Spaceship Battle Game

FPGA Video Game

Mathew Plaza, David Smith, Tao Wang
Electrical and Computer Engineering Department
School of Engineering and Computer Science
Oakland University, Rochester, MI
e-mails: dsmith6@oakland.edu, mathewplaza@oakland.edu, taowang@oakland.edu

***Abstract*—In this project a Spaceship Battle game was created in VHDL. The purpose was to create a fun project while demonstrating the information learned throughout the course.**

INTRODUCTION

The purpose of this project is for the team members to develop mastery over the FPGA by designing a game, datapath, and interfacing with peripheral devices. This game was aimed at being unique, arcade styles, and simple to learn. The spaceship battle game is a simple game, where two players compete to hit the other's ship. To do this, they must move their ship to simultaneously aim at the opponent and dodge the opponent's shots.

The scope of the project was to implement this game on the Nexys 4 board with peripheral I/O hardware. This game required integration of many topics covered throughout the course, including finite state machine design and implementation, VHDL programming, and external hardware interfacing with PS/2 and VGA. A topic that was utilized in this project that was learned outside of the course included the addition of procedures into the Vivado project library. This project can be applied to the industry of video games. The Spaceship Battle game shows the fundamental design of the hardware of a video game.

METHODOLOGY

PS/2 Keyboard Interface

One player's controls are operated on the Nexys board buttons,and the other player will use an external keyboard to make the game more practical for 2 players. The VHDL code provided on Professor Llamocca's website was used as a starting point for this exercise [1]. Figure 1 outlines the keyboard interface.

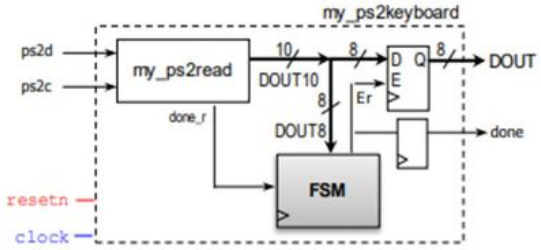

Figure 1: PS2keyboard Block Diagram

The interface has two single-bit inputs which are read serially to produce an output consisting of 8 data bits, a parity bit, and a stop bit. However, the FSM of this circuit underwent a modification from the original design that was provided. Instead of outputting data on the F0 key-up code, the circuit outputs data when a key is pressed or held. When the F0 key-up code is read in DOUT8, a sclr signal is sent to the register, making DOUT = 0.

Game Design

The game's mechanics were broken down into 2 processes: movement and projectiles. Once a player presses their respective fire key (spacebar or A7 button on the Nexys A7), a projectile is created directly in front of that player's ship. The projectile then moves in a straight path vertically until it either hits the other player or reaches the end of the screen. When one player hits the other, the winner's ship will light up, and the other ship is removed from the game. Once the resetn button is pushed, a new game will start. Figure 2 describes the projectile logic. The player on the Nexys A7 controls has their key pushes checked first, so their projectiles will leave slightly before the second player. Through timing simulation it was determined this timing difference is so small it will not impact a real game.

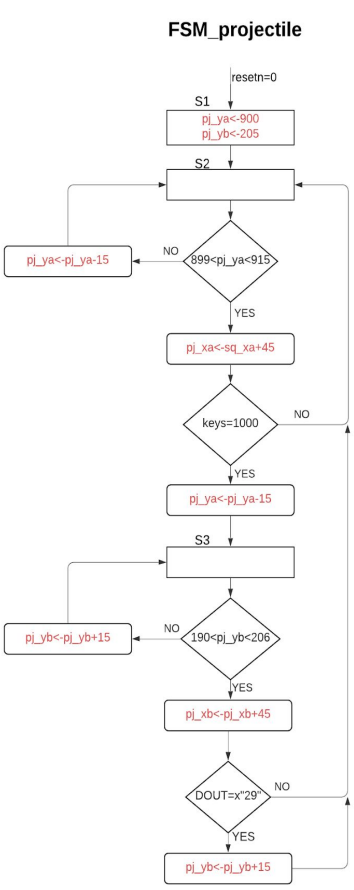

Figure 2: FSM_projectile Block Diagram

The second portion of the game design is the player movement, which is described in Figure 3. In the first state, the initial positions of the players are defined. While the game is running, this FSM continually checks the first player's inputs followed by the second player's inputs, and moves each ship accordingly. Similar to the projectile, one player is checked before the other, but the timing was determined to have no impact on gameplay. A package of shapes (provided by Anton Toni) was created to define the shapes of ships and projectiles [2]. For this game only the two shapes were designed, however the benefit of this implementation allows for easy expansion to include other art. The last component of the game is win detection, which was implemented by checking the position of the projectile each clock cycle to see if they match.

