Connect Four Game

List of Authors (Fabrisio Ballo, David Herweyer, Ryan Thomas)
Electrical and Computer Engineering Department
School of Engineering and Computer Science
Oakland University, Rochester, MI
e-mails: fballo@oakland.edu, dherweyer@oakland.edu, rjthomas3@oakland.edu

***Abstract*—In this project a Connect Four game is created in VHDL. The purpose of this project is to build a source of entertainment, while also displaying what has been learned throughout this semester. The main topic learned from this project is how to interface the Nexys-4 DDR Board with VGA display.**

I. Introduction

The main motivating factor of this project is to create a fun game everyone can enjoy, while displaying skills learned from this course. The Connect Four game is simple, a seven by six game board is displayed using VGA onto a screen, and 2 players are able to verse each other. This is a turn-based game, and the first person to get at least four in a row wins the game! To play the game the user will use the buttons on an NES controller to choose where they want the token to fall, the token will be displayed over top of the game board to show where the player will drop. Once the player decides to drop the token, a button is pressed, and the players turn ends. A reset button is used to reset the game, once a player either wins the game, ties the game, or at any time during the game. Once a player wins the game a message that says “P1 WINS” or “P2 WINS” will be displayed on the screen next to the game board, depending on who won. After displaying the winner on the screen, and once the game board is reset, the seven-segment display will update the total of each players wins and display it passively during the game. If the game board gets full at any point, then the game will tie and neither party will receive a win for that game.

A few topics integrated into the game were learned from this course. Some of the topics include a finite state machine, creating datapaths, VHDL code, and some VGA interface. Although VGA was taught a bit in class, a majority of it was learned outside of class to get the project to work as intended. Since the project is a game, the only application of it would be to have fun and enjoy the time spent playing the game!

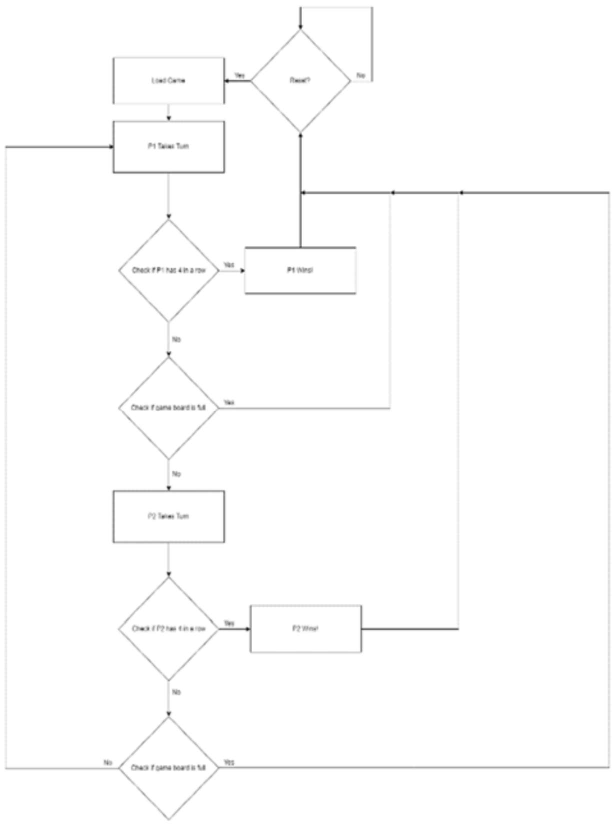
Reset?
Yes
No
Load Game
P1 Takes Turn
Check if P1 has 4 in a row
Yes
P1 Wins!
No
Check if game board is full
Yes
No
P2 Takes Turn
Check if P2 has 4 in a row
Yes
P2 Wins!
No
No
Check if game board is full
Yes

Figure 1. Connect Four Flowchart

II. Methodology

A. *Game Board Creation*

The VGA code provided on the reference website was used as a starting point for this project. The reference code uses the in_RAMgen block in Figure 2 to read the contents of a text file and initialize the data into the BRAM of the Nexys 4 board.

