Gabriel Dimas
May 6, 2022
Section C

Description

This lab was intended to help teach the importance of synchronization in a logic circuit. This lab showed how to use D Flip-Flops, positive edge triggers, ring counters, and a selector. Its function is to load 16-bit numbers, increment or decrement hexadecimal numbers, or continuously increment hex values until it reaches a certain value. It also detects when the 16-bit number is at its unsigned max or unsigned min value.

Design

- **Top_Module_Main**
  - inputs: 16 bit input, btnC, btnU, btnL, btnR, btnD, clkin
  - Outputs: 7-bit output, dp, 4-bit output, 16-bit output,
  - Implementation: Add a **lab4_clks** to slow down the clock from the Basys3 board. The output of this module is the clock used in all of the other modules. Add an **Edge_Detector** module and connect the clock and the btnU input to the inputs of this module. The output of this module is the input for the Up counter of **counterUD16L.** This ensures that only one instance of the module is called, not many times per clock cycle. Add an instance for btnC and ensure that the counter counts up until the button is depressed or the counter reaches a value of 'FFFC' through 'FFFF'. Add an instance of the **RingCounter.** The output of this module is a 4-bit bus used to refresh the value of one of four seven-segment displays. The outputs are inputs to an instance of the **Selector** module, used to select which of the segments will get updated. The 7-bit output of this module is the input for a **hex7seg** instance to display the values on the seven-segment display.
- **Edge_Detector**
  - inputs: btnU, clock
  - outputs: I/O signal
  - Implementation: when the button is pressed and the clock value is high, the output will be high for about two clock cycles. This prevents the output to be high for the entire time the button is pressed to prevent the counter from repeatedly counting up. This is implemented with two flip-flops and an AND gate. This ensures that the current and previous inputs are 1 and 0, respectively. This ensures that there

was no previous output, which would result in a repeated output. Here is a snippet of the waveform and how the Edge_Detecor works[1]:

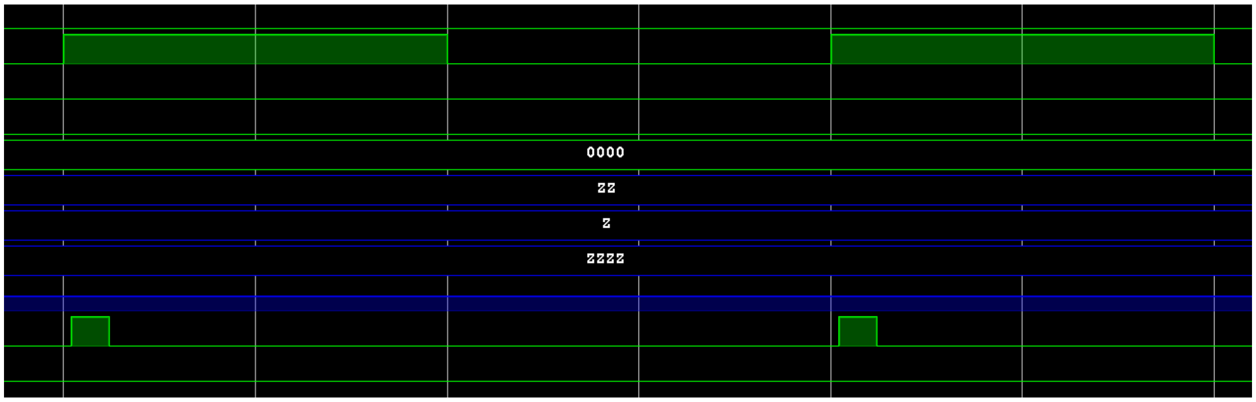
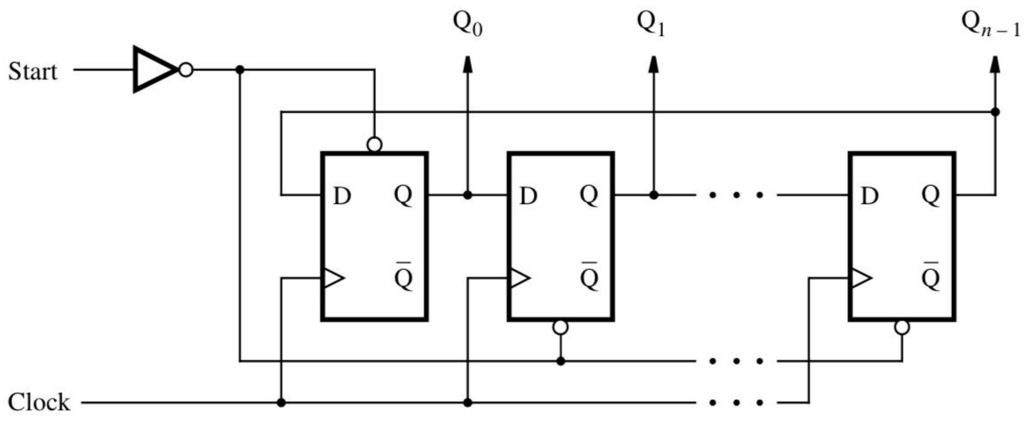


Figure 1

- **Ring_Counter**
    - inputs: digsel, clock
    - outputs: 4-bit bus
    - Implementation: This module works by using four flip-flops. There is only one HIGH bit at any time, and that HIGH bit gets passed to the next flip flop at the positive edge of each clock cycle. This output is then bused together and outputted. This module acts as the project's "HIGH" segment counter. Here is the logic of how it should be implemented[2]:



