NEW HORIZON
COLLEGE OF ENGINEERING
AUTONOMOUS COLLEGE Permanently Affiliated to VTU, Approved by AICTE & UGC
Accredited by NAAC with 'A' Grade

DESIGN AND IMPLEMENTATION OF TIC-TAC-TOE

A MINI PROJECT REPORT

Submitted by

ANJALI R(1NH18EC008)

In partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

IN

ELECTRONICS AND COMMUNICATION

ENGINEERING

NEW HORIZON COLLEGE OF ENGINEERING

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

CERTIFICATE

Certified that the mini project work entitled "**DESIGN AND IMPLEMENTATION OF TIC-TAC-TOE**" carried out **Anjali R (1NH18EC008)** bonafide student of Electronics and Communication Department, New Horizon College of Engineering, Bangalore. The mini project report has been approved as itsatisfies the academic requirements in respect of mini project work prescribed for the said degree.

Project guide	HOD ECE
Ms Monika Gupta	Dr. Sanjeev Sharma
Senior Assistant Professor	B. Tech M. Tech PhD
Dept of ECE	NHCE
NHCE	
-----------------	-------------

External Viva

Name of Examiner	Signature with Date
1.	
2.	

ABSTRACT

A game can be claimed that it is totally understood when one can implement it in various other ways. One of most important aspect to design a game using logic circuits is the understanding of the game rules in depth. Also, one must have complete knowledge of basic logic gates and its functionality.

The friendliness of Tic-tac-toe game makes them ideal as a pedagogical tool for teaching the concepts of good sportsmanship. Tic-tac-toe is a game with simple rules that can be easily implemented with the knowledge of combinational and sequential logic design. It is essential to understand the importance of automation using simple blocks.

This project revolves around the implementation of the game tic tac toe in two ways namely by using logic ICs and simulating using a tool known as Xilinx ISE. The simulation part of the project helps in verification of the logic whereas the logic ICs help in the hands-on experience of the design.

The tic tac toe game is a very good brain exercise as it involves looking ahead and trying to figure out what the person playing against you might do next. This trait of the game makes it more interesting while designing it for a single player using artificial intelligence. Since there are multiple possible moves the opponent can take, it becomes more challenging to develop an artificially intelligent bot.

ACKNOWLEDGEMENT

The satisfaction that accompanies the successful completion of any task would be, but impossible without the mention of the people who made it possible, whose constant guidance and encouragement helped us succeed.

We thank **Dr. Mohan Manghnani**, Chairman of New Horizon Educational Institution, for providing necessary infrastructure and creating a good environment.

We also record here the constant encouragement and facilities extended to us by **Dr. Manjunatha**, Principal, NHCE and **Dr. Sanjeev Sharma**, head of the department of Electronics and Communication Engineering. We extend sincere gratitude to them.

We sincerely acknowledge the encouragement, timely help and guidance given to us by our beloved guide **Ms. Monika Gupta** to complete the project within stipulated time successfully.

Finally, a note of thanks to the teaching and non-teaching staff of the electronics and communication department for their cooperation extended to us, who helped us directly or indirectly in this successful completion of the mini project.

Anjali R (1NH18EC008)

TABLE OF CONTENTS

LIST OF TABLES:

LIST OF FIGURES:

LIST OF ABBREVIATIONS:

Sl. No.	Abbreviation	Abbreviated as:	Page no.
1	ISE	Integrated Synthesis Environment	3
2	OXO	Noughts and Crosses	10
3	EDSAC	Electronic Delay Storage Automatic Calculator	10
4	LED	Light Emitting Diode	14
5	IC	Integrated circuit	14
6	VHDL	VHSIC Hardware Description Language	16
7	VHSIC	Very High-Speed Integrated Circuit	16
8	D flip flops	Data flip flops	16
11	FPGA	Field Programmable Gate Array	16
12	ASIC	Application Specific Integrated Circuit	16
13	CMOS	Complementary Metal Oxide Semiconductor	34
14	MOSFET	Metal Oxide Semiconductor Field Effect Transistor	34
15	TTL	Transistor Transistor Logic	35
16	RTL	Register Transfer Level	38
17	PAR	Place and Route	41
18	IEEE	Institute of Electrical and Electronics Engineers	50
19	EDA	Electronic Design Automation	50

CHAPTER 01

INTRODUCTION

Tic Tac Toe game can be played by two players where the square block (3x3) can be filled with a cross (X) or a circle (O).

A simple thumb rule is used in this game to declare the winner; and that is, when one of the players make a combination of 3 same markers in a horizontal, vertical or diagonal line the program will display which player has won, whether X or O.

Further more we would get into the details of the history involved in the development of the game.

A BRIEF HISTORY :

The Games that are played on three-in-a-row boards can be traced back to ancient Egypt, where those kinds of game boards were being found on the roofing tiles dating from around 1300 Before the Common Era.

During the early times, the variation of tic-tac-toe was found to be played in the Roman Empire, that was probably around the first century Before Christ. It was earlier called as ***terni lapilli*** which means *three pebbles at a time*, and instead of them having many or any number of pieces, each player then only had three pieces, thus they had to keep moving them around to the empty spaces so that the game was on and they could keep playing. As this game became popular, The game's grid markings were found chalked all over Rome. There were even similar kind of games that are closely related to this ancient game, namely *three men's morris* which was also played on a simple grid and it also requires three pieces in a row to finish the game. Picaria, another game of the Puebloans.

The different names of the game are more recent. The first print of reference was to “noughts and crosses” (nought here being an alternative word of zero), it was a British name that was appeared in 1858, in an issue of Notes and Queries. The first print of reference to the game called “Tick-Tack-Toe” was occurred in 1884, but it was more referred to “a children’s game played on a slate”. “Tic-tac-toe” might have also derived from “tick-tack” which is the name of an old version of the Backgammon that was firstly described in the year 1558. And in the early stages of 20th century, United States named it as “tic-tac-toe” removing the name “noughts and crosses”.

In the year 1952, a British computer scientist named Sandy Douglas developed the OXO (or Noughts and Crosses) for the EDSAC (Electronic Delay Storage Automatic Calculator) computer at the university of Cambridge, which became one of the first known video games. The computer player was able to perfectly play the game of tic-tac-toe against a human opponent.

The students of MIT (Massachusetts Institute of Technology) used this Tic-tac-toe to demonstrate the computational power of Tinker toy elements. The Tinker toy computer that is made out of almost only Tinker toys is able to play the tic-tac-toe game perfectly.

THE GAME PLAY :

This is a relatively simple game that is usually played on a grid of 3-by-3 squares, to make it more complex it can even be played by increasing the size of the grid/ board to 4-by-4 or 5-by-5 The goal of tic-tac-toe game is to be the first player/person to get three in a row or a column or a diagonal on a 3-by-3 grid.

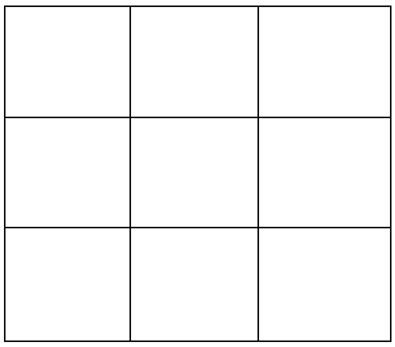


Fig-1(1) 3-by-3 grid of the game

● Consider you are **X**, your opponent as **O**. Players take turns in putting their marks in the empty squares.

● The first player to get three of her/his marks in a row or a column or diagonally is considered as a winner.

● If no player has got 3 marks in a row then the game is considered to be a tie.

To win at tic-tac-toe you got to have the strategy meaning you'll have to figure out what you need to do to win , which includes

● Figuring out how to get 3 **X**s in a row or column or diagonally.

● Another part is how do you stop your opponent in getting 3 **O**s in a row or a column or diagonally.

● When you are done with the first turn of yours, you'll have to think ahead as to where you should place your **X** in the next turn.

● Look at the empty squares and decide which one makes the good choices, that might also make up the three **X**s in a row or column or diagonally.

● Watch your opponent's move so that you could change your next move.

If attention is paid and you always look ahead, you'll never have a chance of losing the game of tic-tac-toe rather it will be a tie.

Here is a sample of the tic-tac-toe game moves.

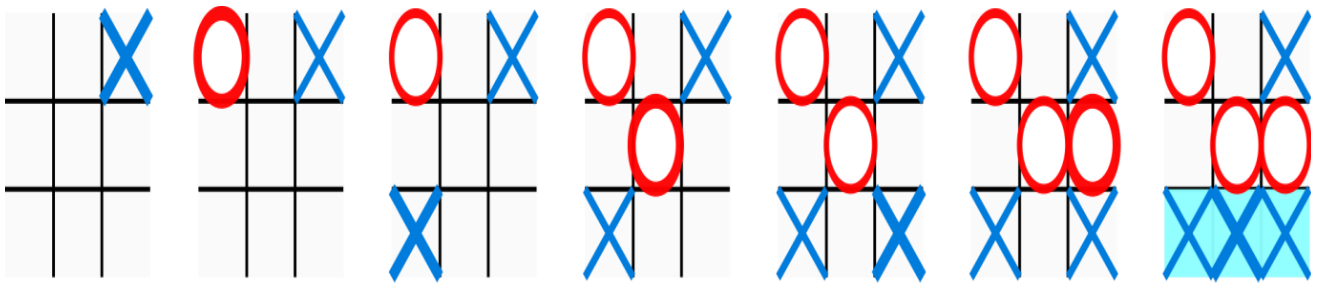


Fig-1(2) The Game Play

CHAPTER 02

LITERATURE SURVEY

Title of the paper	Author & Year of Publication	Outcome	Limitation
Mathematical Recreations	Steve Schaefer 2002	There are 1145 possible games of Tic-Tac-Toe.	Number of games is to consider the decisions faced by the players.
Exploring the possibilities	Clarkson P.C. 2008	Different ways to think and communicate about mathematics.	Artificial intelligence modules should be employed. Which will increase the complexity of the device.
Tic-tac-toe played as a word game	David Silverman 2010	It describes a linguistic version of the game Tic-tac-toe, where the players opt words and the first player to collect words sharing a common letter is declared as the winner	The logic cannot be generalized hence a working model cannot be built rather a code can be written to implement the same.
How to not lose at Tic-Tac-Toe	Thomas Bolan 2013	Always start in the corner nd make 3 combinations of the same markers. It gives the biggest possible chance of winning	There is a slight chance of losing or tying the game if the opponent is a good player.
Introducing Game Theory	Elliott Mendelson 2016	Either a player has a winning strategy or both players have a non-losing strategy.	Description of strategy can be enormously complicated.

Table 2(1)- literature survey

CHAPTER 03

PROBLEM STATEMENT AND EXISTING SYSTEM

PROBLEM STATEMENT:-

Understanding, designing and implementation of the game Noughts and Crosses using various tools.

OBJECTIVES:-

- To understand the functionality of basic logic gates.
- To realize logic of a circuit and build the circuit using software's like Easy EDA
- To implement sequential and combinational logic circuits in Verilog and simulate them using appropriate test benches.

EXISTING SYSTEM :-

Tic-Tac-Toe / Noughts and Crosses / X's and O's is a pencil and paper game for two players who take turns marking the spaces in a 3 x 3 grid. The player who succeeds in placing three consecutive marks (either 'X' OR 'O') in a horizontal, vertical or diagonal row wins the game.

CHAPTER 04

PROPOSED METHODOLOGY AND PROJECT DESCRIPTION

PROPOSED METHODOLOGY :-

There are multiple methodologies used to implement the tic-tac-toe game, in which few of them are

1. Artificial intelligence,
2. Using ICs and LEDs and
3. Describing the hardware of the logic using HDL and dumping it on a target device.

ARTIFICIAL INTELLIGENCE :-

To implement the game of tic tac toe using unbeatable artificial intelligence modules. We employ an algorithm known as Minimax Algorithm in order to achieve this. Minimax algorithm is widely used in domains such as game theory, statics philosophy and artificial intelligence. The minimax algorithm is a recursive or iterative algorithm that is implemented to choose an optimal move for a player assuming that the other player is also taking optimal moves. The main goal of the minimax algorithm is to minimize the maximum loss, in other words minimize the worst-case scenario. The following diagram represents all the possible moves for a given condition.

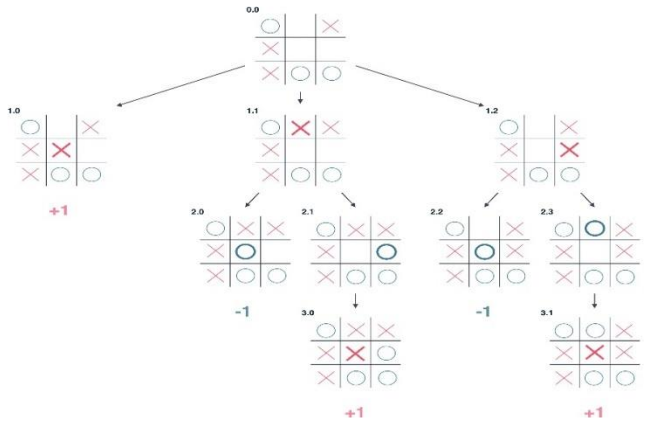


Fig 4(1)-The minimax algorithm

For every possible move there are at least 2 moves which the module analyses before taking the move thus minimizing the number of worst cases. Similar simulations are done for every game move thus making itself an unbeatable opponent.

USING ICs AND LEDs:-

In this method digital ICs are used. Along with few other hardware components such as LEDs and push buttons. The circuit is segmented into blocks namely switch design, player profile generator, all switches pressed(reset), winning logic and umpire circuit.

