CSE100 Lab Report 5

Time trial with state machines

Student Name: Artyom Martirosyan

Lab Sec: 1B

Date: 5/15/2020

Description:

For this assignment the students will be essentially creating a time trial game by creating and implementing a state machine. The student will also be manipulating several LEDs as well as a 8-bit counter and a 4-bit counter. Fortunately, for the students who have already completed the previous lab they will be capable of simply reusing several components as well as creating a few new ones. The main premise of the game is that there are two players: switches 0 and 15 who will click btnU to start the round. Once the round has been started the players will then need to wait until the timer has finished and flip their switches accordingly: if the time is up and one player flips the switch before the other that player wins a point, if the timer is up and both players flip the switch at the same time then both players will be awarded a point. However, if one of the players flips their switch before the timer is up(when time runs out leds 7 and 8 will be lit up) then the other player will be awarded the point, if both players flip their switches before the time is up at the same time then neither player will be awarded a point. In order to do this the student will also need to have a ring counter and a selector due to the fact that the analogue display can only display one value at a time, but if we alternate which value is being displayed as a really fast rate then the values being depicted will seem transparent(there is also a cheat switch which will display the timer(sw3) as well as tying leds 0 and 15 to sw 0 and 15). The user will also need to develop a seven segment hex display converter in order for the values to be properly displayed. For the timer, the clock must count down from an 8 bit value, but rather than creating another module which will hold two 4-bit counters we can simply create a 16 bit counter and ignore the 8 most significant bits of the counter. For the state machine the user must make sure to develop and test their module in order to make sure that it is operating as expected and that each stage of their state machine. The user will also need to synchronise all of the different inputs to the clk output from the provided module.

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:

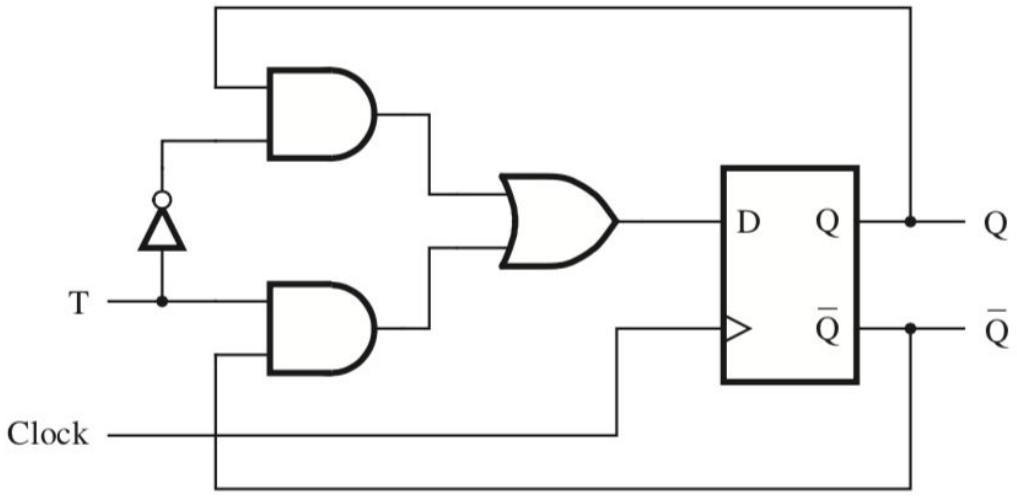


*Figure 1 is a schematic of a T flip-flop provided by the textbook

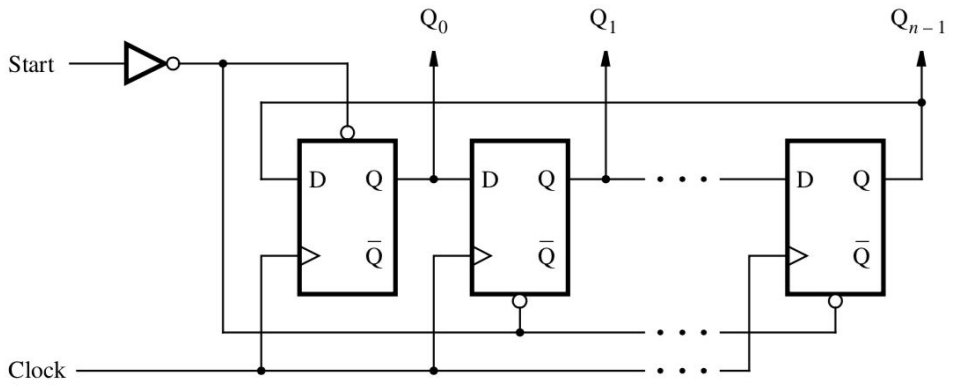
From this schematic the student should be able to create a T-flip flop

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:



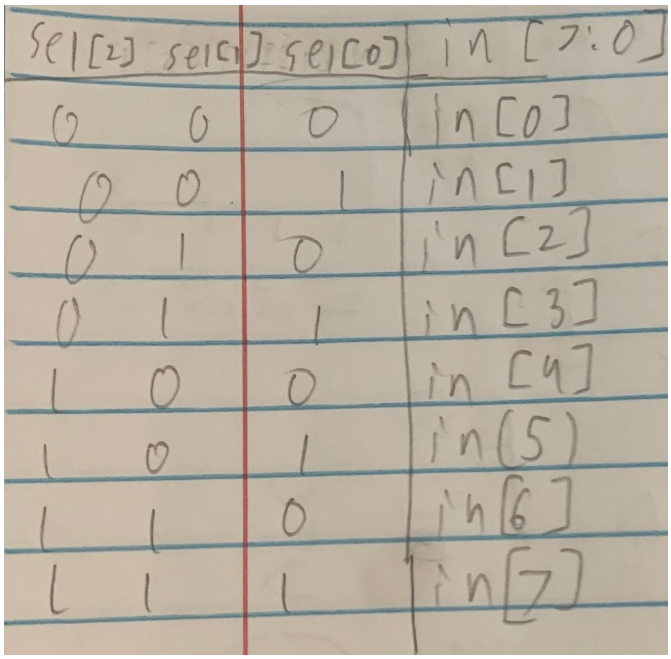
*The figure above(figure 2) is a schematic of an n-bit ring counter provided from the textbook.

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:

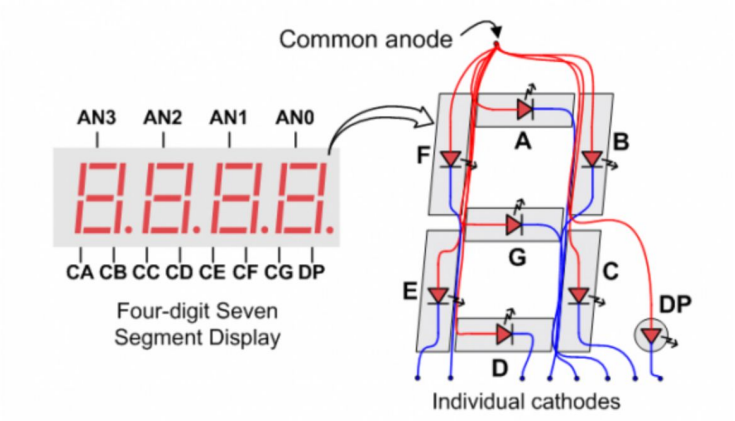
*The figure below(fig 3) is a truth table for m8_le taken from lab manual 3



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:

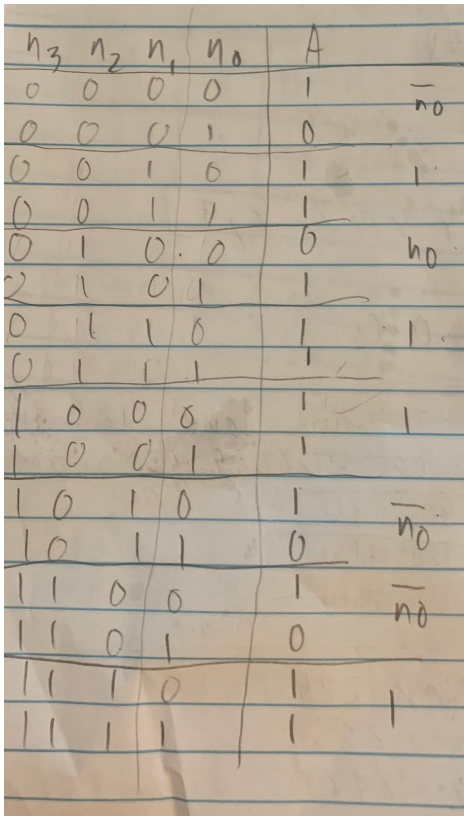


*Figure 5 to the left is a truth table of what A from the display above will represent according to the 4 bit vector it is provided.

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:

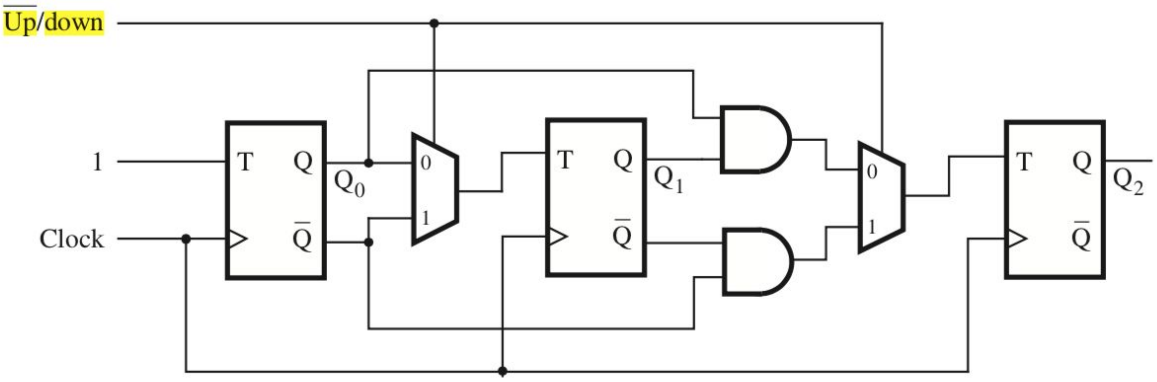
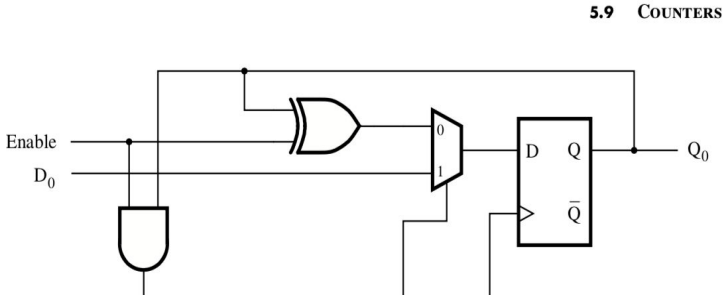


*Figure 6 above is an up/down 3-bit counter without load capabilities

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.

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:

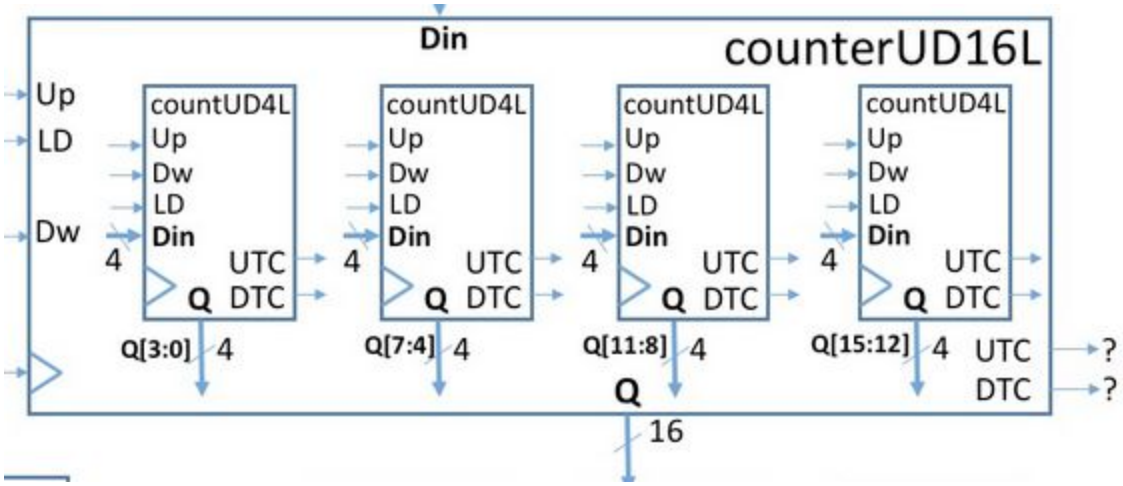


*Figure 8 above is a basic schematic of the 16 bit up/down counter taken from the lab four manual

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.

Random Number Generator:

For this lab each time a round is started the user will need to wait until the timer has finished and the leds have been lit up. However, in order to prevent the players from having an advantage by knowing how long the countdown is, time will have different random times making it impossible for the player to be able to predict when the led will turn on. Below is a schematic of what the random number generator will look like:

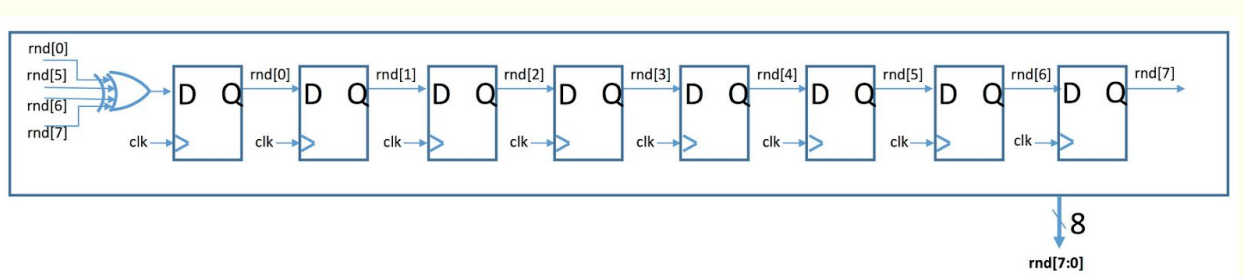


*figure 9 above is the schematic for the random number generator
Since we cannot actually develop truly random numbers we will rather be creating a number using several d-flip flops. By doing this the number will always be different hence making it 'random'. Another note is that the first flip flop must be initialized to 1 since there is no input into this module as well as the fact that we will be setting the two most significant bits to zero in order to make sure that the time isn't too significant as we dont want the players to be waiting several minutes in order to play each round. Though there is no input the generator will always have an eight bit output.

