Student 1: Tambade Atharv Rhushikesh Roll Number: 210260057

Student 2: Jason Gomez Roll Number:210260023

Final project title: 3D Tic Tac Toe

Final project Executive summary: 10

Components: 2 FPGA's, Mechanical Stand, 2 remotes, 64 green bulbs, 64 yellow bulbs

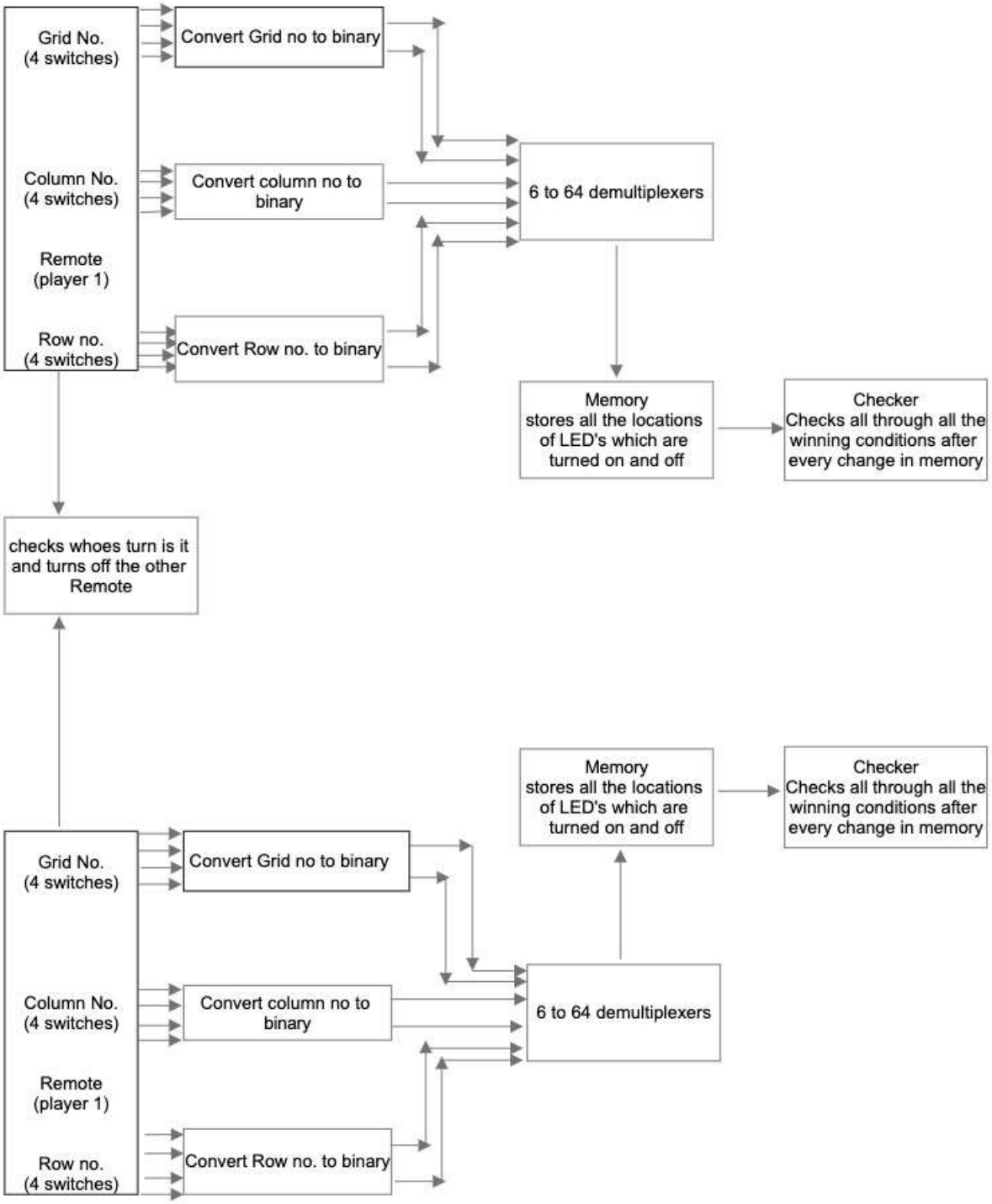
User will have a remote on which 12 buttons are present (to avoid debouncing) 4 each for choosing grid, row and column. To let the player know which number he has chosen we assigned 4 inbuilt LED's on FPGA. Now each will get converted to binary and we will have 6 inputs. We made 1 to 64 demultiplexers in FPGA. So each these 6 inputs will choose 1 unique output out of 64 and light that bulb. The same process is there for other player. We have kept two unidirectional 1 bit channels between the 2 FPGA's to know whose turn it is. We also covered all the 76 winning conditions. We have kept a reset button to reset the game

