Project
Ticket Collector Machine using FSM

Table Number 17 Friday Batch
TA: Rupesh Kumar

Aditya Gupta 2023EE11181
Vanshika Bathla 2023EE30828
Anurag Agarwal 2023MT10312
Vishnu Kumawat 2023EE30758

Date of submission: April 29, 2025

Contents

Project Details

1. Objective

This project aims to design a digital ticket collector system using Verilog HDL that calculates the total fare based on a selected source station, destination station, and the number of tickets, and displays the fare on LEDs.

2. Abstract

This project simulates a basic ticket vending machine. The user selects the source and destination stations using binary inputs, specifies the number of tickets required, and presses a submit button to confirm the selection. The machine computes the total fare based on predefined fare rates and displays it via LEDs. The system is built using a finite state machine (FSM) model with three states: IDLE, CALC_FARE, and SHOW_FARE. A reset option is provided to clear the system and return to the initial state.

Implementation Methodology

The implementation of the Ticket Collector Machine was carried out through the following systematic steps:

Step 1: State Machine Design
Model the system behavior using a Finite State Machine (FSM) with three states:

1. IDLE: Waits for user input and submit signal.
2. CALC_FARE: Computes fare based on input parameters.
3. SHOW_FARE: Displays the calculated fare on output LEDs.

Step 2: Input-Output Assignment
Assign specific switches (sw) for:

1. Source selection (sw[1:0])
2. Destination selection (sw[3:2])
3. Number of tickets (sw[5:4])
4. Submit button (sw[6])

Assign output LEDs (led[7:0]) to display the fare.

Step 3: Verilog Coding
Implement the FSM in Verilog:

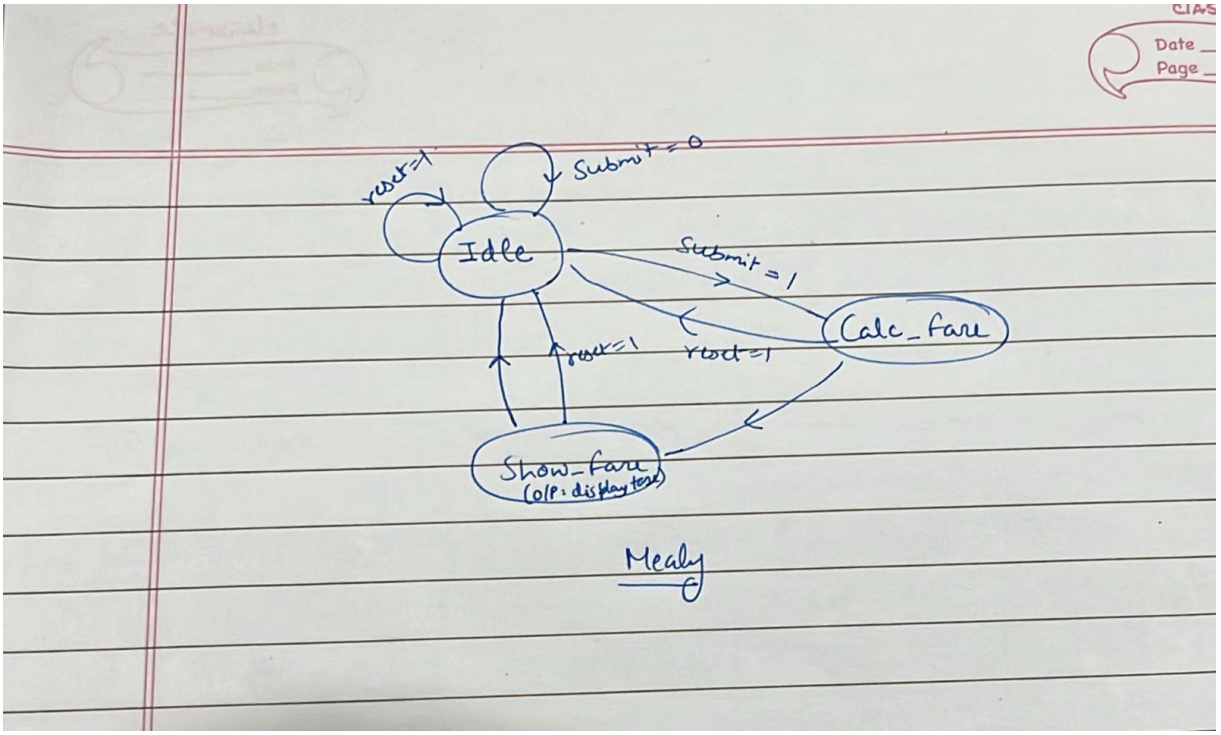
Use typedef enum to define states clearly.
Write synchronous logic for state transitions on clock edges.
Latch the input values on detecting a rising edge of the submit button.
Compute the base fare according to source and destination station codes.
Calculate total fare based on the number of tickets.
Update the output LEDs with the fare value.