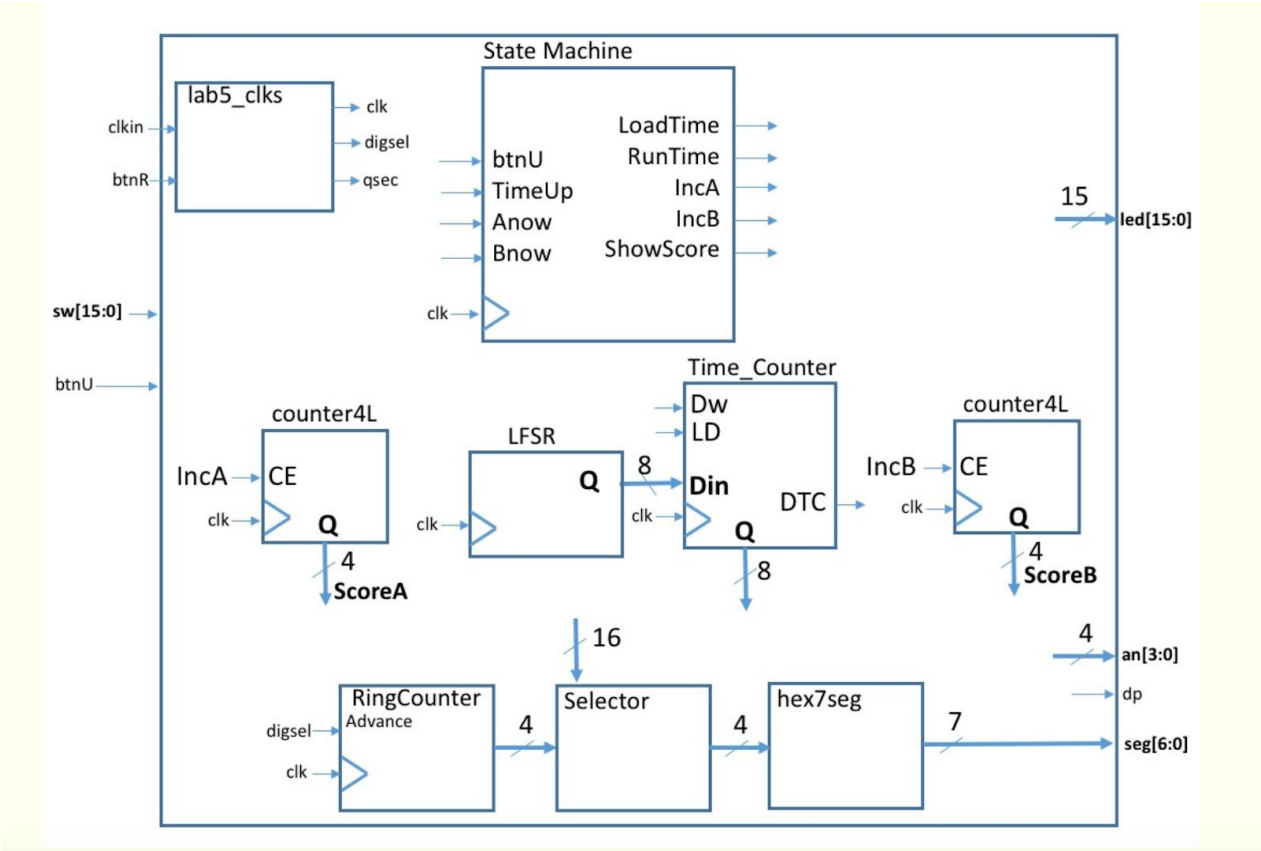
State machine:

For this lab we will also be creating a state machine which will be used to actually run the game. In order to do this we will need to consider several things, starting off with the inputs and outputs as the state machine will be taking btnU, timeup, Anow, Bnow, and the clock. The outputs will be LoadTime, RunTime, IncA, IncB, Showscore, light. From here we will need to keep several things in mind one being that there can only be one stage on at a time meaning that we need to make sure that each state is independent and cannot be combined with other states. Note: the actual logic as well as a schematic will be provided on the design section.

Top Module:

For the top module the user will be taking in several inputs clkin, btnU(starts the rounds), btnR, and sw[15:0](we will only be using switches 0 and 15 and sw 3 for cheat). We will attach the input switches and btnU to the state machine, and the state machine will pretty much control the rest of the modules other than the ring counter, selector, and hex 7 seg which will constantly be alternating showing the score for players a and b as well as the time(if sw 3 is set to high) below is the schematic provided by the original lab manual:

*figure 10 below is an overview of the schematic of out top module for this assignment(provided by the instructors in the lab 5 assignment sheet)



Results:

There weren't many issues implementing the top module, the only issue which originally occurred was related to the hex 7 segment display which was implemented incorrectly meaning that the actual logic was functioning properly, but there was nothing actually being displayed. For the State Machine there were no errors when testing.

Design:

Top schematic:

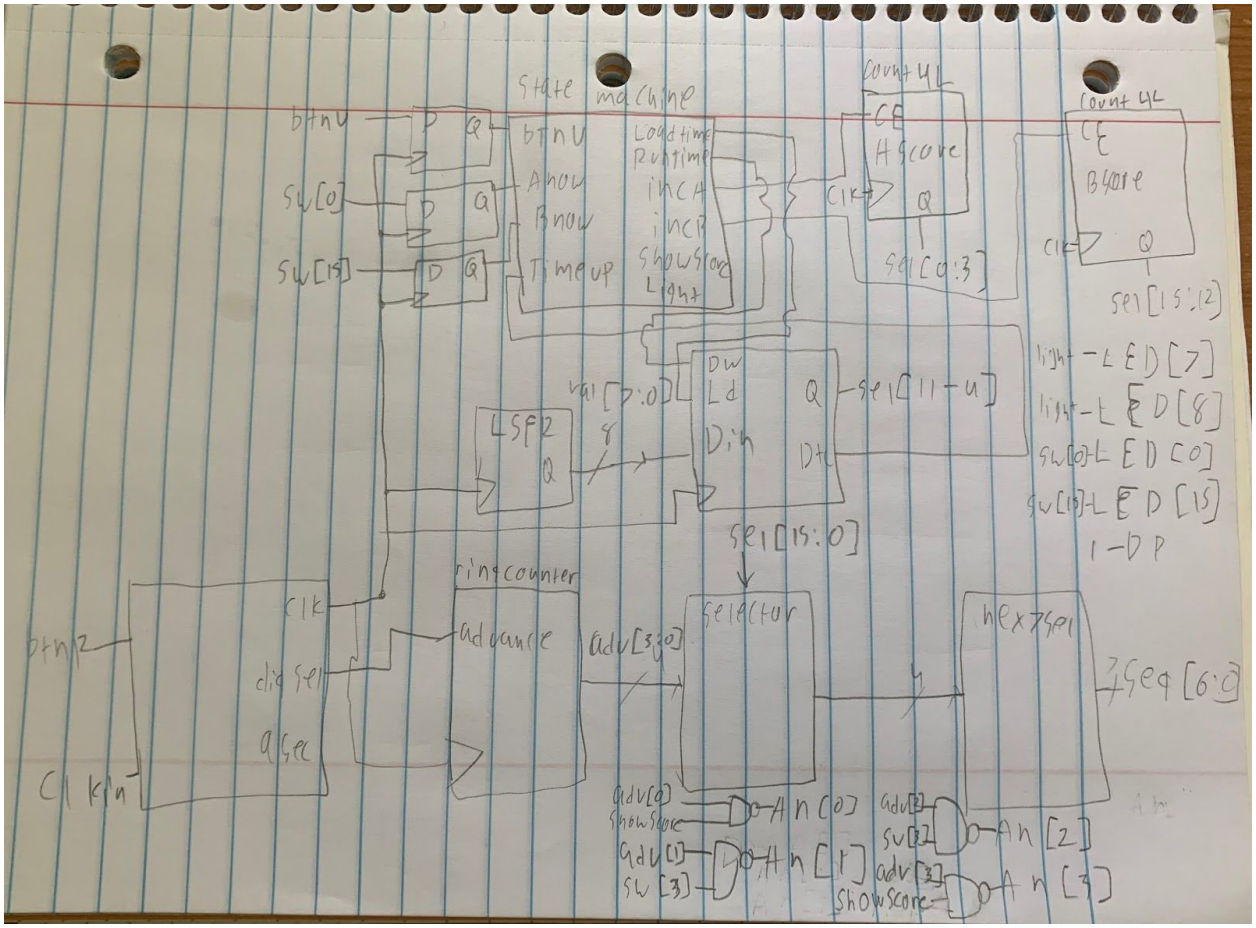


*figure 11 above is a schematic of the top module for Lab 5

The state machine will be outputting when the 16bit counter will start counting down as well as when it loads the new value to start counting down. It will also be deciding who will be winning each round by incrementing the appropriate count4L for A or B. the Analogue devices will only be displaying in certain conditions: for the players scores it will only show their scores when the ring counter has selected to display that value and in between rounds(this state is decided inside of the state machine). In order to display the time sw3 must be active as well as that analogue being selected by the ring counter. Another detail to keep in mind is that the timer will only be counting down 8-bit values, rather than creating a new module which would only run on eight bits we can simply take the 16-bit counter and pad the two most significant bits with zeros hence making an eight bit counter.

LFSR:

For the left shift register or random number generator the user will simply be creating attaching eight d-flip flops and ignoring the two most significant bits as shown below:

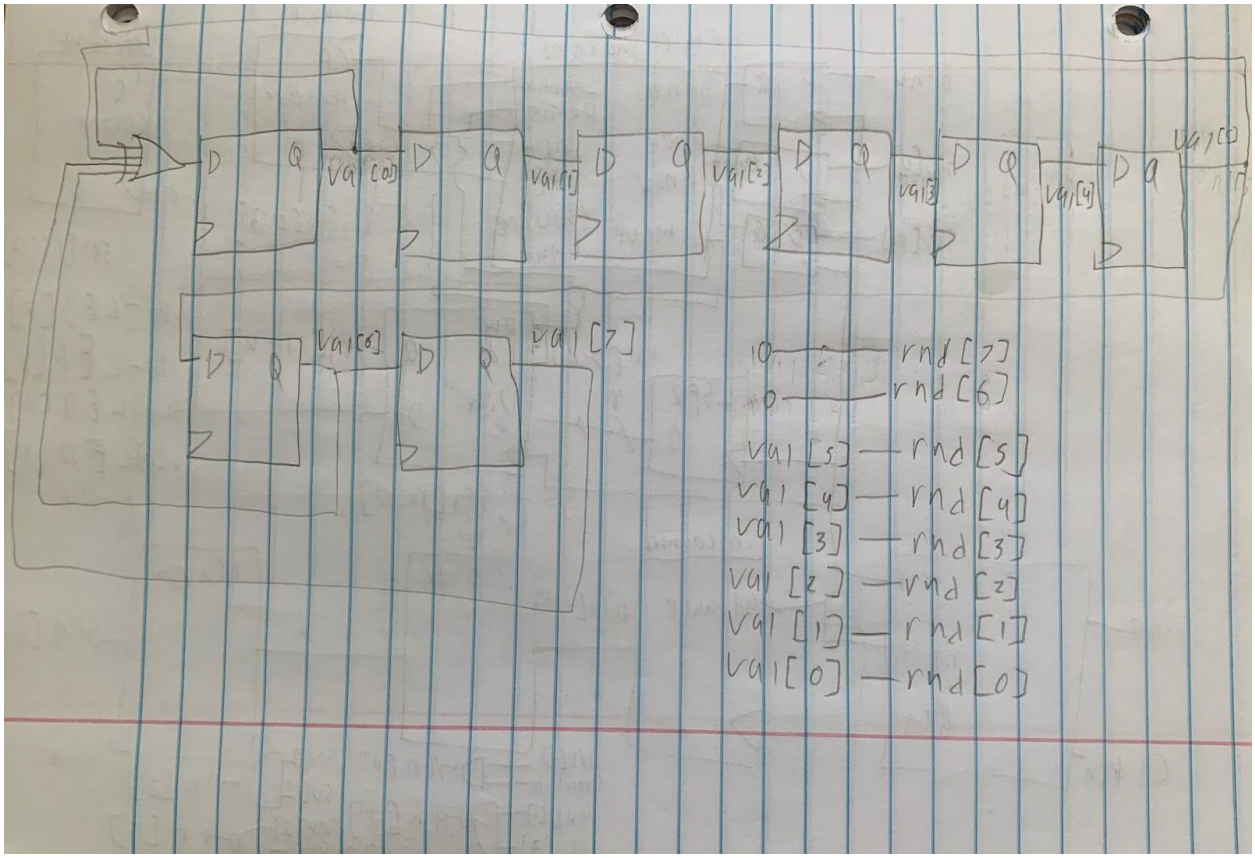


*figure 12 is the schematic for the random number generator

You must also remember to initialize the first flip flop to one and the rest to zero due to that fact that there are no inputs to the module.

