CMPE 100L: Lab5 Write Up

Yanliang Li

February 24, 2017

Contents

1 Description

The purpose of this lab was to create a time estimation game called, Chicken. This game was designed to be played by two players, U and D. When we press the start button, button C, the board will automatically generate a random number and start counting down from that number. Two players have one chance to press button. To win this game, the players want to be the last one to press the button before time runs out.

2 Methods

2.1 Random Number Generator

The first thing we did was to implement a random number generator. As mentioned above, when the game starts, we need to generate a random number and count down from the random number. To get a random number generator, we will use a Linear Feedback Shift Register. The idea of LFSR is that we use the XOR result of some specific bits in the LFSR as the input of the LFSR. We used eight Flip-Flops to implement this LFSR.

One thing to be noticed is that if all the bits in the register are zero then the output will always be zero's. Therefore, we need a way to prevent that. We have done similar thing in the previous labs. We set the initial value of one of the Flip-Flops to one to avoid getting all zero's for the ring counter before. Here we will do the same thing. We set the initial value of first Flip-Flops to one. Another thing was that the wait time would be long if the time counter receives an 8-bits number from the LFSR. Therefore, to avoid that, we managed to use only 7 bits from the LFSR. Also, we need to put a zero as the eighth bit because we are going to need a 8-bit number in the later part.

2.2 Time Counter

The job of a time counter for this lab was to count down the number to zero. From the previous step, the random number generator, we would get a random seven bits number. In this part, we will implement a time counter to count this random number to zero. Professor Schlag mentioned in the lab manual that there are two ways to do this part. The first method is to build a 4-bit loadable counter. The second one is to use two sign changers. I used the latter one in the part. The method I used includes two 4-bit counters and two sign changers. The idea was that we get a random number from the LFSR, then we invert the sign of the random number to negative by using sign changer. After that, we used two counters to count the number up to FF. Because we are using 4-bit counters, we need to separate the eight-bit number to two parts, each with 4-bit, and do the counting up separately. Then, as soon as the number reaches to FF, we use the sign changer to invert the sign again that will make the number back to 00.

2.3 Score Display

Score display was used to keep track of the players' scores. We created another 4-bit counter for this part. Since we always start counting from zero, we do not need input in this part,

neither LD nor TC. We only need count enable, clock, and output for this score display.

2.4 Led Display

One part in this lab was to make the led lights shift from right to left, aka. from LD0 to LD15. When we press start button, the system will generate a random number from LFSR and also, the led lights will shift from right to left. This part was to implement a led shifter to make that work. In order to build a led shifter, we need a count enable, reset, clock and outputs. We need a rest because we would want to reset all the led when they reach to the last led, led15.

2.5 State Machine

In this part, we will build a state machine. The process of building a state machine was tedious no matter how simple I describe later. First of all, we needed to determine how many states there would be. After a long time doing all the work, I came up with eight states. There will be a wait state. This state is for doing nothing but waiting for the button C to be pressed; otherwise, the state will simple stays there and do nothing. When we press button C, we will go to next state. At this state, I called it LED, the LFSR will generate a random number and the LEDs will light up from right to left. After the LED state finishes, the system will automatically go to next state, I called it START, and start counting down from the random number to zero. There are five states after the previous three. They are bothWin, D pressed, U pressed, U wins, and D wins. I would explain further in the result section where I would provide the diagram of my state machine so that it would be easier for me to explain and for you to understand.

3 Results

3.1 Random Number Generator

As mentioned above, for the random number generator part, we used the LFSR. The LFSR is an eight-bit shift register. The design diagram has been shown below.

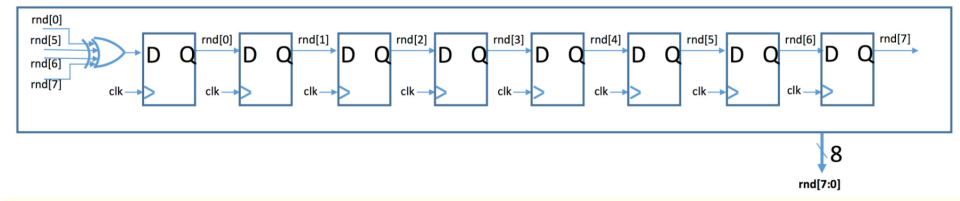

Figure 1: A digram for the random number generator.

At the beginning, we used a XOR gate to do the algorithm from four specific bits in the register. As the diagram indicated, we put the result of the XOR gate into the Flip-Flops. Another diagram of the LFSR is shown below:

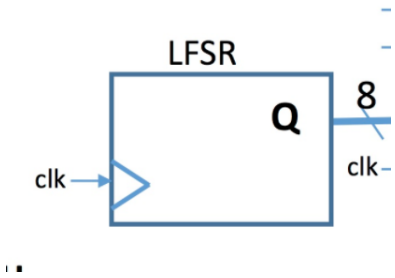

Figure 2: Another digram for the LFSR.

3.2 Time Counter

We used two 4-bit counter and two sign changers for the time counter. The idea has been stated clearly above. We used a sign changer to change the sign for the random number. The number will be split up into two parts. Then two 4-bit counters will count these two 4-bit numbers up to FF. At last, a sign changer will invert FF to 00. From the statement, we can see that the sign changer is always changing sign for numbers. Therefore, we will set the sign changer to always high. Besides, we did not want the two 4-bit numbers to be count up at the same time so we used CE and TC to make sure that the two counters will do their job one after another. Below is the diagram for time counter:

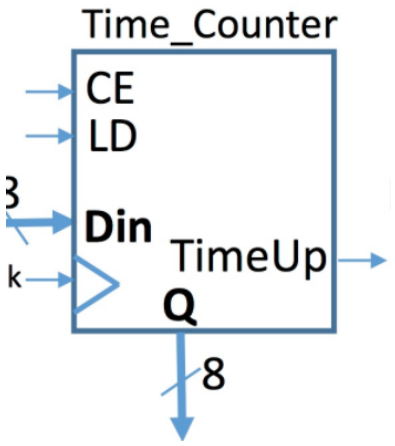

Figure 3: A digram for the time counter.

3.3 Score Display

The purpose of a score display is to keep track of the players' scores. We used a 4-bit counter in this part. Since the score will always be counted from zero, we do not need any input. However, we do need to know when to add score for which player. The input for the score display would be the increment U and increment D. Since we would not know the maximum score that the players could get, we do not need to use TC in this part. Below are the diagrams of score display:

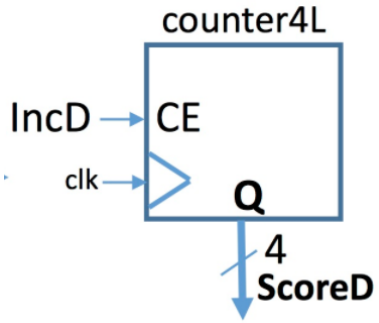

Figure 4: A digram for score D.

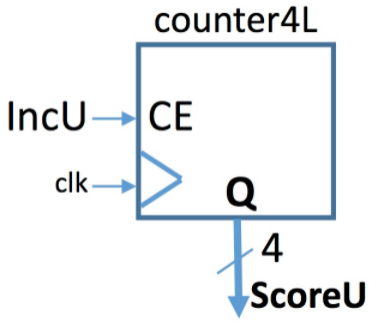

Figure 5: A digram for score U.

3.4 Led Display

As mentioned above, the led shifter is to light from LED0 to LED15 one by one. Since it is a fairly easy task, we only need to use CE, reset, clock and output. When CE is high, the led shifter will start working. The input of other 15 Flip Flops are from the previous Flip Flops' outputs. In the Verilog, we used two Flip Flops. The first one was for the first LED, LED0. The reason that the first LED is special is because the output of this FF would affect all other LEDs. The second FF is a bus FF, representing LED 1 to LED 15. Below is the diagram for LED shifter.

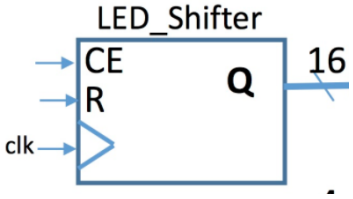

Figure 6: A digram for LED shifter.

3.5 State Machine

Finally, we get to the state machine part. First, I will show you my state machine diagram:

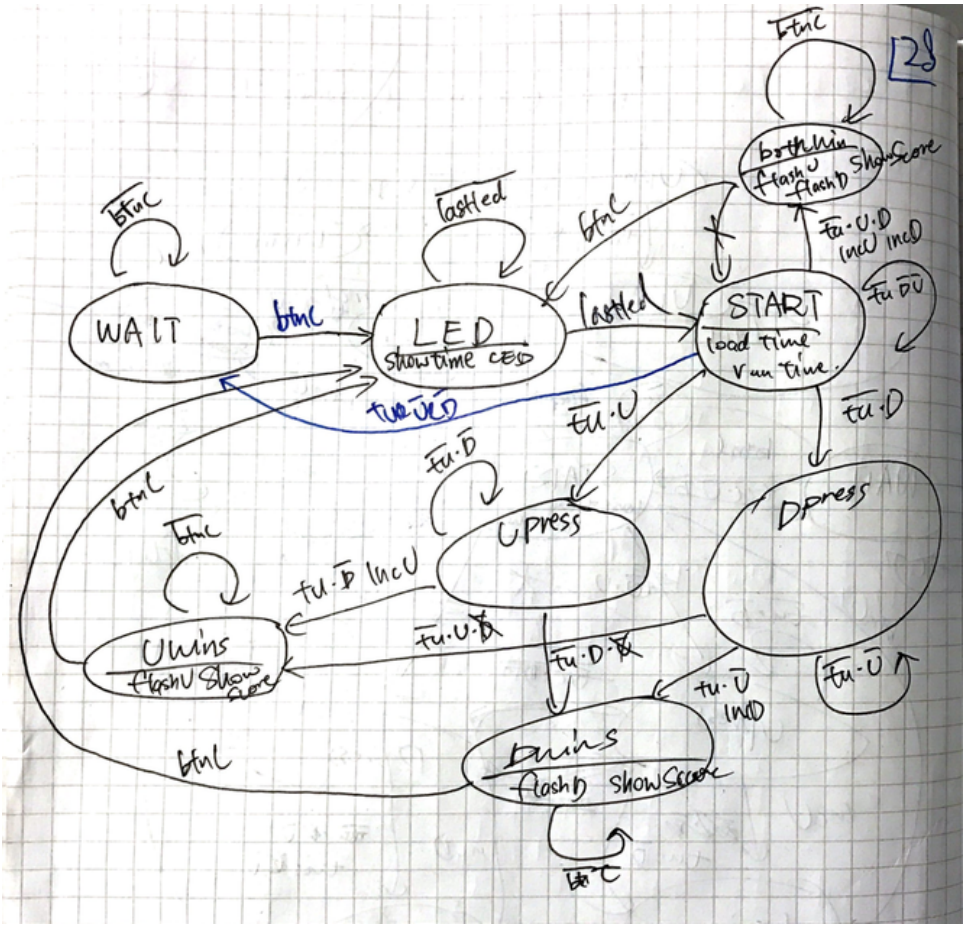

Figure 7: Screenshot of my state machine diagram.

