Khoa Tran
EE 371
April 12, 2021
Lab 1 Report

**Procedure**

**Parking Lot Occupancy Counter**

Approaching this problem, I first drew up the block diagram to figure out how to connect each FSM in order to produce the correct output for the number of cars in the parking lot, keeping track of the sensor inputs for either a car is entering or exiting, and connecting the outputs to the inputs of a counter that increments and decrements in order to get an accurate reading of the number of cars in the parking lot. As seen in the block diagram in figure 1, the switches in the GPIO are connected to the inputs of the parking lot finite state machine, and the outputs are the 7 segment HEX board display, and the LEDs of the GPIO.

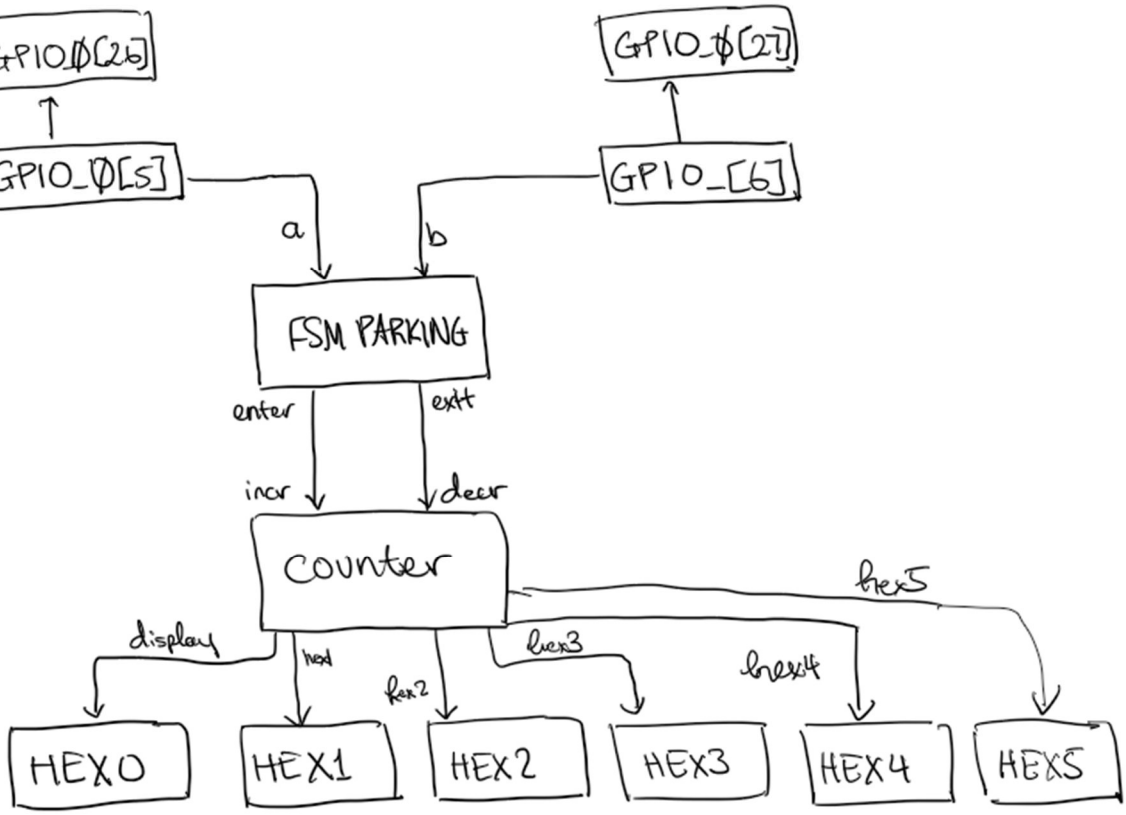


Figure 1: Block diagram for parking lot occupancy counter

**Sensor Input Task**

Approaching this problem, I first drew up the state diagram in order to figure out the number of states that I need and when to progress to each state depending on the inputs of a and b in order to have either exit or enter be true. As seen in figure 1, when the state diagram goes from S0 to

S3 and with a 00 input, then the enter output would be 1. Otherwise, if the state diagram goes from S0 to S1 and with the input of 00, the exit output would be 1.

