David Nguyen

Lab 6

Section C

11/23/2018

Lab 6 Writeup

Purpose:

The purpose of this lab was to learn how to build a state machine to count the number of objects crossing 2 sensors which are simulated using push buttons. This counter counts when a turkey finishes moving from left to right or right to left. If a turkey also stays in one position for more than 4 seconds a time counter will flash with any changes resetting the time counter to 0.

Methods:

Turkey Counter

1. This is the same 8bit counter used in Lab 5 using setting the z output to UTC, or when all the bits of the counter are zero.
2. This counter takes in IncU and IncD as the increments for up and down and outputs the resulting counter into turCountOut.

TimeCounter

1. The TimeCounter consists of 2 4 bit counters. The first counter continuously counts up turkSec using qsec.
2. A wire ‘sec’ is then set to when this counter at turkSec[1 and 0] are on, and qsec is high. This wire is high when 1 sec has passed.
3. The second counter is a counter which keeps track of the time in which the turkey has not switched positions. This counter has an extra output four which is when the counter value equals 4.
4. The inputs to this counter are sec, showtime, and ~four with the Load variable set to cntRes and Din as all zeros.

Sensor Inputs

1. The sensor inputs are high when there is nothing crossing the sensor so btnL and btnR are inverted.
2. This is done by inputting the inverted button presses into a flipflop and using the output for the state machine.
3. Led[15] is also set to the btnLsyn and led[8] is set to btnRsyn.

State Machine

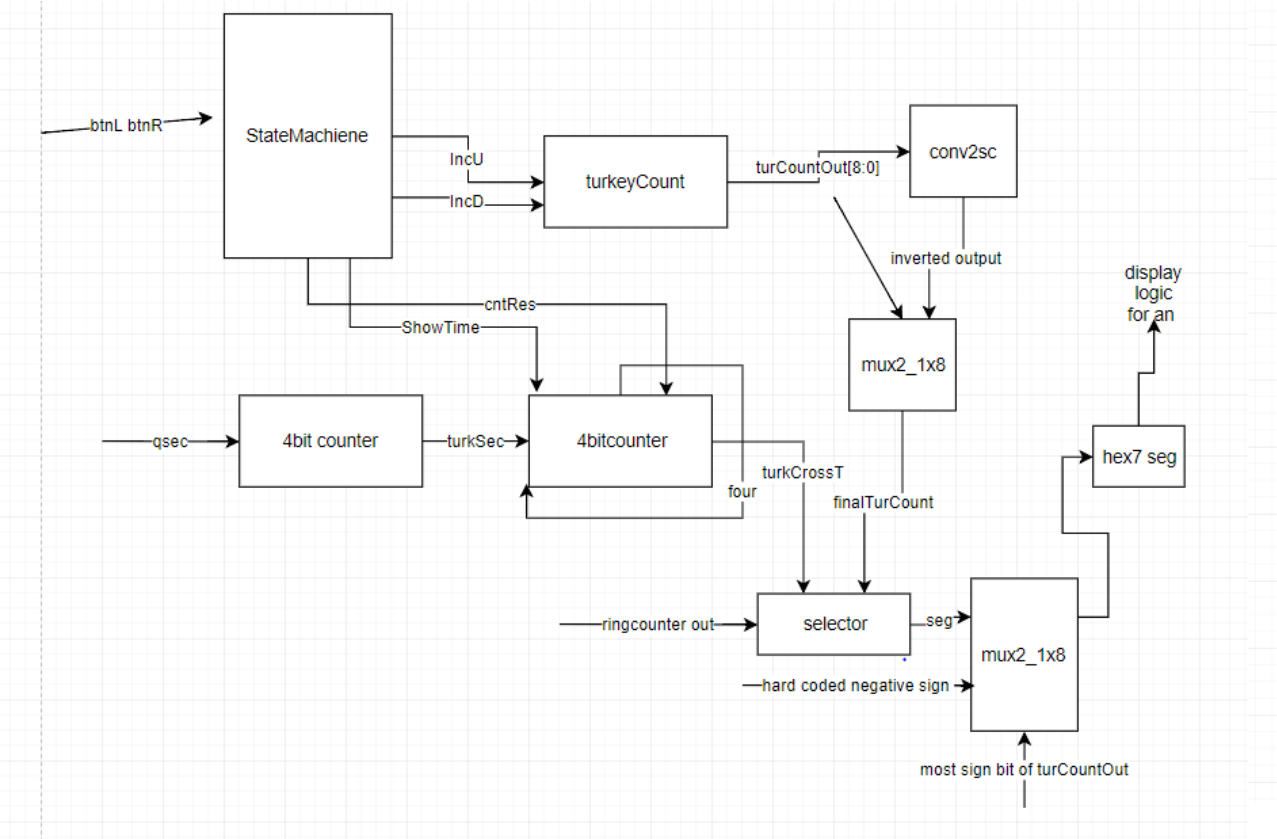
1. The first step of the State Machine is to draw out a diagram with various states which covers all cases considering the inputs btnL, btnR as well as the outputs ShowTime, IncU, IncD, and cntRes.
2. The next step is to create a state table, which is easy through one-hot encoding. The state diagram will have each state in the y -axis, and each input as the x-axis with each input being on or off. There is also a state table for outputs with the x-axis as your outputs.
3. This state table will then provide output logic equations for each of the states and outputs. Each equation is then assigned to the input of a flipflop, one for each state, with the flipflop connected to the IDLE state initialed to 1. The equations for the outputs are then also assigned based off the state table and current states.
4. This state machine has 2 branches. One for left first or right first, this can then go to the next state where both buttons are pressed or if the original button was released it will go back to idle. From the both button pressed state. It will go to the state of just the opposite of the original or back to the original button press. From this final state, it will either go to both buttons are released which is a full crossing or back to both buttons are pressed.

TopMod6

1. The first step of the TopMod is to add the Verilog file lab6_clks.v and the following line should be pasted into TopMod:
   lab6_clks slowit (.clkin(clkin), .greset(btnD), .clk(clk), .digsel(digsel), .qsec(qsec));
   clk is now the system clock, digsel is used to advance the rngCounter,and qsec will be a signal that is high for 1 clock cycle every ¼ of a second
2. The next step is to create 3 flipflops to synchronize btnL, btnD, and btnR to the clk, This is done by having the enable always be on and putting in the each btn as the input to each flipflop. You will now use the output of each flipflop for each button press which is now synchronized with the system clk. You will also need to following instructions for the sensor inputs for btnL and btnR.
5. The next step is to create 1 module of the state machine which takes in the synchronous inputs btnR and btnL and outputs ShowTime, IncU, IncD, and cntRes.
3. These outputs go into the turkeycounter which output this result into turCountOut.
4. turCountOut is then inputted into a 2's complement convertor which inverts the counter result and adds 1.
5. There is a 2_1x8 mux to select between the normal or inverted output with the selector as the most significant bit in turCountOut. The final output is then put into pick[7:0].
6. There are also the 2 counters as mentioned in Timer counter whose 4 bit output is then put into pick[15:12]
7. The next step is to make display logic for the negative sign which is done by using the mux 2_1x8 module which has inputs the original seg and a hardcoded negative sign with the selector as the most significant bit of turCountOut and select[2] .

8. The final step is to have display logic for an[3] which is set to ~((select[3]& ShowTime & ~four) | (select[3]&ShowTime & four & turkSec[1])), which will count up t0 4 and start flashing.

