Final Project
Finite State Machines:
An Adventure Game
ELEE 2640

Benyamain Yacoob

April 23, 2024

Date Performed:	April 5, 2024
Partners:	Ara Oladipo
	Andre Price
Instructor:	Professor Paulik

Contents

1 Objectives

First Objective
Explore the development of finite state machines (FSM).

Second Objective
Implement FSMs in SystemVerilog.

Third Objective
Simulate a simple adventure game.

2 Problem Statement

In this simulation exercise, you will design a finite state machine (FSM) that implements an adventure game! You will then enter the FSM into the SystemVerilog editor in Vivado, then simulate it.

3 Materials

Xilinx integrated synthesis environment (ISE)

4 Exercise #1: Room FSM Completion

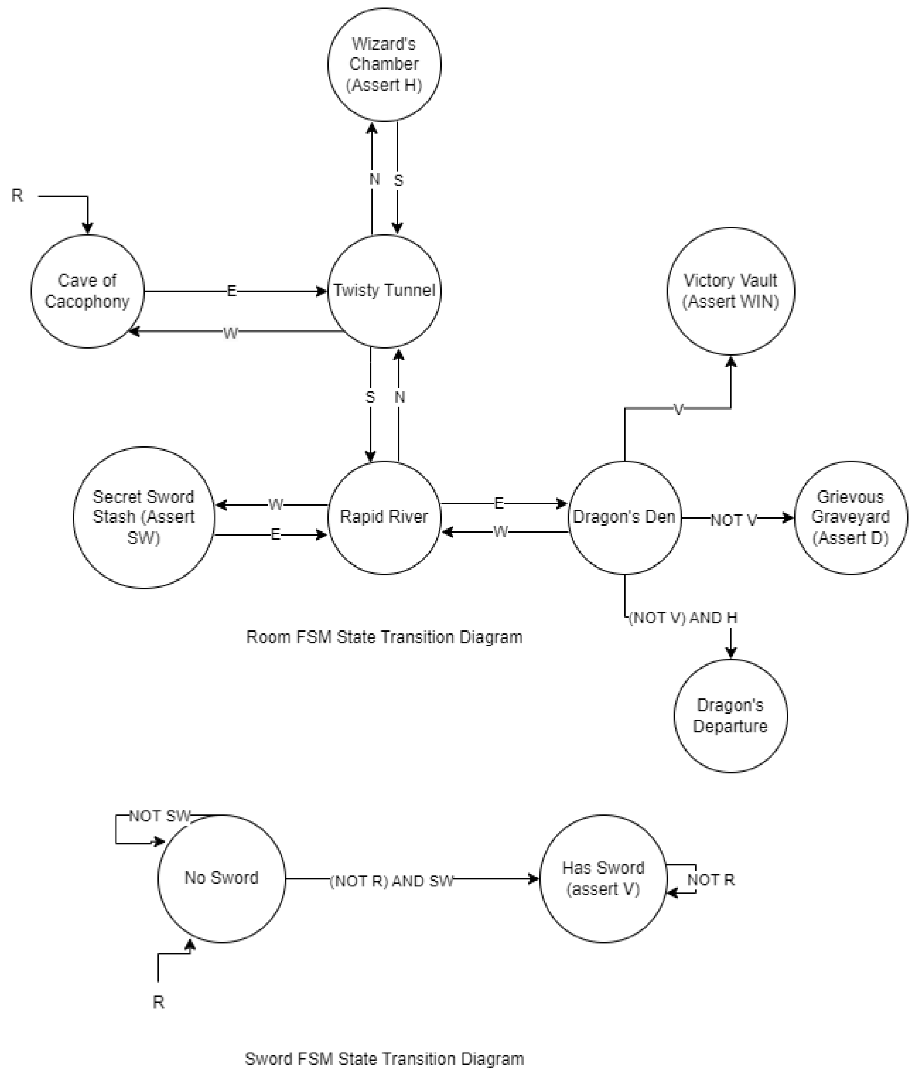


Figure 1: Completed FSM Transition Diagram

4.1 Results and Analysis Discussion

The completed state transition diagram for the Room FSM provides a visual representation of the game flow and the possible transitions between different rooms. It helps in understanding the overall structure of the game and how the player can navigate through various locations. The diagram showcases the interconnectivity of the rooms and the conditions required to move from one room to another. The state transition diagram serves as a blueprint for implementing the game logic. It helps in designing the FSM module and defining the states and transitions accurately.

5 Exercise #2: Input/Output Enumeration, Table Generation

You can find the work for this section here.

5.1 Room FSM

States	
Cave of cacophony	S1
Twisty tunnel	S2
Rapid River	S3
Secret sword stash	S4
Dragon's den	S5
Victory vault	S6
Grievious graveyard	S7
Dragon's departure	S8
Wizard chamber	S9

