Digital Logic Design X-O Game

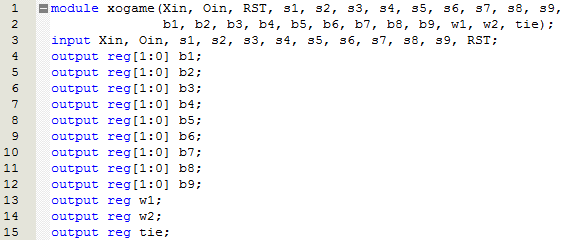
By: Muhammad Ansari

Introduction:

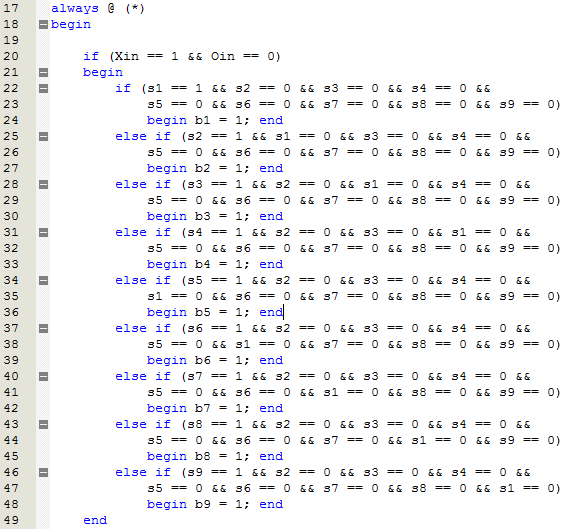
As a computer science student, I chose this topic looking forward to create a famous game by utilizing my knowledge of programming to implement it into digital logic design. Since games like tic-tac-toe are commonly created using software on computers, I wanted to take on the challenge of implementing it into functional Verilog code as well. I had recently created a functional tic-tac-toe game in the Java programming language for a class, so I felt that this will be a new experience for me as I try to re-create my work from computer science into the realm of digital logic design.

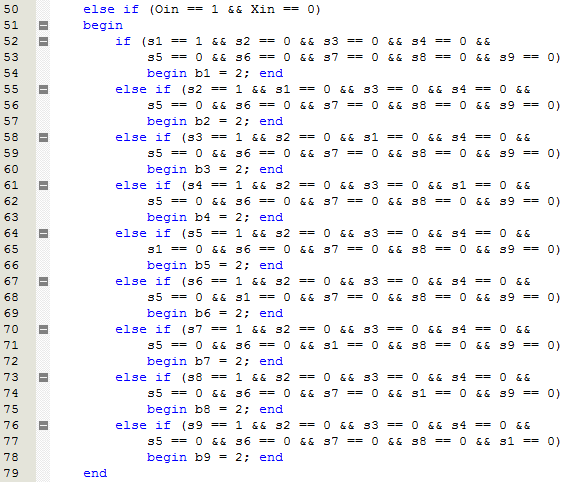
The objective of the game is like that of a typical tic-tac-toe set up. I have designed the code so that two players play the game and take turns inputting their values into the game board. Players will enter either ‘X’ or ‘O’ in alternating fashion onto the game board. When a player successfully places three adjacent ‘X’s or ‘O’s in a horizontal, vertical, or diagonal line, the player wins. If neither of these conditions is met before the game board is completely filled, the game ends in a tie. A new game can be started immediately at any time using the reset function we implemented into the code.

Code:

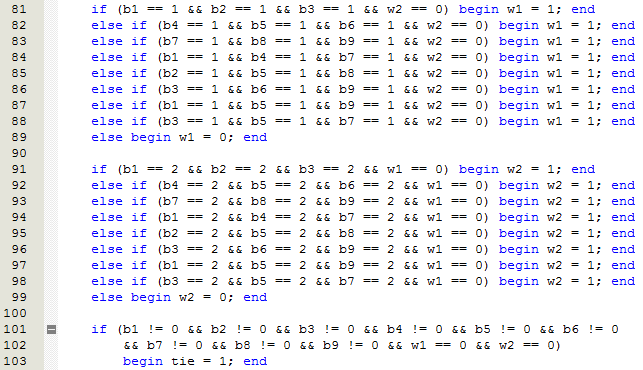


At the start of the code, we declare all of our inputs and outputs. We create nine inputs and nine corresponding outputs which represent the nine blocks on the tic-tac-toe board. They are numbered ‘s1’ to ‘s9’, and ‘b1’ to ‘b9’, with 1 representing the top-left most block on the board, and 9 representing the bottom-right most block. The ‘s1’ to ‘s9’ inputs will use switches to determine which block the player will place their input. The ‘b1’ to ‘b9’ outputs are two-bit outputs with each state representing the current status of each block. When the outputs are in the 00 default state, they represent a blank space. When they are in the 01 state they represent an ‘X’ block and when they are in the 10 state they represent a ‘O’ block. The 11 state is unachievable and unnecessary and can be ignored. The ‘Xin’ and ‘Oin’ inputs are used to determine whether the player will input an ‘X’ or ‘O’. ‘RST’ is the input which resets the entire game. The ‘w1’, ‘w2’, and ‘tie’ outputs are used to display whether the game ended in a player-one win, player-two win, or in a tie respectively.

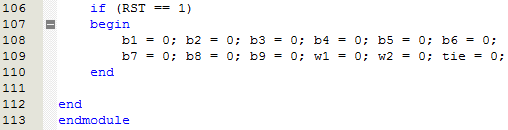




The main functionality of the code is wrapped inside an “always@(\*)” so that the hardware knows to continuously check for inputs from the players at all times. This first block of code is for inputting values into the board. The code first checks whether the player will input an ‘X’ or an ‘O’ and then checks the block in which the player would like to place their input. The if-statements are carefully written to include every instance of input and exclude any possibility of the player attempting to place multiple inputs at a time. When an ‘X’ is placed on the board, the corresponding block’s value is change to 1 in the output. When an ‘O’ is placed on the board, the corresponding block’s value is change to 2 in the output. These outputs will be used to determine the winner in the next block of code.

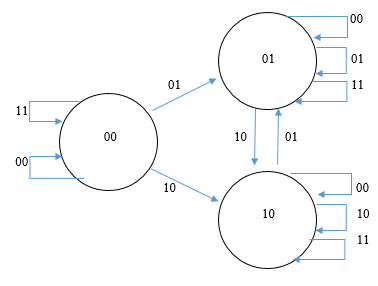


This block of code is used to check after each input whether the game has been won or not. After each input, the code goes through a series of if-statements, each of which describes a win condition for either player-one or player-two. The first three if-statements in each block check the horizontal rows on the game board from top to bottom. The next three if-statements check the three columns on the game board from left to right. The last two if-statements are for checking the diagonals on the game board. When a series of ‘X’s or ‘O’s are created in any of these patterns, the ‘w1’ or ‘w2’ output will display a value of 1, representing a player-one or player-two win respectively. Finally, if all of the blocks on the game board are filled, and no winner has been determined by that point, the ‘tie’ output will become 1, signifying a game that has ended in a tie with no winner.



This last piece of code checks whether the players have inputted the RST. When this is true, the hardware resets all outputs, including the blocks on the game board and all win conditions to 0, essentially starting a new game. The program then finally checks for the next set of inputs.

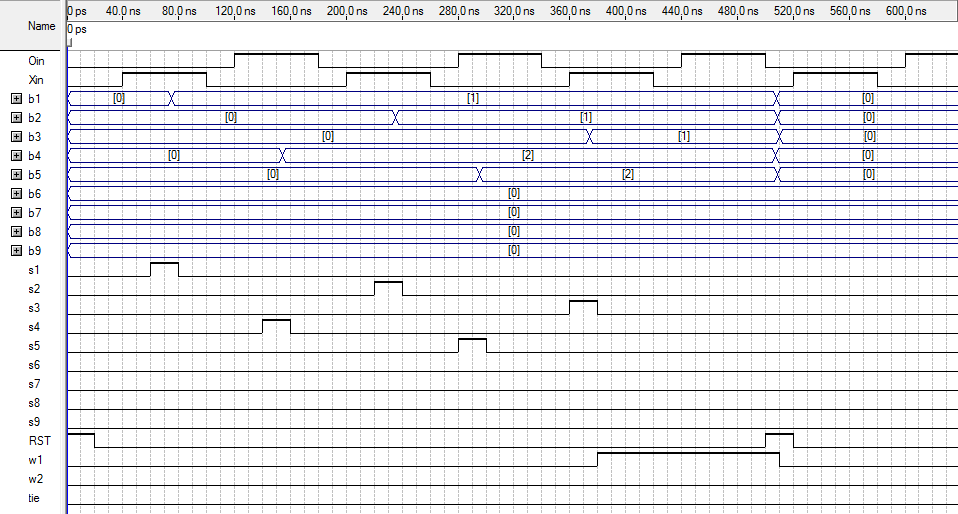
State Diagram:



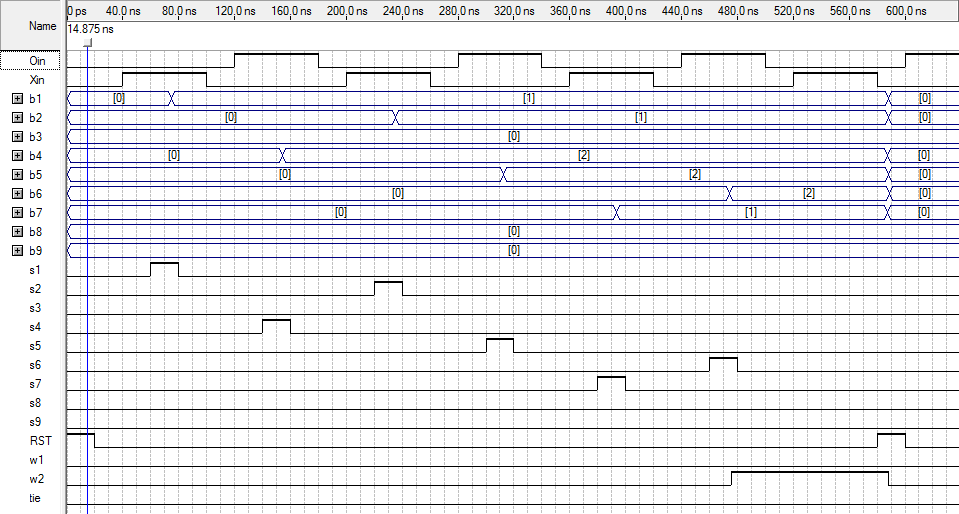
This image shows a state-diagram for only one block on the entire tic-tac-toe game board. Showing all the outputs that one block can achieve through player inputs. I did not show the diagram for all nine blocks together since there are 9 possible outcomes for the entire game board.

Simulation Waveform Results:

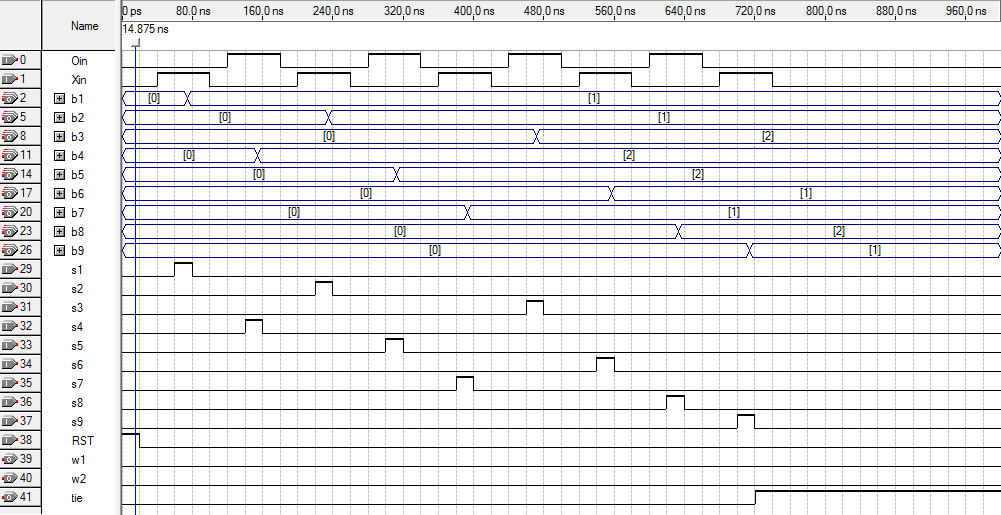
Player-One Win:



Player-Two Win:



Tied Game:



Conclusion:

In conclusion, a fully functional tic-tac-toe game is possible to create through hardware description programming. The code allows the game to run seamlessly, imitating a real-life tic-tac-toe game almost perfectly, giving both players a fun and fair experience playing tic-tac-toe digitally. With hardware capable of reading enough inputs from users and displaying enough outputs necessary for the creation of the tic-tac-toe game board, we can most certainly create a perfectly efficient X-O game. Not only does this apply to tic-tac-toe, but countless other real-world applications that can also be implemented into digital logic design using hardware and hardware programming.