UW Student: Brian Chen
EE 371
January 17, 2025
Lab 1B Report

I. Procedure

A. Task #1

First, I drew out the 2-output / 3-input state diagram for the entrance passcode as shown in Figure 1. Then, I first do the FSM code for "passcode.sv" at first. It's a 9-state Moore FSM.

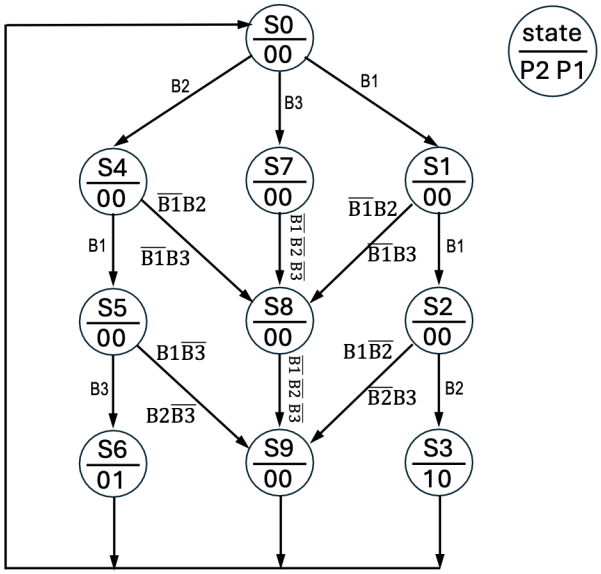

Figure 1: Moore finite state machine of "passcode" module.

Second, I worked on the counter which is "counter.sv" for counting how many cars have been entranced the parking lot.

Third, I used "twoDFF.sv" and which is a D flip-flop code from my EE271 previous lab and also "unserInput.sv" for holding the value that it will only count one time when KEY[3:1] is pressed, no matter how long it pressed (during many clock cycles).

The HDL code "userInput.sv is also contributed by a FSM, and Figure 2 shows the state diagram of the Moore finite-state machine for this module.

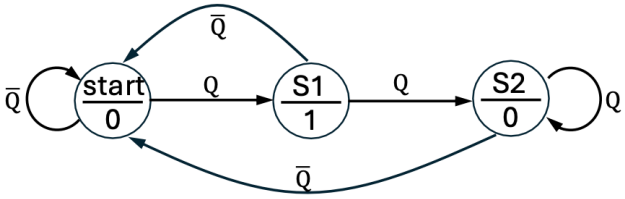

Figure 2: Moore finite state machine of "userInput" module.

Third, I created a higher-level file to integrate “passcode.sv”, “twoDFF.sv”, and “userInput.sv” which is passcode_all.sv.

Fourth, I contributed passcode_all_counter.sv, to implement calling “counter” method for counting number of cars entering the parking lots and sync with “passcode_all.sv”.

The trigger condition “dec” needs to go through the “twoDFF” module and the “userInput” module, so that it will not over count when the signal is stay HIGH in many clock cycles.

Fifth, I wrote “bcd7seg.sv” to driving the seven-segment displays HEX5, HEX4, HEX3, HEX2, HEX1, and HEX0.

Sixth, I contributed “top_cell.sv” for integrating “passcode_all_counter.sv” and “bcd7seg.sv”.

Seventh, I finished “DE1_SoC.sv” main code, which is easily call the method “top_cell”; however, when I tested it on LabsLand, the entrance gate was not working with it. So, I tried to add “signal_extender.sv’ given by Canvas. Then, pull out the entrance gate signal which is for V_GPIO[31] to extend.

GPIO Ports	I/O	Description
V_GPIO[28]	I	Presence Parking 1: Lets you know if there is a car in parking spot 1.
V_GPIO[29]	I	Presence Parking 2: Lets you know if there is a car in parking spot 2.
V_GPIO[30]	I	Presence Parking 3: Lets you know if there is a car in parking spot 3.
V_GPIO[23]	I	Presence at Entrance Gate: Lets you know if there is a car waiting at the entrance
V_GPIO[24]	I	Presence at Exit Gate: Lets you know if there is a car waiting at the exit.
V_GPIO[26]	O	LED Parking 1: Lets you change LED color of parking spot 1 (0 = green, 1 = red).
V_GPIO[27]	O	LED Parking 2: Lets you change LED color of parking spot 2 (0 = green, 1 = red).
V_GPIO[32]	O	LED Parking 3: Lets you change LED color of parking spot 3 (0 = green, 1 = red).
V_GPIO[34]	O	LED Full: Lets you change LED color of the full indicator LED (0 = green, 1 = red).
V_GPIO[31]	O	Open Entrance Gate: Opens the entrance gate. When you send a 1, the gate will stay open until a car enters the lot.
V_GPIO[33]	O	Open Exit Gate: Opens the exit gate. When you send a 1, the gate will stay open until a car leaves the lot.