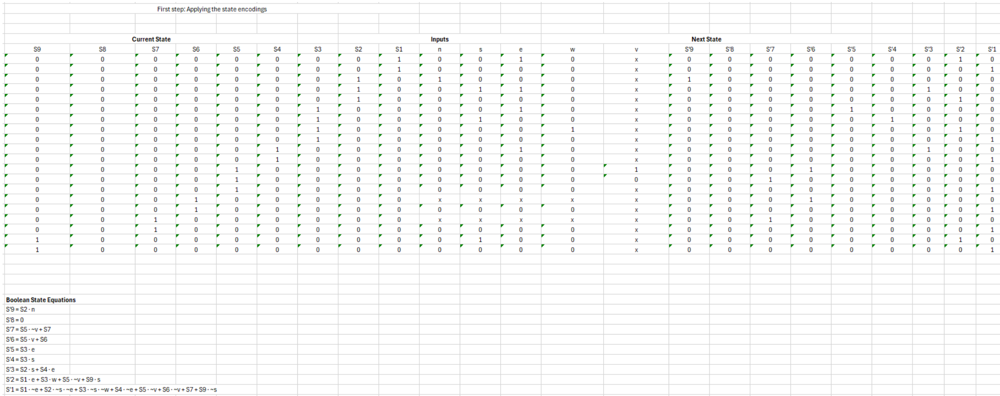
Figure 2: Room FSM States

Room FSM								
	Inputs							
Current State	n	s	e	w	v	Reset	Next State	Output
S1	0	0	1	0	x	0	S2	T
S1	0	0	0	0	x	1	S1	C
S2	1	0	0	0	x	0	S9	W
S2	0	1	1	0	x	0	S3	R
S2	0	0	0	0	x	1	S1	C
S3	0	0	1	0	x	0	S5	D
S3	0	1	0	0	x	0	S4	S
S3	0	0	0	1	x	0	S2	T
S3	0	0	0	0	x	1	S1	C
S4	0	0	1	0	x	0	S3	R
S4	0	0	0	0	x	1	S1	C
S5	0	0	0	0	1	0	S6	V
S5	0	0	0	0	0	0	S7	G
S5	0	0	0	0	x	1	S1	C
S6	x	x	x	x	x	0	S6	WIN
S6	0	0	0	0	x	1	S1	C
S7	x	x	x	x	x	0	S7	D
S7	0	0	0	0	x	1	S1	C
S9	0	1	0	0	x	0	S2	T
S9	0	0	0	0	x	1	S1	C

Figure 3: Room FSM State Transition Diagram

	State Encodings (One-Hot)
S1	000000001
S2	000000010
S3	000000100
S4	000001000
S5	000010000
S6	000100000
S7	001000000
S8	010000000
S9	100000000

Figure 4: One-Hot Encoding



First step: Applying the state encodings

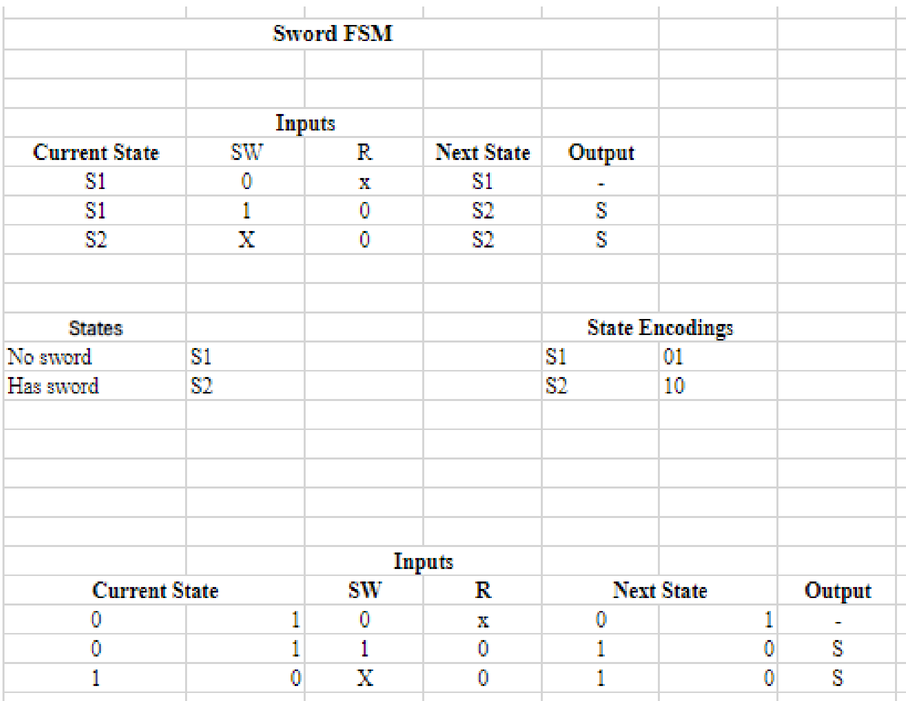
Current State									Inputs					Next State								
S9	S8	S7	S6	S5	S4	S3	S2	S1	n	s	e	w	v	S'9	S'8	S'7	S'6	S'5	S'4	S'3	S'2	S'1
0	0	0	0	0	0	0	0	1	0	0	1	0	x	0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	0	0	1	0	0	0	0	x	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	1	0	1	0	0	0	x	1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0	1	1	0	x	0	0	0	0	0	0	1	0	0
0	0	0	0	0	0	0	1	0	0	0	0	0	x	0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	0	0	0	0	1	0	x	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	1	0	0	0	1	0	0	x	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0	0	0	0	0	1	x	0	0	0	0	0	0	0	1	0
0	0	0	0	0	0	1	0	0	0	0	0	0	x	0	0	0	0	0	0	0	0	1
0	0	0	0	0	1	0	0	0	0	0	1	0	x	0	0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	0	0	0	0	0	0	x	0	0	0	0	0	0	0	0	1
0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	0	0	0	0	x	0	0	0	0	0	0	0	0	1
0	0	0	1	0	0	0	0	0	x	x	x	x	x	0	0	0	1	0	0	0	0	0
0	0	0	1	0	0	0	0	0	0	0	0	0	x	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	x	x	x	x	x	0	0	1	0	0	0	0	0	0
0	0	1	0	0	0	0	0	0	0	0	0	0	x	0	0	0	0	0	0	0	0	1
1	0	0	0	0	0	0	0	0	0	1	0	0	x	0	0	0	0	0	0	0	1	0
1	0	0	0	0	0	0	0	0	0	0	0	0	x	0	0	0	0	0	0	0	0	1

