CC4510 Final Project Space Fighter

Blake Wang

30/05/2024

I. AIM AND OBJECTIVES

The aim of this project is to design a classic space fighter game using an FPGA (Field Programmable Gate Array) learning board Terasic Cyclone V 5CSEMA5F31. The primary objective was to enhance learning from subject CC4510 by applying theoretical concepts in a practical, hands-on project.

II. SPECIFICATION

In a classic space fighter game, players control the fighter by shooting down asteroids that fall from the opposite side of the screen. The goal is to protect the base station and keep the fighter alive. In our design, we refer to asteroids as rocks. Table 1 shows the input and output of the game; we control the fighter on the keyboard and output graphics on a VGA monitor. Second, Table 2 and Fig. 1 describe and illustrate the modes of the game; there are win or loss results according to the performance of players. Finally, Table 3 lists the user-visible states that the user will see on the display. These visible states are mostly related to position change or each game element.

TABLE 1: INPUT AND OUTPUT THE SPACE FIGHTER GAME

Name	Direction	Data Type	Width	Description
game_start (Key0)	input	std_logic	1	When true, game start
Reset (sw_0)	input	std_logic	1	When true, reset the game
move_down (Key2)	input	std_logic	1	When true, move the fighter down
move_up (Key3)	input	std_logic	1	When true move the fighter up
red	output	std_logic(7 down to 0)	8	Color of current pixel
green	output	std_logic(7 down to 0)	8	Color of current pixel
blue	output	std_logic(7 down to 0)	8	Color of current pixel
hsync	output	std_logic	1	Horizontal synchronization
vsync	output	std_logic	1	Vertical synchronization

TABLE 2: MODE OF THE GAME

Name	Description
welcome	Only some text displays the name of the game.
instruction	Show player the goal of the game and prompt player to start the game by pressing Key1
game	game in process
game over	when certain condition trigger, end the game. Shows differently according to the player's performance.

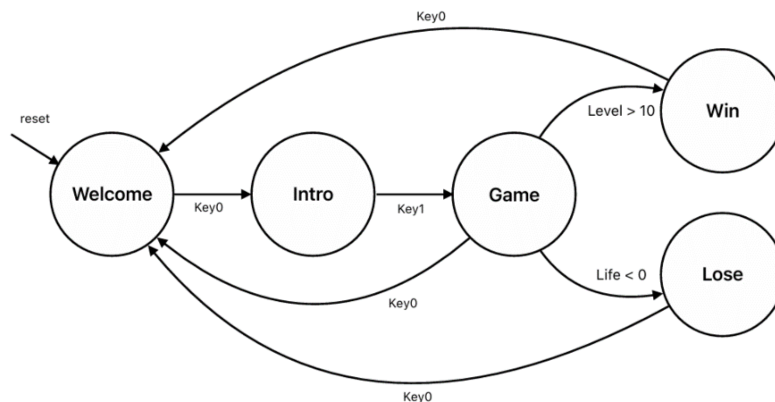


Fig. 1: Game mode controller flow.

TABLE 3: USER-VISIBLE STATE OF THE GAME

Name	Data Type	Data Range	Description
Fighter Y	Integer	0 - 480	y-position of fighter's head
Bullets X	Integer	10 - 630	x-position of bullet
Rock Y	Integer	50 - 430	y-position of rocks
Rock X	Integer	10 - 620	x-position of rocks
Score Bar X	Integer	0 - 100	x-position of score bar
Life Bar X	Integer	0 - 100	x-position of life bar
Base Bar X	Integer	0 - 200	x-position of base bar
Level	Integer	1 - 10	current level of the game

III. SYSTEM BLOCKS DESIGN

A. *Overview*

The game consisted of four main blocks. The connections between each block are shown in Fig. 2. We divided our tasks by blocks, my task focused on building the main control block and the overall connection. In addition, the VGA display block used a reference design from JCU subject resources [1]. This report does not discuss the details of the Data management and VGA blocks. Further information on Data management can be found in the report by my teammate Yuan [2]. In addition, a *25Mhz clock* is run in the overall system, but there are different reference clocks in different objects. Furthermore, our project was quite small, so we retained the graphic design inside each entity. Therefore, in this project, the main control block and graphic management can be considered as a single block. In future design, when the project gets larger, graphic management should be a separate task.

