Eugene Ngo : 1965514

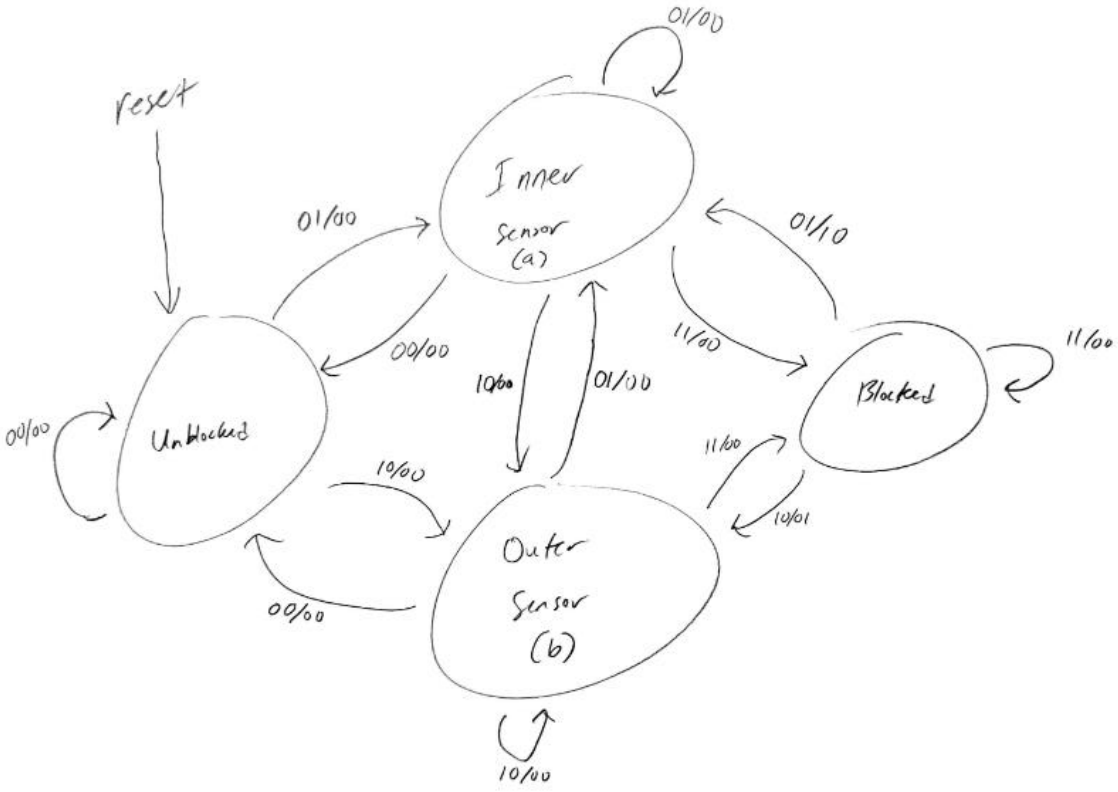
1/13/2023

EE 371 - Lab 1 Report

Procedure

Lab 1 was a one-part lab that refreshed us on creating finite state machines (FSMs) using Quartus. The objective of this lab was to create a parking lot occupancy counter. The GPIO pins on the DE1_SoC were hooked up to switches which represented the inputs from two parking lot sensors. These inputs were also displayed on LEDs. Using the switch inputs, parking lot behaviors could be emulated, such as cars entering and exiting the parking lot. When cars exited or entered the parking lot, the parking lot occupancy counter kept track of the car count. This car count was then displayed on the in-built seven segment display in the DE1_SoC.

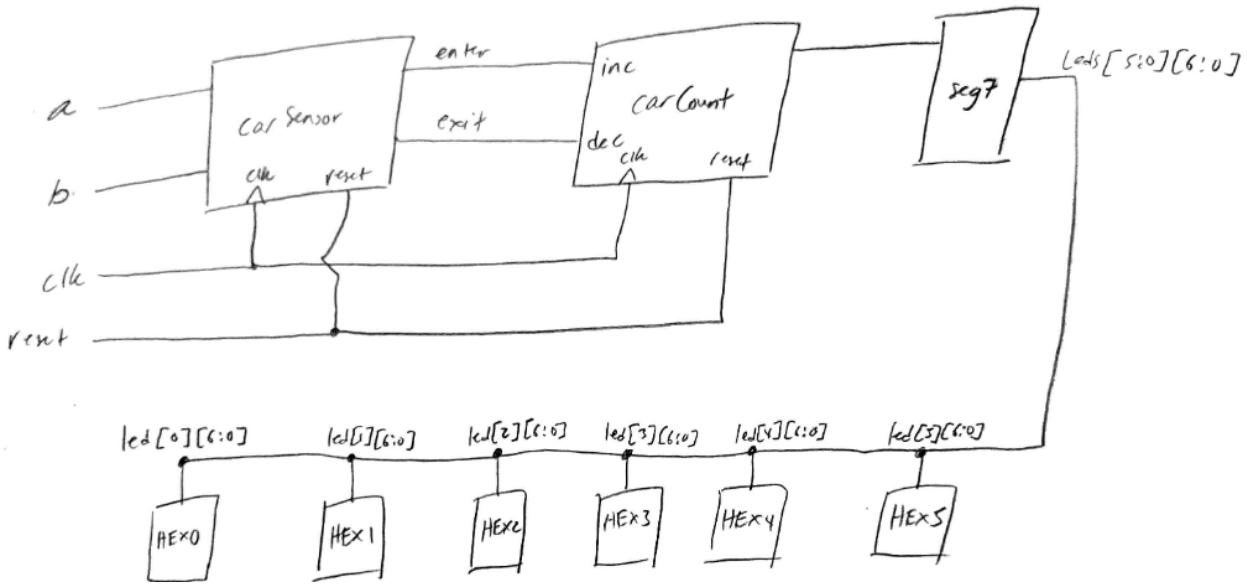
State Diagram:



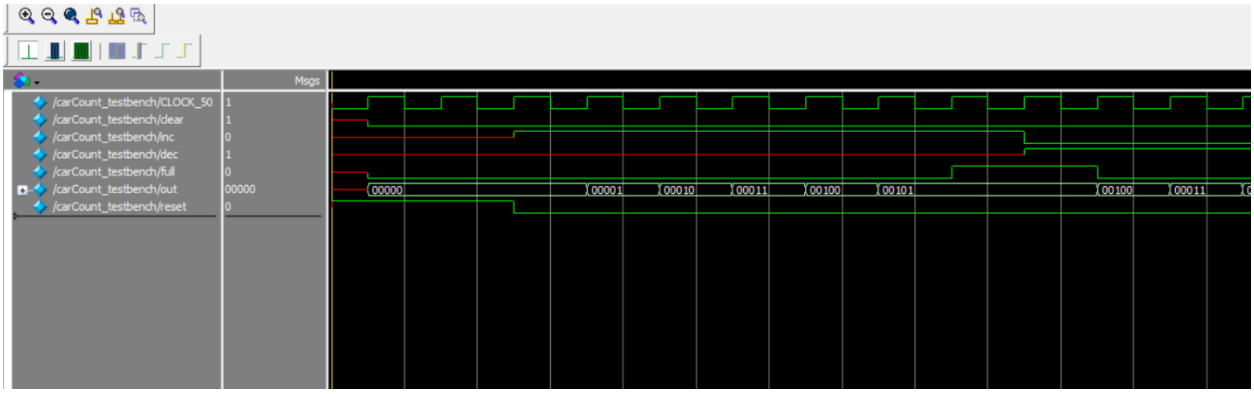
The expected behavior of the parking lot counter was implemented using a finite state machine. Inputs from inner and outer sensors would result in state changes and eventual outputs of a vehicle exiting or entering the parking lot. It was determined that only cars could block both sensors. Thus, the output of the system only adds or removes from the car count when the system transitions from blocked to inner or blocked to outer. The FSM design can be seen above.

The enter and exit output signals from the car detection finite state machine was passed onto a counting unit. The counting unit increments the current count of cars if the enter signal is true and if the current count is below the parking capacity which is 25. The counting unit decrements if the exit signal is true and if the current count is above 0. This design choice was made because car counts outside of this range are not possible and should be ignored. Additionally, it is impossible for both the enter and exit signals to be true at once, thus this case was not considered or implemented. After the counter increments or decrements the count, the count is output. The count value that is generated by the counter is then passed through to the seven segment display unit. The count value is converted from the 5-bit binary encoding to a 7-bit binary encoding for the seven segment display. A count of 0 is converted to “Clear 0”. 25 is converted to “Full 25”. All other values are converted to numbers they represent.