Step 4: Testing and Debugging
Perform testbench simulations to verify:

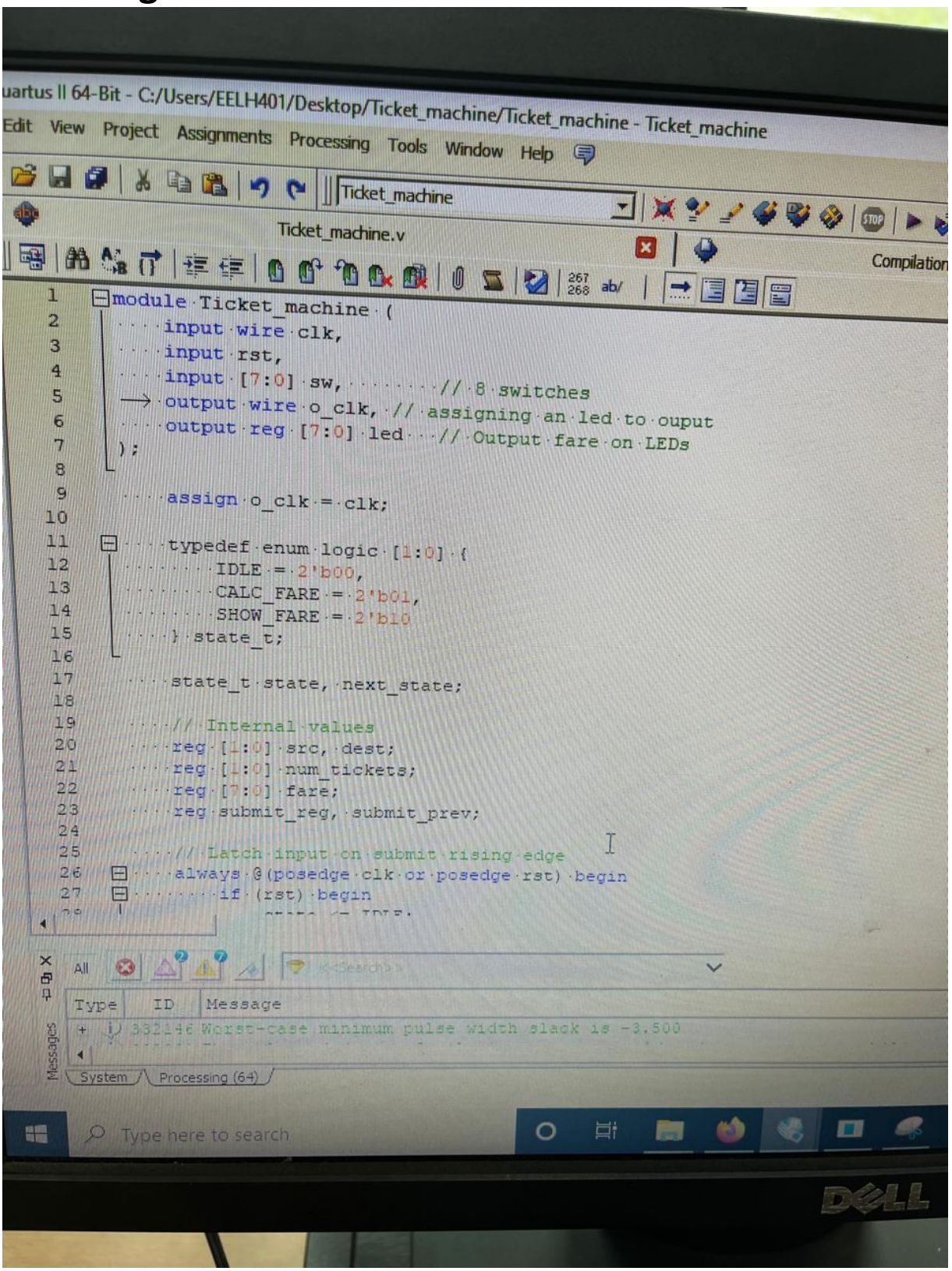
- Correct fare calculation for various source-destination pairs and ticket counts.
- Proper behavior of submit and reset functionality.
- Analyze simulation waveforms to check timing and logic correctness.
- Debug and refine the Verilog code based on simulation results.
- After simulation success, proceed to hardware testing to ensure real-world functionality.