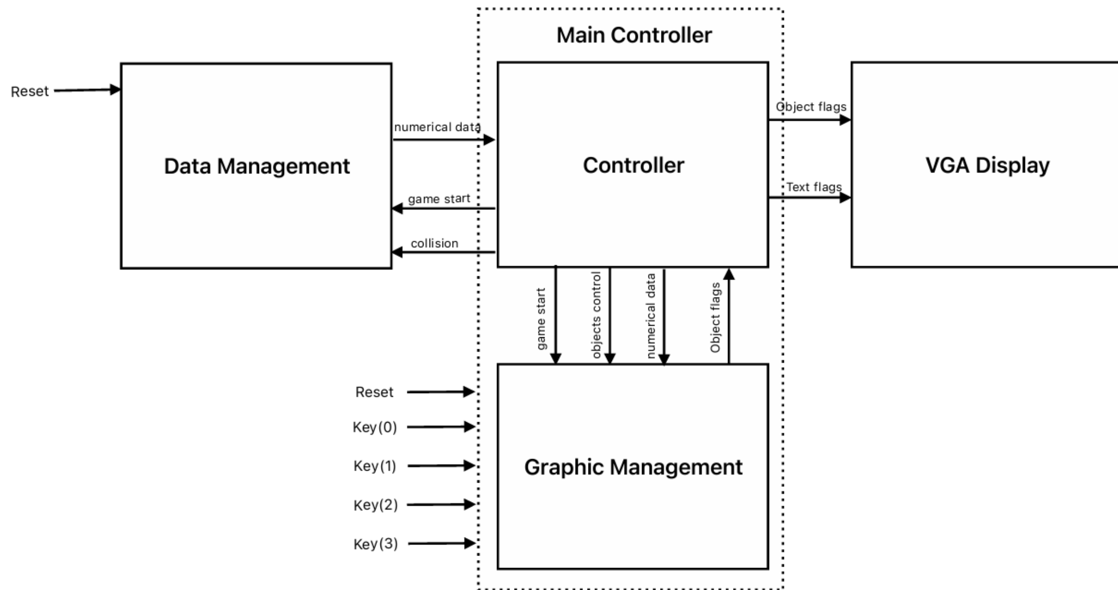


Fig. 2: System Blocks overview.

B. *Main Controller Block Design*

The detailed Design of Controller block is illustrated in Fig. 3. Bullet FSM and Rock FSM specifically draw a bullet and a rock on the screen. Shooting FSM is the main component of the Top Level, and it generates multiple bullets from the Bullet FSM. The Rock generation FSM also generates rocks from Rock FSM. Both connect to the Collision FSM by sending the *bullet_flag* and *rock_flag* signals. If flags overlap, it sends hit signals back to each FSM. All components start working only if *game_s* is true, and all of them are connected to a reset signal and include a reset state. We also built a reference clock for each object, as listed in Table 4.

TABLE 4: REFERENCE CLOCK OF EACH OBJECTS

Name	Cycle of system clock	Description
Fighter	50000	Constant
Bullet	30000 ~ 70000	Change according to level of game
Rock	50000 ~ 120000	Change according to level of game
Collision	1	Same as system clock