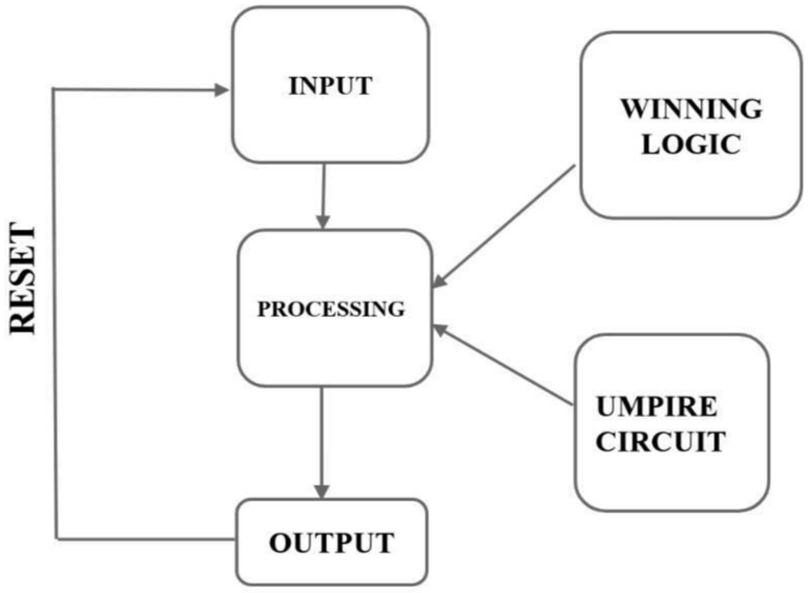


Fig-4(2) Flowchart

SWITCH DESIGN: Controls the move made by the player. Also, it holds the value (high or low) of each state. 9 such circuits are built for all the 9 switches representing the 9 possible positions.

PLAYER PROFILE GENERATOR: keeps check on whose move it is(player one or player two). Which is nothing but XOR operation of all the inputs.

ALL SWITCHES PRESSED: When all the switches are pressed this circuit comes into picture. It resets the switch design circuit and makes the complete circuit ready for the next game play.

WINNING LOGIC: This circuit helps in the detection of the winner. It is ANDing of the possible winning combinations and OR-ing the outputs. Thus, aiding in the detection of the winner.

UMPIRE CIRCUIT: After the winner is detected the next player's move is supposed to be blocked. This is done by the umpire circuit. The umpire circuit is nothing back to back connection of D flip-flops.

USING HDL:-

The tic-tac-toe game can be played on a Field Programmable Gate Array. An FPGA is a target device, the hardware which is to be used is described using any one of the hardware description languages such as system Verilog, VHDL and Verilog. Using hardware description language, we can simulate the design and cross check logic if it is functioning as per the requirements. The logic is written using structural style of coding. Software’s such as Xilinx, altera can be employed to design and simulate. The simulation is done by generating testbenches for the code. Testbench is a piece of code that is used to provide input combinations to test a module. It provides stimuli to the system or design under test.

Fig-4(3) The software used

The methods (2) and (3) are implemented in this project.

Reason for choosing these methods are:

1) Easy output analysis.
2) Simple design.
3) Easy testing and debugging .
4) Effortless hardware generation.

PROJECT DESCRIPTION :-

Since there are different circuits to be combined, we have segmented the project into blocks. The block diagram along with the complete description if the game tic-tac-toe is given below.

THE BLOCK REPRESENTATION OF TIC-TAC-TOE GAME :-

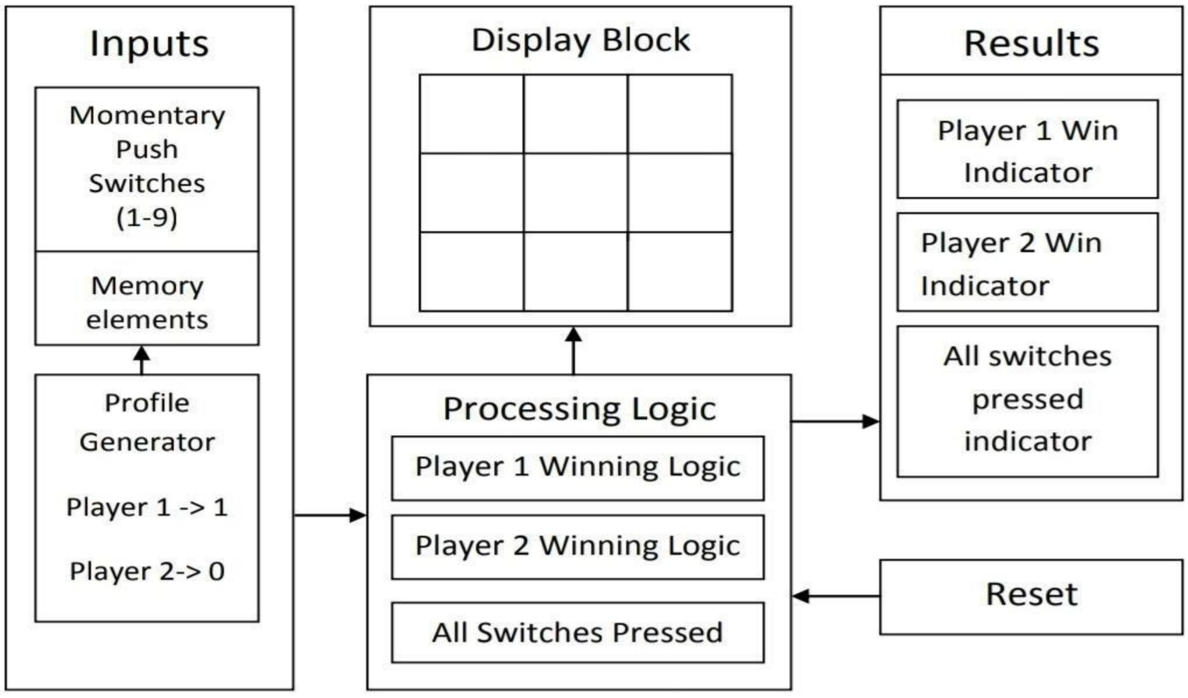


Fig-4(4) Block diagram

Let us now know about each sub block ,the circuit designing and the working part of it.

INPUT BLOCK

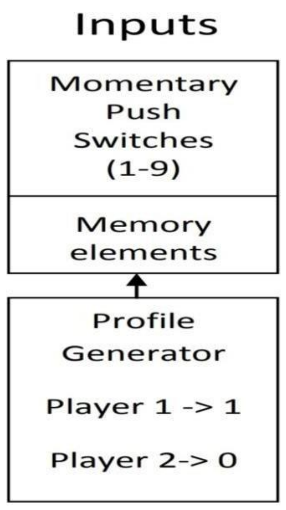


Fig 4(5)- input block

4.2.1(a) WORKING OF THE INPUT BLOCK :-

The user gives the input and these inputs are nothing but the momentary micro switches. We here make use of the D flip flops to know which switch was pressed and to have a track of input. To know which player’s move it was, we store it. Here’s the circuit where one switch circuit is duplicated to nine switches as we are designing a 3-by-3 grid board which has nine empty squares.

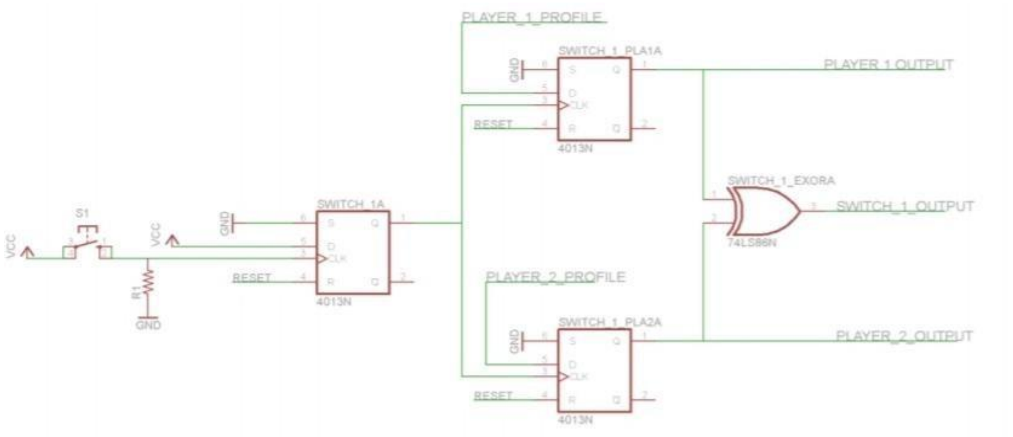


Fig 4(6)- switch design

We make use of D flip flops that are positive edge triggered. We have a master flip flop SWITCH_1A that triggers the flip flop SWITCH_1_PLA1A and SWITCH_2_PLA2A. When switch S1 is pressed by the user, the output 'Q' of SWITCH_1A goes from zero to one i.e., from low to high triggering the other two flip flops.

The data/input for the rest two flip flops comes from the '*Profile Generator Circuit'.* Here the profile generator circuit will have two lines, they are 'PLAYER_1_PROFILE' and 'PLAYER_2_PROFILE'. When it is the first player's turn to play, 'PLAYER_1_PROFILE'line goes to 'one' i.e. 'high' and 'PLAYER_2_PROFILE' line goes to 'zero' i.e. 'low' and vice versa happens.

We shall face two cases when the switch S1 is pressed,

CASE 1 :- when its first player's turn, the output 'Q' from the SWITCH_1_PLA1A will go high.

CASE 2 :- when its second player's turn, the output 'Q' from the SWITCH_1_PLA2A will go high.

To be noted : It is impossible for the output 'Q' from both the switches to go high at the same time. And the XOR can be avoided by directly taking the output 'Q' from the SWITCH_1A .

4.2.1(b) PROFILE GENERATOR CIRCUIT :-

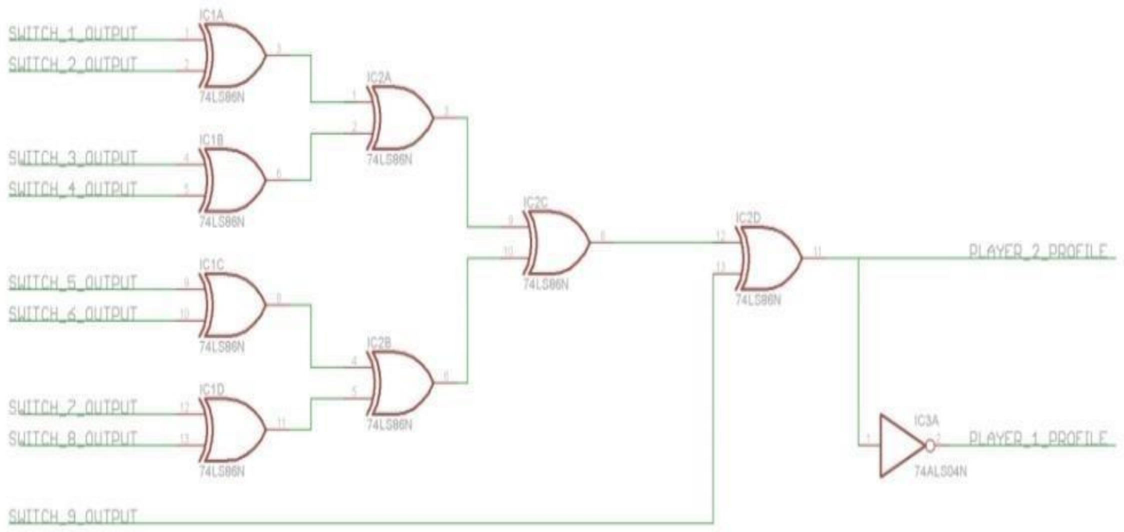


Fig 4(7)- profile generator

This is the circuit that helps us to find out which players turn is it to play currently. As it consists of XOR gates, if no one has played yet the inputs will remain as zero; hence the output for PLAYER_2_PROFILE will be ‘zero’ and the PLAYER_1_PROFILE will be ‘one’ because we have made use of an inverter to invert the output.

Once the game begins, the player1 shall play and one of the inputs goes to ‘one’, which makes the overall XOR output to go to ‘one’, this in turn makes the player2 profile to ‘one’ and PLAYER_1_PROFILE to ‘zero’.

This keeps repeating and we get the results of player1 and player2 alternatively. Basically, it is noticed that for every change in the input, output toggles.

