Habib University

Dhanani School of Science and Engineering

Digital Logic and Design
EE/CS 172/130

T6

Final Project Report
(Milestone 3)

Simon Says Memory Game

Eman Fatima (08595), Fakeha Faisal (08288), Muhammad Ibad Nadeem (08440)

Professor Haseeb Khan
Professor Hur Rizvi

Dec 16th 2023

Table of Contents:

Abstract

This project involves creating a Simon Says Memory Game using four push buttons. The user interacts with the game through those push buttons representing colors: red, blue, green, and yellow. The game begins with the system displaying a random color, and the player must replicate the sequence. With each correct input, the system adds a new color, increasing the difficulty. The score is displayed on the seven-segment FPGA. The display visually represents the sequence by illuminating the corresponding colors. The system is designed to provide an engaging and challenging experience while promoting cognitive development and enhancing memory skills. The project meticulously elucidates the design intricacies, encapsulating the sophistication of digital logic principles deployed in the realization of an engaging and interactive gaming experience.

1. Introduction

This project focuses on designing and implementing a Simon Says game within the context of Digital Logic Design (DLD). The Simon Says game, a classic memory and pattern recognition game, is realized using four distinct colors: red, green, blue, and yellow. The user interacts with the game through a set of four push buttons, each corresponding to one of the four colors. The game is developed on an FPGA Basys 3 board and programmed using Verilog HDL. The game was developed electronically in 1978 as a children's game and we have attempted to implement it through push buttons and a screen.

Fig 1: Simon Says Color Game Electronic version

The game begins with a screen that displays "Game Begin", representing the beginning of the game. Pressing any of the four push buttons will initiate the game and the game's running screen will be displayed. The game screen will have four boxes displaying the colors red, blue, green, and yellow. Added to that, the game title "Simon Says" is displayed on the screen and the user score will be displayed on the FPGA seven-segment display. The user must press the correct push buttons corresponding to the color sequence generated by the game. If the user presses the correct color, the score displayed on the FPGA will increase, and the game will generate the next color in the sequence along with the previous colors. Different music will be played when the game displays a color when the player wins or loses the game through a speaker connected to the FPGA on the pin A-14 (specifically for sound). If the user presses the wrong color and loses, they will be directed towards the game end screen.

2. Implementation

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

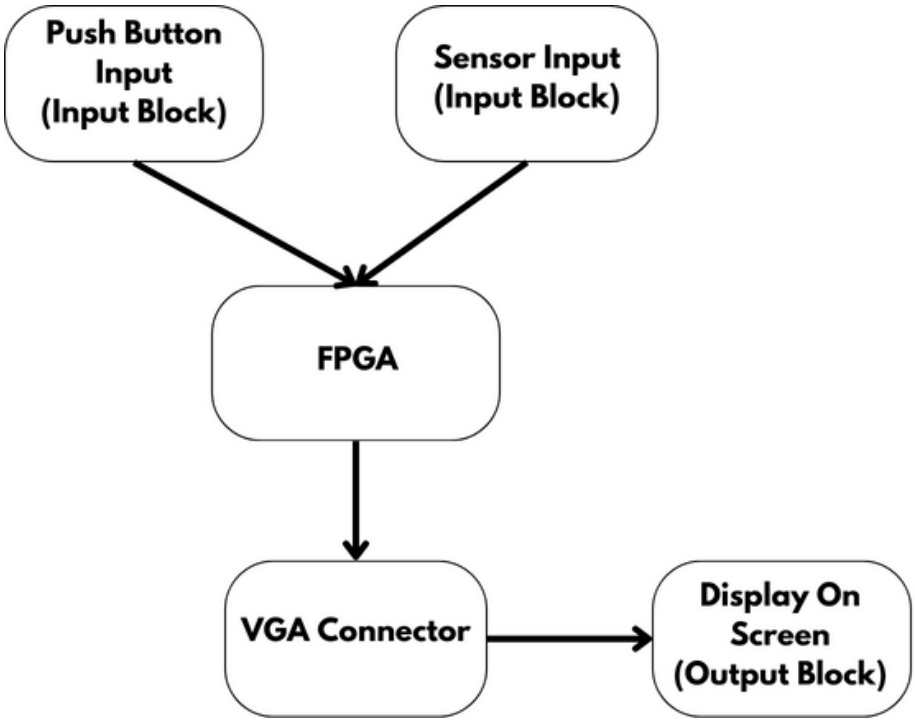


Fig 2: Game Design Overview

2.1. Input Block

The user will interact with the game using push buttons that are connected to the breadboard, which will serve as our source of input. The pressing of the push buttons will be translated on the display screen as the color the user selects.

