Project (Car Parking System)

➢ Aim:

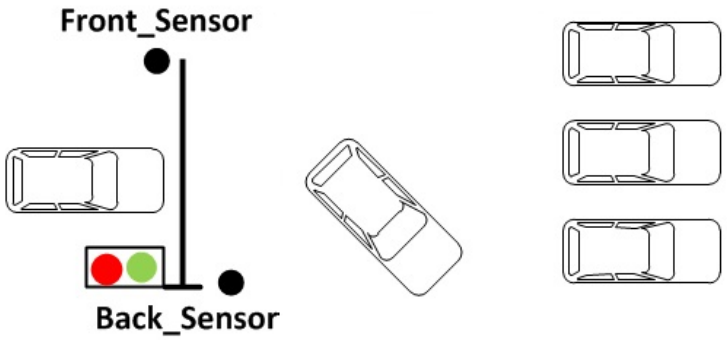
To design a full-fledged car parking system in Verilog using FSM.

➢ Hardware and Software used:

1. Altera Quartus II
2. Modelsim
3. FPGA board (DE-1)

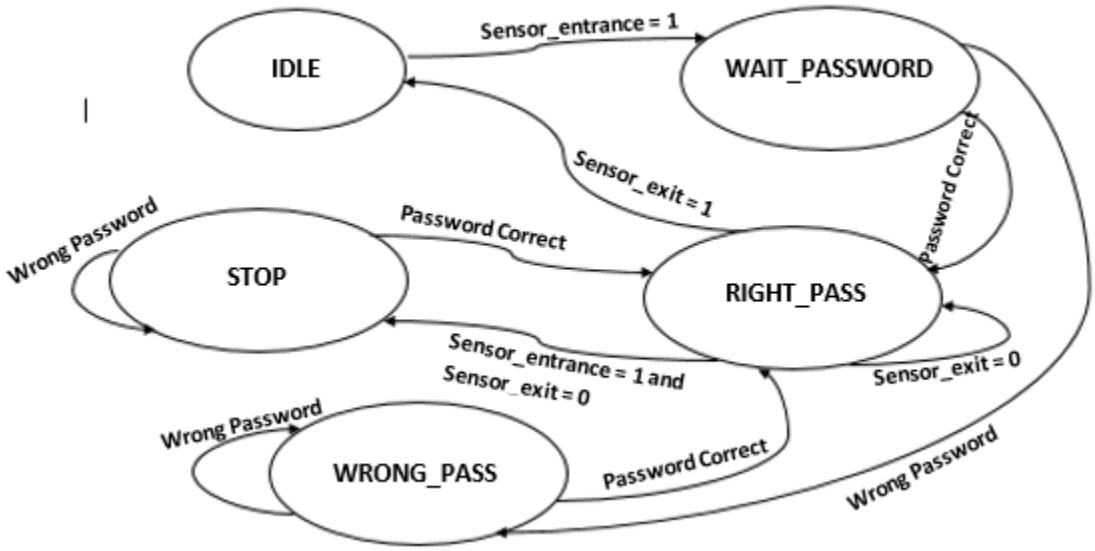
➢ Theory:

This Verilog project presents a car parking system in Verilog using Finite State Machine (FSM). Verilog code and testbench for the car parking system is designed.
The Verilog car parking system is shown in the following figure. There is a front sensor to detect vehicles going to the gate of the car parking system. Another back sensor is to detect if the coming vehicle passed the gate and getting into the car park.



Car Parking System

The car parking system in Verilog operates under the control of a Finite State Machine (FSM) as follows:



Initially, the FSM is in IDLE state. If there is a vehicle coming detected by the front sensor, FSM is switched to WAIT_PASSWORD state for 4 cycles. The car will input the password in this state; if the password is correct, the gate is opened to let the car get in the car park and FSM turns to RIGHT_PASS state; a Green LED will be blinking. Otherwise, FSM turns to WRONG_PASS state; a Red LED will be blinking and it requires the car to enter the password again until the password is correct. When the current car gets into the car park detected by the back sensor and there is the next car coming, the FSM is switched to STOP state and the Red LED will be blinking so that the next car will be noticed to stop and enter the password. After the car passes the gate and gets into the car park, the FSM returns to IDLE state.