(a) An n-bit ring counter

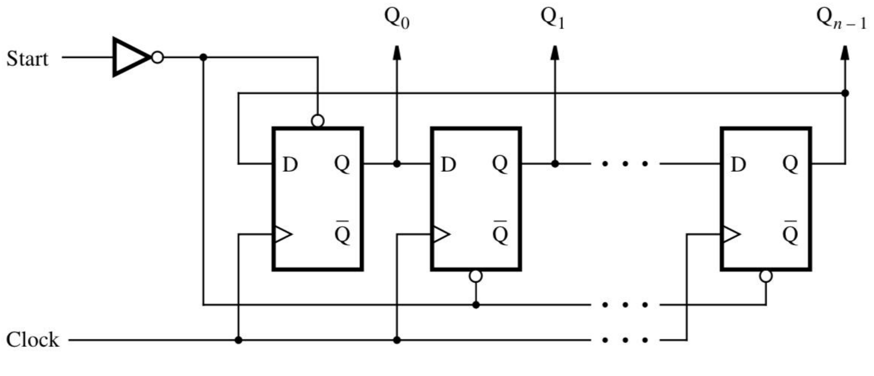
Figure 2

[1] https://www.chipverify.com/verilog/verilog-positive-edge-detector
[2] *Fundamentals of Digital Logic with Verilog Design, Third Edition* by Stephen Brown and Zvonko Vranesic

- **Selector**
  - inputs: 4-bit bus, 16-bit bus
  - outputs: 4-bit bus
  - Implementation: This module takes the input from the **Ring_Counter** to refresh the value on the segment that has been selected. For example, if the ring counter's output is "0100", this means that the an[2] segment will be activated to refresh. This value is passed to the Selector module to refresh the value of the an[2] seven-segment display. When the selector chooses which value to refresh, it will output 4 bits of the 16-bit bus and send that value to the **hex7seg** to display.
- **hex7seg**[3]
  - inputs: 4-bit bus
  - outputs: 7-bit bus
  - Implementation: Imported from Lab 3, this module uses eight 8 to 1 multiplexers and some gates to output the correct segments for the seven-segment display
- **countUD4L**
  - inputs: 4-bit bus, Up, Down, Load, Clock
  - outputs: 4-bit bus
  - Implementation: This module is made of D Flip-Flops to hold the value of the current digit. If the Up button is high, then the counter will count to the next immediate digit using the implementation of a full adder (see **Appendix**). When the Dw button is high, the counter will decrement to the previous immediate value. When the LD button is high, the counter will load the value that is passed in the 4-bit Din bus. This is done by directly inputting the values of that bus into the flip flop input and storing that value. When the stored value reaches "1111" the counter will output UTC high, else it will output UTC low. Similarly, if the value reaches "0000" the counter will output DTC high, else it will output DTC low.
- **counterUD16L**
  - inputs: 16-bit bus, Up, Down, Load, Clock,
  - outputs: UTC, DTC, 16-bit bus
  - Implementation: this counter is created by joining four 4-bit counters together. This module connects the counters in a way so that they count up when it is their turn. When their respective "UP" signal is high, then the counter will do its job to

[3] *Dimas, Gabriel, Lab 3: Multiplexers, Full Adders, and Seven Segment Displays*

move to the next count. When all the counters are UTC high, the 16-bit counter will also be UTC high, else it is low. When all the counters are DTC high, the 16-bit counter will also be DTC high, else it is low. The Q output bus joins all the 4-bit counter buses to create one 16-bit bus. Here is how the module is implemented[4] (without the logic for a decrementer or a loader):



(a) An n-bit ring counter

Figure 3

Testing & Simulation

When testing our lab, ensure that every module works as expected. To begin, know how each of the modules intends to work. For example, know that the **Edge_Detector** module is intended to go HIGH for one clock cycle if the btnU input is pressed. One major module to test was the countUD4L counter. When simulating, ensure that the btnU increments the counter by one, btnD decrements the counter by one, and btnL loads the counter with the correct hexadecimal representation from the 4-bit binary representation. After this works, connect four **countUD4L** to make one **counterUD16L**. To do this, ensure that each of the 4-bit counters increments successively when the previous one has reached its full value. In this lab, when one of the **countUD4L** counters reaches UTC HIGH, this is a signal for the next **countUD4L** counter to begin incrementing as well.

Do the same thing with a decrementer. With decrementing, when DTC is HIGH for one **countUD4L**, this is a signal for the next **countUD4L** to decrement as well. On the other hand, when btnL is HIGH, load 4 of the 16 bits from the output to each of the **countUD4L** counters, and assign them to the outputs of the flip flops. When all of the **countUD4L** UTC values are HIGH, the **counterUD16L** UTC value is also HIGH, same goes with DTC without loss of generality. This was difficult during the first couple of simulations of this module because some

[4] *Fundamentals of Digital Logic with Verilog Design, Third Edition* by Stephen Brown and Zvonko Vranesic

of the connections were not in the correct positions and this contributed to many hours of debugging. Furthermore, this module became less of a hassle when each **countUD4L** counter was connected in series and individually incremented when UTC values were high. During simulation, this was finally the case after many trial and error attempts. See **Appendix** for the waveform for the testing and simulation.

