The Island

List of Authors (Garrett Heiser, Mark Heiser, Alex McInerney, Tyler Wiegand)
Electrical and Computer Engineering Department
School of Engineering and Computer Science
Oakland University, Rochester, MI
e-mails: ghheiser@oakland.edu, maheiser@oakland.edu, ramciner@oakland.edu, tjwiegan@oakland.edu

Abstract: Our project consists of making a game that is a resource survival game. It uses a keyboard interface to increment the value for various actions that may be performed. It then outputs these values via VGA interface to the user. This allows the user to make new decisions based off of the data presented.

I. INTRODUCTION

The island was influenced with our love of video games. We wanted to give the user a rewarding yet stimulating gaming experience that had a sense of difficulty to the game. This was implemented using a keyboard and VGA display. The brain of the project was the finite state machine. Within the finite state machine was all of the important logic in order to tell if the user was winning or losing. The VGA enabled us to use a variety of ideas in order to capture the best player experience. The keyboard was a massive help in giving us the creative freedom to tell the user to click multiple buttons and add different dimensions to our game. Throughout the course of the report each of these topics will be further elaborated upon.

II. METHODOLOGY

A. Game planning

During this phase of the development we had to come up with arguably the most crucial part of the project. We needed to come up with a concept that would inspire generations to come to want to play. We needed something that used an input and output interface in tandem with the FPGA. To start with we decided that we would make a text based game and throughout the course of the game development it had different visions for the future. Eventually we settled upon a resource survival game because of the originality that it brought to the table.

B. VGA Display

We needed to come up with a way to display the information to the user. The only way that seemed like a logical decision was using the vga to display to a monitor. This proved to be one of the trickiest parts of the project. At first, we were going to use the professors vga control code [1]. We then were going to upload 256x256 photos for each state that was active. Yet, this seemed not beneficial for repeatability. The next task involved figuring out how to display multiple letters on the screen at the same time.

a. Creating the letters

The letters for the project were created using paint. We resized the images in paint to 32x32 pixels. This allowed us to place multiple letters next to each other to form sentences. In order to use these newly created letters we needed to change them to text files. The professor provided us with a script to MATLAB that converted the png to a text file with ease. [1] Allowing us to proceed to the next step of our project.

b. understanding HS and VS

These images now needed to be placed on the screen, but they needed to be in certain locations. It was found that the VGA accepts 5 inputs, R, G, B, HS, and VS. It was easy enough to see that RGB signals were in charge of choosing what colors to display onto the VGA display. HS and VS were relatively new parameters to our group. We had heard of them in lecture, yet we had never worked with them. After creating a quick test bench, it was very easy to see that these two parameters acted as counters.

c. RAM

Each of these text files needed to be loaded to the FPGA and held somewhere in order to be called at any given time in the code. We had learned that RAM could hold values for us. So to start the program off we loaded each RAM with a different image text file. Then to call these files to display on the screen we used a simple multiplexer that gave out

each RAM out data depending on the select.

d. Graphing the different images

Finally, we had made it to the point that we could put images onto the VGA. In order to plot one of these letters we needed to be greater than the last image and less than the next image to be displayed. Since each image was exactly 32x32 pixels it was easy to do the math in order to find the locations for each image. The outputs of the FSM also needed to be taken into consideration in choosing which screen to show. For example, the game needed to display the start screen then once the game began the actual game screen needed to be displayed. This is done by keeping track of a day count signal within the finite state machine. Finally, the last part of the VGA was figuring out how to update the number count each time a value was incremented or decremented. This was done by creating a mux that when a different value was presented from the state machine it would pass the proper codes for the numbers that needed to be presented to the user.

C. Finite State Machine

The finite state machine is the brain behind this elaborate game. The first state of the game is used in order to display a screen to the user saying to press B in order move on to the actual game. The FSM will then wait for input from the keyboard controller to move on to the next state. Once the actual game has started you will stay in state 2 until the escape option is selected. The escape button will move the player through the other various states, where the player has the option to collect food or water, build shelter, increase the maximum capacity, or escape. If at any point, the food, water, or shelter values drop to 0, or the turn count increments past 64, the FSM will go to the final "dead" state. Each action in these states requires input from the keyboard controller and depends on which key is pressed. The "win" state is reached when the player increments the escape value above 10. Each state has the same logic repeated. If you click food, water, or shelter that resource increments by 4 while the other two resources will decrement by 1. If you exceed your tank size then your resource will return to only having 1 of that resource. At any given point if you run out of a resource you will die. There is also a day count of 64, if this is exceeded then you will die as well. The only way to win is to use escape 10 times.