➢ Verilog Code:

```
`timescale 1ns / 1ps
module parking_system(
            input clk,reset_n,
 input sensor_entrance, sensor_exit,
 input [1:0] password_1, password_2,
 output wire GREEN_LED,RED_LED,
 output reg [6:0] HEX_1, HEX_2
   );
 parameter IDLE = 3'b000, WAIT_PASSWORD = 3'b001, WRONG_PASS = 3'b010, RIGHT_PASS =
3'b011,STOP = 3'b100;
 // Moore FSM : output just depends on the current state
 reg[2:0] current_state, next_state;
 reg[31:0] counter_wait;
 reg red_tmp,green_tmp;
 // Next state
```

```
always @(posedge clk or negedge reset_n)
begin
if(~reset_n)
current_state = IDLE;
else
current_state = next_state;
end
// counter_wait
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
// change state
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
// LEDs and output, change the period of blinking LEDs here
always @(posedge clk) begin
case(current_state)
IDLE: begin
green_tmp = 1'b0;
red_tmp = 1'b0;
HEX_1 = 7'b1111111; // off
HEX_2 = 7'b1111111; // off
end
WAIT_PASSWORD: begin
green_tmp = 1'b0;
red_tmp = 1'b1;
HEX_1 = 7'b000_0110; // E
HEX_2 = 7'b010_1011; // n
end
WRONG_PASS: begin
green_tmp = 1'b0;
red_tmp = ~red_tmp;
HEX_1 = 7'b000_0110; // E
HEX_2 = 7'b000_0110; // E
end
RIGHT_PASS: begin
green_tmp = ~green_tmp;
red_tmp = 1'b0;
HEX_1 = 7'b000_0010; // 6
HEX_2 = 7'b100_0000; // 0
end
```

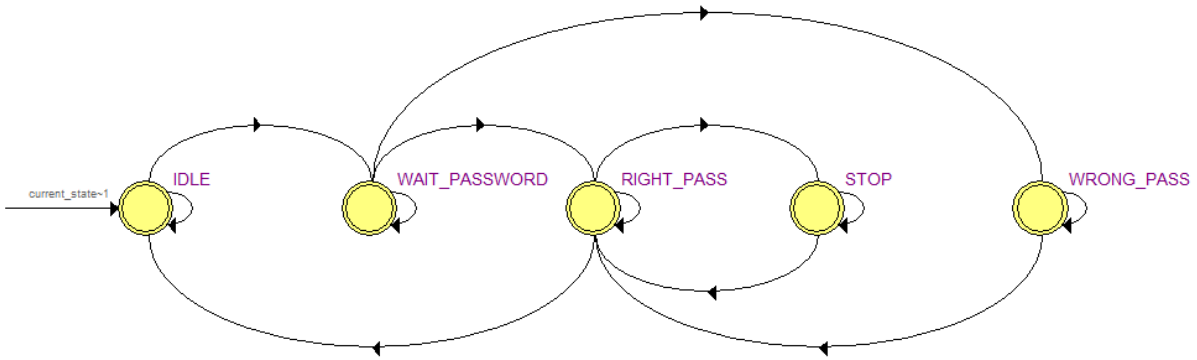
```
STOP: begin
green_tmp = 1'b0;
red_tmp = ~red_tmp;
HEX_1 = 7'b001_0010; // 5
HEX_2 = 7'b000_1100; // P
end
endcase
end
assign RED_LED = red_tmp  ;
assign GREEN_LED = green_tmp;

endmodule
```

- **Test bench Code:**

```
`timescale 1ns / 1ps
// Verilog project: Verilog code for car parking system
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),
```

