Lab 3: Sequential Log, Latches, Flip – Flops, Shift Registers, and Counters

CpE Section 1

Instructor: Telles, Eric

Friday: 4:30 pm – 6:50pm

Thao, Shammah

**Part 1: 4- bit adder circuit**

**Description:** Using the first letter and second and letter of our name we get a constant that will be converted to a binary number. Which will be used as our 4-bit adder.

**Procedures:** Design (create truth table, use K-maps to find equations, and create a circuit schematic diagram) for: A four-bit adder that accepts any 4-bit input and adds a 4-bit binary constant to the input. The output needs five bits

**Engineering Data:**

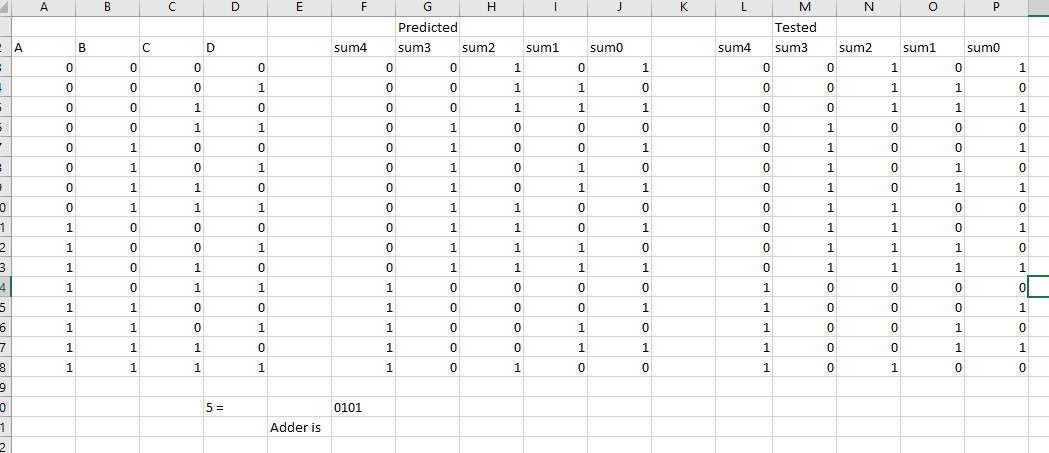
****

Figure :TruthTable

In figure 1 feature above, it shows the original a, b, c, and d numbers and the predicted number which are just the original number added to my constant of 0101. The result was given on the far right.

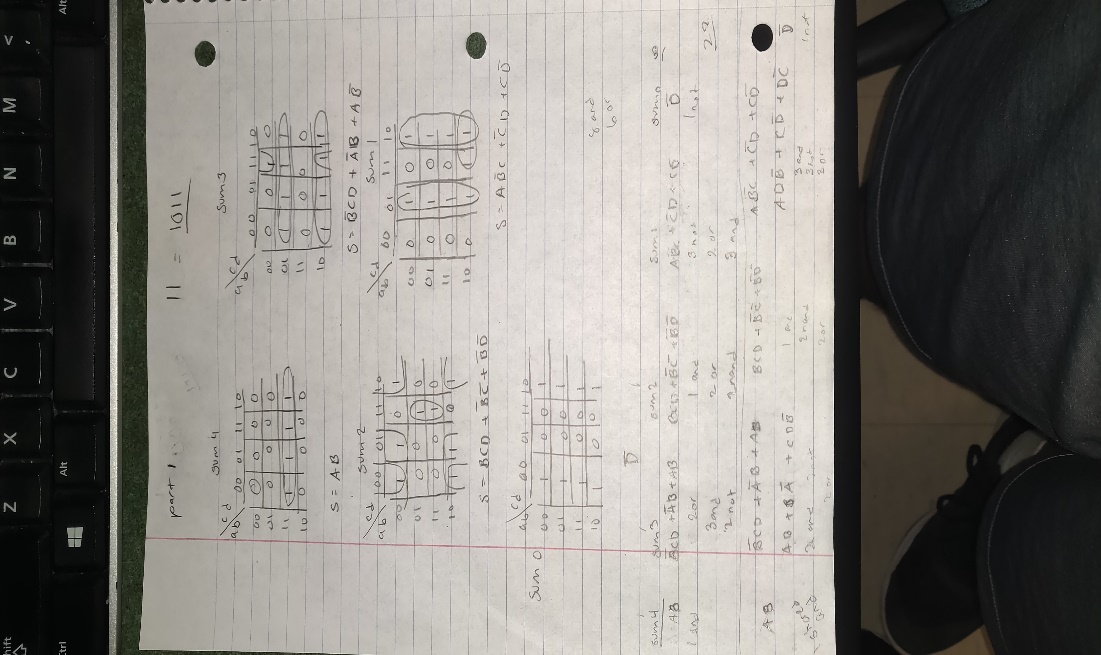


Figure :work

In figure 2, using the truth table from figure 1, I created 5 k-map using 4 section each from the truth table. Which at the end gave me multiple equation which will be used to make a multisim, feature in figure 3.

