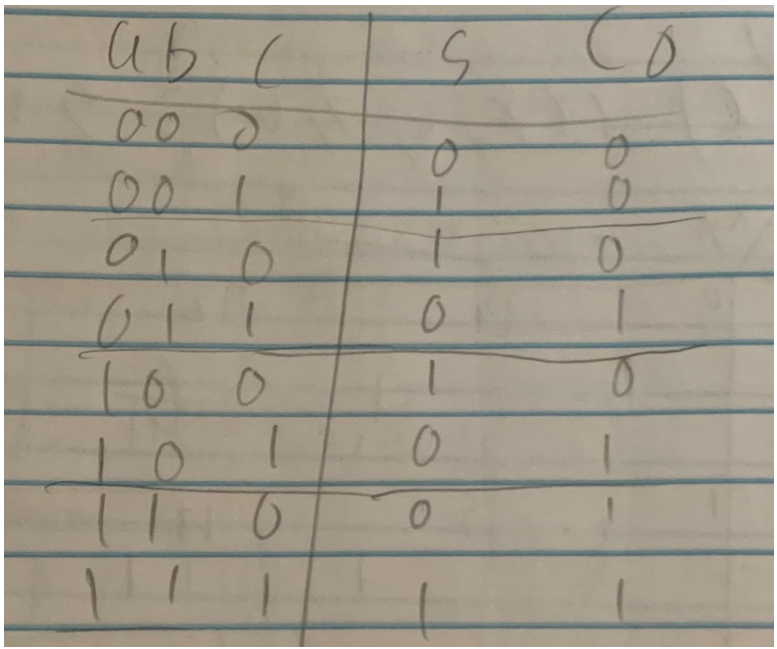


*figure 10 to the left is a truth table of a full adder.

From this we are able to use two 4-1 multiplexer which would use a and b as switches and {1,C,C,0} for Carry out and {C, ~C, ~C, C} for the Sum(note a better schematic will be provided below).

Eight bit adder:

Again this module is taken from lab 3. For the eight bit adder we will simply call 8 full adders and use wires to send the carry from the previous adder to the next one(Note a full schematic will be available below). The formula for overflow is ovfl = (a[7] ~^ b[7]) & (a[7] ^ sum[7]); as we can only have overflow when both numbers are positive or negative they cannot have an overflow if they are both the same sign.

adder/subtractor:

Similar to the module above the schematic will be provided below due to the fact that this module simply took the last eight bits from the 16 bit input and put them into a the two to eight multiplexor(used btnU as switch) and sent that result to the eight bit adder.

State machine:

For this lab we will also be creating a state machine which will keep track of how the turkey is interacting with the sensors(two buttons). Because of this you will need a 7-state state machine which takes clk, left, right as inputs and countup, incup, incdown, resettime, and show clock as outputs(logic and schematic below).

Top Module:

For the top module the user will be taking in several inputs clkin, btnU, btnL, btnR and have an[], dp, led[], seg[] as outputs. There will be several important things to keep track of: one being that the clock currently used isn't representing seconds meaning that we will need to use qsec as the clock input for the 16 bit counter, but you will also need to make sure that you are displaying bits [5:2] as the counter will be incrementing every quarter second meaning Q[0] represents a quarter second, Q[1] represents a half second, Q[2] represents a second and we wish to have a four bit counter counting in seconds therefore we will only be displaying those bits. For the leds we will simply assign led [15] to btnL and led[9] to btnR. There are several methods for displaying a negative number, but the easiest one is to assign bits [11:8] to zero from the selector and use an m2_1x8 multiplexor to select what the value is going to be for the segments of the display. By doing this we can set the most significant bit of the turkey counter

anded with the ringer(so that when we are at the 3rd an value which is going to either display a negative sign or nothing), this would result in the segment either displaying its normal calculated value or the inverse of zero which is the negative sign. I also used the eight bit add/sub module in order to get a the positive number from a negative number in order to do this I simply subtracted the value coming from the turkey counter from 0 and used the most significant bit from the counter as the decider whether the value was going to be subtracted or not. This would give the display module the correct hexadecimal value, but also tell the negative multiplexor if the value is negative or positive so that it can adjust the sign. This is because a negative 1 is f, but you want to display a -1 therefore you will need to convert it into its two's complement, but keep track of whether the number was positive or not.

Results:

There weren't any issues when testing the components individually, the only issue that arose was with the 'clock' counter due to the fact that I had initially not considered the possible issue with continuously counting while in the counting state, this would result in the counter reaching f, but then starting over. Therefore I had also added a condition to counting up which was either when the state machine says to count up and while the Q[5:2] wasn't equal to f.

Design:

Top schematic:

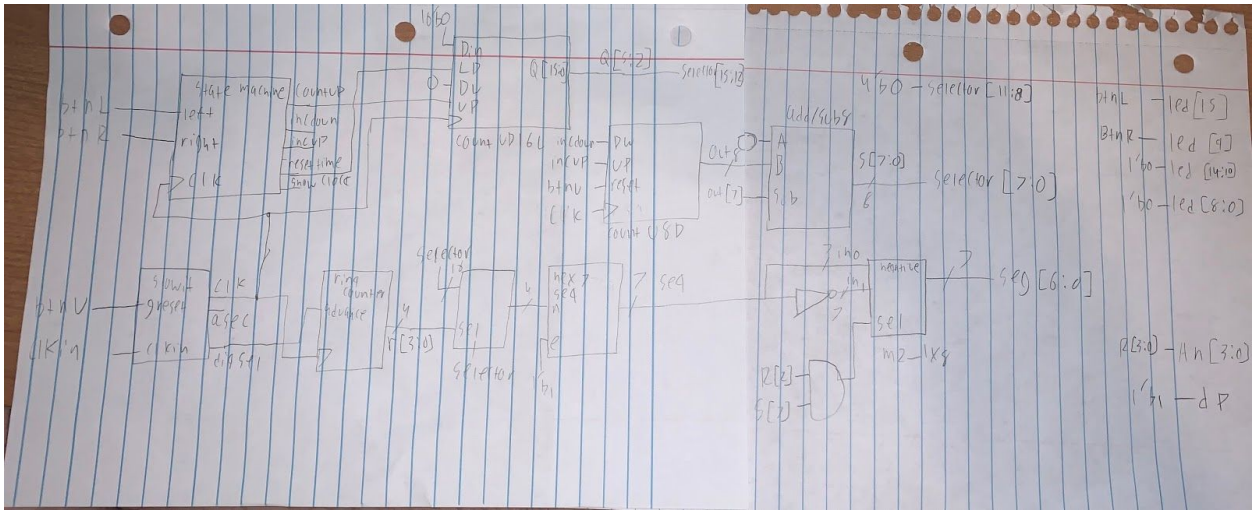


*figure 11 above is a sketch of the schematic for the top module

As seen in the figure above the buttons go into the state machine which will essentially handle incrementing or decrementing appropriately. The 'timer' will be outputting a 16 bit counter, but we are only interested in bits 5:2 as these bits are counting in seconds.

Another thing to keep in mind is the fact that we will be subtracting the counter value from zero in order to have the converted positive value, but we will still need to keep track of the most significant bit from the counter as that will be what is used in order to display the negative number.

8 bit counter:

For the eight bit counter the student will simply attach two four bit counter together as show below:

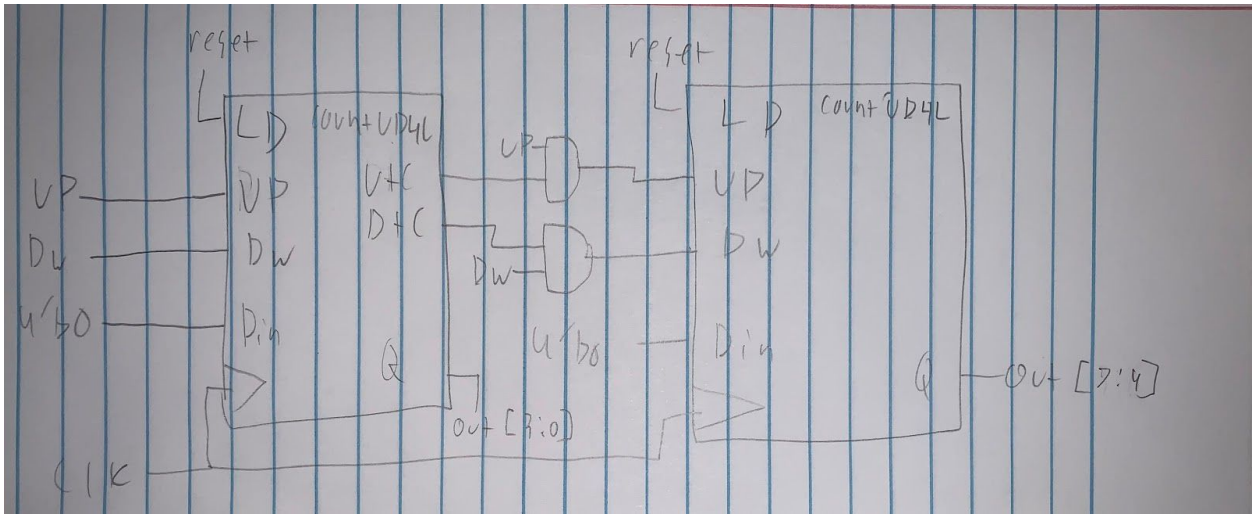
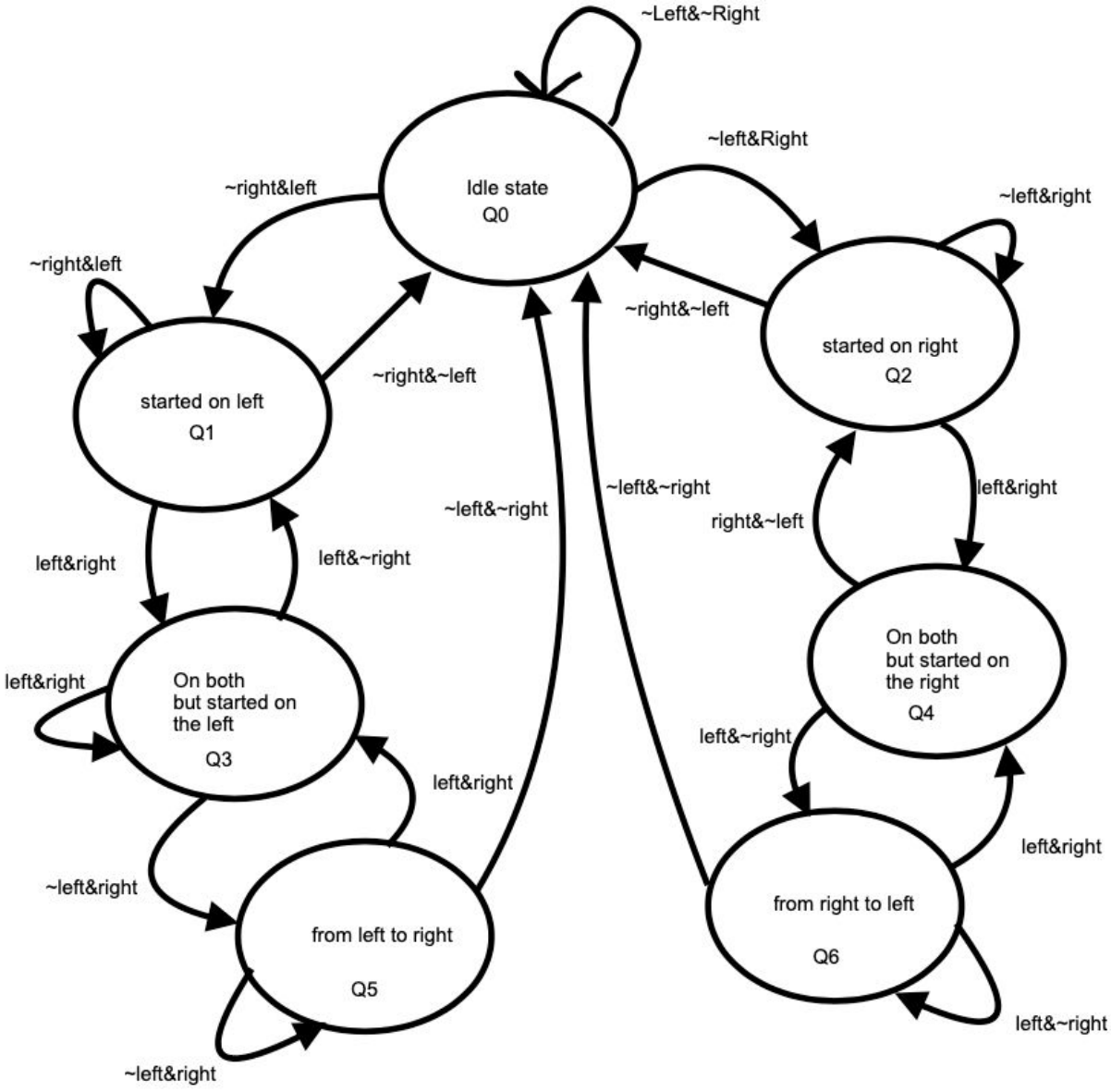


*figure 12 is a sketch of the schematic for the eight bit counter
Keep in mind that the inputs are Up, Dw, clk, reset. Therefore we can use load as reset and simply load in zero when reset is high, another thing to keep in mind is that when the counter is set to Dw or Up the upper four bits will only change when Up is high and UTC is high or Dw is low and DTC is low.

State Machine:

For the state machine we can simply take a look at the model below in order to derive the following circuit(Note: due to the size of the state machine rather than creating a circuit design here is the state diagram):



*The figure above(figure 13) is a state diagram of the state machine implemented for the lab.

We will also need to keep in mind these formulas for outputs and next states:

for reset counter: Q0 & ~right & left | Q0 & ~left & right

for start counter: ~Q0

display: ~Q0

for incrementing up:Q6 & ~left & ~right

for incrementing down:Q5 & ~Left & ~right

Q0: ~left & ~right

Q1: Q0 & ~right & left || Q1 & ~right & left || Q3 & ~right & left

Q2: Q0 & right & ~left || Q2 & right & ~left || Q4 & right & ~left
Q3: Q1 & left & right || Q3 & left & right || Q5 & left & right
Q4: Q2 & left & right || Q4 & left & right || Q6 & left & right
Q5: Q3 & right & ~left || Q5 & right & ~left
Q6: Q4 & ~right & left || Q6 & ~right & left
Q0 is the idle state, we are in this state when there is nothing touching the sensors
Q1 is the state when the turkey has entered from the left side(btnL = 1) it can also be high if the turkey is at Q3, but turns around and gets away from the right sensor.
Q2 is the state when the turkey has entered from the right it can also be high if the turkey is at Q4, but turns around and gets away from the left sensor.
Q3 is the state when both switches are covered, but the turkey started off covering the left wire.
Q4 is the state when both switches are covered, but the turkey started off covering the left wire.
Q5 is the state where the turkey is at the edge of the right side meaning it entered from the left sensor and is not at the edge of the right sensor.
Q5 is the state where the turkey is at the edge of the left side meaning it entered from the right sensor and is not at the edge of the left sensor.
We will only be incrementing/decrementing when the turkey transitions from state Q5 or Q6 back to idle.

Note: the rest of the modules are all used from previous labs therefore I will be attaching the previously written designs from Lab 3 and Lab 4:

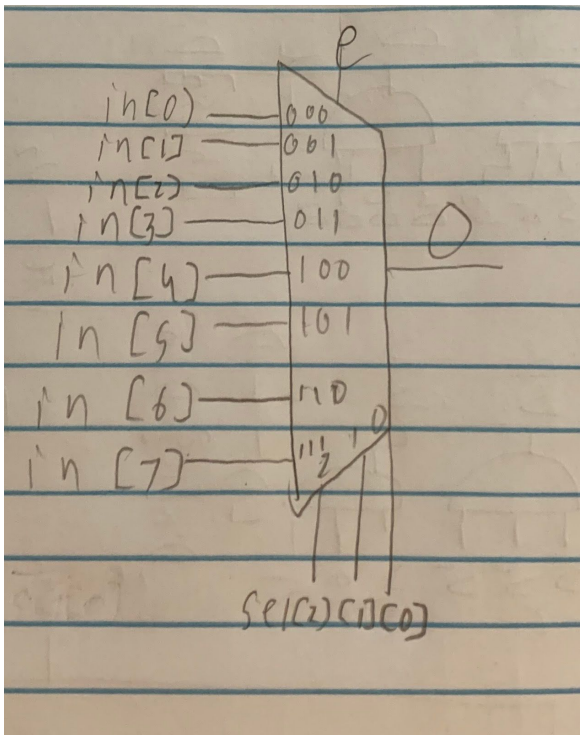
T flip-flop:

(Lab4)Rather than hand drawing the schematic for the t flip-flop please refer to the detailed depiction of a t flip-flop in the methods section above. Keep in mind the reason that these are helpful is that these will make counting up and down easier as the logic can come out of one module making the counter module easier to read. The t flip flop will take in T(one single bit to be represented) and clk as its inputs and have Q and ~Q as its outputs.

M8_1e:

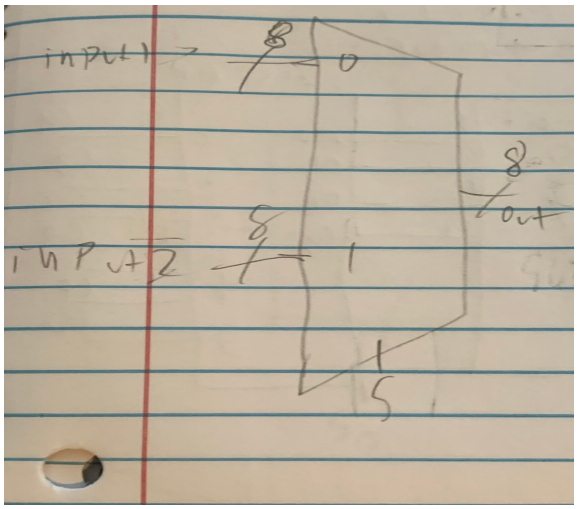
Note: for those who haven't viewed the previous manual the logic for the m8_1 multiplexor was reimplemented from the previous lab(lab 3):

*Figure 14 to the left is the schematic of the 8 to 1 multiplexor.



