Habib University

Dhanani School of Science and Engineering

Digital Logic and Design
EE/CS 172/130

T2

Final Project Report

Space Invaders

Muhammad Bilal Qureshi (08079), Muhammad Saad (08063), Syed Muhammad Ghufran Alvi (08235), Abdullah Amir (08453), Ehzem Sheikh (08518)

Professor Areeba Aziz Rajput

May 11th 2023

Abstract

Space Invaders is a classic shooter game. The game would allow players to control a spaceship and shoot down waves of enemy aliens. Targeted at retro gamers and hobbyists, the game would be designed to be both challenging and entertaining.

The game gives the player an option to either use the joystick or a slider working with an ultrasonic sensor, as an input device to control the spaceship in two directions (left and right) trying to eliminate a wave of aliens by shooting at them, before they either collide with the rocket or reaches the bottom surface, as this will reduce the lives that will be displayed at top right and killing the enemies will increase the score displayed on top left.

Table of Contents:

1. Introduction

The motivation of this game came from our childhood legendary classic retro games “chicken invader” and “star defender”.

Fig 1 : Chicken Invaders*

Fig 2 : Star Defender*

Our project is a single-player shooting game where the player uses the joystick to move the already shooting spaceship to eliminate waves of enemies coming towards the spaceship. The player will have 4 lives which will decrease whenever an enemy collides. There will also be a score counter which will be incremented whenever an enemy is eliminated.

2. Implementation

We will have three main components for this project: input, control unit and the game display (imaging block). The components are described in the following sections.

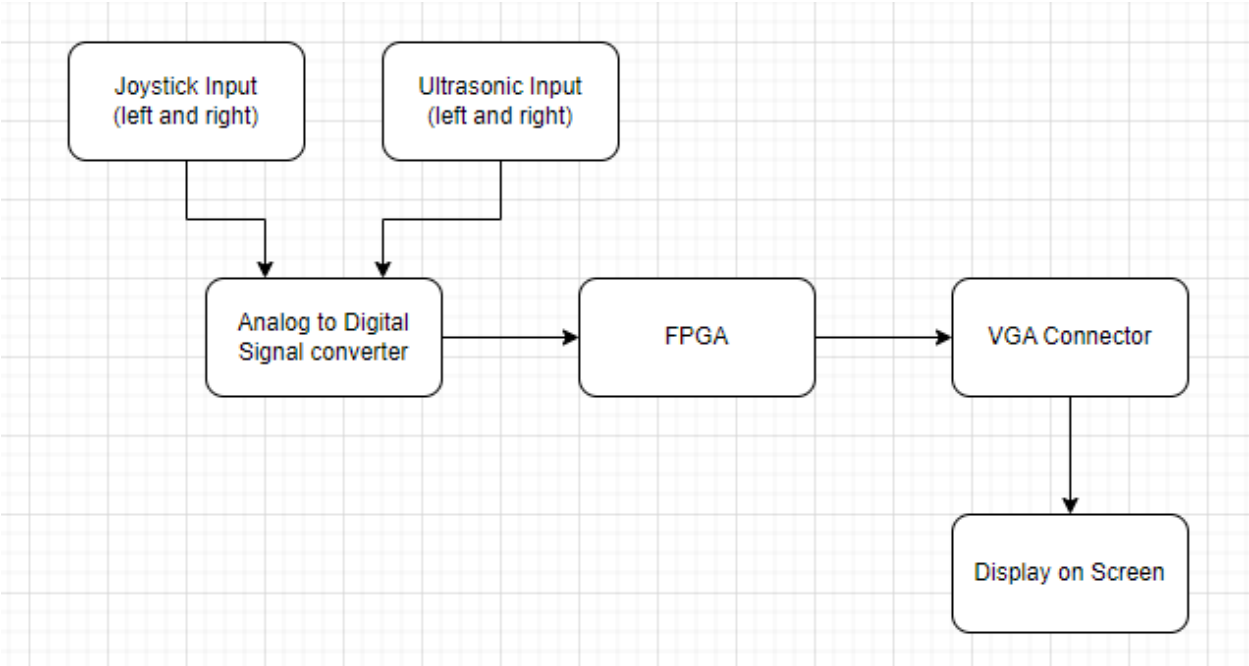

Fig 3 : Game Design Overview

2.1. Input Block

The user will interact with the game using a joystick, or a slider that is working with an ultrasonic sensor, which will serve as our source of input. The movement of the joystick/ slider will be translated on the display screen as the movements for the spaceship, but it will only be horizontal i.e. in left and right directions.
Moving the joystick/ slider to the left will move the spaceship to the left, and moving the joystick/ slider to right will move the spaceship to right.

