National Tsing Hua University Computer Science Department
11210CS210401 Hardware Design and Lab

Final Project Report on:

Legend of Chun-Yi Lee

Project carried out by:

Team 6

with members:

林楷源 111000169

林哲宇 111062311

Under the Supervision of:

Professor Lee, Chun-Yi (李濬屹) and His Teacher Assistants

Table of Contents

Introduction

The project created has the title, Legend of Chun-Yi Lee, a direct inspiration from the Legend of Zelda, a classic game many people adore. The game utilizes a 2D scrolling game design with 8-bit-esque design fashion for the graphics. There will be a number of waves with a variety of monsters to defeat. The storyline takes a fictional story involving a professor from National Tsing Hua University, who becomes the only hope after the university has been infected by an unknown virus that spreads amidst the students. He needs to uncover the truth behind this ordeal and return NTHU to what it once was, a peaceful university where students from all over the world can study and research their respective fields with freedom. Thus, he decides to embark on a journey to rescue NTHU from the clutches of the virus. As the story progresses, there will be several NPCs that he can interact with to obtain items that can assist him in achieving his goal. At the end of the journey, he will encounter a final boss that was the source of the virus, and he has to defeat it. After defeating it, NTHU will be saved and this eventful, unordinary chapter of his life will end, as he moves on to the next. In regards to the technicalities of this project, Verilog is the main programming language used and a Basys 3 FPGA is used to execute the game. There are more materials and equipment used alongside the FPGA, and will be further elaborated in the System Specification section of the report. The class was given plenty of time to plan and create their respective project, however, for this group, the starting date of the project was pushed back, due to other requirements from other classes. There were also several difficulties that the creators ran into as they were programming some of the specifications of the game, which took a significant amount of time to find workarounds. The main thing, however, was that there was not enough LUT in the FPGA spec to fully create the game, thus, we had to make adjustments in the overall game design, so there will be some significant difference from the original game. As such, there were several compromises that even the creators are not necessarily happy about. However, it is of utmost importance to, at the very least, have a game to present by the end of the deadline. With that understanding, readers of this report should note that it was not completely the original vision of the project. Throughout this project report, the details of each part of the creation process of this project will be discussed and explained.

Motivation

The initial idea of this project was raised by 林哲宇, one of the creators of this project. Then, after a back and forth discussion, it was decided that this project will be about a parody version of the Legend of Zelda first edition, which was played on the Nintendo Entertainment System (NES). The idea was, instead of Link, who is the original main character of the game franchise, the professor who is instructing this class will be placed as the main character, which is why the game is called Legend of Chun-Yi Lee. Since the game is parodized, the characters' designs will be different as well. This meant that the creators had to create all of their assets from scratch, which will be seen in the system specifications section. However, at its core, the game is still inspired by the Legend of Zelda. Thus, the gameplay will be very similar to the original game, with a few adjustments here and there. There are also some easter eggs that include other games we enjoy, as well as various materials and equipment that were used in this class. As the creators are limited in what they can achieve with the capabilities of the FPGA and the programming language, the game is by no means up to the standards of other games today. However, it still can serve as a tribute to the legendary game and a fun little project for the class.

System Specification

Storyline

The basic outline of the storyline is written in the introduction. The full version is written below:

> Once upon a time, there was a university called National Tsing Hua University or NTHU. Students are able to study and research any topic of their interest with plenty of aid from the school. They enjoy their campus life, free of worries from dangers of the society. It is a place that nurtures the next generation of young people who will lead the country in different fields in the future.
>
> This university, like many other universities, allows outsiders who are not students to enter freely. This allows many families to come visit the university during the weekends, usually bringing their kids along with them. However, one day, there was a suspicious outsider who entered the premises. No one suspected a thing, because of how often outsiders come inside the university. Suddenly, he said, "it's spitting time," and started spitting all over the place. This brought forth a mysterious virus that humanity has never seen before and infected a majority of the student and faculty population. Only the lucky few who are stuck in laboratories and those who are not on campus at the time were saved from it.
>
> The government then decided to quarantine the university, not allowing anyone out. The surviving students inside the university grounds are starting to give up, but when all hope seems lost, a hero has risen to save NTHU. His name is Chun-Yi Lee, a renowned professor from the CS department. He decided to take matters into his own hands and embark on a journey to return the school back to how it was before. However, when he stepped out of the EECS building, he realized that there were too many infected students and it was impossible to fight all of them. Thus, he devised a strategy, and he calls it, "the best defense is a good offense." He decided to lure infected students and fight them batch by batch. He continued defeating the infected students until the first infected student arrived. His TA, En-Ming Huang,

arrived in time to tell him that only by defeating this infected student can he save the university, but it will be extremely difficult. The professor, along with his wisdom and strength, fought it for a while, and after several days and nights, he finally defeated it. After he defeated it, all the other infected students seemed to have returned to normal and do not remember what happened. Professor Chun-Yi Lee, no, Hero Chun-Yi Lee has saved NTHU. Suddenly, he found a chest beside the final defeated infected student. He opened it and found a golden FPGA, one that he has never seen before in his long career of teaching and researching. He decided to bring it back as a trophy of his victory, and this unusual chapter of his life ends here, and he will forever be remembered as the hero who saved NTHU from the mysterious virus caused by the man, who was identified as "Sukon", the one who will later create many more disasters, but that is another story for another day.

Assets

The assets needed for this game were mostly created during this project. The designs are generally inspired from the items from the Legend of Zelda, with some twists added into it. Listed below are the assets that this game has and used throughout the entirety of the gameplay.

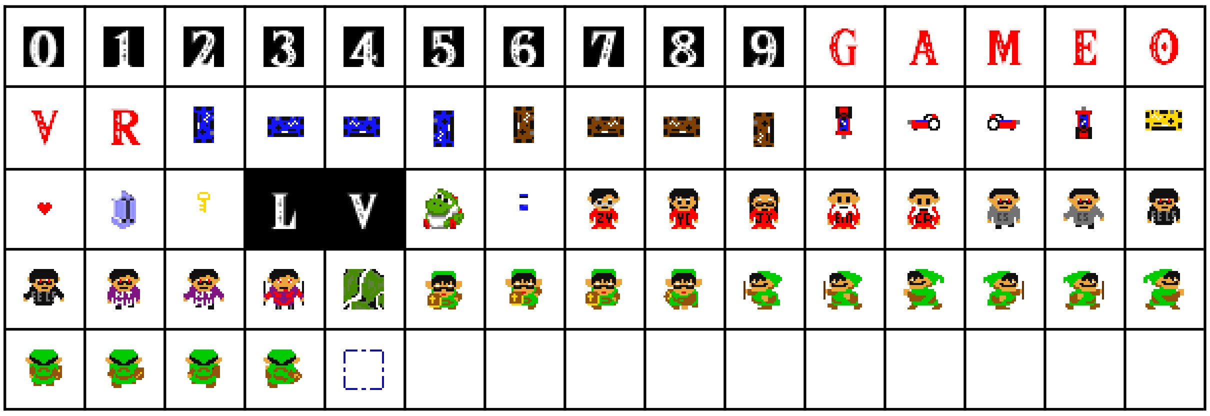

Table 1 - Table of assets used throughout the game

The assets listed are all 20x20 (pixels) in resolution, and the screen resolution of the game is 320x240. These assets are numbers, some letters, Basys 3 FPGA, wooden FPGA, car, golden FPGA, heart, rupee, key, fat yoshi plushie, potion, teacher assistant models, monster models (CS, EECS, NTHU, Boss students), wall, and main character models, respectively.