Winning conditions: On each of the four 4x4 boards, or horizontal planes, there are four columns, four rows, and two diagonals, accounting for 40 lines. There are 16 vertical lines, each ascending from a cell on the bottom board through the corresponding cells on the other boards. There are eight vertically-oriented planes parallel to the sides of the boards, each of these adding two more diagonals (the horizontal and vertical lines of these planes have already been counted). Finally, there are two vertically-oriented planes that include the diagonal lines of the 4x4 boards, and each of these contributes two more diagonal lines—each of these including two corners and two internal cells

To indicate who won there is LED on each player's FPGA which will light up

Final project details:

Theoretical design: 50



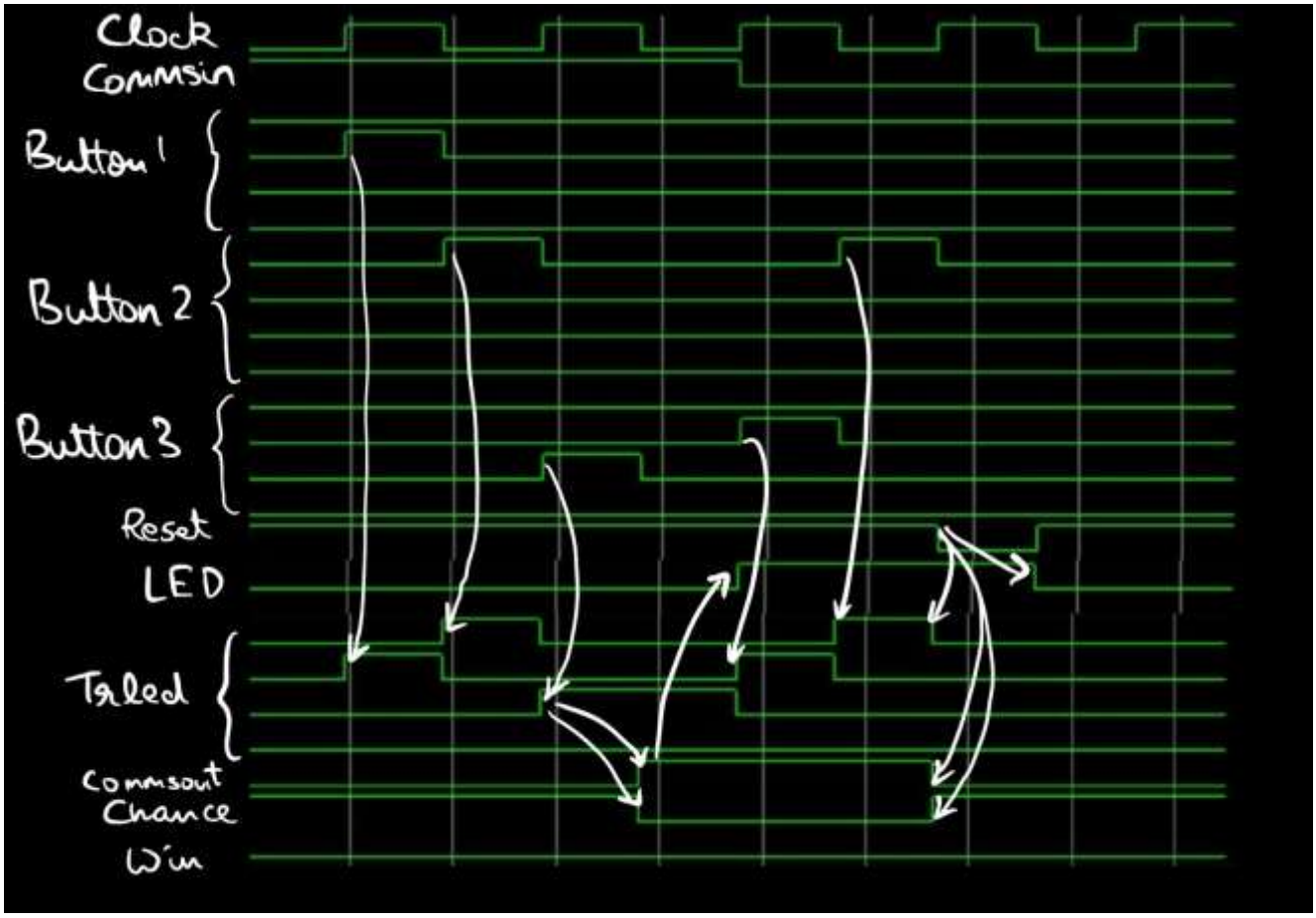
INPUTS

1. commsin : in std_logic; Takes in communication from the other FPGA about when the other player's chance is over
2. button1 : in std_logic_vector(3 downto 0); Takes in inputs of four buttons which choose the stack
3. button2 : in std_logic_vector(3 downto 0); Takes in inputs of four buttons which choose the row
4. button3 : in std_logic_vector(3 downto 0); Takes in inputs of four buttons which choose the column
5. rst : in std_logic; Resets the memory of the FPGA, for an effective game reset both FPGA's have to be reset

OUTPUTS

6. led : out std_logic_vector(63 downto 0); 64 LEDs showing the cells the player has placed their symbol in
7. trled : out std_logic_vector(3 downto 0); (On the FPGA) Shows the most recent selected number among stack, row and column
8. commsout : out std_logic; Gives out communication to the other FPGA when your chance has ended
9. chance : out std_logic; (On the FPGA) Shows whether its your chance or not
10. win : out std_logic; (On the FPGA) Shows whether you have won yet or not

Modelsim



Experiment Implementation: 40

A working model of the game

1) Two players will be there and each one will have a remote in their hands.
