Bert Zhao
EE 371

Abstract: The purpose of this lab is to explore a general review of finite state machines (FSMs). The skills and concepts covered in this lab are representative of the final few labs of EE 271.

Introduction: The objective of this lab is to create a parking lot occupancy counter. We need to design a system for a pair of sensors at a parking lot gate so that we can monitor the traffic of cars that enter and exit the lot. In addition, we must also implement a system that keeps track of the total number of cars in the parking lot at any given time.

Materials:
- **Altera Terasic DE1-SoC Board with power and USB cables**
- **PC with Altera Quartus Prime Lite installed**
- **Red and green LEDs**

Procedure: Parking Lot Occupancy Counter
There are three major modules for the implementation of this design. The first module is an FSM that controls the logic of sensors at the entrance of the parking lot. The inputs are two sensors controlled by KEY[0] and KEY[1]. When the sensors are tripped by a car in a "rolling" pattern (First neither KEY[0] nor KEY[1], then just, KEY[0], then both KEY[0] and KEY[1], then just KEY[1], then back to neither KEY[0] nor KEY[1]), the output signal enter will be high for one clock cycle. The reverse sequence triggers the exit signal for one clock cycle. The secondary outputs are two off board LEDs. When KEY[0] is pressed, a red LED is on and when KEY[1] is pressed, a green LED is on. When reset is on, the module does not receive any inputs from KEY[1] and KEY[0].

State Diagram:

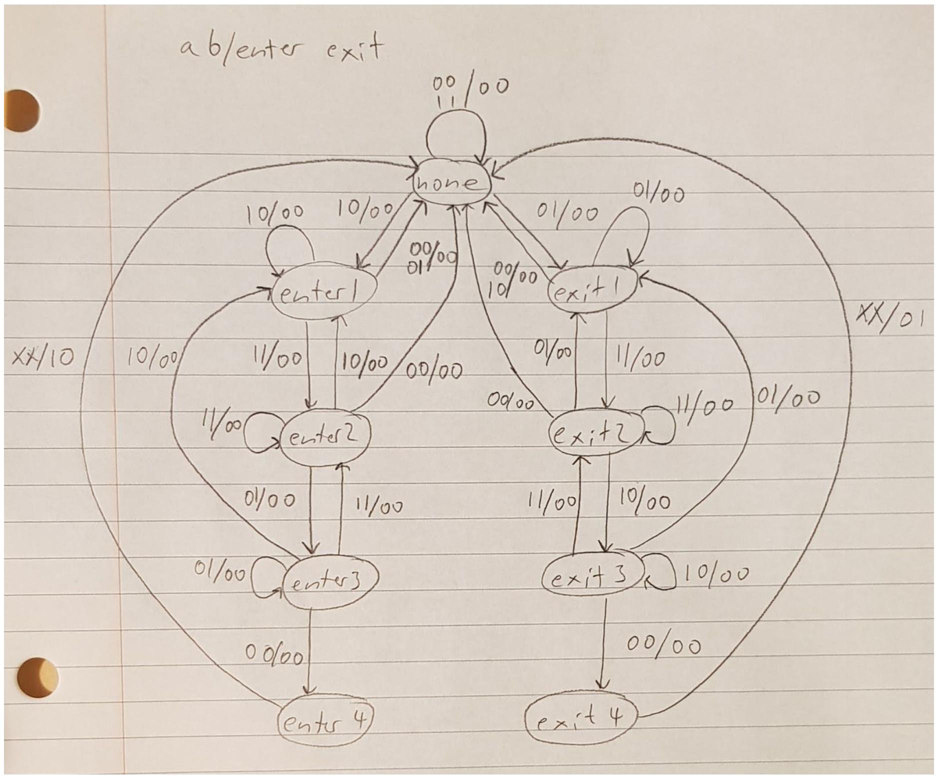


Figure 1: Sensors state diagram

The second module is a counter for the total number of cars in the parking lot. It takes the enter and exit signals from the FSM and outputs information for the total number of cars in the parking lot. Every time an enter or exit signal is received, the counter respectively increments or decrements the number of cars in the parking lot. The counter is a general 0 to 99 circular up/down counter, but it has been limited to a range of 0 to 25 with the circular reset disabled for this lab. The counter outputs two sets of information to the display. The first are two 4-bit numbers for the tens and ones digit. The second is a 5-bit number that tracks the total number of cars in the parking lot. When reset is on, all outputs are set to zero.

The third modules receives counter number and the total number of cars and outputs to all six seven segment displays. The display shows the number of cars in the parking lot ranging from 0 to 25 on HEX1 and HEX0. When the parking lot has no cars, the word “EmPty” is displayed on HEX5 through HEX1 next to the 0 on HEX0. When the parking lot is full, the word “FULL” is displayed on HEX5 through HEX2 next to 25 on HEX1 and HEX0. When reset is on, the display is locked to “EmPty0.”