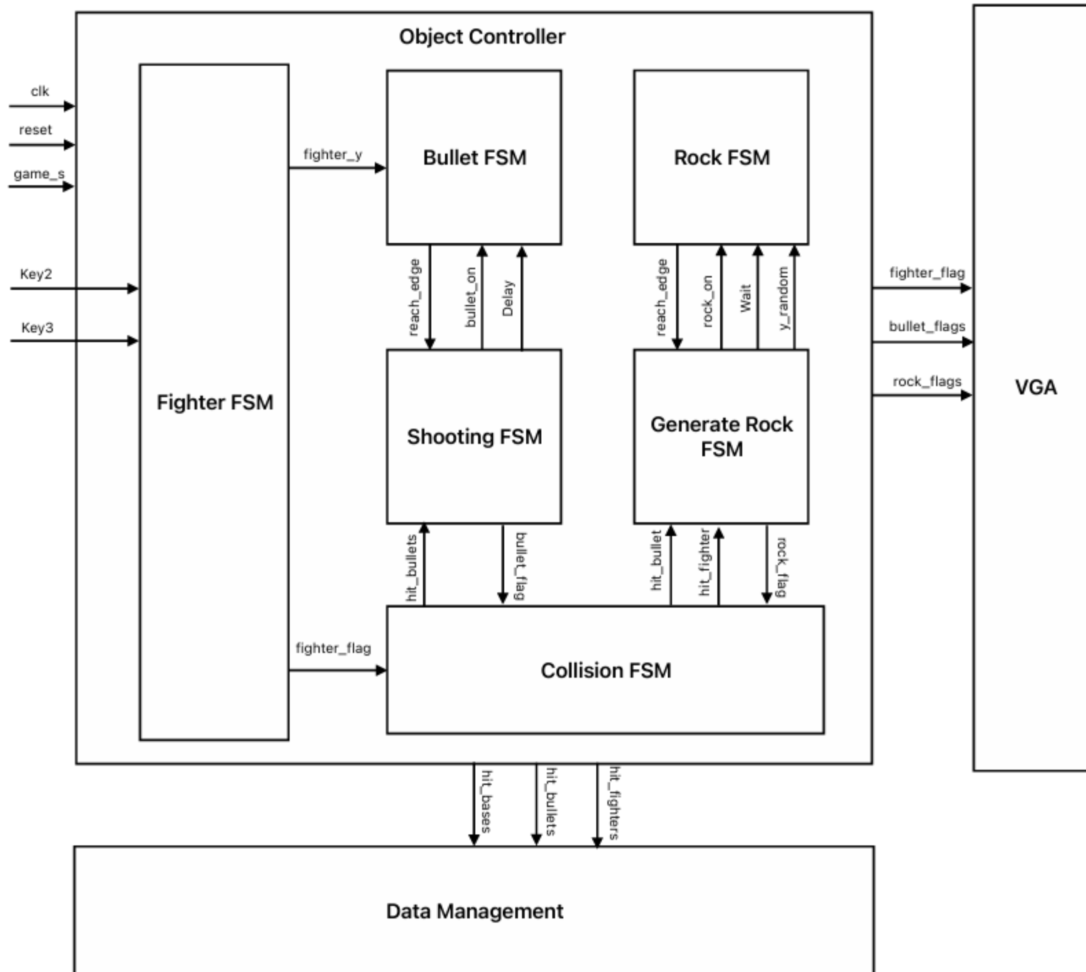


Fig. 3: Detailed design of the Object Controller block.

C. *Object Blocks Design*

1) Fighter FSM

The state diagram of the Fighter FSM is shown in Fig. 4. Each time a signal is received from the buttons, the fighter moves up or down, and we built a reference clock between each detection on the button is 50000 system clock cycles, as we adopted a 25 MHz clock from the reference design at the top level. Therefore, each detection is 0.02s, this is relatively enough for us to play the game smoothly. In addition, the fighter will continue outputting a buffer signal which is the y position of the fighter. This signal is sent to the bullet FSM.

2) Bullet FSM

A state diagram of the Bullet FSM is shown in Fig. 5. It controls bullet movement and draws a bullet on the screen. The bullet always starts from the y position of the fighter, and we set the bullet speed dynamic by adjusting the reference clock cycle. The original speed was 2.4ms per movement, and every level increases 0.16ms per movement. In addition, we open the speed and initial delay port map so that we can adjust this value in the Shooting FSM.

