David Nguyen

Lab 4

Section C

11/9/2018

Lab 4 Writeup

Purpose:

The purpose of this lab was to learn how to build a loadable counter that counted up or down depending on up/down button presses or continuously up counted if the center button was pressed. I learned how create a loadable 3,5, and 16 bit adder using flipflops and logic gates. I also learned how to create a ring counter, selector, and edge detector using flipflops and muxes. I also used the hex7seg module in the previous lab.

Methods:

Counter 3 / 5 / 16:

1. The first step to creating a truth table for countUD3L, with inputs Q2, Q1, and Q0, and CE toggling on as 1 or 0 affecting the 3 Q bits outputs. The next step is to create a K-map for each column of the outputs for the next state which is when CE is on. This can be simplified to the equation Dn = Qn ^ (Qn-1 & Qn-2 …… & Qn-n). This can also be performed for the down counter and the equation Dn = ~(Qn ^ (Qn-1 | Qn-2 | …|Qn-n)).
2. To take care of the logic for Up when btnU, Dw for btnD, and nothing for when btnD&btnU or ~btnD&~btnU. A 4_1mux is used to select the output for {do nothing, add, subtract, do nothing} with the selector of the mux as {Dw, Up}.
3. To make sure the counter is loadable, the output of the 4_1mux is and the bus Din then inputted into a 2_1mux with LD as the selector.
4. This is finally put into the flip flop for the corresponding bit whose output is the used for the next logic equation.
5. To create a 16 bit counter, there are 2 instances each of the 3bit and 5 bit counters, with the same inputs going to each of the counters with additional logic for up and down. UTC should be on when all of the bits are at 1 and DTC should be on all the bits are 0 for all counters.

Selector:

1. The first part which takes in a 4 and 16 bit bus as inputs and outputs a 4 bit bus, is to figure out logic for the selector. The purpose of the selector is:
    - H is x[15:12] when sel=(1000)
    - H is x[11:8] when sel=(0100)
    - H is x[7:4] when sel=(0010)
    - H is x[3:0] when sel=(0001)

- Used from Lab 4 manual

2. With ring counter as the input for the selector, it picks the corresponding 4 bits to then be sent to hex7seg.
3. This is done by anding the 4 bits that correspond to the selector for each of the 4 bits, and then xoring them all together. Because only 1 input in sel can be one, xoring them together means it will output corresponding bits where sel is on.

Ring Counter:

1. The first part of creating the ring counter with inputs clk, advance, reset and outputs a 4 bit bus is understand the purpose of the ring counter. Every time advance is on the output is shifted rotated by 1 bit, for example, 1000 become 01000, and 0001 becomes 1000.
2. This is done by setting 4 flipflops with the first 1 unitized to 1 and the rest to 0. The output is then daisy chained to the input of the next flip flop with the output of the last flip flop daisy chained into the first flipflop.

Edge Detector:

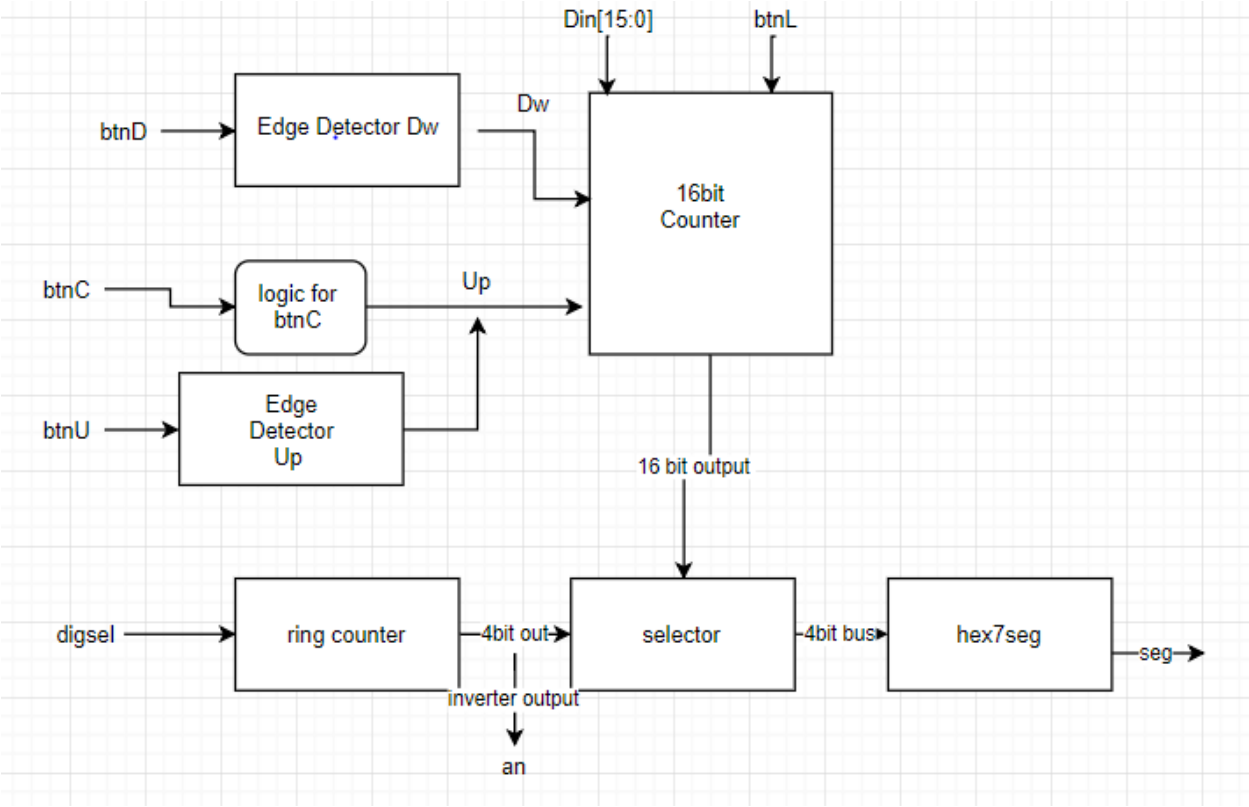
1. The first part of creating the edge detector is create 2 flip flops with the input of the first one be the button press. The input of the second flipflop and output of the first flipflop is assigned to t0 and the output of the second flip flop is assigned to t1.
2. The output is then assigned to t0 & ~ t1. This detects only 1 button press and will not activate until the button is released and pressed again.

