TIC TAC TOE GAME

Alberico Ivan, Pelle Silvana, Tongji University

Abstract — **It is usually common to use logic circuits when it is up to design games, whether it is for hobby or for didactic purposes. For our project, we decided to design a model of Tic Tac Toe game using digital logic components in Proteus, an electronic circuit design software. Tic Tac Toe is a very simple and immediate game, very popular in Western countries, especially among young people when you want to let loose for a while with some friends. There could have been several ways to design the game, however the path we chose seemed to us to be the most accurate in satisfying the characteristics of the game in the most practical and intuitive way. We designed the game using basics logical elements, without relying on microprocessors: the main goal was to put into practice the knowledge learned in class. The whole project is mostly based on sequential logic circuits, especially on the use of flip-flops as elements to store the different states of the game.**

Index Terms— **7-SEGMENT DISPLAY, COUNTERS, FLIP-FLOPS, K-MAP, LEDs, LOGIC GATES, PUSH BUTTONS, PROTEUS SOFTWARE, SEQUENTIAL CIRCUIT, TRUTH TABLE.**

I. Introduction

Tic Tac Toe is one of the simplest yet most-challenging games to be invented. Two players are allowed to play at a time, traditionally one player plays as "X" and the other plays as "O". One player starts by selecting the position of a 3x3 grid, then hands the turn to the next player. This goes on until one player gets three marks in a row (horizontally, vertically or diagonally), or the board fills with no winner resulting in a draw.

Tic Tac Toe is a widely known paper-and-pencil game, and while simple to play, it required not a very immediate and easy procedure to get to the final circuit: many components were used to build the game using only digital logic components.

The whole circuit is composed by 9 LEDs which represent the 9 boxes of the 3x3 grid. Push buttons are used to switch on the LEDs and each one of them represents the moves of both two players: to do that we used bi-color LEDs to represent the two players' moves for each box of the grid, instead of using two different single color LEDs.

The final result is shown by other two LEDs, one for win and one for draw. We also designed a stopwatch that displays the duration of each match.

Inputs given by switches are recognized as either first or second player's move according to the color of the LEDs (red denotes Player 1 and green denotes Player 2) and they are stored in flip-flops. LEDs are connected to the outputs of the flip-flops and winning or drawing condition are checked from there.

Some requirements need to be taken into account to properly develop the game:

- A player should not be able to change other player's move. For example if Player 1 selects the first cell, Player 2 should not be able to select the same cell again.
- Players should have the ability to reset the game at any point through a reset button.
- Once a player wins, the "result" state should be locked. For example, if Player 1 wins first and then Player2 makes a move and satisfies the wining criterion, the output should show only "Player 1" as the winner.

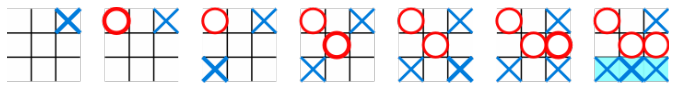


Fig. 1. Example of a match won by player X.

DATE OF SUBSMISSION: 2019/01/05

Ivan Alberico was with University of Bologna Alma Mater Studiorum, Bologna, BO 40138 Italy. He is now with Tongji University, Shanghai, China. (email: ivanalberico1998@gmail.com, telephone number: +86 18621132579).

Silvana Pelle was with University of Bologna Alma Mater Studiorum, Bologna, BO 40138, Italy. She is now with Tongji University, Shanghai, China. (email: silvana080199@gmail.com, telephone number: +86 18621132687).

II. ABBREVIATIONS AND ACRONYMS

- LED – Light-emitting diode
- IC type – Integrated circuit type
- XOR gate – Exclusive or gate
- K-map – Karnaugh map

III. CIRCUIT DESIGN

The whole circuit design is shown in Fig. 2. We are now going to analyze each part in detail, explaining how we got to the final configuration and which were the steps we followed.
We divided the project into five parts: button configuration, players' configuration, winning conditions, draw conditions and stopwatch.

