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: 01_testbench.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
```

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   input  logic [3:0] hMove,
  7   input  logic       clock, reset);
  8
  9   enum logic [2:0] {S0 = 3'b000, S1 = 3'b001,
 10                     S2 = 3'b010, S3 = 3'b011,
 11                     S4 = 3'b100, S5 = 3'b101} currState, nextState;
 12
 13   // next state generation
 14   always_comb
 15     unique case (currState)
 16       S0:
 17         nextState = (hMove == 4'h6) ? S1 : S0;
 18       S1: begin
 19       if (hMove == 4'h1 || hMove == 4'h5 || hMove == 4'h6)
Line contains tabs (each tab replaced by 2 spaces in this print)
 20         nextState = S1;
Line contains tabs (each tab replaced by 2 spaces in this print)
 21       else
Line contains tabs (each tab replaced by 2 spaces in this print)
 22           nextState = (hMove == 4'h9) ? S3 : S2;
 23       end
Line contains tabs (each tab replaced by 2 spaces in this print)
 24       S2:
 25         nextState = S2;
 26       S3: begin
 27       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)
 28         nextState = S3;
Line contains tabs (each tab replaced by 2 spaces in this print)
 29       else
Line contains tabs (each tab replaced by 2 spaces in this print)
 30           nextState = (hMove == 4'h2) ? S5 : S4;
 31       end
 32       S4:
 33         nextState = S4;
 34       S5:
 35         nextState = S5;
 36     endcase
 37
 38
 39   // output generation
 40   always_comb begin
 41    cMove = 4'b0000;
Line contains tabs (each tab replaced by 2 spaces in this print)
 42    win = 1'b1;
Line contains tabs (each tab replaced by 2 spaces in this print)
 43     if (currState == S0) begin
 44       cMove = 4'h5;
 45       win = 0;
 46     end
 47     if (currState == S1) begin
 48       cMove = 4'h1;
 49       win = 0;
 50     end
 51     if (currState == S2) begin
 52       cMove = 4'h9;
 53       win = 1;
 54     end
 55     if (currState == S3) begin
 56       cMove = 4'h3;
 57       win = 0;
 58     end
 59     if (currState == S4) begin
```

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

Lab Code [15 points]
Filename: 03_testbench.sv
AndrewID: xinyew

```
1  `default_nettype none
2
3  module testBench();
4      logic [3:0] cMove, hMove;
5      logic clock, reset, win;
6      myAbstractFSM dut1(.clock(clock),
7                         .reset(reset),
8                         .win(win),
9                         .cMove(cMove),
10                        .hMove(hMove));
11
12     initial begin
13         clock = 0;
14         forever #5 clock = ~clock;
15     end
16
17     initial begin
18         $monitor($time,, "state=%s, cMove=%d, hMove=%d, win=%b",
19                  dut1.currState.name, cMove, hMove, win);
20         // initialize values
21         hMove <= 4'hF;
22         reset <= 1'b1;
23
24         // reset the FSM
25         @(posedge clock); // wait for a positive clock edge
26         @(posedge clock); // one edge is enough, but what the heck
27         @(posedge clock);
28
29         @(posedge clock); // begin cycle 0
30         reset <= 1'b0; // release the reset
31
32
33         // start an example sequence -- not meaningful for the lab
34         hMove <= 4'h6; // these changes are after the clock edge
35                        // which means the state change happens
36                        // AFTER the next clock edge
37         @(posedge clock); // begin cycle 1
38         hMove <= 4'h1;
39
40
41         // reset the FSM
42         @(posedge clock);
43         @(posedge clock);
44         @(posedge clock);
45
46         reset <= 1'b1;
47         @(posedge clock);
48         reset <= 1'b0;
49         //start
50         hMove <= 4'h6;
51         @(posedge clock); // 2-7 TEST
52         hMove <= 4'h9;
53         @(posedge clock);
54         hMove <= 4'h2;
55
56
57         // reset the FSM
58         @(posedge clock);
59         @(posedge clock);
60         @(posedge clock);
61
62         reset <= 1'b1;
63         @(posedge clock);
64         reset <= 1'b0;
65         //start
66         hMove <= 4'h6;
67         @(posedge clock); // 7-2 TEST
68         hMove <= 4'h9;
69         @(posedge clock);
```

