A Report on

# Tic-Tac-Toe using FPGA

for

**Mini Project 2-B (REV- 2019 'C' Scheme) of Third Year, (TE Sem-VI)**

in

**Electronics and Telecommunication Engineering**

by

**Spencer Lobo**
**Shweta Rajput**
**Misha Rai**
**Ganesh Bakshe**

under the guidancce of
**Prof. Madhura Shirodkar**
**Prof. Shailaja Udtewar**

**UNIVERSITY OF MUMBAI**

**Department of Electronics and Telecommunication Engineering**
**Xavier Institute of Engineering**
Mahim(West), Mumbai-400016
(2021-2022)

# CERTIFICATE

This is to certify that the project entitled **Tic-Tac-Toe using FPGA** is a bonafide work of

***Spencer Lobo***
***Shweta Rajput***
***Misha Rai***
***Ganesh Bakshe***

is submitted to the University of Mumbai in partial fulfillment of the requirement for the award of **Mini Project 2-B (REV- 2019 'C' Scheme) of Third Year, (TE Sem-VI) )** in **(Electronics & Telecommunication Engineering)** as laid down by **(University of Mumbai )** during academic year **(2021-22)**

**Internal Examiner** **Reviewer-1**

**External Examiner/Reviewer-2**

# Abstract

Games provide a real source of enjoyment in daily life. Tic-Tac-Toe is a simple and yet an interesting board game. Tic-tac-toe is a world-wide popular two-person game, also spelled tic- tac-toe, and alternatively called Knoghts and Crosses, X's and O's, is a pencil-and-paper game for two players, O and X, who take turns marking the spaces in a 3×3 grid, usually X going first. an the idea of X's and 0's have been applied digitally using FPGA.

# Acknowledgement

We would like to thank our project guide Prof. Madhura Shirodkar and Prof. Shailaja Udtewar who initiated us into learning the subject of Tic-Tac-Toe using FPGA. They has been a source of inspiration and his insight and vision has made it possible for us to pursue and understand the developments in these areas. their patience, encouragement, critique and availability made this dissertation possible.

We are also grateful to Xavier Institute of Engineering Management, Principal Dr. Y. D. Venkatesh and especially the Head of the Department Ms. Vidya Sarode and all the faculty and staff who have helped us to be better acquainted with the recent trends in technology and from whom we have learned so much.

Moreover, we are thankful to all our colleagues and friends for the wonderful years and moments spent at XIE. They indeed have transformed our years at XIE into happy memories, memories which will linger on for decades.

Above all we are grateful to our family because of whose motivation and sacrifice we were able to pursue our Engineering studies. We are immensely grateful to our parents for their sacrifices and encouragement. We can never forget the dreams they have for us and the support they gave us from the very first day they held our hand and led us to school. We hope in the years to come our achievements will indeed make them proud.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

## 1.1 Motivation and Background

Games provide a real source of enjoyment and entertainment in our daily life. A simple game like tic-tac-toe can be a mirror of how people move through obstacles and handle decisions in life. It also shows you that the pluses outweigh the minuses and that you learn how to develop strategies to help you pull through. This two-player game can also be played by the ones who are lonely by playing it digitally with the computer.

## 1.2 Ojective

To make the simple tic-tac-toe game possible to be played between a player and a computer in a digital form using FPGA Spartan 6 board and verilog coding.

## 1.3 Outline and Scope

This dissertation report consists of six chapters. The contents of the chapters are as follows,

**Chapter 2:** Gives us the literature survey and information regarding FPGA Boards, EDA Tools, Technical Papers/Websites Referred and the Problem statement.

**Chapter 3:** Begins with a detailed description of the Methodology which includes the Working of project (state diagram/ flowchart / Modules) and Hardware Software components Description

**Chapter 4:** Implementation and Results

**Chapter 5:** Concludes this report with small summary about the project.

## 1.4 Time Plan

| | Week-1 | Week-2 | Week-3 | Week-4 | Week-5 | Week-6 | Week-7 | Week-8 | Week-9 | Week-10 | Week-11 | Week-12 | Week-13 | Week-14 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 7-1-2022 | 14/1/2022 | 21/1/2022 | 28/1/2022 | 4-2-2022 | 10-2-2022 | 25/2/2022 | 07-03-2022 | 09-03-2022 | 10-3-2022 | 17/3/2022 | 23/3/2022 | 30/3/2022 | 6-4-2022 |
| Comparison of FPGA boards & Introduction to Verilog Programming | | | | | | | | | | | | | | |
| Writing Verilog Code for Basic gates | | | | | | | | | | | | | | |
| Writing Test Bench for Basic gates | | | | | | | | | | | | | | |
| Create UCF File for Basic gates | | | | | | | | | | | | | | |
| Topic Selection Presentation (Group 1 to 6) | | | | | | | | | | | | | | |
| Youtube Video Reffering | | | | | | | | | | | | | | |
| Youtube Video Reffering | | | | | | | | | | | | | | |
| Writing Verilog Code, Testbench & UCF for Full adder | | | | | | | | | | | | | | |
| Writing Verilog Code, Testbench & UCF for Flipflop & | | | | | | | | | | | | | | |
| Writing Verilog Code | | | | | | | | | | | | | | |
| Writing Test Bench | | | | | | | | | | | | | | |
| Writing UCF for Mini Project | | | | | | | | | | | | | | |
| Hardware Implementation | | | | | | | | | | | | | | |
| Trouble Shooting | | | | | | | | | | | | | | |

Figure 1.1: Time Plan

# Chapter 2

# Literature Survey

## 2.1 FPGA Boards

### 2.1.1 Mimas - Spartan 6 FPGA Development Board / Numato Lab

Mimas is an easy to use FPGA Development board featuring Xilinx Spartan-6 FPGA. Mimas is specially designed for experimenting and learning system design with FPGAs. This development board features Xilinx XC6SLX9 TQG144 FPGA with a maximum of 70 user IOs. The USB 2.0 interface provides fast and easy configuration download to the onboard SPI flash. You don't need a programmer or special downloader cable to download the bitstream to the board.
Features:

- FPGA: Spartan-6 XC6SLX9 in TQG144 package
- Flash memory: 16 Mb SPI flash memory (M25P16)
- 100MHz CMOS oscillator
- USB 2.0 interface for On-board flash programming
- FPGA configuration via JTAG and USB

- 8 LEDs and four switches for user-defined purposes
- 70 IOs for user-defined purposes
- Onboard voltage regulators for single power rail operation

### 2.1.2 Zynq-7000 Development Board

The ZedBoard is a low-cost development board for the Xilinx Zynq-7000 all programmable SoC (APSoC).Take advantage of the Zynq-7000 APSoCs tightly coupled ARM processing system and 7-Series programmable logic to create unique and powerful designs with the ZedBoard. This board contains everything necessary to create a Linux, Android, Windows, or other OS/RTOS based design. Additionally, several expansion connectors expose the processing system and programmable logic I/Os for easy user access.
Features:

- Xilinx Zynq-7000 AP SoC XC7Z020-CLG484
- Dual-core ARM Cortex™-A9 processor
- 512 MB DDR3
- 256 MB Quad-SPI Flash
- On-board USB-JTAG Programming
- 10/100/1000 Ethernet
- USB OTG 2.0 and USB-UART
- PS  PL I/O expansion (FMC, Pmod, XADC)
- Analog Devices ADAU1761 SigmaDSP® Stereo, Low Power, 96 kHz, 24-Bit Audio Codec
- Analog Devices ADV7511 High Performance 225 MHz HDMI Transmitter (1080p HDMI, 8-bit VGA, 128x32 OLED)

### 2.1.3 Artix -7 Kinetex Boards

The Xilinx Kintex UltraScale FPGAs are available in -3, -2, -1 speed grades, with -3E devices having the highest performance. The -2LE and -1LI devices can operate at a VCCINT voltage at 0.85V or 0.72V and provide lower maximum static power. When operated at VCCINT = 0.85V, using -2LE and -1LI devices, the speed specification for the L devices is the same as the -2I or -1I speed grades. When operated at VCCINT = 0.72V, the -2LE and -1LI performance and static and dynamic power is reduced.
Features:

- Features the Xilinx Artix-7 FPGA: XC7A35T-1CPG236C
- 33,280 logic cells in 5200 slices (each slice contains four 6-input LUTs and 8 flip-flops)
- 1,800 Kbits of fast block RAM
- UFive clock management tiles, each with a phase-locked loop (PLL)
- 90 DSP slices
- Internal clock speeds exceeding 450 MHz
- On-chip analog-to-digital converter (XADC)
- Digilent USB-JTAG port for FPGA programming and communication
- Designed Exclusively for Vivado Design Suite (Vivado Design Suite WebPACK edition

## 2.2 EDA Tools

### 2.2.1 Xilinx ISE

ISE design suite supports the Spartan-6, Virtex-6, and CoolRunner devices, as well as their previous generation families. ISE design suite runs on and below Windows 10 operating systems.