Results

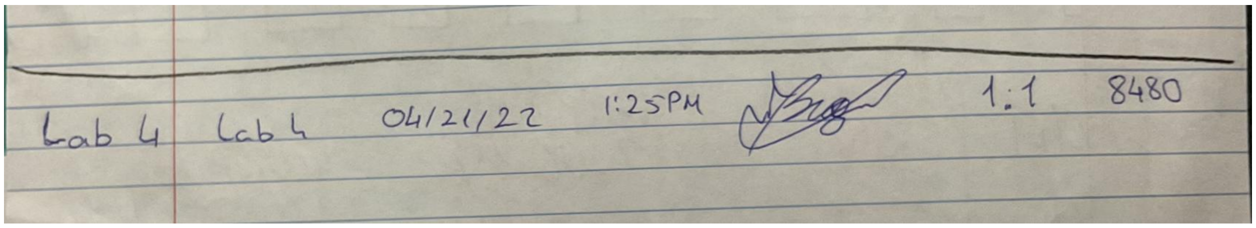
After testing and simulation, the lab became a success. There was lots to learn, mainly how to connect counters together to ensure that they can synchronously work together. In addition to this, the testbenches provided more than helpful feedback before generating bitstreams. When the btnR is held down, the left-most display holds a zero and the others turn off. This acts as a global reset, resetting all the values held previously on the display to zero. This may be done so because this reset resets the flip flops in the schematic to their default hold values. It can also be a state of unknown, where the flip flops cannot know what state to be in, so they are defaulted to zero. When all of the clock inputs are connected to the fast clock, the left-most seven-segment display is high, but the others seem like they are giving an active low signal. Some theories are behind this. It can most likely be that digsel is always a HIGH value for one clock cycle of the original clock[5], so this can explain that the anodes of all the displays are LOW, which is what can be seen.

Conclusion

Overall, this lab resulted in some very interesting findings and real-world applications. For one, it taught how to implement counters using flip flops and how their holding value would remain constant for the entire time until the next input. This lab also taught the importance of synchronization with positive clock edges and the speed of the clock on the Basys3 board. In real life, this type of synchronization is valuable because states should either all change at the same time, or they should change individually but instantaneously. Timing is also important. The clock is the heat beat of the entire design: any issues with it can potentially veer the entire project into a state of asynchronization. If it could be done again, it may have been easier to use smaller 2-bit counters instead of 4-bit counters. There would be no loss of generality if this route is taken, and it would ensure that the module is done fairly quickly. Although I can see the fault with this would be the timing and the inefficiency.

[5] A partial of this response originated from the original working on https://classes.soe.ucsc.edu/cse100/Spring22/lab/lab4/lab4.html

