FPGA-Based Automated Parking System

Digital System Design Project Report

Submitted by:

S B Pranay
Roll No: B24EE045

Department of Electrical Engineering

Indian Institute of Technology Hyderabad

December 3, 2025

Abstract

This report documents the design, implementation, and simulation of an automated parking system controller using Verilog HDL. The system utilizes a Finite State Machine (FSM) to manage vehicle entry, password validation, and visual indicators (LEDs and Seven-Segment Displays). The design was synthesized, and functionality was verified using behavioral simulation waveforms.

1 Introduction

The objective of this project is to design a secure parking entry system. The system detects the presence of a vehicle, requires a two-part password sequence for entry, and manages entry/exit status. The controller is implemented as a synchronous Sequential Logic Circuit.

2 System Architecture

2.1 Finite State Machine (FSM)

The core logic is a Moore Machine with the following states:

- **IDLE (000):** System waits for a car (‘sensor_entrance‘).
- **WAIT_PASSWORD (001):** User has a limited time window to enter credentials.
- **WRONG_PASS (010):** Incorrect password entered; Red LED blinks.
- **RIGHT_PASS (011):** Correct password entered; Green LED blinks, gate opens.
- **STOP (100):** Car has entered but not yet cleared the exit sensor.

2.2 I/O Definition

- **Inputs:** `clk`, `reset_n`, `sensor_entrance`, `sensor_exit`, `password_1`, `password_2`.
- **Outputs:** `GREEN_LED`, `RED_LED`, `HEX_1`, `HEX_2` (7-segment display).

3 RTL Schematic Analysis

The design was synthesized to generate the Register Transfer Level (RTL) and Technology schematics.

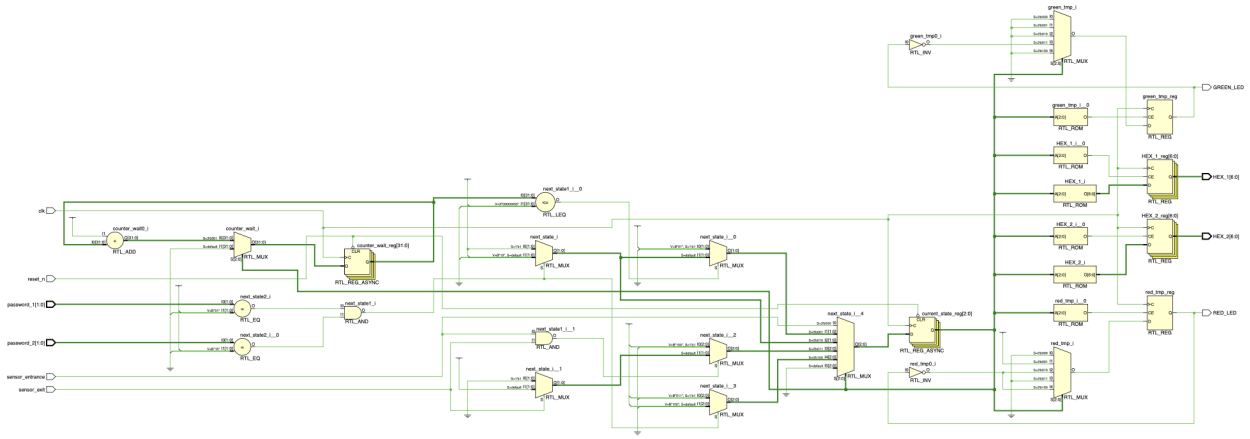

Figure 1: **RTL Schematic:** This view highlights the high-level logic blocks, including the multiplexers for state transitions and comparators for password validation.

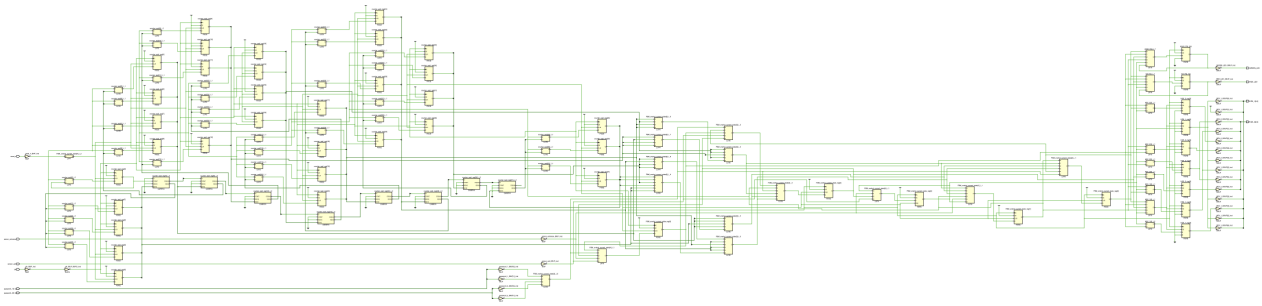

Figure 2: **Technology Schematic:** This detailed view shows the actual gate-level implementation (LUTs, buffers, and flip-flops) post-synthesis.