The ISE Design Suite consists of a type of System Edition builds on top of the Embedded Edition by adding on System Generator for DSP. System Generator for DSP is the industry's leading high-level tool for designing high-performance DSP systems using Xilinx programmable devices, providing system modeling and automatic code generation from Simulink and MATLAB.

### 2.2.2 Vivado

Vivado Design Suite delivers a SoC-strength, IP-centric and system-centric, next generation development environment that has been built from the ground up to address the productivity bottlenecks in system-level integration and implementation. It comes in two editions:
a) Vivado HL Edition
b) Vivado ML Edition

### 2.2.3 EDA Playground

EDA Playground is a free web application that lets users edit, simulate (and view waveforms), synthesise, and share HDL code. Its goal is to accelerate design and testbench development learning by making code sharing easier and providing easier access to simulators and libraries. EDA Playground is intended specifically for small prototypes and examples (it is not intended to be used for a full-blown FPGA or ASIC design).

The usage model is straightforward: enter your code, select your preferred simulator or synthesis tool, and click run. If your simulation dumps waves, simply check a box and the waves will open in a new window after the run. Any code or waveform display, such as this one, can be saved as a static HTML link so that anyone can open it, re-run it, and get the same result.

## 2.3 Technical Papers/Websites Referred

### 2.3.1 Dynamic simulation of electric machines on FPGA boards

The paper that follows describes the implementation of an induction machine dynamic simulation on a Field Programmable Gate Array (FPGA) board. When compared to a typical PC computer, FPGAs can provide significant simulation speed gains, particularly when operations are efficiently parallelized on the board. The basic steps of designing a Runge-Kutta numerical Ordinary Differential Equation (ODE) solver on the FPGA platform are outlined using an example of a free acceleration followed by a step load change. The FPGA simulation results and speed increase are validated in comparison to a Matlab/Simulink simulation.

A field-programmable gate array (FPGA) is a reconfigurable digital logic platform that can execute millions of bit-level operations in parallel in a spatially pro-

grammed environment. FPGAs as computational and non-computational devices are being used in research on the modelling and real-time simulation of various electrical power components. The goal here is to implement an entire dynamic simulation of an induction machine as quickly as possible on a single FPGA board

### 2.3.2 An FPGA Platform for Hyperscalers

FPGAs (Field Programmable Gate Arrays) are making their way into data centers (DC). They are used as accelerators to boost the compute power of individual server nodes and to improve the overall power efficiency. Meanwhile, DC infrastructures are being redesigned to pack ever more compute capacity into the same volume and power envelopes.

Data-center disaggregation is the division of traditional server architecture into a collection of standalone and modular computing memory and storage resources. In practise, this means that traditional rack and blade servers will be replaced by sled and micro-servers. This decision is solely motivated by the performance-per-dollar metric, which is improved by increasing server density and sharing resources such as power supplies, PCB backplanes, cooling, fans, networking uplinks, and other management infrastructure.

## 2.4 Problem Statement

To Implement TIC-TAC-TOE game using Verilog Coding on FPGA Spartan 6 board

# Chapter 3

# Methodology

## 3.1 Working of Project

### 3.1.1 State Diagram

With the help of Figure 3.1, we can see the states and the corresponding to the project's working for that specific state. We can see the states and their position in Table 3.1.

Table 3.1: States and their Position

| **State** | **Position** |
|---|---|
| PLAY = 1, RESET = 0 | IDLE to Player |
| PLAY = 0, RESET = 1 | IDLE condition |
| Illegal Move = 1, | Player |
| Illegal Move = 0 | Player to Computer |
| PC = 0 | Computer |
| No Space = 1, Winner = 1, PC = 1 | Computer to Game Over |
| RESET = 0 | Game Over |
| RESET = 1 | Game Over to Idle |
| No Space = 0 , Winner = 1 | Player to Idle |

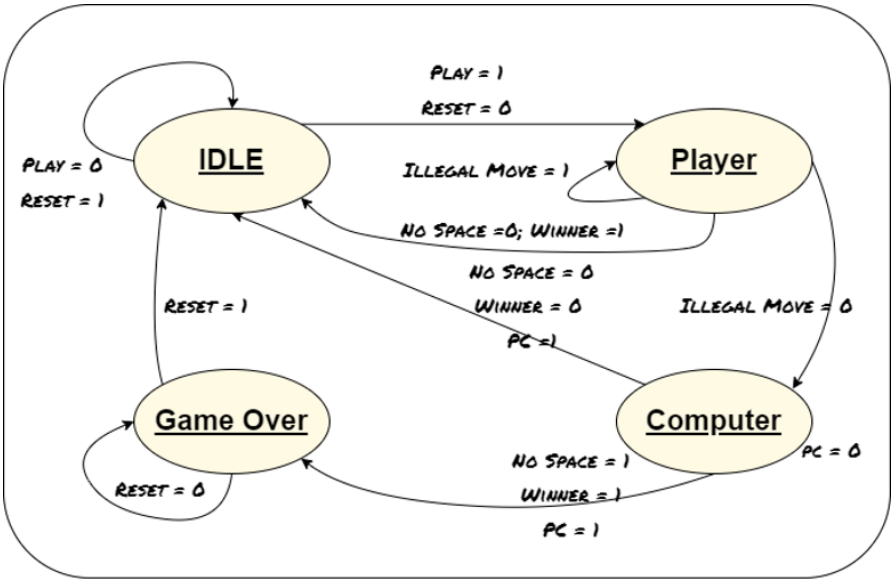



Figure 3.1: State Diagram

### 3.1.2 Flowchart / Block Diagram

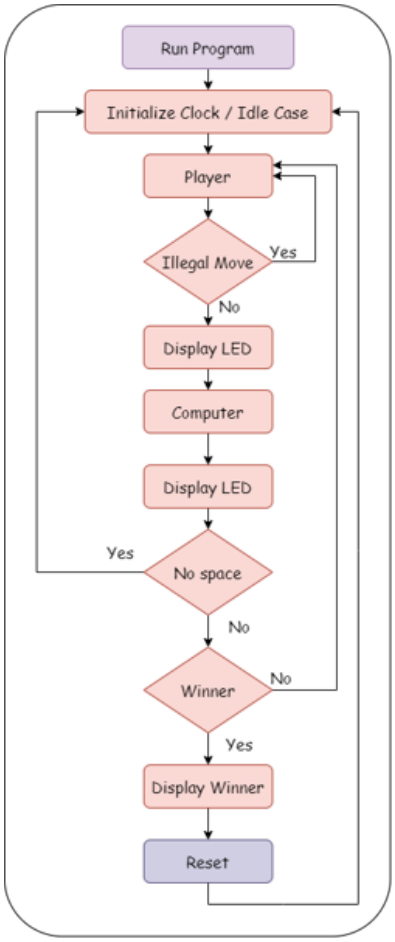



Figure 3.2: Flowchart / Block Diagram

When we run the program, the next step is to initialise the clock, which is also the Idle state, followed by the player going first. There will be no illegal moves if it is the first move (illegal moves are moves that have once been played and are repeated). If there is no illegal move, an LED will light up to show the position of the move, and the computer will then decide and play the move, which will be followed by the LED lighting up.
In the event of an illegal move, the player will be given another opportunity to place his/her move correctly.
After the computer's turn, it will determine whether or not there are any more spaces for the computer to play in. If there are no available spaces, the game will be declared a tie, and the game will be restarted.
If there are any remaining spaces, it will look for the winner using the given condition, i.e. diagonals, horizontals, and verticals. If no winner is determined, the player will proceed to the next move. If a winner is detected, the winner will be displayed using an LED, and the game will be reset.