2.1.1. Joystick

The joystick is interfaced with the BASYS 3 built in XADC as it BAYSYS 3 FPGA provides internal Analog to digital converter that can be used to read the analog input signals of the joystick manually

We need to limit the voltage signal given to FPGA from the joystick between 0 - 0.76 volts, as XDAC is capable of reading signals between 0 - 1 volts, so we are using a voltage divider by connecting a series combination of 15 kilo ohms and 680 ohms resistors. [1]

The connection between the joystick and the XADC PMOD is given below:

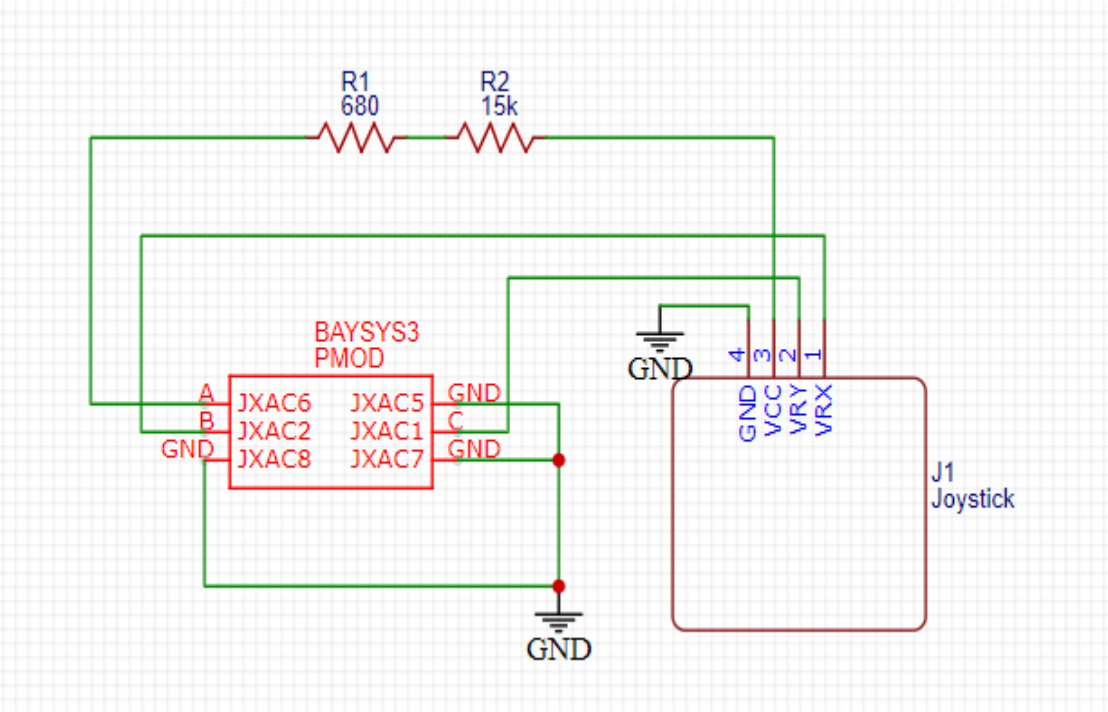

Fig 3: Circuit Diagram for connecting the Joystick to Basys3 Analog Pmod port pinout

2.1.2.Ultrasonic Sensor

Ultrasonic sensor was operated using an arduino. First a pulse was created for the trigger so that it can send a burst of noise to be accepted by the echo pin. Two outputs from the Arduino are being fed to the FPGA as input. We are passing them in binary, meaning there are 2 possibilities for each of them (either being high or low) which gives a total of 4 different combinations. In the first combination when both the outputs are low, we keep the rocket stable at its place, when first output is high and second is low we move the rocket to left, when first output is low and second output is high we move the rocket to the left. Finally there's also a

fourth combination of both inputs being high, but we are not considering it so it is our don’t care condition.

