Implementation of TicTacToe using Minimax algorithm

EE705 VLSI Design Lab

Project Report

Group Members
Anant Kumar - 203070116
Kalpit Agrawal - 203070070
Nimesh Shedge - 20307r006
Rohan Nimesh - 203070084

Accomplishments

1. **FSM Based RTL Design of the Tic Tac Toe Game using Minimax algorithm**
2. **Gate Level Synthesis of the design**
3. **Static Timing Analysis of the design**
4. **Incorporating of an IP block RAM in the design**
5. **Implementing the design to a physical FPGA board (Labsland.com)**

1. Introduction

Artificial intelligence has taken the world by surprise, be it the use of AI in complex critical computation or in solving game problems. One of such games is Tic-Tac-Toe or more commonly known as X's and O's. In which players take turns placing X and O in a 3-by-3 grid and whoever succeeds in placing their mark in three horizontally, vertically or diagonally wins.

To solve this game using AI we will be using the concept of game tree and minimax algorithm. Game tree consists of nodes and root, which is the node where the game begins. At the start of the game the root node is the empty grid with no moves made by any player yet. Each node has a second level called the children node, which calculates the best move that can be made by the player in terms of scores. Every child node has their own children nodes until all the moves have been exhausted.

According to the minimax algorithm each node is assigned a value which helps the algorithm to find the optimal move which will lead to a win or a tie. This value is decided by the final node in the game tree, here we have assumed, win will be assigned with a value of 1, loss -1 and tie a

value of 0. Best score is calculated by traversing different child nodes and simultaneously we get the best move for the AI to take.

2. Theory

2.1 Zero sum game theory

A zero sum game is a mathematical representation of a situation in which, in the case of tic tac toe, 2 players play and the game is won by one when the other side loses, in other words every move can be accustomed to one side winning and the other losing. These games are called zero sum games because, if we take a n-player game, the total gains will always be equal to the total losses. One of the algorithms to solve this problem is minimax algorithm

2.2 Minimax algorithm

Minimax is an artificial intelligence algorithm under game theory that minimizes the loss in a worst case (maximizing) situation. It is an algorithm to choose the next best move in a 2-player game. Here the two players can be considered as a maximising player “MAX” and a minimising player “MIN”. When a current position is fed to this algorithm, it generates a tree, with root node at the top, and its children node. A value is associated with each of the nodes. If the current turn is of MAX then MAX chooses the max(value of the nodes) while if MIN, MIN chooses min(value of the nodes). But as the depth of the tree increases the time to search the tree increases exponentially, fortunately tic tac toe is a small game therefore the algorithm has to search up to a depth of 9 in the worst case.

2.2.1 Example

To understand the algorithm we take an intermediate position in the game as shown in Fig-1. Note that we are starting the game from this state therefore this state is the root node with depth of 3. The algorithm senses that there are 3 places empty and can be filled by the players.

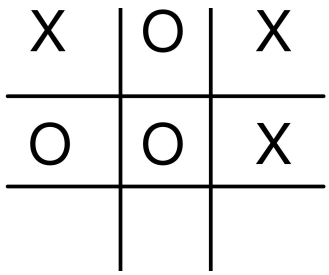


Fig-1:Root node

Assuming the next chance is of AI i.e., X’s. AI can fill one of the three spots, which will lead to addition of 3 branches resulting in 3 children nodes shown in Fig-2. Branch A, B and C, results

from the algorithm placing ‘X’ at each of the empty spots. Notice that in case of branch C, the AI wins the game as it is successful in placing 3 X’s vertically. The value assigned to the branch C is +1 corresponding to the AI winning the game.

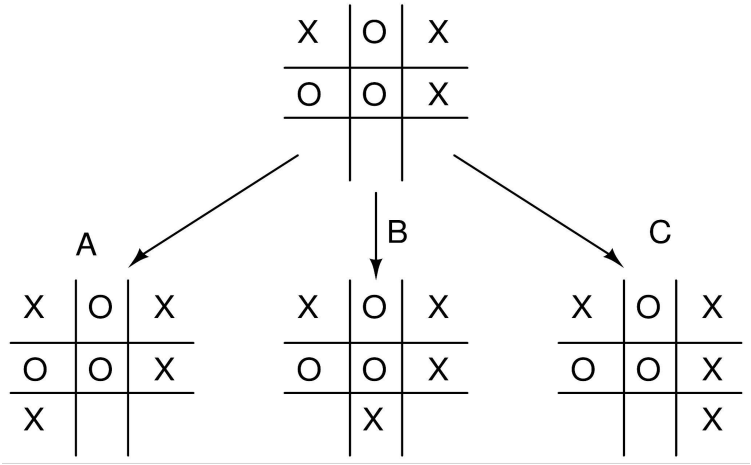


Fig-2 : Root node and its children nodes

Branches A and B have not been assigned any value because the algorithm does not yet know which of these branches will lead to a win, loose or a tie. Further children nodes can be generated by placing O’s in the empty spots, as the next chance is of ‘O’ , i.e., the minimizing player. This leads to formation of branches D, E, F and G, as shown in Fig-3, in branch D the AI loses therefore it is assigned a value of -1. Further branches are also generated by filling the spots with X’s.

