Lab 4
Simon Carballo
1618309
11/13/2020
Section D

Write Up

- **Description:** In this lab we were asked to design and implement D flip-flop counters for our sequential circuit. The counters are used to create a 16-bit counter that can increment, decrement and load values. These values are operated with buttons, and switches that. The outputs for the design will be LEDs to detect UTC, DTC, and load values as well as the four segment displays.

- **Design:** For this design we have the Top Level that calls to 7 subsections which includes: Clock, Edge_Detector(Up, Dw), 16-bit Counter, Ring Counter, Selector, and the Segment_Display Converter. As we go over each subsection we will look more into depth of how each section is designed.
    - Clock:
      This code is provided by the lab manual so I do not know how to go into depth on every detail. Basically this module is the global clock and reset for all of the counters in the Top_Level. It takes in the inputs of clkin from the constraint file, and manipulates it to become the net clk for the sequential design. We also use this to get Dig_sel for the ring counter (Explained in Ring Counter Module).

    - Edge_Detector:
      The Edge Detector is used to input Up and Dw into the 16-bit counter. The Edge Detector will generate a high value for one clock cycle if the past two inputs consist of a 0 followed by a 1. Basically, if the BtnU or BtnD is pressed, it will output one high value which feeds into the counter.

    - 16-bit Counter:
      Assuming that we know how adders work, we will be using 4 counters of 4 bits to create the model for a 16-bit counter.
        - 4-bit counter:
          The four bit counter is created using 4 D flipflops which are connected together using a boolean expression found by creating a truth table:

	Present State				Next State | Up = 1 Dw = 0				Next State | Up = 0 Dw = 1			
Val	Q3	Q2	Q1	Q0	D3	D2	D1	D0	D3	D2	D1	D0
0	0	0	0	0	0	0	0	1	1	1	1	1

1	0	0	0	1	0	0	1	0	0	0	0	0
2	0	0	1	0	0	0	1	1	0	0	0	1
3	0	0	1	1	0	1	0	0	0	0	1	0
4	0	1	0	0	0	1	0	1	0	0	1	1
5	0	1	0	1	0	1	1	0	0	1	0	0
6	0	1	1	0	0	1	1	1	0	1	0	1
7	0	1	1	1	1	0	0	0	0	1	1	0
8	1	0	0	0	1	0	0	1	0	1	1	1
9	1	0	0	1	1	0	1	0	1	0	0	0
10	1	0	1	0	1	0	1	1	1	0	0	1
11	1	0	1	1	1	1	0	0	1	0	1	0
12	1	1	0	0	1	1	0	1	1	0	1	1
13	1	1	0	1	1	1	1	0	1	1	0	0
14	1	1	1	0	1	1	1	1	1	1	0	1
15	1	1	1	1	0	0	0	0	1	1	1	0

- With this truth table I used Kmaps to find the boolean expression for the next state(D).
  - Example: Down State

D3			
1	0	1	0
0	0	1	1
0	0	1	1
0	0	1	1

- D3 = Q1Q2+Q1Q4+Q1Q3+~Q1~Q2~Q3~Q4

D2			
1	0	0	1

0	1	1	0
0	1	1	0
0	1	1	0

- D2 = ~Q2~Q4~Q3+Q4Q2+Q3Q2

D1			
1	1	1	1
0	0	0	0
1	1	1	1
0	0	0	0

- D1 = ~Q4~Q3+Q4Q3

D0			
1	1	1	1
0	0	0	0
0	0	0	0
1	1	1	1

- D0 = ~Q4

- By evaluating these maps we can now grasp a pattern to minimize these boolean expressions using the following formula:
  - Qn^(ΣQn-1)
- Therefore we can make these boolean expressions:
  - d0 = ((q0^(Up|Dw))&~LD)|LD&Din[0];
  - d1 = ((q1^((Up&q0)|(Dw&~q0)))&~LD)|LD&Din[1];
  - d2 = ((q2^((Up&q0&q1)|(Dw&~q0&~q1)))&~LD)|LD&Din[2];
  - d3 = ((q3^((Up&q0&q1&q2)|(Dw&~q0&~q1&~q2)))&~LD)|LD&Din[3];
- On top of the formula we covered, you also see the integration of LD, Din, Up and Dw. You could integrate this into your truth table, but I found it easier to add the inputs after the fact to better understand the expression.
- The LD will decide whether or not the counters will be loaded with values from the SW input (Din).