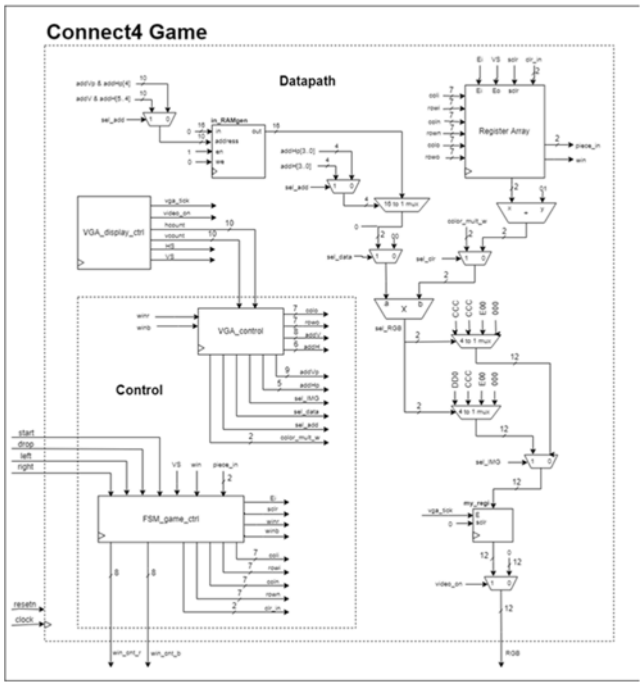

Figure 2. Game Datapath

There are two significant limitations with the memory storage. First it only accepts a square image file that has dimensions that are a power of 2 and secondly it takes a very long time to synthesize if the size of the image exceeds 32x32. For this reason, it was important to fit the image file into as few of addresses as possible. This was accomplished by assigning 1 bit per pixel indicating whether a pixel changes color (value of 1) or retains its color (value of 0). By setting up the game board this way, a 64x64 pixel board position and 7 32x32 pixel letters were fit into 1024 addresses which contain 16 bits or 16 pixels per address. The input text file was generated using Excel so that the full game area could be viewed and edited quickly until it was finalized. Figures 3 and 4 show the creation of parts of the game board in Excel.

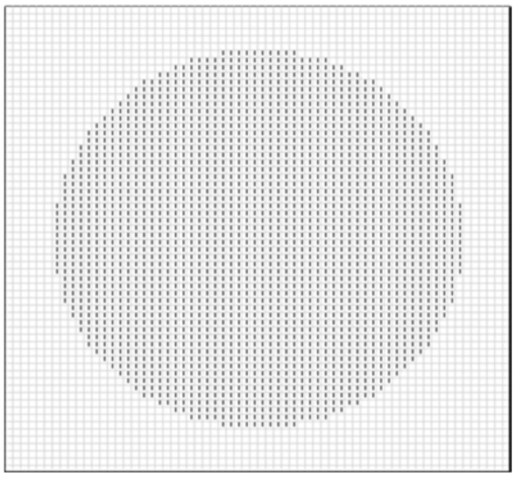

Figure 3. Excel Gameboard Bitmap

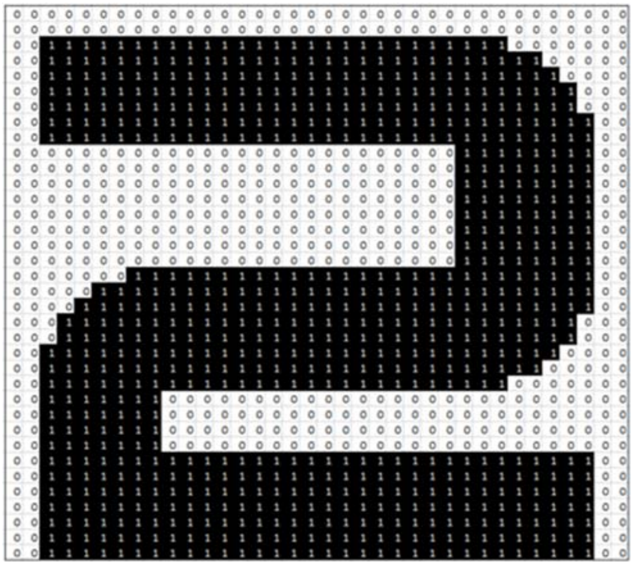

Figure 4. Excel Number 2 Bitmap

B. *VGA Display of Game Board*

After storing the game board in BRAM, the next challenge was piecing the image together on the display. In order to do this, the VGA control block in Figure 2 was created and used to map the memory address, pixel number, color LUT, and input data source to the correct regions of the screen using the hcount and vcount signals. These signals give the horizontal position and vertical position on the display. Each memory address is 16 bits wide and contain 16 pixels so the address location only increments after hcount reaches 15. When vcount increments, the address increases by either 64 for the piece position or 32 for each letter. The 4 least significant bits of hcount were used as inputs to a mux to cycle through the 16 pixels in each address.