Top Level Schematic/ Simulation

1. The top module has inputs clkin, btnR, input btnU, btnD, btnC, btnL, [15:0]sw, and outputs [6:0] seg, dp, [3:0]an,[15:0]led. First set led to sw.
2. The next step is to connect modules and to create logic for Edge Detector, Ring Counter, 16-bit Counter, Selector and hex7seg.
3. The Verilog file lab4_clks.v is also needed and the following line should be pastes in your code:
   lab4_clks slowit (.clkin(clkin), .greset(btnR), .clk(clk), .digsel(digsel));
   clk will now be used to synchronize all of your modules that require a clk input
4. To make sure the counter only goes up when btnC is pressed and not in the range of FFFC- FFFF or up is pressed. You can & every bit from 15-3 inside of the 16 bit counter and check if to see btnC is pressed which is then or with the edge detector of btnU.
5. The leftmost decimal point is lit when the value is FFFF and the right most is lit when the value is 0000.
6. The counter value is then put into selector which is then outputted to hex7seg.
7. The next step is to simulate and run the design by checking the value of counter in the wave viewer setting btnU,C,D, and L, indifferent combinations to check if the output is correct.

Results

1. In this lab when btnR is pressed, it resets the counter value to zero. When btnR is held down, only an[1] turns on and stays at zero without any flickering. This is because btnR also resets the ring counter which output will always be the same. Meaning the selector will always display the same number which is reset to 0.
2. When testing fastclk as the clk input instead of clk the counter counts up even quicker than before because it is a faster clk. Because it is a way faster clock and btnC is pressed, the counter counts up faster than digsel so it goes instantly to FFFC. There is also 3 faint an and 1 bright an because digsel is based of clock but the rest is based off of fastclk. This means the sel is out of sync.



In this design, Din[15-0] and btnL go directly into the 16 bit counter. btnD and btnU go into their respective edge detectors. The edge detector output for down is put into 16 bit counter as Dw and the Up input for the 16 bit counter is the logic for btnC and output for Up edge detector. The ring counter accepts digsel as input and outputs a 4 bit bus that goes into selector. An is the inverted ouput of the ring counter. The selector takes in the 16bit output from the counter and 4bit bus from ring counter and outputs to hex7seg which then outputs the result to be displayed in seg.

To simulate the design for our 3 different counters, I would set Up and Dw on and off frequently and checked the output of the counters. I also tested for pressing both buttons at once and pressing nothing at once. This worked for the 3 and 5 bit counter. But in ran into problems for the 16 bit counter, because created the 16 bit counter using 3 and 5 bit counters, there was a problem connecting my up and down from the first module and the rest of the modules. To fix this, I compounded the Up and down to be t4 = Up & ~Dw & tempUTC[2]& tempUTC[0]& tempUTC[1] and t5 = Dw & ~Up & tempDTC[2]& tempDTC[0]& tempDTC[1]. With temp DTC corresponding to the previous modules and t4 going into the Up and t5 going into the next Dw. I also tried loading data into the simulation and it work as expected.

Testing topMod was also similar , I would set Up and Dw on and off frequently and checked the output of the counter in addition to also testing for btnC. I had problems with the display which would be debugged by using the sim such as the getting logic for dp and inverting the input for an.