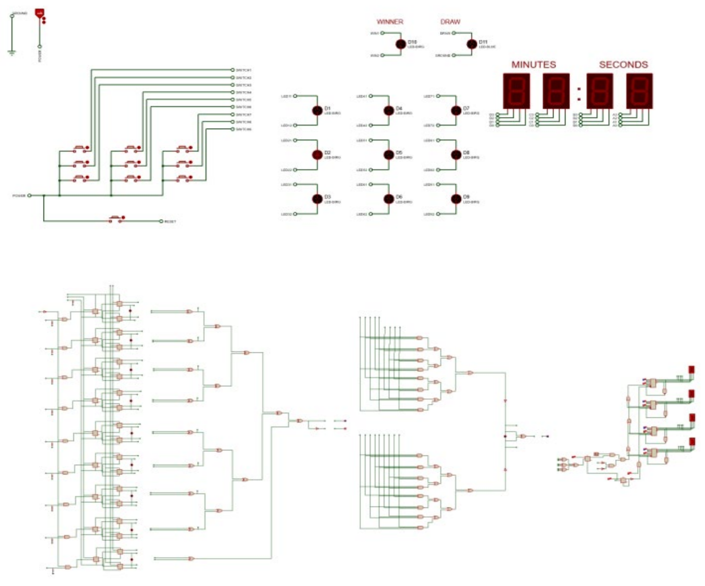


Fig. 2. The whole circuit design in Proteus software.

[1] Since Proteus takes into account the electrical characteristics of the circuit, resistances must be used in order to light up the LEDs. The values of the resistances depend on the color of the LED: usually with 9V batteries resistances of 220 Ω are required for the red, yellow or green color.

*A. **Button configuration***

The first thing we designed was the button configuration of the LEDs. To design this part we firstly had to make sure that each time we pressed a button, the corresponding LEDs lighted up alternatively red or green. Furthermore, once a button had been pressed, it could not have been possible to change the state of the LED anymore even by pressing the buttons several times when it was another player's turn, as stated above in the *Introduction* paragraph.

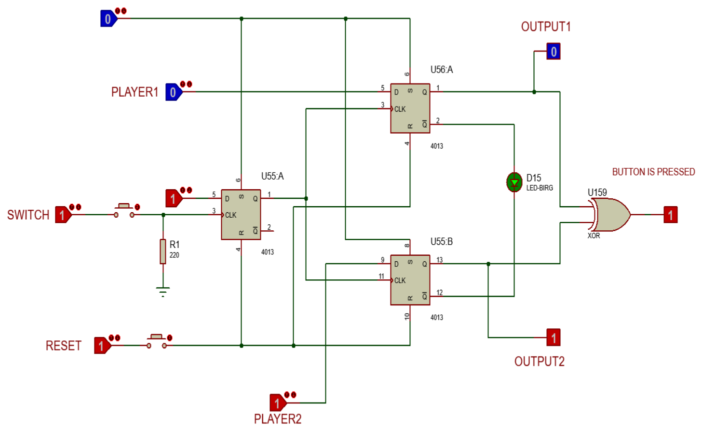


Fig. 3. Button configuration

In general, the main problem of push buttons is that they loose the information instantly as soon as you remove your finger from them. Therefore we needed a mechanism which allowed us to store the memory of the button. For this purpose we used a D flip-flop combination as shown in Fig.3.

The input of the first flip-flop on the left side is connected to a constant high value: when we press the button, the clock input goes from 0 to 1 giving the flip-flop its positive trigger. In this way the output goes from 0 to 1 triggering also the other two flip-flops. The two flip-flop on the right side have as inputs two variables PLAYER1 and PLAYER2 which represent the turns of the players during the match: when PLAYER1=1 then PLAYER2=0 and vice versa (in the first case it means that it is up to the first player to make the move). When the two flip-flops get their positive trigger, they pass their inputs to the outputs and they light up the LED. The direct outputs of the flip-flops are connected to a XOR gate, while their opposites are connected to the bi-color LED which will light up green if PLAYER1=0 and PLAYER2=1, red otherwise, according to the table in Fig. 4 that shows the behavior of a bi-colored LED.

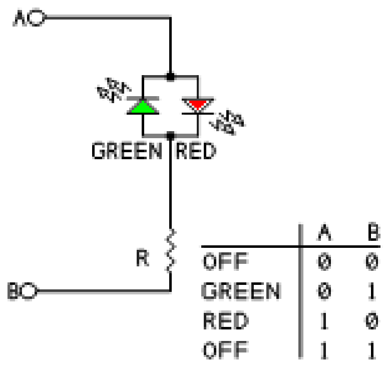


Fig. 4. How a bi-color LED works.

With this configuration no matter how many times we push the button again, the LED will not change its color, even if you do it while is another player's turn. This because the two flip-flops on the left side can have their positive trigger only once, since the output of the first flip-flop cannot go anymore from 0 to 1 due to its constant input 1.

Let us consider a simplified representation, as shown in Fig. 5.