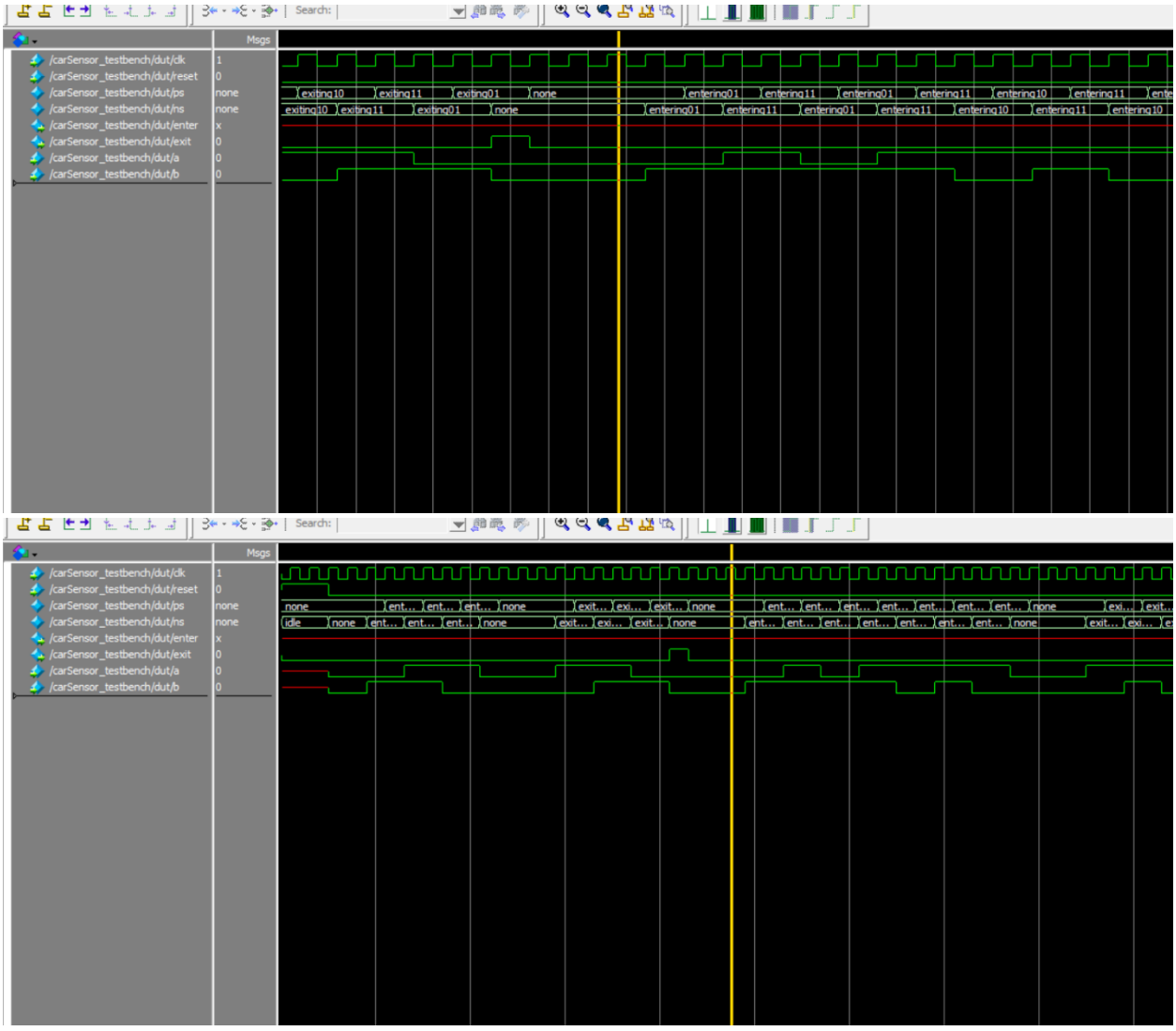
Block Diagram



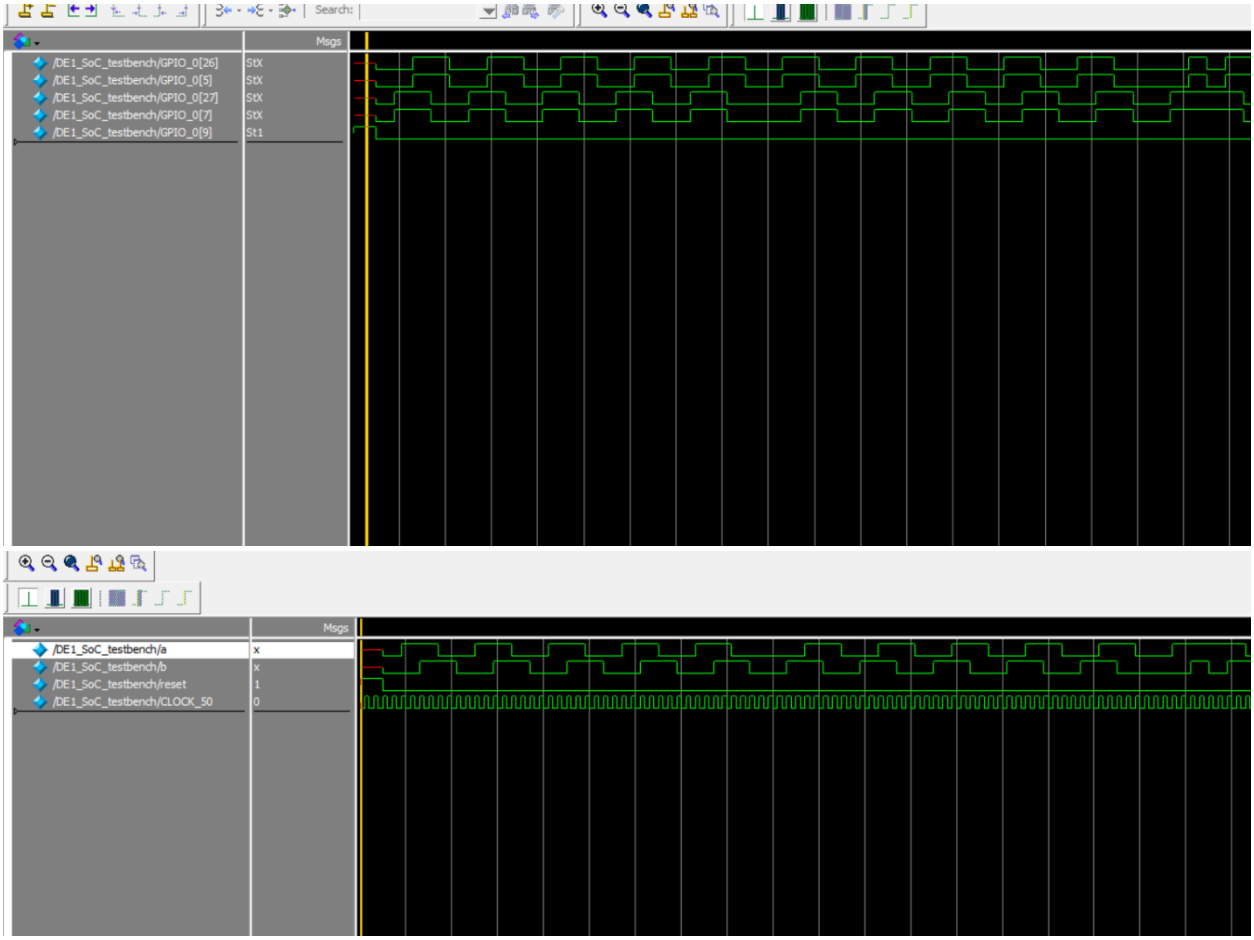
Results



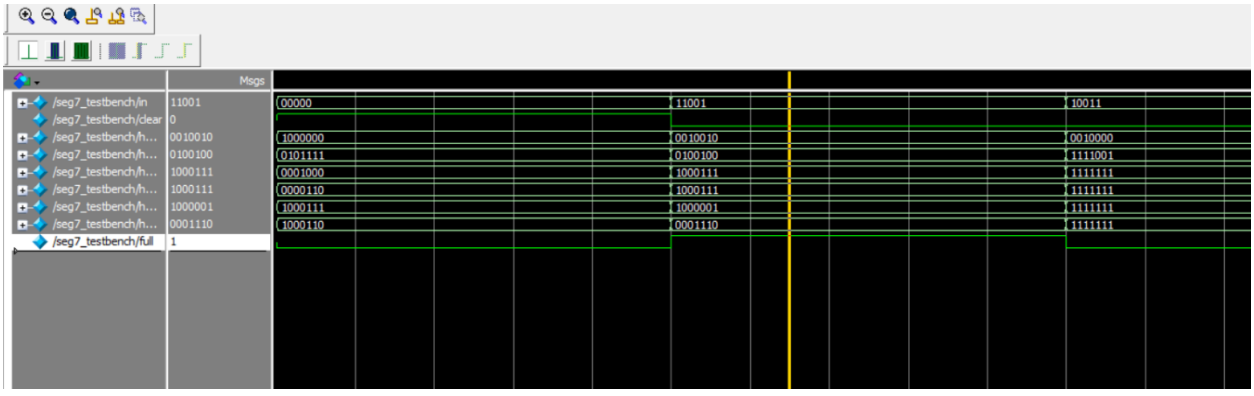
Above is the simulation for the car count which shows the count incrementing to 5 which is set to be the testbench's MAX so after incrementing 5 times, full is set to 1 and is displayed. Then the decrement is tested and decrements the output.



Above are two screenshots of the carSensor showing the present state and next state changing appropriately according to the “enter” and “exit” inputs.

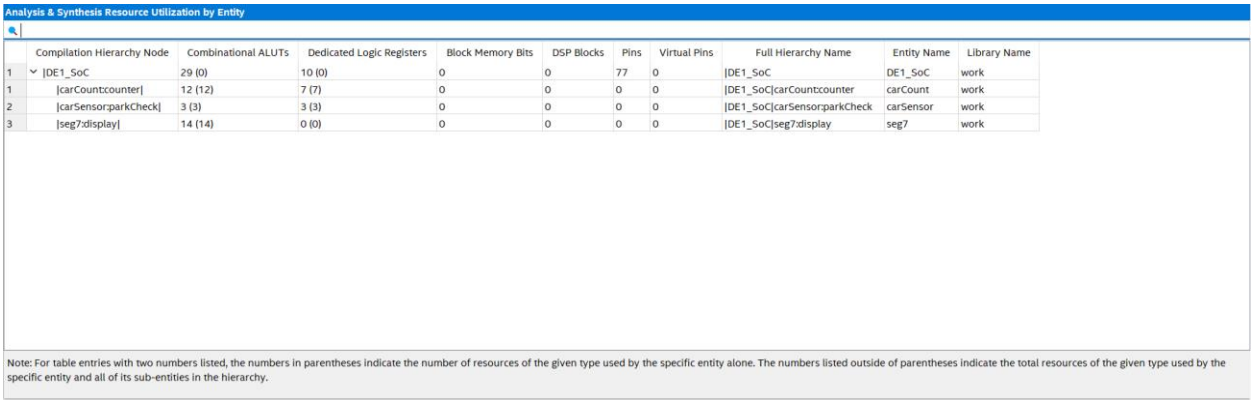


Above are two DE1_SoC screenshots. One shows the GPIO switches and LEDs working together: GPIO 5 switch corresponds to the GPIO 26 LED and the GPIO 7 switch corresponds to the GPIO 27 LED. The second screenshot shows the input signals, a and b, functioning as expected alongside the reset input and the clock.



Above is the seg7 simulation showing 3 cases: ‘clear’ output, ‘full’ output, and a regular numbered output.

Resource Utilization



Analysis & Synthesis Resource Utilization by Entity

	Compilation Hierarchy Node	Combinational ALUTs	Dedicated Logic Registers	Block Memory Bits	DSP Blocks	Pins	Virtual Pins	Full Hierarchy Name	Entity Name	Library Name
1	|DE1_SoC	29 (0)	10 (0)	0	0	77	0	|DE1_SoC	DE1_SoC	work
1	|carCount:counter|	12 (12)	7 (7)	0	0	0	0	|DE1_SoC|carCount:counter	carCount	work
2	|carSensor:parkCheck|	3 (3)	3 (3)	0	0	0	0	|DE1_SoC|carSensor:parkCheck	carSensor	work
3	|seg7:display|	14 (14)	0 (0)	0	0	0	0	|DE1_SoC|seg7:display	seg7	work

Note: For table entries with two numbers listed, the numbers in parentheses indicate the number of resources of the given type used by the specific entity alone. The numbers listed outside of parentheses indicate the total resources of the given type used by the specific entity and all of its sub-entities in the hierarchy.

Overview

This lab was a good refresher on FSMs, the design process and Verilog as a whole. The lab specifications were clear and the material for teaching how to use Labsland was useful. I did not run into any major hang ups while designing the FSM or implementing the FSM, and that is likely because I read the lab spec carefully at the beginning and the lab spec was well written and detailed which helped me design the finite state machine out on paper, and only began implementing the unit in Verilog after I was confident about the logic. Additionally, referencing material from 271 and refreshing myself helped me build the hex display modules. The provided GPIO sheet was quite helpful as well.

Overall, the system works according to how the specification wanted.