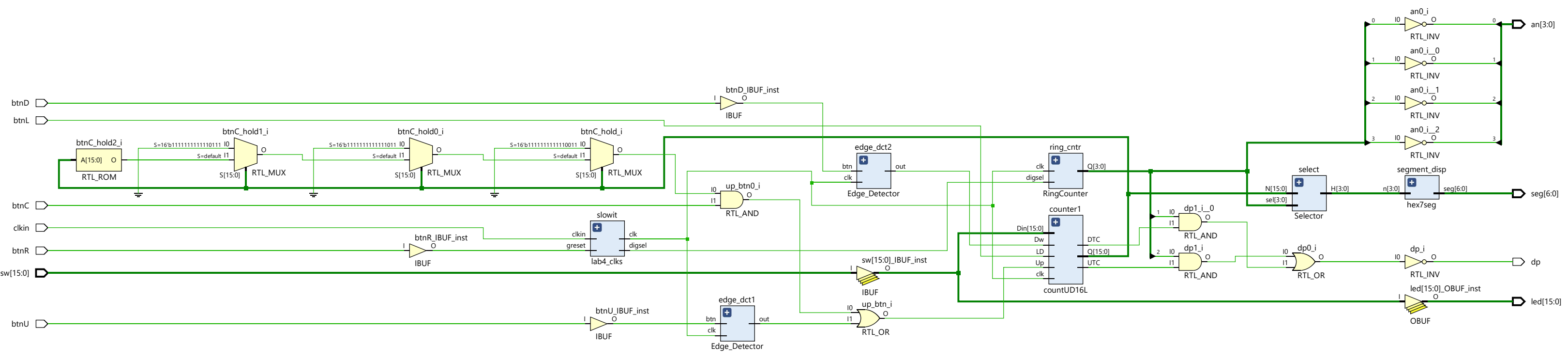
Appendix

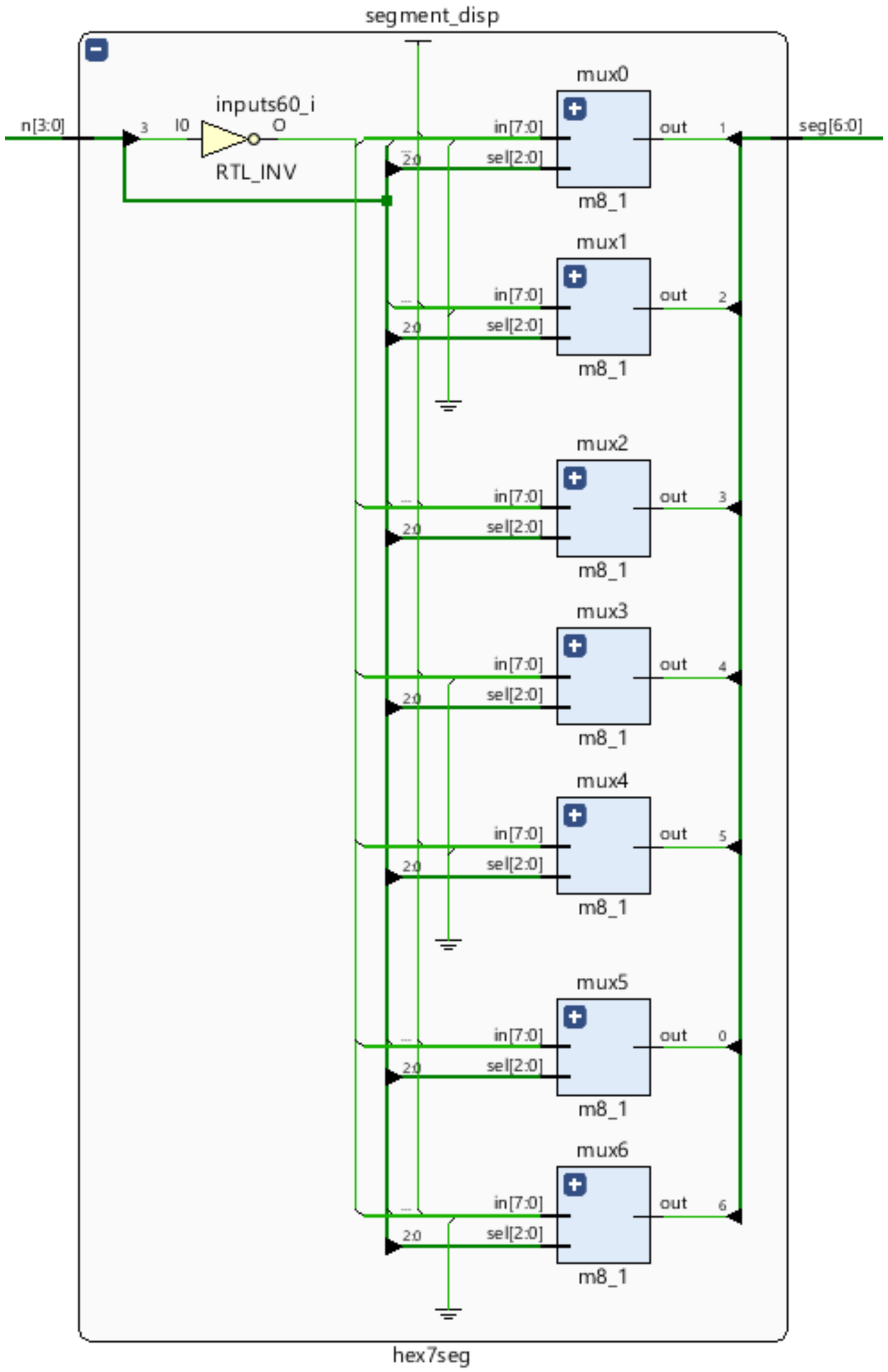


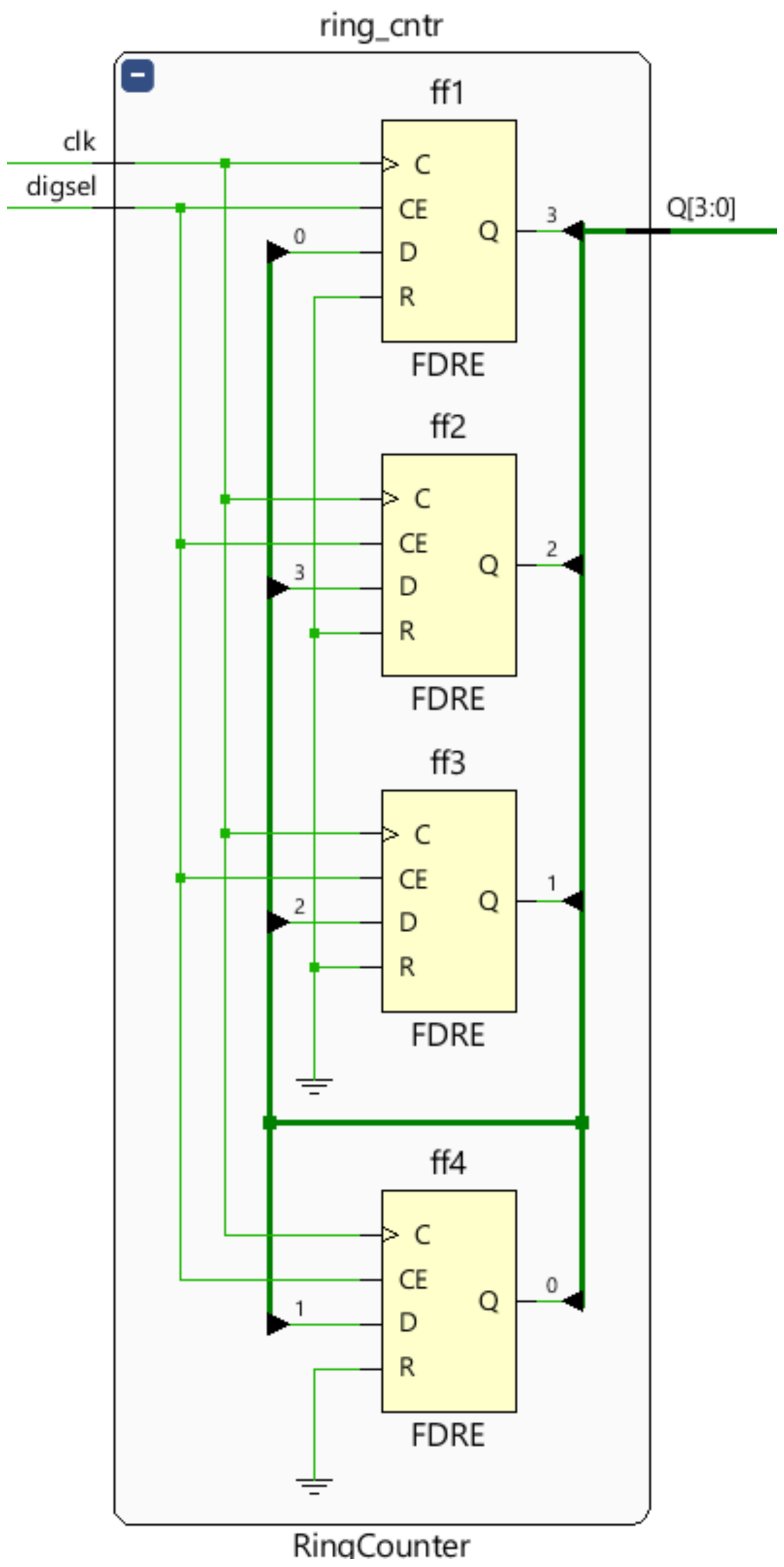
Lab 4 Lab 4 04/21/22 1:25PM [illegible] 1.1 8480