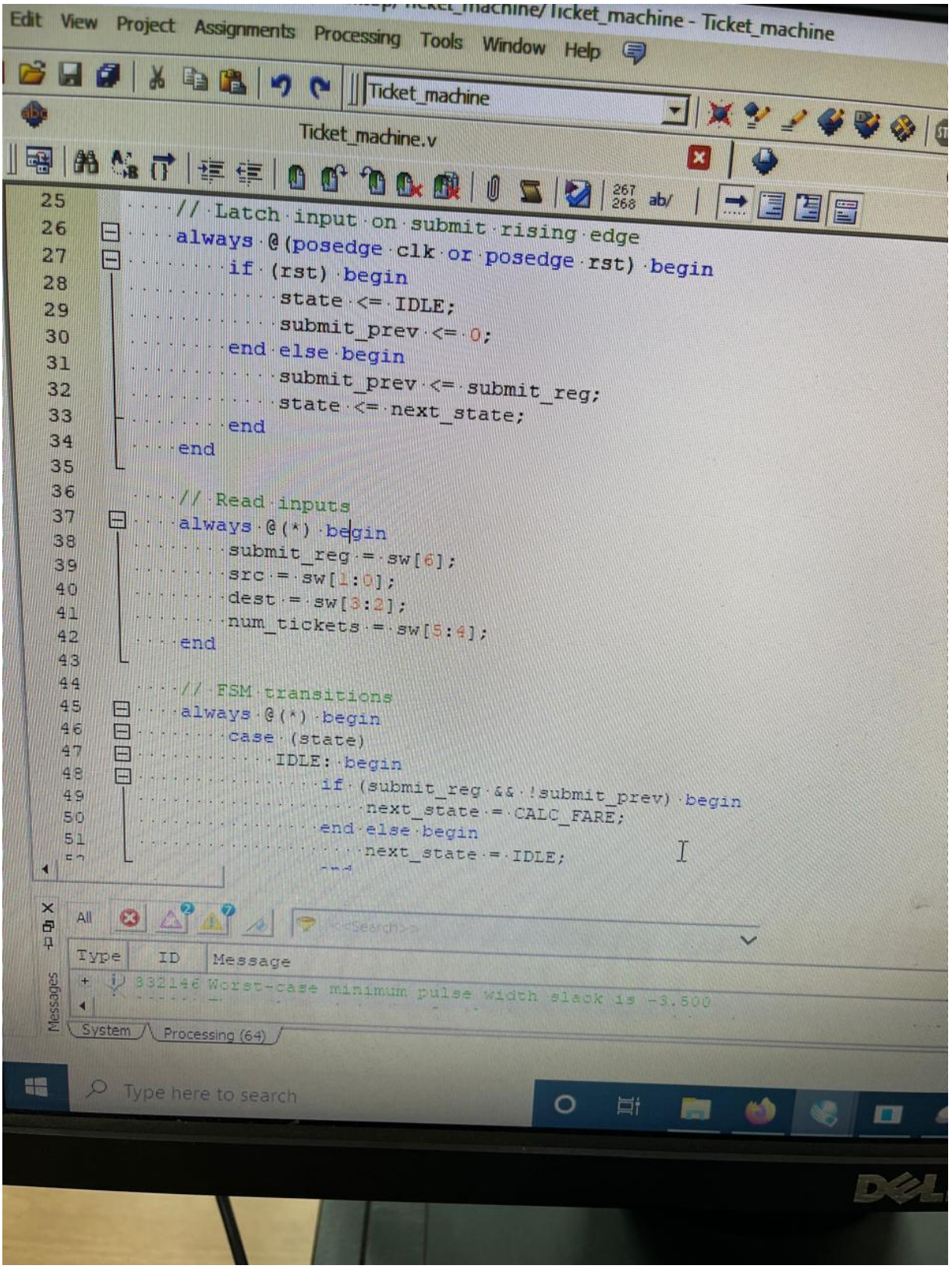
Step 5: Hardware Implementation
Program the Verilog code onto the FPGA development board.
Connect switches and LEDs as per input/output assignment.
Test the system physically by varying inputs and observing LED outputs.

