Habib University

Dhanani School of Science and Engineering

Digital Logic and Design (EE-172L / CS-130L)

Frog Hunter

Group Members:

Muhammad Abubakar Mirza

Rohan Raj

Umar Kashif

Sikander Ahmed

1) Introduction:

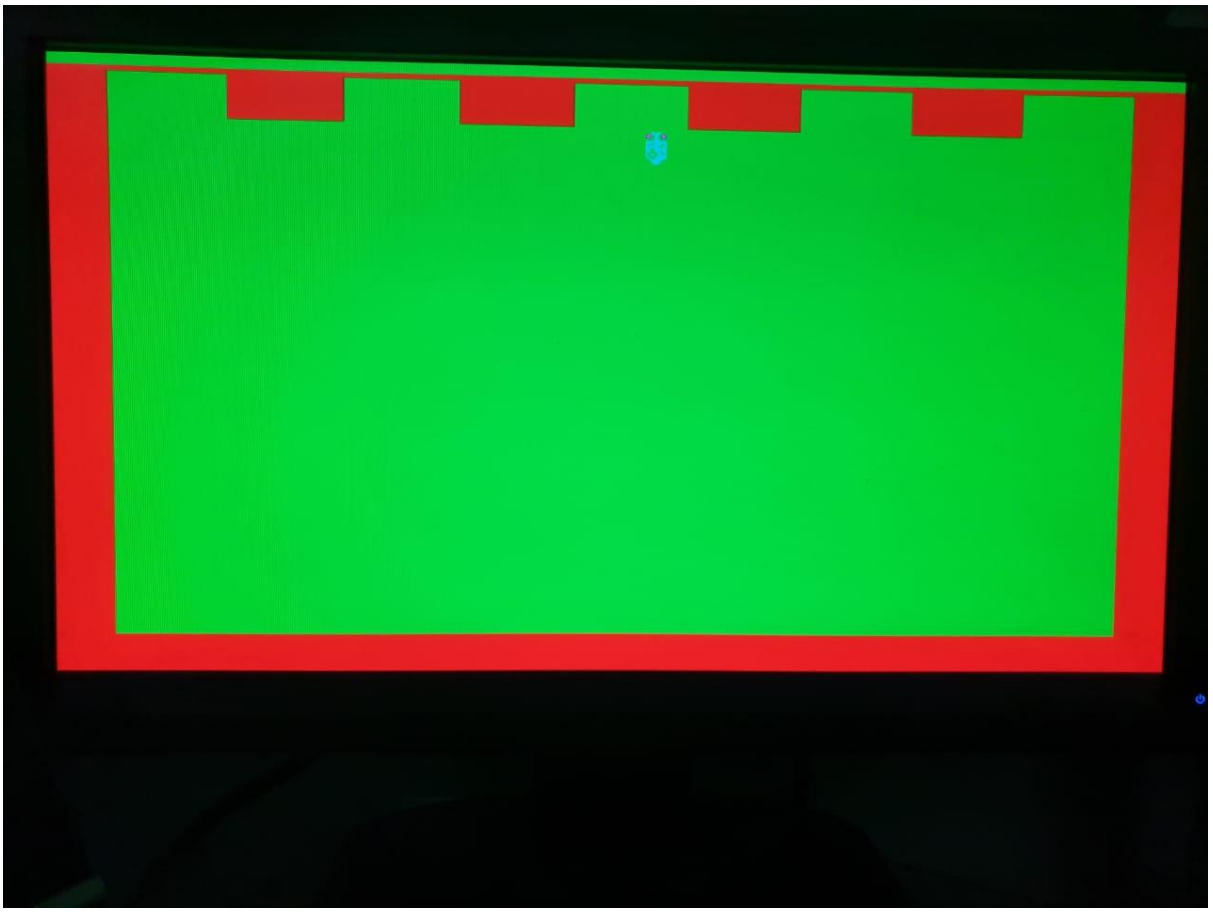
This is a two player game where one player will take the role of a frog, while the other player will take the role of the shooter. The shooter's goal is to shoot the frog three times before the timer runs out, while the frog's goal is to stay alive until the timer runs out.

2) Game Logic:

2.1) Basic Features:

2.1.1) Shooter's Movement:

The shooter is positioned at the bottom most part of the screen and is only allowed to move in the left and right directions which is controlled by the left and right arrow buttons on the FPGA board. It can only shoot upwards (north direction) for which the up arrow button is used.



MAIN SCREEEN:

2.1.1) Frog's Movement:

The frog is free to travel across the screen thanks to the use of a joystick. The capability of the frog travelling through the rightmost portion of the screen and emerging out of the leftmost part of the screen is not supported in this game. Instead, the frog is restricted from leaving a specified boundary on the screen.

2.2) Additional Features (Optional):

Power-ups for the frog and the shooter will be placed randomly on the screen, i.e. random power-ups will be dropped at random locations on the screen for the frog to pick up or the shooter to shoot to gain a personal advantage or a disadvantage.

2.2.1) Power-ups for Frog:

There will be powerups for the frog that will appear randomly on the screen. They will either-

- Make the frog invisible, and for a short amount of time it will be able to withstand any strikes that are dealt to it. The user still has control over the frog, but after five seconds it will no longer be available.
- Make the frog quicker - its existing speed will rise by a factor of 1.5, and this will help him avoid getting shot.
- Boost- Suddenly, the frog will advance a few steps, and the game will take control of it. However, the frog will be immune to death for this brief period of time. After getting the boost the frog will slow down to 0.75 its speed.

2.2.2) Power-ups for the Shooter:

- Make the frog slower - The shooter will have an easier time hitting the frog since the frog's speed will slow down by 0.5x and the frog will run slowly. This condition will last for the next seven seconds, following which the frog will return to its original speed.
- Invisibility- Because the hits will no longer be visible, it will be hard for the user to tell where the hits are located or where they should travel while controlling the frog. The shooter maintains control of the situation for the whole of it. This will be in effect for three seconds.
- Precision shooting- will fixate entirely on the frog for a split second, giving the shooter more time to concentrate on the target and increase their chances of scoring a hit.)