Hardware

The hardwares used in this project are the Basys 3 FPGA board, a VGA cable to connect to the monitor, a keyboard to give inputs for the game, and a speaker connected using PMOD connectors. The code was programmed onto the board using an micro usb connector with Vivado from a computer that contains the code. Due to the fact that there will be no laptops provided during the demonstration, and the creators of this project do not possess a laptop capable of running Vivado, the usage of memory configuration files will be utilized. The code will be programmed onto the board prior and it will only require a power supply from a borrowed laptop on the day.

Logic

The details of what the function of each connection can be seen in the comments of the Verilog files. This section aims to discuss the general function of each module and how it interacts with other modules to work together for the game. To understand the logic structure of the game, it is best to first look at the overall block diagram of the game, which can be seen in figure 1.

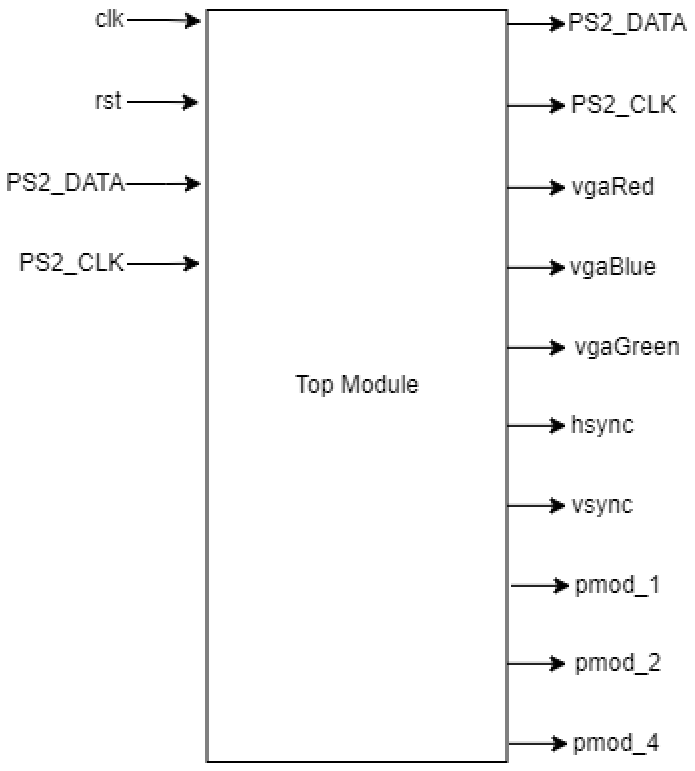

Figure 1 - Block Diagram of Top module.

For the top module, there are quite a number of submodules under the top module. It is to be noted, however, that multiple submodules utilize the same module model and just have different inputs and outputs that connect it. Most of these are for block memory generators and memory address generators. Both of these modules are obtained from the sample code given. The memory address generator is used to decide which pixel to set the color value. It takes in the supposed position values of the pixel as well as counter values for horizontal and vertical and generates an address value. This is then sent as a signal to the block memory generator. The block memory generator is used to take coefficient files via Vivado using IP and generate its pixel data. This pixel data is later connected to the submodule, select_pixel. This processes the pixel data so that the pixel displayed on screen will only be what is needed. For example, if a desired entity is only 20x20, this will avoid the entity taking up the set screen resolution size. Another function of it is that if the entity has different animations (i.e. walking animation), this allows it to select which state to display. The select_pixel module would then output a signal that is connected to the RGB_gen or the RGB generator module which would then decide what the color values of the pixel would be. The block diagram of the modules mentioned in this paragraph can be seen in figures 2-5. It is to be noted that memory address generators, block memory generators, and select_pixel modules are repeatedly used in the top module multiple times. However, the figure only shows a general diagram of the module.

