Project 3: Tic-Tac-Toe

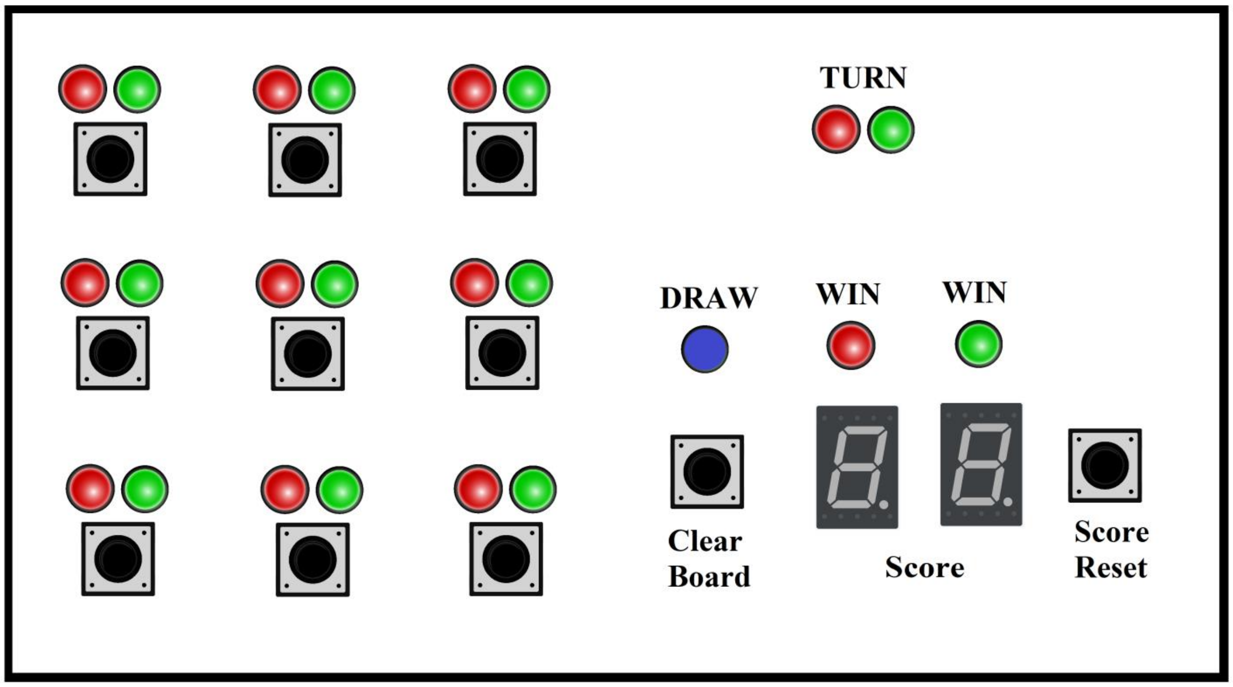
Ahmad Malik
The Cooper Union
Digital Logic Design - ECE150
Professor M. Billoo

Project Description

The objective of this project was to create a two player game using logic gates. I decided to go with tic-tac-toe.

Tic-Tac-Toe is played on a 3x3 grid. Each player assumes a symbol, typically circles or crosses, and takes turn strategically filling the grid. The first person to fill three cells in a row with their symbol (up/down, left/right, or diagonally) wins the game. If all cells were filled and both players were unsuccessful in creating three in a row, that game is a draw. Once the grid is cleared for the next game, the winner of the previous game gets to go first or in the case of a draw, the person who had went first in the previous game.