All the nodes which leads a win, loss or a tie is called a terminal node and values are assigned to these terminal nodes, in Fig-3 nodes C, D, H, I and J are terminal nodes and a value is assigned to the terminal nodes on the basis of a win, loss or a tie of AI, i.e., if the AI loses then the value is -1, if wins then +1 and if it is a tie then a value of 0.

One can ask that if only terminal nodes are assigned the values then how other nodes are assigned a value so that the AI can make the best possible move. Values can not be assigned to non terminal nodes until all the choices are exhausted and the algorithm has reached all the terminal nodes possible. Let us pay attention to the nodes E, F and G in Fig-3, as these nodes are the results of all the possible moves of the minimizing player we assign the minimum value of all the children nodes of these nodes, in other words E will be assigned 0, F will be assigned +1 and G will be assigned 0 as there are no other choices.

For better understanding let's look at the nodes A and B, as these nodes are the result of all the possible moves of the maximizing player, we assign the minimum value of their children nodes to these nodes, therefore A is assigned the minimum of (-1,0) i.e., -1 and B is assigned the minimum of (+1,0) i.e., 0 as the current turn is of MIN player.

To make the best move possible the AI searches tree, in this case the best move possible is the move leading to node C. For a complete understanding, let us assume that node is not an option, then the best possible move would be the one leading to node B as it would be leading to a win (node I) or a tie (node J) rather than a loss through node A to node D.

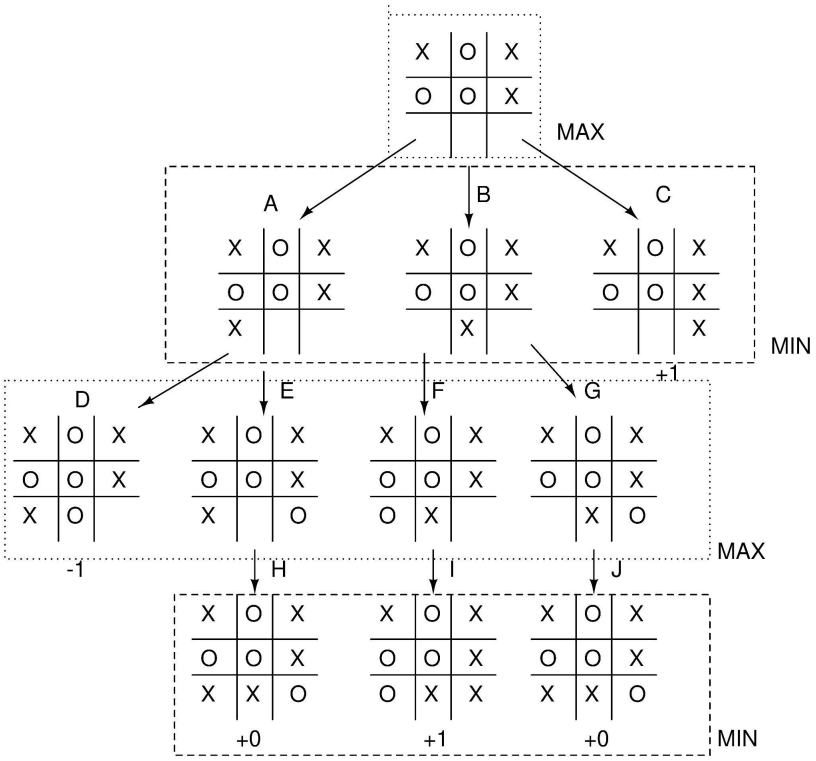


Fig-3: Complete Tree

3. Implementation

Challenge here is to design a recursion in order to fetch all possible moves and its score. But HDLs does not support recursion. VHDL does support recursion till you want to simulate it; this recursive function using VHDL is not synthesizable. In order to make it a synthesizable design which can be implemented into an actual FPGA we have used the help of FSMs and made 9 blocks(structural design) accounting for different empty choices in the board. This implementation is detailed in the below text.

3.1 Overall Block diagram

The following Block diagram shows a complete overview of the program implemented in this project.