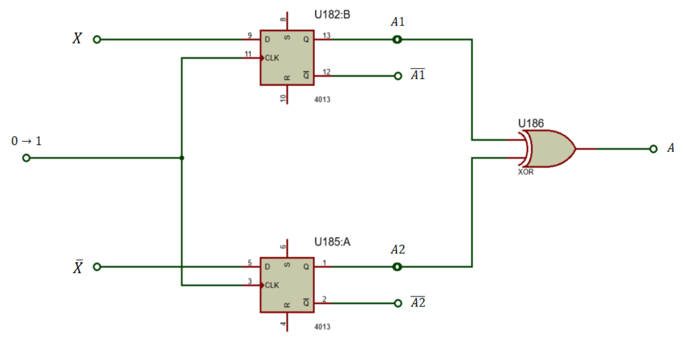


Fig. 5. Schematic representation of the button configuration.

Since we know that PLAYER1 and PLAYER2 are one the opposite of the other, we can define a new variable X such that:

$$X = PLAYER1$$

$$\bar{X} = PLAYER2$$

We know that the state equation for a D flip-flop is

$$Q(t+1) = D$$

so in this case for the two flip-flops we have that

$$A1(t+1) = X$$
$$A2(t+1) = \bar{X}$$

From the state table in Fig. 5 we can notice that the output A is always equal to 1. We know that the output of a XOR gate is equal to 0 if the inputs have the same value, while is equal to 1 if the inputs have opposite values.

The fact that A is always equal to 1 means that a button has been pressed, no matter if it was player1's turn or player2's turn.

Present State		Input	Next State		Output
A1	A2	X	A1	A2	A
0	0	0	0	1	1
0	0	1	1	0	1
0	1	0	0	1	1
0	1	1	1	0	1
1	0	0	0	1	1
1	0	1	1	0	1
1	1	0	0	1	1
1	1	1	1	0	1

Fig. 5. State table of the button configuration

From the bubble diagram in Fig. 6, we can notice how all the states go either in the state 01 or in 10. These are the only two possible states in a match because it is not possible that it is both player's turn at the same time or that it is no one's turn.

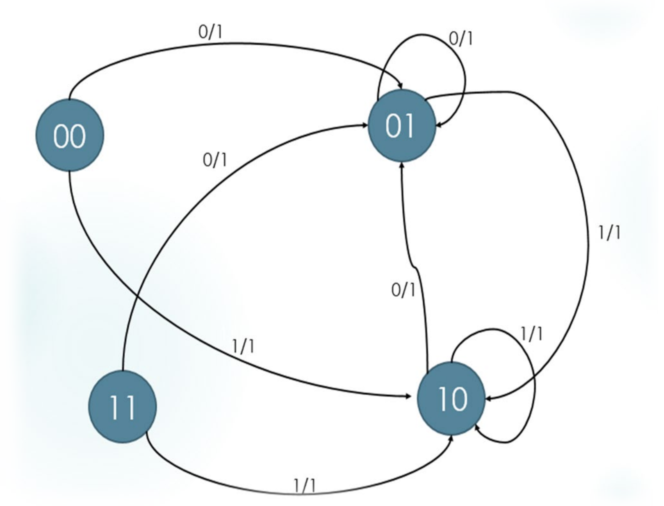


Fig. 6. Bubble diagram of the button configuration.

B. *Players' configuration*

When the game starts, the first move is up to Player 1. The two variables PLAYER1 and PLAYER2 are the inputs of the two flip-flops of each button configuration, as we have seen before.

However when PLAYER1 variable is high, PLAYER2 is low and vice versa. Each time a button is pressed, there is a switch between the values of these two variables: this means that when a player makes a move, then it is up to the other player to make the next step.

Since we have 9 buttons, we should make a 9 variable K-map to find the function that regulates the process. In order to simplify the whole thing, we can also consider less variables, since the logic is the same even if we consider a different number of buttons in the game. For example, let us suppose we have only four buttons: hence, we consider a 4-variable K-map with the relative Truth Table as in Fig. 7 and Fig. 8.

A	B	C	D	PLAYER2	PLAYER1
0	0	0	0	0	1
0	0	0	1	1	0
0	0	1	0	1	0
0	0	1	1	0	1
0	1	0	0	1	0
0	1	0	1	0	1
0	1	1	0	0	1
0	1	1	1	1	0
1	0	0	0	1	0
1	0	0	1	0	1
1	0	1	0	0	1
1	0	1	1	1	0
1	1	0	0	0	1
1	1	0	1	1	0
1	1	1	0	1	0
1	1	1	1	0	1

Fig. 7. Truth Table with only 4 inputs.