Results and Analysis:
All modules in this lab were tested using ModelSim. The parking lot gate FSM was completed in nine states because it treats both the enter and exit processes as their own separate branch as seen in the state diagram. This design accounts for the factor that some cars may change direction in the process of entering or exiting. Therefore, if a car changes directions during. the entering or exiting process, the counter will maintain the previous number of cars in the parking lot. The overall inputs are SW[9] for reset, KEY[1] for sensor a, and KEY[0] for sensor b. The overall outputs are HEX5 through HEX0 for the counter display and the red and green LEDs for the sensor status. The simulation waveforms, block diagram, and flow summary are included below.

ModelSim Screenshots:

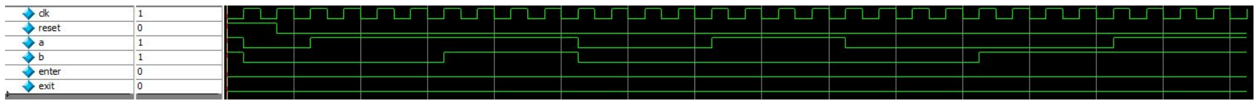


Figure 2: lotGate ModelSim screenshot

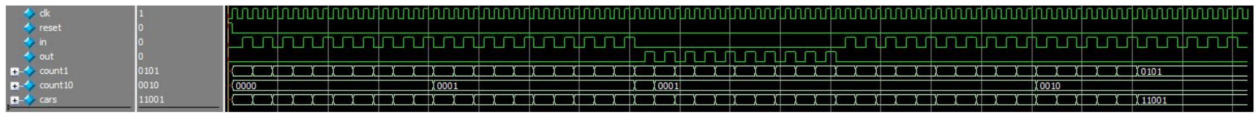


Figure 3: counter ModelSim screenshot

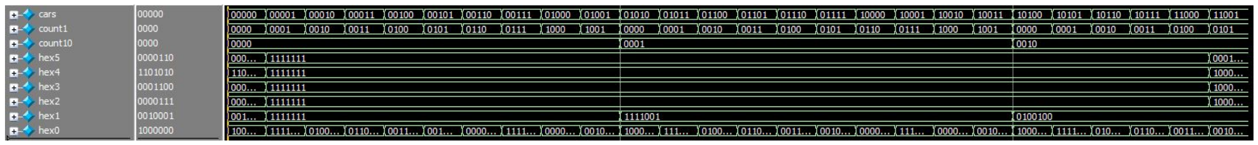


Figure 4: display ModelSim screenshot

Block Diagrams:

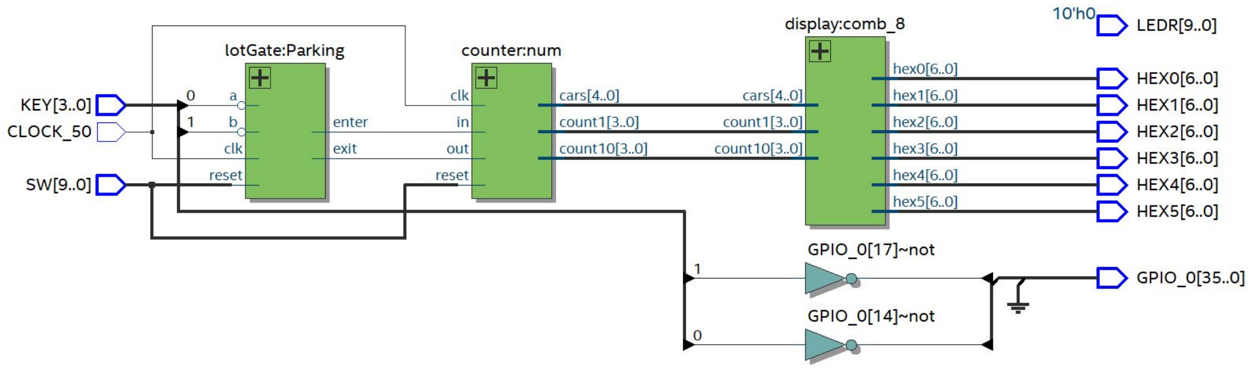
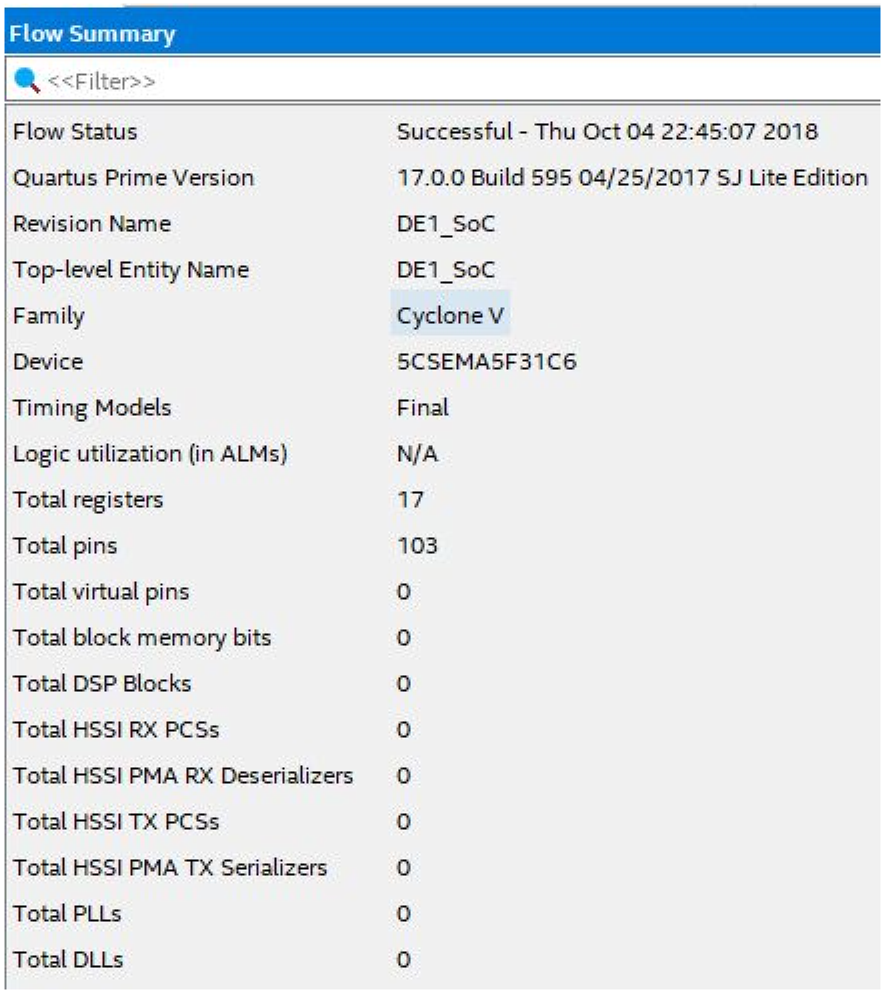


