```
1   `timescale 1ns / 1ps
2   //////////////////////////////////////////////////////////////////
3   //  Class: CECS 360                                             //
4   //  Project name: Project1_CECS360                              //
5   //  File name: Debounce.v                                       //
6   //                                                              //
7   //  Created by Umar Khan on 09/16/2017                          //
8   //                                                              //
9   //                                                              //
10  //  Abstract: Stabilizes the output instead of showing too many //
11  //  transitions.                                                //
12  //  Reference to Pong Chu's Debounce                            //
13  //                                                              //
14  //////////////////////////////////////////////////////////////////
15  module Debounce(clk, reset, sw, db);
16
17  input wire clk, reset;
18  input wire sw;
19  output reg db;
20
21  // symbolic state declaration
22  pulse_maker pulse_maker(clk,reset, pulse);
23
24  //Pulse_Maker     P1 (clk, reset, pulse);
25     localparam [2:0]
26              zero    = 3'b000,
27              wait1_1 = 3'b001,
28              wait1_2 = 3'b010,
29              wait1_3 = 3'b011,
30              one     = 3'b100,
31              wait0_1 = 3'b101,
32              wait0_2 = 3'b110,
33              wait0_3 = 3'b111;
34
35  // number of counter bits (2"N * 2Ons = lOms tick)
36
37   localparam N = 20;
38
39  // signal declaration
40
41        reg [N-1:0] q_reg;
42        wire [N-1 : 0] q_next ;
43        wire pulse;
44        reg [2:0] state_reg , state_next ;
45
46  // body
47  // counter to generate 10 ms tick
48
49     always @ (posedge clk, posedge reset)
50        if (reset)
51           q_reg <= 0;
52        else
53           q_reg <= q_next;
54
55  // next-state logic
56
57     assign q_next = q_reg + 1;
```

```
58
59    // output tick
60
61       assign pulse = (q_reg==0) ? 1'b1 : 1'b0;
62
63    // debouncing FSM
64    // state register
65
66       always @ ( posedge clk , posedge reset)
67          if (reset)
68             state_reg <= zero;
69          else
70             state_reg <= state_next ;
71
72       // next-state logic and output logic
73
74       always @*
75          begin
76             state_next = state_reg; // default state: the same
77             db = 1'b0; // default output: 0
78       case (state_reg)
79
80    zero :
81          if (sw)
82             state_next = wait1_1 ;
83             wait1_1 :
84          if (~sw)
85             state_next = zero;
86          else
87
88          if (pulse)
89             state_next = wait1_2 ;
90    wait1_2 :
91          if (~sw)
92             state_next = zero;
93          else
94          if (pulse)
95             state_next = wait1_3;
96    wait1_3 :
97          if (~sw)
98             state_next = zero;
99          else
100         if (pulse)
101            state_next = one;
102   one :
103         begin
104            db = 1'b1;
105         if (~sw)
106            state_next = wait0_1;
107         end
108   wait0_1 :
109         begin
110            db = 1'b1;
111         if (sw)
112            state_next = one;
113         else
114         if (pulse)
```

```
115             state_next = wait0_2;
116          end
117    wait0_2 :
118          begin
119             db = 1'b1;
120          if (sw)
121             state_next = one;
122          else
123          if (pulse)
124             state_next = wait0_3;
125          end
126    wait0_3 :
127          begin
128             db = 1'b1;
129          if (sw)
130             state_next = one;
131          else
132          if (pulse)
133             state_next = zero;
134          end
135
136          default : state_next = zero;
137
138          endcase
139
140       end
141
142    endmodule
```