At the very beginning, we would be at the WAIT state. If button C is not pressed, then we will stay at WAIT. If button C is pressed, then we will go to the next state, LED. In the LED state, we want to show the time generated from LFSR as well as enable the led shifter. The led shifter will do its job to start lighting from right to left one by one. If last led is checked, then we will move to START. If last led is not yet done, then we stay at LED.

Assume we are at START right now, if both U and D are pressed, that would be a both win case. We go to both win state.

At both win state, the flashes of U and D will both on because they both win, so does show score. In this state, if button C is not pressed, then we stay at both win. If button C is press, then we will go back to LED and start another round.

Assume we are back to START right now, if U is pressed, then we go to U press. In this state, if D is pressed before time up, then we go to D wins. If D is not pressed and time is up, then we go to U wins.

Assume we are back to START right now, if D is pressed, then we go to D press. In this state, if U is pressed before time up, then we go to U wins. If U is not pressed and time is up, then we go to D wins.

For U wins and D wins, the score of U and D will flash respectively. The score will be shown on the display. In all the winning state, if button C is not pressed, then we stay at both win. If button C is press, then we will go back to LED and start another round.

4 Conclusion

This lab was more practical than all the previous ones. I feel like we are not learning new knowledge; however, we are using the knowledge we have to solve more complex problems. The labs are getting harder. However, with the previous four labs experience, we would be able to get it done. Also, this is my first time designing my own state machine diagram. It was hard for me at the first time. I was stuck at the state machine part for days because the whole idea of state machine was new to me. It took me time to practice and try before the final version. Once I got one state machine done, I am confident to design a state machine for other labs. What I learned in this lab was that we could not rush. Especially in some critical parts, like designing state machine. If we miss one point or one state, we might need to start from the beginning. I recommend show your state machine design to anyone in CE100 or the teaching staffs before you precess forward. Overall, it was a good learning experience. Thank you for your time.

5 Appendixes

5.1 Screenshots of Lab notebook

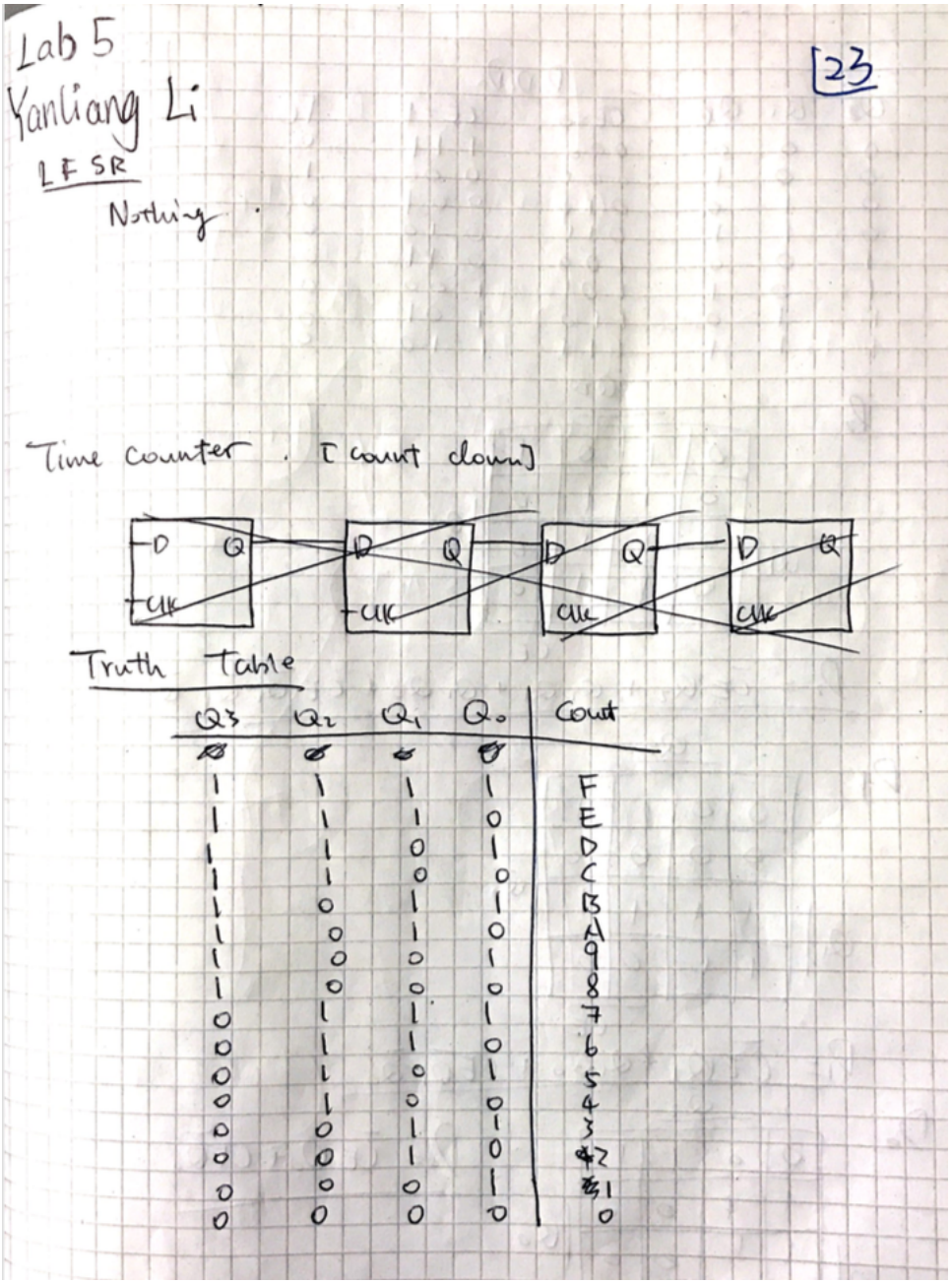

Figure 8: Screenshot of my lab notebook.

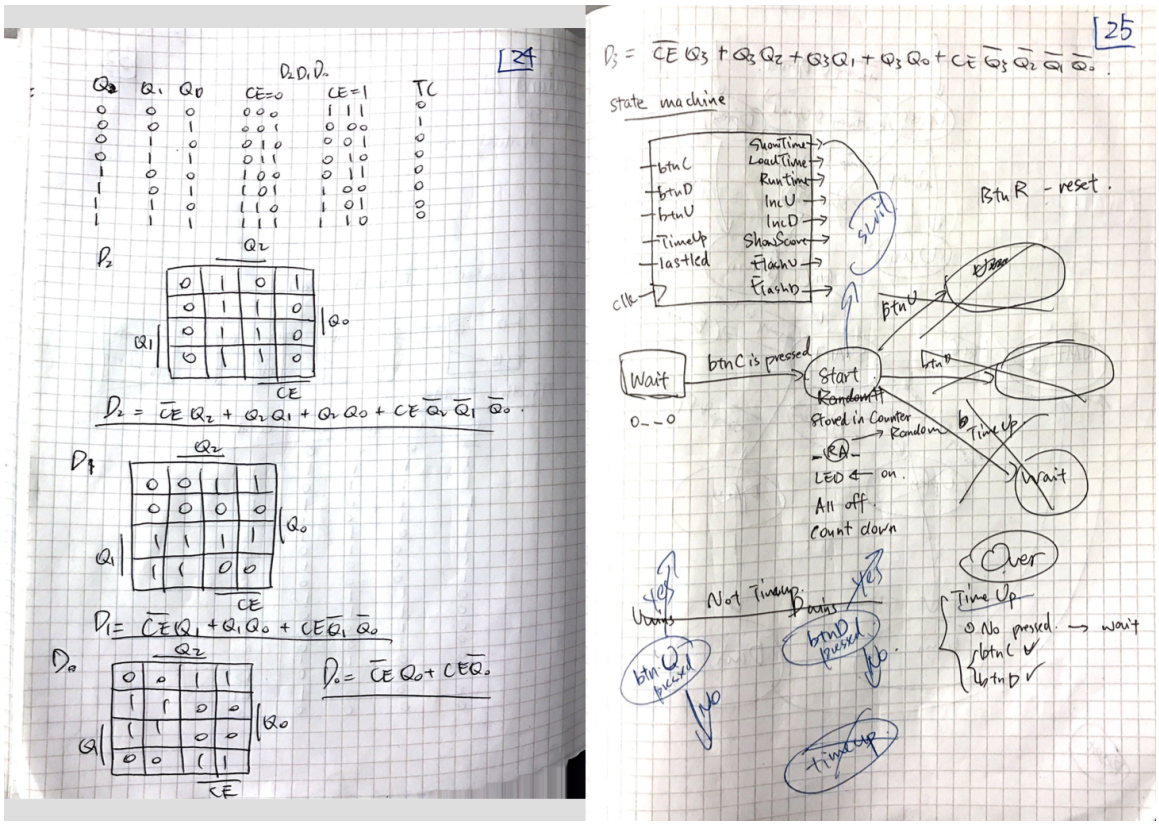

Figure 9: Screenshot of my lab notebook.