Figure 5: DE1_SoC block diagram

Flow Summary



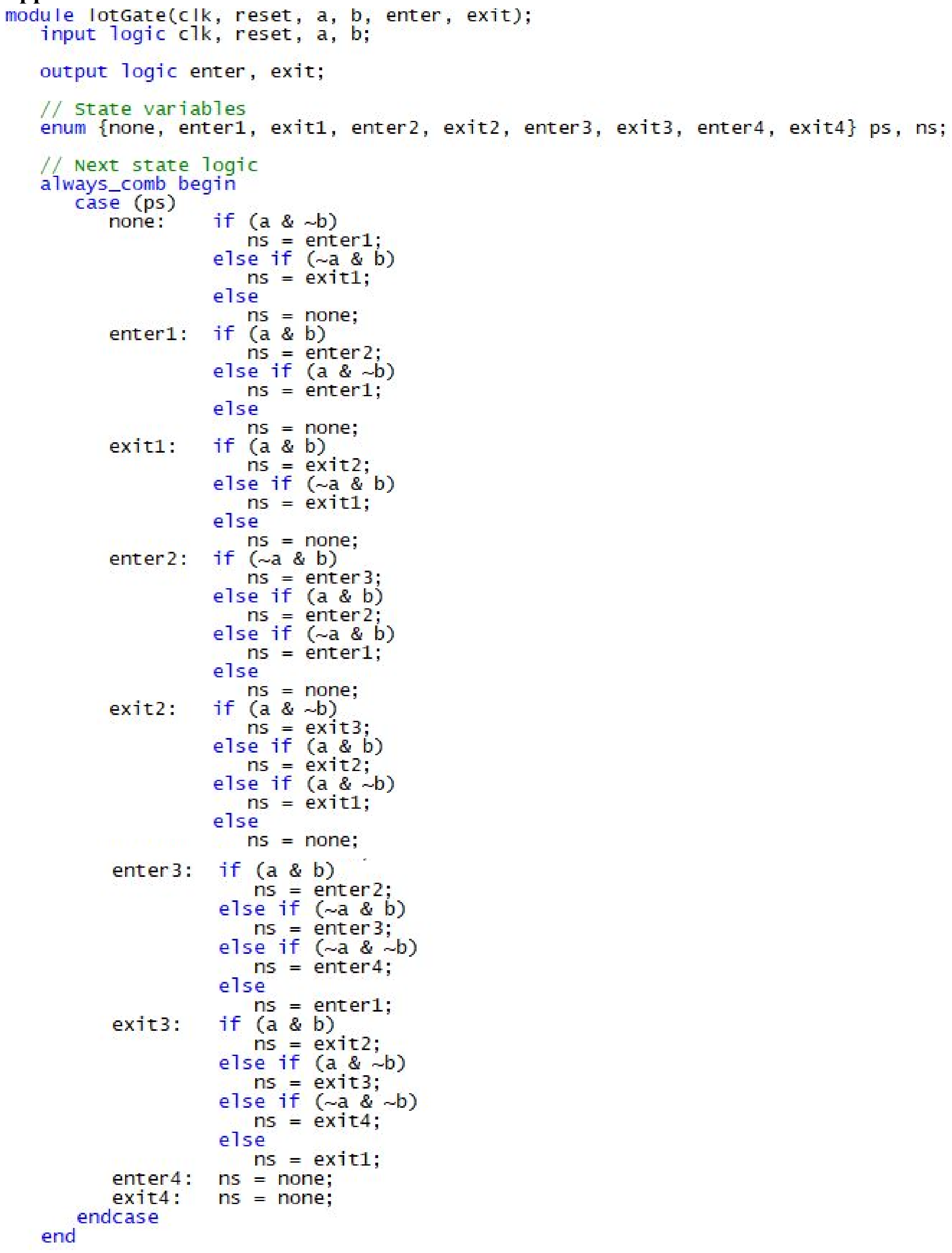
Flow Summary

<<Filter>>

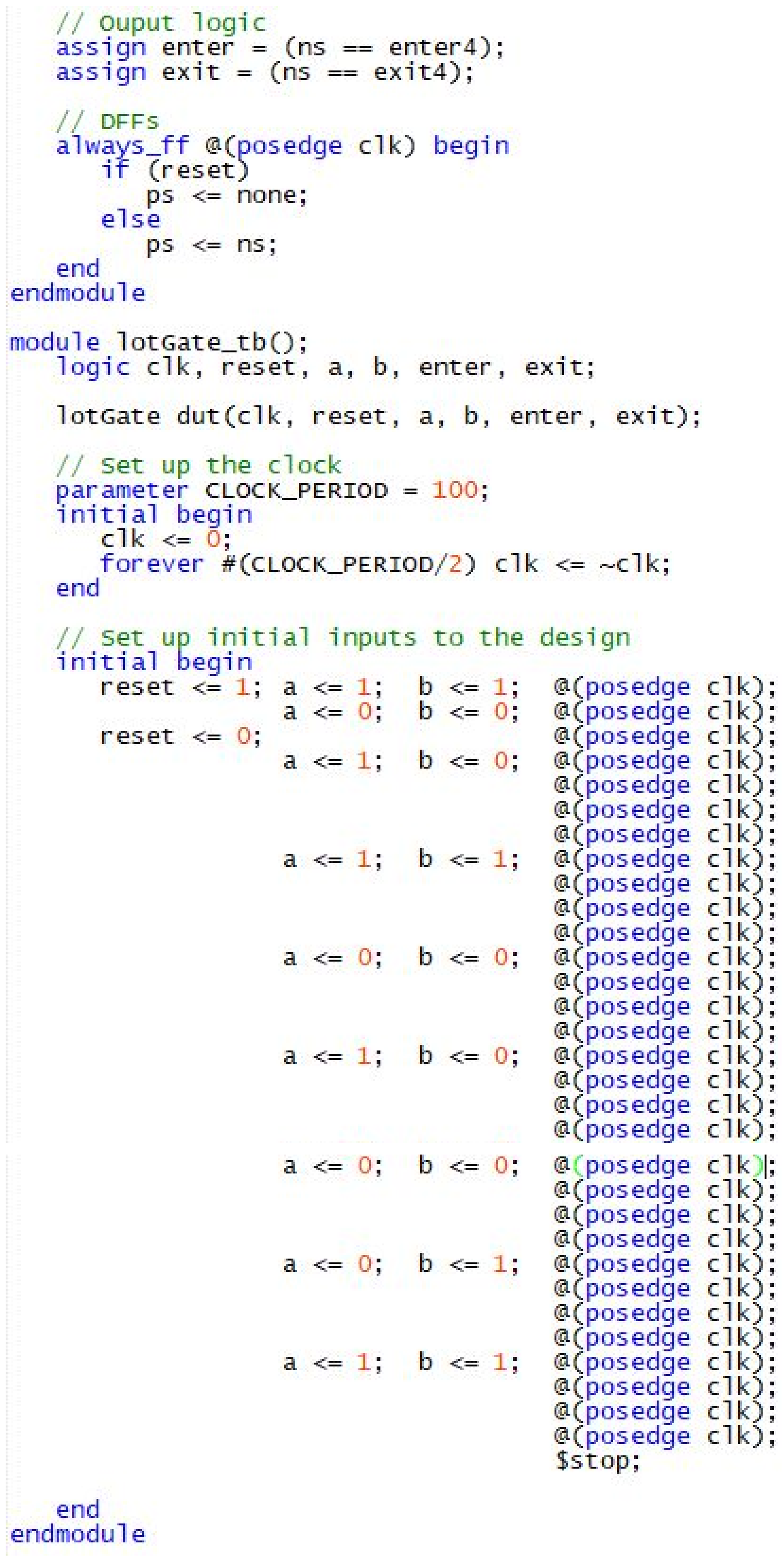
Flow Status	Successful - Thu Oct 04 22:45:07 2018
Quartus Prime Version	17.0.0 Build 595 04/25/2017 SJ Lite Edition
Revision Name	DE1_SoC
Top-level Entity Name	DE1_SoC
Family	Cyclone V
Device	5CSEMA5F31C6
Timing Models	Final
Logic utilization (in ALMs)	N/A
Total registers	17
Total pins	103
Total virtual pins	0
Total block memory bits	0
Total DSP Blocks	0
Total HSSI RX PCSs	0
Total HSSI PMA RX Deserializers	0
Total HSSI TX PCSs	0
Total HSSI PMA TX Serializers	0
Total PLLs	0
Total DLLs	0