Results



The topMod of this lab starts with the inputs btnR and btnL which are then inputed into the state machiene. This outputs IncU and IncD which changes turkey counter. This outputs to a 2's complement convertor which is chosen by a 2_1 mux. This is finally sent to selector. The other part of this diagram starts with the 2 4 bit counters for timerCounter which takes in ShowTime and cntRes. This is then sent to selector whose output is put through another 2_1 mux. Finally, the output of the mux is sent through hex7seg.

To test and simulate the State Machine and Top Module, I tried to make the state machine reach every state by changing btnL, btnR. I then checked to see if the machine was in the correct state if the output. This helped me debug certain typos such as entering the wrong input for certain states. I also tried various cases of the state such as a turkey flying in and covering both sensors, or a turkey flying out from both sensors, I also tried to make the state machine reach every state by changing btnR, and btnR. I then checked each input and output of each state machine to make sure they were correct and went to the right input. I also tested to see if the number counter was incrementing and decrementing the turkey counter properly.

Each of the components function and implementation is described in the methods section with some of the miscellaneous components in the results section

Picture of State Diagram: Picture 1 of Supplementary Material

Logic Equations: Picture 2 of Supplementary Material

Next State and Output: Picture 1 and 2 of Supplementary Material

Various States:

1. Idle – this state is the start of the program and moves onto rightFirst if btnR is pressed or LeftFirst if btnL is pressed. It will stay in this state is none of the buttons are presed or if both he buttons are pressed
2. RightFirst- this state is where the btnR was pressed first which then outputs to itself if only the right button is pressed. If btnR is released meaning both buttons are released it will go back to Idle. If btnL is pressed, meaning both buttons are pressed, it will go to RightLeftBoth.
3. RightLeftBoth - this state will go into itself if both buttons stay pressed. The next state from here is either RightFirst if btnL is released or LeftSecond is btnR is released.
4. LeftSecond – this state is from RightLeftBoth when btnR is released and will stay in this state until another button is released or pressed. If btnR is pressed it will go into state RightLeftBoth. If btnL is released, it will go into RLD(right left done).
5. RLD- this state is from LeftSecond which means a full crossing from right to left which increments the counter up, which turns on IncU. This state automatically goes to IDLE
6. LeftFirst- this state is where the btnL was pressed first which then outputs to itself if only the left button is pressed. If btnL is released meaning both buttons are released it will go back to Idle. If btnR is pressed, meaning both buttons are pressed, it will go to LeftRightBoth.
7. LeftRightBoth - this state will go into itself if both buttons stay pressed. The next state from here is either LeftFirst if btnR is released or RightSecond is btnL is released.
8. RightSecond – this state is from LeftRightBoth when btnL is released and will stay in this state until another button is released or pressed. If btnL is pressed it will go into state LeftRightBoth. If btnR is released, it will go into LRD(left right done).
9. LRD- this state is from RightSecond which means a full crossing from left to right which increments the counter down, which turns on IncD. This state automatically goes to IDLE

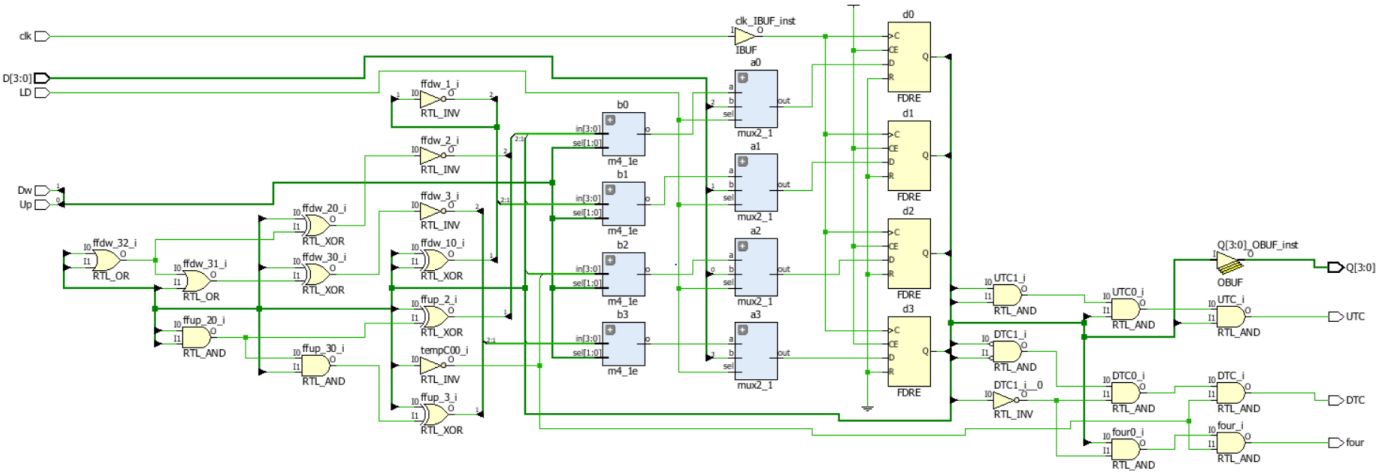
Conclusion

In conclusion, I learned how to build a state machine to create the crossing state machine. The lab first started with creating the smaller modules which were simple, or we already had with minor changes. The next step was to make a diagram of the state machine accounting for every possibility giving us our state table. Using the state table, I was able to get the logical equations for each state and outputs. Finally connecting all the modules in topMod was the most difficult part as there were many inputs and outputs to keep track of. Some difficulties I had were not considering the states by having a turkey fly such as having a turkey cover both sensors or

having a turkey fly off both sensors. If I were to do this lab again, I would make sure my state machine diagram was correct the first time, so I don't have to change the input and outputs of each state and output. I could also optimize this program by not having states that are on for 1 clock cycle, I would make outputs that are based on transition. Saving me from using less states and therefore flipflops for incrementing Up and Down.