### 3.1.3 Modules

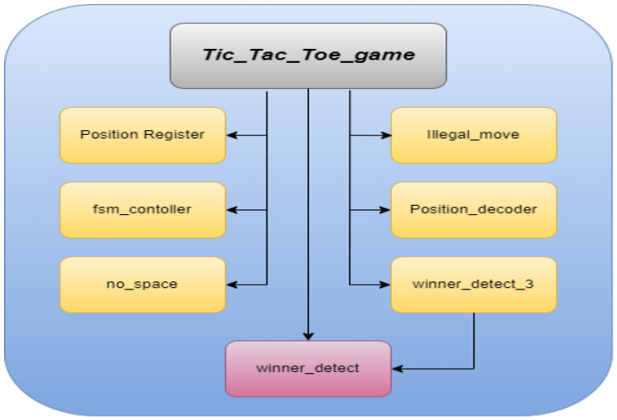



Figure 3.3: Modules

The Top Module is the Tic_Tac_Toe_game which is used to define all input and output pins/position used.
The fsm_controller Module is the and is The FSM is Finite state machine implemented based on the designed state diagram.
The Position Register Module is used to to store player or computer positions.
The no_space module is to detect if no more spaces to play.
The illegal_move module is used to to detect if a player plays on an exist position
The winner_detect_3 module is used for the detection of the 3 positions and determine who the winner is Player: 01 Computer: 10.
The winner_detect module is used to detect the winner .

## 3.2 Hardware Software components Description

### 3.2.1 Hardware Requirements

Table 3.1 lists the components used in circuit simulation.

Table 3.2: List of Components

| Name | Quantity |
|---|---|
| FPGA Spartan6-XC6SLX9_TQ144FPGA board | 1 |
| Push Button | 9 |
| Led (3/4 Terminal) | 11 |
| Breadboard | 3 |
| Connecting Wires | As required |

### 3.2.2 Spartan6-XC6SLX9_TQ144FPGA board

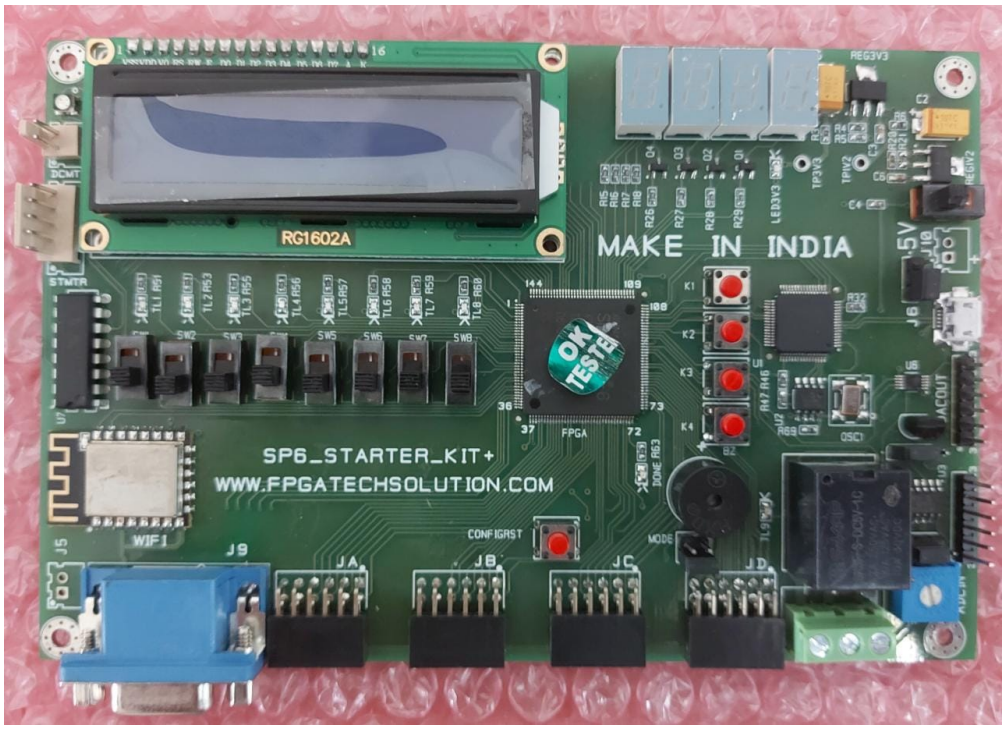



Figure 3.4: Spartan6-XC6SLX9_TQ144FPGA FPGA Board

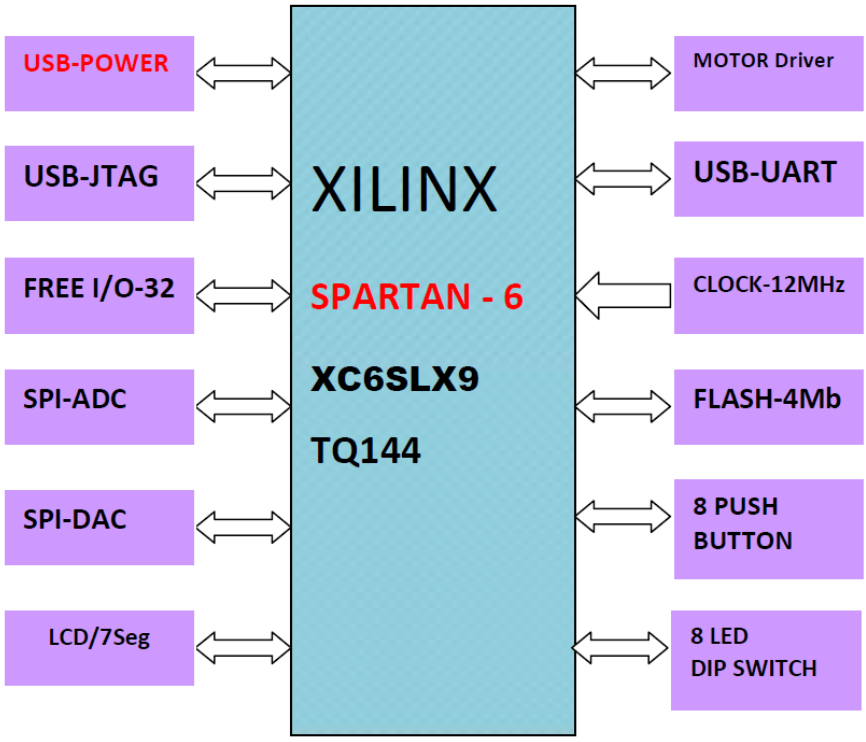
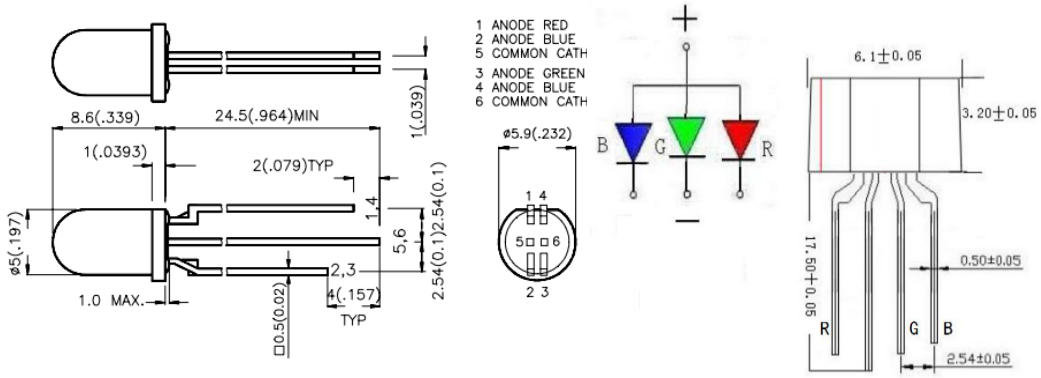



Figure 3.5: XILINX SPARTAN - 6 XC6SLX9TQ144

The Spartan 6 Starter Kit can run on USB power, but it can also be powered by an external 5V DC supply. When the JP2 jumper is set to 2 or 3, power is drawn from the USB connector. When the JP2 jumper is set to 1  2, power is drawn from an external power adaptor.
The SPARTAN6 STARTER KIT+ board includes eight individual LEDs and eight slide switches. When an I/O is configured as an output, an LED is assigned to it to indicate its data status. To provide digital input, a DIP switch is used (i.e. logic 0 and logic 1).

### 3.2.3 3 / 4 Pin LED




(a) 3 Terminal LED (b) 4 Terminal LED

Figure 3.6: LED

A three-pin LED is typically a pair of different-colored LEDs that share an anode or cathode. LEDs can be turned on individually or in groups to form a combination. A three-pin common-cathode bi-color LED.
A 4 pin led consists of RGB (red, green, and blue) LEDs and are most commonly packaged in a 4-pin connector. There are both common cathode and common anode versions.

### 3.2.4 Software components Description

In this project the software used is the Xilinx ISE


Figure 3.7: Xilinx ISE

Xilinx ISE is tightly coupled to the architecture of Xilinx's own chips (the internals of which are highly proprietary) and cannot be used with other vendors' FPGA products, as is common in the commercial electronic design automation sector. ISE enables developers to synthesise ("compile") their designs, perform timing analysis, examine RTL diagrams, simulate the response of a design to various stimuli, and configure the target device with the programmer. The Xilinx ISE also includes the Embedded Development Kit (EDK), Software Development Kit (SDK), and ChipScope Pro. The Xilinx ISE is used primarily for circuit design and synthesis, whereas the ISIM or ModelSim logic simulator is used for system-level testing.