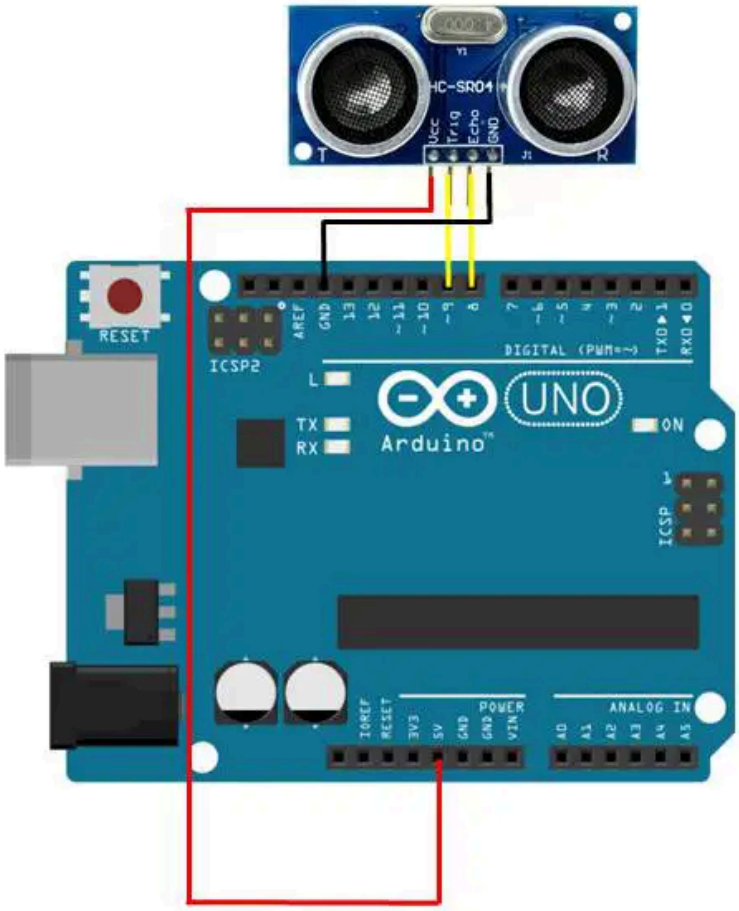

Fig 5: Arduino and Ultrasonic Sensor Connections *

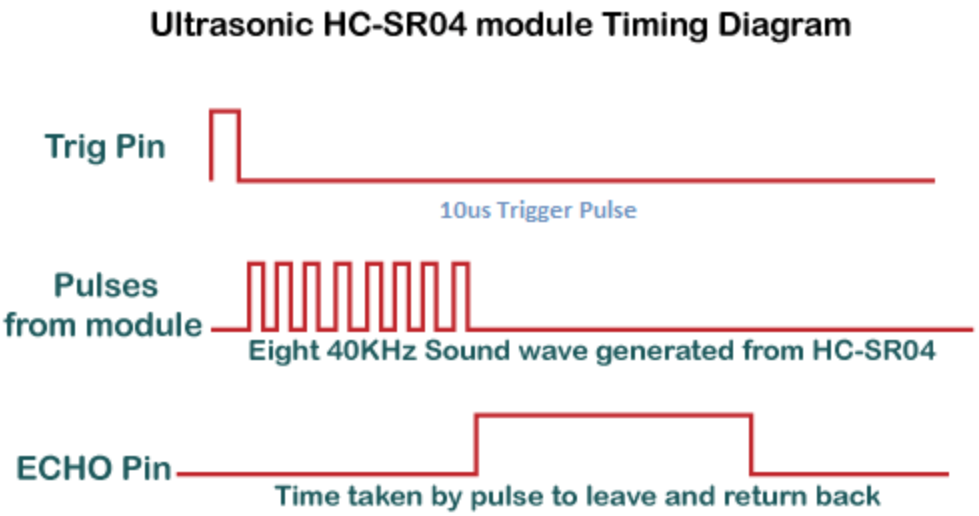

Fig 6 : Waves generated by ultrasonic sensor [3]