Appendix

See following code

```
1   /* Name: Eugene Ngo
2      Date: 1/13/2023
3      Class: EE 371
4      Lab 1: Parking Lot Occupancy Counter*/
5
6   // DE1_SoC is the top-level module that defines the I/Os for the DE-1 SoC board.
7   // DE1_SoC takes three switches from the GPIO as inputs, and outputs to 2 LEDs on the
    breadboard through GPIO and 6 7-bit
8   // hex displays (HEX0-HEX5). It displays "full" or "clear" messages when the parking lot is
    either full or clear, and it
9   // displays the decimal value of the number of cars in the lot accordingly.
10
11  module DE1_SoC #(parameter MAX=25) (HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, GPIO_0, CLOCK_50);
12      output logic [6:0]  HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
13      inout logic [33:0] GPIO_0;
14      input logic CLOCK_50;
15
16     // Assigning and clk to CLOCK_50
17     logic clk;
18     assign clk = CLOCK_50;
19
20     logic enter, exit, full, clear;
21     logic [4:0] counter_out;
22
23     // Outputting a and b to breadboard
24     assign GPIO_0[26] = GPIO_0[5];
25     assign GPIO_0[27] = GPIO_0[7];
26
27     // carSensor parkCheck takes two switches from the breadboard as the input of the two
    parking sensors,
28     // and outputs to enter and exit when an entering or exiting vehicle is detected.
29     carSensor parkCheck (.a(GPIO_0[7]), .b(GPIO_0[5]), .enter, .exit, .clk, .reset(GPIO_0[9
    ]));
30
31     // carCount counter takes enter and exit from sensors, and outputs the car count to
    counter_out.
32     // it als outputs full and clear status to full and clear.
33     carCount #(MAX) counter (.inc(enter), .dec(exit), .out(counter_out), .full, .clear, .clk,
     .reset(GPIO_0[9]));
34
35     // seg7 display takes counter_out, full, and clear from ct, and outputs decimal values
    or status messages to hex displays.
36     seg7 display (.in(counter_out), .full, .clear, .hexout0(HEX0), .hexout1(HEX1), .hexout2(
    HEX2), .hexout3(HEX3), .hexout4(HEX4), .hexout5(HEX5));
37
38  endmodule // DE1_SoC
39
40  // DE1_SoC_testbench tests all expected, unexpected, and edgecase behaviors
41  module DE1_SoC_testbench();
42     logic [6:0]  HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
43     wire [33:0] GPIO_0;
44     logic CLOCK_50;
45
46     DE1_SoC #(5) dut (.HEX0, .HEX1, .HEX2, .HEX3, .HEX4, .HEX5, .GPIO_0, .CLOCK_50);
47
48     // Setting up a simulated clock.
49     parameter CLOCK_PERIOD = 100;
50     initial begin
51        CLOCK_50 <= 0;
52        forever #(CLOCK_PERIOD/2) CLOCK_50 <= ~CLOCK_50; // Forever toggle the clock
53     end
54
55     // Assiging logic to wire
56     logic reset, a, b;
57     assign GPIO_0[9] = reset;
58     assign GPIO_0[7] = a;
59     assign GPIO_0[5] = b;
60
61     // Testing the module
62     initial begin
63        // reset
64        reset <= 1;                    repeat(3) @(posedge CLOCK_50);
65
```

```
66          //enters until full
67          reset <= 0;       a <= 0; b <= 0; repeat(2) @(posedge CLOCK_50);
68                            a <= 1; b <= 0; repeat(2) @(posedge CLOCK_50);
69                            a <= 1; b <= 1; repeat(2) @(posedge CLOCK_50);
70                            a <= 0; b <= 1; repeat(2) @(posedge CLOCK_50);
71                            a <= 0; b <= 0; repeat(2) @(posedge CLOCK_50); // 1st car enters
72                            a <= 1; b <= 0; repeat(2) @(posedge CLOCK_50);
73                            a <= 1; b <= 1; repeat(2) @(posedge CLOCK_50);
74                            a <= 0; b <= 1; repeat(2) @(posedge CLOCK_50);
75                            a <= 0; b <= 0; repeat(2) @(posedge CLOCK_50); // 2nd car enters
76                            a <= 1; b <= 0; repeat(2) @(posedge CLOCK_50);
77                            a <= 1; b <= 1; repeat(2) @(posedge CLOCK_50);
78                            a <= 0; b <= 1; repeat(2) @(posedge CLOCK_50);
79                            a <= 0; b <= 0; repeat(2) @(posedge CLOCK_50); // 3rd car enters
80                            a <= 1; b <= 0; repeat(2) @(posedge CLOCK_50);
81                            a <= 1; b <= 1; repeat(2) @(posedge CLOCK_50);
82                            a <= 0; b <= 1; repeat(2) @(posedge CLOCK_50);
83                            a <= 0; b <= 0; repeat(2) @(posedge CLOCK_50); // 4th car enters
84                            a <= 1; b <= 0; repeat(2) @(posedge CLOCK_50);
85                            a <= 1; b <= 1; repeat(2) @(posedge CLOCK_50);
86                            a <= 0; b <= 1; repeat(2) @(posedge CLOCK_50);
87                            a <= 0; b <= 0; repeat(2) @(posedge CLOCK_50); // 5th car enters, full
88
89      // exits until clear
90                            a <= 0; b <= 0; repeat(2) @(posedge CLOCK_50);
91                            a <= 0; b <= 1; repeat(2) @(posedge CLOCK_50);
92                            a <= 1; b <= 1; repeat(2) @(posedge CLOCK_50);
93                            a <= 1; b <= 0; repeat(2) @(posedge CLOCK_50);
94                            a <= 0; b <= 0; repeat(2) @(posedge CLOCK_50); // 1st car exits
95                            a <= 0; b <= 1; repeat(2) @(posedge CLOCK_50);