2.2.3) Collision Detection for Power-ups:

In this game, the bullet strikes both the frog and the powerups, and the frog consumes the powerups that are dropped for it. Multiple criteria will govern what happens when the frog consumes a given power-up or the shooter fires a specific power-up. For example, when the frog consumes a power-up, the timer increases, thus there must be criteria that are activated when the frog eats this specific powerup, directly tying it to the timer on screen.

2.2.4) Game Status:

Initially, there will be a timer starting at the 1 minute mark and 3 lives for the frog. As the game progresses, the time will keep decreasing. time will be added to the timer if the shooter shoots a specific powerup and the time will decrease from the timer if the frog eats a specific power up.

3) User Flow Diagram:

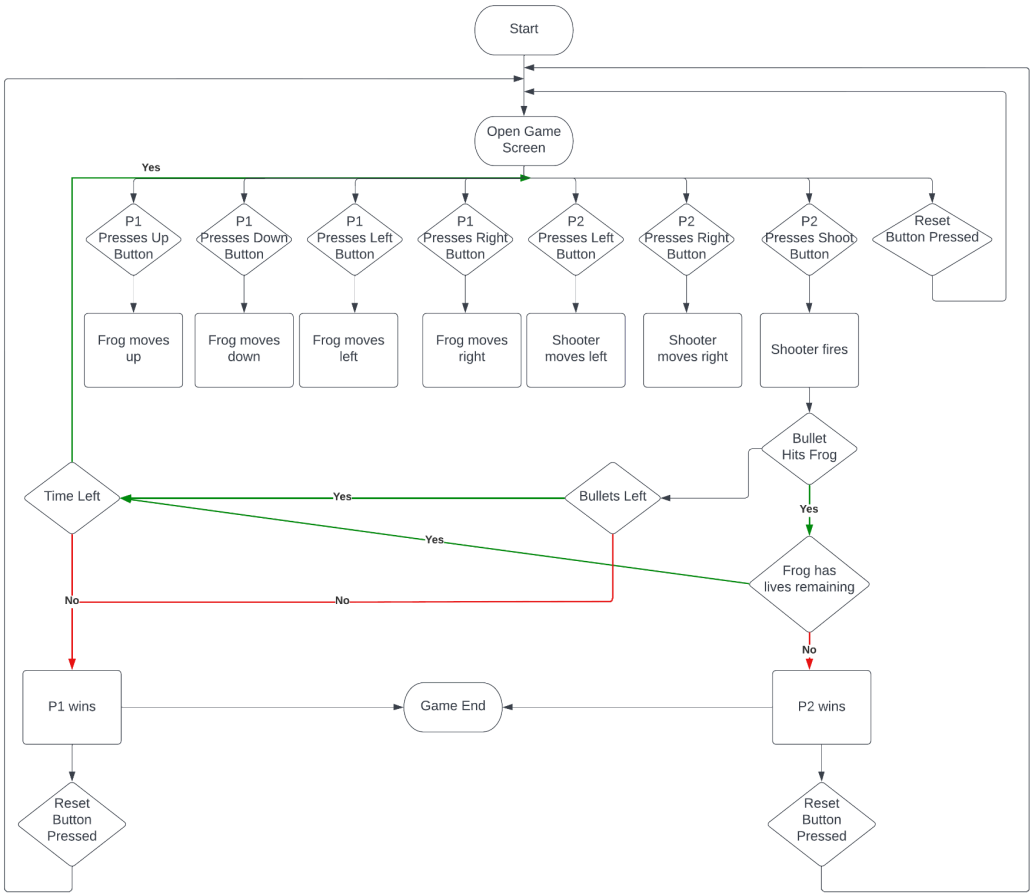


Figure 3.1: Block Diagram for the Game

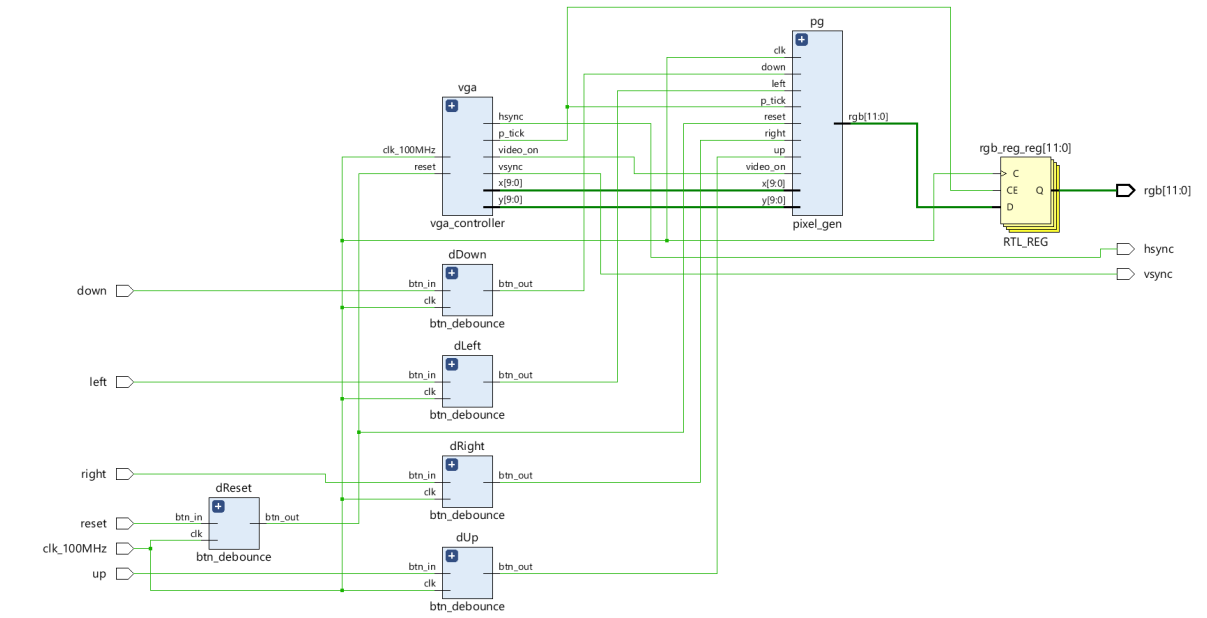


Figure 3.2: Schematic for Top Level Module of the Game

4) Input Block:

We are using two devices for input:

1) FPGA:
    a) Buttons:
        i) If used for Frog:

            Up : T18

            Left: W19

            Right: T17

            Down: U17

        ii) Shooter:

            Left: W19

            Right: T17

            Shoot : T18

    b) Clock Pin: It is used for clock input. The pin used is W5

    c) Seven Segment Displays: It is used to display the timer. The pins used are:

        Enabler for the display: W17

        Enabler for the first seven segment display: W4

        Enabler for the second seven segment display: V4

        Enabler for the third seven segment display: U4

        Enabler for the fourth seven segment display: U2

        S[0]: W7

        S[1]: W6

        S[2]: U8

        S[3]: V8

S[4]: U5

S[5]: V5

S[6]: U7

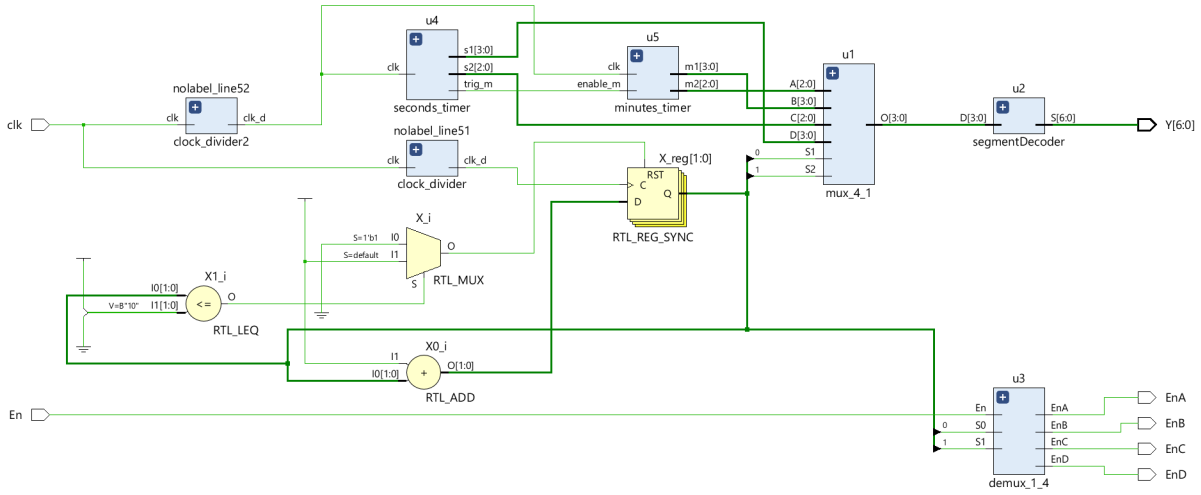


Figure 4.1: Schematic for Top Level Module for Timer

2) Joystick: FPGA buttons are reserved for the movement of the shooter while joystick is used for the movement of the frog.

5) Control Block:

The game consists of three different FSMs:

1) **Frog**
2) **Shooter**
3) **Bullet**

5.1) Frog FSM:

State Transition Table:

Inputs					Input State		Output State	
Reset	Up	Down	Left	Right	S1	S0	S1	S0
1	X	X	X	X	X	X	0	0
0	1	0	0	0	X	X	0	0
0	0	1	0	0	X	X	0	1
0	0	0	1	0	X	X	1	0
0	0	0	0	1	X	X	1	1

Figure 5.1.1: State Transition Table of Frog

State Transition Diagram:

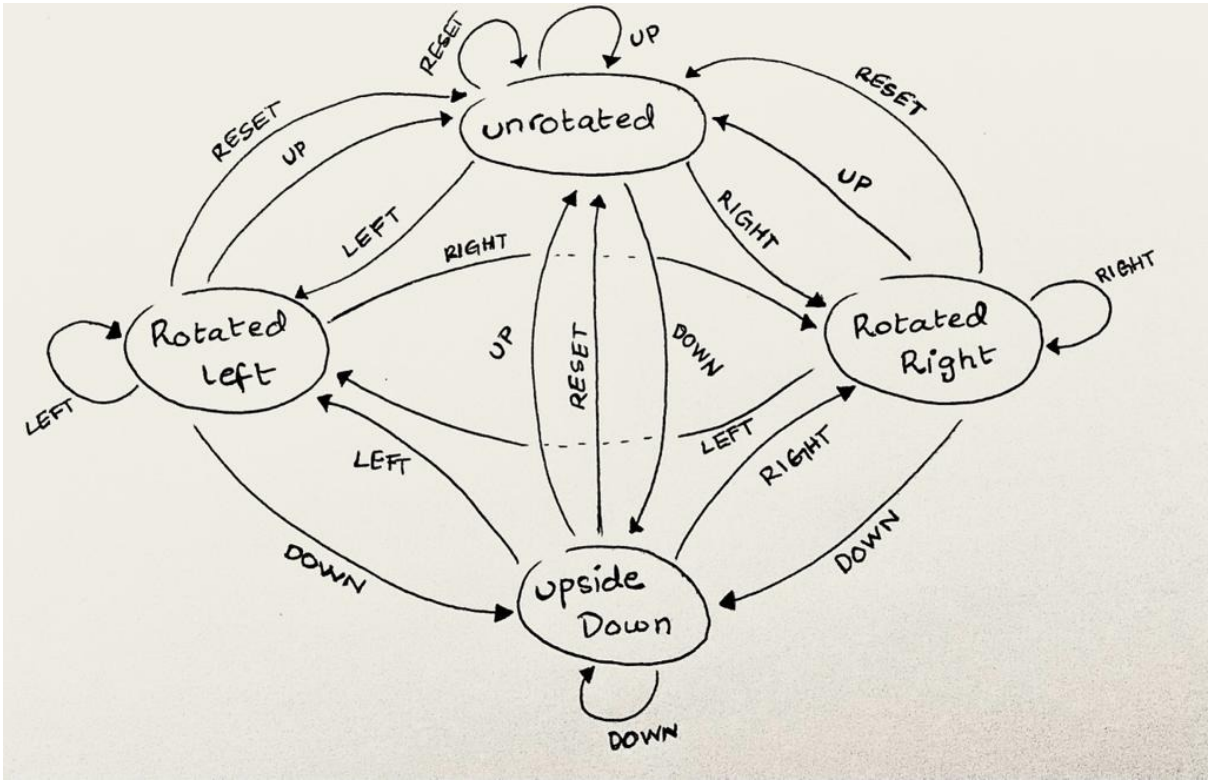


Figure 5.1.2: State Transition Diagram of Frog

State Assignment:

Up: 0 0

Down: 0 1

Left: 1 0

Right: 1 1

Logic Diagram:

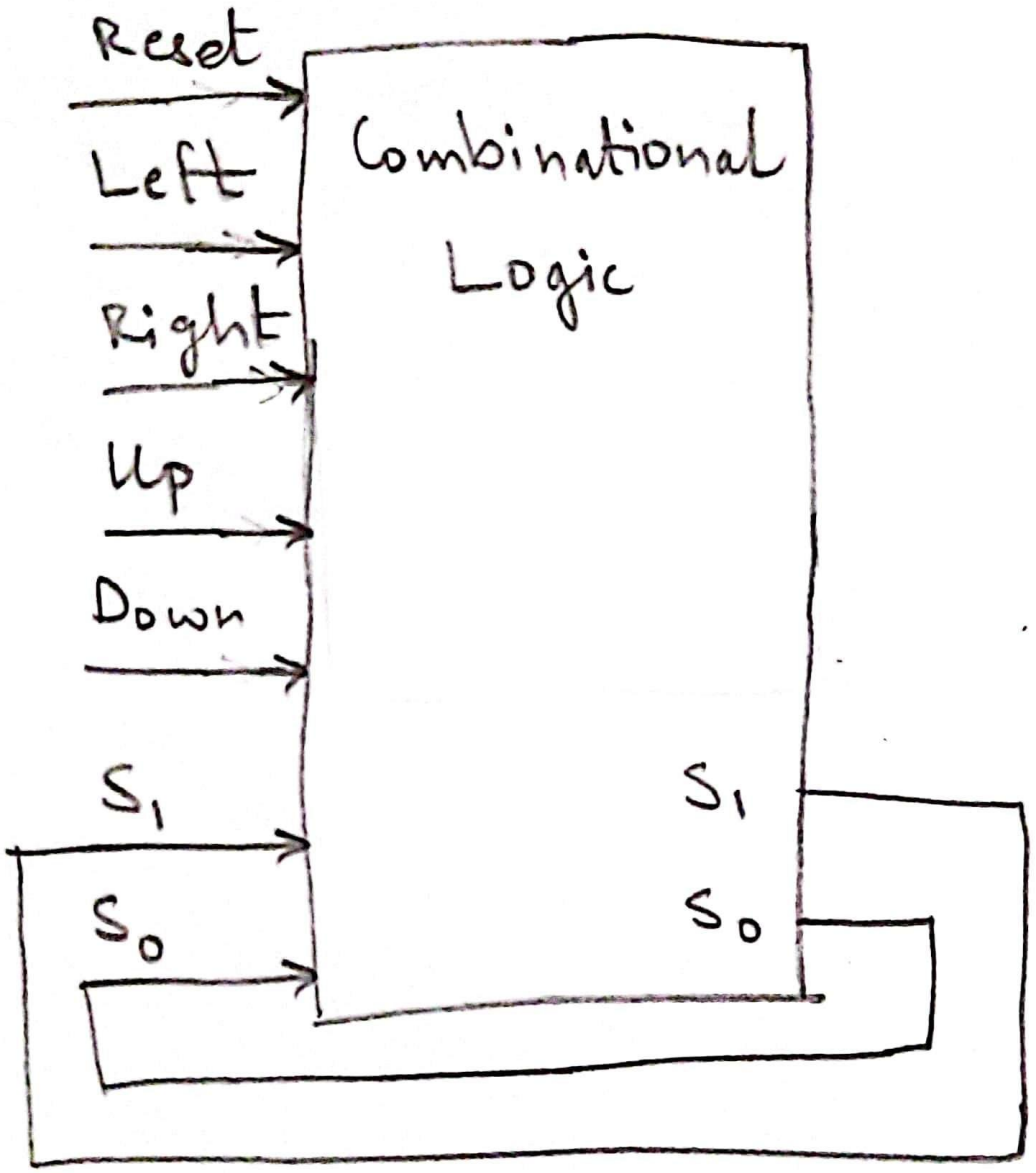


Figure 5.1.2: Logic Diagram of Frog

5.2) Shooter FSM:

State Transition Table:

Inputs			Input State		Output State	
Reset	Left	Right	S1	S0	S1	S0
1	X	X	X	X	0	0
0	0	0	0	0	0	0
0	0	1	X	X	0	1
0	1	0	X	X	1	0
0	1	1	0	0	0	0
0	1	1	0	1	0	1
0	1	1	1	0	1	0
0	1	1	1	1	1	1