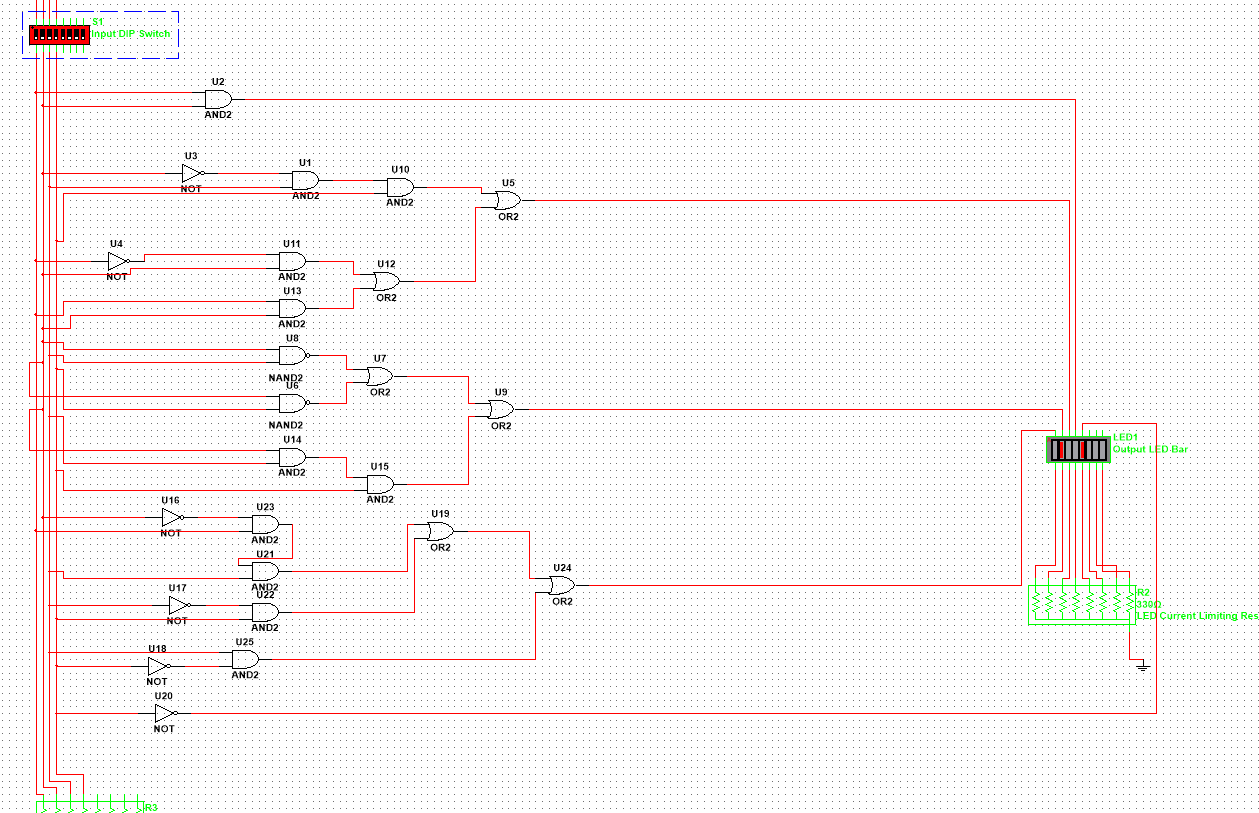


Figure :multisim

**Part 2: f bit adder Verilog**

**Description:** We basically got to plug our equation that we got from part 1 into the program Quartus and write a Verilog code with it.

**Procedure:** - Use your five equations for the ADDER circuit from Lab2 part 1 and write a Data Flow model description in Verilog HDL. Be sure you have 5 equations.

**Engineering Data:**

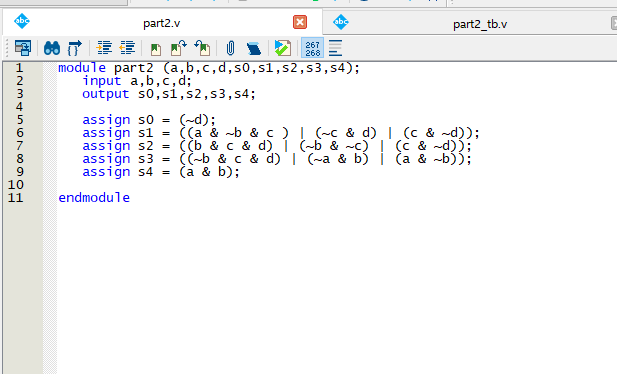


Figure : Verilog

We will have to create a Verilog code using our given equation from figure 2. We did a data flow Verilog which allow us to assign the equation to a variable making it much easier to handle. Feature in figure 4. As the Verilog was test, we were able to get a waveform feature in figure 5.

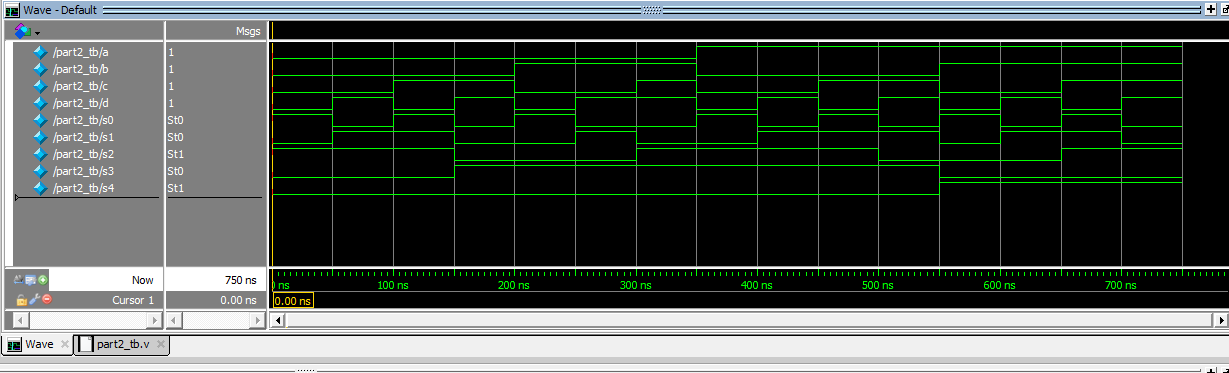


Figure :Waveform

Having to test the verilog and waveform using an fpga, we basically used the pin planner from quartus and assign keys for the fpga, using that after using the switch it works, shown in figure 6, we used the switch 1010 which turned on all our lights, similar to the truth table in figure 1.