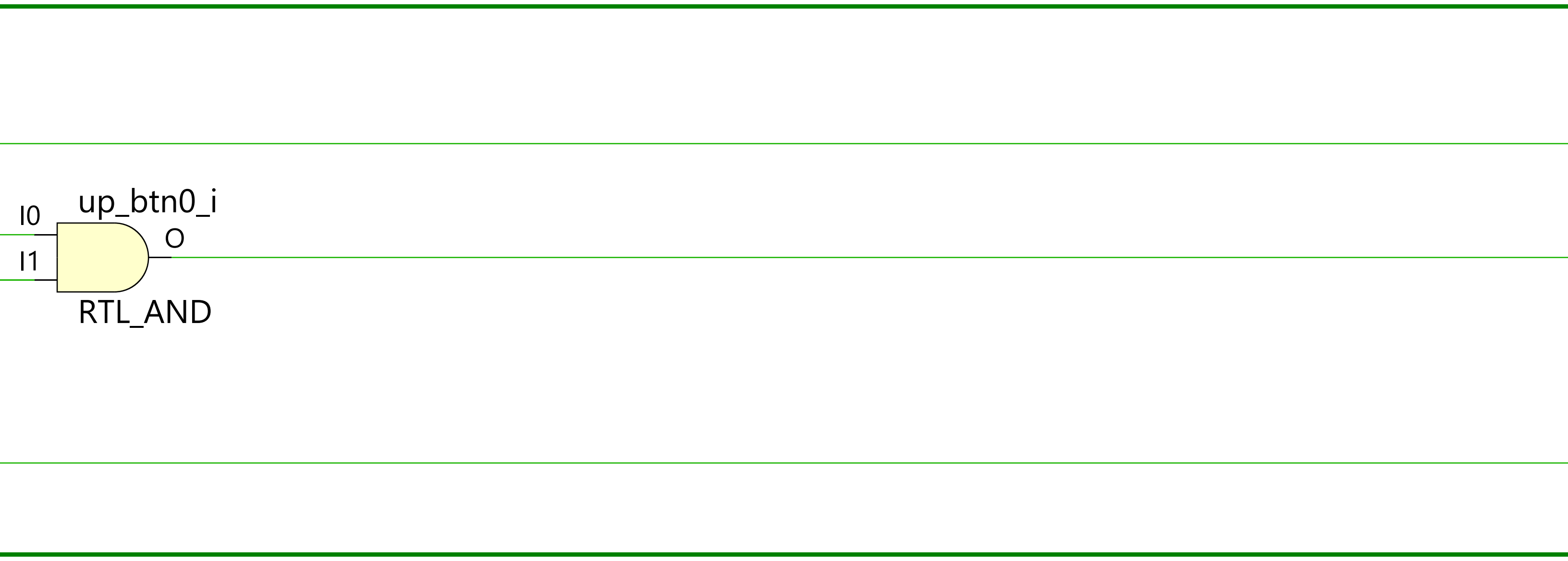
Lab 4 04/21/22 01:25PM 1.1 points Code: 8480

Below are the Lab 4 Schematics, Verilog Code, and Waveform of simulation results.