For the actual logic for this look at the methods section as it is simply a set of and operations which will represent each possible input.

M1_8 :



* figure 15 is a schematic of an 1_8 multiplexor

(Lab 3)As seen above the module is simply a multiplexor which has two eight bit numbers and decides which number is going to be output via a selector.

7segment display:

(note:this was taken from lab writeup 3)Due to the size of the seven segment display it is too difficult to develop a schematic, rather I will explain my logic. As seen in the methods section I created truth tables for each line of the display and I would use n[3:1] as the selecting switches meaning that I would use n[0] as the relation case since it aligned with most of the possible inputs. Once I had developed the inputs for each case(derived from the table mentioned above) I would then call 7 8-1 multiplexer(one for each line) deciding whether the inputted value would need the selected segment to be displayed. I personally believe that this module is more efficient then the logic which was implemented for the previous lab(lab3) since this module was based around multiplexores making the modules less tedious and more developed(easier to understand/debug).

CounterUD16L:

(lab 4) For the 16 bit counter you will be taking in [15:0]Din, Up, Dw, LD, clk, as inputs and UTC, DTC, as well as [15:0]Q as outputs. For the actual counting aspect follow the schematic below as each four bit counter depends on the result from the previous counter(if that counter has reached one of its limits). For assigning UTC and DTC simply and all of the UTCs and DTCs from the four 4bit counters as seen below:

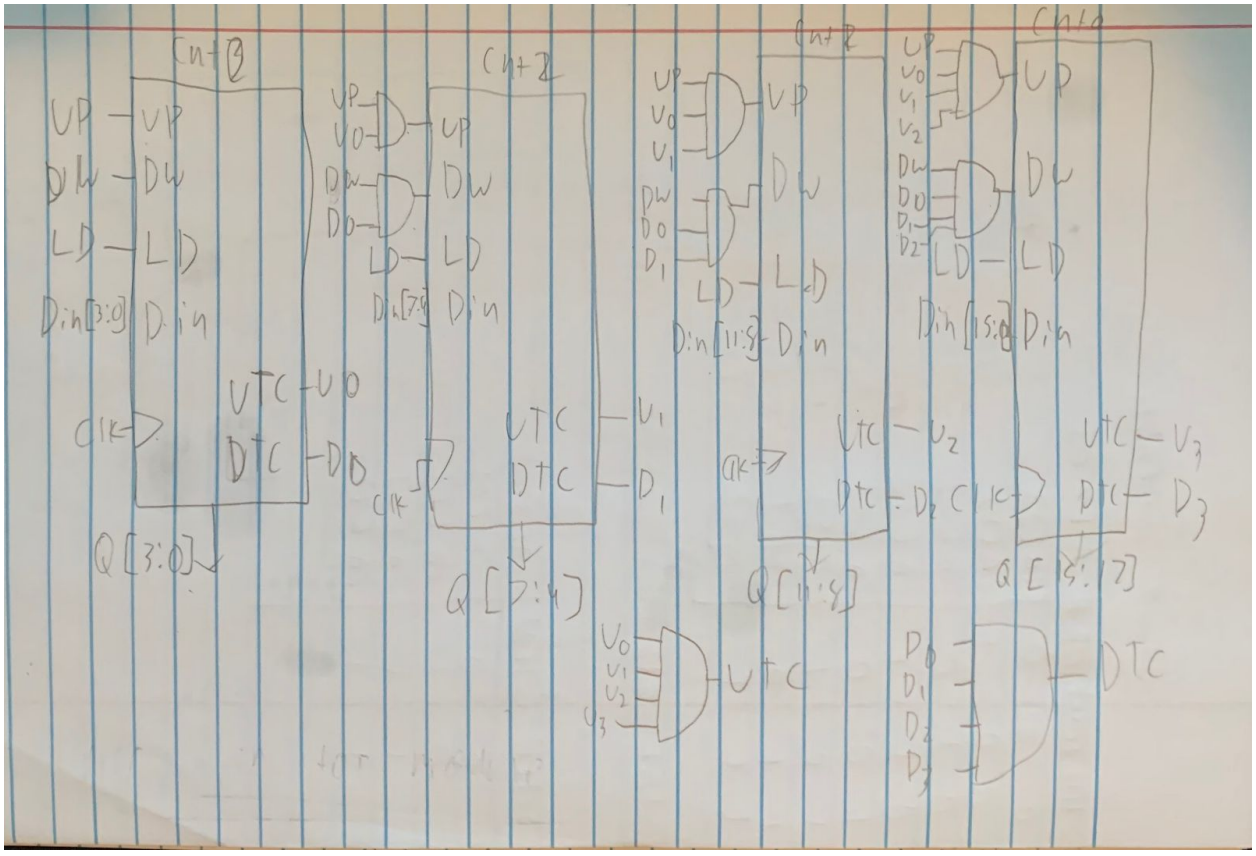
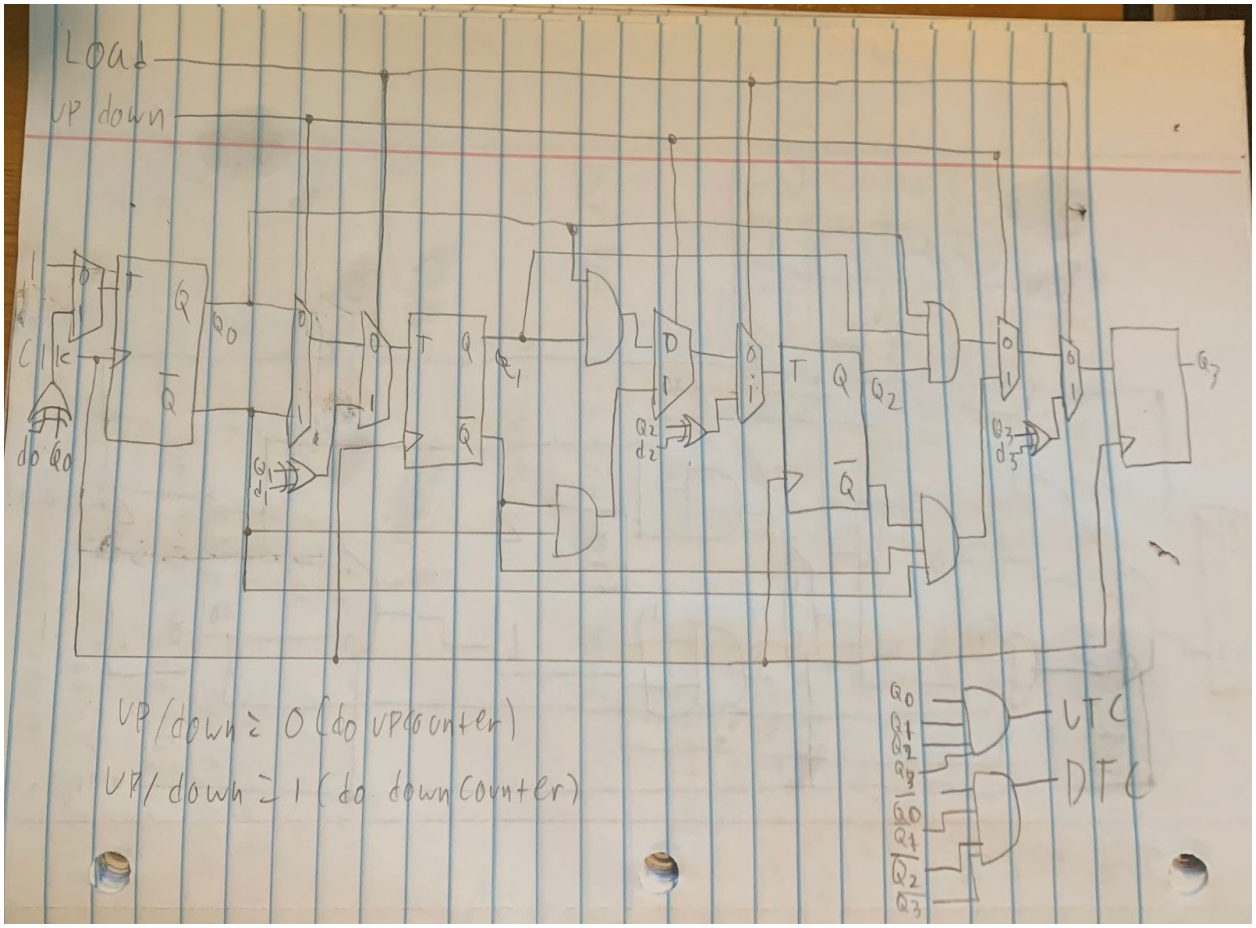


*figure 16 above is the schematic for the 16 bit up/down counter

counterUD4L:

(lab 4)For the 4 bit counter you will be taking in [3:0]Din, Up, Dw, LD, clk, as inputs and UTC, DTC, as well as Q[3:0] for the outputs. For this design we will be following the schematic provided from the textbook above as well as adding an additional multiplexor in order to be able to implement the load feature.



*The figure above(figure 17) is the schematic for a four bit up/down counter

Ring counter:

(Lab 4)For the ring counter the student should simply follow the schematic from the textbook by having four flip flops running into each other in a loop style. Where Q[3:0] are the different outputs from the flip flops.

Selector:

(lab 4)For the selector the inputs are [15:0] N, [3:0] sel, and the outputs is [3:0] H simply had each set of 4 bit multiplied by 4{sel[x]} as seen below:

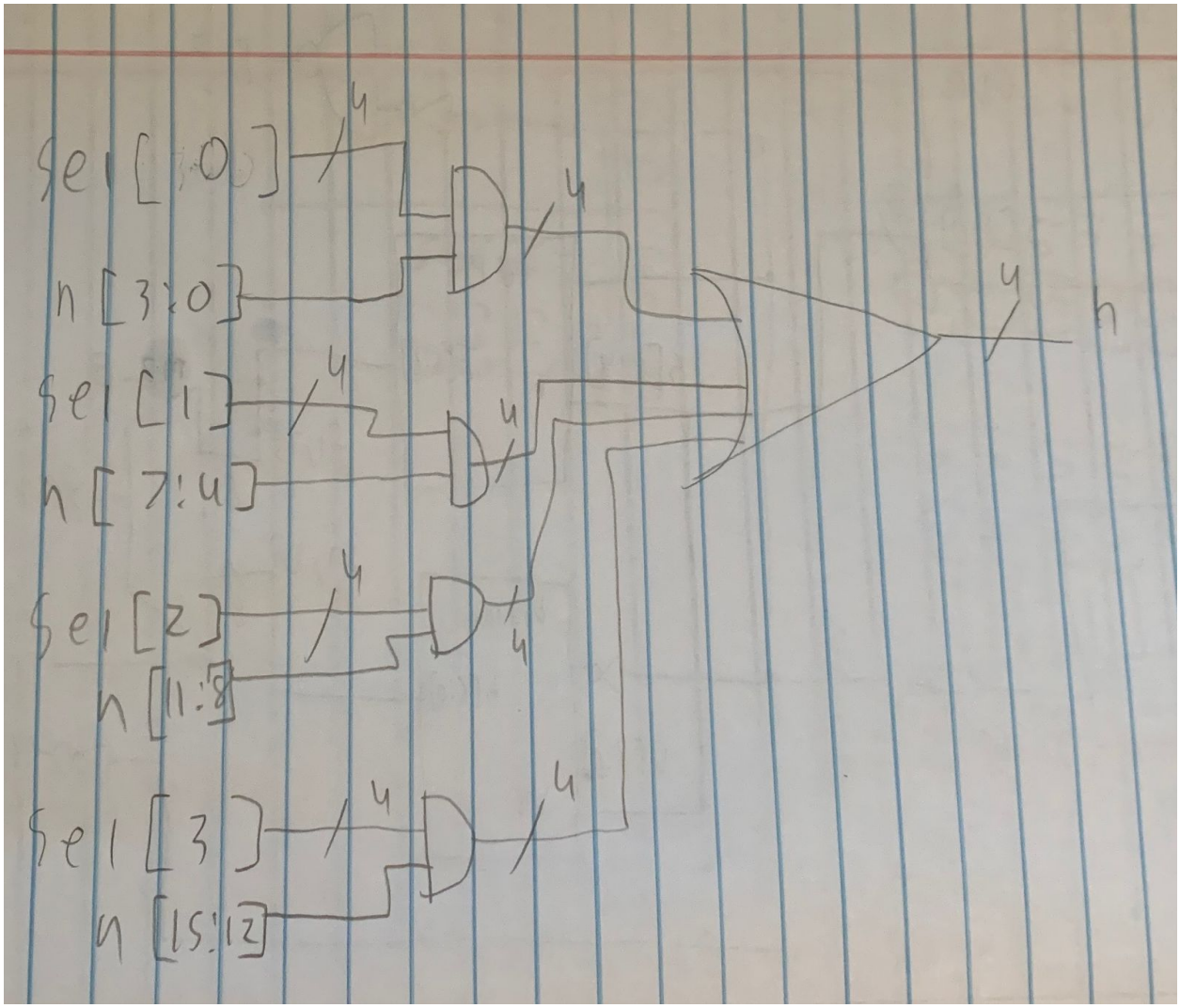


*figure 18 above is the schematic for the selector

For example the code for the first four bits would be:zero = {4{sel[0]}} & N[3:0];

Simulations:

State Machine:

For simulating the State Machine I simply tested out a possible edge case where the turkey would enter in from one side and act ‘indecisive’ by going right and left triggering both sensors before leaving from the same side it entered. I also did some individual tests to make sure that the states were transitioning properly.

Top Module:

For simulating the Top module I first imported the 7seg display file provided by the manual(shows the current values in each display) and from there I tested out all of the possible cases such as Turkey crossing left to right directly. Turkey crossing right to left directly, a turkey entering from right and wandering back and forth before retreating to the right, a turkey entering from right and wandering back and forth before crossing to the left, a turkey entering from left and wandering back and forth before retreating to the left, a turkey entering from left and wandering back and forth before retreating to the right, a turkey entering and wandering back and forth until timer reaches F. All of these provided cases passed and can be seen in the back of the manual in the appendices section.

Questions:

There are no questions for this lab

Conclusion:

This lab was relatively straightforward, the only difficulty was deciding how to approach the negative values and negative sign, but fortunately some of the issues with the negative sign were inadvertently solved before such as: subtracting the negative number in order to invert it; using an m1_8 multiplexor in order to decide whether or not the analogue was going to display a negative number. This lab also gave the student an understanding of how to implement a more complex state machine as the previous one was a very simple 3 stage one which wasn't very complicated. It also gave the students some more experience with using counters as clocks as in this scenario the student needed to understand the logic behind the counter in order to select the appropriate bits in order to be counting in seconds rather than in quarter seconds.

Sources:

Cse 100 Lab 6 manual:

https://classes.soe.ucsc.edu/cse100/Spring20/lab/lab6S20/lab6.html

‘Fundamentals of Digital Logic with Verilog design’, Stephen Brown and Zvonko Vranesic

Basys 3 reference manual:

https://reference.digilentinc.com/reference/programmable-logic/basys-3/reference-manual

Lab writeup 3 and 4

Appendices:

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 04/14/2020 09:45:16 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 ={8{sel}} & in1 | ~{8{sel}} & in0;


endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 04/14/2020 10:18:54 AM
// Design Name:
// Module Name: AddSub8
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module AddSub8(
    input [7:0] A,
    input [7:0] B,
    input sub,
    output [7:0] S,
    output ovfl
    );
    wire [7:0] eight_out;
    m2_1x8 bmux(.in0(B), .in1(~B), .sel(sub), .o(eight_out));
    eight_bit_adder calc(.a(A), .b(eight_out), .c(sub), .sum(S), .ovfl(ovfl));

endmodule
```

```
## This file is a general .xdc for the Basys3 rev B board
## To use it in a project:
## - uncomment the lines corresponding to used pins
## - rename the used ports (in each line, after get_ports) according to the top
level signal names in the project

## Clock signal
set_property PACKAGE_PIN W5 [get_ports clkin]
    set_property IOSTANDARD LVCMOS33 [get_ports clkin]
    create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports clki

## Switches
#set_property PACKAGE_PIN V17 [get_ports {sw[0]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {sw[0]}]
#set_property PACKAGE_PIN V16 [get_ports {sw[1]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {sw[1]}]
#set_property PACKAGE_PIN W16 [get_ports {sw[2]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {sw[2]}]
#set_property PACKAGE_PIN W17 [get_ports {sw[3]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {sw[3]}]
#set_property PACKAGE_PIN W15 [get_ports {sw[4]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {sw[4]}]
#set_property PACKAGE_PIN V15 [get_ports {sw[5]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {sw[5]}]
#set_property PACKAGE_PIN W14 [get_ports {sw[6]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {sw[6]}]
#set_property PACKAGE_PIN W13 [get_ports {sw[7]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {sw[7]}]
#set_property PACKAGE_PIN V2 [get_ports {sw[8]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {sw[8]}]
#set_property PACKAGE_PIN T3 [get_ports {sw[9]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {sw[9]}]
#set_property PACKAGE_PIN T2 [get_ports {sw[10]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {sw[10]}]
#set_property PACKAGE_PIN R3 [get_ports {sw[11]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {sw[11]}]
#set_property PACKAGE_PIN W2 [get_ports {sw[12]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {sw[12]}]
#set_property PACKAGE_PIN U1 [get_ports {sw[13]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {sw[13]}]
#set_property PACKAGE_PIN T1 [get_ports {sw[14]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {sw[14]}]
#set_property PACKAGE_PIN R2 [get_ports {sw[15]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {sw[15]}]
```