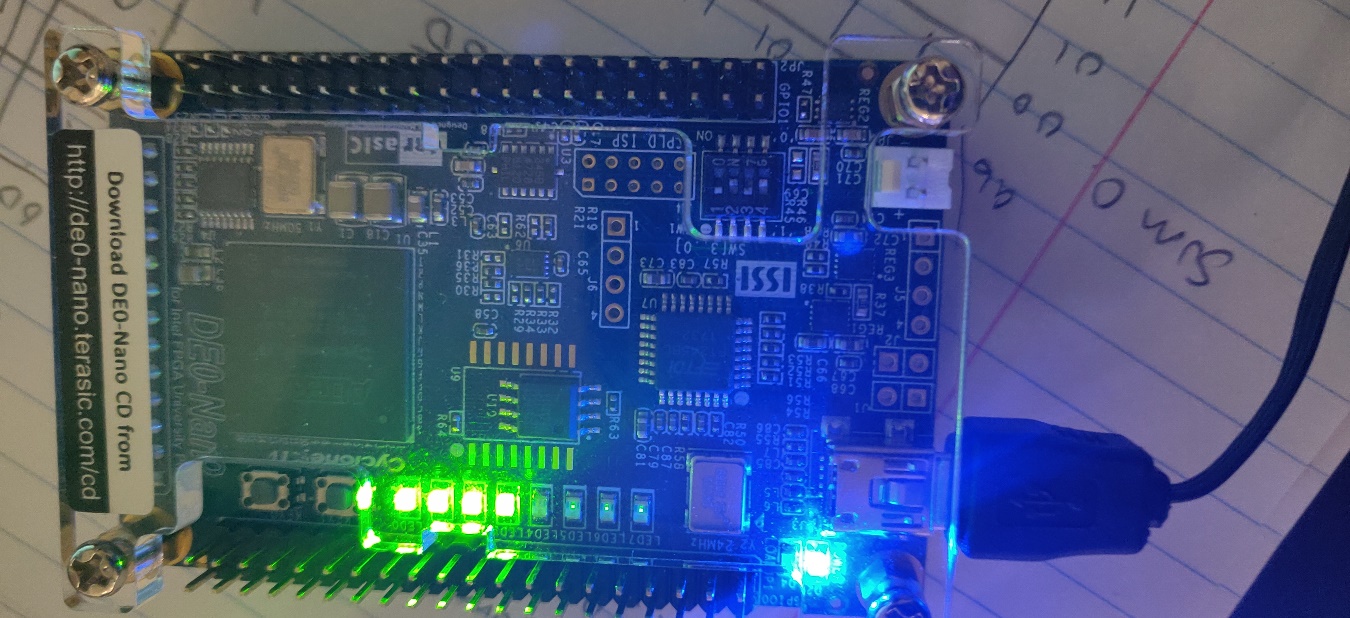
****

Figure :FPGA

**Part 3: full 4-bit adder**

**Description:** We had to create a Verilog code that added 4 bits to any 4 bits.

**Procedure:** Design a full 4-bit adder that adds any 4 bits to any 4 bits. The two 4-bit numbers shall be interpreted as unsigned. Call the 2 numbers A and B. The Function will be A plus B. The number A has four bits called a3, a2, a1 and a0. The number B has four bits called b3, b2, b1 and b0. The adder shall have five outputs called S4 {or CO for Carry Out}, S3 {Sum 3}, S2 {Sum 2}, S1 {Sum 1}, S0 {Sum 0}.

**Engineering Data:**

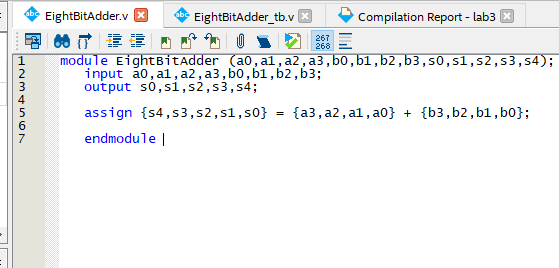


Figure :Verilog

Using data flow again in this verilog coding, we assigned our 4 variables to be able to add another 4 different variable.feature in figure 7

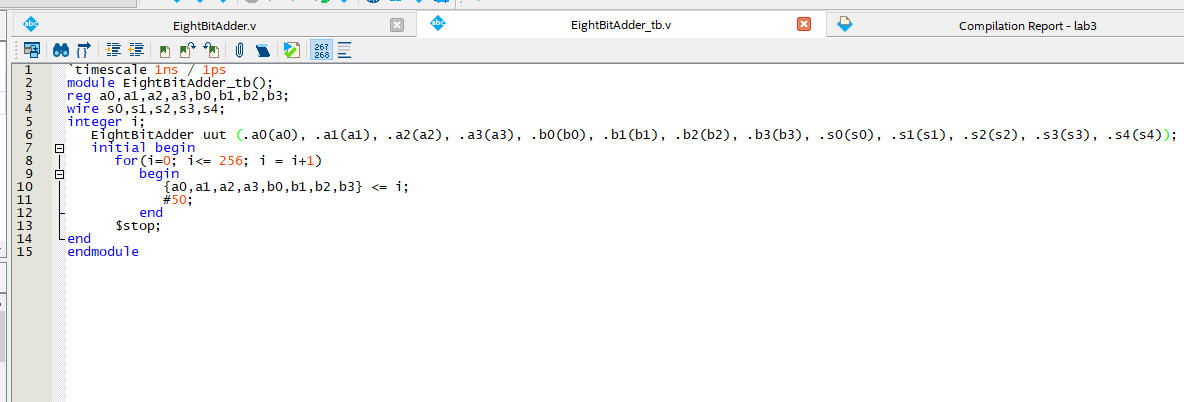


Figure :Testbench

In figure 8, creating the testbench, we made a for look which goes all the way to 256 since 2 ^ 8 is 256, using 8 since we got 8 different variables and 2 since we are using 2 compliments. Using figure 8 testbench, we were able to create the waveform in figure 9.

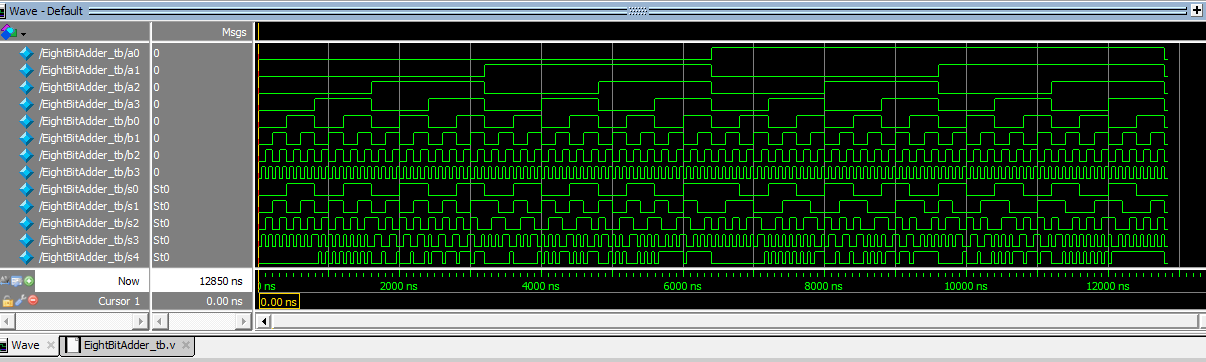


Figure :Waveform

**Part 4: Design of Comparator using gates**

**Description:** We had to design a 4-input comparator that compares 2-bit numbers.

**Procedure:** Design a 4-input, 3-circuit that compared two 2-bit unsigned numbers. You can call these numbers a1 a0 and b1 b0. So here a1 is the most significant bit of input A, and a0 is the least significant bit of input A. This circuit should have 3 outputs, which indicate whether A > B, A = B or A < B. You can label these outputs G (which means a1 a0 > b1 b0, where G stands for greater), E (which means a1 a0 + b1 b0), and L (which means a1 a0 < b1 b0). This is called a comparator circuit, because it compares the binary number a1 a0 with the binary number b1 b0

