Lab Code [15 points]
Filename: 01_struc.sv
AndrewID: xinyew

```
  1 `default_nettype none
  2
  3 module dFlipFlop(
  4   output logic q,
  5   input  logic d, clock, reset);
  6
  7   always_ff @(posedge clock)
  8     if (reset == 1'b1)
  9       q <= 0;
 10     else
 11       q <= d;
 12
 13 endmodule : dFlipFlop
 14
 15 //  ------------------------------------------------------
 16 // |         |  Q2  Q1  Q0  |  desciption
 17 // |  -------|--------------|-----------------------------------
 18 // | State1  |   0   0   0  |  computer #5:
 19 // | State2  |   0   0   1  |  computer #1, #5;
 20 // | State3  |   0   1   0  |  computer #1, #5, #9;     win
 21 // | State4  |   0   1   1  |  computer #1, #3, #5;     win
 22 // | State5  |   1   0   0  |  computer #1, #2, #3, #7; win
 23 // | State6  |   1   0   1  |  computer #1, #3, #5, #7; win
 24 //  ------------------------------------------------------
 25
 26 module myExplicitFSM(
 27   output logic [3:0] cMove,
 28   output logic       win,
 29   output logic       q0, q1, q2,
 30   input  logic [3:0] hMove,
 31   input  logic       clock, reset);
 32
 33   logic d0, d1, d2;
 34
 35   // flip-flops instantiation
 36   dFlipFlop ff0(.d(d0),
 37                 .q(q0),
 38                 .clock(clock),
 39                 .reset(reset)),
 40             ff1(.d(d1),
 41                 .q(q1),
 42                 .clock(clock),
 43                 .reset(reset)),
 44             ff2(.d(d2),
 45                 .q(q2),
 46                 .clock(clock),
 47                 .reset(reset));
 48
 49   // next state generation
 50   assign d2 = ((~q2) & q1 & q0 & (~hMove[3]) & (~hMove[2]) & hMove[1] & (~hMo...
Line length of 87 (max is 80)
 51               ((~q2) & q1 & q0 & (~hMove[3]) & (~hMove[2]) & hMove[1] & (~hMo...
Line length of 86 (max is 80)
 52               ((~q2) & q1 & q0 & (~hMove[3]) & hMove[2] & hMove[1] & hMove[0]) |
 53               ((~q2) & q1 & q0 & hMove[3] & (~hMove[2]) & (~hMove[1]) & (~hMo...
Line length of 86 (max is 80)
 54
 55               (q2 & (~q1) & (~q0) & (~hMove[3]) & (~hMove[2]) & (~hMove[1]) &...
Line length of 89 (max is 80)
 56               (q2 & (~q1) & (~q0) & (~hMove[3]) & (~hMove[2]) & hMove[1] & (~...
Line length of 89 (max is 80)
 57               (q2 & (~q1) & (~q0) & (~hMove[3]) & (~hMove[2]) & hMove[1] & hM...
Line length of 86 (max is 80)
 58               (q2 & (~q1) & (~q0) & (~hMove[3]) & hMove[2] & (~hMove[1]) & (~...
Line length of 89 (max is 80)
 59               (q2 & (~q1) & (~q0) & (~hMove[3]) & hMove[2] & (~hMove[1]) & hM...
Line length of 86 (max is 80)
 60               (q2 & (~q1) & (~q0) & (~hMove[3]) & hMove[2] & hMove[1] & (~hMo...
Line length of 86 (max is 80)
```

```
 61                (q2 & (~q1) & (~q0) & (~hMove[3]) & hMove[2] & hMove[1] & hMove...
Line length of 83 (max is 80)
 62                (q2 & (~q1) & (~q0) & hMove[3] & (~hMove[2]) & (~hMove[1]) & (~...
Line length of 89 (max is 80)
 63                (q2 & (~q1) & (~q0) & hMove[3] & (~hMove[2]) & (~hMove[1]) & hM...
Line length of 86 (max is 80)
 64
 65                (q2 & (~q1) & q0 & (~hMove[3]) & (~hMove[2]) & (~hMove[1]) & hM...
Line length of 86 (max is 80)
 66                (q2 & (~q1) & q0 & (~hMove[3]) & (~hMove[2]) & hMove[1] & (~hMo...
Line length of 86 (max is 80)
 67                (q2 & (~q1) & q0 & (~hMove[3]) & (~hMove[2]) & hMove[1] & hMove...