```
## LEDs
set_property PACKAGE_PIN U16 [get_ports {led[0]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {led[0]}]
set_property PACKAGE_PIN E19 [get_ports {led[1]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {led[1]}]
set_property PACKAGE_PIN U19 [get_ports {led[2]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {led[2]}]
set_property PACKAGE_PIN V19 [get_ports {led[3]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {led[3]}]
set_property PACKAGE_PIN W18 [get_ports {led[4]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {led[4]}]
set_property PACKAGE_PIN U15 [get_ports {led[5]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {led[5]}]
set_property PACKAGE_PIN U14 [get_ports {led[6]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {led[6]}]
set_property PACKAGE_PIN V14 [get_ports {led[7]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {led[7]}]
set_property PACKAGE_PIN V13 [get_ports {led[8]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {led[8]}]
set_property PACKAGE_PIN V3 [get_ports {led[9]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {led[9]}]
set_property PACKAGE_PIN W3 [get_ports {led[10]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {led[10]}]
set_property PACKAGE_PIN U3 [get_ports {led[11]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {led[11]}]
set_property PACKAGE_PIN P3 [get_ports {led[12]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {led[12]}]
set_property PACKAGE_PIN N3 [get_ports {led[13]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {led[13]}]
set_property PACKAGE_PIN P1 [get_ports {led[14]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {led[14]}]
set_property PACKAGE_PIN L1 [get_ports {led[15]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {led[15]}]


##7 segment display
set_property PACKAGE_PIN W7 [get_ports {seg[0]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {seg[0]}]
set_property PACKAGE_PIN W6 [get_ports {seg[1]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {seg[1]}]
set_property PACKAGE_PIN U8 [get_ports {seg[2]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {seg[2]}]
set_property PACKAGE_PIN V8 [get_ports {seg[3]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {seg[3]}]
set_property PACKAGE_PIN U5 [get_ports {seg[4]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {seg[4]}]
```

```
set_property PACKAGE_PIN V5 [get_ports {seg[5]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {seg[5]}]
set_property PACKAGE_PIN U7 [get_ports {seg[6]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {seg[6]}]

set_property PACKAGE_PIN V7 [get_ports dp]
    set_property IOSTANDARD LVCMOS33 [get_ports dp]

set_property PACKAGE_PIN U2 [get_ports {an[0]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {an[0]}]
set_property PACKAGE_PIN U4 [get_ports {an[1]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {an[1]}]
set_property PACKAGE_PIN V4 [get_ports {an[2]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {an[2]}]
set_property PACKAGE_PIN W4 [get_ports {an[3]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {an[3]}]


##Buttons
#set_property PACKAGE_PIN U18 [get_ports btnC]
    #set_property IOSTANDARD LVCMOS33 [get_ports btnC]
set_property PACKAGE_PIN T18 [get_ports btnU]
    set_property IOSTANDARD LVCMOS33 [get_ports btnU]
set_property PACKAGE_PIN W19 [get_ports btnL]
    set_property IOSTANDARD LVCMOS33 [get_ports btnL]
set_property PACKAGE_PIN T17 [get_ports btnR]
    set_property IOSTANDARD LVCMOS33 [get_ports btnR]
#set_property PACKAGE_PIN U17 [get_ports btnD]
    #set_property IOSTANDARD LVCMOS33 [get_ports btnD]



##Pmod Header JA
##Sch name = JA1
#set_property PACKAGE_PIN J1 [get_ports {JA[0]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {JA[0]}]
##Sch name = JA2
#set_property PACKAGE_PIN L2 [get_ports {JA[1]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {JA[1]}]
##Sch name = JA3
#set_property PACKAGE_PIN J2 [get_ports {JA[2]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {JA[2]}]
##Sch name = JA4
#set_property PACKAGE_PIN G2 [get_ports {JA[3]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {JA[3]}]
##Sch name = JA7
```

```
#set_property PACKAGE_PIN H1 [get_ports {JA[4]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {JA[4]}]
##Sch name = JA8
#set_property PACKAGE_PIN K2 [get_ports {JA[5]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {JA[5]}]
##Sch name = JA9
#set_property PACKAGE_PIN H2 [get_ports {JA[6]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {JA[6]}]
##Sch name = JA10
#set_property PACKAGE_PIN G3 [get_ports {JA[7]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {JA[7]}]



##Pmod Header JB
##Sch name = JB1
#set_property PACKAGE_PIN A14 [get_ports {JB[0]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {JB[0]}]
##Sch name = JB2
#set_property PACKAGE_PIN A16 [get_ports {JB[1]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {JB[1]}]
##Sch name = JB3
#set_property PACKAGE_PIN B15 [get_ports {JB[2]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {JB[2]}]
##Sch name = JB4
#set_property PACKAGE_PIN B16 [get_ports {JB[3]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {JB[3]}]
##Sch name = JB7
#set_property PACKAGE_PIN A15 [get_ports {JB[4]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {JB[4]}]
##Sch name = JB8
#set_property PACKAGE_PIN A17 [get_ports {JB[5]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {JB[5]}]
##Sch name = JB9
#set_property PACKAGE_PIN C15 [get_ports {JB[6]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {JB[6]}]
##Sch name = JB10
#set_property PACKAGE_PIN C16 [get_ports {JB[7]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {JB[7]}]



##Pmod Header JC
##Sch name = JC1
#set_property PACKAGE_PIN K17 [get_ports {JC[0]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {JC[0]}]
```

```
##Sch name = JC2
#set_property PACKAGE_PIN M18 [get_ports {JC[1]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {JC[1]}]
##Sch name = JC3
#set_property PACKAGE_PIN N17 [get_ports {JC[2]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {JC[2]}]
##Sch name = JC4
#set_property PACKAGE_PIN P18 [get_ports {JC[3]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {JC[3]}]
##Sch name = JC7
#set_property PACKAGE_PIN L17 [get_ports {JC[4]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {JC[4]}]
##Sch name = JC8
#set_property PACKAGE_PIN M19 [get_ports {JC[5]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {JC[5]}]
##Sch name = JC9
#set_property PACKAGE_PIN P17 [get_ports {JC[6]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {JC[6]}]
##Sch name = JC10
#set_property PACKAGE_PIN R18 [get_ports {JC[7]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {JC[7]}]


##Pmod Header JXADC
##Sch name = XA1_P
#set_property PACKAGE_PIN J3 [get_ports {JXADC[0]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {JXADC[0]}]
##Sch name = XA2_P
#set_property PACKAGE_PIN L3 [get_ports {JXADC[1]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {JXADC[1]}]
##Sch name = XA3_P
#set_property PACKAGE_PIN M2 [get_ports {JXADC[2]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {JXADC[2]}]
##Sch name = XA4_P
#set_property PACKAGE_PIN N2 [get_ports {JXADC[3]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {JXADC[3]}]
##Sch name = XA1_N
#set_property PACKAGE_PIN K3 [get_ports {JXADC[4]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {JXADC[4]}]
##Sch name = XA2_N
#set_property PACKAGE_PIN M3 [get_ports {JXADC[5]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {JXADC[5]}]
##Sch name = XA3_N
#set_property PACKAGE_PIN M1 [get_ports {JXADC[6]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {JXADC[6]}]
##Sch name = XA4_N
```

```
#set_property PACKAGE_PIN N1 [get_ports {JXADC[7]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {JXADC[7]}]



##VGA Connector
#set_property PACKAGE_PIN G19 [get_ports {vgaRed[0]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {vgaRed[0]}]
#set_property PACKAGE_PIN H19 [get_ports {vgaRed[1]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {vgaRed[1]}]
#set_property PACKAGE_PIN J19 [get_ports {vgaRed[2]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {vgaRed[2]}]
#set_property PACKAGE_PIN N19 [get_ports {vgaRed[3]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {vgaRed[3]}]
#set_property PACKAGE_PIN N18 [get_ports {vgaBlue[0]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {vgaBlue[0]}]
#set_property PACKAGE_PIN L18 [get_ports {vgaBlue[1]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {vgaBlue[1]}]
#set_property PACKAGE_PIN K18 [get_ports {vgaBlue[2]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {vgaBlue[2]}]
#set_property PACKAGE_PIN J18 [get_ports {vgaBlue[3]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {vgaBlue[3]}]
#set_property PACKAGE_PIN J17 [get_ports {vgaGreen[0]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {vgaGreen[0]}]
#set_property PACKAGE_PIN H17 [get_ports {vgaGreen[1]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {vgaGreen[1]}]
#set_property PACKAGE_PIN G17 [get_ports {vgaGreen[2]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {vgaGreen[2]}]
#set_property PACKAGE_PIN D17 [get_ports {vgaGreen[3]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {vgaGreen[3]}]
#set_property PACKAGE_PIN P19 [get_ports Hsync]
    #set_property IOSTANDARD LVCMOS33 [get_ports Hsync]
#set_property PACKAGE_PIN R19 [get_ports Vsync]
    #set_property IOSTANDARD LVCMOS33 [get_ports Vsync]


##USB-RS232 Interface
#set_property PACKAGE_PIN B18 [get_ports RsRx]
    #set_property IOSTANDARD LVCMOS33 [get_ports RsRx]
#set_property PACKAGE_PIN A18 [get_ports RsTx]
    #set_property IOSTANDARD LVCMOS33 [get_ports RsTx]


##USB HID (PS/2)
#set_property PACKAGE_PIN C17 [get_ports PS2Clk]
    #set_property IOSTANDARD LVCMOS33 [get_ports PS2Clk]
```

```
    #set_property PULLUP true [get_ports PS2Clk]
#set_property PACKAGE_PIN B17 [get_ports PS2Data]
    #set_property IOSTANDARD LVCMOS33 [get_ports PS2Data]
    #set_property PULLUP true [get_ports PS2Data]


##Quad SPI Flash
##Note that CCLK_0 cannot be placed in 7 series devices. You can access it using the
##STARTUPE2 primitive.
#set_property PACKAGE_PIN D18 [get_ports {QspiDB[0]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {QspiDB[0]}]
#set_property PACKAGE_PIN D19 [get_ports {QspiDB[1]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {QspiDB[1]}]
#set_property PACKAGE_PIN G18 [get_ports {QspiDB[2]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {QspiDB[2]}]
#set_property PACKAGE_PIN F18 [get_ports {QspiDB[3]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {QspiDB[3]}]
#set_property PACKAGE_PIN K19 [get_ports QspiCSn]
    #set_property IOSTANDARD LVCMOS33 [get_ports QspiCSn]
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 04/14/2020 11:25:40 AM
// Design Name:
// Module Name: eight_bit_adder
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module eight_bit_adder(
    input [7:0] a,
    input [7:0] b,
    input c,
    output [7:0] sum,
    output ovfl
    );
    wire t0, t1, t2, t3, t4, t5, t6, t7;
    Full_adder azero(.Cin(c), .a(a[0]), .b(b[0]), .s(sum[0]), .Cout(t0));
    Full_adder aone(.Cin(t0), .a(a[1]), .b(b[1]), .s(sum[1]), .Cout(t1));
    Full_adder atwo(.Cin(t1), .a(a[2]), .b(b[2]), .s(sum[2]), .Cout(t2));
    Full_adder athree(.Cin(t2), .a(a[3]), .b(b[3]), .s(sum[3]), .Cout(t3));
    Full_adder afour(.Cin(t3), .a(a[4]), .b(b[4]), .s(sum[4]), .Cout(t4));
    Full_adder afive(.Cin(t4), .a(a[5]), .b(b[5]), .s(sum[5]), .Cout(t5));
    Full_adder asix(.Cin(t5), .a(a[6]), .b(b[6]), .s(sum[6]), .Cout(t6));
    Full_adder aseven(.Cin(t6), .a(a[7]), .b(b[7]), .s(sum[7]), .Cout(t7));
    assign ovfl = (a[7] ~^ b[7]) & (a[7] ^ sum[7]);

    endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 05/12/2020 10:50:12 AM
// Design Name:
// Module Name: bcd
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module countUD8L(
    input reset,
    input Dw,
    input Up,
    input clk,
    output [7:0] out
    );
    wire high, low, countup, countdown;
    wire [7:0] Qout;
    countUD4L countlower(.Up(Up), .Dw(Dw), .clk(clk), .LD(reset), .Din(4'b0),
.DTC(low), .UTC(high), .Q(out[3:0]));
    countUD4L countupper(.Up(countup), .Dw(countdown), .clk(clk), .LD(reset),
.Din(4'b0), .DTC(), .UTC(), .Q(out[7:4]));
    assign countup = high & Up;
    assign countdown = low & Dw;

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 04/14/2020 09:32:34 AM
// Design Name:
// Module Name: m8_1e
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module m8_1e(
    input [7:0] in,
    input [2:0] sel,
    input e,
    output o
    );
    wire zero, one, two, three, four, five, six, seven;
    assign zero = ~sel[2] & ~sel[1] & ~sel[0];
    assign one = ~sel[2] & ~sel[1] & sel[0];
    assign two = ~sel[2] & sel[1] & ~sel[0];
    assign three = ~sel[2] & sel[1] & sel[0];
    assign four = sel[2] & ~sel[1] & ~sel[0];
    assign five = sel[2] & ~sel[1] & sel[0];
    assign six = sel[2] & sel[1] & ~sel[0];
    assign seven = sel[2] & sel[1] & sel[0];
    assign o = e & (in[0] & zero | in[1] & one | in[2] & two | in[3] & three | in[4]
& four | in[5] & five | in[6] & six | in[7] & seven);
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 04/23/2020 12:03:33 PM
// Design Name:
// Module Name: countUD4L
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module countUD4L(
    input Up,
    input Dw,
    input LD,
    input [3:0] Din,
    input clk,
    output UTC,
    output DTC,
    output [3:0] Q
    );
    wire updown, run;
    wire [3:0] qout, qnot, tin;
    assign run = (Up ^ Dw) | LD;
    assign updown = Dw & ~Up;                                       // assigns
updown so that if down = 1 and up = 0 count down else up
    assign tin[0] = (1'b1 & ~LD) | (LD & (Din[0]^qout[0]));

    assign tin[1] = (~LD & ((updown & qnot[0]) | (~updown & qout[0]))) | (LD &
(Din[1] ^ qout[1]));
    assign tin[2] = (~LD & ((updown & (&qnot[1:0])) | (~updown & (&qout[1:0])))) |
(LD & (Din[2] ^ qout[2]));
    assign tin[3] = (~LD & ((updown & (&qnot[2:0])) | (~updown & (&qout[2:0])))) |
(LD & (Din[3] ^ qout[3]));
    tflop zero (.clk(clk), .t(tin[0]), .enable(run), .q(qout[0]), .notq(qnot[0]));
```

```
    tflop one (.clk(clk), .t(tin[1]), .enable(run), .q(qout[1]), .notq(qnot[1]));
    tflop two (.clk(clk), .t(tin[2]), .enable(run), .q(qout[2]), .notq(qnot[2]));
    tflop three (.clk(clk), .t(tin[3]), .enable(run), .q(qout[3]), .notq(qnot[3]));
    assign Q = qout;
    assign UTC = &(qout[3:0]);
    assign DTC = &(qnot[3:0]);
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 04/06/2020 06:23:41 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 Cin,
    output Cout,
    output s,
    input a,
    input b
    );

    assign s = Cin ^ (a ^ b);
    assign Cout = (a & b) | (Cin & (a ^ b));
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 02/07/2017 11:19:41 AM
// Design Name:
// Module Name: lab4_clks
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////



module lab6_clks(
    input clkin,
    input greset,  //btnU
    output clk,
    output digsel,
    output qsec,
    output fastclk);

       wire clk_int;
       assign fastclk = clk_int;
       clk_wiz_0 my_clk_inst (.clk_out1(clk_int), .reset(greset), .locked(),
.clk_in1(clkin));
       clkcntrl4 slowclk (.clk_int(clk_int), .seldig(digsel), .clk_out(clk),
.qsec(qsec));

       STARTUPE2 #(.PROG_USR("FALSE"), // Activate program event security feature.
Requires encrypted bitstreams.
                   .SIM_CCLK_FREQ(0.0)   // Set the Configuration Clock
Frequency(ns) for simulation.
                                      )
             STARTUPE2_inst (.CFGCLK(),  // 1-bit output: Configuration main clock
output
                             .CFGMCLK(), // 1-bit output: Configuration internal
```

```
oscillator clock output
                              .EOS(),     // 1-bit output: Active high output signal
indicating the End Of Startup.
                              .PREQ(),// 1-bit output: PROGRAM request to fabric
output
                              .CLK(),  // 1-bit input: User start-up clock input
                              .GSR(greset),  // 1-bit input: Global Set/Reset input
(GSR cannot be used for the port name)
                              .GTS(),  // 1-bit input: Global 3-state input (GTS
cannot be used for the port name)
                              .KEYCLEARB(), // 1-bit input: Clear AES Decrypter Key
input from Battery-Backed RAM (BBRAM)
                              .PACK(), // 1-bit input: PROGRAM acknowledge input
                              .USRCCLKO(), // 1-bit input: User CCLK input
                              .USRCCLKTS(), // 1-bit input: User CCLK 3-state enable
input
                               .USRDONEO(), // 1-bit input: User DONE pin output
control
                               .USRDONETS() // 1-bit input: User DONE 3-state enable
output
                             );  // End of STARTUPE2_inst instantiation

endmodule

module clk_wiz_0

 (// Clock in ports
  // Clock out ports
  output        clk_out1,
  // Status and control signals
  input         reset,
  output        locked,
  input         clk_in1
 );
  // Input buffering
  //------------------------------------
wire clk_in1_clk_wiz_0;
wire clk_in2_clk_wiz_0;
  IBUF clkin1_ibufg
   (.O (clk_in1_clk_wiz_0),
    .I (clk_in1));


  // Clocking PRIMITIVE
  //------------------------------------
```

```
// Instantiation of the MMCM PRIMITIVE
//    * Unused inputs are tied off
//    * Unused outputs are labeled unused

wire        clk_out1_clk_wiz_0;
wire        clk_out2_clk_wiz_0;
wire        clk_out3_clk_wiz_0;
wire        clk_out4_clk_wiz_0;
wire        clk_out5_clk_wiz_0;
wire        clk_out6_clk_wiz_0;
wire        clk_out7_clk_wiz_0;

wire [15:0] do_unused;
wire        drdy_unused;
wire        psdone_unused;
wire        locked_int;
wire        clkfbout_clk_wiz_0;
wire        clkfbout_buf_clk_wiz_0;
wire        clkfboutb_unused;
  wire clkout0b_unused;
 wire clkout1_unused;
 wire clkout1b_unused;
 wire clkout2_unused;
 wire clkout2b_unused;
 wire clkout3_unused;
 wire clkout3b_unused;
 wire clkout4_unused;
wire        clkout5_unused;
wire        clkout6_unused;
wire        clkfbstopped_unused;
wire        clkinstopped_unused;
wire        reset_high;