**Engineering Data:**

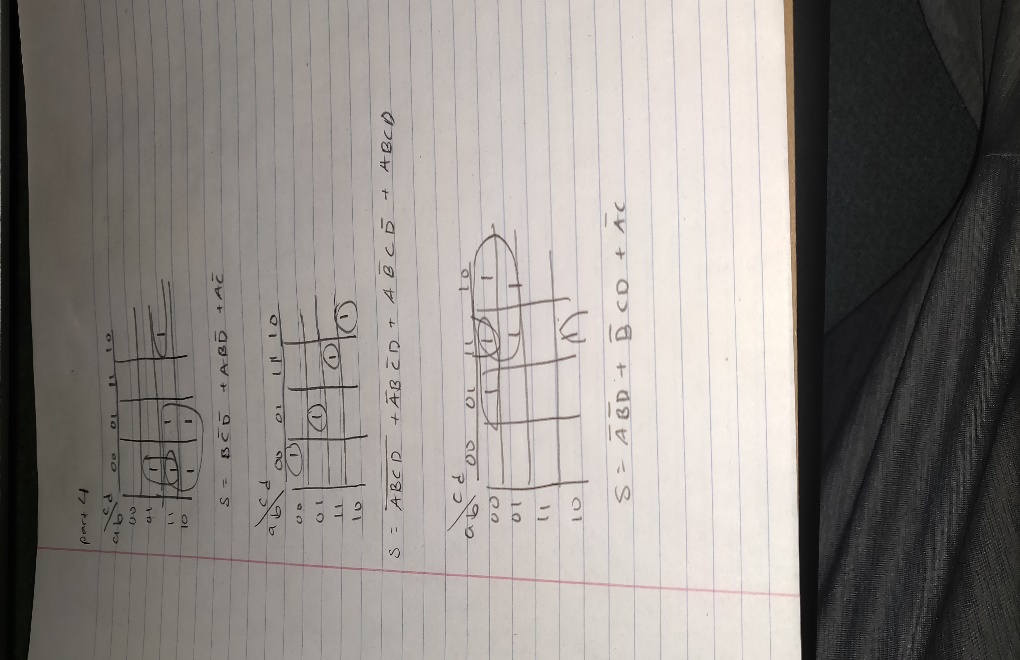
****

Figure :k-map

In figure 10, we got 3 equation using k-map. Which was used to build the Multisim in figure 11.

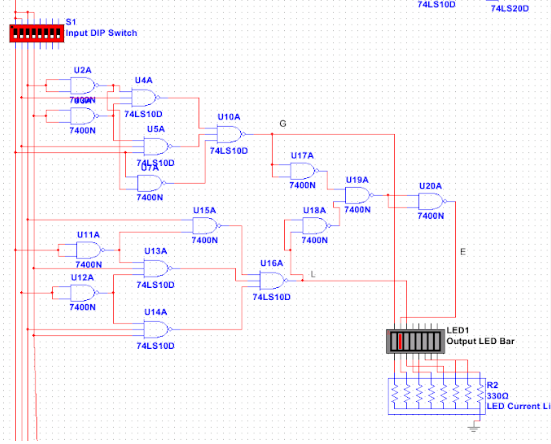
****

Figure :Multsim

The equation that was acquire from figure 10 was then assign to 3 different variable. Using data flow feature in figure 12.

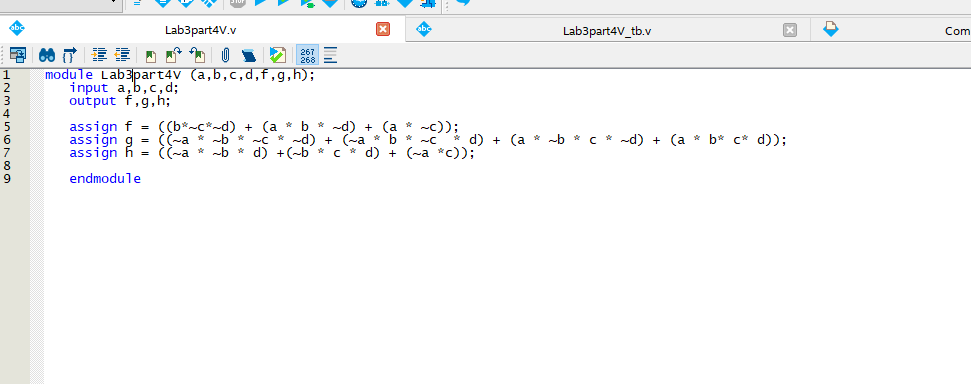
****

Figure :Verilog

Using the long way of doing thing, we made a testbench printing out all the truth table variables. Feature in figure 13.

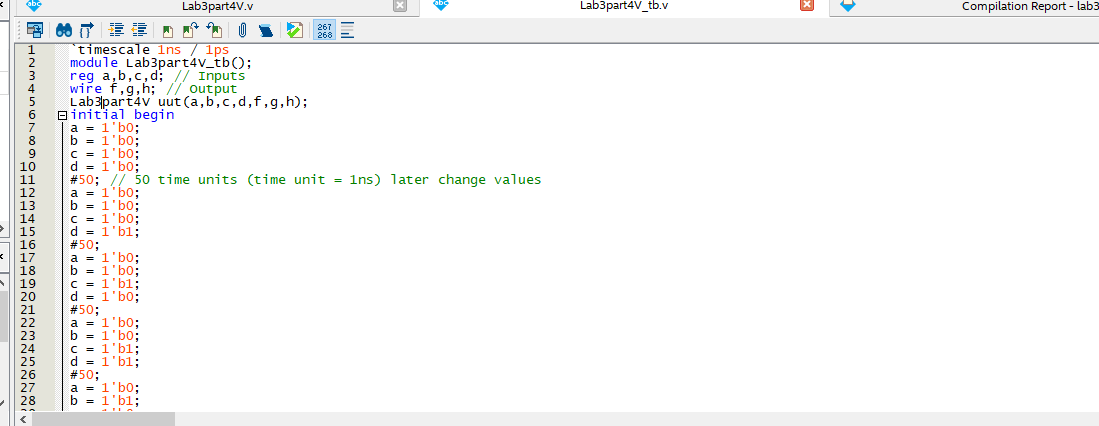
****

Figure :Truth table

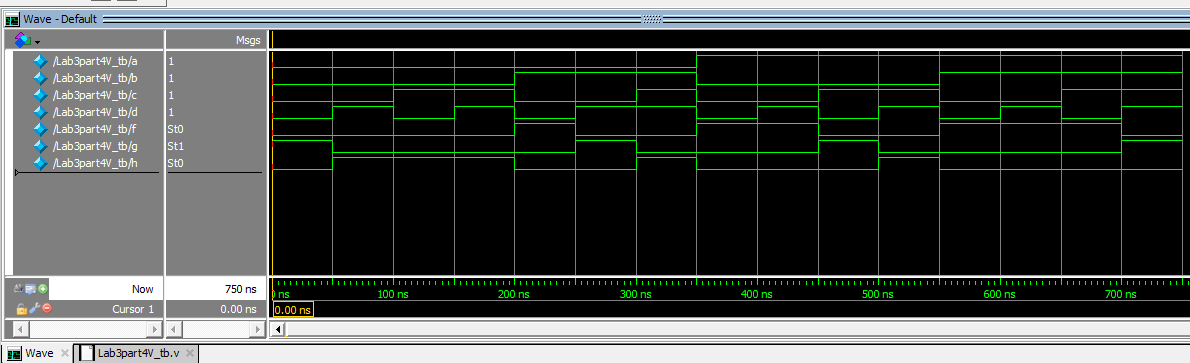
The Verilog and testbench, we made a waveform, feature in figure 14, which shows the 3 different equation at work.