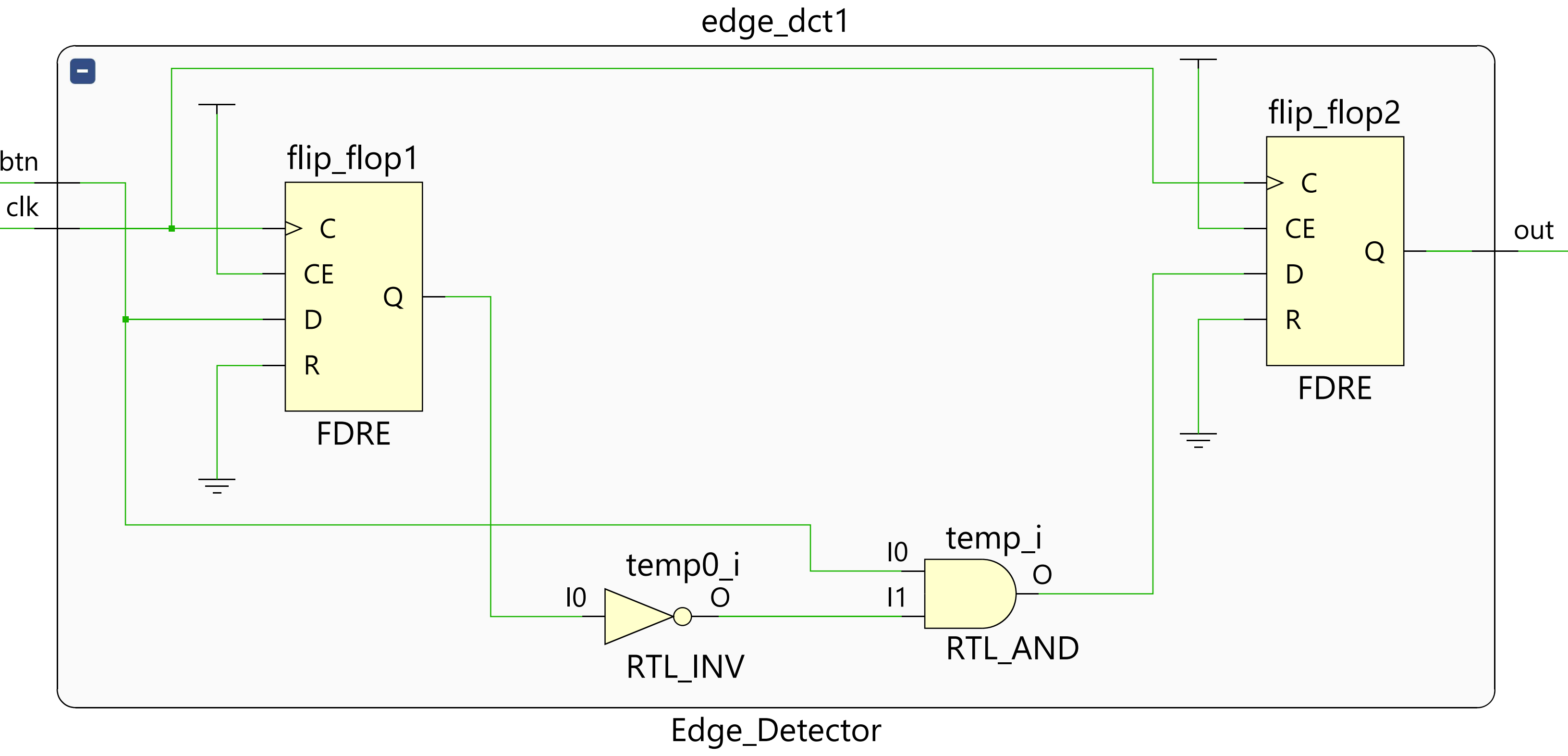

btnD
btnL
btnC
clkin
btnR
sw[15:0]
btnU
btnC_hold2_i
A[15:0]
O
RTL_ROM
btnC_hold1_i
S=16'b1111111111110111
S=default
S[15:0]
RTL_MUX
btnC_hold0_i
S=16'b1111111111111011
btnC_hold_i
S=16'b1111111111110011
btnD_IBUF_inst
IBUF
edge_dct2
btn
clk
out
Edge_Detector
up_btn0_i
RTL_AND
slowit
clkin
greset
clk
digsel
lab4_clks
btnR_IBUF_inst
sw[15:0]_IBUF_inst
btnU_IBUF_inst
edge_dct1
up_btn_i
RTL_OR
ring_cntr
Q[3:0]
RingCounter
counter1
Din[15:0]
Dw
LD
Up
DTC
Q[15:0]
UTC
countUD16L
dp1_i__0
dp1_i
dp0_i
dp_i
RTL_INV
select
N[15:0]
sel[3:0]
H[3:0]
Selector
segment_disp
n[3:0]
seg[6:0]
hex7seg
an0_i
an0_i__0
an0_i__1
an0_i__2
an[3:0]
seg[6:0]
dp
led[15:0]_OBUF_inst
OBUF
led[15:0]


segment_disp
n[3:0]
3
I0
inputs60_i
O
RTL_INV
mux0
mux1
mux2
mux3
mux4
mux5
mux6
in[7:0]
sel[2:0]
2:0
out
m8_1
seg[6:0]
hex7seg