D. Keyboard

Control of the keyboard inputs to the code was handled by two sets of codes working in unison. A PS/2 port was utilized with a USB connection to a keyboard through which a single bit wide data stream could be sent and used to determine the letter pressed on the keyboard and send it to the FSM. The two codes controlling this process included the PS/2 interface for communication with the keyboard and a FSM specifically designed to read the appropriate scan code of the pressed key and send it as a single 8-bit wide bus to the FSM controlling the state of the game.

a. PS/2 Keyboard

The keyboard is programmed to send a scan code, which is unique to each key, which is 8 bits along with a stop and start bit indicating the beginning and end of the scan code. Upon pressing a key, an "F0" signal is sent to the board and shifted in via registers to be detected. After the key is released, the same process is repeated including a '0' stop bit and a '1' stop bit with the scan code to the key being sent and shifted in. Both of these scan codes are then sent to the FSM along with a done signal indication of the full 8-bit scan code being correct.

b. Keyboard FSM

The keyboard FSM is used to control the letter signal to the Finite State Machine for the entire game control. The FSM receives the first done signal and checks that the scan code is an "F0" to indicate a key has been pressed. On the next done signal, the new scan code being sent from the PS/2 keyboard code corresponds to the scan code of the key. The signal is output as a letter to the FSM for the overall game to change the state and control play. The letter signal is returned to all zeroes and the FSM resets to await the next done signal for a new key press.

III. EXPERIMENTAL SETUP

The setup of our project consists of several different modules created in Vivado. One module for each component that was used. For example, one module for the VGA and another for the keyboard. For each module, we created a test bench in order to test the modules separate of one another. We found it easier to debug our vhdl modules when there was less involved. It was easier to test functionality each step of the way in order to achieve the greatest accuracy and no unforeseen bugs. For the VGA module, we also were able to use the VGA screen to troubleshoot. Since when we did the test bench the RGB values were not

able to describe if the project was working. To start we would insert an arbitrary sentence onto the screen to make sure in fact that each image displayed in the proper location. Once this was achieved we needed to move on to trying to change screens from the start, dead, win, and main screen. Once we confirmed that each of these modes were functional we started to create sentences and put the actual game information into the code.

IV. RESULTS

The main result of our game was pretty easy to see. The start screen was initially displayed. Then after pressing B the next screen was to be displayed. In fact, this is what occurred. On the main screen, every time you incremented food, water, or shelter the number incremented was to go up by 4 while the others went down by 1. This occurred each time as well. Another requirement is that every time that you exceed your tank size the amount of that resource is to go to 1. If you run out of any resource at any given time then you are supposed to go to the dead screen. If you manage to survive 64 days without escaping you will also die. The only way to win is to increase the escape value above 10. This is not an easy mission as when you select escape every other value decrements as well. So, if you run out of food, water, or shelter while trying to escape the treacherous island you will die as well.

CONCLUSIONS

From this project, a lot of knowledge was gained. We now know how to interface with an input and an output with an FPGA. There is many improvements that could be made to our project. We could make a high score table in order for contestants to compete against one another. Another improvement could be to make different difficulty levels. This can be done by changing how the program increments each value. As well as easy mode could display your day count so that you can be absolutely certain that you escape the island in a timely fashion. We had some issues that if we didn't assign a value properly it would choose to display random images from the RAM. We could delve more into this to determine why this is happening. In the future, we can use the knowledge we have gained of the VGA to display information that our modules compile to the user. With the keyboard code, we gained the knowledge to be able to submit a ton of very different inputs to our system, rather than the few buttons that the standard FPGA board offers the user.

REFERENCES

[1] Daniel Llamocca. "VHDL Coding for FPGAs". http://www.secs.oakland.edu/~llamocca/VHDLforFPG As.html