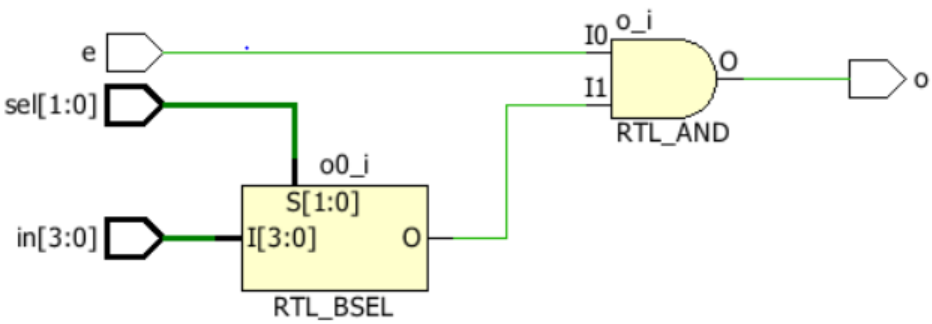
Conclusion

In conclusion, this lab I learned how to create a loadable up and down counter while adding modules together. The lab first started out by creating the counter modules and adding them together to create the 16 bit counters. The next step was to create the other smaller modules to pick what to display or an edge detector. Finally, the smaller modules are connected to the input and outputs of the modules to display to the board. If I were to do this lab again, I would create better truth tables to obtain the equations for adding and subtracting. I could just assume that CE would be pressed because if it was zero it would just be itself. One component I would optimize would be my counters. In this example I assume that up and dw both happen and then I use a mux to pick which one happens. I could optimize this by creating a k map that includes up and dw.

Supplementary material

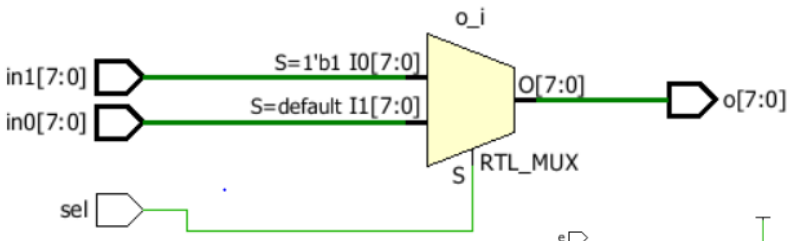
Mux 4x1

```
module m4_1e(
    input [3:0] in,
    input [1:0] sel,
    input e,
    output o
    );
   //assign o = e & ((in[0] &
    assign o = e & (in[sel]);
endmodule
```