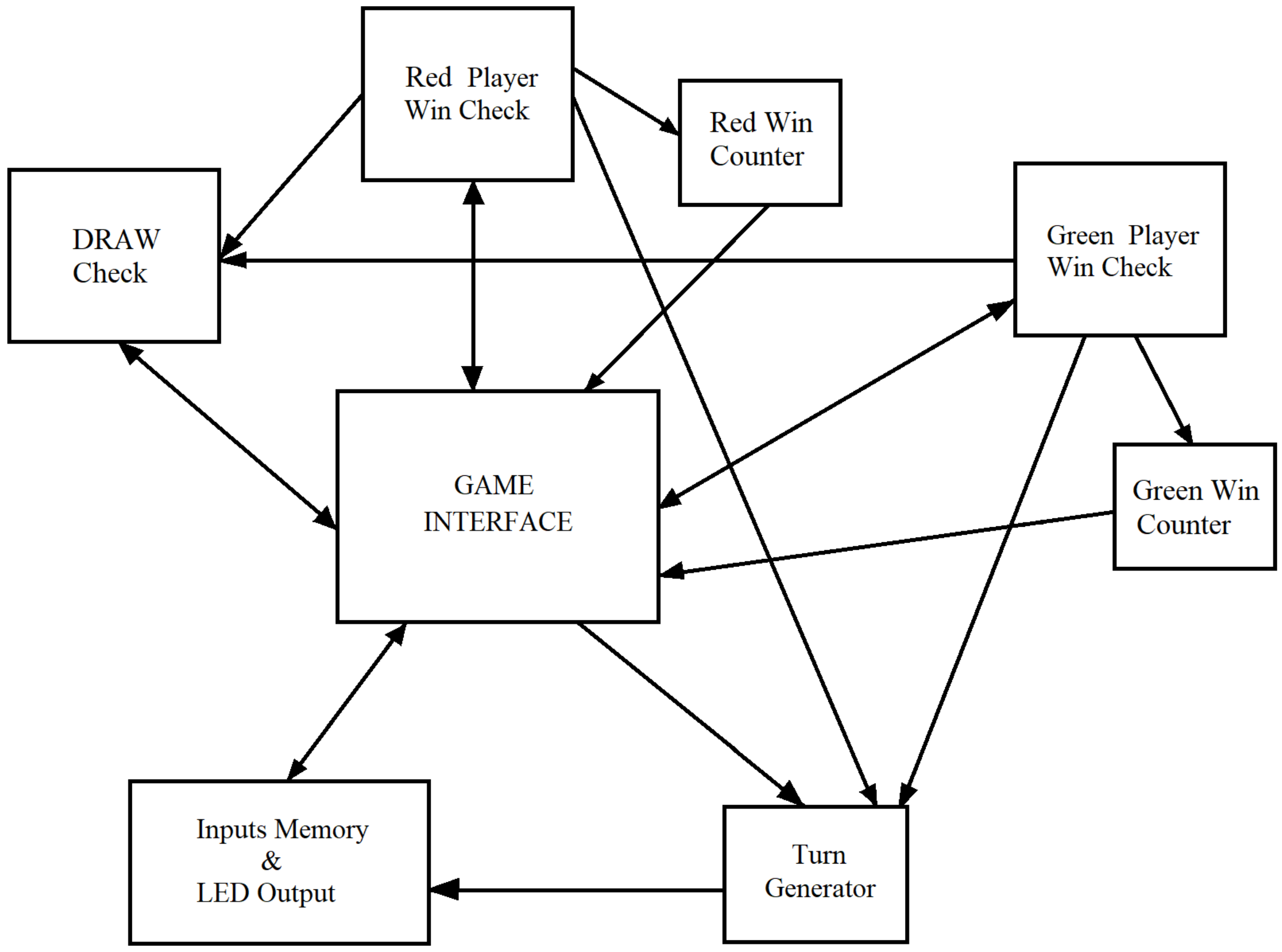
In my adaption of Tic-Tac-Toe, the players will use LED lights, Red and Green, to represent their symbols. Instead of filling a grid, my circuit will uses momentary push buttons arranged in a 3x3 matrix; each pushbutton will represent a vacant cell with two corresponding LED representing its state. I've also decided to include two 7-segment displays to represent the win count of each player. There will be two reset buttons, one for clearing the 3x3 matrix and the other for clearing the win counts. Three LED lights will be used to indicate either a winner or draw. Another two LEDs will be used to determine whose turn it is. The drawing below depicts how the game board should look like.



Theory & Design

Subsections

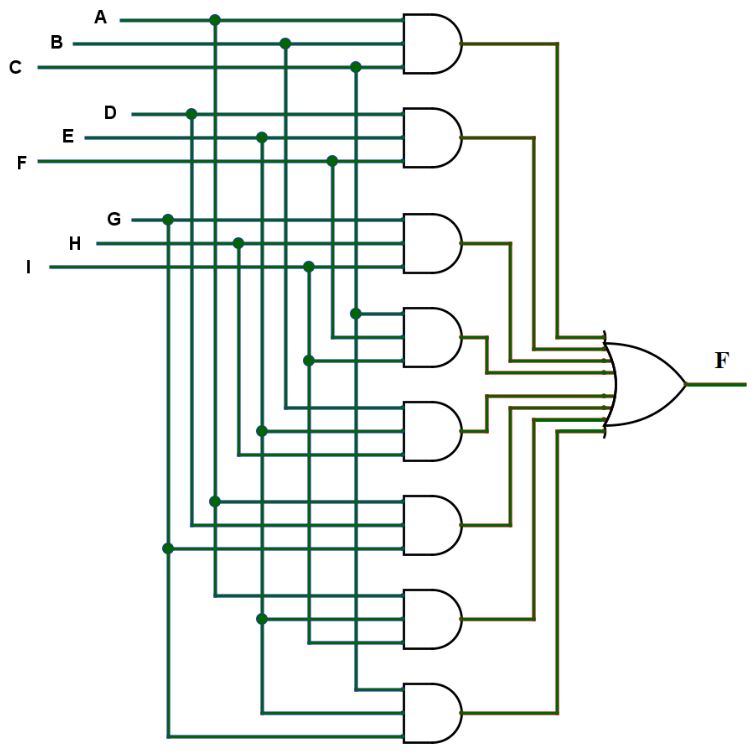
In order to tackle this project, the game needs to be deconstructed into multiple, simpler subsections. For starters, there needs to be a central game interface where everything is displayed and inputs are received from the players. The pushbutton inputs for the 3x3 matrix need to be remembered and displayed in the board. There needs to be logic that controls whose turn it is, whose winning, or if the game is going to draw. There needs to be something that keeps track of the score and the method in which it is displayed. To organize these ideas, I used the following block diagram to split up the individual subsections.



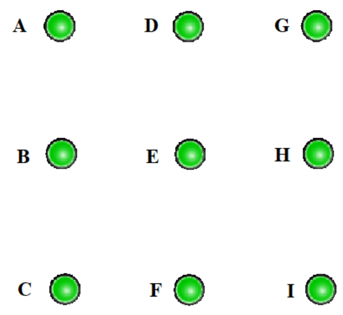
Finding a Winner

The 3x3 push-button matrix uses red and green LED outputs for each of the push-buttons. To determine if a player has won, their needs to be logic that can figure out if either the green LEDs have three in a row, or if the red LEDs made three in a row. Since the green and red LEDs are independently wired, they both can use the same logic circuit that determines if they have won. To find the combinations needed for one of the players to win, I labeled the Green LED matrix cells "A" to "I" and started finding out all the possible combinations that the player could use to win. Once I figured those out, each combination was put through triple-input AND gates whose outputs were sent to a single eight-input OR gate, resulting in a sum of products Boolean expression. The summation of the combinations through the OR gate is necessary because the game requires only one winning combination to be satisfied, resulting in a three in a row (win). Since the labs did not supply us with triple input AND gates or eight input OR gates, they would have to be constructed using the supplied 2-input AND/OR gates. The diagram shows how the labeled matrix was used to find the eight different winning combinations and how they implemented into logic gates. This specific example used green LEDs, but the red LEDs use the same circuit.