Boolean State Equations

S'9 = S2 · n
S'8 = 0
S'7 = S5 · ~v + S7
S'6 = S5 · v + S6
S'5 = S3 · e
S'4 = S3 · s
S'3 = S2 · s + S4 · e
S'2 = S1 · e + S3 · w + S5 · ~v + S9 · s
S'1 = S1 · ~e + S2 · ~s · ~e + S3 · ~s · ~w + S4 · ~e + S5 · ~v + S6 · ~v + S7 + S9 · ~s

Figure 5: Encoded State Transition Diagram and Generated Equations

5.2 Sword FSM



Sword FSM

	Inputs			
Current State	**SW**	**R**	**Next State**	**Output**
S1	0	x	S1	-
S1	1	0	S2	S
S2	X	0	S2	S

States		**State Encodings**	
No sword	S1	S1	01
Has sword	S2	S2	10

		Inputs				
Current State		**SW**	**R**	**Next State**		**Output**
0	1	0	x	0	1	-
0	1	1	0	1	0	S
1	0	X	0	1	0	S

Figure 6: Sword FSM State Transition Table and State Encodings

5.3 Results and Analysis Discussion

This exercise focuses on identifying the inputs and outputs of the Room and Sword FSMs, creating tables to represent the state transitions, and deriving Boolean logic equations. It involves an approach to defining the behavior of the FSMs on the basis of the current state and input conditions. We enumerated the inputs and outputs for each FSM. We did this using Excel. We used X to denote "don't cares", as the directions stated that we do not need to fill in every possible combination of values for all our inputs. Enumerating the inputs and outputs helps in establishing a clear interface for the FSMs.

6 Exercise #3: Individual FSM Modules and Testbenches

In this exercise, the focus is on implementing the individual FSM modules (Room FSM and Sword FSM) using the three always block idiom in SystemVerilog. It also involves creating testbenches to verify the functionality of each FSM module independently. The top-level module serves as the integration point for the individual FSMs and facilitates communication between them. The testbench allows for testing the entire game system and verifying the interactions between the FSMs.

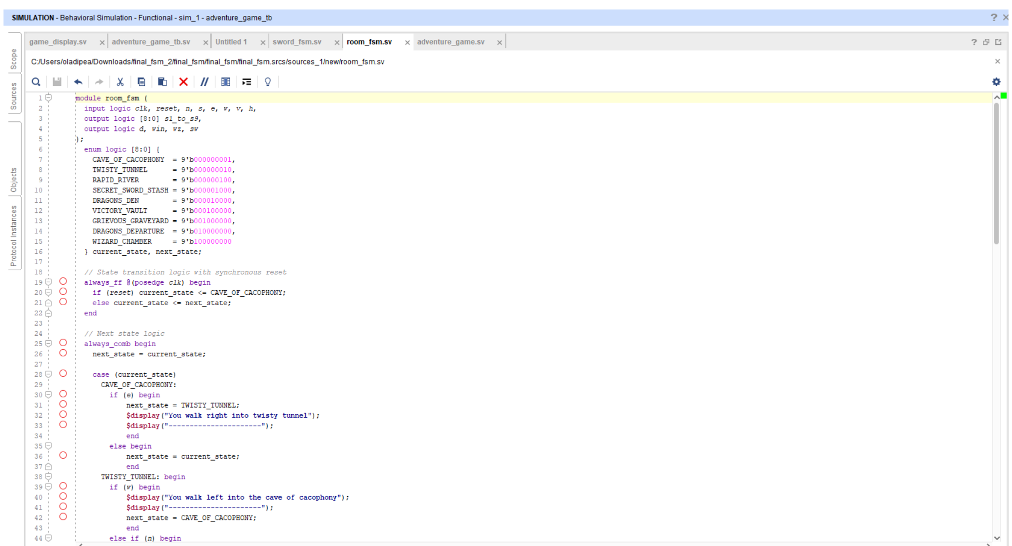


Figure 7: Room FSM Code #1

```
            end
        else if (n) begin
            $display("You walk up and find the wizard's chamber");
            $display("You find a wizard; He has been expecting you, The wizard looks up to you and tells you a secret about the dragon");
            $display("You find out the dragon is your long lost wife, cursed because of her cheating ways....");
            $display("----------------------");
            next_state = WIZARD_CHAMBER;
            end
        else if (s) begin
            $display("You walk down to find a rapid river");
            $display("----------------------");
            next_state = RAPID_RIVER;
            end
        else
            next_state = current_state;
      end
      RAPID_RIVER:
        if (n) begin
            $display("You walk up to the twisty tunnel");
            $display("----------------------");
            next_state = TWISTY_TUNNEL;
            end
        else if (w) begin
            $display("You walk left and find a secret sword stash! You are now prepared to face the evil dragon...or so you hear!");
            $display("----------------------");
            next_state = SECRET_SWORD_STASH;
            end
        else if (e) begin
            next_state = DRAGONS_DEN;
            end
        else
            next_state = current_state;
      SECRET_SWORD_STASH:
        if (e)
        begin
            next_state = RAPID_RIVER;
            $display("You return to the rapid rivers.");
            $display("----------------------");
        end
        else
            next_state = current_state;
      WIZARD_CHAMBER:
        if (s) begin
            $display("You return to move down to the twisty tunnel.");
```

Figure 8: Room FSM Code #2

game_display.sv × | adventure_game_tb.sv × | Untitled 1 × | sword_fsm.sv × | room_fsm.sv * × | adventure_game.sv ×

