EE 354 Project Report: PvP Tic-Tac-Toe

Haotian Xu, Shuangge Wang, Tianyi Yang
Ming Hsieh Dept. of Electrical and Computer Engineering

Viterbi School of Engineering
University of Southern California
Los Angeles, USA
{horacexu, larrywan, tianyiya}@usc.edu

***Abstract*—In this project, our team will build a player-versus-player tic-tac-toe game. We will be using NEXYS 4 FPGA board to accomplish the design. The players can control the game using the buttons on the FPGA board, and the game can be visually displayed on a monitor screen via a VGA cable.**

***Index Terms*—Tic-Tac-Toe, XEXYS 4 FPGA, State Machine**

I. Introduction and Background

The game of tic-tac-toe is a game for two players who take turns marking the spaces in a three-by-three grid with X or O. The player who succeeds in placing three of their marks in a horizontal, vertical, or diagonal row is the winner. It is a solved game, with a forced draw assuming best play from both players. Fig. 1 is a visual representation of the game.

The knowledge used in our project comes from our previous hard work this semester. We use VGA Moving Block files as the basic framework and revise based on them. We referred to the GCD lab while designing our state machine, and we use the constraint file from the 8-bit Divider lab.

II. Design

A. Objective and functions

The main functions of our project is to design a platform for player-versus-player tic-tac-toe game on a FPGA form factor. We will use verilog to program the Nexys 4 FPGA board, fig. 3. Our code structure includes three verilog files, including block.v, top.v, and display_controller.v, and two constrains files, including divider_8_top.xdc and nexys4.xdc[1]. In our game, we assume that the players are well aware of the rules of tic-tac-toe and are rational so that they wouldn't choose to position their chess on a location that's already occupied.

B. State machine

The state diagram consists of 7 states, including Q_{INIT}, $Q_{WAIT1PRESS}$, $Q_{WAIT1RELEASE}$, $Q_{WAIT2PRESS}$, $Q_{WAIT2RELEASE}$, Q_{WIN}, and Q_{DRAW}. Fig. 2 is our design of the state diagram of our game. It is a Mealy state machine.

As the machine starts, the state goes to Q_{INIT}, where all registers are cleared and initialized. Then, the state will transfer to $Q_{WAIT1RELEASE}$ or $Q_{WAIT2RELEASE}$ depending on the status of Player1 (Sw0). These two states correspond to the 1st player and the 2nd player turns. In the $Q_{WAIT1RELEASE}$ or $Q_{WAIT2RELEASE}$ states, if the player presses a button to move the pointer (the position of the chess), the state will go to $Q_{WAIT1PRESS}$ or $Q_{WAIT2PRESS}$ respectively. After the button is released, the state will go back to $Q_{WAIT1RELEASE}$ or $Q_{WAIT2RELEASE}$. These two states can also switch between each other after the player confirms his choice by changing Player1 (turning Sw0). These two states also check for winning or drawing conditions. If any player wins or there is a draw, the state will go to Q_{WIN} or Q_{DRAW}. We also introduced two OFLs, WIN1 and WIN2, indicating the winning condition of Player1 and Player2. These OFLs could be used as tie breakers in the Q_{WIN} state. At any time of the game, if the reset button (BtnC) is pressed, the state will go back to Q_{INIT}, and the game could be restarted.

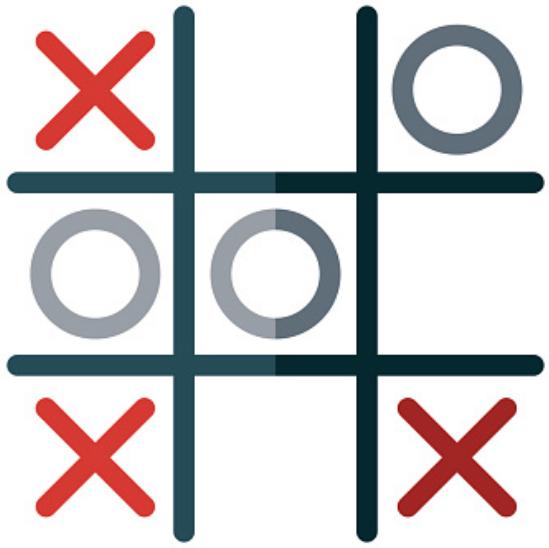

Fig. 1. Tic-Tac-Toe

[1]All source codes to our implementation of tic-tac-toe can be found online at https://github.com/1610165797/verilog-tic-tac-toe

