Tic-Tac-Toe using an input interface (keyboard) and a VGA screen as an output interface

Final Report

Angel Bautista Romero, Antonio Montagna, Paolo Nikaj, Ian Betz
Electrical and Computer Engineering Department
School of Engineering and Computer Science
Oakland University, Rochester, MI
e-mails: bautistaromero@oakland.edu, amontagna@oakland.edu, pnikaj@oakland.edu,ibetz@oakland.edu

1. ABSTRACT

The intent of this project is to help students understand and apply the knowledge earned in the class "Computer Hardware Design" by applying the topics gathered throughout the term. A tic tac toe game is going to be presented, the game will include all the requirements asked on the guidance such as FSM to deploy the logic of the game, Datapath, to store and process information, and demonstration as a proof to verify the digital integration and an external interface with the use of a keyboard and VGA. The topic of the game as well as the inputs and outputs will be discussed in the following part of this report.

I. INTRODUCTION

A Tic-Tac-Toe was created using a Nexys Artix-7 FPGA board, in addition to a keyboard and a vga connected screen. The vga screen will display distinct colors to differentiate the two different players of Tic-Tac-Toe game board, and the keyboard will allow the player to select which box their game piece will be placed in. In the creation of this project we will first create a two player mode which will allow users to play on the vga screen. In the later stages of development of our project, a game process of winning was implemented with the use of LEDs. The team members of this project also sought the creation of a single player version which will allow the player to play against the computer. This will require additional code for the artificial player to determine where to place its game pieces.

In figure 1, a block diagram for our project is shown. This block diagram includes a Keyboard as input, a game memory to store the keyboard input, game process to decide what is the current decision of the game, an FSM to control important decisions, a game output to get ready the information given by the game process, and at the end a VGA display to show the current status of the game.

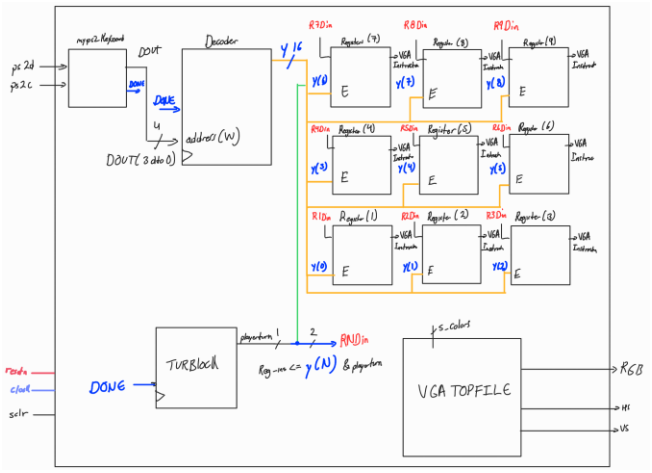


Figure 1. Tic Tac Toe Top File

2. II. METHODOLOGY

Input

The game input was interfaced through a keyboard connected to the Nexys Artix-7 FPGA board over PS2. The number pad is mapped 1 to 1 to the tiles on a tic-tac-toe board so that selecting number "9" will select the top right cell on the board for example as shown in figure 3. The registers will be enabled when the "done" signal is processed once the keyboard is pressed to actuate the decoder. It is important to note that the "my_ps2keyboard" circuit has been given by Llamocca's website [1].