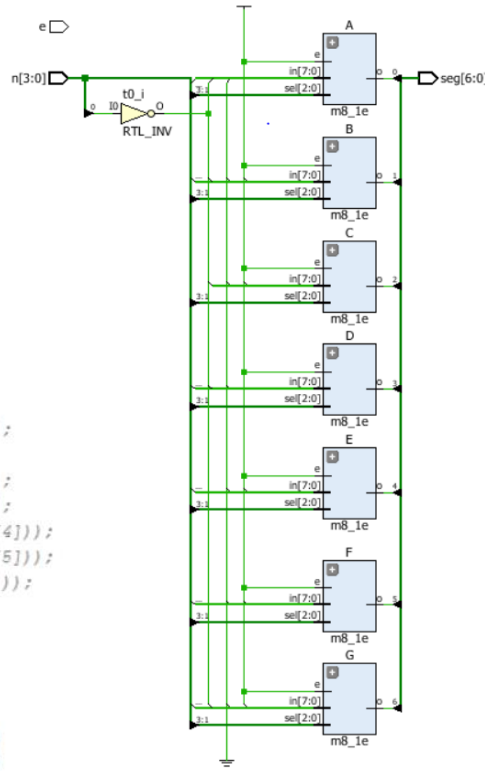
Mux 2_8x1

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



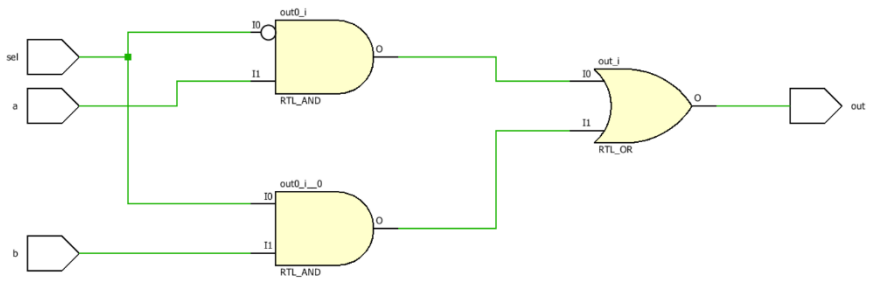
Hex7Seg

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

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

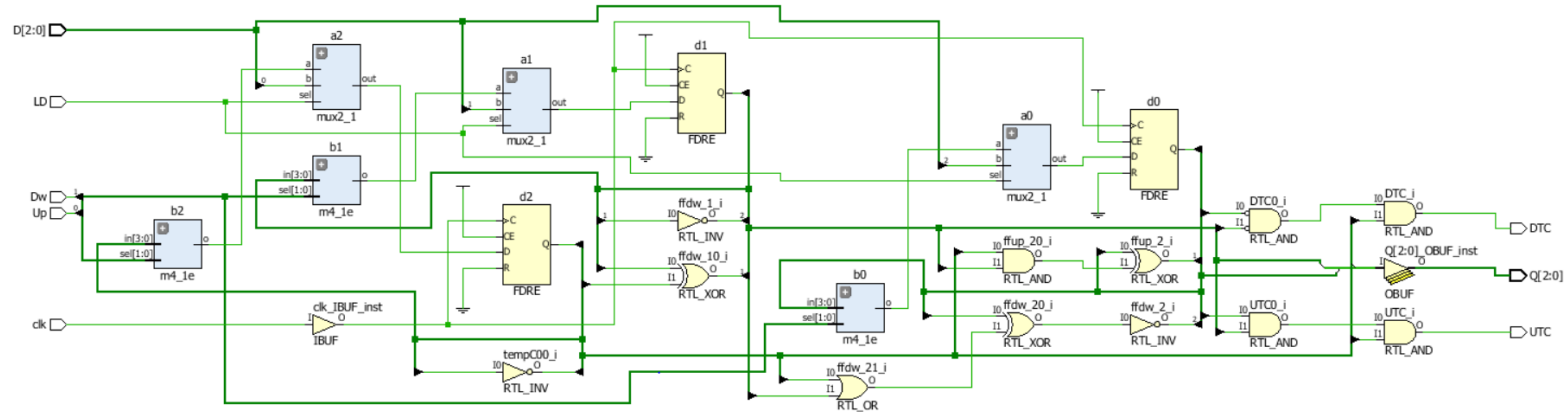


Mux 2_1

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



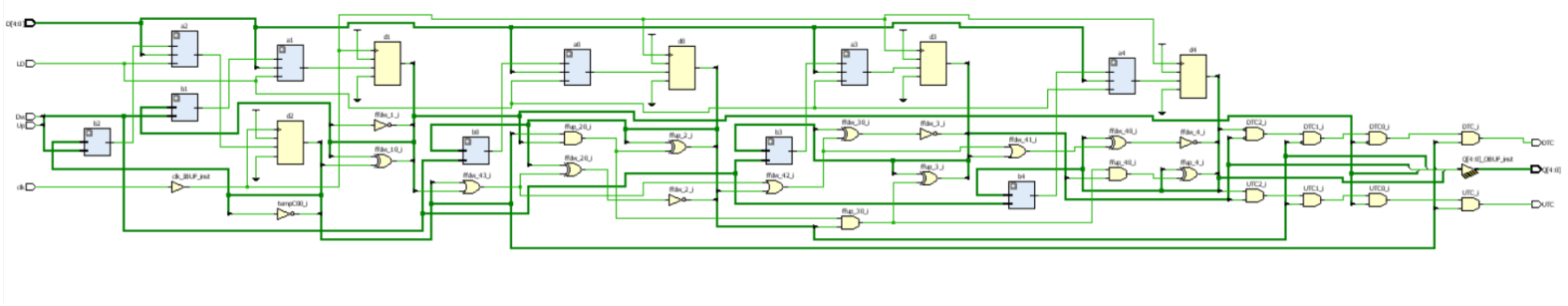
CountUD3L




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

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

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

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

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

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

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

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

    endmodule
```

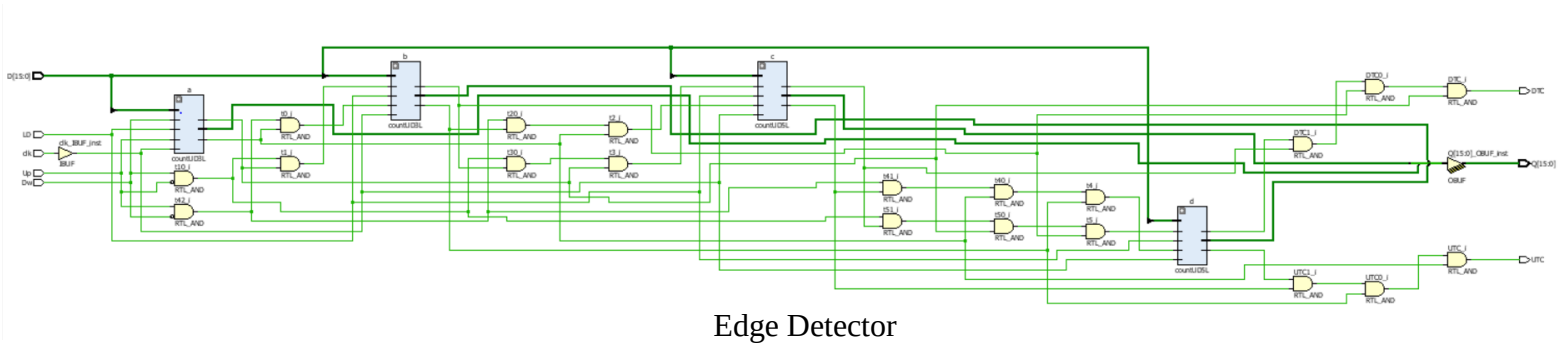
CountUD5L

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

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

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

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

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

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

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

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

    endmodule
```

