FPGA Stacker Game

Jacob Lamberson, Benjamin Rojewski, Nicholas Spanos
Electrical and Computer Engineering Department
School of Engineering and Computer Science
Oakland University, Rochester, MI
e-mails: jlamberson@oakland.edu, brojewski@oakland.edu, nspanos@oakland.edu

***Abstract* — Implementation of the arcade game "Stacker" on a programmable FPGA board as a demonstration of knowledge gained in class.**

I. Introduction

The scope of our project was to come up with a functioning device that showcased our knowledge and comprehension of the topics presented throughout the semester. This report will walk you through our journey towards accomplishing this goal, and we will start with our methodology.

Here, we will discuss our reasoning behind choosing stacker, and we will expand on the three major proponents of the project: the oscillators, the memory storage, and the memory selection and display. We will then discuss our experimental setup and explain how we tested our design's functionality. Afterward, the report will highlight the group's results and discuss the importance of our findings.

Additionally, we should mention that our project incorporates almost all of the learning objectives specified in the course. For example, the project has both combinational and sequential elements, such as registers and finite state machines representing the latter; and decoders and multiplexers representing the former. VHDL was used to implement our design and seamlessly incorporate these components; this implementation was only possible thanks to our grasp of the software. This takeaway is important, as it implies that we have come away from this course learning a practical skill that may serve us well in our future as engineers.

II. Methodology

The game we attempted to implement on our board is commonly present in many arcades and called "Stacker." The following video provides context:

Game Reference

The following is a description of the design methodology of the project, which was subdivided into three distinct sections based on their functions; oscillators to generate the necessary data signal, some form of memory to store data when the user indicates the stop, and a custom-made serializer to send the data concurrently to the 8x8 LED matrix display.

A. Oscillators

The idea behind this part was that a signal of oscillating bits was needed to store in memory to play the game. To implement this portion of the project, three unique FSMs were leveraged along with three genpulse counters and three flip-flops with enable. The genpulse counters are constantly running, and their z outputs that indicate the reset are what signals the FSMs to enter the next or previous state. The FSMs also control the flip-flops' input "D" and their enables. The flip-flops' output "Q" holds onto a value that tells the FSMs whether to go to the next state, or the previous state, in order to make an efficient oscillating signal for later use. There are three of these units in the "Oscillators" portion that are responsible for generating a signal with three blocks, two blocks, or one block, depending on the player's performance during the game. The player may select between each signal using onboard switches as the select line of a multiplexor connected to the three signals. The output of the multiplexor, which is called "data" is sent over to the next portion of the project, the memory of the system.

B. Memory

The memory of the system consists of eight registers with enables that are each eight bits wide. Therefore we have eight rows and eight columns of data, one for each LED on the matrix display. Each register is connected to the "data" output of the oscillator multiplexor at the same time. The register that receives the data for storage is selected through 3 onboard switches that operate a 3-to-8 decoder with an enable. The decoder's 8 outputs are each attached to an enable of a register in the memory bank. The enable of the decoder determines whether it has any output or not, and it is tied to yet another onboard switch, which we have opted to call the "drop switch," since turning it on lets the player see the blocks oscillating on the display, and turning it off stops the blocks where they are, which we called "dropping." The game requires a total of 6 onboard switches; three for memory address, one for memory enable, or the "drop switch," and two to operate the oscillator multiplexor to select how many blocks are in play. The data stored in the eight registers are then sent to the final stage of the datapath, the display portion.

C. *Display Serializer*

The display portion of the project is possibly the most important component of the entire project's design, as it creates the user interface that the player uses to actually play the game. Without it, the board would be doing seemingly nothing, according to the end user. With that being said, the display portion had to begin with the physical component that would be doing the displaying: the LED matrix. For this project, we opted to use the 1088AS 8x8 LED matrix as our display, simply because that is what we had on hand. As the datasheet shows, wiring the matrix is not very straightforward to wire for controllability, but after some testing, it was sorted out. The way the matrix works is as follows: the unit has sixteen pins, eight anodes and eight cathodes. The anodes and cathodes can be thought of as rows and columns controls. When a powered anode lines up with a grounded cathode, current flows, and the LED lights up. For example, if you ground all the cathodes and power one anode, eight leds in the same row will all light up. In order to send our register data to the display, we would need to quickly scan across each register, and send its data to a row of the display to be shown to the player. Since anywhere the power and ground intersect, the current will flow, we couldn't simply ground all the cathodes and send the register data for display, or we wouldn't be able to show separate rows as their own data. To work around this issue, we had to design our own serializer to send data to each row individually, at a rate fast enough that it is undetectable to the human eye. This was implemented using an 8-to-1 multiplexor, a 3-to-8 decoder, a custom FSM, and a genpulse counter. The counter is the beginning of the display subsystem, counting high enough to emit a "z" reset signal every 500 μs, fast enough to be invisible to the human eye. This "z" reset signal is sent to the FSM, which changes states as the "z" becomes high for one clock tick. The FSM outputs a three bit signal that counts up in binary, which is connected to the select line of the 8-to-1 multiplexor. The multiplexor's inputs are connected to each of the eight registers from the memory section, and its output is the data sent to the eight cathode pins of the matrix display. The three bit counting FSM output is also sent to the 3-to-8 decoder, and this decoded output is sent to the anodes, making only one column grounded at a time. With all these components together, the basic function of the display circuit is to send one row of data to the cathodes and ground the respective anodes in order to turn on the correct LEDs in their rows, without impacting the function of any other row on the board. The FSM output is changed so frequently that the LEDs light up and do not flicker to the human eye.