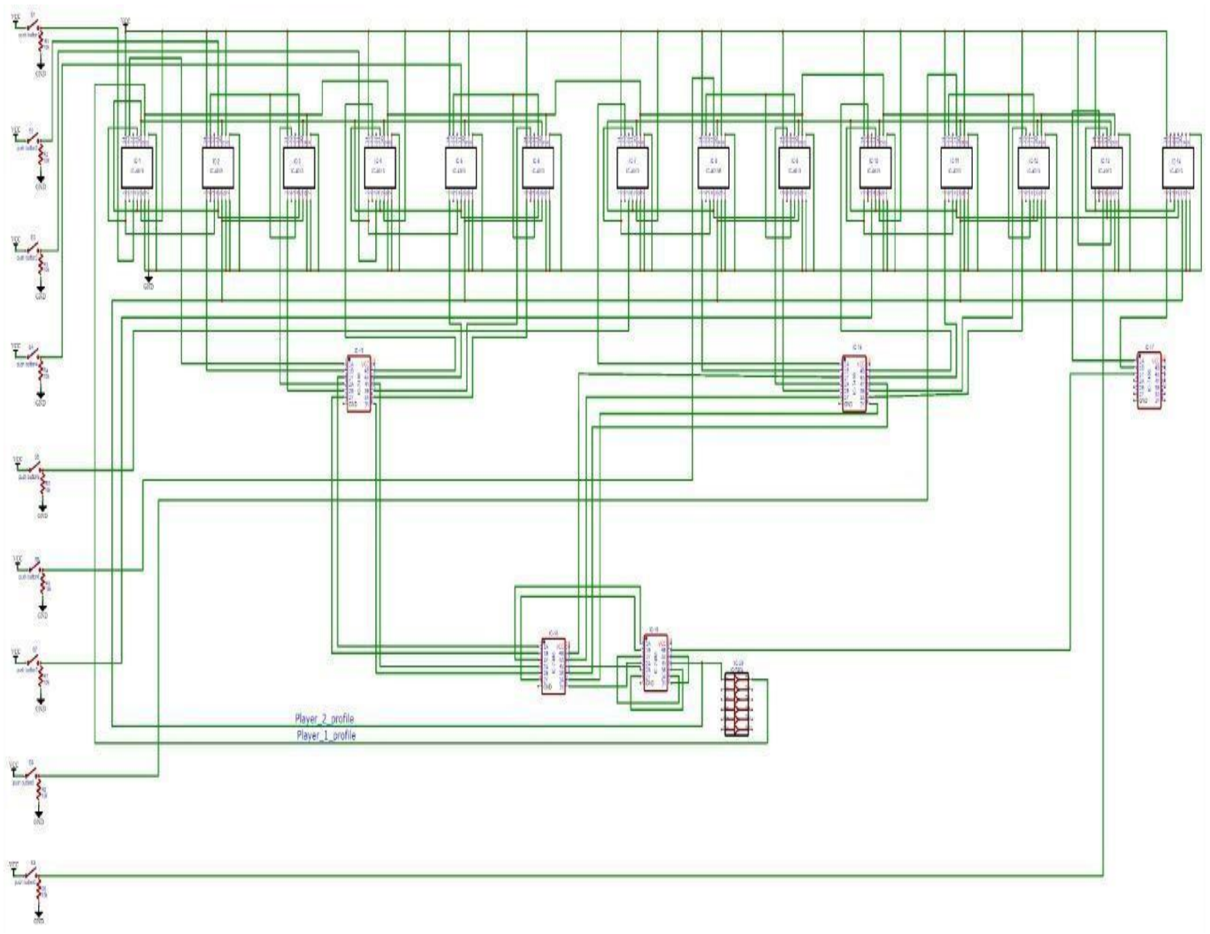


Fig 4(8)- pin diagram of input block

PROCESSING LOGIC

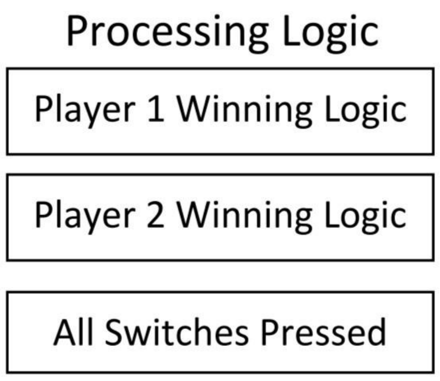


Fig 4(9)- processing block

The inputs of the circuit is given from the block one whose main constituent are switches. As the game progresses, there arises a need to declare the winner. The processing logic aids in the detection of the winner. The rules of the game states that when one of the players make a combination of 3 same markers in a horizontal, vertical or diagonal line the program will display which player has won, whether X or O. these combinations are fed as the inputs to the three input AND gates. There are eight such combinations for a player to be declared as the winner. The eight combinations are as follows: A B C, D E F, G H I, A D G, B E H, C F I, C E G, A E I

When any one of these eight combinations is achieved by the player he/she is declared as the winner.

A	B	C
D	E	F
G	H	I

We employ AND logic in combination with OR logic to achieve this design. If all the three, player switch status required for a win case are high only then will one of the inputs of the OR logic turn high, if any one of the inputs to OR logic are high the output turns high thus declaring the winner. This is duplicated twice to check the win status of each player.

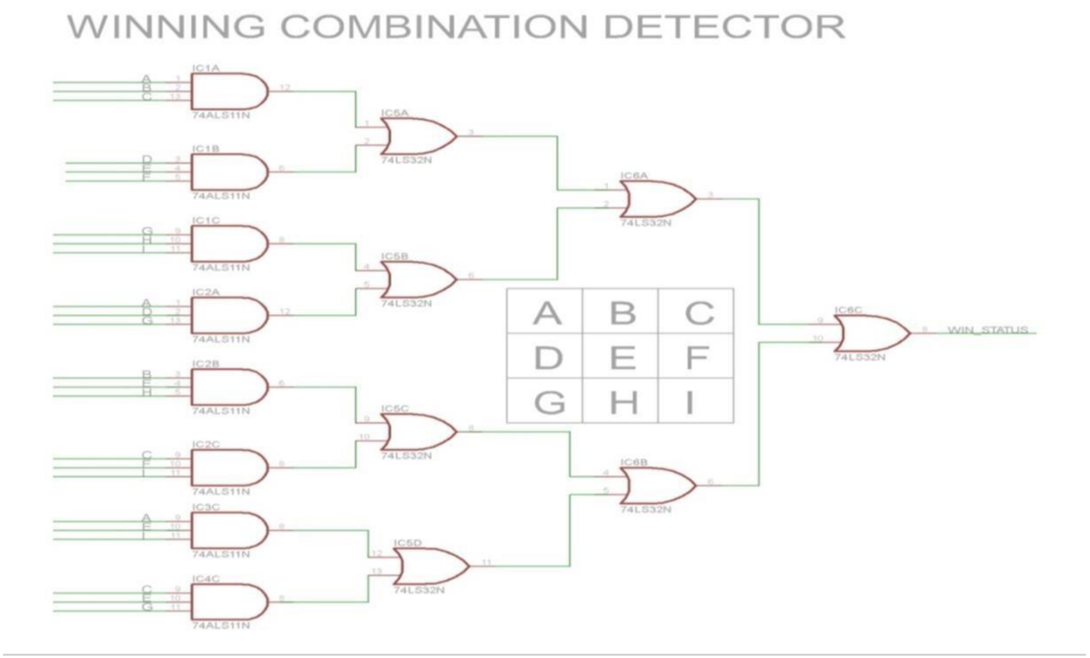


Fig 4(10)- internal diagram of processing logic

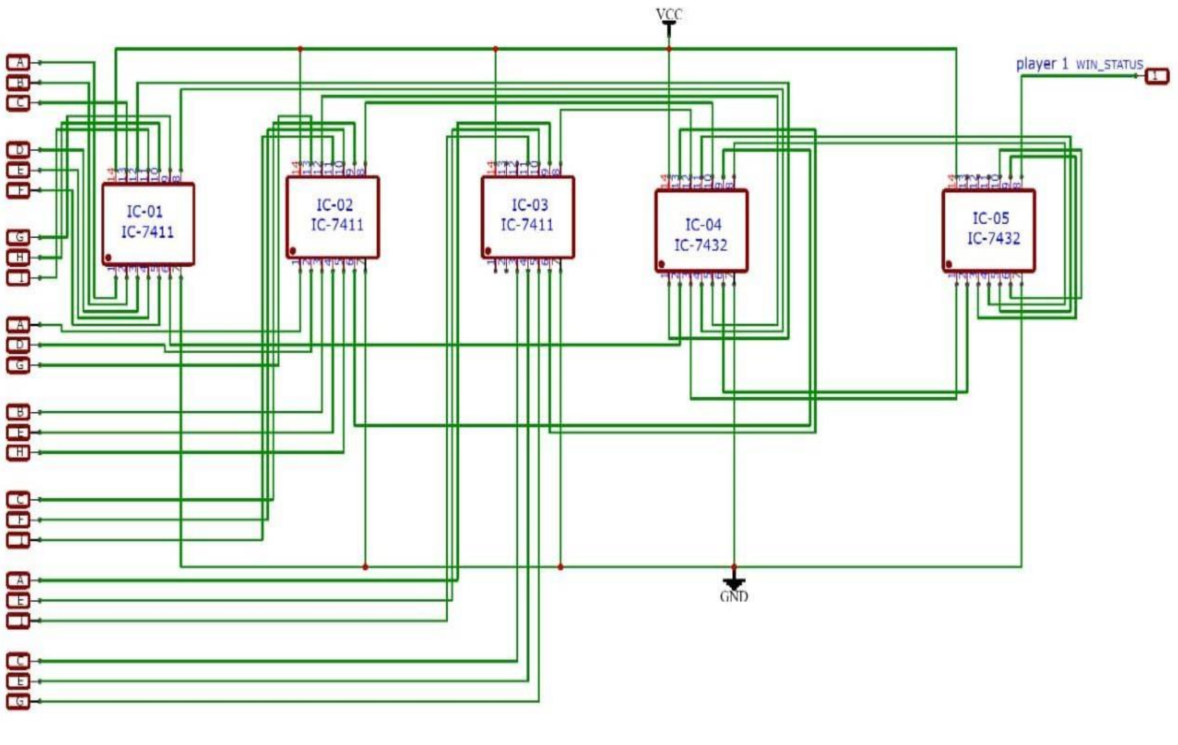


Fig 4(11)- pin diagram of processing logic of player 1

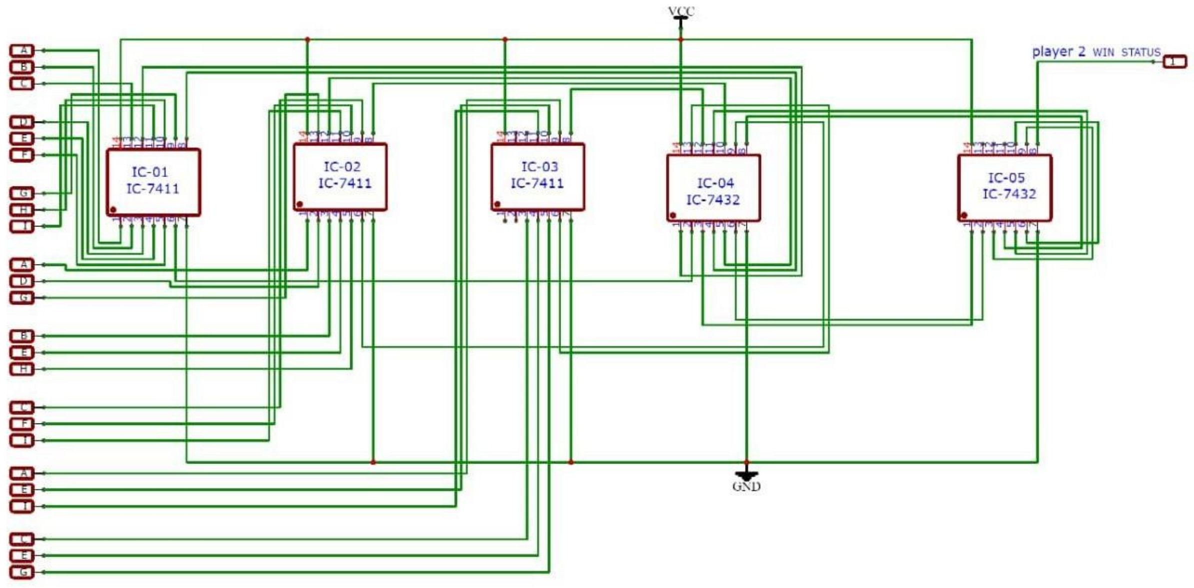


Fig 4(12)- pin diagram of processing logic of player 2

ALL SWITCHES PRESSED/RESET LOGIC :-

In this circuit we employ AND logic using 7408 IC to achieve this design. If all the switches are pressed at once, we need to bring back the circuit to the original state i.e. “Reset”. This is where all switches circuit comes into action. Here we just AND all the outputs from the switches. When all the switches are pressed, the output of the circuit goes high which means it’s a draw or that a new game should be started.

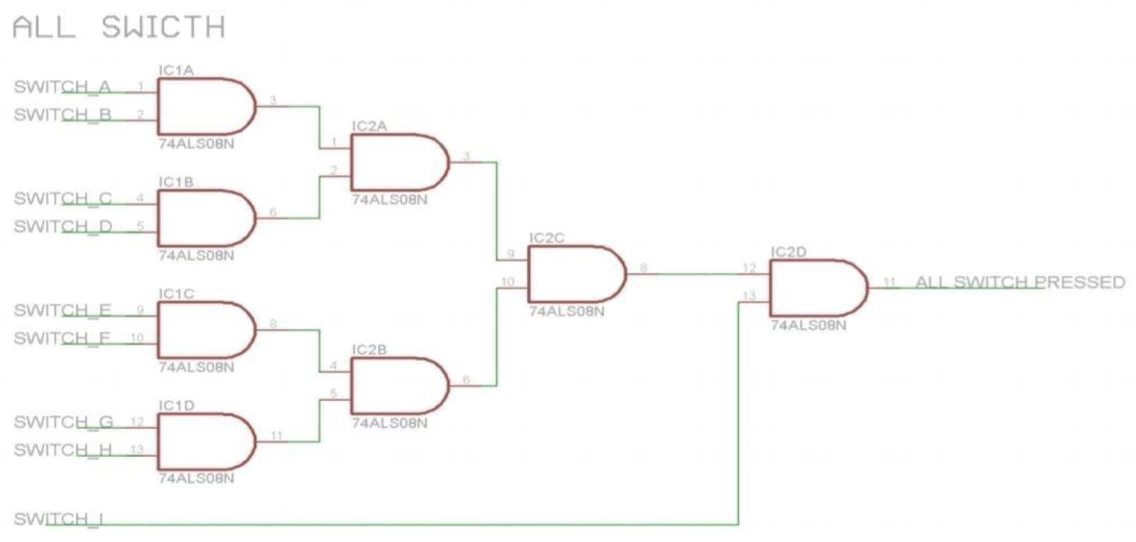


Fig 4(13)- internal diagram of reset logic

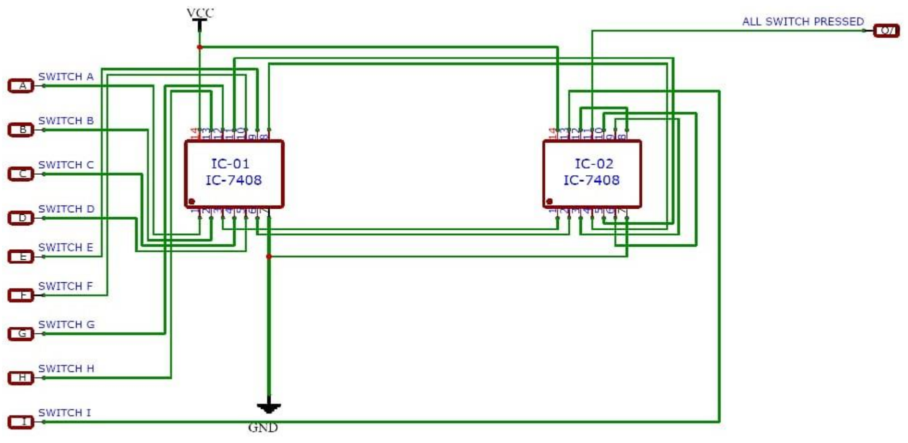


Fig 4(14)- pin diagram of reset logic

UMPIRE CIRCUIT :-

In this circuit, we employ two interconnected D-Flip Flop to achieve this design. We use this circuit to avoid the opponent to get another chance to complete the game while the winner has already been declared.