2.2. Control Block

The game consists of three main states which are represented by three different screen displays:

1. Start Screen
2. Game Screen
3. End Screen

2.2.1. Start Screen

This screen appears as the first screen of the game. It displays the message “Start” .

Fig 7: Game Start Screen

The module controlling the start screen is **StartSprite** which outputs as reg “StartSpriteOn”. When this StartSprite is high, the **StartScreenRom** reads the .mem file and displays the corresponding colors for the pixels of the 640x480 screen.

State Transition: The start screen appears when the switch “SW15” on the FPGA is pushed up. The player is supposed to push the switch down which will make the transition to the main game screen.

2.2.2. Game Screen

Fig 8: Game Screen

This is the screen where the whole gameplay is based. It is dependent on the **Top** Module consisting of **4** main modules: **vga640x480, RocketSprite, AlienSprites, and Bullet Sprites**.

The *Top* module takes in a 100 MHz onboard clock and a reset signal. It outputs VGA horizontal sync and vertical sync signals, 4-bit VGA Red, Green, and Blue color values, and receives input signals for left and right for rocket control. The module instantiates three sub-modules: **vga640x480** for display timing and resolution, **RocketSprite** for the player sprite display, and **AlienSprites** for the alien sprites display. It also loads a color palette from a memory module, which is used to set the background color and color

values for the sprites. The top module assigns the color values based on which sprite is active and draws the sprite pixels in the active area of the screen using the VGA color values. The color palette is being read from the pal24bit.mem file which holds all 192 x 8 bit values(64 colors each with a Red, Green and Blue hex value).

i. VGA Module:

The *VGA* module defines the timings for a 640x480 pixel resolution display at 60Hz with a pixel clock of 25MHz, and provides signals for horizontal sync, vertical sync, active pixel drawing, current pixel position, and pixel clock. **o_active** is video_on, **H_SCAN** is the h_counter, **V_SCAN** is the v_counter while **o_x** and and **o_y** are the pixel x and pixel y positions respectively. Separate modules for h_count, v_count, and clk_div aren't used so that it is easier to debug the code. **pix_clk** is the reduced clock with frequency of 25MHz. However, it is implemented differently to allow for better accuracy. The 16 bit register counter1 (incremented by 4000 hex each CLK pulse) is added to **pix_clk** each CLK pulse. This method **{pix_clk, counter}** is known as a Verilog concatenation operator.

ii. RocketSprites Module:

The *RocketSprites* module is responsible for displaying the rocket. The code defines the rocket's starting position, size, and movement based on input from the joystick. The module also interfaces with a **RocketRom** module to retrieve pixel values for the rocket, which are output as an 8-bit value in **dataout**. Finally, the module checks if the current pixel being drawn is within the rocket's character boundaries, and if so, sets **RSpriteOn** to 1 and updates the **address** variable to retrieve the next pixel value from the **RocketRom** module.

