Gabriel Facco Bettinelli

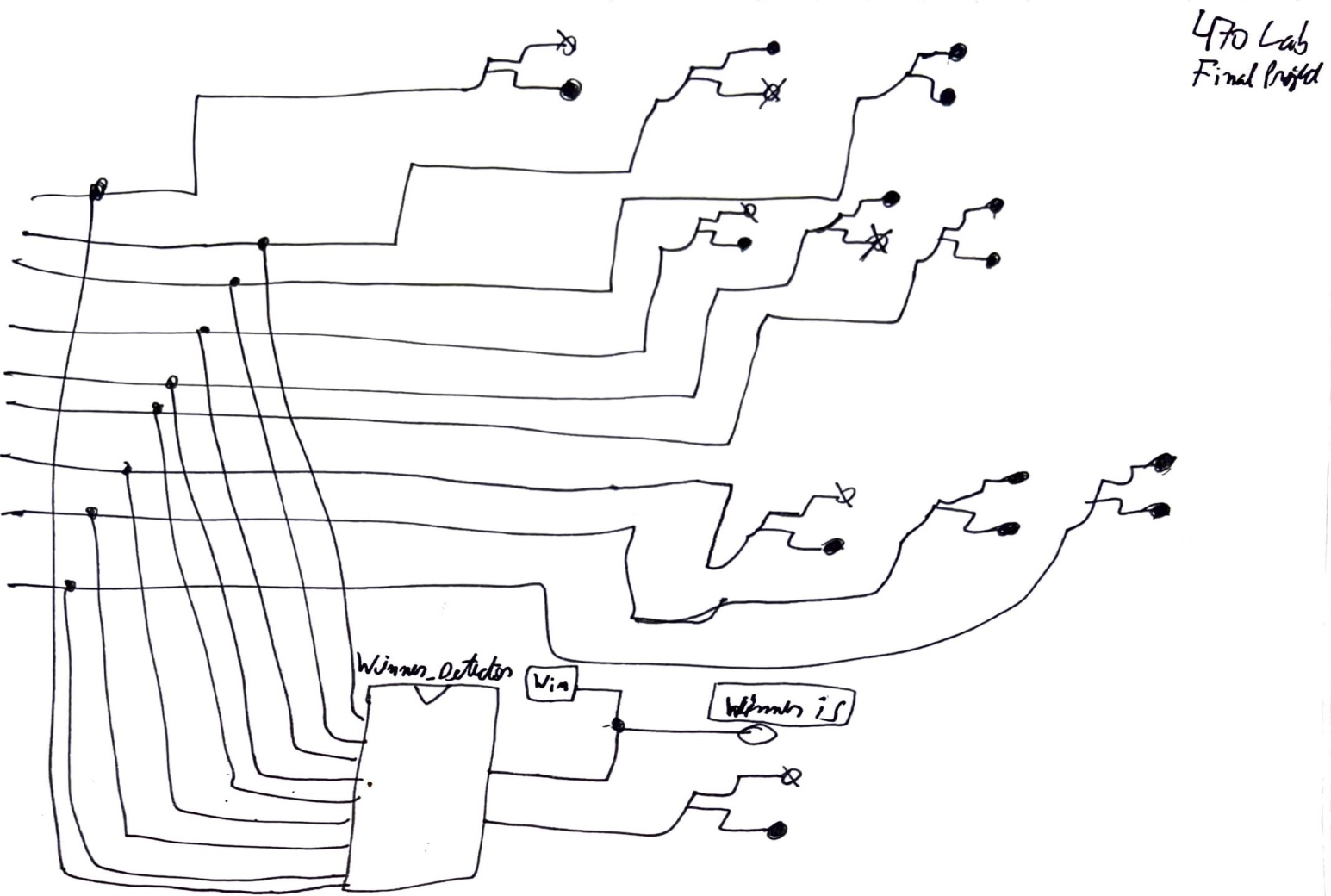
Compe470Lab

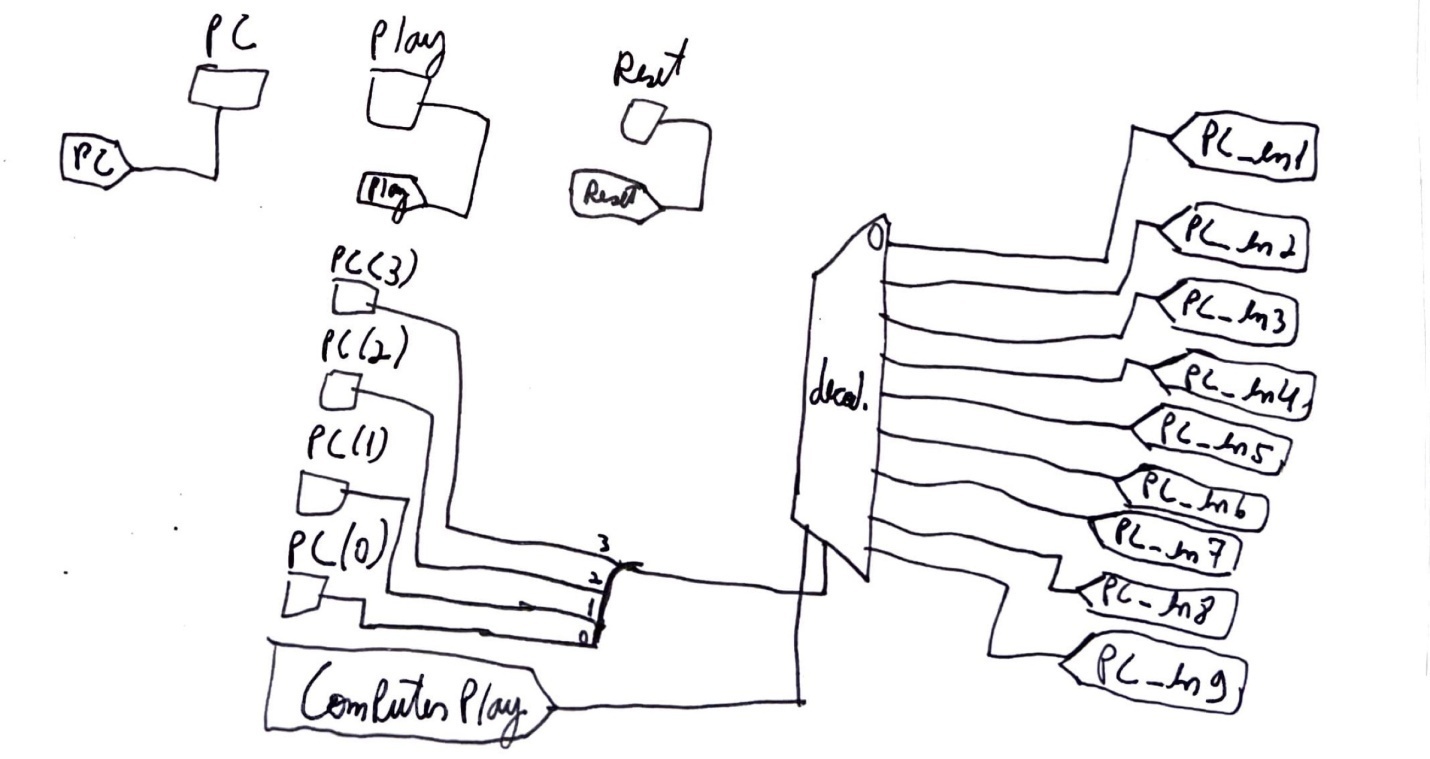
W-0900-1140 Section

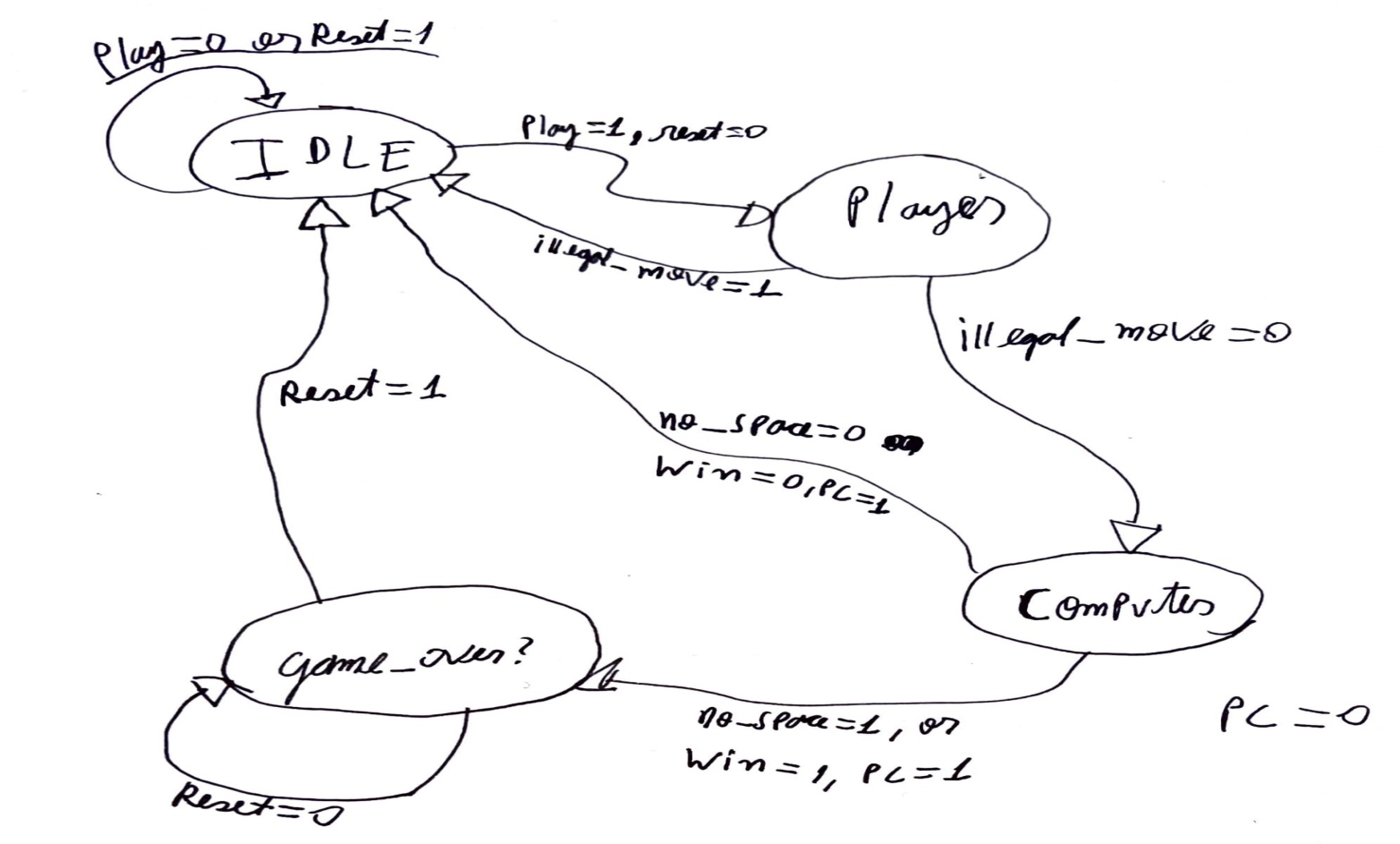
**Final Project – Tic Tac Toe Game on Verilog**

For my final project, I decided to simulate a Tic Tac Toe game on Verilog, using the Vivado IDE.The Tic Tac Toe is game played in a 3x3 grid where the player who makes the first three of their marks in a diagonal, vertical, or horizontal row wins the game. To achieve that, I created a Verilog Module called Tic\_Tac\_Toe\_Game, which contained the logic for the game, which is as follows:  
As the player/ computer plays its turn, a 2-bit value is stored into one of the nine positions in the 3x3 grid like as in the real paper-and-pencil version. 2'b00 is stored into a position when neither the player nor computer played in that position. Similarly, 2'b01 (X) is the value to be stored when the player played in the position and 2'b10 (O) is the value to be saved when the computer played in the position.