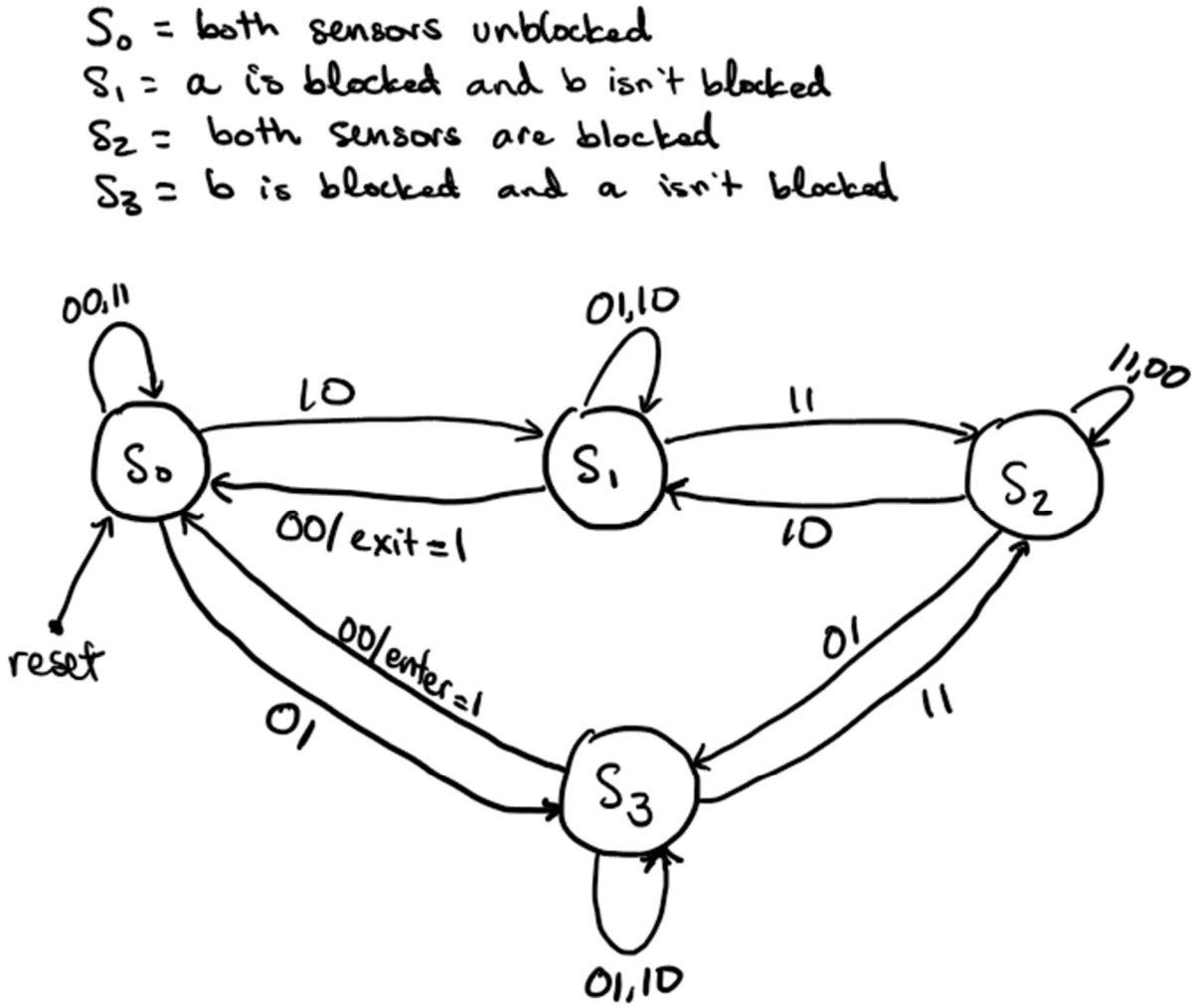


Figure 2: State diagram for sensor detection of exit and enter

**Counter Task**

Approaching this problem, I started by drawing out the state diagram for the counters of 3 bit and 5-bit. Depending on the input of incr and decr, the counter either moves up for incr and down for decr. These values are 1-7 for 3-bit and 31 for 5-bit. However, in this problem, for 3-bit, the maximum capacity of the parking lot is 5 cars, so after 5, the counter will still increment, but the display value will be nothing as it cannot past 5 cars. The same is for 5-bit, as the maximum value is 25, and at that state, the display will be 25, and anything more is nothing. Afterwards, I implemented this design on Verilog with two inputs and six outputs that correspond to the individual 7-segment hex display.