Line length of 83 (max is 80)
 68                (q2 & (~q1) & q0 & (~hMove[3]) & hMove[2] & (~hMove[1]) & (~hMo...
Line length of 86 (max is 80)
 69                (q2 & (~q1) & q0 & (~hMove[3]) & hMove[2] & (~hMove[1]) & hMove...
Line length of 83 (max is 80)
 70                (q2 & (~q1) & q0 & (~hMove[3]) & hMove[2] & hMove[1] & (~hMove[...
Line length of 83 (max is 80)
 71                (q2 & (~q1) & q0 & (~hMove[3]) & hMove[2] & hMove[1] & hMove[0]) |
 72                (q2 & (~q1) & q0 & hMove[3] & (~hMove[2]) & (~hMove[1]) & (~hMo...
Line length of 86 (max is 80)
 73                (q2 & (~q1) & q0 & hMove[3] & (~hMove[2]) & (~hMove[1]) & hMove...
Line length of 82 (max is 80)
 74
 75   assign d1 = ((~q2) & (~q1) & q0 & (~hMove[3]) & (~hMove[2]) & hMove[1] & (~...
Line length of 89 (max is 80)
 76                ((~q2) & (~q1) & q0 & (~hMove[3]) & (~hMove[2]) & hMove[1] & hM...
Line length of 86 (max is 80)
 77                ((~q2) & (~q1) & q0 & (~hMove[3]) & hMove[2] & (~hMove[1]) & (~...
Line length of 89 (max is 80)
 78                ((~q2) & (~q1) & q0 & (~hMove[3]) & hMove[2] & hMove[1] & hMove...
Line length of 83 (max is 80)
 79                ((~q2) & (~q1) & q0 & hMove[3] & (~hMove[2]) & (~hMove[1]) & (~...
Line length of 89 (max is 80)
 80                ((~q2) & (~q1) & q0 & hMove[3] & (~hMove[2]) & (~hMove[1]) & hM...
Line length of 86 (max is 80)
 81
 82                ((~q2) & q1 & (~q0) & (~hMove[3]) & (~hMove[2]) & (~hMove[1]) &...
Line length of 89 (max is 80)
 83                ((~q2) & q1 & (~q0) & (~hMove[3]) & (~hMove[2]) & hMove[1] & (~...
Line length of 89 (max is 80)
 84                ((~q2) & q1 & (~q0) & (~hMove[3]) & (~hMove[2]) & hMove[1] & hM...
Line length of 86 (max is 80)
 85                ((~q2) & q1 & (~q0) & (~hMove[3]) & hMove[2] & (~hMove[1]) & (~...
Line length of 89 (max is 80)
 86                ((~q2) & q1 & (~q0) & (~hMove[3]) & hMove[2] & (~hMove[1]) & hM...
Line length of 86 (max is 80)
 87                ((~q2) & q1 & (~q0) & (~hMove[3]) & hMove[2] & hMove[1] & (~hMo...
Line length of 86 (max is 80)
 88                ((~q2) & q1 & (~q0) & (~hMove[3]) & hMove[2] & hMove[1] & hMove...
Line length of 83 (max is 80)
 89                ((~q2) & q1 & (~q0) & q0 & hMove[3] & (~hMove[2]) & (~hMove[1])...
Line length of 94 (max is 80)
 90                ((~q2) & q1 & (~q0) & hMove[3] & (~hMove[2]) & (~hMove[1]) & hM...
Line length of 86 (max is 80)
 91
 92                ((~q2) & q1 & q0 & (~hMove[3]) & (~hMove[2]) & (~hMove[1]) & hM...
Line length of 86 (max is 80)
 93                ((~q2) & q1 & q0 & (~hMove[3]) & (~hMove[2]) & hMove[1] & hMove...
Line length of 83 (max is 80)
 94                ((~q2) & q1 & q0 & (~hMove[3]) & hMove[2] & (~hMove[1]) & hMove...
Line length of 83 (max is 80)
 95                ((~q2) & q1 & q0 & (~hMove[3]) & hMove[2] & hMove[1] & (~hMove[...
Line length of 83 (max is 80)
 96                ((~q2) & q1 & q0 & hMove[3] & (~hMove[2]) & (~hMove[1]) & hMove...
