ECE 4710 Final Project
VHDL Game: 2048

Authors: Kyle Alspach, Robert Brosig, Grant Parker
Electrical and Computer Engineering Department
School of Engineering and Computer Science
Oakland University, Rochester, MI
E-mails: kalspach@oakland.edu, rmbrosig@oakland.edu, gparker@oakland.edu

***Abstract*- The purpose of this project is to implement the student's current understanding of VHDL and apply it to designing the popular mobile game "2048". To beat the game, equal tiles are added together to reach the sum of 2048. The game is controlled by four controls, up, left, down and right. This is aliased as "wasd" for simplicity of input. Through the use of hardware, several components such as FSM's, counters, comparators, and VGA display controls, the desired implementation will be achieved.**

I. INTRODUCTION

In today's world, video games come in different shapes and sizes. Though most games are coded with programming languages, the same outcome can be achieved through hardware description languages such as VHDL. The topic of this project is the mobile game "2048". This game is played by sliding tiles that are randomly generated and adding them up until you get to the number 2048.

The game starts off with a 4 x 4 matrix and two randomly placed and generated number blocks normally 2's. The game is then operated through user inputs by swiping a finger up, down, left and right across the mobile device's screen. Once equal valued numbers are swiped into each other, they create a new block which is the sum of the previous two.

As mentioned, the end goal is to slide enough of these blocks together until the end value of 2048 is achieved. While the end goal is simple, the user must manage the matrix space and prevent too many number blocks crowding it. If the matrix becomes full and prohibits newly generated blocks from appearing, the game is over.

II. METHODOLOGY

The first step is to start developing a circuit diagram that would best achieve the desired implementation of the game. The biggest obstacle that was faced was the main control circuit, more specifically the FSM. With each movement input and each number tile moving in tandem, there are a lot of conditions that need to be checked in order to move from one state to the next. Each state needs to check to see if the number 2048 exists, what empty spots are there to place new blocks and to check to see if the board is too full to add any more blocks. There are many more scenarios that need to be checked with every input. The input itself shall be handled by the circuit shown on the following page. The UART shall load a register with each character input, and then compare it to each ASCII character value. This is then encoded, ignoring the 0000 case, as that would imply a character input that is not w, a, s, or d. This may cause trouble with this, and if so the design will be modified to include the 0000 case. The output is a combination of a signal from the UART, and the encoded directions, letting the next circuit perform functions when a key is pressed, and knowing which key.

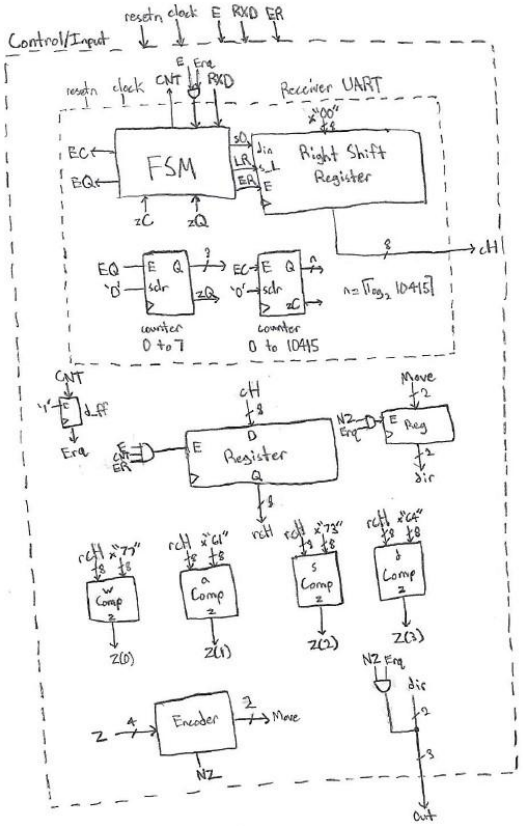

Figure 1 - UART Control Circuit

The next big challenge was to create the VGA display for the project. The first step was taking the VGA display control circuit from Dr. Llamocca's FPGA tutorials and modifying it [1]. This proved to be a challenge given the scope of this project. In order to develop what was needed, the game board was thought of as a graph. Rather than using a RAM generator to store images, it was more efficient to use the graph method to draw each image.