AB \ CD	00	01	11	10
00		1		1
01	1		1	
11		1		1
10	1		1	

Fig. 8. 4-variable K-map.

From the K-map we can simply the function as in the following:

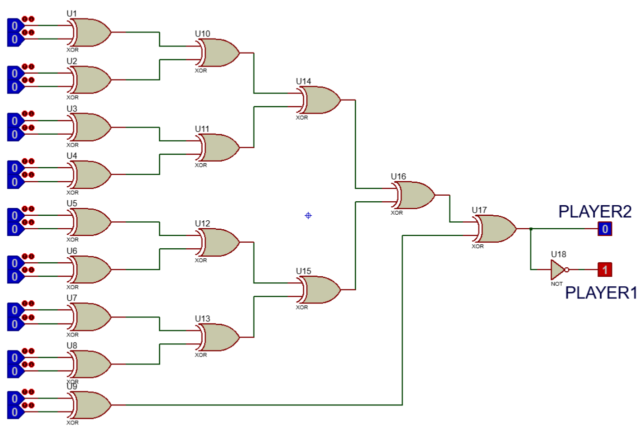
$$\begin{aligned}\text{PLAYER 2} &= \bar{A}\bar{B}\bar{C}D + \bar{A}\bar{B}C\bar{D} + \bar{A}B\bar{C}\bar{D} + \bar{A}BCD + AB\bar{C}D + ABC\bar{D} + A\bar{B}\bar{C}\bar{D} + A\bar{B}CD \\ &= \bar{A}\bar{B}(\bar{C}D + C\bar{D}) + \bar{A}B(CD + \bar{C}\bar{D}) + AB(\bar{C}D + C\bar{D}) + A\bar{B}(\bar{C}\bar{D} + CD) \\ &= (C \oplus D)(\overline{A \oplus B}) + (A \oplus B)(\overline{C \oplus D}) \\ &= (A \oplus B) \oplus (C \oplus D) \\ &= A \oplus B \oplus C \oplus D\end{aligned}$$

Extending the result to all the 9 inputs we have:

$$\text{PLAYER 2} = A \oplus B \oplus C \oplus D \oplus E \oplus F \oplus G \oplus H \oplus I$$

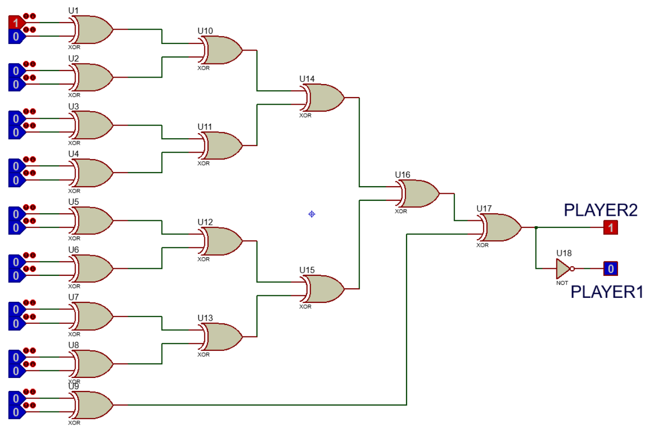
This also means that if any odd number of push buttons is pressed, the output of the final XOR gate will be 1, while for even number of push buttons pressed the output will be 0.

The Proteus simulation is shown in Fig. 9(a) and Fig. 9(b).



(a)

[2] The function represent a XOR gate with 9 inputs. In real life, as in Proteus, we do not have 9-inputs XOR gates, so the function is a combination of 2-inputs XOR gates in parallel.



(b)

Fig. 9. (a) It shows the players configuration when the game has just begun. We can see that in this case no button has been pressed and variable PLAYER1=1, so it is up to the first player to make the move. (b) When we push a button we can see that variables PLAYER1 and PLAYER2 are switched in values, and now it is up to the second player to make the next move.

C. *Winning conditions*

In Tic Tac Toe game a winner is found if there is player who succeeds in placing three of their marks in a horizontal, vertical, or diagonal row before the other player, therefore if one of the eight combinations shown in Fig. 10 is verified.