Supplementary Material

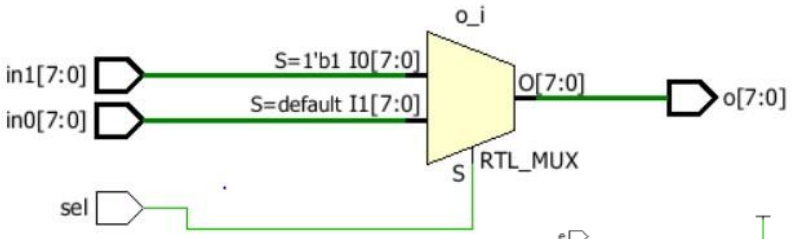
TimeCounter /counterUD4L



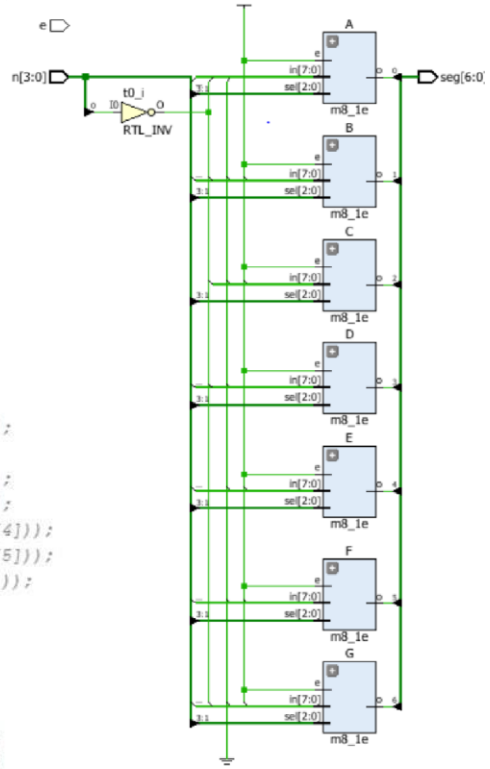

```
module countUD4L(
    input Up,
    input Dw,
    input clk,
    input LD,
    input [3:0] D,
    output [3:0] Q,
    output UTC,
    output DTC,
    output four
    );

      wire [3:0] Din, ffup,ffdw, outC;
      wire [3:0] tempC2,tempC1,tempC0,tempC3;
      wire [1:0] selC = {Dw, Up}; // {down , up}
      assign ffup[0] = ~(Q[0]);
      assign ffdw[0] = ~(Q[0]);
      assign tempC0 = {Q[0],ffdw[0], ffup[0], Q[0]};
      m4_1e b2(.in(tempC0),.sel(selC), .o(outC[0]));// mux for choose
      mux2_1 a2(.a(outC[0]), .b(D[0]), .sel(LD), .out(Din[0]));// mux for load
      FDRE #(.INIT(1'b0) ) d2(.C(clk), .R(reset), .CE(1'b1), .D(Din[0]), .Q(Q[0]));
      assign ffup[1] = (Q[1] ^ Q[0]);
      assign ffdw[1] = ~(Q[1] ^  Q[0]);
      assign tempC1 = {Q[1],ffdw[1], ffup[1], Q[1]};
      m4_1e b1(.in(tempC1),.sel(selC),  .o(outC[1]));// mux for choose
      mux2_1 a1(.a(outC[1]), .b(D[1]), .sel(LD), .out(Din[1]));// mux for load
      FDRE #(.INIT(1'b0) ) d1(.C(clk), .R(reset), .CE(1'b1), .D(Din[1]), .Q(Q[1]));
      assign ffup[2] = (Q[2] ^ (Q[0]&Q[1]));
      assign ffdw[2] = ~(Q[2] ^ (Q[0]|Q[1]));
      assign tempC2 = {Q[2],ffdw[2], ffup[2], Q[2]};
      m4_1e b0(.in(tempC2),.sel(selC), .o(outC[2]));// mux for choose
      mux2_1 a0(.a(outC[2]), .b(D[2]), .sel(LD), .out(Din[2]));// mux for load
      FDRE #(.INIT(1'b0) ) d0(.C(clk), .R(reset), .CE(1'b1), .D(Din[2]), .Q(Q[2]));
      assign ffup[3] = (Q[3] ^ (Q[0]&Q[1]&Q[2]));
      assign ffdw[3] = ~(Q[3] ^ (Q[0]|Q[1]|Q[2]));
      assign tempC3 = {Q[3],ffdw[3], ffup[3], Q[3]};
      m4_1e b3(.in(tempC3),.sel(selC), .o(outC[3]));// mux for choose
      mux2_1 a3(.a(outC[3]), .b(D[3]), .sel(LD), .out(Din[3]));// mux for load
      FDRE #(.INIT(1'b0) ) d3(.C(clk), .R(reset), .CE(1'b1), .D(Din[3]), .Q(Q[3]));
       assign four = Q[2]&~Q[1]&~Q[0];
       assign UTC = (Q[3] & Q[2] & Q[1] & Q[0]);
       assign DTC = (~Q[3] &~Q[2] & ~Q[1] & ~Q[0]);
endmodule
```

Mux 2_8x1

```
module m2_1x8(
    input [7:0] in0,
    input [7:0] in1,
    input sel,
    output [7:0] o
    );
    assign o = sel ? in1:in0;
endmodule
```

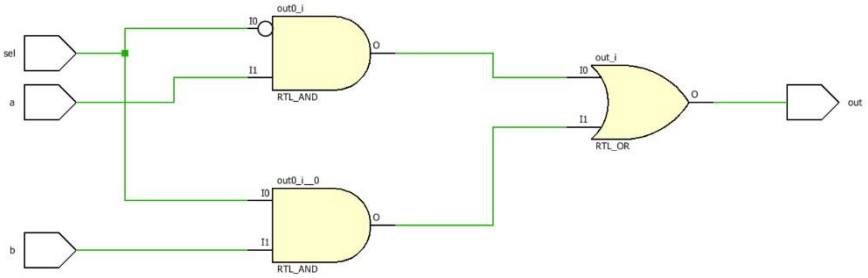


Hex7Seg

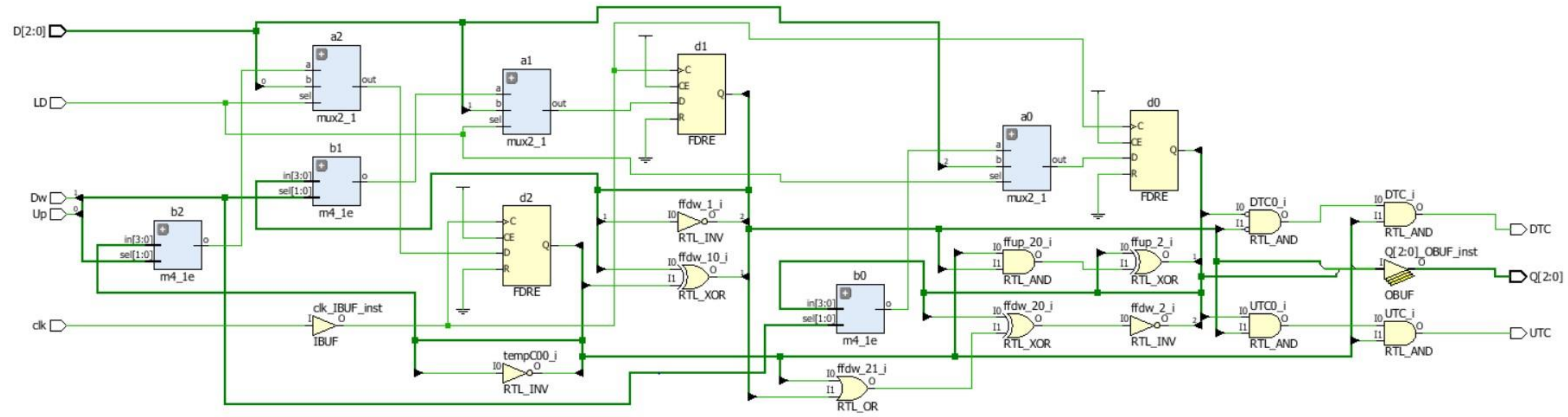



```
module hex7seg(
    input [3:0] n,
    input e,
    output [6:0] seg
    );

    wire t0 = ~n[0];
    wire [6:0] te;
    /*
    //m8_1e A( .in({n[0],1'b0,t0,1'b0,1'b0,n[0],n[0],1'b0}), .sel(n[3:1]), .e(1'b1), .o(seg[0]));
    m8_1e B( .in({1'b0,1'b0,n[0],t0,1'b0,1'b0,1'b0,1'b1}), .sel(n[3:1]), .e(1'b1), .o(seg[1]));
        m8_1e C( .in({1'b0,t0,1'b0,1'b0,1'b0,1'b0,t0,1'b1}), .sel(n[3:1]), .e(1'b1), .o(seg[2]));
        m8_1e D( .in({n[0],1'b0,t0,n[0],n[0],t0,1'b0,n[0]}), .sel(n[3:1]), .e(1'b1), .o(seg[3]));
        m8_1e E( .in({n[0],n[0],1'b1,n[0],n[0],1'b0,1'b0,1'b0}), .sel(n[3:1]), .e(1'b1), .o(seg[4]));
        m8_1e F( .in({n[0],1'b1,1'b0,n[0],1'b0,1'b0,n[0],1'b0}), .sel(n[3:1]), .e(1'b1), .o(seg[5]));
        m8_1e G( .in({1'b1,1'b0,1'b0,n[0],1'b0,1'b0,t0,1'b0}), .sel(n[3:1]), .e(1'b1), .o(seg[6]));
        */
    m8_1e A( .in({1'b0,n[0],n[0],1'b0,1'b0,t0,1'b0,n[0]}), .sel(n[3:1]), .e(1'b1), .o(te[0]));
    m8_1e B( .in({1'b1,t0,n[0],1'b0,t0,n[0],1'b0,1'b0}), .sel(n[3:1]), .e(1'b1), .o(te[1]));
    m8_1e C( .in({1'b1,t0,1'b0,1'b0,1'b0,1'b0,t0,1'b0}), .sel(n[3:1]), .e(1'b1), .o(te[2]));
    m8_1e D( .in({n[0],1'b0,t0,n[0],n[0],t0,1'b0,n[0]}), .sel(n[3:1]), .e(1'b1), .o(te[3]));
    m8_1e E( .in({1'b0,1'b0,1'b0,n[0],n[0],1'b1,n[0],n[0]}), .sel(n[3:1]), .e(1'b1), .o(te[4]));
    m8_1e F( .in({1'b0,n[0],1'b0,1'b0,n[0],1'b0,1'b1,n[0]}), .sel(n[3:1]), .e(1'b1), .o(te[5]));
    m8_1e G( .in({1'b0,t0,1'b0,1'b0,n[0],1'b0,1'b0,1'b1}), .sel(n[3:1]), .e(1'b1), .o(te[6]));
    //assign seg = ~seg;
   assign seg = te;
endmodule
```

```
module mux2_1(
    input a,
    input b,
    input sel,
    output out
    );
    assign out = (~sel&a) | (sel&b);
endmodule
```