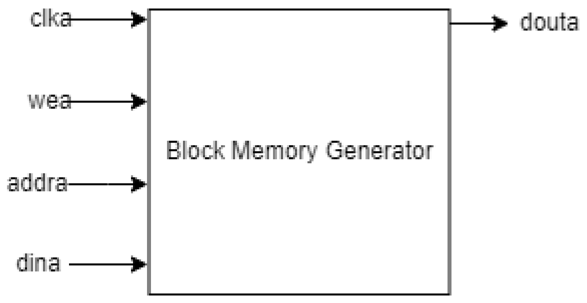

Figure 2 - Block Diagram of Memory Address Generators

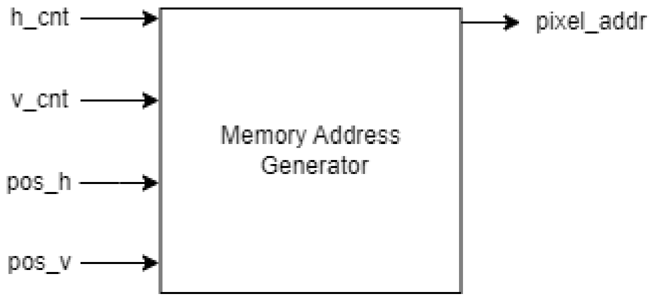

Figure 3 - Block Diagram of Block Memory Generators

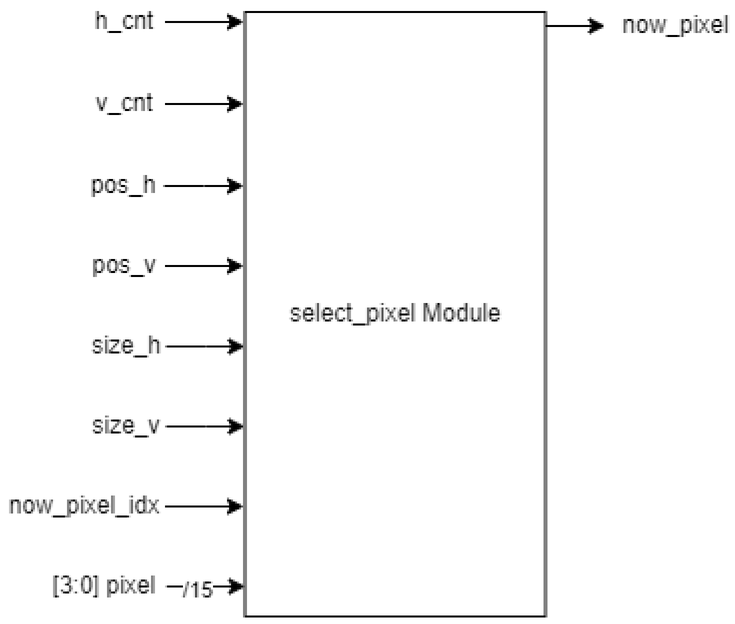

Figure 4 - Block Diagram of select_pixel module

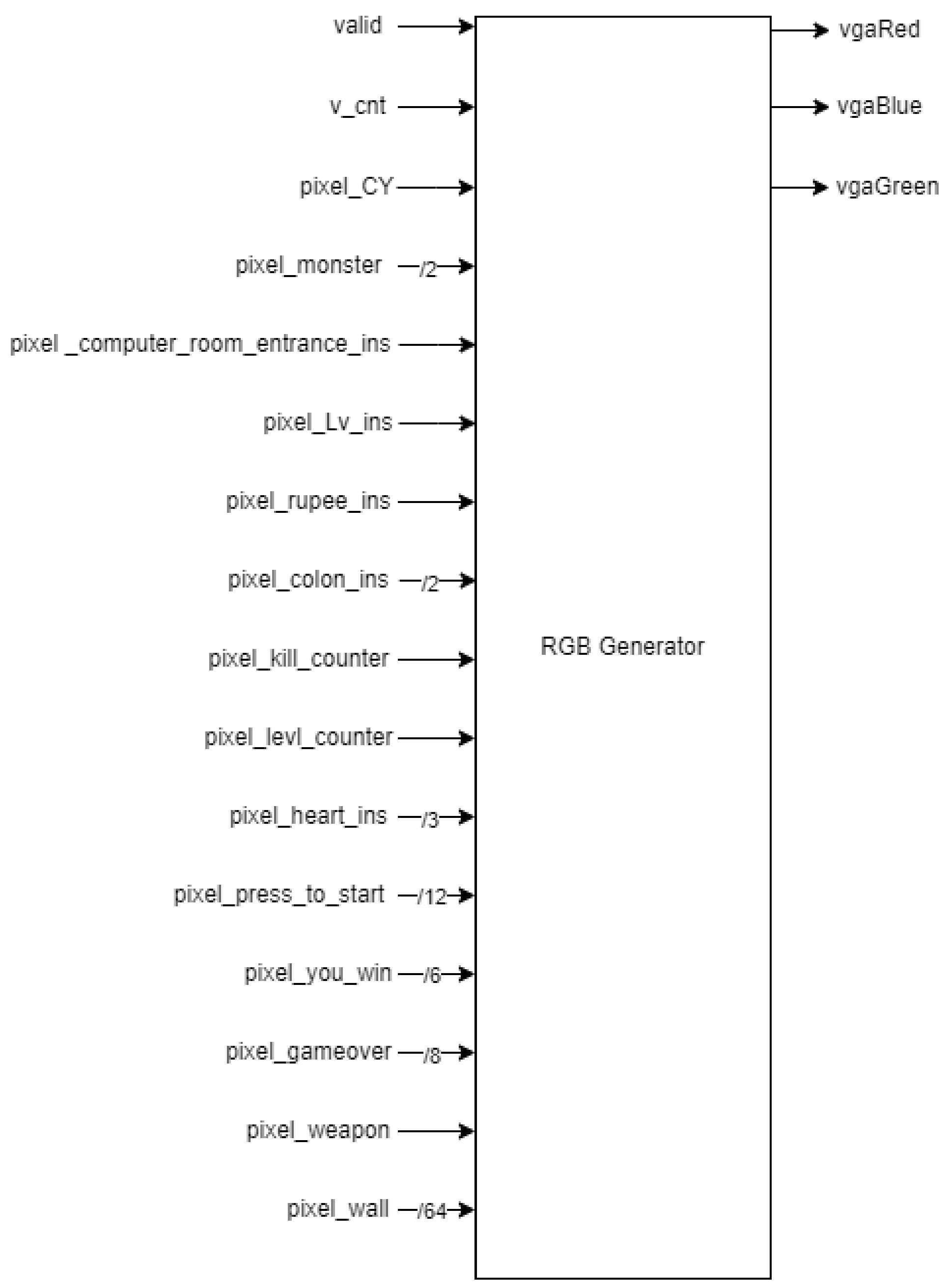

Figure 5 - Block Diagram of RGB_gen

The next module to be discussed is the KeyboardDecoder. This module consists of the protocol of the keyboard to be used for the game. It is also to be noted that a lot of keyboards have different protocols, so it might be possible that this module does not work for another keyboard. This module is also given as a sample code in the previous lab, but how it works basically is the keyboard would send a signal to the FPGA through USB signals, which would get translated to PS2 signals via the PIC24 chip. The PS2 signals will include three types of scancodes, make code, break code, and extend code. Make code represents the key values of each input press, break code represents the action of letting go of the key, and extend code represents duplication of the key. These scancodes are outputted by the KeyboardCtrl module and derived from the PS2 Signals. The next part involves the KeyboardDecoder module, which takes the PS2 data, and last_change which is the key pressed. It also outputs key_valid and key_down to finish decoding the input into a signal for the FPGA to process.

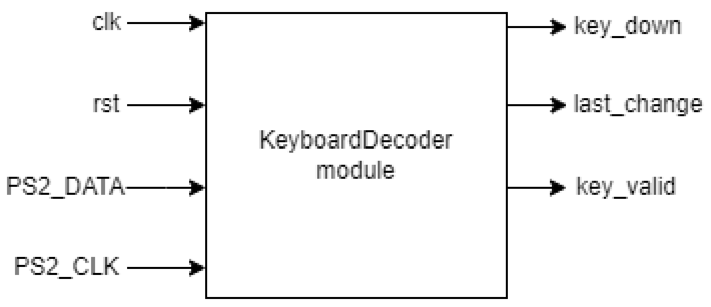

Figure 6 - Block Diagram of KeyboardDecoder

The next module is for the music to be played. There are two pieces of music incorporated into this game. The first one is the main theme of the Legend of Zelda, and the second is an original soundtrack by Ado for the movie One Piece: Red, called Tot Musica. The music module utilizes two PWM modules to control the tone and the beat frequency. There is also a control module that basically takes the beat frequency generated from the PWM module and uses it as its clock signal for a counter to simulate the beat counting. This counter data is then sent to the music sheet module, which consists of an always block with a switch case statement that checks the value of the counter data and outputs the right tone. The tone data is then sent to the top module to be outputted through PMOD connections. For this configuration, a value of 1 of the beat frequency counts as 1/16 of a beat. It is done this way so that it would be easier to recreate a more complicated music beat, especially when dealing with tuplets. Another factor we have to consider is that when the same tone is to be played multiple times in a row, we need to insert a frequency that cannot be heard (i.e. >20000) so that the tone will not be played continuously. Each soundtrack was replicated until approximately the time length of 1:30.

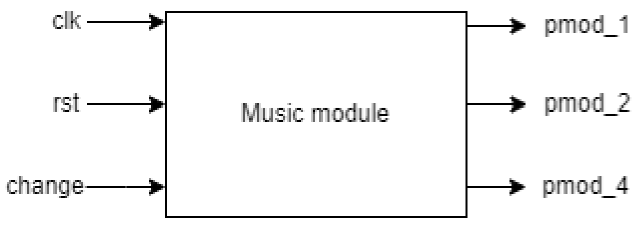

Figure 7 - Block Diagram of Music Module

Then, the core of this game is the state control module. This module’s purpose is to be a central hub for all the state changes and under it, will be the rest of the modules that utilizes the states. In general, the way this game model works is by having a general state machine and several state machines under it. These FSMs consist of both Mealy and Moore machines. To take a further detailed look into this idea, see figure xxx for the general state transition diagram for this game.

