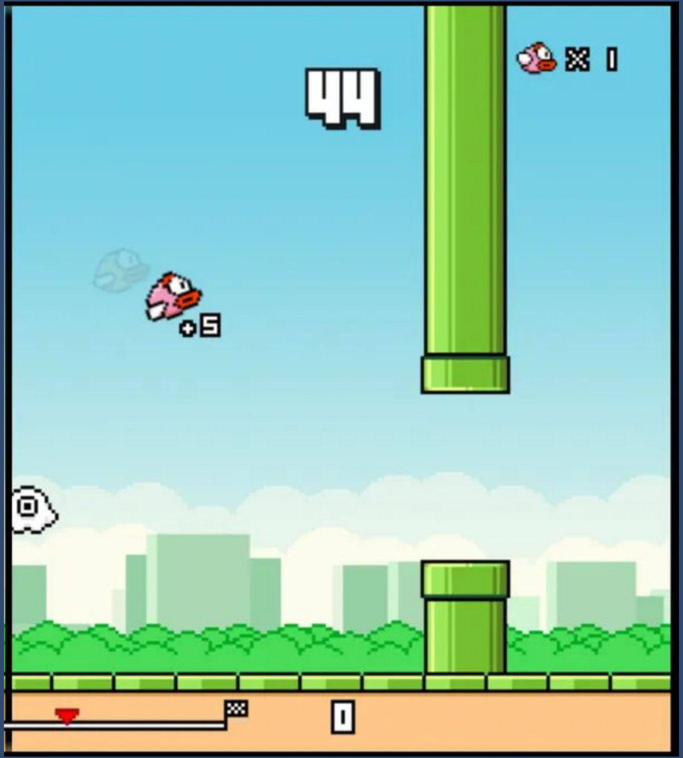
ECE241 Project: Birdy Soar

Laila Assy
Margaret Boychuk
Manahil Saeed

Description

- **Goal**: recreate the "Flappy Bird" mobile gameplay on a VGA-based hardware system
- **Tools used**: FPGA, Quartus, Modelsim, Verilog



Main Components

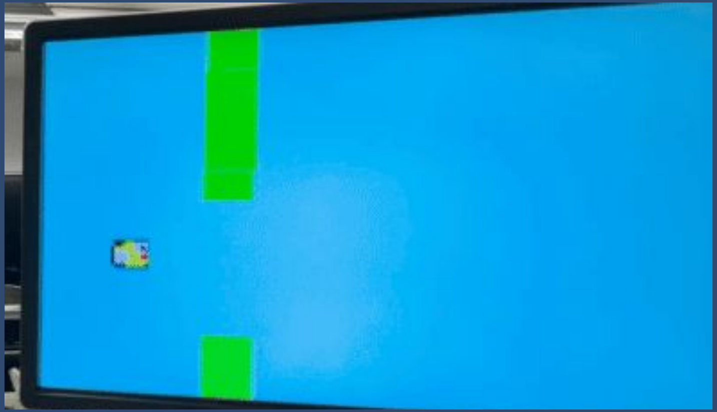
1. Pillar generation
2. Bird movement
3. Collision detection
4. Score tracking

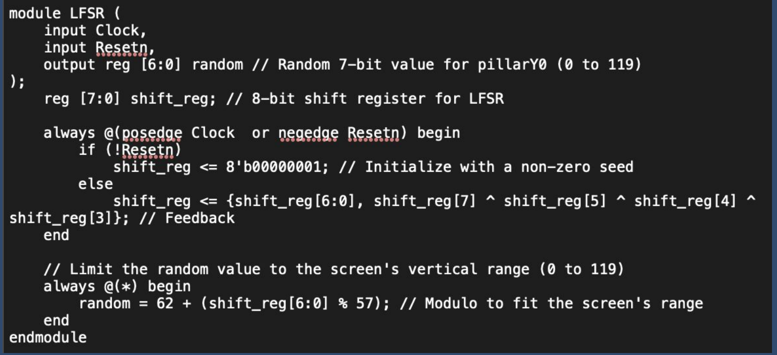
- NOTE: Professor Brown's FSM_Move file was used as a base for the bird and pillar development