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

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

```
212         @(posedge clock); // not 7-2 test
213         hMove <= 4'h9;
214         @(posedge clock);
215         hMove <= 4'h3;
216         @(posedge clock);
217         hMove <= 4'h8;
218         @(posedge clock);
219 
220         // reset the FSM
221         @(posedge clock);
222         @(posedge clock);
223         @(posedge clock);
224         reset <= 1'b1;
225         @(posedge clock);
226         reset <= 1'b0;
227         //start
228         hMove <= 4'h6;
229         @(posedge clock); // not 7-2 test
230         hMove <= 4'h9;
231         @(posedge clock);
232         hMove <= 4'h3;
233         @(posedge clock);
234         hMove <= 4'h7;
235         @(posedge clock);
236         hMove <= 4'h4;
237         @(posedge clock);
238 
239         // reset the FSM
240         @(posedge clock);
241         @(posedge clock);
242         @(posedge clock);
243         reset <= 1'b1;
244         @(posedge clock);
245         reset <= 1'b0;
246         //start
247         hMove <= 4'h6;
248         @(posedge clock); // not 7-2 test
249         hMove <= 4'h9;
250         @(posedge clock);
251         hMove <= 4'h3;
252         @(posedge clock);
253         hMove <= 4'h2;
254         @(posedge clock);
255         hMove <= 4'h4;
256         @(posedge clock);
257 
258         @(posedge clock);
259         @(posedge clock);
260         @(posedge clock);
261         reset <= 1'b1;
262         @(posedge clock);
263         reset <= 1'b0;
264 
265         #1 $finish;
266     end
267 
268 endmodule : testBench
269 
270 
271 module myFSM_test(
272     input logic [3:0] cMove,
273     input logic win,
274     input logic q2, q1, q0,
275     output logic [3:0] hMove,
276     output logic clock, reset);
277 
278 
279 
280 
281 endmodule: myFSM_test
```

Lab Code [15 points]
Filename: 04_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
```

Lab Code [15 points]
Filename: 05_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: 05_chipInterface.sv
AndrewID: xinyew

```
`default_nettype none // Required in every sv file
module chipInterface
  (input logic [3:0] KEY,
  input logic [17:0] SW,
  input logic CLOCK_50,
  output logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, HEX6, HEX7,
  output logic [7:0] LEDG,
  output logic [3:0] LEDR);
  logic [3:0] h_3, h_2, h_1, h_0, c_3, c_2, c_1, c_0;

  logic ww;

  task5(.newGame_L(KEY[0]),
            .clock(CLOCK_50),
        .h3(h_3),
Line contains tabs (each tab replaced by 2 spaces in this print)
        .h2(h_2),
Line contains tabs (each tab replaced by 2 spaces in this print)
        .h1(h_1),
Line contains tabs (each tab replaced by 2 spaces in this print)
            .h0(h_0),
        .c3(c_3),
Line contains tabs (each tab replaced by 2 spaces in this print)
        .c2(c_2),
Line contains tabs (each tab replaced by 2 spaces in this print)
        .c1(c_1),
Line contains tabs (each tab replaced by 2 spaces in this print)
        .c0(c_0),
Line contains tabs (each tab replaced by 2 spaces in this print)
        .enter_L(KEY[3]),
Line contains tabs (each tab replaced by 2 spaces in this print)
        .reset(SW[17]),
Line contains tabs (each tab replaced by 2 spaces in this print)
        .hMove(SW[3:0]),
Line contains tabs (each tab replaced by 2 spaces in this print)
        .cMove(LEDR[3:0]),
Line contains tabs (each tab replaced by 2 spaces in this print)
        .win(ww));
Line contains tabs (each tab replaced by 2 spaces in this print)

  assign LEDG = {ww, ww, ww, ww, ww, ww, ww, ww};
  logic [7:0] blank;
  assign blank = 8'b00000000;

  SevenSegmentDisplay(.BCX0(h_3), .blank(blank), .HEX0(HEX7));
  SevenSegmentDisplay(.BCX0(h_2), .blank(blank), .HEX0(HEX6));
  SevenSegmentDisplay(.BCX0(h_1), .blank(blank), .HEX0(HEX5));
  SevenSegmentDisplay(.BCX0(h_0), .blank(blank), .HEX0(HEX4));
  SevenSegmentDisplay(.BCX0(c_3), .blank(blank), .HEX0(HEX3));
  SevenSegmentDisplay(.BCX0(c_2), .blank(blank), .HEX0(HEX2));
  SevenSegmentDisplay(.BCX0(c_1), .blank(blank), .HEX0(HEX1));
  SevenSegmentDisplay(.BCX0(c_0), .blank(blank), .HEX0(HEX0));


endmodule: chipInterface


```

Lab Code [15 points]
Filename: 05_testbench.sv
AndrewID: xinyew