Figure 5.2.1: State Transition Table of Shooter

State Transition Diagram:

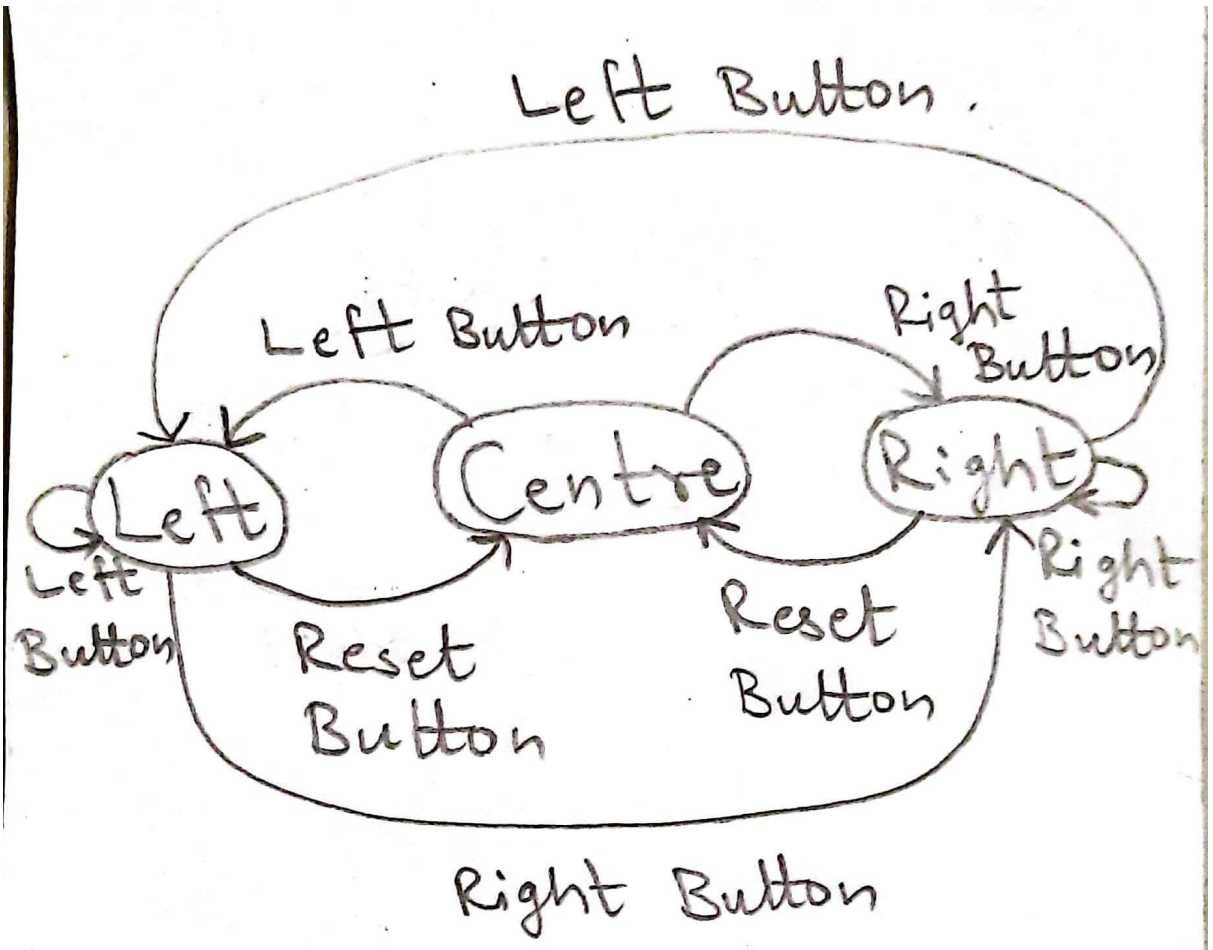


Figure 5.2.1: State Transition Diagram of Shooter

State Assignment:

Left: 01

Right: 10

Logic Diagram:

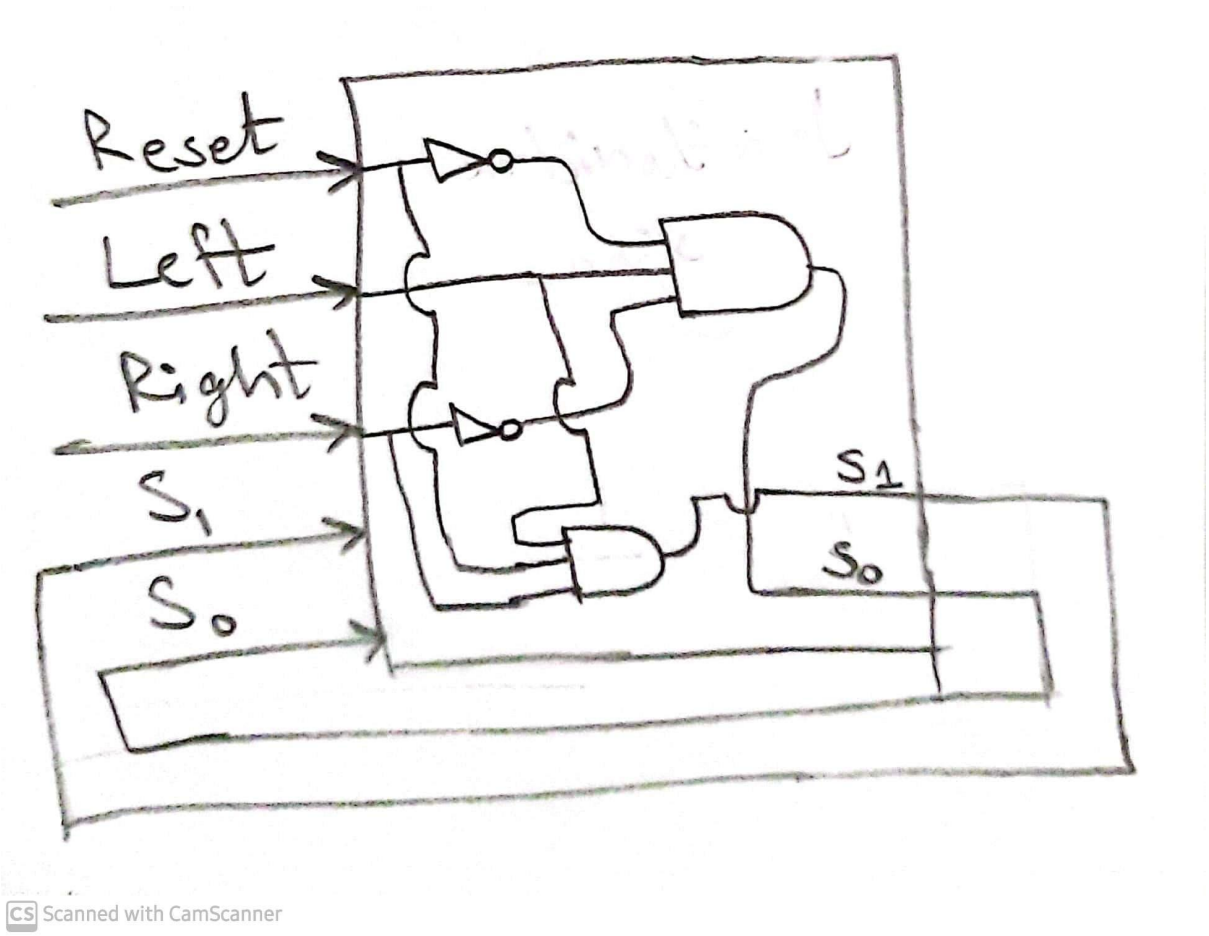


Figure 5.2.2: Logic Diagram of Shooter

5.3) Bullet FSM:

State Transition Table:

Inputs		Input State		Output State	
Reset	Fire	S1	S0	S1	S0
1	X	0	0	0	0
0	0	0	0	0	0
0	1	0	0	0	1
0	X	0	1	1	0
0	X	1	0	1	1

Figure 5.3.1: State Transition Table of Bullet

State Transition Diagram:

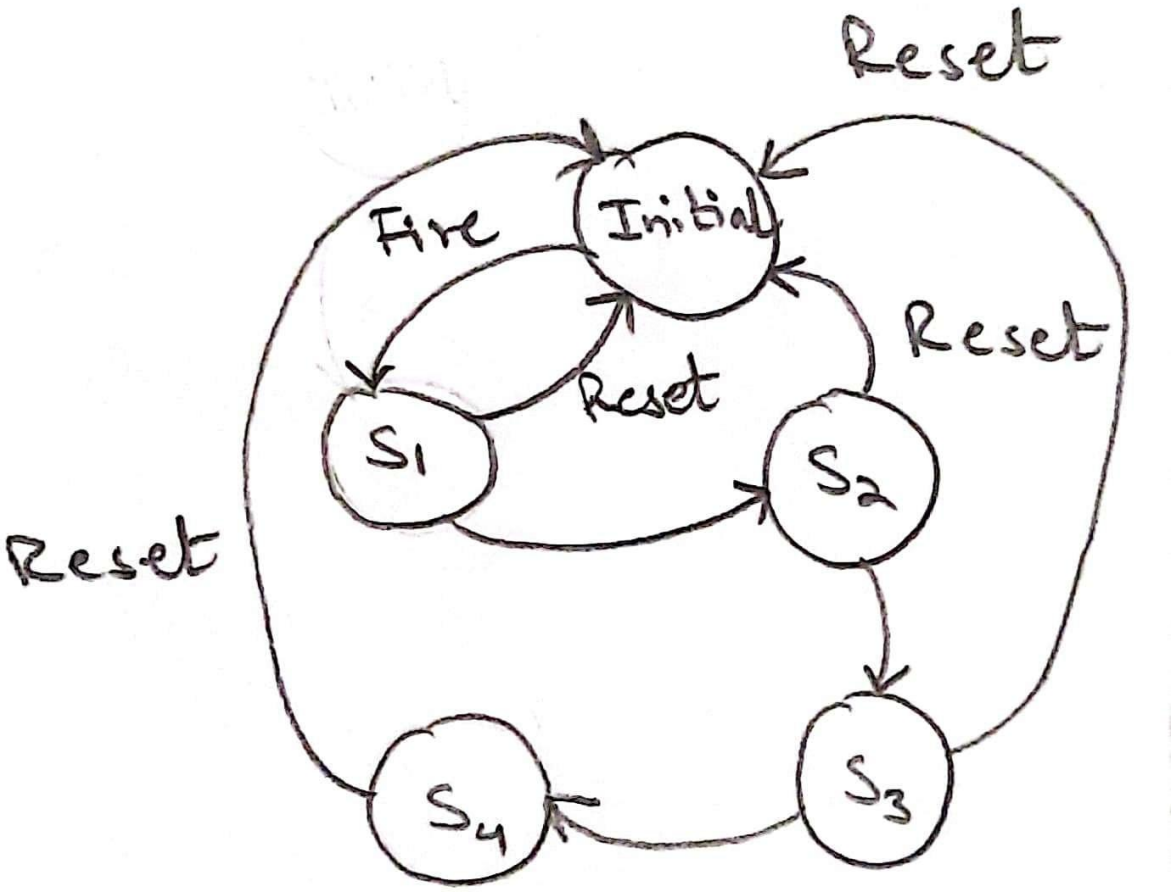


Figure 5.2.1: State Transition Diagram of Bullet

If fire is true, the state keeps on incrementing until it reaches the top or hits the frog.