MMCME2_ADV
#(.BANDWIDTH            ("OPTIMIZED"),
  .CLKOUT4_CASCADE      ("FALSE"),
  .COMPENSATION         ("ZHOLD"),
  .STARTUP_WAIT         ("FALSE"),
  .DIVCLK_DIVIDE        (1),
  .CLKFBOUT_MULT_F      (9.125),
  .CLKFBOUT_PHASE       (0.000),
  .CLKFBOUT_USE_FINE_PS ("FALSE"),
  .CLKOUT0_DIVIDE_F     (36.500),
  .CLKOUT0_PHASE        (0.000),
  .CLKOUT0_DUTY_CYCLE   (0.500),
  .CLKOUT0_USE_FINE_PS  ("FALSE"),
```

```
    .CLKIN1_PERIOD        (10.0))
  mmcm_adv_inst
    // Output clocks
   (
    .CLKFBOUT            (clkfbout_clk_wiz_0),
    .CLKFBOUTB           (clkfboutb_unused),
    .CLKOUT0             (clk_out1_clk_wiz_0),
    .CLKOUT0B            (clkout0b_unused),
    .CLKOUT1             (clkout1_unused),
    .CLKOUT1B            (clkout1b_unused),
    .CLKOUT2             (clkout2_unused),
    .CLKOUT2B            (clkout2b_unused),
    .CLKOUT3             (clkout3_unused),
    .CLKOUT3B            (clkout3b_unused),
    .CLKOUT4             (clkout4_unused),
    .CLKOUT5             (clkout5_unused),
    .CLKOUT6             (clkout6_unused),
     // Input clock control
    .CLKFBIN             (clkfbout_buf_clk_wiz_0),
    .CLKIN1              (clk_in1_clk_wiz_0),
    .CLKIN2              (1'b0),
     // Tied to always select the primary input clock
    .CLKINSEL            (1'b1),
    // Ports for dynamic reconfiguration
    .DADDR               (7'h0),
    .DCLK                (1'b0),
    .DEN                 (1'b0),
    .DI                  (16'h0),
    .DO                  (do_unused),
    .DRDY                (drdy_unused),
    .DWE                 (1'b0),
    // Ports for dynamic phase shift
    .PSCLK               (1'b0),
    .PSEN                (1'b0),
    .PSINCDEC            (1'b0),
    .PSDONE              (psdone_unused),
    // Other control and status signals
    .LOCKED              (locked_int),
    .CLKINSTOPPED        (clkinstopped_unused),
    .CLKFBSTOPPED        (clkfbstopped_unused),
    .PWRDWN              (1'b0),
    .RST                 (reset_high));
  assign reset_high = reset;

  assign locked = locked_int;
// Clock Monitor clock assigning
```

```
//-------------------------------------
 // Output buffering
  //-------------------------------------

  BUFG clkf_buf
   (.O (clkfbout_buf_clk_wiz_0),
    .I (clkfbout_clk_wiz_0));



  BUFG clkout1_buf
   (.O   (clk_out1),
    .I   (clk_out1_clk_wiz_0));




endmodule

module clkcntrl4(
     input clk_int,
     output seldig,
     output clk_out,
     output qsec);

   //wire XLXN_38;
   //wire XLXN_39;
   wire XLXN_44;
   wire XLXN_47;
   wire XLXN_70;
   wire XLXN_71;
   wire XLXN_72;
   wire XLXN_73;
   wire XLXN_74;
   wire XLXN_75;
   wire XLXN_76;
   wire XLXN_77;
   wire XLXN_79;
   wire clkb2_DUMMY;

   GND  XLXI_24 (.G(XLXN_44));

   (* HU_SET = "XLXI_37_73" *)
   CB4CE_MXILINX_clkcntrl4  XLXI_37 (.C(clkb2_DUMMY),
                                     .CE(XLXN_73),
                                     .CLR(XLXN_76),
```

```
                                .CEO(XLXN_72),
                                .Q0(),
                                .Q1(),
                                .Q2(XLXN_74),
                                .Q3(),
                                .TC());
(* HU_SET = "XLXI_38_74" *)
CB4CE_MXILINX_clkcntrl4  XLXI_38 (.C(clkb2_DUMMY),
                                .CE(XLXN_72),
                                .CLR(XLXN_76),
                                .CEO(XLXN_70),
                                .Q0(),
                                .Q1(),
                                .Q2(),
                                .Q3(),
                                .TC());
(* HU_SET = "XLXI_39_75" *)
CB4CE_MXILINX_clkcntrl4  XLXI_39 (.C(clkb2_DUMMY),
                                .CE(XLXN_70),
                                .CLR(XLXN_76),
                                .CEO(XLXN_71),
                               .Q0(),
                                .Q1(),
                                .Q2(),
                                .Q3(XLXN_77),
                                .TC());
//(* HU_SET = "XLXI_40_76" *)
CB4CE_MXILINX_clkcntrl4  XLXI_40 (.C(clk_out),
                                .CE(XLXN_73),
                                .CLR(XLXN_76),
                                .CEO(XLXN_78),
                                .Q0(),
                                .Q1(),
                                .Q2(),
                                .Q3(),
                                .TC(XLXN_75));

CB4CE_MXILINX_clkcntrl4  XLXI_45 (.C(clk_out),
                                .CE(XLXN_78),
                                .CLR(XLXN_76),
                                .CEO(XLXN_79),
                                .Q0(),
                                .Q1(),
                                .Q2(),
                                .Q3(),
                                .TC());
```

```
   CB4CE_MXILINX_clkcntrl4  XLXI_44 (.C(clk_out),
                                    .CE(XLXN_79),
                                    .CLR(XLXN_76),
                                    .CEO(),
                                    .Q0(qsec0),
                                    .Q1(qsec2),
                                    .Q2(),
                                    .Q3(),
                                    .TC());

   AND3  I_12222 (.I0(qsec0),
                  .I1(qsec2),
                  .I2(XLXN_79),
                  //.I3(),
                  .O(qsec3));



   VCC  XLXI_41 (.P(XLXN_73));
   GND  XLXI_43 (.G(XLXN_76));
   BUF  XLXI_328 (.I(clk_int),
                  .O(clkb2_DUMMY));
`ifdef XILINX_SIMULATOR
   BUF  XLXI_336 (.I(XLXN_75),.O(seldig));
   BUF  XLXI_398 (.I(XLXN_75),.O(qsec));
   BUFG  XLXI_399 (.I(clk_int),.O(clk_out));
   //BUFG  XLXI_401 (.I(XLXN_77),.O(clk_out));
`else
   BUF  XLXI_336 (.I(XLXN_75),.O(seldig));
   BUF  XLXI_398 (.I(qsec3),.O(qsec));
   BUFG  XLXI_401 (.I(XLXN_77),.O(clk_out));
`endif

endmodule


module FTCE_MXILINX_clkcntrl4(C,
                              CE,
                              CLR,
                              T,
                              Q);

   parameter INIT = 1'b0;

    input C;
    input CE;
```

```
    input CLR;
    input T;
   output Q;

   wire TQ;
   wire Q_DUMMY;

   assign Q = Q_DUMMY;
   XOR2  I_36_32 (.I0(T),
                  .I1(Q_DUMMY),
                  .O(TQ));
   ///(* RLOC = "X0Y0" *)
   FDCE  I_36_35 (.C(C),
                  .CE(CE),
                  .CLR(CLR),
                  .D(TQ),
                  .Q(Q_DUMMY));
endmodule
`timescale 1ns / 1ps

module CB4CE_MXILINX_clkcntrl4(C,
                               CE,
                               CLR,
                               CEO,
                               Q0,
                               Q1,
                               Q2,
                               Q3,
                               TC);

    input C;
    input CE;
    input CLR;
   output CEO;
   output Q0;
   output Q1;
   output Q2;
   output Q3;
   output TC;

   wire T2;
   wire T3;
   wire XLXN_1;
   wire Q0_DUMMY;
   wire Q1_DUMMY;
   wire Q2_DUMMY;
```

```
wire Q3_DUMMY;
wire TC_DUMMY;

assign Q0 = Q0_DUMMY;
assign Q1 = Q1_DUMMY;
assign Q2 = Q2_DUMMY;
assign Q3 = Q3_DUMMY;
assign TC = TC_DUMMY;
(* HU_SET = "I_Q0_69" *)
FTCE_MXILINX_clkcntrl4 #( .INIT(1'b0) ) I_Q0 (.C(C),
                              .CE(CE),
                              .CLR(CLR),
                              .T(XLXN_1),
                              .Q(Q0_DUMMY));
(* HU_SET = "I_Q1_70" *)
FTCE_MXILINX_clkcntrl4 #( .INIT(1'b0) ) I_Q1 (.C(C),
                              .CE(CE),
                              .CLR(CLR),
                              .T(Q0_DUMMY),
                              .Q(Q1_DUMMY));
(* HU_SET = "I_Q2_71" *)
FTCE_MXILINX_clkcntrl4 #( .INIT(1'b0) ) I_Q2 (.C(C),
                              .CE(CE),
                              .CLR(CLR),
                              .T(T2),
                              .Q(Q2_DUMMY));
(* HU_SET = "I_Q3_72" *)
FTCE_MXILINX_clkcntrl4 #( .INIT(1'b0) ) I_Q3 (.C(C),
                              .CE(CE),
                              .CLR(CLR),
                              .T(T3),
                              .Q(Q3_DUMMY));
AND4  I_36_31 (.I0(Q3_DUMMY),
               .I1(Q2_DUMMY),
               .I2(Q1_DUMMY),
               .I3(Q0_DUMMY),
               .O(TC_DUMMY));
AND3  I_36_32 (.I0(Q2_DUMMY),
               .I1(Q1_DUMMY),
               .I2(Q0_DUMMY),
               .O(T3));
AND2  I_36_33 (.I0(Q1_DUMMY),
               .I1(Q0_DUMMY),
               .O(T2));
VCC  I_36_58 (.P(XLXN_1));
AND2  I_36_67 (.I0(CE),
```

```
                .I1(TC_DUMMY),
                .O(CEO));
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 04/16/2020 09:22:50 AM
// Design Name:
// Module Name: hex7seg
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module hex7seg(
    input e,
    input [3:0] n,
    output [6:0] seg
    );
    wire [7:0] a_in;
    wire [7:0] b_in;
    wire [7:0] c_in;
    wire [7:0] d_in;
    wire [7:0] e_in;
    wire [7:0] f_in;
    wire [7:0] g_in;
    assign a_in = {1'b0, n[0], n[0], 1'b0, 1'b0, ~n[0], 1'b0, n[0]};
    assign b_in = {1'b1, ~n[0], n[0], 1'b0, ~n[0], n[0], 1'b0, 1'b0};
    assign c_in = {1'b1, ~n[0], 1'b0, 1'b0, 1'b0, 1'b0, ~n[0], 1'b0};
    assign d_in = {n[0], 1'b0, ~n[0], n[0], n[0], ~n[0], 1'b0, n[0]};
    assign e_in = {1'b0, 1'b0, 1'b0, n[0], n[0], 1'b1, n[0], n[0]};
    assign f_in = {1'b0, n[0], 1'b0, 1'b0, n[0], 1'b0, 1'b1, n[0]};
    assign g_in = {1'b0, ~n[0], 1'b0, 1'b0, n[0], 1'b0, 1'b0, 1'b1};

    m8_1e ma(.in(a_in), .sel(n[3:1]), .e(e), .o(seg[0]));
    m8_1e mb(.in(b_in), .sel(n[3:1]), .e(e), .o(seg[1]));
    m8_1e mc(.in(c_in), .sel(n[3:1]), .e(e), .o(seg[2]));
    m8_1e md(.in(d_in), .sel(n[3:1]), .e(e), .o(seg[3]));
```

```
    m8_1e me(.in(e_in), .sel(n[3:1]), .e(e), .o(seg[4]));
    m8_1e mf(.in(f_in), .sel(n[3:1]), .e(e), .o(seg[5]));
    m8_1e mg(.in(g_in), .sel(n[3:1]), .e(e), .o(seg[6]));

endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 05/12/2020 12:43:45 PM
// Design Name:
// Module Name: Topmodule
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module Topmodule(
    input clkin,
    input btnL, //left
    input btnR, //right
    input btnU, //reset
    output [15:0] led,
    output [6:0] seg,
    output [3:0] an,
    output dp
    );
    wire clk, digsel, qsec, up, down, reset, show, startcount, countcondition,
negative, left, right;
    wire [3:0] timeshown, ringer, job;
    wire [7:0] amount, turks, segtemp, actualseg;
    wire [15:0] fasttime, selinput;

    lab6_clks slowit (.clkin(clkin), .greset(btnU), .clk(clk), .digsel(digsel),
.qsec(qsec));
    //FDRE #(.INIT(1'b0)) Q0_10 (.C(clk), .CE(1'b1), .D(btnL), .Q(left));
    //FDRE #(.INIT(1'b0)) Q0_20 (.C(clk), .CE(1'b1), .D(btnR), .Q(right));
    assign left = btnL;
    assign right = btnR;
    statemachine mech(.left(left), .right(right), .clk(clk), .incup(up),
.incdown(down), .resettime(reset), .countup(startcount), .showclock(show));
```

```
    countUD8L turkey( .reset(btnU), .Dw(down), .Up(up), .clk(clk), .out(amount));
    counterUD16L timer(.Din(16'h0000), .LD(reset),  .Up(countcondition & qsec),
.Dw(1'b0), .clk(clk), .Q(fasttime), .UTC(), .DTC());
    AddSub8 comp( .A(8'h00), .B(amount), .sub(amount[7]), .S(turks), .ovfl());
    RingCounter ring( .advance(digsel), .clk(clk), .out(ringer));
    Selector selection( .sel(ringer), .N(selinput), .H(job));
    hex7seg segment( .e(1'b1), .n(job), .seg(segtemp[6:0]));
    m2_lx8 displayneg( .in0(segtemp), .in1(~segtemp), .sel(ringer[2] & amount[7]),
.o(actualseg));
    assign seg = actualseg[6:0];
    assign segtemp[7] = 0;

    assign timeshown = fasttime[5:2];
    assign selinput[15:12] = timeshown;
    assign selinput[11:8] = 4'h0;
    assign selinput[7:0] = turks;
    assign dp = 1'b1;
    assign led[15] = btnL;
    assign led[9] = btnR;
    assign an[3] = ~(ringer[3] & show);
    assign an[2] = ~(amount[7] & ringer[2]);
    assign an[1] = ~ringer[1];
    assign an[0] = ~ringer[0];
    assign led[14:10] = 5'b00000;
    assign led[8:0] = 9'b000000000;
    assign countcondition = ~(&timeshown) & startcount;
endmodule
```

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 04/23/2020 07:41:59 AM
// Design Name:
// Module Name: RingCounter
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module RingCounter(
    input advance,
    input clk,
    output [3:0] out
    );
    wire one, two, three, four ,invert;
    FDRE #(.INIT(1'b1)) Q0_FF (.C(clk), .CE(advance), .D(four), .Q(one));
    FDRE #(.INIT(1'b0)) Q1_FF (.C(clk), .CE(advance), .D(one), .Q(two));
    FDRE #(.INIT(1'b0)) Q2_FF (.C(clk), .CE(advance), .D(two), .Q(three));
    FDRE #(.INIT(1'b0)) Q3_FF (.C(clk), .CE(advance), .D(three), .Q(four));
    assign out[0] = one;
    assign out[1] = two;
    assign out[2] = three;
    assign out[3] = four;



endmodule
```

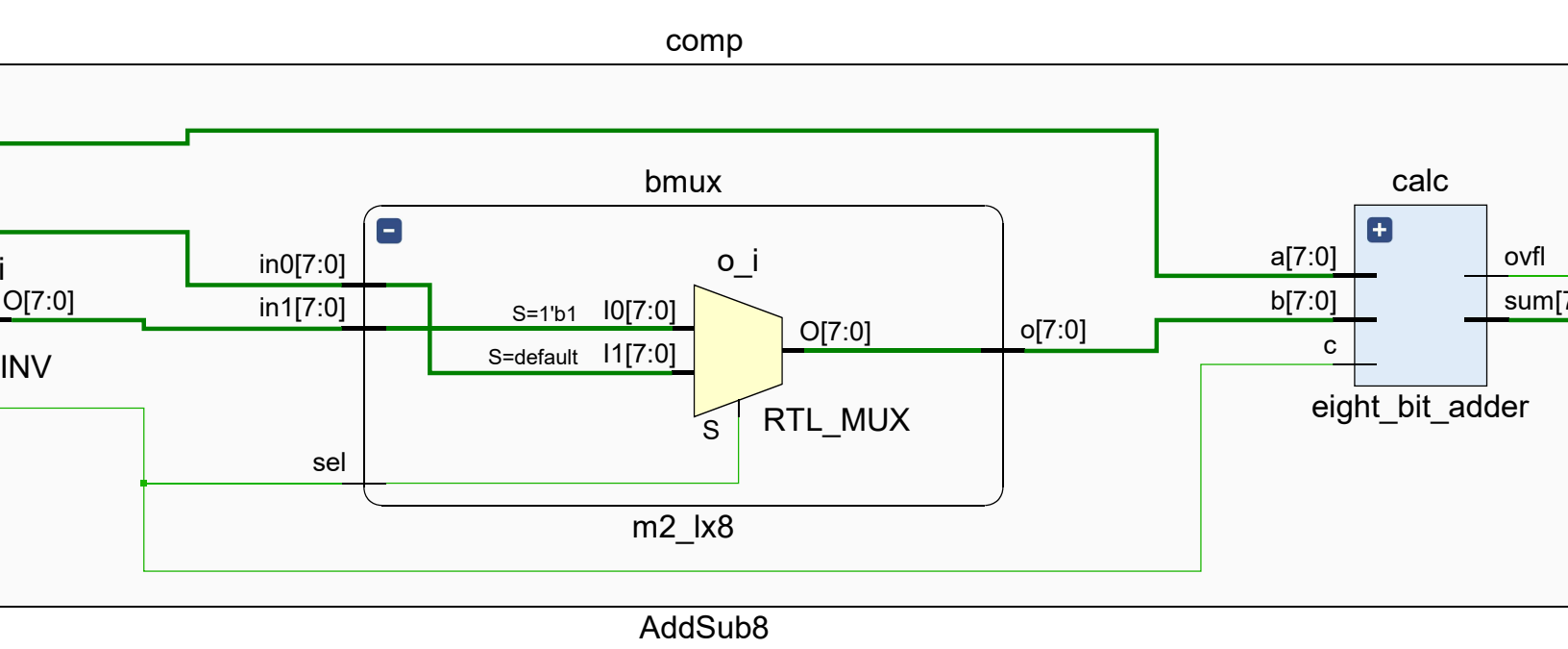
```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 05/12/2020 09:42:55 AM
// Design Name:
// Module Name: statemachine
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module statemachine(
    input left,
    input right,
    input clk,
    output incup,
    output incdown,
    output resettime,
    output countup,
    output showclock
    );
    wire [6:0] D, Q;
    FDRE #(.INIT(1'b1)) Q0_00 (.C(clk), .CE(1'b1), .D(D[0]), .Q(Q[0]));     //idle
state or Q0
    FDRE #(.INIT(1'b0)) Q0_01 (.C(clk), .CE(1'b1), .D(D[1]), .Q(Q[1]));
//starting on the left
    FDRE #(.INIT(1'b0)) Q0_02 (.C(clk), .CE(1'b1), .D(D[2]), .Q(Q[2]));
//starting on the right
    FDRE #(.INIT(1'b0)) Q0_03 (.C(clk), .CE(1'b1), .D(D[3]), .Q(Q[3]));     //from
just left to both
    FDRE #(.INIT(1'b0)) Q0_04 (.C(clk), .CE(1'b1), .D(D[4]), .Q(Q[4]));     //from
just right to both
    FDRE #(.INIT(1'b0)) Q0_05 (.C(clk), .CE(1'b1), .D(D[5]), .Q(Q[5]));     //from
left to right
    FDRE #(.INIT(1'b0)) Q0_06 (.C(clk), .CE(1'b1), .D(D[6]), .Q(Q[6]));     //from
```

