Justin Liang

ECE 156A

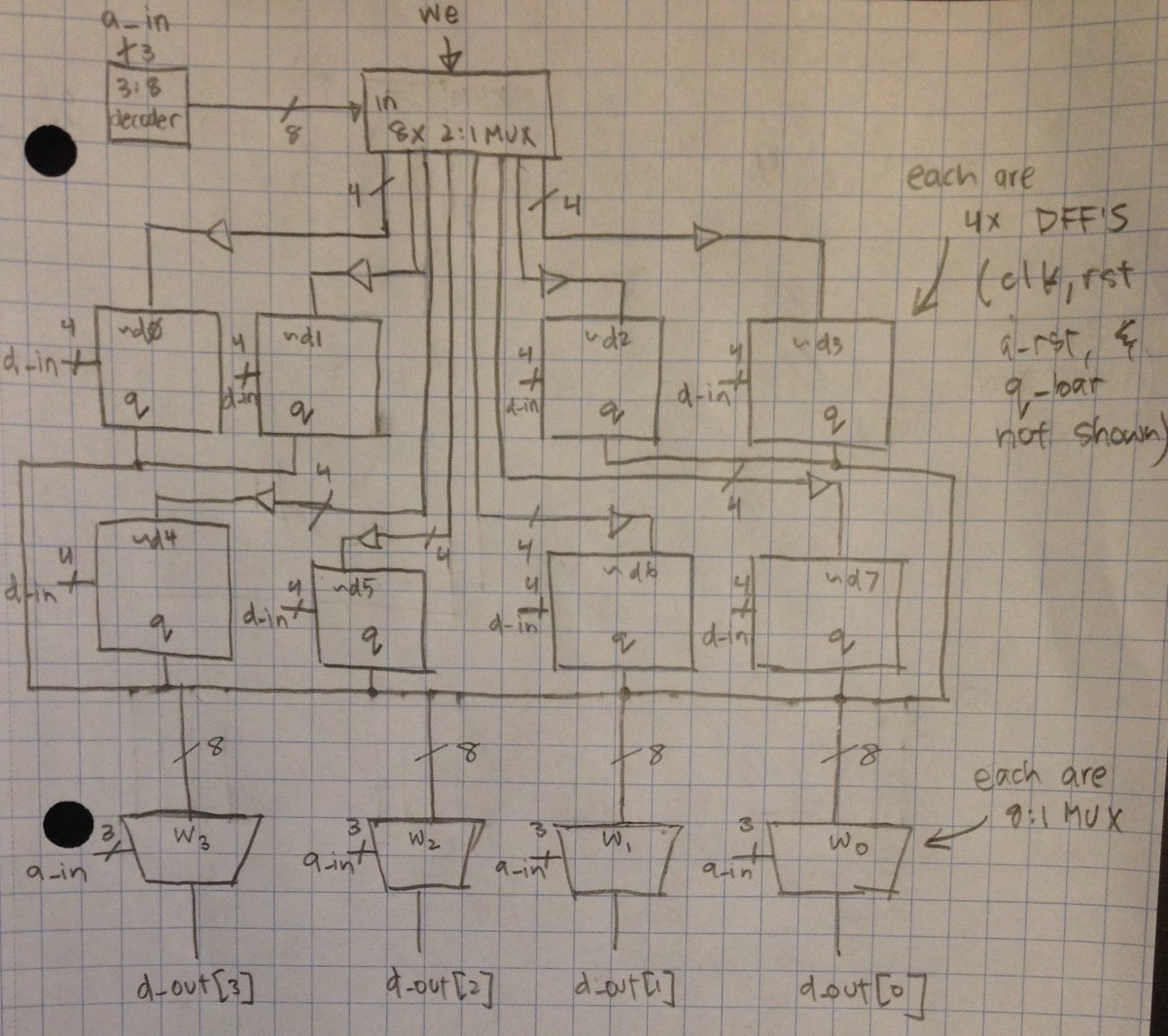
Wang

Due: Oct 29, 2013

**Homework 2**

**Part 1:**

Schematic:



In this schematic, we first start off by introducing a 3:8 decoder. The 3 inputs of the decoder correspond to the 3 different bits of a\_in, while the outputs are fed into eight 2:1 muxes. Next, these eight 2:1 muxes take the inputs from the decoder, and outputs the enable values for each set of D flip flops. The select values for the eight 2:1 muxes are the we (write enable) input. In the schematic, there are eight clusters of 4-bit D flip flops to represent the 8 x 4-bit registers. Each input bit of a cluster of D flip flops correspond to each input bit of d\_in (d\_in[3], d\_in[2], d\_in[1], d\_in[0]). The inversion of the output of the previous eight 2:1 muxes are the enable inputs for each cluster of D flip flops. The output, q, of each cluster of D flip flops are fed into corresponding inputs of the final four 8:1 muxes. These 8:1 muxes are controlled by the three bits of a\_in. Finally, the outputs of the four 8:1 muxes are the individual bits of the final result, d\_out (d\_out[3], d\_out[2], d\_out[1], d\_out[0]). In the schematic, clk, rst, a\_rst, and q\_bar are not shown for simplicity and organization of the circuit. However, in reality, each of the D flip flops have these input values.

Waveform:

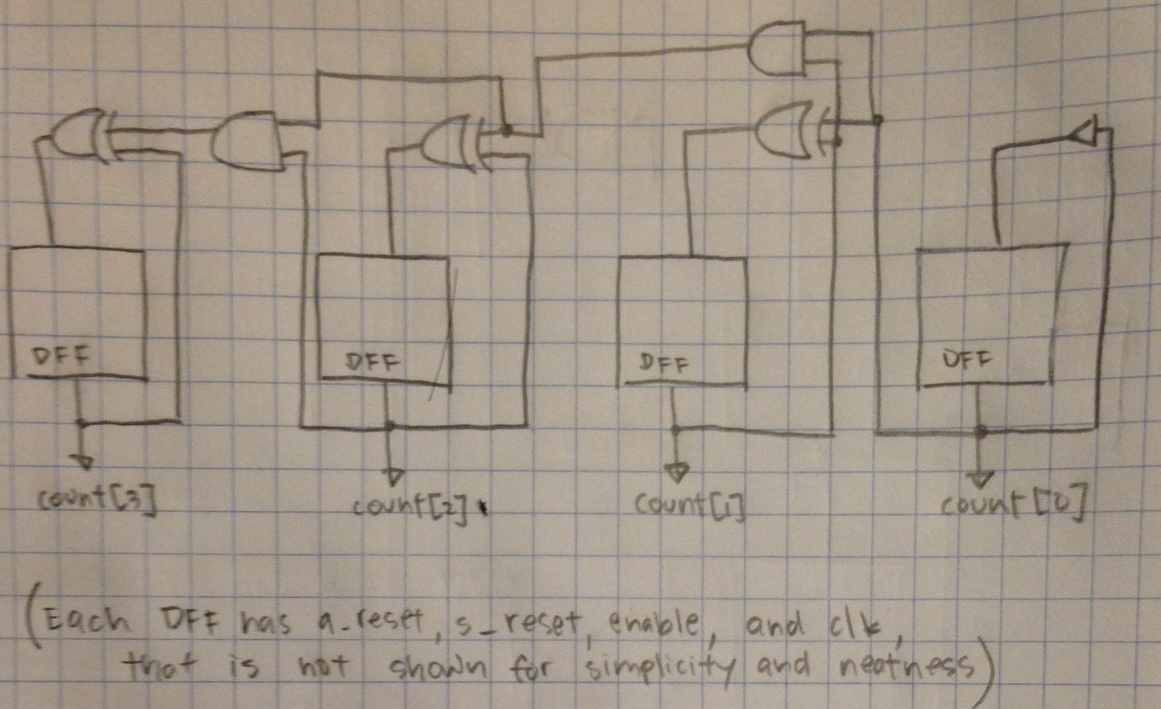
I could not get my D flip flop to work properly with enable and this part of the homework largely dependent on the functionality of the enable input. As a result, I could not produce a proper waveform.