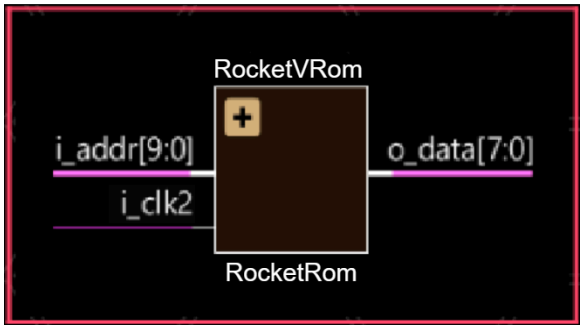

Fig 8 :RocketRom Module

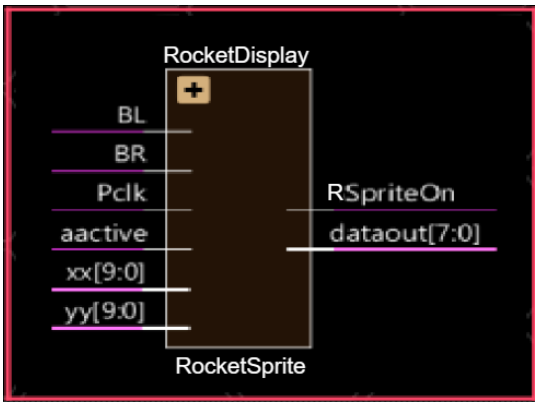

Fig 9 : RocketSprite Module

iii. AlienSprites Module:

The *AlienSprites* module is responsible for displaying alien sprites on the screen. The module takes in current x and y positions, and an active signal which is high during active pixel drawing. The module also contains code to instantiate the **Alien1Rom**, **Alien2Rom**, and **Alien3Rom** modules that load the pixel values for each of the alien sprites from memory. It outputs three signals that determine whether each of the three alien sprites should be on or off, and three eight-bit pixel values for each of the three aliens. The module also sets up the character positions and sizes of the aliens, and uses counters to check whether the current position is within the confines of the aliens. It then updates the appropriate signals and pixel values for each alien based on whether the current position is within that alien's bounds.

Collision is done in the same module, when alien sprite is on and at the same instant bullet sprite (described below) is also on at the same screen position, then alien is sent back to its starting position so it can be respawned.

Lives are also being calculated in the same module, initially the lives are set to 4, which is demonstrated by 3 hearts at the top right, the fourth life being the one where there are no hearts left (last life). Similarly, score is represented by four digits, initially 0, on the top right corner, with first two for high score and last two digits for the current score. With each alien destroyed, it is incremented by either 1 or 2. When an alien crosses the bottom line or collides with the rocket, the lives are reduced by 1 and one of the hearts is removed.

iv. Bullet Sprite Module

The *BulletSprite* module is responsible for displaying the bullet. The code defines the bullet's starting position which is right above the rocket height, its size, and movement based on the rocket's movement. The module also interfaces with a **BulletRom** module to retrieve pixel values for the bullet, which are output as an 8-bit value in **dataout**. Finally, the module checks if the current pixel being drawn is within the bullet's character boundaries, and if so, sets **BSpriteOn** to 1 and updates the **address** variable to retrieve the next pixel value from the **BulletRom** module.

We kept the starting position as startX as X position and startY as Y position, when we move the joystick the value of startX is updated according to the movement and when bullet reaches the top of the screen, it again respawns at at startX which is also the X position of the rocket at that moment. The speed of the bullet has been adjusted accordingly.

State Transition: When the Lives count is reduced to 0, then the game is moved to the end screen. On pressing the reset button, lives are again set to 4 and the score is set to 0. Thus the game returns to its original state.

2.2.3. End Screen:

The end screen will be displayed once the lives are set to 0 from 4. The end screen will display the message of Game Over.

Fig 10: Game End Screen

The module controlling the end screen is **EndSprite** which outputs as reg "EndSpriteOn". When this EndSprite is high, the **EndScreenRom** reads the .mem file and displays the corresponding colors for the pixels of the 640x480 screen.

State Transition: The reset button is supposed to reset the game when the end screen is reached by taking it back to the game screen. But we are turning "SW15" back on to make sure that resetting it leads us to the Start Screen.

2.3. Display Screen (Output Block)

The display screen is generated using the VGA (video graphic array) connector. The display which we are using is standard 640 x 480 pixels.

For imaging on the horizontal axis, we turn the video on from 0 till 639 pixels and for the vertical axis the video is on from 0 till 479 pixels as that is the main display, after that the video is turned off for borders and retracing.

To make the final screen we first used a clock divider to reduce the frequency from 100 MHz to 25 MHz for FPGA to work, then we used h counter to count the pixels of horizontal axis and counter for vertical axis. Then VGA sync was used for turning the video on on the required part of the screen and finally pixel gen is used to create our desired screen.