State Machine:

For the state machine the student must first understand the ‘goal’ and the more significant stages. In this scenario the student will notice there are three important states or stages: when the game is idle(bntU hasn't been pressed, in state 1) and in this stage the score will be shown, when the game has begun on the clock is counting down(state two) in the transition to this stage the clock will load its new value and once it has reached this stage it will start counting down, and finally when the clock has reached zero and the led has turned on showing that the time is up(state 3). From this they will develop the following state machine:

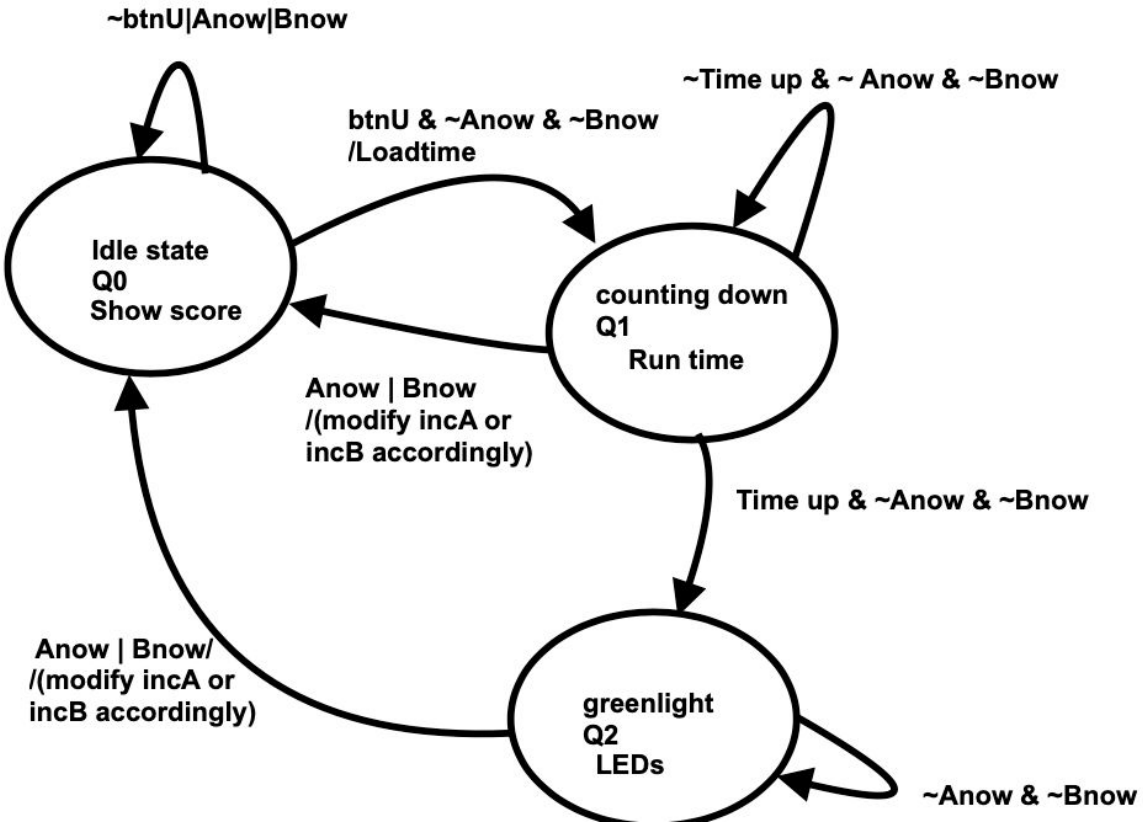


*Figure 13 above is a visual depiction of each state

From the state diagram above we can also figure out the following logic for each input:

LoadTime = the transition from Q0 to Q1 or Q0 & ~Anow & ~Bnow * btnU

Showscore = idle state or Q0

Runtime = counting down state or Q1

Light = greenlight state or Q2

From this we can create the following schematic:

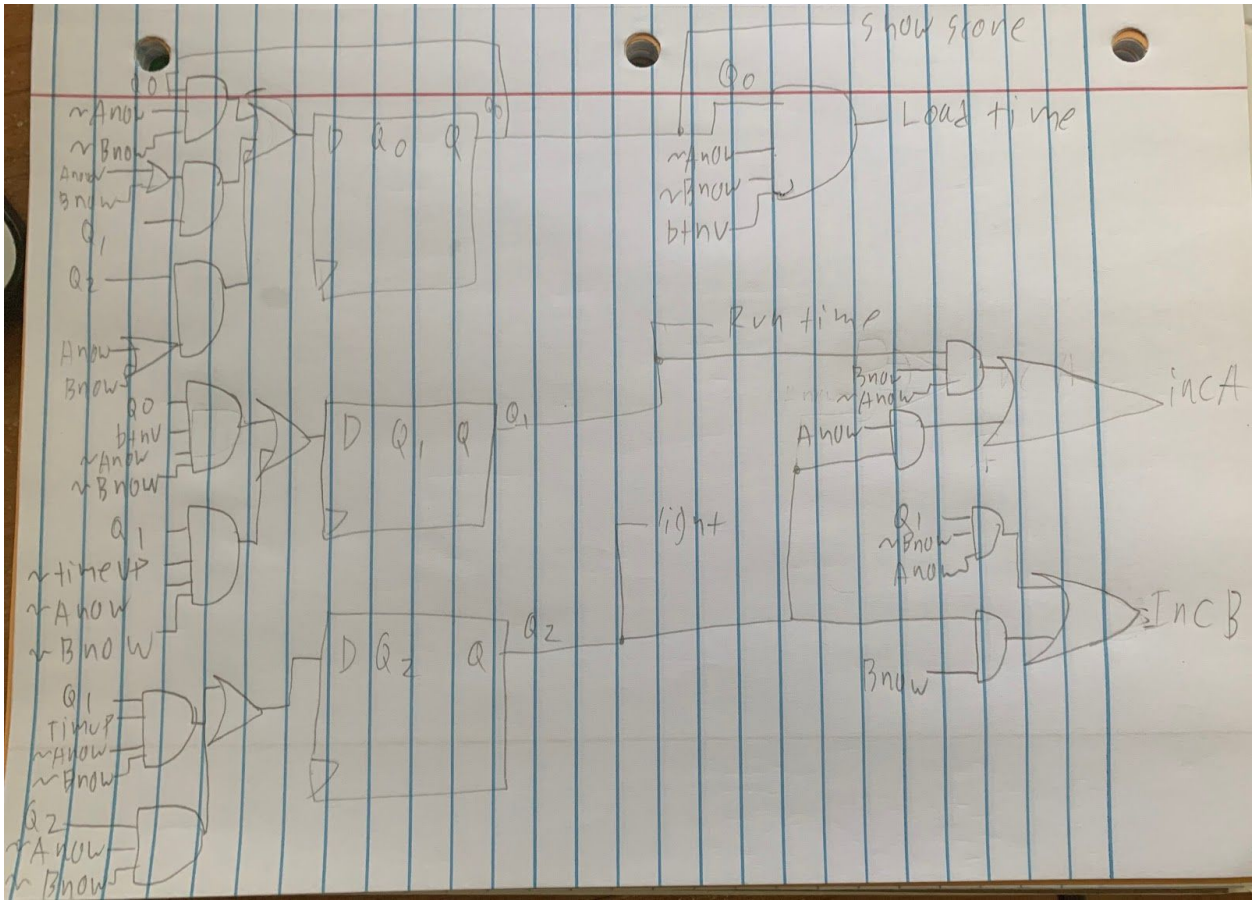


*figure 14 is the schematic for the state machine used in lab 4

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 :

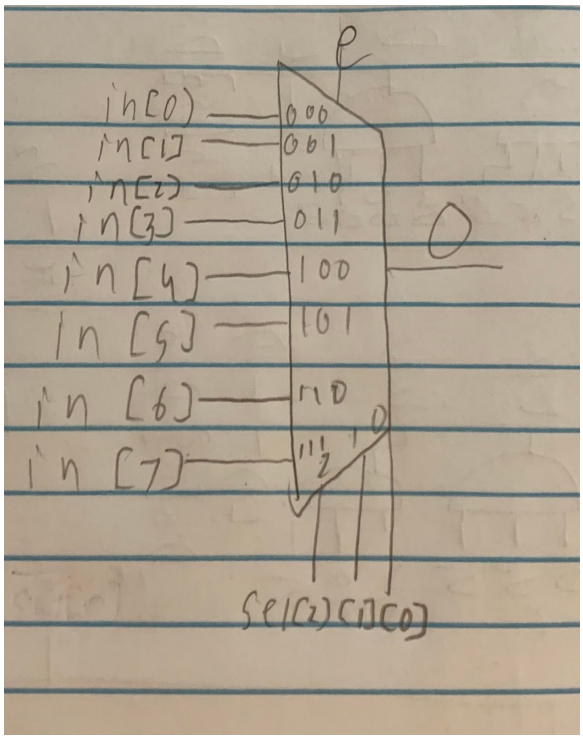
T flip-flop:

(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 15 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.

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:

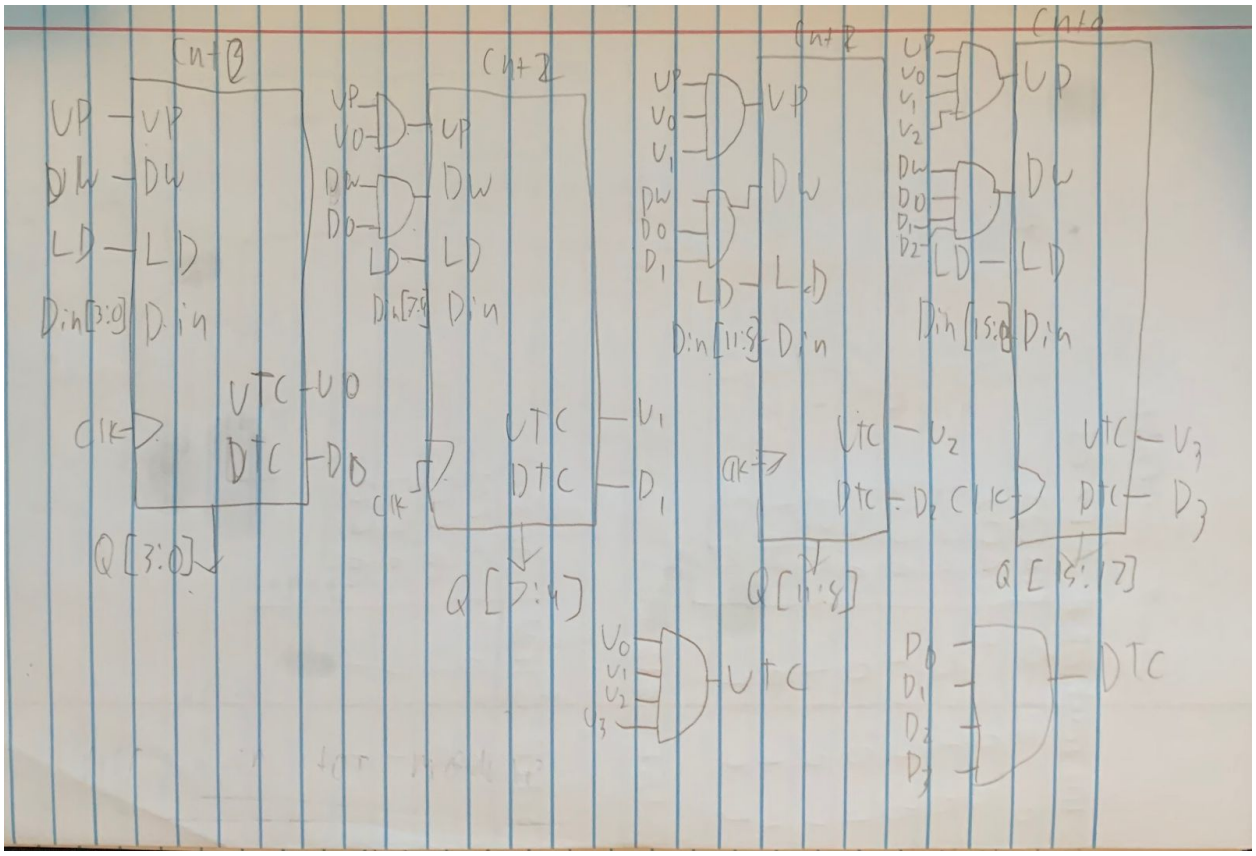
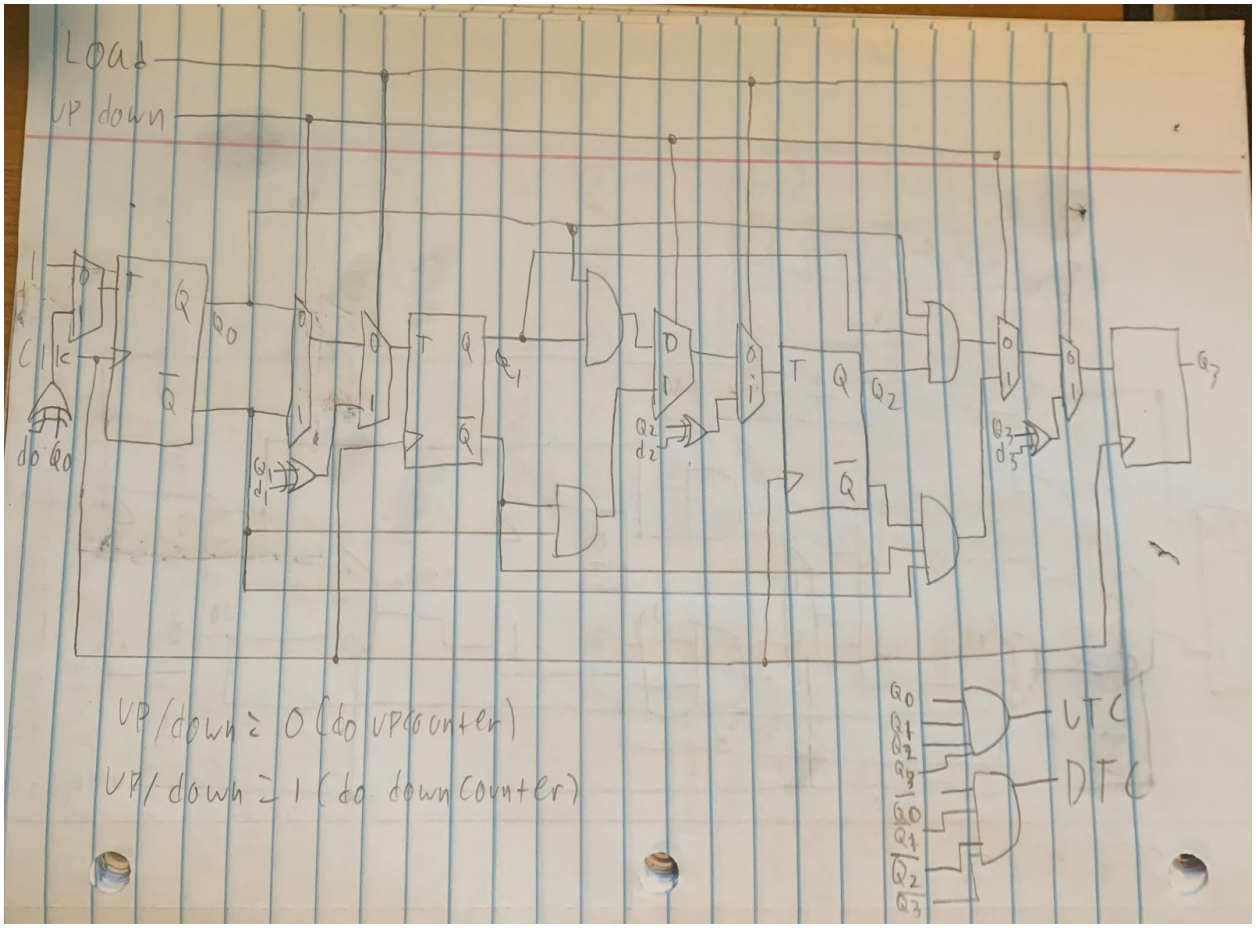


