```
1   // Alan Li
2   // 01/15/2022
3   // EE 371
4   // Lab #1
5
6   // counter takes 1-bit enter and exit as inputs and return 5-bit cout
    as output.
7   // The module functions as its name. When there is an enter signal, the
    cout increse by 1. When there is an exit signal, the cout decrease by 1.
8   // The range for the counter is from 0 to 25.
9
10  module counter(clk, reset, enter, exit, cout);
11
12     input logic clk, reset, enter, exit;
13     output logic [4:0] cout;
14     logic [4:0] ps, ns;
15
16     // Each decimal from 0 to 25 has been assigned with a 5-bit binary
    number.
17     parameter [4:0] zero = 5'b0,
18        one = 5'b1,
19        two = 5'b10,
20        three = 5'b11,
21        four = 5'b100,
22        five = 5'b101,
23        six = 5'b110,
24        seven = 5'b111,
25        eight = 5'b1000,
26        nine = 5'b1001,
27        ten = 5'b1010,
28        eleven = 5'b1011,
29        twelve = 5'b1100,
30        thirteen = 5'b1101,
31        fourteen = 5'b1110,
32        fifteen = 5'b1111,
33        sixteen = 5'b10000,
34        seventeen = 5'b10001,
35        eighteen = 5'b10010,
36        nineteen = 5'b10011,
37        twenty = 5'b10100,
38        twentyone = 5'b10101,
39        twentytwo = 5'b10110,
40        twentythree = 5'b10111,
41        twentyfour = 5'b11000,
42        twentyfive = 5'b11001;
43
44     // 25 states for the counter. Each state represent a different
    number.
45     // When there is a enter signal, the counter goes to next state
    which the number increase by one and vice versa.
46     // At state 0, if there is another exit signal(which should not
    happen in reality), the state will stay at zero.
47     // At state 25, if there is another enter signal, the state will
    stay at 25.
48        case(ps)
49           zero: if(enter)          ns = one;
50                 else               ns = zero;
51           one: if(enter)           ns = two;
```

```
52                else if(exit)     ns = zero;
53                else              ns = one;
54          two: if(enter)          ns = three;
55                else if(exit)     ns = one;
56                else              ns = two;
57          three:if(enter)         ns = four;
58                else if(exit)     ns = two;
59                else              ns = three;
60          four: if(enter)         ns = five;
61                else if(exit)     ns = three;
62                else              ns = four;
63
64          /*
65          five: if(exit)          ns = four;
66                else              ns = five; // for demo purpose
67          */
68
69
70          five: if(enter)         ns = six;
71                else if(exit)     ns = four;
72                else              ns = five; // for lab design purpose
73
74          six: if(enter)          ns = seven;
75                else if(exit)     ns = five;
76                else              ns = six;
77          seven: if(enter)        ns = eight;
78                else if(exit)     ns = six;
79                else              ns = seven;
80          eight:if(enter)         ns = nine;
81                else if(exit)     ns = seven;
82                else              ns = eight;
83          nine: if(enter)         ns = ten;
84                else if(exit)     ns = eight;
85                else              ns = nine;
86          ten:  if(enter)         ns = eleven;
87                else if(exit)     ns = nine;
88                else              ns = ten;
89          eleven:if(enter)        ns = twelve;
90                else if(exit)     ns = ten;