C:/Users/oladipea/Downloads/final_fsm_2/final_fsm/final_fsm/final_fsm.srcs/sources_1/new/room_fsm.sv

```
            $display("You return to move down to the twisty tunnel.");
            $display("----------------------");
            next_state = TWISTY_TUNNEL;
            end
        else
            next_state = current_state;
      DRAGONS_DEN: begin
        $display("You confront the dragon! It is time for the final face-off");
        if  (h)
            next_state = DRAGONS_DEPARTURE;
        else if (v) begin
            next_state = VICTORY_VAULT;
            end
        else if (~v) begin
            next_state = GRIEVOUS_GRAVEYARD;
        end
        else next_state = current_state;
      end
      DRAGONS_DEPARTURE: begin
            $display("You face your cheating wife (now dragon) and show her how much of a chad you are now.");
            $display("She flees in sadness, jealousy, and beta-ness");
            $display("----------------------");
            end
      VICTORY_VAULT: begin
            $display("You slay the dragon!!! You take one of it's horns as it is a well-known aphrodesiac, and head over to the victory vault");
            end
      GRIEVOUS_GRAVEYARD: begin
            $display("You come unprepared to the battle, so you ended up getting killed...by snu snu");
            $display("You may start again");
            $display("----------------------");
      end
      default: next_state = CAVE_OF_CACOPHONY;
    endcase
  end

  //Output logic block
  always_comb
  begin
    sw = sw | (current_state == SECRET_SWORD_STASH);
    wz = wz | (current_state == WIZARD_CHAMBER);
    win = win | (current_state == VICTORY_VAULT) | (current_state == DRAGONS_DEPARTURE);
    d = d | (current_state == GRIEVOUS_GRAVEYARD);
    s1_to_s9 = current_state;
  end
```