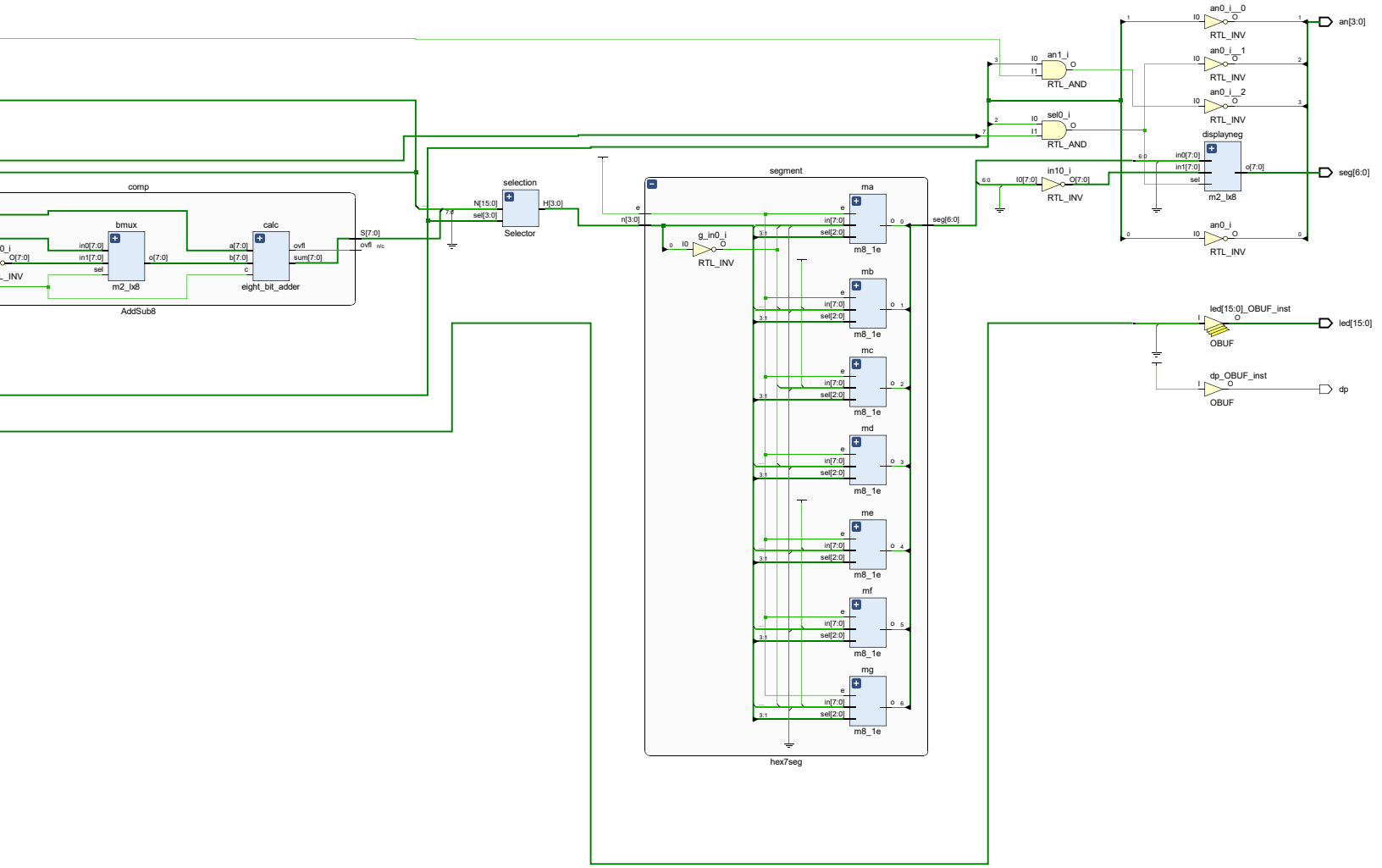
```
right to left
    assign D[0] = ~left & ~right;
    assign D[1] = (Q[0] & ~right & left) | (Q[1] & ~right & left) | (Q[3] & ~right &
left);
    assign D[2] = (Q[0] & ~left & right) | (Q[2] & ~left & right) | (Q[4] & ~left &
right);
    assign D[3] = (Q[1] & left & right) | (Q[3] & left & right) | (Q[5] & right &
left);
    assign D[4] = (Q[2] & right & left) | (Q[4] & right & left) | (Q[6] & right &
left);
    assign D[5] = (Q[3] & ~left & right) | (Q[5] & ~left & right);
    assign D[6] = (Q[4] & ~right & left) | (Q[6] & ~right & left);
    assign showclock = ~Q[0];
    assign countup = ~Q[0];
    assign incup = Q[6] & ~left & ~right;
    assign incdown = Q[5] & ~left & ~right;
    assign resettime = Q[0] & (right ^ left);
endmodule
```

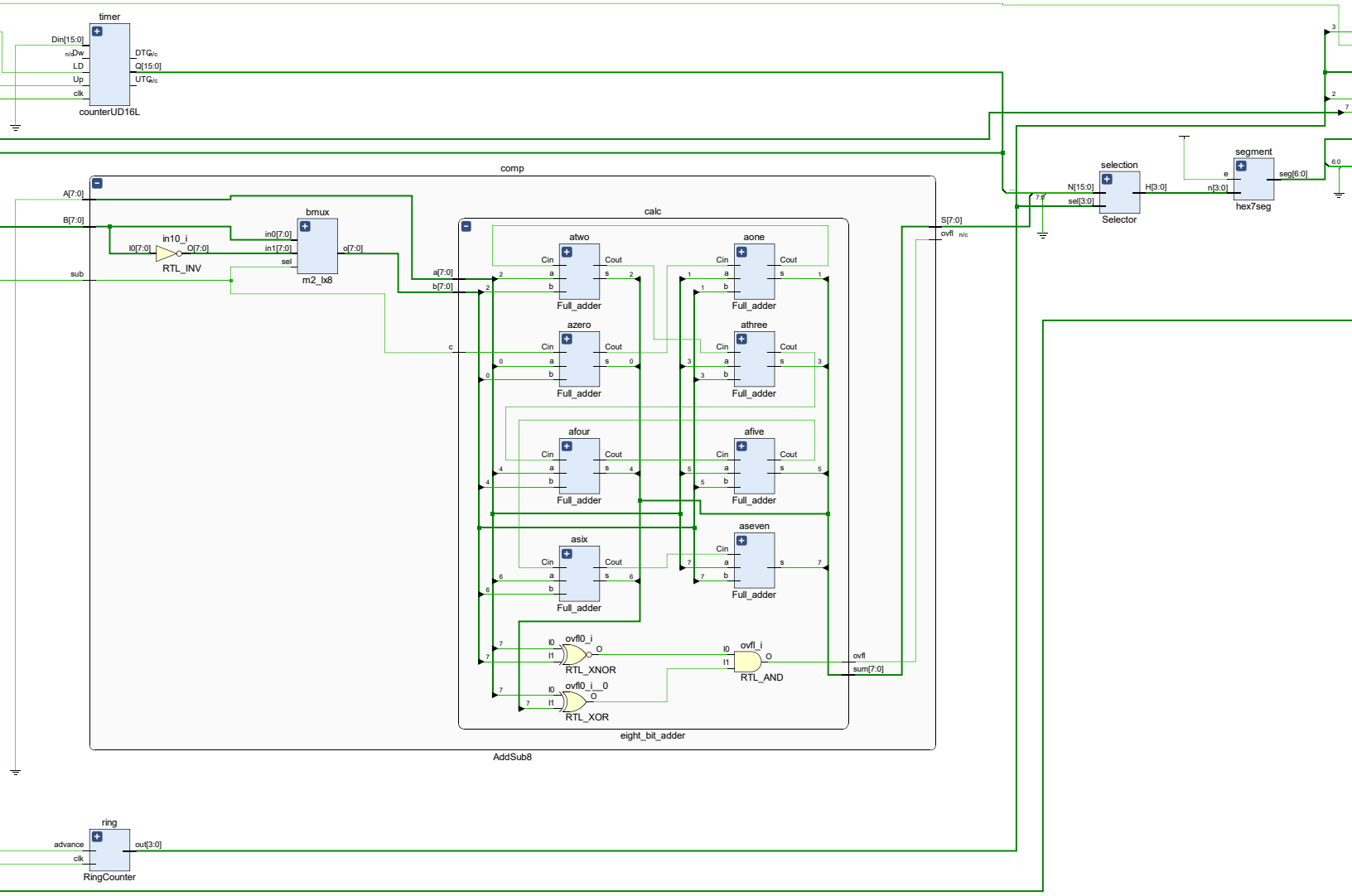
```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 04/23/2020 07:59:34 AM
// Design Name:
// Module Name: Selector
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module Selector(
    input [3:0] sel,
    input [15:0] N,
    output [3:0] H
    );
    wire [3:0] zero, one, two, three;
    assign zero = {4{sel[0]}} & N[3:0];
    assign one = {4{sel[1]}} & N[7:4];
    assign two = {4{sel[2]}} & N[11:8];
    assign three = {4{sel[3]}} & N[15:12];
    assign H = zero | one | two | three;

endmodule
```

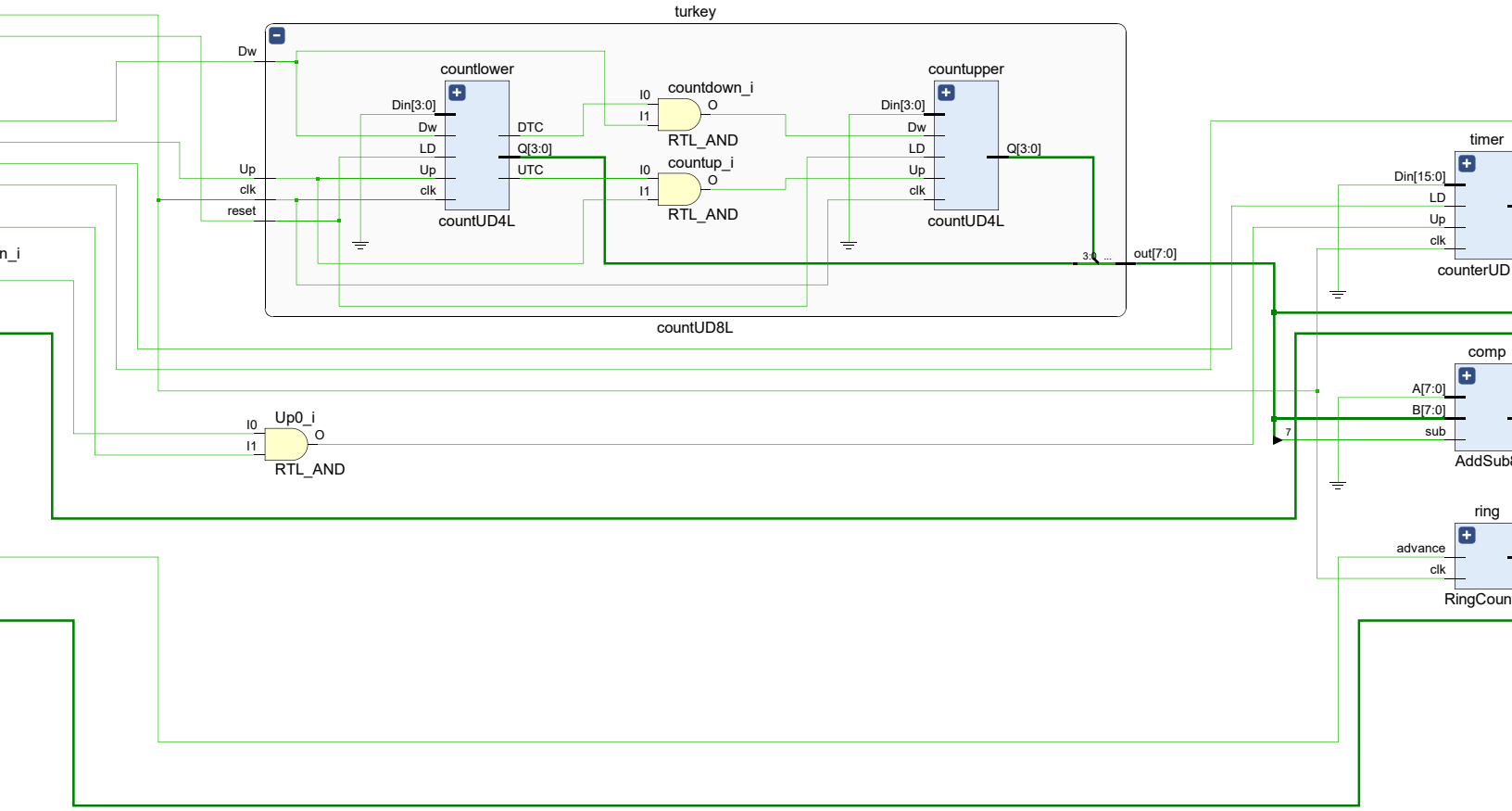
```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 04/23/2020 02:54:10 PM
// Design Name:
// Module Name: counterUD16L
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module counterUD16L(
    input [15:0] Din,
    input Up,
    input LD,
    input Dw,
    input clk,
    output [15:0] Q,
    output UTC,
    output DTC
    );
    wire [3:0] over, under;
    countUD4L first(.Up(Up), .Dw(Dw), .LD(LD), .Din(Din[3:0]), .clk(clk),
.UTC(over[0]), .DTC(under[0]), .Q(Q[3:0]));
    countUD4L second(.Up(Up & over[0]), .Dw(Dw & under[0]), .LD(LD), .Din(Din[7:4]),
.clk(clk), .UTC(over[1]), .DTC(under[1]), .Q(Q[7:4]));
    countUD4L third(.Up(Up & (&over[1:0])), .Dw(Dw & (&under[1:0])), .LD(LD),
.Din(Din[11:8]), .clk(clk), .UTC(over[2]), .DTC(under[2]), .Q(Q[11:8]));
    countUD4L fourth(.Up(Up & (&over[2:0])), .Dw(Dw & (&under[2:0])), .LD(LD),
.Din(Din[15:12]), .clk(clk), .UTC(over[3]), .DTC(under[3]), .Q(Q[15:12]));
    assign UTC = &over;
    assign DTC = &under;
endmodule
```