Line length of 82 (max is 80)
 97
 98   assign d0 = ((~q2) & (~q1) & (~q0) & (~hMove[3]) & hMove[2] & hMove[1] & (~...
Line length of 89 (max is 80)
 99
```

```
100                ((~q2) & (~q1) & q0 & (~hMove[3]) & (~hMove[2]) & (~hMove[1]) &...
Line length of 89 (max is 80)
101                ((~q2) & (~q1) & q0 & (~hMove[3]) & hMove[2] & (~hMove[1]) & hM...
Line length of 86 (max is 80)
102                ((~q2) & (~q1) & q0 & (~hMove[3]) & (~hMove[2]) & (~hMove[1]) &...
Line length of 89 (max is 80)
103
104
105                ((~q2) & q1 & q0 & (~hMove[3]) & (~hMove[2]) & (~hMove[1]) & hM...
Line length of 86 (max is 80)
106                ((~q2) & q1 & q0 & (~hMove[3]) & (~hMove[2]) & hMove[1] & (~hMo...
Line length of 86 (max is 80)
107                ((~q2) & q1 & q0 & (~hMove[3]) & (~hMove[2]) & hMove[1] & hMove...
Line length of 83 (max is 80)
108                ((~q2) & q1 & q0 & (~hMove[3]) & hMove[2] & (~hMove[1]) & hMove...
Line length of 83 (max is 80)
109                ((~q2) & q1 & q0 & (~hMove[3]) & hMove[2] & hMove[1] & (~hMove[...
Line length of 83 (max is 80)
110                ((~q2) & q1 & q0 & hMove[3] & (~hMove[2]) & (~hMove[1]) & hMove...
Line length of 83 (max is 80)
111
112                (q2 & (~q1) & q0 & (~hMove[3]) & (~hMove[2]) & (~hMove[1]) & hM...
Line length of 86 (max is 80)
113                (q2 & (~q1) & q0 & (~hMove[3]) & (~hMove[2]) & hMove[1] & (~hMo...
Line length of 86 (max is 80)
114                (q2 & (~q1) & q0 & (~hMove[3]) & (~hMove[2]) & hMove[1] & hMove...
Line length of 83 (max is 80)
115                (q2 & (~q1) & q0 & (~hMove[3]) & hMove[2] & (~hMove[1]) & (~hMo...
Line length of 86 (max is 80)
116                (q2 & (~q1) & q0 & (~hMove[3]) & hMove[2] & (~hMove[1]) & hMove...
Line length of 83 (max is 80)
117                (q2 & (~q1) & q0 & (~hMove[3]) & hMove[2] & hMove[1] & (~hMove[...
Line length of 83 (max is 80)
118                (q2 & (~q1) & q0 & (~hMove[3]) & hMove[2] & hMove[1] & hMove[0]) |
119                (q2 & (~q1) & q0 & q0 & hMove[3] & (~hMove[2]) & (~hMove[1]) & ...
Line length of 91 (max is 80)
120                (q2 & (~q1) & q0 & hMove[3] & (~hMove[2]) & (~hMove[1]) & hMove...
Line length of 82 (max is 80)
121
122    // output logic generation
123    assign cMove[3] = (~q2) & q1 & (~q0);
124
125    assign cMove[2] = ((~q2) & (~q1) & (~q0)) |
126                      (q2 & (~q1) & q0);
127
128    assign cMove[1] = ((~q2) & q1 & q0) |
129                      (q2 & (~q1) & (~q0)) |
130                      (q2 & (~q1) & q0);
131
132    assign cMove[0] = ((~q2) & (~q1) & (~q0)) |
133                      ((~q2) & (~q1) & q0) |
134                      ((~q2) & q1 & (~q0)) |
135                      ((~q2) & q1 & q0) |
136                      (q2 & (~q1) & q0);
137
138    assign win = ((~q2) & q1 & (~q0)) |
139                 (q2 & (~q1) & (~q0)) |
140                 (q2 & (~q1) & q0);
141
142 endmodule : myExplicitFSM
143
```

Lab Code [15 points]
Filename: 02_abstract.sv
AndrewID: xinyew

```
  1 `default_nettype none
  2
  3 module myAbstractFSM (
  4   output logic [3:0] cMove,
  5   output logic       win,
  6   output logic       q2, q1, q0,
  7   input  logic [3:0] hMove,
  8   input  logic       clock, reset);
  9
 10   enum logic [2:0] {S0 = 3'b000, S1 = 3'b001,
 11                     S2 = 3'b010, S3 = 3'b011,
 12                     S4 = 3'b100, S5 = 3'b101} currState, nextState;
 13
 14   // next state generation
 15   always_comb
 16     unique case (currState)
 17       S0:
 18         nextState = (hMove == 4'h6) ? S1 : S0;
 19       S1: begin
 20       if (hMove == 4'h1 || hMove == 4'h5 || hMove == 4'h6)
Line contains tabs (each tab replaced by 2 spaces in this print)
 21         nextState = S1;
Line contains tabs (each tab replaced by 2 spaces in this print)
 22       else
Line contains tabs (each tab replaced by 2 spaces in this print)
 23           nextState = (hMove == 4'h9) ? S3 : S2;
 24       end
Line contains tabs (each tab replaced by 2 spaces in this print)
 25       S2:
 26         nextState = S2;
 27       S3: begin
 28       if (hMove == 4'h1 || hMove == 4'h4 || hMove == 4'h5 || hMove == 4'h6 || h...