1. Pillar Generation

Highlights

- Initial Position
  - The pillar's initial X-coordinate (pillarX0) is set to the rightmost position of the screen.
  - The initial Y-coordinate (pillarY0) is randomly generated using an LFSR to create varying gap sizes.
  - The pillar is drawn pixel by pixel, with the FSM controlling the drawing process and the VGA controller displaying the pixels on the screen.
- Pillar Movement
  - The pillar moves leftward across the screen using a counter.
  - When the pillar reaches the left edge of the screen, its X-coordinate is reset to the rightmost position, and a new random Y-coordinate is generated.
- Randomized pillar logic
  - Pseudo-random pillar gaps generated by wrapping a vertical line around the top and bottom boundary

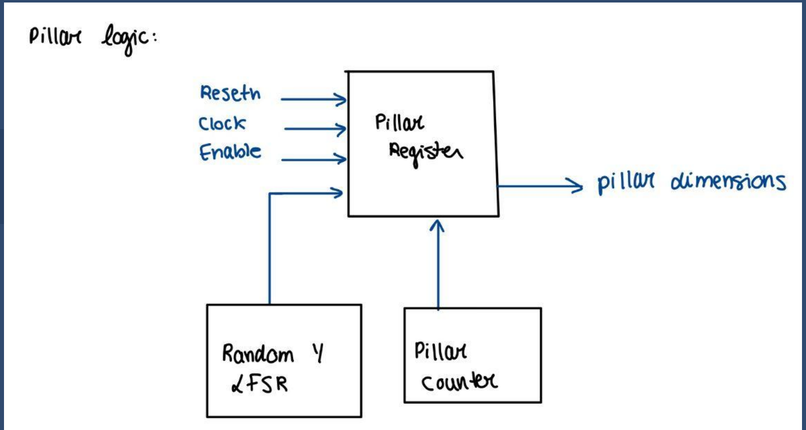


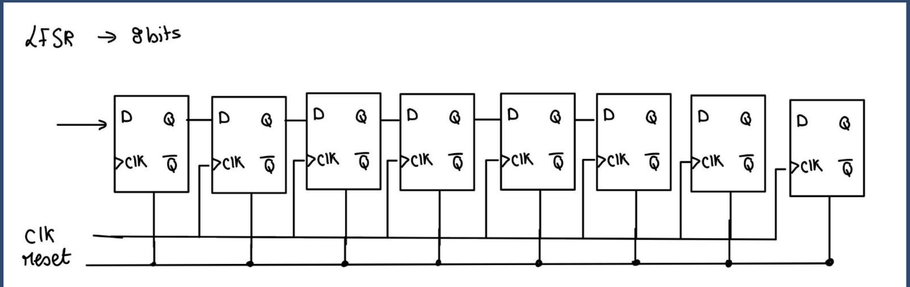


```
module LFSR (
    input Clock,
    input Resetn,
    output reg [6:0] random // Random 7-bit value for pillarY0 (0 to 119)
);
    reg [7:0] shift_reg; // 8-bit shift register for LFSR

    always @(posedge Clock  or negedge Resetn) begin
        if (!Resetn)
            shift_reg <= 8'b00000001; // Initialize with a non-zero seed
        else
            shift_reg <= {shift_reg[6:0], shift_reg[7] ^ shift_reg[5] ^ shift_reg[4] ^
shift_reg[3]}; // Feedback
    end

    // Limit the random value to the screen's vertical range (0 to 119)
    always @(*) begin
        random = 62 + (shift_reg[6:0] % 57); // Modulo to fit the screen's range
    end
endmodule
```

