Tic-Tac-Toe Game using FPGA

Surya Pramod(ES16BTECH11015)
Dinesh Reddy(EE15BTECH11007)

1 INTRODUCTION

Tic Tac Toe is a very popular paper-and-pencil game in a 3x3 grid for two players. The player who makes the first three of their marks in a diagonal, vertical, or horizontal row wins the game.
Rules of the game:
It's a two player game. When any of the player plays the game, a 2-bit value is stored into one of the nine positions in the 3x3 grid like Xs/Os in the real paper-pencil version.
The player plays the game by pressing the corresponding button. Red/Green LED is lit in a position when the position is played by the players respectively. The players wins the game when successfully placing three similar Xs/Os values in the following row 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).
The winner detecting circuit is designed to find the winner when the above winning rule is matched.
To detect an illegal move, a comparator is needed to check if the current position was already played by either of the players.
Moreover, "No space" detector is to check if all the positions are played and no winner is found.

2 Pre-requisites

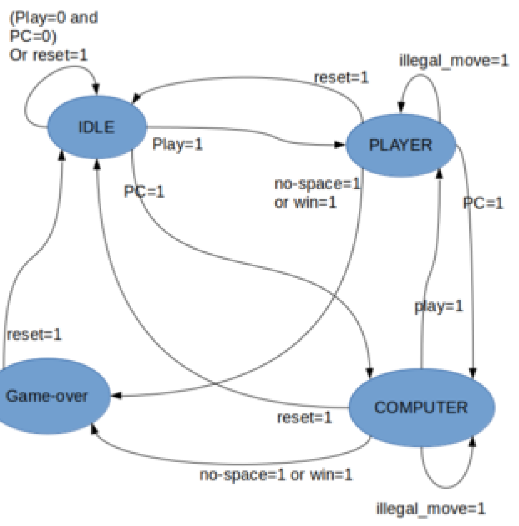
1. Raspberry pi, Ico board, miscellaneous electrical components.
2. Fair knowledge on verilog HDL.
3. Should be comfortable with simulating verilog HDL projects in their favourite simulator.
4. Understand FSM.

3 Setup

1. Please install iverilog or any other verilog HDL simulator. Here, I am using iverilog.
   $ sudo apt-get install iverilog
2. Install the required packages to work on Ico board by following the *Document*
3. Get a local repository of our project from github
   $ git pull https://github.com/CHINTADINESH /tic_tac_toe_extension

4 Verilog Code

To control the Tic-Tac-Toe game, a FSM controller is designed as shown below.



1.IDLE(00): When waiting for the players to play or when resetting the circuit, the FSM is at the IDLE state.
2.PLAYER(01): The player turns to play and "01" to be stored into the decoded position.
3.COMPUTER(10): The computer turns to play and "01" to be stored into the decoded position.
4. Game-over(11): The game is finished when there is a winner or no more space to play.
Inputs of the controller of the Tic-Tac-Toe game
a. Reset :

- Reset = 1 : Reset the game when in the Game-Over state.
- Reset = 0 : The game begins.

b. Play:

- Play = 1 : When in the IDLE state, play = 1 is to switch the controller to the PLAYER state and the player plays.

- Play = 0 : Stay in the IDLE state.

c. PC :

- PC = 1 : When in COMPUTER state, PC = 1 is to switch to the IDLE state and the computer plays.
- PC = 0 : stay in COMPUTER state.

d. Illegal-move :

- Illegal-move = 0 : When in PLAYER state, Illegal-move = 0 is to switch to COMPUTER state and let computer plays when PC = 1.
- Illegal-move = 1 : Illegal moving from the player/computer and switch to the IDLE state.

e. No-space :

- No-space = 0 : still have space to play, continue the game.
- No-space = 1 : no more space to play, game over, and need to reset the game before playing again.

f. Win :

- Win = 0 : Still waiting for the winner.
- Win = 1 : There is a winner, finish the game, and need to reset the game before playing again.