        - Finally to make this an official adder we need the sum and the cout which is expressed as Q[3:0] for sum and UTC, DTC for cout. The UTC and DTC is made by anding the sum or nanding the sums.

    - In order to make this an official 16-bit adder we call 4 4-bit counting modules and connect them together using wires. Using the UTC and DTC as inputs for the next 4-bit module

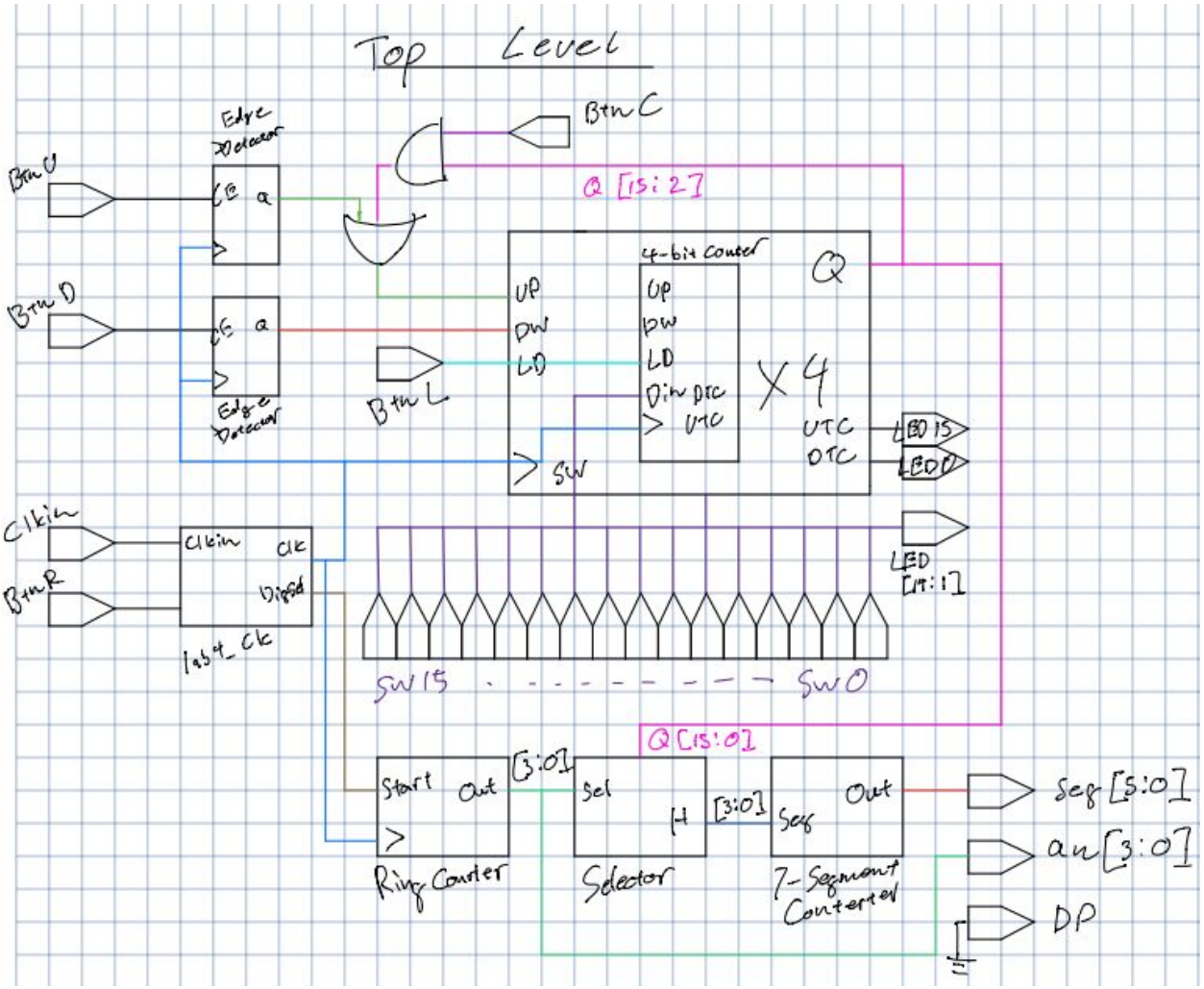
- Ring Counter:
  The ring counter is used as encode to create a one-hot/single active in 4-bit bus. This is used to set values for the selector. The module is created using a start(CE) and the clock(clk) which iterates through 4 flip flops as one state. In order to do this we connect the 4 flip flops in a complete loop and initialize one of the flip flops. This way we would get the outputs 0001, 0010, 0100, 1000, repeat.