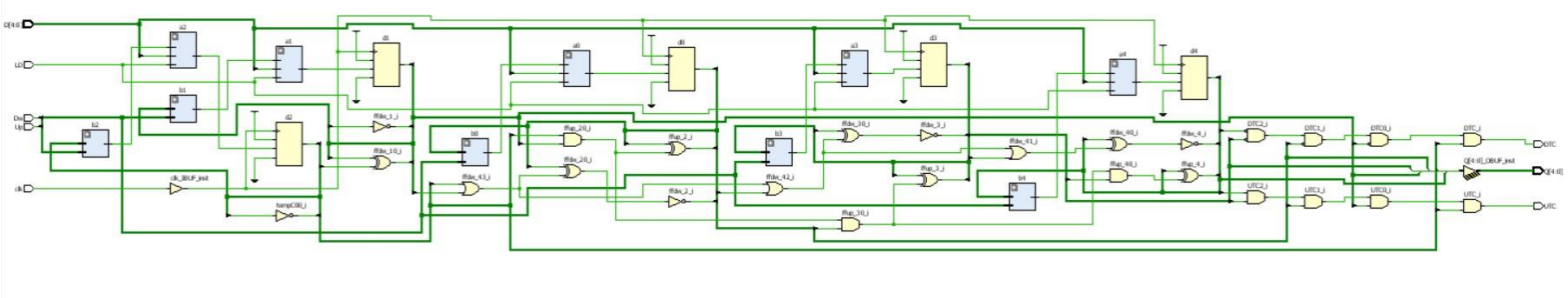
CountUD3L



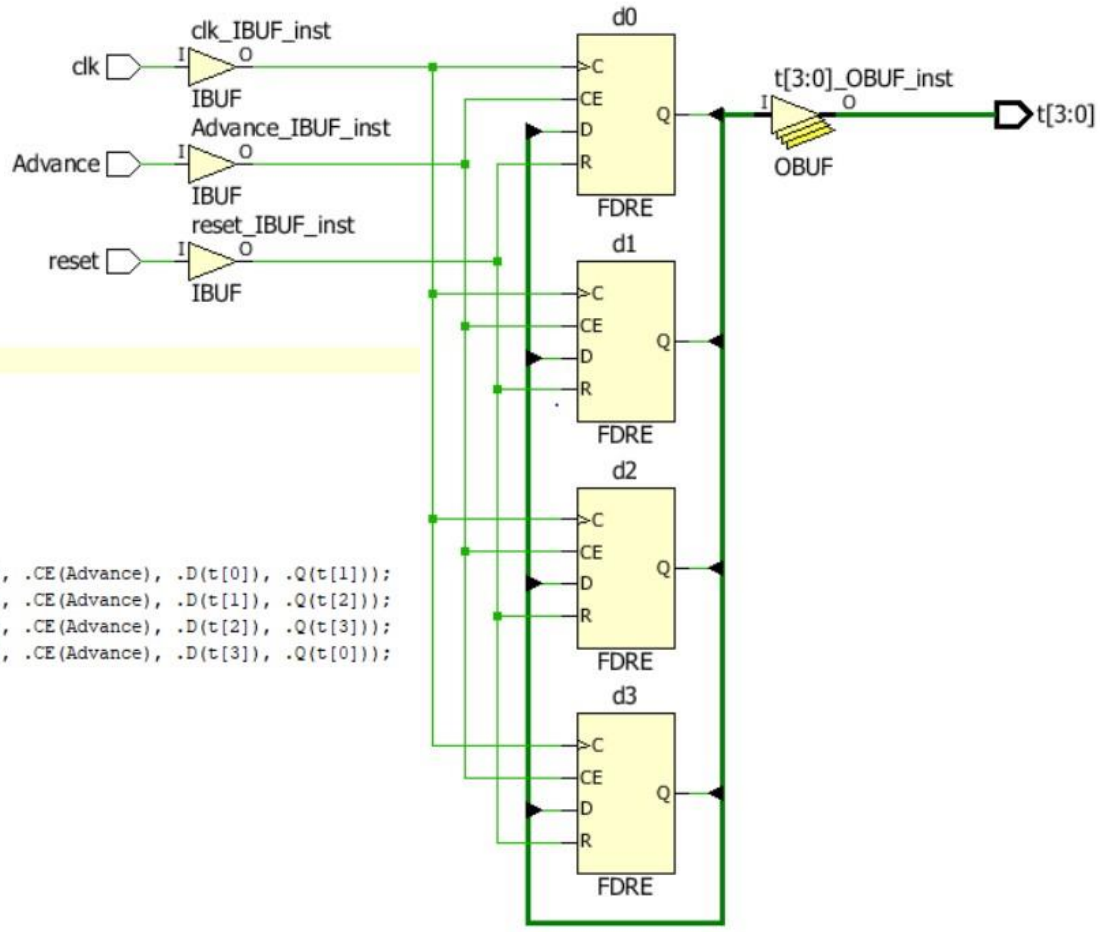

```
module countUD3L(
    input clk,
    input Up,
    input Dw,
    input LD,
    input [2:0] D,
    output [2:0] Q,
    output UTC,
    output DTC
    );

    wire enable = LD | (Up ^ Dw);
    //assign LD = 1'b0;
    wire [2:0] Din, ffup,ffdw, outC;
    wire [3:0] tempC2,tempC1,tempC0;
    wire [1:0] selC = {Dw, Up};

    assign ffup[0] = ~(Q[0]);
    assign ffdw[0] = ~(Q[0]);
    assign tempC0 = {Q[0],ffdw[0], ffup[0], Q[0]};
    m4_1e b2(.in(tempC0),.sel(selC), .o(outC[0]));// mux for choose

    mux2_1 a2(.a(outC[0]), .b(D[0]), .sel(LD), .out(Din[0]));// mux for load
    FDRE #(.INIT(1'b0) ) d2(.C(clk), .R(reset), .CE(1'b1), .D(Din[0]), .Q(Q[0]));

    assign ffup[1] = (Q[1] ^ Q[0]);
    assign ffdw[1] = ~(Q[1] ^  Q[0]);
    assign tempC1 = {Q[1],ffdw[1], ffup[1], Q[1]};
    m4_1e b1(.in(tempC1),.sel(selC),  .o(outC[1]));// mux for choose

    mux2_1 a1(.a(outC[1]), .b(D[1]), .sel(LD), .out(Din[1]));// mux for load
    FDRE #(.INIT(1'b0) ) d1(.C(clk), .R(reset), .CE(1'b1), .D(Din[1]), .Q(Q[1]));

    assign ffup[2] = (Q[2] ^ (Q[0]&Q[1]));
    assign ffdw[2] = ~(Q[2] ^ (Q[0]|Q[1]));
    assign tempC2 = {Q[2],ffdw[2], ffup[2], Q[2]};
    m4_1e b0(.in(tempC2),.sel(selC), .o(outC[2]));// mux for choose
    mux2_1 a0(.a(outC[2]), .b(D[2]), .sel(LD), .out(Din[2]));// mux for load
    FDRE #(.INIT(1'b0) ) d0(.C(clk), .R(reset), .CE(1'b1), .D(Din[2]), .Q(Q[2]));

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

    endmodule
```