Figure 6: Flow Summary

Conclusion:
Overall, this lab provides a brief review of FSMs to refresh the important concepts and skills used in EE 271. The logic for the parking lot gate is just another example of a multiple input FSM that can be used to run more complex logic systems The total time taken to complete the full lab was about 10 hours.

Appendix:


```
module lotGate(clk, reset, a, b, enter, exit);
   input logic clk, reset, a, b;

   output logic enter, exit;

   // State variables
   enum {none, enter1, exit1, enter2, exit2, enter3, exit3, enter4, exit4} ps, ns;

   // Next state logic
   always_comb begin
      case (ps)
         none:    if (a & ~b)
                     ns = enter1;
                  else if (~a & b)
                     ns = exit1;
                  else
                     ns = none;
         enter1:  if (a & b)
                     ns = enter2;
                  else if (a & ~b)
                     ns = enter1;
                  else
                     ns = none;
         exit1:   if (a & b)
                     ns = exit2;
                  else if (~a & b)
                     ns = exit1;
                  else
                     ns = none;
         enter2:  if (~a & b)
                     ns = enter3;
                  else if (a & b)
                     ns = enter2;
                  else if (~a & b)
                     ns = enter1;
                  else
                     ns = none;
         exit2:   if (a & ~b)
                     ns = exit3;
                  else if (a & b)
                     ns = exit2;
                  else if (a & ~b)
                     ns = exit1;
                  else
                     ns = none;
         enter3:  if (a & b)
                     ns = enter2;
                  else if (~a & b)
                     ns = enter3;
                  else if (~a & ~b)
                     ns = enter4;
                  else
                     ns = enter1;
         exit3:   if (a & b)
                     ns = exit2;
                  else if (a & ~b)
                     ns = exit3;
                  else if (~a & ~b)
                     ns = exit4;
                  else
                     ns = exit1;
         enter4:  ns = none;
         exit4:   ns = none;
      endcase
   end
```