Let us consider that Player 1 has won the game. The output of the Winning combination for player 1 detector turns high. This output is connected to the umpire circuit to the clock pin thus we can conclude that win status of player one triggers the first(top) flipflop and the win status of player 2 triggers the second flip-flop. The negated output of one flip-flop is connected to the D input of the other and vice versa. This interconnection ensures that Player 2 does not getting another move to complete the game while the Player 1 has already been declared the winner.

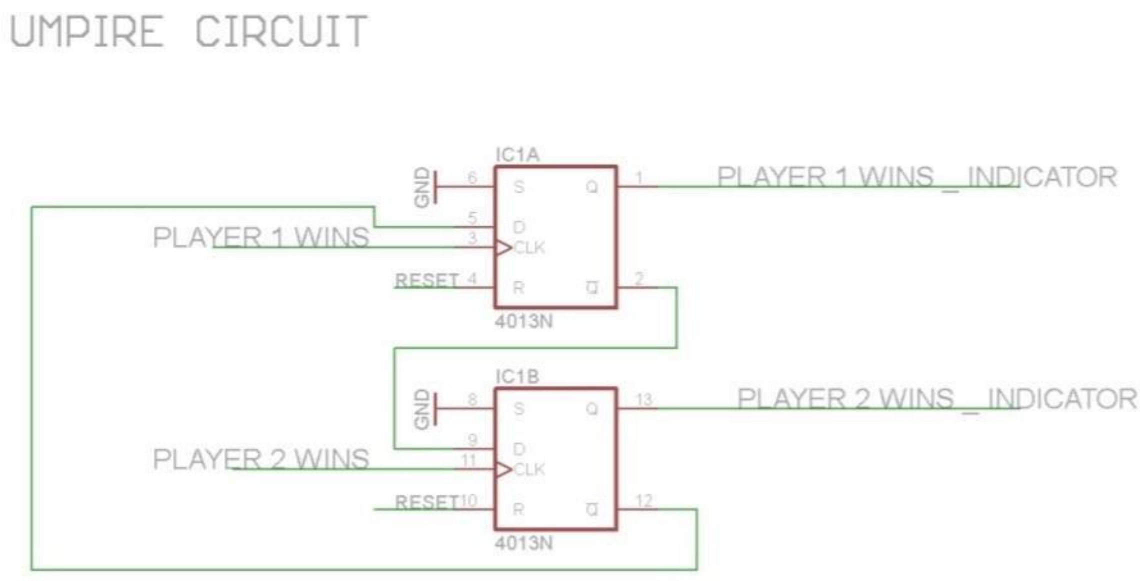


Fig 4(15)- internal diagram of umpire circuit

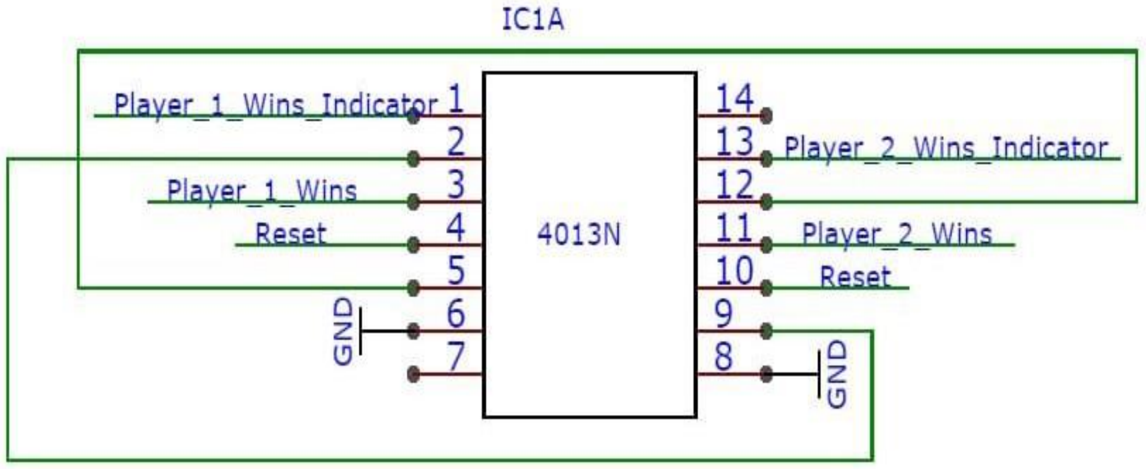


Fig 4(16)- pin diagram of umpire circuit

DISPLAY BLOCK :-

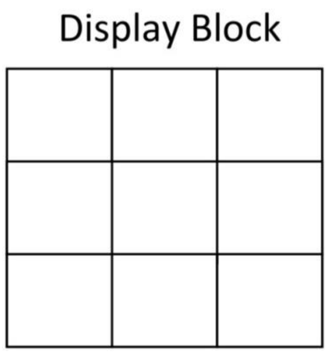


Fig-4(17)-The output/display block

We represent the players with two different colors, thus we make use of this hardware component Dual color LED.

The Dual color LED’s are arranged on a 3-by-3 board in a matrix form. Outputs from the ‘switch block’ are given to the corresponding LED. One leg of the LED is given to player1 and another leg of the LED to player2.

Here the color red represents player1 and green represents player2. If the LED glows ‘red’, it basically means that it was the first player’s move and if the LED glows ‘green’ it means it was the second player’s move.

CHAPTER 05

HARDWARE DESCRIPTION AND SOFTWARE DESCRIPTION

The following components have been used to build this circuit.

1. IC-4013 (Dual D flip-flop)
2. IC-7486 (XOR gate)
3. IC-7411 (3 input AND gate)
4. IC-7404 (NOT gate)
5. IC-7432 (OR gate)
6. Dual color LED
7. Push buttons
8. Bread board
9. Resistors
10. 9V battery

A detailed explanation about these elements are given as follows for better understanding of the circuit.

THE DUAL D FLIP-FLOP

IC number used- 4013

In case the mentioned IC is not available, it can be replaced with the following ICs 74HC74, 74HC174, 74LS75, 74LS74, 74LS175, 74LS273, 4013B, 40174B.

A few specifications of the IC used are

- Asynchronous Set-Reset Capability
- Medium-Speed Operation: 16 MHz (Typical) Clock Toggle Rate at 10-V Supply
- Maximum Input Current Of 1-µA at 18 V

● Over Full Package Temperature Range: – 100 nA at 18 V and 25°C

Dual D flip-flop IC consists of two identical D flip flops in it. Each D flipflop consists of a single input D, RESET ,SET, CLOCK ,and outputs Q and negotiation of Q. D stands for DATA and the Flip flop is used to store values on data lines. D flop-flop is designed from gated S-R flipflop with an inverter inserted in between the S and the R inputs to allow the single D (data) input. When clock is '1', if input is '0',then it follows 'reset' condition else 'set' condition.

It can be constructed using transmission gates.

A few applications od D flip-flop are

1. used to create delay- lines in digital signal processors.
2. used as Frequency divider.
3. counter display at bank, token counter, microwave oven timer, event detector.

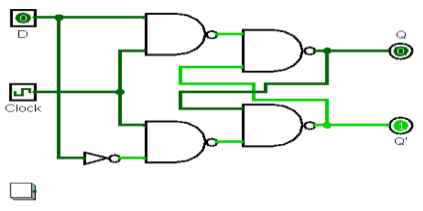


Fig 5(1)-D flip-flop internal circuit

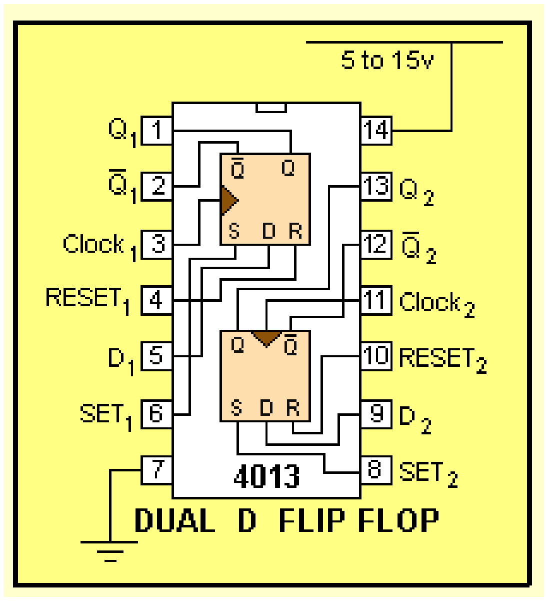


Fig 5(2)- IC 4013 internal pin diagram

PIN NUMBERS	OPERATIONS
1	Q1
2	Q1 bar
3	Clock1
4	Reset1
5	Data1
6	Set1
7	VSS
8	Set2
9	Data2
10	Reset2
11	Clock2
12	Q2 bar
13	Q2
14	VDD

Table 5(1)- pin description of IC 4013

THE XOR GATE

IC number used- 7486

In case the mentioned IC is not available, it can be replaced with the following ICs SN74HCS86, SN74HCS86-Q1, SN74LVS1G86-Q1, SN74LV86A-Q1, CD74ACT86-EP, SN74LV86A-EP, SN74LVC86A-EP

Absolute Maximum Ratings of the IC used are:

- Supply Voltage 7V
- Input Voltage 5.5V
- Operating Free Air Temperature Range 0°C to +70°C
- Storage Temperature Range −65°C to +150°C

XOR gate is also known as EOR,EXOR, Exclusive OR. It is 14 pin IC. XOR gate gives output as True(high) if the inputs are not alike, else the output is False(low). XOR gate can be constructed using MOSFET's. Available XOR versions are TTL logic XOR gate, CMOS logic XOR gate. A few applications of XOR gate are: implementing binary addition in computers and implementation of subtractors, comparators and controlled inverters.

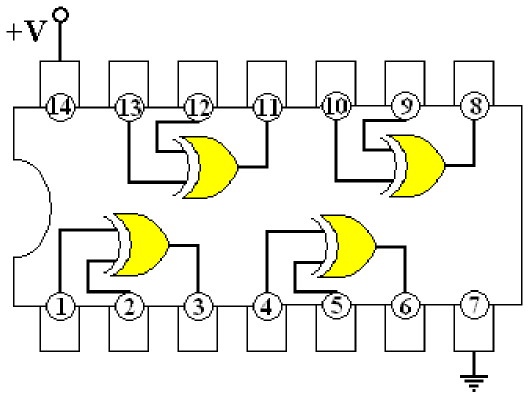


Fig 5(3)- IC 7486 internal pin diagram

PIN NUMBERS	OPERATIONS
1	IN_1A
2	IN_2A
3	OUT_A
4	IN_1B
5	IN_2B
6	OUT_B
7	GROUND
8	OUT_C
9	IN_1C
10	IN_2C
11	OUT_D
12	IN_1D
13	IN_2D
14	Vcc

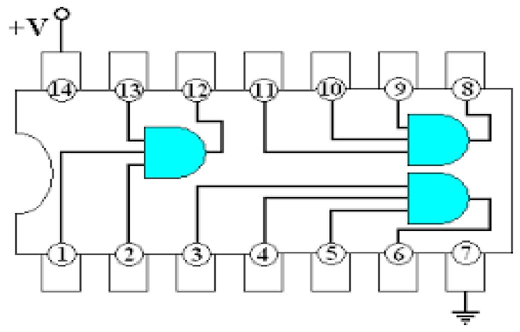
Table 5(2)- pin description of IC7486

THE THREE INPUT AND GATE

IC number used- 7411

It is 14 PIN IC. AND gate follows logical multiplication. AND gate gives output as high (logic 1) only if all inputs are high (logic 1) else the output remains as low (logic 0). The applications of AND logic include enable gate and inhibit gate. Inhibit gates help developers represent a scenario in which the output event occurs if and only if all the input events occur, also an extra conditional event must occur. In other words, inhibit gate is nothing but an AND gate with an additional input (event) AND gate can be used constructed using n channel or p channel MOSFET's. Logical AND gate can be available in TTL logic gate, CMOS logic gate.

PIN NUMBERS	OPERATIONS
1	IN_1A
2	IN_2A
3	IN_1B
4	IN_2B
5	IN_3B
6	OUT_B
7	GROUND
8	OUT_C
9	IN_1C
10	IN_2C
11	IN_3C
12	OUT_A
13	IN_3A
14	Vcc



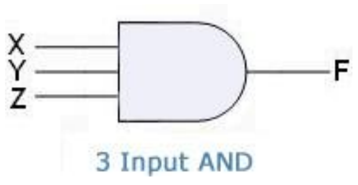


Fig 5(4)- IC 7411 internal pin diagram

Table 5(3)-pin description of IC 7411

THE INVERTOR

IC number used- 7404

It is 14 pin IC. The output of the NOT gate is the inverted/ complemented for of the input thus giving it the name invertor. The NOT gate is also known as "The Digital inverter" or "inverting buffer". The NOT gate can be used to swap between two voltage levels. The functionality of NOT gate is nothing but, the out put is low when the input is high and vice-versa.