The TicTacToe Block once started it talks to the 9th minimax element block depending on the number of cells empty in the current board(if 5 cells empty then 5th Block) by asserting whichever check_next(i) (here check_next(9)) signal of the corresponding block and rest as 0 . The 9th element block returns the best move(best_move_i,j) among the 9 possible empty cells. Until this, the TicTacToe block waits depending on the state of the 9th block until complete and done_next is 1 and best move is available.

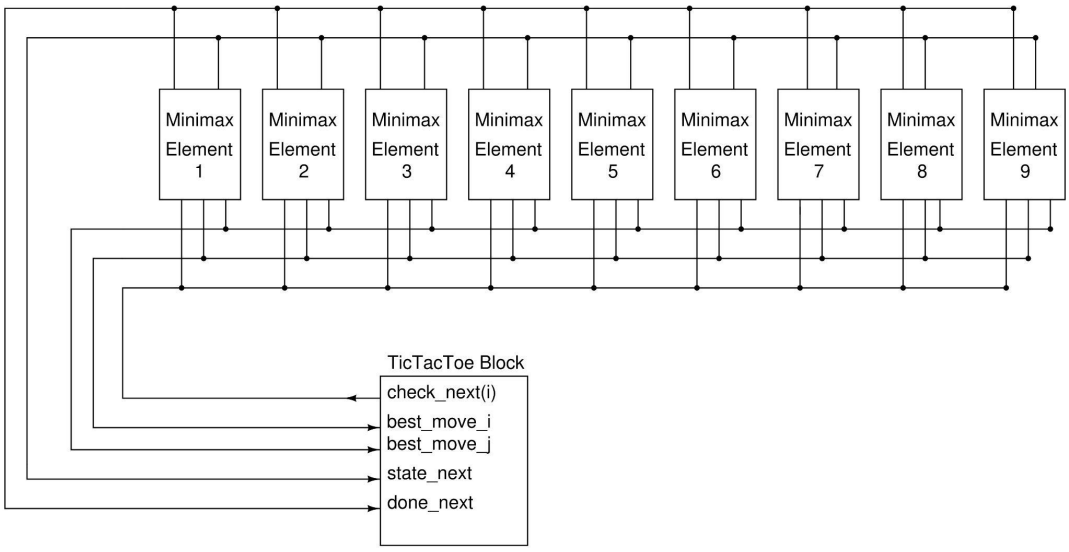


Fig-4: Block Diagram

3.2 Talk between Minimax Elements

Between two Minimax element blocks, one of the blocks acts as parent while other as child. The parent sends the current board to the child after filling the desired unfilled position of the block while the child returns the best score corresponding to its child. The parent waits until the state of the child becomes complete. A simplified schematic is shown in figure below.

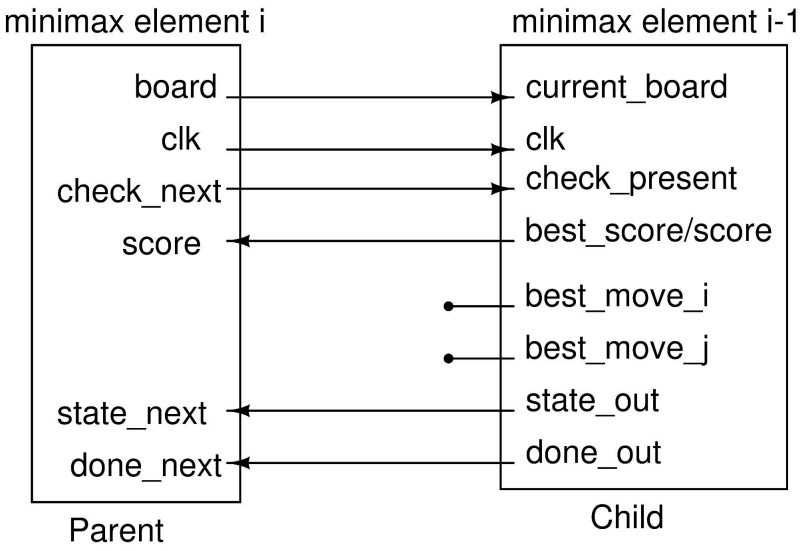


Fig-5: Interconnection between two minimax elements

3.3 The Minimax element

In the minimax element block, the signal current_board tells the program the current situation about the board. The best_move_i and best_move_j signals specifies the best moves available to AI. The score signal outputs the current best score of the node, but as mentioned above, to output the score of non terminal nodes each minimax element has to wait for the whole tree to construct. The done_out signal outputs ‘1’ when all the nodes have been checked for the best score and now the best move and best scores are available. The state_out signal outputs the current state of the program/minimax element block. A simplified block diagram of minimax element block is shown below.