# Chapter 4

# Implementation and Result

## 4.1 Module Creation and Interfacing

Verilog code was written for Tic-Tac-Toe. and the Figure 4.1 shows the verilog code was successfully executed.

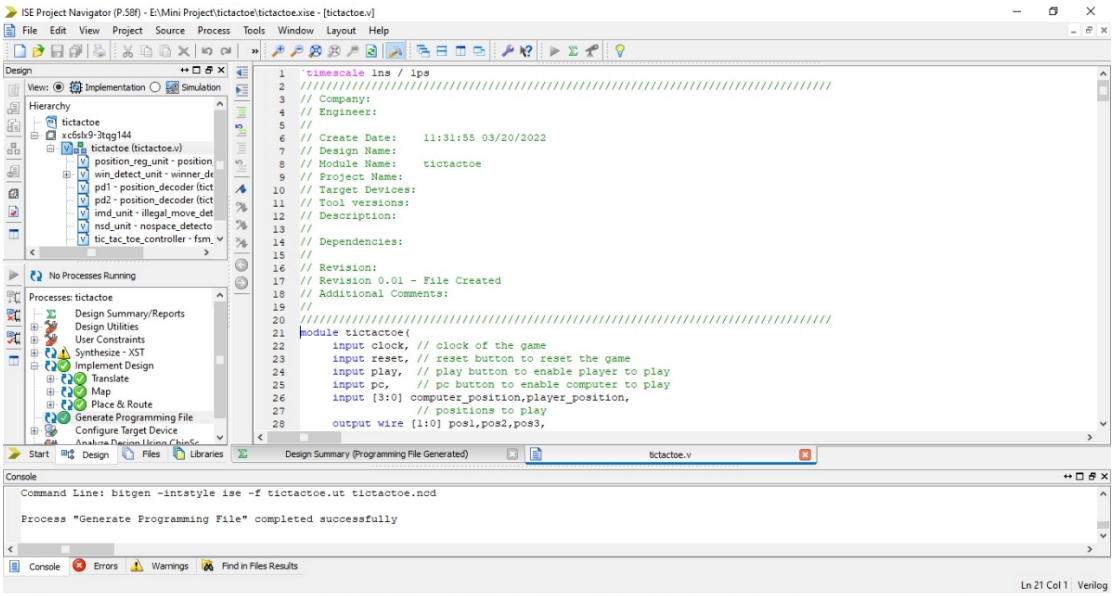



Figure 4.1: Verilog Code and Execution

## 4.2 Test Bench

For the above verilog code test-bench was created and the results can be observed. We got two result for the test-bench, one with forcing the clock and one with initializing the clock. Figure 4.2 Shows the Test-Bench Execution with Forced Clock and Constant and Figure 4.3 shows Test-Bench Execution With Clock.