*figure 16 above is the schematic for the 16 bit up/down counter

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:

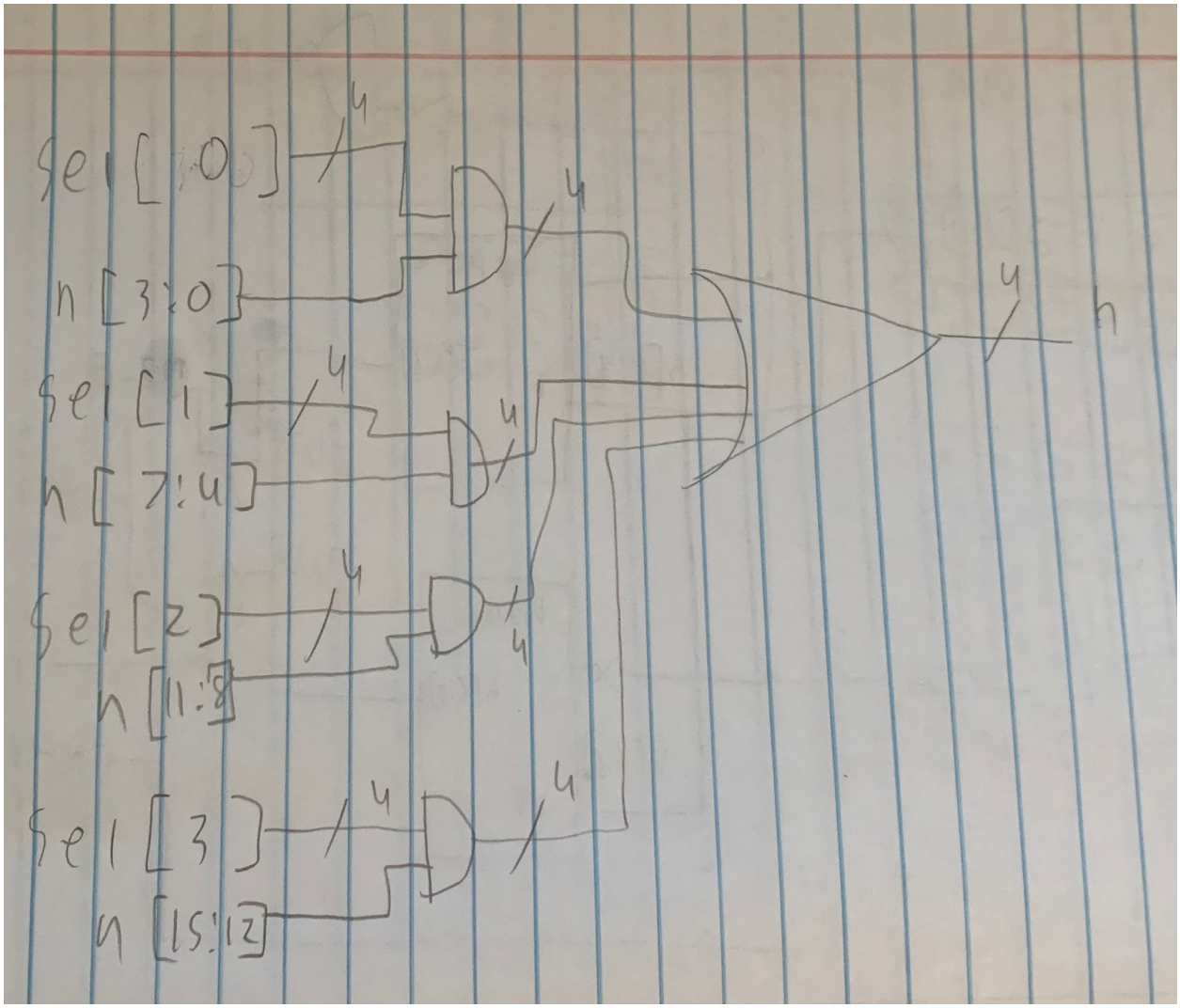


*figure 18 above is the schematic for the selector

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

Simulations:

Random number generator:

For simulating the LFSM(random number generator) I simply ran the program for 4000 nanoseconds in order to make sure that the outputs were different.

State Machine:

For simulating the State Machine I simply tested out all edge cases such as attempting to click btnU while switch 1 or 2 was on in order to make sure that it wouldn't start going into different stages. I also tested out all of the possible outcomes from inputs such as one or both players clicking the switch before time ran out as well as testing the same cases but with time actually running out.

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 one player flipping the switch before time ran out as well as both flipping before time ran out and the and both/one switch being switched before the timer had finished.

Questions:

There are no questions for this lab

Conclusion:

There were some challenges with actually designing the state machine, but once the machine was properly designed and was revised if needed it functioned as expected. This lab helped strengthen the students' understanding of how to create and implement a state machine as well as giving the student some experience with bit vector manipulation(for the clock and random number generator). This experiment also gave students some experience with what a random number generator looks like as well as giving them some experience with how to properly shift bits in a bitvector. This experiment also taught the students how to implement previously designed modules in order to build off of them and create a larger project(used our counters and selectors and more modules from previous labs).

Sources:

Cse 100 Lab 5 manual:

https://classes.soe.ucsc.edu/cse100/Spring20/lab/lab5S20/lab5.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

Appendices:

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 05/07/2020 07:39:57 AM
// 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 btnR,
    input btnU,
    input [15:0] sw,
    output [3:0] an,
    output dp,
    output [15:0] led,
    output [6:0] seg
    );
    wire TimeUp, LoadTime, RunTime, IncA, IncB, ShowScore, digsel, qsec, clk, light,
Anow, Bnow, you;
    wire [15:0] down, selval, countdown;
    wire [3:0] ringval, selected;
    lab5_clks slowit(.clkin(clkin), .greset(btnR), .clk(clk), .digsel(digsel),
.qsec());
    assign qsec = 1'b1;
    FDRE #(.INIT(1'b0)) Q4_0a (.C(clk), .CE(1'b1), .D(sw[0]), .Q(Anow));
    FDRE #(.INIT(1'b0)) Q4_0b (.C(clk), .CE(1'b1), .D(sw[15]), .Q(Bnow));
    FDRE #(.INIT(1'b0)) Q4_0c (.C(clk), .CE(1'b1), .D(btnU), .Q(you));
    Statemachine mech(.clk(clk), .btnU(you), .TimeUp(TimeUp), .Anow(Anow),
.Bnow(Bnow), .light(light), .LoadTime(LoadTime), .RunTime(RunTime), .IncA(IncA),
.IncB(IncB), .ShowScore(ShowScore));
    assign down[15:8] = 8'h00;
```

```
    LFSR rand(.clk(clk), .rnd(down[7:0]));
    counterUD16L Time_counter(.Din(down), .Up(1'b0), .LD(LoadTime), .Dw(RunTime &
qsec), .UTC(), .clk(clk), .DTC(TimeUp), .Q(countdown)); //Q = selval
    assign selval[11:4] = countdown[7:0];
    countUD4L a(.Up(IncA), .Dw(1'b0), .LD(1'b0), .Din(), .clk(clk), .UTC(), .DTC(),
.Q(selval[3:0])); // Q = selval
    countUD4L b(.Up(IncB), .Dw(1'b0), .LD(1'b0), .Din(), .clk(clk), .UTC(), .DTC(),
.Q(selval[15:12])); // Q = selval
    RingCounter ringer(.advance(digsel), .clk(clk), .out(ringval));
    Selector select(.sel(ringval), .N(selval), .H(selected));
    hex7seg display(.e(1'b1), .n(selected), .seg(seg));
    assign led[0] = sw[0];
    assign led[15] = sw[15];
    assign led[7] = light;
    assign led[8] = light;
    assign an[0] = ~(ringval[0] & ShowScore);
    assign an[1] = ~(ringval[1] & sw[3]);
    assign an[2] = ~(ringval[2] & sw[3]);
    assign an[3] = ~(ringval[3] & ShowScore);
    assign dp = 1'b1;
    assign led[14:9] = 1'b0;
    assign led[6:1] = 1'b0;
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: 04/16/2020 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: 05/06/2020 11:01:27 AM
// Design Name:
// Module Name: LFSR
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module LFSR(
    input clk,
    output [7:0] rnd
    );
    wire first;
    wire [7:0] loop;
    assign first = loop[0] ^ loop[5] ^ loop[6] ^ loop[7];
    FDRE #(.INIT(1'b1)) Q5_00 (.C(clk), .CE(1'b1), .D(first), .Q(loop[0]));
    FDRE #(.INIT(1'b0)) Q5_01 (.C(clk), .CE(1'b1), .D(loop[0]), .Q(loop[1]));
    FDRE #(.INIT(1'b0)) Q5_02 (.C(clk), .CE(1'b1), .D(loop[1]), .Q(loop[2]));
    FDRE #(.INIT(1'b0)) Q5_03 (.C(clk), .CE(1'b1), .D(loop[2]), .Q(loop[3]));
    FDRE #(.INIT(1'b0)) Q5_04 (.C(clk), .CE(1'b1), .D(loop[3]), .Q(loop[4]));
    FDRE #(.INIT(1'b0)) Q5_05 (.C(clk), .CE(1'b1), .D(loop[4]), .Q(loop[5]));
    FDRE #(.INIT(1'b0)) Q5_06 (.C(clk), .CE(1'b1), .D(loop[5]), .Q(loop[6]));
    FDRE #(.INIT(1'b0)) Q5_07 (.C(clk), .CE(1'b1), .D(loop[6]), .Q(loop[7]));
    assign rnd[5:0] = loop[5:0];
    assign rnd[6] = 1'b0;
    assign rnd[7] = 1'b0;

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 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: 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
```

```
`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: 05/06/2020 11:21:45 PM
// Design Name:
// Module Name: Statemachine
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module Statemachine(
    input btnU,
    input TimeUp,
    input Anow,
    input Bnow,
    input clk,
    output LoadTime,
    output RunTime,
    output light,
    output IncA,
    output IncB,
    output ShowScore
    );
    wire[2:0] D, Q;
    FDRE #(.INIT(1'b1)) Q0_30 (.C(clk), .CE(1'b1), .D(D[0]), .Q(Q[0]));     //idle
state or Q0
    FDRE #(.INIT(1'b0)) Q0_31 (.C(clk), .CE(1'b1), .D(D[1]), .Q(Q[1]));
//counting time Q1
    FDRE #(.INIT(1'b0)) Q0_32 (.C(clk), .CE(1'b1), .D(D[2]), .Q(Q[2]));
//greenlight Q2

    assign LoadTime = Q[0] & ~Anow & ~Bnow & btnU;
    assign ShowScore = Q[0];
    assign RunTime = Q[1];
```

```
    assign light = Q[2];
    assign IncA = (Q[1] & Bnow & ~Anow) | (Q[2] & Anow);
    assign IncB = (Q[1]& ~Bnow & Anow) | (Q[2] & Bnow);
    assign D[0] = (Q[0] & (~btnU | Anow | Bnow)) | (Q[1] & (Anow | Bnow)) | (Q[2] &
(Anow | Bnow));
    assign D[1] = (Q[0] & (btnU & ~Anow & ~Bnow)) | (Q[1] & (~TimeUp & ~Anow & ~Bnow)
    assign D[2] = (Q[1] & TimeUp & ~Anow & ~Bnow) | (Q[2] & ~Anow & ~Bnow);
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
```