Pillar Logic & Randomization

Main logic behind pillar module development





General idea behind pseudo-random pillar generation

2. Bird Movement & User Input

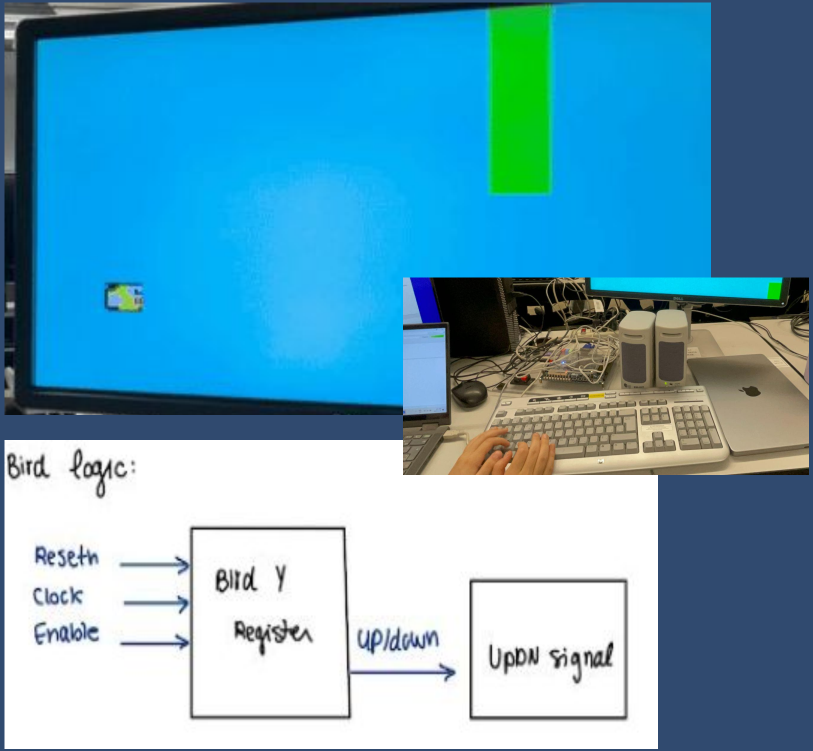
Highlights

Memory Management

- **Memory Block:** The bird is read from a ROM module containing pixel data.
- **Color Mapping:** The pixel colors are loaded into the VGA controller for rendering on the screen.
- The current X and Y coordinates of the bird are used to calculate the address in the ROM.

Movement Control

- **Position Management:**
  - bird's initial position is defined at a fixed (X, Y) coordinate
  - Controlled by a register to track its vertical position (birdY).
- The bird's movement is updated every clock cycle
- The user's input (up or down) is captured through the keyboard.



3. Collision Detection

Highlights

<u>Collision Logic</u>

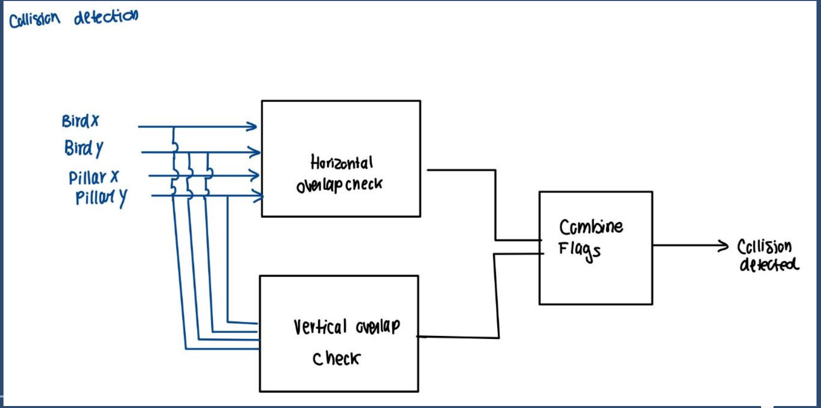
- checking if the bird's X and Y coordinates, (including width, height) overlap with the pillar's X and Y coordinates (including width, height)
- If a collision is detected, the FSM transitions to the game over state (state M)

<u>Game Reset</u>

- All game elements (bird position, pillar positions, and score) are reset to their **initial states**.
- The game_reset signal (KEY[0]) is used to reset the counters and registers associated with the bird, pillars, and other game elements.