CountUD5L

```
module countUD5L(
    input clk,
    input Up,
    input Dw,
    input LD,
    input [4:0] D,
    output [4:0] Q,
    output UTC,
    output DTC
    );

    wire enable = LD | (Up ^ Dw);
    //assign LD = 1'b0;
    wire [4:0] Din, ffup,ffdw, outC;
    wire [3:0] tempC2,tempC1,tempC0,tempC3,tempC4;
    wire [1:0] selC = {Dw, Up};

    assign ffup[0] = ~(Q[0]);
    assign ffdw[0] = ~(Q[0]);
    assign tempC0 = {Q[0],ffdw[0], ffup[0], Q[0]};
    m4_1e b2(.in(tempC0),.sel(selC), .o(outC[0]));// mux for choose
    mux2_1 a2(.a(outC[0]), .b(D[0]), .sel(LD), .out(Din[0]));// mux for load
    FDRE #(.INIT(1'b0) ) d2(.C(clk), .R(reset), .CE(1'b1), .D(Din[0]), .Q(Q[0]));

    assign ffup[1] = (Q[1] ^ Q[0]);
    assign ffdw[1] = ~(Q[1] ^  Q[0]);
    assign tempC1 = {Q[1],ffdw[1], ffup[1], Q[1]};
    m4_1e b1(.in(tempC1),.sel(selC),  .o(outC[1]));// mux for choose
    mux2_1 a1(.a(outC[1]), .b(D[1]), .sel(LD), .out(Din[1]));// mux for load
    FDRE #(.INIT(1'b0) ) d1(.C(clk), .R(reset), .CE(1'b1), .D(Din[1]), .Q(Q[1]));

    assign ffup[2] = (Q[2] ^ (Q[0]&Q[1]));
    assign ffdw[2] = ~(Q[2] ^ (Q[0]|Q[1]));
    assign tempC2 = {Q[2],ffdw[2], ffup[2], Q[2]};
    m4_1e b0(.in(tempC2),.sel(selC), .o(outC[2]));// mux for choose
    mux2_1 a0(.a(outC[2]), .b(D[2]), .sel(LD), .out(Din[2]));// mux for load
    FDRE #(.INIT(1'b0) ) d0(.C(clk), .R(reset), .CE(1'b1), .D(Din[2]), .Q(Q[2]));

    assign ffup[3] = (Q[3] ^ (Q[0]&Q[1]&Q[2]));
    assign ffdw[3] = ~(Q[3] ^ (Q[0]|Q[1]|Q[2]));
    assign tempC3 = {Q[3],ffdw[3], ffup[3], Q[3]};
    m4_1e b3(.in(tempC3),.sel(selC), .o(outC[3]));// mux for choose
    mux2_1 a3(.a(outC[3]), .b(D[3]), .sel(LD), .out(Din[3]));// mux for load
    FDRE #(.INIT(1'b0) ) d3(.C(clk), .R(reset), .CE(1'b1), .D(Din[3]), .Q(Q[3]));

    assign ffup[4] = (Q[4] ^ (Q[0]&Q[1]&Q[2]&Q[3]));
    assign ffdw[4] = ~(Q[4] ^ (Q[0]|Q[1]|Q[2]|Q[3]));
    assign tempC4 = {Q[4],ffdw[4], ffup[4], Q[4]};
    m4_1e b4(.in(tempC4),.sel(selC), .o(outC[4]));// mux for choose
    mux2_1 a4(.a(outC[4]), .b(D[3]), .sel(LD), .out(Din[4]));// mux for load
    FDRE #(.INIT(1'b0) ) d4(.C(clk), .R(reset), .CE(1'b1), .D(Din[4]), .Q(Q[4]));

    assign UTC = (Q[4] & Q[3] & Q[2] & Q[1] & Q[0]);
    assign DTC = (~Q[4] & ~Q[3] &~Q[2] & ~Q[1] & ~Q[0]);

    endmodule
```

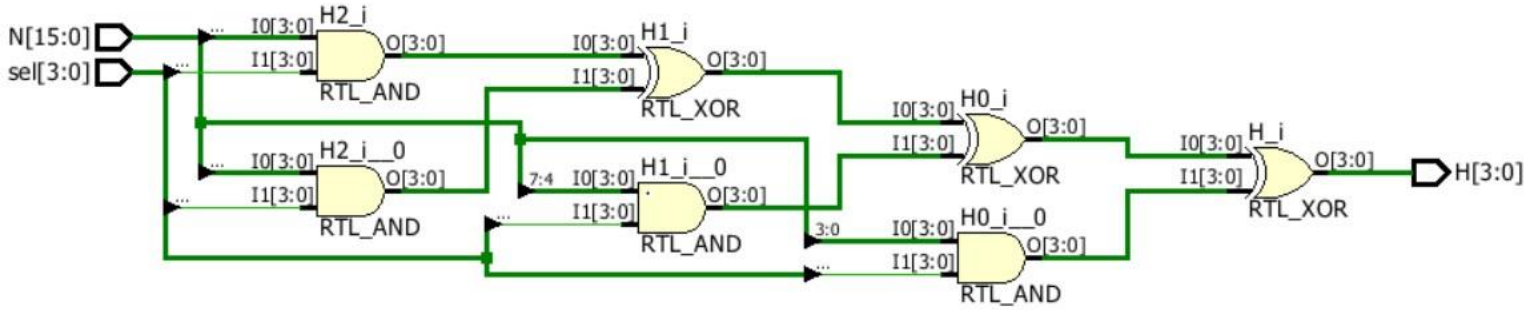


Ring Counter




```
module RingCNT(
    input clk,
    input Advance,
    input reset,
    output [3:0]t
    );

   FDRE #(.INIT(1'b1) ) d0(.C(clk), .R(reset), .CE(Advance), .D(t[0]), .Q(t[1]));
   FDRE #(.INIT(1'b0) ) d1(.C(clk), .R(reset), .CE(Advance), .D(t[1]), .Q(t[2]));
   FDRE #(.INIT(1'b0) ) d2(.C(clk), .R(reset), .CE(Advance), .D(t[2]), .Q(t[3]));
   FDRE #(.INIT(1'b0) ) d3(.C(clk), .R(reset), .CE(Advance), .D(t[3]), .Q(t[0]));

endmodule
```