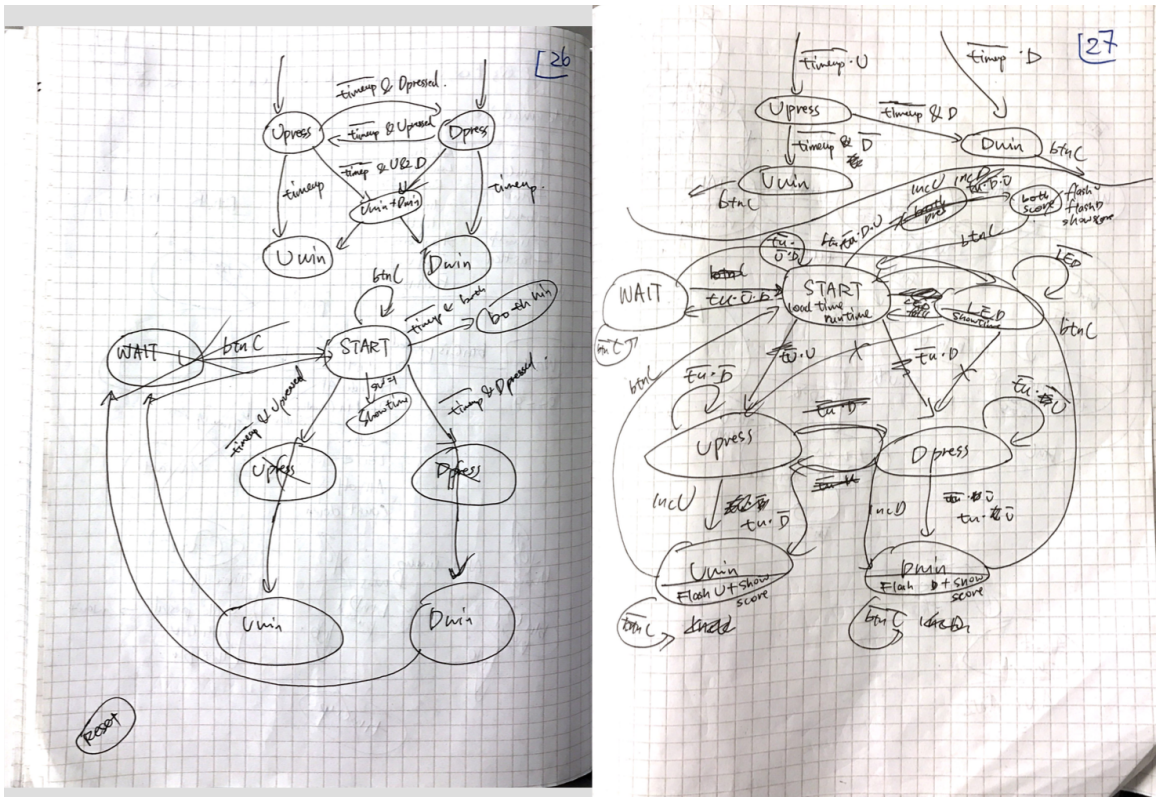

Figure 10: Screenshot of my lab notebook.

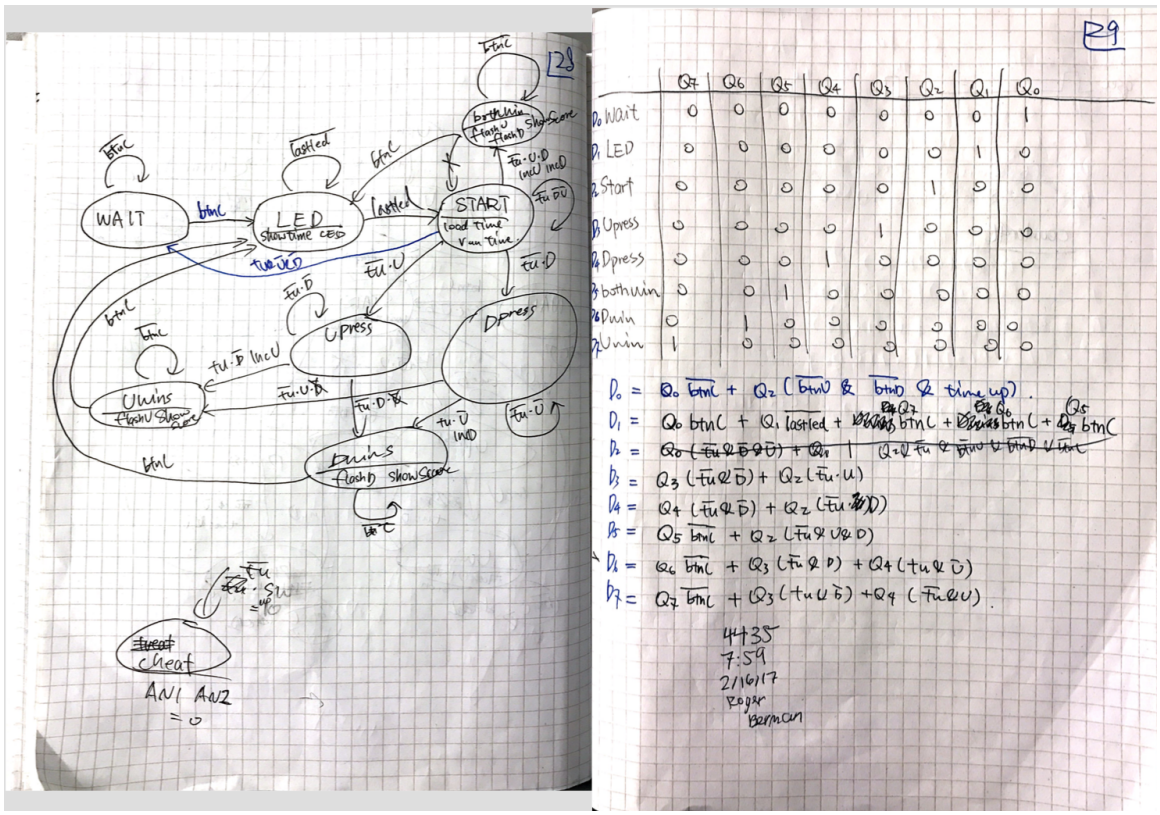

Figure 11: Screenshot of my lab notebook.

5.2 Verilogs

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 02/06/2017 04:42:50 PM
// Design Name:
// Module Name: hex7seg
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


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

    wire t0, t1;
    assign t0 = 0;
    assign t1 = 1;

    wire [2:0] sel;
    assign sel = n[3:1];

    wire [7:0] CA, CB, CC, CD, CE, CF, CG;

    assign CA = {t0, n[0], n[0], t0, t0, ~n[0], t0, n[0]};
    assign CB = {t1, ~n[0], n[0], t0, ~n[0], n[0], t0, t0};
    assign CC = {t1, ~n[0], t0, t0, t0, t0, ~n[0], t0};
    assign CD = {n[0], t0, ~n[0], n[0], n[0], ~n[0], t0, n[0]};
    assign CE = {t0, t0, t0, n[0], n[0], t1, n[0], n[0]};
    assign CF = {t0, n[0], t0, t0, n[0], t0, t1, n[0]};
    assign CG = {t0, ~n[0], t0, t0, n[0], t0, t0, t1};

    m8_1e seg0 (.in(CA[7:0]), .sel(sel), .e(e), .o(seg[0]) );
```

Figure 12: The verilog code of seven segment.

```
    m8_1e seg1 (.in(CB[7:0]), .sel(sel), .e(e), .o(seg[1]));
    m8_1e seg2 (.in(CC[7:0]), .sel(sel), .e(e), .o(seg[2]));
    m8_1e seg3 (.in(CD[7:0]), .sel(sel), .e(e), .o(seg[3]));
    m8_1e seg4 (.in(CE[7:0]), .sel(sel), .e(e), .o(seg[4]));
    m8_1e seg5 (.in(CF[7:0]), .sel(sel), .e(e), .o(seg[5]));
    m8_1e seg6 (.in(CG[7:0]), .sel(sel), .e(e), .o(seg[6]));


endmodule
```

Figure 12: The verilog code of seven segment.

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 02/16/2017 08:59:32 AM
// Design Name:
// Module Name: top_level
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

module top_level(
    input clkin,
    input btnR,
    input btnU,
    input btnC,
    input btnD,
    input sw15,
    output [15:0] led,
    output [3:0] an,
    output dp,
    output [6:0] seg
    );

    wire TU, lastled, showtime, loadtime, runtime, iU, iD, showscore, fU, fD, SW15;
    wire [15:0] Q16;

    wire [7:0] rnd;
    wire digsel, clk, qsec;
    wire [3:0] inSEL, outSEL;
    wire CEled, CEtime;
    wire flash;
    wire one;

    assign one = 1'b1;
    assign CEled =  qsec & showtime;
```