Input	**Functionalities**
Push Buttons	Four buttons for each color
FPGA V-18 Button	Reset

2.1.1. Push Buttons

Inputs will be taken through four push buttons which we have integrated using a circuit made on the breadboard which maps onto the four squares of colors on the screen. We have four variables for these squares and the square that brightens up (indicates being pressed) gets mapped on the variable. This information is then sent back to the control block. Our FPGA is the source of the voltage and the ground for this circuit. The power pins of the FPGA provide a voltage of about 3.3V.

I/O Ports

Name	Direction	Neg ...	Package Pin	Fixed	Bank	I/O Std	Vcco
All ports (36)							
an (4)	OUT			☑	34	LVCMOS33*	3.300
an[3]	OUT		W4	☑	34	LVCMOS33*	3.300
an[2]	OUT		V4	☑	34	LVCMOS33*	3.300
an[1]	OUT		U4	☑	34	LVCMOS33*	3.300
an[0]	OUT		U2	☑	34	LVCMOS33*	3.300
blue (4)	OUT			☑	14	LVCMOS33*	3.300
blue[3]	OUT		J18	☑	14	LVCMOS33*	3.300
blue[2]	OUT		K18	☑	14	LVCMOS33*	3.300
blue[1]	OUT		L18	☑	14	LVCMOS33*	3.300
blue[0]	OUT		N18	☑	14	LVCMOS33*	3.300
green (4)	OUT			☑	14	LVCMOS33*	3.300
green[3]	OUT		D17	☑	14	LVCMOS33*	3.300
green[2]	OUT		G17	☑	14	LVCMOS33*	3.300
green[1]	OUT		H17	☑	14	LVCMOS33*	3.300
green[0]	OUT		J17	☑	14	LVCMOS33*	3.300
red (4)	OUT			☑	14	LVCMOS33*	3.300
red[3]	OUT		N19	☑	14	LVCMOS33*	3.300
red[2]	OUT		J19	☑	14	LVCMOS33*	3.300
red[1]	OUT		H19	☑	14	LVCMOS33*	3.300
red[0]	OUT		G19	☑	14	LVCMOS33*	3.300
seg (7)	OUT			☑	34	LVCMOS33*	3.300
seg[0]	OUT		W7	☑	34	LVCMOS33*	3.300
seg[1]	OUT		W6	☑	34	LVCMOS33*	3.300
seg[2]	OUT		U8	☑	34	LVCMOS33*	3.300
seg[3]	OUT		V8	☑	34	LVCMOS33*	3.300
seg[4]	OUT		U5	☑	34	LVCMOS33*	3.300
seg[5]	OUT		V5	☑	34	LVCMOS33*	3.300
seg[6]	OUT		U7	☑	34	LVCMOS33*	3.300

Scalar ports (13)

BTN0	IN		R18	☑	14	LVCMOS33*	3.300
BTN1	IN		P17	☑	14	LVCMOS33*	3.300
BTN2	IN		M19	☑	14	LVCMOS33*	3.300
BTN3	IN		L17	☑	14	LVCMOS33*	3.300
CLK	IN		W5	☑	34	LVCMOS33*	3.300
h_sync	OUT		P19	☑	14	LVCMOS33*	3.300
LED0	OUT		L1	☑	35	LVCMOS33*	3.300
LED1	OUT		P1	☑	35	LVCMOS33*	3.300
LED2	OUT		N3	☑	35	LVCMOS33*	3.300
LED3	OUT		P3	☑	35	LVCMOS33*	3.300
RST	IN		U18	☑	14	LVCMOS33*	3.300
SND	OUT		A14	☑	16	LVCMOS33*	3.300
v_sync	OUT		R19	☑	14	LVCMOS33*	3.300