Line length of 90 (max is 80)
Line contains tabs (each tab replaced by 2 spaces in this print)
 29         nextState = S3;
Line contains tabs (each tab replaced by 2 spaces in this print)
 30       else
Line contains tabs (each tab replaced by 2 spaces in this print)
 31           nextState = (hMove == 4'h2) ? S5 : S4;
 32       end
 33       S4:
 34         nextState = S4;
 35       S5:
 36         nextState = S5;
 37     endcase
 38
 39
 40   // output generation
 41   always_comb begin
 42    cMove = 4'b0000;
Line contains tabs (each tab replaced by 2 spaces in this print)
 43    win = 1'b1;
Line contains tabs (each tab replaced by 2 spaces in this print)
 44     if (currState == S0) begin
 45       cMove = 4'h5;
 46       win = 0;
 47     end
 48     if (currState == S1) begin
 49       cMove = 4'h1;
 50       win = 0;
 51     end
 52     if (currState == S2) begin
 53       cMove = 4'h9;
 54       win = 1;
 55     end
 56     if (currState == S3) begin
 57       cMove = 4'h3;
 58       win = 0;
 59     end
```

```
60      if (currState == S4) begin
61        cMove = 4'h2;
62        win = 1;
63      end
64      if (currState == S5) begin
65        cMove = 4'h7;
66        win = 1;
67      end
68    end
69
70    // register
71    always_ff @(posedge clock)
72      if (reset)
73        currState <= S0;
74      else
75        currState <= nextState;
76
77  endmodule: myAbstractFSM
78
```

Lab Code [15 points]
Filename: HexDisplay.sv
AndrewID: xinyew

```
1  `default_nettype none
2
3  module SevenSegmentDisplay
4    (input  logic [3:0] BCX0,
5     input  logic [7:0] blank,
6     output logic [6:0] HEX0);
7
8    always_comb begin
9      HEX0 = 7'b0000000;
10     if (~blank[0])
11       case (BCX0)
12         4'h0: HEX0 = 7'b0111111;
13         4'h1: HEX0 = 7'b0000110;
14         4'h2: HEX0 = 7'b1011011;
15         4'h3: HEX0 = 7'b1001111;
16         4'h4: HEX0 = 7'b1100110;
17         4'h5: HEX0 = 7'b1101101;
18         4'h6: HEX0 = 7'b1111101;
19         4'h7: HEX0 = 7'b0000111;
20         4'h8: HEX0 = 7'b1111111;
21         4'h9: HEX0 = 7'b1100111;
22         4'ha: HEX0 = 7'b1110111;
23         4'hb: HEX0 = 7'b1111100;
24         4'hc: HEX0 = 7'b0111001;
25         4'hd: HEX0 = 7'b1011110;
26         4'he: HEX0 = 7'b1111001;
27         4'hf: HEX0 = 7'b1110001;
28         default: HEX0 = 7'b0000000;
29       endcase
30     HEX0 = ~HEX0;
31   end
32
33
34 endmodule : SevenSegmentDisplay
```

Lab Code [15 points]
Filename: chipInterface.sv
AndrewID: xinyew

```
  1 `default_nettype none // Required in every sv file
  2 module chipInterface
  3   (input logic [3:0] KEY,
  4   input logic [17:0] SW,
  5   output logic [6:0] HEX0,
  6   output logic [7:0] LEDG);
  7   logic [3:0] c;
  8
  9   logic ww;
 10   myAbstractFSM(.clock(KEY[0]), .reset(SW[17]), .hMove(SW[9:6]), .cMove(c), ....