Changing the color of each position in the game board was done by multiplying the input data by a value held in a 7x7 register array of 2 bit registers. Each register holds a value that determines the multiplier that is applied to the input data. The multiplier is a value from 1 to 3 and changes only the pixels that were assigned a 1 in the memory. The result is a 2 bit value from 00 to 11 that is used to select from one of two LUTs containing 12 bit colors. This multiplier value is only updated at the end of a frame when VS is low to avoid changing only part of a piece. Updating the output registers in this way required another 7x7 register array to store the inputs values that are written while VS is high until the end of a frame when the output registers are ready to be updated.

C. *Game Logic*

Once the interface for displaying the game board and changing the color of the pieces was completed, the next task was adding the game logic. The FSM block in Figure 2 is the main component in the game logic and was implemented as shown in Figure 5.

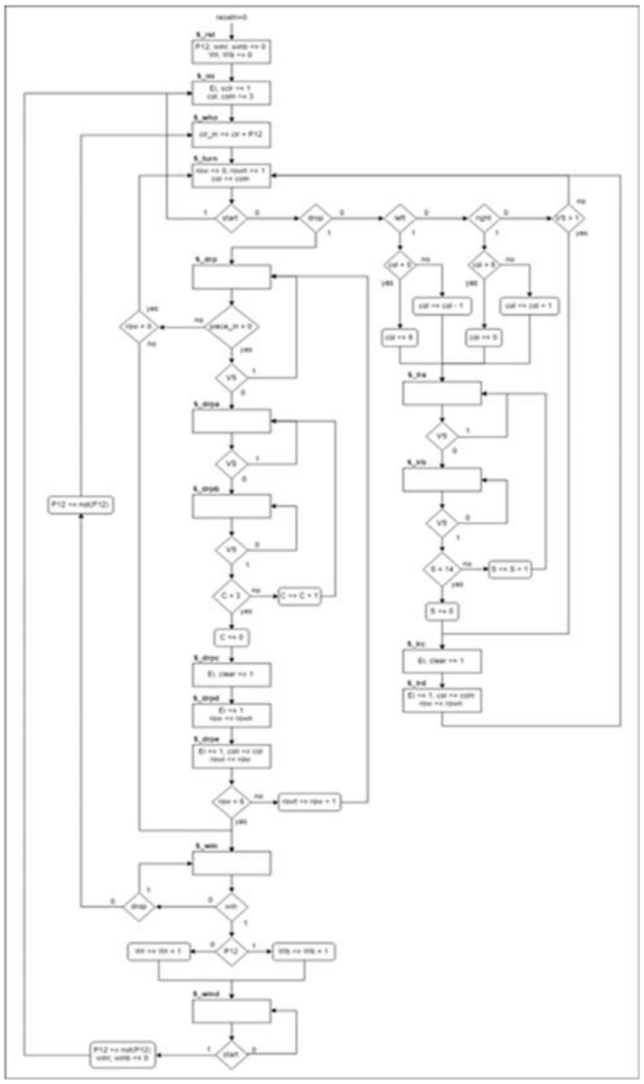

Figure 5. Game FSM

The first function of the FSM is initializing the game board to empty on startup and on the event that the start button is pressed. This was achieved by adding a synchronous clear to the register arrays to clear both the input and output registers. The next function is determining which player's turn it is. The P12 signal was added to accomplish this task. This signal is inverted at the end of a turn so that the other player can take their turn. The P12 signal is also used to determine which multiplier value will be written to the register array.

The third function is constraining a turn to only intentional valid moves. To accomplish this goal the buttons were prioritized so that there would not be multiple button presses registered at once. This means that if two buttons are held at the same time the higher priority button will be the only one that the FSM acts on. There was also a 15 frame delay added between updates when the left or right buttons are held down so that the user can see the movement of the piece above the board. If the drop button is pressed, the FSM checks for a piece below the current position before moving it down. If the next piece is not empty then the piece has reached its final position. If piece never entered the game board then the FSM will loop back to the current players turn so that they can select a valid move. A 4 frame delay is built in to the drop loop to make a visible dropping animation that is more realistic.

The final function of the FSM is checking for a winner and then updating the game display and scoreboard if there is. An array of 4 input AND gates was added to the register array block in Figure 2 to check for all possible win conditions for both players. The FSM is needed to determine whether a piece is in its final position when a win condition is set. If it is then the FSM will set either the winr signal for red or winb signal for black to high which will light up either P1 WINS or P2 WINS on the sides of the game board and increment the appropriate scoreboard.