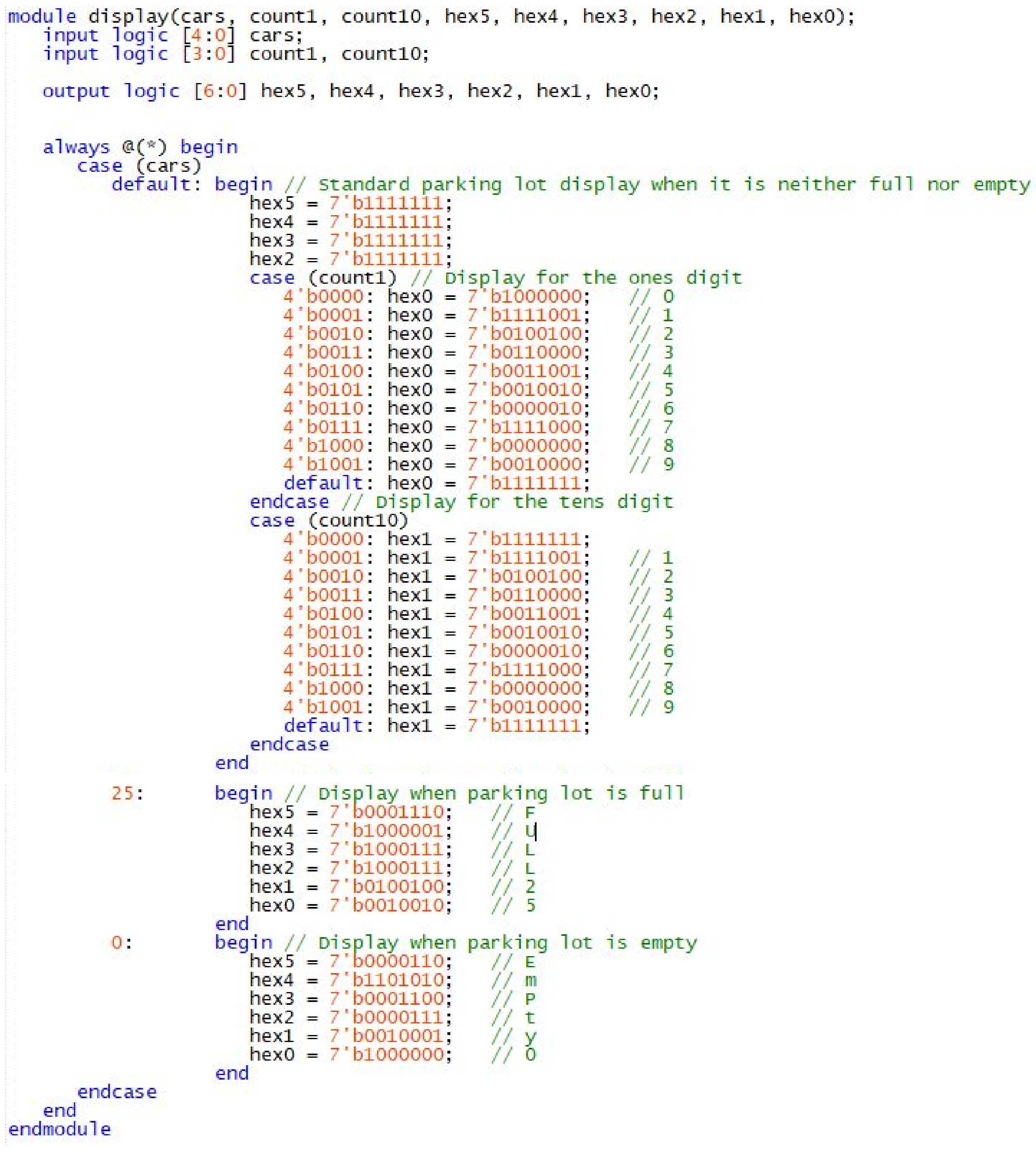


```
   // Ouput logic
   assign enter = (ns == enter4);
   assign exit = (ns == exit4);

   // DFFs
   always_ff @(posedge clk) begin
      if (reset)
         ps <= none;
      else
         ps <= ns;
   end
endmodule

module lotGate_tb();
   logic clk, reset, a, b, enter, exit;

   lotGate dut(clk, reset, a, b, enter, exit);

   // Set up the clock
   parameter CLOCK_PERIOD = 100;
   initial begin
      clk <= 0;
      forever #(CLOCK_PERIOD/2) clk <= ~clk;
   end

   // Set up initial inputs to the design
   initial begin
      reset <= 1; a <= 1;  b <= 1;  @(posedge clk);
                  a <= 0;  b <= 0;  @(posedge clk);
      reset <= 0;                   @(posedge clk);
                  a <= 1;  b <= 0;  @(posedge clk);
                                    @(posedge clk);
                                    @(posedge clk);
                                    @(posedge clk);
                  a <= 1;  b <= 1;  @(posedge clk);
                                    @(posedge clk);
                                    @(posedge clk);
                                    @(posedge clk);
                  a <= 0;  b <= 0;  @(posedge clk);
                                    @(posedge clk);
                                    @(posedge clk);
                                    @(posedge clk);
                  a <= 1;  b <= 0;  @(posedge clk);
                                    @(posedge clk);
                                    @(posedge clk);
                                    @(posedge clk);
                  a <= 0;  b <= 0;  @(posedge clk);
                                    @(posedge clk);
                                    @(posedge clk);
                                    @(posedge clk);
                  a <= 0;  b <= 1;  @(posedge clk);
                                    @(posedge clk);
                                    @(posedge clk);
                                    @(posedge clk);
                  a <= 1;  b <= 1;  @(posedge clk);
                                    @(posedge clk);
                                    @(posedge clk);
                                    @(posedge clk);
                                    $stop;

   end
endmodule
```