3) Shooting FSM

A state diagram of Shooting FSM is shown in Fig. 6. This component is one of the main components at the top level and directly controls the on and off the bullet on the screen according to the collision detection signal between the bullet and rock. In addition, we introduce a delay state because we want each bullet to shoot from different times so that we can continuously observe the bullet shooting out, and there are four bullets on the screen, so we set a delay counter of 0, 160, 320, and 480 for each bullet accordingly. This means that the first cycle of each bullet must wait for the counter to finish counting before the bullet is turned on.

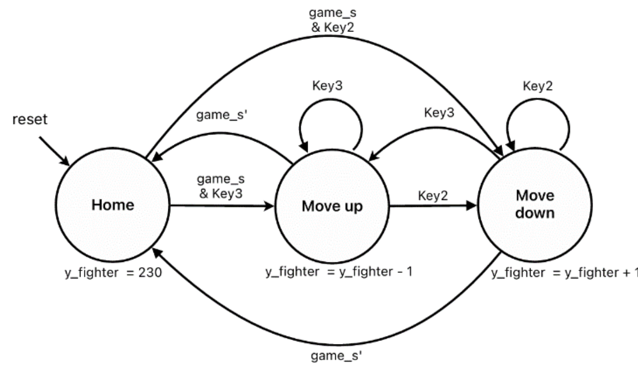


Fig. 4: Fighter FSM state diagram.

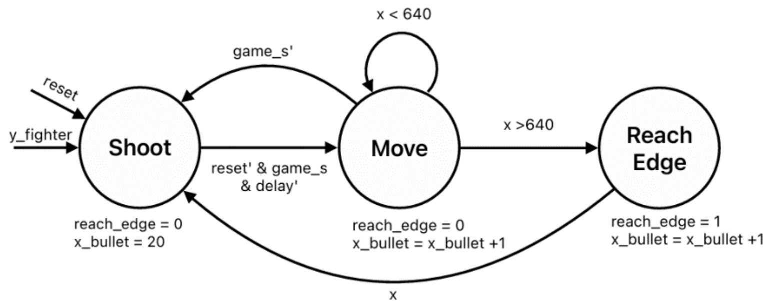


Fig. 5: Bullet FSM state diagram.

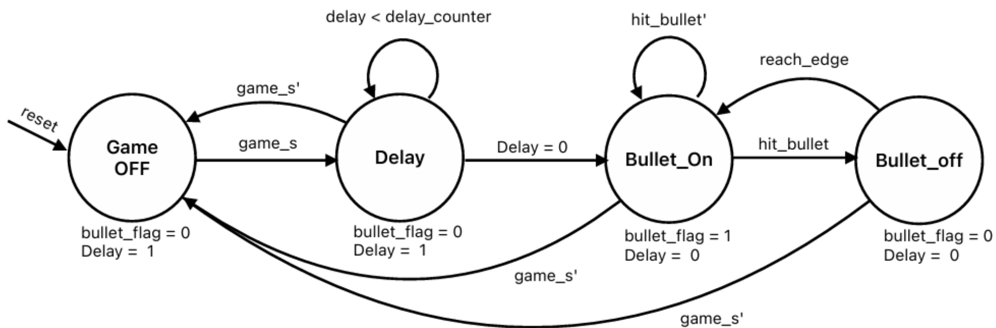


Fig. 6: Shooting FSM state diagram.

4) Rock FSM

A state diagram of the Rock FSM is shown in Fig. 7. This component draws a rock and keeps it continuously moving from the right to the left side of the screen. The rock starts from the right side of the screen again once it reaches the edge of the left. We open the speed and generate the y-position port map, so we can change this value in the Generate Rock FSM which inputs a *random_y* signal to the Rock FSM.

5) Generate Rock FSM

State diagram of Generate Rock FSM is shown in Fig. 8. This component is also one of the main components at the top level, which directly controls the on and off the rock on the screen according to collision detection signals. We set a wait state because we did not want the first cycle of rock to show, given the player times to prepare.