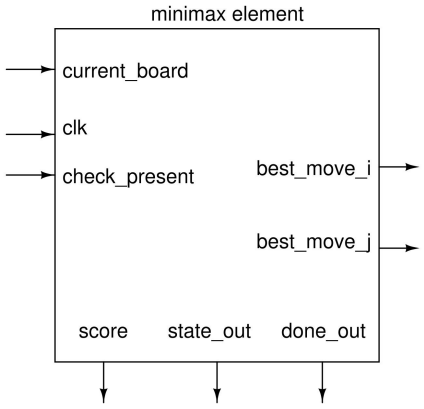


Fig-6: Minimax Element Block

3.4 Working

As mentioned before, in the worst case the depth is 9 therefore we have used 9 minimax elements as shown in Fig-4. Each minimax element has 3 input signals and 5 output signals as shown in Fig-5. After the human player plays, the situation of the current board is stored in the “board” signal in “TicTacToe block”. Now the TicTacToe block passes this current board to the desired minimax element/block from where it needs to calculate the best move (best_move_i,best_move_j).

For example the player has played and the AI is left with 5 Boxes as an option to pick one. The TicTacToe block passes this to the 5th minimax element and check_next(5) is asserted to 1 while the rest check_next are 0’s. 5th Minimax element now runs a For Loop and Fills the 5 empty cells one by one, correspondingly get its score for each traversing node by passing the current situation of the board(4 empty cells) and compares it to the previous best score and depending on that the 5th element block returns the best move for the TicTacToe block/AI to take.The corresponding minimax element(4th) places all the next possible moves one by one and passes the updated current_board signal to the next element(3rd) and this keeps on traversing until we reach the farthest leaf or 1st minimax element. Till then all the preceding elements keep waiting for the complete tree so that best scores(minimum for the next human(MIN) turn and max for upcoming AI (MAX) turn) can be assigned to each node and the root gives back the best_move_i,best_move_j to the tictactoe block.

3.5 FSM of Minimax Element Block

Following FSM Diagram shows the FSM of minimax element blocks.
The initial state of any block is “ready” once it gets a signal check_present as 1 it goes to state “start” if it is not busy. thereafter, that minimax element block runs a loop to pick one of the leftover choices on the board and then goes to state “current_check”. In current_check it checks whether this move will result in a win or loss. If Win or Loss occurs it returns back to state “start” to pick another choice giving the score as +1 or -1 for win or loss respectively. If no Win or Loss occurs then it sends the current board to the next child to get the score and hence it goes to state “fetch”. Then it waits until the next child computes the score under state “state_fetch”. Once the state of next child is “complete” it fetches the score from the next child in state “fetch_result” and returns back to state “start” to pick another empty choice and repeat it until all the choices have been used to compute the best score depending on a MIN or a MAX block/player. Once complete it shifts to state “complete” , where it resets the busy and done flag and returns back to state “ready” for the new board to compute.

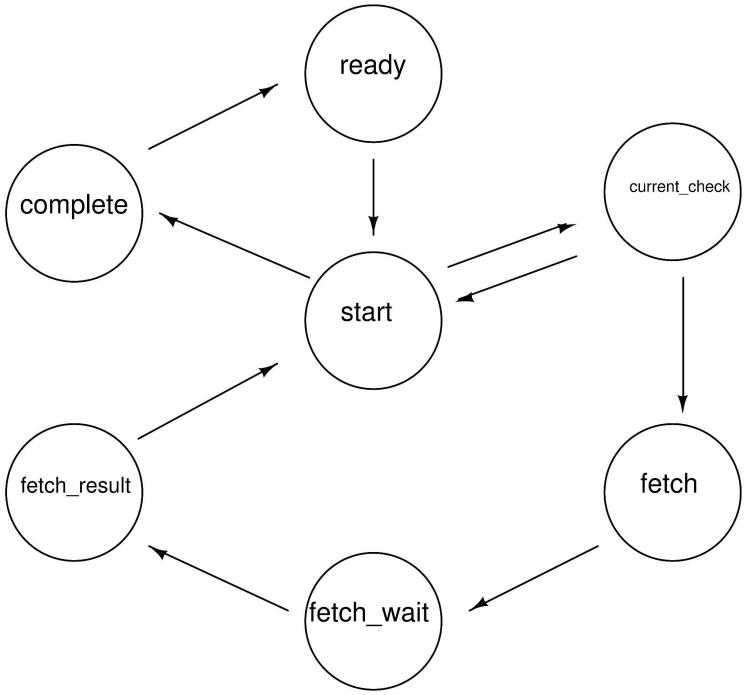


Fig-7: FSM of Minimax Block

4. Simulation

4.1 RTL using Test Bench

The following simulation was generated for two different human inputs. As you can see in the below graph the two fetch_wait which corresponds to the longest delay because the game starts with 9 empty cells, which decreases as the number of empty cells reduces.