Figure :waveform

**Part 5: Unsigned 3-bit comparator**

**Description:** we had to make a 3-bit comparator

**Procedure:** Design an unsigned 3-bit comparator such that “A” is compared to “B”.

Inputs: A three bits (a2, a1, a0)

Outputs: E A equal to B B three bits (b2, b1, b0) L A less than B G A greater than B E, L, and G are each one bit

**Engineering Data:**

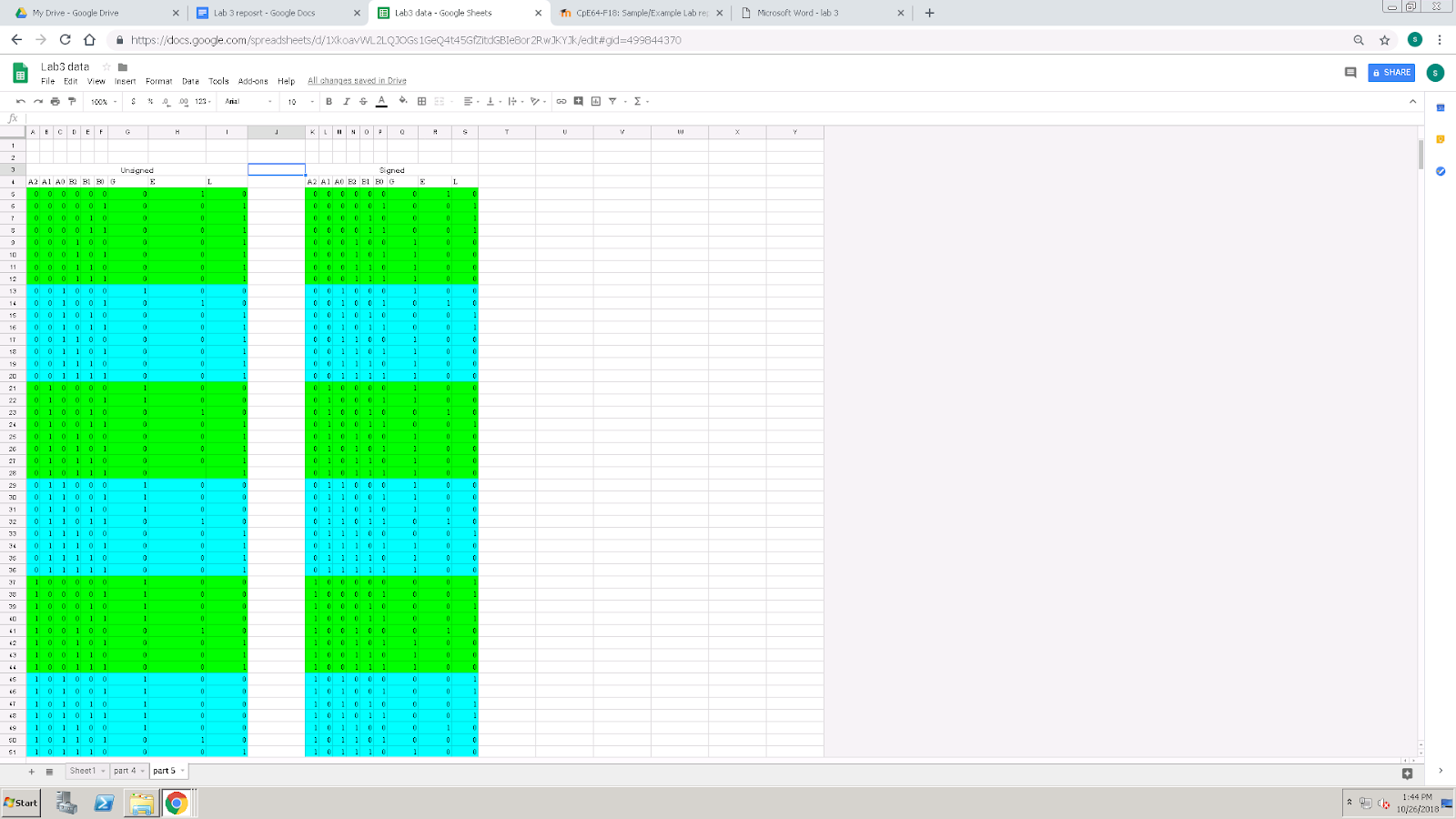


Figure :truth table

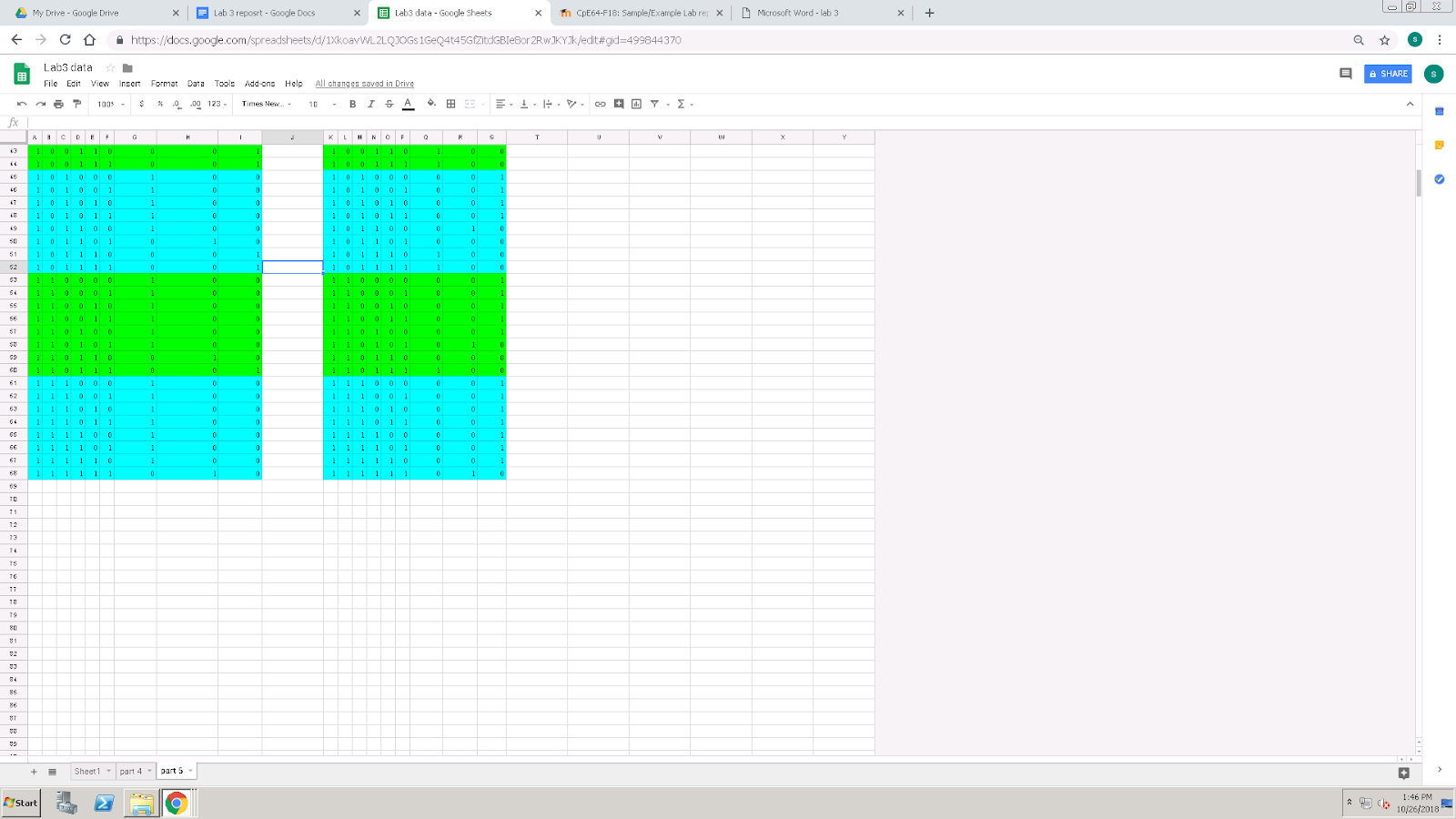


Figure 16: truthtable

The truth table feature in figure 15 and 16 are a combine truth table what was made to compare 3 bits

