```
1   `timescale 1ns / 1ps
2   /**********************************************************************
3    * File Name: displayController.v
4    * Project: Counter using AISO
5    * Designer: Marc Cabote
6    * Email: marcdominic011@gmail.com
7    * Rev. Date: 20 September, 2017
8    *
9    * Purpose: The Debounce module is designed to stabilize the rapid succession
10   *           of binary input signals before transmitting a single, stable
11   *          output signal when a button is pressed or a switch is flipped.
12   *          In this project, when the clock is active and the down button
13   *          is pressed, the state machine waits for the input to
14   *          be stable before it outputs the confirmed stable signal.
15   *
16   *
17   * Notes:   -  This module is driven by a finite state machine.
18   *          -  The codes below are directly(with slight modifcations)
19   *             from Pong Chu's Book Verilog By Examples 3rd Edition.
20   *             It can be found on pages 131-133.
21   *
22   **********************************************************************/
23  module debounce(input  clk, rst, sw,
24                  output reg   db);
25
26    //symbolic state declaration
27    localparam [2:0]
28                zero   = 3'b000,
29                zero_1 = 3'b001,
30                zero_2 = 3'b010,
31                zero_3 = 3'b011,
32                one    = 3'b100,
33                one_1  = 3'b101,
34                one_2  = 3'b110,
35                one_3  = 3'b111;
36
37    // localparam N = 20; // 100 Mhz clock
38
39    //signal declaration
40    //output tick;
41    wire tick;
42    reg [19:0] count;
43    reg  [2:0] state_reg, nextState;
44
45    //body
46
47    //Modified from Pong Chu to generate 10 ms tick
48    //================================
49    // counter to generate 10 ms tick
50    //================================
51    assign tick = (count == 999999);
52     //assign tick = clk;
53    always @ (posedge clk, posedge rst)
54       if (rst)  count <= 20'b0; else
55       if (tick) count <= 20'b0; else //check if tick is 1
56                 count <= count + 20'b1; //increment count
57
```

```
58      //================================
59      // debouncing FSM
60      //================================
61      //state register
62      always @ (posedge clk, posedge rst)
63         if (rst) state_reg <= zero; else
64                  state_reg <= nextState;
65
66      //ns and output logic
67
68      always @ (*) begin
69         nextState = state_reg; //default state: the same
70         db = 1'b0;             //default output = 0
71         case (state_reg)
72            zero:
73               if (sw) nextState = one_1;
74            one_1:
75               if (~sw)  nextState = zero; else
76               if (tick) nextState = one_2;
77            one_2:
78               if (~sw)  nextState = zero; else
79               if (tick) nextState = one_3;
80            one_3:
81               if (~sw)  nextState = zero; else
82               if (tick) nextState = one;
83            one: begin
84               db = 1'b1;
85               if (~sw)  nextState = zero_1; end
86            zero_1: begin
87               db = 1'b1;
88               if (sw)   nextState = one; else
89               if (tick) nextState = zero_2; end
90            zero_2: begin
91               db = 1'b1;
92               if (sw)   nextState = one; else
93               if (tick) nextState = zero_3; end
94            zero_3: begin
95               db = 1'b1;
96               if (sw)   nextState = one; else
97               if (tick) nextState = zero; end
98            default :    nextState = zero;
99         endcase
100     end
101
102
103
104  endmodule
105
106
```