LED Matching Game

Julian Hakim, Matthew Binkowski, Jacob Hamameh, Stefan Maiorano
julianhakim@oakland.edu, mjbinkowski@oakland.edu, jacobhamameh@oakland.edu, smaiorano@oakland.edu
Electrical and Computer Engineering Department
School of Engineering and Computer Science
Oakland University, Rochester, MI

***Abstract*—The LED Matching Game was made by describing digital circuits on Vivado using VHDL. An initial circuit was designed, then it was described using VHDL. After debugging, there were some design changes to ensure that the game works as intended. One of several design changes involved using the 7-segment displays on the FPGA board instead of wiring an LED array. This simplified the design but presented a new challenge: using a serializer to display different segments simultaneously at different speeds. After research and testing, the serializer and 7-segment displays were successful for the design. The project was successful and worked exactly as intended, but when it comes to complexity in the game there are a lot more features that can be implemented.**

I. Introduction

This report will cover every step in how we got our project from an idea to a working design. Beginning with our initial ideas and designs. The issues we faced and concepts that had to be scrapped or changed due to poor functionality. How we adapted to using different components or modified components to fit our needs. The results obtained from testing and simulation. Finally, a discussion on our results and an overall conclusion.

Our main motivation in this project was to create something that was not extremely simple or complex but still represented the topics learned in this class as completely as possible. We believe that we did just that, using many major components and even designing completely new ones using strategies discussed in class.

As for our components, we utilized ring counters, comparators, adders, modulo counters, finite state machines, seven segment decoders, serializers, multiplexors, decoders, and seven segment displays. However, not all these components were used as they were. Of course, the main finite state machine was designed from scratch just for our project. The modulo counters needed new counts to be calculated to fit our timing needs. An enable was created for the adder to properly calculate the score. Two different types of seven segment decoders were created to properly display the score and game. Finally, the serializer was expanded to have the capacity to output data to eight seven segment displays by increasing the size of the multiplexor, decoder, and finite state machine. The main concept we learned here was the serializer. We knew of it but had never seen it functioning. Additionally, our project had a strange functionality compared to simply displaying numbers for a timer or a similar device.

This project was mainly designed to be a game. Of course, it is a small game that is limited by the size of the board that was used. Still, the first thought was of entertainment and having fun through a simple game. Maybe something that could be in an arcade, movie theater, or restaurant that you could win prizes from.

II. Methodology

A. Shifting Bits

The initial plan was to use shift registers to shift bits and light up the rows of LEDs. This was the most obvious example of shifting data across rows, it also had the capability of pausing and adding more complex bit configurations if more difficulty was needed. However, we had trouble getting the bits to wrap around in the same patterns. After much trial and error, we found out that ring counters were a perfect solution that did exactly what we needed and were much easier to use. They weren't capable of complex formations of bits, but they were able to be controlled by other counters to speed up the counting, still allowing additional difficulty. In fact, this speeding up is something that was implemented later.

B. Scoring Points

For the game to be able to display the score, we implemented a comparator between each two ring counters, therefore giving us two comparators for the three ring counters that control the sequencing of the rows. The comparators determine when the paused LEDs are lined up. Then, we tied the outputs of the comparators to an adder that will determine the final score. Although we found that we needed to add an enable condition to the adder, otherwise the score would increase every time the ring counter outputs matched the previous output even when the rows were not paused. After adding the enable to the adder we were able to control when the score was summed up without it increasing throughout the game. We controlled this enable with our finite state machine.

C. Serializer/Decoders

The serializer and decoders were very closely related in this project. Typically, a serializer only includes a single decoder, but we needed two types of decoders. The score decoder was a normal decoder used for most seven segment

displays. The input for this decoder was simply mapped to the output of the adder. However, a special decoder was needed to display the outputs of the ring counters.

We needed to get the proper signals to input into these decoders. The bits of the ring counters were combined in a specific way in the VHDL code. For each counter in order, the least significant to most significant bits were concatenated into signals A-G. These were 3-bit signals that would be mapped to the three rows present in the seven-segment display. Our specialized decoders would take these signals and light up the proper row on the display depending on the 1's and 0's present in the signals.

As previously stated, a serializer usually uses one decoder and a multiplexor to select which data goes into it. As we had two types of decoders and needed eight displays to be used, we merely fed the decoded inputs directly into the multiplexor.

D. Finite State Machine

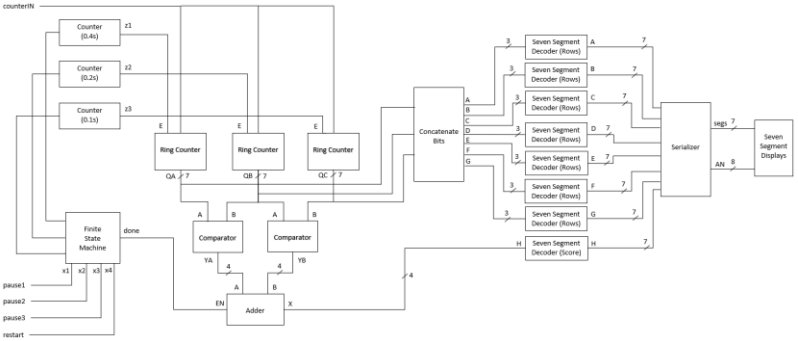
One of the most important parts of the game is pausing the rows of LEDs. This is the main element of gameplay and is needed to end the game. For this very important function, we needed to design a finite state machine. The FSM always had four states where three were dedicated to pausing the rows and one was for displaying the paused rows and enabling the adder to calculate the score at the end of the game.