4 Verilog Implementation

4.1 Main Module: `parking_system.v`

```
`timescale 1ns / 1ps

module parking_system(
    input clk,
    input reset_n,
    input sensor_entrance,
    input sensor_exit,
    input [1:0] password_1,
    input [1:0] password_2,
    output wire GREEN_LED,
    output wire RED_LED,
    output reg [6:0] HEX_1,
    output reg [6:0] HEX_2
);
    // State Encoding
    parameter IDLE = 3'b000,
              WAIT_PASSWORD = 3'b001,
              WRONG_PASS = 3'b010,
              RIGHT_PASS = 3'b011,
              STOP = 3'b100;

```

```
    reg[2:0] current_state, next_state;
    reg[31:0] counter_wait;
    reg red_tmp, green_tmp;

    // 1. Sequential Logic: State Memory
    always @(posedge clk or negedge reset_n) begin
        if(~reset_n)
            current_state <= IDLE;
        else
            current_state <= next_state;
    end

    // 2. Sequential Logic: Wait Counter
    always @(posedge clk or negedge reset_n) begin
        if(~reset_n)
            counter_wait <= 0;
        else if(current_state == WAIT_PASSWORD)
            counter_wait <= counter_wait + 1;
        else
            counter_wait <= 0;
    end

    // 3. Combinational Logic: Next State Logic
    always @(*) begin
        case(current_state)
            IDLE: begin
                if(sensor_entrance == 1) next_state = WAIT_PASSWORD;
                else next_state = IDLE;
            end
            WAIT_PASSWORD: begin
                if(counter_wait <= 3) next_state = WAIT_PASSWORD;
                else begin
                    if((password_1 == 2'b01) && (password_2 == 2'b10))
                        next_state = RIGHT_PASS;
                    else
                        next_state = WRONG_PASS;
                end
            end
            WRONG_PASS: begin
                if((password_1 == 2'b01) && (password_2 == 2'b10))
                    next_state = RIGHT_PASS;
                else
                    next_state = WRONG_PASS;
            end
            RIGHT_PASS: begin
                if(sensor_entrance == 1 && sensor_exit == 1) next_state =
    STOP;
                else if(sensor_exit == 1) next_state = IDLE;
                else next_state = RIGHT_PASS;
            end
            STOP: begin
                if((password_1 == 2'b01) && (password_2 == 2'b10))
    next_state = RIGHT_PASS;
                else next_state = STOP;
            end
            default: next_state = IDLE;
        endcase
    end

    // 4. Output Logic (LEDs and HEX)
```

```
    always @(posedge clk) begin
        case(current_state)
            IDLE: begin
                green_tmp <= 1'b0; red_tmp <= 1'b0;
                HEX_1 <= 7'b1111111; HEX_2 <= 7'b1111111;
            end
            WAIT_PASSWORD: begin
                green_tmp <= 1'b0; red_tmp <= 1'b1;
                HEX_1 <= 7'b000_0110; HEX_2 <= 7'b010_1011;
            end
            WRONG_PASS: begin
                green_tmp <= 1'b0; red_tmp <= ~red_tmp;
                HEX_1 <= 7'b000_0110; HEX_2 <= 7'b000_0110;
            end
            RIGHT_PASS: begin
                green_tmp <= ~green_tmp; red_tmp <= 1'b0;
                HEX_1 <= 7'b000_0010; HEX_2 <= 7'b100_0000;
            end
            STOP: begin
                green_tmp <= 1'b0; red_tmp <= ~red_tmp;
                HEX_1 <= 7'b001_0010; HEX_2 <= 7'b000_1100;
            end
        endcase
    end
    assign RED_LED = red_tmp;
    assign GREEN_LED = green_tmp;
endmodule
```