2) They will give 3 inputs from 12 buttons they have after pressing each button they will get to know which grid, column, row they have chosen by LED's on FPGA.
3) After doing this the LED which he has chosen will light up and his remote will get disabled until next player plays this will keep happening till winning condition occurs.
4) When one of the player wins winning LED will light up on his FPGA

Connection with Physics 50 bonus marks

The answer of this includes game theory so Let's start with the question why we chose to make 3D tic tac toe instead of 2D and also why 4×4×4 instead of 3×3×3.

According to Game theory problem with 2D tic Tac toe is if 2 players play optimally the outcome will always be a draw and playing optimally is very easy in the 2D case so the game becomes boring.

According to game theory in 3x3x3 version of the game cannot end in a draw and is easily won by the first player unless a rule is adopted that prevents the first player from taking the center cell. In that case, the game is easily won by the second player. By banning the use of the center cell altogether, the game is easily won by the first player so the game is not fair inherently and after choosing the centre cell it's really easy to spot the winning strategy now let's discuss what is so special about 4×4×4 one.

To our disappointment it's not the case that no winning strategy exists, but it's really difficult to predict by a human without using a computer and that makes the game more fun. First complete first player win strategy was published by Oren Patashnik but the computer assisted proof consumed 1500 hours of computer time and the strategy comprised choices for 2929 difficult "strategic positions". This what makes the game intrinsically fun because even if the game is inherently not fair, the difficulty in coming up with a winning strategy makes it such that on the surface it feels fair

The game was solved again by Victor Allis using proof number search. Proof Number search is a game tree search algorithm with applications mostly in endgame solvers, but also for sub-goals during games.