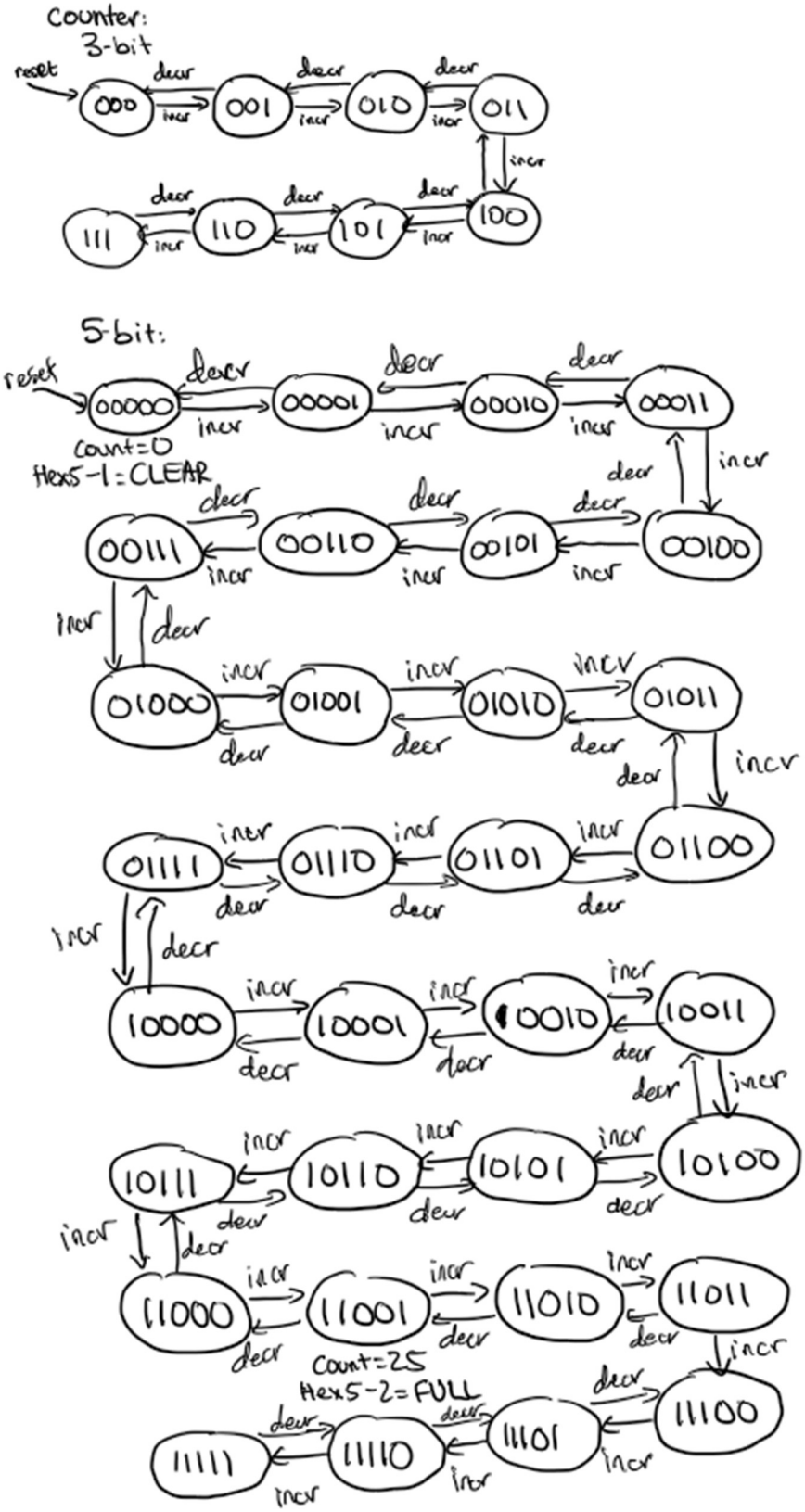


Figure 3: State diagram 3- and 5-bit counters

## Results

### Sensor Input Task:

For the first part, I tested if the finite state machine will go through the correct states by checking if enter increments on the input sequence of 10, 11, 01, 00. I also tested for the opposite of enter, the exit output as it should increment on the input sequence of 01, 11, 10, 00. As seen in figure 4, the sequence of inputs produced the correct outputs for both enter and exit. For the simulations, the input and output progresses if the clock is at the positive edge.

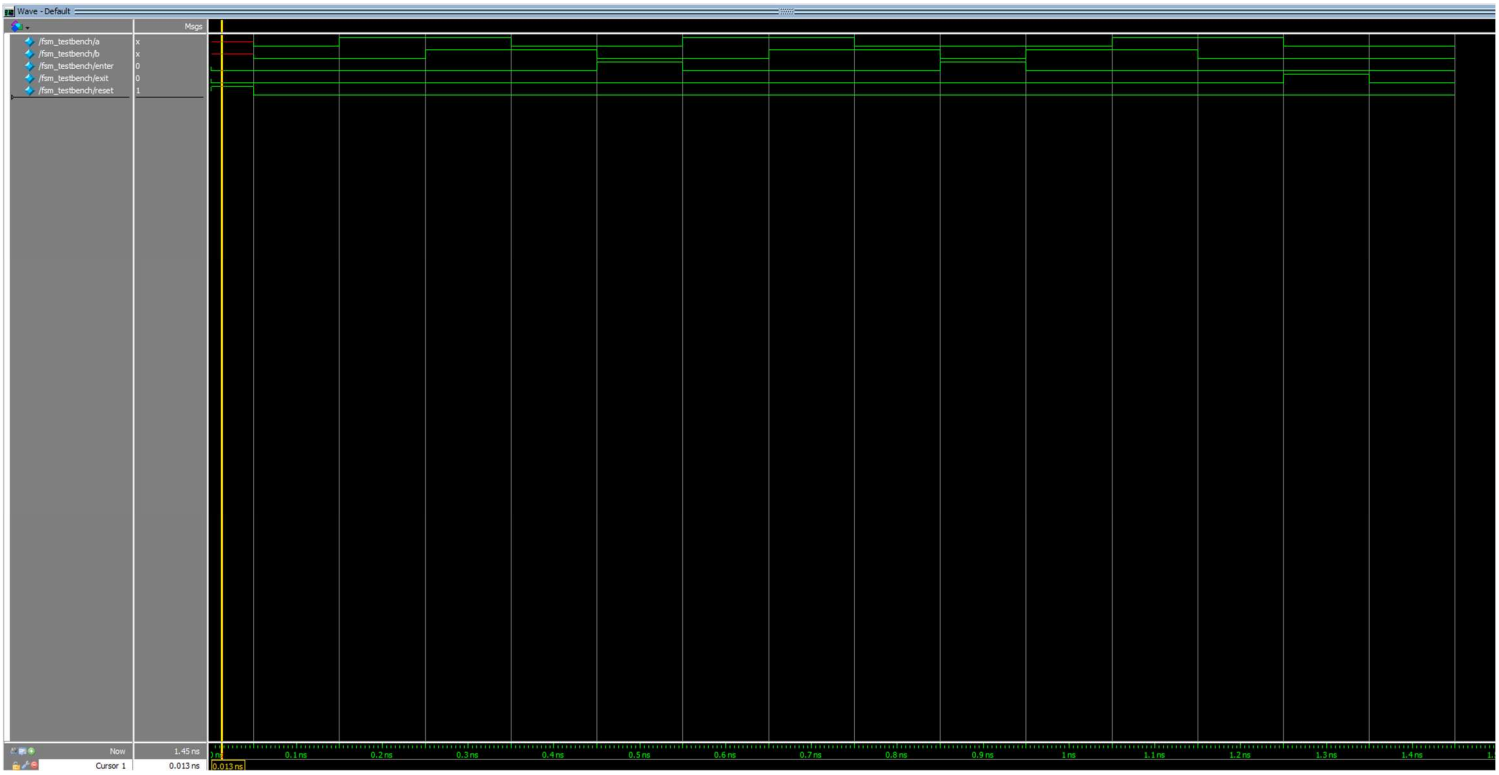


Figure 4: The waveform generated by the FSM for the Sensor Inputs

### Counter Task:

For this simulation, I tested the inputs of incr and decr in order to check if the values of display, hex1, hex2, hex3, hex4, and hex5 are correct in order to be transferred to the DE1_SoC module and the 7-segment hex display board. As seen in the waveform generated by the counter module in figure 5, the hex display board starts at “CLEAR0” then progresses to “01” and continues to count when incr is true. It was difficult to fit all the simulations towards the maximum capacity of 25, however, I was able to run the simulation individually, and confirmed the output at the count of 25 to be “FULL25”. As a result, this waveform simulation shows that my counter module works perfectly with the inputs of incr and decr.