96                            a <= 1; b <= 1; repeat(2) @(posedge CLOCK_50);
97                            a <= 1; b <= 0; repeat(2) @(posedge CLOCK_50);
98                            a <= 0; b <= 0; repeat(2) @(posedge CLOCK_50); // 2nd car exits
99                            a <= 0; b <= 1; repeat(2) @(posedge CLOCK_50);
100                           a <= 1; b <= 1; repeat(2) @(posedge CLOCK_50);
101                           a <= 1; b <= 0; repeat(2) @(posedge CLOCK_50);
102                           a <= 0; b <= 0; repeat(2) @(posedge CLOCK_50); // 3rd car exits
103                           a <= 0; b <= 1; repeat(2) @(posedge CLOCK_50);
104                           a <= 1; b <= 1; repeat(2) @(posedge CLOCK_50);
105                           a <= 1; b <= 0; repeat(2) @(posedge CLOCK_50);
106                           a <= 0; b <= 0; repeat(2) @(posedge CLOCK_50); // 4th car exits
107                           a <= 0; b <= 1; repeat(2) @(posedge CLOCK_50);
108                           a <= 1; b <= 1; repeat(2) @(posedge CLOCK_50);
109                           a <= 1; b <= 0; repeat(2) @(posedge CLOCK_50);
110                           a <= 0; b <= 0; repeat(2) @(posedge CLOCK_50); // 5th car exits, clear
111
112         // direction changes while entering
113                           a <= 0; b <= 0; repeat(2) @(posedge CLOCK_50);
114                           a <= 1; b <= 0; repeat(2) @(posedge CLOCK_50);
115                           a <= 1; b <= 1; repeat(2) @(posedge CLOCK_50);
116                           a <= 1; b <= 0; repeat(2) @(posedge CLOCK_50);
117                           a <= 1; b <= 1; repeat(2) @(posedge CLOCK_50);
118                           a <= 0; b <= 1; repeat(2) @(posedge CLOCK_50);
119                           a <= 1; b <= 1; repeat(2) @(posedge CLOCK_50);
120                           a <= 0; b <= 1; repeat(2) @(posedge CLOCK_50);
121                           a <= 0; b <= 0; repeat(2) @(posedge CLOCK_50);
122         // direction changes while exiting
123                           a <= 0; b <= 0; repeat(2) @(posedge CLOCK_50);
124                           a <= 0; b <= 1; repeat(2) @(posedge CLOCK_50);
125                           a <= 1; b <= 1; repeat(2) @(posedge CLOCK_50);
126                           a <= 0; b <= 1; repeat(2) @(posedge CLOCK_50);
127                           a <= 1; b <= 1; repeat(2) @(posedge CLOCK_50);
128                           a <= 1; b <= 0; repeat(2) @(posedge CLOCK_50);
129                           a <= 1; b <= 1; repeat(2) @(posedge CLOCK_50);
130                           a <= 1; b <= 0; repeat(2) @(posedge CLOCK_50);
131                           a <= 0; b <= 0; repeat(2) @(posedge CLOCK_50);
132
133         $stop;
134     end
135 endmodule // DE1_SoC_testbench
```

```
/* Name: Eugene Ngo
   Date: 1/13/2023
   Class: EE 371
   Lab 1: Parking Lot Occupancy Counter*/

// carSensor takes inputs from two sensors, a and b, and output "1" to either enter or exit for 1 clock cycle
// whenever an entering or exiting vehicle is detected.
module carSensor (a, b, enter, exit, clk, reset);
   input logic a, b, clk, reset;
   output logic enter; // car entering
   output logic exit; //  car exiting

   enum {none, entering01, exiting01, entering11, exiting11, entering10, exiting10, idle} ps, ns;

   // Logic for next state
   always_comb begin
      case(ps)
         none: if (~a & ~b) ns = none;
               else if (~a & b) ns = entering01;
               else if (a & ~b) ns = exiting10;
               else ns = idle;
         entering01: if (~a & ~b) ns = none;
               else if (~a & b) ns = entering01;
               else if (a & ~b) ns = idle;
               else ns = entering11;
         exiting01: if (~a & ~b) ns = none;
               else if (~a & b) ns = exiting01;
               else if (a & ~b) ns = idle;
               else ns = exiting11;
         entering11: if (~a & ~b) ns = none;
               else if (~a & b) ns = entering01;
               else if (a & ~b) ns = entering10;
               else ns = entering11;
         exiting11: if (~a & ~b) ns = none;
               else if (~a & b) ns = exiting01;
               else if (a & ~b) ns = exiting10;
               else ns = exiting11;
         entering10: if (~a & ~b) ns = none;
               else if (~a & b) ns = idle;
               else if (a & ~b) ns = entering10;
               else ns = entering11;
         exiting10: if (~a & ~b) ns = none;
               else if (~a & b) ns = idle;
               else if (a & ~b) ns = exiting10;
               else ns = exiting11;
         idle: if (~a & ~b) ns = none;
               else ns = idle;
      endcase
   end // always_comb

   //output logic for exiting: outputs 1 to exit when an exiting vehicle is detected.
   always_comb begin
      case(ps)
         exiting01: if (~a & ~b) exit = 1'b1;
                    else exit = 1'b0;
         default: exit = 1'b0;
      endcase
   end // always_comb

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

// carSensor_testbench tests all expected, unexpected, and edgecase behaviors
module carSensor_testbench();