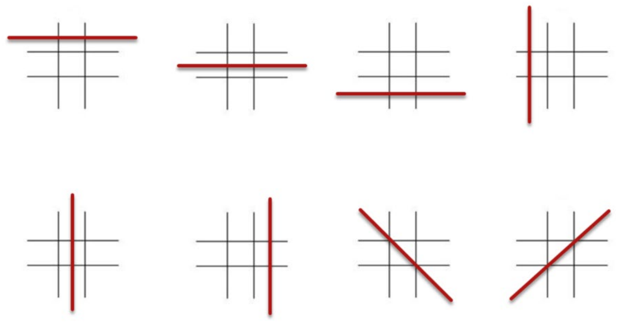


Fig. 10. Winning cases in Tic Tac Toe game.

Referring to the values in Fig. 11, we can say that in terms of variables there is a winner as soon as one of the following values becomes high: ABC, DEF, GHI, ADG, BEH, CFI, AEI and CEG. Therefore, to find the function that regulates the winning condition we connect the values of each triplet to 3-input AND gates that become high when all the three values connected are high. Then we connect all these eight gates to an OR gate, because a winner is found when at least one of the previous gates is high.

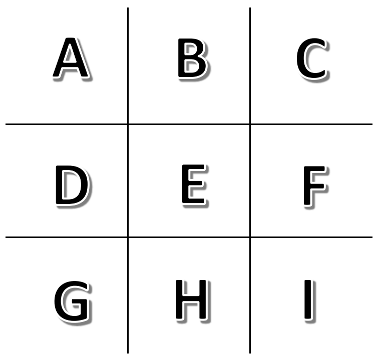


Fig. 11. The 9 boxes of the grid named with letters for A to I.

The function Y that states if a winner is found or not, is defined as follows:

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

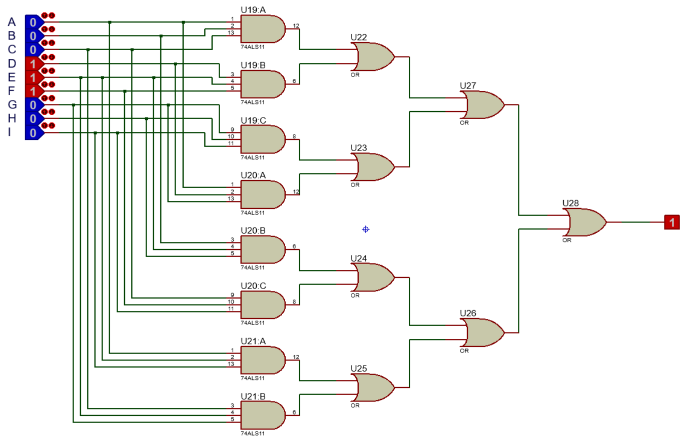


Fig. 12. Example of a match won with the combination DEF.

However, in designing the circuit we need to distinguish if the game is won by Player 1 or Player 2: we still need the function defined above but we also need to separate the two cases.

Let us consider the button configuration seen before in Fig. 5. To define WIN1 (the variable that states if Player 1 has won or not) we use the outputs of the first flip-flops of the button's configuration (the upper ones in the right side), instead with WIN2 we use the outputs of the second flip-flops. Hence:

$$WIN1 = A1 + B1 + C1 + D1 + E1 + F1 + G1 + H1 + I1$$

$$WIN2 = A2 + B2 + C2 + D2 + E2 + F2 + G2 + H2 + I2$$

Then we connect these two variables, followed both by a NOT gate, to the pins of the winning LED, as shown in Fig. 13.

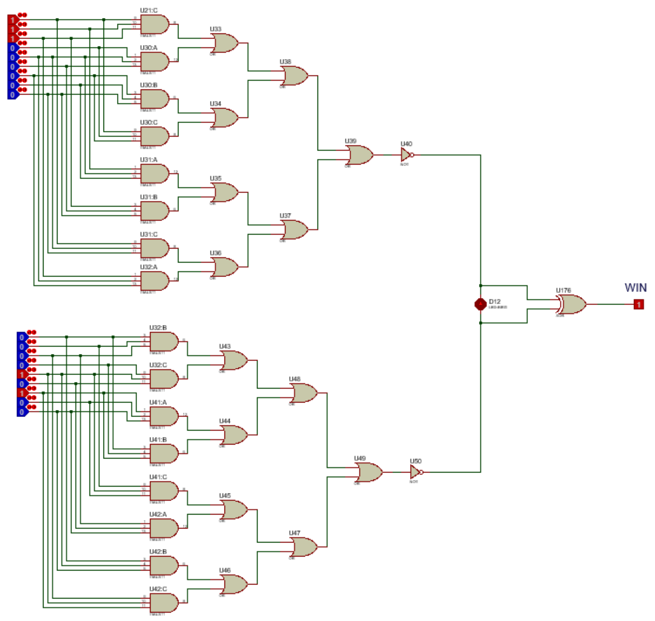


Fig. 13. Example of a match won by Player 1.