D. *NES Controller*

The primary goal of interfacing the NES controller to the connect four board, is to provide a more user-friendly interface to the users. The NES controller primarily consists of a parallel in serial out shift register connected to the 8 buttons on the face of the controller.

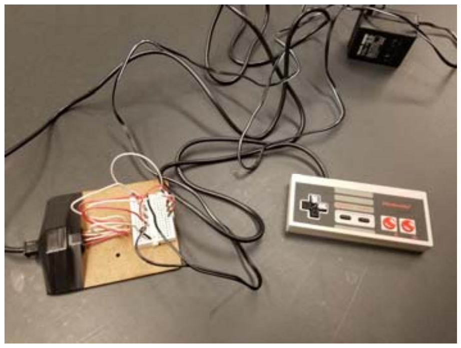

Figure 6. NES Controller

In order to interface with this shift register, 3 communication lines are required. The required timing must follow the timing diagram seen below.

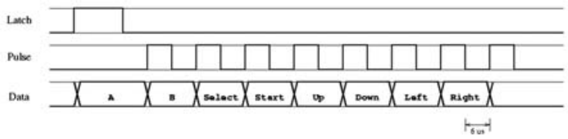

Figure 7. Timing Diagram [3]

In order to interface this controller to the game board a control unit and datapath had to be created. First a 12 us pulse has to be sent out in order to latch the button data on the controller. Then the bits are shifted out 1 at a time at the end of each pulse. The data line is read in at the falling edge of each pulse seen in the timing diagram above.

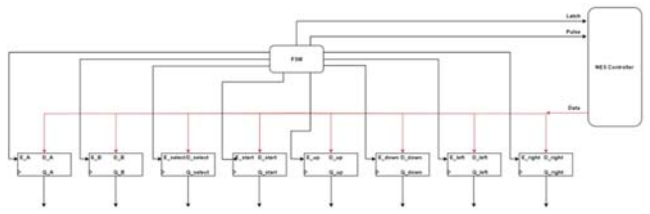
Figure 8. NES Controller Datapath

In order to generate the required timings for the controller, a 20 state FSM was created that used three my_genpulse counters to time a 60 Hz refresh rate, a 12 us latch, and each 6 us pulse. The FSM was implemented as seen below.

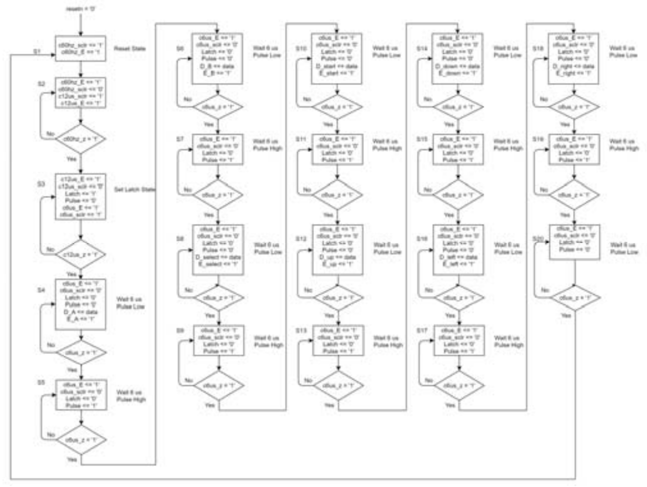
Figure 9. NES Controller FSM

The results of the state machine that was implemented can be seen in the test bench below where there is an initial 12 us pulse, followed by alternating high as low 6 us pulses. On each of the low pulses, the register corresponding to the appropriate button is enabled in order to read in the value.

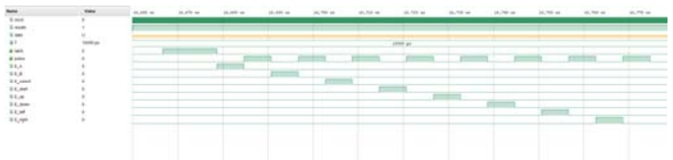
Figure 10. NES Controller Testbench

E. Seven Segment Display

The seven segment display was added to show both player's win totals in decimal values from 0 to 99. To achieve this, P1 and P2 were hard wired into 4 of the 8 seven segment displays and the other 4 were used to show the score. The main game code outputs an 8 bit standard logic vector for each score which is then converted to a BCD value that is used to determine which segments of the display to light up for the scores.