```
## 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]}]
```

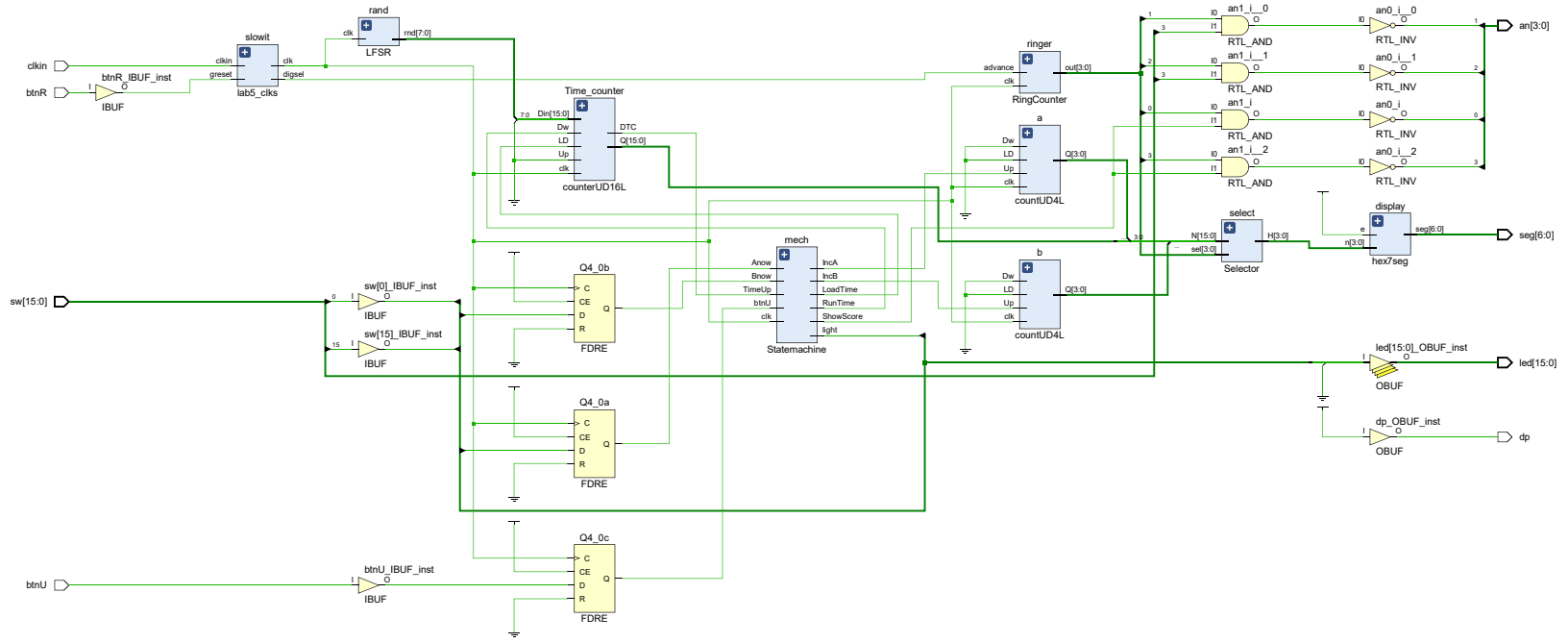
```
set_property PACKAGE_PIN V5 [get_ports {seg[5]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {seg[5]}]
set_property PACKAGE_PIN U7 [get_ports {seg[6]}]
    set_property IOSTANDARD LVCMOS33 [get_ports {seg[6]}]

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
```

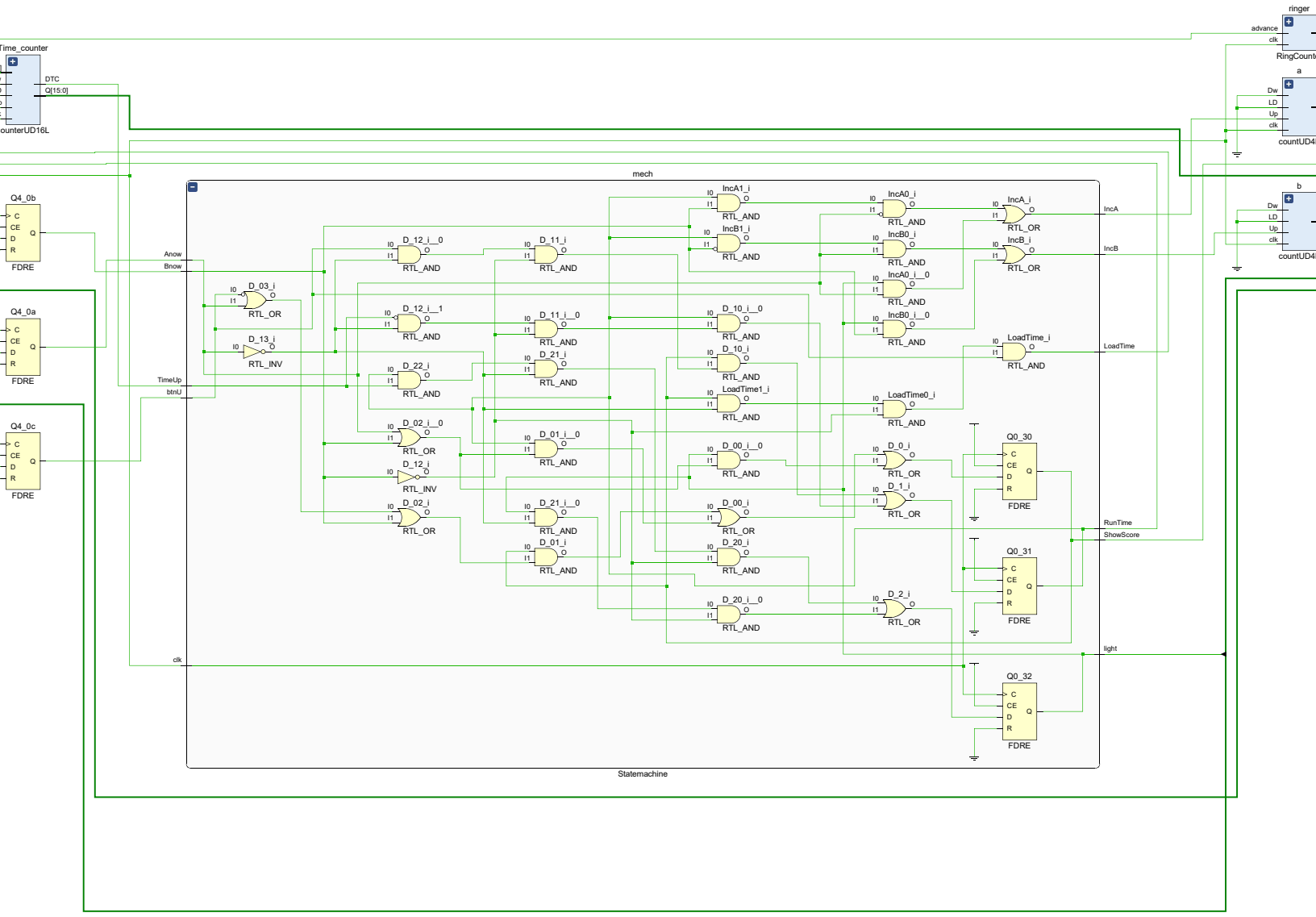
```
#set_property PACKAGE_PIN H1 [get_ports {JA[4]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {JA[4]}]
##Sch name = JA8
#set_property PACKAGE_PIN K2 [get_ports {JA[5]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {JA[5]}]
##Sch name = JA9
#set_property PACKAGE_PIN H2 [get_ports {JA[6]}]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

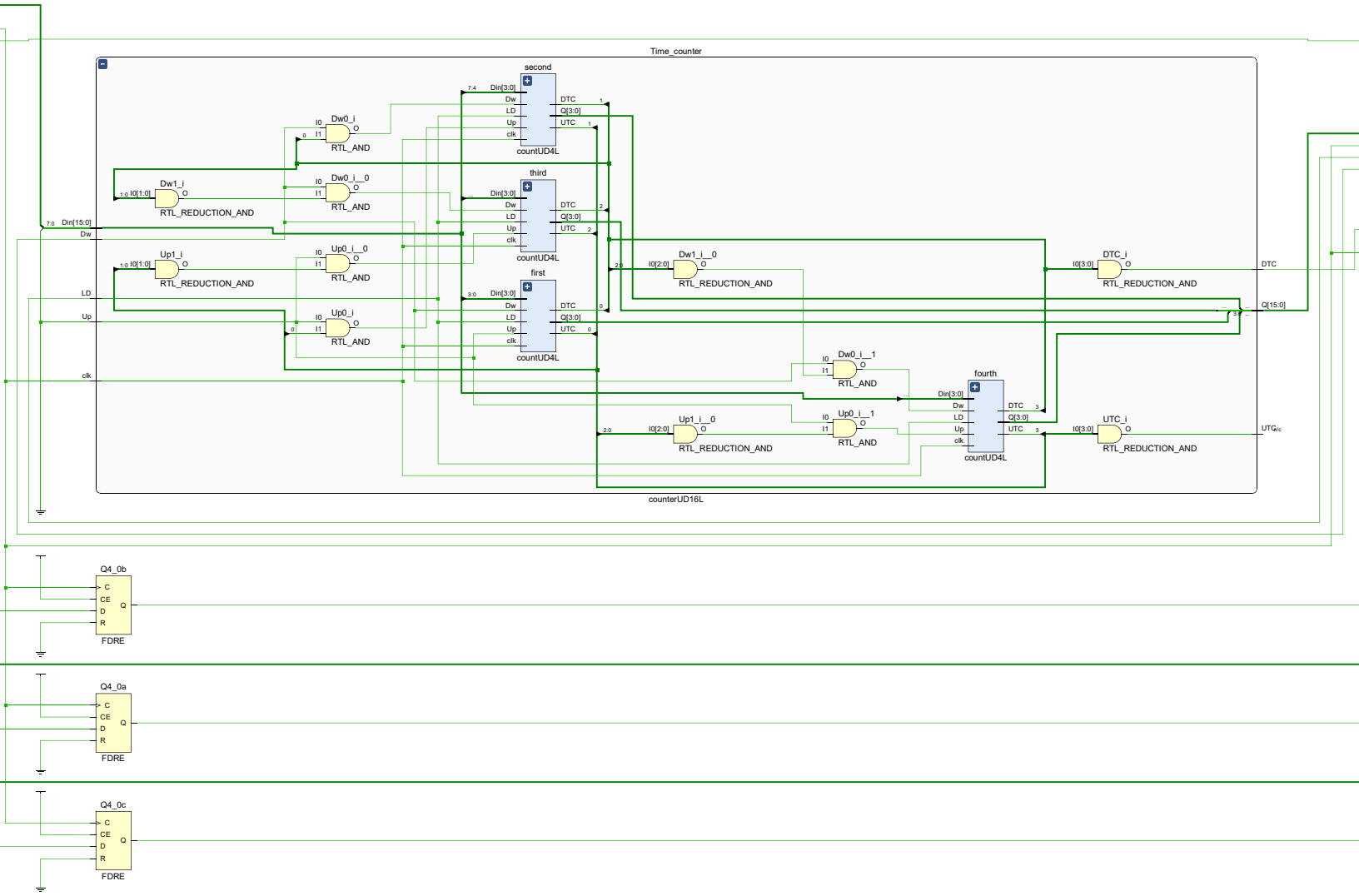
```
##Sch name = JC2
#set_property PACKAGE_PIN M18 [get_ports {JC[1]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {JC[1]}]
##Sch name = JC3
#set_property PACKAGE_PIN N17 [get_ports {JC[2]}]
    #set_property IOSTANDARD LVCMOS33 [get_ports {JC[2]}]
##Sch name = JC4
#set_property PACKAGE_PIN P18 [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]
```

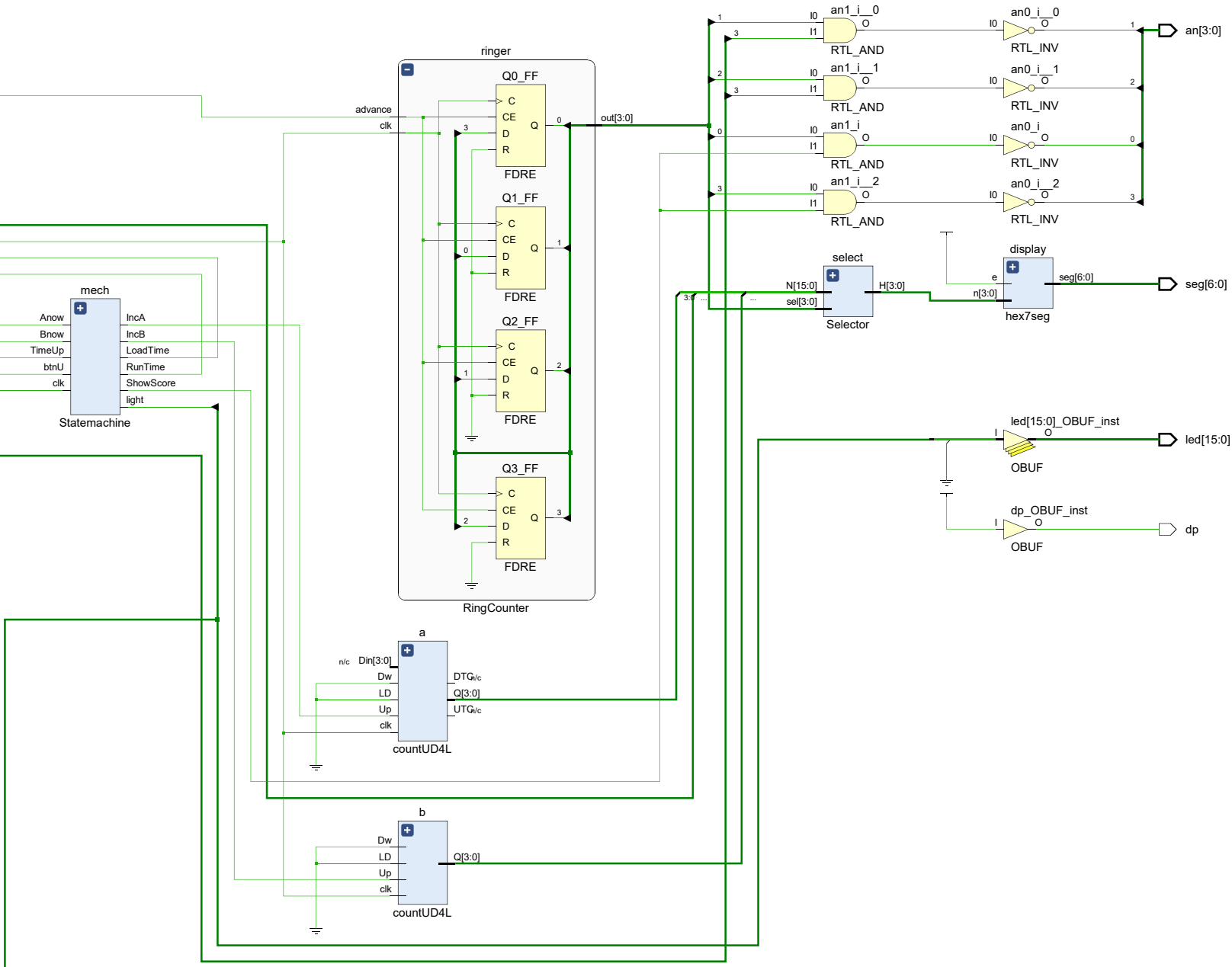
```
    #set_property PULLUP true [get_ports PS2Clk]
#set_property PACKAGE_PIN B17 [get_ports PS2Data]
    #set_property IOSTANDARD LVCMOS33 [get_ports PS2Data]
    #set_property PULLUP true [get_ports PS2Data]


##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]
```