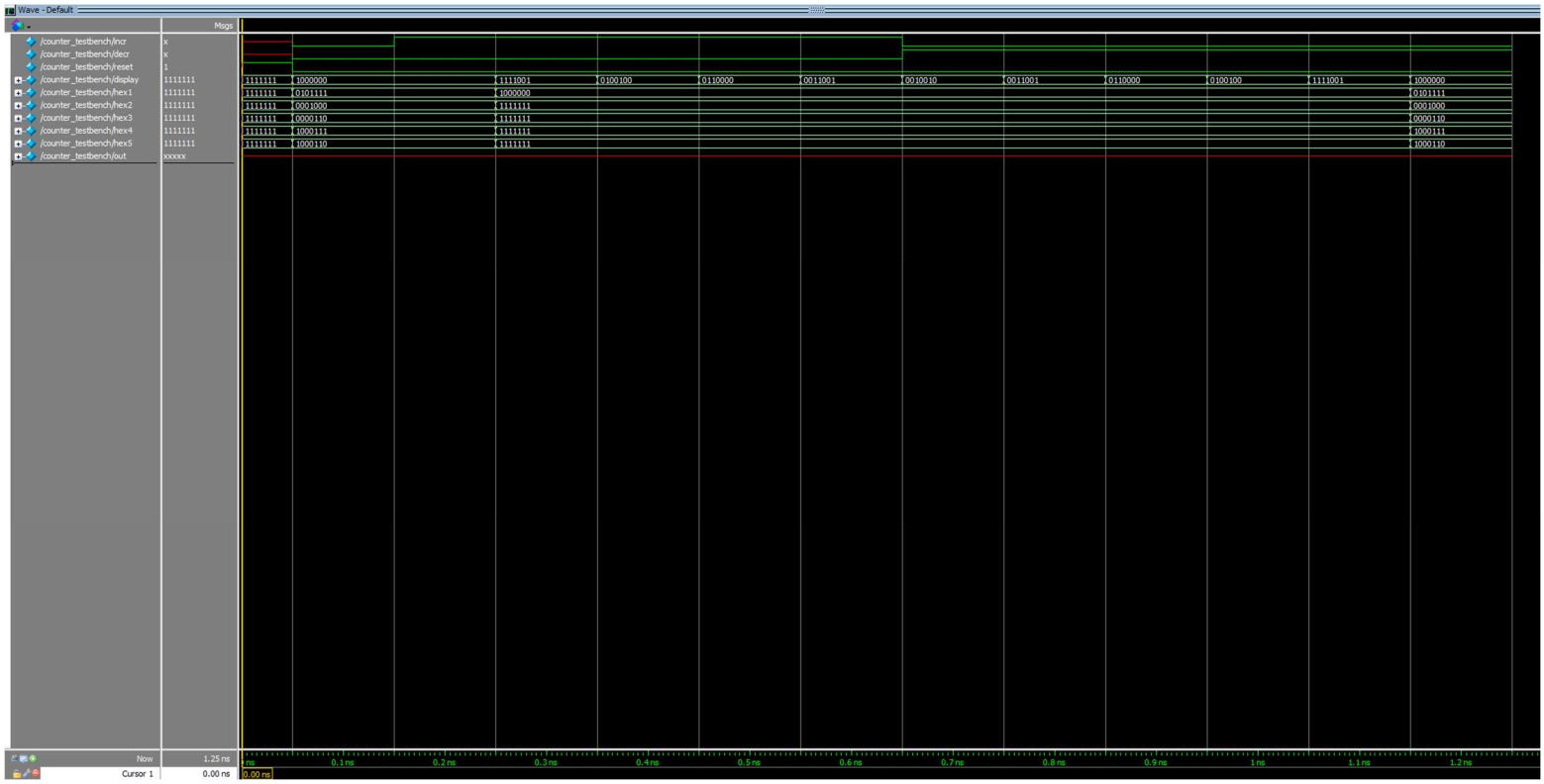


Figure 5: The waveform generated by the wind module

The size of FSM (fsm parking module) is 27+7, which equals 34. This is the size of the design in terms of FPGA logic and DFF resources with the cost of the clock divider as the cost of the clock divider isn't given.