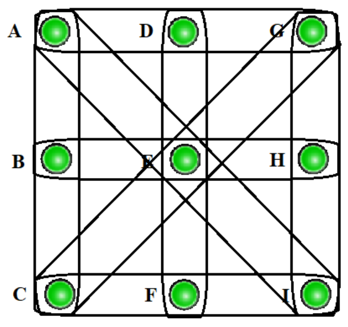
Logic Gate Representation



3x3 Green LED Matrix



Possible Win Configurations:



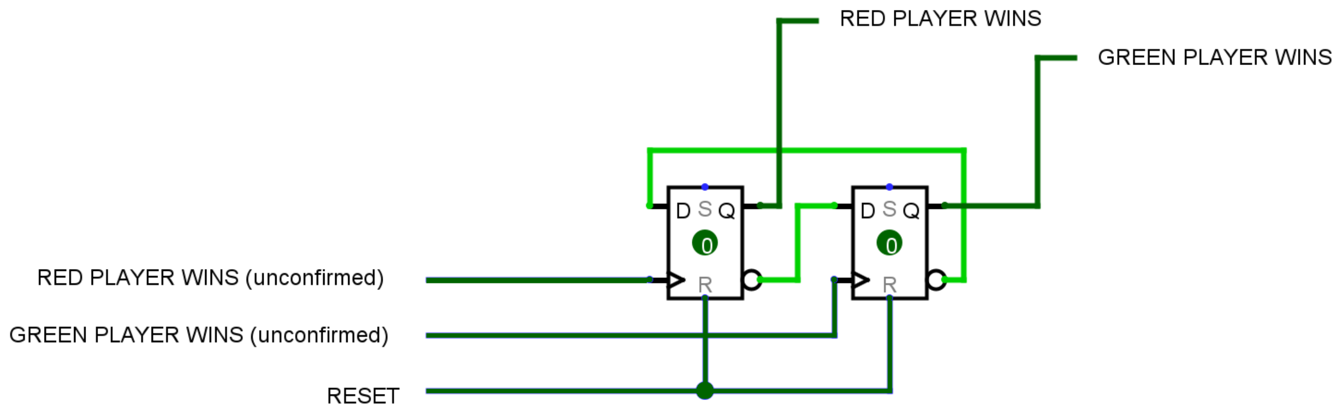
Possible Win Combinations:

- **ABC**
- **DEF**
- **GHI**
- **ADG**
- **BEH**
- **CFI**
- **CEG**
- **AEI**

Boolean Expression:

$F = ABC + DEF + GHI + ADG + BEH + CFI + CEG + AEI$

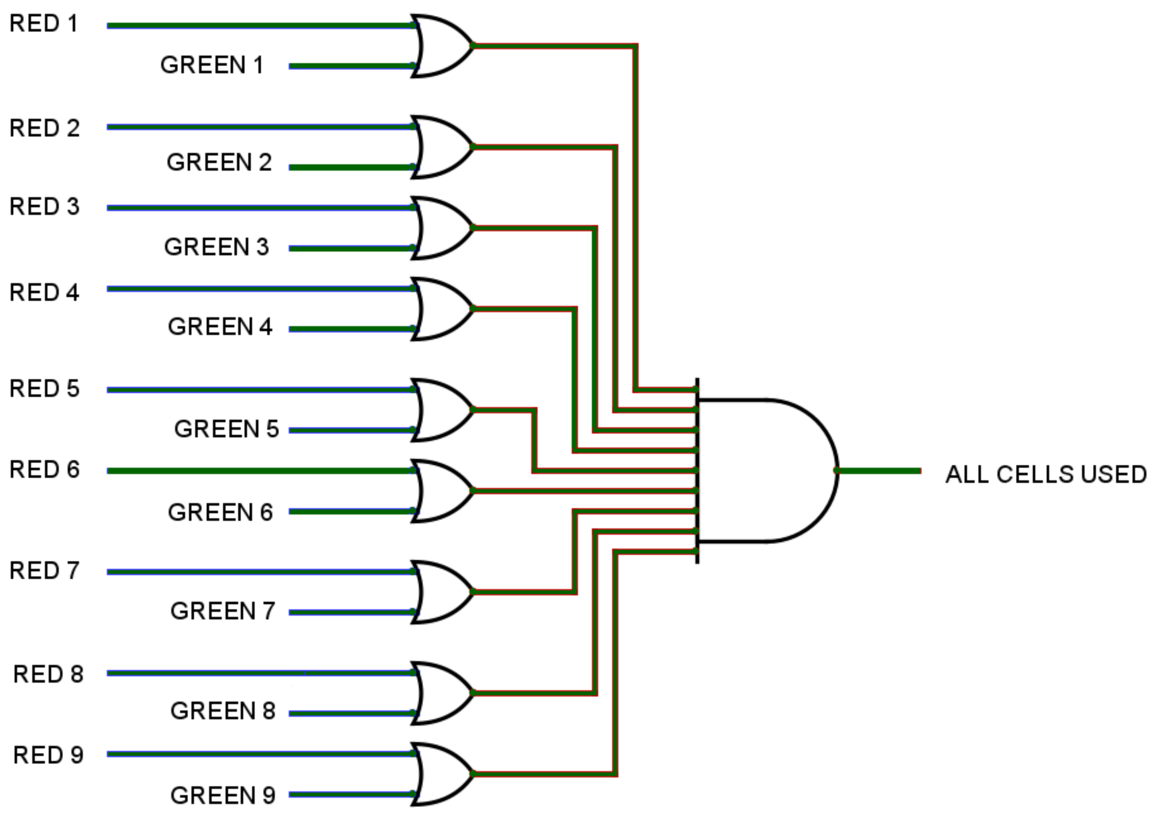
Now that we know when a player or color has won, what's to stop the other player from continuing to play and win as well? Clearly, there needs to be an extra form or security for both the outputs of the winning logic circuits so that if green has already won, then red cannot win later, or vice versa.