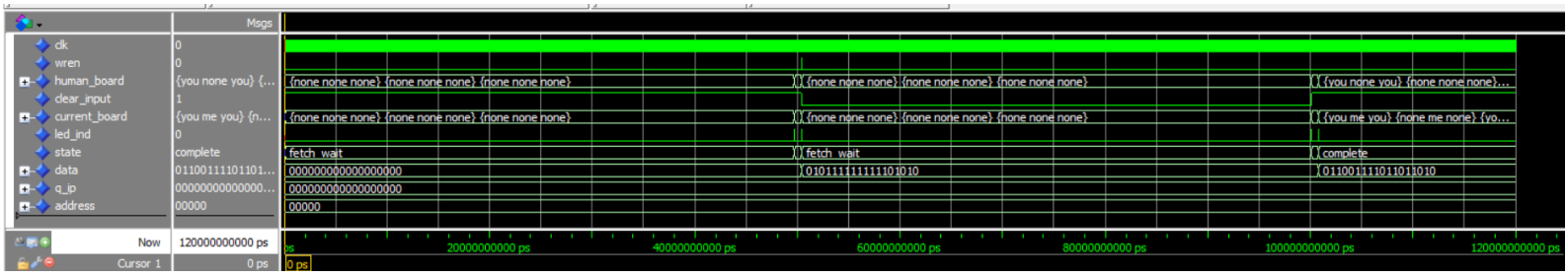
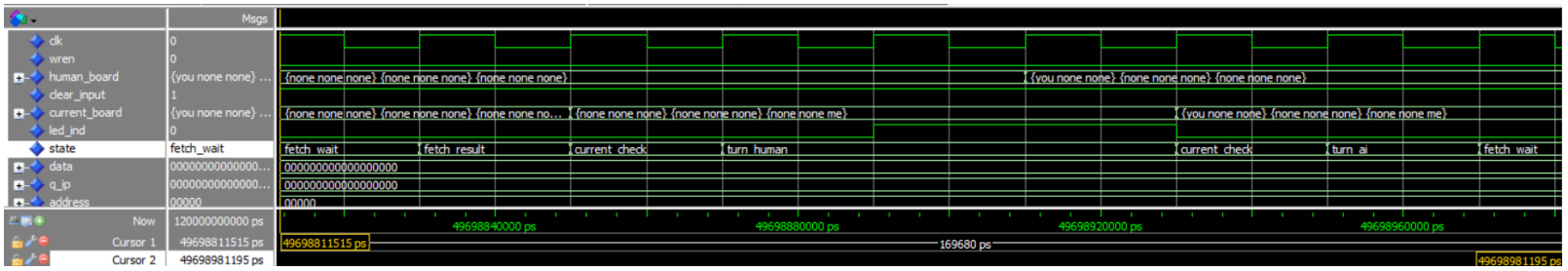


Fig-8: RTL simulation using test bench

Another simulation wave view is the zoomed in version of the above waveform where it shows three transition points. One where the AI gets the result and plays its best move while the other

where it waits for the human to play then after again the AI starts to get the best move for this human play.