Analysis & Synthesis Resource Utilization by Entity

<<Filter>>

| | 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 | 27 (0) | 7 (0) | 0 | 0 | 87 | 0 | \|DE1_SoC | DE1_SoC | work |
| 1 | \|counter:count\| | 23 (23) | 5 (5) | 0 | 0 | 0 | 0 | \|DE1_SoC\|counter:count | counter | work |
| 2 | \|fsm:parking\| | 4 (4) | 2 (2) | 0 | 0 | 0 | 0 | \|DE1_SoC\|fsm:parking | fsm | work |

Figure 6: Analysis and Synthesis Resource Utilization of the DE1_SoC

## Final Product

The overarching goal of this project was to design a system that takes in the inputs of two sensors in order to detect the movement of cars going in or out of a parking lot. The goal was also to develop a counter to keep track of the number of cars currently in the parking lot. Overall, Overall, this project was designed to fit the goals as it was able to take in the inputs of two switches (sensors) and traverse through the finite state machine in order to determine if a car has entered or exited the parking lot. From then, the counter takes care of the number of cars, incrementing the value when a car enters and decrementing when a car exits. At 25 cars, the hex board displays full and if another car enters then it displays nothing as that is not possible. I was able to develop each part of the design, from the parking lot sensor inputs to the counters, both work in conjunction to develop a parking lot occupancy counter system. The difference on what was asked, and the finished product is minimal as it executed inputs with the correct outputs.

# Appendix: SystemVerilog Code

## 1) fsm.sv

```
1   //Khoa Tran
2   //04/12/2021
3   //CSE 371
4   //Lab 1, Task 1
5   //This module represents the finite state machine for the parking lot as it represents
6   //the input of sensor a and b. As a and b are one-bit inputs, the module goes through
7   //the order of progression of the two inputs in order to determine either an enter or
8   //an exit output value.
9
10  //Implementing the fsm module with the inputs clk, reset, a, and b. The enter and
11  //exit variable outputs are a one-bit value indicating either the parking lot got one
12  //more or less car. This fsm module allows for reset to start the input sequence and
13  //current state over again. Lastly, the module progress to the next input value when clk is
    a positive edge.
14  module fsm (clk, reset, a, b, enter, exit);
15     input logic clk, reset; //input for clk, reset
16     input logic a, b; //input for sensor a and b
17     output logic exit; //output for exit
18     output logic enter; //output for enter
19
20     enum {S0, S1, S2, S3} ps, ns; //all five states, present state, next state
21
22     always_comb begin //comb next state logic
23        case (ps)
24           S0: if (a & ~b) ns = S1; //initial state, both sensor is unblocked
25                  else if (~a & b) ns = S3;
26                  else ns = S0;
27           S1: if (a & b) ns = S2; //state of sensor a blocked, and b unblocked
28                  else if (~a & ~b) ns = S0;
29                  else ns = S1;
30           S2: if (a & ~b) ns = S1; //state of both sensors blocked
31                  else if (~a & b) ns = S3;
32                  else ns = S2;
33           S3: if (a & b) ns = S2; //state of sensor b blocked, and a unblocked
34                  else if (~a & ~b) ns = S0;
35                  else ns = S3;
36        endcase
37     end
38
39     // equation for output of enter and exit as it depends on present state and input, a and
    b.
40     assign enter = ((ps == S3) & ~a & ~b);
41     assign exit = ((ps == S1) & ~a & ~b);
42
43        //sequential logic (DFFS)
44        always_ff @(posedge clk) begin
45           if (reset)
46              ps <= S0;
47           else
48              ps <= ns;
49        end
50  endmodule
51
52  //Module test the output of the fsm module by running a sequence of inputs
53  //on a and b and testing if the output, enter and exit, is correct.
54  module fsm_testbench();
55
56     logic clk, reset, a, b, enter, exit;
57
58     //devices under test
59     fsm dut (.clk, .reset, .a, .b, .enter, .exit);
60
61     //clock setup
62     parameter clock_period = 100;
63
64     initial begin
65        clk <= 0;
66        forever #(clock_period /2) clk <= ~clk;
67     end
68     //initial simulation
69     initial begin
70        reset <= 1;                   @(posedge clk);
71        reset <= 0; a<=0; b<=0;       @(posedge clk);
72        reset <= 0; a<=1; b<=0;       @(posedge clk);
73        reset <= 0; a<=1; b<=1;       @(posedge clk);
74        reset <= 0; a<=0; b<=1;       @(posedge clk);
```





```
75        reset <= 0; a<=0; b<=0;       @(posedge clk);
76        reset <= 0; a<=1; b<=0;       @(posedge clk);
77        reset <= 0; a<=1; b<=1;       @(posedge clk);
78        reset <= 0; a<=0; b<=1;       @(posedge clk);
79        reset <= 0; a<=0; b<=0;       @(posedge clk);
80        reset <= 0; a<=0; b<=1;       @(posedge clk);
81        reset <= 0; a<=1; b<=1;       @(posedge clk);
82        reset <= 0; a<=1; b<=0;       @(posedge clk);
83        reset <= 0; a<=0; b<=0;       @(posedge clk);
84        reset <= 0; a<=0; b<=0;       @(posedge clk);
85
86        $stop; //end simulation
87     end
88  endmodule
89
```