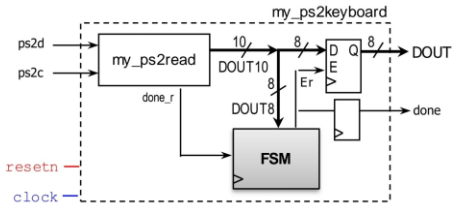


Figure 2. my_ps2kayboard circuit

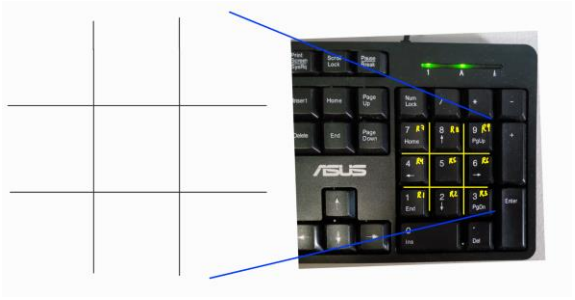


Figure 3. Input Memory logic

The input processing block then enters data into nine registers by using a decoder registering the four Least significant bits from “my_ps2keyboard” as shown in figure 4. The decoder will enable the selected register. Each of these registers contains two bits of input data from the turn block and the signal from the output of the decoder to indicate which register the information is going. The first bit will indicate whether the cell has been selected by either player or is still blank and available to be claimed. The second bit will indicate which player has chosen the cell. The value of this field will be irrelevant if the first bit does not indicate that it has been chosen. Data from the number pad is fed the same to all registers but the decoder block will decide which registers will have their value updated via the enable signals. The outputs from these registers will be combined into one eighteen bit number that the VGA display logic will use to determine which cells are displayed as which color.

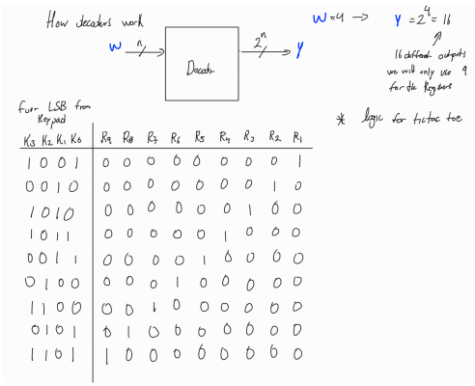


Figure 4. Decoder set of values

Figure 4 shows how the decoder will work according to the different outputs of the keyboard assignment. These values are unique and easy to follow depending on which section of the game is being played.

Game Logic

The game starts off with a clear board and ready for player one to make a move. Once a move is made and an output is received from the keyboard, the “Turn Block” will alternate a bit from high to low signaling that the next move is for player two. Once another output is received from the keyboard, then the “Turn Block” will go back from low to high. The keyboard outputs are decoded and assigned to 9 registers that represent the game board and each hold 2 bits. One bit tells which player is making the turn and the other bit tells the VGA to enable the specific block for its corresponding location. The player bit is read by the VGA as the instruction for what color to display in the block. So far, we are running the game in an infinite loop where the two players have to be responsible for not overriding each other’s moves. The players are also responsible for resetting the game with the reset button after each game. We are doing this to make sure that we get a functioning game first. Once we get this nailed down, we will work on creating additional logic instructions that account for the possible outcomes and displaying the winner either through the onboard LEDs or the seven-segment displays. The next step we will take will be to implement a code that will run a check on whether the blocks are already occupied by a player’s move so that an override is not possible.

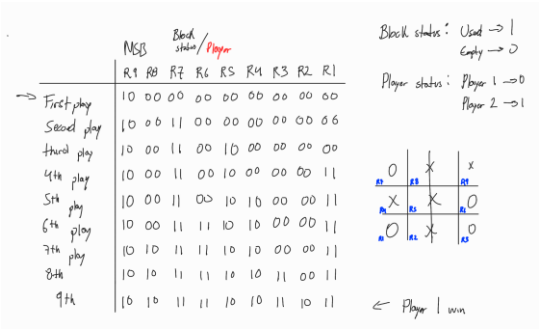
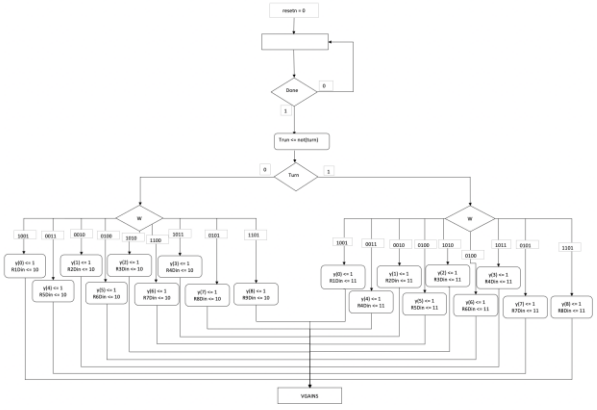