91                else              ns = eleven;
92          twelve:if(enter)        ns = thirteen;
93                else if(exit)     ns = eleven;
94                else              ns = twelve;
95          thirteen:if(enter)      ns = fourteen;
96                else if(exit)     ns = twelve;
97                else              ns = thirteen;
98          fourteen: if(enter)     ns = fifteen;
99                else if(exit)     ns = thirteen;
100               else              ns = fourteen;
101         fifteen:if(enter)       ns = sixteen;
102               else if(exit)     ns = fourteen;
103               else              ns = fifteen;
104         sixteen: if(enter)      ns = seventeen;
105               else if(exit)     ns = fifteen;
106               else              ns = sixteen;
107         seventeen:if(enter)     ns = eighteen;
108               else if(exit)     ns = sixteen;
109               else              ns = seventeen;
```

```
110             eighteen:if(enter)      ns = nineteen;
111                   else if(exit)     ns = seventeen;
112                   else              ns = eighteen;
113             nineteen:if(enter)      ns = twenty;
114                   else if(exit)     ns = eighteen;
115                   else              ns = nineteen;
116             twenty:if(enter)        ns = twentyone;
117                   else if(exit)     ns = nineteen;
118                   else              ns = twenty;
119             twentyone:if(enter)     ns = twentytwo;
120                   else if(exit)     ns = twenty;
121                   else              ns = twentyone;
122             twentytwo:if(enter)     ns = twentythree;
123                   else if(exit)     ns = twentyone;
124                   else              ns = twentytwo;
125             twentythree:if(enter)   ns = twentyfour;
126                   else if(exit)     ns = twentytwo;
127                   else              ns = twentythree;
128             twentyfour:if(enter)    ns = twentyfive;
129                   else if(exit)     ns = twentythree;
130                   else              ns = twentyfour;
131             twentyfive:if(exit)     ns = twentyfour;
132                   else              ns = twentyfive;
133             endcase
134       end
135
136       // When reset is pressed, the counter will reset to state 0.
137       always @(posedge clk) begin
138          if(reset) begin
139             ps <= zero;
140          end
141          else begin
142             cout <= ps;
143             ps <= ns;
144          end
145       end
146    endmodule
147
148    // counter_testbench tests all expected behavior that the parking lot
       occupancy counter system in the lab may encounter
149    module counter_testbench();
150       logic clk, reset, enter, exit;
151       logic [4:0]cout;
152
153       counter dut (.clk(clk), .reset(reset), .enter(enter), .exit(exit), .
       cout(cout));
154
155       parameter CLOCK_PERIOD = 100;
156
157       initial begin
158          clk <= 0;
159          forever #(CLOCK_PERIOD/2) clk <= ~clk;
160       end
161
162       // 30 cars enters, the counter will reach 25 and stay there
163       // 30 cars exit(for simulation), the counter will reach 0 and stay
       there
164       initial begin
```

```
165         reset <= 1;             @(posedge clk);
166         reset <= 0;             @(posedge clk);
167         enter <= 1; exit <= 0; repeat(30) @(posedge clk);
168         enter <= 0; exit <= 1; repeat(30) @(posedge clk);
169         $stop;
170       end
171   endmodule
172
173
174
```

```
1   // Alan Li
2   // 01/15/2022
3   // EE 371
4   // Lab #1
5
6   // DE1_Soc takes 34-bit GPIO_0 and return 1-bit enterIndicator and
    exitIndicator, 5-bit countIndicator and inputA, inputB
7   // This serves as the top-level module for the parking lot occupancy
    counter system
8
9   module DE1_SoC(CLOCK_50, GPIO_0, HEX5, HEX4, HEX3, HEX2, HEX1, HEX0);
10
11      // GPIO_0[5] is connected to the reset switch, GPIO_0[6] is
    connected to the sensorA switch, GPIO_0[7] is connected to the sensorB
    switch
12      inout logic [33:0] GPIO_0;
13      input logic CLOCK_50;
14      output logic [6:0] HEX5, HEX4, HEX3, HEX2, HEX1, HEX0;
15      logic enterIndicator, exitIndicator;
16      logic [4:0] countIndicator;
17      logic inputA, inputB;
18
19
20      assign GPIO_0[26] = GPIO_0[6]; // when switch A(sensor A) is
    enabled/triggered the corresponding LED turns on
21      assign GPIO_0[27] = GPIO_0[7]; // when switch B(sensor B) is
    enabled/triggered the corresponding LED turns on
22
23      // userInput inA and inB takes GPIO[6] and GPIO[7] as input
    parameters and return Q as inputA and inputB respectively
24      userInput inA (.clk(CLOCK_50), .D(GPIO_0[6]), .Q(inputA));
25      userInput inB (.clk(CLOCK_50), .D(GPIO_0[7]), .Q(inputB));
26
27      // parkingLotSensors myfsm takes CLOCK_50 as clk, GPIO[5], inputA,
    input B as parameters to reset, A, B
28      // and returns enter and exit as enterIndicator and exitIndicator
    respectively
29      parkingLotSensors myfsm (.clk(CLOCK_50), .reset(GPIO_0[5]), .A(inputA
    ), .B(inputB), .enter(enterIndicator), .exit(exitIndicator));
30
31      // counter mycounter takes CLOCK_50 as clk, GPIO_0[5],
    enterIndicator and exitIndicator as reset, enter and exit
32      // it returns enter, exit and cout as countIndicator[4:0]
33      counter mycounter (.clk(CLOCK_50), .reset(GPIO_0[5]), .enter(
    enterIndicator), .exit(exitIndicator), .cout(countIndicator[4:0]));
34
35      // hexDisplay mydisplay takes CLOCK_50 as clk, countIndicator[4:0]
    as inputCount
36      // and returns HEX5, HEX4, HEX3, HEX2, HEX1, HEX0 as HEX5, HEX4,
    HEX3, HEX2, HEX1, HEX0 respectively
37      hexDisplay mydisplay (.clk(CLOCK_50), .inputCount(countIndicator[4:0
    ]),
38                            .HEX5(HEX5), .HEX4(HEX4), .HEX3(HEX3), .HEX2(
    HEX2), .HEX1(HEX1), .HEX0(HEX0));
39  endmodule
40
41
42  // DE1_SoC_testbench tests all expected behavior that the parking lot
```

```
    occupancy counter system in the lab may encounter
43  module DE1_SoC_testbench();
44
45     logic clk, reset, A, B;
46     logic [6:0] HEX5, HEX4, HEX3, HEX2, HEX1, HEX0;
47
48     DE1_SoC dut (.CLOCK_50(CLOCK_50), .GPIO_0(GPIO_0),
49                             .HEX5(HEX5), .HEX4(HEX4), .HEX3(HEX3), .HEX2(
    HEX2), .HEX1(HEX1), .HEX0(HEX0));
50
51     //Set up the clock.
52     parameter CLOCK_PERIOD = 100;
53
54     initial begin
55        clk <= 0;
56        forever #(CLOCK_PERIOD/2) clk <= ~clk;
57
58     end
59
60     initial begin
61        reset <= 1;       @(posedge clk);   // cycle through car entering
62        reset <= 0;       @(posedge clk);
63                          @(posedge clk);
64                          @(posedge clk);
65        {A,B} <= 2'b00;   @(posedge clk);
66        {A,B} <= 2'b10;   @(posedge clk);
67        {A,B} <= 2'b11;   @(posedge clk);
68        {A,B} <= 2'b01;   @(posedge clk);
69        {A,B} <= 2'b00;   @(posedge clk);
70
71        {A,B} <= 2'b00;   @(posedge clk);
72        {A,B} <= 2'b10;   @(posedge clk);
73        {A,B} <= 2'b11;   @(posedge clk);
74        {A,B} <= 2'b01;   @(posedge clk);
75        {A,B} <= 2'b00;   @(posedge clk);
76
77        {A,B} <= 2'b00;   @(posedge clk);
78        {A,B} <= 2'b10;   @(posedge clk);
79        {A,B} <= 2'b11;   @(posedge clk);
80        {A,B} <= 2'b01;   @(posedge clk);
81        {A,B} <= 2'b00;   @(posedge clk);
82
83        {A,B} <= 2'b00;   @(posedge clk);
84        {A,B} <= 2'b10;   @(posedge clk);
85        {A,B} <= 2'b11;   @(posedge clk);
86        {A,B} <= 2'b01;   @(posedge clk);
87        {A,B} <= 2'b00;   @(posedge clk);   // 4 cars entered
88
89        {A,B} <= 2'b00;   @(posedge clk);
90        {A,B} <= 2'b01;   @(posedge clk);
91        {A,B} <= 2'b11;   @(posedge clk);
92        {A,B} <= 2'b10;   @(posedge clk);
93        {A,B} <= 2'b00;   @(posedge clk);
94
95        {A,B} <= 2'b00;   @(posedge clk);
96        {A,B} <= 2'b01;   @(posedge clk);
97        {A,B} <= 2'b11;   @(posedge clk);
98        {A,B} <= 2'b10;   @(posedge clk);
```

```
 99          {A,B} <= 2'b00;    @(posedge clk);
100
101          {A,B} <= 2'b00;    @(posedge clk);
102          {A,B} <= 2'b01;    @(posedge clk);
103          {A,B} <= 2'b11;    @(posedge clk);
104          {A,B} <= 2'b10;    @(posedge clk);
105          {A,B} <= 2'b00;    @(posedge clk);    // 3 cars exiting
106
107          reset <= 1;        @(posedge clk);    // cycle through car exiting
108          reset <= 0;        @(posedge clk);
109                             @(posedge clk);
110                             @(posedge clk);
111          {A,B} <= 2'b00;    @(posedge clk);
112          {A,B} <= 2'b01;    @(posedge clk);
113          {A,B} <= 2'b11;    @(posedge clk);
114          {A,B} <= 2'b10;    @(posedge clk);
115          {A,B} <= 2'b00;    @(posedge clk);
116                             @(posedge clk);
117          $stop;
118       end
119    endmodule
120
```

```
1   // Alan Li
2   // 01/15/2022
3   // EE 371
4   // Lab #1
5
6   // hexDisplay takes 5-bit inputCount as inputs and return 7-bits HEX5,
    HEX4, HEX3, HEX2, HEX1, HEX0 as output.
7   // Upon start, the HEX display will display "clear0", when enter signal
    is given, the HEX display the counter on HEX1 and HEX0. The counter
    will keep adding up until reach 25.
8   // Then the hex display will diaplay "FULL25".
9   // If the counter reach 0, the hex display will display "clear0" as
    there is no car in the parking lot
10
11  module hexDisplay(clk, inputCount, HEX5, HEX4, HEX3, HEX2,HEX1, HEX0);
12     input clk;
13     input logic [4:0] inputCount;
14     output logic [6:0] HEX5, HEX4, HEX3, HEX2, HEX1, HEX0;
15
16     // 7-bit parameter for the display to display different numbers or
    characters
17     //                                  6543210
18     parameter [6:0]   zero =   7'b1000000,
19                       one =    7'b1111001,
20                       two =    7'b0100100,
21                       three =  7'b0110000,
22                       four =   7'b0011001,
23                       five =   7'b0010010,
24                       six =    7'b0000010,
25                       seven =  7'b1111000,
26                       eight =  7'b0000000,
27                       nine =   7'b0011000,
28                       F =      7'b0001110, //FULL
29                       U =      7'b1000001,
30                       L =      7'b1000111,
31
32                       C =      7'b1000110, //CLEAR
33                       E =      7'b0000110,
34                       A =      7'b0001000,
35                       R =      7'b1001100,
36                       blk =    7'b1111111;
37
38
39     // The hex are driven off upon start.
40     // The hex will display counters when enter or exit signal is given
41     // When the counter is 0, it will display "clear0". When full it
    will display "full25"
42     always_comb begin
43        HEX5 = blk;
44        HEX4 = blk;
45        HEX3 = blk;
46        HEX2 = blk;
47        HEX1 = blk;
48        HEX0 = blk;
49        case(inputCount)
50           0: begin HEX5 = C; HEX4 = L; HEX3 = E; HEX2 = A; HEX1 = R; HEX0
     = zero; end
51           1: begin HEX5 = blk; HEX4 = blk; HEX3 = blk; HEX2 = blk; HEX1 =
```

```
      blk; HEX0 = one; end
52              2: begin HEX5 = blk; HEX4 = blk; HEX3 = blk; HEX2 = blk; HEX1 =
      blk; HEX0 = two; end
53              3: begin HEX5 = blk; HEX4 = blk; HEX3 = blk; HEX2 = blk; HEX1 =
      blk; HEX0 = three; end
54              4: begin HEX5 = blk; HEX4 = blk; HEX3 = blk; HEX2 = blk; HEX1 =
      blk;HEX0 = four; end
55              5: begin HEX5 = F; HEX4 = U; HEX3 = L; HEX2 = L;HEX1 = blk;
     HEX0 = five; end // for demo purpose
56
57              //5: begin HEX5 = blk; HEX4 = blk; HEX3 = blk; HEX2 = blk;
     HEX1 = blk;HEX0 = five; end
58              6: begin HEX5 = blk; HEX4 = blk; HEX3 = blk; HEX2 = blk; HEX1 =
      blk;HEX0 = six; end
59              7: begin HEX5 = blk; HEX4 = blk; HEX3 = blk; HEX2 = blk; HEX1 =
      blk;HEX0 = seven; end
60              8: begin HEX5 = blk; HEX4 = blk; HEX3 = blk; HEX2 = blk; HEX1 =
      blk;HEX0 = eight; end
61              9: begin HEX5 = blk; HEX4 = blk; HEX3 = blk; HEX2 = blk; HEX1 =
      blk;HEX0 = nine; end
62              10: begin HEX5 = blk; HEX4 = blk; HEX3 = blk; HEX2 = blk;HEX1 =
      one; HEX0 = zero; end
63              11: begin HEX5 = blk; HEX4 = blk; HEX3 = blk; HEX2 = blk;HEX1 =
      one; HEX0 = one; end
64              12: begin HEX5 = blk; HEX4 = blk; HEX3 = blk; HEX2 = blk;HEX1 =
      one; HEX0 = two; end
65              13: begin HEX5 = blk; HEX4 = blk; HEX3 = blk; HEX2 = blk;HEX1 =
      one; HEX0 = three; end
66              14: begin HEX5 = blk; HEX4 = blk; HEX3 = blk; HEX2 = blk;HEX1 =
      one; HEX0 = four; end
67              15: begin HEX5 = blk; HEX4 = blk; HEX3 = blk; HEX2 = blk;HEX1 =
      one; HEX0 = five; end
68              16: begin HEX5 = blk; HEX4 = blk; HEX3 = blk; HEX2 = blk;HEX1 =
      one; HEX0 = six; end
69              17: begin HEX5 = blk; HEX4 = blk; HEX3 = blk; HEX2 = blk;HEX1 =
      one; HEX0 = seven; end
70              18: begin HEX5 = blk; HEX4 = blk; HEX3 = blk; HEX2 = blk;HEX1 =
      one; HEX0 = eight; end
71              19: begin HEX5 = blk; HEX4 = blk; HEX3 = blk; HEX2 = blk;HEX1 =
      one; HEX0 = nine; end
72              20: begin HEX5 = blk; HEX4 = blk; HEX3 = blk; HEX2 = blk;HEX1 =
      two; HEX0 = zero; end
73              21: begin HEX5 = blk; HEX4 = blk; HEX3 = blk; HEX2 = blk;HEX1 =
      two; HEX0 = one; end
74              22: begin HEX5 = blk; HEX4 = blk; HEX3 = blk; HEX2 = blk;HEX1 =
      two; HEX0 = two; end
75              23: begin HEX5 = blk; HEX4 = blk; HEX3 = blk; HEX2 = blk;HEX1 =
      two; HEX0 = three; end
76              24: begin HEX5 = blk; HEX4 = blk; HEX3 = blk; HEX2 = blk;HEX1 =
      two; HEX0 = four; end
77              25: begin HEX5 = F; HEX4 = U; HEX3 = L; HEX2 = L;HEX1 = two;
     HEX0 = five; end
78            endcase
79        end
80
81    endmodule
82
83    // counter_testbench tests all expected behavior that the parking lot
```

```
    occupancy counter system in the lab may encounter
84  module hexDisplay_testbench();
85     logic clk;
86     logic [4:0] inputCount;
87     logic [6:0] HEX5, HEX4, HEX3, HEX2, HEX1, HEX0;
88
89     hexDisplay dut (.clk(clk), .inputCount(inputCount), .HEX5(HEX5), .
    HEX4(HEX4), .HEX3(HEX3), .HEX2(HEX2), .HEX1(HEX1), .HEX0(HEX0));
90
91     parameter CLOCK_PERIOD = 100;
92
93     initial begin
94        clk <= 0;
95        forever #(CLOCK_PERIOD/2) clk <= ~clk;
96     end
97
98     initial begin
99        inputCount <= 0;  @(posedge clk);
100       inputCount <= 10; @(posedge clk);
101       inputCount <= 20; @(posedge clk);
102       inputCount <= 25; @(posedge clk);
103       $stop;
104    end
105 endmodule
106
```

```
1   // Alan Li
2   // 01/15/2022
3   // EE 371
4   // Lab #1
5
6   // parkingLotSensors takes in 1-bit A, B and clk as inputs and return
    1-bit enter and exit as outputs.
7
8   module parkingLotSensors(clk, reset, A, B, enter, exit);
9      input logic A, B, clk, reset;
10     output logic enter, exit;
11
12     // States
13     enum {unblocked, beginIn, blockIn, almostIn, beginOut, blockOut,
    almostOut} ps, ns;
14
15     // The parking lot sensor can have six cases as list below. beginIn,
    blockin and almostIn are pairs with almostOut, blockOut and beginOut.
16     // Assume the entrance is north-south direction. If car is entering
    from North, it will trigger sensor A first.
17     // If car is exiting from South it will trigger sensor B first.
18     // For example, beginIn and almost out both indicate that only
    sensorA is blocked, they only differences is that beginIn means the car
    is going in the south direction and almostOut is car going north.
19     // This "case pair" is designed to tackle car changing direction
    issue.
20     always_comb
21        begin
22           enter = 0;
23           exit = 0;
24
25           case(ps)
26
27              unblocked:
28                 if (A & ~B)          ns = beginIn;      // car begin to
    enter
29                 else if (~A & B)     ns = beginOut;     // car begin to
    exit
30                 // else if (~A & ~B)    ns = unblocked; // impossible
31                 else                 ns = unblocked;    // nothing happens
32
33              beginIn:
34                 if (A & B)           ns = blockIn;      // car is
    halfway entering
35                 else if (~A & ~B)    ns = unblocked;    // car just
    enters then back up
36                 // else if (A & ~B)     ns = beginIn;    // impossible
37                 else                 ns = beginIn;      // car does not
    move
38
39              blockIn:
40                 if (~A & B)          ns = almostIn;     // car almost
    enters (trigger both sensors)
41                 else if (A & ~B)     ns = beginIn;      // car backs up
42                 // else if (A & B)      ns = blockIn;    // impossible
43                 else                 ns = blockIn;      // car does not
    move
44
```

```
45                almostIn:
46                   if (~A & ~B)
47                      begin
48                                              ns = unblocked;    // car enters
49                                              enter = 1;
50                                              exit = 0;
51                      end
52                   else if (A % B)            ns = blockIn;      // car backs up
53                   // else if (~A & B)           ns = almostIn;  // impossible
54                   else                       ns = almostIn;     // car does not
      move
55
56                beginOut:
57                   if (A & B)                 ns = blockOut;     // car is
      halfway exiting
58                   else if (~A & ~B)          ns = unblocked;    // car backs up
59                   // else if (~A & B)           ns = beginOut;  // impossible
60                   else                       ns = beginOut;     // car does not
      move
61
