Final Project: Tic-Tac-Toe

Joseph Popoviciu, Sami Elias, Nash Russak

Electrical Engineering Department, ECE 3300.02 Digital Circuit Design Using Verilog
California State Polytechnic University, California, United States of America

***Abstract*— This paper is a detailed explanation on how to create a tic-tac-toe simulator using Verilog and a Nexys a7 board. The project will utilize the following aspects of the board including switches, LEDs, RGBs, buttons, and the seven-segment displays. The output will be displayed onto a monitor screen using the VGA connections with the board and will be controlled using the buttons and with a keyboard that will be implemented using UART connections. The goal of this project is to use these aspects of the Nexys a7 board to create a project that will effectively use those parts. A tic-tac-toe simulation was chosen because of its ability to accurately display all of the functions of the board and because of its flexibility in being able to use multiple input devices and coding strategies. The result of our work has created a simulation that allows the user to effectively play a game of tic-tac-toe using the different features of the Nexys a7 board.**

***Keywords*— Electrical engineering, FPGA, Nexys a7, UART, Verilog**

I. Introduction

To display the full capabilities of the Nexys A7 board from Digilent our group decided to create a tic-tac-toe simulation that would utilize many of the features found on the board. The features of the board that will be used include the switches, buttons, LEDs, RGBs, and seven-segment display. Our simulation will work by outputting the display onto a monitor through the VGA port found on the board. Furthermore, the device will also be able to be controlled with a keyboard using UART connections. The final product will be a tic-tac-toe simulation that will allow the user to place the marker of "X" or "O" onto a specific spot until three in a row of either marker is achieved, the game will then be repeated once that requirement has been met. We will be designing our simulation using Vivado and will be writing the code with the language HDL or hardware description language. This paper discusses the details of our process of creating our project which include research, coding implementation, and troubleshooting.

II. Research

There are many different coding techniques useful for different situations. Some of the techniques that we researched include using if-else statements, case statements, and finite state machines. This same thought process went into designing the code that would be used for our UART connections to the keyboard. We decided to implement the same coding strategies for both topics.

A. Design Process for Tic-Tac-Toe code

The research that went into our project started with designing how we wanted our tic-tac-toe simulation to work. The main idea of our simulation was to have the buttons on the Nexys A7 board control the movement of the marker that will be displayed on the monitor. The buttons would also be used to place the marker on the designated spot. If enabled, the keyboard that will be connected would also be used as another option to input the markers onto the monitor. It would be programmed to allow the user to use the arrow keys to move the marker around and the enter key to place the marker. The switches would be used to enable certain aspects of our code such as enabling the keyboard or the speed of different clocks that would be in our code. The LEDs will be utilized by indicating whether our enable is currently working or not by having the light turn on to indicate it is active. The RGBs will be used to indicate what is happening during the game. A green light will indicate that the game is currently active, a blue light will indicate that player 2 or the "O" markers won the game, and a red light will indicate that player 1 or the "X" markers won the game. Finally, the seven-segment display will be used to show the remaining number of rounds and how many rounds player 1 and player 2 have won.

When researching techniques, the simplest one that we used to implement our code would be if-else statements. If-else statements first create a requirement for the if part of the statement and when that requirement is met the rest of the code under it would be executed. When the requirement is not met, then the else part of the statement would be executed instead and execute any code that is part of the else statement. For case statements, the process is similar where a requirement first has to be met until the rest of the code is executed. In a case statement, when the requirement is met, the code will then search for another requirement from a list of values within the case statement where if a specific variable is equal to that value, then that part of the code will be executed. The case statement in our project is paired with either nonblocking or blocking always blocks and a clock to produce the specific requirements for both the first and second part of the execution. The last technique that we implemented was finite state machines which use two separate registers to store data for the present state and next state of our code. The present state will not continue to the next state until a certain action has been performed. It differs from using a clock where a specific action needs to be performed by the user to proceed to the next state while a clock will continuously cycle and would eventually meet its requirement.