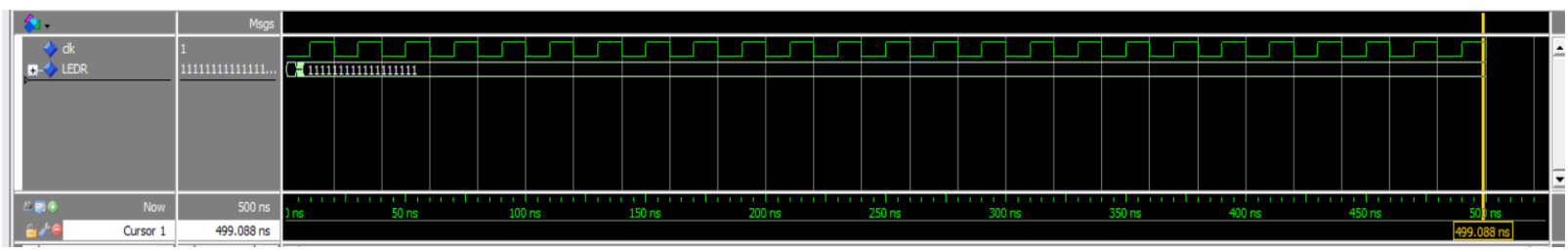
Following is the output of Modelsim Transcript.

```
Transcript
#
# vsim -t 1ps -L altera -L lpm -L sgate -L altera_mf -L altera_lnsim -L cycloneive -L rtl_work -L work -voptargs="+acc"  tictactoe_tb
# vsim -t 1ps -L altera -L lpm -L sgate -L altera_mf -L altera_lnsim -L cycloneive -L rtl_work -L work -voptargs=""+acc"" tictactoe_tb
# Start time: 19:58:32 on May 16,2021
# Loading std.standard
# Loading std.textio(body)
# Loading ieee.std_logic_1164(body)
# Loading ieee.numeric_std(body)
# Loading work.commonpackage(body)
# Loading work.tictactoe_tb(sim)
# Loading work.minimaxelemtpkg
# Loading work.tictactoe(rtl)
# Loading work.minimaxelemt_1(rtl)
# Loading work.minimaxelemt_2(rtl)
# Loading work.minimaxelemt_3(rtl)
# Loading work.minimaxelemt_4(rtl)
# Loading work.minimaxelemt_5(rtl)
# Loading work.minimaxelemt_6(rtl)
# Loading work.minimaxelemt_7(rtl)
# Loading work.minimaxelemt_8(rtl)
# Loading work.minimaxelemt_9(rtl)
#
# add wave *
# view structure
# .main_pane.structure.interior.cs.body.struct
# view signals
# .main_pane.objects.interior.cs.body.tree
# run 120 ms
# ** Note: Win
#    Time: 50433110 ns  Iteration: 0  Instance: /tictactoe_tb/dut
# ** Note: Win
#    Time: 100833550 ns  Iteration: 0  Instance: /tictactoe_tb/dut

VSIM 2>
```