<u>Module Calls</u>

- UpDn_count: game_reset signal is connected to the *Resetn* input of these modules, ensuring that the counters are reset when the game restarts
- Regn module: store the bird's initial X-coordinate. The game_reset signal is connected to the *Resetn* input to reset the bird's initial position.



4. Score Tracking

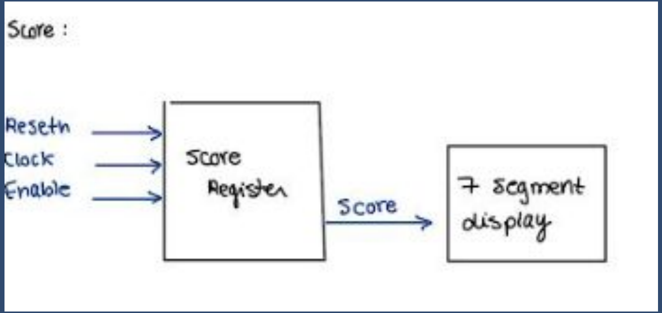
Highlights

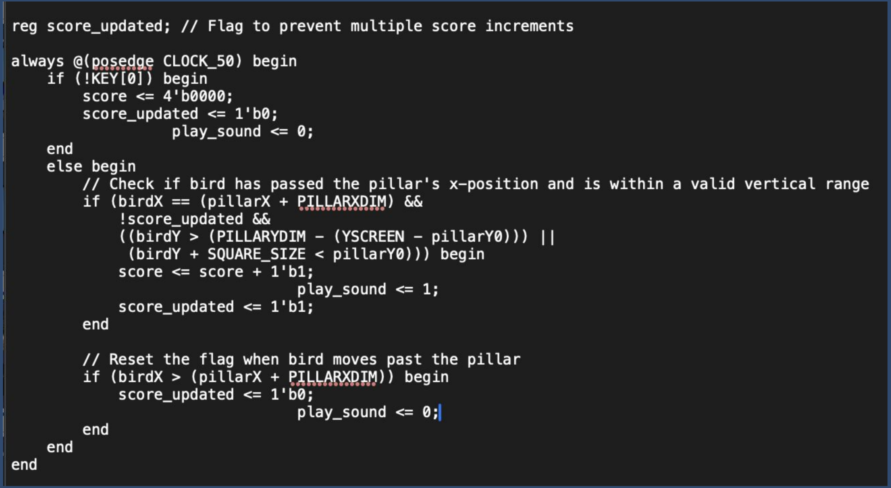
Method: 4-bit register score is used to store the current score.

Debugging: flag *score_updated* is used to prevent multiple score increments for a single point.

- When a point is scored, the score_updated flag is set to 1.

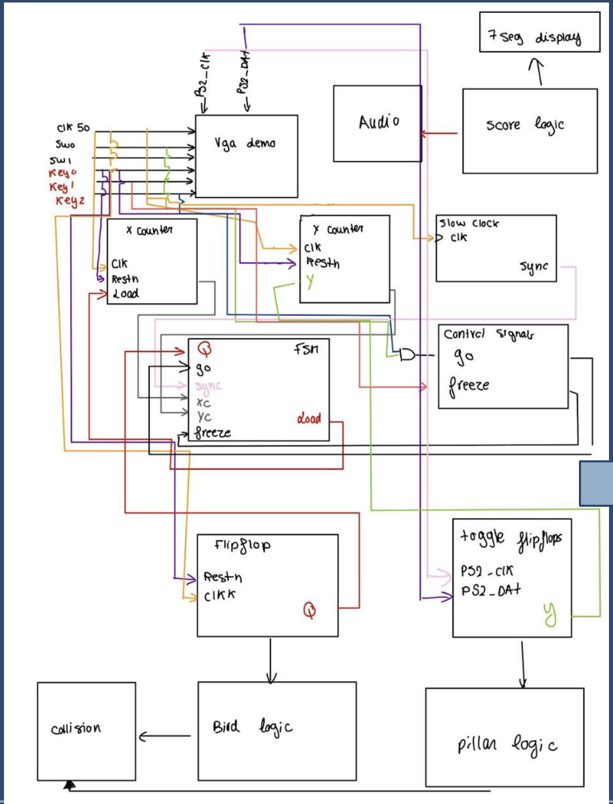
- Continuously update score based on pillar's position with respect to the bird
- When KEY[0] is clicked to replay, the score is re-initialized to 0 and restarts
- Utilizes audio component (magnified by 10) to indicate increase in score