- Selector:
  The selector is used to select a segment of it's inputs and output it with it's respective weight. This module is made with just boolean expressions with the inputs being the ring counter and the 16-bit counter and the output being the input to a seven_segment converter. As we review above the ring counter will feed a set of 4-bits with one active bit therefore giving it a state value for a certain part of the 16-bit you want to pass through. In this case, we want to pass through every 4-bits of the counter starting from 0. In pseudo code, it looks like this:
    - H is N[15:12] when sel=(1000)
    - H is N[11:8] when sel=(0100)
    - H is N[7:4] when sel=(0010)
    - H is N[3:0] when sel=(0001)

  When we put it in verilog it looks like this:
    - H = (N[15:12] & {4{(sel[3] & ~sel[2] & ~sel[1] & ~sel[0])}})|
    - (N[11:8] & {4{(~sel[3] & sel[2] & ~sel[1] & ~sel[0])}})|
    - (N[7:4] & {4{(~sel[3] & ~sel[2] & sel[1] & ~sel[0])}})|
    - (N[3:0] & {4{(~sel[3] & ~sel[2] & ~sel[1] & sel[0])}});

- Seven Segment Converter:
  Finally we have the Seven Segment Converter. This module has been reviewed in previous lab reports and it works exactly the same. It takes in 4 inputs and uses boolean expressions to output the proper values to its respective segments.

- Top Level:



- In this Diagram you can see how each module is connected with each other to make the sequential circuit work. It may seem like there is only one segment display, and that is because the ring counter is also connected to the 4 segments(an) which displays the proper values according to the state of the counter.

- **Testing & Simulation:**
  This was a very stressful lab as it is the first use of clocks. This meant that there were timing loop errors to account for. Timing loops are accidentally created by creating a pathway that can loop back into itself. This error is common, and could cause many hours of stressful digging to find the problem. My design had a system loop which prevented me from running simulations until the weekend when I figured out the cause. Simulations were run with the provided test fill, and making the necessary adjustments to test other functions. One of the TAs was a great help in explaining how to debug specific wires with the simulation which really helped figure out the small undetectable problems in my design.

- **Results:**
    1. By holding down btnR the display an[3:1] goes dark. This is basically caused by resetting all of the counters. Meaning that the clk will stay at a constant low leaving the ring counter to only display a single display. I only think this is the reason because the output of the ring counter is the power to the displays.
    2. When switching from clk to the fastclk, the displays an[3:1] are dimmer, this is most likely because of the speed it is oscillating. This can be proven by hitting btnC, which counts to the Max FFFC immediately. On top of that the Up and Dw sometimes skips a number because of how fast the clock is oscillating creating a glitch in the edge detector.

- **Conclusion:**
  In conclusion I learned how to design a sequential circuit of D Flip Flop Counters, that can be used to store information and display through multiple seven segments. I better understand how to create a truth table for the counters and the benefits to having counters. I did feel a lot of frustration with the amount of errors I had to deal with during the designing process so I should learn to better organize my code for ease of future troubleshooting.