5 Simulation

1. We can use iverilog to simulate the project. You are however free to use any simulator.
   $ iverilog -o tic_tac_toe tb_tttg.v tttg.v

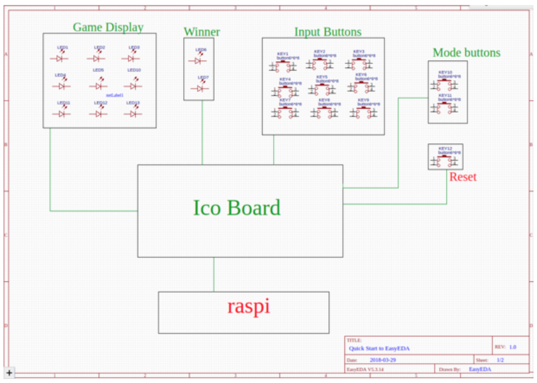
   ```
   dinesh@UB18:~/Desktop/tic_tac_toe_
   extension$ ls
   Makefile tb_tttg.v
   tic_tac_toe_extension2x2.tar.gz tttg.v
   README.md tic_tac_toe_extension2x2
   tttg.pcf
   dinesh@UB18: /Desktop/tic_tac_toe_
   extension$ iverilog -o tic_tac_toe
   tb_tttg.v tttg.v
   ```

2. If iverilog works properly, you will get an binary named tic_tac_toe in the project directory.

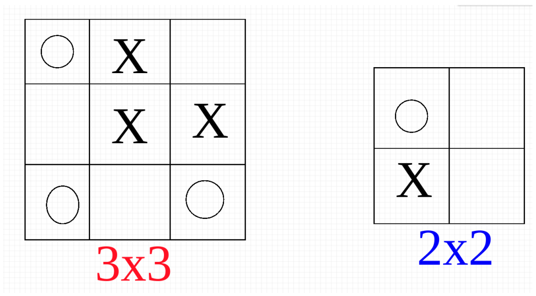
   ```
   dinesh@UB18:~/Desktop/tic_tac_toe_
   extension$ ls
   Makefile tb_tttg.v
   tic_tac_toe_extension2x2
   tttg.pcf README.md tic_tac_toe
   tic_tac_toe_extension2x2.tar.gz tttg.v
   ```

3. Now execute the binary.
   $ ./tic_tac_toe

6 Circuit Diagram



7 3x3 vs 2x2 variants of the game



8 Synthesis

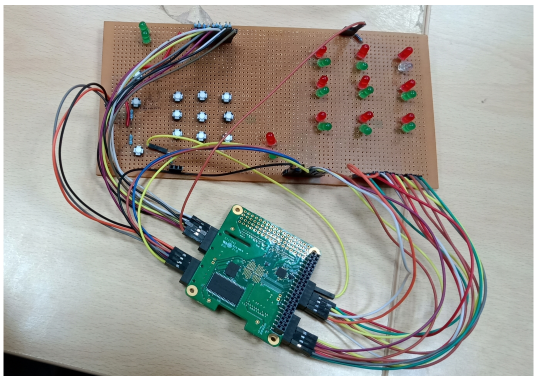
If you have followed the above mentioned *document* Then you are all set to synthesize the project on icoboard. The following command will dump the verilog on the ico board.
$ make v_fname=tttg

```
............................
............................
............................
After placement:
PIOs 31 / 206
PLBs 37 / 960
BRAMs 0 / 32
place time 4.13s
route...
pass 1, 1 shared.
pass 2, 0 shared.
After routing:
span_4 198 / 29696
span_12 59 / 5632
route time 5.99s
write_txt tttg.asc...
icetime -d hx8k -c 25 tttg.asc
// Reading input .asc file..
// Reading 8k chipdb file..
// Creating timing netlist..
// Timing estimate:  9.93 ns (100.68 MHz)
// Checking 40.00 ns (25.00 MHz) clock
constraint:  PASSED.
icepack tttg.asc tttg.bin
```

```
icoprog -p tttg.bin
reset..
cdone:  low
programming..
cdone:  low
```

The output should be something like shown above.

9 3x3 variant of the game



10 2x2 variant of the game