Table 1: GPIO mapping and port descriptions.

II. Result

A. Task #1

So, what I did for testbench is test if I pressed “KEY2, KEY1, KEY3”, “KEY1, KEY1, KEY2” in order when there is a car waiting at entrance gate, and I see what happened with all the modules. After those, I checked when the car is leaving for a few times for checking if the counter and others module work as I thought. Figure 3 shows the simulated waveform of the 3D parking lot simulator

generated by Modelsim. For the demonstration of this design, I provide the recorded video. Please access the video via the hyperlink: https://www.youtube.com/watch?v=0ZIBwFXs_O8 .

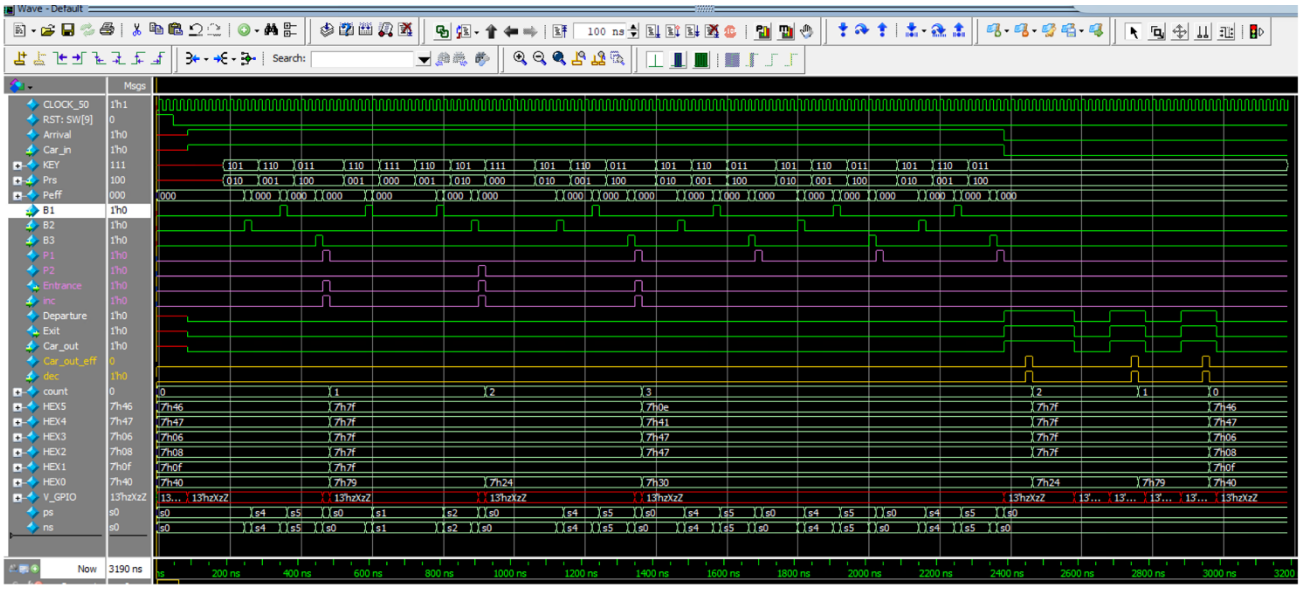

Figure 3: The simulated waveform of the 3D parking lot simulator generated by Modelsim.

III. Final Product

In this LAB, I became familiar with the GPIO interface and the LabsLand environment. The experimental results showed that by combining FPGA and GPIO, I could successfully control the parking and vehicle entry and exit status in the 3D parking lot simulator.