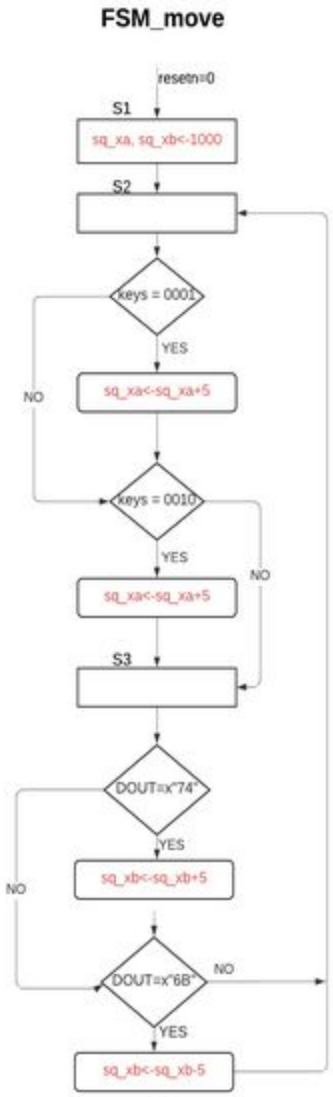

Figure 3: FSM_move Block Diagram

VGA Display of Game

To display the game, a VGA screen was chosen because the Nexys FPGA has a VGA connector on board, allowing for relatively easy integration. The VHDL code for the top file and the VGA driver is a variant of the one created by Anton Toni [2]. From a high level, this portion works by cycling through every pixel by row and column and outputting red, green, and blue information for each pixel. The projectile and ship shapes are stored for easy access, requiring an initial position and color as inputs. Figure 4 shows the modified VGA controller logic, which cycles through each horizontal and vertical position.

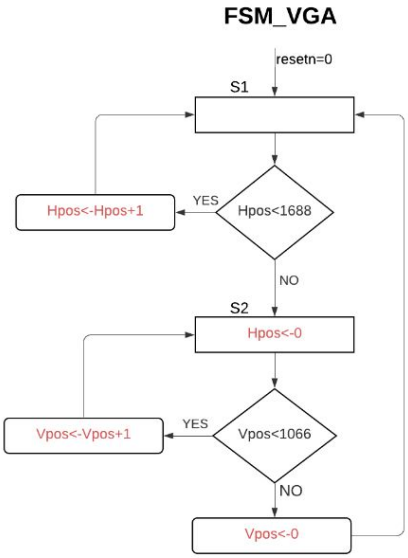

Figure 4: FSM_VGA Block Diagram

LED Display of Score

To keep track of score, a serializer was used to output on the integrated 7-segment LED displays. A single display was used for each player, allowing for scoring up to 9 points. Each score is kept through a counter that receives an enable signal when a win is detected. This code was developed for an earlier lab, and adapted for two screens. Figure 5 outlines the adapted serializer. Each time the FSM receives an enable signal from the counter, it selects the screen address and data from the mux because data can only be sent to one screen at a time.

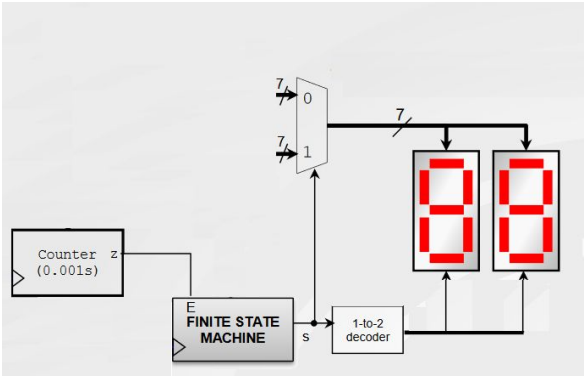

Figure 5: LED Display Implementation

EXPERIMENTAL SETUP

The game was implemented on a Nexys A7 FPGA along with VHDL coding in Vivado 2019.2.

All code was written inside the Vivado program. Each component/module was designed and simulated separately from one another to ensure success between each component. It was possible to see if any mistakes were made in the code from the timing and behavioral diagrams. After completing any troubleshooting needed, we implemented the code in the overall design and then moved on to the next component. This allowed the project to be completed smoothly in increments and in a timely manner.

The code was written for a 1688x1066 VGA screen, and would need minor changes to work with a different resolution screen.

RESULTS

The Spaceship Battle game works as intended, and is considered successful. Figure 6 shows a game in progress. Figure 7 shows a player's victory screen. No bugs in the game have been found through testing, however there may still be some that are undiscovered.