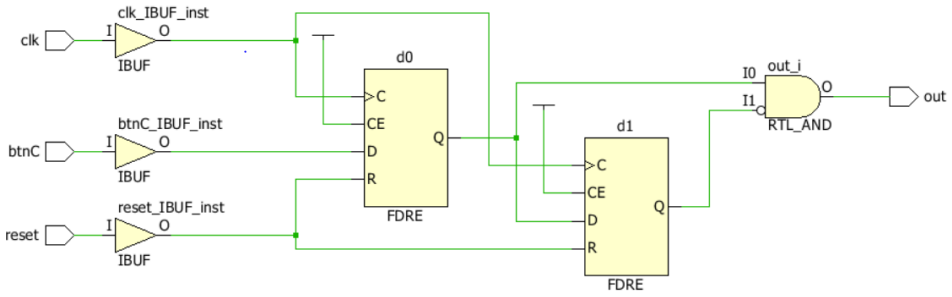


countUD16L

```
module countUD16L(
    input clk,
    input Up,
    input Dw,
    input LD,
    input [15:0] D,
    output [15:0] Q,
    output UTC,
    output DTC
     );

 wire [3:0] tempUTC, tempDTC;
 wire t0,t1,t2,t3,t4,t5;

    countUD3L a(.clk(clk), .Up(Up), .Dw(Dw), .LD(LD), .D(D[2:0]), .Q(Q[2:0]), .UTC(tempUTC[0]) , .DTC(tempDTC[0]));
    assign t0 = Up & ~Dw & tempUTC[0];
    assign t1 = Dw & ~Up & tempDTC[0];
    countUD3L b(.clk(clk), .Up(t0), .Dw(t1), .LD(LD), .D(D[5:3]), .Q(Q[5:3]), .UTC(tempUTC[1]) , .DTC(tempDTC[1]));
    assign t2 = Up & ~Dw & tempUTC[1]& tempUTC[0];
    assign t3 = Dw & ~Up & tempDTC[1]& tempDTC[0];
    countUD5L c(.clk(clk), .Up(t2), .Dw(t3), .LD(LD), .D(D[10:6]), .Q(Q[10:6]), .UTC(tempUTC[2]) , .DTC(tempDTC[2]));
    assign t4 = Up & ~Dw & tempUTC[2]& tempUTC[0]& tempUTC[1];
    assign t5 = Dw & ~Up & tempDTC[2]& tempDTC[0]& tempDTC[1];
   countUD5L  d(.clk(clk), .Up(t4), .Dw(t5), .LD(LD), .D(D[15:11]), .Q(Q[15:11]), .UTC(tempUTC[3]) , .DTC(tempDTC[3]));
    //swap is correct output, beacuse order was reverse in other counters
    assign UTC = tempUTC[3] &tempUTC[2] &tempUTC[1] &tempUTC[0];
    assign DTC = tempDTC[3] &tempDTC[2] &tempDTC[1] &tempDTC[0];


endmodule
```