Fig 2.5: Input ports integrated in Verilog

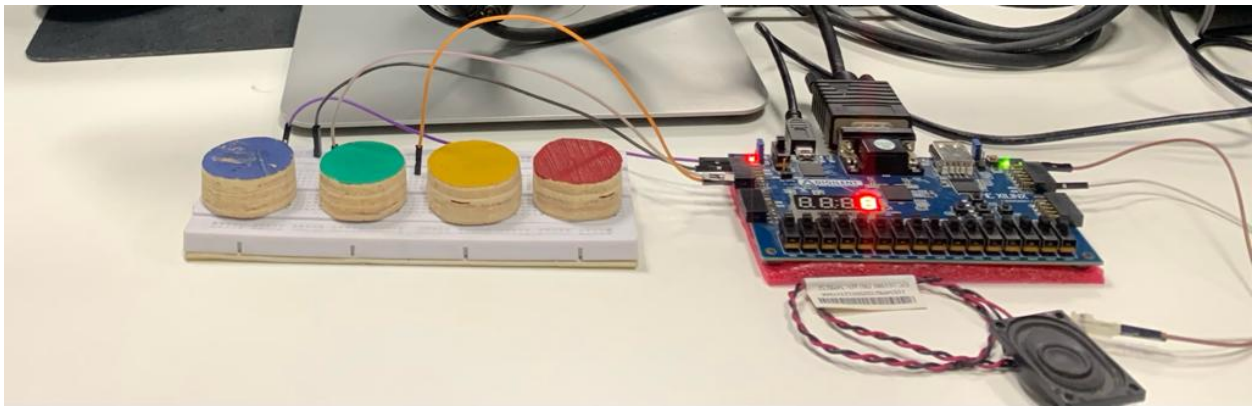


Fig 3: Push Buttons as the external source of input

2.1.2. V-18 Push Button of Basys-3 FPGA

We have simply used the V-18 button of the FPGA board to implement the reset. This button resets the game at any instant and switches the screen back to the starting screen.

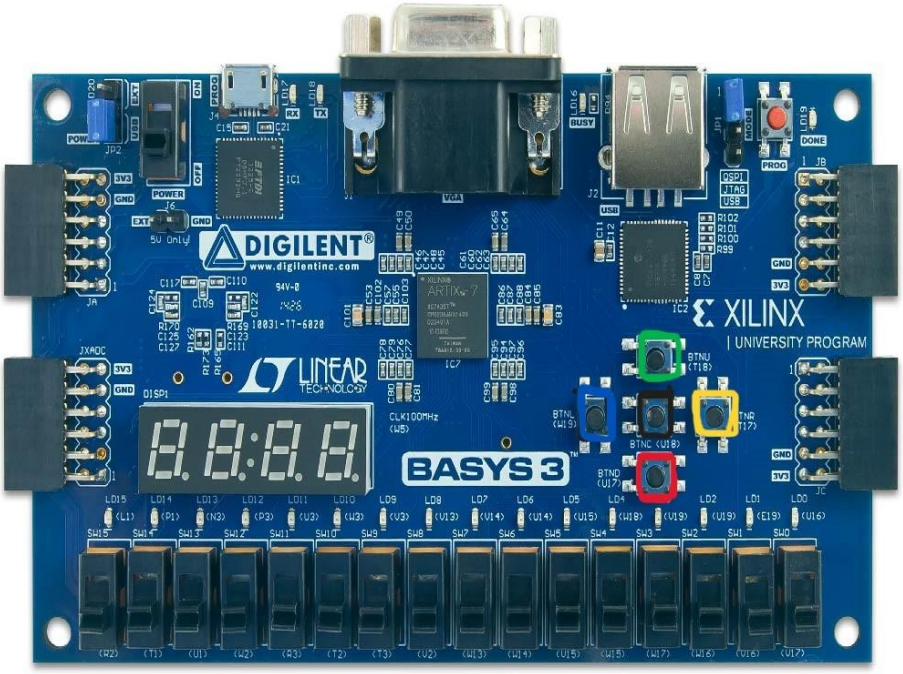


Fig 4: Basys 3 FPGA layout

Here we have annotated the four pins being used with their respective colors and the middle button annotated with black color representing the reset key

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 “Game Begin”.

Fig 5: Game Begin Screen

The modules controlling the start screen are **clock_divider, h_counter, v_counter, vga_sync, and pixel_gen.** The module pixel_gen has three conditions based on the state of the game: when the game state is equal to the Power On state, the start screen is displayed.

State Transition: The start screen appears when the control block has the starting state. The player is supposed to press any one of the four push buttons which will result in the transition from the start screen to the game screen.