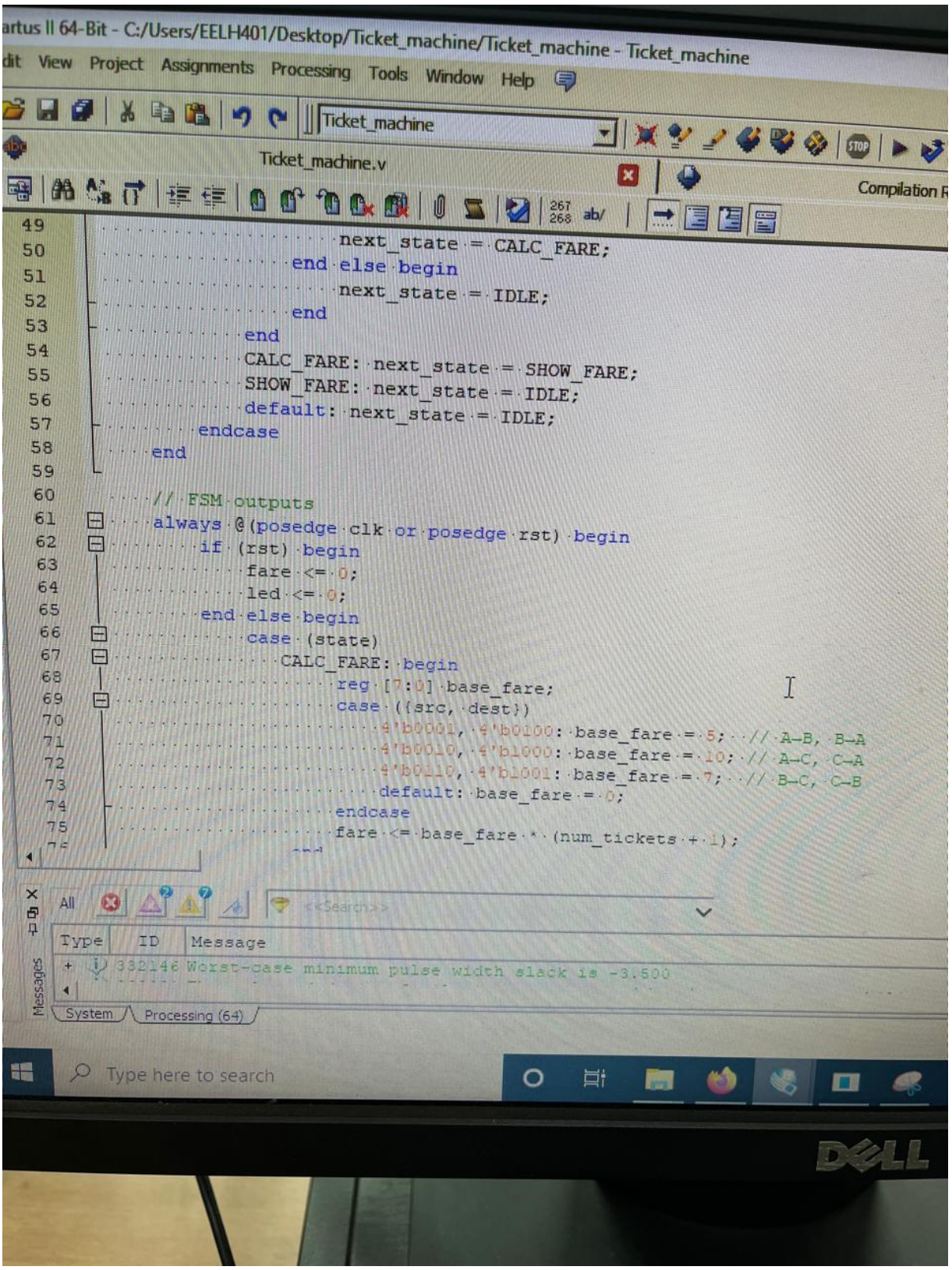
3. State Diagram

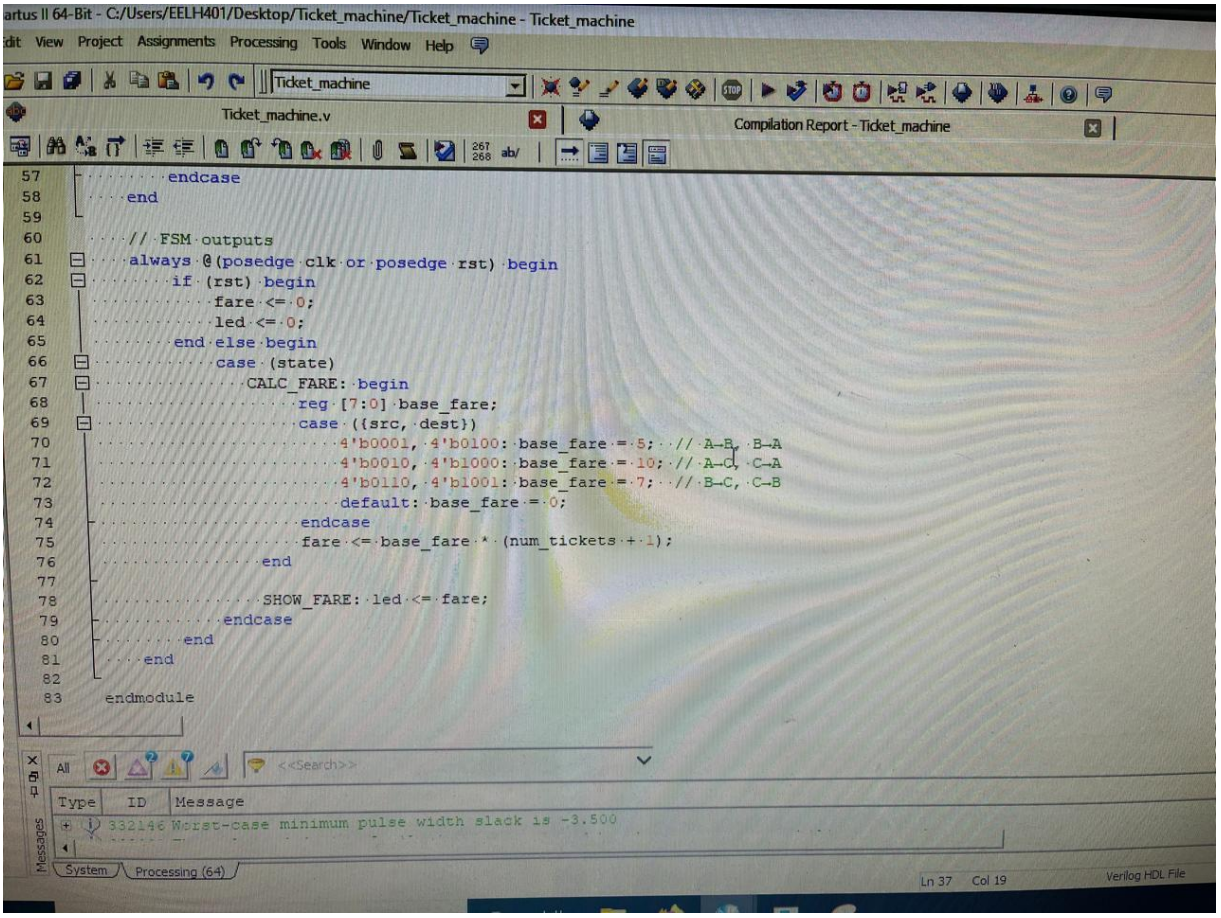


4.Verilog Code




Edit View Project Assignments Processing Tools Window Help
Ticket_machine - Ticket_machine
Ticket_machine
Ticket_machine.v
// Latch input on submit rising edge
always @(posedge clk or posedge rst) begin
if (rst) begin
state <= IDLE;
submit_prev <= 0;
end else begin
submit_prev <= submit_reg;
state <= next_state;
end
end
// Read inputs
always @(*) begin
submit_reg = sw[6];
src = sw[1:0];
dest = sw[3:2];
num_tickets = sw[5:4];
end
// FSM transitions
always @(*) begin
case (state)
IDLE: begin
if (submit_reg && !submit_prev) begin
next_state = CALC_FARE;
end else begin
next_state = IDLE;
All
Type ID Message
Messages
System
Processing (64)
Type here to search


artus II 64-Bit - C:/Users/EELH401/Desktop/Ticket_machine/Ticket_machine - Ticket_machine