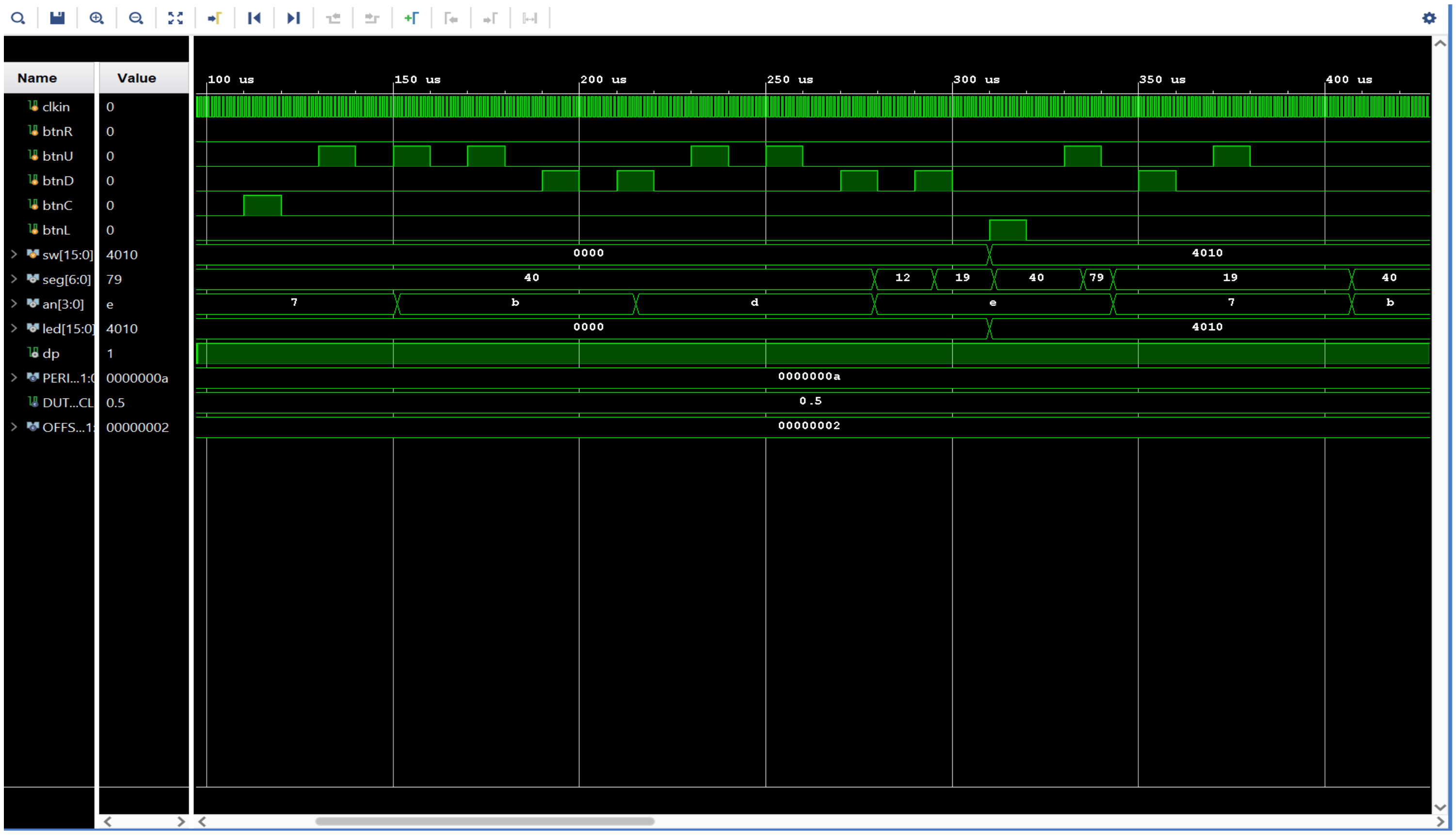

ring_cntr
clk
digsel
ff1
C
CE
D
R
Q
FDRE
0
3
Q[3:0]
ff2
3
2
FDRE
ff3
2
1
FDRE
ff4
1
0
FDRE
RingCounter



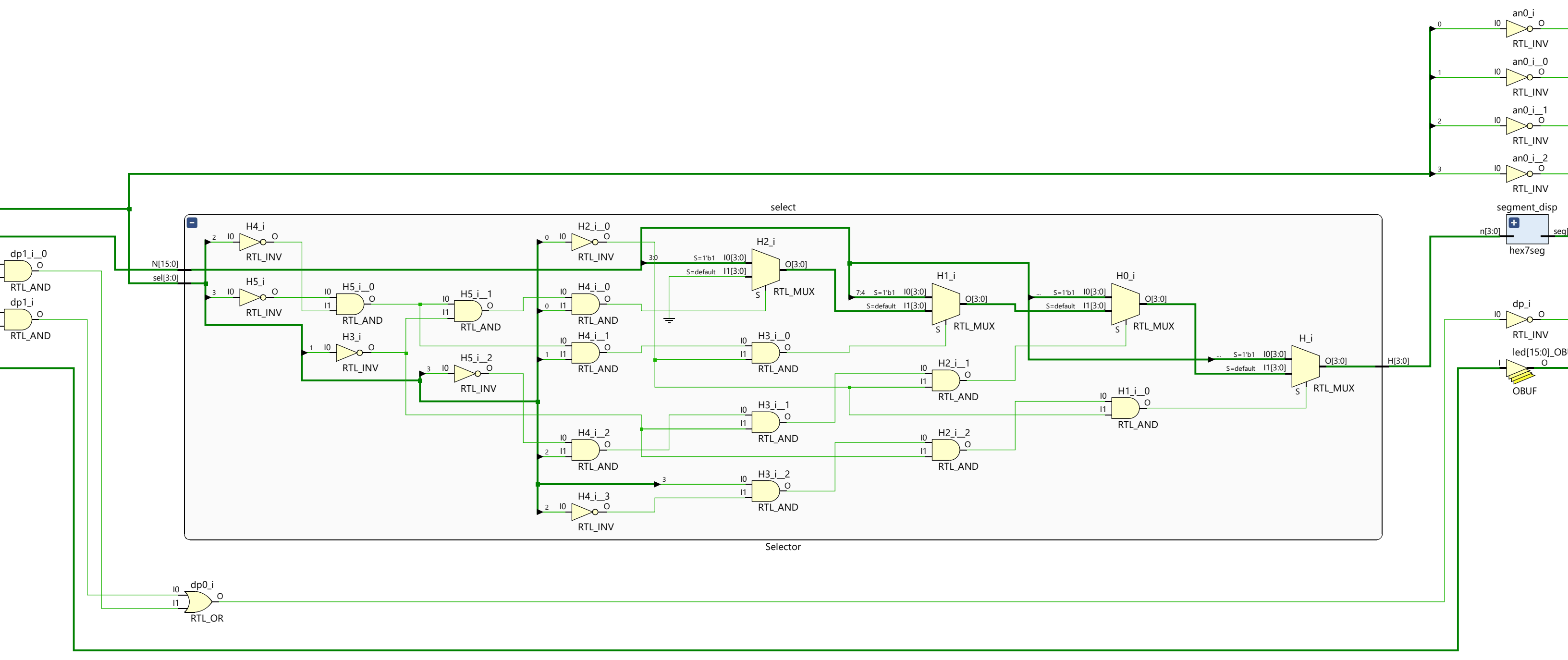
I0
up_btn0_i
O
I1
RTL_AND



edge_dct1
btn
clk
flip_flop1
C
CE
D
R
Q
FDRE
flip_flop2
C
CE
D
R
Q
FDRE
out
temp0_i
I0
O
RTL_INV
temp_i
I0
I1
O
RTL_AND
Edge_Detector