2.2.2. Game Screen

Fig 6: Game Screen

This is the screen where the whole gameplay is based. It is dependent on the **Top** Module consisting of five modules: **clock_divider, h_counter, v_counter, vga_sync, and pixel_gen and simon.**

i. Pixelgen Module:

The "pixel_gen" Verilog module serves as a pixel generator designed for creating colored rectangles on a display. Operating on a pixel clock (clk_d) and receiving input coordinates (pixel_x and pixel_y), the module intelligently selects colors for pixels based on their positions within predefined rectangles. The rectangles, denoted by distinct colors (blue, green, yellow, and red), are determined by conditional statements evaluating the pixel's coordinates. The color intensity of each rectangle is modulated by four input signals (LED0 to LED3). The module employs a structured case-based approach to assign specific RGB values to pixels within each colored region, facilitating the creation of dynamic and configurable displays. Additionally,

defaulting to black for pixels outside the specified rectangles, this module offers flexibility in generating visually appealing patterns or images on the display by leveraging the interactive control provided by the LED inputs.

ii. Vga_sync Module:

The "vga_sync" module in Verilog generates synchronization signals for a VGA display system. It takes horizontal and vertical count signals as inputs, producing synchronization signals (h_sync and v_sync), as well as indicating whether the video is active (video_on). The module calculates current pixel coordinates (x_loc and y_loc) based on input counts. Parameters are defined for display and porch sizes. Conditional assignments ensure proper synchronization by considering display areas, front and back porches, and retrace intervals for both horizontal and vertical directions. This module plays a crucial role in coordinating pixel generation and display timing for VGA output.

iii. h_counter Module:

The "h_counter" Verilog module is a horizontal counter designed to count clock cycles and generate synchronization signals for a VGA display system. It takes a clock signal (clk) as an input and produces two outputs: h_count, representing the horizontal count, and trig_v, a trigger signal. The module initializes these outputs to zero and increments h_count on each rising edge of the clock until it reaches 799. At this point, it sets trig_v to 1 and resets h_count to 0. The module essentially counts clock cycles, providing horizontal synchronization for the VGA system and triggering vertical synchronization (trig_v) when a complete horizontal line is scanned.

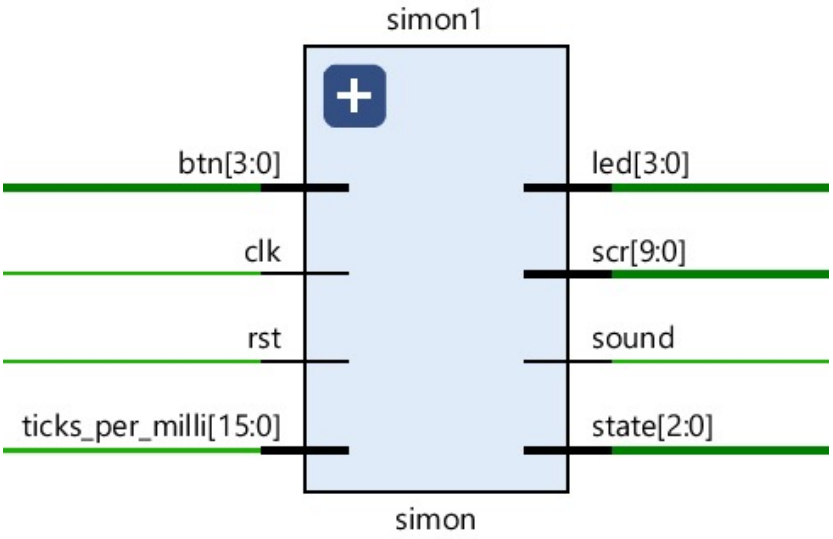
iv. V_counter Module:

The "v_counter" Verilog module serves as a vertical counter for a VGA display system. It takes a clock signal (clk) and an enable signal (enable_v) as inputs, producing the v_count output representing the vertical count. The module initializes v_count to zero and increments it on each rising edge of the enable signal until it reaches 524. When the count exceeds 524 or the enable signal is not active (enable_v is not equal to 1), the counter resets to 0. This module plays a crucial role in counting vertical lines and contributes to the synchronization of the VGA display.

v. Simon Module:

The "Simon" Verilog module implements a simplified version of the Simon game, an electronic memory game. The module incorporates game logic, button input processing, LED output control, and sound generation. The game has several states, including initialization, gameplay, user input, level progression, and game over. The LED outputs (led) represent the sequence of

tones that the player must remember and replicate. The module includes tone sequences for the game, success, and game-over scenarios. It also manages a score (scr) that increases with each successful level completion. The game features sound feedback, and the play module generates sounds based on the specified frequency (sound_freq). The state machine within the module controls the game flow and transitions between states based on user input and game progression, providing an interactive experience for the player.



vi. Top Module:

The "top" Verilog module integrates various modules to create a VGA display system with interactive features. The module takes clock (CLK), reset (RST), and button inputs (BTN0 to BTN3) and produces LED outputs (LED0 to LED3), horizontal and vertical synchronization signals (h_sync and v_sync), and RGB values (red, green, blue) for pixel color. Additionally, it includes a sound output (SND). The clock is divided using a clock divider module (clock_divider), and the resulting clock (CLKD) is used to drive a Simon game (simon1). The Simon game is controlled by buttons and outputs to LEDs and sound. The horizontal and vertical counters (h_counter and v_counter) generate counts that contribute to the VGA synchronization, and the vga_sync module coordinates the timing of the display. Finally, the pixel_gen module assigns colors to pixels based on their coordinates and the state of LEDs. This top-level module orchestrates the functionality of the individual modules to create an interactive VGA display with a Simon game and colored pixel generation.

State Transition: If the user presses an incorrect color and loses, the game moves to the end screen.

2.2.3. End Screen:

The end screen will be displayed once the player presses the incorrect color. The end screen will display the message of Game End.

Fig 7: Game End Screen

The modules controlling the end screen are **clock_divider, h_counter, v_counter, vga_sync, and pixel_gen**. The last of the three conditions is when the game state is equal to the Game over, the end screen is displayed.

State Transition: On pressing the reset button, the score is set back to zero and the game begin screen is displayed.

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 to 640 pixels, and for the vertical axis the video is on from 0 to 480 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 the horizontal axis and the v_counter for the vertical axis. Then Vga_sync was used for turning the video on on the required part of the screen and finally pixel gen was used to create our desired screen.