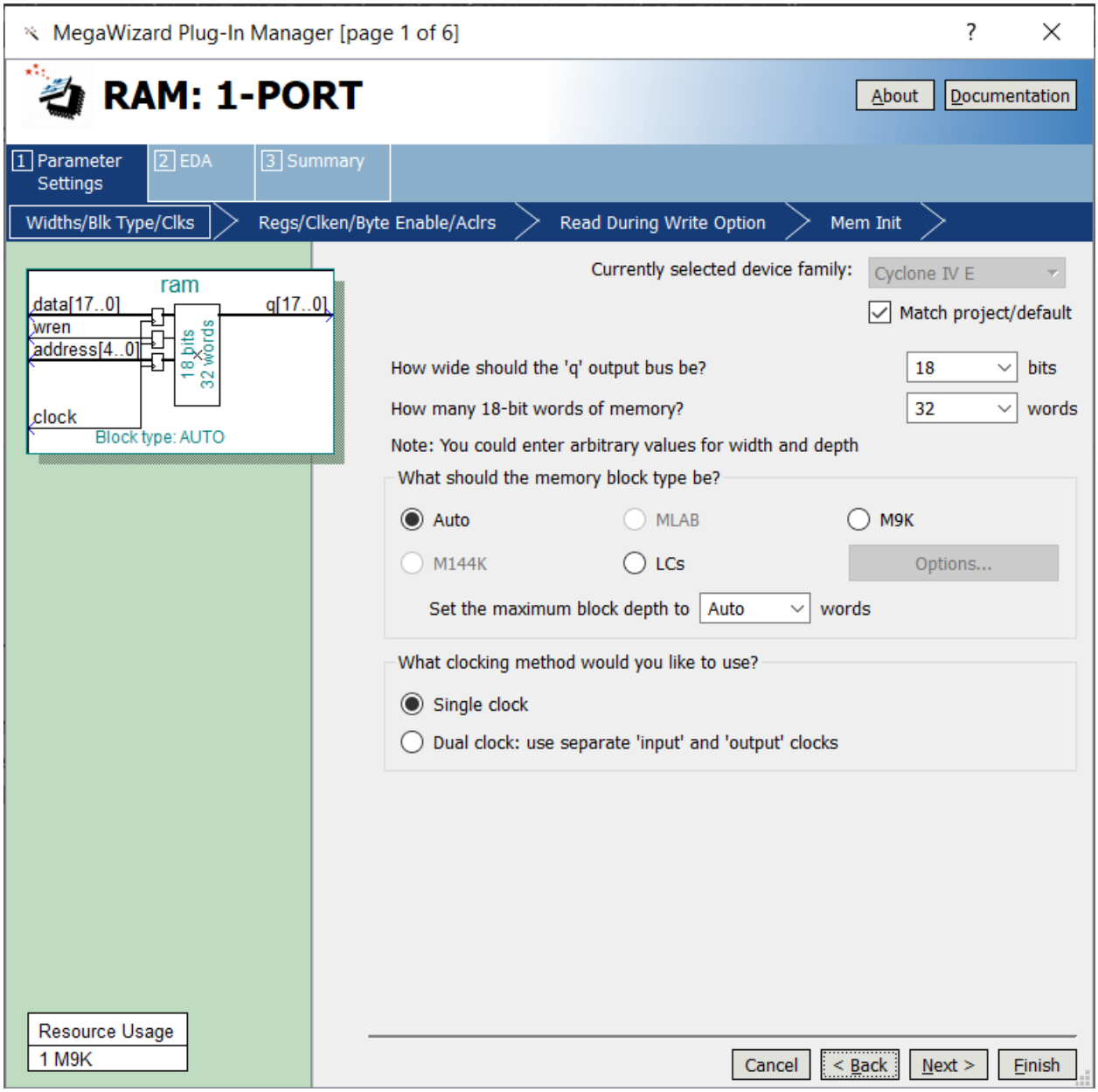
4.2 Gate Level Simulation

The output of the gate level simulation shown below depicts two signals clk and LEDR(17:0). LEDR is the current state of the board and clk is input to the tictactoe_game entity.



4.3 Using Quartus IP catalog

We have used a quartus IP block of RAM in our design of minimax. The RAM block is used to store the final status of the board for checking the history. Here we have used a single clock RAM.
Since we have a total 9 places on board with 3 options as input for each place we have used an 18 bit wide data bus and we have chosen to take the minimum number of words in the memory, so address bus is 5 bit wide or 32 words total.



4.4 Static timing analysis

After doing multiple iterations in Quartus timing analyzer,

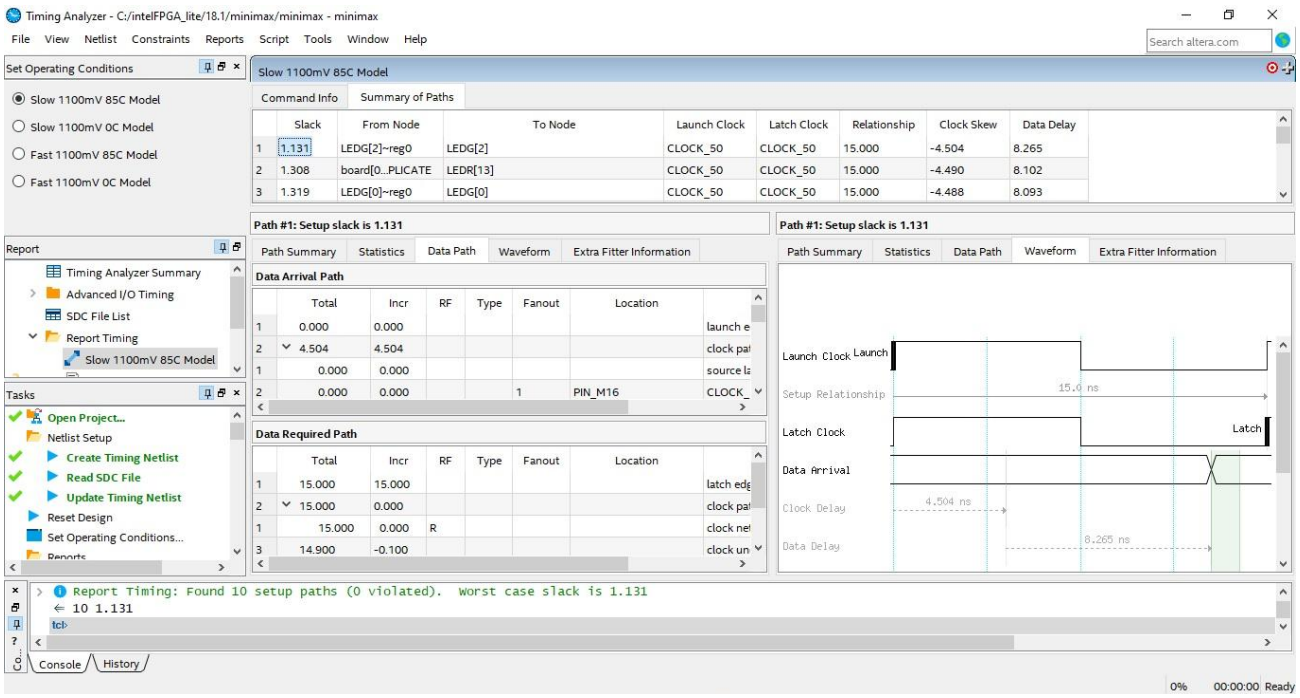


Fig-9: Static Timing analysis

Model	Setup time slack	Hold time slack	Fmax
Slow 1100 mV 85C	1.131 ns	0.284 ns	72.1 MHz (timing analyzer)

4.5 Labsland FPGA

We modified the code such that it would run on Labsland FPGA. To play the game on DE2-115 board on labsland we need to understand how the input is provide, what is the current board position and how to clear the board for a new game

4.5.1 Providing Inputs

As we know TicTacToe is a 3x3 grid game, so we need 9 switches to provide the input. The switch configuration is shown in Fig- .