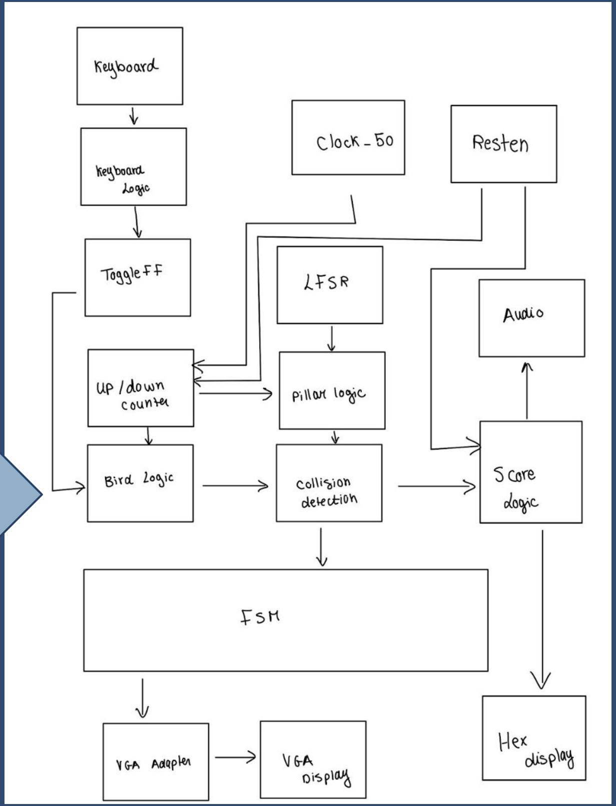




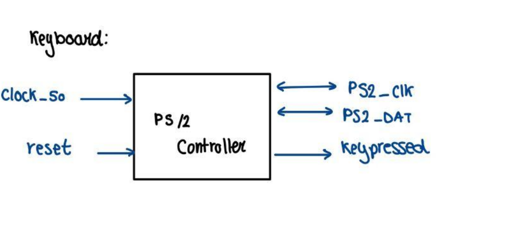
```
reg score_updated; // Flag to prevent multiple score increments

always @(posedge CLOCK_50) begin
    if (!KEY[0]) begin
        score <= 4'b0000;
        score_updated <= 1'b0;
                  play_sound <= 0;
    end
    else begin
        // Check if bird has passed the pillar's x-position and is within a valid vertical range
        if (birdX == (pillarX + PILLARXDIM) &&
            !score_updated &&
            ((birdY > (PILLARYDIM - (YSCREEN - pillarY0))) ||
             (birdY + SQUARE_SIZE < pillarY0))) begin
            score <= score + 1'b1;
                                play_sound <= 1;
            score_updated <= 1'b1;
        end

        // Reset the flag when bird moves past the pillar
        if (birdX > (pillarX + PILLARXDIM)) begin
            score_updated <= 1'b0;
                                play_sound <= 0;
        end
    end
end
```