RED PLAYER WINS
GREEN PLAYER WINS
RED PLAYER WINS (unconfirmed)
GREEN PLAYER WINS (unconfirmed)
RESET
D S Q
0
R
D S Q
0
R

The circuit above takes the input of one of the winning circuits and remembers it through the use of one of the D-Flip-Flops. The moment its clock is triggered, it locks the other D-Flip-Flop from taking any inputs and changing states. Both flip flops have the ability to lock each other's states so that either red or green player can lock in their win. Once the game is over, the users can clear the state of this circuit by engaging the "Clear Board" button on the main interface; this will be routed to the "RESET" of both D-Flip-Flops so that they are ready for the next winner.

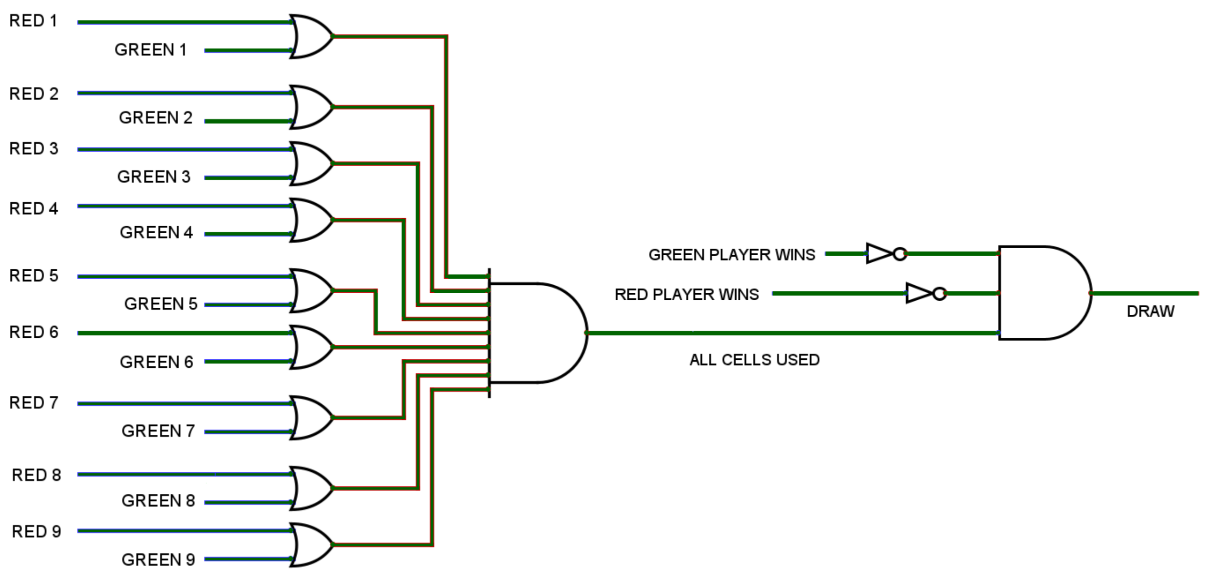
Checking For a DRAW

In Tic-Tac-Toe, if all nine cells of the game board have been used and no winner prevails, the game is a draw. Since each cell in my game is shared by a red and green light, if either of them are on, it would indicate that the cell is used. Therefore, an OR gate can be used to see if either of the LEDs are on. By attaching an OR gate to each pair of LEDs, a total of nine OR gates will be used and all of them have to be triggered for there to be a draw. A nine-input AND gate can be used for the purpose of ensuring that all cells are used and if all OR gates have been triggered. The result is a single output that is HIGH when all cells of the 3x3 matrix are used. The logic circuit below demonstrates this. Note: since the lab does not provide nine-input AND gates, they will be replaced by a series of two-input AND gates.