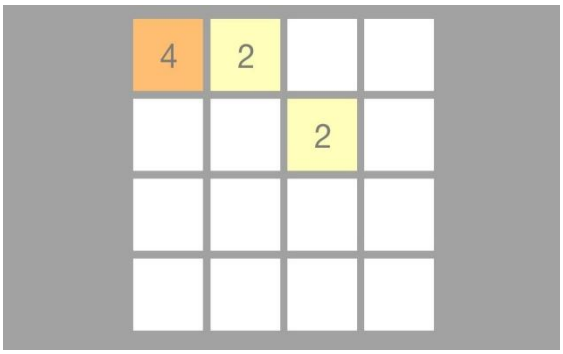

Figure 2 - Playing board

In Figure 2, this is what the rough playing board will look like. Each box will be passed a value from the FSM and will display the corresponding number. Each number is drawn separately as opposed to storing it as a series of addresses to save time in simulation and synthesis.

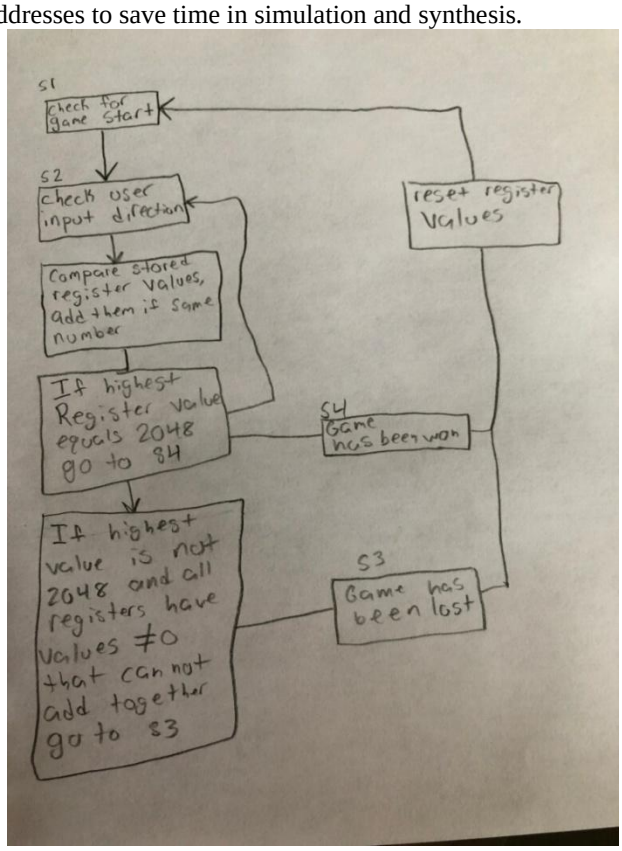

Figure 3 - Rough draft of FSM

After further development, the final FSM included 7 separate states: start, direction, move, merge, check highest value, win, lose, and random. The first state in the FSM was the start state. This state set all the array values to zero and then waited for a start signal to be sent. Once this signal was sent two random numbers were placed into two array components to be displayed on the VGA display. In order to simplify the randomly placed values the values could only be placed in the outer columns of the array. After random numbers are placed it goes to the direction state.

The direction state waits for input signal telling it which direction was selected, then it sets a btn_direction signal equal to that value to be used in later states. Once a proper direction signal is sent the FSM moves onto the move state. While in the move state the FSM looks at the btn_direction signal and depending on which direction is selected it shifts all values in that direction. During this phase it looks at every possible combination that allows the numbers to move into blank spaces. Once the numbers have been moved into the blank spaces it moves onto the merge state.

In the merge state depending on the btn_direction the FSM looks for values that are equal and adds them together. Once added together it places the new number in the array value that was already being used and brings in a zero from the opposite side of the movement. After all possible cases have been checked for all the rows or columns depending on the btn_direction the FSM moves onto the check highest value state.

The check highest value does exactly what it sounds like. Once in this state it first checks all the values to see if one of the array values is equal to 2048 if it is, it generates a winner signal and goes to the win state. If not it checks to see if any of the array values are equal to one another in a way that can be moved and merged together. If there is a pair that is equal the game goes to the random state, if not and none of the array values equal each other then the game is lost and the loser signal is generated. Once the loser signal is generated the FSM goes to the lose state.

In the random state a new number either 2 or 4 is randomly placed on the board. After being placed the FSM goes back to the direction state where the cycle begins again. During the win state the display lets you know you won then it goes back to the start state and waits for a start signal. Similarly, the lose state does the same thing it just displays that the player has lost. Therefore the final FSM design can be seen below.