```
`default_nettype none

module testbench();
  logic [3:0] cMove, hMove;
  logic [3:0] h3, h2, h1, h0, c3, c2, c1, c0;
  logic win, clock, reset, enter_L, newGame_L;

  task5 DUT (.*);

  initial begin
    $monitor($time,, "state: %30s cMove: %d hMove: %d win: %b \
// h3: %d h2: %d h1: %d h0: %d c3: %d c2: %d c1: %d c0: %d",
                      DUT.currState.name, cMove, hMove, win,
                      h3, h2, h1, h0, c3, c2, c1, c0);
    // init
    clock = 0;
    reset = 1;
    reset <= 0;

    forever #10 clock = ~clock;
  end

  initial begin
    // C_5
    hMove <= 4'd4;
    enter_L <= 1'd0;
    @(posedge clock) // #10 C_5_I
    @(posedge clock) // #30
    enter_L <= 1'd1;
    @(posedge clock) // #50 C_5
    hMove <= 4'd6;
    @(posedge clock) // #70 C_5
    enter_L <= 1'd0;
    @(posedge clock) // #90 C_5_H_6_E
    @(posedge clock) // #110 C_5_H_6_E
    enter_L <= 1'd1;
    @(posedge clock) // #130 C_1_5_H_6

    enter_L <= 1'd0;
    @(posedge clock) // #150 C_1_5_H_6_I
    enter_L <= 1'd1;
    @(posedge clock) // #170 C_1_5_H_6

    hMove <= 4'd9;
    @(posedge clock); // #190 C_1_5_H_6
    enter_L <= 1'd0;
    @(posedge clock); // #210 C_1_5_H_6_9_E
    enter_L <= 1'd1;
    @(posedge clock); // #230 C_1_3_5_H_6_9
    enter_L <= 1'd0;
    @(posedge clock); // #250 C_1_3_5_H_6_9_I
    @(posedge clock); // #270 C_1_3_5_H_6_9_I
    enter_L <= 1'd1;
    @(posedge clock); // #290 C_1_3_5_H_6_9
    hMove <= 4'd2;
    @(posedge clock); // #310 C_1_3_5_H_6_9
    enter_L <= 1'd0;
    @(posedge clock); // #330 C_1_3_5_H_2_6_9_E
    @(posedge clock); // #350 C_1_3_5_H_2_6_9_E
    enter_L <= 1'd1;
    @(posedge clock); // #370 C_1_3_5_7_H_2_6_9_W
    @(posedge clock); // #390 C_1_3_5_7_H_2_6_9_W
    newGame_L <= 1'd0;
    @(posedge clock); // #410 C_1_3_5_7_H_2_6_9_W_N
    @(posedge clock); // #430 C_1_3_5_7_H_2_6_9_W_N
    newGame_L <= 1'd1;
    @(posedge clock); // #450 C_5
    hMove <= 4'd6;
    enter_L <= 1'd0;
```

```
 70      @(posedge clock); // #470 C_5_H_6_E
 71      enter_L <= 1'd1;
 72      @(posedge clock); // #490 C_1_5_H_6
 73      enter_L <= 1'd0;
 74      @(posedge clock); // #470 C_5_H_6_E
 75      enter_L <= 1'd1;
 76      @(posedge clock); // #490 C_1_5_H_6
 77      enter_L <= 1'd0;
 78      @(posedge clock); // #470 C_5_H_6_E
 79      enter_L <= 1'd1;
 80      @(posedge clock); // #490 C_1_5_H_6
 81      enter_L <= 1'd0;
 82      @(posedge clock); // #470 C_5_H_6_E
 83      enter_L <= 1'd1;
 84      @(posedge clock); // #490 C_1_5_H_6
 85      enter_L <= 1'd0;
 86      @(posedge clock); // #470 C_5_H_6_E
 87      enter_L <= 1'd1;
 88      @(posedge clock); // #490 C_1_5_H_6
 89      enter_L <= 1'd0;
 90      @(posedge clock); // #470 C_5_H_6_E
 91      enter_L <= 1'd1;
 92      @(posedge clock); // #490 C_1_5_H_6
 93      enter_L <= 1'd0;
 94      @(posedge clock); // #470 C_5_H_6_E
 95      enter_L <= 1'd1;
 96      @(posedge clock); // #490 C_1_5_H_6
 97      enter_L <= 1'd0;
 98      @(posedge clock); // #470 C_5_H_6_E
 99      enter_L <= 1'd1;
100      @(posedge clock); // #490 C_1_5_H_6
101      enter_L <= 1'd0;
102      @(posedge clock); // #470 C_5_H_6_E
103      enter_L <= 1'd1;
104      @(posedge clock); // #490 C_1_5_H_6
105      enter_L <= 1'd0;
106      @(posedge clock); // #470 C_5_H_6_E
107      enter_L <= 1'd1;
108      @(posedge clock); // #490 C_1_5_H_6
109      enter_L <= 1'd0;
110      @(posedge clock); // #470 C_5_H_6_E
111      enter_L <= 1'd1;
112      @(posedge clock); // #490 C_1_5_H_6
113      enter_L <= 1'd0;
114      @(posedge clock); // #470 C_5_H_6_E
115      enter_L <= 1'd1;
116      @(posedge clock); // #490 C_1_5_H_6
117      enter_L <= 1'd0;
118      @(posedge clock); // #470 C_5_H_6_E
119      enter_L <= 1'd1;
120      @(posedge clock); // #490 C_1_5_H_6
121      enter_L <= 1'd0;
122      @(posedge clock); // #470 C_5_H_6_E
123      enter_L <= 1'd1;
124      @(posedge clock); // #490 C_1_5_H_6
125      enter_L <= 1'd0;
126      @(posedge clock); // #470 C_5_H_6_E
127      enter_L <= 1'd1;
128      @(posedge clock); // #490 C_1_5_H_6
129      enter_L <= 1'd0;
130      @(posedge clock); // #470 C_5_H_6_E
131      enter_L <= 1'd1;
132      @(posedge clock); // #490 C_1_5_H_6
133       enter_L <= 1'd0;
134      @(posedge clock); // #470 C_5_H_6_E
135      enter_L <= 1'd1;
136      @(posedge clock); // #490 C_1_5_H_6
137      enter_L <= 1'd0;
138      @(posedge clock); // #470 C_5_H_6_E
139      enter_L <= 1'd1;
140      @(posedge clock); // #490 C_1_5_H_6
```