To see if we got the right logic implementation, I created a Verilog TestBench to Simulate a Tic Tac Toe Game between a player (user) and the computer. As we instantiate the game on the TestBench, we are able to analyze the WaveForm Simulation given by Verilog and see which player won by seeing the wave value of the ‘who’ variable.

 I started my design with a hand draw schematic of how the logic circuit of the game would look like.

* Here, we have all the 9 positions on the grid, and the X is showing the grid slots that the user chooses to play. The Winner\_Detector circuit is designed to find the winner of the game. 
* Also, to detect an illegal move, a comparator is needed to check if the current position was already played by either the computer or player. Moreover, “No space” detector is to check if all the positions are played and no winner is found.

Advancing on the logic required to Control the game on Verilog a FSM controller is one of the easiest ways to control a 1x1 game, since Tic Tac Toe is also played in turns, so I designed the follow **Finite State Machine** to control the Tic Tac Toe game:

**The FSM States were the following:**

**1.IDLE(00)** – When waiting for the user/pc to play or when game is over and we need to reset the game.

**2. Player(01)** – When is the Player turn to play and “01” is stored into the decoded position.

**3. Computer(10)** – When is the Computer turns to play and “10” is stored into the decoded position.

**4.Game\_Over(11)** – When the game is finished and we have a winner or if we have No More Space to play.

And My **Main Logic Module(Tic\_Tac\_Toe\_Game) contained the following inputs:**

1.Reset. (reset = 1, resets the game when in Game\_Over state.) (reset = 0, begins the game)

2. Play. (play = 1, is to switch the controller to Player State and the player plays) (play = 0, stays in the IDLE State)

3. PC. (PC =1, is to switch to the IDLE state and the Computer plays) (PC = 0, stay in Computer State)

4.Illegal\_Move. (Illegal\_Move = 0, when in Player Mode is to switch to Computer State and let computer plays when PC-1) (Illega\_Move = 1, Illegal Move from the Player or Computer and Switch to IDLE State.