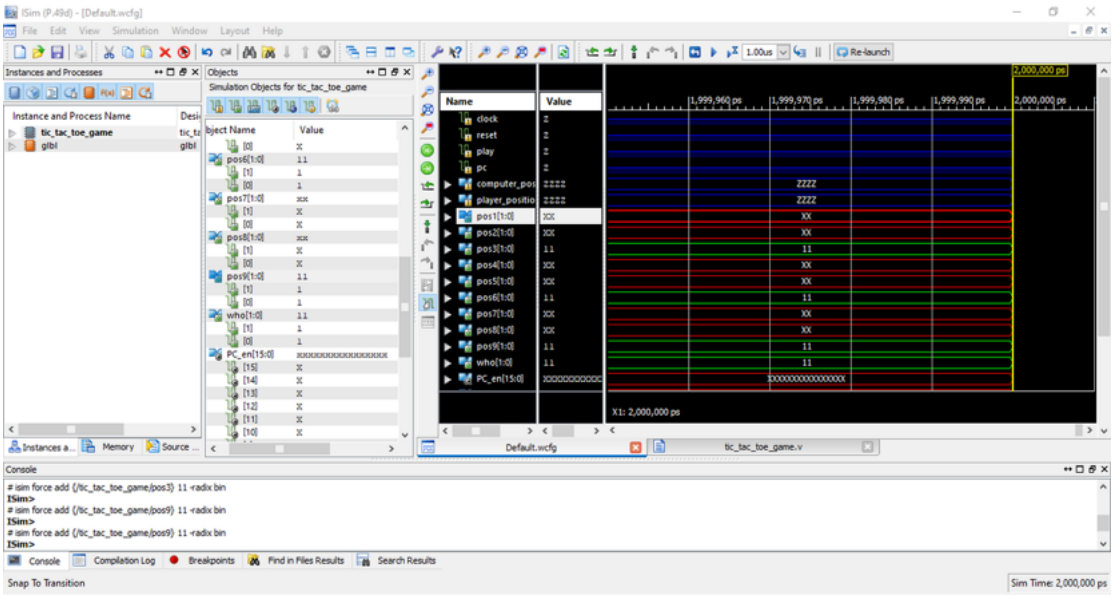


Figure 4.2: Test-Bench Execution with Forced Clock and Constant

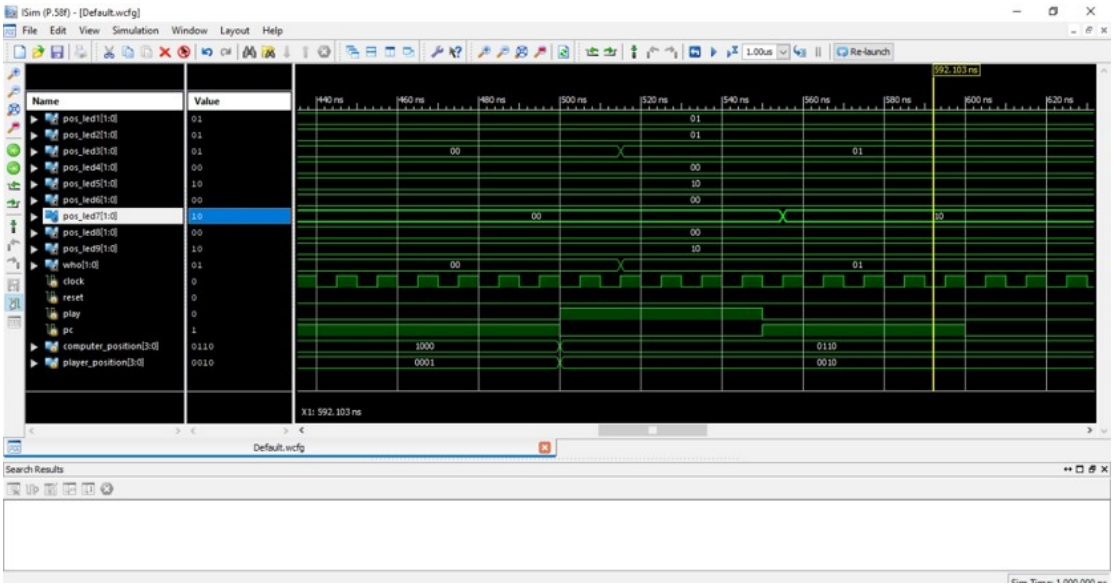


Figure 4.3: Test-Bench Execution With Clock

## 4.3 RTL and Technology Schematic

Using XIlinx Software the below RTL diagrams were generated

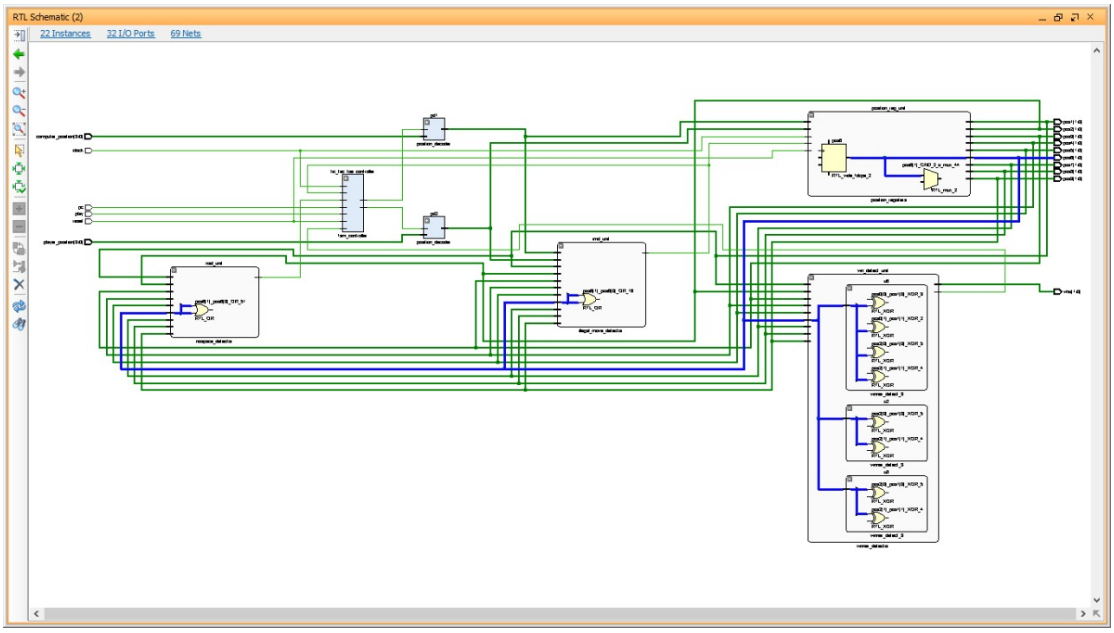



Figure 4.4: RTL Logic Diagram

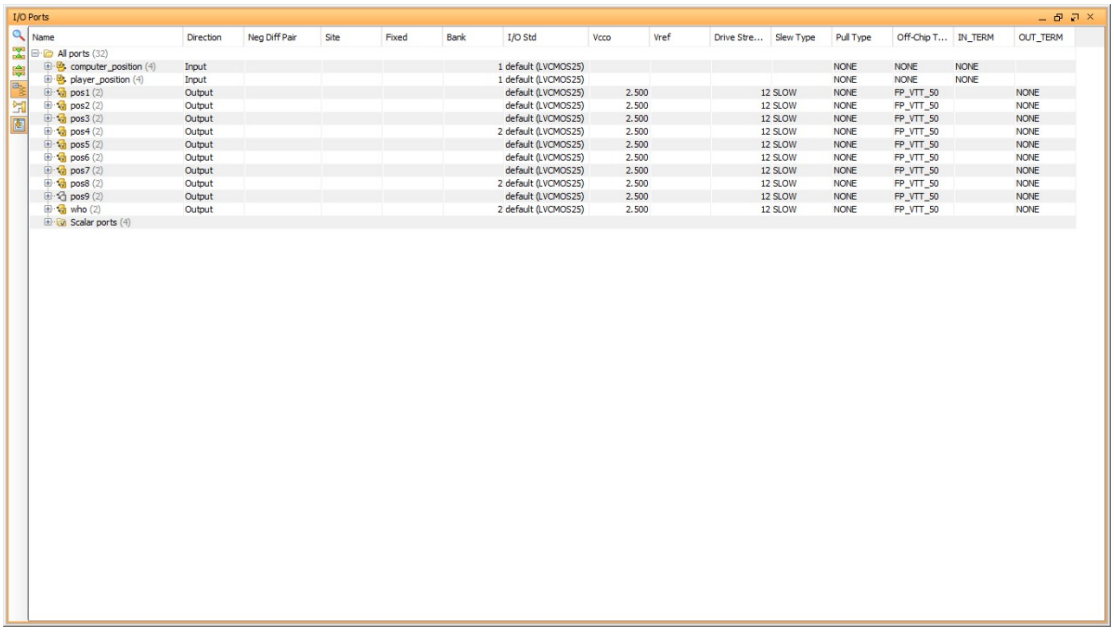



Figure 4.5: RTL Input-Output Pins

## 4.4 UCF file

Using the Spartan6-XC6SLX9_TQ144FPGA Manual, we were able to identify the ports that will be used in our verilog code. The UCF code was used to initialise those ports, and a UCF was created. When the UCF file was simulated, there were no errors.

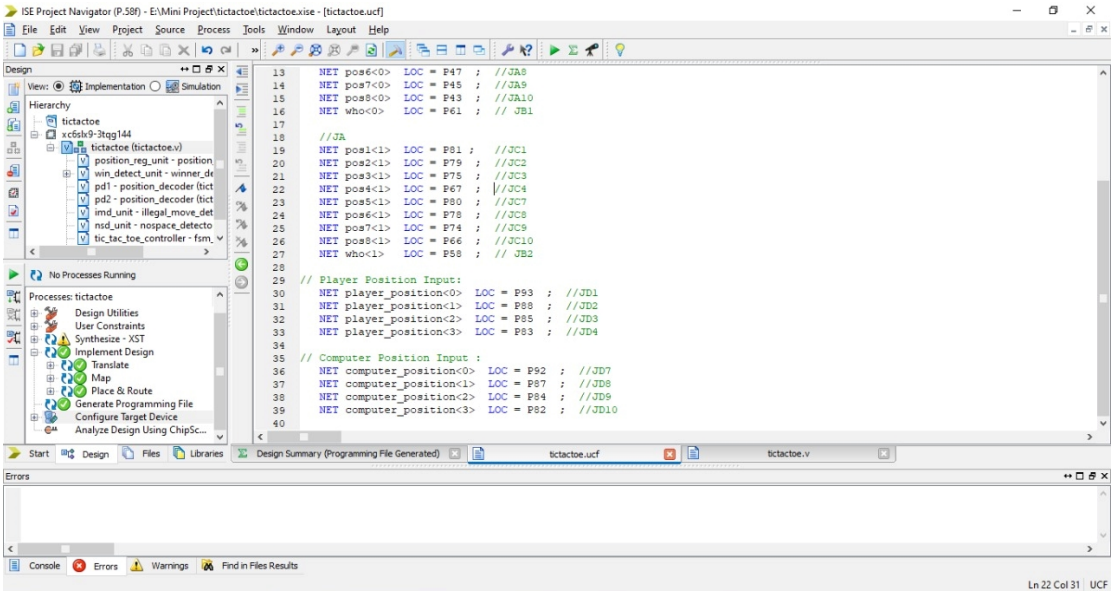



Figure 4.6: UCF Code Execution

## 4.5 Configuration of Target Device

The file was configured with the Spartan6-XC6SLX9_TQ144FPGA board using the UCF generated in section 4.4. The result, as shown in Figure 4.7, was a "Program Succeeded" text, indicating that the program was successfully burned to the Spartan6-XC6SLX9_TQ144FPGA Board.

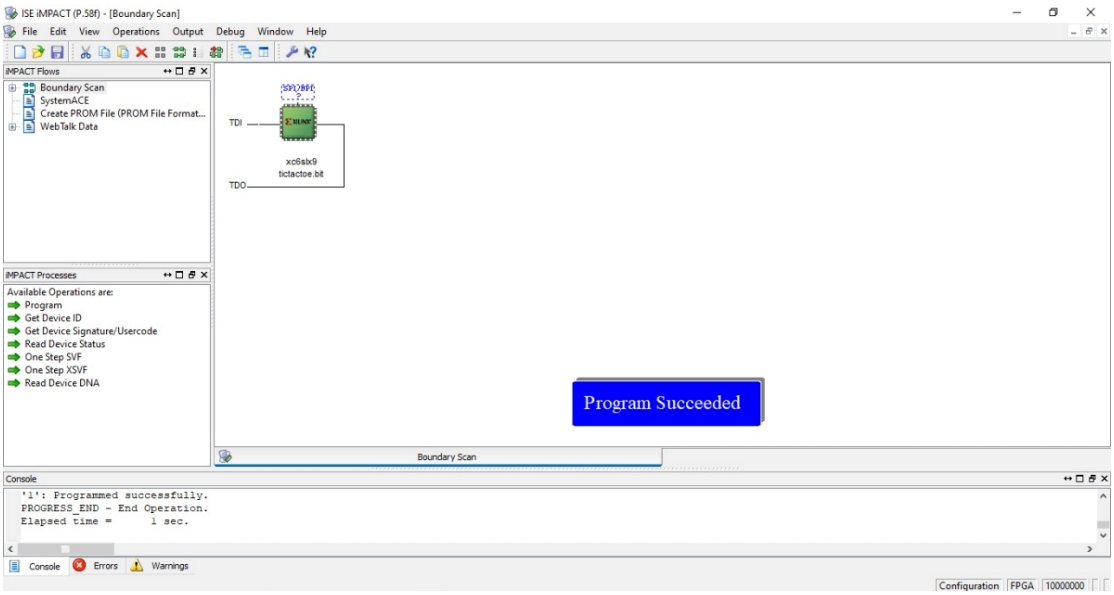



Figure 4.7: Configuration of Target Device Successful

## 4.6 Hardware Implementation

Figure 4.8 depicts the hardware implementation that was used to connect the Spartan6-XC6SLX9_TQ144FPGA board and burn the UCF file.