```
141     enter_L <= 1'd0;
142     @(posedge clock); // #470 C_5_H_6_E
143     enter_L <= 1'd1;
144     @(posedge clock); // #490 C_1_5_H_6
145     enter_L <= 1'd0;
146     @(posedge clock); // #470 C_5_H_6_E
147     enter_L <= 1'd1;
148     @(posedge clock); // #490 C_1_5_H_6
149     enter_L <= 1'd0;
150     @(posedge clock); // #470 C_5_H_6_E
151     enter_L <= 1'd1;
152     @(posedge clock); // #490 C_1_5_H_6
153     enter_L <= 1'd0;
154     @(posedge clock); // #470 C_5_H_6_E
155     enter_L <= 1'd1;
156     @(posedge clock); // #490 C_1_5_H_6
157     enter_L <= 1'd0;
158     @(posedge clock); // #470 C_5_H_6_E
159     enter_L <= 1'd1;
160     @(posedge clock); // #490 C_1_5_H_6
161     enter_L <= 1'd0;
162     @(posedge clock); // #470 C_5_H_6_E
163     enter_L <= 1'd1;
164     @(posedge clock); // #490 C_1_5_H_6
165     enter_L <= 1'd0;
166     @(posedge clock); // #470 C_5_H_6_E
167     enter_L <= 1'd1;
168     @(posedge clock); // #490 C_1_5_H_6
169     enter_L <= 1'd0;
170     @(posedge clock); // #470 C_5_H_6_E
171     enter_L <= 1'd1;
172     @(posedge clock); // #490 C_1_5_H_6
173     enter_L <= 1'd0;
174     @(posedge clock); // #470 C_5_H_6_E
175     enter_L <= 1'd1;
176     @(posedge clock); // #490 C_1_5_H_6
177     enter_L <= 1'd0;
178     @(posedge clock); // #470 C_5_H_6_E
179     enter_L <= 1'd1;
180     @(posedge clock); // #490 C_1_5_H_6
181     enter_L <= 1'd0;
182     @(posedge clock); // #470 C_5_H_6_E
183     enter_L <= 1'd1;
184     @(posedge clock); // #490 C_1_5_H_6
185     enter_L <= 1'd0;
186     @(posedge clock); // #470 C_5_H_6_E
187     enter_L <= 1'd1;
188     @(posedge clock); // #490 C_1_5_H_6
189     enter_L <= 1'd0;
190     @(posedge clock); // #470 C_5_H_6_E
191     enter_L <= 1'd1;
192     @(posedge clock); // #490 C_1_5_H_6
193     enter_L <= 1'd0;
194     @(posedge clock); // #470 C_5_H_6_E
195     enter_L <= 1'd1;
196     @(posedge clock); // #490 C_1_5_H_6
197     enter_L <= 1'd0;
198     @(posedge clock); // #470 C_5_H_6_E
199     enter_L <= 1'd1;
200     @(posedge clock); // #490 C_1_5_H_6
201     enter_L <= 1'd0;
202     @(posedge clock); // #470 C_5_H_6_E
203     enter_L <= 1'd1;
204     @(posedge clock); // #490 C_1_5_H_6
205     enter_L <= 1'd0;
206     @(posedge clock); // #470 C_5_H_6_E
207     enter_L <= 1'd1;
208     @(posedge clock); // #490 C_1_5_H_6
209     enter_L <= 1'd0;
210     @(posedge clock); // #470 C_5_H_6_E
211     enter_L <= 1'd1;
```