IV. Appendix

A. Block diagram

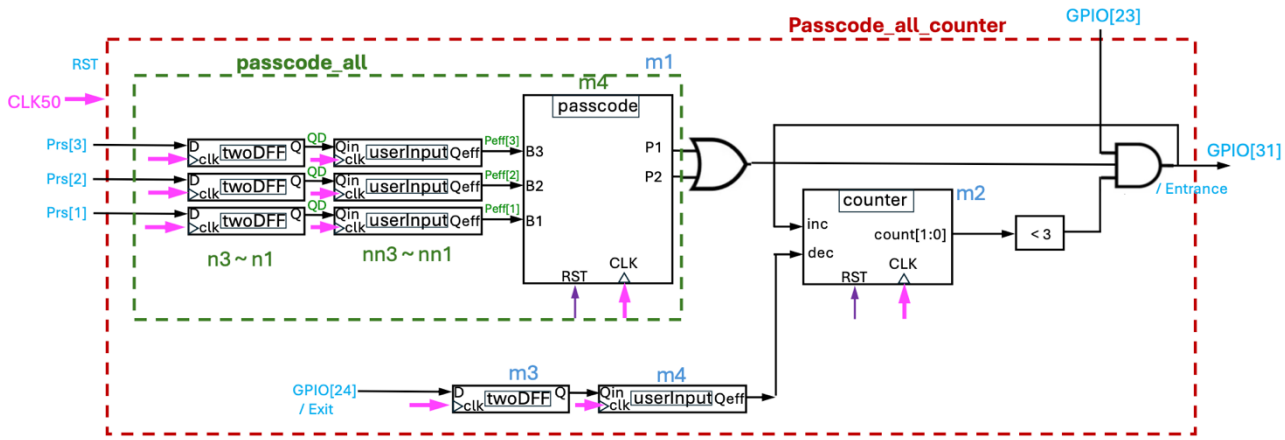

Figure 4: The block diagram of the module “passcode_all_counter”.

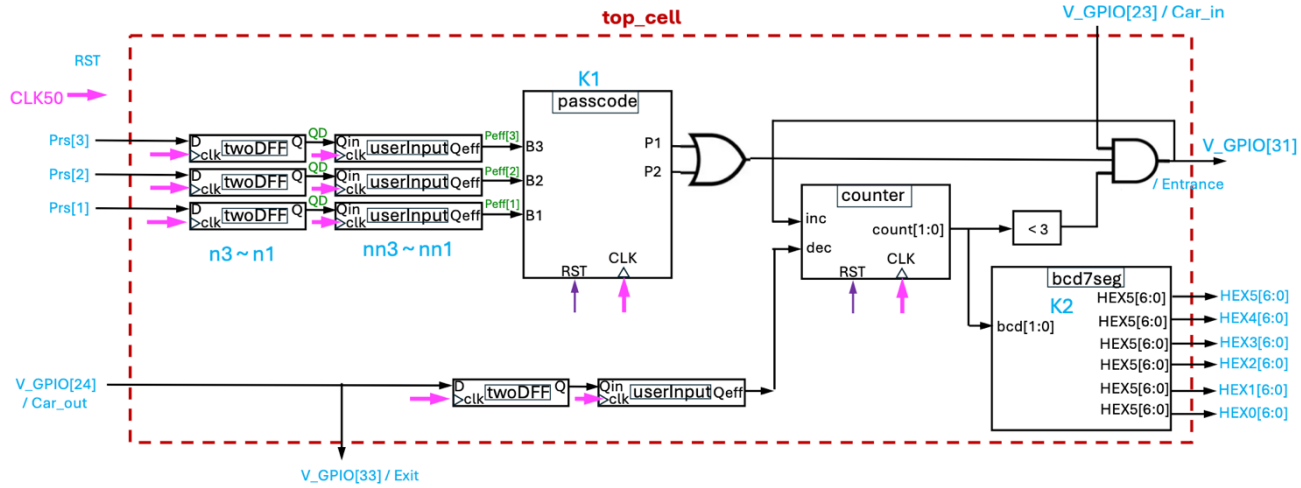

Figure 5: The block diagram of the module “top_cell”.