Figure 9: Room FSM Code #3

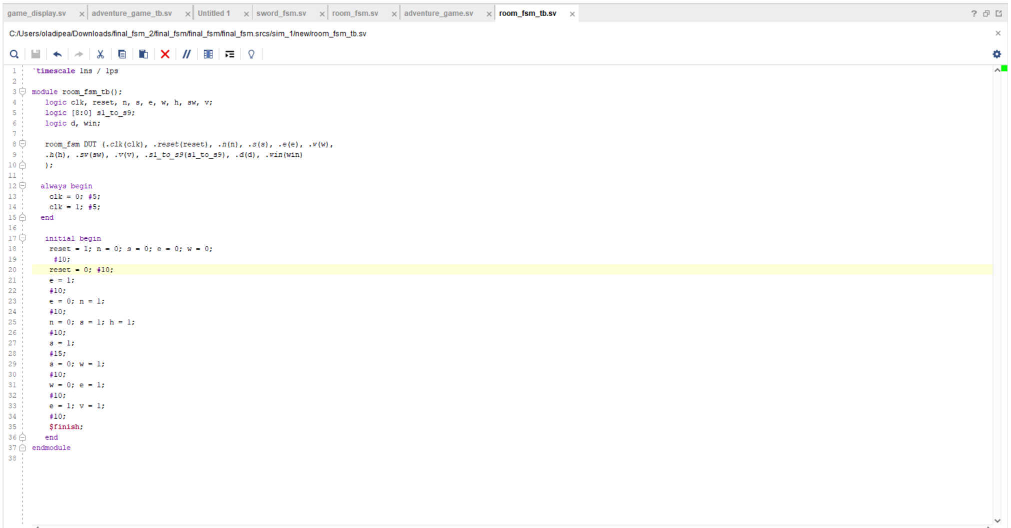


Figure 10: Room FSM Testbench Code

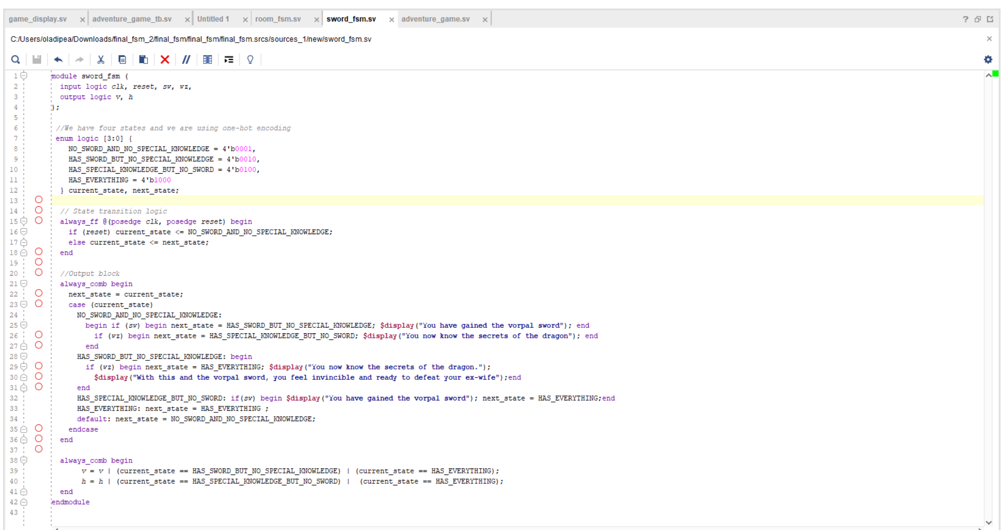


Figure 11: Sword FSM Code

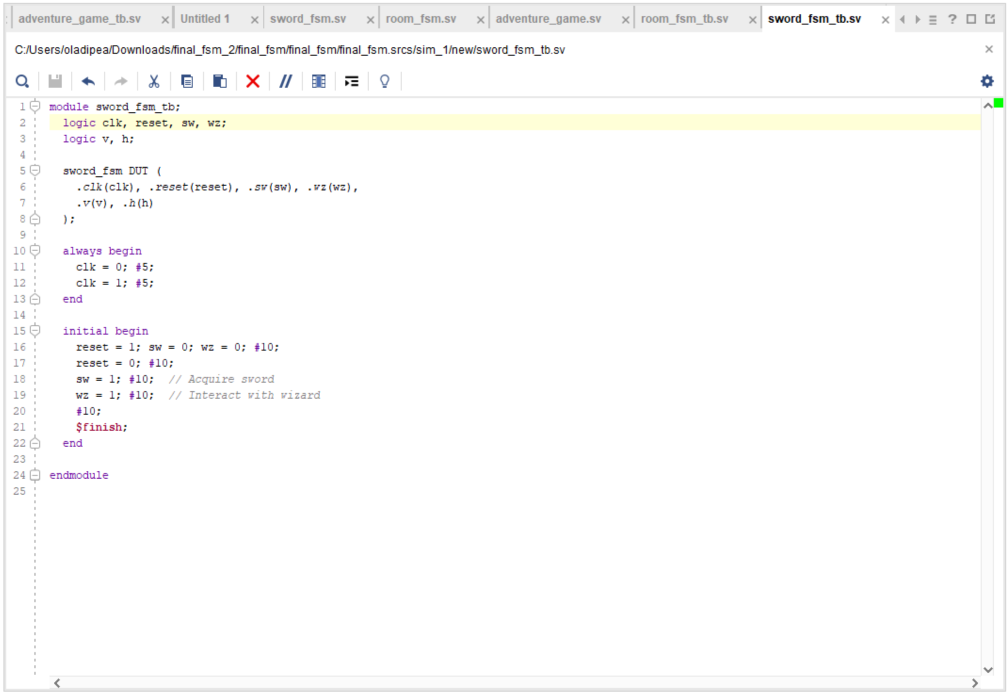


Figure 12: Sword FSM Testbench Code

6.1 Results and Analysis Discussion

The implementation of the individual FSM modules (Room FSM and Sword FSM) using the three always block idiom in SystemVerilog resulted in a structured and modular design. The three always blocks, namely the always_ff block for the state register, the always_comb block for the next state logic, and another always_comb block for the output logic, provided a clear separation of concerns and made the code more readable and maintainable.

The Room FSM module successfully captured the behavior of the game's room transitions based on the player's actions. It accurately represented the states and transitions defined in the state transition diagram, ensuring that the player could navigate through the game world as intended. The Sword FSM module, on the other hand, effectively managed the state of the sword, determining whether the player had acquired the sword or not and handling the special ability granted by the wizard.

To verify the functionality of each FSM module, testbenches were created. The testbenches simulated various scenarios and input sequences to thoroughly test the behavior of the FSMs. By providing controlled inputs and observing the output, the testbenches helped to identify any discrepancies between the expected and actual behavior of the modules.

7 Exercise #4: Joint High-Level FSM Module and Testbench

The game consists of two communicating FSMs: the Room FSM, which keeps track of the player's current location, and the Sword FSM, which keeps track of whether the player has acquired the sword. The objective of the game is to navigate through various rooms, find the Vorpal Sword, and reach the Victory Vault while avoiding the dragon in the Dragon Den.

The provided code snippet shows the implementation of the high-level adventure_game module, which instantiates the room_fsm and sword_fsm modules. It also includes a game_display module for displaying the game state on the 7-segment displays. The adventure_game module takes inputs such as the clock, reset, and directional controls (n, s, e, w) and outputs the current room state, win, and dead signals.