   logic a, b, clk, reset, enter, exit;
```

```
72      logic CLOCK_50;
73
74      carSensor dut (.a(b), .b(a), .clk(CLOCK_50), .reset, .enter, .exit);
75
76      // Setting up a clock.
77      parameter CLOCK_PERIOD = 100;
78      initial begin
79         CLOCK_50 <= 0;
80         forever #(CLOCK_PERIOD/2) CLOCK_50 <= ~CLOCK_50; // toggle the clock forever
81      end // initial
82
83      initial begin
84         // reset
85         reset <= 1;                      repeat(3) @(posedge CLOCK_50);
86
87         //enters
88         reset <= 0;       a <= 0; b <= 0; repeat(2) @(posedge CLOCK_50);
89                           a <= 1; b <= 0; repeat(2) @(posedge CLOCK_50);
90                           a <= 1; b <= 1; repeat(2) @(posedge CLOCK_50);
91                           a <= 0; b <= 1; repeat(2) @(posedge CLOCK_50);
92                           a <= 0; b <= 0; repeat(2) @(posedge CLOCK_50);
93         //exits
94                           a <= 0; b <= 0; repeat(2) @(posedge CLOCK_50);
95                           a <= 0; b <= 1; repeat(2) @(posedge CLOCK_50);
96                           a <= 1; b <= 1; repeat(2) @(posedge CLOCK_50);
97                           a <= 1; b <= 0; repeat(2) @(posedge CLOCK_50);
98                           a <= 0; b <= 0; repeat(2) @(posedge CLOCK_50);
99
100        // direction changes while entering
101                          a <= 0; b <= 0; repeat(2) @(posedge CLOCK_50);
102                          a <= 1; b <= 0; repeat(2) @(posedge CLOCK_50);
103                          a <= 1; b <= 1; repeat(2) @(posedge CLOCK_50);
104                          a <= 1; b <= 0; repeat(2) @(posedge CLOCK_50);
105                          a <= 1; b <= 1; repeat(2) @(posedge CLOCK_50);
106                          a <= 0; b <= 1; repeat(2) @(posedge CLOCK_50);
107                          a <= 1; b <= 1; repeat(2) @(posedge CLOCK_50);
108                          a <= 0; b <= 1; repeat(2) @(posedge CLOCK_50);
109                          a <= 0; b <= 0; repeat(2) @(posedge CLOCK_50);
110
111        // direction changes while exiting
112                          a <= 0; b <= 0; repeat(2) @(posedge CLOCK_50);
113                          a <= 0; b <= 1; repeat(2) @(posedge CLOCK_50);
114                          a <= 1; b <= 1; repeat(2) @(posedge CLOCK_50);
115                          a <= 0; b <= 1; repeat(2) @(posedge CLOCK_50);
116                          a <= 1; b <= 1; repeat(2) @(posedge CLOCK_50);
117                          a <= 1; b <= 0; repeat(2) @(posedge CLOCK_50);
118                          a <= 1; b <= 1; repeat(2) @(posedge CLOCK_50);
119                          a <= 1; b <= 0; repeat(2) @(posedge CLOCK_50);
120                          a <= 0; b <= 0; repeat(2) @(posedge CLOCK_50);
121        $stop;
122     end // intial
123
124  endmodule // carSensor_testbench
125
```

```
1   /* Name: Eugene Ngo
2      Date: 1/13/2023
3      Class: EE 371
4      Lab 1: Parking Lot Occupancy Counter*/
5
6   // carCount takes two inputs (inc, dec). It adds 5'b00001 to out when inc is true, and
    subtracts 5'b00001
7   // from out when dec is true. Out has a minimum value of 5'b00000 and a maximum value
    determined by the
8   // parameter (25 by default).
9   module carCount #(parameter MAX=25) (inc, dec, out, full, clear, clk, reset);
10
11     input logic inc, dec, clk, reset;
12     output logic [4:0] out;
13     output logic full, clear;
14
15     // Sequential logic for counting up and counting down depending on the input.
16     always_ff @(posedge clk) begin
17        if (reset) begin
18           out <= 5'b00000;
19           full <= 1'b0;
20           clear <= 1'b0;
21        end
22        else if (inc & out < MAX) begin //increment when not at max
23           out <= out + 5'b00001;
24           clear <= 1'b0;
25        end
26        else if (dec & out > 5'b00000) begin // decrement when not at min
27           out <= out - 5'b00001;
28           full <= 1'b0;
29        end
30        else if (out == MAX) begin // hold value at max, output full
31           out <= MAX;
32           full <= 1'b1;
33        end
34        else if (out == 5'b00000) begin // hold value at min, output clear
35           out <= 5'b00000;
36           clear <= 1'b1;
37        end
38        else
39           out <= out; // hold value otherwise
40     end // always_ff
41
42  endmodule
43
44  // carCount_testbench tests all expected, unexpected, and edgecase behaviors
45  module carCount_testbench();
46     logic inc, dec, full, clear, reset;
47     logic [4:0] out;
48     logic CLOCK_50;
49
50     carCount #(5) dut (.inc, .dec, .out, .full, .clear, .clk(CLOCK_50), .reset);
51
52     // Setting up the clock.
53     parameter CLOCK_PERIOD = 100;
54     initial begin
55        CLOCK_50 <= 0;
56        forever #(CLOCK_PERIOD/2) CLOCK_50 <= ~CLOCK_50; // toggle the clock forever
57     end // initial
58
59     initial begin
60        reset <= 1;                          repeat(3) @(posedge CLOCK_50); // reset
61        reset <= 0; inc <= 1;                repeat (7) @(posedge CLOCK_50); // inc past max limit
62                    inc <= 0; dec <= 1; repeat(7) @(posedge CLOCK_50); // dec past min limit