At this point it was useful to us to define a new variable WIN that indicates if the game has a winner, no matter if it is Player 1 or Player 2. It is defined as follows:

$$WIN = \overline{WIN1} \oplus \overline{WIN2}$$

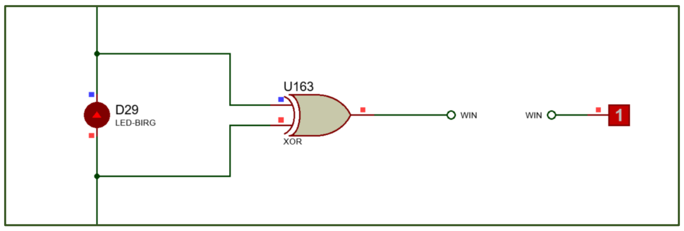


Fig. 14. The variable WIN.

In fact, this new variable is equal to 1 only when one of the two players has won the game.

Now it is very important that, after a winner is found, the game must end right away. This means that when a player wins, the remaining unpressed buttons should be blocked by modifying the system.

Therefore we have to block the other remaining switches from making any changes in the circuit. To do that, an additional AND gate should be used right after each switch of the circuit. These AND gate should connect the input coming from each switch and the winning input followed by a NOT gate, as shown in Fig. 15: it means that as long as a winner is not found the two inputs of the AND gates will be high (when we push the button) and so the output. This will not let the player light the LEDs when a winner is found.

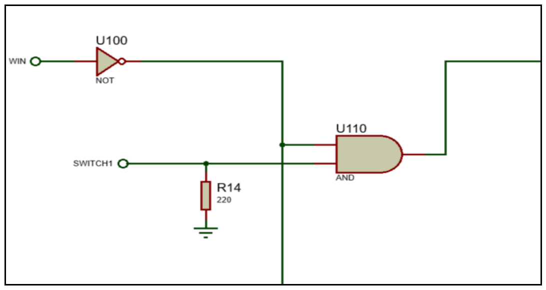


Fig. 15. How to block the game once the winner is found.

D. Draw condition

If all the buttons are pressed and a winner is not found we have a draw. An example of a draw is show in Fig. 16.

To get to the function that regulates the draw condition we use

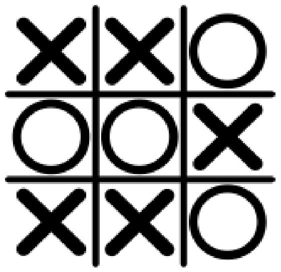


Fig. 16. Example of draw condition with no winner.

3-input AND gates to take the inputs from the buttons. Then we connect those gates and the WIN variable (followed by a NOT gate) to an additional AND gate that is connected to a single-color LED: this LED will light up if the draw condition is verified.

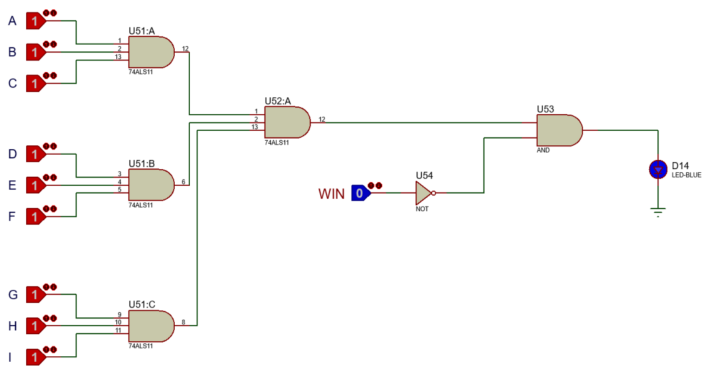


Fig. 17. All button pressed and WIN=0 lead to the draw condition.