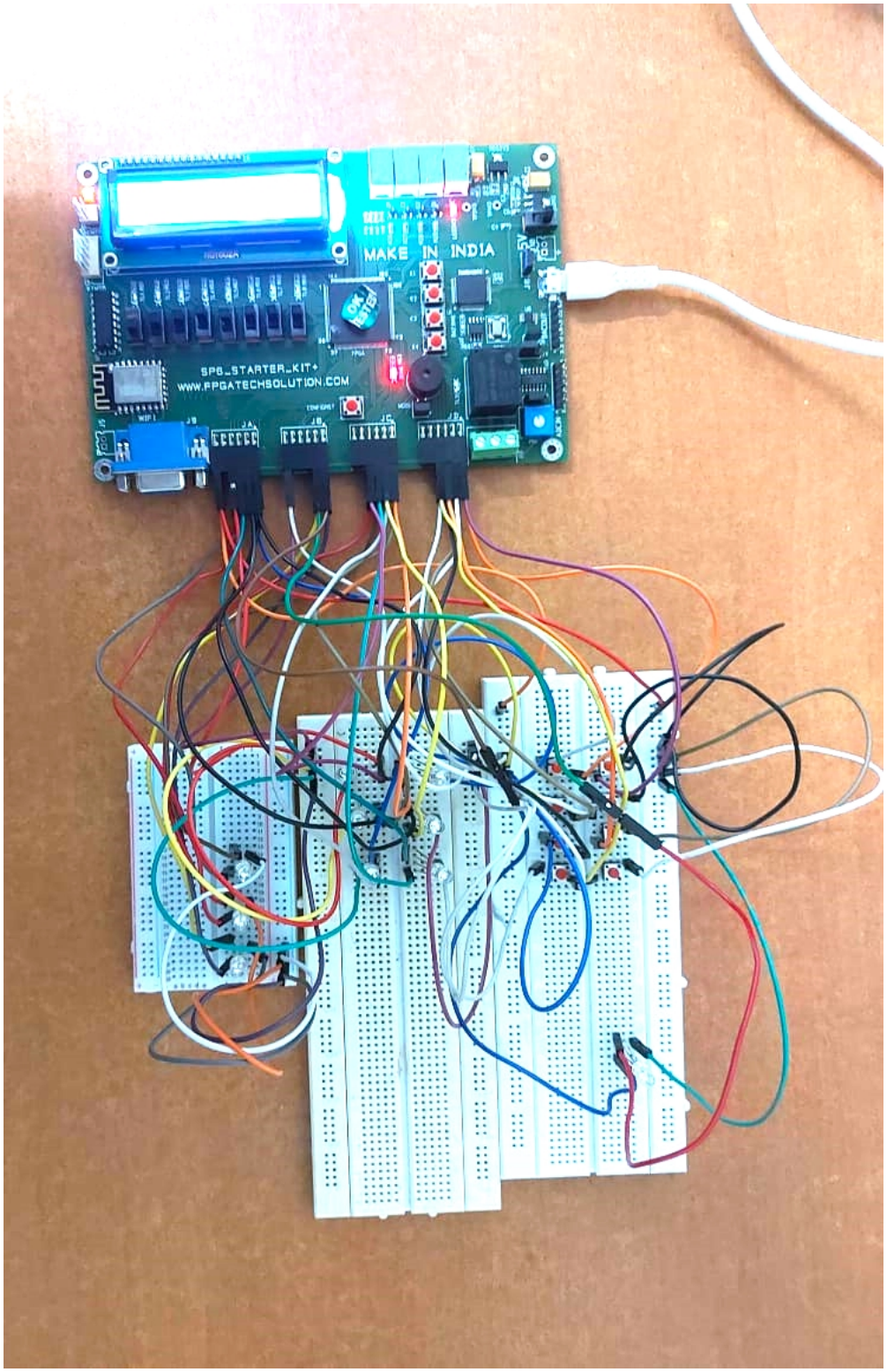



Figure 4.8: Implemented Hardware

# Chapter 5

# Conclusion

In this project we have tried to Implement Tic – Tac –Toe Using Spartan6-XC6SLX9_TQ144FPGA FPGA board.

With the help of the Pre-Synthesis simulation we were able to see the results with the help of a waveform where we could give (force) input which resulted in accurate output; the winner was detected.

The UCF file was successfully created and the code was successful burnt on the board; given by the "Program Succeeded" message was seen on the Xilinx ISE software.

# Reference


[1] "Tic Tac Toe on FPGA Platform - Blog - FPGA - element14 Community." https://community.element14.com/technologies/fpga-group/b/blog/posts/tic-tac-toe-on-fpg a-platform.

[2] "Tic Tac Toe Game in Verilog and LogiSim - FPGA4student.com." https://www.fpga4student.com/2017/06/tic-tac-toe-game-in-verilog-and-logisim.html.

[3] http://www.diva-portal.org/smash/get/diva2:357030/fulltext01.pdf

[4] "LED pinouts - 2, 3, 4-pin and more — LEDnique." http://lednique.com/leds-with-more-than-two-pins/.

[5] "GitHub - fpgatechsolution/MINI_SP6_FPGA." https://github.com/fpgatechsolution/MINI_SP6_FPGA.

[6] "Prototyping with FPGA - YouTube." https://www.youtube.com/playlist?list=PLYBDk4yLHCMynEx_qLtLyLM04T-CffUdAs.

# Appendix

**Verilog Code:**

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date:    11:31:55 03/20/2022
// Design Name:
// Module Name:    tictactoe
// Project Name:
// Target Devices:
// Tool versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////
module tictactoe(
     input clock, // clock of the game
     input reset, // reset button to reset the game
     input play,  // play button to enable player to play
     input pc,    // pc button to enable computer to play
     input [3:0] computer_position,player_position,
                  // positions to play
     output wire [1:0] pos1,pos2,pos3,pos4,pos5,pos6,pos7,pos8,pos9,
                  // LED display for positions
                  // 01: Player
                  // 10: Computer
```

```
     output wire[1:0]who
                  // who the winner is
     );
 wire [15:0] PC_en;// Computer enable signals
 wire [15:0] PL_en; // Player enable signals
 wire illegal_move; // disable writing when an illegal move is
detected
 //wire [1:0] pos1,pos2,pos3,pos4,pos5,pos6,pos7,pos8,pos9;//
positions stored
 wire win; // win signal
 wire computer_play; // computer enabling signal
 wire player_play; // player enabling signal
 wire no_space; // no space signal
 // position registers
  position_registers position_reg_unit(
      clock, // clock of the game
      reset, // reset the game
      illegal_move, // disable writing when an illegal move is
detected
      PC_en[8:0], // Computer enable signals
      PL_en[8:0], // Player enable signals
      pos1,pos2,pos3,pos4,pos5,pos6,pos7,pos8,pos9// positions
stored
      );
 // winner detector
 winner_detector
win_detect_unit(pos1,pos2,pos3,pos4,pos5,pos6,pos7,pos8,pos9,win,who
);
 // position decoder for computer
 position_decoder pd1(computer_position,computer_play,PC_en);
 // position decoder for player
 position_decoder pd2(player_position,player_play,PL_en);
 // illegal move detector
  illegal_move_detector imd_unit(
   pos1,pos2,pos3,pos4,pos5,pos6,pos7,pos8,pos9,
   PC_en[8:0], PL_en[8:0],
```

```
   illegal_move
   );
 // no space detector
 nospace_detector nsd_unit(
   pos1,pos2,pos3,pos4,pos5,pos6,pos7,pos8,pos9,
   no_space
    );
 fsm_controller tic_tac_toe_controller(
     clock,// clock of the circuit
     reset,// reset
     play, // player plays
     pc,// computer plays
     illegal_move,// illegal move detected
     no_space, // no_space detected
     win, // winner detected
     computer_play, // enable computer to play
     player_play // enable player to play
     );
endmodule
// Position registers

// to store player or computer positions
// when enabling by the FSM controller

module position_registers(
      input clock, // clock of the game
      input reset, // reset the game
      input illegal_move, // disable writing when an illegal move is
detected
      input [8:0] PC_en, // Computer enable signals
      input [8:0] PL_en, // Player enable signals
      output reg[1:0] pos1,pos2,pos3,pos4,pos5,pos6,pos7,pos8,pos9//
positions stored
      );
 // Position 1
```