62                blockOut:
63                   if (A & ~B)                ns = almostOut;    // car almost
      exits (trigger both sensors)
64                   else if (~A & B)           ns = beginOut;     // car backs up
65                   // else if (A & B)            ns = blockOut;  // impossible
66                   else                       ns = blockOut;     // car does not
      move
67
68                almostOut:
69                   if (~A & ~B)                                  // car exits
70                      begin
71                         ns = unblocked;
72                         enter = 0;
73                         exit = 1;
74                      end
75                   else if (A & B)            ns = blockOut;     // car backs up
76                   // else if (A & ~B)           ns = almostOut; // impossible
77                   else                       ns = almostOut;    // car does not
      move
78
79                /*
80                default:
81                   begin
82                      enter = 0;
83                      exit = 0;
84                   end
85                */
86             endcase
87          end
88
89
90       // if reset, the parking lot sensor goes to unblocked case
91       always_ff @(posedge clk)
92          begin
93             if (reset)
94                ps <= unblocked;
95             else ps <= ns;
96          end
```

```
 97  endmodule
 98
 99  // parkingLotSensor_testbench tests all expected behavior that the
     parking lot occupancy counter system in the lab may encounter
100  module parkingLotSensor_testbench();
101     logic clk, reset, A, B, enter, exit;
102
103     parkingLotSensors dut (.clk(clk), .reset(reset), .A(A), .B(B), .enter
     (enter), .exit(exit));
104
105     parameter CLOCK_PERIOD = 100;
106
107     initial begin
108        clk <= 0;
109        forever #(CLOCK_PERIOD/2) clk <= ~clk;
110     end
111
112     // I have already simulate the case where car enters and exits
     without changing directions in DE1_SoC
113     // For this testbench I will simulate the case where the car changes
     direction multiple times
114     initial begin
115        reset <= 1;          @(posedge clk);
116        reset <= 0;          @(posedge clk);
117        {A, B} <= 2'b00;     @(posedge clk);
118        {A, B} <= 2'b10;     @(posedge clk);
119        {A, B} <= 2'b11;     @(posedge clk);
120        {A, B} <= 2'b01;     @(posedge clk);
121        {A, B} <= 2'b11;     @(posedge clk);
122        {A, B} <= 2'b10;     @(posedge clk);
123        {A, B} <= 2'b11;     @(posedge clk);
124        {A, B} <= 2'b01;     @(posedge clk);
125        {A, B} <= 2'b00;     @(posedge clk); // simulation for car enters
126
127        {A, B} <= 2'b00;     @(posedge clk);
128        {A, B} <= 2'b01;     @(posedge clk);
129        {A, B} <= 2'b11;     @(posedge clk);
130        {A, B} <= 2'b10;     @(posedge clk);
131        {A, B} <= 2'b11;     @(posedge clk);
132        {A, B} <= 2'b01;     @(posedge clk);
133        {A, B} <= 2'b11;     @(posedge clk);
134        {A, B} <= 2'b10;     @(posedge clk);
135        {A, B} <= 2'b00;     @(posedge clk); // simulation for car exits
136        $stop;
137     end
138  endmodule
139
140
141
142
```

```
1   // Alan Li
2   // 01/15/2022
3   // EE 371
4   // Lab #1
5
6   // userInput take 1-bit D as user input and return Q as output
7   // This module utilize two DFF to reduce the possibility of
    metastability by adding latency
8
9   module userInput(clk, D, Q);
10     input clk, D;
11     output logic Q;
12     logic temp;
13
14     always_ff @(posedge clk) begin
15        temp <= D;
16        Q <= temp;
17     end
18
19  endmodule
```