Figure 13: The verilog code of top level.

```
    assign CEtime = qsec & runtime;
    assign lastled = led[15];

    //lab5_clks
    lab5_clks clkstop ( .clkin(clkin), .greset(btnR), .clk(clk), .digsel(digsel),
.qsec(qsec));
    //state machine
    state_machine statetop (.clk(clk), .btnC(btnC), .btnU(btnU), .btnD(btnD),
.TimeUp(TU), .lastled(lastled), .ShowTime(showtime),
                            .LoadTime(loadtime), .RunTime(runtime), .IncU(iU),
.IncD(iD), .ShowScore(showscore), .FlashU(fU), .FlashD(fD));
    //led shifter
    led_shifter shifttop ( .clk(clk), .CE(CEled), .R(~showtime), .Q(led[15:0]));

    //scroe display
    score_display scoreU (.clk(clk), .CE(iU), .Q(Q16[15:12]));
    score_display scoreD (.clk(clk), .CE(iD), .Q(Q16[3:0]));

    //LFSR
    LFSR randomtop (.clk(clk), .R(rnd[7:0]));

    //time counter
    time_counter counttop (.clk(clk), .CE(CEtime), .LD(loadtime), .TimeUp(TU),
.Din(rnd[7:0]), .Q(Q16[11:4]));
   //ring counter
    ring_counter ringtop (.clk(clk), .Advance(digsel), .Q(inSEL[3:0]) );
    //seclector
    selector selecttop (.sel(inSEL[3:0]), .out(outSEL[3:0]), .in(Q16[15:0]));
    //hex7seg
    hex7seg hextop (.n(outSEL[3:0]), .e(one), .seg(seg[6:0]));


    FDRE #( .INIT(1'b0)) switch15 (.C(clk), .CE(one), .D(sw15), .Q(SW15));

    //delay, to delay the flash light
    delay delaytop (.clk(clk), .qsec(qsec), .Q(flash));

    assign an[3] = ~(inSEL[3] & (showscore| (fU & flash^fD)));
    assign an[2] = ~((SW15|showtime) & inSEL[2]);
    assign an[1] = ~((SW15|showtime) & inSEL[1]);
    assign an[0] = ~(inSEL[0] & (showscore|(fD & flash^fU)));
    assign dp = 1;

endmodule
```

Figure 13: The verilog code of top .

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 02/13/2017 03:13:46 PM
// Design Name:
// Module Name: Time_Counter
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

module time_counter(
    input CE,
    input LD,
    input [7:0] Din,
    input clk,
    output TimeUp,
    output [7:0] Q
    );


    wire SIGN;
    assign SIGN = 1'b1;
    wire [1:0] ce, tc;
    wire [7:0] inQ, outQ;
    assign ce[0] = CE;
    assign ce[1] = tc[0] & CE;

    signChanger invert (.sign(SIGN), .b(Din[7:0]), .d(inQ[7:0]));
    count4L count1 (.clk(clk), .CE(ce[0]), .LD(LD), .D(inQ[3:0]), .Q(outQ[3:0]),
.TC(tc[0]));
    count4L count2 (.clk(clk), .CE(ce[1]), .LD(LD), .D(inQ[7:4]), .Q(outQ[7:4]),
.TC(tc[1]));
    signChanger invert2 (.sign(SIGN), .b(outQ[7:0]), .d(Q[7:0]));

    assign TimeUp = ~Q[7] & ~Q[6] & ~Q[5] & ~Q[4] & ~Q[3] & ~Q[2] & ~Q[1] & ~Q[0];
```

Figure 14: The verilog code of time counter.

```
endmodule
```

Figure 14: The verilog code of time counter.

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 02/15/2017 01:00:43 PM
// Design Name:
// Module Name: state_machine_logic
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

module state_machine(
    input btnC,
    input btnU,
    input btnD,
    input TimeUp,
    input lastled,
    input clk,
    output ShowTime,
    output LoadTime,
    output RunTime,
    output IncU,
    output IncD,
    output ShowScore,
    output FlashU,
    output FlashD
    );

    wire [7:0] D;
    wire [7:0] Q;
    wire one;
    assign one = 1'b1;
    wire btnD1, btnC1, btnU1;

    FDRE #( .INIT(1'b0)) btnd (.C(clk), .CE(one), .D(btnD), .Q(btnD1));
    FDRE #( .INIT(1'b0)) btnc (.C(clk), .CE(one), .D(btnC), .Q(btnC1));
```

Figure 15: The verilog code of state machine.

```
    FDRE #( .INIT(1'b0)) btnu (.C(clk), .CE(one), .D(btnU), .Q(btnU1));

    assign D[0] = (Q[0] & ~btnC1) | (Q[2] & ~btnU1 & ~btnD1 & TimeUp);
    assign D[1] = (Q[1] & ~lastled) | (Q[5] & btnC1) | (Q[6] & btnC1) | (Q[7] &
btnC1) | (Q[0] & btnC1);
    assign D[2] = (Q[1] & lastled) | (Q[2] & ~TimeUp & ~btnU1 & ~btnD1 & ~btnC1);
    assign D[3] = (Q[3] & ~btnD1 & ~TimeUp) | (Q[2] & btnU1 & ~btnD1 & ~TimeUp);
    assign D[4] = (Q[4] & ~btnU1 & ~TimeUp) | (Q[2] & btnD1 & ~btnU1 & ~TimeUp);
    assign D[5] = (Q[5] & ~btnC1) | (Q[2] & btnU1 & btnD1 & ~TimeUp);
    assign D[6] = (Q[6] & ~btnC1) | (Q[3] & btnD1 & ~TimeUp) | (Q[4] & TimeUp);
    assign D[7] = (Q[7] & ~btnC1) | (Q[4] & btnU1 & ~TimeUp) | (Q[3] & TimeUp);


    assign ShowTime = Q[1] & ~lastled;
    assign LoadTime = Q[0] & btnC1 | Q[2] & btnC1 | Q[5] & btnC1 | Q[6] & btnC1 |
Q[7] & btnC1;
    assign RunTime = Q[2] | Q[3] |
Q[4]|(Q[2]&btnU1&~btnD1&~TimeUp)|(Q[3]&btnD1&~TimeUp)|(Q[2]&btnD1&~btnU1&~TimeUp)|(Q[
4]&btnU1&~TimeUp);
    assign IncU = (Q[2] & btnU1 & btnD1 & ~TimeUp) | (Q[4] & btnU1 & ~TimeUp) | (Q[3]
& TimeUp);
    assign IncD = (Q[2] & btnU1 & btnD1 & ~TimeUp) | (Q[4] & TimeUp) | (Q[3] & btnD1
& ~TimeUp);
    assign ShowScore = Q[0] ;
    assign FlashU = Q[7] | Q[5];
    assign FlashD = Q[6] | Q[5];

    FDRE #( .INIT(1'b1)) state0 (.C(clk), .CE(one), .D(D[0]), .Q(Q[0]));
    FDRE #( .INIT(1'b0)) state[7:1] (.C({7{clk}}), .CE({7{one}}), .D(D[7:1]),
.Q(Q[7:1]));


endmodule
```

Figure 15: The verilog code of state machine.

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 02/13/2017 05:14:01 PM
// Design Name:
// Module Name: count4L
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////



module count4L(
    input clk,
   input CE,
   input LD,
   input [3:0] D,
   output [3:0] Q,
   output TC
   );

   wire [3:0] inD;

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

   assign TC = Q[3] & Q[2] & Q[1] & Q[0];

   FDRE #( .INIT(1'b0)) zero (.C(clk), .CE(LD|CE), .D(inD[0]), .Q(Q[0]));
   FDRE #( .INIT(1'b0)) one (.C(clk), .CE(LD|CE), .D(inD[1]), .Q(Q[1]));
   FDRE #( .INIT(1'b0)) two (.C(clk), .CE(LD|CE), .D(inD[2]), .Q(Q[2]));
   FDRE #( .INIT(1'b0)) three (.C(clk), .CE(LD|CE), .D(inD[3]), .Q(Q[3]));
```

Figure 16: The verilog code of counter4L.

```
endmodule
```

Figure 16: The verilog code of counter4L.

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 02/13/2017 03:13:20 PM
// 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,
    output [7:0] R
    );

    wire one;
    assign one = 1'b1;

    wire random;

    assign random = rnd[0]^rnd[5]^rnd[6]^rnd[7];
    FDRE #(.INIT(1'b1)) rand0 (.C(clk), .CE(one), .D(random), .Q(rnd[0]));
    FDRE #(.INIT(1'b0)) rand1 (.C(clk), .CE(one), .D(rnd[0]), .Q(rnd[1]));
    FDRE #(.INIT(1'b0)) rand2 (.C(clk), .CE(one), .D(rnd[1]), .Q(rnd[2]));
    FDRE #(.INIT(1'b0)) rand3 (.C(clk), .CE(one), .D(rnd[2]), .Q(rnd[3]));
    FDRE #(.INIT(1'b0)) rand4 (.C(clk), .CE(one), .D(rnd[3]), .Q(rnd[4]));
    FDRE #(.INIT(1'b0)) rand5 (.C(clk), .CE(one), .D(rnd[4]), .Q(rnd[5]));
    FDRE #(.INIT(1'b0)) rand6 (.C(clk), .CE(one), .D(rnd[5]), .Q(rnd[6]));
    FDRE #(.INIT(1'b0)) rand7 (.C(clk), .CE(one), .D(rnd[6]), .Q(rnd[7]));

    assign R ={1'b0, rnd[6:0]};
```

Figure 17: The verilog code of LFSR.

```
endmodule
```

Figure 17: The verilog code of LFSR

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 02/13/2017 05:28:38 PM
// Design Name:
// Module Name: FullAdder
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module FullAdder(
  input cin,
  input a,b,
  output s,
  output cout
  );
  wire e;
  assign e = 1;
  wire zero = 0;

  wire [3:0] x;
  wire [1:0] y;
  wire [3:0] w;
  assign x = {cin,~cin,~cin,cin};
  assign y = {a, b};

  assign w = {e, cin, cin, zero};

  m4_1e sum (.o(s), .in(x[3:0]), .sel(y[1:0]), .e(e) );
  m4_1e carry (.o(cout), .in(w[3:0]), .sel(y[1:0]), .e(e));


endmodule
```

Figure 18: The verilog code of full adder.

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 02/14/2017 12:30:58 PM
// Design Name:
// Module Name: delay
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////



module delay(
    input clk,
    input qsec,
    output Q
    );

    wire [1:0] q;
    FDRE #(.INIT(1'b1) ) ff_instance_1095 (.C(clk), .CE(qsec),.R(1'b0), .D(q[1]),
.Q(q[0]));
    FDRE #(.INIT(1'b0) ) ff_instance_1096 (.C(clk), .CE(qsec),.R(1'b0), .D(q[0]),
.Q(q[1]));
    assign Q = q[1];

endmodule
```

Figure 19: The verilog code of delay flash.