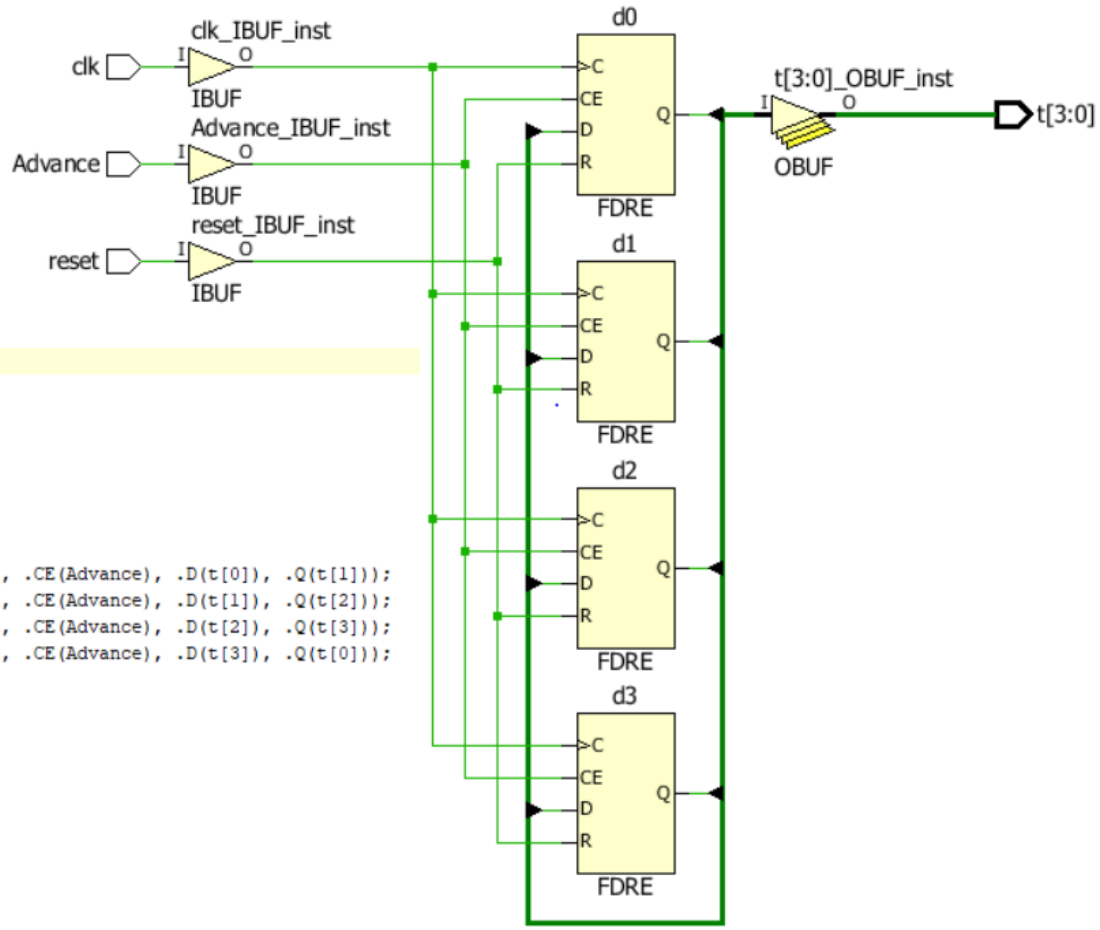


Edge Detector



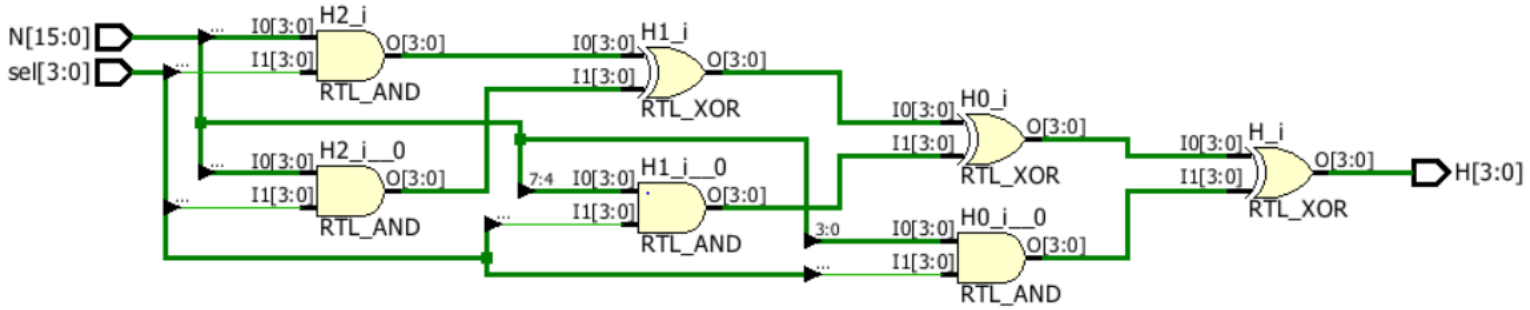

```
module edgeDect(
    input btnC,
    input clk,
    input reset,
    output out
    );
    wire t0,t1;
    FDRE #(.INIT(1'b0) ) d0(.C(clk), .R(reset), .CE(1'b1), .D(btnC), .Q(t0));
    FDRE #(.INIT(1'b0) ) d1(.C(clk), .R(reset), .CE(1'b1), .D(t0), .Q(t1));
    assign out = t0 & ~t1;
    endmodule
```