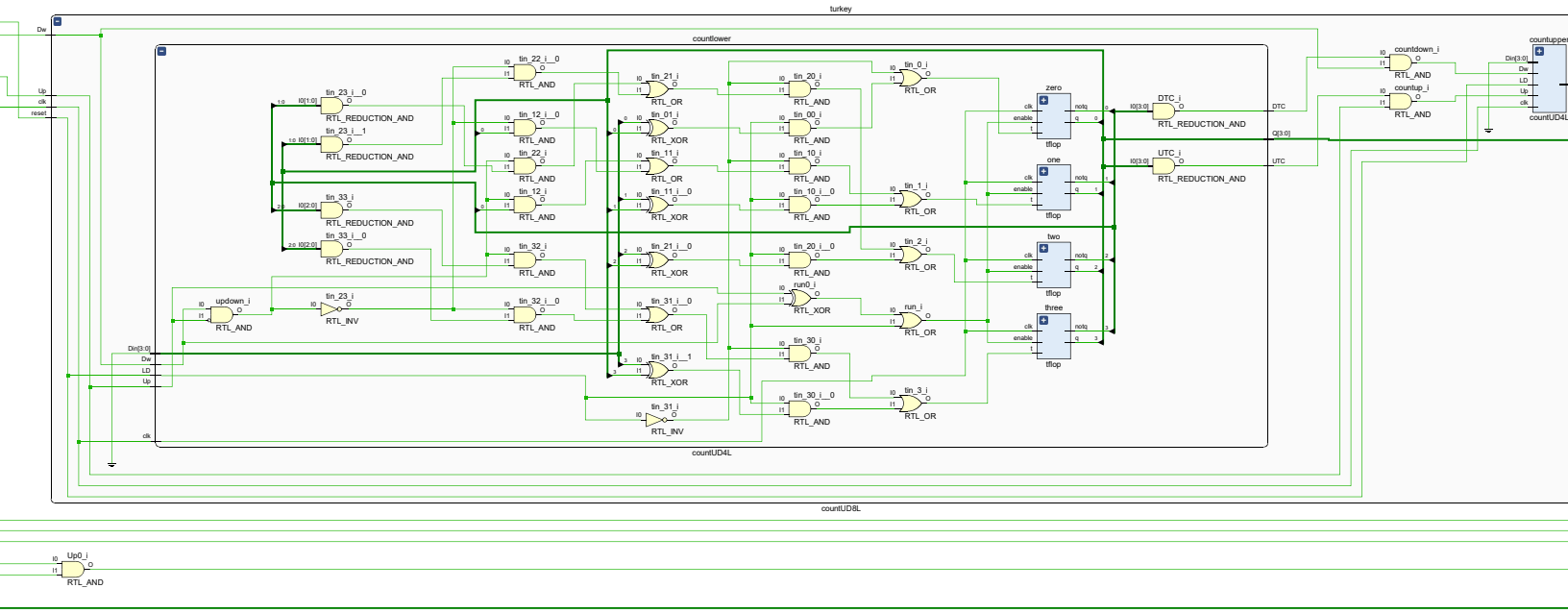
```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 04/23/2020 11:06:07 AM
// Design Name:
// Module Name: tflop
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module tflop(
    input t,
    input clk,
    input enable,
    output q,
    output notq
    );
    wire out, nout, din;
    assign din = (t & nout) | (~t & out);
    FDRE #(.INIT(1'b0)) Q5_FF (.C(clk), .CE(enable), .D(din), .Q(out));
    assign nout = ~out;
    assign q = out;
    assign notq = nout;

endmodule
```


comp
bmux
calc
i
O[7:0]
INV
in0[7:0]
in1[7:0]
sel
S=1'b1
I0[7:0]
S=default
I1[7:0]
o_i
O[7:0]
S
RTL_MUX
o[7:0]
m2_lx8
a[7:0]
b[7:0]
c
ovfl
sum[
eight_bit_adder
AddSub8


comp
bmux
in0[7:0]
in1[7:0]
sel
o[7:0]
m2_lx8
calc
a[7:0]
b[7:0]
c
ovfl
sum[7:0]
eight_bit_adder
S[7:0]
ovfl
n/c
AddSub8
selection
N[15:0]
sel[3:0]
H[3:0]
Selector
segment
e
n[3:0]
g_in0_i
I0
O
RTL_INV
ma
mb
mc
md
me
mf
mg
in[7:0]
sel[2:0]
o
m8_1e
seg[6:0]
hex7seg
an1_i
I0
I1
O
RTL_AND
sel0_i
RTL_AND
in10_i
I0[7:0]
O[7:0]
RTL_INV
an0_i__0
an0_i__1
an0_i__2
an0_i
RTL_INV
displayneg
in0[7:0]
in1[7:0]
sel
o[7:0]
m2_lx8
an[3:0]
seg[6:0]
led[15:0]_OBUF_inst
I
O
OBUF
led[15:0]
dp_OBUF_inst
OBUF
dp

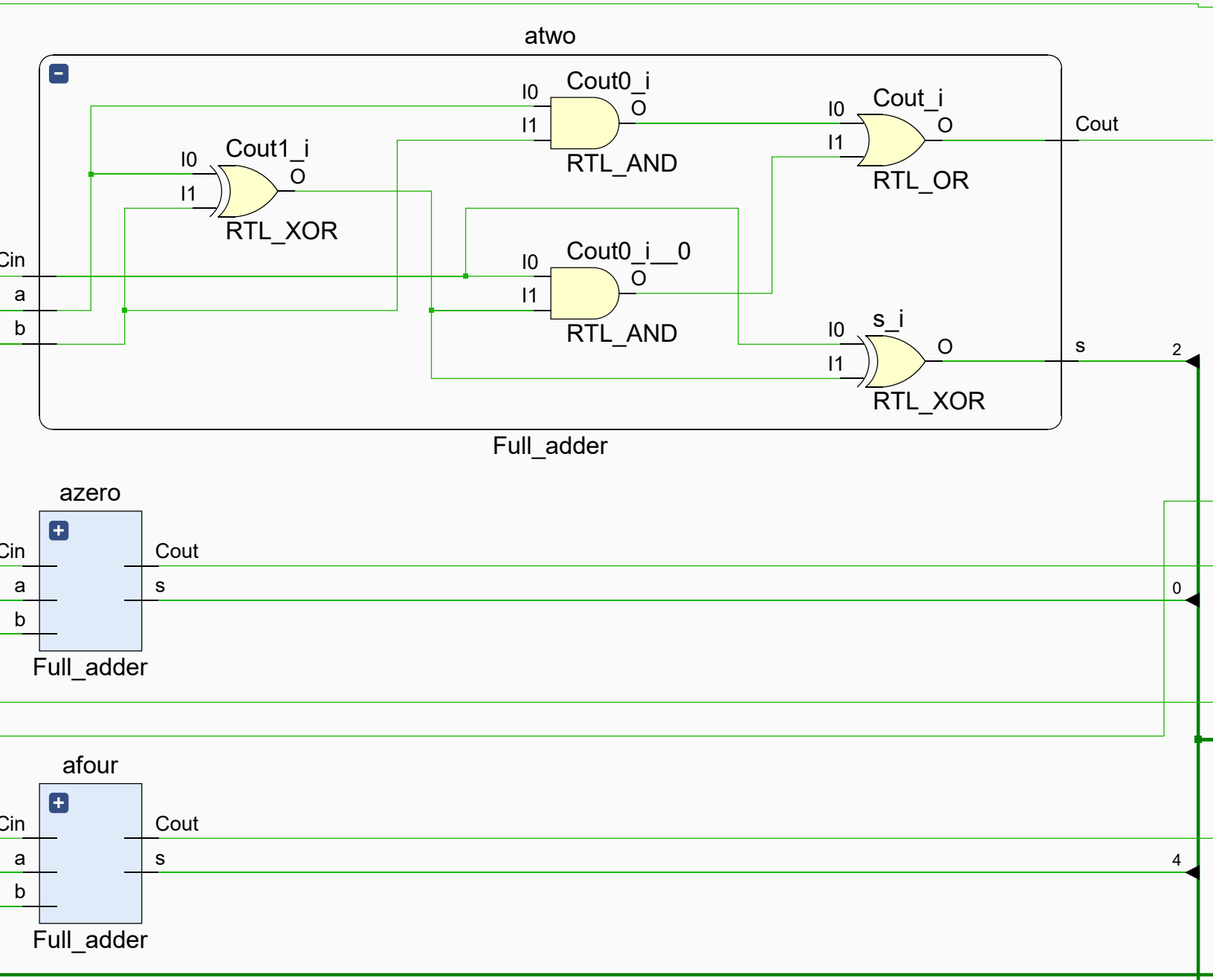

timer
Din[15:0]
n/cDw
LD
Up
clk
DTG/c
Q[15:0]
UTG/c
counterUD16L
comp
A[7:0]
B[7:0]
sub
in10_i
I0[7:0]
O[7:0]
RTL_INV
bmux
in0[7:0]
in1[7:0]
sel
o[7:0]
m2_lx8
calc
a[7:0]
b[7:0]
c
atwo
aone
azero
athree
afour
afive
asix
aseven
Cin
Cout
a
b
s
Full_adder
ovfl0_i
RTL_XNOR
ovfl_i
RTL_AND
ovfl0_i__0
RTL_XOR
eight_bit_adder
S[7:0]
ovfl
n/c
sum[7:0]
AddSub8
selection
N[15:0]
sel[3:0]
H[3:0]
Selector
segment
e
n[3:0]
seg[6:0]
hex7seg
ring
advance
clk
out[3:0]
RingCounter

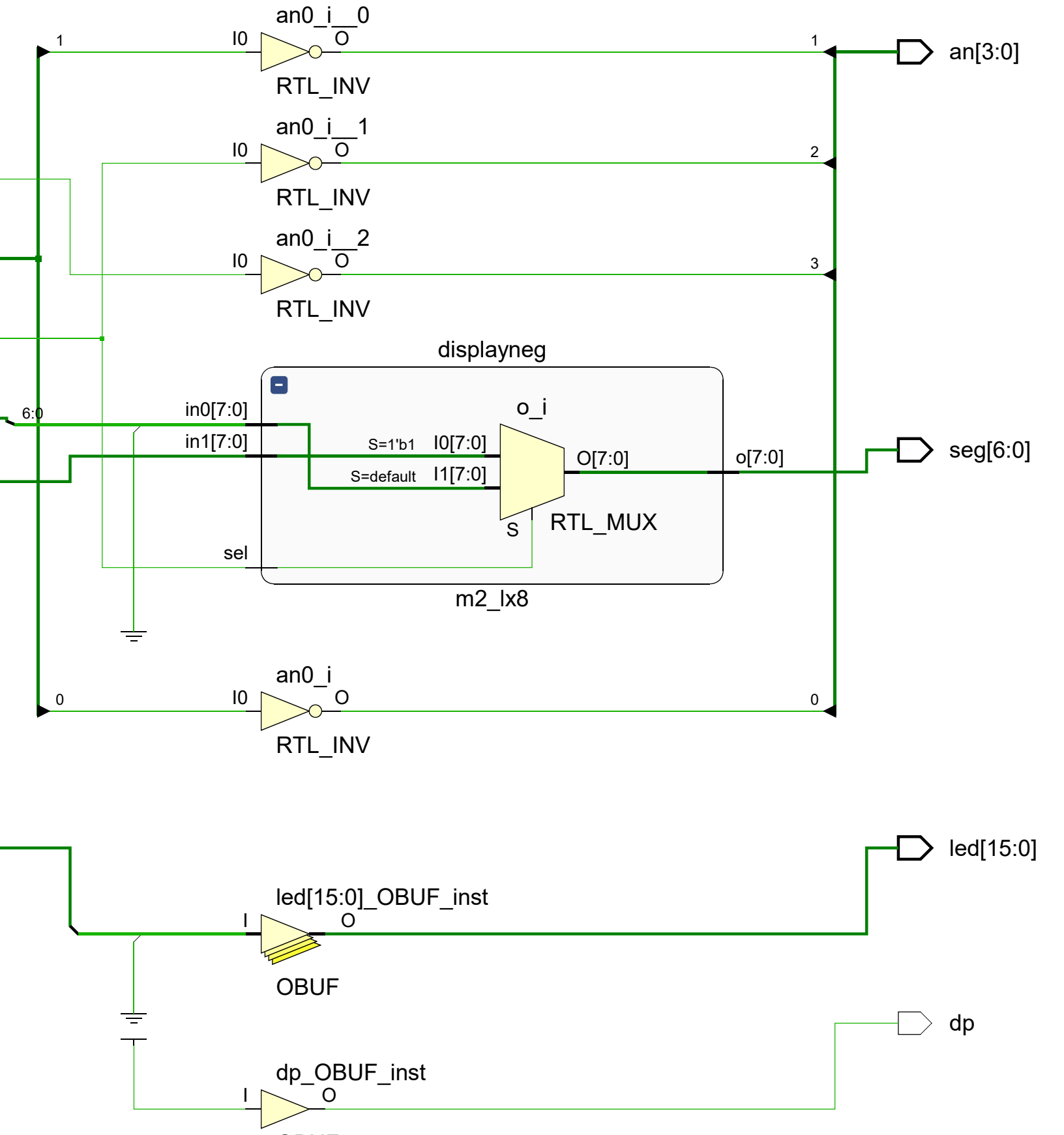

turkey
Dw
countlower
Din[3:0]
Dw
LD
Up
clk
DTC
Q[3:0]
UTC
countUD4L
countdown_i
I0
I1
O
RTL_AND
countup_i
countupper
Q[3:0]
Up
clk
reset
out[7:0]
countUD8L
timer
Din[15:0]
LD
Up
clk
counterUD
comp
A[7:0]
B[7:0]
sub
AddSub
ring
advance
clk
RingCoun
Up0_i

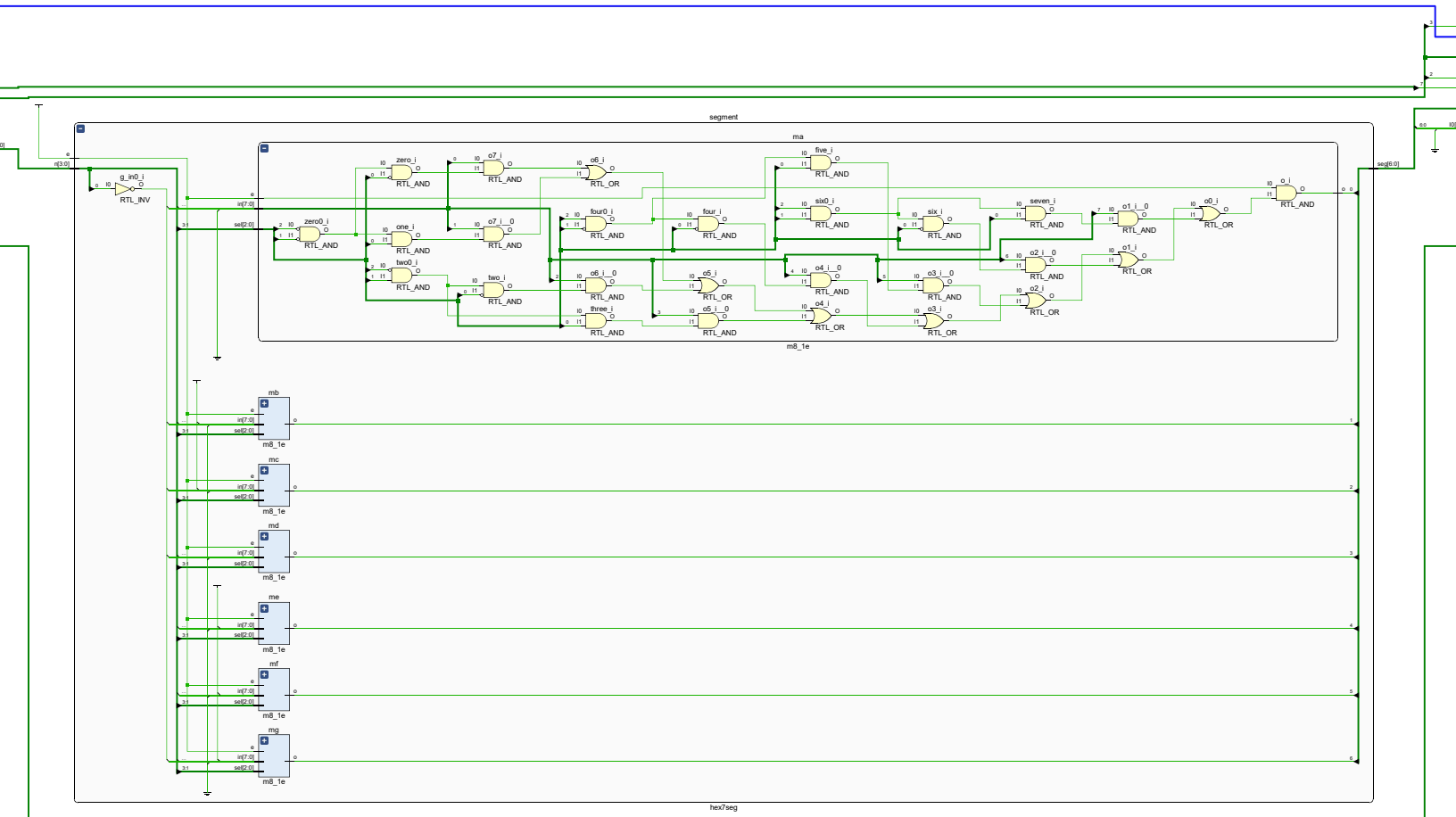

turkey
countlower
countUD4L
countUD8L
Dw
Up
clk
reset
Din[3:0]
LD
countupper
countdown_i
countup_i
Up0_i
RTL_AND
RTL_OR
RTL_XOR
RTL_INV
RTL_REDUCTION_AND
tflop
zero
one
two
three
DTC_i
UTC_i
DTC
UTC
Q[3:0]