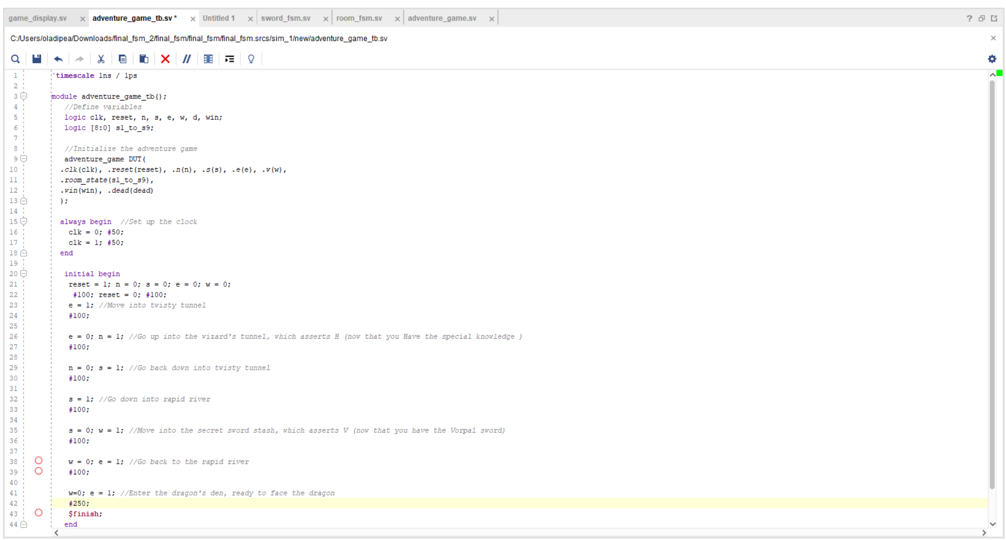


Figure 13: Adventure Game Testbench Code

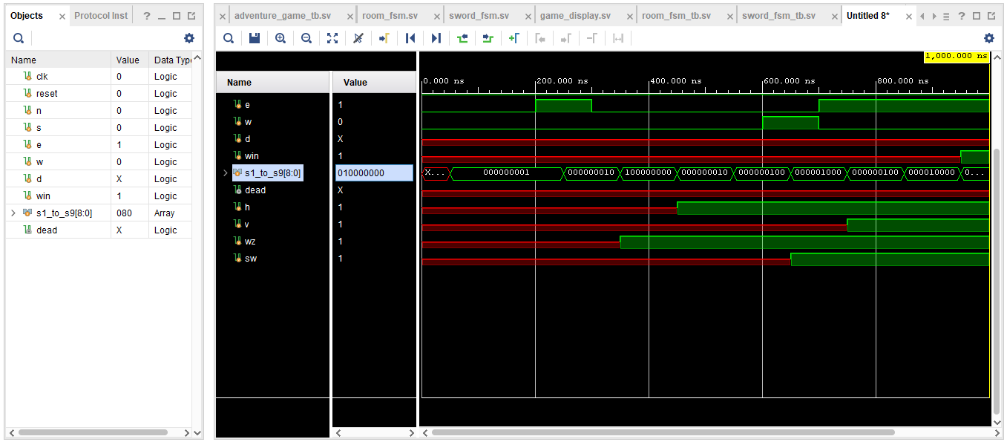
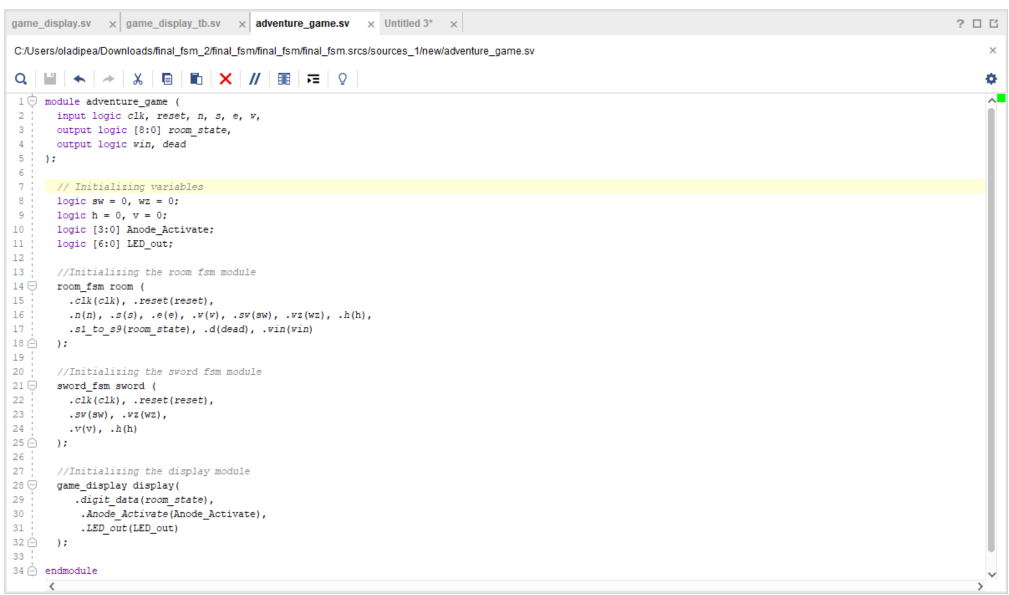


Figure 14: Adventure Game Waveform



game_display.sv | game_display_tb.sv | adventure_game.sv | Untitled 3*

C:/Users/oladipea/Downloads/final_fsm_2/final_fsm/final_fsm/final_fsm.srcs/sources_1/new/adventure_game.sv

```
module adventure_game (
  input logic clk, reset, n, s, e, w,
  output logic [8:0] room_state,
  output logic win, dead
);

  // Initializing variables
  logic sw = 0, wz = 0;
  logic h = 0, v = 0;
  logic [3:0] Anode_Activate;
  logic [6:0] LED_out;

  //Initializing the room fsm module
  room_fsm room (
    .clk(clk), .reset(reset),
    .n(n), .s(s), .e(e), .w(w), .sw(sw), .wz(wz), .h(h),
    .s1_to_s9(room_state), .d(dead), .win(win)
  );

  //Initializing the sword fsm module
  sword_fsm sword (
    .clk(clk), .reset(reset),
    .sw(sw), .wz(wz),
    .v(v), .h(h)
  );

  //Initializing the display module
  game_display display(
     .digit_data(room_state),
      .Anode_Activate(Anode_Activate),
      .LED_out(LED_out)
  );

endmodule
```

Figure 15: Adventure Game Code

7.1 Results and Analysis Discussion

The adventure_game module serves as the top level module, connecting the Room FSM, Sword FSM, and Game Display Modules. It provides a clean interface for the game inputs and outputs.

The room_fsm and sword_fsm modules are instantiated within the adventure_game module, allowing them to communicate and share relevant signals such as sw (sword acquired) and wz (wizard's tunnel). This modular design promotes code reusability and maintainability.

The game_display module is responsible for driving the 7-segment displays to show the current state of the game. It takes the room_state as input and outputs the appropriate signals for the display. The testbench code covers a specific scenario in which the player moves through the rooms, acquires the sword, and faces the dragon in the Dragon Den. It demonstrates the expected behavior of the game logic and helps verify the correctness of the FSM implementations.

The joint high-level FSM module brings together the individual FSMs and forms the backbone of the game system. It enables coordination and synchronization between the Room and Sword FSMs.

8 Exercise #5: Multiplexed Display and Modified Top-Level Module

This exercise focuses on implementing a multiplexed display module to control 7-segment displays and LEDs to display game information. It also involves modifying the top-level module to incorporate the display functionality and creating a test-bench to simulate the gameplay with the display outputs.