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 01/24/2017 12:33:16 PM
// Design Name:
// Module Name: eight_bit_adder
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module eight_bit_adder(
    input [7:0] a,
    output [7:0] s
    );

    wire [7:0] t;
    wire [7:0] invert;
    assign invert = ~a;
    wire one;
    assign one=1;
    wire zero;
    assign zero=0;


    FullAdder add0 ( .s(s[0]), .a(invert[0]), .b(zero), .cin(one), .cout(t[0]));
    FullAdder add1 ( .s(s[1]), .a(invert[1]), .b(zero), .cin(t[0]), .cout(t[1]));
    FullAdder add2 ( .s(s[2]), .a(invert[2]), .b(zero), .cin(t[1]), .cout(t[2]));
    FullAdder add3 ( .s(s[3]), .a(invert[3]), .b(zero), .cin(t[2]), .cout(t[3]));
    FullAdder add4 ( .s(s[4]), .a(invert[4]), .b(zero), .cin(t[3]), .cout(t[4]));
    FullAdder add5 ( .s(s[5]), .a(invert[5]), .b(zero), .cin(t[4]), .cout(t[5]));
    FullAdder add6 ( .s(s[6]), .a(invert[6]), .b(zero), .cin(t[5]), .cout(t[6]));
    FullAdder add7 ( .s(s[7]), .a(invert[7]), .b(zero), .cin(t[6]), .cout(t[7]));
endmodule
```

Figure 20: The verilog code of eight bit adder.

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 02/13/2017 05:30:22 PM
// Design Name:
// Module Name: m4_1e
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module m4_1e(
  input [3:0] in,
  input [1:0] sel,
  input e,
  output o
  );

  assign o = (e & ~sel[1] & ~sel[0] & in[0]) | (e & ~sel[1] & sel[0] & in[1]) | (e &
sel[1] & ~sel[0] & in[2]) | (e & sel[1] & sel[0] & in[3]);

endmodule
```

Figure 21: The verilog code of four bit counter.

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 02/06/2017 07:43:49 PM
// 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
    );

    assign o = ( e & (~sel[2] & ~sel[1] & ~sel[0]) & in[0]) | ( e & (~sel[2] &
~sel[1] & sel[0]) & in[1]) | ( e & (~sel[2] & sel[1] & ~sel[0]) & in[2])
                | ( e & (~sel[2] & sel[1] & sel[0]) & in[3]) | ( e & (sel[2] &
~sel[1] & ~sel[0]) & in[4]) | ( e & (sel[2] & ~sel[1] & sel[0]) & in[5])
                | ( e & (sel[2] & sel[1] & ~sel[0]) & in[6]) | ( e & (sel[2] & sel[1]
& sel[0]) & in[7]);
endmodule
```

Figure 22: The verilog code of eight bit counter.

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 02/13/2017 05:05:35 PM
// Design Name:
// Module Name: signChanger
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module signChanger(
   input sign,
   input [7:0] b,
   output [7:0] d
   );
   wire [7:0] new;
   wire [7:0] sign0;
   assign sing0={8{sign}};

   eight_bit_adder TwosComp ( .s(new[7:0]), .a(b[7:0]) );
   assign d = (sign0 & new) | (~(sign0) & b);

endmodule
```

Figure 23: The verilog code of sign changer.

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 02/16/2017 10:22:01 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] in,
   output [3:0] out
   );
   wire [3:0]s;

   assign s[0] = ~sel[3] & ~sel[2] & ~sel[1] & sel[0];
   assign s[1] = ~sel[3] & ~sel[2] & sel[1] & ~sel[0];
   assign s[2] = ~sel[3] & sel[2] & ~sel[1] & ~sel[0];
   assign s[3] = sel[3] & ~sel[2] & ~sel[1] & ~sel[0];



   assign out = ({4{s[0]}} & in[3:0]) | ({4{s[1]}} & in[7:4]) | ({4{s[2]}} &
in[11:8]) | ({4{s[3]}} & in[15:12]);

endmodule
```

Figure 24: The verilog code of selector.

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 02/15/2017 12:18:51 PM
// Design Name:
// Module Name: score_display
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module score_display(
    input CE,
    input clk,
    output [3:0] Q
    );

    wire [3:0] inD;

    assign inD[3] = CE & (Q[3]^Q[2] & Q[1] & Q[0]);
    assign inD[2] = CE & (Q[2]^Q[1] & Q[0]);
    assign inD[1] =  CE & (Q[1]^Q[0]);
    assign inD[0] = (CE ^ Q[0]);

    FDRE #( .INIT(1'b0)) zero (.C(clk), .CE(CE), .D(inD[0]), .Q(Q[0]));
    FDRE #( .INIT(1'b0)) one (.C(clk), .CE(CE), .D(inD[1]), .Q(Q[1]));
    FDRE #( .INIT(1'b0)) two (.C(clk), .CE(CE), .D(inD[2]), .Q(Q[2]));
    FDRE #( .INIT(1'b0)) three (.C(clk), .CE(CE), .D(inD[3]), .Q(Q[3]));
endmodule
```

Figure 25: The verilog code of score display.

```
                                                `timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 02/13/2017 03:14:03 PM
// Design Name:
// Module Name: LED_Shifter
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////

module led_shifter(
    input CE,
    input R,
    input clk,
    output [15:0] Q
    );


    FDRE #(.INIT(1'b0)) LEDshift0 (.C(clk), .CE(CE), .R(R), .D(CE), .Q(Q[0]));
    FDRE #(.INIT(1'b0)) LEDshift15to1 (.C({15{clk}}), .CE({15{CE}}), .R({15{R}}),
.D(Q[14:0]), .Q(Q[15:1]));
endmodule
```

Figure 26: The verilog code of led shifter.

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 02/06/2017 03:33:33 PM
// Design Name:
// Module Name: ring
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


module ring_counter(
    input Advance,
    input clk,
    output [3:0] Q
    );
////////////////////ring
    FDRE #( .INIT(1'b1)) ring0 (.C(clk), .CE(Advance), .D(Q[3]), .Q(Q[0]));
    FDRE #( .INIT(1'b0)) ring1 (.C(clk), .CE(Advance), .D(Q[0]), .Q(Q[1]));
    FDRE #( .INIT(1'b0)) ring2 (.C(clk), .CE(Advance), .D(Q[1]), .Q(Q[2]));
    FDRE #( .INIT(1'b0)) ring3 (.C(clk), .CE(Advance), .D(Q[2]), .Q(Q[3]));