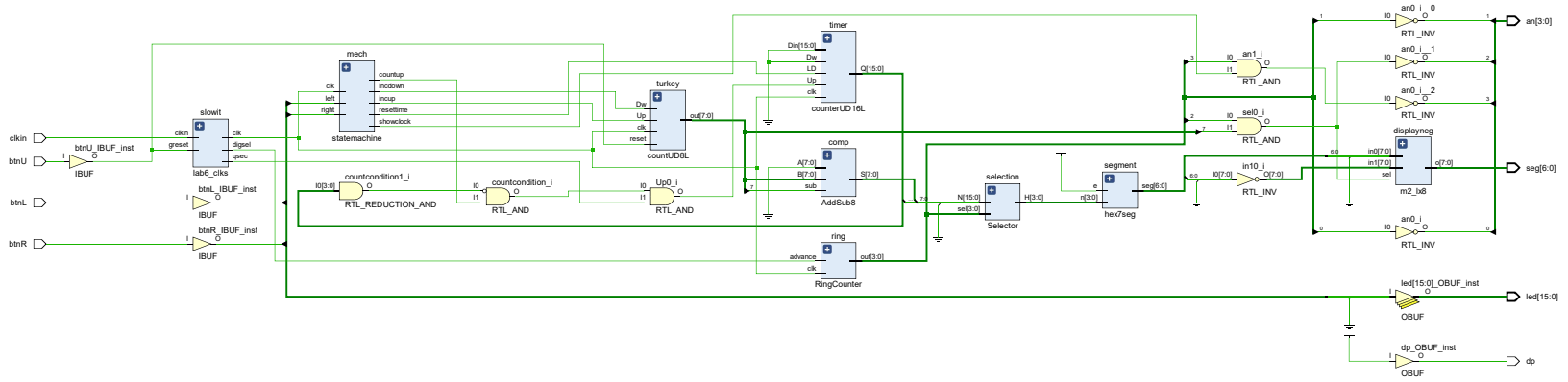
comp

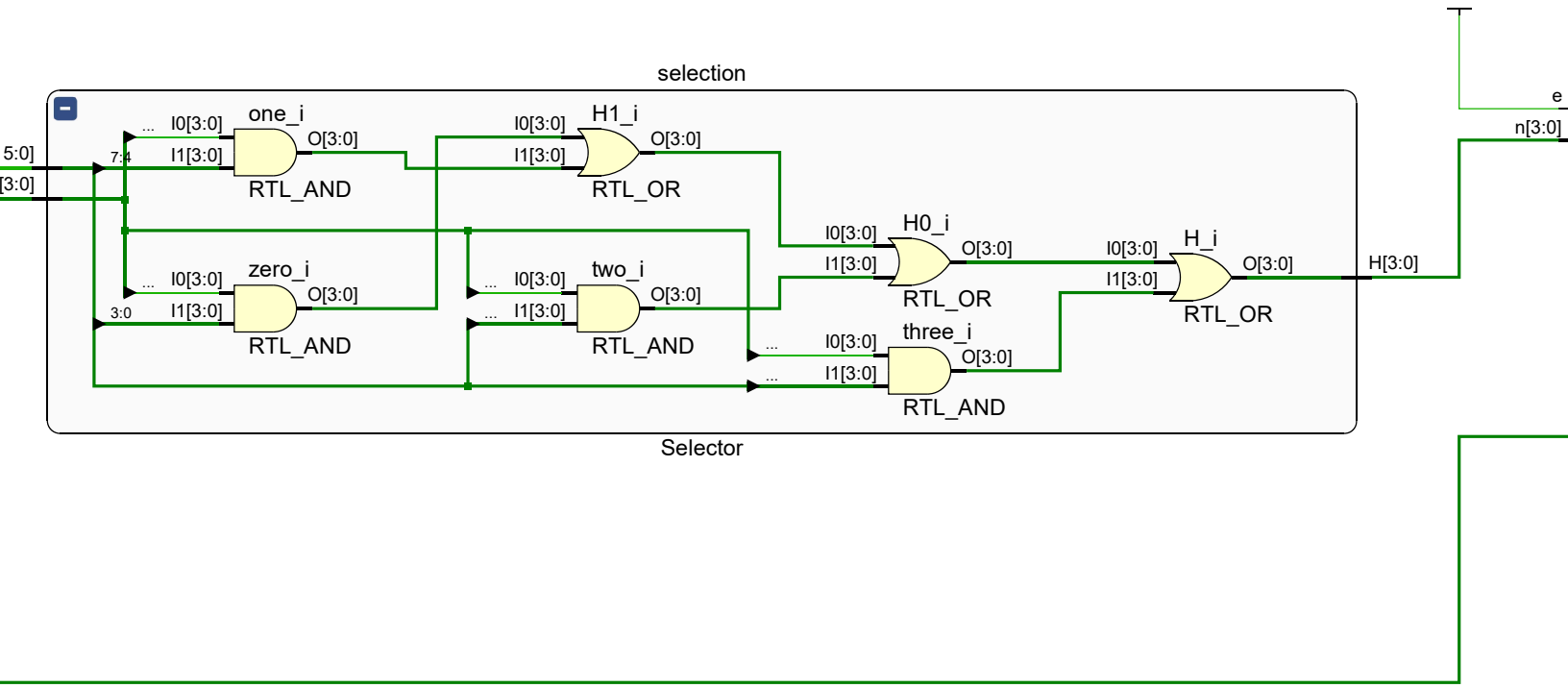
calc

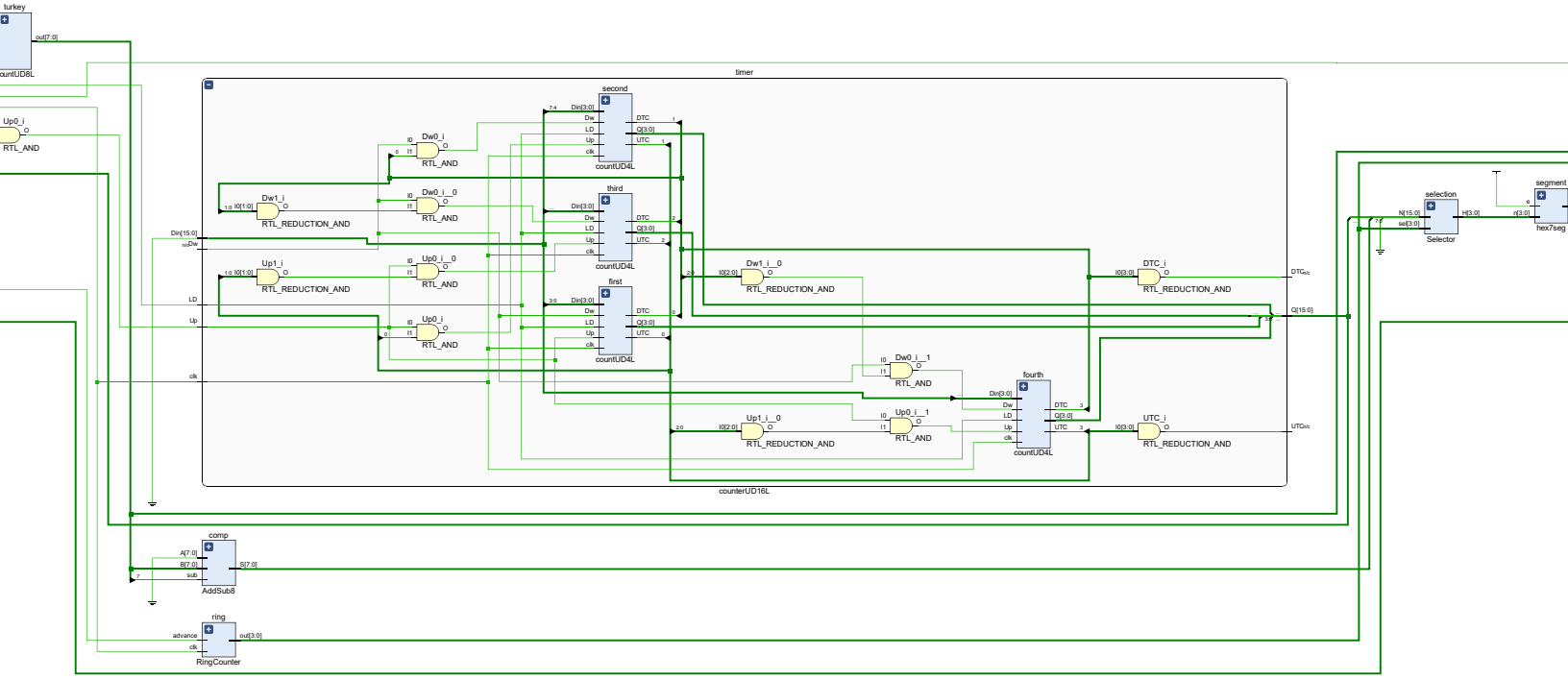

atwo
Cout0_i
I0
I1
O
RTL_AND
Cout_i
RTL_OR
Cout
Cout1_i
RTL_XOR
Cout0_i__0
s_i
s
2
Cin
a
b
Full_adder
azero
0
afour
4

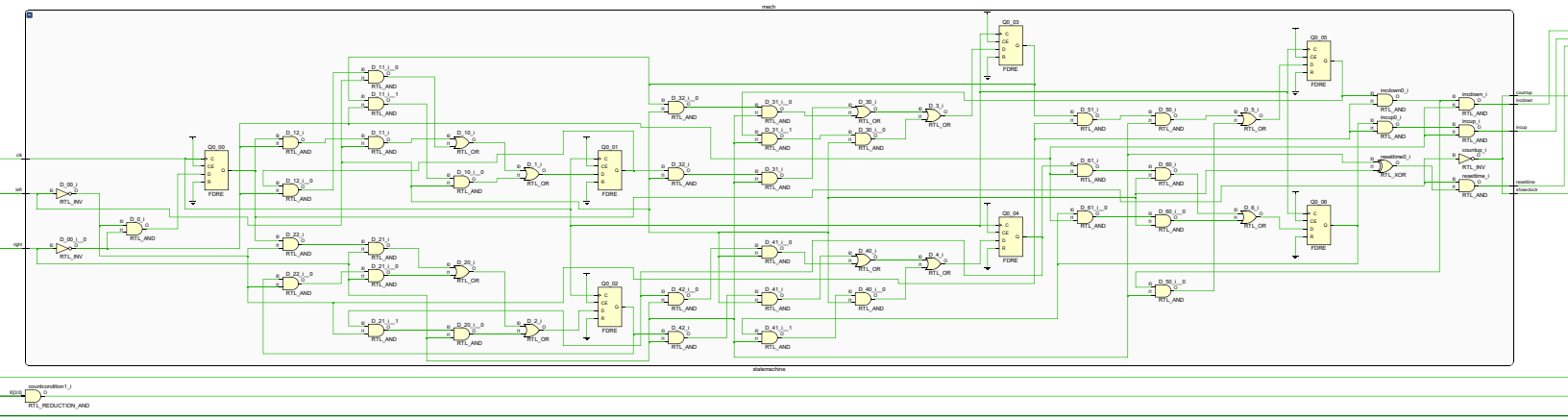

an0_i__0
I0
O
RTL_INV
1
1
an[3:0]
an0_i__1
I0
O
RTL_INV
2
an0_i__2
I0
O
RTL_INV
3
displayneg
6:0
in0[7:0]
in1[7:0]
o_i
S=1'b1
I0[7:0]
S=default
I1[7:0]
O[7:0]
o[7:0]
seg[6:0]
S
RTL_MUX
sel
m2_lx8
an0_i
0
I0
O
RTL_INV
0
led[15:0]
led[15:0]_OBUF_inst
I
O
OBUF
dp
dp_OBUF_inst
I
O


segment
ma
m8_1e
hex7seg
RTL_INV
RTL_AND
RTL_OR
mb
mc
md
me
mf
mg

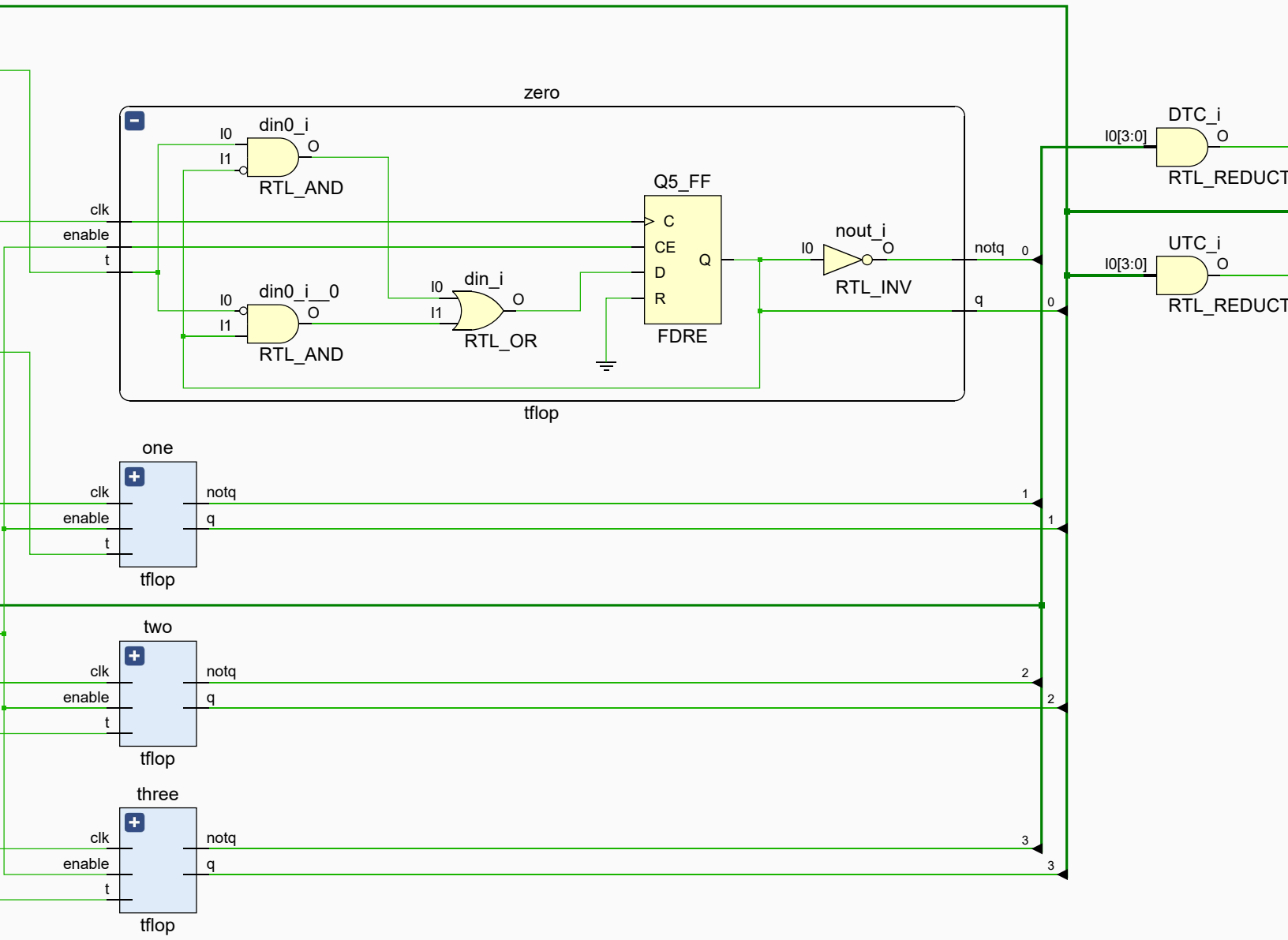

clkin
btnU
btnL
btnR
btnU_IBUF_inst
IBUF
btnL_IBUF_inst
IBUF
btnR_IBUF_inst
IBUF
slowit
clkin
greset
clk
digsel
qsec
lab6_clks
mech
clk
left
right
countup
incdown
incup
resettime
showclock
statemachine
countcondition1_i
I0[3:0]
O
RTL_REDUCTION_AND
countcondition_i
I0
I1
O
RTL_AND
turkey
Dw
Up
clk
reset
out[7:0]
countUD8L
Up0_i
I0
I1
O
RTL_AND
timer
Din[15:0]
Dw
LD
Up
clk
Q[15:0]
counterUD16L
comp
A[7:0]
B[7:0]
sub
S[7:0]
AddSub8
ring
advance
clk
out[3:0]
RingCounter
selection
N[15:0]
se[3:0]
H[3:0]
Selector
segment
n[3:0]
seg[6:0]
hex7seg
an1_i
I0
I1
O
RTL_AND
sel0_i
I0
I1
O
RTL_AND
in10_i
I0[7:0]
O[7:0]
RTL_INV
an0_i_0
an0_i_1
an0_i_2
an0_i
I0
O
RTL_INV
displayneg
in0[7:0]
in1[7:0]
sel
o[7:0]
m2_lx8
an[3:0]
seg[6:0]
led[15:0]_OBUF_inst
OBUF
led[15:0]
dp_OBUF_inst
OBUF
dp


selection
5:0]
[3:0]
7:4
3:0
...
I0[3:0]
I1[3:0]
one_i
O[3:0]
RTL_AND
zero_i
RTL_AND
H1_i
RTL_OR
two_i
RTL_AND
H0_i
RTL_OR
three_i
RTL_AND
H_i
RTL_OR
H[3:0]
e
n[3:0]
Selector


timer
second
third
first
fourth
countUD4L
Dw0_i
Dw1_i
Up0_i
Up1_i
RTL_AND
RTL_REDUCTION_AND
Dw0_i_0
Up0_i_0
Dw1_i_0
Dw0_i_1
Up0_i_1
Up1_i_0
DTC_i
UTC_i
counterUD16L
turkey
countUD8L
Up0_i
RTL_AND
Din[15:0]
LD
Up
clk
Q[15:0]
selection
Selector
segment
hex7seg
comp
AddSub8
ring
RingCounter


mech
statemachine
clk
left
right
countcondition1_i
RTL_REDUCTION_AND

turkey


zero
din0_i
I0
I1
O
RTL_AND
Q5_FF
C
CE
D
R
Q
FDRE
nout_i
RTL_INV
din0_i__0
din_i
RTL_OR
clk
enable
t
notq
q
tflop
DTC_i
I0[3:0]
RTL_REDUCT
UTC_i
one
two
three

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 05/12/2020 12:20:57 PM
// Design Name:
// Module Name: statesim
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module statesim();
reg left, right, clk;
wire incup, incdown, resettime, countup, showclock;

statemachine mech(.left(left), .right(right), .clk(clk), .incup(incup),
.incdown(incdown), .resettime(resettime), .countup(countup), .showclock(showclock));
 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
    #1000;
    left = 1'b0;
    right = 1'b0;
    #100;
```