Oren_Patashnik_algorithm.pdf : Oren Patashnik algorithm link

Group Theoretical Structures in the game

The corner cells and the internal cells are actually equivalent via an automorphism; likewise for face and edge cells. The group of automorphisms of the game contains 192 automorphisms. It is made up of combinations of the usual rotations and reflections that reorient or reflect the cube, plus two that scramble the order of cells on each line. If a line comprises cells A, B, C and D in that order, one of these exchanges inner cells for outer ones (such as B, A, D, C) for all lines of the cube, and the other exchanges cells of either the inner or the outer cells (A, C, B, D or equivalently D, B, C, A) for all lines of the cube. Combinations of these basic automorphisms generate the entire group of 192 it Is shown by R. Silver. The link is given below

Automorphisms_3D_tictactoe.pdf

Appendix:

Our project idea is still the same but we have to change lot of things while trying to implement it. We faced following problems while implementing the idea

1) VHDL doesn't work like normal programming languages so the problem was their can't exist 2 processes having same arguments and same variables changing ,So we have to change our code multiple times to make it compatible with this feature of language.
2) The problem on which we invested most of our time was to take input from the user. First we thought of using the number pad which we had in our lab. We understood it's working then

wrote a code which we thought will work but it was not compatible for that and also we were not able to find the reason why it didn't work so after wasting 2 days on it we moved on from that idea. The we thought making that using normal switches in lab but the main problem involving that was signal debouncing , So we made a debouncer using "IC555" but it didn't work properly we think that the reason is in FPGA processes are to fast so even if slight noise after using debouncing is getting recorded in FPGA. So till writing this report we have 2 ideas write a code which will work sort of debouncer and include that in FPGA. We have another idea if this doesn't we will directly use 12 buttons so even if in debounces it won't create any problem because we wish to take only 1 input from a button so 1 player won't use the same button twice in his turn so the debouncing doesn't create any issue, but the only 1 issue we have in that is FPGA has only 72 pins on which we can give inputs and outputs but now we need 78-79 input and output portals so we have to use 6 inbuilt buttons on the FPGA. Then also there could be chance that we are 1 pin short.

CODE :

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity TicTac3D is
    port (
              clk : in std_logic;
              commsin : in std_logic;
              button1 : in std_logic_vector(3 downto 0);
              button2 : in std_logic_vector(3 downto 0);
              button3 : in std_logic_vector(3 downto 0);
              rst : in std_logic;
        led : out std_logic_vector(63 downto 0) := x"0000000000000000";
              trled : out std_logic_vector(3 downto 0) := "0000";
              commsout : out std_logic := '0';
              chance : out std_logic := '1';
              win : out std_logic := '0'
    );
end entity;

architecture tic of TicTac3D is
      signal temp : Integer := 0;
      signal grid : Integer := 0;
      signal chgrid : std_logic  := '0';
      signal row : Integer := 0;
      signal chrow : std_logic  := '0';
      signal col : Integer := 0;
      signal chcol : std_logic  := '0';
      signal sigled : std_logic_vector(63 downto 0) := x"0000000000000000";
begin

      main : process(clk)
      begin

            --Full Reset, to be done after FPGA starts due to starting up
errors

            if (rst = '0') then
                  led <= x"0000000000000000";
                  trled <= "0000";
                  commsout <= '0';
```

```
      chance <= '1';
      sigled <= x"0000000000000000";
      led <= x"0000000000000000";
      chgrid <= '0';
      chrow <= '0';
      chcol <= '0';

else
      null;

end if;


--start

if (commsin = '1') then
      commsout  <= '0';
      chance <= '1';

else
      null;

end if;

--input

if (button1 = "0001") then
      grid <= 0;
      chgrid <= '1';
      trled <= button1;

elsif (button1 = "0010") then
      grid <= 1;
      chgrid <= '1';
      trled <= button1;

elsif (button1 = "0100") then
      grid <= 2;
      chgrid <= '1';
      trled <= button1;

elsif (button1 = "1000") then
      grid <= 3;
      chgrid <= '1';
      trled <= button1;

else
      null;

end if;

if (button2 = "0001") then
      row <= 0;
      chrow <= '1';
      trled <= button2;

elsif (button2 = "0010") then
      row <= 1;
      chrow <= '1';
      trled <= button2;

elsif (button2 = "0100") then
      row <= 2;
```

```
                  chrow <= '1';
                  trled <= button2;

            elsif (button2 = "1000") then
                  row <= 3;
                  chrow <= '1';
                  trled <= button2;

            else
                  null;

            end if;

            if (button3 = "0001") then
                  col <= 0;
                  chcol <= '1';
                  trled <= button3;

            elsif (button3 = "0010") then
                  col <= 1;
                  chcol <= '1';
                  trled <= button3;

            elsif (button3 = "0100") then
                  col <= 2;
                  chcol <= '1';
                  trled <= button3;

            elsif (button3 = "1000") then
                  col <= 3;
                  chcol <= '1';
                  trled <= button3;

            else
                  null;

            end if;