After researching more into these different techniques, we had to figure out what type of modules would be needed to allow the simulation to function. Within our project, the clock manager and counter modules would be the most significant because many of the other modules are reliant on them. The clock manager is used to create different clocks that have varying speeds depending on what is required for our other modules. The counters are used by the clock to count upwards or downwards and store the information to be used with various connected modules. Other modules we added later were debouncers which allow the users to have only one input be read for each button press instead of multiple times.

B. *UART connections*

When designing the UART connections the same techniques were used to allow the board to communicate with the monitor we were using. Mainly case statements and always block were used to allow the keyboard to be used as an input device. Always blocks allow for a specific action to continuously occur whenever a specific requirement is met.

III. Coding process

When implementing our code each of the coding techniques and module types that were previously researched were used. The choice of coding techniques and modules made the coding process easier since they are commonly used together and easy to implement.

A. *Creating the Tic-Tac-Toe Simulation*

Throughout our project, the main techniques that we used with our different modules were if-else statements and case statements mainly due to their ease of understanding and ease to see exactly how our code operates. The use of clocks, if-else statements, and case statements made it easy to combine the two to create variables that would hold the value of the clock and perform specific actions with each specific bit number. To create this we made a counter with each specific clock that would be connected to the correct module.

We also wanted to show the users how many rounds they've won so we implemented a counter using a finite state machine that would use a wire to connect to our seven-segment display. The seven-segment display will be used to show the number of games won by each player.

The design is simple with each successive 1 that is inputted into each state will allow the code to go to the next state. When a 0 is inputted, it is implied that a reset of the code has been activated which would bring the state back to state zero regardless of the current position. Each position in the state diagram will display a different number depending on what state it is currently in.

In Fig 2, as seen below, we have posted the design implementation that is created from Verilog. The design implementation creates a model of all the different blocks that were created in our code. As seen from our design many decoders and gates were used to implement our simulation.

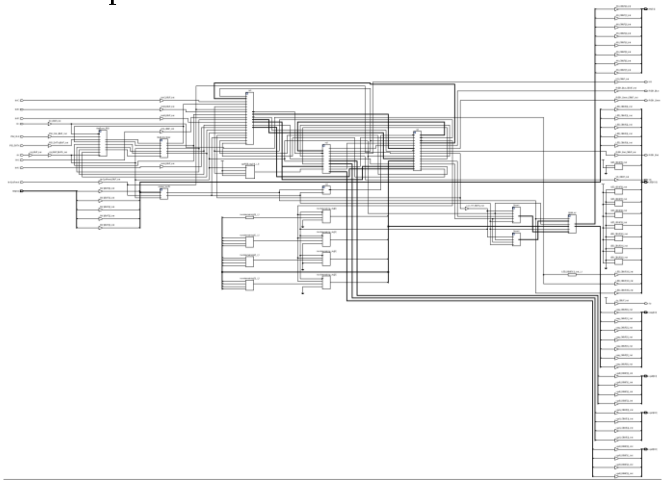

Fig. 2 Design implementation for our tic-tac-toe simulation created through Verilog.

B. *Implementing the UART Connections*

When creating our code for our UART connections we decided to use a keyboard that could be used as another input device to play our simulation. Since the Nexys A7 board reads the keyboard as a ps/2 input device we needed to convert the data received before it could be displayed on the monitor. This part was done by creating a receiver module that would interpret the data and then send it to the board so it could be correctly outputted. Within the module case statements and always blocks were used to constantly be interpreting the data and sending out the data for the correct case statement. Furthermore, we added a module to interpret the signals coming from the keyboard to emulate the same functions as our onboard buttons. To do so we used a nonblocking always block in combination with if-else statements.