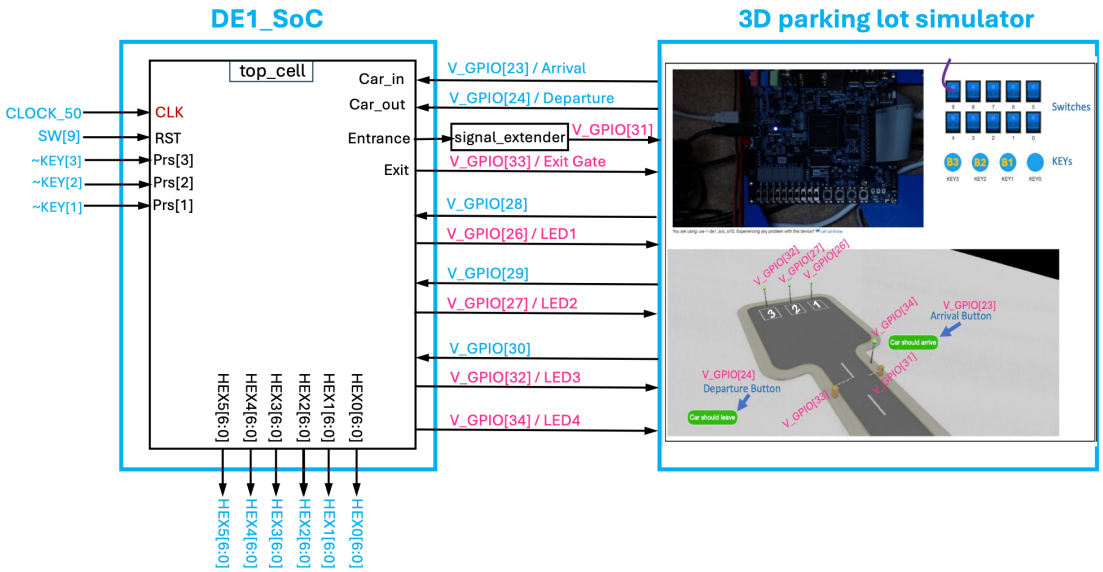

Figure 6: The block diagram of the whole design.

B. DE1_SoC.sv

```
module DE1_SoC(HEX5, HEX4, HEX3, HEX2, HEX1, HEX0, KEY, SW, LEDR, V_GPIO, CLOCK_50);
// define ports
    output logic [6:0]  HEX5, HEX4, HEX3, HEX2, HEX1, HEX0;
    output logic [9:0]  LEDR;
    input  logic [3:1]  KEY;
    input  logic [9:0]  SW;
    input logic CLOCK_50;
    inout logic [35:23] V_GPIO;

    logic hold, hold2; // for signal_extender

    top_cell top (.CLK(CLOCK_50), .RST(SW[9]), .Prs(~KEY),
                  .Car_in(V_GPIO[23]), .Car_out(V_GPIO[24]),
              .Entrance(hold), .Exit(V_GPIO[33]),
                 .HEX5, .HEX4, .HEX3, .HEX2, .HEX1, .HEX0);

    signal_extender s1(.in(hold), .out(hold2), .reset(SW[9]), .clk(CLOCK_50));

assign V_GPIO[31] = hold2;
assign V_GPIO[26] = V_GPIO[28];  // specify the color of LED1.
assign V_GPIO[27] = V_GPIO[29];  // specify the color of LED2.
assign V_GPIO[32] = V_GPIO[30];  // specify the color of LED3.

// specify the color (0:green / 1:red) of the full indicaror LED.
assign V_GPIO[34] = V_GPIO[28] & V_GPIO[29] & V_GPIO[30];

endmodule // DE1_SoC
```

C. top_cell.sv

```
module top_cell (CLK, RST, Prs, Car_in, Car_out, Entrance, Exit, HEX5, HEX4, HEX3, HEX2, HEX1, HEX0);
    input logic CLK, RST;
   input logic [3:1] Prs;
   input logic Car_in, Car_out;
   output logic Entrance, Exit;
   output logic [6:0] HEX5, HEX4, HEX3, HEX2, HEX1, HEX0;
   logic [1:0] count;


    passcode_all_counter K1 (.CLK, .RST, .Prs, .Car_in, .Car_out, .Entrance, .Exit, .count);

    bcd7seg              K2 (.bcd(count), .HEX5, .HEX4, .HEX3, .HEX2, .HEX1, .HEX0);

endmodule
```