Figure 5. Truth table from a game

FSM of the game logic



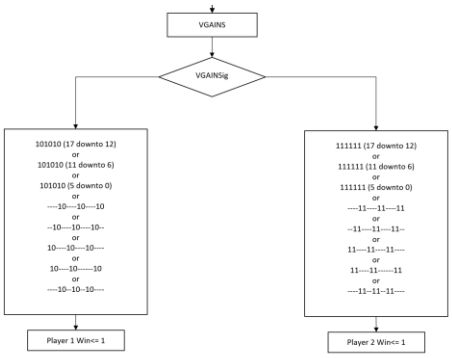


Figure 6. FSM game logic

Figure xx represents how the game logic is operational when the game is started. In the first state, the game will start after a "resetn" is pressed, the following process will wait until the "done" signal from the "my_ps2keyboard" signal is actuated, this happens when a key from the numpath is pressed. this it will actuate the game logic as well as the register selection of the memory. Depending on what the decoder is registered, the game logic will indicate the input instruction to the chosen register, this will contain two bits and they will vary depending on the player currently playing.

18 generated values will keep the current values of used registers as well as change values of unused registers. This will generate the different sections on the vga. After 18 bits are generated on the "VGAins" signal. The FSM block will concatenate the values and send them on a signal to start the win process of the game. In this process, the FSM will determine when a player has successfully completed a series of follow straight three areas as a traditional Tic Tac Toe. Once there is a winner, the FSM will instantly update a series of LEDs on the FPGA to choose a winner. After there is a winner, the game will be needed to start from zero, thus the player will reset the board of the game.

Output

The output from the game logic block of code will be in the form of an eighteen bit number, this eighteen bit number will contain all of the information needed to display the game. Each register's output will be two bits, the first bit will say whether or not the registers are assigned space is blank, while the second bit will signify which player has occupied the space, if it is occupied. This eighteen bit value will be the combined values outputted from nine registers, register nine being at the most significant bit, and register one at the least significant bit. The output value will be used in order to track the games progress as well as to inform the VGA Display.

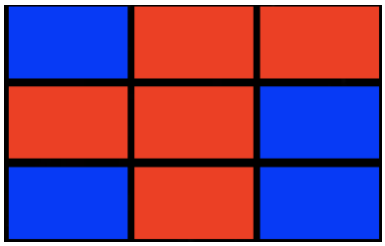


Figure 7. VGA Display for Sample Game

VGA Display

The VGA display portion of this project is executed by retrieving the eighteen bit code that is outputted by the game logic component outlined above in the Output section. This binary value is then deconstructed in order to display the gameplay. The first two bits, least significant, come from register one, which represents the bottom right space on the tic-tac-toe game board. Bit zero represents which player occupies the space, and bit one represents whether or not the space is occupied. Similar logic follows the remaining registers, with register nine being represented by bits 16 and 17. Register nine is used for the upper right space on the gameboard. The eighteen bit output is continuously updated as the game progresses. The VGA display is divided into nine ranges, one given to each space of the gameboard. These spaces will remain blank (black) until occupied by a player, then they will be filled with their player's color, i.e. red for player one and blue for player two. The VGA will continue to be filled as the players make inputs, until the game is finished and reset, at which time the display will be cleared and reset back to a blank screen.

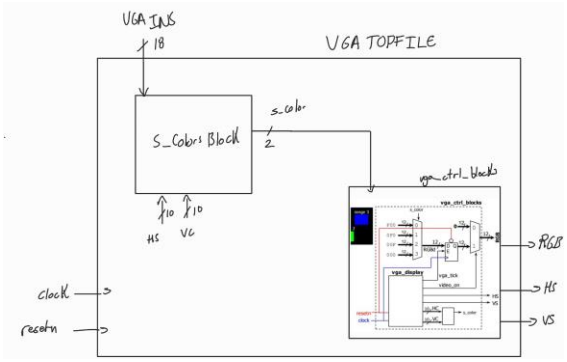


Figure 8. VGA Circuit Diagram

3. III. Experimental Setup

The game will be tested by connecting the display to the VGA port found on the Nexys Artix-7 FPGA board. Then the keyboard will be connected to the board via a PS2 to USB type-A adapter. The board itself will be powered through the micro USB port that is also used for programming.

We will then test a number of different conditions and make sure that each player can still claim each cell. The

game needs to be efficient enough to run without much delay in the time it takes to reflect a player's choice but given the processing power at our disposal with these FPGA boards, that should not be a significant roadblock.