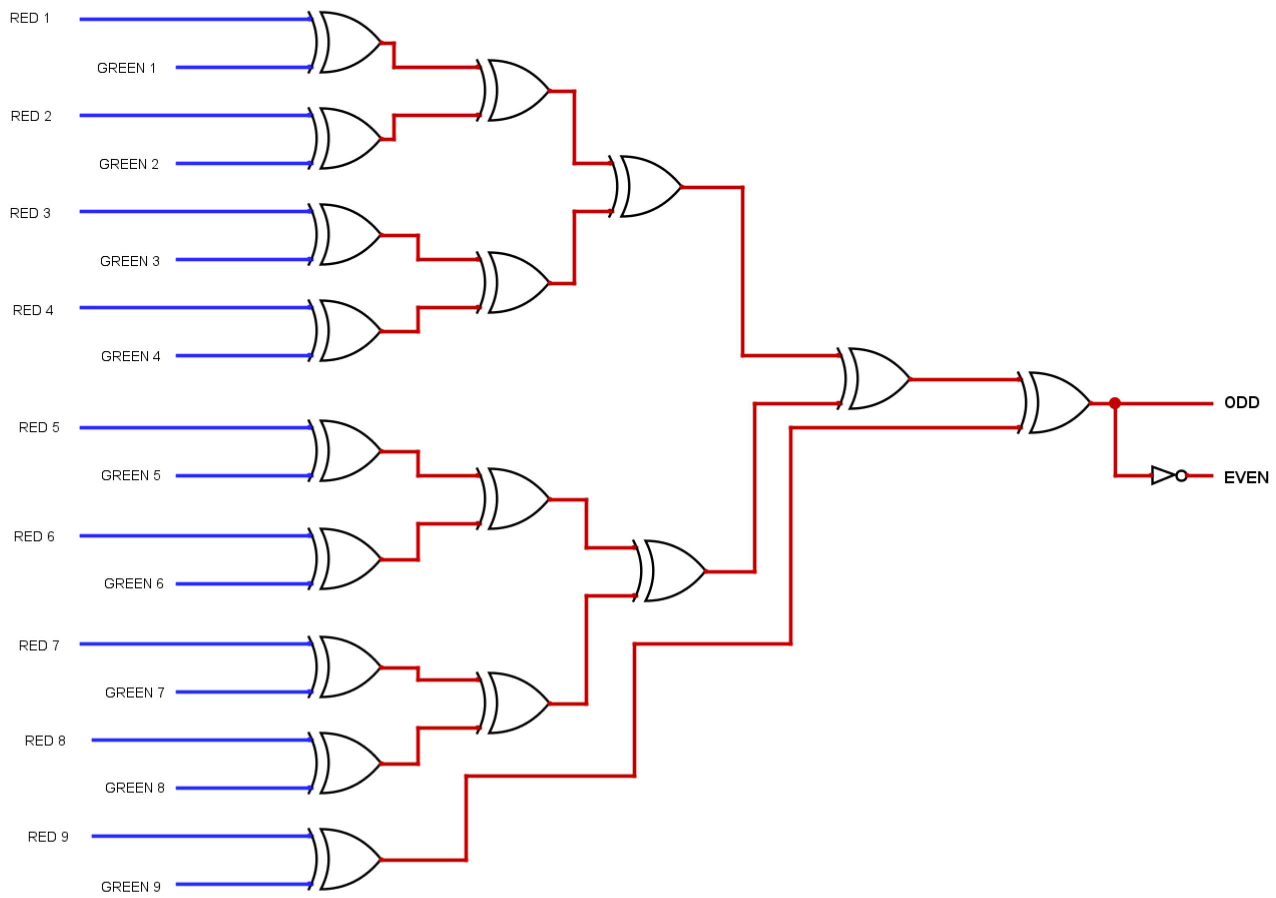
To determine whether the game is actually a draw, there should also be no winners. This means that if either of the winning circuits discussed earlier output a High, then the game is no longer a draw regardless if all cells have been filled. To fix this issue, I used two NOT gates and a three-input AND gate to take the inputs of the green and red "win circuits" along with the "all cells used" input to create a single output that confirms a draw. Below is the implementation of this idea into the final "Check Draw" circuit.

CHECK DRAW

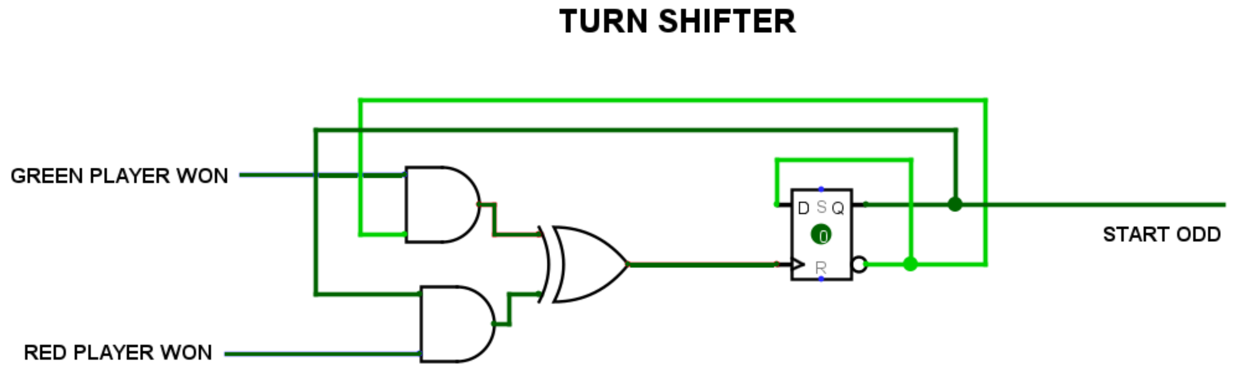


Whose Turn?