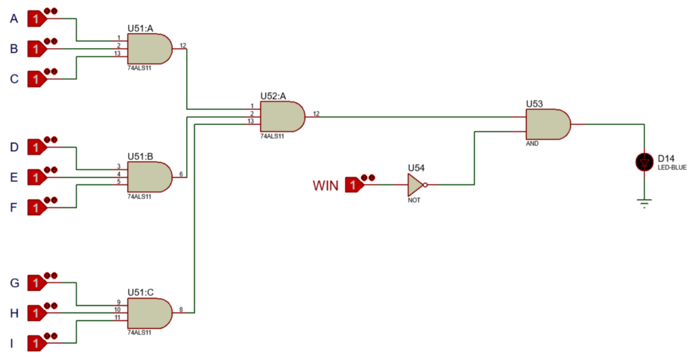
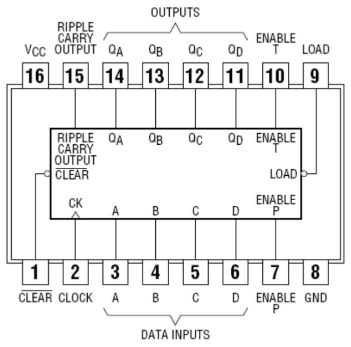


Fig. 18. If a winner is found, even if all buttons are pressed the LED will not light up

E. *Stopwatch*

We also thought it could have been useful to design a stopwatch to track the duration of each match. To do that we used 74161 4-bit binary counters. As we can see from the function table in Fig. 19, with this type of counters the input is enabled when the load is equal to 0. Therefore to load the input data, the clear input must be equal to 1 and the load input must be equal to 0. For this reason, when we want to start counting over, we should connect a NAND gate to the load rather than an AND gate.



Clear	Clock	Load	Count	Function
0	X	X	X	Clear outputs to 0
1	↑	0	X	Load input data
1	↑	1	1	Count to next binary value
1	↑	1	0	No change in output

Fig. 19. IC type 74161 binary counter with parallel load and function table

The stop-watch starts counting when one of the 9 buttons is pressed, and stops when we have either a winner or a draw condition. When we press the reset button the stopwatch should be reinitialized. The results of the counters are shown in 7-segment displays.

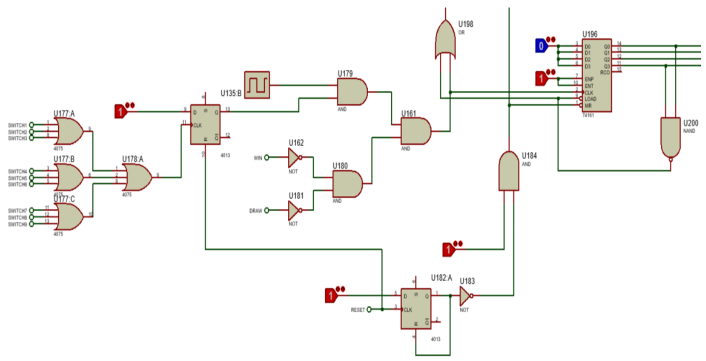


Fig. 20. Configuration of the inputs of the stop-watch.

Referring to Fig. 21, starting from the bottom, the first two displays represent the seconds, while the other two the minutes.

The first counter counts up to 9: when it reaches the value of 9, the NAND gate value is 0, so the inputs are loaded and the counter starts counting again from 0. In this transition the clock of the next counter is triggered and therefore the second counter increases of 1 each time the first one goes from 9 to 0.
The second counter counts up to 5. When we have the value 5 in the second counter and the first one goes from 9 to 0, then they are both resetted to 0. Here the third counter is triggered and it is increased of 1, representing the first minute gone.
The maximal value the stopwatch can reach is 99 minutes and 59 seconds. Then it restarts from 00:00.

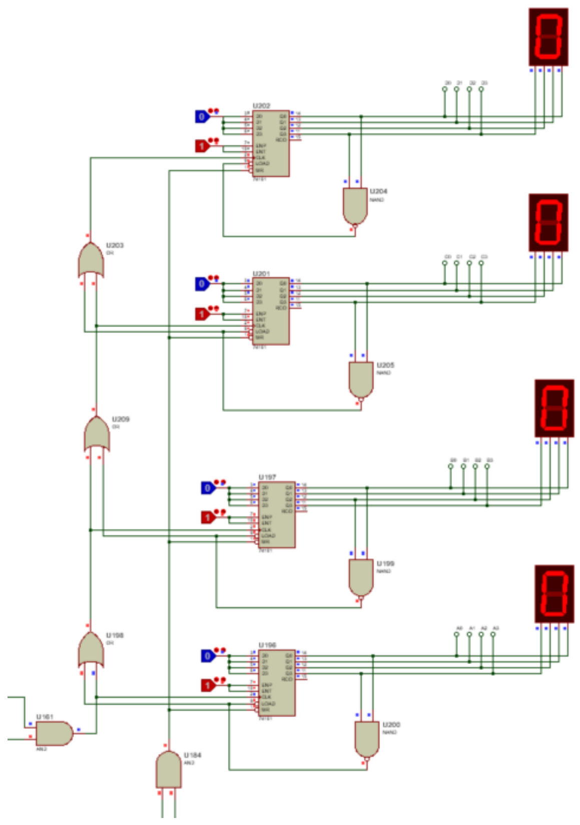


Fig. 21. Stopwatch design.