Waveform Diagram of Lab 4



an0_i
an0_i__0
an0_i__1
an0_i__2
RTL_INV
select
Selector
segment_disp
hex7seg
n[3:0]
N[15:0]
sel[3:0]
H4_i
H5_i
H3_i
H5_i__0
H5_i__1
H5_i__2
H2_i__0
H4_i__0
H4_i__1
H4_i__2
H4_i__3
H3_i__0
H3_i__1
H3_i__2
H2_i
H1_i
H0_i
H_i
H2_i__1
H2_i__2
H1_i__0
RTL_AND
RTL_MUX
S=1'b1
S=default
I0[3:0]
I1[3:0]
O[3:0]
H[3:0]
dp1_i__0
dp1_i
dp0_i
RTL_OR
dp_i
led[15:0]_OBUF
OBUF

```
module Top_Module_main(
    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, [15:0]led
    );

    //some module with 3 input, 1 output. IN: 16bit bus, btnC, clk
    wire up_btn, down_btn, btnU_press;
    wire digsel, clk;
    wire utc, dtc, btnC_hold;
    wire [15:0]bit16out;
    assign led = sw;    //turns on the LED to the corresponding switch

    lab4_clks slowit (.clkin(clkin), .greset(btnR), .clk(clk), .digsel(digsel));
//DONE

    Edge_Detector edge_dct1(.clk(clk), .btn(btnU), .out(btnU_press));  //1 input for
OR gate
    assign btnC_hold = (bit16out == 16'b1111111111110011) ? 0  :      //fffc
                       (bit16out == 16'b1111111111111011) ? 0  :      //fffd
                       (bit16out == 16'b1111111111110111) ? 0  :      //fffe
                       (bit16out == 16'b1111111111111111) ? 0  : 1;   //ffff
    assign up_btn = (btnC_hold & btnC) | btnU_press;

    Edge_Detector edge_dct2(.clk(clk), .btn(btnD), .out(down_btn)); //input for
16bit cntr
    countUD16L counter1 (.clk(clk), .Up(up_btn), .Dw(down_btn),
        .LD(btnL), .Din(sw), .UTC(utc), .DTC(dtc), .Q(bit16out));   //may change UTC
and DTC outputs



    wire [3:0]Qring;
    RingCounter ring_cntr (.digsel(digsel), .clk(clk), .Q(Qring));

    assign an[0] = !Qring[0];
    assign an[1] = !Qring[1];
    assign an[2] = !Qring[2];
    assign an[3] = !Qring[3];
    assign dp = ~((Qring[1] & utc) | (Qring[2] & dtc)) ;
   // assign dp = ~(Qring[2] & utc);
    //Selector module with IN: 3bit sel from RingCounter, OUT: 4bit H
    wire [3:0]sel;
    Selector select(.sel(Qring), .N(bit16out), .H(sel));
   // assign dp = (~Qring[2] & ~utc) | (~Qring[1] & ~dtc);//~(utc | dtc) & (Qring[2]
| Qring[3]);
```

```
    //hex7seg module with IN 4bit bus from Selector, OUT: 7bit to svnseg_disp
     hex7seg segment_disp (.n(sel), .seg(seg));






endmodule
```

```
//status: SEEMING TO WORK
module Edge_Detector(
    input clk, input btn,
    output out
);
    wire feed, temp;
   FDRE #(.INIT(1'b0)) flip_flop1 (.C(clk), .R(1'b0), .CE(1'b1), .D(btn), .Q(feed));
   assign temp = (btn & !feed);   //src1
   FDRE #(.INIT(1'b0)) flip_flop2 (.C(clk), .R(1'b0), .CE(1'b1), .D(temp), .Q(out));
endmodule
```

```
//status: COMPLETE
module countUD16L(
    input clk,
    input Up,
    input Dw,
    input LD,   //iff pressed, load the switch value
    input [15:0] Din,
    output UTC, output DTC, output [15:0] Q
    );
    wire [3:0] utc,  dtc;
    wire m1, m2, m3, d1, d2, d3;
    countUD4L count4bit1 (.Up(Up), .Dw(Dw), .LD(LD), .Q(Din[3:0]), .clk(clk),   //cnt
        .UTC(utc[0]), .DTC(dtc[0]), .Qout(Q[3:0]));
    assign m1 = Up & utc[0];
    assign d1 = Dw & dtc[0];

    countUD4L count4bit2 (.Up(m1), .Dw(d1), .LD(LD), .Q(Din[7:4]), .clk(clk),   //cnt
        .UTC(utc[1]), .DTC(dtc[1]), .Qout(Q[7:4]));
    assign m2 = Up & utc[0] & utc[1];
    assign d2 = Dw & dtc[0] & dtc[1];

    countUD4L count4bit3 (.Up(m2), .Dw(d2), .LD(LD), .Q(Din[11:8]), .clk(clk),  //cnt
        .UTC(utc[2]), .DTC(dtc[2]), .Qout(Q[11:8]));
    assign m3 = Up & utc[0] & utc[1] & utc[2];
    assign d3 = Dw & dtc[0] & dtc[1] & dtc[2];

    countUD4L count4bit4 (.Up(m3), .Dw(d3), .LD(LD), .Q(Din[15:12]), .clk(clk), //cnt
        .UTC(utc[3]), .DTC(dtc[3]), .Qout(Q[15:12]));

    assign UTC = utc[0]&utc[1]&utc[2]&utc[3] ? 1 : 0;
    assign DTC = dtc[0]&dtc[1]&dtc[2]&dtc[3] ? 1 : 0;

endmodule
```

```
//status: OPERATIONAL
module RingCounter(
    input digsel,
    input clk,
    output [3:0]Q
    );
    //wire start;
    FDRE #(.INIT(1'b1)) ff1 (.C(clk), .R(1'b0), .CE(digsel), .D(Q[0]), .Q(Q[3]));
    FDRE #(.INIT(1'b0)) ff2 (.C(clk), .R(1'b0), .CE(digsel), .D(Q[3]), .Q(Q[2]));
    FDRE #(.INIT(1'b0)) ff3 (.C(clk), .R(1'b0), .CE(digsel), .D(Q[2]), .Q(Q[1]));
    FDRE #(.INIT(1'b0)) ff4 (.C(clk), .R(1'b0), .CE(digsel), .D(Q[1]), .Q(Q[0]));
endmodule
```

```
//status: COMPLETE
module Selector(
  input [3:0] sel,
  input [15:0] N,
  output [3:0] H
  );

  assign H = ( sel[3] & !sel[2] & !sel[1] & !sel[0]) ? N[15:12]:
             (!sel[3] &  sel[2] & !sel[1] & !sel[0]) ? N[11:8] :
             (!sel[3] & !sel[2] &  sel[1] & !sel[0]) ? N[7:4]  :
             (!sel[3] & !sel[2] & !sel[1] & sel[0])  ? N[3:0]  : 0;
endmodule
```