DESIGN OF A TRAIN TICKET MACHINE

AT81.06 VLSI DESIGN

INSTRUCTORS: Dr. Krit Athikulwongse

Mr. Chatchai Pruetong

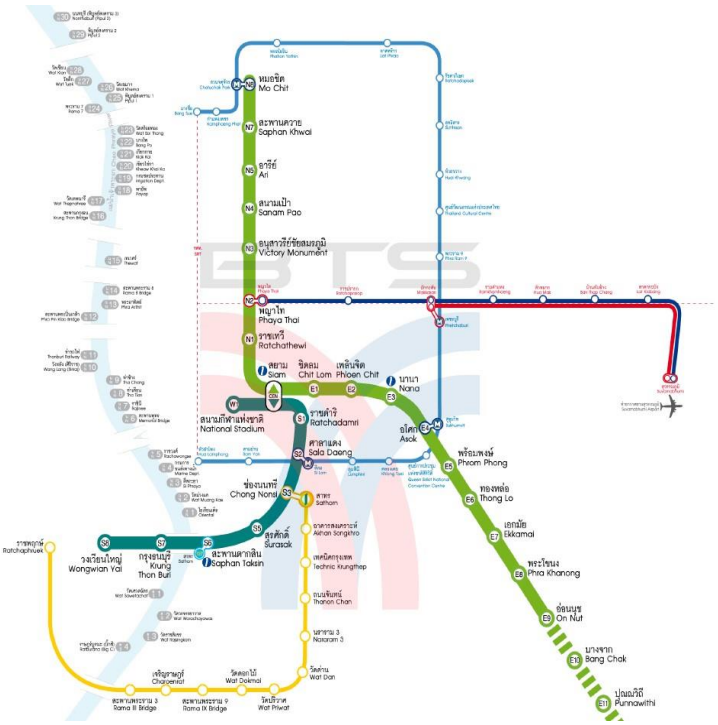
Prepared by:

Mohammad Ashiqur Noor (st117813)

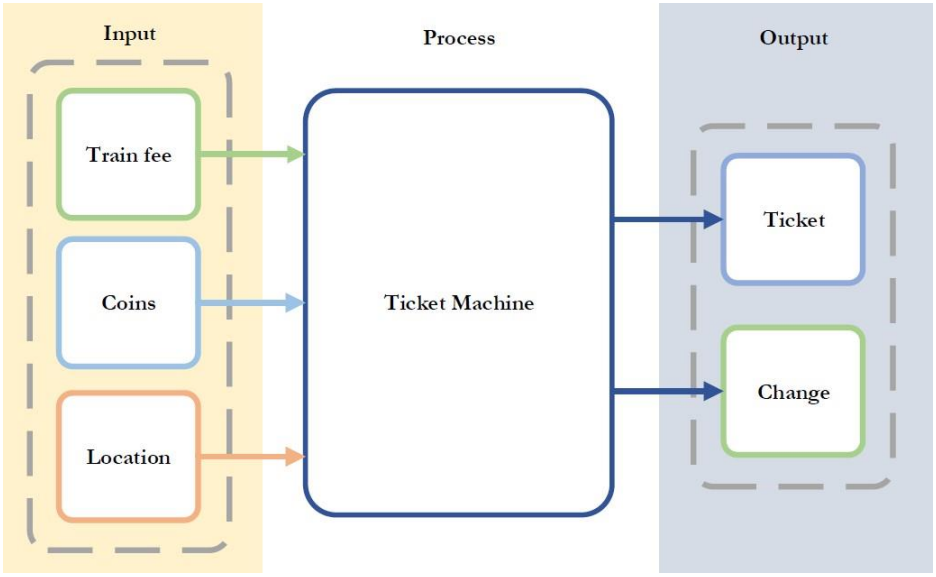
Introduction

The train ticket machine, a type of vending machine, efficiently dispenses tickets in paper or electronic form to the user. This innovative system significantly reduces the workload of the staff at the counter, enhancing the overall efficiency of the ticketing process for travelers.

This project is specifically designed for passengers using the BTS system to travel inside Bangkok. The map below shows the tram lines for BTS, subway, and airport rail links. In this project, we focus on only two lines, green and dark green, to make the ticket machine model. We will implement this design using Verilog Hardware Description Language (HDL) and Moore Finite State Machine (FSM) approach. The second diagram shows the train ticket fees in baht according to the number of stations the user wants to travel.



Design and Operation



The ticket machine for the BTS will dispense the ticket after it collects the information from the passenger. The information consists of source and destination locations and the ticket fee, which will be shown in the display interface. The passenger must press the buttons corresponding to the desired locations and provide the train fee by inserting the coins. After collecting all the data, the appropriate ticket will be dispensed, and the change will be made if the passenger has given more money than the machine requires. It will also wait for the passenger to take the ticket and change, and after that, it will go back to its initial state, waiting for the next passenger. If a person goes somewhere in the middle of the operation, the other person can reset the machine back to its original state and the change, if available, is returned to the user.

Inputs and Outputs

The inputs used in designing the train ticket machine are as follows:

a. Train Fee
b. Coins
c. Location (Hardcoded)

The outputs of the train ticket machine are as follows:

i. Ticket
ii. Change

State Diagrams

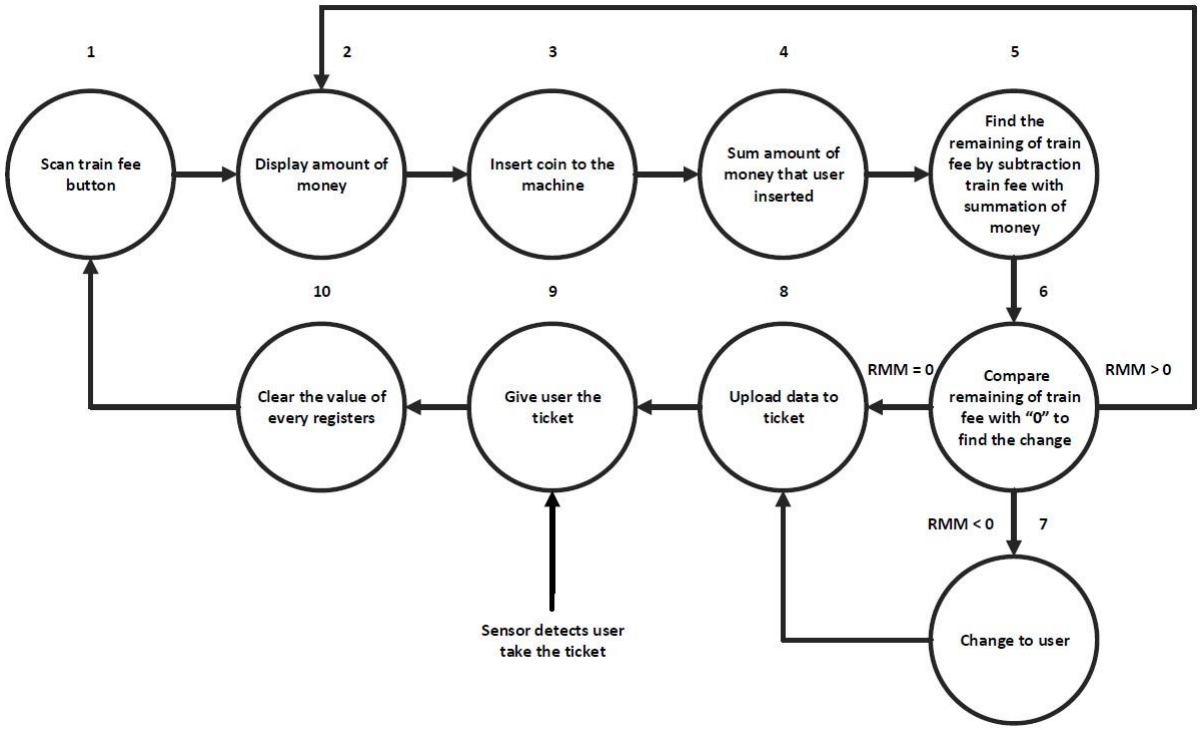


Fig 1: Main State Diagram

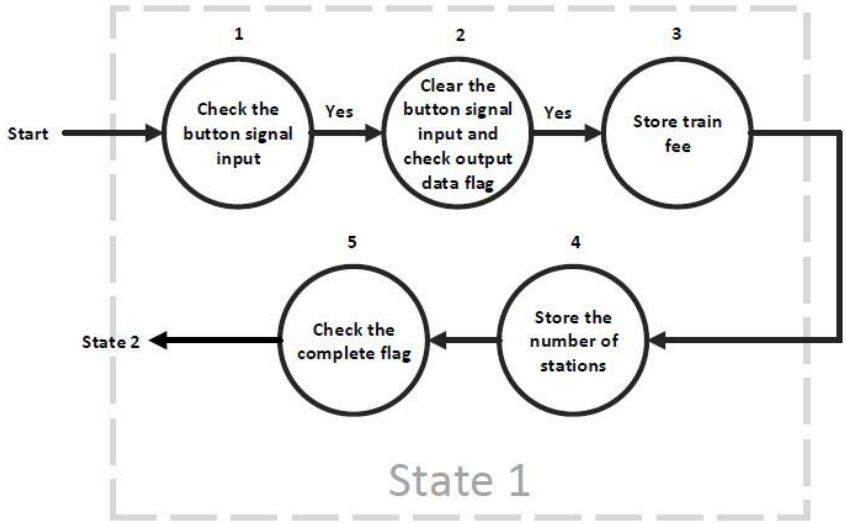


Fig 2: State Diagram of State 1

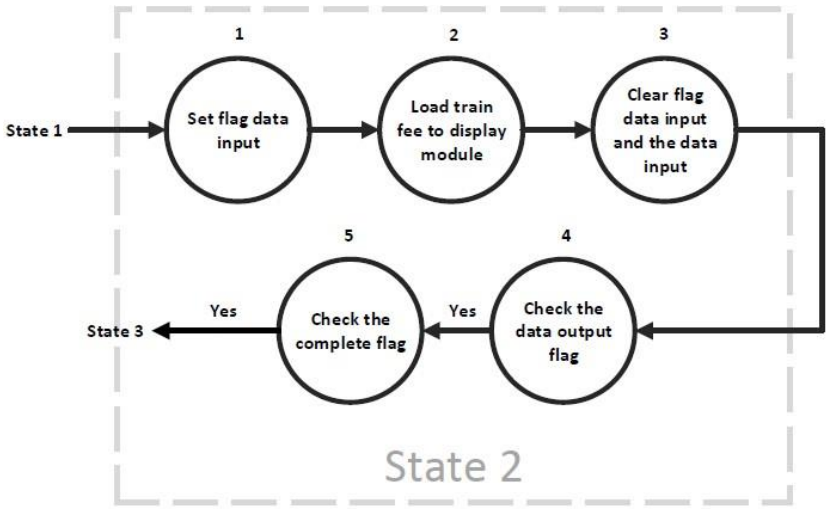


Fig 3: State Diagram of State 2

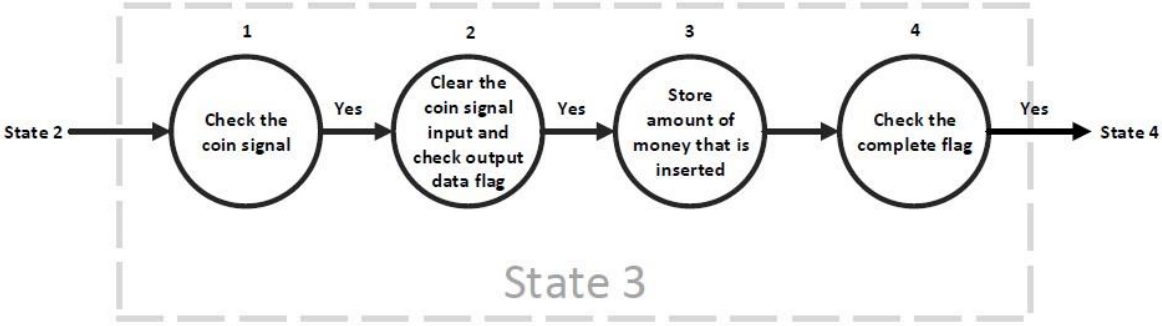


Fig 4: State Diagram of State 3

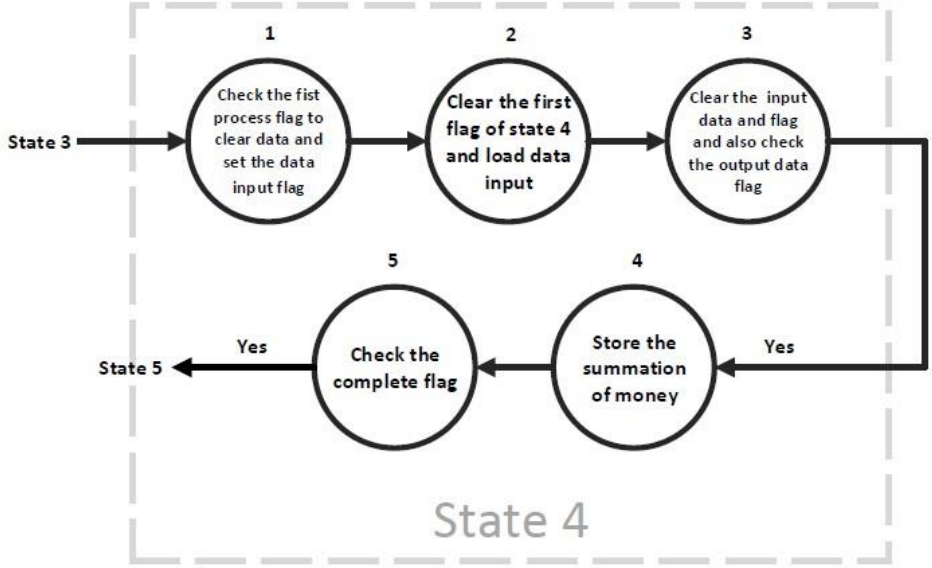


Fig 5: State Diagram of State 4

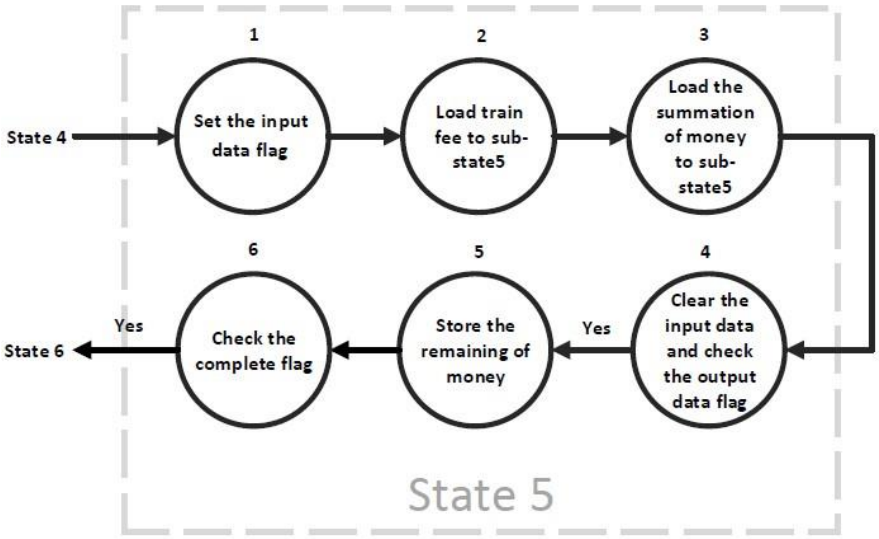


Fig 6: State Diagram of State 5

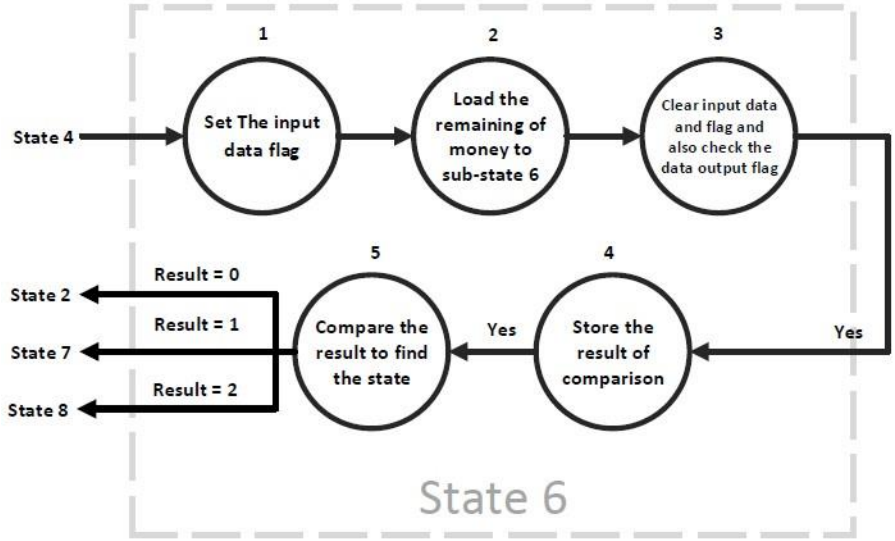


Fig 7: State Diagram of State 6

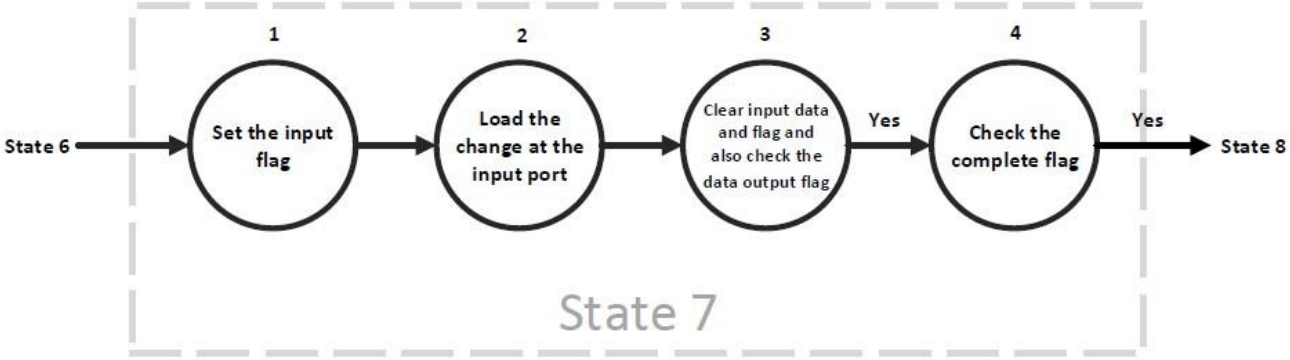


Fig 8: State Diagram of State 7

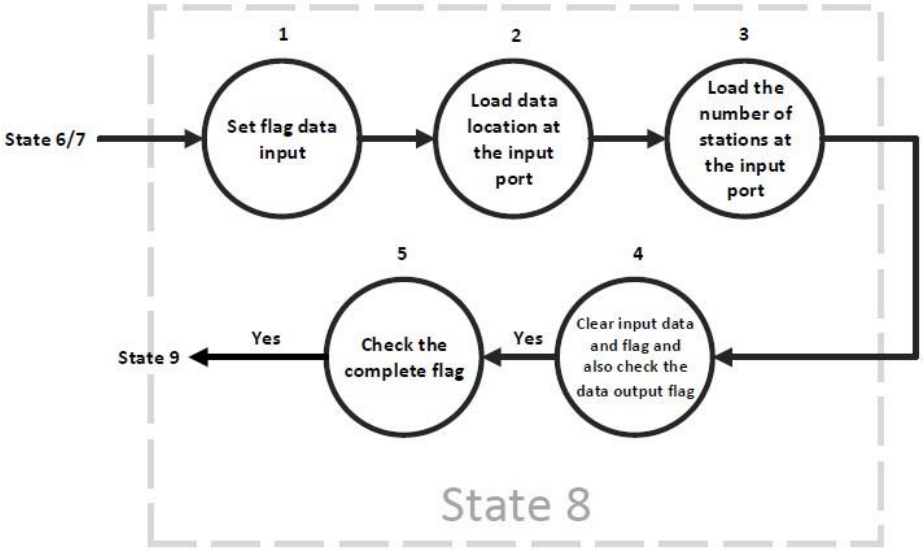


Fig 9: State Diagram of State 8