Virtual testing in Vivado was also performed to ensure that the registers would store values as expected. In figure seven below, we see that the commands are being simulated for each of the cells and the register values are changing accordingly. The turn flip-flop is also alternating as expected which changes what color is being displayed. The PS2 logic block was bypassed in order to more easily simulate the data coming into the decoder. Only the "data in" and "done" signals were simulated in the test bench (aside from clock and resetn). Other simulations also showed that the win condition logic with LED's worked which was helpful instead of playing the game each time in order to test that it worked.

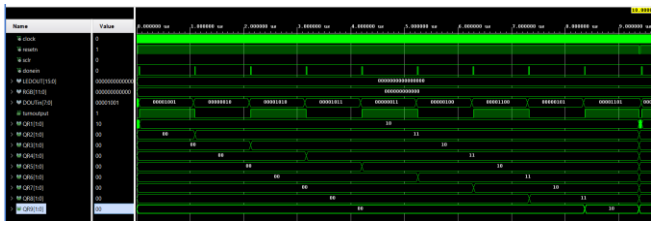


Figure 9: Testbench simulation

IV. Results

The main result that was desired from this project was the ability to play a game of tic-tac-toe with two players on a computer monitor using a keyboard as input. From a purely subjective analysis, we have achieved this goal. We could look at the response time of the game to determine how much latency is experienced between a button press and the corresponding cell changing state on screen to determine the efficiency of the game state logic for a quantitative result. This analysis may not be necessary though as the circuit is mostly combinational and the latency is not perceptible to the human eye. As far as anyone can tell, the screen updates upon an input instantaneously.

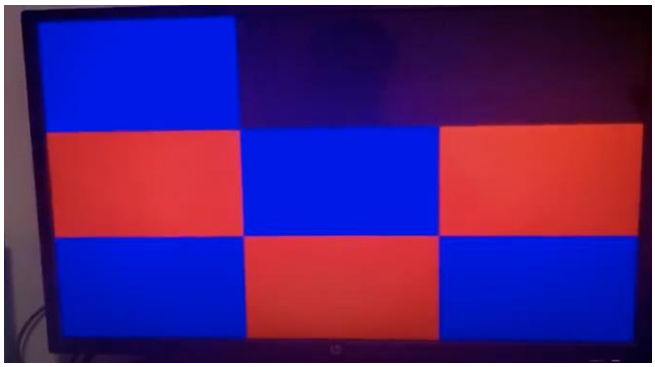


Figure 10: Game in Action

Conclusions

This project could be further improved by implementing the following features.

1. AI Opponent
2. Real X's and O's
3. Logic to handle games that end in a draw
4. Ability to keep score over several games
5. Logic to prevent players from taking claimed cells

AI could be implemented by utilizing the flip - flopping active player bit as an enable for the AI circuit. A simple AI wouldn't even require picking the most optimal cell but instead just choose a random space that is currently not occupied. If the random space is chosen by the AI then pick the next open cell.

X's and O's could be implemented to make the game more appealing. The current implementation uses red and blue squares to indicate each player as shown in the "Output" section above. This would be done using the code from the professor and by converting image files to txt files using matlab. the images would then be used as sprites in gameplay.

Games that end in a draw state could be detected by when all cells have been claimed but none of the eight win conditions have been met.

In order to keep track of points for each player across multiple games, a counter for each player would be needed. Then the ability to reset all game variables except for these counters would need to be implemented.

Cell claimed state is already stored in the eighteen bit vector that describes the game board state. This value would need to be piped back to the decoder block where it is then decided if the move that was just executed is valid or not.

References

[1] "VHDL Coding for FPGAs" Director: Daniel Llamocca - Associate Professor. Access on March 15th, 2022. VHDL Coding for FPGAs (oakland.edu)

[2] Diligent, "Nexys A7 Reference Manual," Nexys A7 Reference Manual [Reference.Digilentinc]. [Online]. Accessed on March 15th 2022. https://reference.digilentinc.com/reference/programma ble-logic/nexys-a7/reference-manual.

[3] "FPGA Based Tic-Tac-Toe Game with VGA Output" Matthew Button, Chris Lair, Kacper Wojtowicz Electrical and Computer Engineering Department School of Engineering and Computer Science Oakland University.

[Online] Accessed on March 15th 2022. Paper Title (use style: paper title) (oakland.edu)