**Part 2a**:

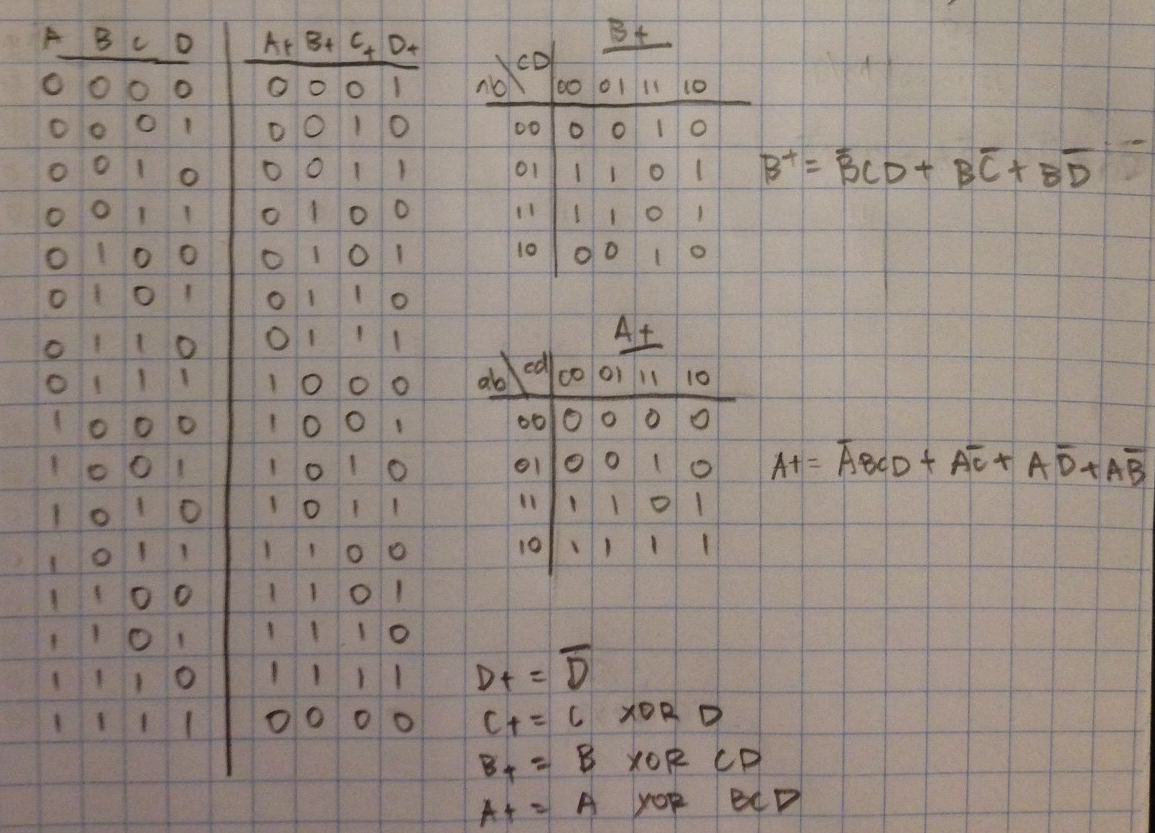
Discussion:

In the upcnt, I am implementing this using a structural design with 4 D flip flops, 2 AND gates, and 3 XOR gates. I wrote down the state diagram for the counter, produced the k-map for count[3] and count[2]. The reason why I didn’t produce a k-map for count[1], and count[0] is that with inspection, I can see that count[0]+ = ~count[0] and that count[1]+ = count[1] XOR count[0]. With the other equations, it was harder to just see from inspection so I had to do k-maps for count[3] and count[2] and find boolean equations. From there, I can see that it follows the same pattern: count[3]+ = count[3] XOR count[2]count[1]count[0] and count[2]+ = count[2] XOR count[1]count[0].