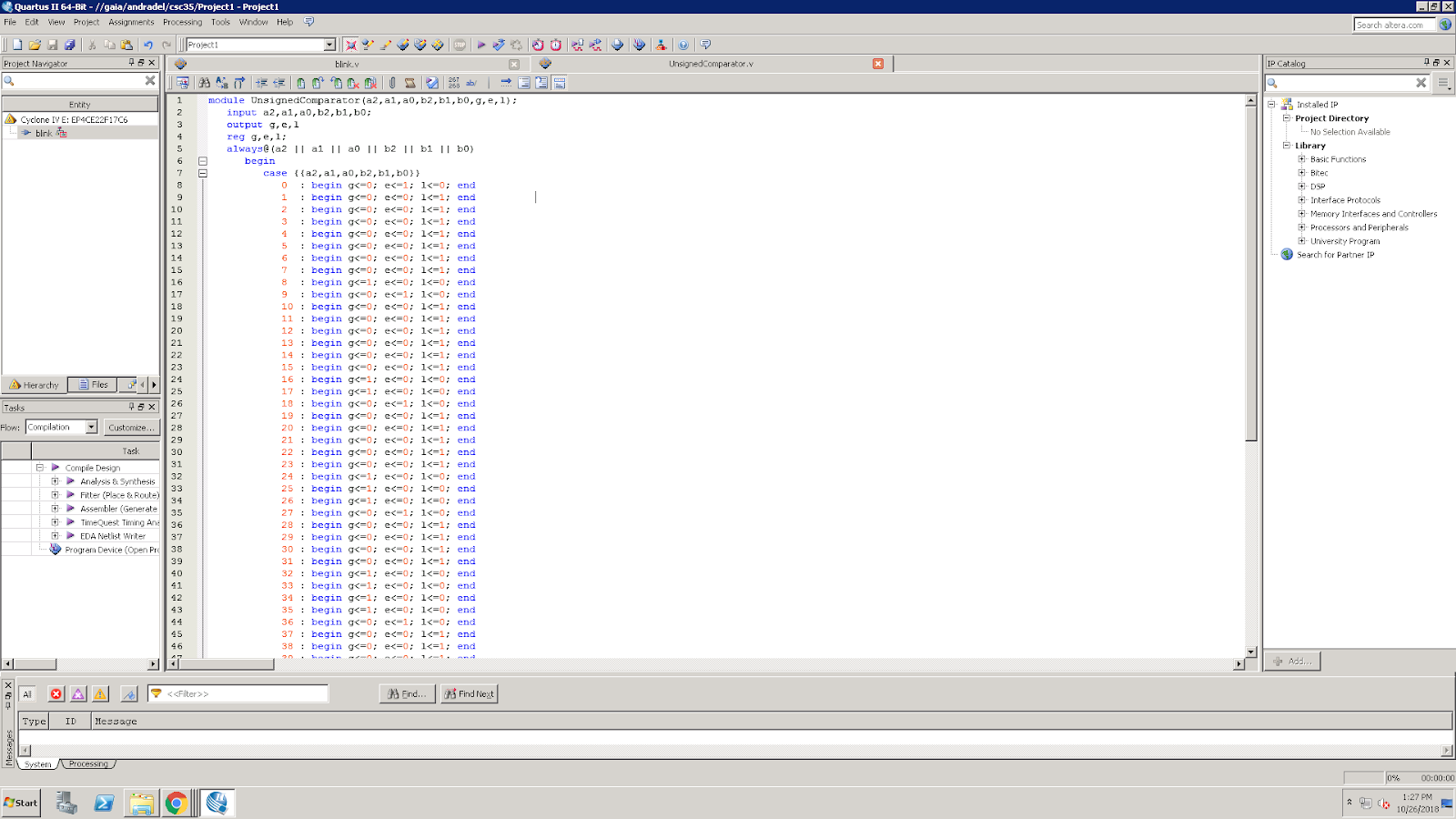


Figure :Verilog

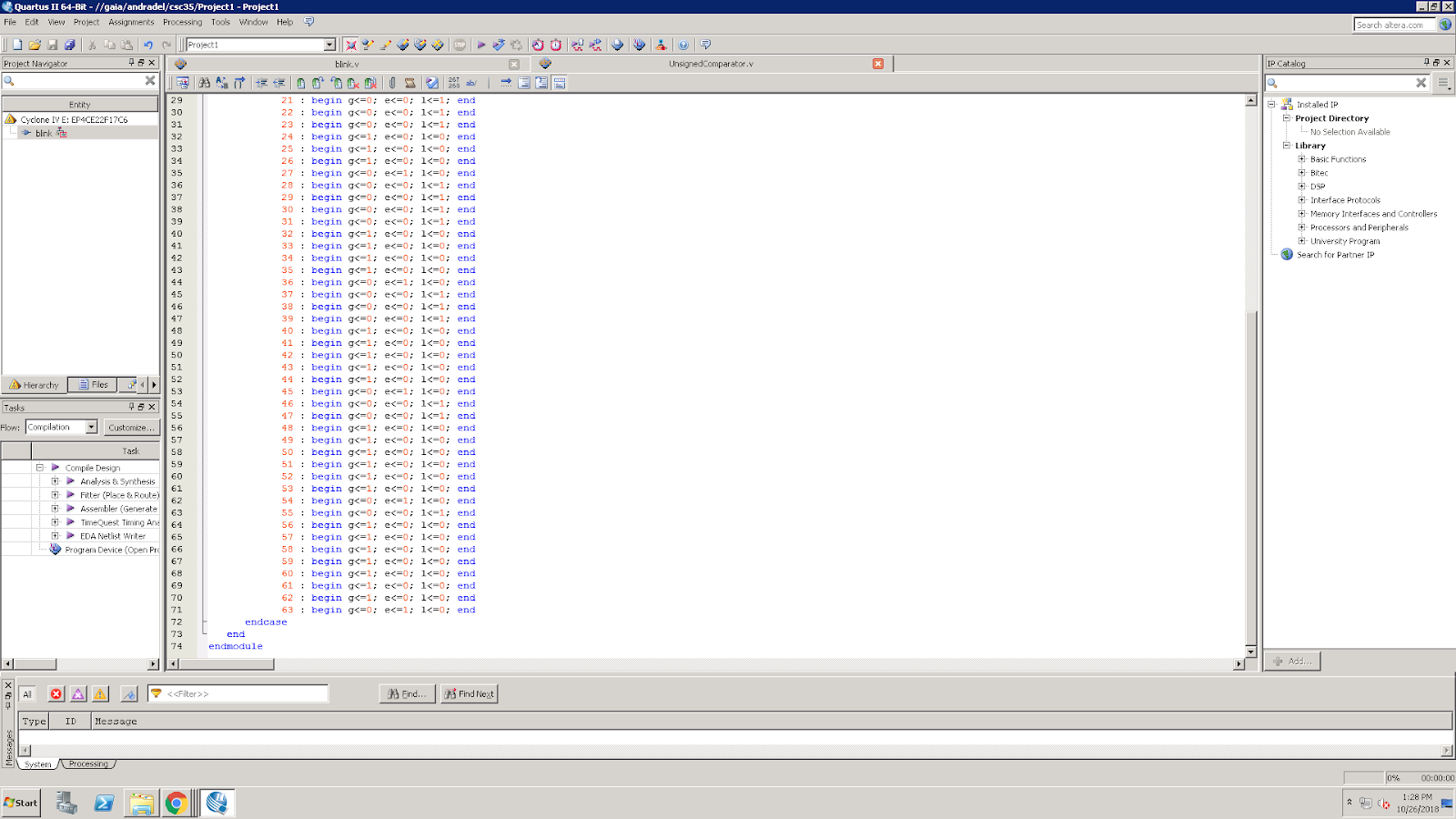


Figure :Verilog

The Verilog code that was made, feature in 17 and 18, was used as a switch case.

**Part 6: design a 3-bit comparator signed and unsigned switch**

**Description:** We basically had to edit the code from part 5 and make it so it’s less wordy

**Procedure:** - While the solution for Part 5 was better than doing a six variable K-map, it still was a lengthy design solution (lots of lines of Verilog code). You would suspect that Verilog is better than this! Verilog has the following "logical" operators: = = equality, > greater than, < less than, >= greater than or equal, =< less than or equal to, and! = inequality. Be adventuresome and redesign a solution