```
212     @(posedge clock); // #490 C_1_5_H_6
213     enter_L <= 1'd0;
214     @(posedge clock); // #470 C_5_H_6_E
215     enter_L <= 1'd1;
216     @(posedge clock); // #490 C_1_5_H_6
217     enter_L <= 1'd0;
218     @(posedge clock); // #470 C_5_H_6_E
219     enter_L <= 1'd1;
220     @(posedge clock); // #490 C_1_5_H_6
221     enter_L <= 1'd0;
222     @(posedge clock); // #470 C_5_H_6_E
223     enter_L <= 1'd1;
224     @(posedge clock); // #490 C_1_5_H_6
225     enter_L <= 1'd0;
226     @(posedge clock); // #470 C_5_H_6_E
227     enter_L <= 1'd1;
228     @(posedge clock); // #490 C_1_5_H_6
229     #1 $finish;
230   end
231 endmodule : testbench
```

Lab Code [15 points]
Filename: 05_theBigGame.sv
AndrewID: xinyew

```
`default_nettype none

module task5 (
  output logic [3:0] cMove,
  output logic       win,
  output logic [3:0] h3, h2, h1, h0, c3, c2, c1, c0,
  input  logic [3:0] hMove,
  input  logic       clock, reset, enter_L, newGame_L);

  // C_1_2_H_3_4_E_I means that
  // computer's move: 1, 2
  // human move: 3, 4
  // enter_L pressed
  // this is an invalid move
  // 'W' means that computer wins
  // 'N' means that newGame_L pressed
  enum logic [5:0] {
                    C_5 = 6'd0,
                    C_5_I = 6'd1,
                    C_5_H_6_E = 6'd2,
                    C_1_5_H_6 = 6'd3,
                    C_1_5_H_6_I = 6'd4,
                    C_1_5_H_2_6_E = 6'd5,
                    C_1_5_H_3_6_E = 6'd6,
                    C_1_5_H_4_6_E = 6'd7,
                    C_1_5_H_6_7_E = 6'd8,
                    C_1_5_H_6_8_E = 6'd9,
                    C_1_5_H_6_9_E = 6'd10,
                    C_1_5_9_H_2_6_W = 6'd11,
                    C_1_5_9_H_2_6_W_N = 6'd12,
                    C_1_5_9_H_3_6_W = 6'd13,
                    C_1_5_9_H_3_6_W_N = 6'd14,
                    C_1_5_9_H_4_6_W = 6'd15,
                    C_1_5_9_H_4_6_W_N = 6'd16,
                    C_1_5_9_H_6_7_W = 6'd17,
                    C_1_5_9_H_6_7_W_N = 6'd18,
                    C_1_5_9_H_6_8_W = 6'd19,
                    C_1_5_9_H_6_8_W_N = 6'd20,
                    C_1_3_5_H_6_9 = 6'd21,
                    C_1_3_5_H_6_9_I = 6'd22,
                    C_1_3_5_H_2_6_9_E = 6'd23,
                    C_1_3_5_H_4_6_9_E = 6'd24,
                    C_1_3_5_H_6_7_9_E = 6'd25,
                    C_1_3_5_H_6_8_9_E = 6'd26,
                    C_1_2_3_5_H_4_6_9_W = 6'd27,
                    C_1_2_3_5_H_4_6_9_W_N = 6'd28,
                    C_1_2_3_5_H_6_7_9_W = 6'd29,
                    C_1_2_3_5_H_6_7_9_W_N = 6'd30,
                    C_1_2_3_5_H_6_8_9_W = 6'd31,
                    C_1_2_3_5_H_6_8_9_W_N = 6'd32,
                    C_1_3_5_7_H_2_6_9_W = 6'd33,
                    C_1_3_5_7_H_2_6_9_W_N = 6'd34
                   } currState, nextState;

  // next state generation
  always_comb begin
    unique case (currState)
      C_5: begin
        if (~enter_L) begin
          if (hMove != 4'd6)
            nextState = C_5_I;
          else
            nextState = C_5_H_6_E;
        end
      else
Line contains tabs (each tab replaced by 2 spaces in this print)
        nextState = C_5;
Line contains tabs (each tab replaced by 2 spaces in this print)
      end
```