endmodule
```

Figure 27: The verilog code of ring coutner.

////////////////////////////////////

5.3 Simulation

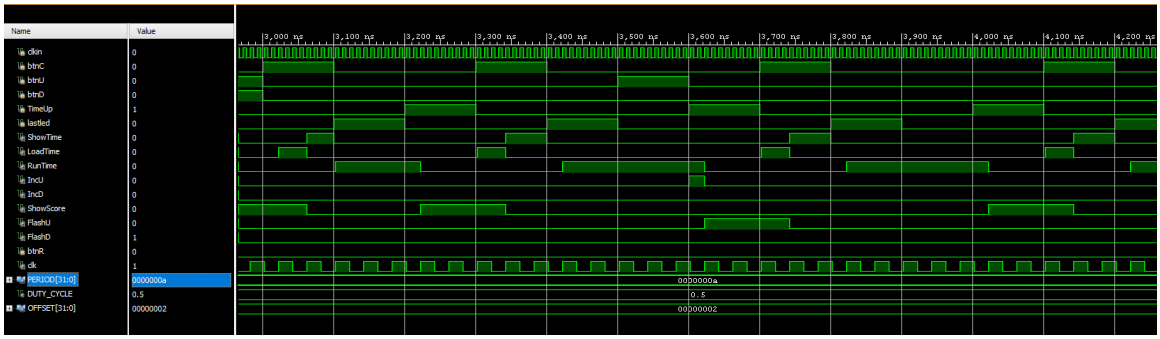

Figure 28: The top-level schematic.

5.4 Schematic

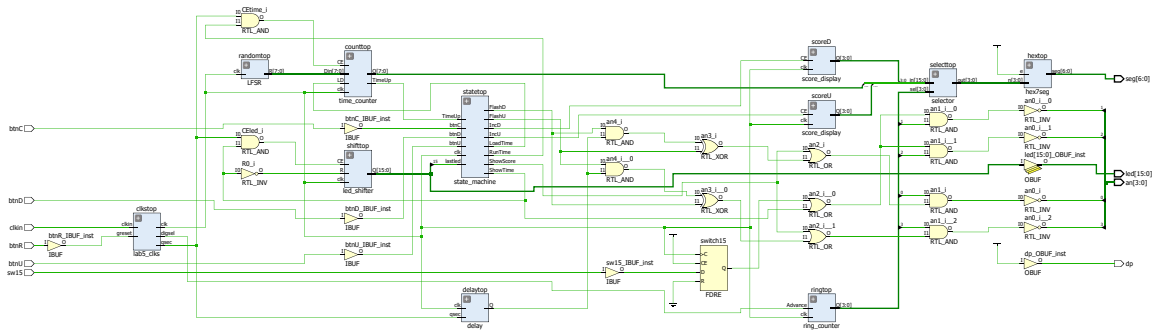

Figure 29: The top-level schematic.

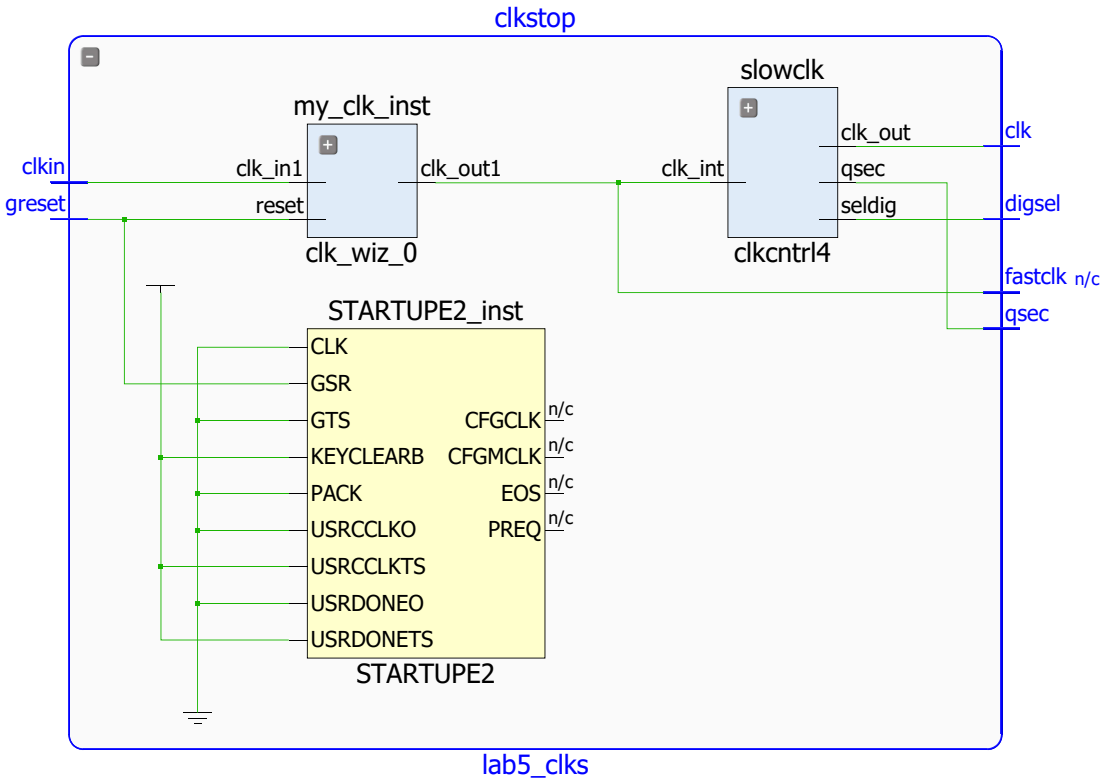

Figure 30: The schematic for clks.

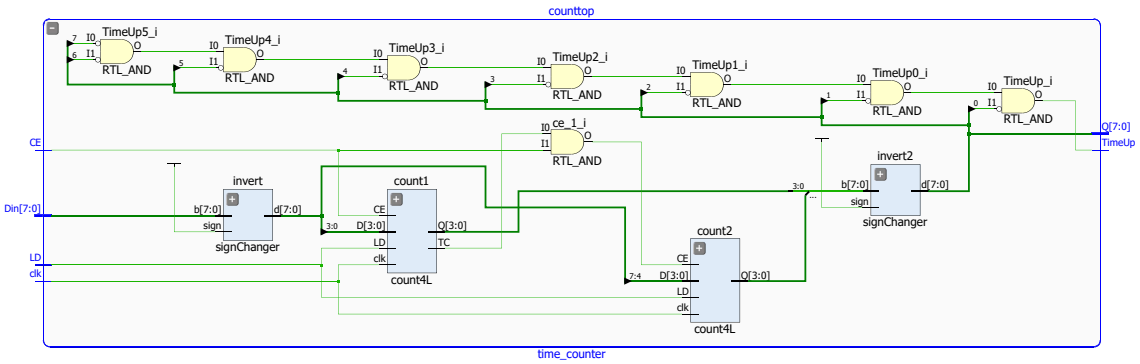