to the comparator problem by using the relational operators. Compile, download, and test your design. Again, include the segment of the report file that included the test bench source code. Compare the equations with part 5 and comment about any differences in the equations using the two techniques done in each part.

**Engineering Data:**

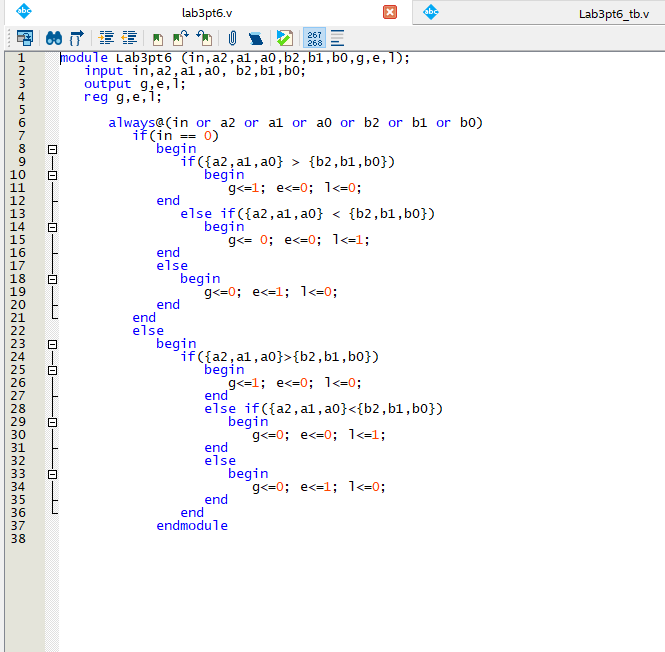
****

Figure :Verilog

Figure 18, was basically the Verilog code from figure 17 and 18, shorten up using if statement and if then statement.

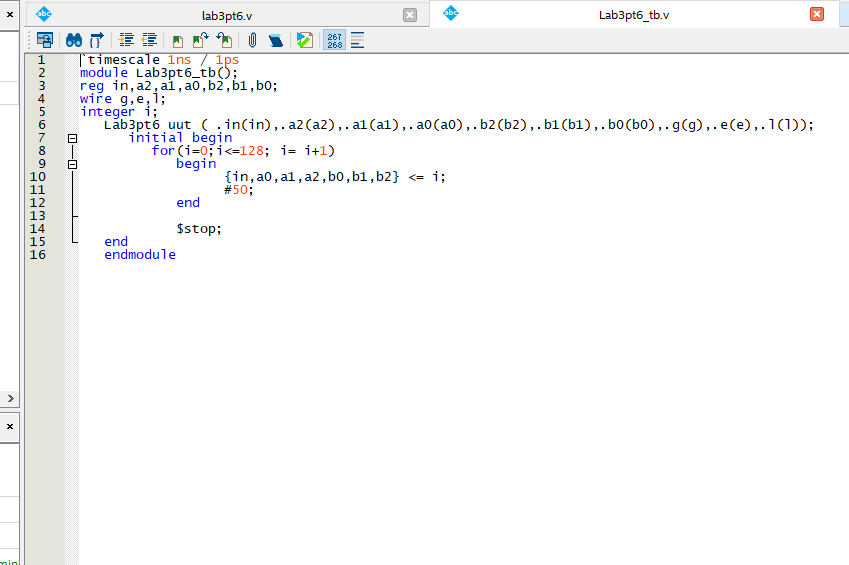
****

Figure :testbench

The testbench for figure 18 is feature in figure 19 which also used a for loop to make it less tedious. The Verilog and testbench was then tested to give up a waveform feature in figure 20.