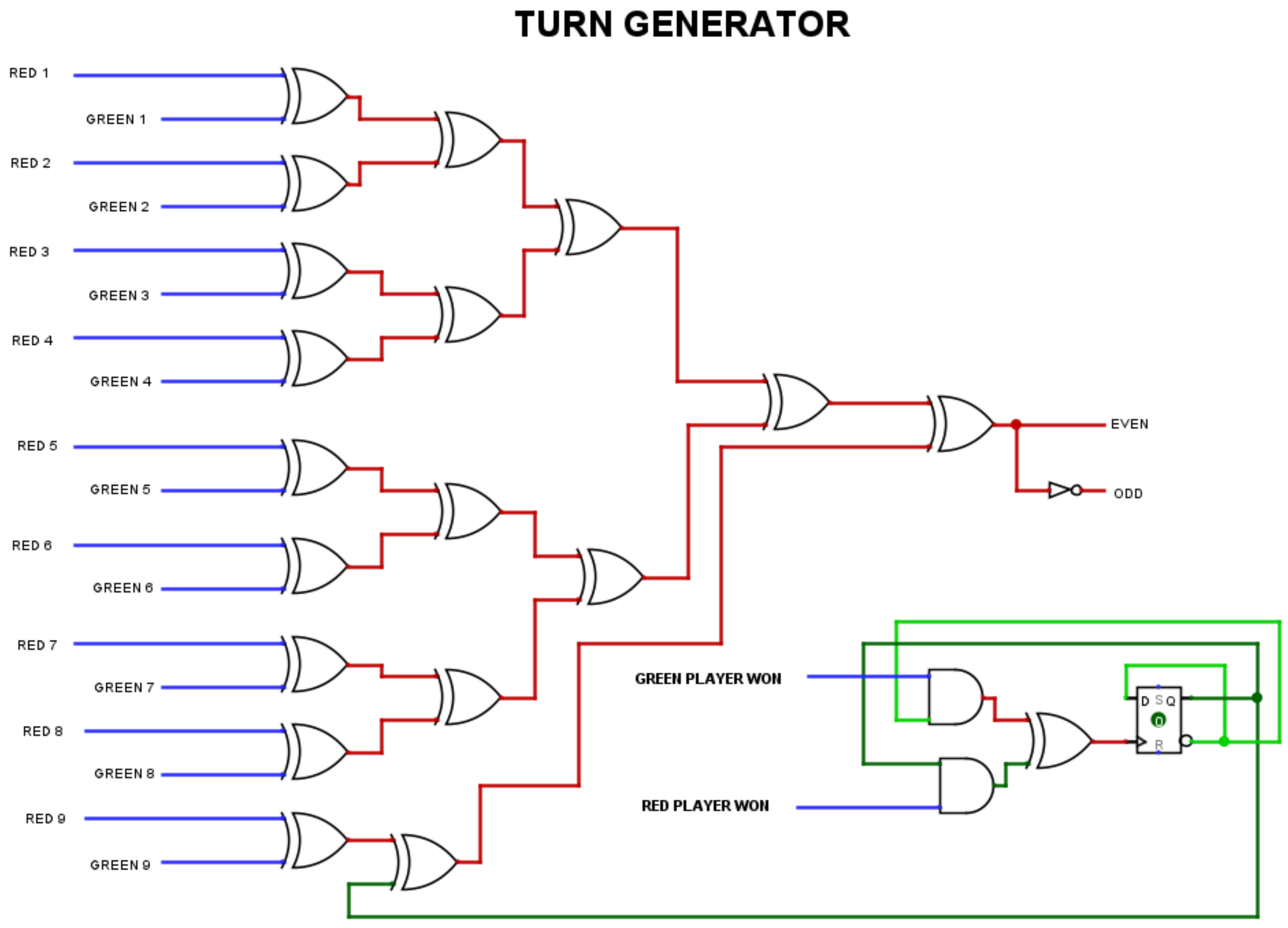
In Tic-Tac-Toe, the first person to pick a cell is arbitrary, so I randomly decided that Red player should always start the game; then the green player goes, then the red, then the green, and the pattern repeats. If I want to know whose turn it is, given that I know who went first, I can just count all the used cells and if the number of used cells is even, it's Green's turn, and if it's odd, it's Red's turn. Using this idea, I implemented the same principle used in parity generators, and created the following circuit that checks how many LEDs are ON (cells that are used), and outputs a HIGH or LOW based on if the number of LEDs that are ON is odd or even. The Output of this circuit is routed to the main game interface where a red LED lights when the "EVEN" output is High, and a green LED lights when the "ODD" output is HIGH; this lets the players know whose turn it is.



In my project description, I mentioned that I wanted to let the person who won the previous game, go first in the next; this makes the game more fair and interesting. Based on my current circuit design, when the board is cleared, the parity generator above will always say that it is Red's turn. If green had won in the previous match, then to let green start first, the parity generator needs to be shifted by one. This extra "one" has to be constantly ON throughout the game or else the game will immediately shift the turn to red. Since indicator that shows who has won is only ON between the time that somebody has won and when the game is reset, it has to be treated like a momentary indicator. Thus, sequential logic will be utilized (D-FlipFlop) along with a few gates that determine whose turn it should be based who has won and who has won before. The following logic diagram is what I came up with:

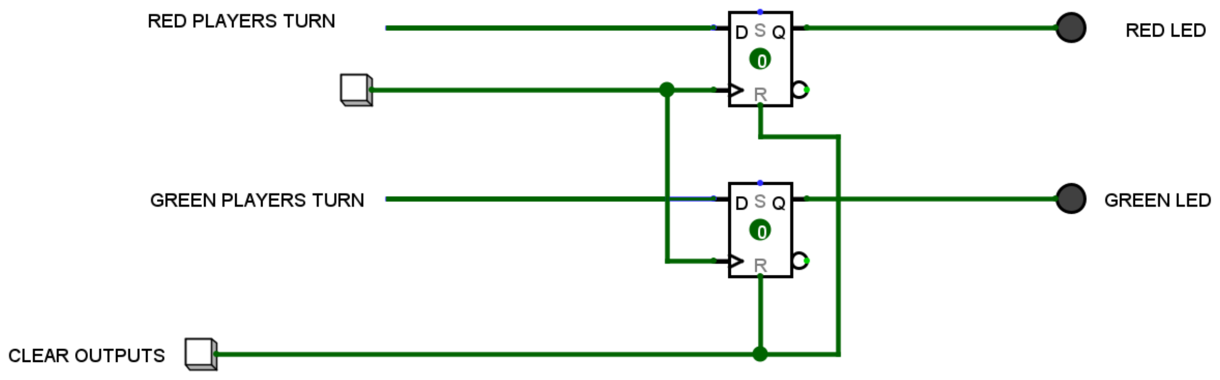


Normally, the "Start Odd" output is LOW, but if the green player wins, it becomes HIGH; it will remain in that position until Red wins. The output of this circuit is wired back to the parity generator through the use of a XOR gate whose second input is wired to the output of one of the other XOR gates. The final "Turn Generator" with the turn shifter attached is shown below.



