

```
1  /*
2  Ben Davis
3  1/11/23
4  EE 371
5  Lab 2, Task 2
6
7  This module counts up to 15 for the 32x4 RAM.
8  It takes in three inputs, a reset, a plus one (inc), and
9  a minus one (dec). It returns the number of cars in the lot
10 as a 4 bit output (out).
11
12 */
13
14 module counter_31 (clk, rst, out);
15
16     input logic clk, rst; //clock, reset, increment, decrement
17     output logic [4:0] out; //number of cars inside lot
18
19     enum {s0, s1, s2, s3, s4, s5, s6, s7, s8 //states to count the # of cars
20         , s9, s10, s11, s12, s13, s14, s15
21         , s16, s17, s18, s19, s20, s21, s22
22         , s23, s24, s25, s26, s27, s28, s29
23         , s30, s31} ps, ns;
24
25     //always ff block to reset the state to zero
26     //or to move to the next state if need be
27     always_ff @(posedge clk) begin
28         if(rst) begin
29             ps <= s0;
30         end else begin
31             ps <= ns;
32         end
33     end
34
35
36     //the block to tell each state if it goes to the
37     //next or previous state
38     always_comb begin
39         case(ps)
40
41             s0: begin out <= 5'b00000; ns <= s1; end
42
43             s1: begin out <= 5'b00001; ns <= s2; end
44
45             s2: begin out <= 5'b00010; ns <= s3; end
46
47             s3: begin out <= 5'b00011; ns <= s4; end
48
49             s4: begin out <= 5'b00100; ns <= s5; end
50
51             s5: begin out <= 5'b00101; ns <= s6; end
52
53             s6: begin out <= 5'b00110; ns <= s7; end
54
55             s7: begin out <= 5'b00111; ns <= s8; end
56
57             s8: begin out <= 5'b01000; ns <= s9; end
58
59             s9: begin out <= 5'b01001; ns <= s10; end
60
61             s10: begin out <= 5'b01010; ns <= s11; end
62
63             s11: begin out <= 5'b01011; ns <= s12; end
64
65             s12: begin out <= 5'b01100; ns <= s13; end
66
67             s13: begin out <= 5'b01101; ns <= s14; end
68
69             s14: begin out <= 5'b01110; ns <= s15; end
70
71             s15: begin out <= 5'b01111; ns <= s16; end
72
73             s16: begin out <= 5'b10000; ns <= s17; end
```

```
74
75     s17: begin out <= 5'b10001; ns <= s18; end
76
77     s18: begin out <= 5'b10010; ns <= s19; end
78
79     s19: begin out <= 5'b10011; ns <= s20; end
80
81     s20: begin out <= 5'b10100; ns <= s21; end
82
83     s21: begin out <= 5'b10101; ns <= s22; end
84
85     s22: begin out <= 5'b10110; ns <= s23; end
86
87     s23: begin out <= 5'b10111; ns <= s24; end
88
89     s24: begin out <= 5'b11000; ns <= s25; end
90
91     s25: begin out <= 5'b11001; ns <= s26; end
92
93     s26: begin out <= 5'b11010; ns <= s27; end
94
95     s27: begin out <= 5'b11011; ns <= s28; end
96
97     s28: begin out <= 5'b11100; ns <= s29; end
98
99     s29: begin out <= 5'b11101; ns <= s30; end
100
101     s30: begin out <= 5'b11110; ns <= s31; end
102
103     s31: begin out <= 5'b11111; ns <= s0; end
104
105     endcase
106   end
107 endmodule
108
109 module counter_31_testbench();
110
111     logic clk, rst; //repeating logic variables
112     logic [4:0] out;
113
114     //test counter_25 module
115     counter_31 dut (.clk, .rst, .out);
116
117     // clock setup
118     parameter clock_period = 100;
119
120     initial begin
121         clk <= 0;
122         forever #(clock_period / 2) clk = ~clk;
123     end // of clock setup
124
125     //an instance where it counts up to five with a
126     //space after the first increment, and then
127     //decrements twice
128     initial begin
129         rst <= 0; @(posedge clk);
130         rst <= 0; @(posedge clk);
131         rst <= 0; @(posedge clk);
132         rst <= 0; @(posedge clk);
133         rst <= 0; @(posedge clk);
134         rst <= 0; @(posedge clk);
135         rst <= 0; @(posedge clk);
136         rst <= 0; @(posedge clk);
137         rst <= 0; @(posedge clk);
138         $stop;
139     end
140 endmodule
```