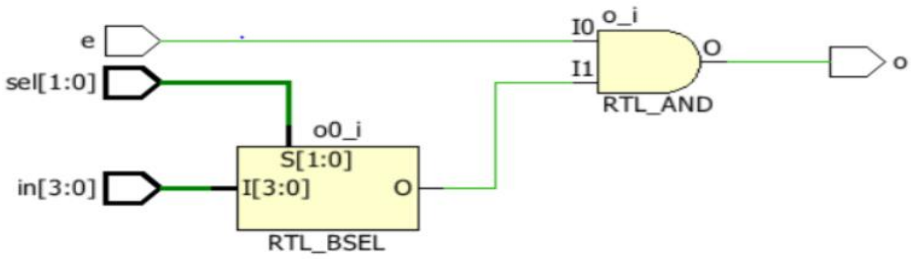
Selector




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

Mux 4x1

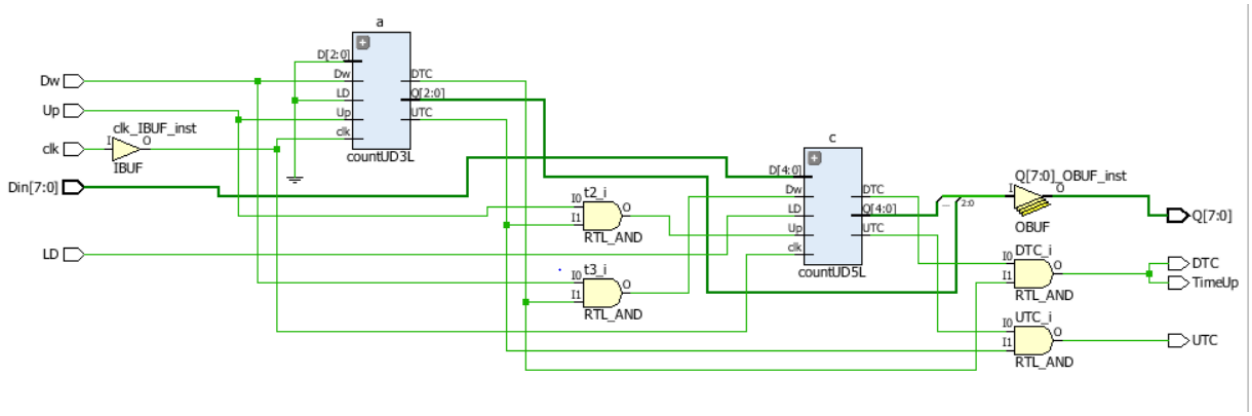
```
module m4_le(
    input [3:0] in,
    input [1:0] sel,
    input e,
    output o
    );
   //assign o = e & ((in[0] &
    assign o = e & (in[sel]);
endmodule
```



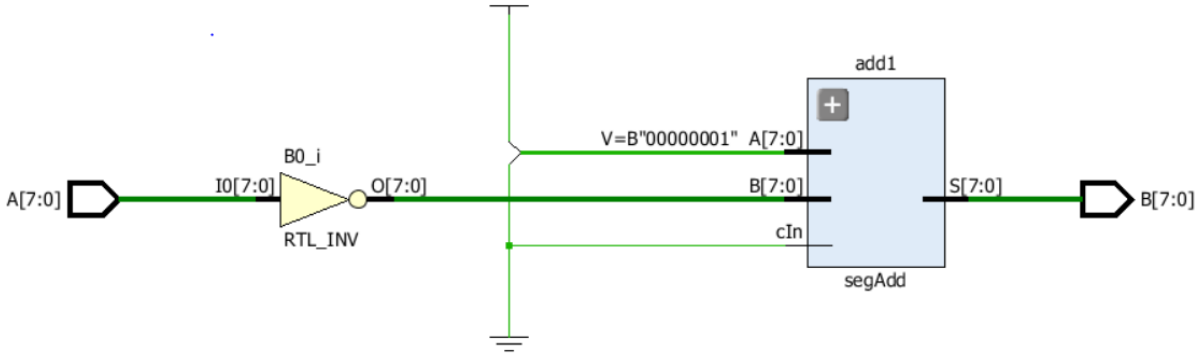
Turkey Counter

```
module turkeyCount(
    input Up,
    input Dw,
    input clk,
    input LD,
    input [7:0] Din,
    output [7:0] Q,
    output UTC,
    output DTC,
    output TimeUp
    );

        wire [1:0] tempUTC, tempDTC;
        wire t0,t1,t2,t3;
        countUD3L a(.clk(clk), .Up(Up), .Dw(Dw), .LD(1'b0), .D(3'b000), .Q(Q[2:0]), .UTC(tempUTC[0]) , .DTC(tempDTC[0]));
        assign t2 = Up & tempUTC[0];
        assign t3 = Dw & tempDTC[0];
        countUD5L c(.clk(clk), .Up(t2), .Dw(t3), .LD(LD), .D(Din[7:3]), .Q(Q[7:3]), .UTC(tempUTC[1]) , .DTC(tempDTC[1]));
        assign UTC = tempUTC[1] &tempUTC[0];
        assign DTC = tempDTC[1] &tempDTC[0];

        assign TimeUp = DTC;


endmodule
```

\



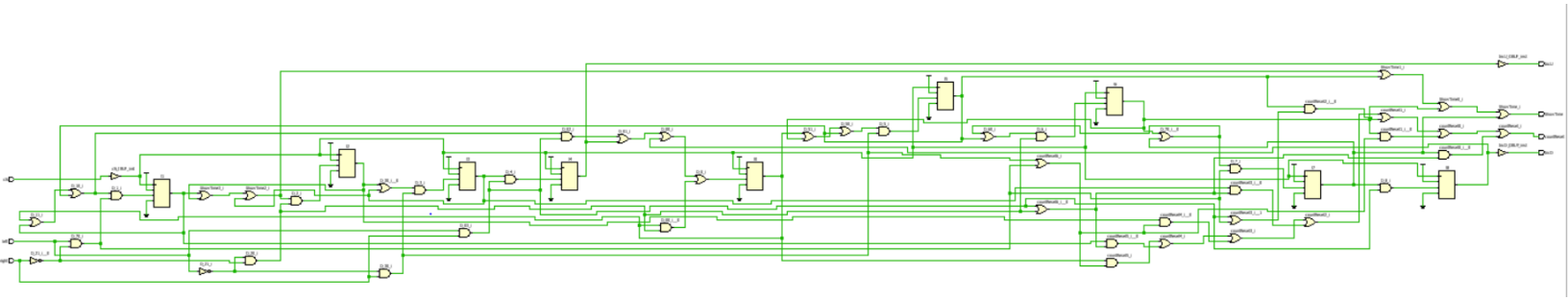
2’s complement convertor




```
module conv2sc(
    input [7:0] A,
    output [7:0] B
    );
    wire [7:0] TB;
    wire t5;
    segAdd add1( .A({8'b00000001}), .B(~A), .cIn(1'b0), .S(B), .cOut(t5));

    // wire [7:0] C;
    //segAdd add1( .A({1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0,1'b0}), .B(TB), .
    //assign B = C;

endmodule
```