## 2) counter.sv

```
1   //Khoa Tran
2   //04/12/2021
3   //CSE 371
4   //Lab 1, Counter Task 2
5   //This module implements a sequential circuit that acts as a register
6   //that goes through a specific state sequence. In this module, it is a
7   //counter that has a specific n-bit size, and counts from 0 to 2^(n-1)
8   //in binary. This module counts upwards at positive clock edge. This
9   //module allows reset to make the output go back to 0 when reset is called.
10  //As the counter increases or decreases, the 7-bit values for the display of hex0
11  //to hex5 is given in a combinational circuit in order to display the current
12  //number of cars in the parking lot. At 0 cars, the display is "CLEAR0" and at 25
13  //cars, it displays "FULL25". For all other values, it will just display the value.
14
15  //Implements counter module with input incr, decr, reset, clk, display, hex1, hex2,
16  //hex3, hex4, and hex5, along with a parameter to initialize the n-bit size desired.
17  //The output variable is display and hex1 through hex5, which are the outputs
18  //to the display on the 7-segment HEX board as it shows the current count of cars
19  //incrementing when incr is 1 and decrementing when decr is 1. This module counts from 0 to
    2^(n-1)
20  //in binary and counts upwards at positive clock edge. This
21  //module allows reset to make the output go back to 0 when reset is called. As the
    counter's count
22  //increase or decrease, the display outputs the 7-bit binary output for the HEX board
23  //displaying the current digital value. When the binary value equals 25,
24  //the parking lot is full and shows a message displaying that.
25  module counter #(parameter width = 5) (incr, decr, reset, clk, display, hex1, hex2, hex3,
    hex4, hex5);
26      logic [width - 1:0] out; //current value
27      input logic incr, decr, reset, clk; //input incr, decr, reset, and clk
28      output logic [6:0] display; //outputs for HEX0-5
29      output logic [6:0] hex1;
30      output logic [6:0] hex2;
31      output logic [6:0] hex3;
32      output logic [6:0] hex4;
33      output logic [6:0] hex5;
34
35      //7-seg hex display for values zero through nine
36      logic [6:0] zero, one, two, three, four, five, six, seven, eight, nine;
37      assign zero = 7'b1000000;
38      assign one = 7'b1111001;
39      assign two = 7'b0100100;
40      assign three = 7'b0110000;
41      assign four = 7'b0011001;
42      assign five = 7'b0010010;
43      assign six = 7'b0000010;
44      assign seven = 7'b1111000;
45      assign eight = 7'b0000000;
46      assign nine = 7'b0010000;
47
48      //sequential (DFFS) incr and decr out
49      always_ff @(posedge clk)
50      begin
51          if (reset)
52              out <= 0;
53          else if (incr)
54              out <= out + 1;
55          else if (decr)
56              out <= out - 1;
57      end
58
59      //combinational for cases of out values 1-25
60      always_comb begin
61              case (out)
62                  5'b00000: begin //0 (CLEAR0)
63                              hex5 = 7'b1000110;
64                              hex4 = 7'b1000111;
65                              hex3 = 7'b0000110;
66                              hex2 = 7'b0001000;
67                              hex1 = 7'b0101111;
68                              display = zero;
69                              end
70                  5'b00001: begin
71                              hex5 = 7'b1111111;
72                              hex4 = 7'b1111111;
73                              hex3 = 7'b1111111;
```



```
74                              hex2 = 7'b1111111;
75                              hex1 = zero;
76                              display = one;
77                              end
78                  5'b00010: begin
79                              hex5 = 7'b1111111;
80                              hex4 = 7'b1111111;
81                              hex3 = 7'b1111111;
82                              hex2 = 7'b1111111;
83                              hex1 = zero;
84                              display = two;
85                              end
86                  5'b00011: begin
87                              hex5 = 7'b1111111;
88                              hex4 = 7'b1111111;
89                              hex3 = 7'b1111111;
90                              hex2 = 7'b1111111;
91                              hex1 = zero;
92                              display = three;
93                              end
94                  5'b00100: begin
95                              hex5 = 7'b1111111;
96                              hex4 = 7'b1111111;
97                              hex3 = 7'b1111111;
98                              hex2 = 7'b1111111;
99                              hex1 = zero;
100                             display = four;
101                             end
102                 5'b00101: begin
103                             hex5 = 7'b1111111;
104                             hex4 = 7'b1111111;
105                             hex3 = 7'b1111111;
106                             hex2 = 7'b1111111;
107                             hex1 = zero;
108                             display = five;
109                             end
110                 5'b00110: begin
111                             hex5 = 7'b1111111;
112                             hex4 = 7'b1111111;
113                             hex3 = 7'b1111111;
114                             hex2 = 7'b1111111;
115                             hex1 = zero;
116                             display = six;
117                             end
118                 5'b00111: begin
119                             hex5 = 7'b1111111;
120                             hex4 = 7'b1111111;
121                             hex3 = 7'b1111111;
122                             hex2 = 7'b1111111;
123                             hex1 = zero;
124                             display = seven;
125                             end
126                 5'b01000: begin
127                             hex5 = 7'b1111111;
128                             hex4 = 7'b1111111;
129                             hex3 = 7'b1111111;
130                             hex2 = 7'b1111111;
131                             hex1 = zero;
132                             display = eight;
133                             end
134                 5'b01001: begin
135                             hex5 = 7'b1111111;
136                             hex4 = 7'b1111111;
137                             hex3 = 7'b1111111;
138                             hex2 = 7'b1111111;
139                             hex1 = zero;
140                             display = nine;
141                             end
142                 5'b01010: begin
143                             hex5 = 7'b1111111;
144                             hex4 = 7'b1111111;
145                             hex3 = 7'b1111111;
146                             hex2 = 7'b1111111;
147                             hex1 = one;
148                             display = zero;
149                             end
```

```
150                     5'b01011: begin
151                             hex5 = 7'b1111111;
152                             hex4 = 7'b1111111;
153                             hex3 = 7'b1111111;
154                             hex2 = 7'b1111111;
155                             hex1 = one;
156                             display = one;
157                             end
158                     5'b01100: begin
159                             hex5 = 7'b1111111;