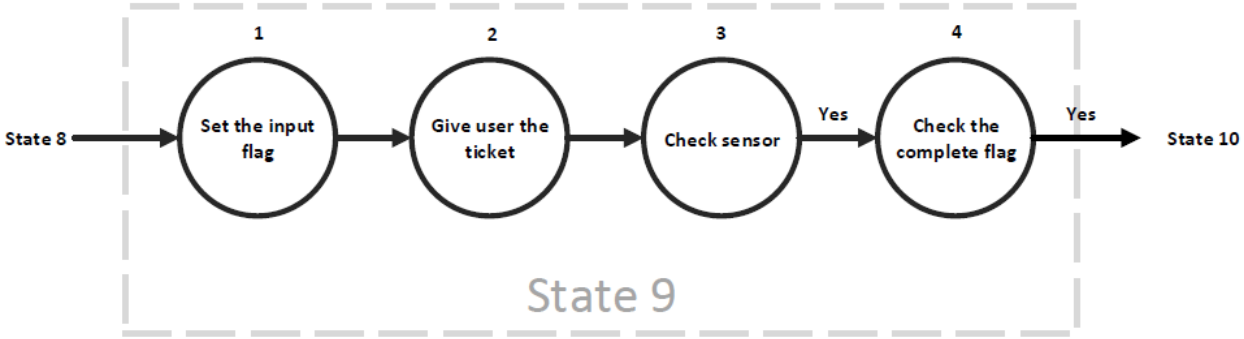


Fig 10: State Diagram of State 9

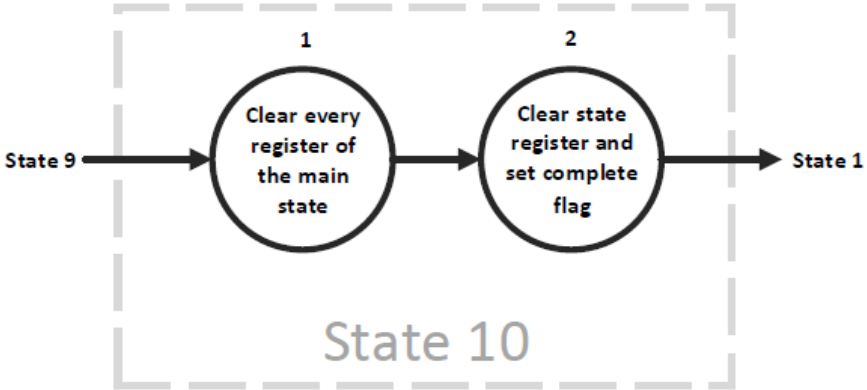


Fig 11: State Diagram of State 10

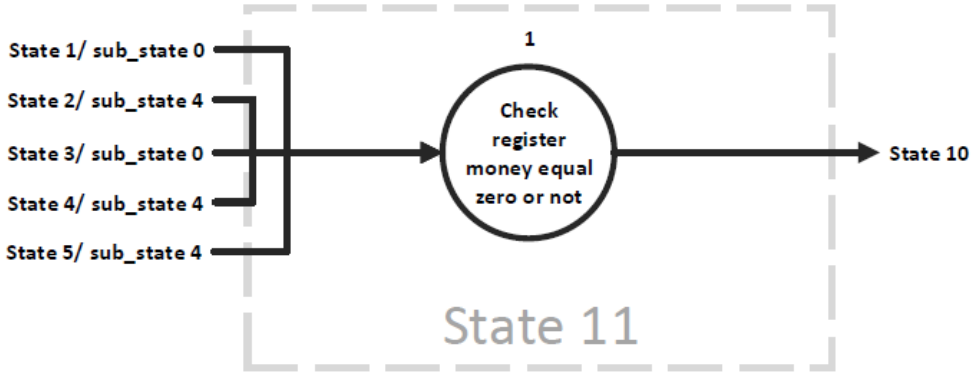


Fig 12: State Diagram of State 11

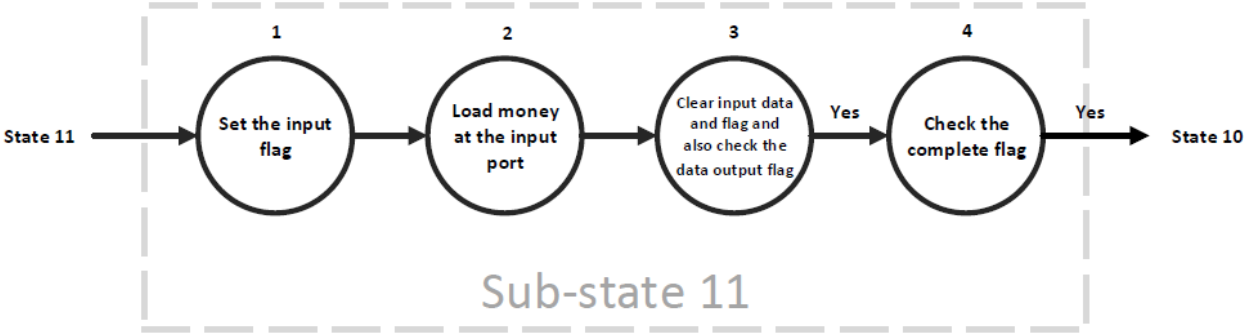


Fig 13: State Diagram of State 12

Design Testing

The design undergoes rigorous testing to ensure its reliability. The Verilog HDL code is simulated in ModelSim software, and the inputs are thoroughly tested. The output waveforms are displayed and verified, providing a comprehensive assessment of the design's functionality. The code is divided into two parts for clarity: Source Code and Test bench Code.