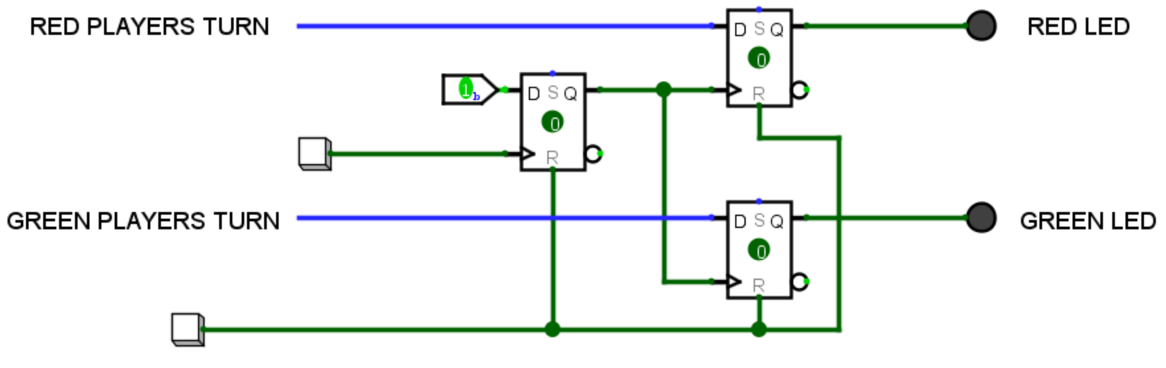
Memory Storage & Outputs

All of the logic circuits discussed earlier rely on the steady states of the LEDs on the 3x3 matrix. Although there are two LEDs (green and red) in each cell of the matrix, only one should be ON and it should remain ON until the game ends and the board is cleared. When a push button is triggered, either the green or red LED from the button's cell should light, and should remain lightened which hints at the need for sequential logic (D-FlipFlops). Which LED should light is based on whose turn it is, thus the necessity of the "Turn Generator" that was found before. This idea is shown below using two D-FlipFlops.



The problem with this circuit is that anyone can claim a cell that was already claimed before by the other player. This is not allowed in this game because once a cell has been chosen by a player, it remains in their possession until the game ends and the boards is cleared. To fix this issue, an extra D-Flip Flop is added between the push button and the clock inputs of the other two D-FlipFlops so that if the push button is triggered once, its output is locked and held constant so that the states of the other Two Flip Flops are also jammed until the "Clear Outputs" button is triggered when the game ends or when the players need to clear the board. The following logic circuit will be duplicated for each cell of the matrix, and all the inputs, besides the push buttons for each cell, will be tied together; the "Clear Outputs" will be coupled to a single pushbutton in the game interface labeled "Clear Board".

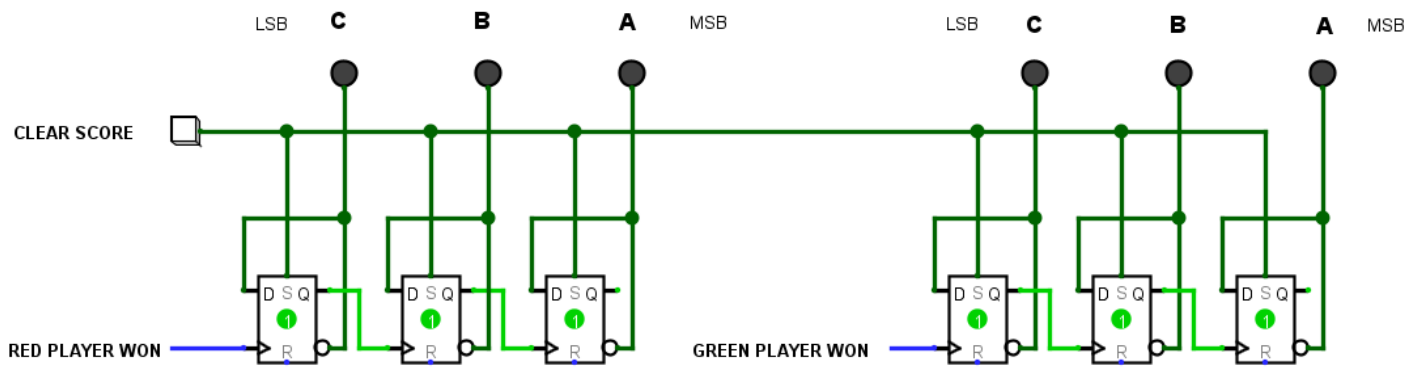
MEMORY OUTPUT



Score Display

Naturally, Tic-Tac-Toe is really quick game so players may want to play a number of matches. Each match will have a winner unless it is a draw, in which case nobody wins. To keep track of the wins of each player, I will be using two 7-Segment displays along with a single “score reset” button that clears the scores if need be.

First, there needs to be something that can keep track of the score in binary so that the binary data can be sent a decoder which sends its output to the 7-Segment display. In the last project, I created a simple counter that outputted a binary value based on a clock input. Instead of a clock input, I could wire the momentary output of the winner circuit of each player to their each individual counter. I will be using three D-FlipFlops for each counter, which means a maximum binary value of 111, the decimal equivalent of 7; an extra D-FlipFlop would bump the decimal value to 15, but the 7-Seg display can only output up to 9, so it’s not worth the extra FlipFlop. The counters for each player are shown below.