Ring Counter




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

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

endmodule
```

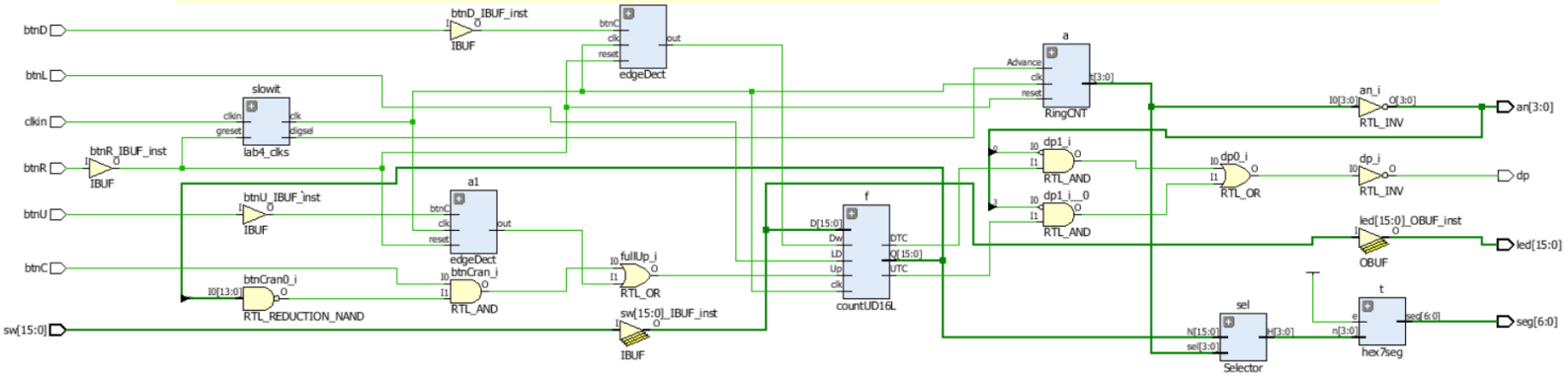
Selector



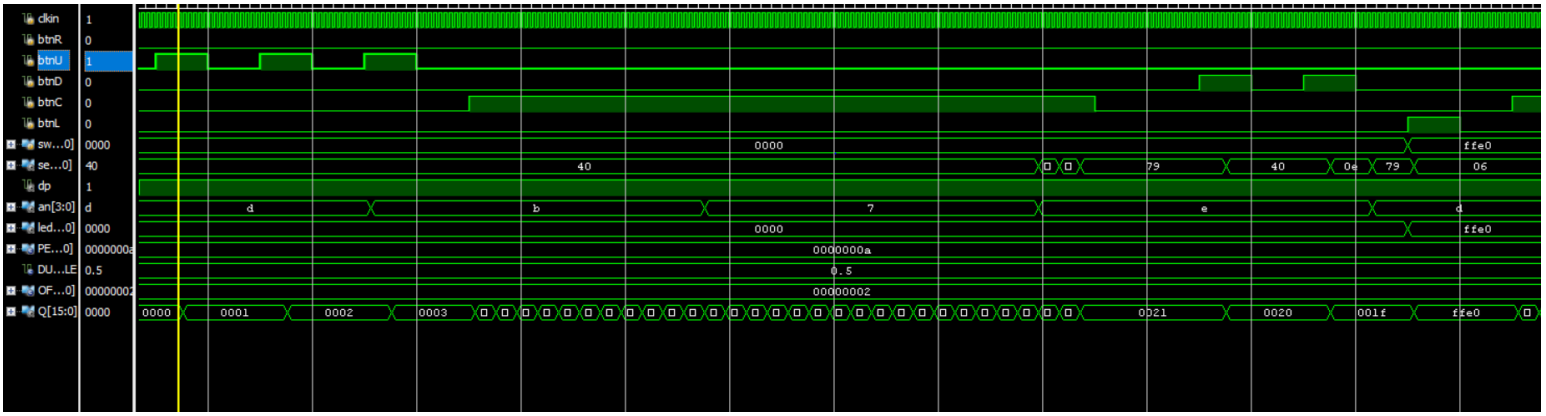

```
module Selector(
    input [3:0] sel,
    input [15:0] N,
    output [3:0] H
    );
      assign H = ((N[15:12]&{4{sel[3]}}) ^(N[11:8]&{4{sel[2]}})^(N[7:4]&{4{sel[1]}}) ^(N[3:0]&{4{sel[0]}}));
endmodule
```

Top Module

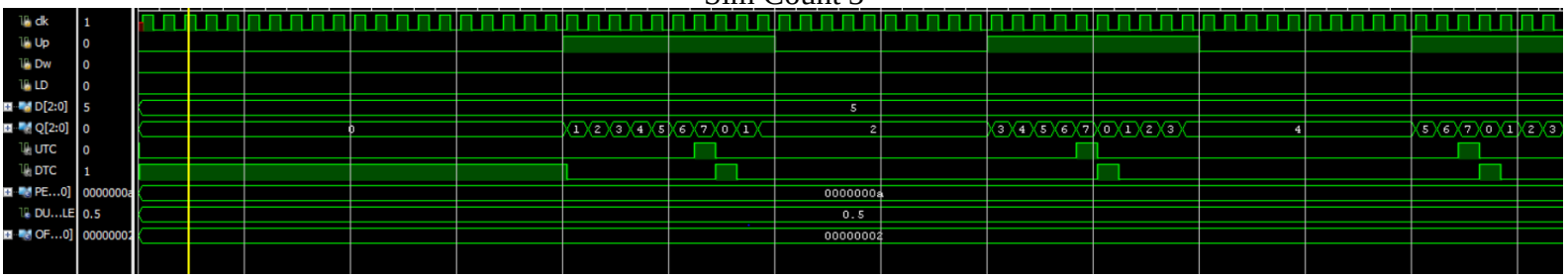
```
module topMod(
    input clkin,
    input btnR,
    input btnU,
    input btnD,
    input btnC,
    input btnL,
    input [15:0]sw,
    output [6:0] seg,
    output dp,
    output [3:0]an,
    output [15:0]led
    );
    wire clk;
    wire edgeOutU, edgeOutD, UTC, DTC, btnCran, fullUp, digsel;
    wire [15:0]Q;
    wire [3:0] hex7;
    wire [3:0] rndsel;
    wire [6:0] segtemp;