```
    left = 1'b1;
    #100;
    right = 1'b1;
    #100;
    right = 1'b0;
    #100;
    right = 1'b1;
    #100;
    left = 1'b0;
    #100;
    left = 1'b1;
    #100;
    right = 1'b0;
    #100;
    left = 1'b0;
    #100;

    end
endmodule
```

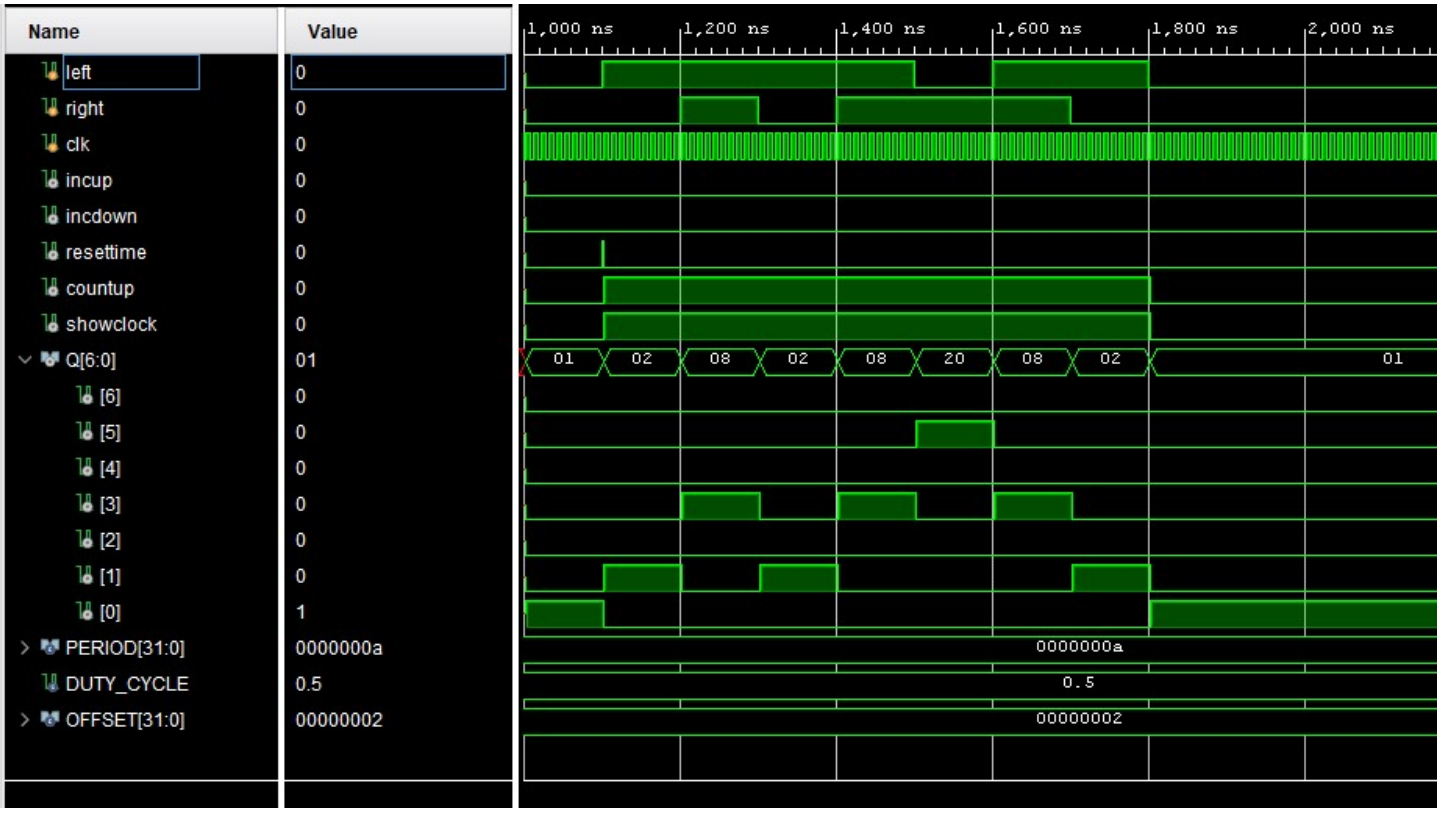
```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 05/07/2020 08:21:26 AM
// Design Name:
// Module Name: top_sim
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module top_sim();
    reg clkin, btnR, btnU, btnL;
    wire [6:0] seg;
    wire dp;
    wire [3:0] an;
    wire [15:0] led;
    parameter PERIOD = 10;
    parameter real DUTY_CYCLE = 0.5;
    parameter OFFSET = 2;
    wire [7:0] D7Seg3,D7Seg2,D7Seg1,D7Seg0; // Change the Radix of these signals to
ASCII
    show_7segDisplay  showit (.seg(seg),.dp(dp),.an(an),
     .D7Seg0(D7Seg0),.D7Seg1(D7Seg1),.D7Seg2(D7Seg2),.D7Seg3(D7Seg3));
     Topmodule turkeytracker(.clkin(clkin), .btnR(btnR), .btnU(btnU), .btnL(btnL),
.seg(seg), .dp(dp), .an(an), .led(led));
    initial    // Clock process for clkin
    begin
        #OFFSET
          clkin = 1'b1;
        forever
        begin
            #(PERIOD-(PERIOD*DUTY_CYCLE)) clkin = ~clkin;
        end
    end
```

```
initial
begin
btnU = 1'b0;
btnL = 1'b0;
btnR = 1'b0;
#1000;
btnL = 1'b1;     //start of left to right
#100;
btnR = 1'b1;
#100;
btnL = 1'b0;
#100;
 btnR = 1'b0;    //end of left to right
#100;
btnR = 1'b1;     //start of right to left
#100;
btnL = 1'b1;
#100;
btnR = 1'b0;
#100;
btnL = 1'b0;     //end of left to right
#100;
btnR = 1'b1;     //starts at right
#100;
btnL = 1'b1;     //is between
#100;
btnR = 1'b0;     //is near end of left
#100;
btnR = 1'b1;     //comes back to between
#100;
btnL = 1'b0;     //is at right edge of sensor
#100;
btnR = 1'b0;     //leaves from right side
#100;
btnR = 1'b1;     //starts off in the right side
#100;
btnL = 1'b1;     //goes in between
#100;
btnR = 1'b0;     //goes to edge of left
#100;
btnR = 1'b1;     //comes back to the inbetween area
#100;
btnL = 1'b0;     //goes into the edge of right
#100;
btnL = 1'b1;     //comes back to middle
```

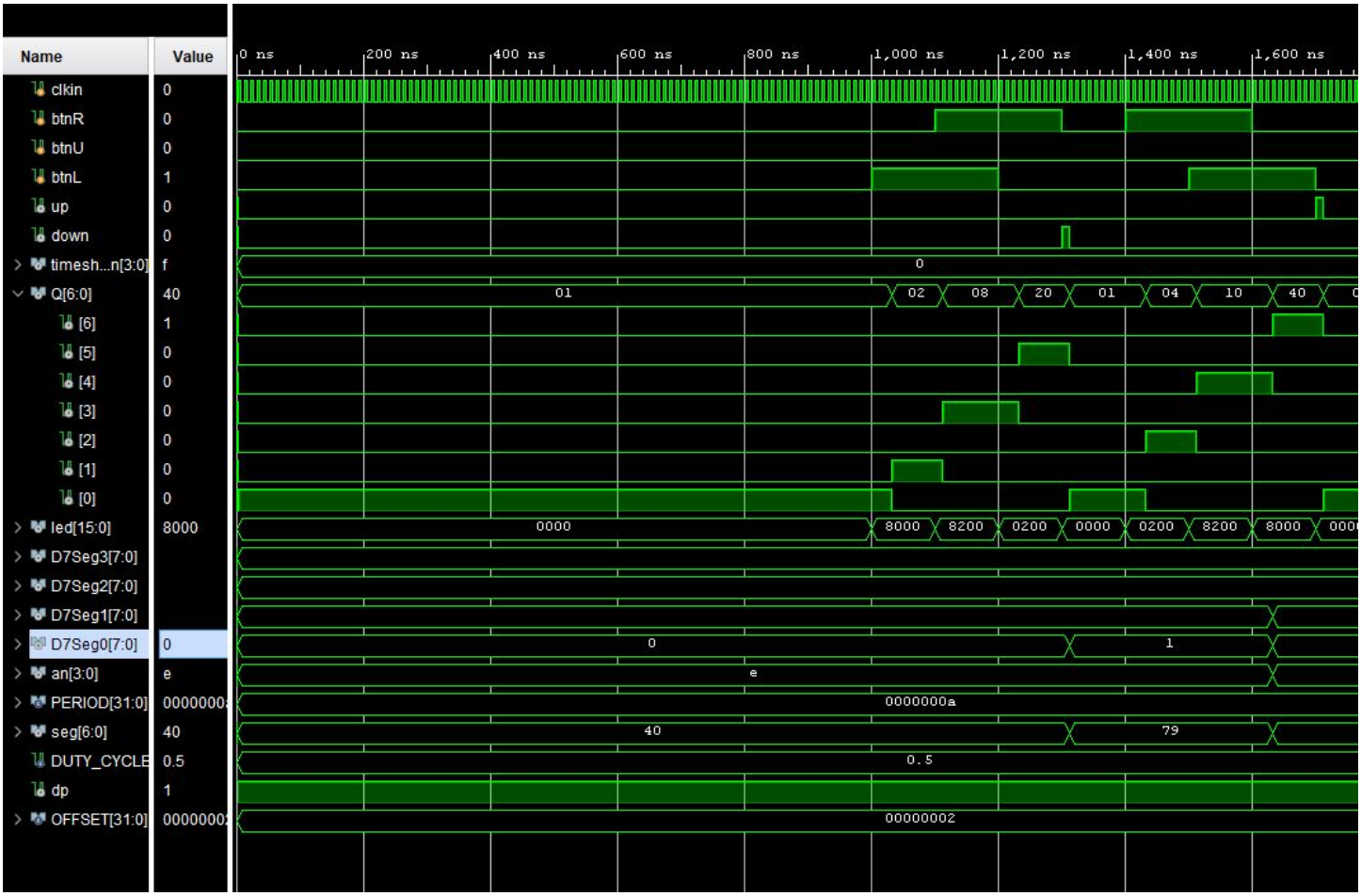
```
   #100;
   btnR = 1'b0;     //goes to edge of left
   #100;
   btnL = 1'b0;     //enters/exits from right to left
   #100;
   btnL = 1'b1;     //starts at the left and will wander in between before leaving le
   #100;
   btnR = 1'b1;     //is in between
   #100;
   btnL = 1'b0;     //is at the edge of right
   #100;
   btnL = 1'b1;     //is back in between
   #100;
   btnR = 1'b0;     //is not back at the edge of left
   #100;
   btnL = 1'b0;     //left from left
   #100;
   btnL = 1'b1;     //enters from left goes back and forth and ends up leaving right
   #100;
   btnR = 1'b1;     //is at the in between area
   #100;
   btnL = 1'b0;     //is at the edge of right
   #100;
   btnL = 1'b1;     //is back in the in between area
   #100;
   btnR = 1'b0;     //is at the edge of left
   #100;
   btnR = 1'b1;     //is back in the in between area
   #100;
   btnL = 1'b0;     //is at the edge of right
   #100;
   btnR = 1'b0;     //leaves from left to right
   #100;
   btnR = 1'b1;     //starts off at right and walks around until time ends and then
leaves
   #100;
   btnL = 1'b1;     //is in the inbetween area
   #100;
   btnR = 1'b0;     //is on the edge of left
   #100;
   btnR = 1'b1;     //is in the inbetween area again
   #100;
   btnR = 1'b0;     //is on the edge of left again
   #1000000000;
   //btnL = 1'b0;     //finally leaves
    end
```

```
endmodule
```

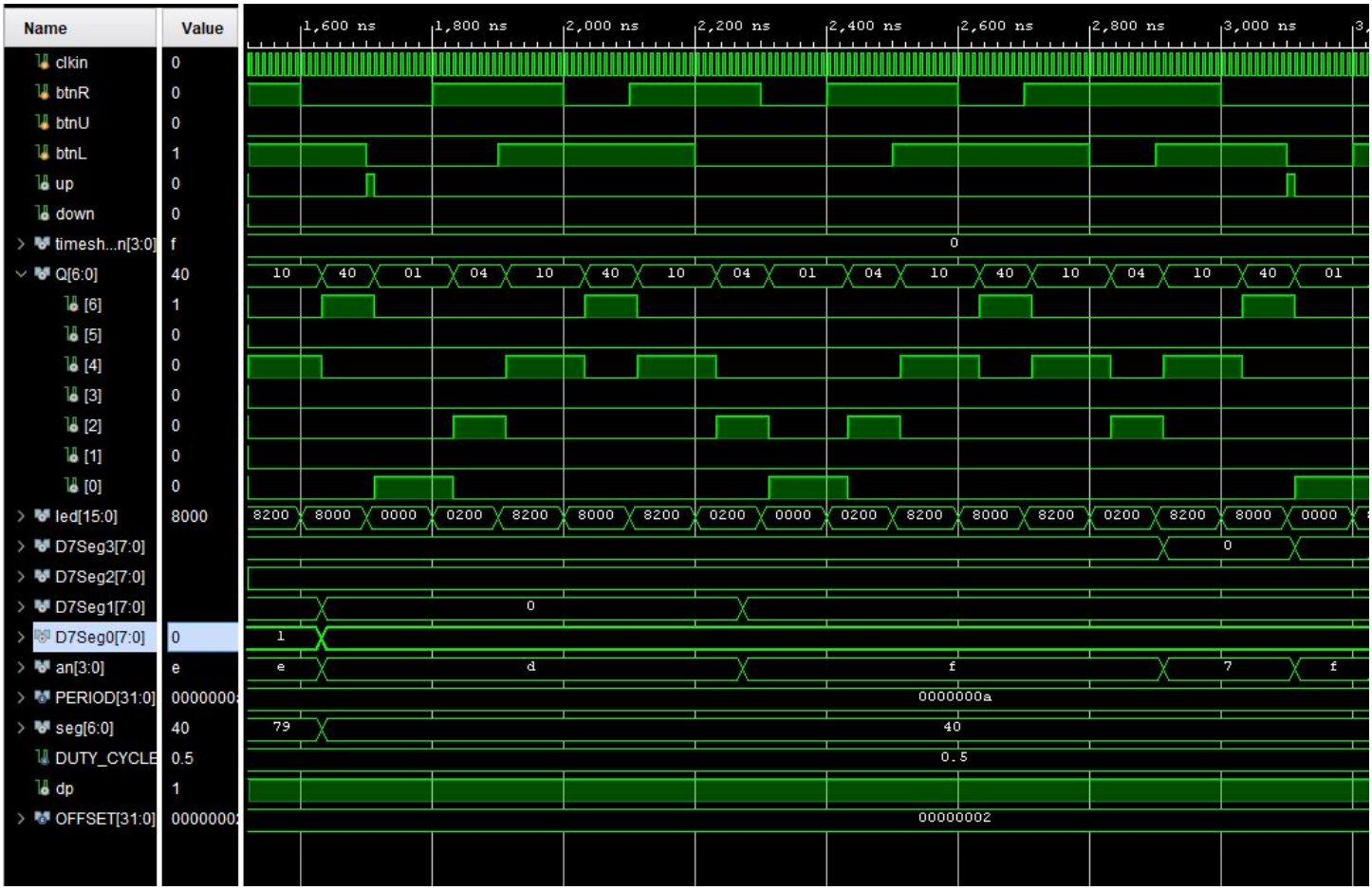
State machine simulation results:



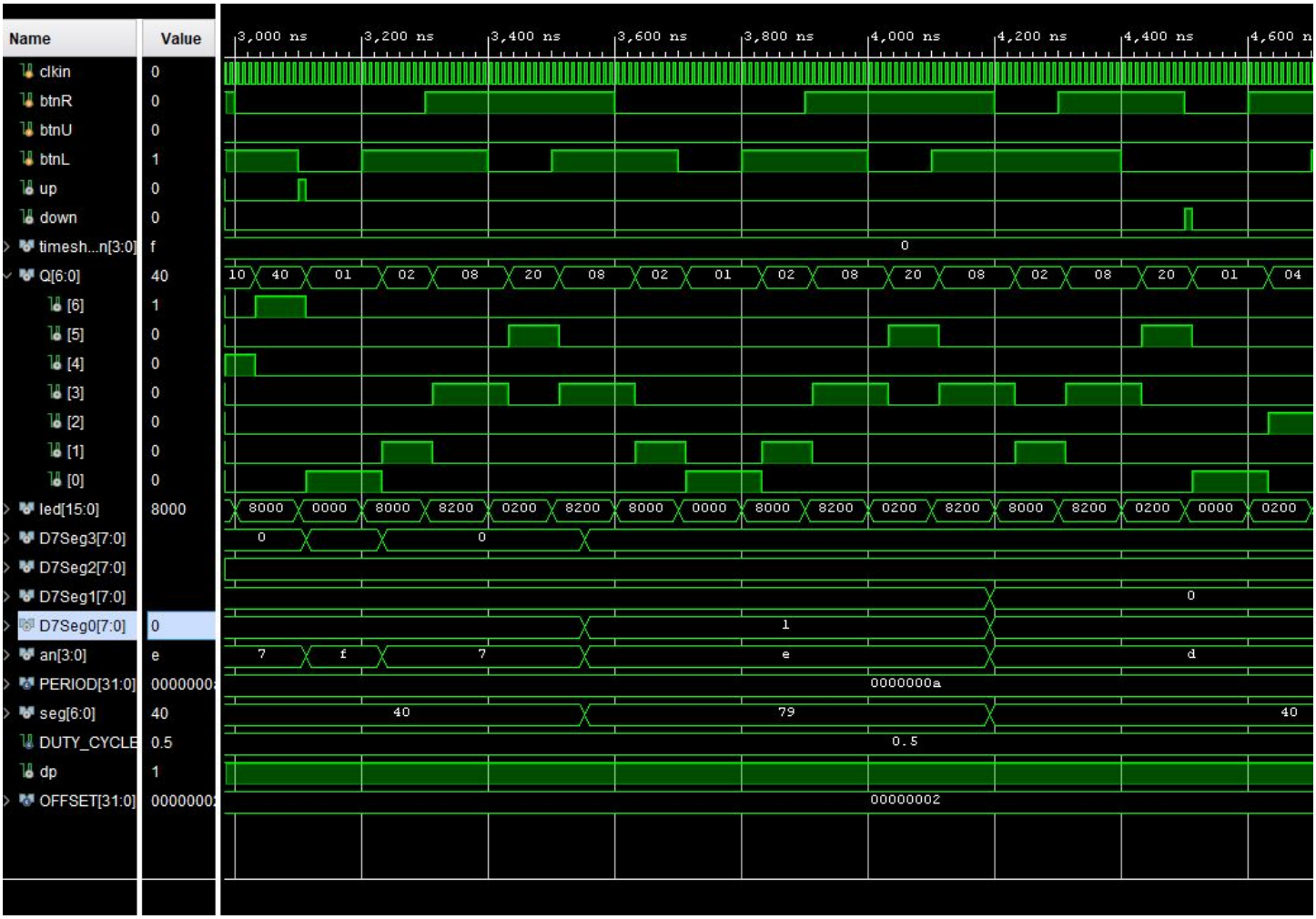
A) Turkey crossing left to right directly.
B) Turkey crossing right to left directly.
results



C)Turkey entering from right and wandering back and forth before retreating to the right.
D) Turkey entering from right and wandering back and forth before crossing to the left.
results



E) Turkey entering from left and wandering back and forth before retreating to the left.
F) Turkey entering from left and wandering back and forth before retreating to the right.
results:



G) Turkey entering and wandering back and forth until timer reaches F.
resut: