David Nguyen

Lab 4

Section C

11/16/2018

Lab 5 Writeup

Purpose:

The purpose of this lab was to learn how to build a state machine for a time-based game of chicken that depended on btnU, btnD, and btnC. This game loaded up a random time and the winner would be the last person to press their button before time ran out, but the button can only be pressed once per match. In this lab, we included a LFSR to generate a random number, counter, and state machine.

Methods:

Random Number Generator

1. To start the Random Number Generator, there are 8 flipflops daisy chained to one another such as the output of 1 is input of 2 and output of 2 in input of 3, until 8.
2. The input of the first flip flop is the output of the xor of the flipflops 0,5,6, and 7. This will give a sequence of numbers that seem random because the LFSR is read at random times.
3. The final step to this is to set the initial value of the first flipflop to 1, else the output will always be 0. The output of this module the an 8bit bus of the output of the flipflops.

TimeCounter

1. The first step of creating the TimeCounter is to declare an instance of an UpDown 3 and 5 bit counter. This creates an 8bit count that will go up or down, although only the down part of the counter is needed.
2. To connect the 3 and 5 bit counters, the there will be a 3bit input and output for the 3 bit counter, and a 5bit output for the 5 bit counter. The Up, Dw, LD of the first counter will be the input Up, Dw, LD. The input of the second counter will be UP&(UTC of first counter), DW&(DTC of first counter), and LD.
3. The DTC and UTC output of TimeCounter are if the UTC or DTC of both counters are high for the respective output. Finally, there is an output TimeUp, which will be declared to UTC of TimeCounter.

Score Display

1. The Score Display is a 4bit counter to keep track of each players score, this can be done by creating a new separate 4 bit counter, or by using a 5 bit counter and ignoring the 5^{th} bit.

2. To create a new 4bit counter, one can take the code of the 5 bit counter, change the input and output to 4 bits, delete the logic, code, and flipflop for the 5^{th} bit, and finally change UTC and DTC to have a 4bit counter.
3. To use an existing 5 bit counter, one will declare a 5bit counter in their Top Module and output the counter to a 5bit bus wire. Then they would assign another 4bit bus wire to [3:0]5bit bus wire.
4. This counter is able to keep track of players scores which is then the output is display to the board.

Led Shifter

1. The Led Shifter is a module that requires 16 daisy chained flipflops similar to the Random Number Generator up to 16, a 16bit shift register.
2. All flipflops are initialized to zero but the input of the first flipflop is 1. This means the output of the flipflops start at 00000, 00001, 00011, …. ,11111. Which the output is set to the Led meaning the Leds act as a loading bar from 0 to 16 when the enable is on.
3. The enable of this module is qsec & ShowTime, meaning it only shifts when time is suppose to be displayed on the board(acts as a loading the time onto the board) every quarter of a second.
4. The reset of this module is lastled(led[15]) & ~ShowTime, this resets the Leds to all 0, until it is time to show the time again(when ShowTime becomes on again).

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 btnU, btnD, btnC, TimeUp, and lastLed as well as the outputs ShowTime, LoadTime, RunTime, IncU, IncD, ShowScore, FlashU, and FlashD.
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 diagram 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.

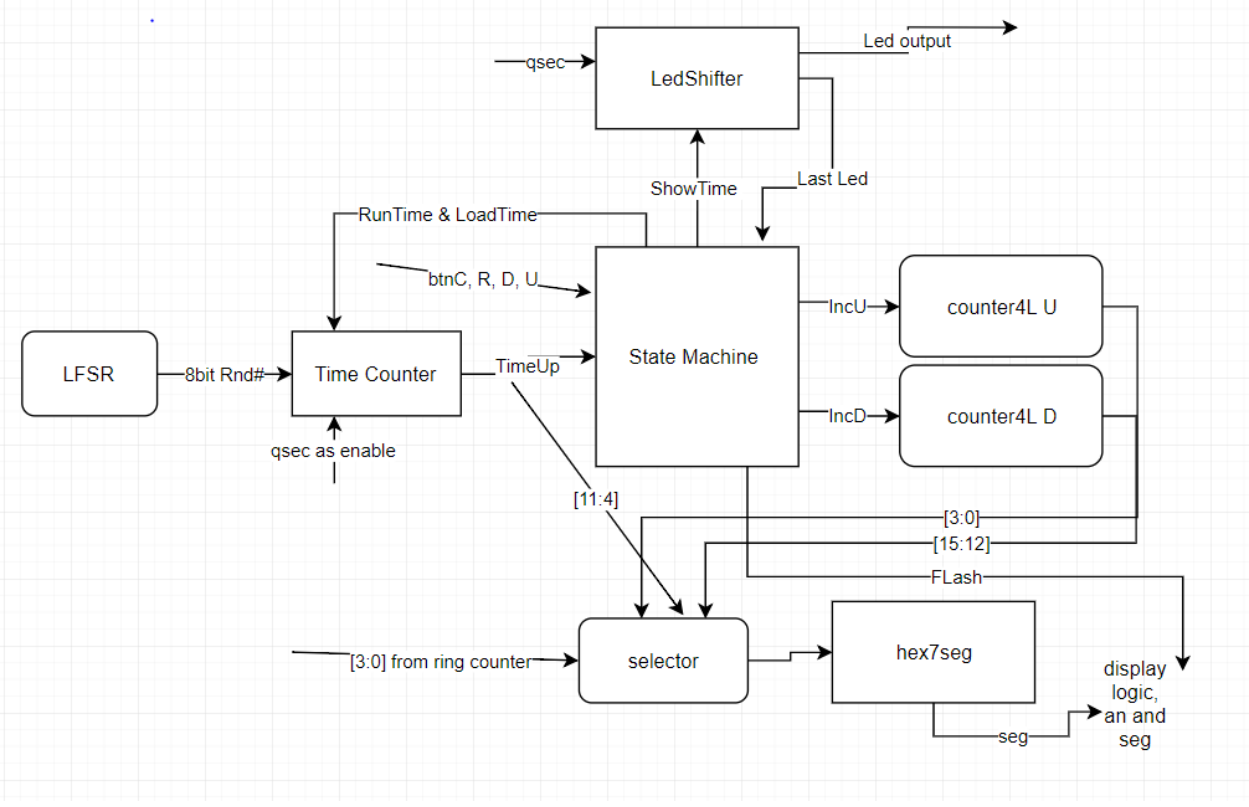
TopMod5

1. The first step of the TopMod is to add the Verilog file lab5_clks.v and the following line should be pasted into TopMod:
   ab5_clks slowit (.clkin(clkin), .greset(btnR), .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 btnU, btnD, and btnC 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.

3. The next step is to create 1 module of the Random Number generator, whose output is the Load Input of the TimeCounter. A Led Shifter module is created using logic from step 3 and 4 of Led Shifter for reset and enable.
4. The TimeCounter has input Up(0), Dw(qsec&RunTime)$_{\text{every quarter second}}$,LD(LoadTime), and Din(Rnd#Gen output). The output of this is then an 8 bit bus to display the time left on the game and TimeUp.
5. The next step is to create 2 4 bit counters for each player score with the Up increment being IncU and IncD for each counter. The output is then 2 4bit bus used to display the score of each player on the board.
6. Adding the state machine to the top level, is fairly easy as wires are used for each input and output making sure that each input and output is accounted for.
7. The next step is to have create a 16 bit wire with the {first 4bits as your ScoreCounter1, 8bits of time left, and 4 bits of ScoreCounter 2}, this is ran through selector with the ring counter and outputted to hex7seg.
8. The final step to the TopMod is the display logic for an. An 0 and 3 flash every half a second when that respective player wins. This is done by having a 2 bit counter and only using the output of the 2nd bit in logic for flashing. This logic is the inverse of (ShowScore&~flash) | (ShowScore&Flash&2nd bit). FlashU and FlashD are put in place of flash with select[0] and [3] also added to the logic. An 1 and 2 are supposed to be on when ShowTime is active or when sw[15] is active. The logic to this is the inverse of (ShowTime | sw[15])&sel[2 or 1].

Results



The topMod of this lab starts with LFSR which outputs an 8bit random number to time counter. TimeCounter then takes in RunTime & LoadTime from the State Machine based off the button presses. The output TimeUp is then fed back into the State Machine and an 8 bit output is put into selector at [11:4]. Led Shifter takes in ShowTime from the State Machine and inputs LastLed back into the state machine while also setting the led output. The State Machine then turns on IncU and IncD based off the button pressed which act as the enable for the 2 score counters. The 2 score counters output their output into selector at [3:0] and [15:12]. Selector takes in a 4 bit bus from ring counter and sends the output to hex7seg. Finally hex7seg and the 3 flash outputs from the state machine determine the display logic for seg and an.

To test and simulate the State Machine, I tried to make the state machine reach every state by changing btnC,, btnU, btnD, timeUp, and LastLed. 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.

To test and simulate the TopModule, I also tried to make the state machine reach every state by changing btnC,, btnU, and btnD. 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 simulated sw15 to see if the cheat switch worked for an 1 and 2.

Picture of State Diagram: Picture 5 of Supplementary Material

Logic Equations: Picture 3 of Supplementary Material

Next State and Output: Picture 3 and 2 of Supplementary Material

VariousStates:

1. Idle – this state is the start of the program and moves onto ledload when btnC is pressed and ShowScore is on.
2. Ledload- this state is waiting for the leds to load up from 0 to 16 and moves onto accept state when lastled is on. This turns on LoadTime and ShowTime.
3. Accept- this state is waiting for a btnD or U press either going to UFirst, Dfirst, or Tie depending on the button press. If no buttons are pressed before time up is on, the next state is Idle. This turns on Runtime.
4. Tie – this state is when both buttons are pressed at the same time incrementing U and D for 1 clock cycle then goes into state TFlash.
5. UFirst - this state is from accept and is when the first button pressed is U and timeUp is off. It will stay in this state until time is up which goes to Uwin, or it will go to Dwin if btnD is pressed.
6. DFirst - this state is from accept and is when the first button pressed is D and timeUp is off. It will stay in this state until time is up which goes to Dwin, or it will go to Uwin if btnU is pressed.
7. Uwin- this state is when player Uwin and is on for 1 clock cycle incrementing U and automatically going to Uflash the next clock cycle.

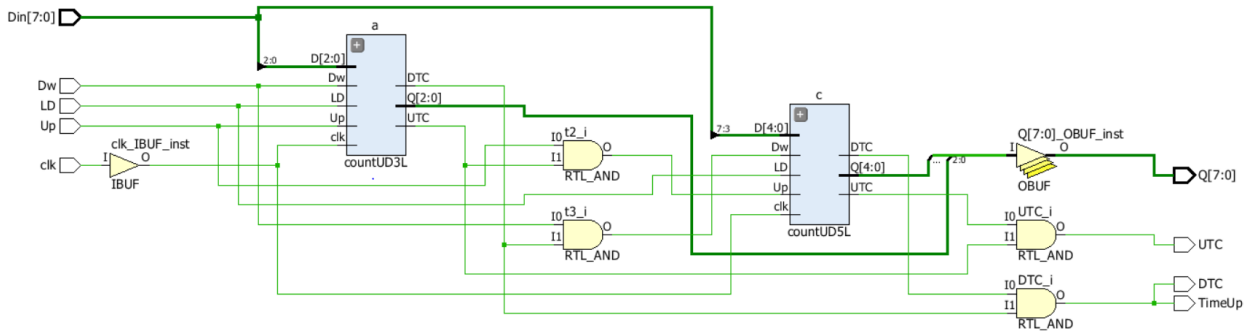
8. Uwin- this state is when player Dwin and is on for 1 clock cycle incrementing D and automatically going to Dflash the next clock cycle.
9. UFlash- this is the state after Uwin where player U's score flashes until btnC is pressed which then brings the state back to LedLoad.
10. DFlash- this is the state after Dwin where player D's score flashes until btnC is pressed which then brings the state back to LedLoad.
11. TFlash- this is the state after tie where both player's scores flashes until btnC is pressed which then brings the state back to LedLoad.

Conclusion

In conclusion, I learned how to build a state machine from flipflops which acted as a game of chicken. The lab first started with creating the smaller modules which were simple, or we already had. 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. 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.

Supplementary Material

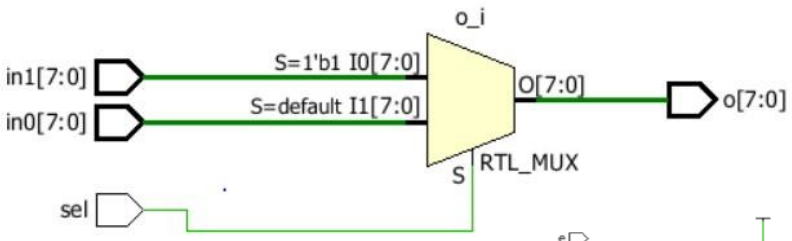
TimeCounter



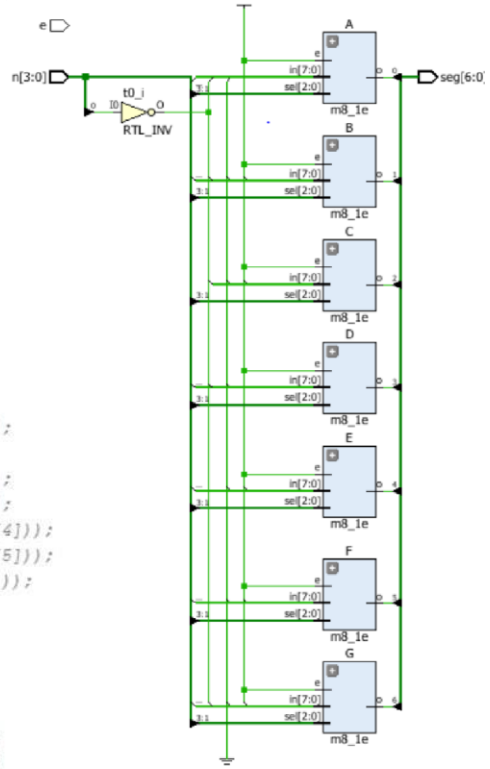

```
module Time_Counter(
    input Dw,
    input Up,
    input LD,
    input [7:0]Din,
    input clk,
    output TimeUp,
    output UTC,
    output DTC,
    output [7:0]Q
    );
    wire [1:0] tempUTC, tempDTC;
    wire t0,t1,t2,t3;
    countUD3L a(.clk(clk), .Up(Up), .Dw(Dw), .LD(LD), .D(Din[2:0]), .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
```

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
```



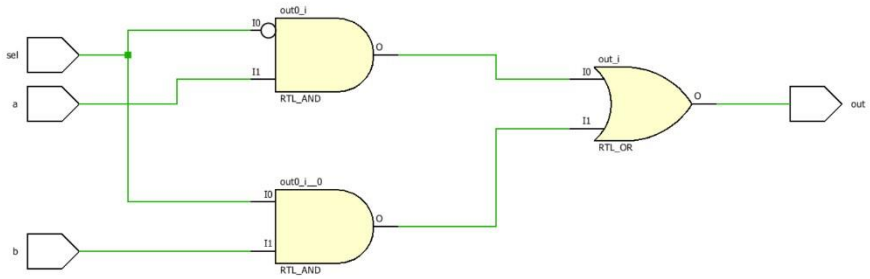
Hex7Seg



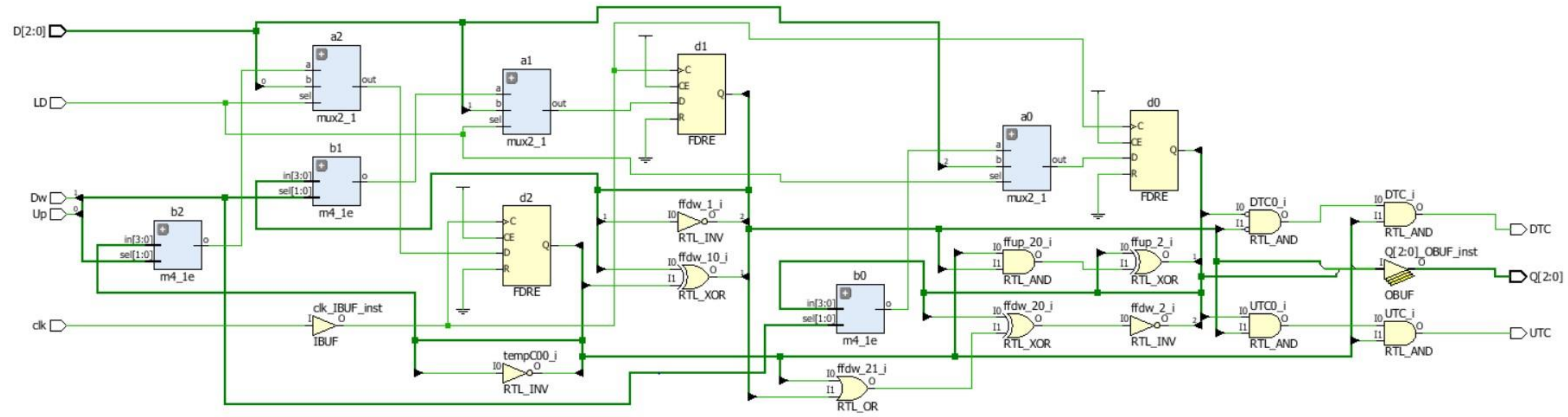

```
module hex7seg(
    input [3:0] n,
    input e,
    output [6:0] seg
    );

    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,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
```

Mux 2_1

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



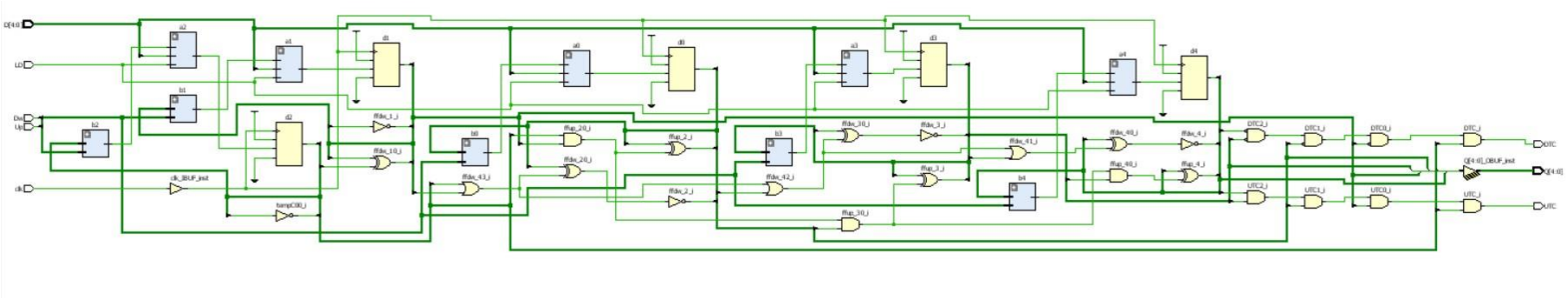
CountUD3L



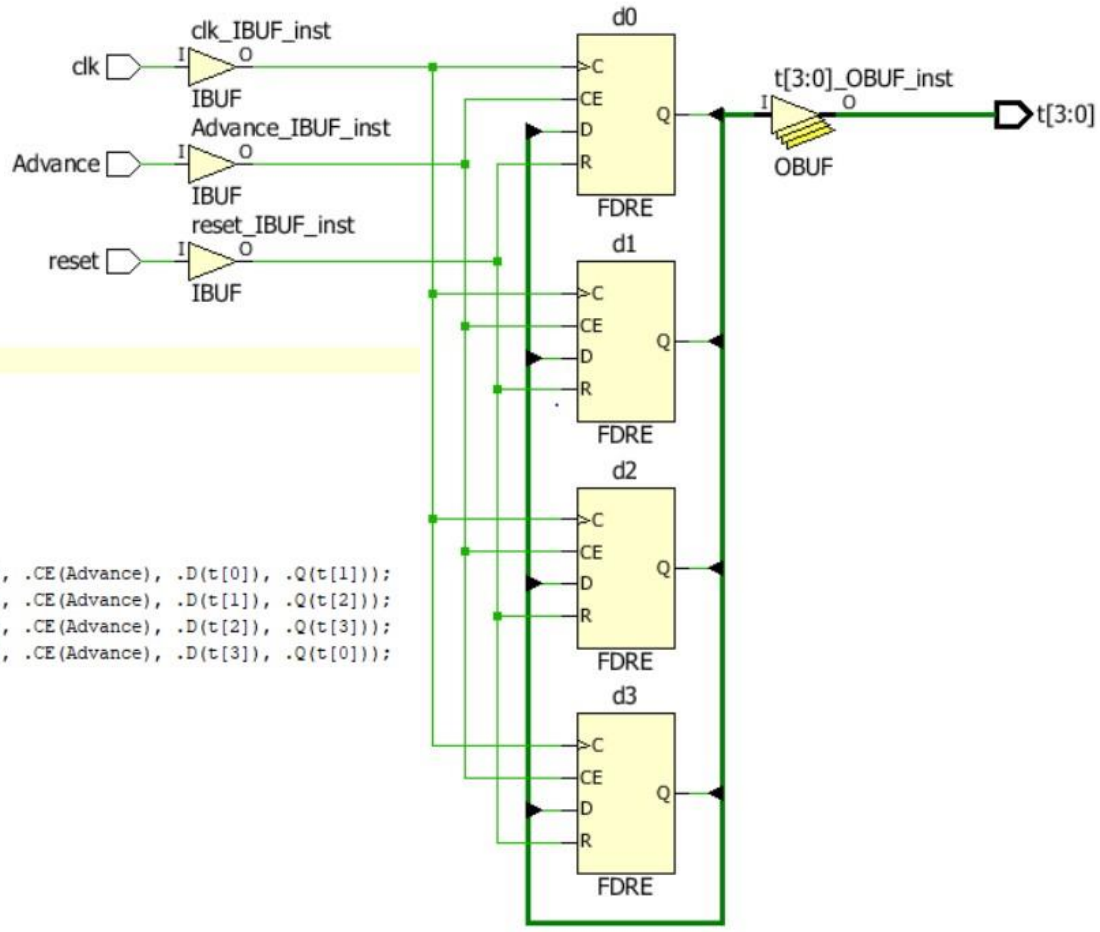

```
module countUD3L(
    input clk,
    input Up,
    input Dw,
    input LD,
    input [2:0] D,
    output [2:0] Q,
    output UTC,
    output DTC
    );

    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
```

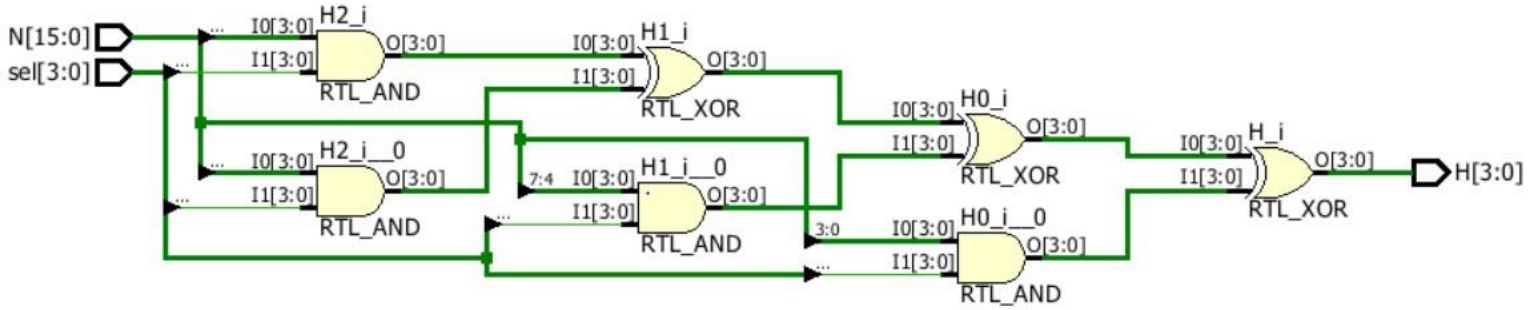


Ring Counter




```
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
```

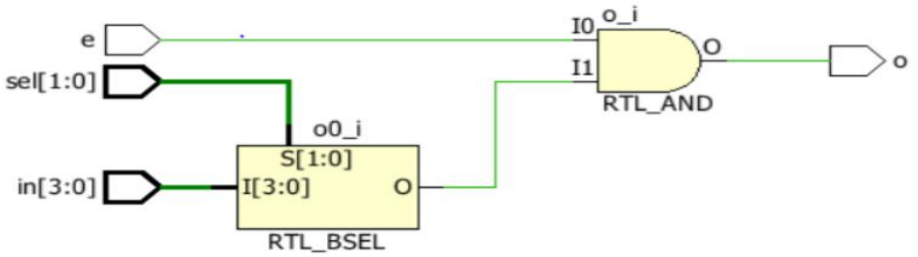
Selector



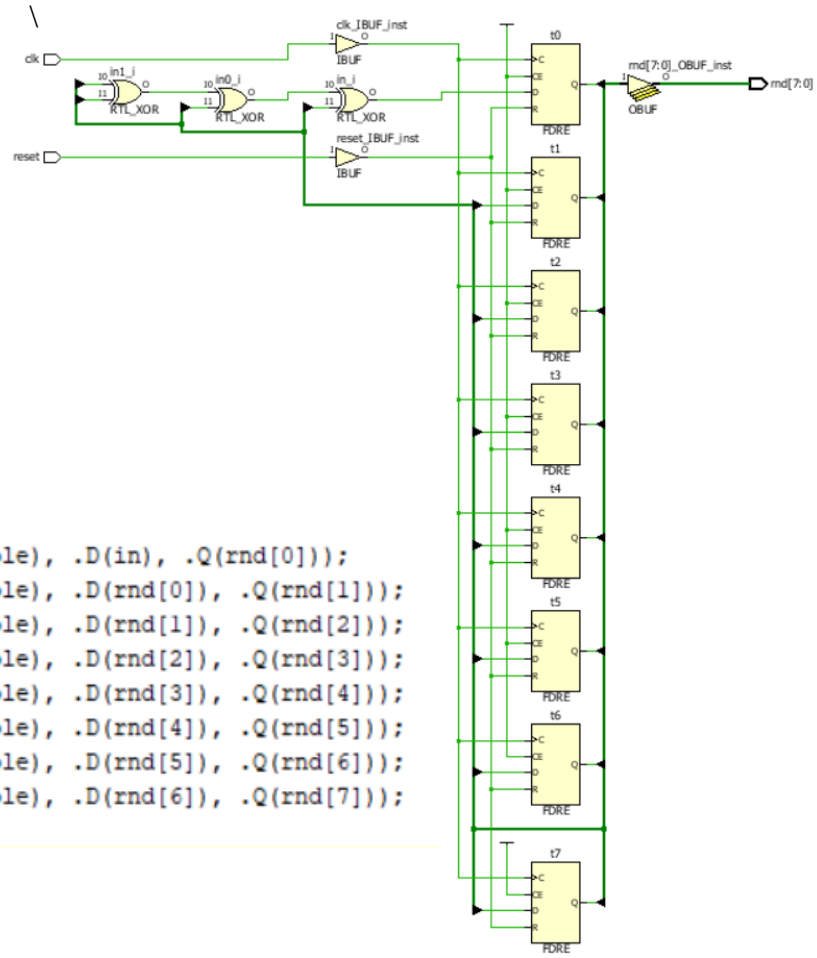

```
module Selector(
    input [3:0] sel,
    input [15:0] N,
    output [3:0] H
    );
      assign H = ((N[15:12]&{4{sel[3]}}) ^(N[11:8]&{4{sel[2]}})^(N[7:4]&{4{sel[1]}}) ^(N[3:0]&{4{sel[0]}}));
endmodule
```

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
```

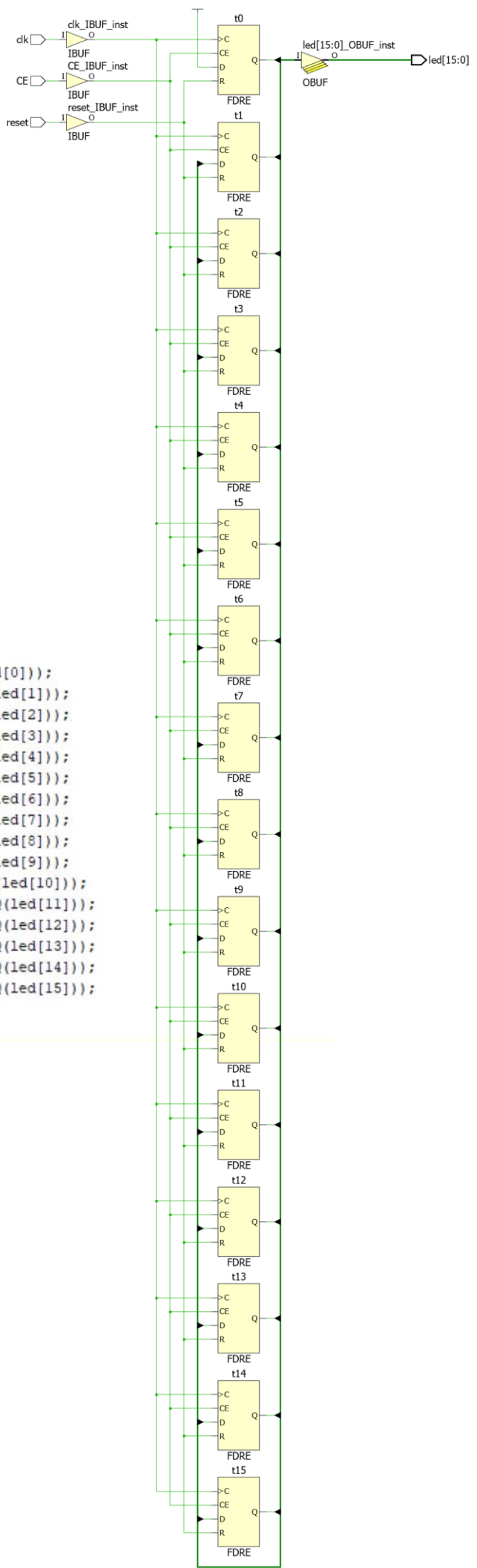


Random Number Generator



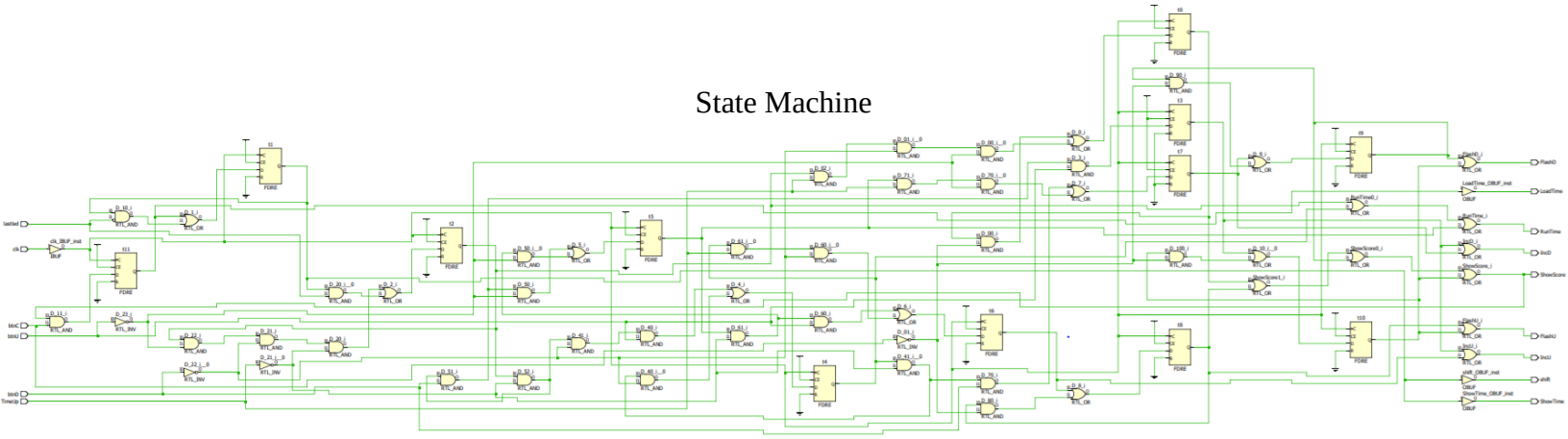

```
module rndNum(
    input clk,
    input reset,
    output [7:0]rnd
    );
    //wire [7:0]Q;

    wire in = rnd[0]^rnd[5]^rnd[6]^rnd[7];
    wire enable = 1'b1;
    FDRE #(.INIT(1'b1) ) t0(.C(clk), .R(reset), .CE(enable), .D(in), .Q(rnd[0]));
    FDRE #(.INIT(1'b0) ) t1(.C(clk), .R(reset), .CE(enable), .D(rnd[0]), .Q(rnd[1]));
    FDRE #(.INIT(1'b0) ) t2(.C(clk), .R(reset), .CE(enable), .D(rnd[1]), .Q(rnd[2]));
    FDRE #(.INIT(1'b0) ) t3(.C(clk), .R(reset), .CE(enable), .D(rnd[2]), .Q(rnd[3]));
    FDRE #(.INIT(1'b0) ) t4(.C(clk), .R(reset), .CE(enable), .D(rnd[3]), .Q(rnd[4]));
    FDRE #(.INIT(1'b0) ) t5(.C(clk), .R(reset), .CE(enable), .D(rnd[4]), .Q(rnd[5]));
    FDRE #(.INIT(1'b0) ) t6(.C(clk), .R(reset), .CE(enable), .D(rnd[5]), .Q(rnd[6]));
    FDRE #(.INIT(1'b0) ) t7(.C(clk), .R(reset), .CE(enable), .D(rnd[6]), .Q(rnd[7]));
endmodule
```

Led Shifter

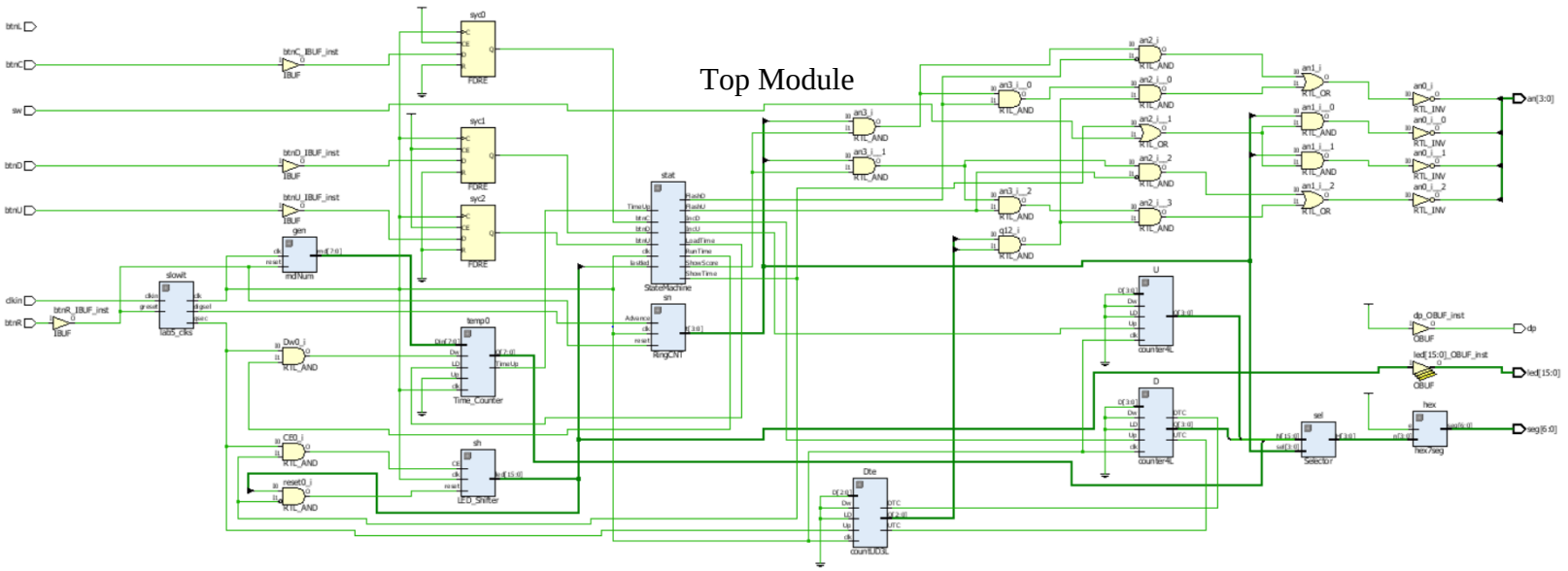



```
module LED_Shifter(
    input reset,
    input CE,
    input clk,
    output [15:0]led
    );
        wire enable = CE;
        FDRE #(.INIT(1'b0) ) t0(.C(clk), .R(reset), .CE(enable), .D(1'b1), .Q(led[0]));
        FDRE #(.INIT(1'b0) ) t1(.C(clk), .R(reset), .CE(enable), .D(led[0]), .Q(led[1]));
        FDRE #(.INIT(1'b0) ) t2(.C(clk), .R(reset), .CE(enable), .D(led[1]), .Q(led[2]));
        FDRE #(.INIT(1'b0) ) t3(.C(clk), .R(reset), .CE(enable), .D(led[2]), .Q(led[3]));
        FDRE #(.INIT(1'b0) ) t4(.C(clk), .R(reset), .CE(enable), .D(led[3]), .Q(led[4]));
        FDRE #(.INIT(1'b0) ) t5(.C(clk), .R(reset), .CE(enable), .D(led[4]), .Q(led[5]));
        FDRE #(.INIT(1'b0) ) t6(.C(clk), .R(reset), .CE(enable), .D(led[5]), .Q(led[6]));
        FDRE #(.INIT(1'b0) ) t7(.C(clk), .R(reset), .CE(enable), .D(led[6]), .Q(led[7]));
        FDRE #(.INIT(1'b0) ) t8(.C(clk), .R(reset), .CE(enable), .D(led[7]), .Q(led[8]));
        FDRE #(.INIT(1'b0) ) t9(.C(clk), .R(reset), .CE(enable), .D(led[8]), .Q(led[9]));
        FDRE #(.INIT(1'b0) ) t10(.C(clk), .R(reset), .CE(enable), .D(led[9]), .Q(led[10]));
        FDRE #(.INIT(1'b0) ) t11(.C(clk), .R(reset), .CE(enable), .D(led[10]), .Q(led[11]));
        FDRE #(.INIT(1'b0) ) t12(.C(clk), .R(reset), .CE(enable), .D(led[11]), .Q(led[12]));
        FDRE #(.INIT(1'b0) ) t13(.C(clk), .R(reset), .CE(enable), .D(led[12]), .Q(led[13]));
        FDRE #(.INIT(1'b0) ) t14(.C(clk), .R(reset), .CE(enable), .D(led[13]), .Q(led[14]));
        FDRE #(.INIT(1'b0) ) t15(.C(clk), .R(reset), .CE(enable), .D(led[14]), .Q(led[15]));

endmodule
```

State Machine

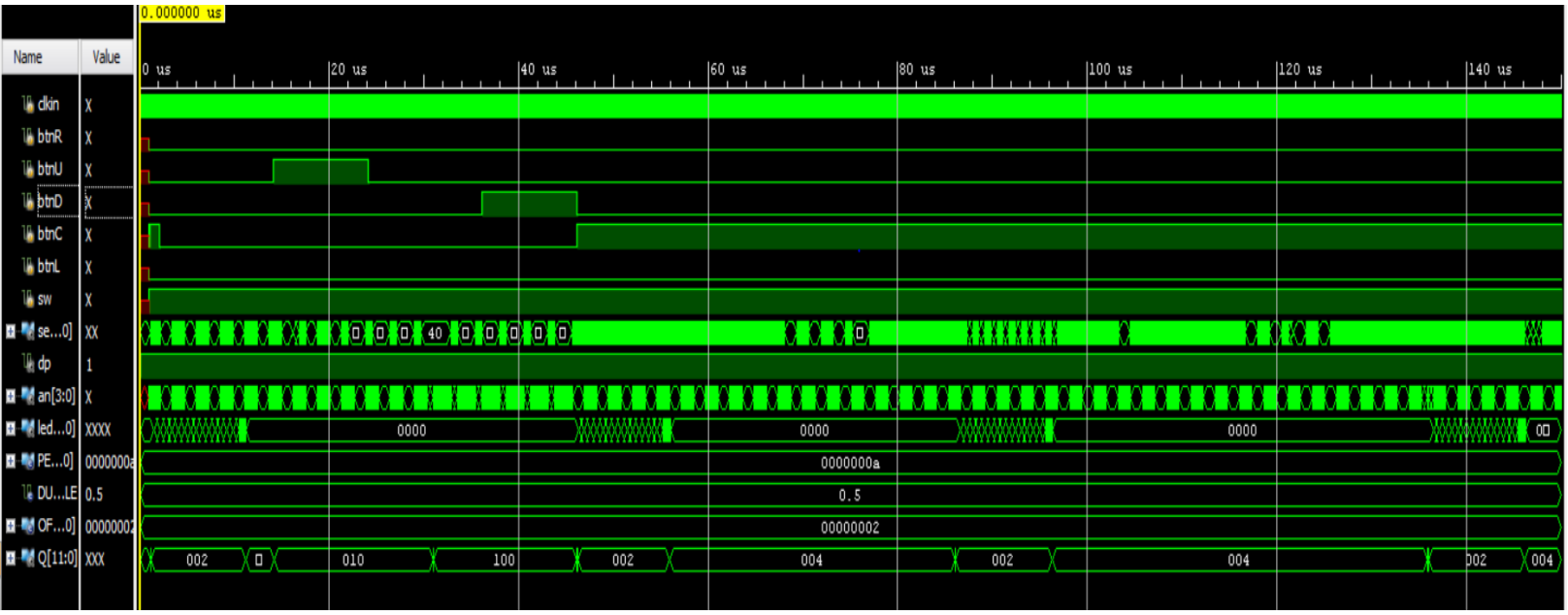


```
module StateMachine(
    input clk,
    input btnC,
    input btnU,
    input btnD,
    input TimeUp,
    input lastled,
    output ShowTime,
    output LoadTime,
    output RunTime,
    output IncU,
    output IncD,
    output ShowScore,
    output FlashU,
    output FlashD,
    output shift
    );
    wire[11:0] D, Q;
    wire enable = 1'b1;
    wire btnCsyn,btnDsyn,btnUsyn;
    assign btnCsyn = btnC;
    assign btnDsyn = btnD;
    assign btnUsyn = btnU;

//    FDRE #(.INIT(1'b0)) syc0 (.C(clk), .CE(enable), .D(btnC), .Q(btnCsyn
//    FDRE #(.INIT(1'b0)) syc1 (.C(clk), .CE(enable), .D(btnD), .Q(btnDsyn
//    FDRE #(.INIT(1'b0)) syc2 (.C(clk), .CE(enable), .D(btnU), .Q(btnUsyn


    //Idle
    assign D[0] = Q[0]&~btnCsyn | Q[2]&TimeUp&~btnDsyn&~btnUsyn;
    FDRE #(.INIT(1'b1)) t0 (.C(clk), .CE(enable), .D(D[0]), .Q(Q[0]));

    //wait
    assign D[1] = Q[11] | Q[1]&~lastled;
    FDRE #(.INIT(1'b0)) t1 (.C(clk), .CE(enable), .D(D[1]), .Q(Q[1]));

    //Accp
    assign D[2] = Q[2]&~btnUsyn&~btnDsyn&~TimeUp | Q[1]&lastled;
    FDRE #(.INIT(1'b0)) t2 (.C(clk), .CE(enable), .D(D[2]), .Q(Q[2]));
    //TIE
    assign D[3] =  Q[2]&~TimeUp&btnDsyn&btnUsyn;
    FDRE #(.INIT(1'b0)) t3 (.C(clk), .CE(enable), .D(D[3]), .Q(Q[3]));
    //UFirst
    assign D[4] =  Q[2]&~TimeUp&~btnDsyn&btnUsyn | Q[4]&~TimeUp&~btnDsyn;
    FDRE #(.INIT(1'b0)) t4 (.C(clk), .CE(enable), .D(D[4]), .Q(Q[4]));

    //Dfirst
    assign D[5] =  Q[2]&~TimeUp&btnDsyn&~btnUsyn | Q[5]&~TimeUp&~btnUsyn;
    FDRE #(.INIT(1'b0)) t5 (.C(clk), .CE(enable), .D(D[5]), .Q(Q[5]));

    //Uwin
    assign D[6] =  Q[5]&~TimeUp&btnUsyn | Q[4]&TimeUp&~btnDsyn;
    FDRE #(.INIT(1'b0)) t6 (.C(clk), .CE(enable), .D(D[6]), .Q(Q[6]));

    //Dwin
    assign D[7] =  Q[4]&~TimeUp&btnDsyn | Q[5]&TimeUp&~btnUsyn;
    FDRE #(.INIT(1'b0)) t7 (.C(clk), .CE(enable), .D(D[7]), .Q(Q[7]));

    //UFlash
    assign D[8] =  Q[6] | Q[8]&~btnCsyn;
    FDRE #(.INIT(1'b0)) t8 (.C(clk), .CE(enable), .D(D[8]), .Q(Q[8]));

    //Dflash
    assign D[9] =  Q[7] | Q[9]&~btnCsyn;
    FDRE #(.INIT(1'b0)) t9 (.C(clk), .CE(enable), .D(D[9]), .Q(Q[9]));
    //Tflash
    assign D[10] =  Q[3] | Q[10]&~btnCsyn;
    FDRE #(.INIT(1'b0)) t10 (.C(clk), .CE(enable), .D(D[10]), .Q(Q[10]));
     //Load
       assign D[11] = (Q[0]|Q[8]|Q[9]|Q[10])&btnCsyn ;
       FDRE #(.INIT(1'b0)) t11 (.C(clk), .CE(enable), .D(D[11]), .Q(Q[11]));

    assign ShowTime = Q[1];
    assign LoadTime = Q[11];
    assign RunTime = Q[2]| Q[4]| Q[5];
    assign IncU = Q[3]| Q[6];
    assign IncD = Q[3]| Q[7];
    assign ShowScore = Q[0]| Q[8]| Q[9]| Q[10];
    assign FlashU = Q[8]| Q[10];
    assign FlashD =  Q[9]| Q[10];
    assign shift = Q[1];

endmodule
```

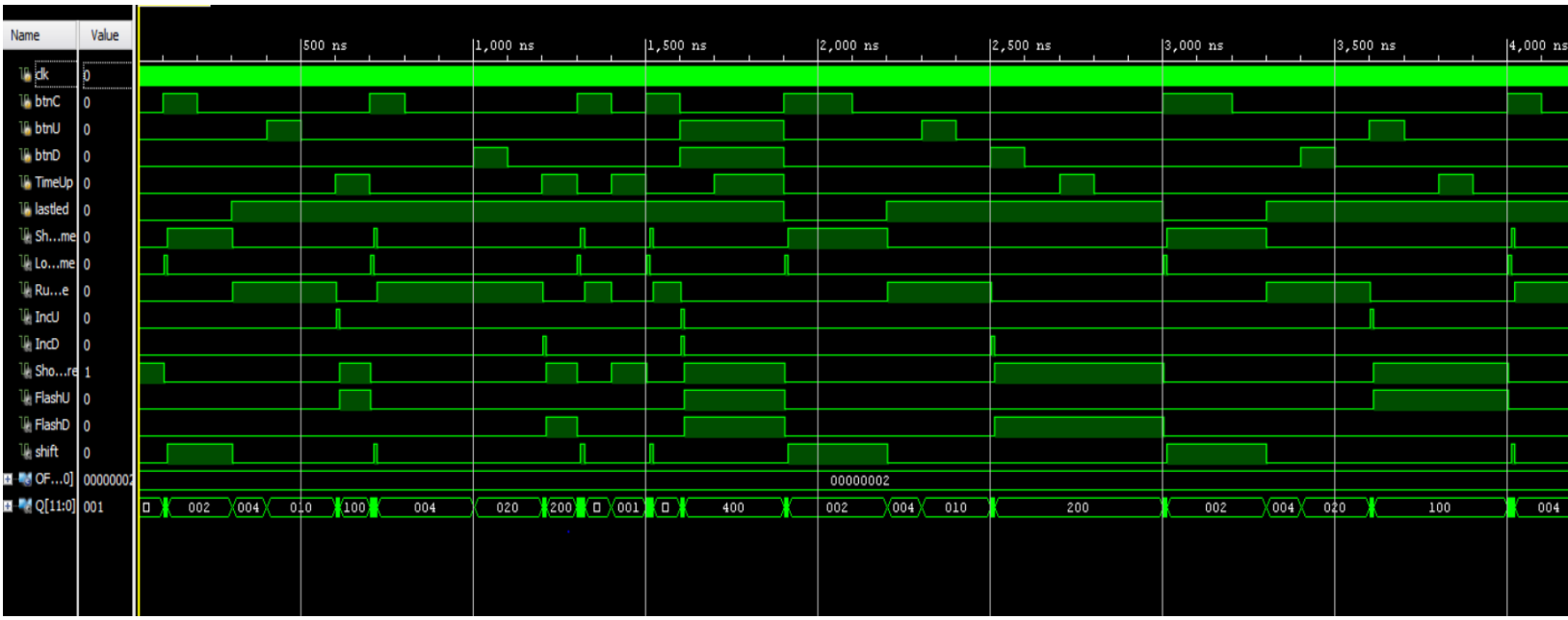



```
module TopMod5(
    input clkin,
    input btnR,
    input btnU,
    input btnC,
    input btnL,
    input btnD,
    input sw,
    output [15:0] led,
    output [6:0]seg,
    output [3:0]an,
    output dp
    );
   wire clk, digsel, qsec;
   wire TimeUp, lastled, ShowTime, LoadTime, RunTime, IncU, IncD, ShowScore, FlashU, FlashD, shift;
   wire [7:0] timerIn, timerOut;
   wire [3:0] ScoreU, ScoreD, select,dam;
   wire [15:0] pick, tempLed;
   lab5_clks slowit (.clkin(clkin), .greset(btnR), .clk(clk), .digsel(digsel), .qsec(qsec));


    wire btnCsyn,btnDsyn,btnUsyn;


      FDRE #(.INIT(1'b0)) syc0 (.C(clk), .CE(1'b1), .D(btnC), .Q(btnCsyn));
      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(btnU), .Q(btnUsyn));


   LED_Shifter sh(.reset(lastled & ~ShowTime),.CE(qsec & ShowTime),.clk(clk),.led(led));
  // assign led = tempLed & {16{shift}};
   assign lastled = led[15] ;

   rndNum gen(.clk(clk),.reset(btnR),.rnd(timerIn));
   wire [7:0]tempIn  = timerIn;
   assign tempIn[8] = 1'b0;

   Time_Counter temp0(.Dw(qsec&RunTime),.Up(1'b0),.LD(LoadTime),.Din(tempIn),.clk(clk),.TimeUp(TimeUp),.Q(timerOut));

   wire teme0, teme1, teme2, teme3;

   counter4L U(.Up(IncU),.Dw(1'b0),.clk(clk),.LD(1'b0),.D({1'b0,1'b0,1'b0,1'b0}),.Q(ScoreU),.UTC(teme0),.DTC(teme1));
   counter4L D(.Up(IncD),.Dw(1'b0),.clk(clk),.LD(1'b0),.D({1'b0,1'b0,1'b0,1'b0}),.Q(ScoreD),.UTC(teme2),.DTC(teme3));
    StateMachine stat(.clk(clk),.btnC(btnCsyn),.btnU(btnUsyn),.btnD(btnDsyn),.TimeUp(TimeUp),.LoadTime(LoadTime),.RunTime(RunTime),
         .lastled(lastled),.ShowTime(ShowTime),.IncU(IncU),.IncD(IncD),.ShowScore(ShowScore),
         .FlashU(FlashU),.FlashD(FlashD),.shift(shift));

   RingCNT sn(.clk(clk),.Advance(digsel),.reset(btnR),.t(select));

   assign pick[3:0] = ScoreD;
   assign pick[11:4] = timerOut;
   assign pick[15:12] = ScoreU;

   Selector sel(.sel(select),.N(pick),.H(dam));

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

     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 ql2 = temp3[0] & temp3[1];
     assign an[0] = ~((select[0]&ShowScore)&~FlashD  |   (select[0]&ShowScore)&FlashD&ql2) ;

     assign an[1] = ~((select[1]&(ShowTime | sw)));
     assign an[2] = ~((select[2]&(ShowTime | sw)));

     assign an[3] =  ~((select[3]&ShowScore)&~FlashU | (select[3]&ShowScore)&FlashU&ql2) ;

    assign dp = 1'b1;

endmodule
```

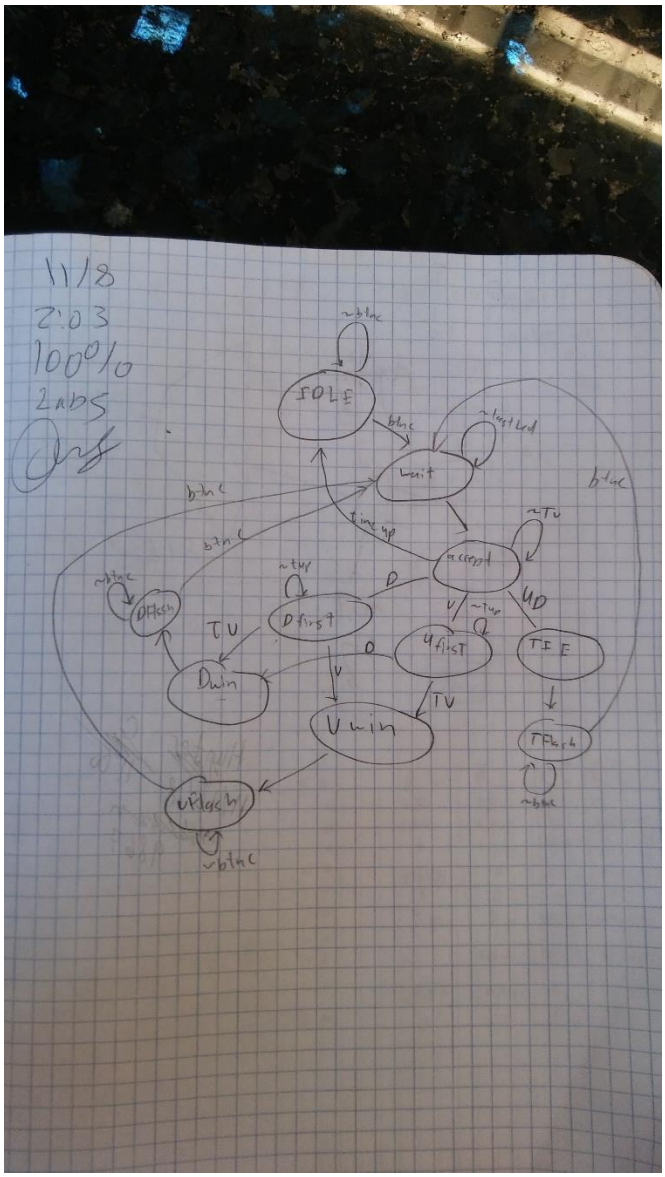
TopMod Sim



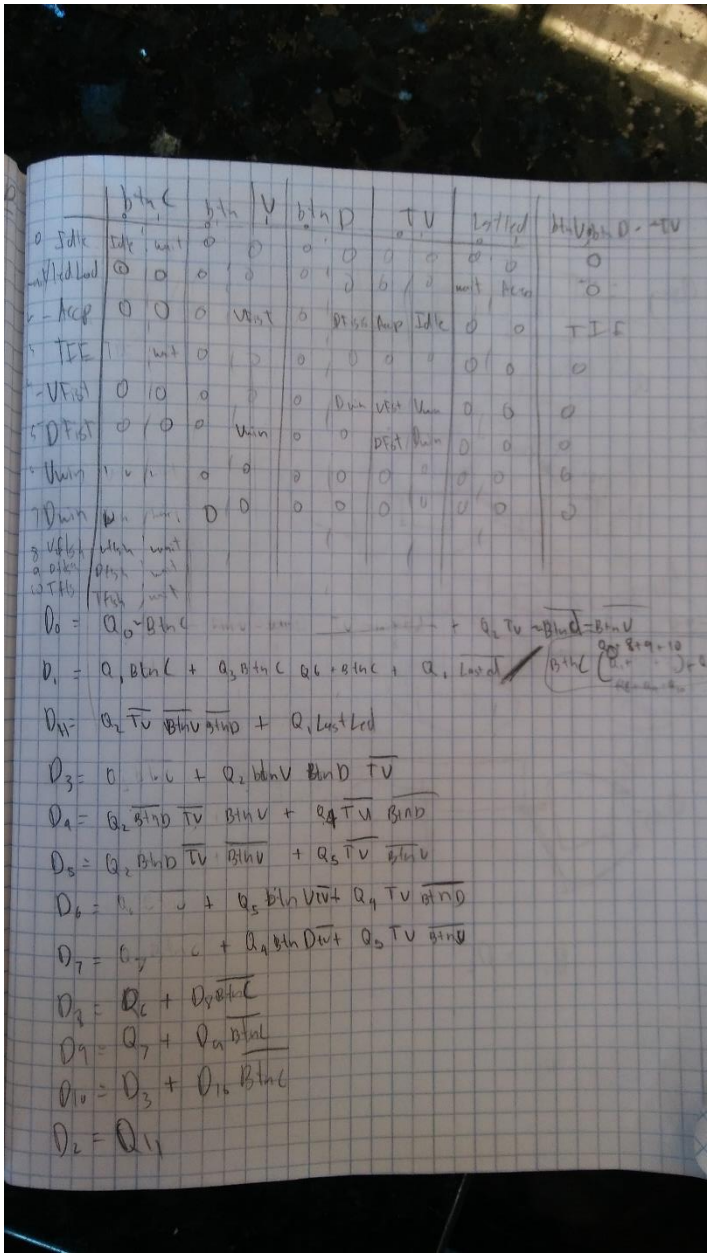
State Machine Sim



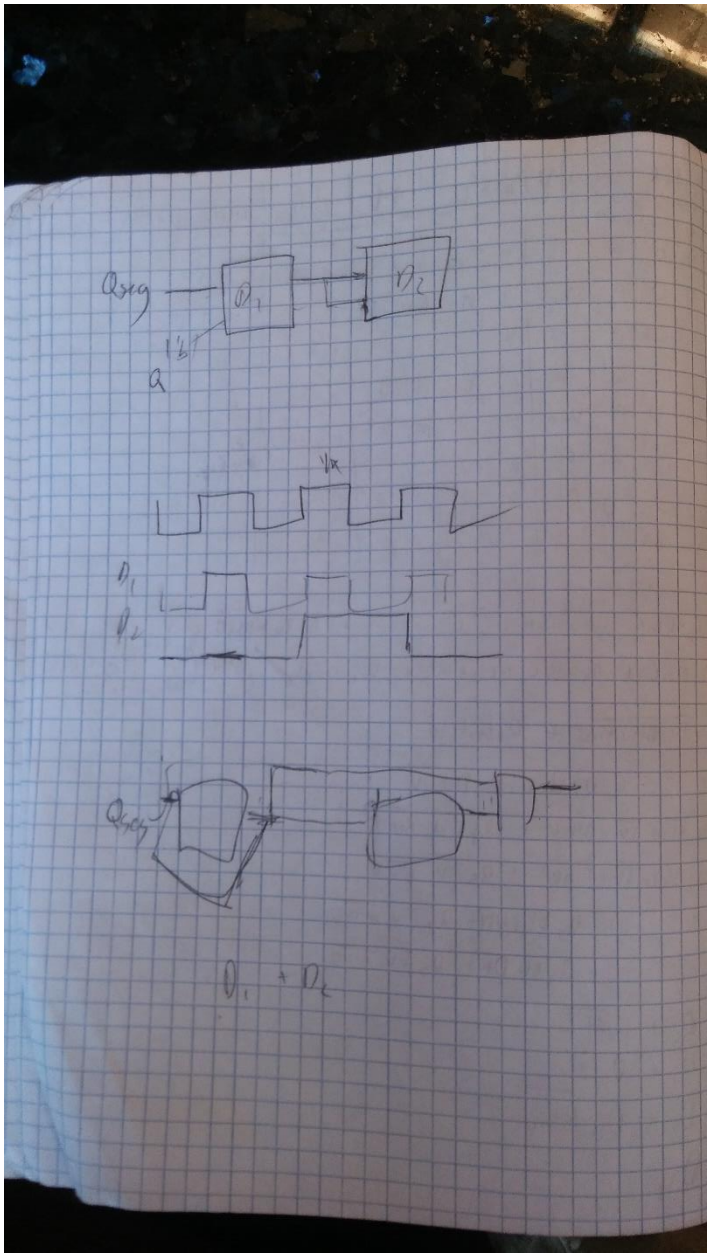
Picture 5



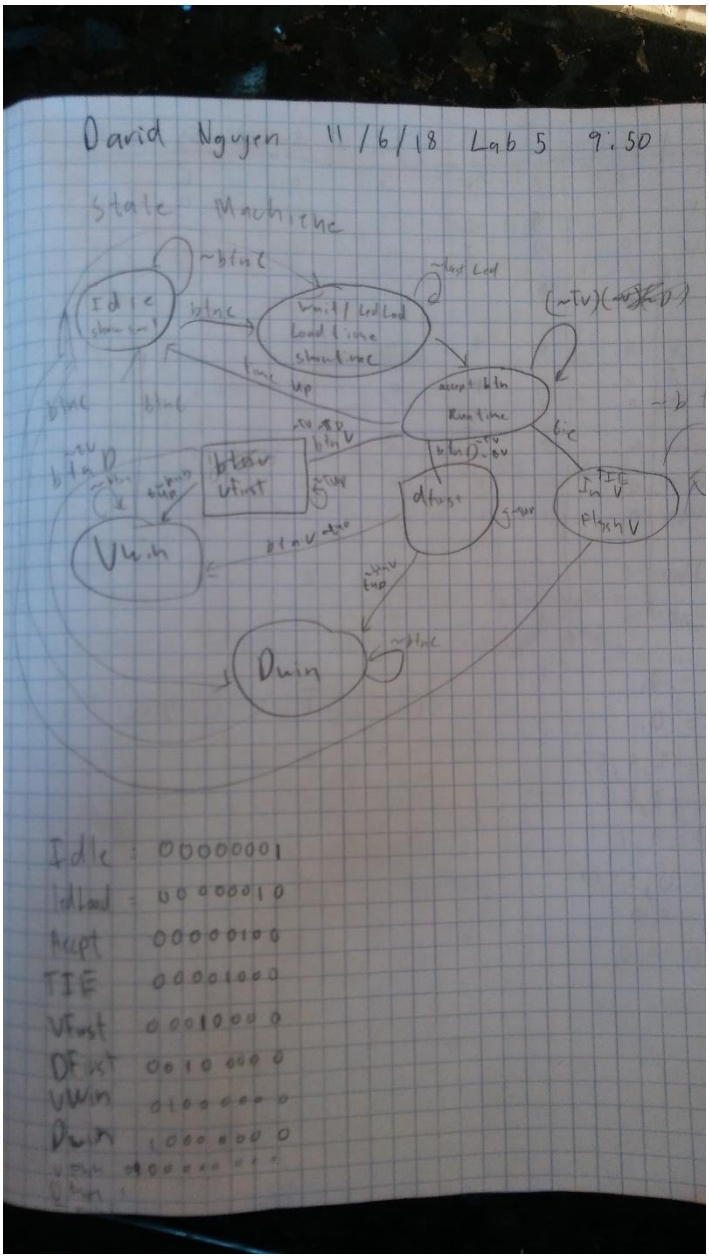
Picture 3



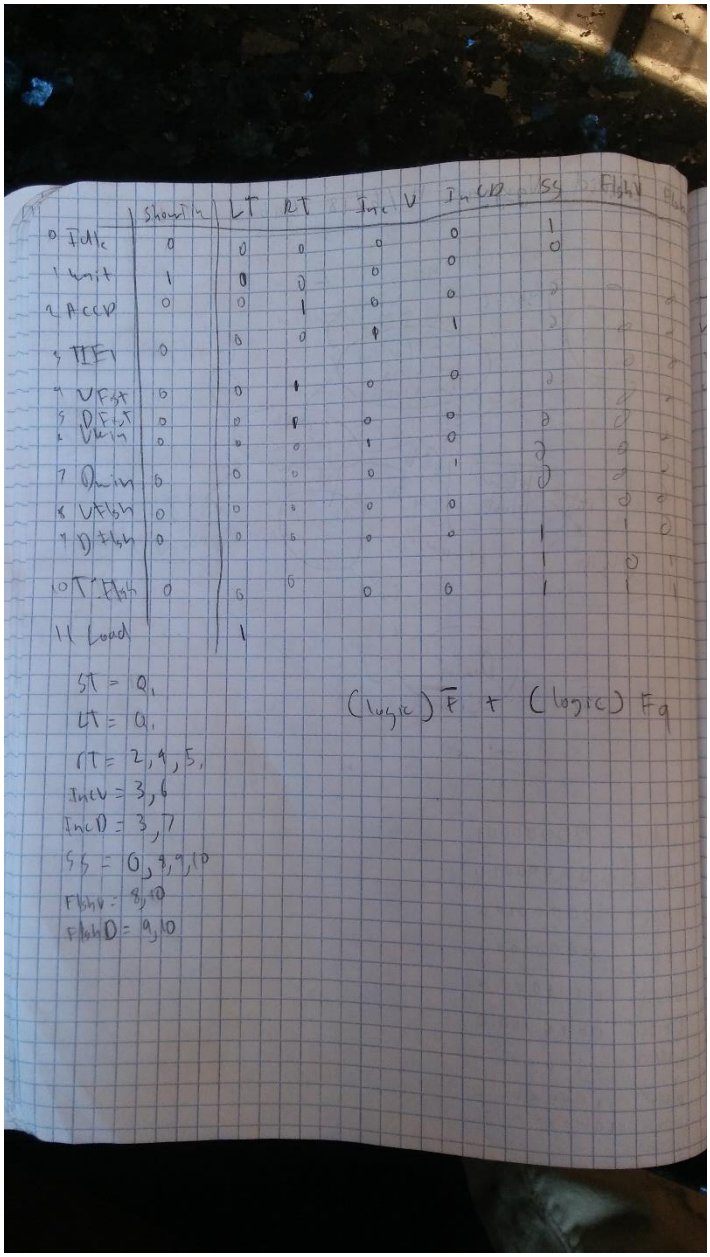
Picture 4



Picture 1



Picture 2


Shown
LT
RT
IncV
IncD
SS
FlshV
FlshD
0 Idle
1 Wait
2 Accp
3 TEF
4 VFast
5 DFast
6 VMain
7 DMain
8 VFlsh
9 DFlsh
10 T. Flsh
11 Load
ST = Q1
LT = Q1
RT = 2, 4, 5,
IncV = 3, 6
IncD = 3, 7
SS = 0, 8, 9, 10
FlshV = 8, 10
FlshD = 9, 10
(logic) F̄ + (logic) Fq