Line length of 86 (max is 80)
 11   assign LEDG = {ww, ww, ww, ww, ww, ww, ww, ww};
 12   logic [7:0] blank;
 13   assign blank = 8'b00000000;
 14
 15   SevenSegmentDisplay DUT2 (.BCX0(c),
 16                             .blank(blank),
 17                             .HEX0(HEX0));
 18
 19 endmodule: chipInterface
 20
 21
 22
```

Lab Code [15 points]
Filename: tb.sv
AndrewID: xinyew

```
`default_nettype none

module testBench();
    logic w1, w2, w3, w4, w5, w6;
    logic [3:0] w7, w8;
    myExplicitFSM dut1(.clock(w1),
                       .reset(w2),
                       .q1(w3),
                       .q2(w4),
                       .q0(w5),
                       .win(w6),
                       .cMove(w7),
                       .hMove(w8));
    myFSM_test dut2(.clock(w1),
                       .reset(w2),
                       .q1(w3),
                       .q2(w4),
                       .q0(w5),
                       .win(w6),
                       .cMove(w7),
                       .hMove(w8));

endmodule : testBench


module myFSM_test(
    input logic [3:0] cMove,
    input logic win,
    input logic q2, q1, q0,
    output logic [3:0] hMove,
    output logic clock, reset);



    initial begin
        clock = 0;
        forever #5 clock = ~clock;
    end

    initial begin
        $monitor($time,, "state=%b, cMove=%d, hMove=%d, win=%b",
                 {q2, q1, q0}, cMove, hMove, win);
        // initialize values
        hMove <= 4'hF;
        reset <= 1'b1;

        // reset the FSM
        @(posedge clock); // wait for a positive clock edge
        @(posedge clock); // one edge is enough, but what the heck
        @(posedge clock);

        @(posedge clock); // begin cycle 0
        reset <= 1'b0; // release the reset


        // start an example sequence -- not meaningful for the lab
        hMove <= 4'h6; // these changes are after the clock edge
                       // which means the state change happens
                       // AFTER the next clock edge
        @(posedge clock); // begin cycle 1
        hMove <= 4'h1;


        // reset the FSM
        @(posedge clock);
        @(posedge clock);
        @(posedge clock);

        reset <= 1'b1;
```

```
 70          @(posedge clock);
 71          reset <= 1'b0;
 72          //start
 73          hMove <= 4'h6;
 74          @(posedge clock); // 2-7 TEST
 75          hMove <= 4'h9;
 76          @(posedge clock);
 77          hMove <= 4'h2;
 78
 79
 80          // reset the FSM
 81          @(posedge clock);
 82          @(posedge clock);
 83          @(posedge clock);
 84
 85          reset <= 1'b1;
 86          @(posedge clock);
 87          reset <= 1'b0;
 88          //start
 89          hMove <= 4'h6;
 90          @(posedge clock); // 7-2 TEST
 91          hMove <= 4'h9;
 92          @(posedge clock);
 93          hMove <= 4'h7;
 94
 95
 96          // reset the FSM
 97          @(posedge clock);
 98          @(posedge clock);
 99          @(posedge clock);
100
101          reset <= 1'b1;
102          @(posedge clock);
103          reset <= 1'b0;
104          //start
105          hMove <= 4'h6;
106          @(posedge clock); // not 9 test
107          hMove <= 4'h5;
108
109
110          // reset the FSM
111          @(posedge clock);
112          @(posedge clock);
113          @(posedge clock);
114
115          reset <= 1'b1;
116          @(posedge clock);
117          reset <= 1'b0;
118          //start
119          hMove <= 4'h6;
120          @(posedge clock); // not 7-2 test
121          hMove <= 4'h9;
122          @(posedge clock);
123          hMove <= 4'h4;
124
125
126          // reset the FSM
127          @(posedge clock);
128          @(posedge clock);
129          @(posedge clock);
130
131          reset <= 1'b1;
132          @(posedge clock);
133          reset <= 1'b0;
134          //start
135          hMove <= 4'h4;  // not 6 test
136          @(posedge clock); // not
137
138          // reset the FSM
139          @(posedge clock);
140          @(posedge clock);