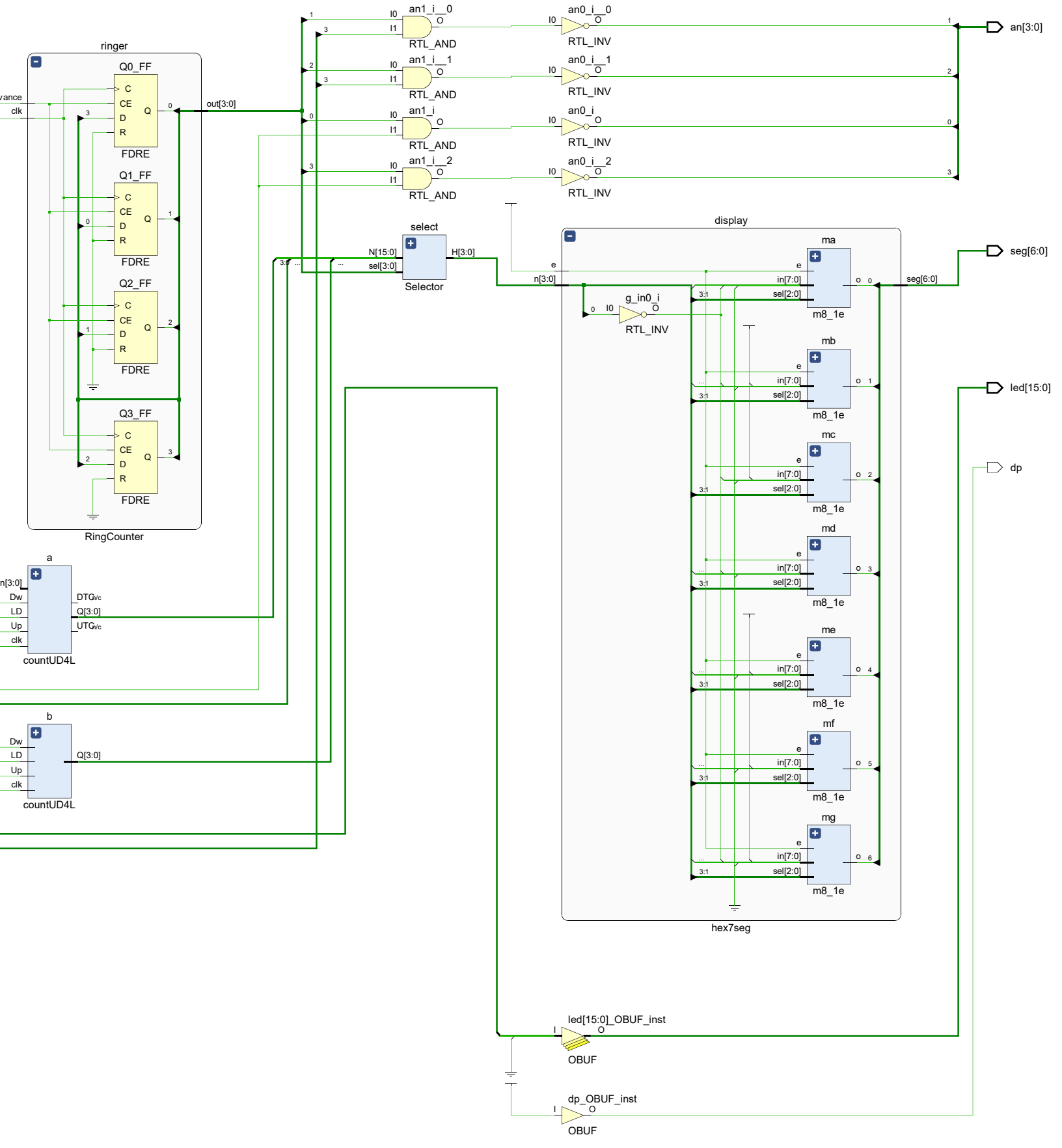

clkin
btnR
btnR_IBUF_inst
IBUF
slowit
clkin
greset
clk
digsel
lab5_clks
rand
clk
rnd[7:0]
LFSR
Time_counter
7:0
Din[15:0]
Dw
LD
Up
clk
DTC
Q[15:0]
counterUD16L
ringer
advance
clk
out[3:0]
RingCounter
a
Dw
LD
Up
clk
Q[3:0]
countUD4L
b
Dw
LD
Up
clk
Q[3:0]
countUD4L
mech
Anow
Bnow
TimeUp
btnU
clk
IncA
IncB
LoadTime
RunTime
ShowScore
light
Statemachine
an1_i__0
an1_i__1
an1_i
an1_i__2
I0
I1
O
RTL_AND
an0_i__0
an0_i__1
an0_i
an0_i__2
RTL_INV
an[3:0]
select
N[15:0]
sel[3:0]
H[3:0]
Selector
display
e
n[3:0]
seg[6:0]
hex7seg
sw[15:0]
sw[0]_IBUF_inst
sw[15]_IBUF_inst
IBUF
Q4_0b
Q4_0a
Q4_0c
C
CE
D
R
Q
FDRE
led[15:0]_OBUF_inst
led[15:0]
OBUF
dp_OBUF_inst
dp
btnU
btnU_IBUF_inst

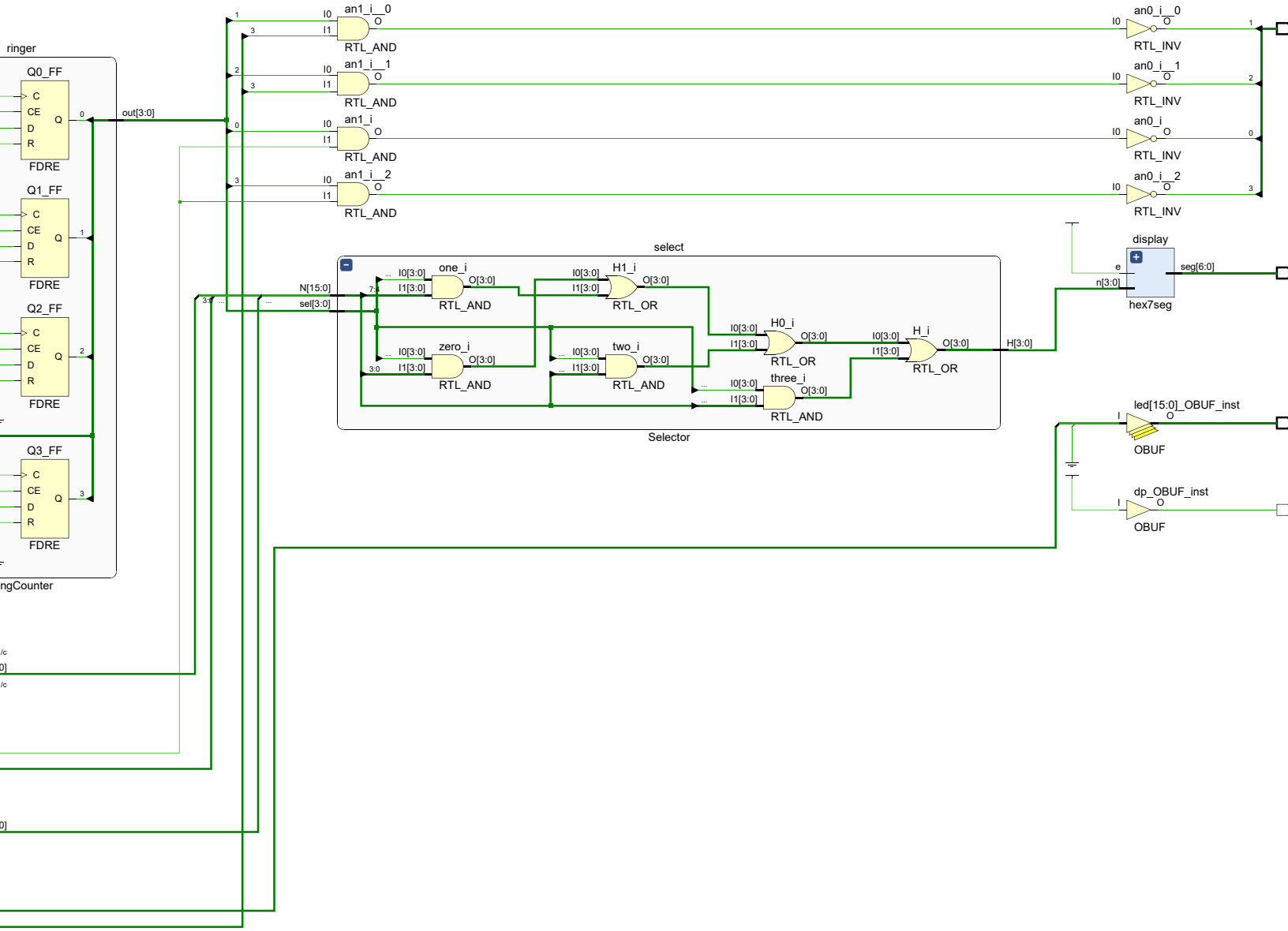

ringer
advance
clk
RingCount
a
Dw
LD
Up
clk
countUD4
b
Dw
LD
Up
clk
countUD4
Time_counter
DTC
Q[15:0]
ounterUD16L
mech
Q4_0b
C
CE
D
R
Q
FDRE
Anow
Bnow
Q4_0a
FDRE
TimeUp
btnU
Q4_0c
FDRE
clk
D_03_i
RTL_OR
D_13_i
RTL_INV
D_12_i__0
RTL_AND
D_12_i__1
RTL_AND
D_22_i
RTL_AND
D_02_i__0
RTL_OR
D_12_i
RTL_INV
D_02_i
RTL_OR
D_11_i
RTL_AND
D_11_i__0
RTL_AND
D_21_i
RTL_AND
D_01_i__0
RTL_AND
D_21_i__0
RTL_AND
D_01_i
RTL_AND
IncA1_i
RTL_AND
IncB1_i
RTL_AND
D_10_i__0
RTL_AND
D_10_i
RTL_AND
LoadTime1_i
RTL_AND
D_00_i__0
RTL_AND
D_00_i
RTL_OR
D_20_i
RTL_AND
D_20_i__0
RTL_AND
IncA0_i
RTL_AND
IncB0_i
RTL_AND
IncA0_i__0
RTL_AND
IncB0_i__0
RTL_AND
LoadTime0_i
RTL_AND
D_0_i
RTL_OR
D_1_i
RTL_OR
D_2_i
RTL_OR
IncA_i
RTL_OR
IncB_i
RTL_OR
LoadTime_i
RTL_AND
IncA
IncB
LoadTime
Q0_30
Q0_31
Q0_32
FDRE
RunTime
ShowScore
light
Statemachine