PIN NUMBERS	OPERATIONS
1	IN_A
2	OUT_A
3	IN_B
4	OUT_B
5	IN_C
6	OUT_C
7	GROUND
8	OUT_D
9	IN_D
10	OUT_E
11	IN_E
12	OUT_F
13	IN_F
14	Vcc

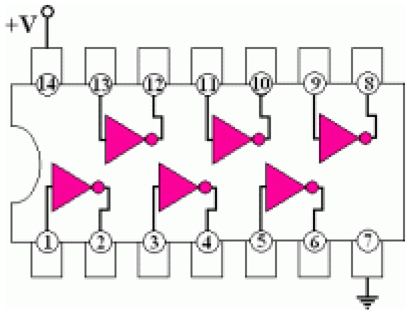


Fig 5(5)- IC 7404 internal pin diagram

Table 5(4)-pin description of IC 7404

The Not gate can be designed using multiple ways such as: Using single NMOS or PMOS transistor coupled with resistor and using BJT(bipolar junction transistor) or TTL(transistor-transistor logic).

THE OR GATE

IC number used- 7432

In case the mentioned IC is not available, it can be replaced with the following ICs 74LS32, 74HC32,74AC32,74LVC32.

The IC used is a 14 pin IC. The output of the or gate is high if any one of the input is high, under all other circumstances the output remains low. It is complementary of AND gate. The OR gate is designed using diodes, transistors and MOSFETs. The OR gate is mainly available in CMOS IC's families.

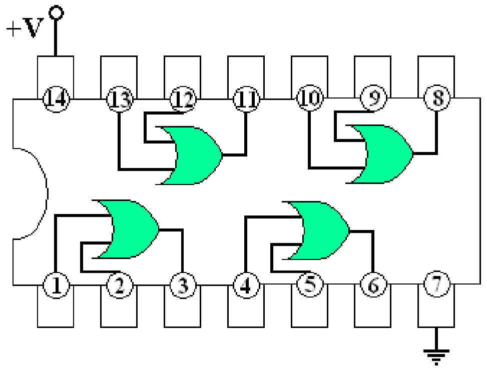


Fig 5(6)- IC 7432 internal diagram

PIN NUMBERS	OPERATIONS
1	IN_1A
2	IN_2A
3	OUT_A
4	IN_1B
5	IN_2B
6	OUT_B
7	GROUND
8	OUT_C
9	IN_1C
10	IN_2C
11	OUT_D
12	IN_1D
13	IN_2D
14	Vcc

Table 5(5)-pin description of IC 74

THE DUAL COLOUR LEDs

A dual color LED (light emitting device) is a device which emits two different colors of light , mainly red and green ,rather than normal single-color LED's. LED's produce different colors depending on photons produced at different wavelengths.

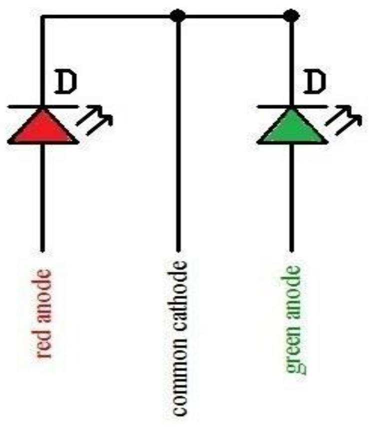


Fig 5(7)- dual color LEDs

PUSH BUTTONS

Push buttons are used in electronics mainly to control the process. This push button is made out of a very hard material using plastic/metal. This device can be easily used by humans to depress and push.

Fig 5(8)- push buttons

RESISTORS:

Resistor used in this device is 10K. Resistors are passive two-terminal electrical component that implements electrical resistance as a circuit element. In electronic circuits, resistors are used to reduce current flow, adjust signal levels, to divide voltages, bias active elements, and terminate transmission lines, among other uses. Resistors are what are called "Passive Devices", that is they contain no source of power or amplification but only attenuate or reduce the voltage or current signal passing through them. This attenuation results in electrical energy being lost in the form of heat as the resistor resists the flow of electrons through it. Color coding is used in resistors to calculate the magnitude of the resistance in ohms.

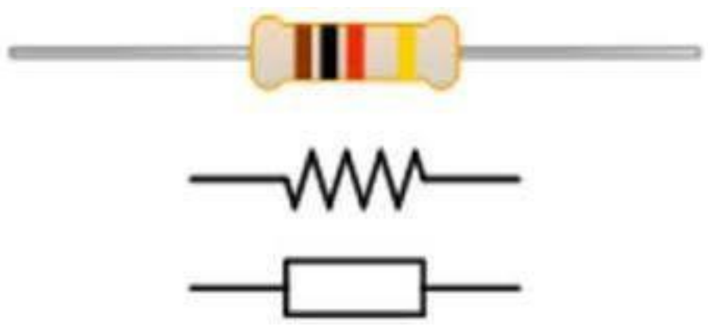


Fig 5(9): Resistor block and circuit diagram

BREADBOARD:

A breadboard is a solderless device for temporary prototype with electronics and test circuit designs. Most electronic components in electronic circuits can be interconnected by inserting their leads or terminals into the holes.

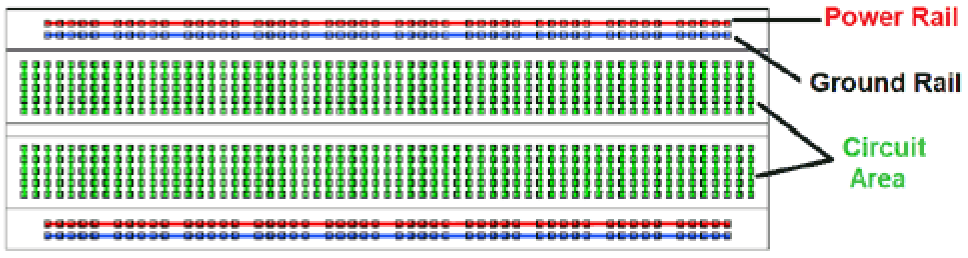


Fig-5(10): bread board

The holes covered by the green line are shorted. A pair of possitive and negitive supply lines are provided for easy connections. A bread board hepls in hassel free connection of curcuit elements.

THE POWER SUPPLY:

A 9-volt battery is used to power up the circuit. The battery has both terminals in a snap connector on one end. The smaller circular (male) terminal is positive, and the larger hexagonal (female) terminal is the negative contact. The battery used here is an alkaline battery.. The voltage is near 1.25 volts at 50% discharge point.

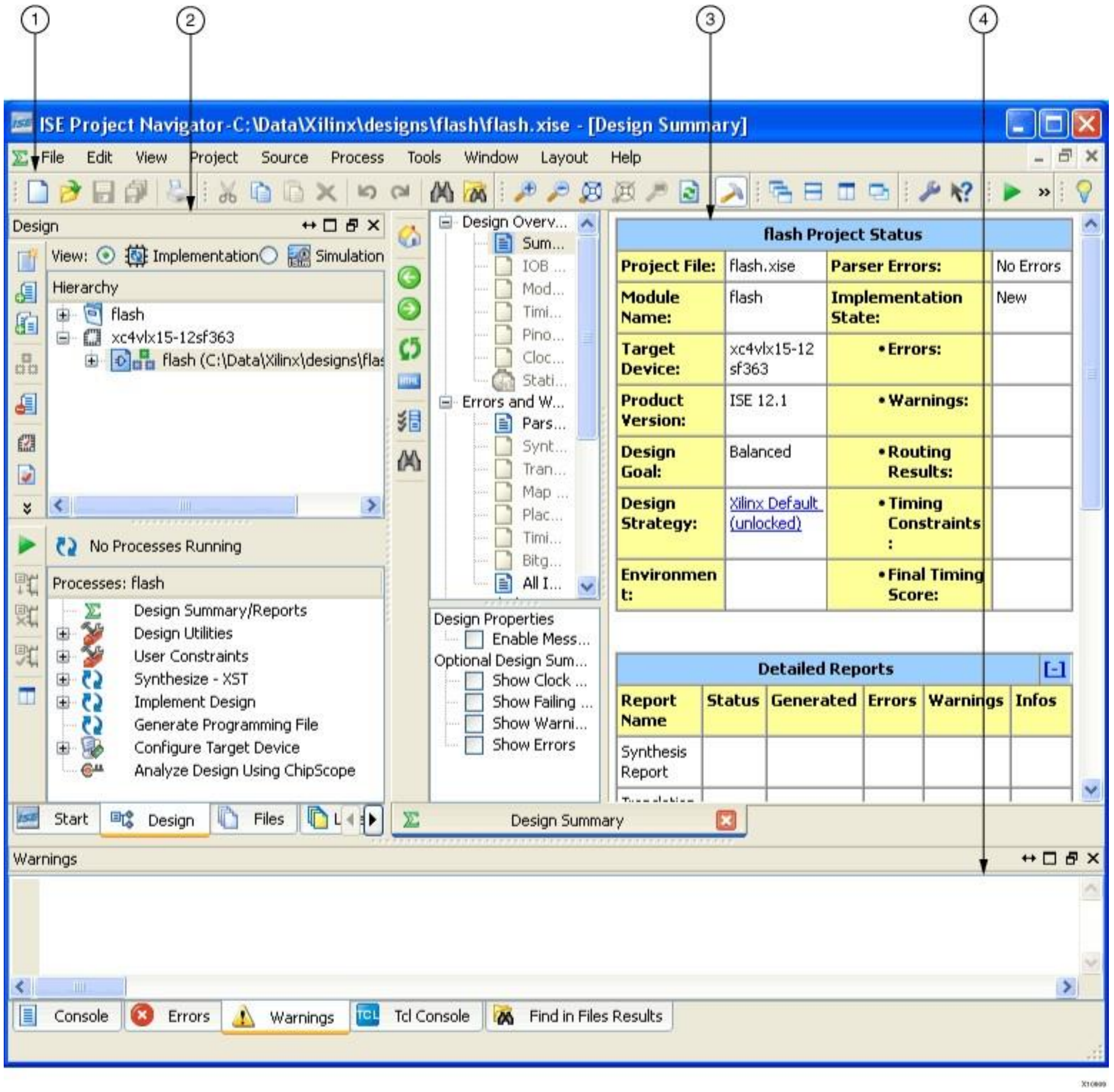
Fig-5(11): A nine-volt battery

SOFTWARE USED: XILINX ISE 12.2

Xilinx ISE (**I**ntegrated **S**ynthesis **E**nvironment) 12.2 version software controls the design flow of all aspects through the project navigator interface. The access to all of the design entry and design implementation tools are easily available and the access to all the files and documents associated with the project is easy.

PROJECT NAVIGATOR INTERFACE

Project Navigator interface by default is segregated into four panel sub-windows, as seen in the figure down below. Left side of the project navigator consists of the start, design, files and libraries panels, these include display and access to the source files in the project as well as access to running processes for the currently selected source. The start panel gives quick access to opening projects as well as frequently get the reference material, documentation and tutorials. Bottom of the project navigator consists of the console, errors and warnings panels, these panels display status messages, errors, and warnings. Right side of the project navigator is a multi-document interface (MDI) window referred to as the workspace which enables you to view design reports, text files, schematics, and simulation waveforms. Each window can be rescaled, undocked from project navigator, moved to a new location within the main project navigator window, tiled, layered, or closed. The view - panels menu commands to open or close panels. The layout - load default layout to restore the default window layout. These windows are discussed in more detail in the following sections.



flash Project Status			
Project File:	flash.xise	Parser Errors:	No Errors
Module Name:	flash	Implementation State:	New
Target Device:	xc4vlx15-12 sf363	• Errors:	
Product Version:	ISE 12.1	• Warnings:	
Design Goal:	Balanced	• Routing Results:	
Design Strategy:	Xilinx Default (unlocked)	• Timing Constraints:	
Environment:		• Final Timing Score:	

Report Name	Status	Generated	Errors	Warnings	Infos
Synthesis Report					

Fig 5(12)- Project Navigator

DESIGN PANEL

The design panel gives access to view, hierarchy, and processes panes.

5.13A VIEW PANEL

View pane radio buttons allows us to view the source modules associated with the implementation or simulation design view in the hierarchy pane. If it is select simulation, selection of a simulation phase from the drop-down list is necessary.

5.13B HIERARCHY PANE

Hierarchy pane shows the project name, the target device, user documents and design source files associated with the selected design view. The view pane at the top of the Design panel shows only those source files associated with the selected design view such as implementation or simulation. Every hierarchy pane file has an associated icon. The icon indicates the file type (HDL file, schematic, core, or text file for example). For a complete list of possible source types and their associated icons, check out the "Source File Types" topic in the ISE help. Go to project navigator, select Help > Help Topics to view the ISE help. If a file contains lower levels of hierarchy, the icon has a plus sign (+) to the left of the name. Expand the hierarchy is possible by clicking the plus sign (+). You can open a file for editing by double-clicking on the filename.

C PROCESSES PANE

Processes pane is context sensitive while it changes based upon the source type selected in the sources pane and the top-level source in the project. Processes pane can run the functions necessary to define, run, and analyze your design. It provides access to the following functions:

I. DESIGN SUMMARY/REPORTS

It enables us to access the design reports, messages, and summary of results data. Message filtering can also be performed.

II. DESIGN UTILITIES

It enables us to access the symbol generation, instantiation templates, viewing command line history, and simulation library compilation.

III. USER CONSTRAINTS

It enables us to access the editing location and timing constraints.

Synthesis gives access to check the syntax, synthesis, view RTL or technology schematic, and synthesis reports. Available processes vary depending on the synthesis tools you use.

IV. IMPLEMENT DESIGN

It gives access to implementation tools and post-implementation analysis tools.

V. GENERATE PROGRAMMING FILE

It gives access to bitstream generation.

VI. CONFIGURE TARGET DEVICE

It gives access to configuration tools for creating programming files and programming the device. The Processes pane has dependency management technology where the tools keep track of which processes have been run and which processes need to be run. Graphical status indicators display the state of the flow at any given time. When selecting a process in the flow, the software automatically runs the processes necessary to get to the desired step for example, when the Implement Design process runs, project navigator also runs the synthesis process because implementation is dependent on up-to-date synthesis results. To view a running log of command line arguments used on the current project, expand design utilities and select view command line log file.