With all of our subdivisions completed, the last step is to put them all together. Once connected, the data is generated by the oscillators, told where to be stored by the memory bank, and quickly scanned through to be displayed by the display serializer.

III. Experimental Setup

Our project setup incorporated both software and hardware solutions to test the functionality of our final design. As previously mentioned, we coded our project in VHDL, programmed it into an FPGA board, and then translated our code into a hardware application via an 8x8 LED matrix board. This may sound simple, but there were a few hiccups along the way that we were able to correct thanks to VHDL's tools and our experience with hardware. On the software end, VHDL's synthesis and behavioral simulation were critical in highlighting our errors. For example, the initial code for the memory storage and address section ignored the address input in the decoder, as each register enable was mapped to the same enable used for the decoder. This was not something we initially realized, but after running a few simulations and retracing our code, we found the issue in the local top file and fixed it. Once we had finished diagnosing errors and our simulations ran according to plan, we created an xdc file for the FPGA board to talk with our code.

As far as hardware goes, we had three main components: Our Artix A7-50T board, a breadboard, and an 8x8 LED matrix board. The Artix-A7 is undoubtedly the most important of the three as it served as the foundation for the project. The board was used to implement the code, but we also incorporated six of its switches, the CPU reset button, and both sets of p-mod ports into our project. The switches and reset button were particularly important as they were used as inputs to the system. Flipping switch 1 dictates when to stop the oscillation and switches 2 & 3 were used to count in 2- bit binary to select the block count displayed. Switches 13, 14, and 15 were used as a 3-bit binary address for the memory storage section and the CPU reset button can be pressed to reset the board. The p-mod ports were used to connect the FPGA with the bread board and the 8x8 LED board, where we visually displayed our code. The visuals took the form of 3, 2 or 1 lit LEDs oscillating back and forth in a predetermined row, where the row was sent 3.3 volts and then the individual diodes were grounded, in order to prevent the entire row from lighting up. When we first tested our design, we burnt out our matrix board because the current was too high, so we decided to add a 220-ohm resistor in series with each row of the display.

IV. Results

After the project was set up as described in the experimental setup, the project operated as shown below.

Video Demonstration

We can see that each row is able to be controlled by address switches, as well as how many blocks appear on that row. It is also shown that previous rows persist on the matrix. We had three separate major components to this project, and we learned from each. The oscillator portion involved FSM's, which required unique construction of each one, as well as how to time them properly for the rest of the circuit. The data storage component required taking what we learned from Lab 5's RAM emulator example, and expanding on it so it could handle the data given from the oscillators, as well as pass it to the display efficiently. Finally, the display portion required taking inspiration from the seven-segment display example for a stopwatch and adding a serializer to make sure the data is cycled through in order to display game data correctly.

There were several challenges faced while completing this project. During development, there were three major issues we had to overcome and solve; the first was the wiring diagram of the 8x8 display, as it was very confusing, but after some time with it, we were able to wire it up correctly to our needs. After it was wired up correctly, we had an issue where the display was inverted, meaning the blocks that needed to be lit were unlit, and the background was lit. This was then fixed by inverting the cathode signal to the columns on the display. Finally, there was an issue we still don't fully understand, but have solved. For some unexplained reason, the oscillators were generating data that would skip every other position on the display, only on odd sets of blocks, or even. to get smooth block movement, we had to tune the genpulse counters attached to each FSM until it gave smooth output. The hypothesis as to why this was happening is that it may simply be a nuance of the display hardware used, but this issue only required some guess-and-check tuning until all the oscillators were outputting smooth signals for use.

Besides these brief interruptions, development went smoothly and had no major setbacks or troubles, and the results we got from the project were to be expected from the amount of work and effort we put into it.

Conclusions

We gained a lot of experience during the development of this project. The points that stood out the most were adjusting scope and working as a team to complete our goal. Our scope originally included a VGA display along with game logic, so all we would need is a button to "drop" the blocks and a button to reset. Partway through development, it became clear that a system like that would simply not be possible in the amount of time we had. Collectively agreeing to simplify the scope to a 8x8 display, and to use switches and the honor system to control game logic gave us the time we needed to complete our project and make it function. If we had more time, we would be able to add these functions to the project. However, our lack of these things is not from lack of knowledge, but instead of time. We can all say that we are very proud of how we did, and are happy with the results and how they function.