Logic Diagram:

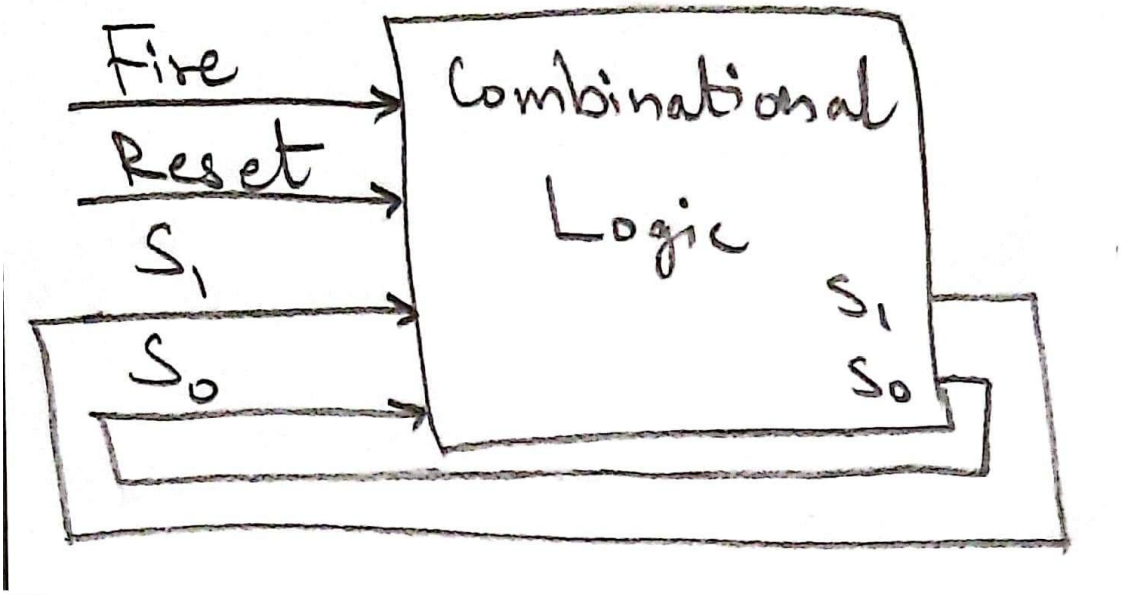


Figure 5.3.2: Logic Diagram of Bullet

Reset pin: U18

State Transition Diagram of the Game:

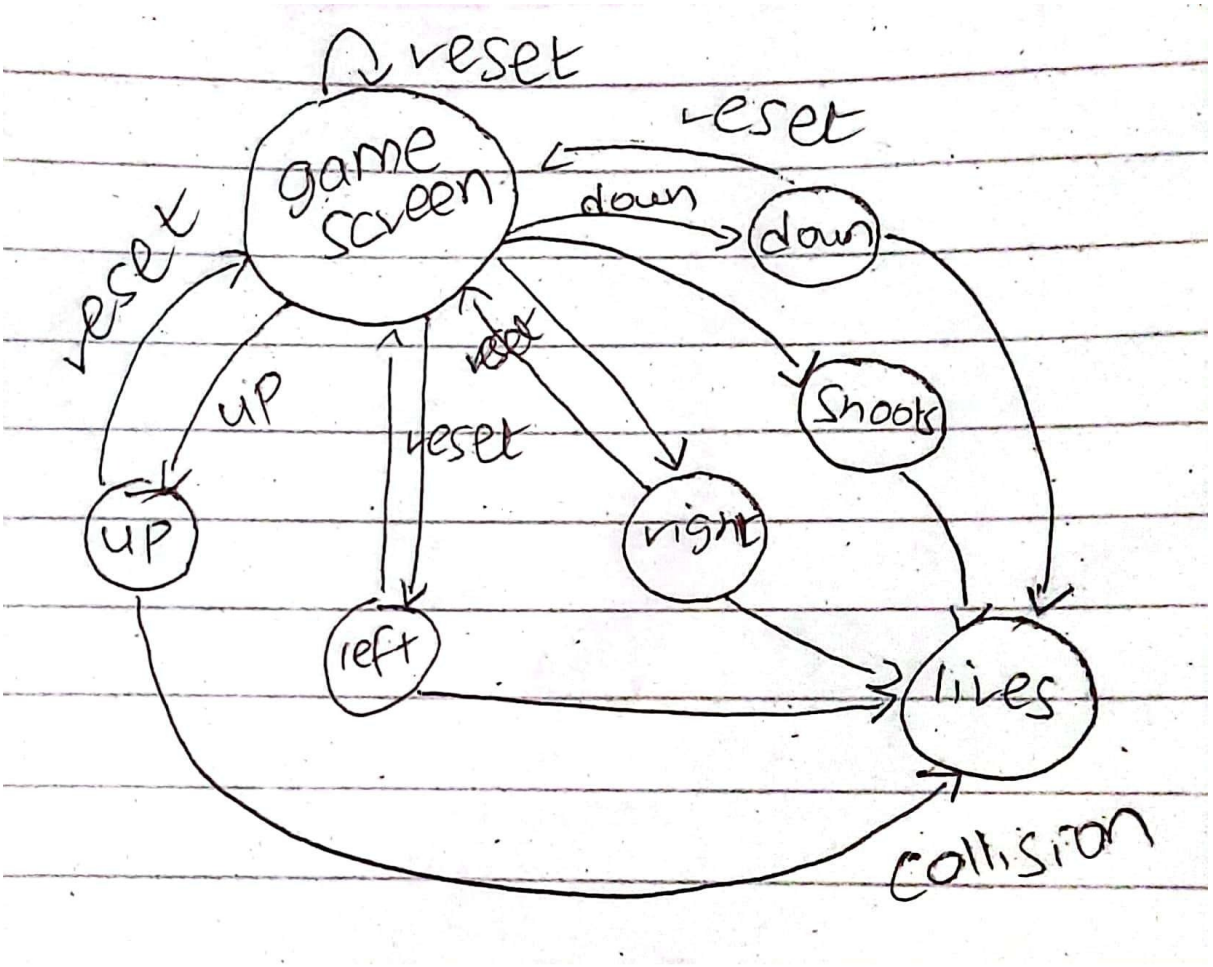


Figure 5.4: State Transition Table of the Game

6) Output Block:

The h sync and v sync modules receive a 25MHz clock generated by the clock divider module. The pixel x (the horizontal pixel location) on the screen corresponds to the h count, and the v sync module increases the v count when the value of h count reaches its maximum (right most corner of the screen). This is how all of the screen's pixels are accessed.