Figure 31: The schematic for counter.

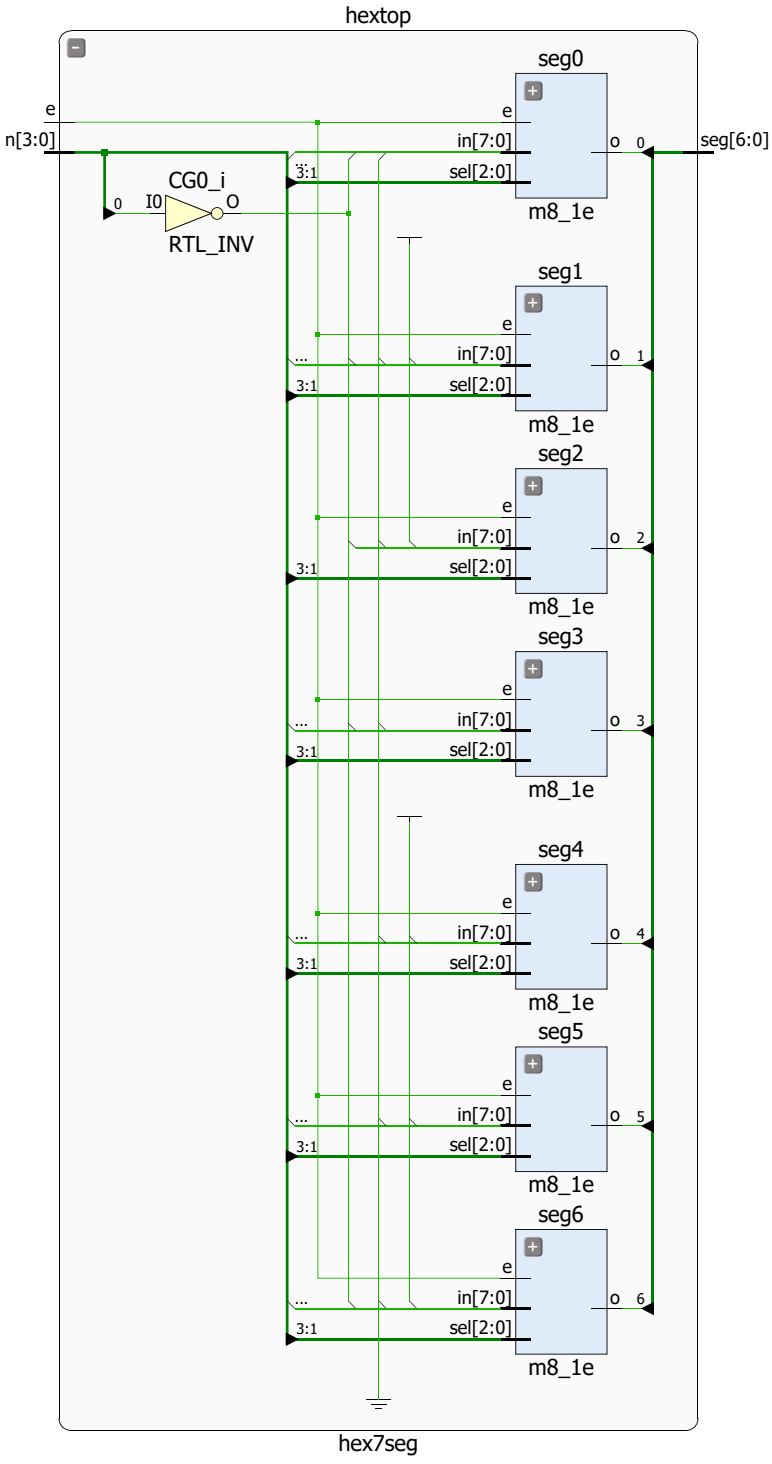

Figure 32: The schematic for hex.

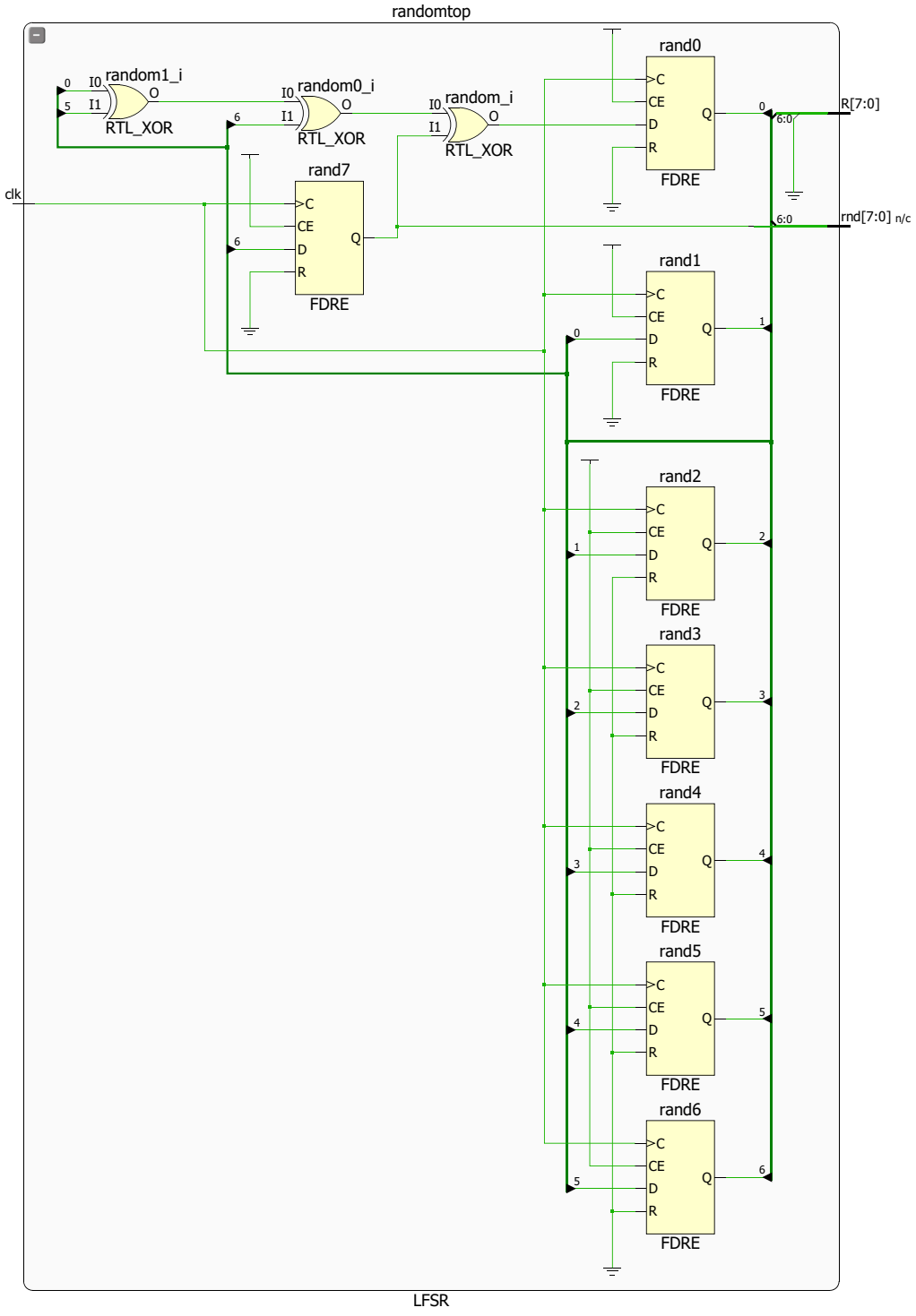

Figure 33: The schematic for LFSR.

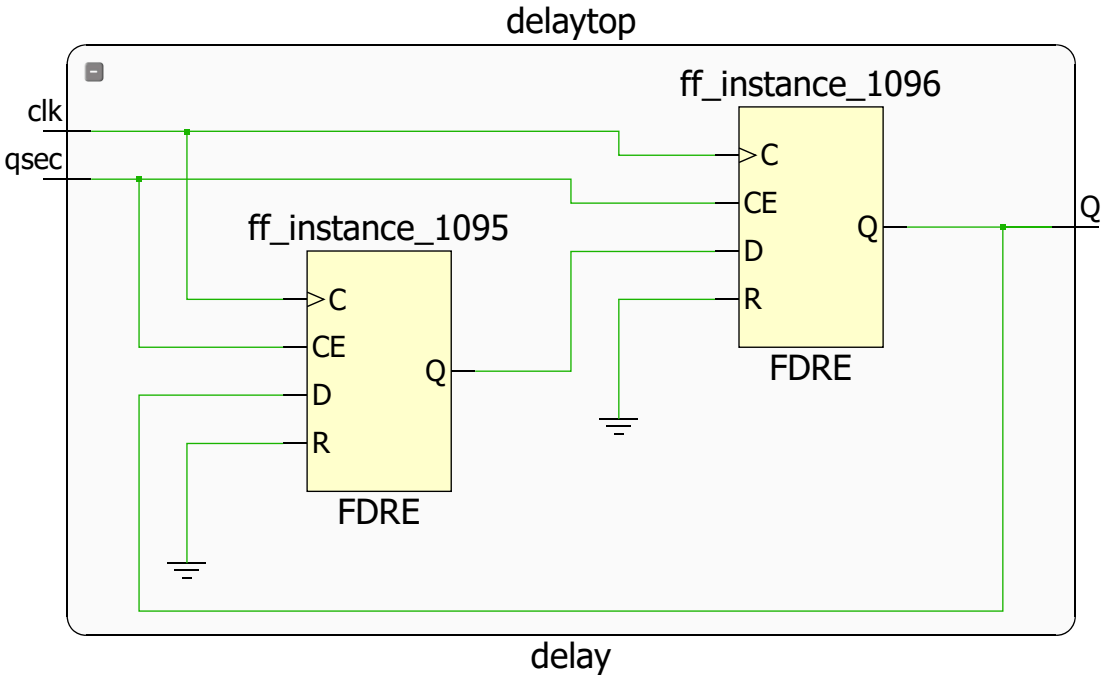

Figure 34: The schematic for delay.

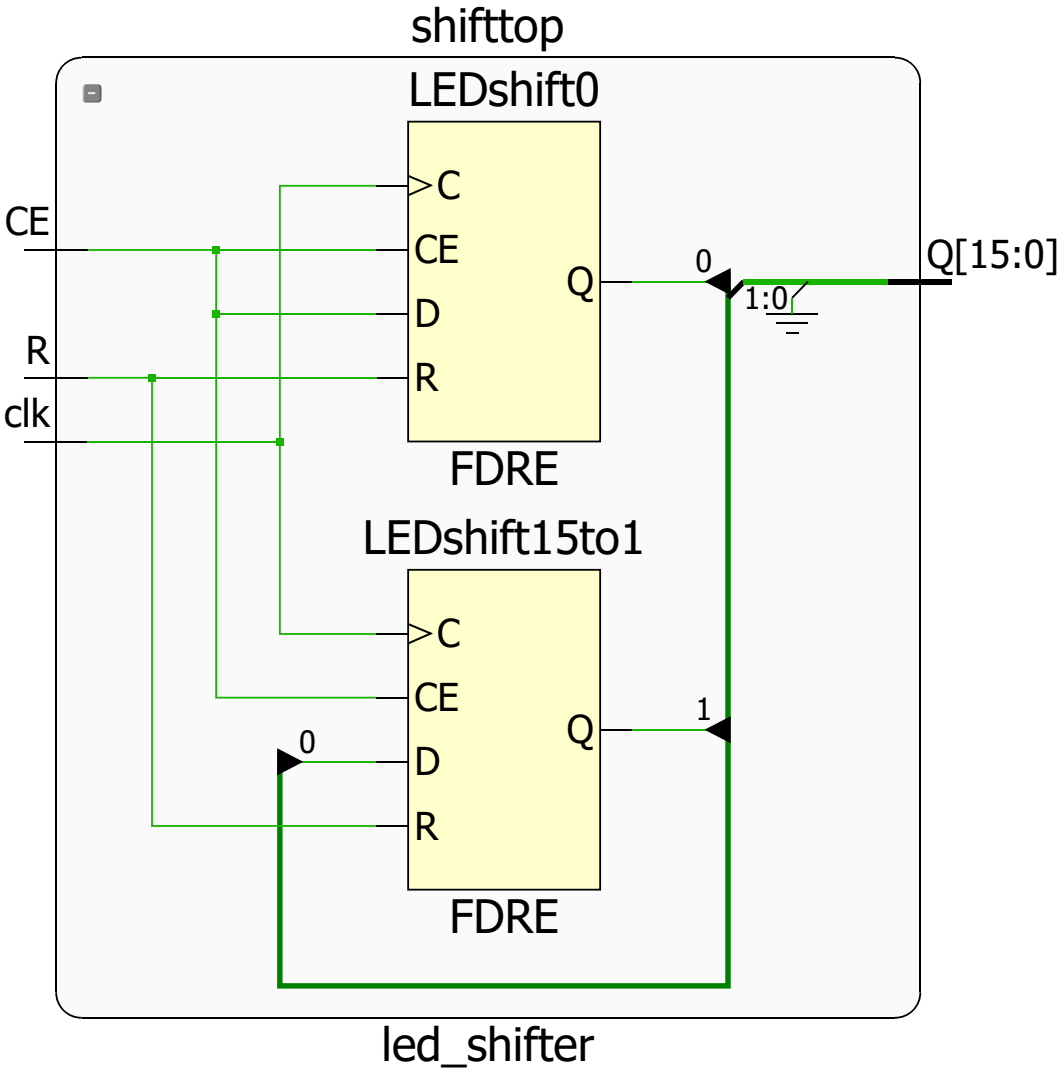

Figure 35: The schematic for led shifter.