```
68        C_1_5_H_6: begin
69          if (~enter_L)
70            unique case (hMove)
71              4'd1,
72              4'd5,
73              4'd6: nextState = C_1_5_H_6_I;
74              4'd2: nextState = C_1_5_H_2_6_E;
75              4'd3: nextState = C_1_5_H_3_6_E;
76              4'd4: nextState = C_1_5_H_4_6_E;
77              4'd7: nextState = C_1_5_H_6_7_E;
78              4'd8: nextState = C_1_5_H_6_8_E;
79              4'd9: nextState = C_1_5_H_6_9_E;
80            endcase
81          else
82            nextState = C_1_5_H_6;
83        end
84        C_1_3_5_H_6_9:
85          if (~enter_L)
86            unique case (hMove)
87              4'd1,
88              4'd3,
89              4'd5,
90              4'd6,
91              4'd9: nextState = C_1_3_5_H_6_9_I;
92              4'd2: nextState = C_1_3_5_H_2_6_9_E;
93              4'd4: nextState = C_1_3_5_H_4_6_9_E;
94              4'd7: nextState = C_1_3_5_H_6_7_9_E;
95              4'd8: nextState = C_1_3_5_H_6_8_9_E;
96            endcase
97          else
98            nextState = C_1_3_5_H_6_9;
99        C_5_H_6_E:
100         if (enter_L)
101           nextState = C_1_5_H_6;
102         else
103           nextState = C_5_H_6_E;
104       C_1_5_H_2_6_E:
105         if (enter_L)
106           nextState = C_1_5_9_H_2_6_W;
107         else
108           nextState = C_1_5_H_2_6_E;
109       C_1_5_H_3_6_E:
110         if (enter_L)
111           nextState = C_1_5_9_H_3_6_W;
112         else
113           nextState = C_1_5_H_3_6_E;
114       C_1_5_H_4_6_E:
115         if (enter_L)
116           nextState = C_1_5_9_H_4_6_W;
117         else
118           nextState = C_1_5_H_4_6_E;
119       C_1_5_H_6_7_E:
120         if (enter_L)
121           nextState = C_1_5_9_H_6_7_W;
122         else
123           nextState = C_1_5_H_6_7_E;
124       C_1_5_H_6_8_E:
125         if (enter_L)
126           nextState = C_1_5_9_H_6_8_W;
127         else
128           nextState = C_1_5_H_6_8_E;
129       C_1_5_H_6_9_E:
130         if (enter_L)
131           nextState = C_1_3_5_H_6_9;
132         else
133           nextState = C_1_5_H_6_9_E;
134       C_1_3_5_H_2_6_9_E:
135         if (enter_L)
136           nextState = C_1_3_5_7_H_2_6_9_W;
137         else
138           nextState = C_1_3_5_H_2_6_9_E;
```

```
139         C_1_3_5_H_4_6_9_E:
140           if (enter_L)
141             nextState = C_1_2_3_5_H_4_6_9_W;
142           else
143             nextState = C_1_3_5_H_4_6_9_E;
144         C_1_3_5_H_6_7_9_E:
145           if (enter_L)
146             nextState = C_1_2_3_5_H_6_7_9_W;
147           else
148             nextState = C_1_3_5_H_6_7_9_E;
149         C_1_3_5_H_6_8_9_E:
150           if (enter_L)
151             nextState = C_1_2_3_5_H_6_8_9_W;
152           else
153             nextState = C_1_3_5_H_6_8_9_E;
154
155         C_1_5_9_H_2_6_W:
156           if (~newGame_L)
157             nextState = C_1_5_9_H_2_6_W_N;
158           else
159             nextState = C_1_5_9_H_2_6_W;
160         C_1_5_9_H_3_6_W:
161           if (~newGame_L)
162             nextState = C_1_5_9_H_3_6_W_N;
163           else
164             nextState = C_1_5_9_H_3_6_W;
165         C_1_5_9_H_4_6_W:
166           if (~newGame_L)
167             nextState = C_1_5_9_H_4_6_W_N;
168           else
169             nextState = C_1_5_9_H_4_6_W;
170         C_1_5_9_H_6_7_W:
171           if (~newGame_L)
172             nextState = C_1_5_9_H_6_7_W_N;
173           else
174             nextState = C_1_5_9_H_6_7_W;
175         C_1_5_9_H_6_8_W:
176           if (~newGame_L)
177             nextState = C_1_5_9_H_6_8_W_N;
178           else
179             nextState = C_1_5_9_H_6_8_W;
180         C_1_3_5_7_H_2_6_9_W:
181           if (~newGame_L)
182             nextState = C_1_3_5_7_H_2_6_9_W_N;
183           else
184             nextState = C_1_3_5_7_H_2_6_9_W;
185         C_1_2_3_5_H_4_6_9_W:
186           if (~newGame_L)
187             nextState = C_1_2_3_5_H_4_6_9_W_N;
188           else
189             nextState = C_1_2_3_5_H_4_6_9_W;
190         C_1_2_3_5_H_6_7_9_W:
191           if (~newGame_L)
192             nextState = C_1_2_3_5_H_6_7_9_W_N;
193           else
194             nextState = C_1_2_3_5_H_6_7_9_W;
195         C_1_2_3_5_H_6_8_9_W:
196           if (~newGame_L)
197             nextState = C_1_2_3_5_H_6_8_9_W_N;
198           else
199             nextState = C_1_2_3_5_H_6_8_9_W;
200
201         C_5_I:
202           if (enter_L)
203             nextState = C_5;
204           else
205             nextState = C_5_I;
206         C_1_5_H_6_I:
207           if (enter_L)
208             nextState = C_1_5_H_6;
209           else
```