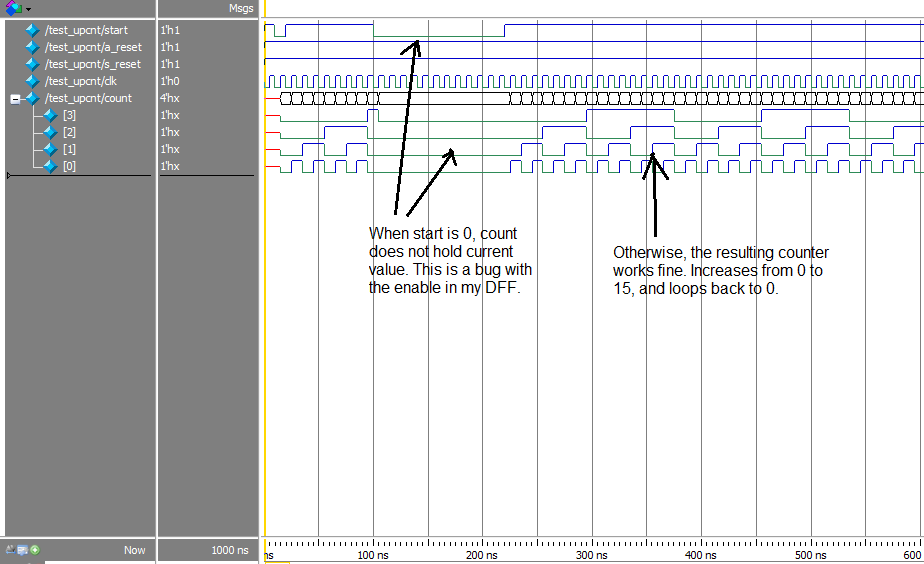
Schematic:



K-Maps and Boolean Equations:



Waveform:

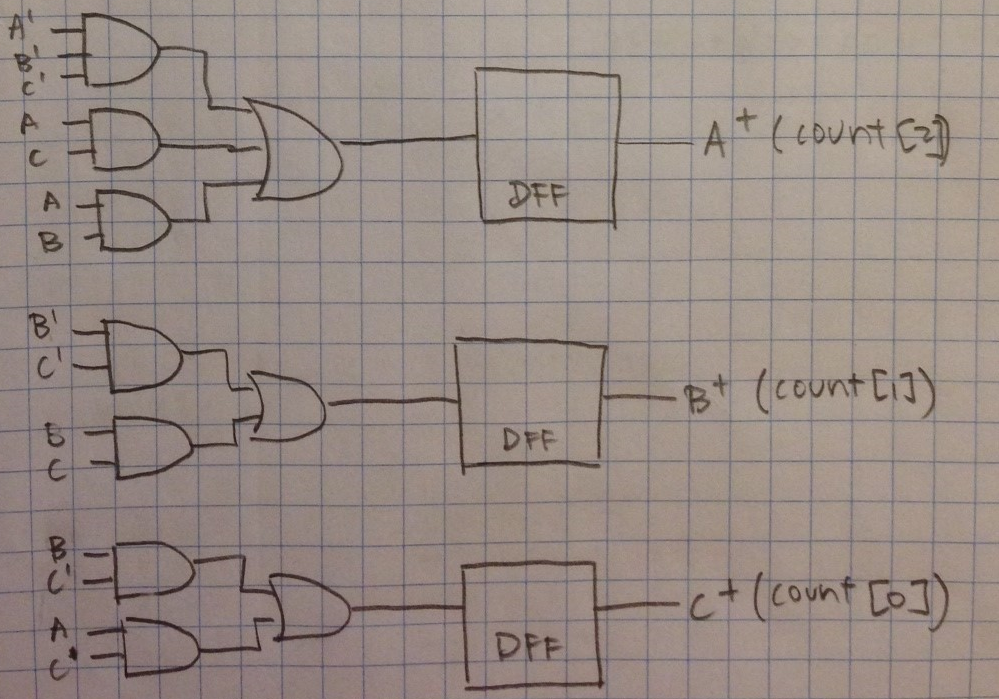


**Part 2b:**

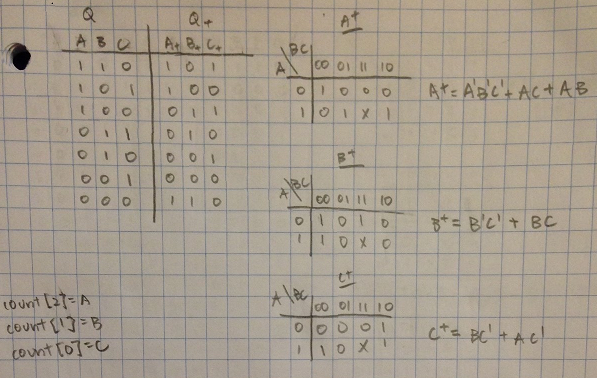
Discussion:

In the dwncnt, I followed the same procedure as I did for finding the boolean equations for upcnt. However, I was unable to find any patterns in the next state, so I had to make K-maps and construct boolean equations for each bit (count[2], count[1], count[0]). I implemented dwncnt using 7 AND gates, 3 OR gates, and 3 D flip flops.