6) Collision Detection FSM

A state diagram of collision detection is shown in Fig. 9. The collision has three different signals: one signal sent to the life counter, one signal sent to the counter, and one signal sent to the base counter. After every detection, it always returns to the idle state for the next detection. Each detection is a system clock cycle.

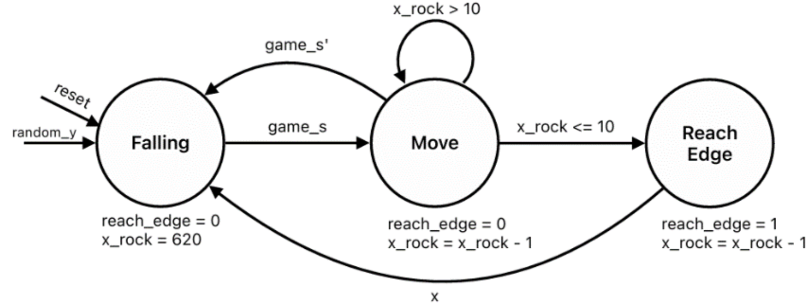


Fig. 7: Rock RSM state diagram.

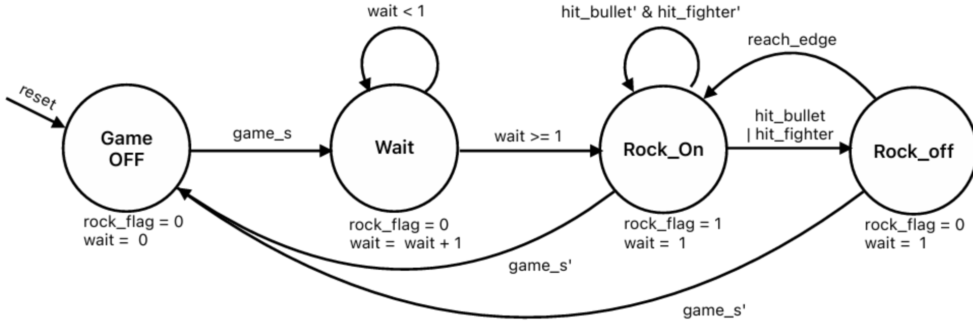


Fig. 8: Generate Rock FSM state diagram.

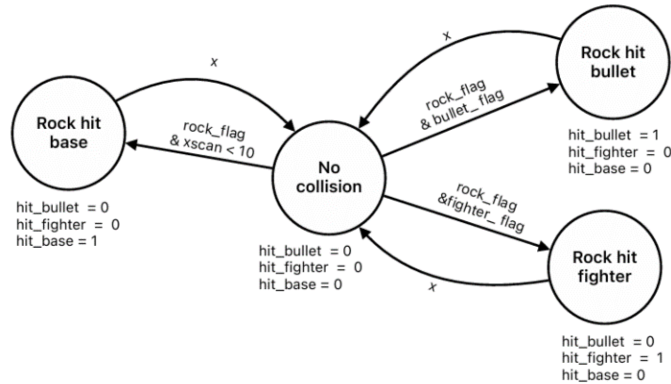


Fig. 9: Collision detection FSM state diagram.

IV. RESULT AND DISCUSSION

A. Multiple Rock and Bullet generation

We built the bullet and rock based on the Pong game reference design. However, if we want multiple rocks and bullets on the screen, multiple entities must be created. To achieve this, we built two main entities: shooting and rock generation. Bullet and rock entities were set as components in these entities. Thus, we will be able to create as many bullets and rocks on the screen as possible by adding more entities in shooting and generating rock entitles. In addition, to prevent bullet overlapping, we created a delay variable that was input from the shooting entity. Given a screen size of 640 pixels and four bullets, we only needed to add 160 movements between each bullet to ensure that all bullets appear on the screen simultaneously.