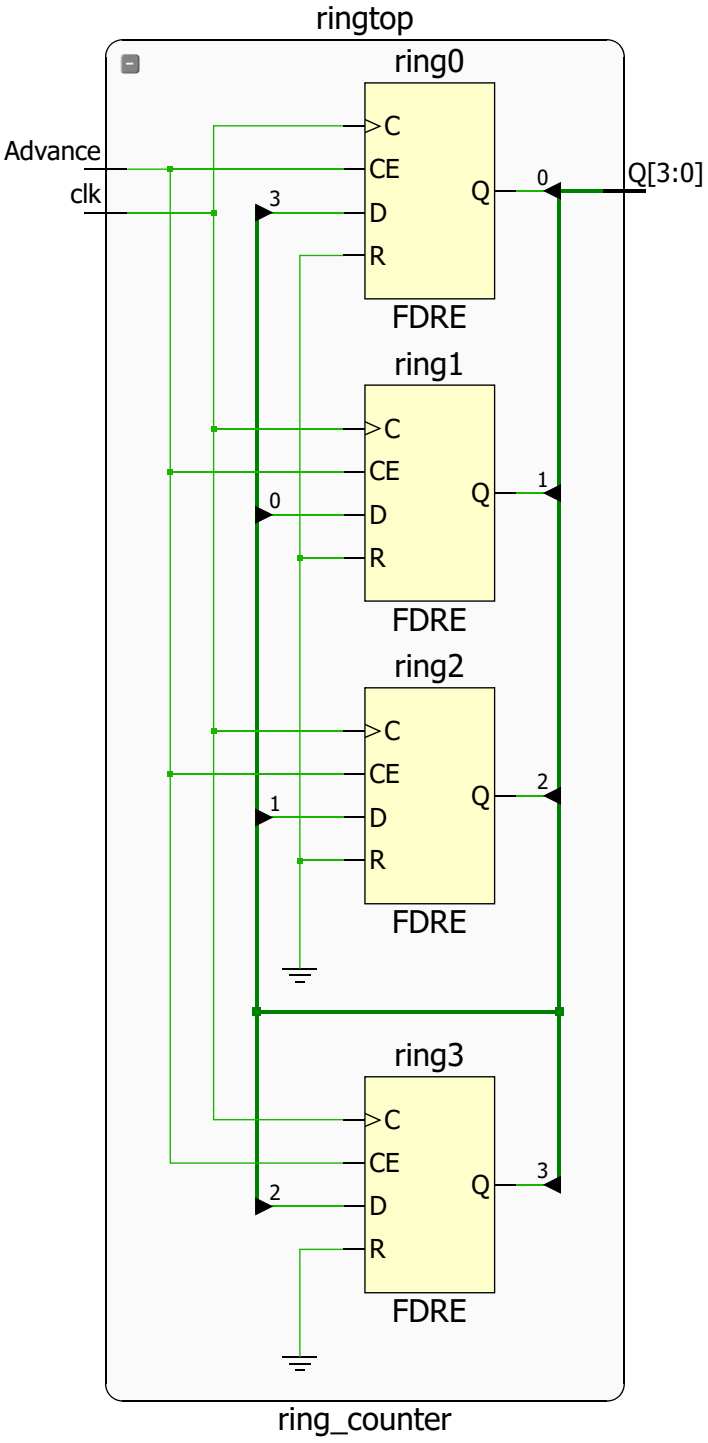

Figure 36: The schematic for ring coutner.

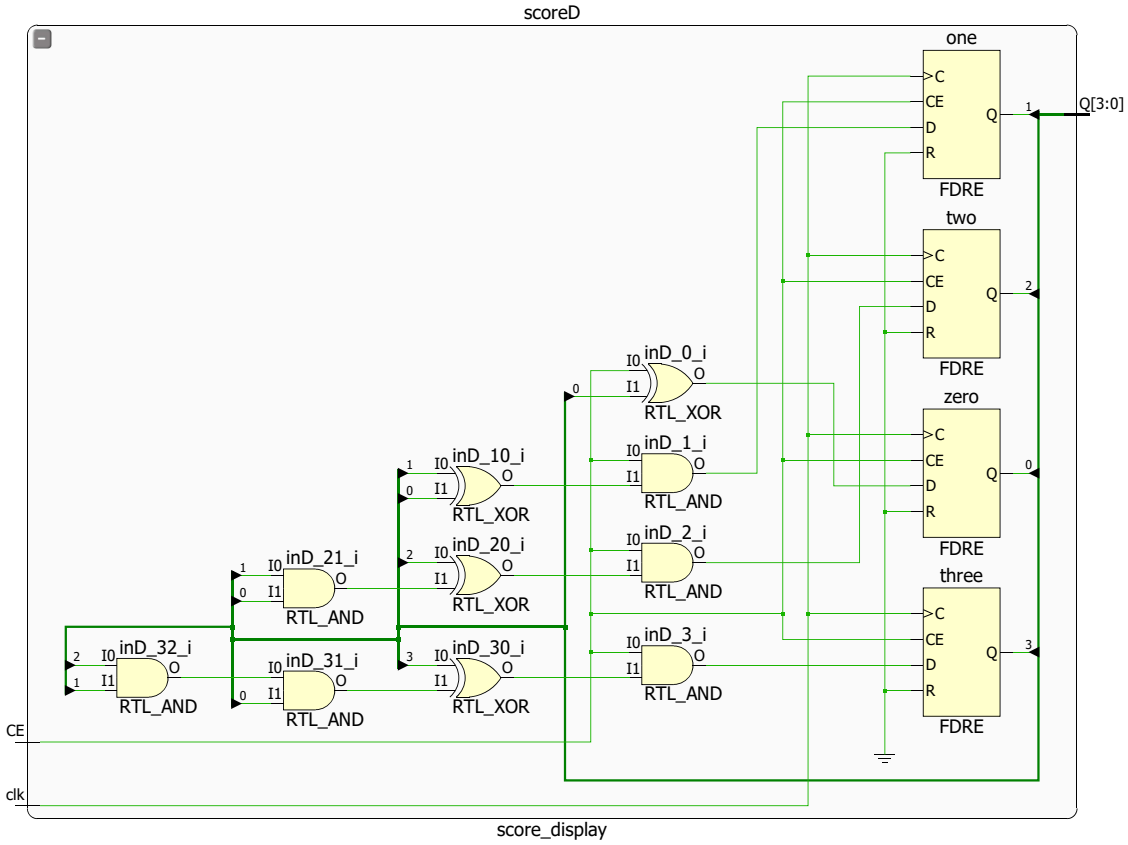

Figure 37: The schematic for score D.

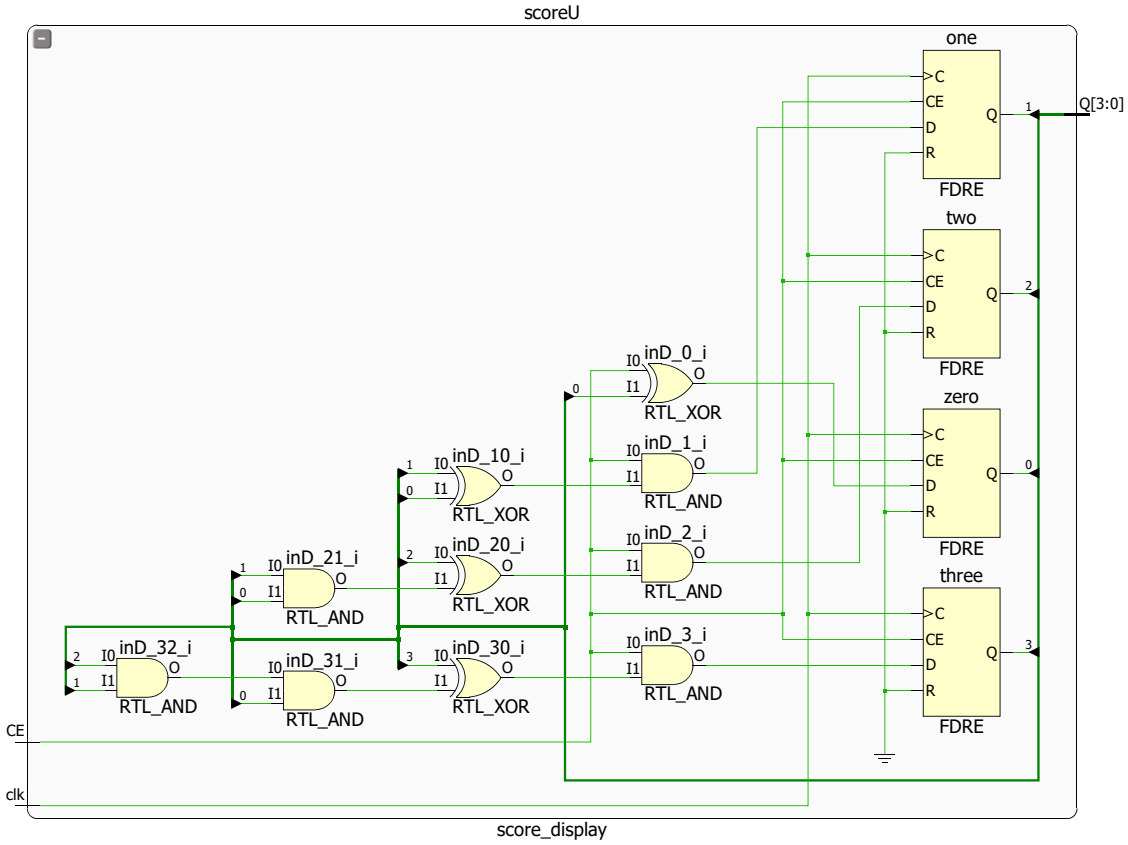

Figure 38: The schematic for scoreU.

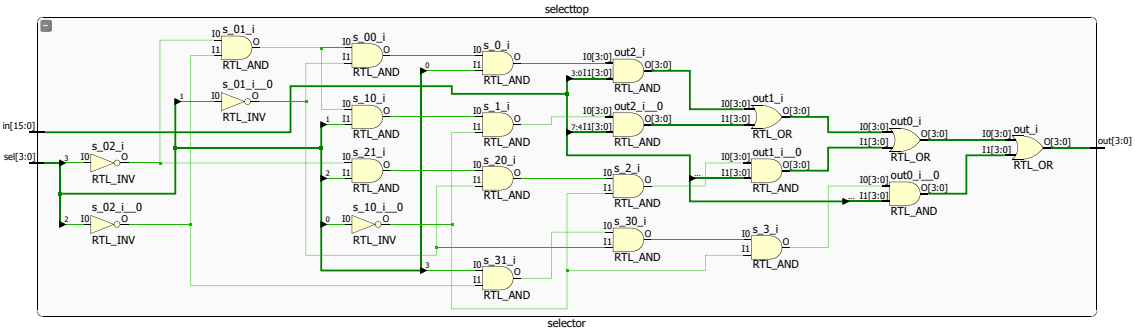

Figure 39: The schematic for slector.

6 Reference

References

[1] M. Schlag. CMPE 100 Lab 1. Winter 2017.