dit View Project Assignments Processing Tools Window Help
Ticket_machine
Ticket_machine.v
Compilation R
next_state = CALC_FARE;
end else begin
next_state = IDLE;
end
end
CALC_FARE: next_state = SHOW_FARE;
SHOW_FARE: next_state = IDLE;
default: next_state = IDLE;
endcase
end
// FSM outputs
always @(posedge clk or posedge rst) begin
if (rst) begin
fare <= 0;
led <= 0;
end else begin
case (state)
CALC_FARE: begin
reg [7:0] base_fare;
case ({src, dest})
4'b0001, 4'b0100: base_fare = 5; // A→B, B→A
4'b0010, 4'b1000: base_fare = 10; // A→C, C→A
4'b0110, 4'b1001: base_fare = 7; // B→C, C→B
default: base_fare = 0;
endcase
fare <= base_fare * (num_tickets + 1);
All
Type ID Message
332146 Worst-case minimum pulse width slack is -3.600
Messages
System Processing (64)
Type here to search
DELL


artus II 64-Bit - C:/Users/EELH401/Desktop/Ticket_machine/Ticket_machine - Ticket_machine
Edit View Project Assignments Processing Tools Window Help
Ticket_machine
Ticket_machine.v
Compilation Report - Ticket_machine
endcase
end
// FSM outputs
always @(posedge clk or posedge rst) begin
if (rst) begin
fare <= 0;
led <= 0;
end else begin
case (state)
CALC_FARE: begin
reg [7:0] base_fare;
case ({src, dest})
4'b0001, 4'b0100: base_fare = 5; // A→B, B→A
4'b0010, 4'b1000: base_fare = 10; // A→C, C→A
4'b0110, 4'b1001: base_fare = 7; // B→C, C→B
default: base_fare = 0;
endcase
fare <= base_fare * (num_tickets + 1);
end
SHOW_FARE: led <= fare;
endcase
end
end
endmodule
All
<<Search>>
Type ID Message
332146 Worst-case minimum pulse width slack is -3.500
Messages
System
Processing (64)
Ln 37 Col 19
Verilog HDL File

5. **Testbench code**

Input:

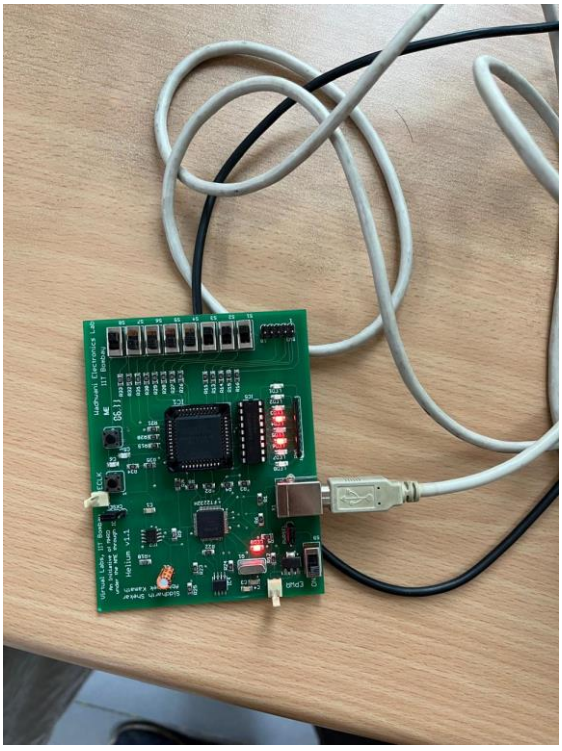
```
`timescale 1ns/1ps

module tb_ticket_machine_fsm_2;
    reg clk, rst;
    reg [7:0] sw;
    wire [7:0] led;

    ticket_machine_fsm uut (.clk(clk), .rst(rst), .sw(sw), .led(led));

    // Clock generation
    always #5 clk = ~clk;

    // Submit a route and wait for response
    task submit_trip(input [1:0] src, input [1:0] dest, input [1:0] num_tickets, input string label);
        begin
            sw = {1'b0, 1'b1, num_tickets, dest, src}; // submit bit is bit[6]
            #10;
            sw[6] = 0; // falling edge
            #100; // allow FSM to cycle through CALC_FARE and SHOW_FARE
            $display("Fare for %s: %d", label, led);
        end
    endtask

    initial begin
        // Enable waveform dump for EPWave
        $dumpfile("dump.vcd");
        $dumpvars(0, tb_ticket_machine_fsm_2);

        clk = 0; rst = 1; sw = 0;
        #20; rst = 0;

        // Test A->B (src=00, dest=01), 1 ticket
        submit_trip(2'b00, 2'b01, 2'b00, "A->B, 1 ticket");  // 5 * (0+1) = 5

        // Test B->A (src=01, dest=00), 2 tickets
        submit_trip(2'b01, 2'b00, 2'b01, "B->A, 2 tickets"); // 5 * (1+1) = 10

        // Test A->C (src=00, dest=10), 4 tickets
        submit_trip(2'b00, 2'b10, 2'b11, "A->C, 4 tickets"); // 10 * (3+1) = 40

        // Test C->A (src=10, dest=00), 3 tickets
        submit_trip(2'b10, 2'b00, 2'b10, "C->A, 3 tickets"); // 10 * (2+1) = 30

        // Test B->C (src=01, dest=10), 2 tickets
        submit_trip(2'b01, 2'b10, 2'b01, "B->C, 2 tickets"); // 7 * (1+1) = 14

        // Test invalid route (e.g., A->A)
        submit_trip(2'b00, 2'b00, 2'b10, "A->A, 3 tickets"); // 0 * (2+1) = 0

        #20;
        $finish;
    end
endmodule
```

Output:

```
[2025-04-29 10:04:05 UTC] iverilog '-Wall' '-g2012' design.sv testbench.sv  && unbuffer vvp a.out
warning: Some design elements have no explicit time unit and/or
       : time precision. This may cause confusing timing results.
       : Affected design elements are:
       :   -- module ticket_machine_fsm declared here: design.sv:1
VCD info: dumpfile dump.vcd opened for output.
Fare for A->B, 1 ticket:   5
Fare for B->A, 2 tickets:  10
Fare for A->C, 4 tickets:  40
Fare for C->A, 3 tickets:  30
Fare for B->C, 2 tickets:  14
Fare for A->A, 3 tickets:   0
testbench.sv:51: $finish called at 700000 (1ps)
Finding VCD file...
./dump.vcd
[2025-04-29 10:04:06 UTC] Opening EPWave...
Done
```

Output Waveform for input: A→B, 1 ticket

CPLD Board Images demonstrating output



Input: A→C, passengers=4
Output: 40

Input: C→B, passengers=4
Output: 28

6.Inference

The project successfully implements a Ticket Collector Machine using Verilog based on an FSM (Finite State Machine) design. The system correctly accepts the source, destination, and number of tickets as inputs, detects the submit signal, computes the total fare based on predefined fare rules, and displays the result on LEDs. Both **simulations** through testbenches and hardware testing confirmed the correct functionality of the system.