160                             hex4 = 7'b1111111;
161                             hex3 = 7'b1111111;
162                             hex2 = 7'b1111111;
163                             hex1 = one;
164                             display = two;
165                             end
166                     5'b01101: begin
167                             hex5 = 7'b1111111;
168                             hex4 = 7'b1111111;
169                             hex3 = 7'b1111111;
170                             hex2 = 7'b1111111;
171                             hex1 = one;
172                             display = three;
173                             end
174                     5'b01110: begin
175                             hex5 = 7'b1111111;
176                             hex4 = 7'b1111111;
177                             hex3 = 7'b1111111;
178                             hex2 = 7'b1111111;
179                             hex1 = one;
180                             display = four;
181                             end
182                     5'b01111: begin
183                             hex5 = 7'b1111111;
184                             hex4 = 7'b1111111;
185                             hex3 = 7'b1111111;
186                             hex2 = 7'b1111111;
187                             hex1 = one;
188                             display = five;
189                             end
190                     5'b10000: begin
191                             hex5 = 7'b1111111;
192                             hex4 = 7'b1111111;
193                             hex3 = 7'b1111111;
194                             hex2 = 7'b1111111;
195                             hex1 = one;
196                             display = six;
197                             end
198                     5'b10001: begin
199                             hex5 = 7'b1111111;
200                             hex4 = 7'b1111111;
201                             hex3 = 7'b1111111;
202                             hex2 = 7'b1111111;
203                             hex1 = one;
204                             display = seven;
205                             end
206                     5'b10010: begin
207                             hex5 = 7'b1111111;
208                             hex4 = 7'b1111111;
209                             hex3 = 7'b1111111;
210                             hex2 = 7'b1111111;
211                             hex1 = one;
212                             display = eight;
213                             end
214                     5'b10011: begin
215                             hex5 = 7'b1111111;
216                             hex4 = 7'b1111111;
217                             hex3 = 7'b1111111;
218                             hex2 = 7'b1111111;
219                             hex1 = one;
220                             display = nine;
221                             end
222                     5'b10100: begin
223                             hex5 = 7'b1111111;
224                             hex4 = 7'b1111111;
225                             hex3 = 7'b1111111;
```



```
226                             hex2 = 7'b1111111;
227                             hex1 = two;
228                             display = zero;
229                             end
230                     5'b10101: begin
231                             hex5 = 7'b1111111;
232                             hex4 = 7'b1111111;
233                             hex3 = 7'b1111111;
234                             hex2 = 7'b1111111;
235                             hex1 = two;
236                             display = one;
237                             end
238                     5'b10110: begin
239                             hex5 = 7'b1111111;
240                             hex4 = 7'b1111111;
241                             hex3 = 7'b1111111;
242                             hex2 = 7'b1111111;
243                             hex1 = two;
244                             display = two;
245                             end
246                     5'b10111: begin
247                             hex5 = 7'b1111111;
248                             hex4 = 7'b1111111;
249                             hex3 = 7'b1111111;
250                             hex2 = 7'b1111111;
251                             hex1 = two;
252                             display = three;
253                             end
254                     5'b11000: begin
255                             hex5 = 7'b1111111;
256                             hex4 = 7'b1111111;
257                             hex3 = 7'b1111111;
258                             hex2 = 7'b1111111;
259                             hex1 = two;
260                             display = four;
261                             end
262                     5'b11001: begin //25 (FULL25)
263                             hex5 = 7'b0001110;
264                             hex4 = 7'b1000001;
265                             hex3 = 7'b1000111;
266                             hex2 = 7'b1000111;
267                             hex1 = two;
268                             display = five;
269                             end
270                     default: begin
271                             hex5 = 7'b1111111;
272                             hex4 = 7'b1111111;
273                             hex3 = 7'b1111111;
274                             hex2 = 7'b1111111;
275                             hex1 = 7'b1111111;
276                             display = 7'b1111111;
277                             end
278                     endcase
279             end
280     endmodule
```

```
281
282   //Module test the output of the counter module by running a sequence of inputs
283   //on reset, incr and decr to test if the output of display, hex1, hex2, hex3, hex4, and hex5.
284   //is correct along the positive edges, by making sure out counts up by 1 when incr is 1 and
285   //decrements when decr is 1.
286   module counter_testbench#(parameter width = 5)();
287
288       logic incr, decr, reset, clk;
289       logic [6:0] display;
290       logic [6:0] hex1;
291       logic [6:0] hex2;
292       logic [6:0] hex3;
293       logic [6:0] hex4;
294       logic [6:0] hex5;
295       logic [width - 1:0] out;
296
297       //devices under test
298       counter dut (.incr, .decr, .reset, .clk, .display, .hex1, .hex2, .hex3, .hex4, .hex5);
299
300       //clock setup
301       parameter clock_period = 100;
```



```
302
303       initial begin
304          clk <= 0;
305          forever #(clock_period /2) clk <= ~clk;
306       end
307       //initial simulation
308       initial begin
309          reset <= 1;                              @(posedge clk);
310          reset <= 0; incr<=0; decr<=0;            @(posedge clk);
311          reset <= 0; incr<=1; decr<=0;            @(posedge clk);
312          reset <= 0; incr<=1; decr<=0;            @(posedge clk);
313          reset <= 0; incr<=1; decr<=0;            @(posedge clk);
314          reset <= 0; incr<=1; decr<=0;            @(posedge clk);
315          reset <= 0; incr<=1; decr<=0;            @(posedge clk);
316          reset <= 0; incr<=0; decr<=1;            @(posedge clk);
317          reset <= 0; incr<=0; decr<=1;            @(posedge clk);
318          reset <= 0; incr<=0; decr<=1;            @(posedge clk);
319          reset <= 0; incr<=0; decr<=1;            @(posedge clk);
320          reset <= 0; incr<=0; decr<=1;            @(posedge clk);
321          reset <= 0; incr<=0; decr<=1;            @(posedge clk);
322
323          $stop; //end simulation
324       end
325   endmodule
326
```