SW8	SW7	SW6
SW5	SW4	SW3
SW2	SW1	SW0

Fig-9 : Switch configuration

4.5.2 Current Board position

In addition to providing inputs we need to know the current board position, which we have chosen to show using red LEDs namely LEDR0 to LEDR17. The LED configuration is shown in Fig-10.

LEDR 16/17	LEDR 14/15	LEDR 12/13
LEDR 10/11	LEDR 8/9	LEDR 6/7
LEDR 4/5	LEDR 2/3	LEDR 0/1

Fig-10: LEDR configuration

Odd numbered LEDs including LEDR0 are OFF when the board is occupied by AI's move i.e., X. Even numbered LEDs are OFF when the board is occupied by the human's move i.e., O. When AI has won the game a green LED named LEDG1 becomes ON. To reset the game press KEY0.

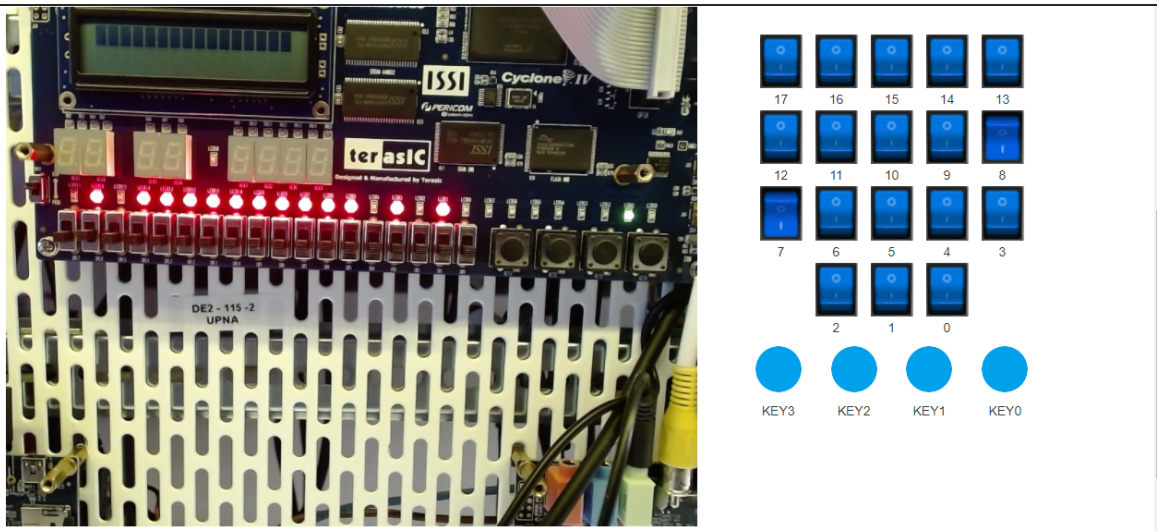


Fig-11: A game of Tic Tac Toe on DE2-115 on Labsland

5. Future Work

As explained above, the minimax algorithm is a search algorithm. It constructs the tree then assigns the scores to each node and calculates the best possible move. This calculation takes time, in order to decrease the delay of this algorithm we can implement alpha beta pruning in the programming.

To understand alpha beta pruning let us redirect our attention to the example taken in [2.2.1]. Pay attention to scores of nodes A, B and C which are -1, 0 and +1 respectively. To make the best possible move the algorithm chooses the branch leading to C i.e., maximum of scores of A, B and C (max{-1,0,+1}). It is evident that after the calculator of the score of C, the algorithm need not calculate the scores of A and B because C has the highest possible score.

References

1. Wikipedia - Zero Sum Game
URL-https://en.wikipedia.org/wiki/Zero-sum_game#:~:text=In%20game%20theory%20and%20economic,they%20will%20sum%20to%20zero (accessed on 12.05.21)

2. Wikipedia - Minimax URL-https://en.wikipedia.org/wiki/Minimax (accessed on 12.05.21)

3. Minimax Algorithm in Game Theory
URL-https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-3-tic-tac-toe-ai-finding-optimal-move/ (accessed on 2.05.21)