B. Collision detection method

Initially, we calculated the detection points using x and y locations. However, we discovered that this approach would make development more complex when dealing with rocks and fighters of various sizes and shapes. Consequently, we opted for a pixel overlapping method. As the FPGA board constantly scans all pixels, we only need to determine whether the object pixel signal overlaps. By incorporating this mechanism, we can avoid the challenges associated with changes in the shape and size of rocks, which will be advantageous for future developments.

C. Random rock generation

We use *yscan* as our random generation of the rock, as each rock has a different speed, so when reaching the edge, *yscan* will have a different value. We used this value to reset the rock location in y. However, this is not very random in the process; once the game starts, all rock locations have already been calculated which makes it easy to predict the next rock location. Therefore, we introduce more uncertainty into the game; that is, if the rock is hit, it is generated before it reaches the edge when it is below a certain point near the flight. This outer uncertainty makes new locations difficult to predict. Thus, the game is more fun.

D. Timing Analysis

In our game, the moving speeds of rocks and bullets are controlled by introducing a reference clock. Each entity has a different reference value, which causes a delay in the collision detection. We encountered issues with scores and life counting. These values decreased at unexpected speeds. We assumed this was the timing issue, as the bullet moves faster than the rock, so when hit, there might have been more cycles on detection, causing multiple decreases before the rock was turned off, as shown in Fig. 10. However, we keep it as it is because in physical thinking, speed can relate to the amount of damage. Therefore, this makes the game more enjoyable. However, future work should investigate the timing issue to gain more control over the development of the game.

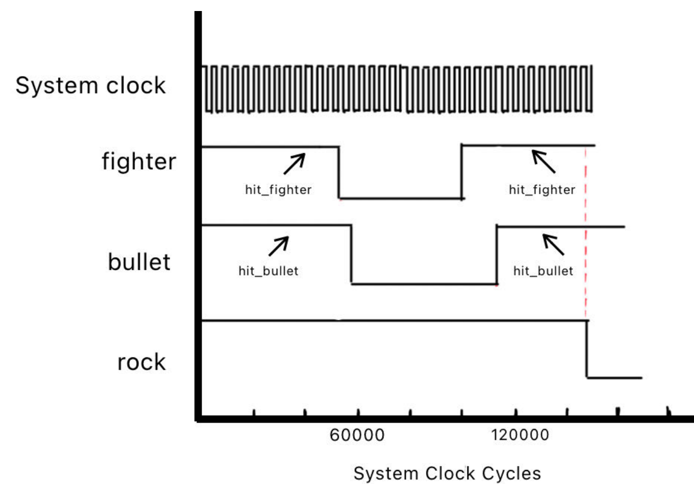


Fig. 10: Timing analysis for collision detection.

E. Debugging

We assigned all top-level entities with an enable signal, and during the debugging process, we could test one entity each time and close all others. Several issues were encountered, as listed below:

1) Collision detection

During the development process, issues related to collision detection were encountered. Initially, we used a two-bit signal for different collision detections because there are multiple possibilities for detection. However, this approach failed to output the expected signal, likely because of programming or timing issues. To expedite the process, we separated different collisions into 1 bit signal each, which resolved the issue, although some redundant code was introduced. Future developments should aim to reduce this redundancy.

2) Missing Conditions

We encountered an issue where the screen would turn off when detecting a collision, we found that we did not include 'else' conditions in some drawing processes. This oversight has led the system to an undefined state. Therefore, ensuring all possible conditions are considered is crucial.

3) Button Signal Handling

Initially, we allowed the use of buttons to start the game, but this caused the second page to be skipped because we did not add a delay clock between each detection. To address this issue, we used only one button to enter the next mode, and the next mode did not refer to this button. However, in the next development process, we can introduce a proper reference clock to the mode change FSM.

V. Conclusion

In this study, we successfully designed and implemented a classic space fighter game using an FPGA learning board. The design process involves specifying game inputs and outputs, defining game modes, and outlining user-visible states. We constructed the system using several key blocks including the main controller, data management, and VGA display blocks. Detailed FSM diagrams have been developed for various game elements, such as fighters, bullets, rocks, and collision detection.