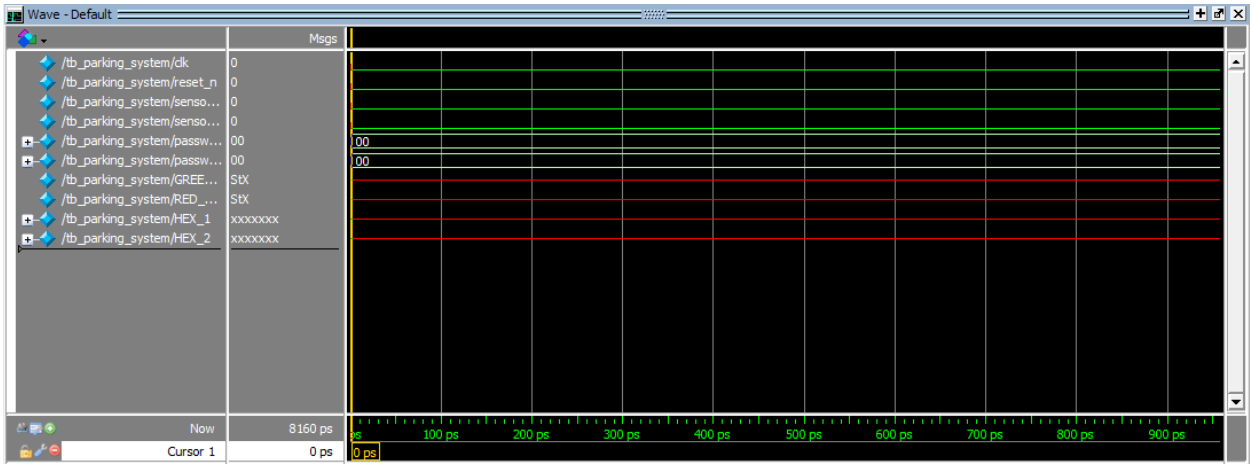
```
.HEX_2(HEX_2)
);
initial begin
clk = 0;
forever #10 clk = ~clk;
end
initial begin
// Initialize Inputs
reset_n = 0;
sensor_entrance = 0;
sensor_exit = 0;
password_1 = 0;
password_2 = 0;
// Wait 100 ns for global reset to finish
#100;
    reset_n = 1;
#20;
sensor_entrance = 1;
#1000;
sensor_entrance = 0;
password_1 = 1;
password_2 = 2;
#2000;
sensor_exit =1;

// Add stimulus here
end
endmodule
```

➢ State machine diagram:

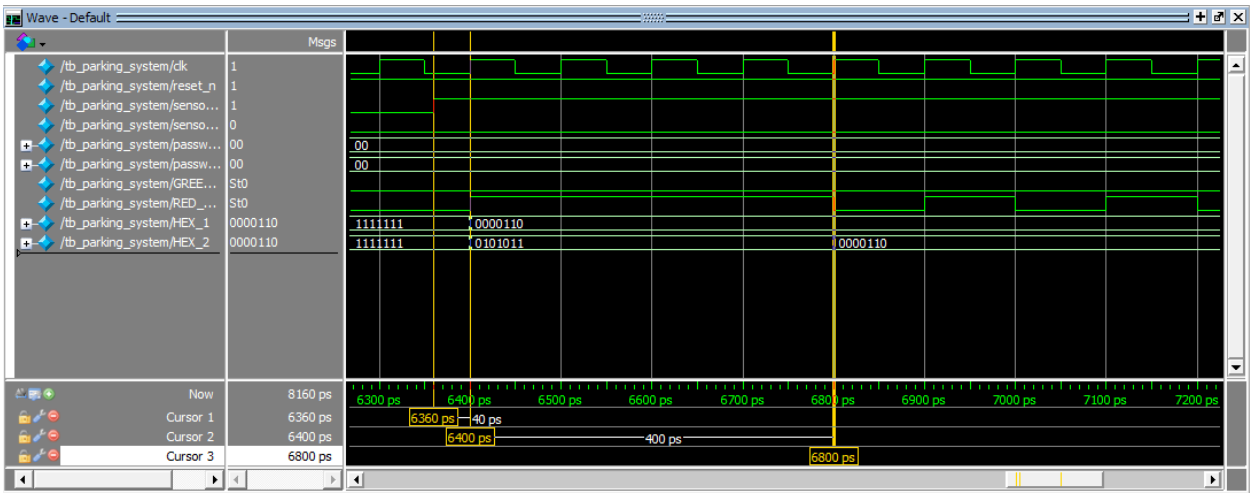


➢ Simulation waveform

1. Initial state

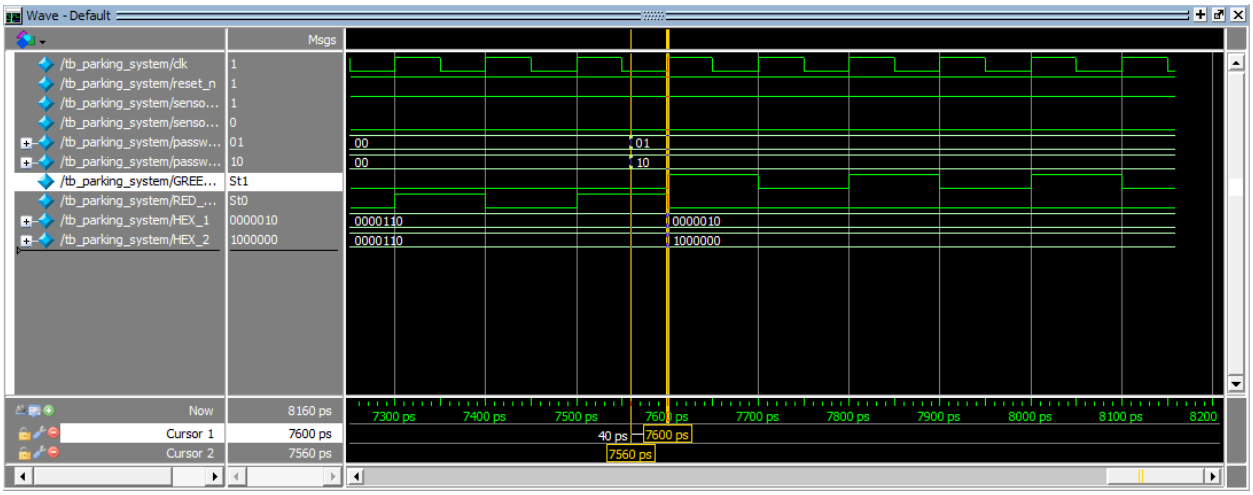


2. When Car is detected



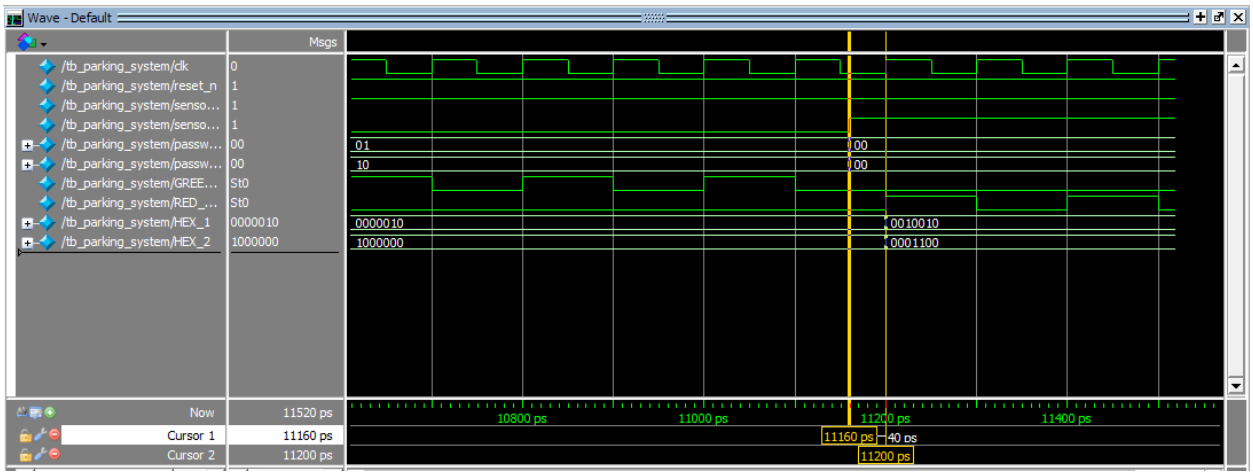
Here sensor_entrance = 1 at 6360ps (cursor 1), which indicates the car is detected. For next four cycles the system will wait for password entry (6400-6800ps (cursor 2 – cursor 3)). During this “wait” state red LED is ON. After that in this case password detected is wrong. So red LED keeps on blinking. Machine is in wrong_pass state.

3. When correct password is detected



Here, at 7560ps (Cursor 2) password variable obtains correct value. So, on next positive edge of clock (cursor 1) red LED stops blinking and green LED starts blinking which indicates that password is correct, gate is open, you can park your vehicle.

4. When next vehicle comes



Here, at 11160ps (Cursor 1) both sensor_entrance and sensor_exit are equal to one. Which indicates that the previous vehicle was parked and new vehicle is waiting for parking. We can see that at 11200ps (Cursor 2) green LED is turned OFF and red LED is turned on. Now, if correct password is entered then machine will move to right_pass state and will proceed accordingly. Otherwise it remains in “stop” state.

➢ Conclusion:

In this project we applied the concepts which we have studied in previous labs, to create a FSM Model for Car Parking System. We designed a code in Verilog which helps us in getting the desired output. The machine provides different outputs according to its present state.
This machine is Moore machine so output depends only on the current state.