```
210           nextState = C_1_5_H_6_I;
211       C_1_3_5_H_6_9_I:
212         if (enter_L)
213           nextState = C_1_3_5_H_6_9;
214         else
215           nextState = C_1_3_5_H_6_9_I;
216
217       C_1_5_9_H_2_6_W_N:
218         if (newGame_L)
219           nextState = C_5;
220         else
221           nextState = C_1_5_9_H_2_6_W_N;
222       C_1_5_9_H_3_6_W_N:
223         if (newGame_L)
224           nextState = C_5;
225         else
226           nextState = C_1_5_9_H_3_6_W_N;
227       C_1_5_9_H_4_6_W_N:
228         if (newGame_L)
229           nextState = C_5;
230         else
231           nextState = C_1_5_9_H_4_6_W_N;
232       C_1_5_9_H_6_7_W_N:
233         if (newGame_L)
234           nextState = C_5;
235         else
236           nextState = C_1_5_9_H_6_7_W_N;
237       C_1_5_9_H_6_8_W_N:
238         if (newGame_L)
239           nextState = C_5;
240         else
241           nextState = C_1_5_9_H_6_8_W_N;
242       C_1_2_3_5_H_4_6_9_W_N:
243         if (newGame_L)
244           nextState = C_5;
245         else
246           nextState = C_1_2_3_5_H_4_6_9_W_N;
247       C_1_2_3_5_H_6_7_9_W_N:
248         if (newGame_L)
249           nextState = C_5;
250         else
251           nextState = C_1_2_3_5_H_6_7_9_W_N;
252       C_1_2_3_5_H_6_8_9_W_N:
253         if (newGame_L)
254           nextState = C_5;
255       else
Line contains tabs (each tab replaced by 2 spaces in this print)
256         nextState = C_1_2_3_5_H_6_8_9_W_N;
Line contains tabs (each tab replaced by 2 spaces in this print)
257       C_1_3_5_7_H_2_6_9_W_N:
258         if (newGame_L)
259           nextState = C_5;
260         else
261           nextState = C_1_3_5_7_H_2_6_9_W_N;
262     endcase
263   end
264
265
266
267 always_comb begin
268     cMove = 4'b0000;
Line contains tabs (each tab replaced by 2 spaces in this print)
269     win = 1'b1;
Line contains tabs (each tab replaced by 2 spaces in this print)
270       {c3, c2, c1, c0, h3, h2, h1, h0} = 32'b0;
271     if (currState == C_5) begin
272       cMove = 4'h5;
273       win = 0;
274       c3 = 4'd5;
275     end
276
```

```
277     if (currState == C_5_I) begin
278       cMove = 4'h5;
279       win = 0;
280       c3 = 4'd5;
281     end
282
283     if (currState == C_5_H_6_E) begin
284       cMove = 4'h5;
285       win = 0;
286       c3 = 4'd5;
287     end
288
289     if (currState == C_1_5_H_6) begin
290       cMove = 4'h1;
291       win = 0;
292       c3 = 4'd1;
293       c2 = 4'd5;
294       h3 = 4'd6;
295     end
296
297     if (currState == C_1_5_H_6_I) begin
298       cMove = 4'h1;
299       win = 0;
300       c3 = 4'd1;
301       c2 = 4'd5;
302       h3 = 4'd6;
303     end
304
305
306     if (currState == C_1_5_H_2_6_E) begin
307       cMove = 4'h1;
308       win = 0;
309       c3 = 4'd1;
310       c2 = 4'd5;
311       h3 = 4'd6;
312     end
313     if (currState == C_1_5_H_3_6_E) begin
314       cMove = 4'h1;
315       win = 0;
316       c3 = 4'd1;
317       c2 = 4'd5;
318       h3 = 4'd6;
319     end
320     if (currState == C_1_5_H_4_6_E) begin
321       cMove = 4'h1;
322       win = 0;
323       c3 = 4'd1;
324       c2 = 4'd5;
325       h3 = 4'd6;
326     end
327     if (currState == C_1_5_H_6_7_E) begin
328       cMove = 4'h1;
329       win = 0;
330       c3 = 4'd1;
331       c2 = 4'd5;
332       h3 = 4'd6;
333     end
334     if (currState == C_1_5_H_6_8_E) begin
335       cMove = 4'h1;
336       win = 0;
337       c3 = 4'd1;
338       c2 = 4'd5;
339       h3 = 4'd6;
340     end
341     if (currState == C_1_5_H_6_9_E) begin
342       cMove = 4'h1;
343       win = 0;
344       c3 = 4'd1;
345       c2 = 4'd5;
346       h3 = 4'd6;
347     end
```