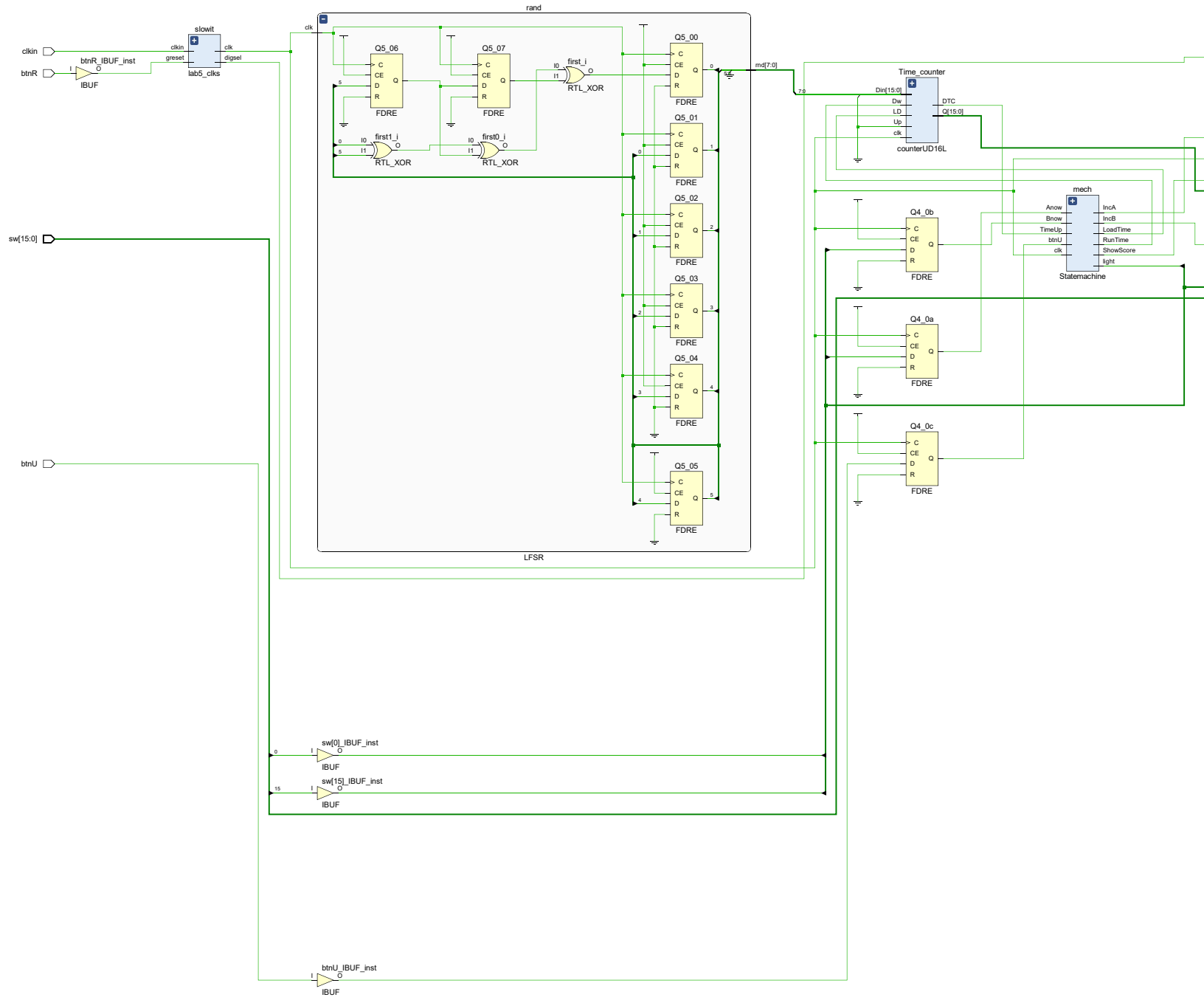

Time_counter
second
third
first
fourth
countUD4L
RTL_AND
RTL_REDUCTION_AND
Dw0_i
Dw0_i__0
Dw0_i__1
Dw1_i
Dw1_i__0
Up0_i
Up0_i__0
Up0_i__1
Up1_i
Up1_i__0
DTC_i
UTC_i
Din[15:0]
Dw
LD
Up
clk
DTC
Q[15:0]
UTC
counterUD16L

Q4_0b
Q4_0a
Q4_0c
C
CE
D
R
Q
FDRE

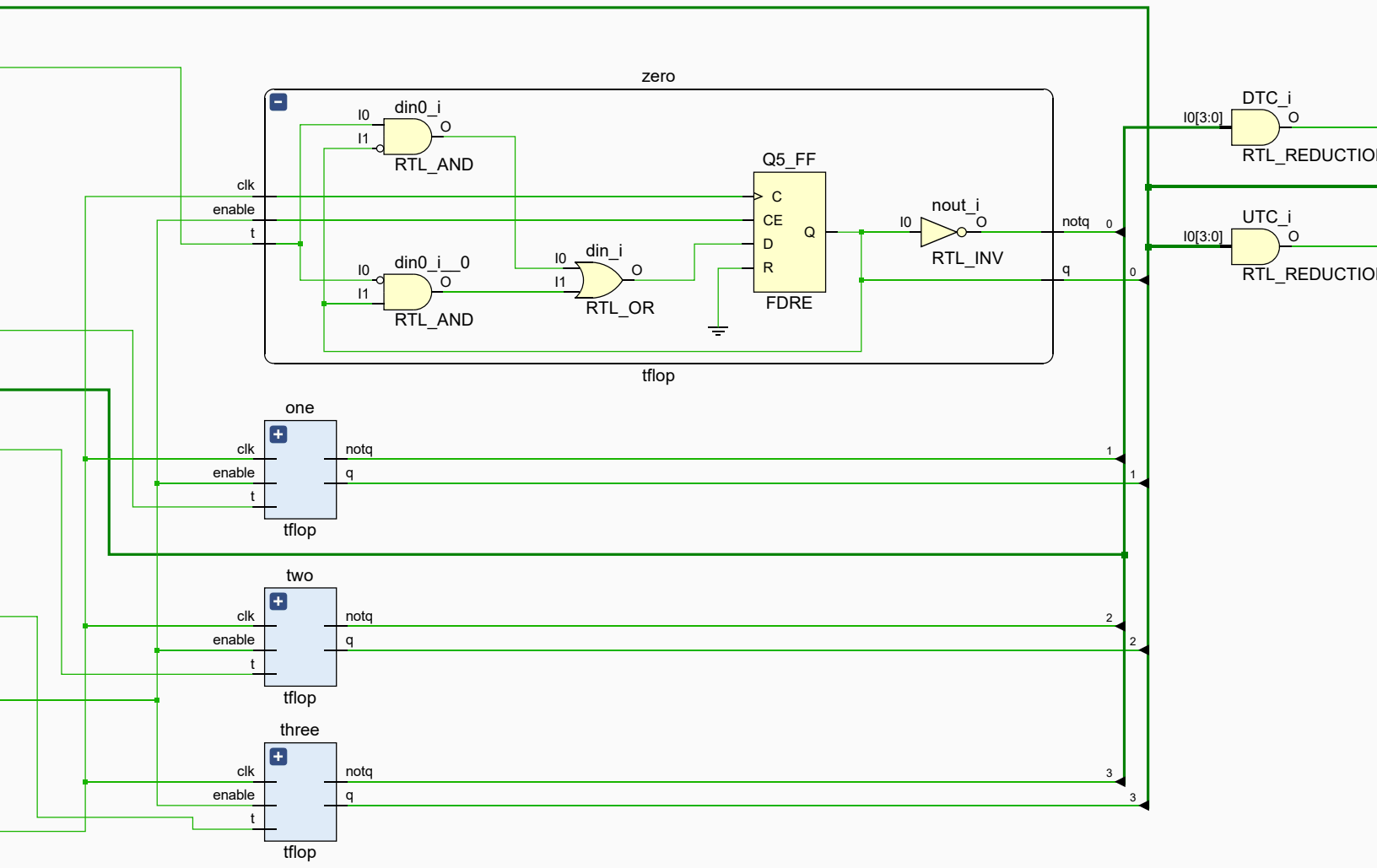

ringer
Q0_FF
Q1_FF
Q2_FF
Q3_FF
FDRE
advance
clk
out[3:0]
RingCounter
mech
Anow
Bnow
TimeUp
btnU
clk
IncA
IncB
LoadTime
RunTime
ShowScore
light
Statemachine
a
b
Din[3:0]
Dw
LD
Up
DTG
Q[3:0]
UTG
countUD4L
an1_i__0
an1_i__1
an1_i
an1_i__2
RTL_AND
an0_i__0
an0_i__1
an0_i
an0_i__2
RTL_INV
an[3:0]
select
N[15:0]
sel[3:0]
H[3:0]
Selector
display
hex7seg
seg[6:0]
led[15:0]_OBUF_inst
led[15:0]
OBUF
dp_OBUF_inst
dp


ringer
Q0_FF
Q1_FF
Q2_FF
Q3_FF
FDRE
RingCounter
out[3:0]
an1_i__0
an1_i__1
an1_i
an1_i__2
RTL_AND
an0_i__0
an0_i__1
an0_i
an0_i__2
RTL_INV
an[3:0]
select
Selector
N[15:0]
sel[3:0]
H[3:0]
display
ma
mb
mc
md
me
mf
mg
m8_1e
g_in0_i
hex7seg
seg[6:0]
led[15:0]
dp
led[15:0]_OBUF_inst
dp_OBUF_inst
OBUF
a
b
countUD4L


ringer
Q0_FF
Q1_FF
Q2_FF
Q3_FF
FDRE
out[3:0]
ngCounter
an1_i__0
an1_i__1
an1_i
an1_i__2
RTL_AND
an0_i__0
an0_i__1
an0_i
an0_i__2
RTL_INV
select
N[15:0]
sel[3:0]
one_i
zero_i
two_i
three_i
H1_i
H0_i
H_i
RTL_OR
H[3:0]
Selector
display
hex7seg
seg[6:0]
led[15:0]_OBUF_inst
dp_OBUF_inst
OBUF


clkin
btnR
btnR_IBUF_inst
IBUF
slowit
clkin
greset
clk
digsel
lab5_clks
rand
clk
Q5_06
Q5_07
first_i
RTL_XOR
first1_i
first0_i
FDRE
Q5_00
Q5_01
Q5_02
Q5_03
Q5_04
Q5_05
md[7:0]
LFSR
sw[15:0]
btnU
sw[0]_IBUF_inst
sw[15]_IBUF_inst
btnU_IBUF_inst
Time_counter
Din[15:0]
Dw
LD
Up
clk
DTC
Q[15:0]
counterUD16L
Q4_0b
Q4_0a
Q4_0c
mech
Anow
Bnow
TimeUp
btnU
clk
IncA
IncB
LoadTime
RunTime
ShowScore
light
Statemachine

Time_counter



```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 05/06/2020 11:15:11 AM
// Design Name:
// Module Name: lfsrsim
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module lfsrsim();
reg clk;
wire [7:0] rnd;

LFSR rand(.clk(clk), .rnd(rnd));


 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
    #2000;
    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;
    reg [15:0] sw;
    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 timetrial(.clkin(clkin), .btnR(btnR), .btnU(btnU), .sw(sw),
.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
```

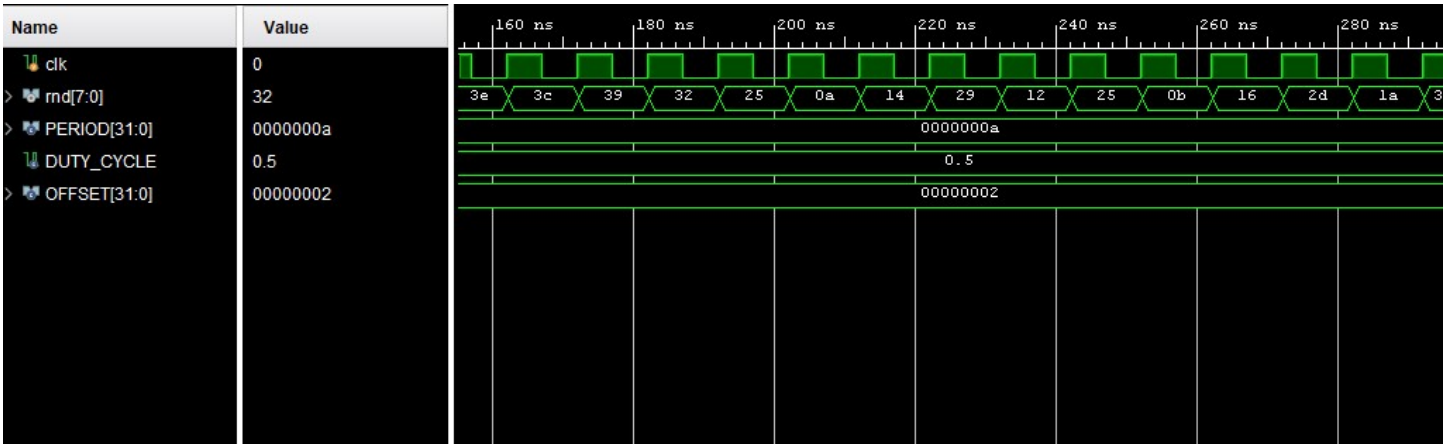
```
 end

initial
begin
btnU = 1'b0;
btnR = 1'b0;
sw = 16'h0000;
sw[3] = 1'b1;
#1100;
btnU = 1'b1;
#100;
btnU = 1'b0;
#700;
sw[0] = 1'b1;    //a does it first
#100;
sw[0] = 1'b0;
btnU = 1'b1;
#100;
btnU = 1'b0;
#3000;
sw[15] = 1'b1;   //b does it first
#100;
sw[15] = 1'b0;
btnU = 1'b1;
#100;
btnU = 1'b0;
#2000;
sw[15] = 1'b1;   //both at same time
sw[0] = 1'b1;
#100;
sw[15] = 1'b0;
sw[0] = 1'b0;
btnU = 1'b1;
#100;
btnU = 1'b0;
#100;
sw[15] = 1'b1;   //b went early
#100;
sw[15] = 1'b0;
btnU = 1'b1;
#100;
btnU = 1'b0;
#100;
sw[0] = 1'b1;    //a went early
#100;
sw[15] = 1'b0;
```

```
   sw[0] = 1'b0;
   btnU = 1'b1;
   #100;
   btnU = 1'b0;
   #100;
   sw[15] = 1'b1;   //both went early
   sw[0] = 1'b1;
   #100;
    end
endmodule
```