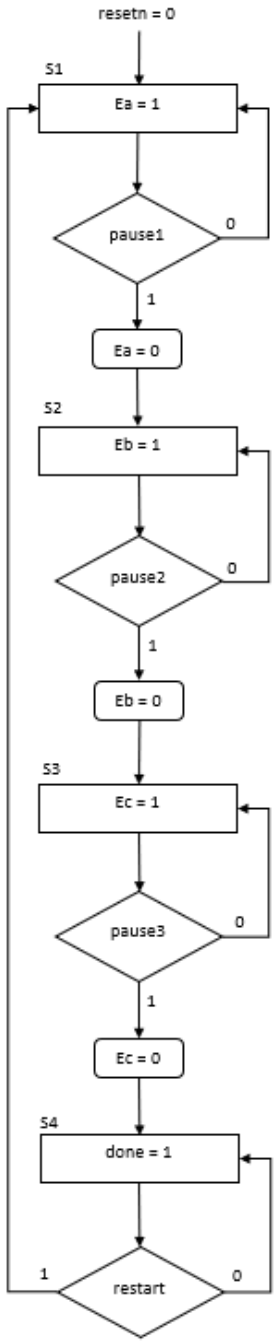
First, we tried using one button since a single input would be simpler for coding and implementation. However, we ran into major issues with the bouncing of the button causing rapid switching between states. Instead of needlessly complicating our code, we decided to change our approach.

Next, we used a single switch. This solved the debouncing issue at the cost of less ease of pausing the rows as the switch would have to be constantly moved back and forth. Additionally, it could be confusing to keep track of a switch for the player since it has two positions. Therefore, this still was not the correct answer.

Finally, we decided on using four buttons. Three for pausing the counters and one for restarting the game. This seems inefficient but we had space on the board for this change. It also allows for the user to keep track of which row they are pausing and where they are in the game.

E. Block Diagram and FSM





III. Experimental Setup

Verifying the functionality of the project involved several steps in Vivado – synthesis, simulation, implementation, and generating bitstream – and physically testing the board to ensure it works as intended. Specifically, VHDL was written in Vivado 2021.1 to describe the circuits from the block diagram and the Nexys Artix-7 50T was the FPGA board used to program the circuit. First, we wrote a test bench to check the following:

1. All values are defined.
2. The ring counter works properly by keeping only 1 bit on at a time and wrapping around once it reaches the least significant bit.
3. The pausing function works.

After debugging, we ensured that all these conditions were satisfied and programmed the board; however, there were several issues. The decoder did not display the segments as we wanted, but this was simply fixed by reviewing and carefully correcting the decoder values. There was bouncing going on with the button, but instead of adding a specific denouncer circuit, we added a pause button for each row and designed the finite state machine so it only cares about the first press for a period.

Lastly, an issue and flaw in our test simulations was that the values were shifting faster than the eye can see. After making all the counter speeds faster to be viewable on the simulation, it turns out that the counter was not used properly. With debugging via simulation and physical tests, this issue was fixed.

IV. Results

After weeks of designing, testing, and debugging we were able to achieve what we wanted while collaborating as a team. The final result was a functioning LED matching game. The game utilizes all 3 rows of each seven segment display on our FPGA board, although we would have liked to include more rows, that is all that is available in each display.

While testing, we noticed our serilaizer was not displaying the score properly on the last display because of the position of our seven segment decoders within the serilaizer. We found that decoding the display signal before the multiplexor made it easier to implement two different types of decoders rather than placing them after the multiplexor. Here is a link to watch a short clip of our game functioning: https://youtube.com/shorts/3KeSnldMaz0.

When planning to use a button to pause the rows, we knew we would run into bouncing issues and the game functioning incorrectly. To mitigate this issue, we decided to use a separate button for each row instead of implementing a debouncing circuit. This allows our finite state machine to have a dedicated button for each state that is responsible for pausing the LEDS. Therefore, when one row is paused with a certain button, no matter how many times it bounces, the game will not be affected. This is because the finite state machine will simply move to the next state which controls the next row and requires a different button to be pressed to pause that row. We used this technique as a way of simplifying the circuit and making it more reliable.

Conclusion

The key takeaways from this experience would be collaborating as a group to implement and utilize our knowledge to make our idea a reality. Even if you think your initial design is perfect, there can always be modifications to help improve your project. When our group first brainstormed how we want to make the game, we thought the approach would be to use LEDS in rows on a breadboard. As time went by as to how we would construct the design, we believed that it would be far simpler to use the 7 segment displays on the FPGA board to show which LEDs are lit up while still meeting all requirements. Something that could be improved for our project could be manipulating the speeds of each row using multiple switches to make it harder for the user to stack the LED rows. Since the user is limited to connecting only three LED rows, having a method to make it more difficult would make this game more effective and enjoyable.

References

Some VHDL circuits written by Dr. Llamocca were used as references and components to complete the project.

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