```
 always @(posedge clock or posedge reset)
 begin
  if(reset)
   pos1 <= 2'b00;
  else begin
   if(illegal_move==1'b1)
    pos1 <= pos1;// keep previous position
   else if(PC_en[0]==1'b1)
    pos1 <= 2'b10; // store computer data
   else if (PL_en[0]==1'b1)
    pos1 <= 2'b01;// store player data
   else
    pos1 <= pos1;// keep previous position
  end
 end
 // Position 2
 always @(posedge clock or posedge reset)
 begin
  if(reset)
   pos2 <= 2'b00;
  else begin
   if(illegal_move==1'b1)
    pos2 <= pos2;// keep previous position
   else if(PC_en[1]==1'b1)
    pos2 <= 2'b10; // store computer data
   else if (PL_en[1]==1'b1)
    pos2 <= 2'b01;// store player data
   else
    pos2 <= pos2;// keep previous position
  end
 end
// Position 3
 always @(posedge clock or posedge reset)
 begin
```

```
 if(reset)
  pos3 <= 2'b00;
 else begin
  if(illegal_move==1'b1)
   pos3 <= pos3;// keep previous position
  else if(PC_en[2]==1'b1)
   pos3 <= 2'b10; // store computer data
  else if (PL_en[2]==1'b1)
   pos3 <= 2'b01;// store player data
  else
   pos3 <= pos3;// keep previous position
 end
end
// Position 4
always @(posedge clock or posedge reset)
begin
 if(reset)
  pos4 <= 2'b00;
 else begin
  if(illegal_move==1'b1)
   pos4 <= pos4;// keep previous position
  else if(PC_en[3]==1'b1)
   pos4 <= 2'b10; // store computer data
  else if (PL_en[3]==1'b1)
   pos4 <= 2'b01;// store player data
  else
   pos4 <= pos4;// keep previous position
 end
end
// Position 5
always @(posedge clock or posedge reset)
begin
 if(reset)
  pos5 <= 2'b00;
```

```
 else begin
  if(illegal_move==1'b1)
   pos5 <= pos5;// keep previous position
  else if(PC_en[4]==1'b1)
   pos5 <= 2'b10; // store computer data
  else if (PL_en[4]==1'b1)
   pos5 <= 2'b01;// store player data
  else
   pos5 <= pos5;// keep previous position
 end
end
// Position 6
always @(posedge clock or posedge reset)
begin
 if(reset)
  pos6 <= 2'b00;
 else begin
  if(illegal_move==1'b1)
   pos6 <= pos6;// keep previous position
  else if(PC_en[5]==1'b1)
   pos6 <= 2'b10; // store computer data
  else if (PL_en[5]==1'b1)
   pos6 <= 2'b01;// store player data
  else
   pos6 <= pos6;// keep previous position
 end
end
// Position 7
always @(posedge clock or posedge reset)
begin
 if(reset)
  pos7 <= 2'b00;
 else begin
  if(illegal_move==1'b1)
```

```
   pos7 <= pos7;// keep previous position
  else if(PC_en[6]==1'b1)
   pos7 <= 2'b10; // store computer data
  else if (PL_en[6]==1'b1)
   pos7 <= 2'b01;// store player data
  else
   pos7 <= pos7;// keep previous position
 end
end
// Position 8
always @(posedge clock or posedge reset)
begin
 if(reset)
  pos8 <= 2'b00;
 else begin
  if(illegal_move==1'b1)
   pos8 <= pos8;// keep previous position
  else if(PC_en[7]==1'b1)
   pos8 <= 2'b10; // store computer data
  else if (PL_en[7]==1'b1)
   pos8 <= 2'b01;// store player data
  else
   pos8 <= pos8;// keep previous position
 end
end
// Position 9
always @(posedge clock or posedge reset)
begin
 if(reset)
  pos9 <= 2'b00;
 else begin
  if(illegal_move==1'b1)
   pos9 <= pos9;// keep previous position
  else if(PC_en[8]==1'b1)
```

```
    pos9 <= 2'b10; // store computer data
   else if (PL_en[8]==1'b1)
    pos9 <= 2'b01;// store player data
   else
    pos9 <= pos9;// keep previous position
  end
 end
endmodule

// FSM controller to control how player and computer play the TIC
TAC TOE GAME

module fsm_controller(
     input clock,// clock of the circuit
     input reset,// reset
     play, // player plays
     pc,// computer plays
     illegal_move,// illegal move detected
     no_space, // no_space detected
     win, // winner detected
     output reg computer_play, // enable computer to play
     player_play // enable player to play
     );
// FSM States
parameter IDLE=2'b00;
parameter PLAYER=2'b01;
parameter COMPUTER=2'b10;
parameter GAME_DONE=2'b11;
reg[1:0] current_state, next_state;
// current state registers
always @(posedge clock or posedge reset)
begin
 if(reset)
  current_state <= IDLE;
```

```
 else
  current_state <= next_state;
end
 // next state
always @(*)
 begin
 case(current_state)
 IDLE: begin
  if(reset==1'b0 && play == 1'b1)
   next_state <= PLAYER; // player to play
  else
   next_state <= IDLE;
  player_play <= 1'b0;
  computer_play <= 1'b0;
 end
 PLAYER:begin
  player_play <= 1'b1;
  computer_play <= 1'b0;
  if(illegal_move==1'b0)
   next_state <= COMPUTER; // computer to play
  else
   next_state <= IDLE;
 end
 COMPUTER:begin
  player_play <= 1'b0;
  if(pc==1'b0) begin
   next_state <= COMPUTER;
   computer_play <= 1'b0;
  end
  else if(win==1'b0 && no_space == 1'b0)
  begin
   next_state <= IDLE;
   computer_play <= 1'b1;// computer to play when PC=1
  end
```

```
  else if(no_space == 1 || win ==1'b1)
  begin
   next_state <= GAME_DONE; // game done
   computer_play <= 1'b1;// computer to play when PC=1
  end
 end
 GAME_DONE:begin // game done
  player_play <= 1'b0;
  computer_play <= 1'b0;
  if(reset==1'b1)
   next_state <= IDLE;// reset the game to IDLE
  else
   next_state <= GAME_DONE;
 end
 default: next_state <= IDLE;
 endcase
 end
endmodule
// NO SPACE detector
// to detect if no more spaces to play
module nospace_detector(
   input [1:0] pos1,pos2,pos3,pos4,pos5,pos6,pos7,pos8,pos9,
   output wire no_space
    );
wire temp1,temp2,temp3,temp4,temp5,temp6,temp7,temp8,temp9;
// detect no more space
assign temp1 = pos1[1] | pos1[0];
assign temp2 = pos2[1] | pos2[0];
assign temp3 = pos3[1] | pos3[0];
assign temp4 = pos4[1] | pos4[0];
assign temp5 = pos5[1] | pos5[0];
assign temp6 = pos6[1] | pos6[0];
assign temp7 = pos7[1] | pos7[0];
assign temp8 = pos8[1] | pos8[0];
```

```
assign temp9 = pos9[1] | pos9[0];
// output
assign no_space =(((((((temp1 & temp2) & temp3) & temp4) & temp5) &
temp6) & temp7) & temp8) & temp9);
endmodule


// Illegal move detector
// to detect if a player plays on an exist position


module illegal_move_detector(
   input [1:0] pos1,pos2,pos3,pos4,pos5,pos6,pos7,pos8,pos9,
   input [8:0] PC_en, PL_en,
   output wire illegal_move
   );
wire temp1,temp2,temp3,temp4,temp5,temp6,temp7,temp8,temp9;
wire temp11,temp12,temp13,temp14,temp15,temp16,temp17,temp18,temp19;
wire temp21,temp22;
// player : illegal moving
assign temp1 = (pos1[1] | pos1[0]) & PL_en[0];
assign temp2 = (pos2[1] | pos2[0]) & PL_en[1];
assign temp3 = (pos3[1] | pos3[0]) & PL_en[2];
assign temp4 = (pos4[1] | pos4[0]) & PL_en[3];
assign temp5 = (pos5[1] | pos5[0]) & PL_en[4];
assign temp6 = (pos6[1] | pos6[0]) & PL_en[5];
assign temp7 = (pos7[1] | pos7[0]) & PL_en[6];
assign temp8 = (pos8[1] | pos8[0]) & PL_en[7];
assign temp9 = (pos9[1] | pos9[0]) & PL_en[8];
// computer : illegal moving
assign temp11 = (pos1[1] | pos1[0]) & PC_en[0];
assign temp12 = (pos2[1] | pos2[0]) & PC_en[1];
assign temp13 = (pos3[1] | pos3[0]) & PC_en[2];
assign temp14 = (pos4[1] | pos4[0]) & PC_en[3];
assign temp15 = (pos5[1] | pos5[0]) & PC_en[4];
assign temp16 = (pos6[1] | pos6[0]) & PC_en[5];
```

```
assign temp17 = (pos7[1] | pos7[0]) & PC_en[6];

assign temp18 = (pos8[1] | pos8[0]) & PC_en[7];

assign temp19 = (pos9[1] | pos9[0]) & PC_en[8];

// intermediate signals

assign temp21 =(((((((temp1 | temp2) | temp3) | temp4) | temp5) |
temp6) | temp7) | temp8) | temp9);

assign temp22 =(((((((temp11 | temp12) | temp13) | temp14) |
temp15) | temp16) | temp17) | temp18) | temp19);

// output illegal move

assign illegal_move = temp21 | temp22 ;

endmodule

// To decode the position being played, a 4-to-16 decoder with high
active output is needed.