63        $stop;
64     end
65  endmodule // counter_testbench
```

```
/* Name: Eugene Ngo
   Date: 1/13/2023
   Class: EE 371
   Lab 1: Parking Lot Occupancy Counter*/

// seg7 outputs correct decimal value to hex displays based on the output given by the counter.
// It also displays "FULL" and "CLEAR" according to the indicator outputs given by the counter.
 module seg7 (in, full, clear, hexout0, hexout1, hexout2, hexout3, hexout4, hexout5);

   input logic full, clear;
   input logic [4:0] in;
   output logic [6:0] hexout0, hexout1, hexout2, hexout3, hexout4, hexout5;

   // Assigning hex display variables on necessary numbers.
   logic [6:0] hex0, hex1, hex2, hex3, hex4, hex5, hex6, hex7, hex8, hex9;
   assign hex0 = 7'b1000000; // 0
   assign hex1 = 7'b1111001; // 1
   assign hex2 = 7'b0100100; // 2
   assign hex3 = 7'b0110000; // 3
   assign hex4 = 7'b0011001; // 4
   assign hex5 = 7'b0010010; // 5
   assign hex6 = 7'b0000010; // 6
   assign hex7 = 7'b1111000; // 7
   assign hex8 = 7'b0000000; // 8
   assign hex9 = 7'b0010000; // 9

   // Assigning hex display variables on necessary letters.
   logic [6:0] hexf, hexu, hexl, hexc, hexe, hexa, hexr, hexoff;
   assign hexf = 7'b0001110; // F
   assign hexu = 7'b1000001; // U
   assign hexl = 7'b1000111; // L
   assign hexc = 7'b1000110; // C
   assign hexe = 7'b0000110; // E
   assign hexa = 7'b0001000; // A
   assign hexr = 7'b0101111; // R
   assign hexoff = 7'b1111111; // off

   // Logic for hexout0: 26 different cases for 26 numbers. (0-25)
   always_comb begin
      case(in)
         5'b00000: hexout0 = hex0;
         5'b00001: hexout0 = hex1;
         5'b00010: hexout0 = hex2;
         5'b00011: hexout0 = hex3;
         5'b00100: hexout0 = hex4;
         5'b00101: hexout0 = hex5;
         5'b00110: hexout0 = hex6;
         5'b00111: hexout0 = hex7;
         5'b01000: hexout0 = hex8;
         5'b01001: hexout0 = hex9;
         5'b01010: hexout0 = hex0;
         5'b01011: hexout0 = hex1;
         5'b01100: hexout0 = hex2;
         5'b01101: hexout0 = hex3;
         5'b01110: hexout0 = hex4;
         5'b01111: hexout0 = hex5;
         5'b10000: hexout0 = hex6;
         5'b10001: hexout0 = hex7;
         5'b10010: hexout0 = hex8;
         5'b10011: hexout0 = hex9;
         5'b10100: hexout0 = hex0;
         5'b10101: hexout0 = hex1;
         5'b10110: hexout0 = hex2;
         5'b10111: hexout0 = hex3;
         5'b11000: hexout0 = hex4;
         5'b11001: hexout0 = hex5;
        default: hexout0 = 7'bX;
      endcase
   end // always_comb

   // Logic for hexout1: 26 different cases for 26 numbers. (0-25)
```

```
72      always_comb begin
73         case(in)
74            5'b00000: hexout1 = hexr;
75            5'b00001: hexout1 = hexoff;
76            5'b00010: hexout1 = hexoff;
77            5'b00011: hexout1 = hexoff;
78            5'b00100: hexout1 = hexoff;
79            5'b00101: hexout1 = hexoff;
80            5'b00110: hexout1 = hexoff;
81            5'b00111: hexout1 = hexoff;
82            5'b01000: hexout1 = hexoff;
83            5'b01001: hexout1 = hexoff;
84            5'b01010: hexout1 = hex1;
85            5'b01011: hexout1 = hex1;
86            5'b01100: hexout1 = hex1;
87            5'b01101: hexout1 = hex1;
88            5'b01110: hexout1 = hex1;
89            5'b01111: hexout1 = hex1;
90            5'b10000: hexout1 = hex1;
91            5'b10001: hexout1 = hex1;
92            5'b10010: hexout1 = hex1;
93            5'b10011: hexout1 = hex1;
94            5'b10100: hexout1 = hex2;
95            5'b10101: hexout1 = hex2;
96            5'b10110: hexout1 = hex2;
97            5'b10111: hexout1 = hex2;
98            5'b11000: hexout1 = hex2;
99            5'b11001: hexout1 = hex2;
100           default: hexout1 = 7'bx;
101        endcase
102     end // always_comb
103
104     // Logic for hexout5 - hexout2: display letters when full or clear, turn off otherwise.
105     always_comb begin
106        if (full) begin
107           hexout5 = hexf;
108           hexout4 = hexu;
109           hexout3 = hexl;
110           hexout2 = hexl;
111        end
112        else if (clear) begin
113           hexout5 = hexc;
114           hexout4 = hexl;
115           hexout3 = hexe;
116           hexout2 = hexa;
117        end
118        else begin
119           hexout5 = hexoff;
120           hexout4 = hexoff;
121           hexout3 = hexoff;
122           hexout2 = hexoff;
123        end
124     end // always_comb
125
126  endmodule // seg7
127
128  // seg7_testbench tests all expected, unexpected, and edgecase behaviors
129  module seg7_testbench();
130    logic full, clear;
131    logic [4:0] in;
132    logic [6:0] hexout0, hexout1, hexout2, hexout3, hexout4, hexout5;
133
134    seg7 dut (.in, .full, .clear, .hexout0, .hexout1, .hexout2, .hexout3, .hexout4, .hexout5);
135
136    initial begin
137       in = '0; clear = 1; full = 0; #10; // testing clear output
138       in = 5'b11001; clear = 0; full = 1; #10; // testing full output
139       in = 5'b10011; clear = 0; full = 0; #10; // testing regular output
140       $stop;
141    end // initial
142 endmodule // seg7_testbench
```