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

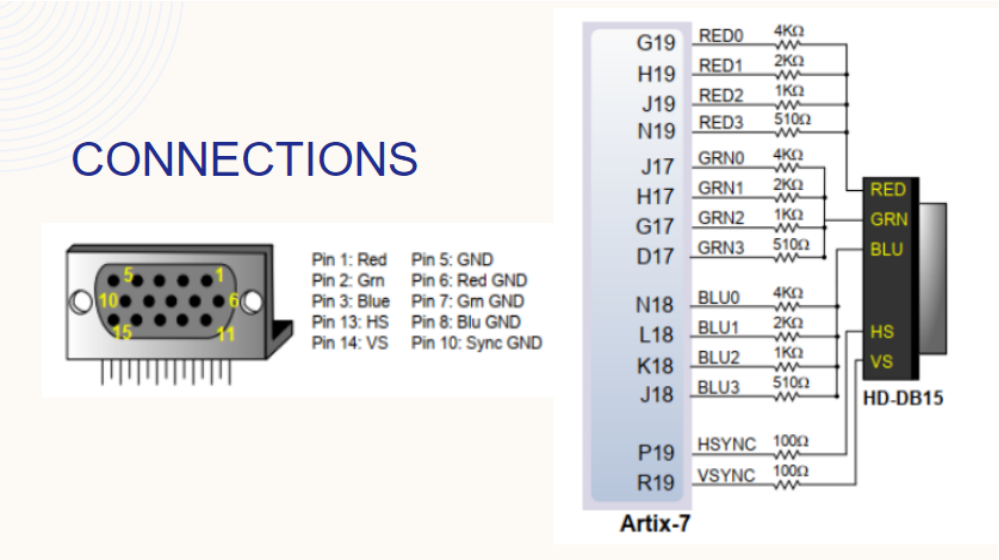


Fig 8: VGA Connections and RGB Ports

3. User Flow Diagram

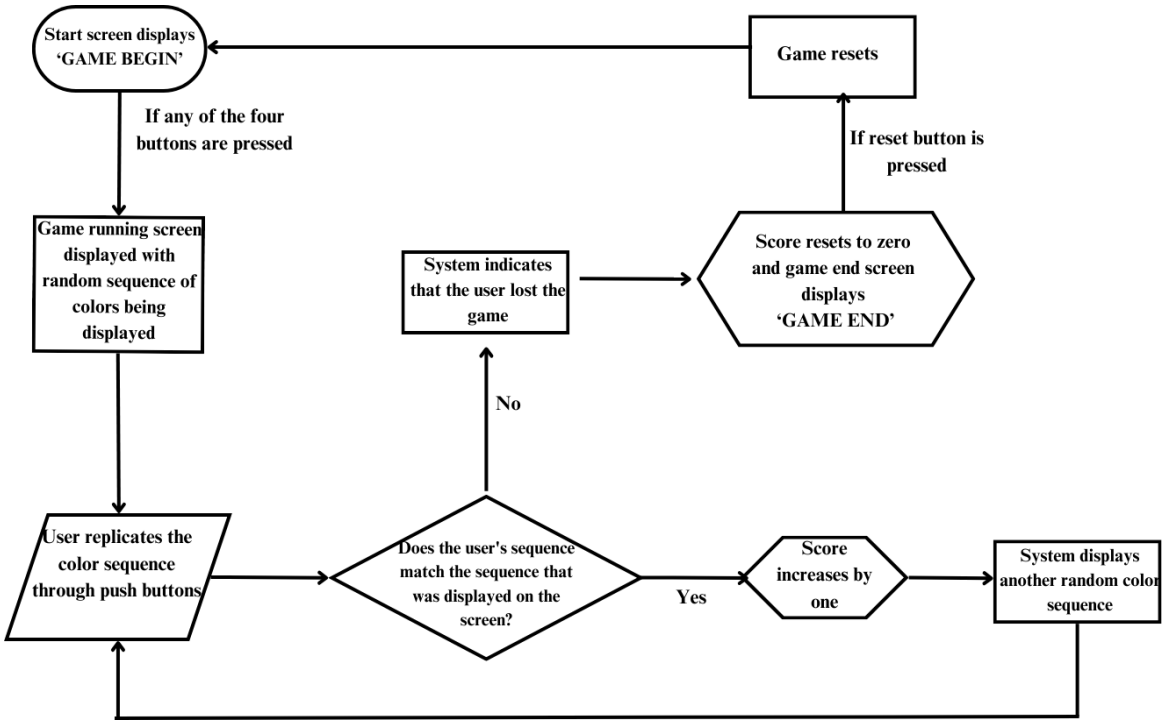


Fig 12: User Flow Diagram

The game starts with the start screen and upon clicking any one of the four push buttons, the game running screen is displayed. If the user replicates the correct color sequence, the score increments by one and the system displays another color, repeating the cycle. On the other hand, if the wrong color is pressed, the game end screen is displayed indicating that the user has lost. Afterward, the player has to press the push button on the FPGA to reset the game and go back to the game start screen.

4. FSM

The start and end states of the game, dictated by the start game and end game screens, take the push buttons of the FPGA and four colors as input for state transitions. Furthermore, our main game itself gives outputs based on any kind of input from the user (the next color sequence based on the correct sequence pressed by the user) or instead terminates automatically when the incorrect color is pressed, decreasing the count to zero and moving to the "Game Over" screen. This output can be seen as dependent on the current state and current inputs hence, we can conclude that the FSM implemented in our game is a **Mealy Machine** as the output of our game is dependent on the input block.

5. Major Challenges Faced

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

i) **Implementation of push buttons:** It was very hard for us to integrate the push buttons within our project using the breadboard and connecting it with the FPGA. We resolved this issue by getting assistance from an Electrical Engineering student Hania Kashif (Batch of 2026) who had implemented the same push buttons before.

ii) **Writing the code for Start and End Screens:** We had no experience in writing the verilog code for creating screens with writing phrases on them. Getting the coordinates right for each letter and making them look visually aesthetic was quite hard and consumed a major chunk of our time. We analyzed the code for a start screen we found on Git Hub and then implemented our screens.

6. GitHub Link

The code for the project can be found in a GitHub repository following this link:

https://github.com/IbadNadeem/DLD-Simon-Says

7. References

[1] simon-game-verilog - Wokwi ESP32, STM32, Arduino Simulator

*Image taken from the web or some other resource