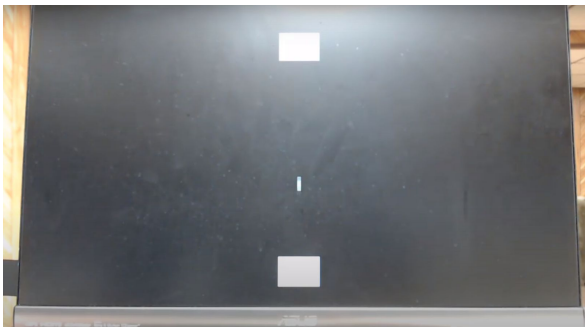

Figure 6: Spaceship Battle Game in Progress

Figure 7: Player Wins

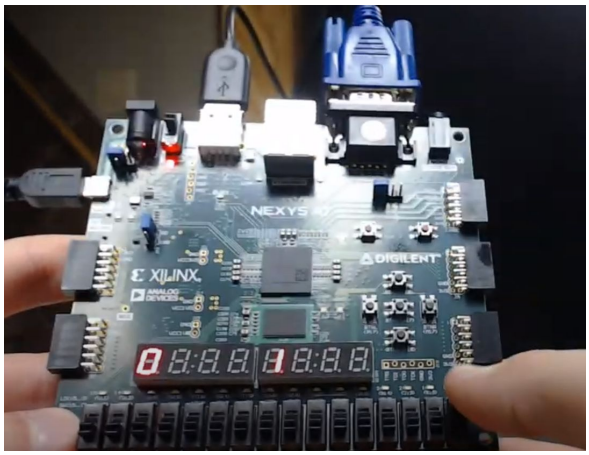

Figure 8: Score Display Example

Testbenches were used to debug and verify correct implementation of each part of the game. Figure 9 shows an example testbench where both players are pressing keys and moving simultaneously. Keys[3..0] and DOUT[7..0] represent the player inputs received by the game. sq_xa and sq_xb show the horizontal position of each player on the screen. When the move signal is received from the player inputs, keys=1 and DOUT=x"6B", the player position begins to move until a different input is received. The "fire" command is then simulated with inputs keys=8 and DOUT=x"29". When these commands are sent, the projectile vertical positions, pj_ya and pj_yb, begin to increase. When one player's projectile reaches the other side of the screen and hits the other ship, win_a goes high to signal this round of the game is over.

Figure 10 is an example simulation of the scores output to the LED displays. On the rising edge of win_a, the counter for score A increments. Counter B increments on the rising edge of win_b.

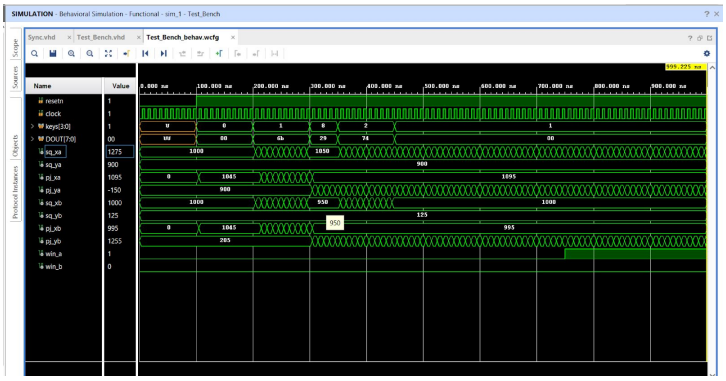

Figure 9: Movement Timing Simulation

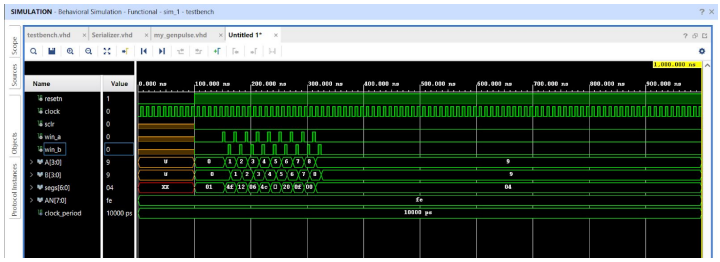

Figure 10: Scoring Simulation

CONCLUSIONS

Overall, the game creation was a success. While the project seemed overwhelming at first, breaking down the game into smaller blocks allowed us to focus on each section individually before creating the final product. The VGA interface was the least familiar to us and required the effort to perfect. Although the goals were achieved and the game is functional, there is room for further improvement. Some potential additions could include a single player mode, sound effects, a start screen, or more complex art. Each of these improvements could be constructed as independent blocks and added to the whole project.

REFERENCES

[1] Llamocca, Daniel. "VHDL Coding for FPGAs." VHDL Coding for FPGAs. Web 23 April 2020. http://www.secs.oakland.edu/~llamocca/VHDLforFPGAs.html

[2] Toni, Anton. "VGA". Github. Web 18 April 2020. https://github.com/AntonZero/VGA.