    lab4_clks slowit (.clkin(clkin), .greset(btnR), .clk(clk), .digsel(digsel));
     assign led = sw;

     assign btnCran = btnC & ~&(Q[15:2]);
    edgeDect a1(.btnC(btnU),.reset(btnR),.clk(clk),.out(edgeOutU));
    assign fullUp = btnCran | edgeOutU;


    edgeDect a2(.btnC(btnD),.reset(btnR),.clk(clk),.out(edgeOutD));

    countUD16L f(.clk(clk),.Up(fullUp),.Dw(edgeOutD),.LD(btnL),.D(sw),.Q(Q),.UTC(UTC),.DTC(DTC));

    RingCNT a(.clk(clk), .Advance(digsel),.reset(btnR),.t(rndsel));
    assign an = ~rndsel;
    Selector sel(.sel(rndsel),.N(Q),.H(hex7));
    hex7seg t(.n(hex7),.e(1'b1),.seg(seg));

     //assign seg = segtemp;
     assign dp = ~((~an[0]&DTC)|(~an[3]&UTC));
     endmodule
```



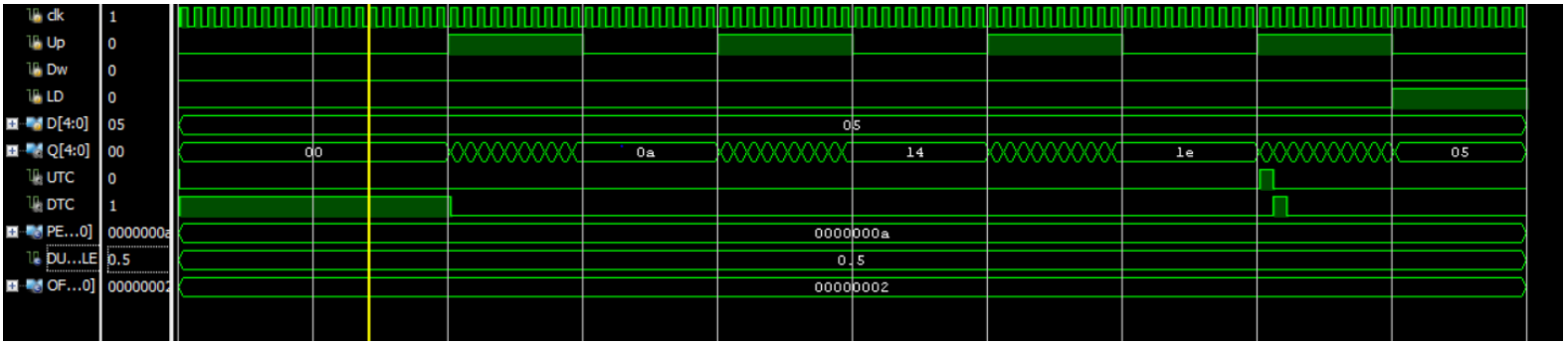
Top Sim



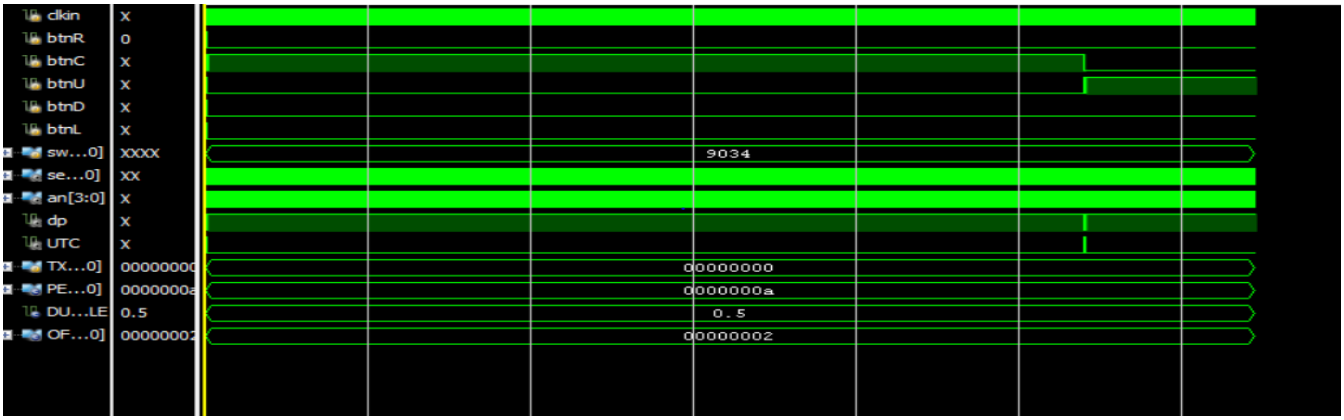
Sim Count 3



Sim Count 5



Test sim



Notebook Pages