Our results demonstrate the successful generation and control of multiple rocks and bullets, employing methods such as pixel overlapping for collision detection and introducing pseudo-randomness in rock generation. During debugging, several issues related to collision detection, signal handling, and condition management were resolved. However, there are still some bugs in the game, such as timing issues with scores and life counting. It is time consuming during the debugging process, and a better debugging method might be applied in the next development cycle.

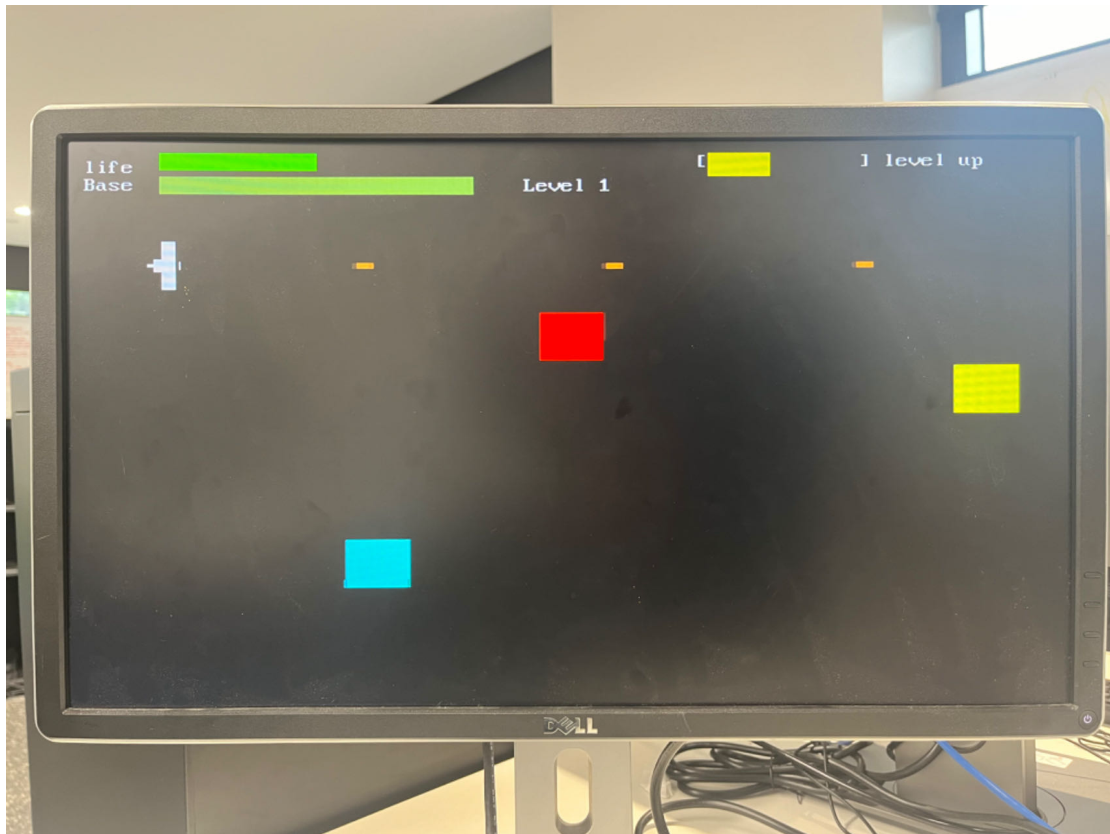
In conclusion, through this project, we have an overall understanding of the VHDL coding structure and digital system design. Future work could focus on refining timing controls, reducing code redundancy, and enhancing the randomness of game elements to improve gameplay and robustness further.

Reference

[1] M. Small, "CC4510 Design Project- Pong Game," James Cook University, 2023.

[2] H. Yuan, "CC4510 Final Project Space Fighter " James Cook University, 2024.

Appendix 1: An Image of the Space Fighter Game



APPENDIX 2: TOP LEVEL ALL CONNECTION LAYOUT