Ramzey Ghanaim

Lab 2: Tic-Tac-Toe

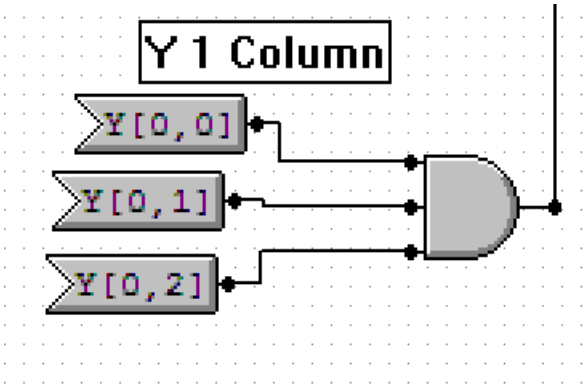
Introduction

In this lab I was created a game of tic-tac-toe to determine the winner of the game. The game is a basic 3 by 3 grid where players can put X's and O's, one at a time, to get a combination of three X's or 3 O's in a row. In my case, I used Blue and Orange instead of X and O, however the same game mechanics still apply. The way I designed this game was in a program called Multimedia Logic in which I used logic to determine the winner of the game.

Solving the Problem

The game starts out with a switch can be used to represent X and O (blue and orange in my case). When the control dial is at 0, each spot on the grid is blank. When the dial is set to 2, a blue square replaces the blank space. . When each dial is switched to 4, an orange square replaces the blank space. There are 9 dials. Each dial corresponds to each position on the grid in the same order the dials are positioned. Each dial had 3 outputs to the grid. The first was for blue, the second for orange, and the third controlled the blank square. If the dial was moved to 2, the output that represented blue would be active, or 1, while the other two outputs are zero (inactive). This same thing applies to orange as well.

I solved the game by using a brute-force method. This method means that I checked each possible combination to determine a winner. For example, looking at the first column, if we wanted blue to win, the first dial AND the second AND the third dial all have to output 1 for blue. To check this I had the blue output from each switch go into an AND gate to check if each switch was outputting 1. If all three were outputting a 1, then blue wins since there are three blues in a row. I replicated this 8 times for the 8 different combinations of blue winning. Then, I was left with 8 different outputs/cases. However I only wanted one so it could light up a single light if blue wins. I used OR gates to combine these 8 outputs in OR gates. I used OR gates because, if the first case, OR the second, OR the third case, etc. became true, the light would turn on. Next I did the same exact thing for orange, only this time I had the switch outputs come from the orange output rather than the blue. This was the only difference between the two colors. In total I checked for 16 different combinations of winning. For simplicity, X represented orange, and Y represented blue. Here's an example of checking the first column for blue:



Each number in the brackets corresponds to a location on the board in a [column, row] order. The order starts at zero and goes until 2. For example, the top left location of the game is: Y [0,0]. The location below this location is: color [0,1]. This pattern continues for the rest of the locations on the tic-tac-toe grid, and repeats for Orange with the letter X instead of Y.

Ties

In this game, a tie is considered when both blue and orange win. Normally this would not be the case in the real game for for the purposes of this lab, it is. I have one AND gate that checks to see if blue AND orange's lights are on, and if so, outputs a 1 to turn on the "Tie" LED.

Gates and Transistors

In my design I used a total of 25 gates in this project. 12 gates were used to check if blue won, and 12 more were used to check if orange won. The last gate checks to see if there is a tie. It is the combinations of multiple transistors that makes up a gate. The number of transistors for these AND and OR gates can be calculated using the formula: 2n+1, where n is the number of inputs in each gate. When doing the calculation, my project has a total of 194 transistors. I could have used less transistors if I used more inputs in each gate. I would have been able to eliminate the need for a few OR gates if I had more inputs in a single OR gate. Basically, instead of combining inputs into multiple OR gates, I could have just combined these inputs into a single OR gate, and thus lowering the amount of gates/transistors used in the project.

Changes and Revisions

In this lab I made one major revision. When I first created the lab I did not understand that pointers existed in Multimedia Logic. My first design had the entire logic design and game on a single schematic page. It was very well organized, however with the number of wires needed to construct the logic for this game, it looked a little un-appealing. Since I already had the logic down All I did was cut and paste the logic gates to different schematic sheets, and added pointers to the inputs. I also added pointers to the dials. Overall, pointers made my program look much simpler and more appealing than it did before.

Conclusion

In this lab the logic used to construct his game was very simple. I did not necessarily enhance my understanding of logic. However, I was able to learn organization mechanics by using nodes and pointers. Although, I would have to say the biggest take away I have from this lab is that simple logic can be used to construct a game, which to me shows how logic is truly the basics of the computer.