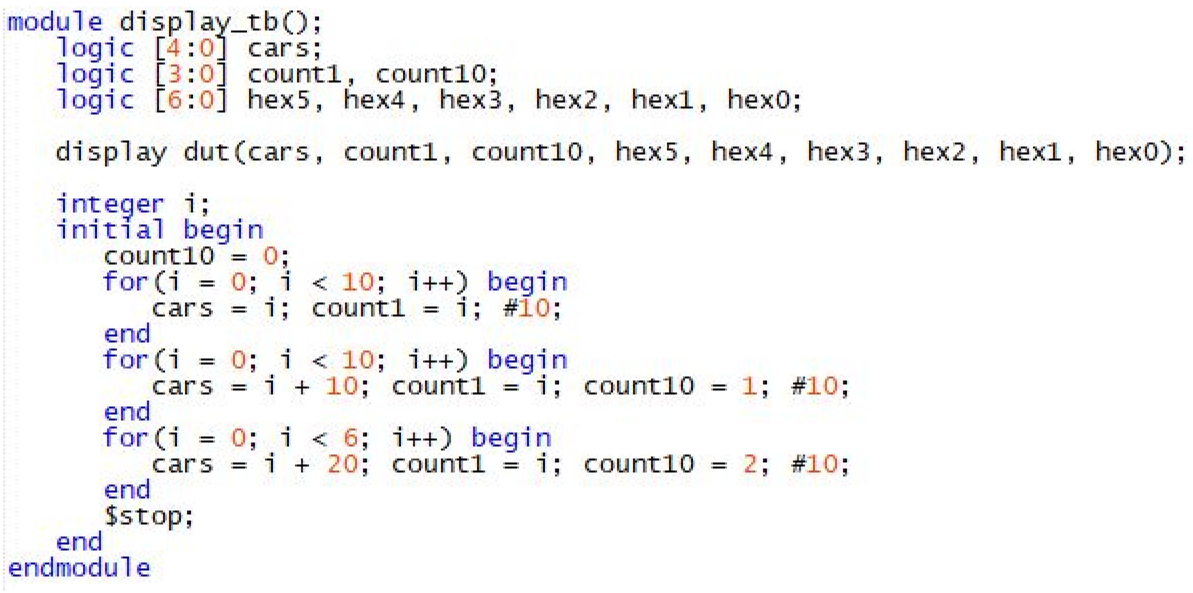
Code 1: lotGate.sv

```
module counter(clk, reset, in, out, count10, count1, cars);
   input logic clk, reset, in, out;

   output logic [4:0] cars;
   output logic [3:0] count1, count10;

   /*
      Gereral 0 to 99 circular up/down counter
      For this module, counter is limited from 0 to 25
      with circular restart disabled
   */
   always_ff @(posedge clk) begin
      if (reset) begin
         count1 <= 0;
         count10 <= 0;
         cars <= 0;
      end else if (in && cars < 25) begin // Up counter when lot is not full and car enters
         cars <= cars + 1;
         if (count1 < 9) // increment ones digit
            count1 <= count1 + 1;
         else begin // increment tens digit if ones digit reaches 9
            if (count10 < 9)
               count10 <= count10 + 1;
            else // restart counter at 00 if counter goes above 99
               count10 <= 0;
            count1 <= 0;
         end
      end else if (out && cars > 0) begin // Down counter when lot is not empty can car leaves
         cars <= cars - 1;
         if (count1 > 0) // decrement ones digit
            count1 <= count1 - 1;
         else begin // decrement tens digit if ones digit reaches 0
            if (count10 > 0)
               count10 <= count10 - 1;
            else // restart counter at 99 if counter goes below 00
               count10 <= 9;
            count1 <= 9;
         end
      end
   end
endmodule

module counter_tb();
   logic clk, reset, in;
   logic [6:0] displayOut10, displayOut1;
   logic [3:0] count1, count10;
   logic [7:0] guesses;

   counter dut(reset, in, out, displayOut10, displayOut1);

   // Set up the clock.
   parameter CLOCK_PERIOD = 100;
   initial begin
      clk <= 0;
      forever #(CLOCK_PERIOD/2) clk <= ~clk;
   end

   // Set up initial inputs to the design. Each line is a clock cycle.
   integer i;
   initial begin
      reset <= 1; in <= 0; @(posedge clk);
      reset <= 0;
      for(i=0; i<30; i++) begin
         in <= 1; @(posedge clk);
         in <= 0; @(posedge clk);
      end
      $stop;
   end
endmodule
```