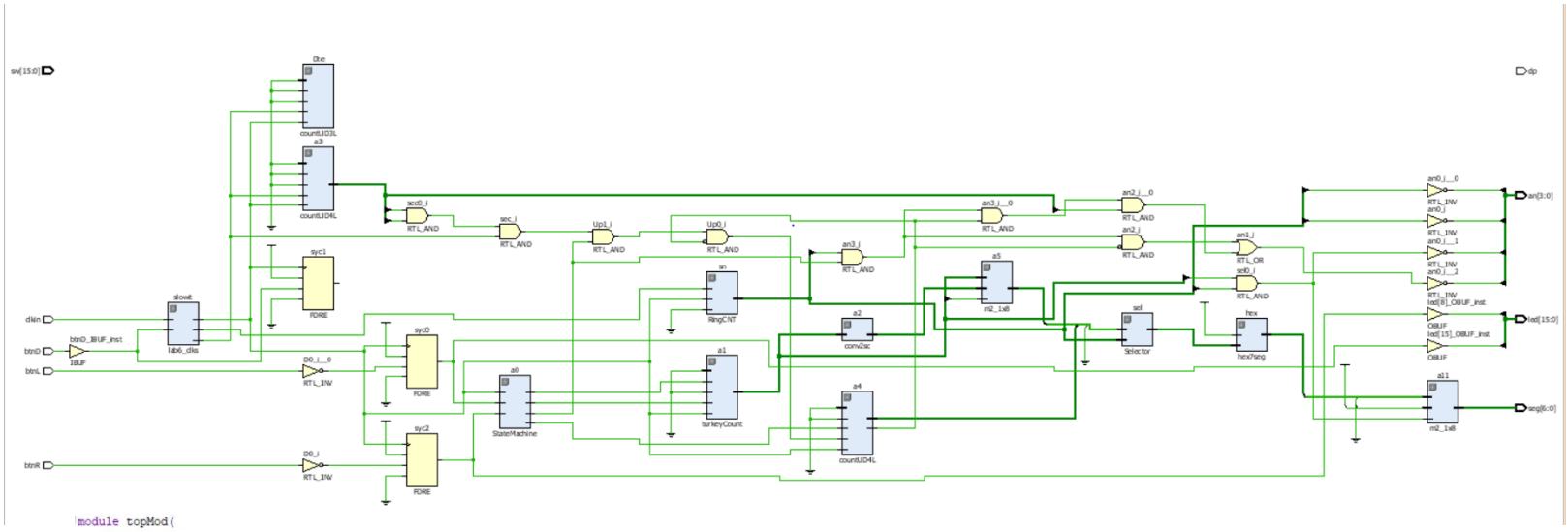
Uses modules segAdd, and FullAdder which are modules not listed by used in previous labs

State Machine



```
module StateMachine(
    input clk,
    input left,
    input right,
    output ShowTime,
    output countReset,
    output IncU,
    output IncD
    );
    wire[8:0] D, Q;
    wire enable = 1'b1;


    //Idle
    assign D[0] = (Q[0]|Q[1]|Q[5])&(left&right)  |  Q[4] | Q[8] | (Q[0]&(~left&~right));
    FDRE #(.INIT(1'b1)) t0 (.C(clk), .CE(enable), .D(D[0]), .Q(Q[0]));

    //Right First
    assign D[1] = (Q[0]| Q[1]| Q[5])&(left&~right);
    FDRE #(.INIT(1'b0)) t1 (.C(clk), .CE(enable), .D(D[1]), .Q(Q[1]));

    // right to left
    assign D[2] = (Q[2]| Q[1]| Q[3])&(~left&~right);
    FDRE #(.INIT(1'b0)) t2 (.C(clk), .CE(enable), .D(D[2]), .Q(Q[2]));

    //left after right
    assign D[3] =(Q[2]| Q[3])&(~left&right);
    FDRE #(.INIT(1'b0)) t3 (.C(clk), .CE(enable), .D(D[3]), .Q(Q[3]));

    //Right to left done
    assign D[4] =  (Q[3])&(left&right);
    FDRE #(.INIT(1'b0)) t4 (.C(clk), .CE(enable), .D(D[4]), .Q(Q[4]));

    //left first
    assign D[5] =  (Q[0]| Q[6]| Q[5])&(~left&right);
    FDRE #(.INIT(1'b0)) t5 (.C(clk), .CE(enable), .D(D[5]), .Q(Q[5]));

    //left to right
    assign D[6] =  (Q[6]| Q[7]| Q[5])&(~left&~right);
    FDRE #(.INIT(1'b0)) t6 (.C(clk), .CE(enable), .D(D[6]), .Q(Q[6]));

    assign D[7] =  (Q[6]| Q[7])&(left&~right);
    FDRE #(.INIT(1'b0)) t7 (.C(clk), .CE(enable), .D(D[7]), .Q(Q[7]));

    //left to right done
    assign D[8] =  Q[7]&(left&right);
    FDRE #(.INIT(1'b0)) t8 (.C(clk), .CE(enable), .D(D[8]), .Q(Q[8]));

    assign ShowTime = Q[1]|Q[2]|Q[3]|Q[5]|Q[6]|Q[7];
    assign IncU = Q[4];
    assign IncD = Q[8];
    assign countReset = (Q[0]&((right&~left)|(~right&left)))|
    (Q[1]&((right&left)|(~right&~left)))|
    (Q[2]&((right&~left)|(~right&left)))|
    (Q[3]&((right&left)|(~right&~left)))|
    (Q[5]&((right&left)|(~right&left )))|
    (Q[6]&((right&~left)|(~right&left)))|
    (Q[7]&((~right&~left)|(right&left)));


endmodule
```