Source Code:

```
module Main_TFMC(clkm, rstm, reset_button, sensor_t, DATA_inm, state_cmpm, return_state);
        input clkm, rstm;
        input reset_button, sensor_t;
        input [7:0] DATA_inm;
        output state_cmpm;
        output [3:0] return_state;
        reg state_cmpm, rst_cmp;
        reg frt_fg;             // first operation flag
        reg rst_but_act;
        reg [3:0] return_state;
        reg [3:0] sub_state, main_state;
        reg [7:0] data_mem1, coin_m;
        reg signed [7:0] rmm;
        reg [7:0] train_fee, NMS, money;
        parameter locate = 8'b0010_0001;

        wire out_RDY1, state_cmp1;
        wire [7:0] DATA_out1;
        reg [7:0] button;

        reg in_RDY2;
        wire out_RDY2, state_cmp2;
        wire [7:0] DATA_out2;
        reg [7:0] DATA_in2;

        wire out_RDY3, state_cmp3;
        wire [7:0] DATA_out3;
        reg [7:0] coin_in;


        reg frt_fg4, in_RDY4;
        reg [7:0] DATA_in4;
        wire out_RDY4, state_cmp4;
        wire [7:0] DATA_out4;

        reg in_RDY5;
        reg [7:0] DATA_in5;
        wire out_RDY5, state_cmp5;
        wire [7:0] DATA_out5;

        reg in_RDY6;
        reg signed[7:0] DATA_in6;
        wire out_RDY6, state_cmp6;
        wire [7:0] DATA_out6;

        reg in_RDY7;
        wire out_RDY7, state_cmp7;
        wire [7:0] DATA_out7;
        reg[7:0] DATA_in7;

        reg in_RDY8;
        wire out_RDY8, state_cmp8;
        wire [7:0] DATA_out8;
        reg[7:0] DATA_in8;

        reg in_RDY9;
        wire ticket_out, state_cmp9;


button_scan state1(.clk(clkm), .rst(rstm), .but(button), .state_cmp1(state_cmp1), .out_RDY1(out_RDY1), .DATA_out1(DATA_out1));
Display state2(.clk(clkm), .rst(rstm), .in_RDY2(in_RDY2), .DATA_in2(DATA_in2), .state_cmp2(state_cmp2), .out_RDY2(out_RDY2), .DATA_out2(DATA_out2) );
Ins_coin state3(.clk(clkm), .rst(rstm), .coin_in(coin_in), .state_cmp3(state_cmp3), .out_RDY3(out_RDY3), .DATA_out3(DATA_out3));
sum_coins state4(.clk(clkm), .rst(rstm), .frt_fg4(frt_fg4), .in_RDY4(in_RDY4), .DATA_in4(DATA_in4), .state_cmp4(state_cmp4), .out_RDY4(out_RDY4),  .DATA_out4(DATA_out4));
tf_rmm state5(.clk(clkm), .rst(rstm), .in_RDY5(in_RDY5), .DATA_in5(DATA_in5), .state_cmp5(state_cmp5), .out_RDY5(out_RDY5), .DATA_out5(DATA_out5)) ;
compare state6(.clk(clkm), .rst(rstm), .in_RDY6(in_RDY6), .DATA_in6(DATA_in6), .state_cmp6(state_cmp6), .out_RDY6(out_RDY6), .DATA_out6(DATA_out6) );
change state7(.clk(clkm), .rst(rstm), .in_RDY7(in_RDY7), .DATA_in7(DATA_in7), .state_cmp7(state_cmp7), .out_RDY7(out_RDY7), .DATA_out7(DATA_out7)) ;
updata state8(.clk(clkm), .rst(rstm), .in_RDY8(in_RDY8), .DATA_in8(DATA_in8), .state_cmp8(state_cmp8), .out_RDY8(out_RDY8), .DATA_out8(DATA_out8)) ;
Ticketout state9(.clk(clkm), .rst(rstm), .sensor_t(sensor_t), .in_RDY9(in_RDY9), .state_cmp9(state_cmp9), .ticket_out(ticket_out));
```

```
always @(posedge clkm, posedge rstm) begin
        if(rstm) begin
                frt_fg <= 1;
                sub_state <= 4'b0000;
                main_state <= 4'b0000;
                end
        else begin
                if(reset_button)
                        rst_but_act <= 1;
                else begin
                        case(main_state)
                        4'b0000 : begin                                                 // state 1
                                  case(sub_state)
                                  4'b0000 : begin
                                            state_cmpm <= 0;
                                            if(DATA_inm != 0) begin
                                                    button <= DATA_inm;                 // give the input button signal to sub-state 1
                                                    sub_state <= sub_state + 1;
                                                    end
                                            else begin
                                                    if(rst_but_act == 1) begin          // when user press the button reset
                                                            main_state <= 4'b1010;
                                                            sub_state <= 4'b0000;
                                                            end
                                                    end
                                            end
                                  4'b0001 : begin
                                            button <= 8'b0000_0000;
                                            if(out_RDY1)
                                                    sub_state <= sub_state + 1;
                                            end
                                  4'b0010 : begin
                                            train_fee <= DATA_out1;             // store train fee to train fee variable
                                            sub_state <= sub_state + 1;
                                            end
                                  4'b0011 : begin
                                            NMS <= DATA_out1;                   // store the number of station  to number of station variable
                                            sub_state <= sub_state + 1;
                                            end
                                  4'b0100 : begin
                                            if(state_cmp1) begin
                                                    main_state <= main_state + 1;
                                                    sub_state <= 0;
                                                    state_cmpm <= 1;
                                                    end
                                            end
                                  endcase
                                  return_state <= main_state;
                                  end
                        4'b0001 : begin                                             // state 2
                                  case(sub_state)
                                  4'b0000 : begin
                                            state_cmpm <= 0;
                                            in_RDY2 <= 1;
                                            sub_state <= sub_state + 1;
                                  4'b0001 : begin
                                            if(frt_fg) begin
                                                    DATA_in2 <= train_fee;      // Display train fee for the first process
                                                    end
                                            else begin
                                                    DATA_in2 <= rmm;            // Display the remaining of money for the second, third.... process
                                                    end
                                            sub_state <= sub_state + 1;
                                            end
                                  4'b0010 : begin
                                            in_RDY2 <= 0;
                                            DATA_in2 <= 8'b0000_0000;
                                            sub_state <= sub_state + 1;
                                            end
                                  4'b0011 : begin
                                            if(out_RDY2)
                                                sub_state <= sub_state + 1;
                                            end
                                  4'b0100 : begin
                                            if(state_cmp2) begin
                                                    if(rst_but_act == 1)                // when user press the button reset
                                                            main_state <= 4'b1010;
                                                    else
                                                            main_state <= main_state + 1;
                                                    sub_state <= 0;
                                                    state_cmpm <= 1;
                                                    end
```

```
                end
        endcase
        return_state <= main_state;
        end
4'b0010 : begin                                                 // state 3
        case(sub_state)
        4'b0000 : begin
                state_cmpm <= 0;
                if(DATA_inm != 0) begin
                        coin_in <= DATA_inm;                    // give the input coin signal to sub-state3
                        sub_state <= sub_state + 1;
                        end
                else begin
                        if(rst_but_act == 1) begin              // when user press the button reset
                                main_state <= 4'b1010;
                                sub_state <= 4'b0000;
                                end
                        end
                end
        4'b0001 : begin
                coin_in <= 8'b0000_0000;
                if(out_RDY3)
                        sub_state <= sub_state + 1;
                end
        4'b0010 : begin
                coin_m <= DATA_out3;
                sub_state <= sub_state + 1;
                end
        4'b0011 : begin
                if(state_cmp3) begin
                        main_state <= main_state + 1;
                        sub_state <= 0;
                        state_cmpm <= 1;
                        end
                end
        endcase
        return_state <= main_state;
        end
4'b0011 : begin                                 // state 4
        case(sub_state)
        4'b0000 : begin
                if(frt_fg) begin                // check whether this is the first process or not to clear the register
                        frt_fg4 <= frt_fg;
                        frt_fg <= 0;
                        end
                else
                        frt_fg4 <= 0;

                state_cmpm <= 0;
                in_RDY4 <= 1;
                sub_state <= sub_state + 1;
                end
        4'b0001 : begin
                frt_fg4 <= 0;
                DATA_in4 <= coin_m;             // load money that user inserted to sub-state 4
                sub_state <= sub_state + 1;
                end
        4'b0010 : begin
                in_RDY4 <= 0;
                DATA_in4 <= 8'b0000_0000;
                if(out_RDY4)
                        sub_state <= sub_state + 1;
                end
        4'b0011 : begin
                money <= DATA_out4;                     //  get the summation of money that user insert
                sub_state <= sub_state + 1;
                end
        4'b0100 : begin
                if(state_cmp4) begin
                        if(rst_but_act == 1)            // when user press the button reset
                                main_state <= 4'b1010;
                        else
                                main_state <= main_state + 1;
                        sub_state <= 0;
                        state_cmpm <= 1;
                        end
                end
        endcase
        return_state <= main_state;
        end
4'b0100 : begin                                         // state 5
        case(sub_state)
        4'b0000 : begin
                state_cmpm <= 0;
```

```
                in_RDY5 <= 1;
                sub_state <= sub_state + 1;
                end
        4'b0001 : begin
                DATA_in5 <= train_fee;                  // give the train fee to sub-state 5
                sub_state <= sub_state + 1;
                end
        4'b0010 : begin
                DATA_in5 <= money;                      // give amount of money to sub-state 5
                sub_state <= sub_state + 1;
                end
        4'b0011 : begin
                in_RDY5 <= 0;
                DATA_in5 <= 8'b0000_0000;
                if(out_RDY5)
                        sub_state <= sub_state + 1;
                end
        4'b0100 : begin
                rmm <= DATA_out5;                       // get the remaining of money that user have to insert
                sub_state <= sub_state + 1;
                end
        4'b0101 : begin
                if(state_cmp5) begin
                        if(rst_but_act == 1)            // when user press the button reset
                                main_state <= 4'b1010;
                        else
                                main_state <= main_state + 1;
                        sub_state <= 0;
                           state_cmpm <= 1;
                           end
                 end
          endcase
          return_state <= main_state;
          end
4'b0101 : begin                                                 // state 6
          case(sub_state)
          4'b0000 : begin
                    state_cmpm <= 0;
                    in_RDY6  <= 1;
                    sub_state <= sub_state + 1;
                    end
          4'b0001 : begin
                    DATA_in6 <= rmm;                            // give the remaining of money to sub-state 6
                    sub_state <= sub_state + 1;
                    end
          4'b0010 : begin
                    in_RDY6  <= 0;
                    DATA_in6 <= 8'b0000_0000;
                    if(out_RDY6)
                            sub_state <= sub_state + 1;
                    end
          4'b0011 : begin
                    data_meml  <= DATA_out6;                    // the result of comparation
                    sub_state <= sub_state + 1;
                    end


      4'b0100 : begin
                if(state_cmp6) begin
                       if(data_meml == 8'b0000_0001) begin // if the result of camparison is 1 that means user insert money more than train fee
                              money <= 8'b0000_0000;
                              main_state <= 4'b0110;
                              end
                       else begin
                              if(data_meml == 8'b0000_0010)       // if the result of comparison is 2 that means user insert money less than train fee
                                     main_state <= 4'b0001;
                              else begin
                                     money <= 8'b0000_0000;      // if the result of caomparison is 0 that means user insert money equal train fee
                                     main_state <= 4'b0111;
                              end
                       end
                       sub_state <= 0;
                       state_cmpm <= 1;
                       end
                end
      endcase
```

```
                return_state <= main_state;
                end
        4'b0110 : begin                                                 // state 7
                case(sub_state)
                4'b0000 : begin
                        state_cmpm <= 0;
                        in_RDY7 <= 1;
                        sub_state <= sub_state + 1;
                        end
                4'b0001 : begin
                        DATA_in7 <= rmm;                                // load the remaining as the change to give the user
                        sub_state <= sub_state + 1;
                        end
                4'b0010 : begin
                        in_RDY7 <= 0;
                        DATA_in7 <= 8'b0000_0000;
                        if(out_RDY7)
                                sub_state <= sub_state + 1;
                        end
                4'b0011 : begin
                        if(state_cmp7) begin
                                rmm <= 8'b0000_0000;
                                main_state <= main_state + 1;
                                sub_state <= 0;
                                state_cmpm <= 1;
                                end
                        end
                endcase
                return_state <= main_state;
                end

        4'b0111 : begin                                                 // state 8
                case(sub_state)
                4'b0000 : begin
                        state_cmpm <= 0;
                        in_RDY8 <= 1;
                        sub_state <= sub_state + 1;
                        end
                4'b0001 : begin
                        DATA_in8 <= locate;                             // load the location of the machine to sub-state 8
                        sub_state <= sub_state + 1;
                        end
                4'b0010 : begin
                        DATA_in8 <= NMS;                                // load the number of station of the machine to sub-state 8
                        sub_state <= sub_state + 1;
                        end
                4'b0011 : begin
                        in_RDY8 <= 0;
                        DATA_in8 <= 8'b0000_0000;
                        if(out_RDY8)
                                sub_state <= sub_state + 1;
                        end
                4'b0100 : begin
                        if(state_cmp8) begin
                                main_state <= main_state + 1;
                                sub_state <= 0;
                                state_cmpm <= 1;
                                end
                        end

                endcase
                return_state <= main_state;
                end
        4'b1000 : begin                                                 // state 9
                case(sub_state)
                4'b0000 : begin
                        state_cmpm <= 0;
                        in_RDY9 <= 1;                                   // sent the signal to tell the machine to give user the ticket
                        sub_state <= sub_state + 1;
                        end
                4'b0001 : begin
                        in_RDY9 <= 0;
                        if(ticket_out)                                  // wait for the ticket out
                                sub_state <= sub_state + 1;
                        end
                4'b0010 : begin
                        if(sensor_t)                                    // wait for user take the ticket
                                sub_state <= sub_state + 1;
                        end
                4'b0011 : begin
                        if(state_cmp9) begin
                                main_state <= main_state + 1;
                                sub_state <= 0;
                                state_cmpm <= 1;
                                end
                        end
                endcase
                return_state <= main_state;
                end
```

```
                        4'b1001 : begin                                         // state 10
                                  case(sub_state)
                                  4'b0000 : begin
                                            state_cmpm <= 0;                    // clear the registers in the main function
                                            return_state <= 4'b0000;
                                            frt_fg <= 1;
                                            data_meml <= 8'b0000_0000;
                                            rmm <= 8'b0000_0000;
                                            train_fee <= 8'b0000_0000;
                                            NMS<= 8'b0000_0000;
                                            money <= 8'b0000_0000;
                                            sub_state <= sub_state + 1;
                                            end
                                  4'b0001 : begin
                                            sub_state <= 4'b0000;
                                            main_state  <= 4'b0000;
                                            state_cmpm <= 1;
                                            end
                                  endcase
                                  end

/* Reset state ------------------------------------------------------------------------ */
                        4'b1010 : begin                                         // state 11 : handle the process when user press the reset button
                                  if(money != 0) be                             // check user insert money or not? if user inserted money,
                                            case(sub_state)                     // the machine have to give money back (this state work as state 7)
                                            4'b0000 : begin
                                                      state_cmpm <= 0;
                                                      in_RDY7 <= 1;
                                                      sub_state <= sub_state + 1;
                                                      end


                                            4'b0001 : begin
                                                      DATA_in7 <= money;
                                                      sub_state <= sub_state + 1;
                                                      end
                                            4'b0010 : begin
                                                      in_RDY7 <= 0;
                                                      DATA_in7 <= 8'b0000_0000;
                                                      if(out_RDY7)
                                                            sub_state <= sub_state + 1;
                                                      end
                                            4'b0011 : begin
                                                      if(state_cmp7) begin
                                                            rst_but_act <= 0;
                                                            money <= 8'b0000_0000;
                                                            main_state <= 4'b1001;
                                                            sub_state <= 0;
                                                            state_cmpm <= 1;
                                                            end
                                                      end
                                            endcase
                                            end
                                  end
                        endcase
                end
        end
end
endmodule
```