Code 2: counter.sv

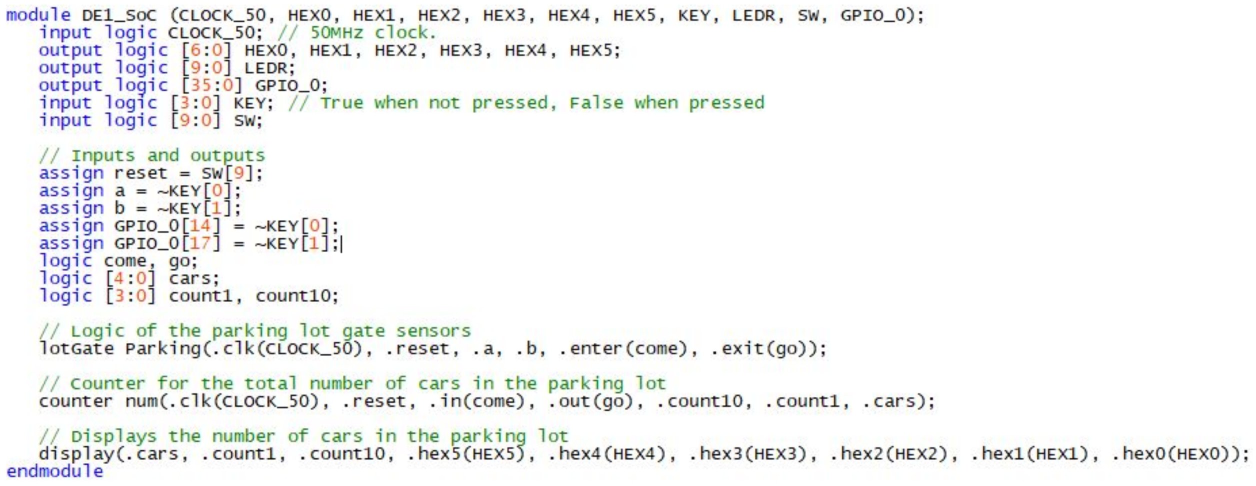


```
module display(cars, count1, count10, hex5, hex4, hex3, hex2, hex1, hex0);
   input logic [4:0] cars;
   input logic [3:0] count1, count10;

   output logic [6:0] hex5, hex4, hex3, hex2, hex1, hex0;


   always @(*) begin
      case (cars)
         default: begin // Standard parking lot display when it is neither full nor empty
                     hex5 = 7'b1111111;
                     hex4 = 7'b1111111;
                     hex3 = 7'b1111111;
                     hex2 = 7'b1111111;
                     case (count1) // Display for the ones digit
                        4'b0000: hex0 = 7'b1000000;    // 0
                        4'b0001: hex0 = 7'b1111001;    // 1
                        4'b0010: hex0 = 7'b0100100;    // 2
                        4'b0011: hex0 = 7'b0110000;    // 3
                        4'b0100: hex0 = 7'b0011001;    // 4
                        4'b0101: hex0 = 7'b0010010;    // 5
                        4'b0110: hex0 = 7'b0000010;    // 6
                        4'b0111: hex0 = 7'b1111000;    // 7
                        4'b1000: hex0 = 7'b0000000;    // 8
                        4'b1001: hex0 = 7'b0010000;    // 9
                        default: hex0 = 7'b1111111;
                     endcase // Display for the tens digit
                     case (count10)
                        4'b0000: hex1 = 7'b1111111;
                        4'b0001: hex1 = 7'b1111001;    // 1
                        4'b0010: hex1 = 7'b0100100;    // 2
                        4'b0011: hex1 = 7'b0110000;    // 3
                        4'b0100: hex1 = 7'b0011001;    // 4
                        4'b0101: hex1 = 7'b0010010;    // 5
                        4'b0110: hex1 = 7'b0000010;    // 6
                        4'b0111: hex1 = 7'b1111000;    // 7
                        4'b1000: hex1 = 7'b0000000;    // 8
                        4'b1001: hex1 = 7'b0010000;    // 9
                        default: hex1 = 7'b1111111;
                     endcase
                  end
         25:      begin // Display when parking lot is full
                     hex5 = 7'b0001110;    // F
                     hex4 = 7'b1000001;    // U
                     hex3 = 7'b1000111;    // L
                     hex2 = 7'b1000111;    // L
                     hex1 = 7'b0100100;    // 2
                     hex0 = 7'b0010010;    // 5
                  end
         0:       begin // Display when parking lot is empty
                     hex5 = 7'b0000110;    // E
                     hex4 = 7'b1101010;    // m
                     hex3 = 7'b0001100;    // P
                     hex2 = 7'b0000111;    // t
                     hex1 = 7'b0010001;    // y
                     hex0 = 7'b1000000;    // 0
                  end
      endcase
   end
endmodule
```



```
module display_tb();
   logic [4:0] cars;
   logic [3:0] count1, count10;
   logic [6:0] hex5, hex4, hex3, hex2, hex1, hex0;

   display dut(cars, count1, count10, hex5, hex4, hex3, hex2, hex1, hex0);

   integer i;
   initial begin
      count10 = 0;
      for(i = 0; i < 10; i++) begin
         cars = i; count1 = i; #10;
      end
      for(i = 0; i < 10; i++) begin
         cars = i + 10; count1 = i; count10 = 1; #10;
      end
      for(i = 0; i < 6; i++) begin
         cars = i + 20; count1 = i; count10 = 2; #10;
      end
      $stop;
   end
endmodule
```

Code 3: display.sv



```
module DE1_SoC (CLOCK_50, HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, KEY, LEDR, SW, GPIO_0);
   input logic CLOCK_50; // 50MHz clock.
   output logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
   output logic [9:0] LEDR;
   output logic [35:0] GPIO_0;
   input logic [3:0] KEY; // True when not pressed, False when pressed
   input logic [9:0] SW;

   // Inputs and outputs
   assign reset = SW[9];
   assign a = ~KEY[0];
   assign b = ~KEY[1];
   assign GPIO_0[14] = ~KEY[0];
   assign GPIO_0[17] = ~KEY[1];
   logic come, go;
   logic [4:0] cars;
   logic [3:0] count1, count10;

   // Logic of the parking lot gate sensors
   lotGate Parking(.clk(CLOCK_50), .reset, .a, .b, .enter(come), .exit(go));

   // Counter for the total number of cars in the parking lot
   counter num(.clk(CLOCK_50), .reset, .in(come), .out(go), .count10, .count1, .cars);

   // Displays the number of cars in the parking lot
   display(.cars, .count1, .count10, .hex5(HEX5), .hex4(HEX4), .hex3(HEX3), .hex2(HEX2), .hex1(HEX1), .hex0(HEX0));
endmodule
```

Code 4: DE1_SoC.sv