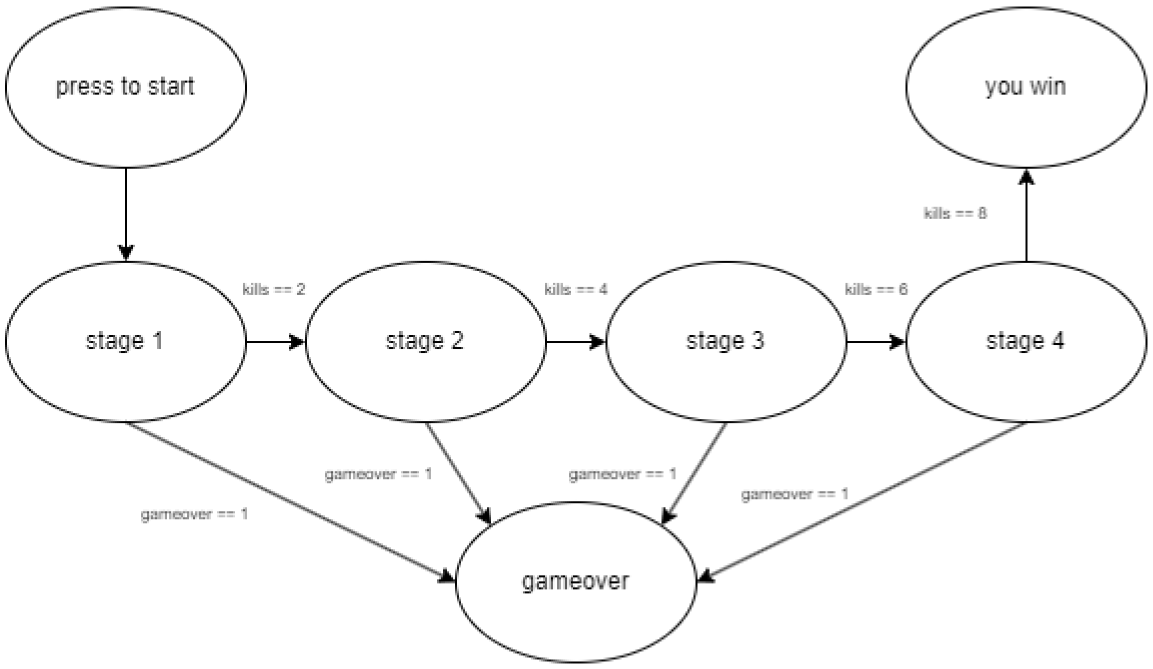

Figure 8 - General State Transition Diagram

As one can see in the state transition diagram, the game in general has three main states, i.e. press-to-start stage, in-game stages (which consist of 4 stages) , game-over stage, and you-win stage. For the press-to-start stage, to transition to the next stage, it simply requires a space bar input from the user. Then, for the in-game stages, it is required to defeat the monsters present in the scene before being able to move to the next stage. There are four in-game stages, ranging from level 1 to 4. Each level consists of a different monster with different specs (further details in the paragraph dedicated to the monster). After defeating the final monster in the fourth stage, the state will transition into the last state, which is the you-win stage. This indicates that the player has already beat the game. If the player is to input another space bar signal, it will reset the game back to the first state, which is the press-to-start stage.

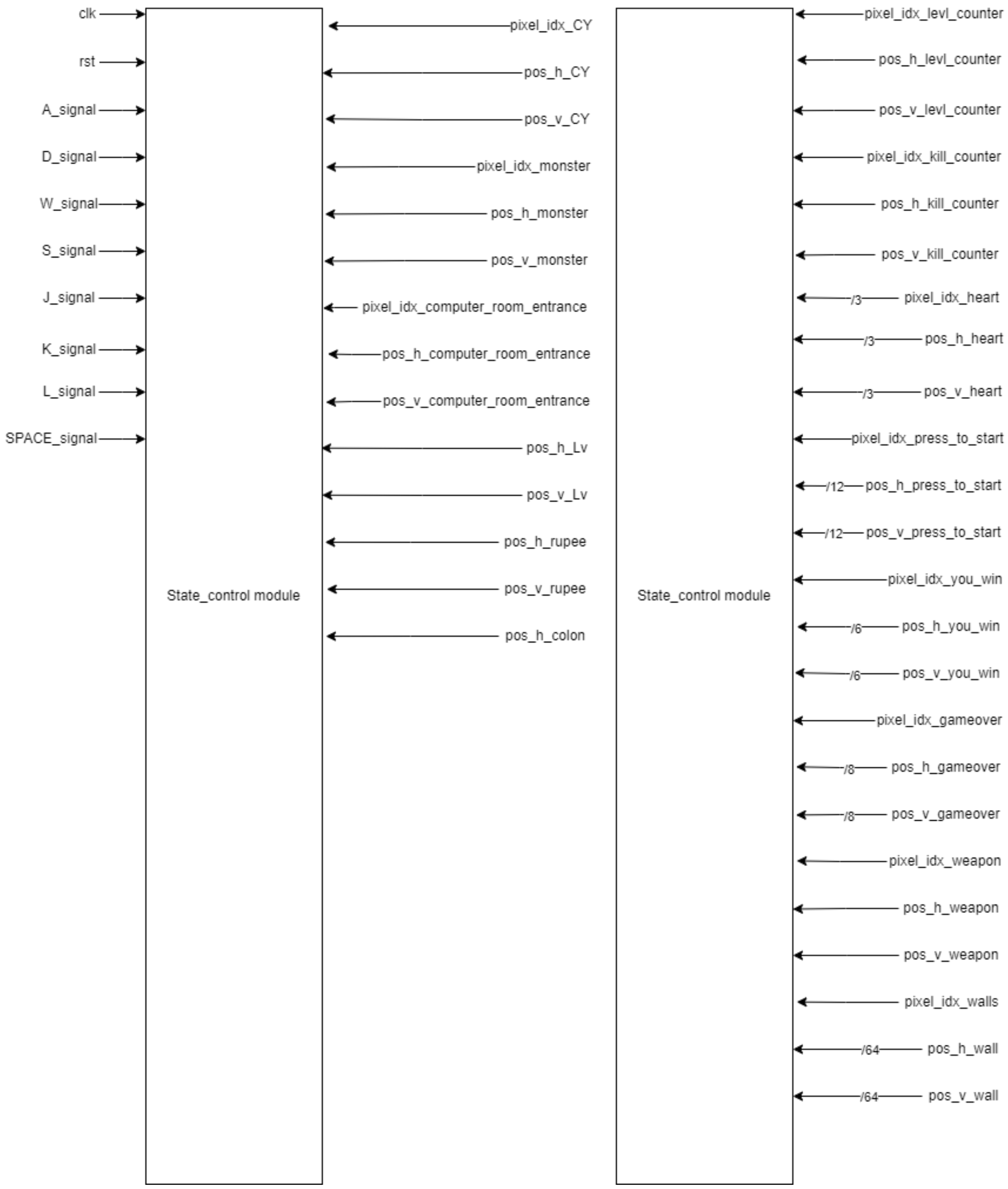

Figure 9 - Block Diagram of State Control Module

Now, since this game essentially involves the player controlling a character to defeat monsters, this module(maincharacter) would use different states to represent the current state of the character based on the input and the character's current state. There are several states for the characters. Some of these states are also impacted by the general stage state. First, the character's collision with monster's will depend on the current stage since different stage's monsters' collision logic is under different states based on which stage the monster is supposed to appear in. The finite state machine which shows if the character is hit by the monster or not is called is_attacked. The input for this state machine is from the collision logic module that checks whether or not the monster’s model’s pixel coincides with the main character’s pixel. If it does, it will take a heart away from the main character. While it does so, it will also force the character to enter a state of being attacked, which will cause the character to flash and be invincible for a few moments. If the character’s hearts were to be depleted, the character would die and this would send a signal to the general state machine and the state would transition into the game-over stage. Now, if the user is to give any of the WASD movement keys signal input, the character will begin walking towards whichever direction the user presses (W - upwards, A - downwards, S - left, D - right). This mechanism simply increases or decreases the character pixel position on the screen based on the input. However, if the movement is simply left this way, the character would simply glide across the screen, and that wouldn’t make it appealing, which is why animations are added. To do so, another state machine is utilized. This state machine takes in the input from the user (WASD) and the character’s current state (where the character is facing, whether the character is moving, and whether the character is attacking). For example, if the character is walking forward, and the input is W again, this would cause the character to switch legs as he walks. This is the same for different directions. Also, if there is no input from the user, whichever direction the character is facing, it will continue to face that way and stay/transition into a stand position. Now, if the user inputs an attack signal, the character will attack whichever direction it's facing. Each state for the character has its own asset, and by switching through these states, the images will cycle through creating an animated effect for the character. As for the weapon of the character, the logic behind it is simply that if the user inputs an attack signal, the weapon will spawn in front of the character depending on which way the character is facing. This in a sense is the same as the character attacking in front of him. The weapon also has collision logic with the monster, but this time if the weapon is to hit the monster, the monster module will receive a signal notifying that it is hit by a weapon, and this will cause the monster to die, while also having the being hit animation. Finally, the character has a wall collision logic with the walls in the game. This wall module basically detects whether the character’s next movement will coincide with the wall and sends a collision signal to the character’s module to stop it from going into the wall.