TopMod

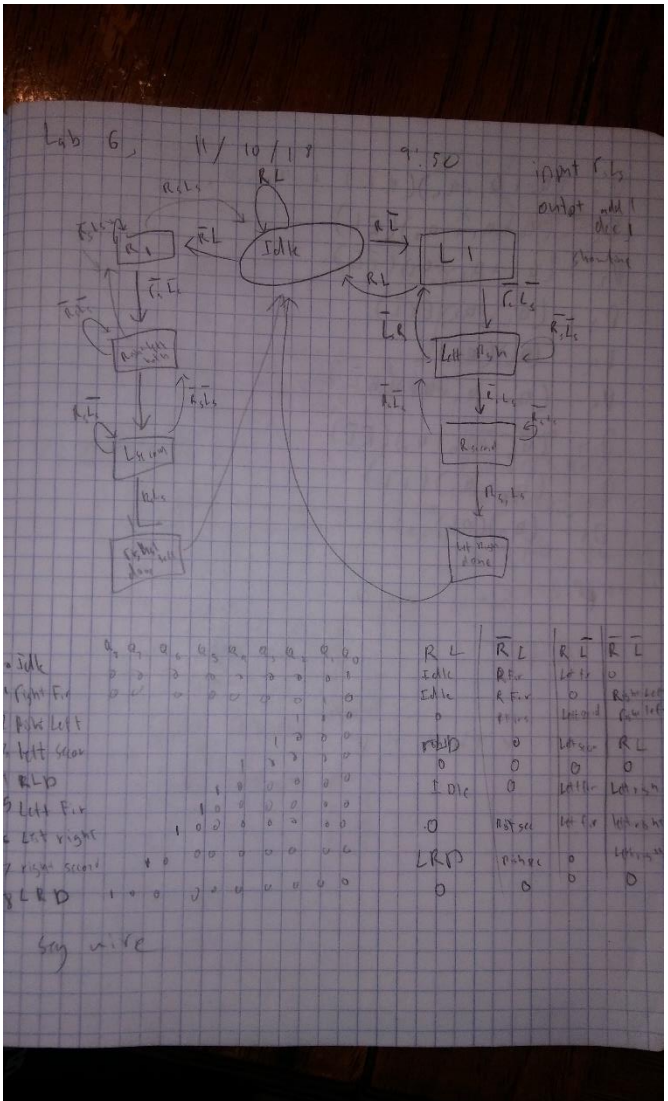


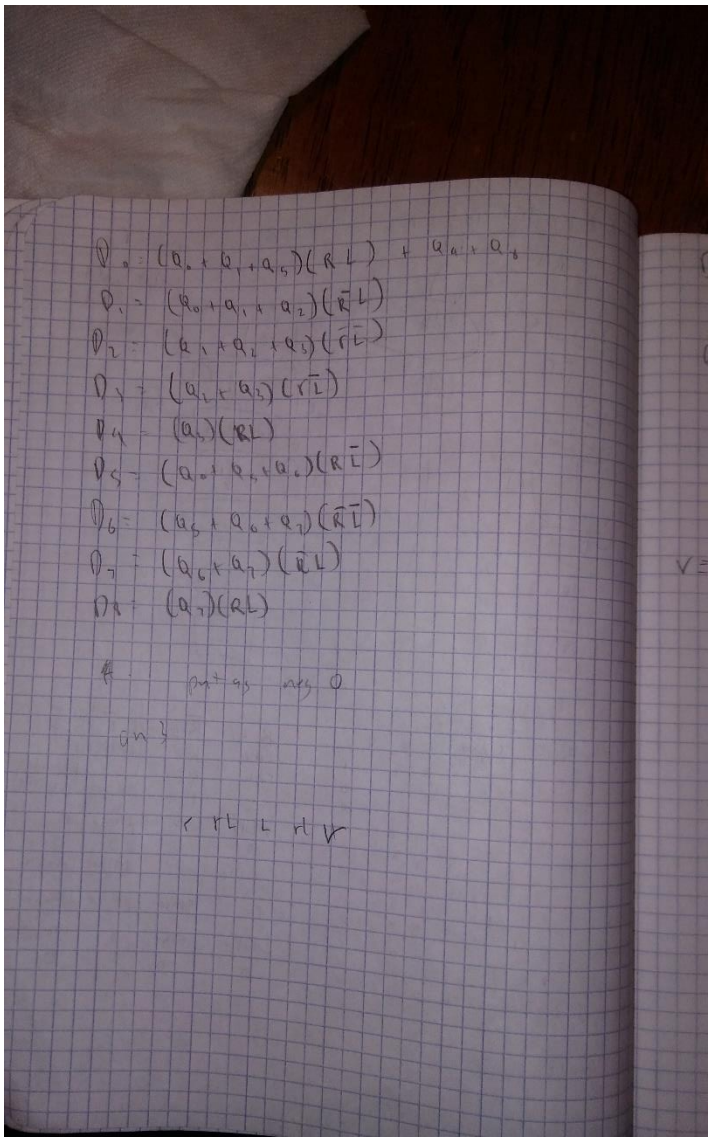
```
module topMod(
    input clkin,
    input btnD,
    input btnR,
    input btnL,
    input [15:0]sw,
    output [6:0] seg,
    output dp,
    output [3:0]an,
    output [15:0]led
    );
    wire clk, digsel, qsec;
    wire ShowTime, IncU, IncD, Flash, cntRes, UTC, DTC, four;
    wire [7:0] turCountOut, invTCount, finalTcount;
     wire [15:0] pick;
     wire [3:0] turkCrossT,turkSec, select,dam;

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

    wire btnLsyn,btnDsyn,btnRsyn;
    FDRE #(.INIT(1'b0)) syc0 (.C(clk), .CE(1'b1), .D(~btnL), .Q(btnLsyn));
    FDRE #(.INIT(1'b0)) syc1 (.C(clk), .CE(1'b1), .D(btnD), .Q(btnDsyn));
    FDRE #(.INIT(1'b0)) syc2 (.C(clk), .CE(1'b1), .D(~btnR), .Q(btnRsyn));

    assign led[15] = btnLsyn;
    assign led[8] = btnRsyn;

    StateMachine a0(.clk(clk),.left(btnLsyn),.right(btnRsyn),.ShowTime(ShowTime),.countReset(cntRes),.IncU(IncU),.IncD(IncD));
    turkeyCount a1(.Up(IncU),.Dw(IncD),.clk(clk),.LD(1'b0),.Din(8'b00000000),.Q(turCountOut));

    //counvert to 2sc
    conv2sc a2(.A(turCountOut),.B(invTCount));
    //choose between 2sc or turcount based on most significant bit,
    m2_1x8 a5(.in0(turCountOut),.in1(invTCount),.sel(turCountOut[7]),.o(finalTcount));

    //counter out will be 1 sec
    countUD4L a3(.Up(qsec),.Dw(1'b0),.clk(clk),.LD(1'b0),.D(4'b0000),.Q(turkSec));
    wire sec = turkSec[1] & turkSec[0] & qsec;
    //cross count
    countUD4L a4(.Up(sec & ShowTime & ~four ),.Dw(1'b0),.clk(clk),.LD(cntRes),.D(4'b0000),.Q(turkCrossT),.four(four));
     //display for negative number, use mux,
     wire [6:0] neg = {1'b1,1'b1,1'b1,1'b1,1'b1,1'b1,1'b1};
     wire [6:0] pos = {1'b1,1'b1,1'b1,1'b1,1'b1,1'b1,1'b1};
     wire [7:0] sign;
     // choose seg


     RingCNT sn(.clk(clk),.Advance(digsel),.t(select));
     assign pick[7:0] = finalTcount;
     assign pick[11:8] = {4'b0000};//timerOut;
    assign pick[15:12] = turkCrossT;

     Selector sel(.sel(select),.N(pick),.H(dam));
     wire [6:0]intempSeg;
     wire [7:0] outtempSeg;

     hex7seg hex(.n(dam),.e(1'b1),.seg(intempSeg));


     assign seg = outtempSeg[6:0];
    m2_1x8 a11(.in0({1'b0,intempSeg}),.in1(8'b0111111),.sel(turCountOut[7]& select[2]),.o(outtempSeg));
    //2bit counter for flash
    wire[2:0] temp3;

    countUD3L Dte(.Up(qsec),.Dw(1'b0),.clk(clk),.LD(1'b0),.D({1'b0,1'b0,1'b0}),.Q(temp3),.UTC(teme2),.DTC(teme3));
    wire q12 = temp3[1];


    assign an[0] = ~select[0]; //~((select[0]&ShowScore)&~FlashD  |   (select[0]&ShowScore)&FlashD&q12) ;
    assign an[1] = ~select[1]; //~((select[1]&(ShowTime | sw)));
    assign an[2] = ~((select[2]&(turCountOut[7])));
    assign an[3] =  ~((select[3]& ShowTime & ~four) | (select[3]&ShowTime & four & turkSec[1])) ;



ndmodule
```

TopMod Sim

Picture 1

Picture 2





Picture 3