III. Experimental Setup

Most of the main game code was checked for the correct operation by viewing the output on a VGA display. The initial game board for instance was verified by outputting it on VGA as shown in Figure 11.

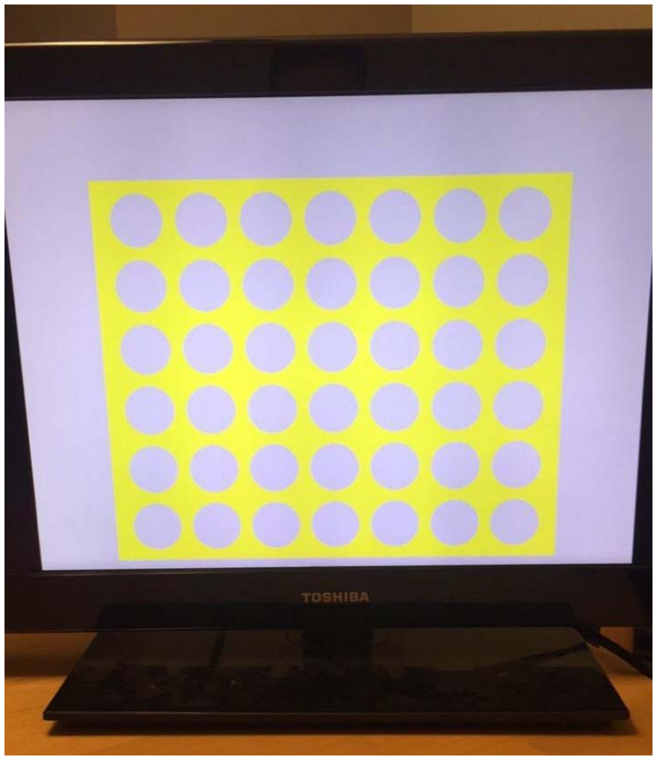
Figure 11. VGA Connect Four Board

It is not very practical to run the VGA interface in a testbench due to the relatively long amount of time that it takes for a full frame to be written. For this reason almost the entire game code was tested iteratively on a VGA display whenever a new feature was added in. Some signals were temporarily hard coded to a value or mapped to a switch or buttons on the Nexys 4 board so that a function could be checked before the control circuitry was developed. The only part that was verified using a testbench for the main game code was the FSM. A special project containing only the FSM and register array that it was in charge of updating was created so that it could be isolated from the VGA timing and tested for the correct functionality. The testbench created the VS signal with a much smaller delay so that the simulation would not take such a long time to run. During testing of the FSM a button press of start, drop, left, and right were all simulated and the register values were monitored to see if the multiplier value moved to the correct location. It was expected that when the right or left buttons were pressed, the register column would increase or decrease by one and the corresponding register would show the correct 2 bit number. The expected result of pressing the drop button in the testbench was that the value would move down row by row until it reached the bottom of the game board. Last the start button was expected to clear the contents of the register array.

The controller FSM was also tested using a testbench to check the timing. The expectation was that the intended enable signal would go high relative to the 3 clock signals that were being generated.

The 7 segment display was tested by setting the inputs to switches and viewing the results on the seven segment display. Since a good portion of this code was provided it was not necessary to create a testbench as one was already included.

IV. RESULTS

The results of this project were overall positive. Many problems were encountered early on but in the end the game functioned exactly as intended. In Figure 12 below a player 2 (black) diagonal win is shown. Figure 13 shows a tie game which doesn't allow for any more moves other than restarting the game. This is because the FSM will not allow a piece to be dropped in a column that is already full.