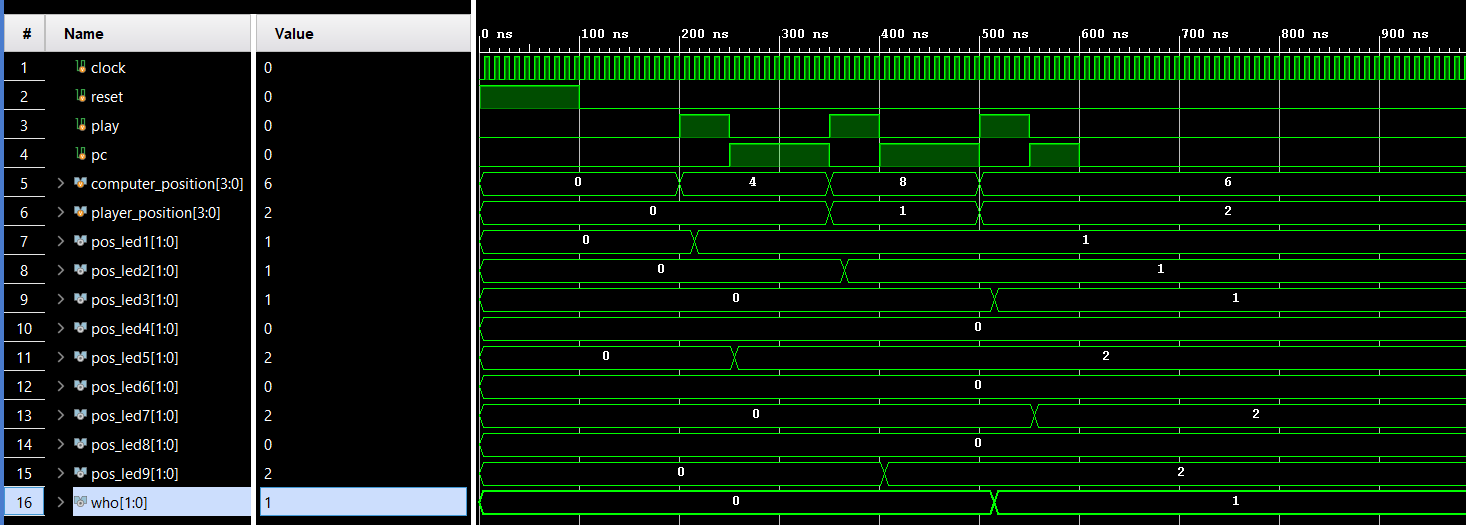
5.No\_Space. (No\_Space = 0, still have space to play, continue the game) (No\_Space = 1, means no more space to play, game over and need to reset the game)

6.Win.(also an Output) (Win = 0, We are still waiting for a Winner of the game) (Win = 1, someone won the game and we need to reset the game)

**And The Following OUTPUTS:**

1. wire[1:0] pos1, pos2, pos3, pos4, pos5, pos6, pos7, pos8 and pos9 representing all nine possible play locations, and to store the Player/PC Position. It player has the position, a “01” will be stored in the wire, if the PC has the position, a “10” will be stored in the wire.
2. who[1:0] to show and store the code of the User that won the Game. A “01 will be stored in the who if the player won the game and a “10” will be stored in the who if the PC won the game.

Lastly, I set up my TestBench in way that the User would win the game by marking in slot units 1,2 and 3, in sequence and the computer would mark off units 4,5 and 8, not making a game. So, after we simulate our Vivado Code into the Behavioral Simulation of Vivado, we get the following WaveForm and we can see that the Simulation resulted in the correct winner of the Game, which was player 1, and we can see that the “who” value is 1, which refers to Player 1(user).



**Vivado Code:**

**Tic\_Tac\_Toe\_Game.v:**

module Tic\_Tac\_Toe\_Game(

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

// The FSM is implemented based on the designed state diagram

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

**Verilog TestBench:**

**tb\_tic\_tac\_toe.v:**

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