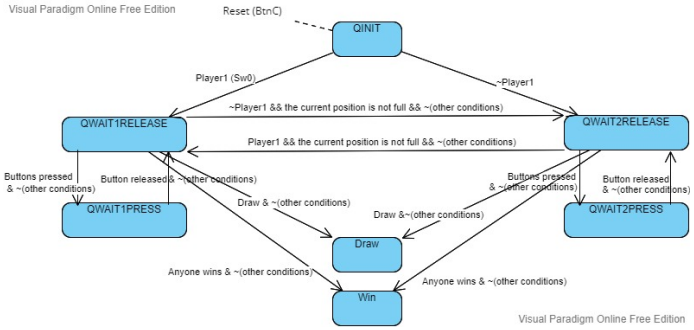

Fig. 2. State Diagram

C. User Interface

As the game appears on the screen, users can use buttons and switches on the Nexy4 board to control the game, and SSDs can show who should play and the game's results (Player1 win, Player2 win, and draw). The central button (BtnC) is used to reset the game. The up, down, left, and right buttons can be used to move the pointer in corresponding directions. Once the player selects the position through buttons where he wants to put his piece, he can change the switch 0 (Sw0) to confirm his actions and move to the other player's turn. The 2nd player can follow the same procedure to put his piece and switch back to the 1st player. If the 1st player is playing, the SSD1 and SSD0 will display "04". If the 2nd player is playing, the SSD1 and SSD0 will display "10". If one of the players win the game, the SSD1 and SSD0 will display "20". If there is a draw, the SSD1 and SSD0 will display "40". The SSD4 can display who wins the game or if there is a draw. When SSD4 display "1", it means the 1st player wins. When SSD4 display "2", it means the 2nd player wins. When SSD4 display "3", it means there is a draw.

D. Display

The first component we want to display is the chessboard. We want to display a 3×3 tic-tac-toe chess board. This is achieved by drawing 9 separate squares. This board will be displayed and fixed on the screen upon initialization of the state machine.

The second component we want to display is the crosshair, which can be moved to the desired position by each player to posit their chess. We will assign the cross, $+$, to player 1 and circle, $\circ$, to player 2. Once the player finalized their decision on which location to place their chess, we will update the chess board accordingly on the display.

The third component we want to display is the chess, which is stored in two 9-bit registers. They describes the current positioning of all of the chess on the board.

The last component we want to display is the message for game result. Once the state moves to done state, we will display a message on the SSD that either says 1 (Player 1 Wins), 2 (Player 2 Wins), or 3 (Draw).

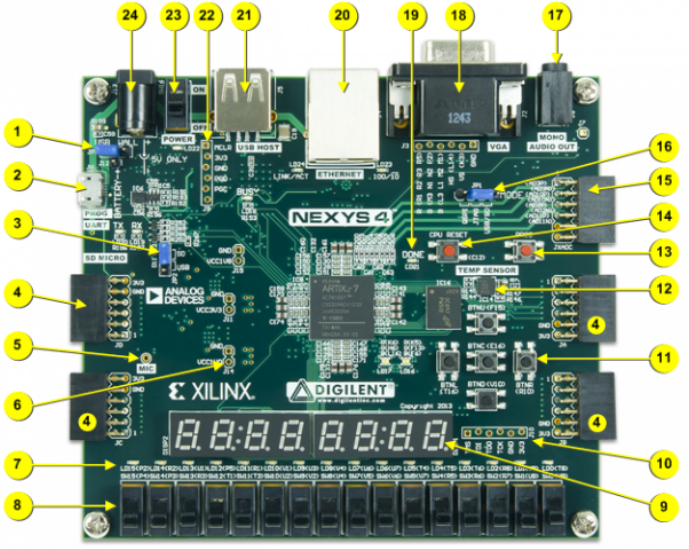

Fig. 3. Nexys 4

III. Test Methodology

First, we divided our testing methodology into two parts. In the first part we tested the state machine design. We set multiple states for both player to use the Nexys 4 board's buttons to control the board pointer and also use the switches on the Nexys 4 to switch rounds between the two players. Before we moved to test the state machine on the VGA display, we used the SSDs on the Nexys 4 board to display state information and player number in order to check if the all the states are entered, exited and transitioned appropriately. During this process, we found and fixed a number of bugs that leads to the Tic-Tac-Toe machine not being initialized and variables not resetting correctly. We assigned a two digit hexadecimal code to each state so we can visually see it on the SSDs. We verified all cases for the game results, including player 1 win, player 2 win and draw. For the second part we tested the VGA display. We built and refurbished our game board from repeated testing and verification in this part, because programming the display requires careful adjustment on the horizontal and vertical axis values. We also tested the pointers and symbols for each player on the display by making tiny tweaks and adjustments to our programmed shapes. Finally, we merged the state machine design and the VGA control to test the whole game, and during testing we debugged a few minor issues relating to state transitions and moving the pointer.

IV. Conclusion and Future Work

During the process of building this project, we were able to take the ideas and methodologies from previous labs and integrate them to fit our game' need. We also learned from scratch how to use and program the VGA display. I think our Tic-Tac-Toe design can be enhanced by making the user interface more aesthetically pleasing. This can be done by exploring more features and techniques in programming the

VGA display. Moreover, the core design of our game can be applied to a larger game board and future students of this class can potentially develop a Gobang game or games of similar rules based on the logic of our tic-tac-toe game. For this semester's lab, I think the experience has been interesting and useful. The labs are very good practices to learn how to program in Verilog and how to use synthesize and simulation tools. We also explored many functionalities of the Nexys 4 board. However, I think that in some labs the instructions are not as explicit and clear as other labs and we spent more time trying to figure out what to do. In some cases we have videos guiding us step by step but in others we might have to create files or modules with very few guidance. In general, I think we gained a lot of hands-on experience and skills with hardware programming in the lab, which is extremely important for this course.

V. Appendix

This is the repository to our code: https://github.com/1610165797/verilog-tic-tac-toe.