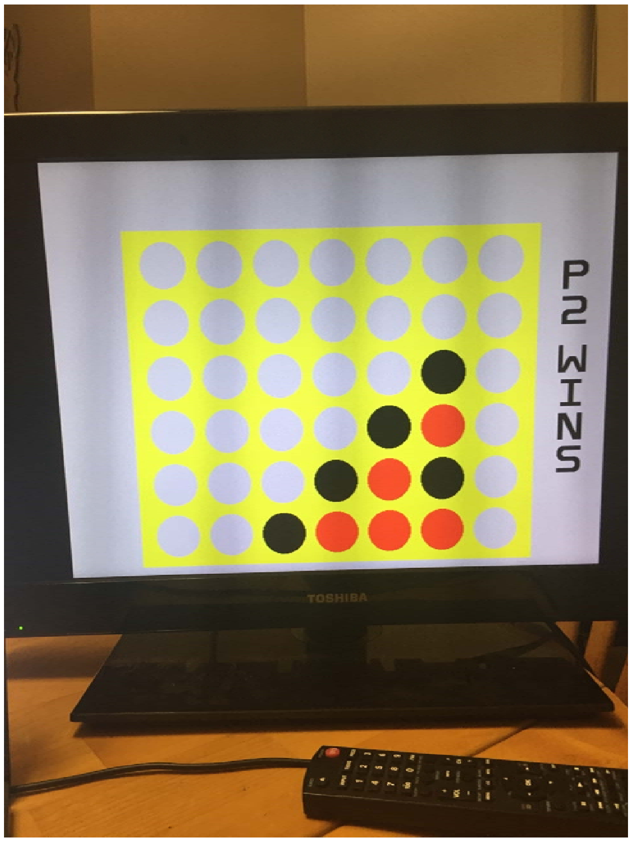

Figure 12. Player 2 Diagonal Win

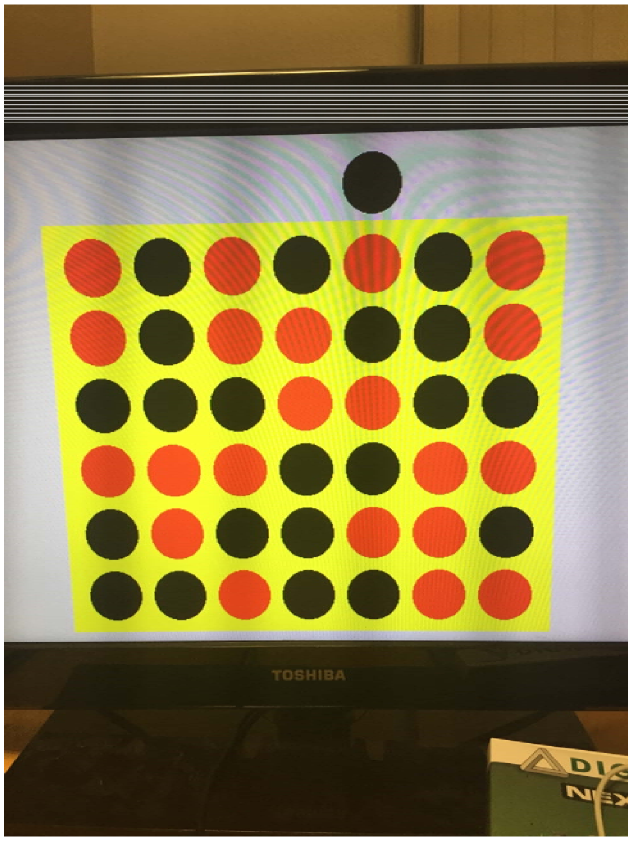

Figure 13. Tie Game Condition

The score also works correctly and is only reset if the reset button is pressed. If a new game is started the score persists.

The most difficult issues to resolve were when a game piece would inexplicably overwrite another piece. This issue did not occur very often which added to the confusion. The root cause of the overwriting issue was determined to be the drop button press timing relative to VS. If the button was pressed again before VS went low, then the output register was not able to update and the FSM would not know there was a piece there. This was corrected in the FSM by ensuring that VS is low when the output register is read from in the FSM.

V. CONCLUSION

Overall when working on this project a lot was learned such as the importance of the sensitivity list in a process. Another takeaway is that it is very important to verify each part of a large VHDL project before moving on or it makes debugging an issue extremely difficult. Also testbenches are almost always worth the extra work when trying to debug a complex state machine. Although the end goal was achieved and everything worked as intended, there are a few improvements that could be added to the project. There are three main improvements that come to mind. The first is to add a second NES controller to eliminate the need to pass the controller back and forth and make the game feel a little more fun. Next, a sound bit file of a physical game piece dropping would be added to the game each time a player went their turn to better simulate the feel of a real Connect Four game. Lastly, with a lot more time, a whole game menu would be added to allow a user to choose between a selection of games, while still maintaining the player scores. This would make the project feel more like an arcade style game.

VI. REFERENCES

[1] R. Haskell and D. Hanna, VHDL by example. Auburn Hills: LBE Books, 2016.

[2] "Lesson 105 - Example 71: VGA Stripes", YouTube, 2012. [Online]. Available: https://www.youtube.com/watch?v=7j7brGz7u6M. [Accessed: 12- Mar- 2019].

[3] https://tresi.github.io/nes/ (Needs fixed formatting)