D. passcode_all_counter.sv

```
module passcode_all_counter (CLK, RST, Prs, Car_in, Car_out, Entrance, Exit, count);
    input logic CLK, RST;
   input logic [3:1] Prs;
   input logic Car_in, Car_out;
    output logic Entrance, Exit;
   output logic [1:0] count;

   logic Car_out_QD, Car_out_eff;
   logic P1, P2;

   assign Entrance = Car_in & (P1 | P2) & (count < 3);


   assign Exit = Car_out;

    passcode_all m1 (.CLK(CLK), .RST(RST), .Prs(Prs), .P2(P2), .P1(P1));
    counter      m2 (.clk(CLK), .reset(RST), .inc(Entrance), .dec(Car_out_eff), .count(count));
    twoDFF       m3 (.CLK(CLK), .RST(RST), .D(Car_out), .Q(Car_out_QD));
    userInput    m4 (.CLK(CLK), .RST(RST), .Qin(Car_out_QD), .Qeff(Car_out_eff));
endmodule
```

E. bcd7seg.sv

```
6   module bcd7seg(bcd, HEX5, HEX4, HEX3, HEX2, HEX1, HEX0);
7   input logic [1:0] bcd;
8   output logic [6:0] HEX5, HEX4, HEX3, HEX2, HEX1, HEX0;
9
10  parameter [6:0] Blank = 7'b111_1111; // 7'h7f
11  parameter [6:0] A = 7'b000_1000; // 7'h08
12  parameter [6:0] C = 7'b100_0110; // 7'h46
13  parameter [6:0] E = 7'b000_0110; // 7'h06
14  parameter [6:0] F = 7'b000_1110; // 7'h0e
15  parameter [6:0] L = 7'b100_0111; // 7'h47
16  parameter [6:0] r = 7'b000_1111; // 7'h0f
17  parameter [6:0] U = 7'b100_0001; // 7'h41
18  parameter [6:0] Chr_0 = 7'b100_0000;  // 7'h40
19  parameter [6:0] Chr_1 = 7'b111_1001;  // 7'h79
20  parameter [6:0] Chr_2 = 7'b010_0100;  // 7'h24
21  parameter [6:0] Chr_3 = 7'b011_0000;  // 7'h30
22
23  always_comb begin
24  case(bcd)
25      3'b000:  {HEX5, HEX4, HEX3, HEX2, HEX1, HEX0} = {C, L, E, A, r, Chr_0};
26      3'b001:  {HEX5, HEX4, HEX3, HEX2, HEX1, HEX0} = {Blank, Blank, Blank, Blank, Blank,
    Chr_1};
27      3'b010:  {HEX5, HEX4, HEX3, HEX2, HEX1, HEX0} = {Blank, Blank, Blank, Blank, Blank,
    Chr_2};
28      3'b011:  {HEX5, HEX4, HEX3, HEX2, HEX1, HEX0} = {F, U, L, L, Blank, Chr_3};
29     default: {HEX5, HEX4, HEX3, HEX2, HEX1, HEX0} = {Blank, Blank, Blank, Blank, Blank,
    Blank};
30  endcase
31  end
32
33  endmodule
```

F. passcode_all.sv

```
1   module passcode_all (CLK, RST, Prs, P2, P1);
2       input logic CLK, RST;
3      input logic [3:1] Prs;
4       output logic P2, P1;
5
6      logic [3:1] QD, Peff;
7
8       twoDFF     n3 (.CLK(CLK), .RST(RST), .D(Prs[3]), .Q(QD[3]));
9      twoDFF     n2 (.CLK(CLK), .RST(RST), .D(Prs[2]), .Q(QD[2]));
10      twoDFF     n1 (.CLK(CLK), .RST(RST), .D(Prs[1]), .Q(QD[1]));
11      userInput nn3 (.CLK(CLK), .RST(RST), .Qin(QD[3]), .Qeff(Peff[3]));
12      userInput nn2 (.CLK(CLK), .RST(RST), .Qin(QD[2]), .Qeff(Peff[2]));
13      userInput nn1 (.CLK(CLK), .RST(RST), .Qin(QD[1]), .Qeff(Peff[1]));
14
15      passcode m4 (.CLK(CLK), .RST(RST), .B3(Peff[3]), .B2(Peff[2]), .B1(Peff[1]), .P2(P2),
    .P1(P1));
16
17  endmodule
```

G. counter.sv

```
1   module counter(clk, reset, inc, dec, count);
2
3      input logic clk, reset, inc ,dec;
4      output logic [1:0] count;
5
6      //counter logic
7
8      always_ff @(posedge clk or posedge reset) begin
9            if (reset)
10              count <= 2'b00; // reset count to 0
11
12           else begin
13              if (inc && !dec && count < 2'b11)
14                 count <= count + 1; // Increment if below max
15
16              else if (dec && !inc && count > 2'b00)
17                 count <= count - 1 ; // Decrement iff above 0
18
19           end
20     end
21
22  endmodule
23
```