High-Level Block Diagram

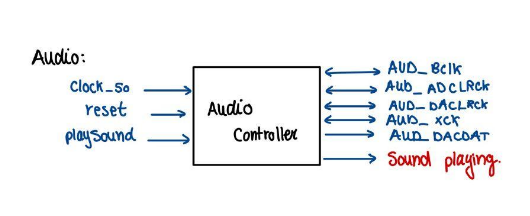




Keyboard & Audio



- Input signals from PS2_CLK & PS2_DAT
- PS2 controller decoded the outputs
    - PS2_KEY_DATA : which key is pressed
    - PS2_key_pressed : when is a key pressed
- Signals are then fed into the FSM

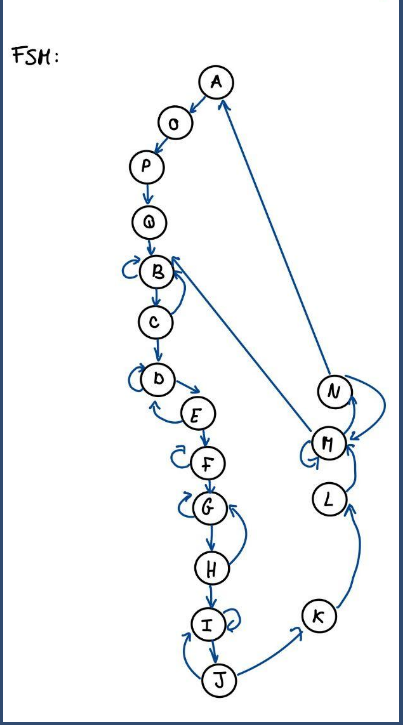


- Adds sounds effects everytime the score increments
    - Score +1 , then play_sound signal is triggered