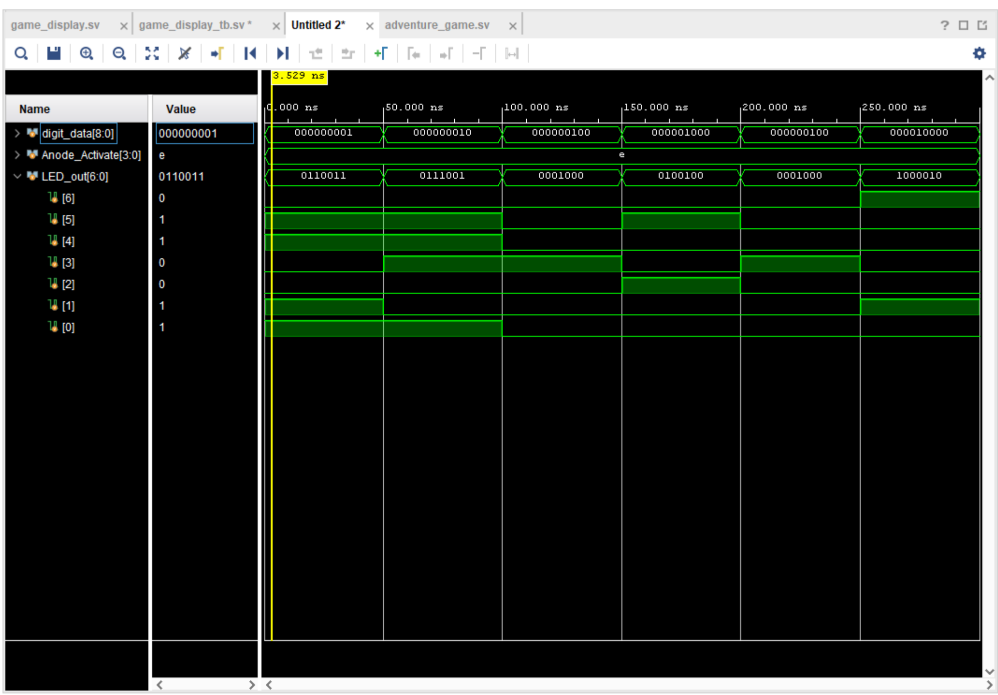
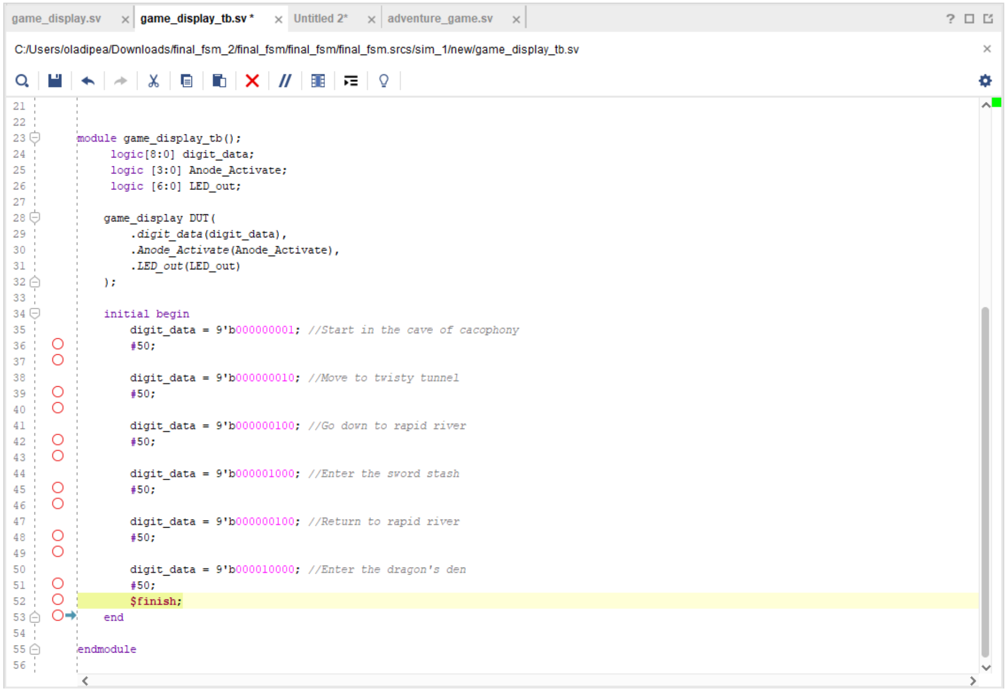