H. twoDFF.sv

```
6   module twoDFF (CLK, RST, D, Q);
7   input logic CLK, RST;
8   input logic D;
9   output logic Q;
10  logic Q1;
11
12      // DFFs
13      always_ff @(posedge CLK or posedge RST) begin
14          if (RST) begin
15            Q  <= 0;
16           Q1 <=0;
17        end
18          else begin
19            Q  <= Q1;
20           Q1 <= D;
21         end
22      end
23
24  endmodule
25
```

I. userInput.sv

```
/ ------------------------------------------------------------------- /
module userInput (CLK, RST, Qin, Qeff);
input logic CLK, RST;
input logic Qin;
output logic Qeff;

// State variables
    typedef enum logic [1:0] {
        start, s1, s2
    } state_t;
    state_t ps, ns;

    // Next State logic
always_comb begin
    ns = start; // Default to avoid latches
    case (ps)
        start: if (Qin == 1) ns = s1;
               else ns = start;

        s1:    if (Qin == 1) ns = s2;
               else ns = start;

      s2:    if (Qin == 1) ns = s2;
               else ns = start;
       default: ns = start;
    endcase
end

    // Output logic
    always_comb begin
        Qeff = 0;  // Default to avoid latches
        case (ps)
            start: Qeff = 0;
            s1:    Qeff = 1;
            s2:    Qeff = 0;
            default: Qeff = 0;
        endcase
    end

    // DFFs
    always_ff @(posedge CLK or posedge RST) begin
        if (RST)
            ps <= start;
        else
            ps <= ns;
    end

endmodule
```

J. passcode.sv

```
module passcode (CLK, RST, B3, B2, B1, P2, P1);
    input logic CLK, RST, B3, B2, B1;
    output logic P2, P1;

// State variables
    typedef enum logic [3:0] {
    s0, s1, s2, s3, s4, s5, s6, s7, s8, s9
    } state_t;
    state_t ps, ns;

    // assign ps_out = ps;

    // Next State logic
always_comb begin
    ns = s0; // Default to avoid latches
    case (ps)
        s0:    if (B1 == 1)
                   ns = s1;
               else if (B2 == 1)
                ns = s4;
            else if (B3 == 1)
                ns = s7;
               else ns = s0;

        s1:    if (B1 == 1)
                   ns = s2;
               else if ((B2 == 1) || (B3 == 1))
                ns = s8;
               else ns = s1;

        s2:    if (B2 == 1)
                   ns = s3;
               else if ((B1 == 1) || (B3 == 1))
                ns = s9;
               else ns = s2;

        s3:    ns = s0;

        s4:    if (B1 == 1)
                   ns = s5;
               else if ((B2 == 1) || (B3 == 1))
                ns = s8;
               else ns = s4;

        s5:    if (B3 == 1)
                   ns = s6;
               else if ((B1 == 1) || (B2 == 1))
                ns = s6;
               else ns = s5;

        s6:    ns = s0;

        s7:    if ((B1 == 0) && (B2 == 0) && (B3 == 0))
                ns = s7;
            else
               ns = s8;

        s8:    if ((B1 == 0) && (B2 == 0) && (B3 == 0))
                ns = s8;
            else
               ns = s9;

        s9:    ns = s0;

       default: ns = s0;
    endcase
end

    // Output logic
    always_comb begin
        {P2, P1} = 2'b00;  // Default to avoid latches
        case (ps)
            s0: {P2, P1} = 2'b00;
            s1: {P2, P1} = 2'b00;
            s2: {P2, P1} = 2'b00;
```

```
76                  s3: {P2, P1} = 2'b10;
77                  s4: {P2, P1} = 2'b00;
78                  s5: {P2, P1} = 2'b00;
79                  s6: {P2, P1} = 2'b01;
80                  s7: {P2, P1} = 2'b00;
81                  s8: {P2, P1} = 2'b00;
82                  s9: {P2, P1} = 2'b00;
83                  default: {P2, P1} = 2'b00;
84              endcase
85          end
86          // DFFs
87
88          always_ff @(posedge CLK or posedge RST) begin
89              if (RST)
90                  ps <= s0;
91              else
92                  ps <= ns;
93          end
94
95      endmodule
96
```