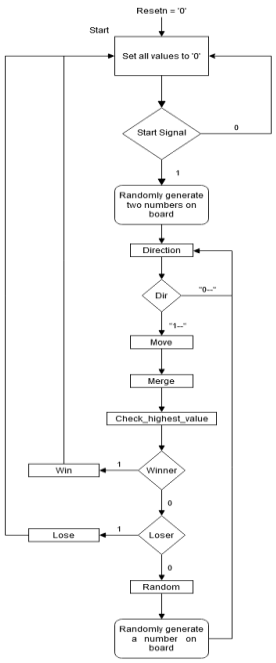

Figure 4 - Final draft of FSM

III. EXPERIMENTAL SETUP

In order to test the circuit design, the students must first download PuTTY onto a computer. While the FPGA does not need to be programmed by the computer if it was already programmed by a separate one, PuTTY is used to communicate serially to the board, acting as the control system. Any feedback will be sent to the VGA screen, rather than serially back to the computer.

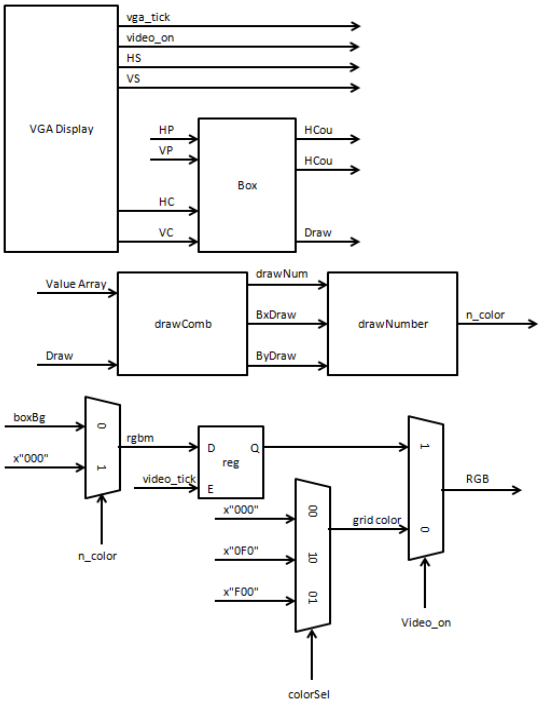

Figure 5 - VGA output circuit

After creating the FSM, it needed to go through testing before the VGA display was implemented into it. For this a simulation was done using the FSM, many bugs were found in the beginning stages of the testing. The first bug that occurred was when going into the direction state all the array values would go to zero. After looking through the output portion of the FSM it was realized that there were no outputs during the direction state. In order to fix this all of the array values had to be set equal to one another in order to pass the values from the previous state into the direction state.

The next problem that occurred was in the first FSM the merge state was before the move state. This was found not to work because it would look to merge numbers together before it got rid of the blank spaces between them. Therefore it would not merge values until the next direction was chosen causing the initial merge that was wanted not to happen or happened a move later since you would have to choose the same direction again in order for it to merge. This was fixed by moving the move state to occur before the merge state. This allowed for the numbers to move together then merge together in the same cycle.

The biggest problem faced in the FSM was the implementation of the random number generator. The random number generator was used to first randomly select a row and a number for the numbers placed into the array at the start of the game. It was also used during the random state for introducing new numbers into the board and the end of every move. The original thought was for the new number to be able to be randomly placed in any open array value but this proved very difficult. Instead depending on which btn_direction was selected the random number would be placed in an open array value on the opposite side of the direction. Therefore, if the btn_direction was up the random number would be placed in one of the open spots on the bottom row of the array. After simplifying the possible cases for the new randomly placed numbers it was tested. In testing it was found that there was a bug when placing the random number. If the random place number was different before and after the clock cycle that changed the state to random it would place two random numbers in the row or column. If the rand place number was the same though it would only place one number. This bug was never fixed through countless modifications to the random state. A concept that was thought of but never had the chance to be implemented to fix this was to maybe how the states change on the falling edge of the clock cycle. If this were to happen the random place number should have been the same before and after because it was a separate component that generated a new number at every rising clock tick. Once accepting the fact that two random numbers may be generated the simulation of the FSM looked to be properly functioning. The numbers moved and merged together correctly and when all of the array values filled up with numbers that did not match each other it stated the game was lost.