Figure 16: Game Display Waveform

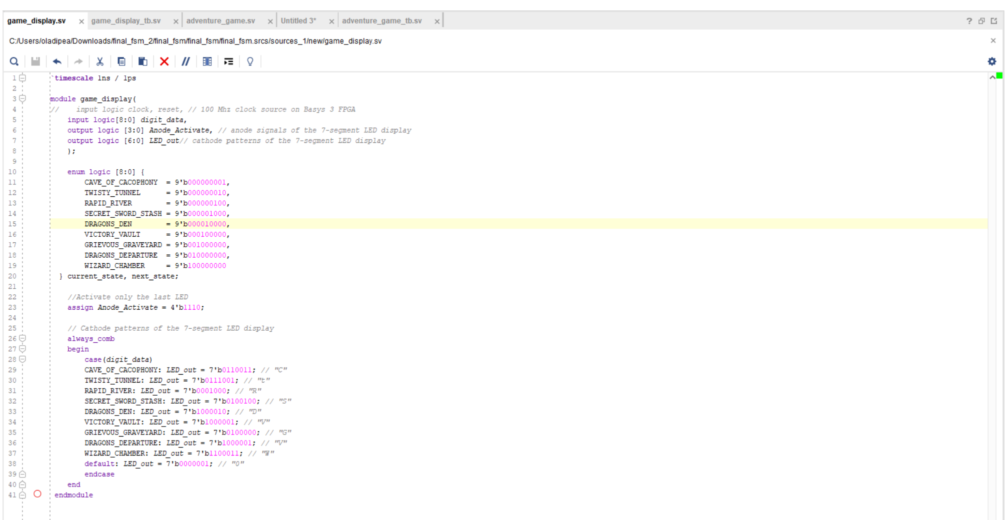


game_display.sv | game_display_tb.sv * | Untitled 2* | adventure_game.sv

C:/Users/oladipea/Downloads/final_fsm_2/final_fsm/final_fsm/final_fsm.srcs/sim_1/new/game_display_tb.sv

```
module game_display_tb();
    logic[8:0] digit_data;
    logic [3:0] Anode_Activate;
    logic [6:0] LED_out;

  game_display DUT(
      .digit_data(digit_data),
      .Anode_Activate(Anode_Activate),
      .LED_out(LED_out)
  );

  initial begin
      digit_data = 9'b000000001; //Start in the cave of cacophony
      #50;

      digit_data = 9'b000000010; //Move to twisty tunnel
      #50;

      digit_data = 9'b000000100; //Go down to rapid river
      #50;

      digit_data = 9'b000001000; //Enter the sword stash
      #50;

      digit_data = 9'b000000100; //Return to rapid river
      #50;

      digit_data = 9'b000010000; //Enter the dragon's den
      #50;
      $finish;
  end

endmodule
```

Figure 17: Game Display Testbench Code



game_display.sv | game_display_tb.sv | adventure_game.sv | Untitled 3* | adventure_game_tb.sv

C:/Users/oladipea/Downloads/final_fsm_2/final_fsm/final_fsm/final_fsm.srcs/sources_1/new/game_display.sv

```
`timescale 1ns / 1ps

module game_display(
//    input logic clock, reset, // 100 Mhz clock source on Basys 3 FPGA
    input logic[8:0] digit_data,
    output logic [3:0] Anode_Activate, // anode signals of the 7-segment LED display
    output logic [6:0] LED_out// cathode patterns of the 7-segment LED display
    );

    enum logic [8:0] {
        CAVE_OF_CACOPHONY  = 9'b000000001,
        TWISTY_TUNNEL      = 9'b000000010,
        RAPID_RIVER        = 9'b000000100,
        SECRET_SWORD_STASH = 9'b000001000,
        DRAGONS_DEN        = 9'b000010000,
        VICTORY_VAULT      = 9'b000100000,
        GRIEVOUS_GRAVEYARD = 9'b001000000,
        DRAGONS_DEPARTURE  = 9'b010000000,
        WIZARD_CHAMBER     = 9'b100000000
  } current_state, next_state;

    //Activate only the last LED
    assign Anode_Activate = 4'b1110;

    // Cathode patterns of the 7-segment LED display
    always_comb
    begin
        case(digit_data)
        CAVE_OF_CACOPHONY: LED_out = 7'b0110011; // "C"
        TWISTY_TUNNEL: LED_out = 7'b0111001; // "t"
        RAPID_RIVER: LED_out = 7'b0001000; // "R"
        SECRET_SWORD_STASH: LED_out = 7'b0100100; // "S"
        DRAGONS_DEN: LED_out = 7'b1000010; // "D"
        VICTORY_VAULT: LED_out = 7'b1000001; // "V"
        GRIEVOUS_GRAVEYARD: LED_out = 7'b0100000; // "G"
        DRAGONS_DEPARTURE: LED_out = 7'b1000001; // "V"
        WIZARD_CHAMBER: LED_out = 7'b1100011; // "W"
        default: LED_out = 7'b0000001; // "0"
        endcase
    end
endmodule
```

Figure 18: Game Display Code

```
# run 1000ns
You walk right into twisty tunnel
----------------------
You walk up and find the wizard's chamber
You find a wizard; He has been expecting you, The wizard looks up to you and tells you a secret about the dragon
You find out the dragon is your long lost wife, cursed because of her cheating ways....
----------------------
You now know the secrets of the dragon
You return to move down to the twisty tunnel.
----------------------
You walk down to find a rapid river
----------------------
```

Figure 19: TCL Console Display

```
----------------------
You walk left and find a secret sword stash! You are now prepared to face the evil dragon...or so you hear!
----------------------
You have gained the vorpal sword
You return to the rapid rivers.
----------------------
You confront the dragon! It is time for the final face-off
You face your cheating wife (now dragon) and show her how much of a chad you are now.
She flees in sadness, jealousy, and beta-ness
----------------------
```

Figure 20: TCL Console Display

8.1 Results and Analysis Discussion

The multiplexed display module adds visual feedback to the game. It allows for displaying relevant game information, such as the current room, sword status, win/lose conditions, and special abilities. The modified top-level module integrates the display functionality with the game logic, providing a complete game system.

9 Results and Conclusion

This section discusses the results of the final project. The discussion provides in-depth explanations of the results and any discrepancies between the results and theoretical expectations. It also explores potential sources of error and discusses the accuracy of the tests, the debugging process, and what was learned from these experiences.

The final project succeeded in requiring the group to work very hard just to complete the code. Although we were able to finish the exercise, it was difficult to get a working code and implementation that met the expectations of the report. We hope that for future reference, more readable tutorials are provided to ensure a final project that is of high quality to all parties involved, whether that be the professor who reads the reports, the teacher assistants who give input, or group members who do the work to the best of their ability.

10 Group Contributions

The work for this simulation assignment was divided among three individuals: Benyamain Yacoob, Andre Price Jr., and Ara Oladipo. Ara and Benyamain collaborated using the Vivado software and writing in SystemVerilog. It should be noted, however, that this was a group effort and all members were present to answer any questions related to the circuits. Andre led the effort to organize the document, format it correctly, and write the analysis for this report.