IV. PROBLEMS ENCOUNTERED

During the design of the project we had mainly two problems: one was the problem to not let other players overwrite moves and the other one was dealing with 9 inputs when we had to find the function of the players' configuration (although this latter turned out to be a very simple situation).

Our first approach in designing the button configuration was the one shown in Fig. 22.

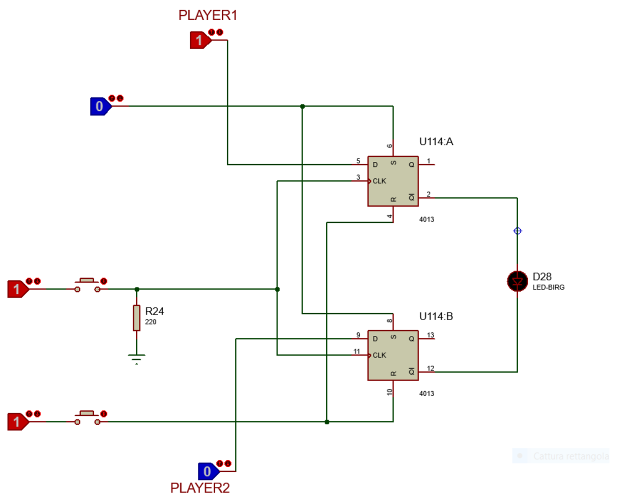


Fig. 22. Our first design of the button configuration.

This configuration did not satisfy all the requirements of the game. Although the LED lighted up as we wanted, this configuration allowed the other player to change the previous moves because the two flip-flops had their positive trigger each time the button was pressed.
.
It was only by placing the additional flip-flop that we were able to solve the problem: the flip-flop tracked the state of the button, so that if it was already pressed it could not be possible to change its value in the following moves.

The second problem was to find the function of the players' configuration. To get to the final expression we had to make a Truth Table with 9 inputs, and then a 9-variable K-map that was too complicated and required a lot of time. To solve the problem we thought to reduce the number of the variables since the logic the regulated the process was the same either we had more buttons or less buttons: the function just had to switch the players' value at each turn. We considered only 4 inputs.

However we found out that the function was simpler than we thought: a XOR gate of all the inputs, that gave 1 if an odd number of inputs were high, 0 if an even number of inputs were high.

References

The idea to make this project came after seeing this work on the website https://www.instructables.com/id/Arduino-Touch-Tic-Tac-Toe-Game/. It is the same game designed with Arduino: however, the project was made with only codes and without relying on logic gates.

[1] M. Morris Mano, Michael D. Ciletti, "Synchronous Sequential Logic" in *Digital Design, 4*th edition, Pearson Education & Publishing House of Electronics, China, 2008, pp. *182–225.*

[2] M. Morris Mano, Michael D. Ciletti, "Registers and Counters" in *Digital Design, 4*th edition, Pearson Education & Publishing House of Electronics, China, 2008, pp. *242–269.*

V. Conclusion

After designing even a simple game such as Tic Tac Toe, a new sensibility for the effort that goes into creating digitally controlled devices was obtained. There are numerous methods and approaches that can be taken to achieve the same goal, some simpler than others which was a valuable lesson. Designing even what it seems to be simple may take you a lot of time and might not be very immediate. We hope this was only a beginning and that someday we will be able to design also more complicated things.

VI. Biography

Ivan Alberico was born in Marcianise, CE, Italy in 1998. In 2014, at the age of 15 he studied for 8 months in Ireland, at St. Joseph School, Drogheda, Louth. He graduated from the scientific high school "F. Quercia" in Marcianise in 2017. Since September 2017 he moved to Bologna, where he started his degree in Automation Engineering at the University of Bologna, Alma Mater Studiorum. He is currently in Shanghai, China, studying at Tongji University to get the double degree.

Silvana Pelle was born in Locri, RC, Italy in 1999. She graduated from the classic high school "Ivo Oliveti" in Locri in 2017. Since September 2017 she is studying Automation Engineering in the University of Bologna and she is currently staying in Shanghai studying at Tongji University.