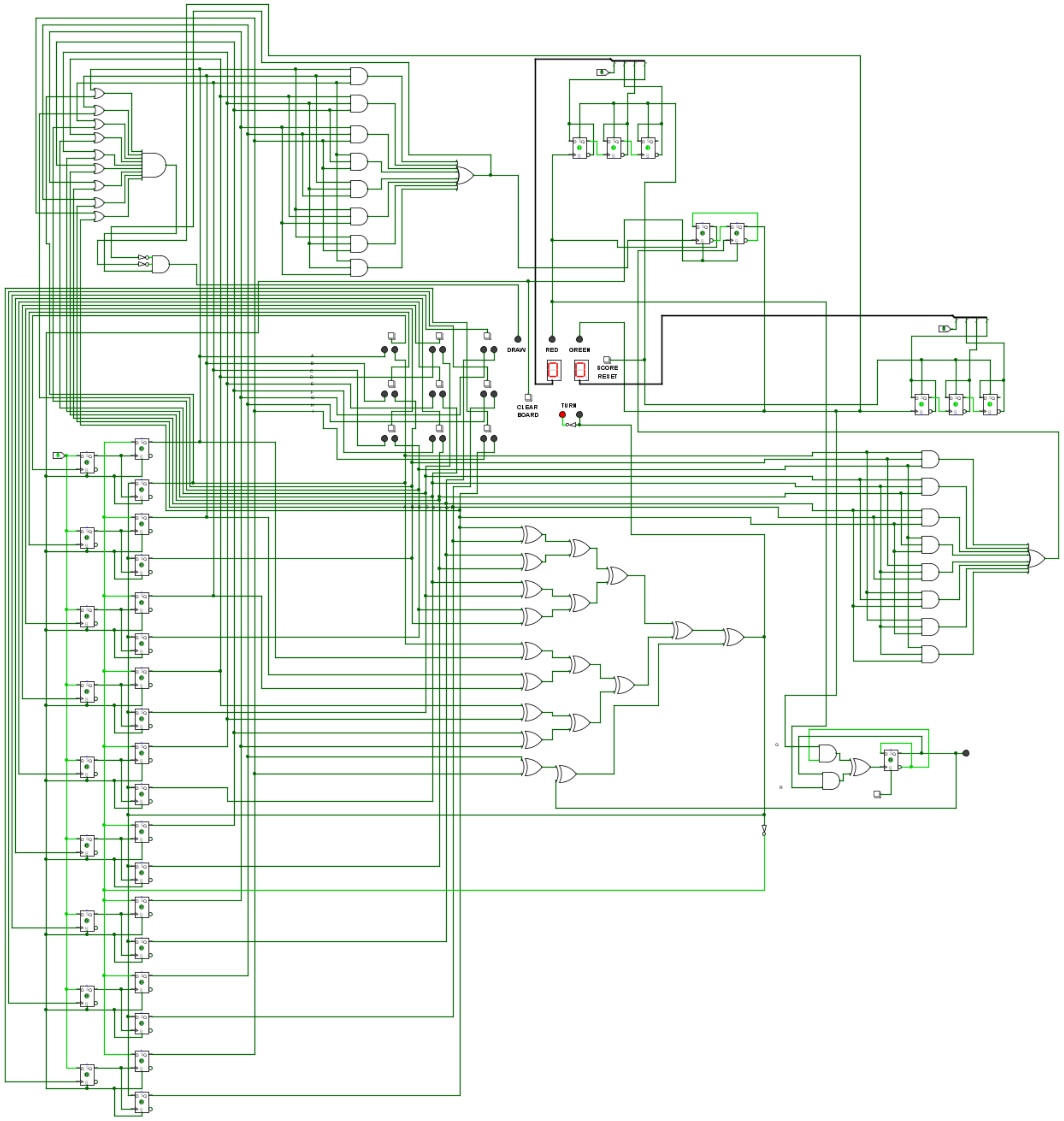


The outputs (A-C) will be sent to a decoder that will take counter’s inputs and transform them into the signal needed for the 7-Segment display to work. Here is a truth table of how the Decoder interprets the inputs of the counter and the corresponding outputs to the display.

NOTE: 4-bit Decoders require four binary inputs. The counter in my game outputs only 3-Bits, so the the fourth input of the decoder will need to be grounded, Hence the reason why the MSB for the Input is "0".

Decimal Digit	Input lines				Output lines							Display pattern
	0	A	B	C	a	b	c	d	e	f	g	
0	0	0	0	0	1	1	1	1	1	1	0	0
1	0	0	0	1	0	1	1	0	0	0	0	1
2	0	0	1	0	1	1	0	1	1	0	1	2
3	0	0	1	1	1	1	1	1	0	0	1	3
4	0	1	0	0	0	1	1	0	0	1	1	4
5	0	1	0	1	1	0	1	1	0	1	1	5
6	0	1	1	0	1	0	1	1	1	1	1	6
7	0	1	1	1	1	1	1	0	0	0	0	7

TIC-TAC-TOE FINAL ASSEMBLY ON LOGISIM



Conclusion

Constructing Tic-Tac-Toe required a new way of thinking about a problem since it required breaking up the project into multiple independent parts. Many of the parts depended on each other so if I hadn’t broken the game apart into smaller manageable parts, it would have been really difficult knowing where to start. Tackling each problem one at a time allowed me to put everything together like a puzzle and it was rewarding when it all worked out in the logisim simulator. All that’s left to do is to put it together in Tinkerkad using real ICs and see if it works for real. My only concern is how the game interface will look since everything needs to be staggered when placed on a breadboard; it may look nothing like Tic-Tac-Toe.

Looking back, this project was quite daunting at first because there were so many possibilities and I wasn’t sure if the simple logic design skills I had learned would be enough to build a complicated game. I didn’t want to make a very simple game because that would be cheating on my own abilities and I didn’t want to attempt a really complicated game because I may not have all the ideas or components needed to build it. Tic-Tac-Toe seemed like the perfect balance between complicated and simple. Although the project didn’t require timers, I believe constructing this game covered most of the ideas discussed in the class, from understanding truth tables to using decoders and counters.