```

```
141         @(posedge clock);
142         reset <= 1'b1;
143         @(posedge clock);
144         reset <= 1'b0;
145         //start
146         hMove <= 4'h4;  // not 6 test
147         @(posedge clock); // not
148
149
150
151         // reset the FSM
152         @(posedge clock);
153         @(posedge clock);
154         @(posedge clock);
155         reset <= 1'b1;
156         @(posedge clock);
157         reset <= 1'b0;
158         //start
159         hMove <= 4'h6;
160         @(posedge clock); // not 7-2 test
161         hMove <= 4'h9;
162         @(posedge clock);
163         hMove <= 4'h4;
164
165
166         // reset the FSM
167         @(posedge clock);
168         @(posedge clock);
169         @(posedge clock);
170         reset <= 1'b1;
171         @(posedge clock);
172         reset <= 1'b0;
173         //start
174         hMove <= 4'h6;
175         @(posedge clock); // not 7-2 test
176         hMove <= 4'h9;
177         @(posedge clock);
178         hMove <= 4'h4;
179
180
181         // reset the FSM
182         @(posedge clock);
183         @(posedge clock);
184         @(posedge clock);
185         reset <= 1'b1;
186         @(posedge clock);
187         reset <= 1'b0;
188         //start
189         hMove <= 4'h2;
190         @(posedge clock); // not 7-2 test
191         hMove <= 4'h3;
192         @(posedge clock);
193         hMove <= 4'h9;
194
195
196         // reset the FSM
197         @(posedge clock);
198         @(posedge clock);
199         @(posedge clock);
200         reset <= 1'b1;
201         @(posedge clock);
202         reset <= 1'b0;
203         //start
204         hMove <= 4'h6;
205         @(posedge clock); // not 7-2 test
206         hMove <= 4'h9;
207         @(posedge clock);
208         hMove <= 4'h4;
209
210         // reset the FSM
211         @(posedge clock);
```

```
212         @(posedge clock);
213         @(posedge clock);
214         reset <= 1'b1;
215         @(posedge clock);
216         reset <= 1'b0;
217         //start
218         hMove <= 4'h6;
219         @(posedge clock); // not 7-2 test
220         hMove <= 4'h9;
221         @(posedge clock);
222         hMove <= 4'h3;
223         @(posedge clock);
224         hMove <= 4'h4;
225
226                 // reset the FSM
227         @(posedge clock);
228         @(posedge clock);
229         @(posedge clock);
230         reset <= 1'b1;
231         @(posedge clock);
232         reset <= 1'b0;
233         //start
234         hMove <= 4'h6;
235         @(posedge clock); // not 7-2 test
236         hMove <= 4'h9;
237         @(posedge clock);
238         hMove <= 4'h3;
239         @(posedge clock);
240         hMove <= 4'h8;
241         @(posedge clock);
242
243         // reset the FSM
244         @(posedge clock);
245         @(posedge clock);
246         @(posedge clock);
247         reset <= 1'b1;
248         @(posedge clock);
249         reset <= 1'b0;
250         //start
251         hMove <= 4'h6;
252         @(posedge clock); // not 7-2 test
253         hMove <= 4'h9;
254         @(posedge clock);
255         hMove <= 4'h3;
256         @(posedge clock);
257         hMove <= 4'h7;
258         @(posedge clock);
259         hMove <= 4'h4;
260         @(posedge clock);
261
262
263         // reset the FSM
264         @(posedge clock);
265         @(posedge clock);
266         @(posedge clock);
267         reset <= 1'b1;
268         @(posedge clock);
269         reset <= 1'b0;
270         //start
271         hMove <= 4'h6;
272         @(posedge clock); // not 7-2 test
273         hMove <= 4'h9;
274         @(posedge clock);
275         hMove <= 4'h3;
276         @(posedge clock);
277         hMove <= 4'h2;
278         @(posedge clock);
279         hMove <= 4'h4;
280         @(posedge clock);
281
282         @(posedge clock);
```

```
283         @(posedge clock);
284         @(posedge clock);
285         reset <= 1'b1;
286         @(posedge clock);
287         reset <= 1'b0;
288
289
290
291
292         #1 $finish;
293     end
294 endmodule: myFSM_test
```