After the FSM was developed and tested, it was combined with the VGA display circuit as seen in Figure 4. This circuit was built using Dr. Llamocca's VGA display circuit [1]. The circuit works by taking the output horizontal and vertical counts to determine when to draw what on the screen. The first step was to create the 4x4 grid to display the boxes the numbers will be stored in. Since the output display of the base VGA display circuit was 640 x 480, each box was drawn using 16 coordinates. The coordinates were then compared with horizontal and vertical counts and a draw signal was outputted. Each box was then combined into one signal and compared with a case statement to determine when it needed to be drawn and with what array value it was.

The next step after this was to draw the numbers for the array value in each box. This proved to be the toughest challenge of the entire VGA display. To aid in the creation of this, Microsoft Excel was used to map out each number. Since each box was 150 x 110 pixels, the grid used in excel was scaled down. Once the layout was figured out for each number, it was then written out with a case statement. The process read the value from each box and the horizontal and vertical counts. Using a series of if statements, each horizontal and vertical counts were checked to figure out when to output a signal to draw each number.

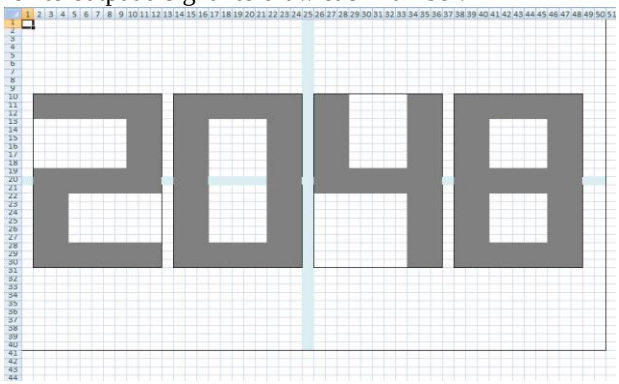

Figure 6 - Excel Bitmap representation

Once every circuit was created, the last step was to use a series of multiplexors to hold each 12-bit color needed to color whatever element needed to be colored. Each number had its own background color and was written out along with the corresponding number into a register. The register was controlled by the video tick outputted from the original VGA display circuit. This was sent to the final multiplexor that sent out the final RGB signal to the VGA output. Connected to the last multiplexor was one more that held the grid colors. By default, the grid was black. Once the player achieves 2048, the grid turns green. If the player fills up the grid and can no longer make any more movements, the grid turns red to signify that the game was lost.

IV. RESULTS

While the FSM and receiver worked together, the VGA display circuit did not display the correct table. During testing, each component had various issues that had to be remedied, including the receiver only accepting the first input after a reset, the FSM performing the move and random number generation actions multiple times, and the VGA display having various difficulties in conversion. Given the lack of time, a focus was placed upon the FSM, as it is important that the circuit work at all.

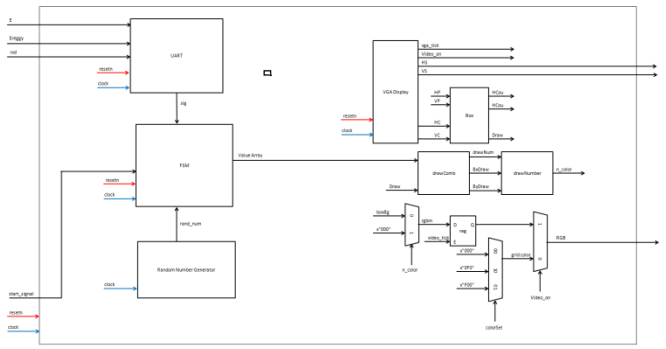

Figure 7 - Top level circuit

CONCLUSIONS

The students gained more knowledge of how to properly communicate with external hardware using the NEXYS A-7 board, create a complex logic-driven FSM, and how to create a display using a VGA display. While the result of the circuit was not as intended, the process of making each component was still valuable, teaching the students both how to make each part, and the importance of keeping enough time budgeted for debugging. While each component by themselves mostly worked, corners had to be cut towards the deadline, especially when making sure each component worked together. This is a suboptimal solution, and one that the students have learned from.

REFERENCES

[1] Llamocca, Daniel. VHDL Coding for FPGAs,

http://www.secs.oakland.edu/~llamocca/VHDLforFPGAs.html