The top-level module vga controller (the primary display module), which generates and updates the output of the screen coming from the Control Block, calls the pixel gen module to assign an RGB value to each pixel on the screen.

The pixel gen module also calls the bitmaps for the first screen, the main game screen, the shooter pad, and the final result screen to get information on where to print which pixels. This is programmed into Basys 3 using VGA.

The most essential part of the video game is the VGA module, which enables visuals to be displayed on a screen. Let's start by looking at the board's VGA port's technical specifications.

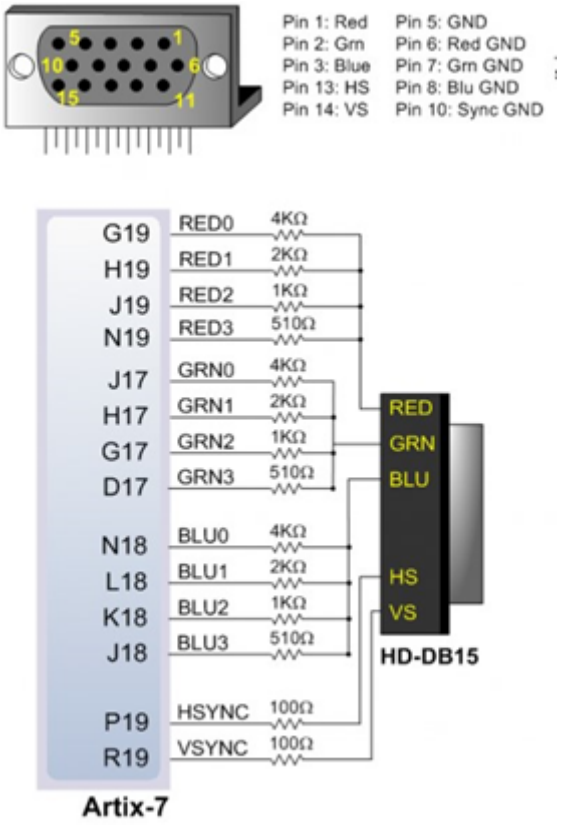


Figure 6.1: Basys 3 VGA Port

The handbook states that this VGA port provides 12-bit depth colour, with 4-bits for red, green, and blue. This means that a total of 163 = 4096 colours can be displayed.
To establish a functioning display system, a video controller circuit must be built in the FPGA to drive the sync and colour signals at the proper timing.

However as we have only specified only 3 bits for pixels (RGB), we can have a combination of total 8 colors [1].

Three-bit VGA color combinations			
Red (R)	**Green (G)**	**Blue (B)**	**Resulting color**
0000	0000	0000	black
0000	0000	1111	blue
0000	1111	0000	green
0000	1111	1111	cyan
1111	0000	0000	red
1111	0000	1111	magenta
1111	1111	0000	yellow
1111	1111	1111	white

Figure 6.2: VGA Colour Combinations

The following inputs go into the pixel gen module:

100MHz clock

Reset button

Up button

Down button

Left button

Right button

Position of x coordinate

Position of y coordinate

Video signal

25MHz pixel/second rate signal

6.1) Graphics:

The graphics are categorized into three categories:

1) Background
2) Objects:
    a) Frog
    b) Shooter
    c) Bullet
3) Timer

6.2) Modules:

A separate module is created for each movement. Movements can be categorized into two types:

1) Movement of Frog:

    In case of FPGA Buttons:

    a) Left
    b) Right
    c) Up
    d) Down

    In case of Joystick:

        All directions

2) Movement of Shooter:
    a) Left
    b) Right

The functionality of the movement modules of both of these objects will be the same. For example, the functionality of the frog moving to the left and the shooter moving to the left will be the same.

In order to display these, a parent module pixel_gen.v is created which calls all of these modules. The display will be determined using conditional "if" statements. This module is also responsible for what is displayed in the background. The output of this module is a 12 bit register which stores the colour codes of the colours to be displayed. The colours are displayed on a particular area which is determined using "if" statements. Currently, the background consists of different solid colours only.

6.3) Seven Segment Display:

Currently, the timer is displayed on the four seven segment displays on FPGA where the first two segments are used to display minutes while the last two segments are used to display seconds. The timer is set to a default value of one minute.

7) Sprites:

The steps to load the sprites in the fpga are:

1) Download the image you want and crop it to the appropriate dimensions.
2) Use the coetool to export the 8 bit file of the image.
3) Using vivado load the ip file in the block ram memory generator.

The image can be accessed by reading the sram module [2].

In order to use pictures for the elements of the game, we have made the following sprites:

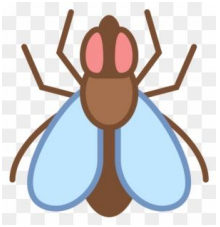
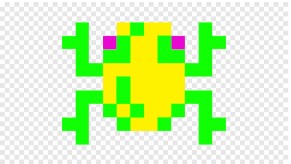


Figure 7.1

Figure 7.2

Figure 7.3

Figure 7.4



Figures 7.5

References:

[1] FPGAPrototypingByVerilogExamples

[2] Nikkatsa, “Space invaders FPGA game,” FPGAWORLD, 11-Feb-2017. [Online]. Available: https://fpgaworld.wordpress.com/2016/05/20/space-invaders-fpga-game/. [Accessed: 14-Dec-2022].