// When a button is pressed, a player will play and the position at
IN [3:0] will be decoded

// to enable writing to the corresponding registers

module position_decoder(input[3:0] in, input enable, output wire
[15:0] out_en);

 reg[15:0] temp1;

 assign out_en = (enable==1'b1)?temp1:16'd0;

 always @(*)

 begin

 case(in)

 4'd0: temp1 <= 16'b0000000000000001;

 4'd1: temp1 <= 16'b0000000000000010;

 4'd2: temp1 <= 16'b0000000000000100;

 4'd3: temp1 <= 16'b0000000000001000;

 4'd4: temp1 <= 16'b0000000000010000;

 4'd5: temp1 <= 16'b0000000000100000;

 4'd6: temp1 <= 16'b0000000001000000;

 4'd7: temp1 <= 16'b0000000010000000;

 4'd8: temp1 <= 16'b0000000100000000;

 4'd9: temp1 <= 16'b0000001000000000;

 4'd10: temp1 <= 16'b0000010000000000;

 4'd11: temp1 <= 16'b0000100000000000;

 4'd12: temp1 <= 16'b0001000000000000;
```

```
 4'd13: temp1 <= 16'b0010000000000000;
 4'd14: temp1 <= 16'b0100000000000000;
 4'd15: temp1 <= 16'b1000000000000000;
 default: temp1 <= 16'b0000000000000001;
 endcase
end
endmodule
// winner detector circuit
// to detect who the winner is
// We will win when we have 3 similar (x) or (O) in the following
pairs:
// (1,2,3); (4,5,6);(7,8,9); (1,4,7); (2,5,8);(3,6,9);
(1,5,9);(3,5,7);
module winner_detector(input [1:0]
pos1,pos2,pos3,pos4,pos5,pos6,pos7,pos8,pos9, output wire winner,
output wire [1:0]who);
wire win1,win2,win3,win4,win5,win6,win7,win8;
wire [1:0] who1,who2,who3,who4,who5,who6,who7,who8;
winner_detect_3 u1(pos1,pos2,pos3,win1,who1);// (1,2,3);
winner_detect_3 u2(pos4,pos5,pos6,win2,who2);// (4,5,6);
winner_detect_3 u3(pos7,pos8,pos9,win3,who3);// (7,8,9);
winner_detect_3 u4(pos1,pos4,pos7,win4,who4);// (1,4,7);
winner_detect_3 u5(pos2,pos5,pos8,win5,who5);// (2,5,8);
winner_detect_3 u6(pos3,pos6,pos9,win6,who6);// (3,6,9);
winner_detect_3 u7(pos1,pos5,pos9,win7,who7);// (1,5,9);
winner_detect_3 u8(pos3,pos5,pos6,win8,who8);// (3,5,7);
assign winner = (((((((win1 | win2) | win3) | win4) | win5) | win6)
| win7) | win8);
assign who = (((((((who1 | who2) | who3) | who4) | who5) | who6) |
who7) | who8);
endmodule
// winner detection for 3 positions and determine who the winner is
// Player: 01
// Computer: 10
module winner_detect_3(input [1:0] pos0,pos1,pos2, output wire
winner, output wire [1:0]who);
wire [1:0] temp0,temp1,temp2;
```

```
wire temp3;
assign temp0[1] = !(pos0[1]^pos1[1]);
assign temp0[0] = !(pos0[0]^pos1[0]);
assign temp1[1] = !(pos2[1]^pos1[1]);
assign temp1[0] = !(pos2[0]^pos1[0]);
assign temp2[1] = temp0[1] & temp1[1];
assign temp2[0] = temp0[0] & temp1[0];
assign temp3 = pos0[1] | pos0[0];
// winner if 3 positions are similar and should be 01 or 10
assign winner = temp3 & temp2[1] & temp2[0];
// determine who the winner is
assign who[1] = winner & pos0[1];
assign who[0] = winner & pos0[0];
endmodule
```

**Test-Bench Code**

```
`timescale 1ns / 1ps
module tb_tic_tac_toe;
 // Inputs
 reg clock;
 reg reset;
 reg play;
 reg pc;
 reg [3:0] computer_position;
 reg [3:0] player_position;

 // Outputs
 wire [1:0] pos_led1;
 wire [1:0] pos_led2;
 wire [1:0] pos_led3;
 wire [1:0] pos_led4;
 wire [1:0] pos_led5;
 wire [1:0] pos_led6;
 wire [1:0] pos_led7;
 wire [1:0] pos_led8;
 wire [1:0] pos_led9;
 wire [1:0] who;

 // Instantiate the Unit Under Test (UUT)
 tic_tac_toe_game uut (
  .clock(clock),
  .reset(reset),
  .play(play),
  .pc(pc),
  .computer_position(computer_position),
  .player_position(player_position),
  .pos1(pos_led1),
  .pos2(pos_led2),
```

```
 .pos3(pos_led3),
 .pos4(pos_led4),
 .pos5(pos_led5),
 .pos6(pos_led6),
 .pos7(pos_led7),
 .pos8(pos_led8),
 .pos9(pos_led9),
 .who(who)
);
// clock
initial begin
clock = 0;
forever #5 clock = ~clock;
end
initial begin
 // Initialize Inputs
 play = 0;
 reset = 1;
 computer_position = 0;
 player_position = 0;
 pc = 0;
 #100;
 reset = 0;
 #100;
 play = 1;
 pc = 0;
 computer_position = 4;
 player_position = 0;
 #50;
 pc = 1;
 play = 0;
 #100;
 reset = 0;
 play = 1;
```

```
    pc = 0;
    computer_position = 8;
    player_position = 1;
    #50;
    pc = 1;
    play = 0;
    #100;
    reset = 0;
    play = 1;
    pc = 0;
    computer_position = 6;
    player_position = 2;
    #50;
    pc = 1;
    play = 0;
    #50
    pc = 0;
    play = 0;
    end

endmodule
```

**UCF Code:**

```
NET clock  LOC = P95  ;
//Switches
     NET reset  LOC = P26  ;
     NET play  LOC = P24  ;
     NET pc  LOC = P16  ;
//Leds
 //JA
     NET pos1<0>  LOC = P48 ;    //JA1
     NET pos2<0>  LOC = P46  ;  //JA2
     NET pos3<0>  LOC = P44  ;  //JA3
     NET pos4<0>  LOC = P41  ;  //JA4
     NET pos5<0>  LOC = P50  ;  //JA7
     NET pos6<0>  LOC = P47  ;  //JA8
     NET pos7<0>  LOC = P45  ;  //JA9
     NET pos8<0>  LOC = P43  ;   //JA10
     NET who<0>  LOC = P61  ;  // JB1

     //JA
     NET pos1<1>  LOC = P81 ;    //JC1
     NET pos2<1>  LOC = P79  ;  //JC2
     NET pos3<1>  LOC = P75  ;  //JC3
     NET pos4<1>  LOC = P67  ;  //JC4
     NET pos5<1>  LOC = P80  ;  //JC7
     NET pos6<1>  LOC = P78  ;  //JC8
     NET pos7<1>  LOC = P74  ;  //JC9
     NET pos8<1>  LOC = P66  ;   //JC10
     NET who<1>  LOC = P58  ;  // JB2

// Player Position Input:
     NET player_position<0>  LOC = P93  ;  //JD1
     NET player_position<1>  LOC = P88  ;  //JD2
     NET player_position<2>  LOC = P85  ;  //JD3
     NET player_position<3>  LOC = P83  ;  //JD4
```

```
// Computer Position Input :
      NET computer_position<0>  LOC = P92  ;  //JD7
      NET computer_position<1>  LOC = P87  ;  //JD8
      NET computer_position<2>  LOC = P84  ;  //JD9
      NET computer_position<3>  LOC = P82  ;  //JD10
```