Test Bench Code:

```
module Main_TFMC_test;

reg clk, rst;
reg reset_button, sensor_t;
reg [7:0] DATA_inm;
wire state_cmpm;
wire [3:0] return_state;
reg [3:0] state_test, sub_state_test;
reg [3:0] j;
integer i;

Main_TFMC maint(.clkm(clk), .rstm(rst), .reset_button(reset_button), .sensor_t(sensor_t), .DATA_inm(DATA_inm), .state_cmpm(state_cmpm), .return_state(return_state));

initial
        begin
             clk = 0;
             rst = 1;
        #10  clk = 1;
             rst = 0;
             j = 3'b000;
             state_test = 4'b0000;
             sub_state_test = 4'b0000;
        end

always begin
for(i=0;i<200;i=i+1) begin
        #10 clk = 1;
        #10 clk = 0;
        end
end

always @(posedge clk) begin
        case(state_test)
        4'b0000 : begin                                                 // for testing state 1
                  case(sub_state_test)
                  4'b0000 : begin
                            DATA_inm = 8'b0001_0000;
                            sub_state_test = sub_state_test + 1;
                            end
                  4'b0001 : begin
                            //reset_button = 1;                         // user press the reset button in state 1
                            DATA_inm = 8'b0000_0000;
                            sub_state_test = sub_state_test + 1;
                            end


                   4'b0010 : begin
                             //reset_button = 0;                            // clear the reset button signal
                             if(state_cmpm == 1) begin
                                     state_test = state_test + 1;
                                     sub_state_test = 0;
                                     end
                             end
                   endcase
                   end
        4'b0001 : begin                                                 // for testing state 2
                  if(state_cmpm == 1)
                         state_test = state_test + 1;
                  end
        4'b0010 : begin                                                 // for testing state 3
                  case(sub_state_test)
                  4'b0000 : begin
                            DATA_inm = 8'b0000_1000;                    // insert 10 bath
                            sub_state_test = sub_state_test + 1;
                            end
                  4'b0001 : begin
                            //if(j == 2)
                                    //reset_button = 1;                 // user press the button in state 3 after they insert coin 2 times
                            DATA_inm = 8'b0000_0000;
                            sub_state_test = sub_state_test + 1;
                            //j = j + 1;
                            end
                  4'b0010 : begin
                            //reset_button = 0;
                            if(state_cmpm == 1) begin
                                    state_test = state_test + 1;
                                    sub_state_test = 0;
                                    end
                            end
                  endcase
                  end
        4'b0011 : begin                                                 // for testing state 4
                  if(state_cmpm == 1)
                         state_test = state_test + 1;
                  end
        4'b0100 : begin                                                 // for testing state 5
                  if(state_cmpm == 1)
                         state_test = state_test + 1;
                  end
```

```
        4'b0101 : begin                                                 // for testing state 6
                  case(sub_state_test)
                  4'b0000 : begin
                            if(state_cmpm == 1)
                                    sub_state_test = sub_state_test + 1;
                            end
                  4'b0001 : begin
                            if(return_state == 4'b0001)
                                    state_test = 4'b0001;
                            else begin
                                    if(return_state == 4'b0110)
                                            state_test = 4'b0110;
                                    else
                                            state_test = 4'b0111;
                                    end
                            sub_state_test = 0;
                            end
                  endcase
                  end
        4'b0110 : begin                                                 // for testing state 7
                  if(state_cmpm == 1)
                          state_test = state_test + 1;
                  end
        4'b0111 : begin                                                 // for testing state 8
                  if(state_cmpm == 1)
                          state_test = state_test + 1;
                  end
        4'b1000 : begin                                                 // for testing state 9
                  case(sub_state_test)
                  4'b0000 : begin
                            sub_state_test = sub_state_test + 1;
                            end
                  4'b0001 : begin
                            sub_state_test = sub_state_test + 1;
                            end
                  4'b0010 : begin
                            sensor_t = 1;
                            sub_state_test = sub_state_test + 1;
                            end
                  4'b0011 : begin
                            sensor_t = 0;
                            if(state_cmpm == 1) begin
                                    state_test = state_test + 1;
                                    sub_state_test = 0;

                                     end
                             end
                  endcase
                  end
        4'b1001 : begin                                                  // for testing state 10
                  if(state_cmpm == 1)
                          state_test = 4'b0000;
                  end
        endcase
end
endmodule
```