FILES PANEL

Files panel gives a plain changeable list of all the source files in the project. They can be sorted by any of the columns in the view. Properties for each file can be viewed and modified by right-clicking on the file and selecting Source Properties.

LIBRARIES PANEL

Libraries panel helps us to manage HDL libraries and their associated HDL source files. Creating, viewing and editing libraries with their associated sources can be done.

CONSOLE PANEL

Console gives all standard output from processes run from project navigator. It displays errors, warnings, and information messages. Errors are signified by a red X next to the message; while warnings have a yellow exclamation mark (!).

ERRORS PANEL

Errors panel shows only error messages. Other console messages are filtered out.

WARNING PANEL

Warnings panel shows only warning messages. Other console messages are filtered out.

ERROR NAVIGATION TO SOURCE

Navigation from a synthesis error or warning message in the console, errors, or warnings panel to the location of the error in a source HDL file can be done. To implement it, select the error or warning message, right-click the mouse and select go to source from the right-click menu. The HDL source file opens, and the cursor moves to the line with the error.

ERROR NAVIGATION ANSWER RECORD

Navigation from an error or warning message in the console, errors, or warnings panel to relevant answer records on the support page of the Xilinx website can be implemented. To navigate to the answer record, select the error or warning message, right-click the mouse, and select go to answer record from the right-click menu. The default web browser pops up and shows all answer records applicable to this message.

WORKSPACE

The workspace is where design editors, viewers, and analysis tools open. These include ISE text editor, schematic editor, constraint editor, design summary/report viewer, RTL, technology viewers and timing analyzer.

DESIGN SUMMARY/REPORT VIEWER

The design summary gives a summary of key design data and enables the access to all of the messages and detailed reports from the synthesis and implementation tools. The summary lists high-level information about the project, including overview information, a device utilization summary, performance data gathered from the place and route (PAR) report, constraints information, and summary information from all reports with links to the individual reports. System settings report link gives information on environment variables and tool settings used during the design implementation.

USER INTERFACE

The main interface of the ISE is the Project Navigator, which incorporates the design hierarchy- Sources, a source code editor-Workplace, an output console-Transcript and a processes tree-Processes.

Design hierarchy consists of design files- modules, whose reliance's are translated by the ISE and shown as a tree structure. For single-chip designs there may be one main module where the other modules are included by the main module, similar to the main( ) subroutine in C++ language programs. Design constraints are stated in modules, which include pin configuration and mapping.

Processes hierarchy describes the operations that the ISE will perform on the currently active module. The hierarchy includes compilation functions, their dependency functions, and other utilities. The window also denotes issues or errors that arise with each function.

Transcript window provides status of currently running operations, and informs engineers on design issues. Such issues may be filtered to show warnings, errors, or both.

SIMULATION

System-level testing may be performed with ISIM or the ModelSim logic simulator where these test programs must also be written in HDL languages. Test bench programs may include simulated input signal waveforms or which observes and verifies the outputs of the device under test.

ModelSim or ISIM may be used to perform the following simulations: To execute logical verification and to ensure the module gives expected results. To carry out behavioural verification and to verify logical and timing issues. For post-place & route simulation and to verify behaviour after placement of the module within the reconfigurable logic of the FPGA.

SYNTHESIS

It is used to convert higher-level abstraction of the design into actual components at the gate and flip flops levels whose output is the netlist, which is a list of gates and a list of their interconnections. As the synthesis tool converts the design description into hardware, it is also called as “design compiler” or “silicon compiler”.

HARDWARE DESCRIPTION LANGUAGE

HDL is a textual method of documenting the circuits and feeding them into the simulators in the textual form, as opposed to the graphic form. It leads to a top-down design methodology (specified and tested at a high level) and they are designed to be technology independent.

It can describe a digital system at three different levels- behavioural (functional description), data flow (logic equations) and structural (in terms of subcomponents).

It is a must have tool for modern engineers. Two popular HDLs are Verilog and VHDL. Verilog was a proprietary language in 1984, and was later owned and opened up by Cadence, in order to create a vendor-independent language specification, and to prevent the industry from shifting to VHDL.

VHDL

VHDL (VHSIC Hardware Description Language) is very fast emerging as one of the most important electronic design languages in both the military and commercial electronic design areas.

It was developed by the government (Designed and sponsored by US department of defence), and became IEEE standard in 1987.

It is based on “ADA” language which is strongly type casted and is case insensitive.

It is for large and complex systems (Difficult to learn, but more powerful) because it is in a structured way.

VERILOG

Verilog HDL is a Hardware Description Language. It is a language used for reporting a digital system like a network switch, microprocessor, memory or a flip−flop. It means, by using HDL we can report any digital hardware at any level. Designs, which are reported in HDL are independent of technology, very easy for designing and debugging, and are normally more useful than schematics, particularly for large circuits.

It was developed commercially (Designed by Gateway design system corporation), became IEEE standard in 1995.

It is based on “C” language which is mildly type casted and is case-sensitive.

It comes with less syntax and fewer constructs (Easier to learn, and less powerful) because it is a simpler way.

HARDWARE DESCRIPTION LANGUAGE

HDL is a textual method of documenting the circuits and feeding them into the simulators in the textual form, as opposed to the graphic form. It leads to a top-down design methodology (specified and tested at a high level) and they are designed to be technology independent.

It can describe a digital system at three different levels- behavioural (functional description), data flow (logic equations) and structural (in terms of subcomponents).

It is a must have tool for modern engineers. Two popular HDLs are Verilog and VHDL. Verilog was a proprietary language in 1984, and was later owned and opened up by Cadence, in order to create a vendor-independent language specification, and to prevent the industry from shifting to VHDL.

CHAPTER 06

RESULT AND DISCUSSION

The project was completed successfully. The goal of the project was to design a 3*3 Tic Tac Toe game using Verilog programming to understand it in a deeper level. The game rule stated that if a player gets any of the horizontal or vertical or even the diagonal lines full then the player is declared as winner, as the strategies of each player is different from the other.

This project aims to make the game more interesting and less predictable.

Our project was also being implemented using Verilog coding , this program is being played optimally and there are higher winning chances of the player. If both the players play optimally then it is destined that no one will lose. As it doesn't matter whether you play first or second.

The logic of the game has been drafted successfully using the software Easy EDA. And the hardware part for all the blocks has been generated successfully.

As we are satisfied to show that it is only limited in board game and a code and this could be worked out on multiple different complex games such as Ludo ,connect 4 , chess, etc.

INPUT BLOCK

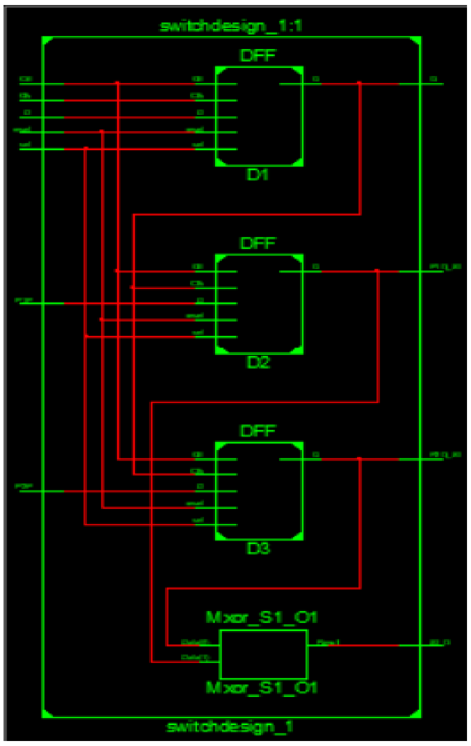
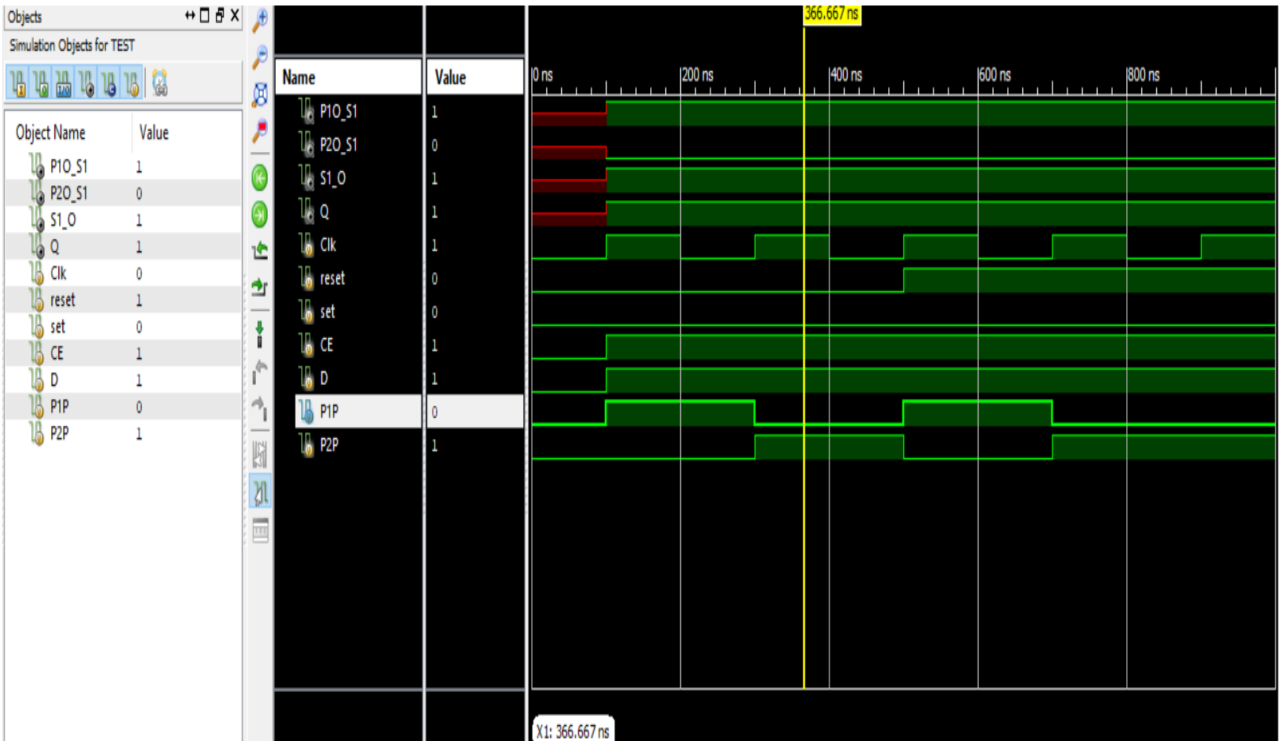


Fig 6(1)-RTL schematic of input block

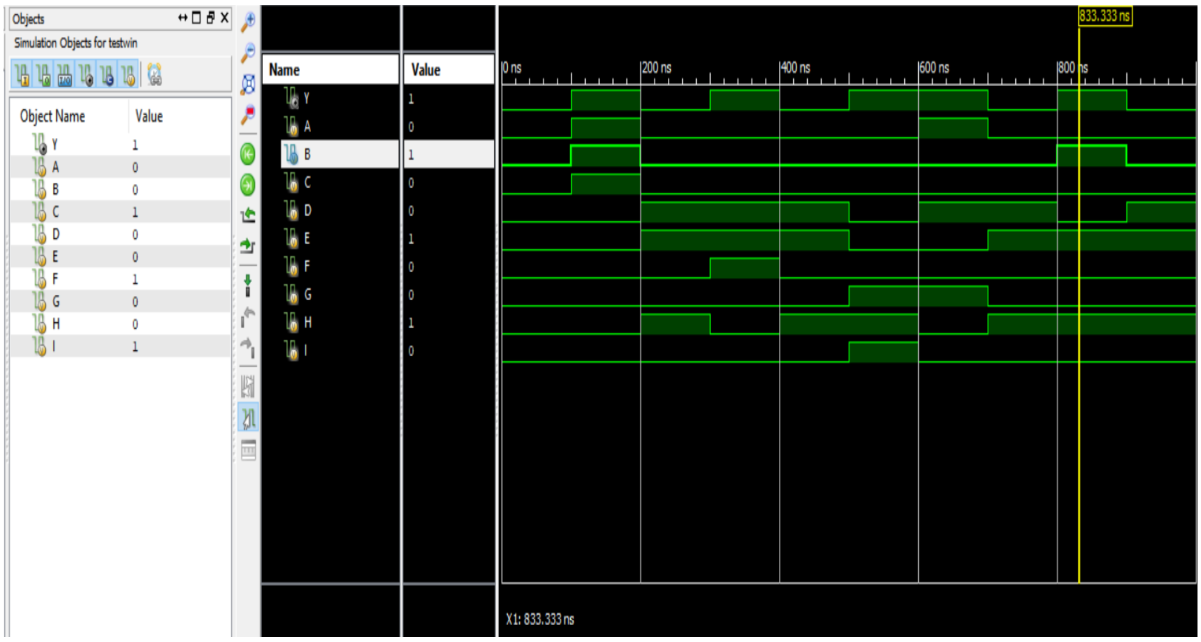


Object Name	Value
P1O_S1	1
P2O_S1	0
S1_O	1
Q	1
Clk	0
reset	1
set	0
CE	1
D	1
P1P	0
P2P	1