IV. ASSEMBLY

The parts that were used in the assembly of our hardware include the Nexys A7 board, a computer with Verilog, a USB-connected keyboard, and a monitor capable of using VGA or HDMI connections. In our case, we used a VGA to HDMI adapter

A. *Connecting the Hardware*

All the parts used are directly plugged into the Nexys A7 board with the VGA connection being the only exception. To plug in the monitor, we used an HDMI to VGA converter to plug it into the board. The ports for the USB and the computer are readily available though. The board is first connected to a computer that has Verilog installed to allow the code to run. The keyboard and monitor ports will then be configured by the code running on the computer to be used correctly. The results of the code should then be displayed onto the monitor connected by the VGA wire.

V. TESTING

In order to troubleshoot our code, we first had to finish debugging the actual code to allow it to run and then we could see what was not being correctly displayed or working correctly. Each aspect of the board had to be tested to see what was not being outputted from our devices.

A. *Troubleshooting Simulation*

When troubleshooting our code, we first had to make sure all the syntaxes of our code were correct. The recurring problem that we faced was having ports wired incorrectly due to there being so many different modules. Since the code used for tic-tac-toe was so complex it was difficult which outputs signified the game being over, and further which players won that round. This proved to be challenging when creating the finite state machine counter to display each player's score.

When the simulation started to work the problem that gave us the most trouble were the RGB lights that indicated who won the round. When first run the lights appeared to not even be working, but after further testing of the simulation we found out that they did work but only if a victory was achieved on the top or middle rows of the game. The top row caused the blue RGB to go off while the middle row caused the red RGB to go off. Upon further testing of our code, we saw that instead of coding it so the markers would determine the win color, the position on the field was what determined the RGB color output.

B. Troubleshooting FPGA and UART Connections

When figuring out the UART connections to the keyboard and monitor we first had to find out what type of connections the board had. We were unaware that the board only read in ps/2 input and due to that many of our previous codes did not work until we looked it up. Unfortunately, when assigning our input in the button control module we were using ASCII to hex instead of PS/2 to hex.

VI. CONCLUSION

To finish our project to create a tic-tac-toe simulation using the Nexys A7 board, a considerable amount of research and troubleshooting had to be done. By researching commonly used techniques used in Verilog like using if-else statements, case statements, and finite state machines we strengthened our understanding of Verilog and implement them into our code. By researching these topics, we were able to create counters, clocks, and debouncers that would help run the rest of the code. From completing our project, we have gained a greater understanding of how to use Verilog and how to create connections using UART. By having to utilizing many aspects of the Nexys A7 board it allowed us to see that there are many ways to use those parts in different ways. Other simulation projects that we have thought of as a result include creating a Simon says game that could use the LED's more and a different input device such as a mouse to play.

REFERENCES

[1] Arpadi, Sergiu. "Nexys Video PS/2 Keyboard Demo." *Nexys Video PS/2 Keyboard Demo-Digilent Reference*, https://digilent.com/reference/learn/programmable-logic/tutorials/nexys-video-axi-ps2-keyboard/start.

[2] Artvvb. "TicTacToe/SRC/HDL at Master · Artvvb/Tictactoe." *GitHub*, https://github.com/artvvb/TicTacToe/tree/master/src/hdl.

[3] Aseddin. "Ece_3300/fsm_counter.V At Main · ASEDDIN/ece_3300." *GitHub*, 26 Mar. 2021, https://github.com/aseddin/ece_3300/blob/main/8%20FSMs/Source/fsm_counter.v.

[4] Brown, Arthur. "Nexys A7 Reference Manual." *Nexys A7 Reference Manual - Digilent Reference*, https://digilent.com/reference/programmable-logic/nexys-a7/reference-manual.

[5] Digilent. "Digilent/Nexys-A7-100T-Keyboard." *GitHub*, https://github.com/Digilent/Nexys-A7-100T-Keyboard.

[6] "Tic Tac Toe Game in Verilog and LogiSim." *FPGA Projects, Verilog Projects, VHDL Projects - FPGA4student.Com*, https://www.fpga4student.com/2017/06/tic-tac-toe-game-in-verilog-and-logisim.html.

[7] "UART, Serial Port, RS-232 Interface." *UART in VHDL and Verilog for an FPGA*, https://www.nandland.com/vhdl/modules/module-uart-serial-port-rs232.html.