            --memory

            if (chgrid = '1' and chrow = '1' and chcol = '1' and commsin =
'1') then
                  sigled(16*grid + 4*row + col) <= '1';
                  commsout <= '1';
                  chance <= '0';
                  chgrid <= '0';
                  chrow <= '0';
                  chcol <= '0';

            else
                  null;

            end if;

            for i in 0 to 63 loop
                  led(i) <= sigled(i);

            end loop;

      end process main;


      --winning condition
```

```
	winner : process(clk)
	begin

		-- Reset condition
		if (rst = '0') then
			win <= '0';
		end if;

		--Checking x-axis
		for grid in 0 to 3 loop
			for row in 0 to 3 loop
				temp <= 16*grid + 4*row;
				if (sigled(temp) = '1' and sigled(temp + 1) = '1'
and sigled(temp + 2) = '1' and sigled(temp + 3) = '1') then
					win <= '1';
				end if;
			end loop;
		end loop;

		--Checking y-axis
		for grid in 0 to 3 loop
			for col in 0 to 3 loop
				temp <= 16*grid + col;
				if (sigled(temp) = '1' and sigled(temp + 1*4) = '1'
and sigled(temp + 2*4) = '1' and sigled(temp + 3*4) = '1') then
					win <= '1';
				end if;
			end loop;
		end loop;

		--Checking z-axis
		for row in 0 to 3 loop
			for col in 0 to 3 loop
				temp <= 4*row + col;
				if (sigled(temp) = '1' and sigled(temp + 1*16) = '1'
and sigled(temp + 2*16) = '1' and sigled(temp + 3*16) = '1') then
					win <= '1';
				end if;
			end loop;
		end loop;

		--Checking xy diagonals
		for grid in 0 to 3 loop
			temp <= 16*grid;
			if (sigled(temp) = '1' and sigled(temp + 1*5) = '1' and
sigled(temp + 2*5) = '1' and sigled(temp + 3*5) = '1') then
				win <= '1';
			end if;
			temp <= 16*grid + 3;
			if (sigled(temp) = '1' and sigled(temp + 1*3) = '1' and
sigled(temp + 2*3) = '1' and sigled(temp + 3*3) = '1') then
				win <= '1';
			end if;
		end loop;

		--Checking xz diagonals
		for row in 0 to 3 loop
			temp <= 4*row;
			if (sigled(temp) = '1' and sigled(temp + 1*17) = '1' and
sigled(temp + 2*17) = '1' and sigled(temp + 3*17) = '1') then
				win <= '1';
			end if;
			temp <= 4*row + 3;
```

```
                  if (sigled(temp) = '1' and sigled(temp + 1*15) = '1' and
sigled(temp + 2*15) = '1' and sigled(temp + 3*15) = '1') then
                        win <= '1';
                  end if;
            end loop;

            --Checking yz diagonals
            for col in 0 to 3 loop
                  temp <= col;
                  if (sigled(temp) = '1' and sigled(temp + 1*20) = '1' and
sigled(temp + 2*20) = '1' and sigled(temp + 3*20) = '1') then
                        win <= '1';
                  end if;
                  temp <= col + 12;
                  if (sigled(temp) = '1' and sigled(temp + 1*12) = '1' and
sigled(temp + 2*12) = '1' and sigled(temp + 3*12) = '1') then
                        win <= '1';
                  end if;
            end loop;

            --Checking xyz diagonals
            if (sigled(0) = '1' and sigled(21) = '1' and sigled(42) = '1'
and sigled(63) = '1') then
                  win <= '1';
            elsif (sigled(3) = '1' and sigled(22) = '1' and sigled(41) = '1'
and sigled(60) = '1') then
                  win <= '1';
            elsif (sigled(12) = '1' and sigled(25) = '1' and sigled(38) =
'1' and sigled(51) = '1') then
                  win <= '1';
            elsif (sigled(15) = '1' and sigled(26) = '1' and sigled(37) =
'1' and sigled(48) = '1') then
                  win <= '1';
            end if;

      end process winner;

end architecture;
```