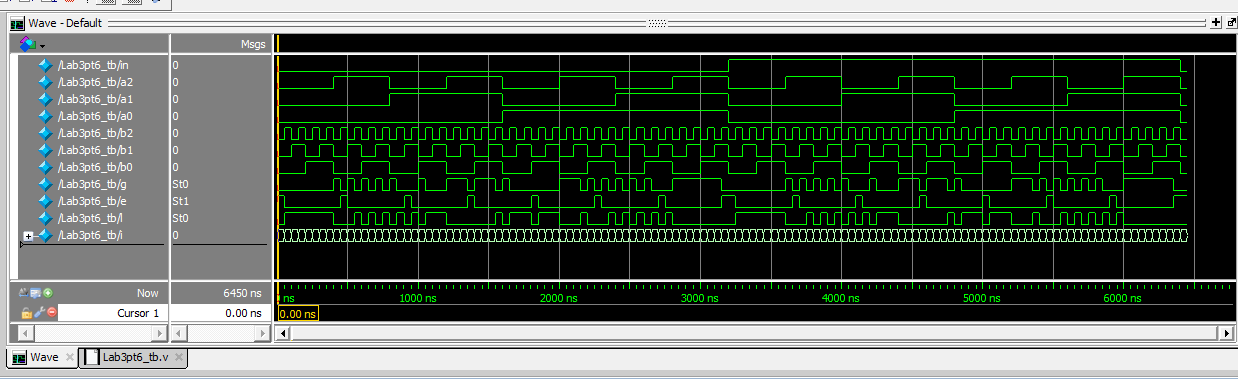
****

Figure :waveform

**Part 7 Special design**

**Description:** we make a Verilog design for an 8 input and 2 output.

**Procedure:** Design a logic system that has eight inputs and two LED outputs. Use your eight pin dip switches as inputs.

**Engineering Data:**

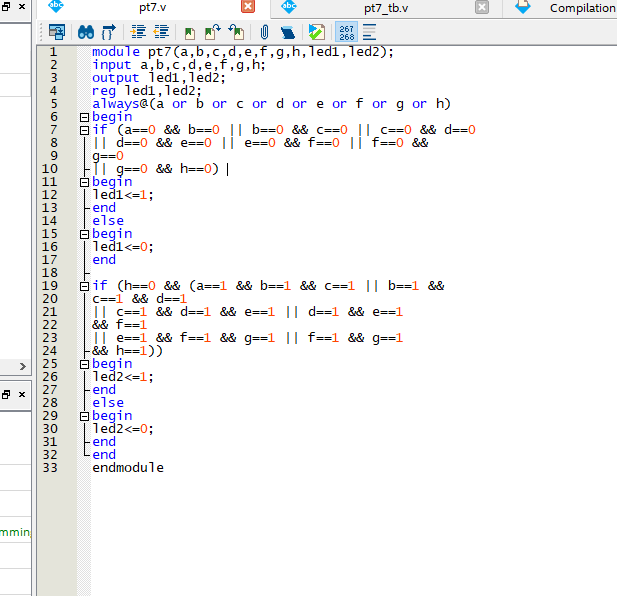
****

Figure :Verilog

Figure 21, shows the Verilog coding of and 8 input and 2 output coding, using if statement. Trying to figure out the flip-flop switch so it changes every time something is different.

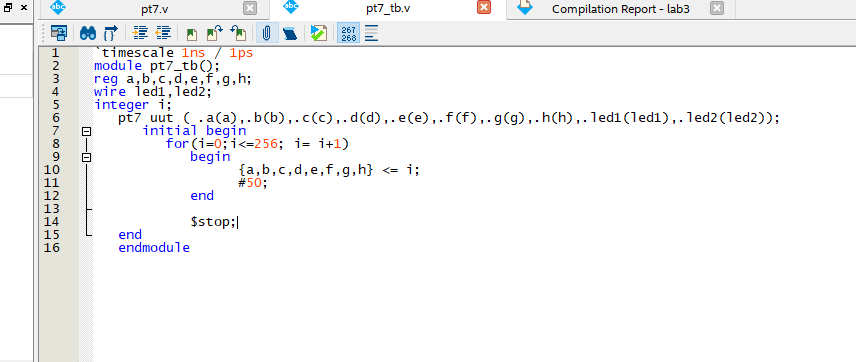
****

Figure :testbench

In figure 22, we have a test bench, using a for loop to create an output of a truth table. Using figure 22 and 21 we made a waveform feature in figure 23.

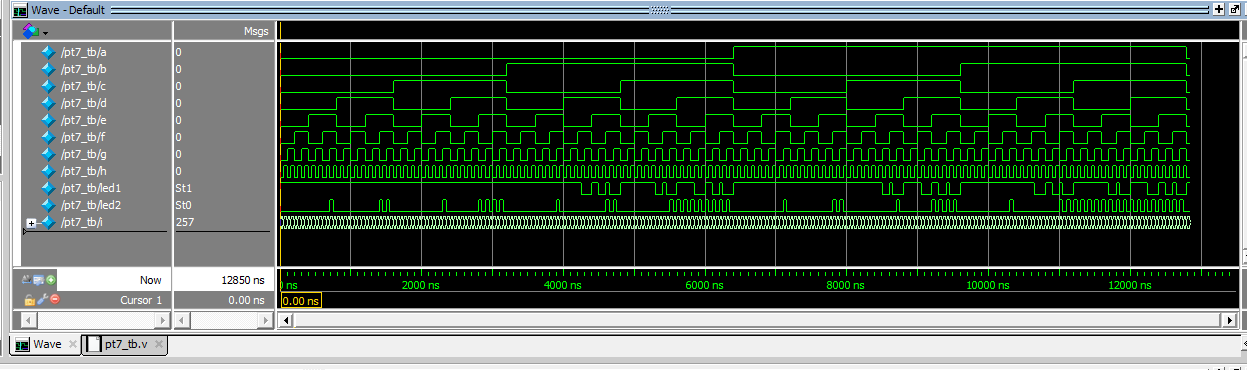
****

Figure :waveform

**Conclusion:**

In conclusion this lab was hard to do since it was time crunched into one week for such a long lab. The hardest part about this lab is trying to figure out something that haven’t been taught in class yet, such as flip flops and comparators. Part 1 of the lab was simple but as we went on it got more different since we had to figure some part out our self. One helpful resource was the sample lab from previous years, it gave us hints to how to get the correct solution