Name	Value
P1O_S1	1
P2O_S1	0
S1_O	1
Q	1
Clk	1
reset	0
set	0
CE	1
D	1
P1P	0
P2P	1

Fig 6(2)-simulated behavioral model of input block

PROCESSING LOGIC



Objects

Simulation Objects for testwin

Object Name	Value
Y	1
A	0
B	0
C	1
D	0
E	0
F	1
G	0
H	0
I	1

Name	Value
Y	1
A	0
B	1
C	0
D	0
E	1
F	0
G	0
H	1
I	0

Fig 6(3)-simulated behavioral model of processing logic

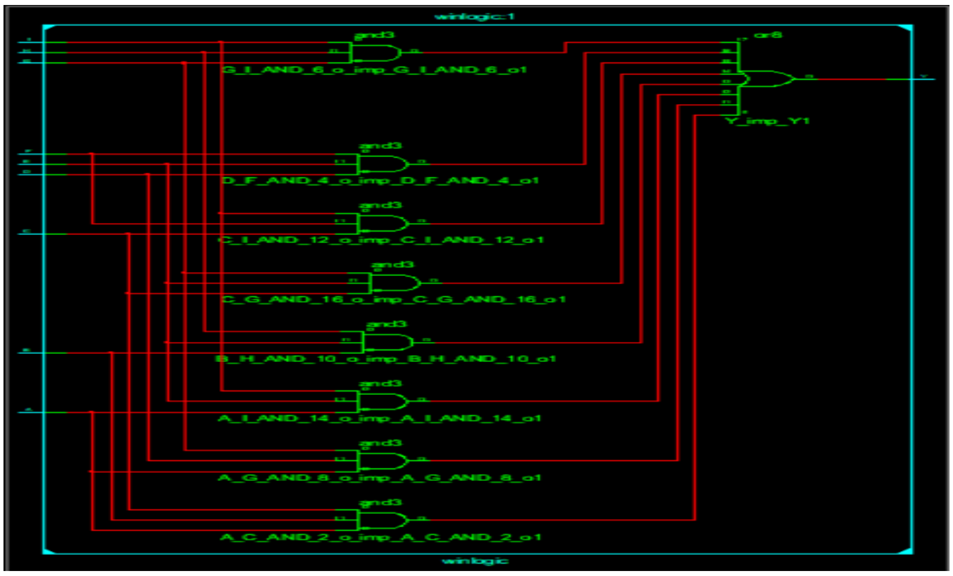
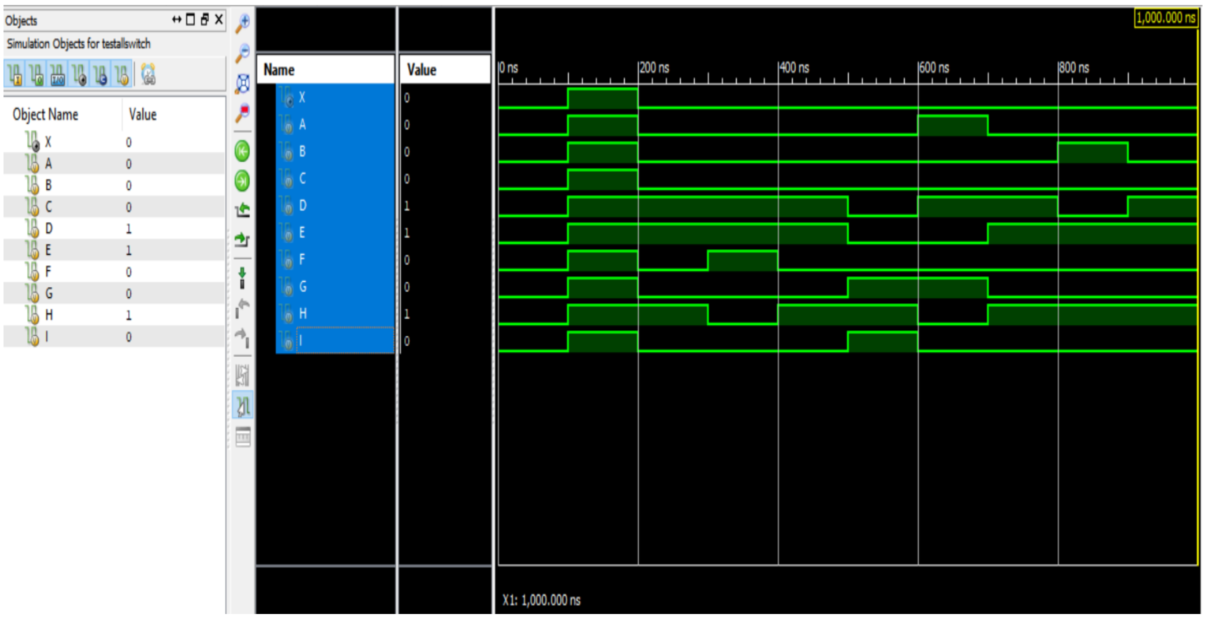

winlogic:1
and3
G_I_AND_6_o_imp_G_I_AND_6_o1
or8
Y_imp_Y1
D_F_AND_4_o_imp_D_F_AND_4_o1
C_I_AND_12_o_imp_C_I_AND_12_o1
C_G_AND_16_o_imp_C_G_AND_16_o1
B_H_AND_10_o_imp_B_H_AND_10_o1
A_I_AND_14_o_imp_A_I_AND_14_o1
A_G_AND_8_o_imp_A_G_AND_8_o1
A_C_AND_2_o_imp_A_C_AND_2_o1
winlogic

Fig 6(4)-RTL schematic of processing logic

ALL SWITCHES PRESSED/RESET



Object Name	Value
X	0
A	0
B	0
C	0
D	1
E	1
F	0
G	0
H	1
I	0

Fig 6(5)-simulated behavioral model of reset logic

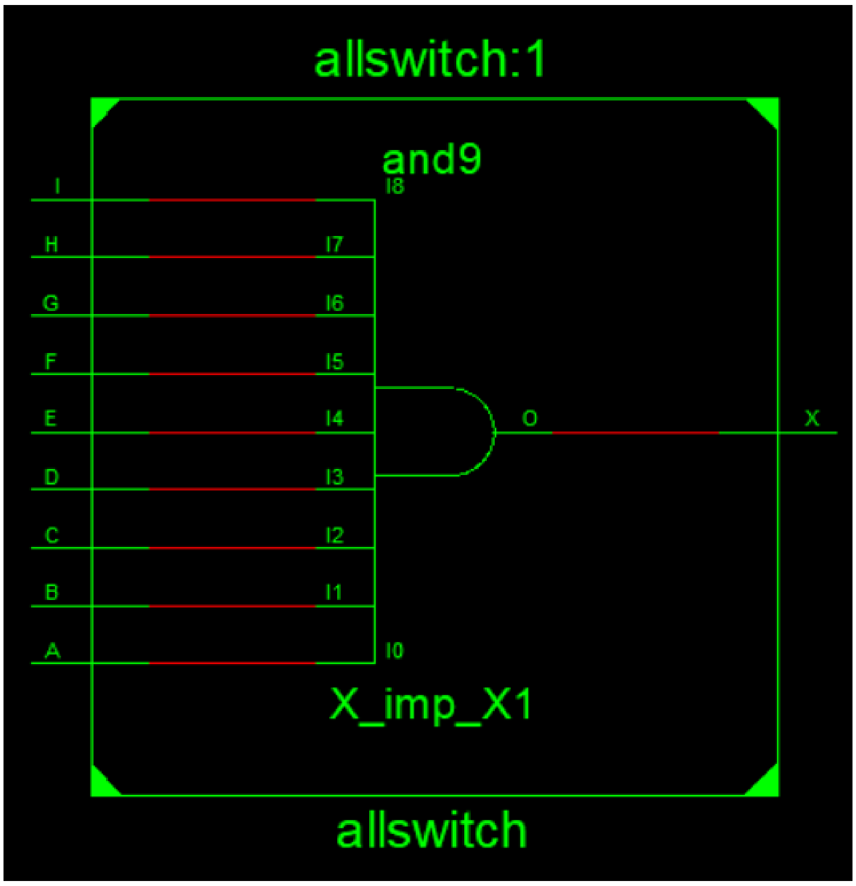


Fig 6(6)-RTL schematic of reset logic

UMPIRE CIRCUIT

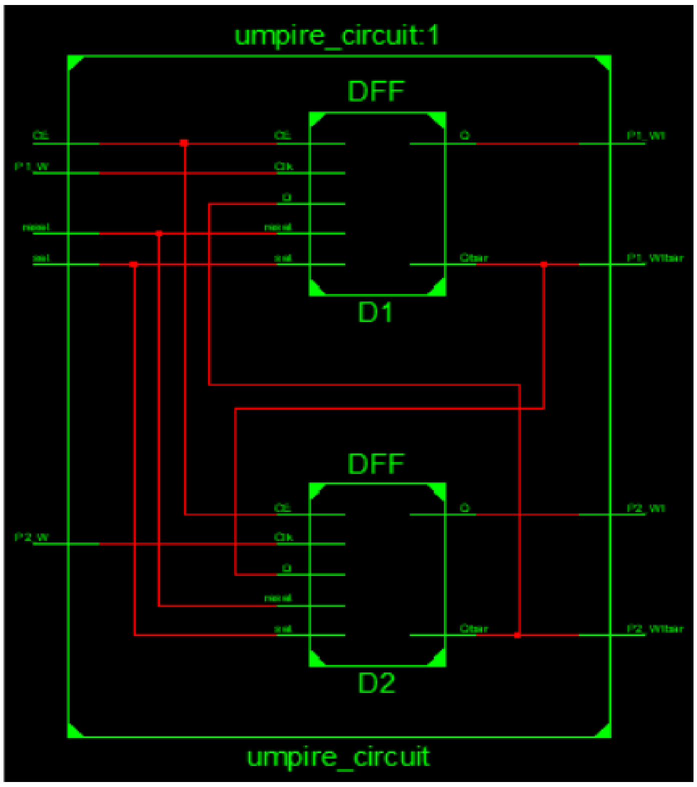


Fig 6(7)-RTL schematic of umpire circuit

Fig 6(8)-simulated behavioral model of umpire circuit

CHAPTER 07

APPLICATONS AND ADVANTAGES

APPLICATIONS:-

- Simple logic gate IC's are used, which made us to understand it's functionality very easily.
- It is power efficient as it can be powered up using a 9V battery.
- Easy EDA software made us to generate hardware components very easily.
- Used Verilog to implement Basic sequential circuits and combinational circuits
- Simulating the circuits using Xilinx software was easy ,as we studied about software in our academic course.

ADVANTAGES:-

- Optimal game.
- Very efficient in terms of time and it is a simple game.
- Helps in strategic thinking.
- Helps in improving concentration.

DRAW-BACKS:-

- Does not contain higher levels of game play.

CHAPTER 08

FUTURE SCOPE

Our project could be implemented in the future by changing and modifying few things such as:

1) One change can be done by adding the player's fingerprint.

2) Adding user profiles and scores.

3) Allow the players to choose the mode of the game.

CHAPTER 09

CONCLUSION

The Tic Tac Toe game is very familiar to all age groups. Intelligence can be a property of any purpose driven decision maker. This is the basic idea that the game follows. Here in this project, an algorithm of playing Tic Tac Toe has been presented and simulated that works in an efficient way.

Overall, the system works without any errors. Different ways of winning the game was also present in the results obtained.

REFERENCES

[1] Anurag Bhatt, Pratul Varshney and Kalyanmoy Dev, “**Evolution of No-loss Strategies for the game Tic Tac Toe”,** KanGAL report number 2007002.

[2] D.W.Bliss, P.A.Parker, A.R. Margetts , **“Simulation transmission and reception for improved wireless network performance”** , Statistical Signal Processing 2007 IEEE/SP 14th Workshop on , pp. 478-482,2007.

[3] D. Fogel, **“Using evolutionary programming to create neural network that are capable of playing tic-tac-toe,** “in Proceedings of ICNN93, pp.875-880, 1993.

[4] Kevin Crowley, Robert S. Siegler (1993), **"Flexible Strategy Use in Young Children's Tic-Tac-Toe".** *Cognitive Science*. **17** (4): 531–561.

[5]. Solomon W. Golomb and Alfred W. Hales (2002), "Hypercube Tic-tac-toe", MSRI Publications.

[6]. Sandy Douglas (1952), "OXO aka - Noughts and Crosses, the first video game" D.S.Cohen

APPENDIX – A

VERILOG SUMMARY

The Verilog language is a hardware description language for modelling digital circuits that can range from the simple connection of gates to complex systems.

Verilog is case sensitive. An identifier must start with a letter or underscore and must not start with a number or a dollar($) sign.

1. MODULE:

A module represents a logical component in a digital system. Each module has an interface for specifying the signals for communication with other modules.

```
module module_name

(input port_name_list,

output port_name_list,

inout port_name_list);

statements;

endmodule
```

2. DATAFLOW MODEL:

The dataflow model specifies the circuit in a form similar to Boolean algebra. Hence, this model is best suited for describing circuit when a given set of Boolean equations.

Continuous assignment: the assign statement is used to provide continuous assignment of values onto nets. The **assign** statement is evaluated anytime when any of its inputs changes, and the result of the evaluation is propagated to the output net.

Syntax:

```
assign net_identifier = expression;
assign net_identifier <= expression;
assign net_identifier = function;
```

3. STRUCTURAL MODEL:

The structural model allows manuel connection of primitive gates and module components together using nets to build larger modules.

Structural architecture first defines its component and then proceeds to describe the structure in which the components connect to give the circuit.