4.2 Testbench Module: tb_parking_system.v

```
`timescale 1ns / 1ps

module tb_parking_system;

    // Inputs
    reg clk;
    reg reset_n;
    reg sensor_entrance;
    reg sensor_exit;
    reg [1:0] password_1;
    reg [1:0] password_2;

    // Outputs
    wire GREEN_LED;
    wire RED_LED;
    wire [6:0] HEX_1;
    wire [6:0] HEX_2;

    // Instantiate the Unit Under Test (UUT)
    parking_system uut (
        .clk(clk),
        .reset_n(reset_n),
        .sensor_entrance(sensor_entrance),
        .sensor_exit(sensor_exit),
        .password_1(password_1),
        .password_2(password_2),
        .GREEN_LED(GREEN_LED),
        .RED_LED(RED_LED),
        .HEX_1(HEX_1),
```

```
        .HEX_2(HEX_2)
    );

    // Clock generation (toggles every 5ns = 100MHz clock)
    always #5 clk = ~clk;

    initial begin
        // Initialize Inputs
        clk = 0;
        reset_n = 0;
        sensor_entrance = 0;
        sensor_exit = 0;
        password_1 = 0;
        password_2 = 0;

        // Reset the system
        #100;
        reset_n = 1;

        // Scenario: Car Arrives
        #20;
        sensor_entrance = 1;
        #50;

        // Wait for state machine to ask for password
        // Scenario: Enter WRONG Password first
        password_1 = 2'b00;
        password_2 = 2'b00;
        #50;

        // Scenario: Enter CORRECT Password (01 and 10)
        password_1 = 2'b01;
        password_2 = 2'b10;
        #100;

        // Scenario: Car enters fully (Exit sensor triggers)
        sensor_exit = 1;
        sensor_entrance = 0; // car left entrance
        #50;
        sensor_exit = 0;     // car left exit

        #100;
        $finish;
    end

endmodule
```

5 Simulation Results

The testbench `tb_parking_system.v` was executed to verify the state transitions.

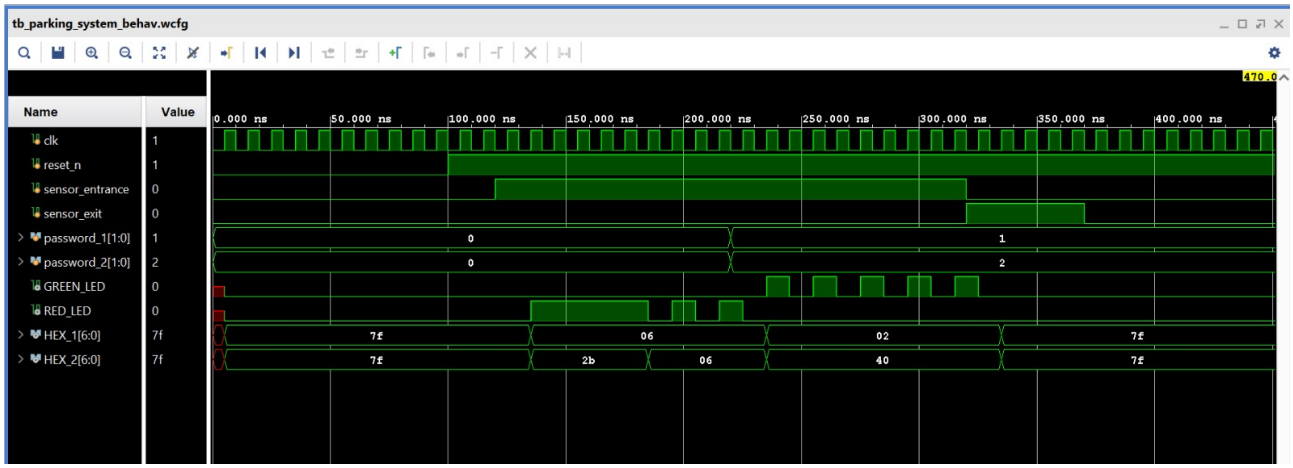

Figure 3: **Behavioral Simulation Waveform:** The simulation verifies the following sequence: (1) System Reset, (2) Car Entrance detected, (3) Wrong Password entered (Red LED active), (4) Correct Password entered (Green LED toggles).

5.1 Testbench Analysis

As seen in Figure 3:

1. At **100ns**, reset is released.
2. At **120ns**, `sensor_entrance` goes High, triggering the password wait state.
3. At **170ns**, an incorrect password is detected.
4. At **270ns**, the correct password (`password_1 = 01, password_2 = 10`) is applied. The `GREEN_LED` output begins to toggle, and the HEX displays update to show the entry success code.

6 Conclusion

The Automated Parking System was successfully designed and simulated. The Verilog implementation correctly handles state transitions based on sensor inputs and password validation logic. The RTL and synthesis schematics confirm the hardware realization of the FSM, and the simulation waveforms validate the timing and logic correctness.