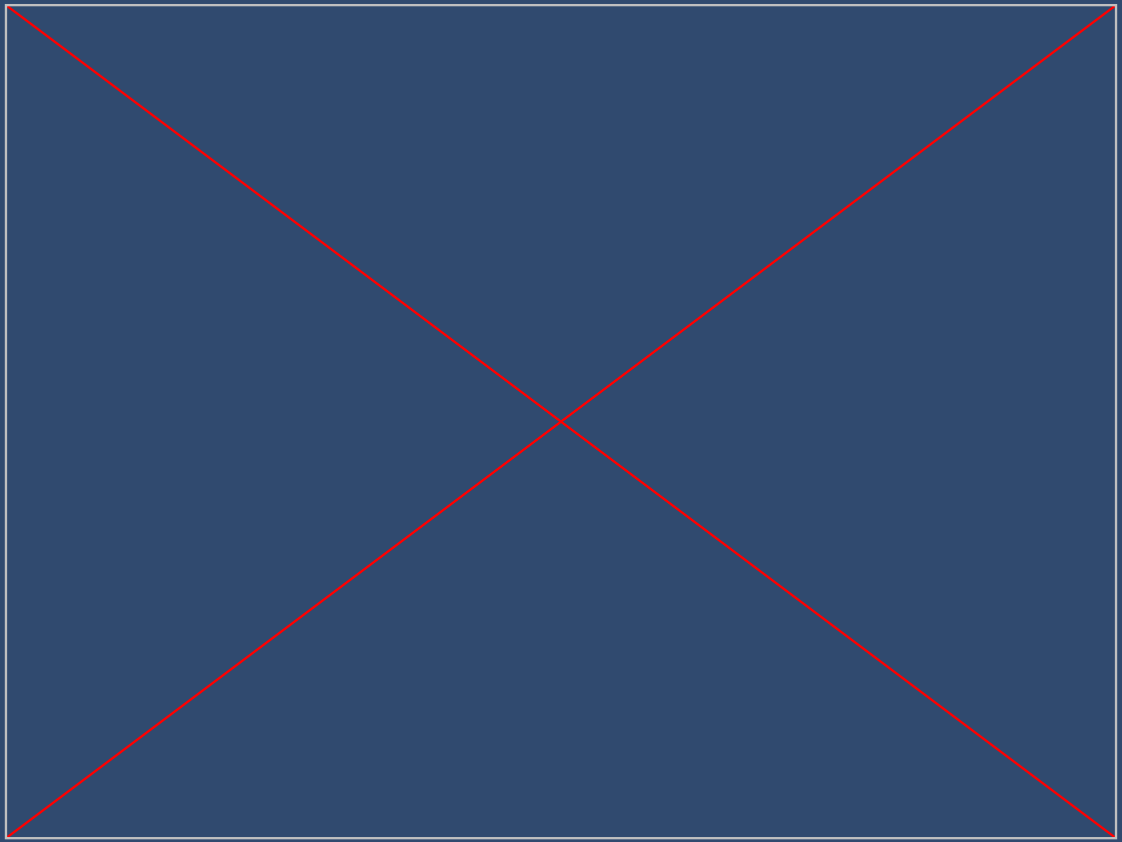
Finite State Machine

A	Initial State
B	Drawing Birds pixels
C	Move to the next row of the bird and draw
D	Drawing Pillar
E	Pillar drawing move to the next row
F	Wait for Sync Signal
G	Erase bird pixel by pixel
H	Erase the birds pixels from the next row
I	Erase pillars row by row

J	Erase the pillar's pixels from the next row
K	Updates the game state
L	Collision handling state
M	Draw the "Game Over" screen
N	Draw the next row of the "Game Over" screen
O	Erase the "Game Over" screen
P	Erases the "Game Over" screen from the next row
Q	Resets game



Demonstration



Bugs and Issues

<u>Merging Bird and Pillar Codes</u>

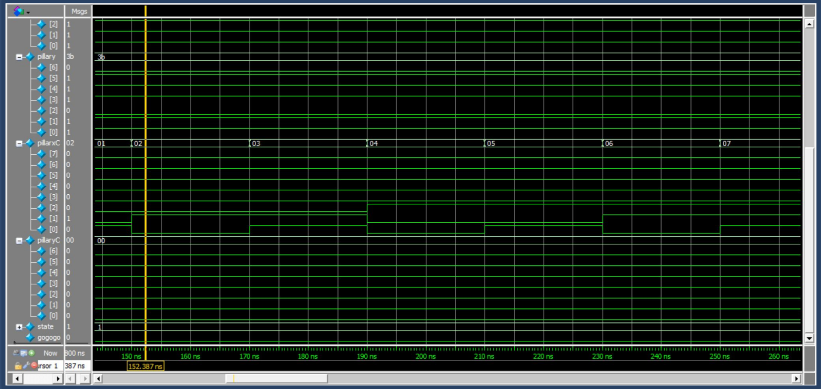
- <u>Issue</u>: Integrating bird movement and pillar generation logic caused synchronization issues
- <u>Solution</u>: Introduced a single clock-driven FSM to synchronize all game elements

<u>Multiple VGA Ports Error</u>

- <u>Issue:</u> Error occurred due to multiple drivers trying to access the VGA output simultaneously.
- <u>Solution</u>: Ensured only one VGA controller instance was active in the final design

<u>Collision Detection Issues</u>

- <u>Issue:</u> first iteration of collision logic caused false positives or missed detections.
- <u>Solution:</u> debugged logic using ModelSim simulations and LEDR feedback for real-time visualization.



Future Steps

- Introduce increase in game difficulty
  - Increase speed at which pillars appear
  - Smaller gaps between pillars

- Call objects from memory for cleaner VGA
  - Call a flappy bird with moving wings using ROM memory for cleaner graphics and gameplay
  - Introduce dynamic background elements (ex. moving clouds, pillar pattern)

- Multiplayer system
  - Introduce another bird and more keys on keyboard to allow for multiplayer gameplay
  - Multiple lives or duels

Work Distribution

Laila	Margaret	Manahil
● Bird implementation (Movement and VGA) ● Keyboard component ● Audio component ● Game start screen VGA	● Bird implementation ● Game collision and reset logic ● Combine bird and pillar VGA ● End screen	● Pillar implementation - Randomized, wrap reset ● Collision detection logic ● Score implementation ● Game reset logic