Following pin configuration is used to connect the VGA connector to the FPGA board.

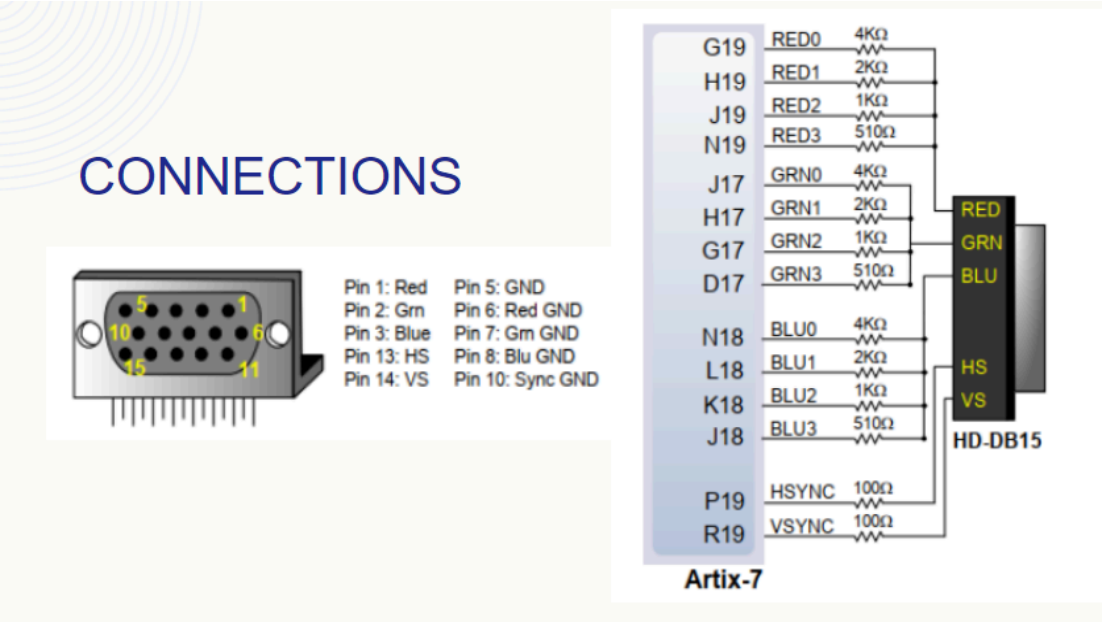

Fig 11: VGA Connections and RGB Ports*

4. User Flow Diagram

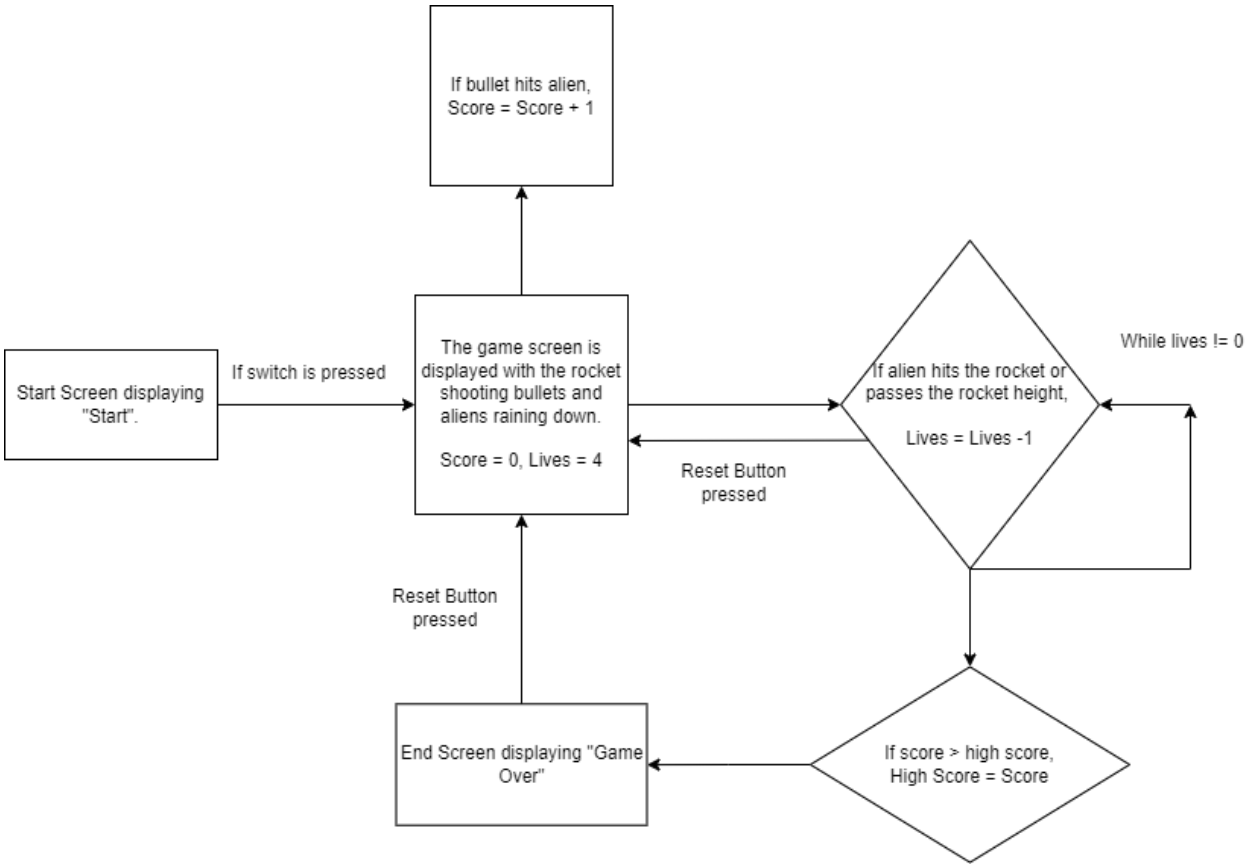

Fig 12: User Flow Diagram

The game starts with the start screen and an option of selecting the input, when the user turns SW15 on, it moves to the main screen of the game. If the bullet collides with an enemy the score is updated as score = score + 1 and if the alien passes the end line then lives are decreased by one, if live comes to 0 the game moves to the end screen. To play the game again, the user has to press the reset button.

5. FSM

Although the start and end states of the game, dictated by the start game and end game screens, take the switch and reset button as input for state transitions, our main game itself does not make any state changes based on any kind of input from the user and instead terminates automatically when the lives count decreases to zero moving to the “Game Over” screen. This transition to the end state can be seen as solely dependent on the gameplay state i.e. the next state depends on the initial state. So we can conclude that the FSM implemented in our game is a **Moore Machine** as our next state of our game is not dependent on the input block.

6. Major Challenges Faced

The major challenges faced in making of the project were as follows:

i) **Implementation of the joystick**, especially integrating it with our own project. The issue was resolved by taking assistance of Muhammad Ali Naqvi (CS 2025) who had worked on the code.

ii) We faced a lot of **issues in displaying the rocket, aliens, score and lives using .mem files.** Some files gave distorted results and the errors generated were incomprehensible. Synchronizing these files all together was an issue as each used a different palette. We found it very late in our project that each .mem file reads from its own palette, and once that was clear our issues with the displays were quickly resolved.

iii) **Integrating ultrasonic sensor directly with FPGA** was very challenging, we took some assistance from our senior who sent us a github repo [4], we amended the code according to our need by using clock divider and other changes but still it did not work, so we had to use Arduino to take the readings from the Ultrasonic sensor and then send it to FPGA.

7. Code

The code for the project can be found in a GitHub repository following this link:
https://github.com/aa08453/DLD_Final_Project.git

References

[1] Syed Muhammad Ali Naqvi, *Interfacing Joystick using Basys3 XADC*, muhammadali74/Basys3-Joystick-Interfacing (github.com)

[2] https://www.javatpoint.com/arduino-ultrasonic-distance-sensor

[3] https://github.com/borisfba/fpga-distance/blob/master/sonic.v

*Image taken from the web or some other resource