Simulation Results

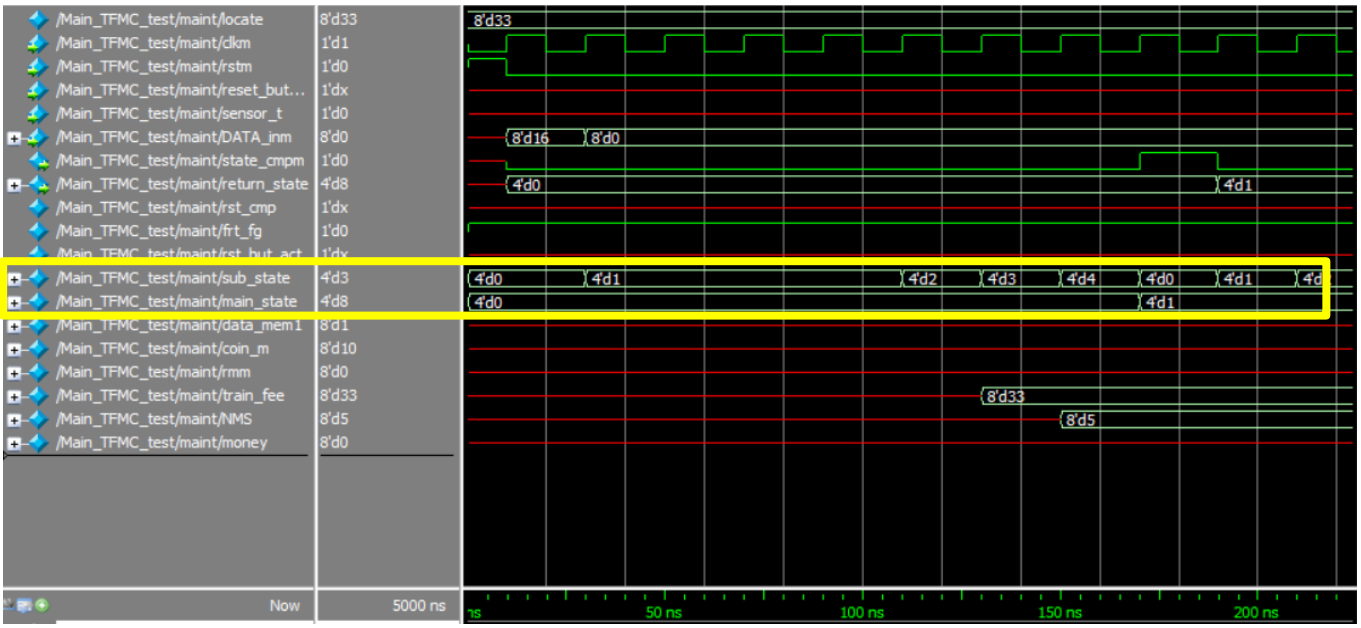


Fig 14: Output waveforms showing the state transition of the central state and sub-states

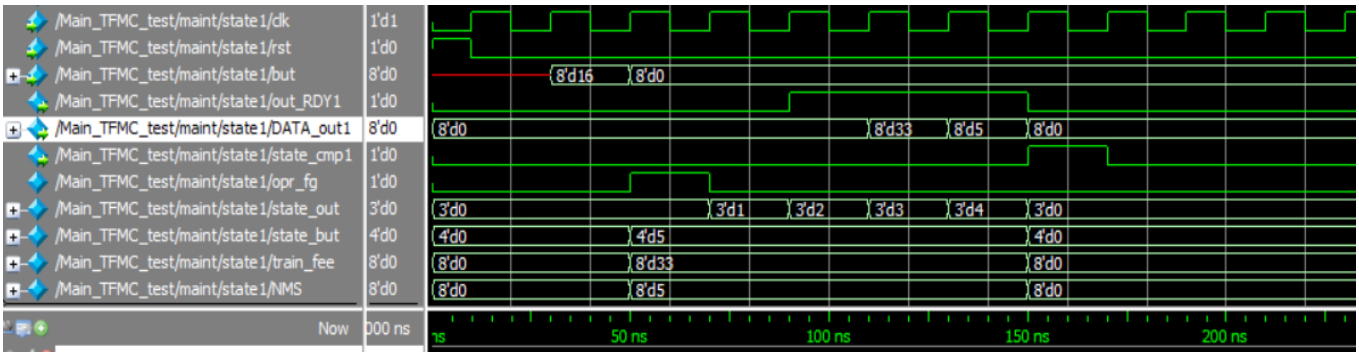


Fig 15: Output waveforms of State 1 showing button 4 when pressed and the train fee of 33 baht with five stations needed to travel

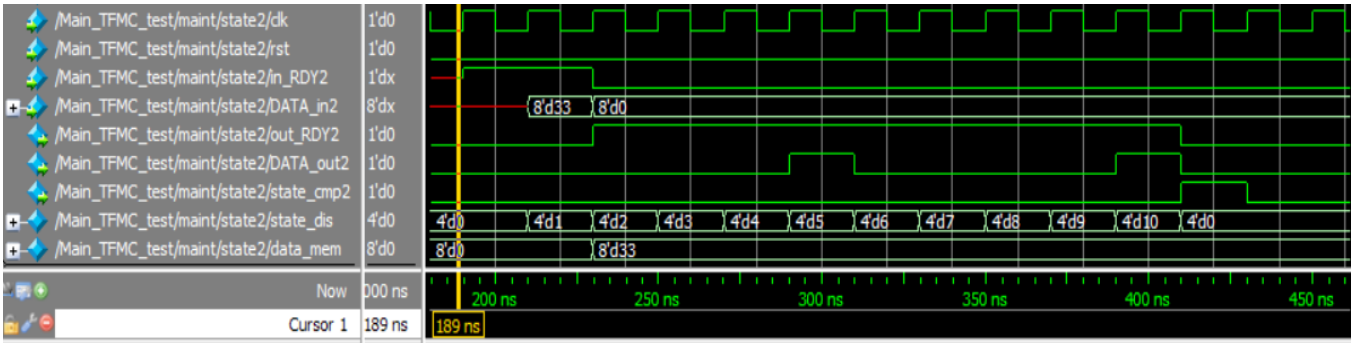


Fig 16: Output waveforms of State 2 showing the serial data of the amount of money to be displayed on the screen