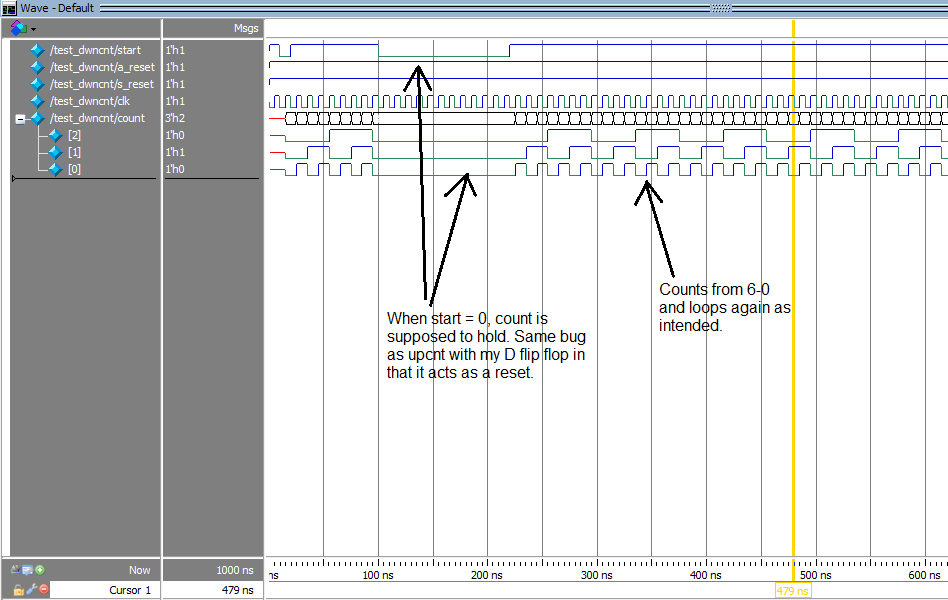
Schematic:



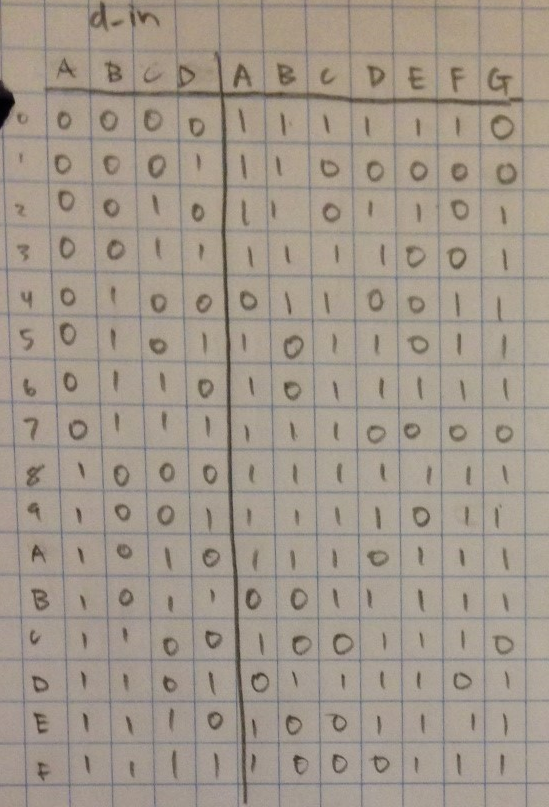
K-maps and Boolean Equations:



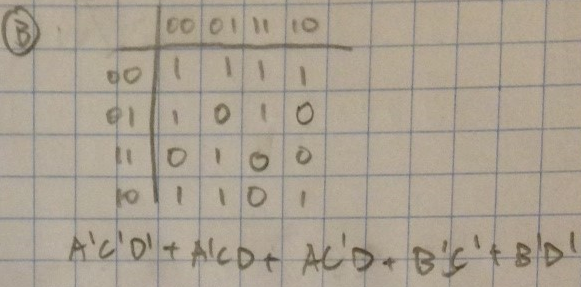
Waveform:

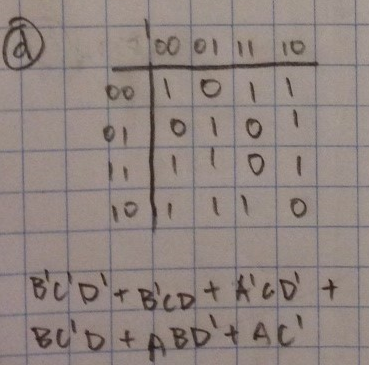


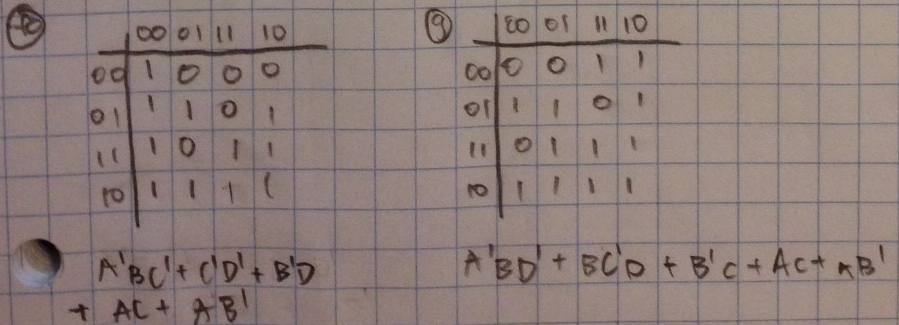
**Part 3:**

In this part of the homework, we are asked to construct a 7 segment display. First, I constructed the truth table for each segment of the display. Below is the corresponding truth table. 

After constructing the truth table, I constructed a corresponding K-map for each segment of the display. With the K-maps, I found the boolean equations with the least amount of literals for each segment. Below are the K-maps:

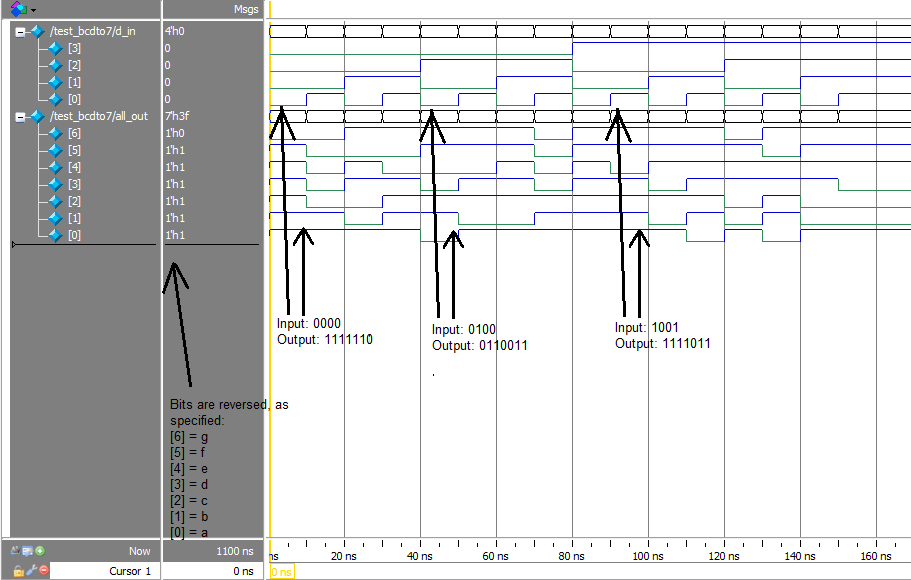






After constructing the K-maps, I just had to program each boolean equation into primitive gates into Verilog. Below is the resulting waveform. As you can see, I have chosen random inputs with its corresponding outputs to verify that the truth table and waveforms are correct.

Waveform:



Conclusion:

In this lab, I had a great amount of trouble getting the enable on the D flip flop to work. Since the D flip flop is the building block for parts 1 and 2, I could not produce the correct waveforms. In part 1, the circuit is completely dependent on the D flip flop’s enable input to work because the enable input is used to control which register is displayed. In part 2, the circuit is somewhat dependent on the enable input of the D flip flop, but not entirely. As a result, I could not get the counter to hold the current value when enable is not asserted. Part 3 was the hardest in terms of the number of the K-maps and Boolean equation minimizations. Other than the D flip flop’s enable not working, everything else was fairly easy and I am certain that my Verilog code is correct for each part of the homework.