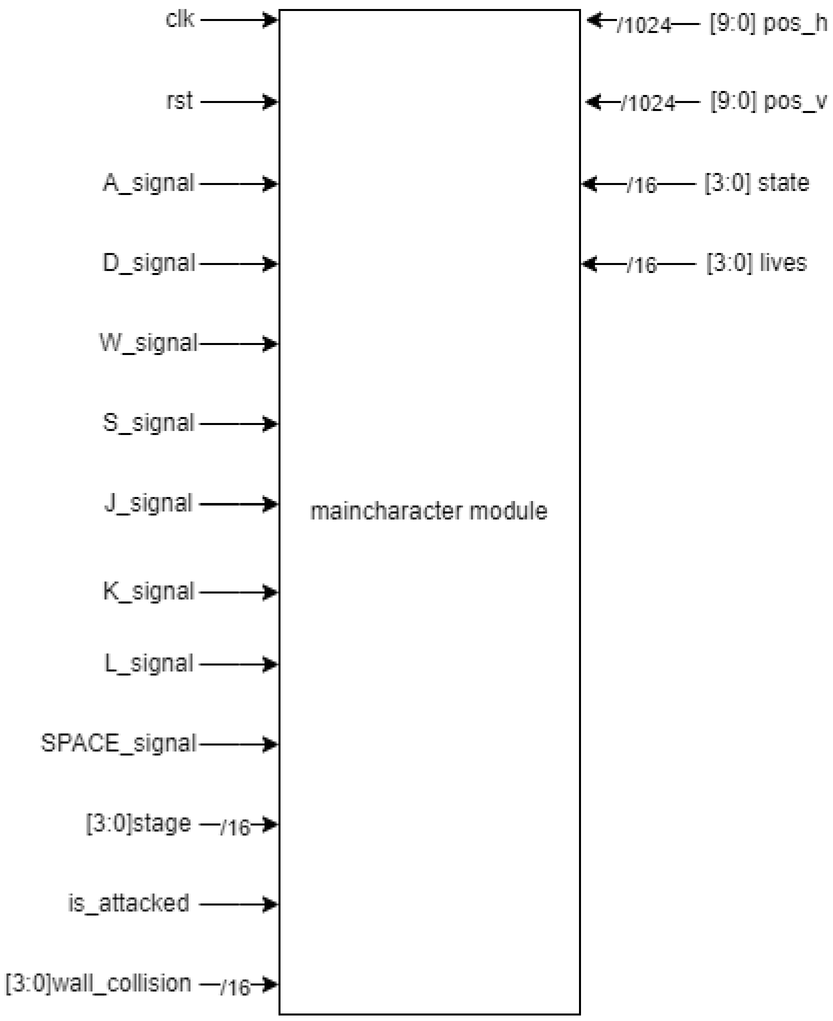

Figure 10 - Block Diagram of Maincharacter Module

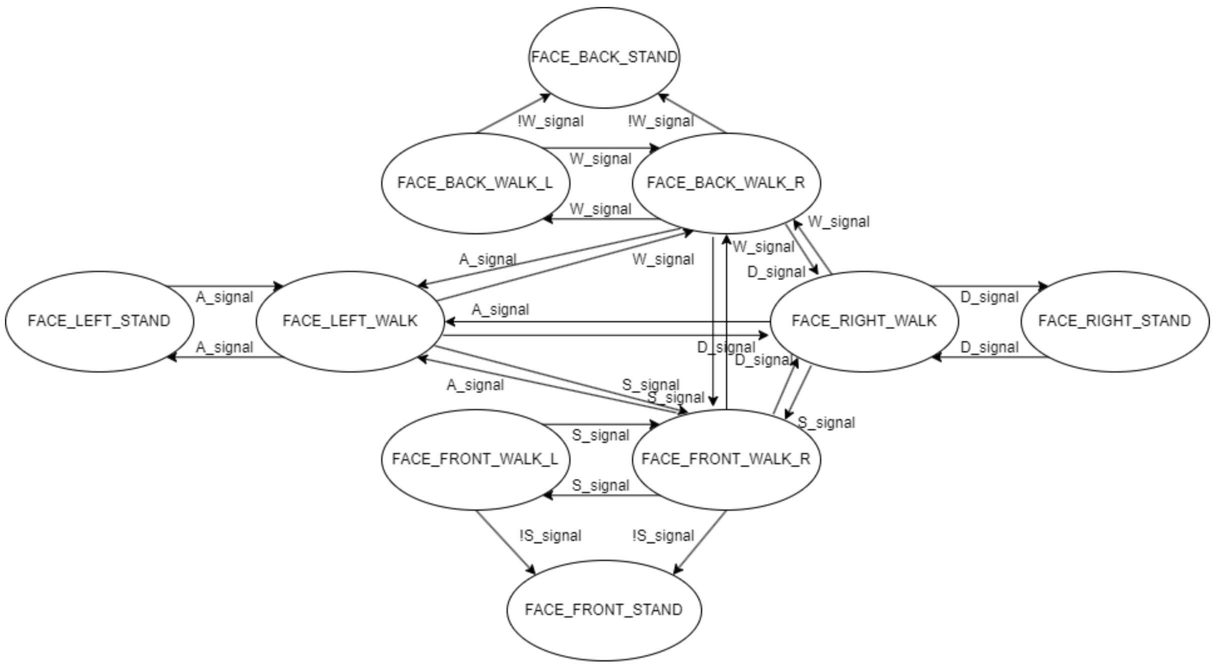

Figure 11 - State Transition Diagram for Main Character Walking Animation

Finally, there are the monster modules. These modules control the behavior of how the monsters interact with other assets in the scene. The movement of the monster is randomized using a linear feedback shift register, since Verilog does not have a random function built-in, thus, it requires a LFSR to simulate the random number generator. However, it is to be noted that after a certain number of generated numbers, the number will begin to repeat itself in the same pattern. However, this certain number is great enough that it can be considered random for our use case. This generated random number is then applied with modulo to obtain a directional value. There are also cases added such that if the monster is to hit the wall, it will start walking in another direction, so that it will not keep walking into the wall. This mechanism is also implemented with a counter, so that the monster will continue walking in one direction until the counter reaches a certain number, then it would change direction, depending on the random number. Next, for the collision physics, it has the same logic as how the main character module works with walls and other assets, so please check the previous paragraph for more information. What differs with the collision with a weapon is that all monsters do not have life values, which means that they die after one hit. There are also different types of monsters with different difficulties. The type of monster is dependent on the current stage state. The difference in their specs is only their speed. The first monster, CS student, has a speed value of 1, i.e. their model moves through the screen 1 pixel per clock cycle. The second and third monsters, EECS student, and NTHU student, have a speed value of 2, i.e. their model moves through the screen 2 pixels per clock cycle. The final monster, Boss student, has a speed value of 4, i.e. their model moves through the screen 4 pixels per clock cycle. It is to be noted also that the monsters' initial position is at the rightmost entrance at the door, where they enter the scene.

