Ramzey Ghanaim

Logic Design Basics 1

Introduction

In this lab I was introduced to how the program Multimedia Logic works (MML). Once I had an understanding of the program I was able to apply my knowledge of logic to replicate De Morgan's law, a given truth table, and a guessing game on schematics in MML.

Part A

In Part A of this lab I demonstrated De Morgan's law. This law is what relates AND and OR gates. The law states: A' AND B' = (A OR B)'. The circuit I created in Multi-Media Logic (MML) represents this law. To represent A' and B' I used an inverter on these two inputs. I then combined them to an AND gate to represent the AND function and put that output on a LED. To represent A OR B I sent the two inputs into an OR gate. However only (A OR B)' can relate to the AND gate A OR B alone. Therefore I inverted the OR gate, to create a NOR gate. This inversion is what takes the place of ' in (A OR B)'. I sent the output of this NOR gate to an LED and checked the different combinations of A and B to confirm De Morgan's law.

Part B: Sum of Products

Original Design

In Part B of this lab I was given truth table and was asked to create a logic design based off of it. I started by looking at the three cases of IN 1, IN 2, and IN 3 that produced an output of 1. From here I worked on each case one at a time. In order to create a product of the inputs to get an output of 1 I inverted all of the logic zeros and then had the three inputs into an AND gate. I left the logic 1 inputs alone since they did not need to be inverted to create a logic 1 output of the AND gate. After the necessary inversions, each of the three cases went to their own AND gate. These are the products, and only one of these 3 AND gates has the potential to produce an output of 1 at any of the possible cases. So I combined them into an OR gate to select the output of 1 that may be produced from one of the three AND gates. This OR gate produced one output which was connected to the LED. The OR gate represents the sum of all of the three products (AND gates). Hence, I created a Sum of Products.

Part B: NAND only Schematic

When changing my design from or and AND gates to just NAND gates I replicated inverters, AND and OR gates that I had in my original design. Every time a single input is inputted twice into the same NAND gate, the NAND gate functions as an inverter. So my first step was to re-draw all inverters. Next, I put the inverted inputs into 3 different NAND gates, similar to how I put inverted inputs into AND gates in the original design. Each NAND gate once

again represents the three different conditions that can be used to have an output of logic 1. However, this time the three NAND gates will give the opposite output that I want. Combining these three NAND gates into another NAND gate acted as the three AND gates being combined into an OR gate. Since I used a NAND gate to combine the other three NAND gates, there was a "double inversion" as I like to call it and the three NAND gates giving the opposite output that I wanted was eradicated. The correct logic combinations was the result.

Part B: Transistor Count

In the original design I used a total of 36 transistors. Each inverter has two transistors while each AND and OR gate have six. Summing the total came out to be 36 transistors. The second design consisted of only NAND gates. For this design I had a total of 40 transistors. Each NAND gate consists of four transistors, and there were ten NAND gates in my design. NAND and OR gates seem to have less transistors than AND and OR gates. This is result of physics and logic. When combining 2 p-type and 2 n-type transistors, depending on the orientation of these transistors, (parallel or series) a NAND or NOR gate is crated. Since AND and NOR gates are the opposites of NAND and NOR gates, AND and OR gate are created simply by adding an inverter to a NAND and NOR gates. As a result, this adds an extra 2 transistors to AND and OR gates.

Part C

Simplifying Part B

In part C I reduced the circuit I made in part B due to boolean algebra. The sum of products equation that represented the logic gates in part B was:

$$Q = A'B'C' + AB'C' + ABC'$$

Where each product represented the three cases when a logic 1 was the output of the truth table, as described earlier in the report. C is input 2, B is input 1 and A is input 0 I simplified this equation by firs factoring out AC' from the second two cases:

$$Q= A'B'C'+ AC'(B'+B)$$

Next, I used the law: A' + A = 1 since no matter what, either A or A' is going to be 1. I know that this is an OR gate because B'+B is using addition:

$$Q= A'B'C' + AC'(1)$$

$$\Rightarrow \quad Q = A'B'C' + AC'$$

From here I factored out C' to get:

$$Q= C' (A'B'+A)$$

And lastly, I used the property A'B+A = A+B to get:

Q= C' (A+B')

And this is the final equation which I used to represent the diagram in part C. First I had an OR gate to combine A and B with an inverter on B. Then I combined the OR gate to an inverted C to an AND gate to produce the correct output. In this design, I had a total of 16 transistors.

Part C: NAND only Schematic

Similar to part B's NAND schematic, when I re-created the circuit using only NAND gates I just replicated what I created in the first schematic of part C. Once again I put one input into one NAND gate twice to replicate inverters. I then used a NAND gate to replicate an OR gate, by inverting and combining IN 0 and IN 1 into a NAND gate. A double inversion occurred in Input 1 which allowed me to remove the extra 4 transistors. To replicate the AND gate used in part C's first schematic, I used a NAND gate and once again, used another NAND gate as an inverter to get the correct output since the NAND gate reversed what the AND gate would normally produce. This logic design has 20 transistors.

Part D

In part D I used a random number generator to create a number for the user to guess. From what I have learned from AP Computer Science in high school the computer generates a number and it gets multiplied, added subtracted or divided by another number. This in turns leads me to believe that there is a formula to calculate a so-called "random" number. Since computers function so logically there really is only logic following a formula that can output a "random" number.

Part C: Design

For the design of part D, I started out with a truth table that contained the two switches the user is going to use to guess the number:

A	B	Output	Symbol
0	0	1	A'B'
0	1	1	A'B
1	0	1	AB'
1	1	1	AB

From here I replicated the truth table on a schematic diagram for both the user's input, and the random number generator's input. After this, I wanted to check to see if each case from the user's switches matched with the same case from the number generator. To do this I had 4 AND gates that compared each of the cases from the generator and the user input to check to see if they match. Lastly, these AND gates converged to an OR gate to see if any of the cases matched up, and if so, it produces the output of 1 to turn on the light.

Conclusion

In this lab I was able to further enhance my understanding of Logic by practicing boolean algebra, and designing logic schematics for a variety of different scenarios. I was also able to get a better understanding of how powerful NAND gates are since they can replicate any kind of function. Lastly I was able to apply my knowledge of logic to create a simple guess the number game. After creating this game, I now completely understand that all computers are made of is just logic calculations of 1's and 0's.