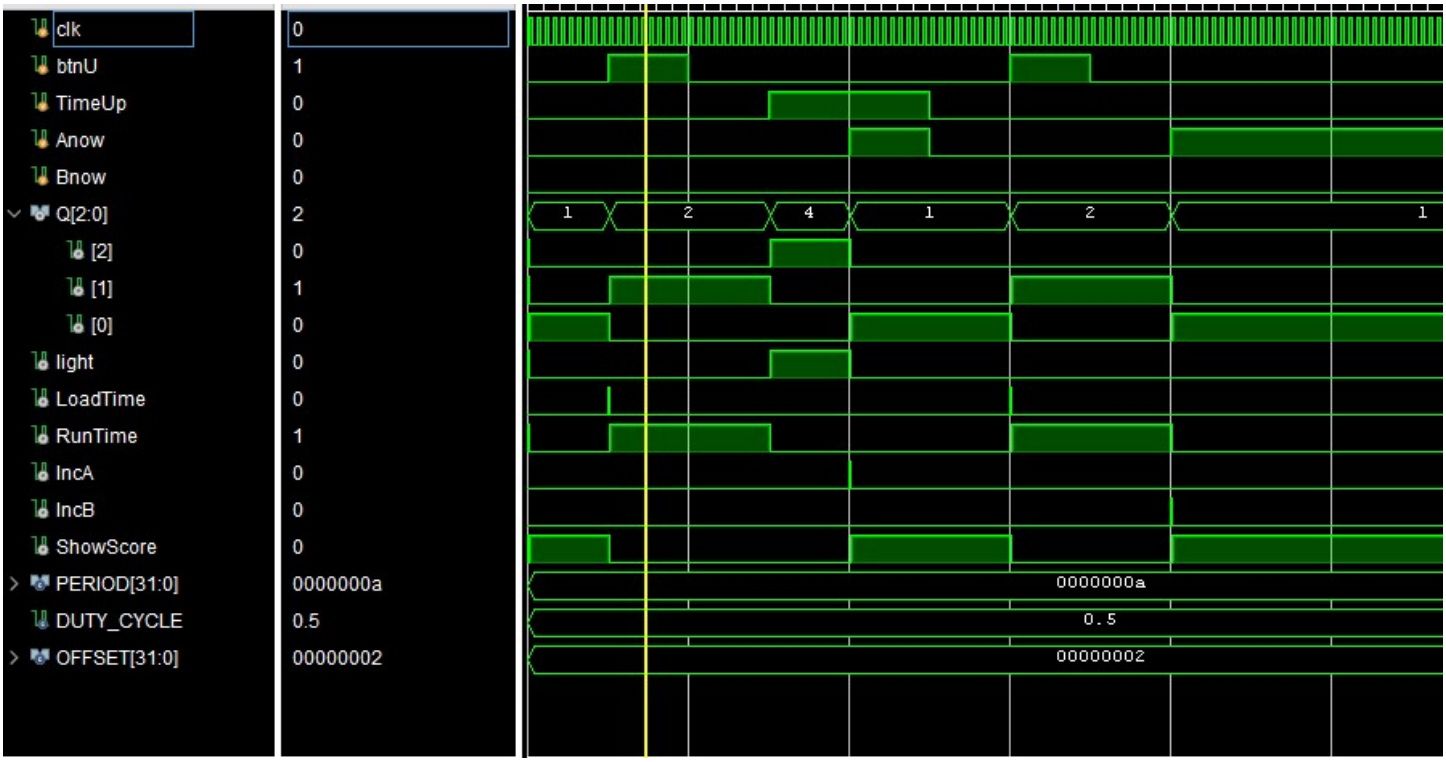
```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 05/06/2020 11:44:10 PM
// Design Name:
// Module Name: statemachinesim
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module statemachinesim();
reg clk, btnU, TimeUp, Anow, Bnow;
wire light, LoadTime, RunTime, IncA, IncB, ShowScore;

Statemachine mech(.clk(clk), .btnU(btnU), .TimeUp(TimeUp), .Anow(Anow), .Bnow(Bnow),
.light(light), .LoadTime(LoadTime), .RunTime(RunTime), .IncA(IncA), .IncB(IncB),
.ShowScore(ShowScore));

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
    btnU = 1'b0;
    TimeUp = 1'b0;
```

```
    Anow = 1'b0;
    Bnow = 1'b0;
    #100;
    btnU = 1'b1;
    #100;
    btnU = 1'b0;
    #100;
    TimeUp = 1'b1;
    #100;
    Anow = 1'b1;
    #100;
    Anow = 1'b0;
    TimeUp = 1'b0;
    #100;
    btnU = 1'b1;
    #100;
    btnU = 1'b0;
    #100;
    Anow = 1'b1;
    end
endmodule
```

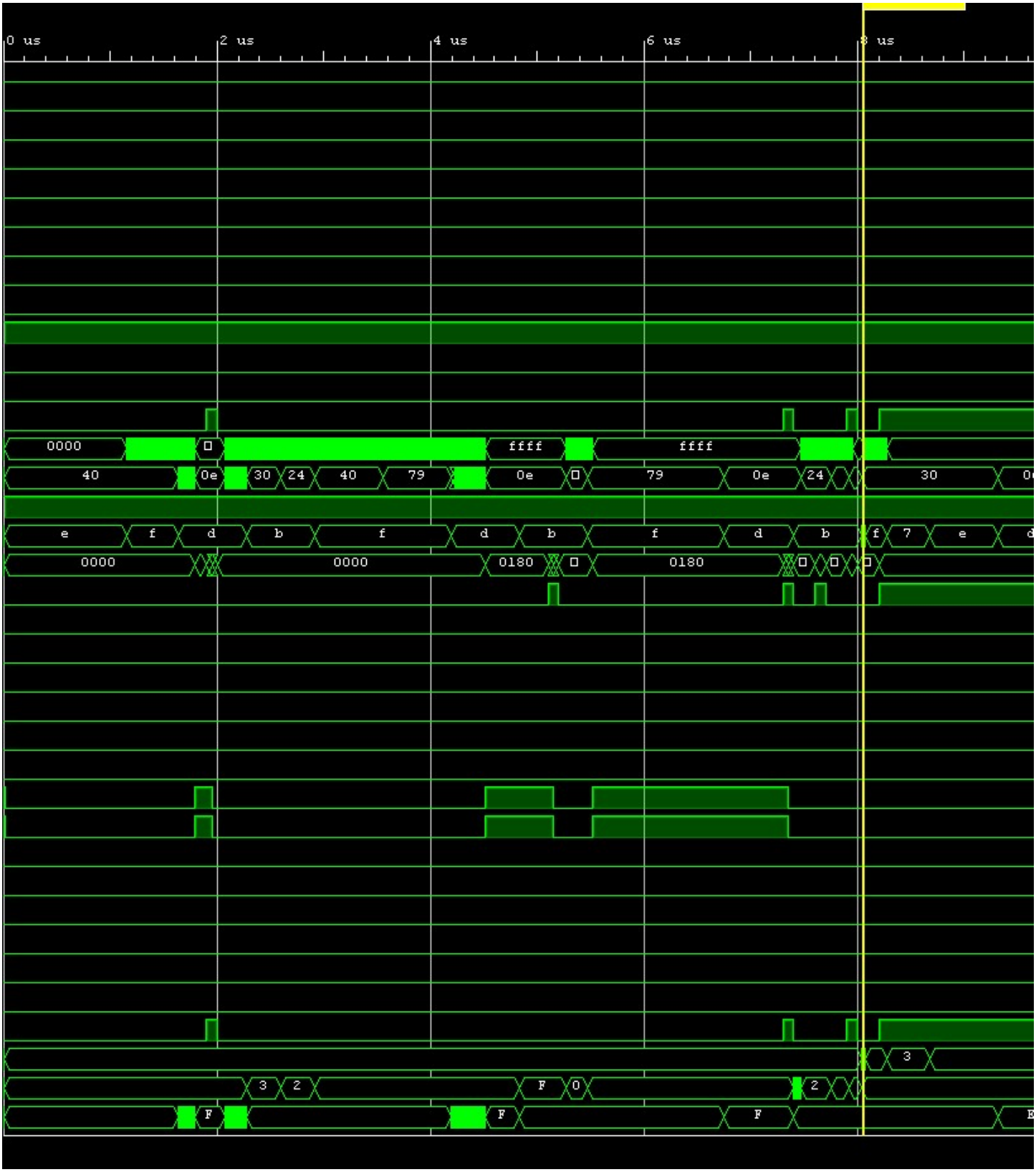
LFSR simulation results



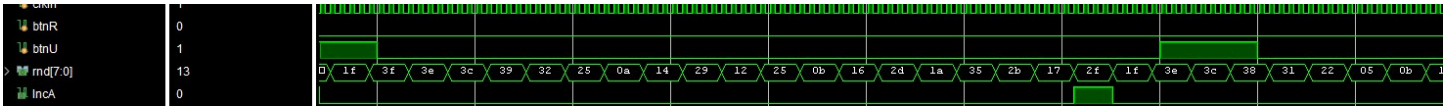
State Machine simulation



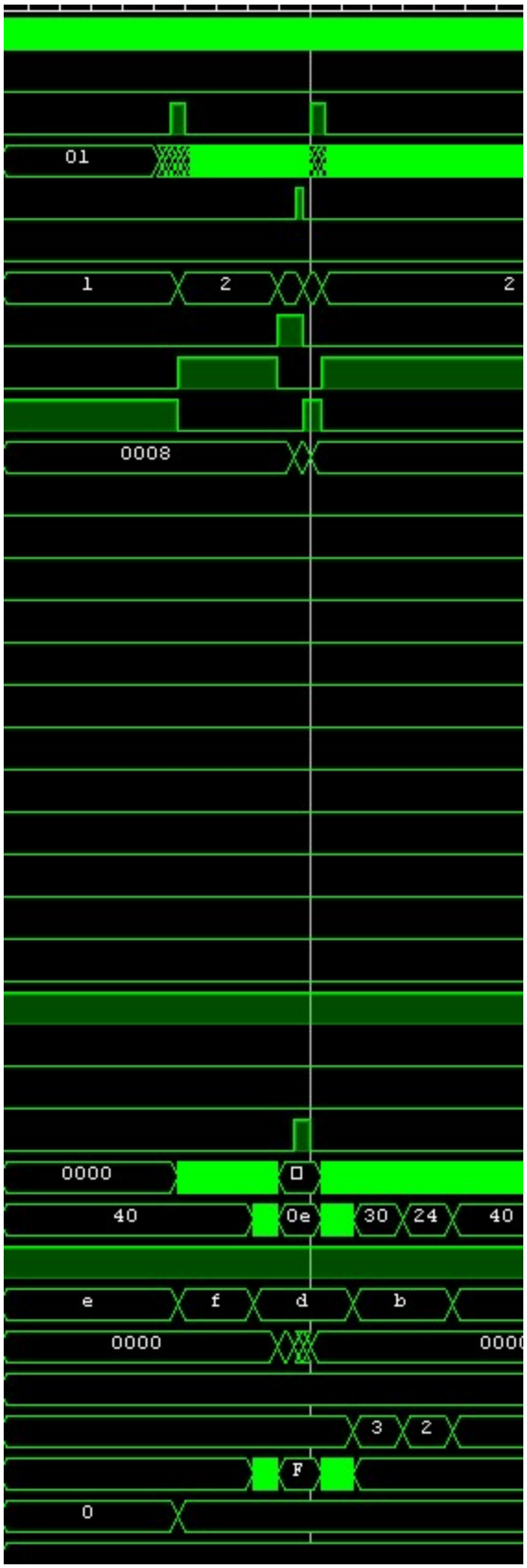
Note: due to the size of the led section and the quickness of the random number generator both of these simulation results are on different screenshots



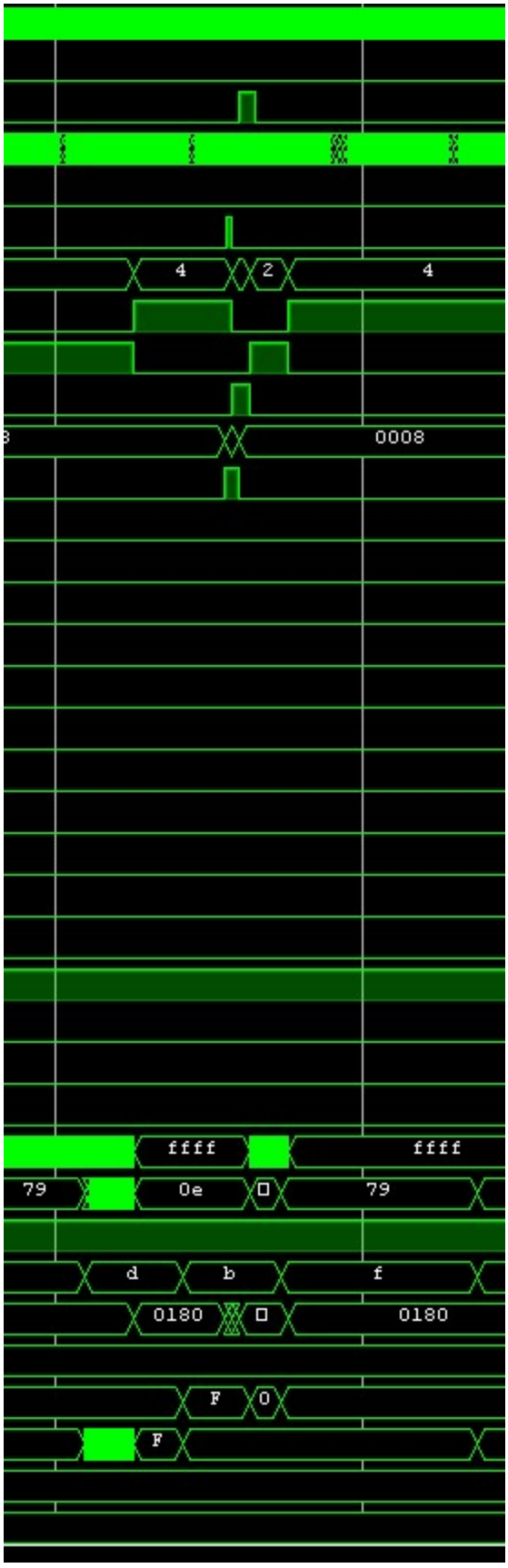
Random number generator results from the top simulation.



Player A winning by being first after the green light.



Player B winning by being first after the green light.



Player B winning by Player A flipping the switch before the green light.
Player A winning by Player B flipping the switch before the green light.
Players A and B both winning by flipping their switches at the same time after the green light.
Players A and B both losing by flipping their switches at the same time before the green light.