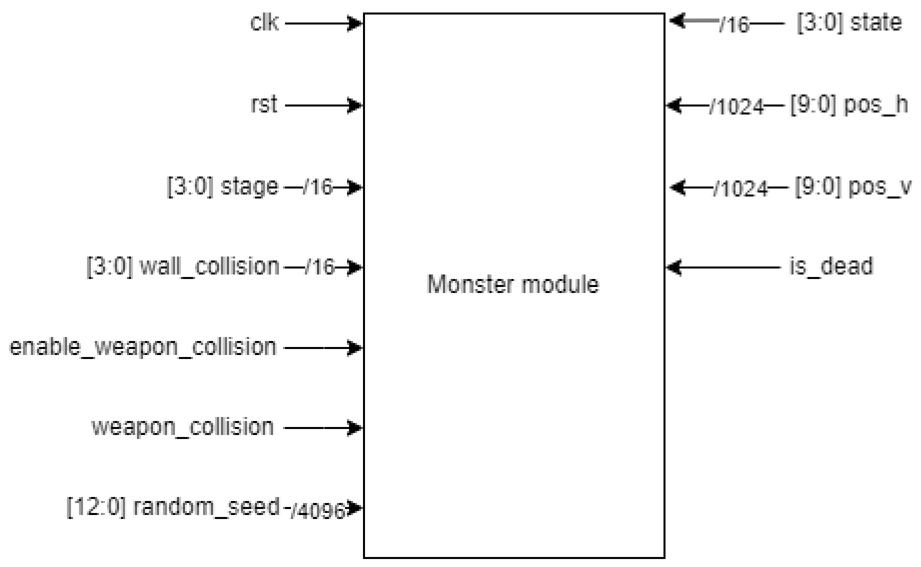

Figure 12 - Block Diagram of Monster Module

Game Results

Game Structure

This section will be divided into three parts, the initial idea, the process of changing, and the final result. There were plenty of changes mainly due to the lack of time in our hands. There were a lot of issues encountered whilst creating the game. These include but are not limited to the lack of memory on the board for all the initial assets, and the lack of LUTs of the board (more details to be mentioned later).

The initial idea of the game is to have a structure similar to the first edition of the Legend of Zelda, where the player will control the main character and go through the game with stages and collect stronger weapons and items along the way until reaching the boss. There were supposed to be ten stages, including a boss fight stage, and a puzzle stage, and the layout of each stage can be seen in figures 13 and 14.

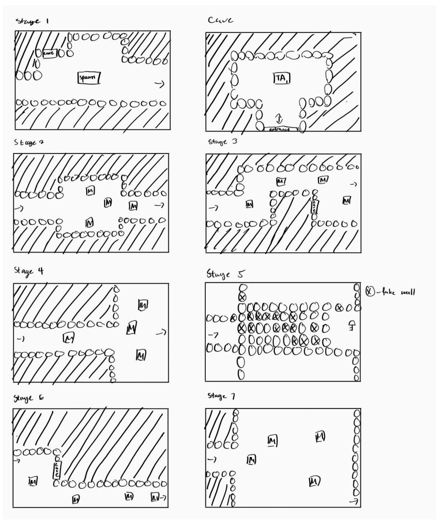

Figure 13 - Stages 1~7 of the Game

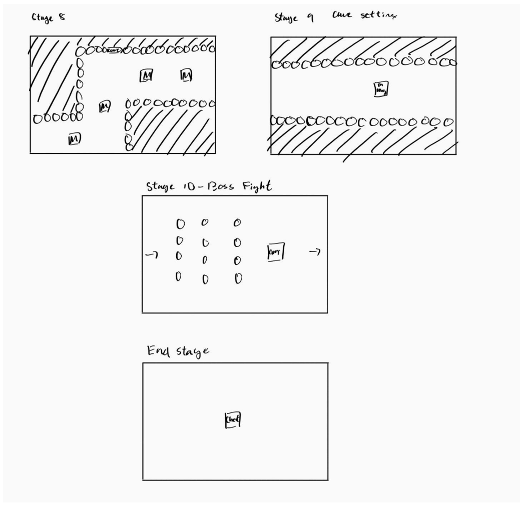

Figure 14 - Stages 8~11 of the Game

As seen in figures xxx and xxx, the player was supposed to be able to go through these stages, defeat monsters, obtain rupees (in-game currency), and purchase stronger items and weapons from the teacher assistants present in the caves (computer classrooms in setting). After defeating the boss at the end, the player would have obtained a trophy for defeating the boss, and the game would've ended there. However, due to the aforementioned issues, there were several changes made.

While the important modules were getting implemented, an issue came to light, which is the lack of LUTs in the Basys 3 board. When the first scene and the interactions between entities were finished, the LUT usage was already at 95% according to the statistics on Vivado. It was later identified that it was due to the block memory generator that is causing this issue. Thus, we decided to make the change to have only one scene and switch the game to a tower defense style, but with the same combat mechanics as before. Later, a more efficient way was discovered to cut down the LUT usage by half. However, it was already too late to revert the change, as there wouldn't be enough time to program the rest of the game. Therefore, it was decided the change would be final.

The new structure basically has monsters coming into the scene through an entrance and the player has to defeat them before reaching a new wave. This will continue on until the ninth wave, which is the boss wave and the final wave. There will be a significantly stronger boss and who will enter the scene, alongside several other monsters. After defeating the boss, the player will be able to obtain the trophy similar to the last design. The game would then end there.