Example: (full adder using two half adders)

```
module ha(x,y,sm,cy);    // half adder module
input x,y;
output sm,cy;
assign sm=x^y;
assign cy=x&y;
endmodule                      // end of module

module or_m (p,q,r);    // OR logic module
input p,q;
output r;
assign r=p | q;
endmodule                      // end of module

module fa(a,b,ci,s,co);// full adder module
input a,b,ci;
output s,co;
wire t1,t2,t3;
ha i1(a,b,t1,t2);    // Instantiate two Half Adders: HA1, HA2
ha i2(t1,ci,s,t3);    // The ports are matched by position
or_m i3(t2,t3,co);
endmodule                      // end of module
```

4. CONCURRENT OR PARALLEL STATEMENTS:

```
assign [#delay] signal_name = expression;
assign signal_name = condition ? expression_T : expression_F;
initial
begin
        sequential-statements
end
always @(sensitivity-list);
begin
       sequential-statements
end
```

5. SEQUENTIAL STATEMENTS(Procedural Statements)

a. Blocking statements

```
always @(sensitivity_list)
begin
blocking statements using the = operator
end
```

b. Non-blocking statements

```
always @(sensitivity_list)
begin
non-blocking statements using the <= operator
end
```

c.

```
if(condition)
sequential statements
else if (condition)
  Sequential statements}
[else sequential statements]
```

d.

```
if -then-else
if (condition)
   statement1;
else if(condition)
   statement2;
else
   statement3;
```

6. PREDEFINED DATA TYPES

Unlike VHDL, all data types are predefined by the verilog language and not by the user. Some of the popular predefined types are:

nets	connections between hardware elements (declared with keywords such as **wire)**
variables	data storage elements that can retain values (declared with keywords such as **reg)**
integer	an integer is a variable data type (declared with the keyword **integer)**
vectors	wire or reg data types can be declared as vectors(multiple bits)

7. VERILOG OPERATORS BY DECREASING PRECEDENCE:

a. Bitwise Operators

Character	Operation performed
~	Invert each bit
&	And each bit
|	Or each bit
^	Xor each bit
^~ or ~^	Xnor each bit

b. Reduction Operators

Character	Operation performed
&	And all bits
~&	Nand all bits
|	Or all bits
~|	Nor all bits
^	Xor all bits
^~ or ~^	Xnor all bits

c. Relational Operators

Character	Operation performed
>	Greater than
<	Smaller than
>=	Greater than or equal
<=	Smaller than or equal
==	Equality
!=	Inequality
===	Case equality
!===	Case inequality

d. Operators Precedence

Operators precedence
+, -, !, ~ (Unary)
+,- (Binary)
<<, >>
<,>,<=,>=
==, !=
&
^, ^~ or ~^
|
&&
||
?:

8. DECLARATION EXAMPLES

```
input A,B,C;
output d;
wire [7:0] A;
reg C;
reg[7:0] A;
reg signed [7:0] A=8'hA5;
parameter constant_name= contanst_value;
```

9. BUILT IN PRIMITIVES

Built in primitive types	Primitives
n-input gates	and, nand, nor, or, xnor, xor
n-output gates	buff, not
Three-state gates	bufif0,bufif1,botif0,notif1

APPENDIX – B

VERILOG CODE

1. BLOCK ONE: THE INPUT BLOCK

```
module DFF(Clk,CE,reset,D,set,Q);
input Clk,CE,reset,D,set;
output reg Q;
 always @(posedge(Clk))
    begin
        if(Clk == 1)
        begin
            if (CE == 1)
                Q = D;
            else if(set == 1)
             Q = 1;
         else if (reset == 1)
                 Q = 0;
      end
    end
endmodule

module profile_generator( P1P,P2P,S1_O ,S2_O ,S3_O ,S4_O ,S5_O
,S6_O ,S7_O ,S8_O ,S9_O );
inout S1_O ,S2_O ,S3_O ,S4_O ,S5_O ,S6_O ,S7_O ,S8_O ,S9_O;
output P1P,P2P;
wire x1,x2,x3;
assign x1 = ((S1_O^S2_O)^(S3_O^S4_O));
assign x2 = ((S5_O^S6_O)^(S7_O^S8_O));
assign x3 = (x1^x2);
assign  P2P  =  (x3^S9_O);
assign P1P = (~(x3^S9_O));
```

```
endmodule

module
switchdesign_1(Clk,reset,set,CE,D,P1P,P2P,P1O_S1,P2O_S1,S1_O,Q);
input Clk,P1P,P2P,reset,set,CE,D;
output P1O_S1,P2O_S1;
inout S1_O,Q;
DFF D1 (Clk,CE,reset,D,set,Q);
DFF D2 (Q,CE,reset,P1P,set,P1O_S1);
DFF D3 (Q,CE,reset,P2P,set,P2O_S1);
assign S1_O = P1O_S1 ^ P2O_S1;
endmodule

module
switchdesign_2(Clk,reset,set,CE,D,P1P,P2P,P1O_S2,P2O_S2,S2_O,Q);
input Clk,P1P,P2P,reset,set,CE,D;
output P1O_S2,P2O_S2;
inout S2_O,Q;
DFF D1 (Clk,CE,reset,D,set,Q);
DFF D2 (Q,CE,reset,P1P,set,P1O_S2);
DFF D3 (Q,CE,reset,P2P,set,P2O_S2);
assign S2_O = P1O_S2^P2O_S2;
endmodule

  module
switchdesign_3(Clk,reset,set,CE,D,P1P,P2P,P1O_S3,P2O_S3,S3_O,Q);
input Clk,P1P,P2P,reset,set,CE,D;
output P1O_S3,P2O_S3;
inout S3_O,Q;
DFF D1 (Clk,CE,reset,D,set,Q);
DFF D2 (Q,CE,reset,P1P,set,P1O_S3);
DFF D3 (Q,CE,reset,P2P,set,P2O_S3);
assign S3_O = P1O_S3^P2O_S3;
endmodule
```

```
module
switchdesign_4(Clk,reset,set,CE,D,P1P,P2P,P1O_S4,P2O_S4,S4_O,Q);
input Clk,P1P,P2P,reset,set,CE,D;
output P1O_S4,P2O_S4;
inout S4_O,Q;
DFF D1 (Clk,CE,reset,D,set,Q);
DFF D2 (Q,CE,reset,P1P,set,P1O_S4);
DFF D3 (Q,CE,reset,P2P,set,P2O_S4);
assign S4_O = P1O_S4^P2O_S4;
endmodule

module
switchdesign_5(Clk,reset,set,CE,D,P1P,P2P,P1O_S5,P2O_S5,S5_O,Q);
input Clk,P1P,P2P,reset,set,CE,D;
output P1O_S5,P2O_S5;
inout S5_O,Q;
DFF D1 (Clk,CE,reset,D,set,Q);
DFF D2 (Q,CE,reset,P1P,set,P1O_S5);
DFF D3 (Q,CE,reset,P2P,set,P2O_S5);
assign S5_O = P1O_S5^P2O_S5;
endmodule

module
switchdesign_6(Clk,reset,set,CE,D,P1P,P2P,P1O_S6,P2O_S6,S6_O,Q);
input Clk,P1P,P2P,reset,set,CE,D;
output P1O_S6,P2O_S6;
inout S6_O,Q;
DFF D1 (Clk,CE,reset,D,set,Q);
DFF D2 (Q,CE,reset,P1P,set,P1O_S6);
DFF D3 (Q,CE,reset,P2P,set,P2O_S6);
assign S6_O = P1O_S6^P2O_S6;
endmodule
```

```
module
switchdesign_7(Clk,reset,set,CE,D,P1P,P2P,P1O_S7,P2O_S7,S7_O,Q);
input Clk,P1P,P2P,reset,set,CE,D;
output P1O_S7,P2O_S7;
inout S7_O,Q;
DFF D1 (Clk,CE,reset,D,set,Q);
DFF D2 (Q,CE,reset,P1P,set,P1O_S7);
DFF D3 (Q,CE,reset,P2P,set,P2O_S7);
assign S7_O = P1O_S7^P2O_S7;
endmodule

module
switchdesign_8(Clk,reset,set,CE,D,P1P,P2P,P1O_S8,P2O_S8,S8_O,Q);
input Clk,P1P,P2P,reset,set,CE,D;
output P1O_S8,P2O_S8;
inout S8_O,Q;
DFF D1 (Clk,CE,reset,D,set,Q);
DFF D2 (Q,CE,reset,P1P,set,P1O_S8);
DFF D3 (Q,CE,reset,P2P,set,P2O_S8);
assign S8_O = P1O_S8^P2O_S8;
endmodule

module
switchdesign_9(Clk,reset,set,CE,D,P1P,P2P,P1O_S9,P2O_S9,S9_O,Q);
input Clk,P1P,P2P,reset,set,CE,D;
output P1O_S9,P2O_S9;
inout S9_O,Q;
DFF D1 (Clk,CE,reset,D,set,Q);
DFF D2 (Q,CE,reset,P1P,set,P1O_S9);
DFF D3 (Q,CE,reset,P2P,set,P2O_S9);
assign S9_O = P1O_S9^P2O_S9;
endmodule
```

TEST BENCH FOR BLOCK INPUT(INPUT BLOCK)

2. BLOCK TWO: PROCESSING LOGIC

```
module winlogic_player_1( Y_1,A,B,C,D,E,F,G,H,I);
output Y_1;
input A,B,C,D,E,F,G,H,I;
wire x1,x2,x3, x4,x5,x6;
assign x1=((A&B&C)|(D&E&F));
assign x2=((G&H&I)|(A&D&G));
assign x3=((B&E&H)|(C&F&I));
assign x4=((A&E&I)|(G&E&C));
assign x5=(x1|x2);
assign x6=(x3|x4);
assign Y_1=(x5|x6);
endmodule

module winlogic_player_2( Y_2,A,B,C,D,E,F,G,H,I);
output Y_2;
input A,B,C,D,E,F,G,H,I;
wire x1,x2,x3, x4,x5,x6;
assign x1=((A&B&C)|(D&E&F));
assign x2=((G&H&I)|(A&D&G));
assign x3=((B&E&H)|(C&F&I));
assign x4=((A&E&I)|(G&E&C));
assign x5=(x1|x2);
assign x6=(x3|x4);
assign Y_2=(x5|x6);
endmodule
```

TEST BENCH FOR BLOCK TWO(PROCESSING LOGIC)

```
 A=1 ; B=1 ; C=1 ; D=1 ; E=1 ; F=1 ; G=1 ; H = 1; I=1 ;  #100;
 A=0 ; B=0 ; C=0 ; D=1 ; E=1 ; F=0 ; G=0 ; H =1 ; I=0 ;  #100;
A=0 ; B=0 ; C=0 ; D=1 ; E=1 ; F=1 ; G=0 ; H =0 ; I=0 ;  #100;
 A=0 ; B=0 ; C=0 ; D=1 ; E=1 ; F=0 ; G=0 ; H =1 ; I=0 ;  #100;
 A=0 ; B=0 ; C=0 ; D=0 ; E=0 ; F=0 ; G=1 ; H =1 ; I=1 ;  #100;
 A=1 ; B=0 ; C=0 ; D=1 ; E=0 ; F=0 ; G=1 ; H =0 ; I=0 ;  #100;
A=0 ; B=0 ; C=0 ; D=1 ; E=1 ; F=0 ; G=0 ; H =1 ; I=0 ;  #100;
 A=0 ; B=1 ; C=0 ; D=0 ; E=1 ; F=0 ; G=0 ; H =1 ; I=0 ;  #100;
 A=0 ; B=0 ; C=0 ; D=1 ; E=1 ; F=0 ; G=0 ; H =1 ; I=0 ;  #100;
```

3. BLOCK THREE: RESET

```
module reset( X,A,B,C,D,E,F,G,H,I);
output X;
input A,B,C,D,E,F,G,H,I;
wire x1,x2,x3;
assign x1=((A&B)&(C&D));
assign x2=((E&F)&(G&H));
assign x3=(x1&x2);
assign X= x3&I;
endmodule
```

TEST BENCH FOR BLOCK THREE(RESET)

```
A=1 ; B=1 ; C=1 ; D=1 ; E=1 ; F=1 ; G=1 ; H = 1; I=1 ;  #100;
A=0 ; B=0 ; C=0 ; D=1 ; E=1 ; F=0 ; G=0 ; H =1 ; I=0 ;  #100;
A=0 ; B=0 ; C=0 ; D=1 ; E=1 ; F=0 ; G=0 ; H =1 ; I=0 ;  #100;
A=0 ; B=0 ; C=0 ; D=0 ; E=0 ; F=0 ; G=1 ; H =1 ; I=1 ;  #100;
A=1 ; B=0 ; C=0 ; D=1 ; E=0 ; F=0 ; G=1 ; H =0 ; I=0 ;  #100;
A=0 ; B=0 ; C=0 ; D=1 ; E=1 ; F=0 ; G=0 ; H =1 ; I=0 ;  #100;
A=0 ; B=1 ; C=0 ; D=0 ; E=1 ; F=0 ; G=0 ; H =1 ; I=0 ;  #100;
A=0 ; B=0 ; C=0 ; D=1 ; E=1 ; F=0 ; G=0 ; H =1 ; I=0 ;  #100;
A=1 ; B=0 ; C=0 ; D=1 ; E=1 ; F=0 ; G=0 ; H =1 ; I=0 ;  #100;
A=0 ; B=1 ; C=0 ; D=1 ; E=1 ; F=0 ; G=0 ; H =1 ; I=0 ;  #100;
A=0 ; B=0 ; C=1 ; D=1 ; E=1 ; F=0 ; G=0 ; H =1 ; I=0 ;  #100;
```

4. BLOCK FOUR: UMPIRE CIRCUIT

```
module DFF(Clk,CE,reset,D,set,Q,Qbar);
input Clk,CE,reset,D,set;
output Q,Qbar;
reg temp;
 always @(posedge(Clk))
    begin
        if(Clk == 1)
        begin
            if (CE == 1)
                temp = D;
            else if(set == 1)
             temp = 1;
         else if (reset == 1)
                 temp = 0;
      end
    end
assign Q = temp;
assign Qbar= ~temp;
endmodule

module
umpire_circuit(P1_W,P2_W,set,reset,P1_WI,P2_WI,P1_WIbar,P2_WIbar
,CE);
input P1_W,P2_W,set,reset,CE;
output P1_WI,P2_WI;
inout P1_WIbar,P2_WIbar;

DFF D1 (P1_W,CE,reset,P2_WIbar,set,P1_WI,P1_WIbar);
DFF D2 (P2_W,CE,reset,P1_WIbar,set,P2_WI,P2_WIbar);
endmodule
```

TEST BENCH FOR BLOCK FOUR(UMPIRE CIRCUIT)