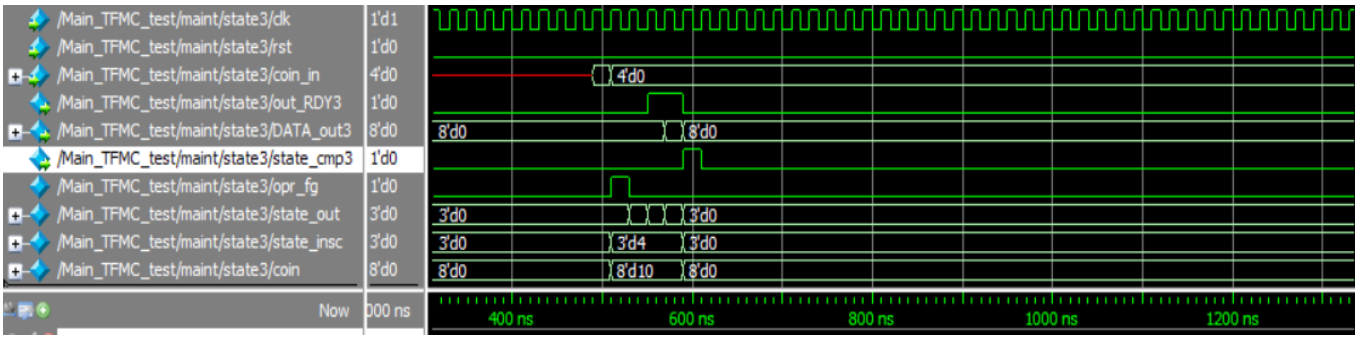


Fig 17: Output waveforms of State 3 showing the 10 baht coin inserted

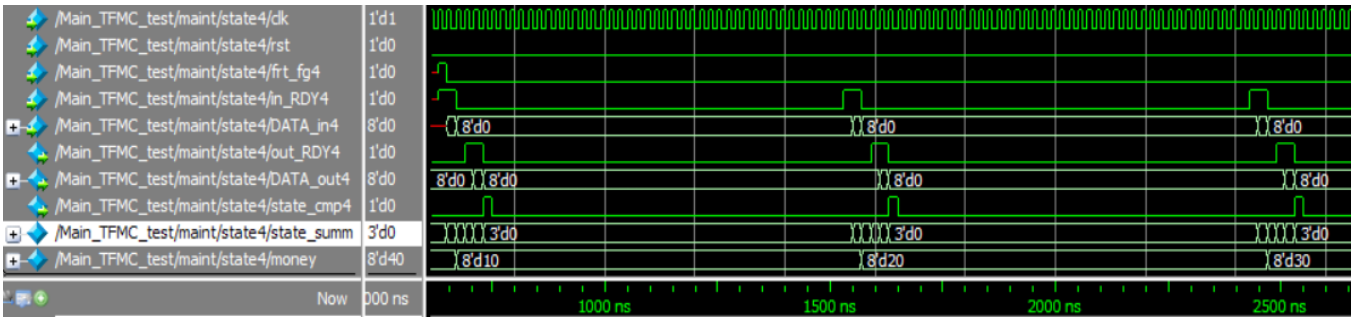


Fig 18: Output waveforms of State 4 showing the summation of coins when 10 baht is inserted multiple times

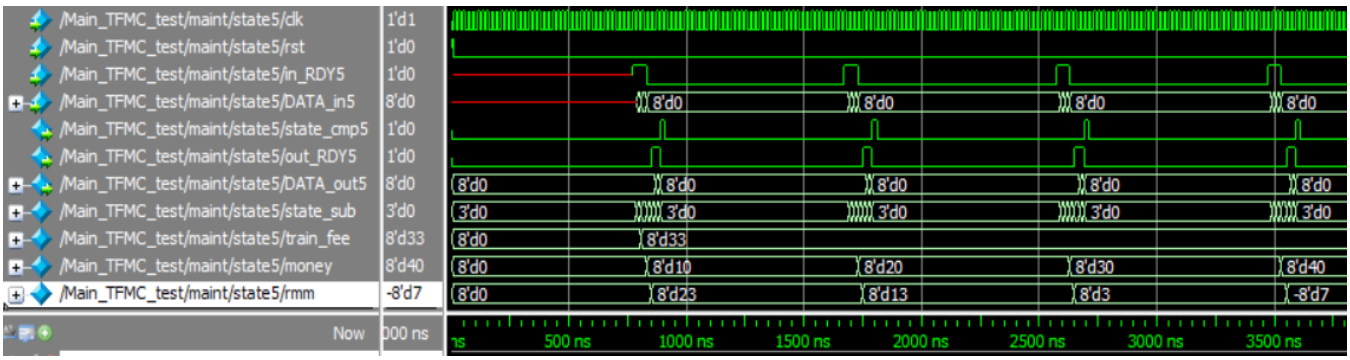


Fig 19: Output waveforms of State 5 showing the remaining money after subtracting the training fee when a 10 baht coin is inserted four times

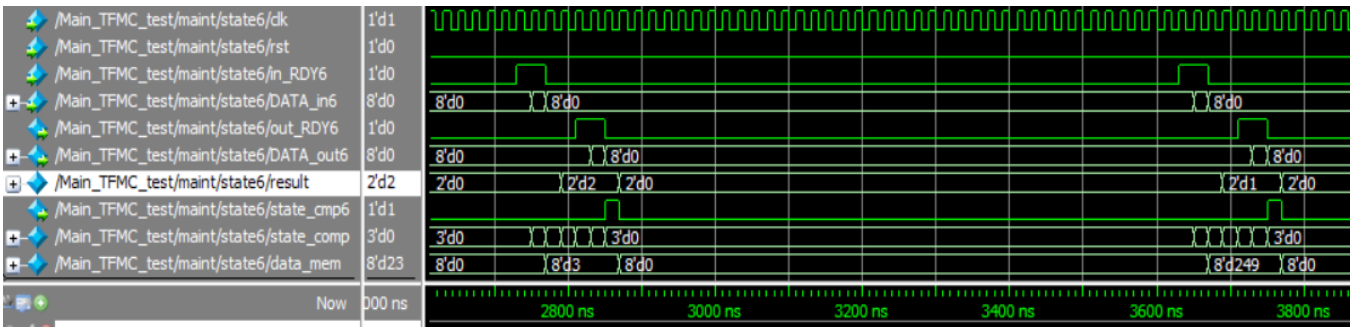


Fig 20: Output waveforms of State 6 showing the comparison to find out whether there is a change or not

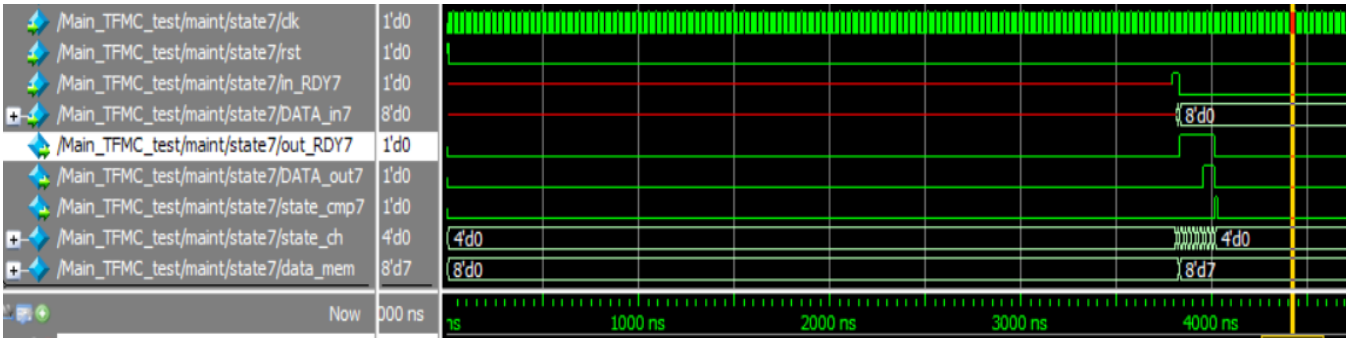


Fig 21: Output waveforms of State 7 showing the change of 7 baht given to the user

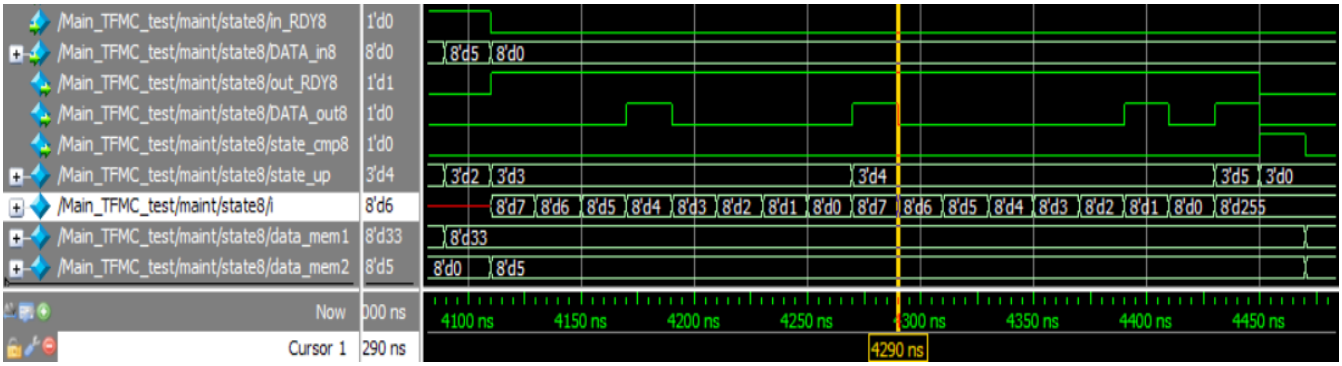


Fig 22: Output waveforms of State 8 showing the data (00100001) getting uploaded on the ticket where the first 3 bits denote the line number and the last 5 bits indicate the number of stations