Game Snippet

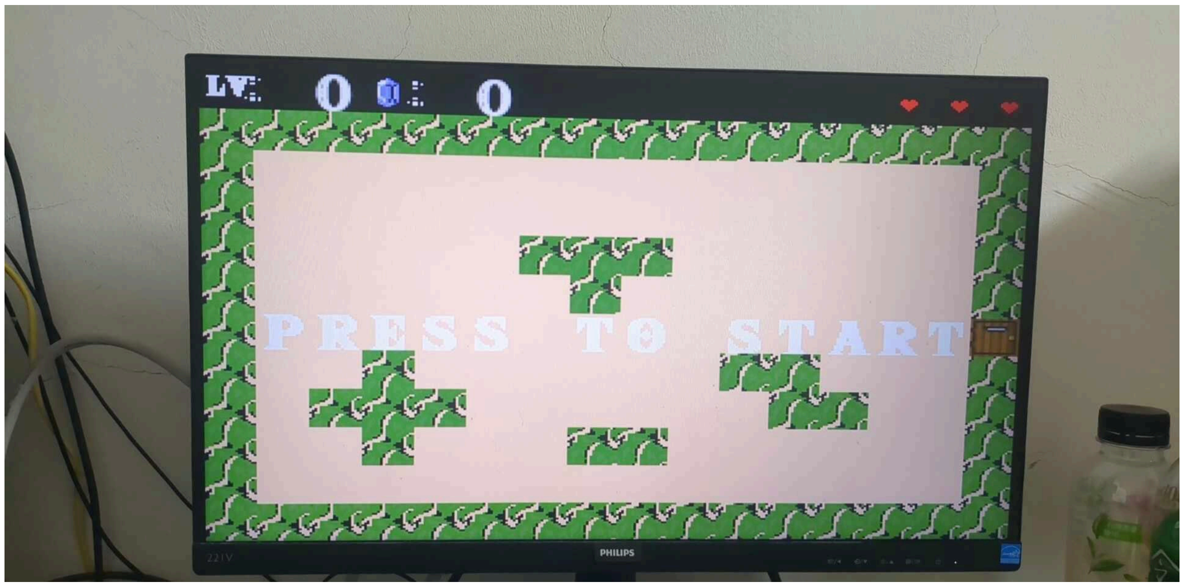

Figure 15 - Press to Start Screen Snippet

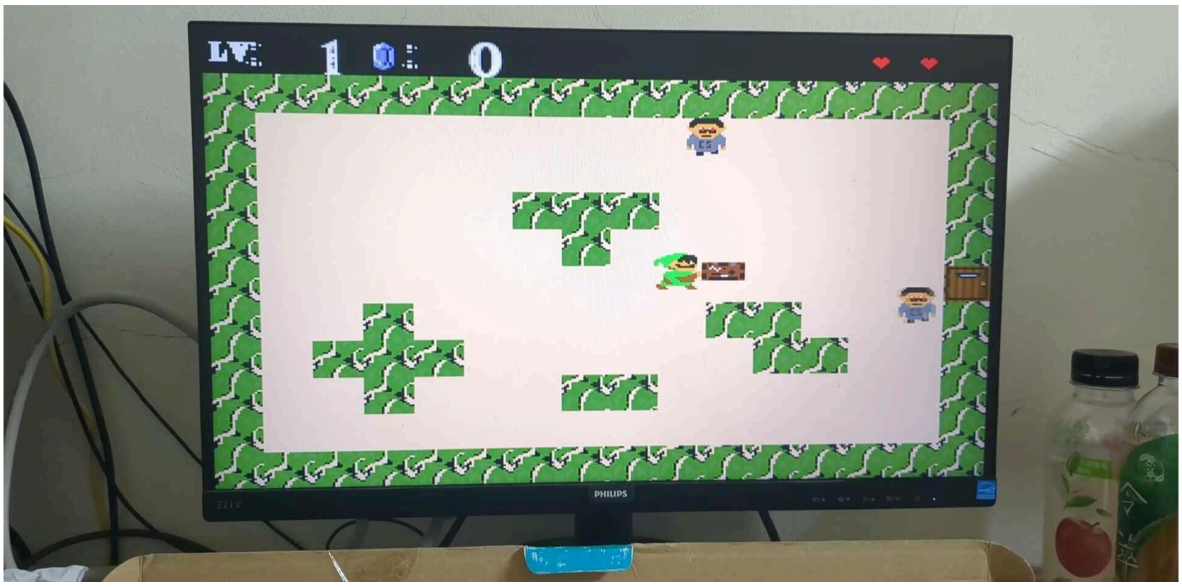

Figure 16 - Level 1 Screen Snippet

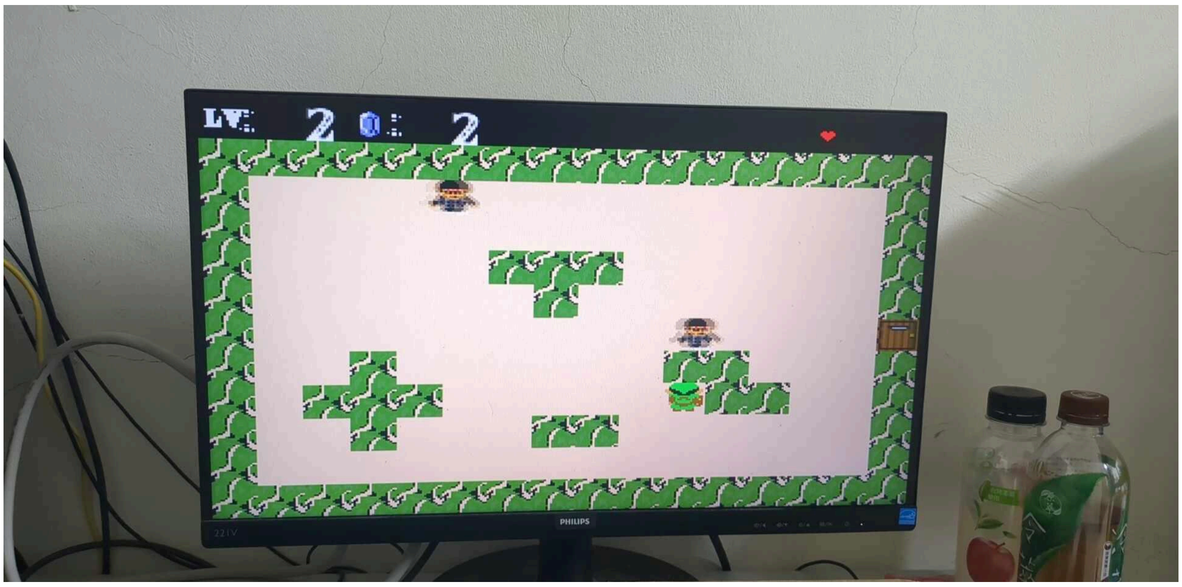

Figure 17 - Level 2 Screen Snippet

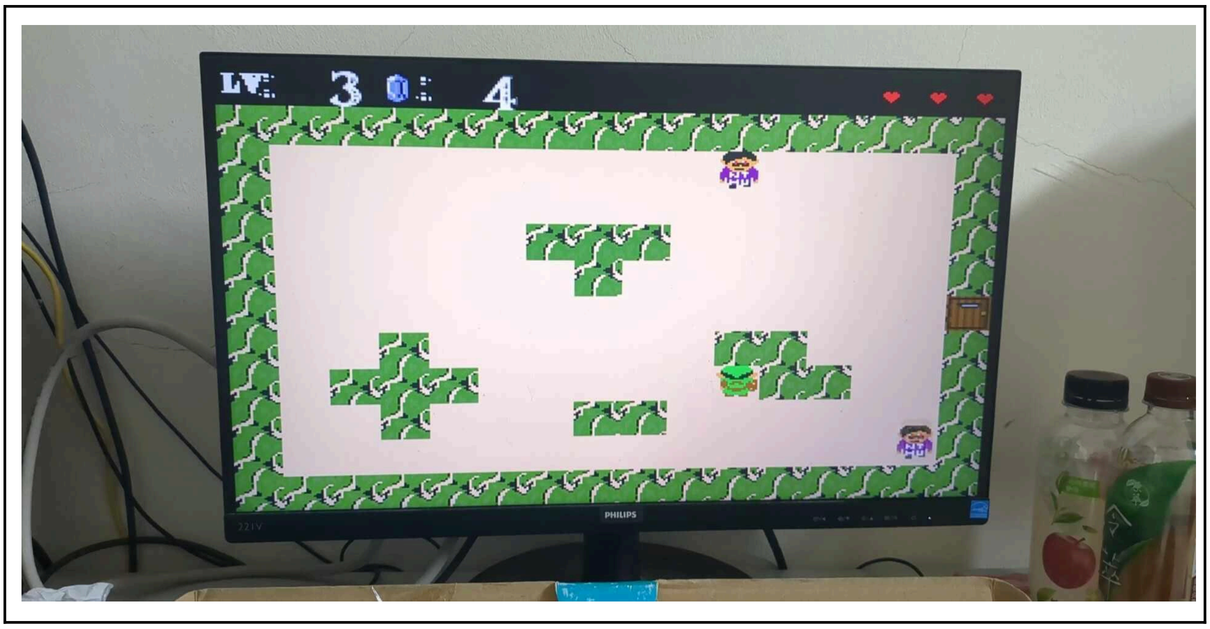

Figure 18 - Level 3 Screen Snippet

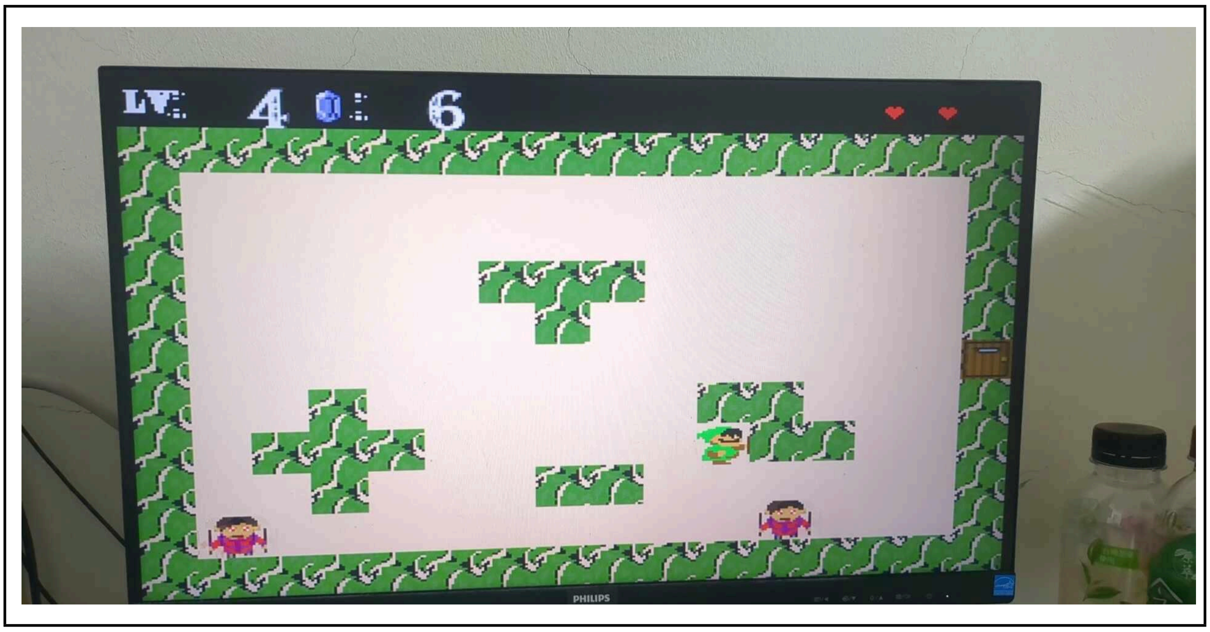

Figure 19 - Level 4 Screen Snippet (Boss)

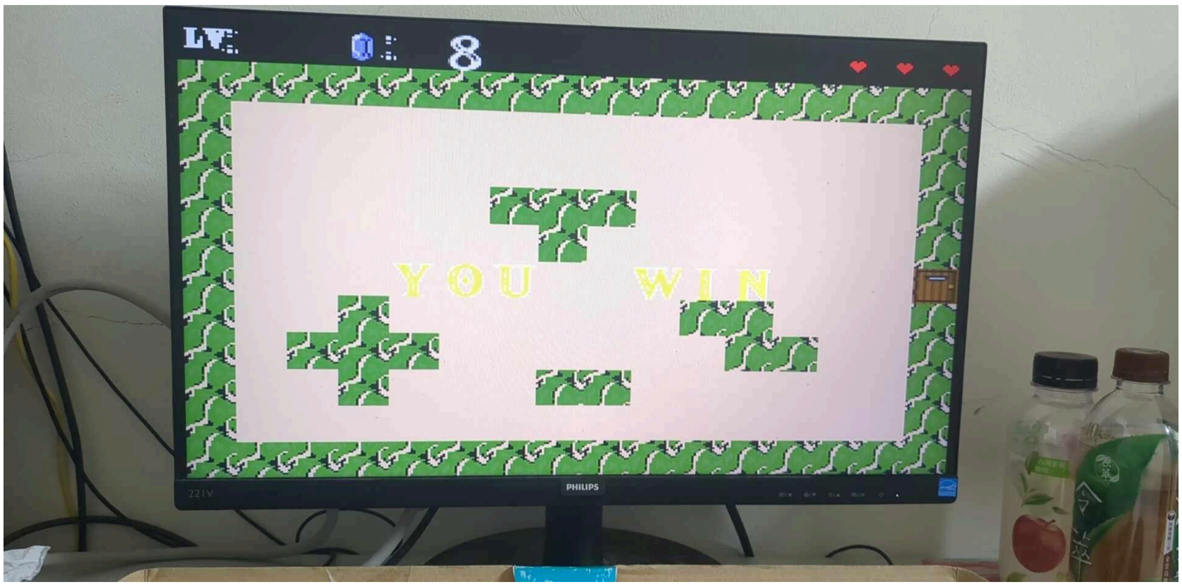

Figure 20 - You Win Screen Snippet

Conclusion

The project was a game inspired by the Legend of Zelda, with some of its elements changed to have an original look. The game utilized plenty of Verilog modules, from hardware protocols to game logic code. Verilog usually isn't the ideal programming language for game development, however, it is still possible to create a game using it, but there will be restrictions and inconveniences. Some of these that we encountered are the capability limit of the FPGA, as well as other things like needing to implement something more complicated in order to display text and other things. This, however, allowed us to learn a lot more about the topic that we are supposed to learn in this course. The output was not necessarily the same with the initial idea that was in the proposal, due to plenty of factors that came along the way, which was discussed in earlier sections of the report. Overall, the project was successful, albeit having some inconveniences and issues that arose midway. The end result was satisfactory but not what we had envisioned in the start of the planning of the project. However, after taking into consideration the limited time we had and the limitations of the hardware, we