## 3) DE1_SoC.sv

```
1   //Khoa Tran
2   //04/12/2021
3   //CSE 371
4   //Lab 1, DE1_SoC Task3
5   //This module, DE1_SoC takes the inout swtiches of the GPIO_0, Clock_50 input,
6   //in order to have the switches be connected to the LED on the breadboard. These
7   //switches are connected to the fsm of the parking lot problem, as the first GPIO
8   //switch is connected to input a, and the other switch connected to input b.
9   //From these inputs, the fsm will output a value for the counter to either increment
10  //or decrement. Afterwards, the output of the counter indicates the amount of cars in the
    parking
11  //lot, displaying the values on the hex 7-segment display. The reset is connected to
12  //the third switch on the breadboard, allowing the state of the parking lot to go back
13  //to zero cars. Additionally, implements a clock that is connected to LEDR[5] and will
14  //express the input whenever the clock, and led is on with the positve edge.
15  module DE1_SoC (CLOCK_50, HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, LEDR, GPIO_0);
16     input logic CLOCK_50; // 50MHz clock
17     output logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
18     output logic [9:0] LEDR;
19
20     inout logic [33:0] GPIO_0;
21
22
23     logic reset;
24     logic [31:0] div_clk;
25
26     assign reset = GPIO_0[7]; //third switch connected to reset
27     parameter whichClock = 22; // 0.75 Hz clock
28
29     assign GPIO_0[26] = GPIO_0[5]; //connecting switches to leds
30     assign GPIO_0[27] = GPIO_0[6];
31
32     clock_divider cdiv (.clock(CLOCK_50), .reset(reset), .divided_clocks(div_clk));
33
34     logic clkSelect;
35
36     assign clkSelect = CLOCK_50; // for simulation
37
38     //assign clkSelect = div_clk[whichClock]; // for board
39
40     //LEDR[0] connected to clk
41     assign LEDR[0] = clkSelect;
42
43     // logic variables for output of fsm
44     logic enterOut;
45     logic exitOut;
46
47     fsm parking(.clk(clkSelect), .reset(reset), .a(GPIO_0[5]), .b(GPIO_0[6]), .enter(enterOut
    ), .exit(exitOut));
48     //output of fsm goes into counter for count and display of values
49     counter count(.incr(enterOut), .decr(exitOut), .reset(reset), .clk(clkSelect), .display(
    HEX0), .hex1(HEX1), .hex2(HEX2), .hex3(HEX3), .hex4(HEX4), .hex5(HEX5));
50
51
52  endmodule
53
54  //Module test the output of the DE1_SoC module by running a sequence of inputs
55  //on the switches on the GPIO[5] and GPIO[6] and GPIO[7] as the reset
56  //in order to test if the output on the HEX0, HEX1, HEX2, HEX3, HEX4, and HEX5
57  //is correct along the positive edges.
58  module DE1_SoC_testbench();
59     logic CLOCK_50;
60     logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
61     logic [9:0] LEDR;
62     wire [33:0] GPIO_0;
63     logic Sw0, Sw1, reset;
64
65     DE1_SoC dut (.CLOCK_50, .HEX0, .HEX1, .HEX2, .HEX3, .HEX4, .HEX5, .LEDR, .GPIO_0);
66
67     assign GPIO_0[5] = Sw0;
68     assign GPIO_0[6]= Sw1;
69     assign GPIO_0[7] = reset;
70
71     parameter clock_period = 100;
72
73        initial begin
```



```
74           CLOCK_50 <= 0;
75           forever #(clock_period /2) CLOCK_50 <= ~CLOCK_50;
76
77        end //initial
78
79
80     initial begin
81           reset <= 1; @(posedge CLOCK_50);
82           reset <= 0; @(posedge CLOCK_50);
83
84           Sw0 <= 0; Sw1 <= 0; @(posedge CLOCK_50);
85           Sw0 <= 1; Sw1 <= 0; @(posedge CLOCK_50);
86           Sw0 <= 1; Sw1 <= 1; @(posedge CLOCK_50);
87           Sw0 <= 0; Sw1 <= 1; @(posedge CLOCK_50);
88           Sw0 <= 0; Sw1 <= 0; @(posedge CLOCK_50);
89           Sw0 <= 1; Sw1 <= 0; @(posedge CLOCK_50);
90           Sw0 <= 1; Sw1 <= 1; @(posedge CLOCK_50);
91           Sw0 <= 0; Sw1 <= 1; @(posedge CLOCK_50);
92           Sw0 <= 0; Sw1 <= 0; @(posedge CLOCK_50);
93           Sw0 <= 0; Sw1 <= 1; @(posedge CLOCK_50);
94           Sw0 <= 1; Sw1 <= 1; @(posedge CLOCK_50);
95           Sw0 <= 1; Sw1 <= 0; @(posedge CLOCK_50);
96           Sw0 <= 0; Sw1 <= 0; @(posedge CLOCK_50);
97           Sw0 <= 0; Sw1 <= 0; @(posedge CLOCK_50);
98
99     $stop;
100 end
101
102 endmodule
103
104 //clock_divider module has inputs of the clock, reset, and the
105 //32 bit divided_clock which allows to sequence through and
106 //output whenever the postive edge has been reached on the clock
107 // divided_clocks[0] = 25MHz, [1] = 12.5Mhz, ... [23] = 3Hz, [24] = 1.5Hz, [25] = 0.75Hz, ...
108 module clock_divider (clock, reset, divided_clocks);
109    input logic reset, clock;
110    output logic [31:0] divided_clocks = 0;
111
112    always_ff @(posedge clock) begin
113       divided_clocks <= divided_clocks + 1;
114    end
115
116 endmodule
```