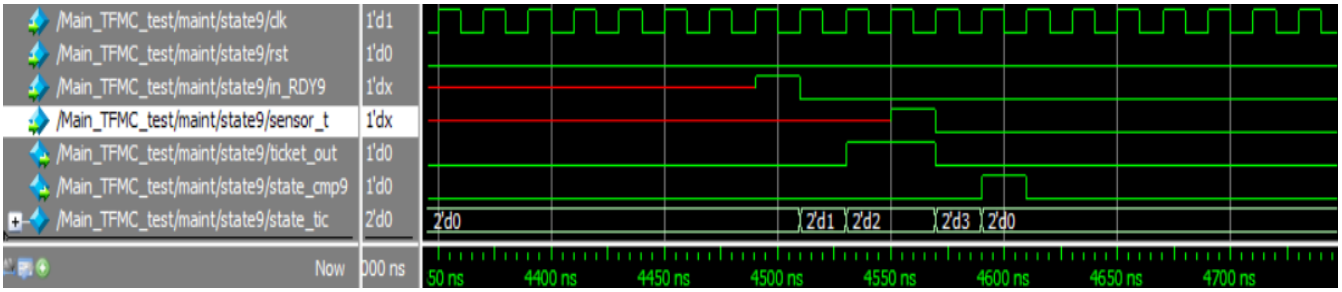


Fig 23: Output waveforms of State 9 showing that the user has taken the ticket by detecting it from a sensor

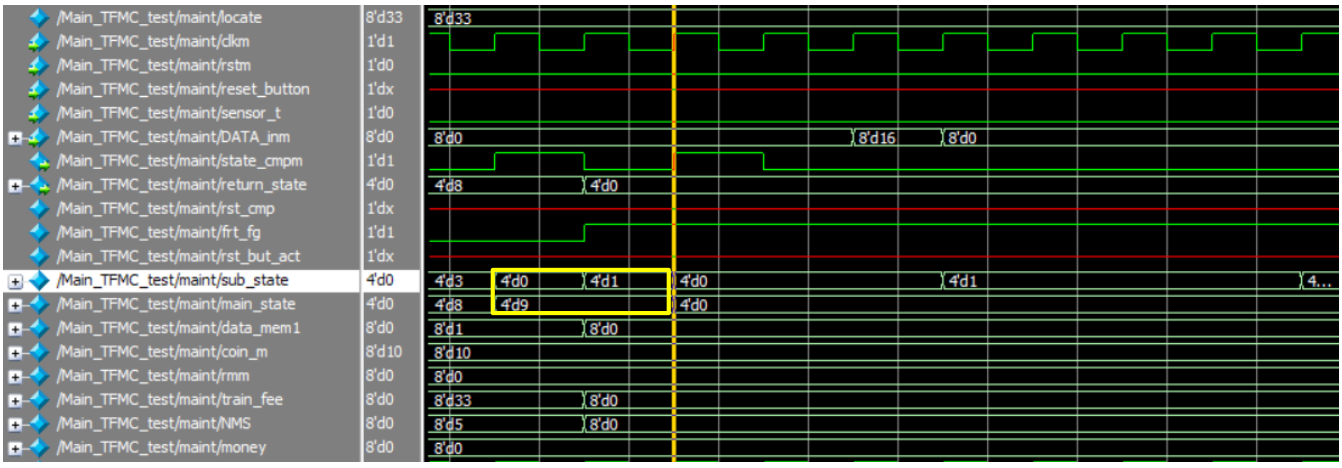


Fig 24: Output waveforms of State 10, which shows every register of the central state is cleared

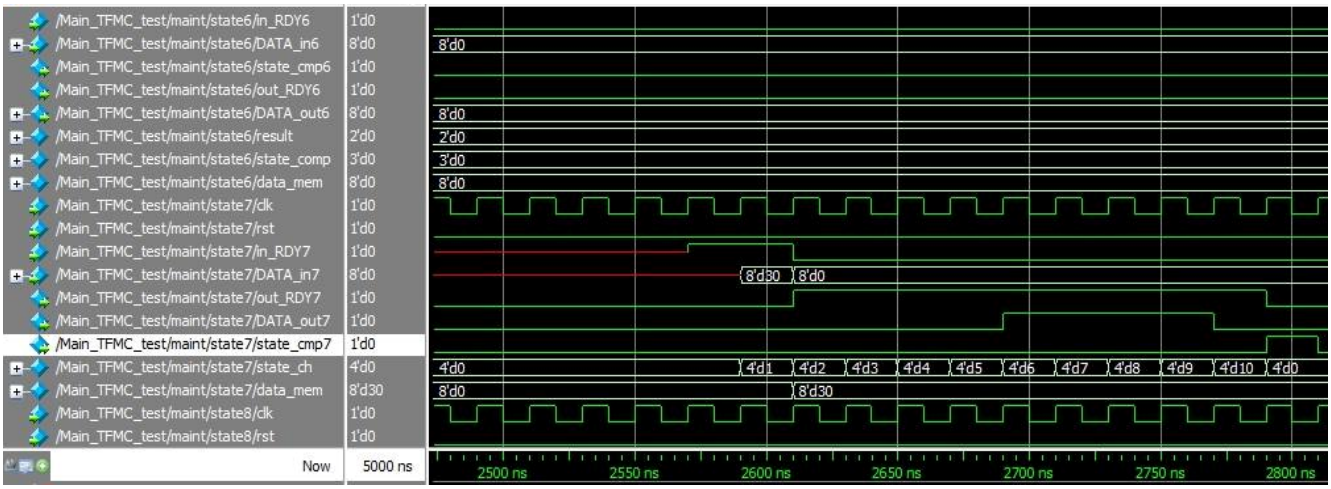


Fig 25: Output waveforms showing the change given to the user when the reset button is pressed

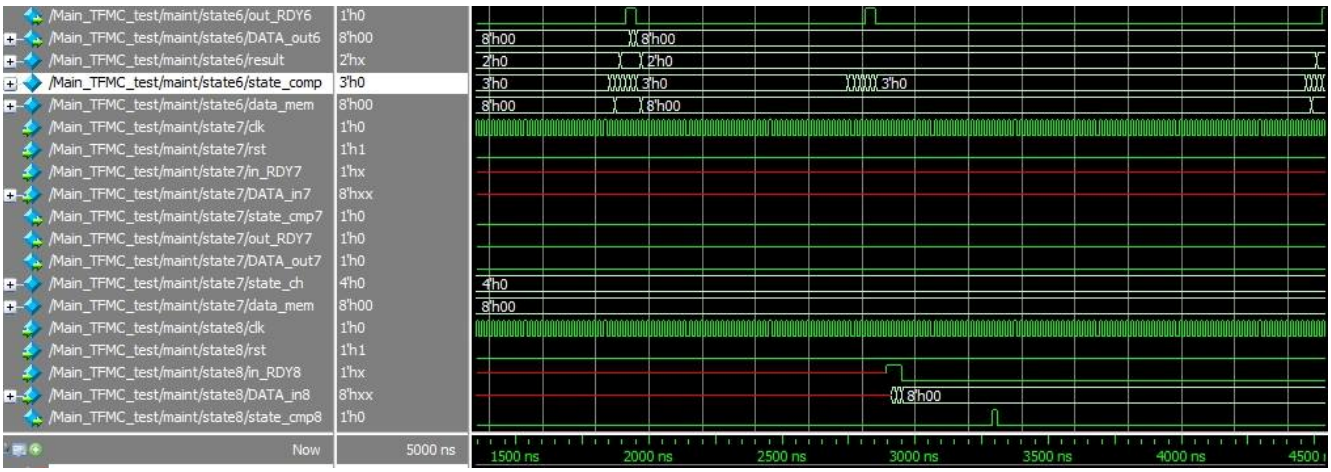


Fig 26: Output waveforms showing the result when the train fee is equal to the money inserted by the user

Conclusion

After seeing and verifying the output waveforms, we concluded that our train ticket vending machine using Verilog Hardware Description Language (HDL) and Moore Finite State Machine (FSM) approach has been achieved with all the objectives implemented. Thus, we can use this project in a real-world scenario where the passengers can get the ticket without standing in a long queue at the ticket counters.

Note: The definition of each state is attached to this report