are able to accept this project as our output. It still remains a fact that it could've been done better. Both of us had learned a lot from this project, from learning more about VGA controllers with FPGA to creating game logic with Verilog. It was quite stressful along the way due to the workload at hand, but it was quite a fulfilling experience that we are glad and thankful to have as a part of our university course.

Delegation of Work

Name	Tasks	Name	Tasks

林楷源	Added ideas for a twist in the project (so that it isn’t just a copy of the original game).	林哲宇	Proposal of the initial idea for the project (take ideas from the Legend of Zelda First Edition).
	Designing of most of the assets present in the game (can be seen in the assets section under System Specification).		Responsible for all VGA aspect of the project (code)(e.g. RGB gen, etc.).
	Written the storyline and interaction lines between Characters.		Fleshed out most of the interaction between entities and other actions of entities
	Provided the code for the main character’s movement animation and the initial structure for monster’s movements		Converted all assets into .coe files and implemented them in the game.
	Created background music using piano sheets from pre-existing OST.		Provided the code for scene switching part of the game
	Written the report for this project		

Table 2 - Delegation of Work of Each Person in This Project

References

Inspired by:

The Legend of Zelda created by Shigeru Miyamoto and Takashi Tezuka, and published by Nintendo

Background for Title Page Banner

Grabrela. (2020). *The Legend of Zelda*. Pixilart. Retrieved from https://www.pixilart.com/art/the-legend-of-zelda-a2504df5a9c1006.