```
348
349
350     if (currState == C_1_5_9_H_2_6_W) begin
351       cMove = 4'h7;
352       win = 1;
353       c3 = 4'd1;
354       c2 = 4'd5;
355       c1 = 4'd9;
356       h3 = 4'd2;
357       h2 = 4'd6;
358     end
359     if (currState == C_1_5_9_H_2_6_W_N) begin
360       cMove = 4'h9;
361       win = 1;
362       c3 = 4'd1;
363       c2 = 4'd5;
364       c1 = 4'd9;
365       h3 = 4'd2;
366       h2 = 4'd6;
367     end
368
369     if (currState == C_1_5_9_H_3_6_W) begin
370       cMove = 4'h9;
371       win = 1;
372       c3 = 4'd1;
373       c2 = 4'd5;
374       c1 = 4'd7;
375       h3 = 4'd3;
376       h2 = 4'd6;
377     end
378     if (currState == C_1_5_9_H_3_6_W_N) begin
379       cMove = 4'h9;
380       win = 1;
381       c3 = 4'd1;
382       c2 = 4'd5;
383       c1 = 4'd9;
384       h3 = 4'd3;
385       h2 = 4'd6;
386     end
387
388     if (currState == C_1_5_9_H_4_6_W) begin
389       cMove = 4'h9;
390       win = 1;
391       c3 = 4'd1;
392       c2 = 4'd5;
393       c1 = 4'd9;
394       h3 = 4'd4;
395       h2 = 4'd6;
396     end
397     if (currState == C_1_5_9_H_4_6_W_N) begin
398       cMove = 4'h9;
399       win = 1;
400       c3 = 4'd1;
401       c2 = 4'd5;
402       c1 = 4'd9;
403       h3 = 4'd4;
404       h2 = 4'd6;
405     end
406
407     if (currState == C_1_5_9_H_6_7_W) begin
408       cMove = 4'h9;
409       win = 1;
410       c3 = 4'd1;
411       c2 = 4'd5;
412       c1 = 4'd9;
413       h3 = 4'd6;
414       h2 = 4'd7;
415     end
416     if (currState == C_1_5_9_H_6_7_W_N) begin
417       cMove = 4'h9;
418       win = 1;
```

```
419         c3 = 4'd1;
420         c2 = 4'd5;
421         c1 = 4'd9;
422         h3 = 4'd6;
423         h2 = 4'd7;
424       end
425
426
427
428       if (currState == C_1_5_9_H_6_8_W) begin
429         cMove = 4'h9;
430         win = 1;
431         c3 = 4'd1;
432         c2 = 4'd5;
433         c1 = 4'd9;
434         h3 = 4'd6;
435         h2 = 4'd8;
436       end
437       if (currState == C_1_5_9_H_6_8_W_N) begin
438         cMove = 4'h9;
439         win = 1;
440         c3 = 4'd1;
441         c2 = 4'd5;
442         c1 = 4'd9;
443         h3 = 4'd6;
444         h2 = 4'd8;
445       end
446
447
448       if (currState == C_1_3_5_H_6_9) begin
449         cMove = 4'h3;
450         win = 0;
451         c3 = 4'd1;
452         c2 = 4'd3;
453         c1 = 4'd5;
454         h3 = 4'd6;
455         h2 = 4'd9;
456       end
457       if (currState == C_1_3_5_H_6_9_I) begin
458         cMove = 4'h3;
459         win = 0;
460         c3 = 4'd1;
461         c2 = 4'd3;
462         c1 = 4'd5;
463         h3 = 4'd6;
464         h2 = 4'd9;
465       end
466
467       if (currState == C_1_3_5_H_2_6_9_E) begin
468         cMove = 4'h3;
469         win = 0;
470         c3 = 4'd1;
471         c2 = 4'd3;
472         c1 = 4'd5;
473         h3 = 4'd2;
474         h2 = 4'd6;
475         h1 = 4'd9;
476       end
477
478       if (currState == C_1_3_5_H_4_6_9_E) begin
479         cMove = 4'h3;
480         win = 0;
481         c3 = 4'd1;
482         c2 = 4'd3;
483         c1 = 4'd5;
484         h3 = 4'd4;
485         h2 = 4'd6;
486         h1 = 4'd9;
487       end
488
489       if (currState == C_1_3_5_H_6_7_9_E) begin
```

```
490       cMove = 4'h3;
491       win = 0;
492       c3 = 4'd1;
493       c2 = 4'd3;
494       c1 = 4'd5;
495       h3 = 4'd6;
496       h2 = 4'd7;
497       h1 = 4'd9;
498     end
499
500     if (currState == C_1_3_5_H_6_8_9_E) begin
501       cMove = 4'h3;
502       win = 0;
503       c3 = 4'd1;
504       c2 = 4'd3;
505       c1 = 4'd5;
Line contains tabs (each tab replaced by 2 spaces in this print)
506       h3 = 4'd6;
507       h2 = 4'd8;
508       h1 = 4'd9;
509     end
510
511     if (currState == C_1_2_3_5_H_4_6_9_W) begin
512       cMove = 4'h2;
513       win = 1;
514       c3 = 4'd1;
515       c2 = 4'd2;
516       c1 = 4'd3;
517       c0 = 4'd5;
518       h3 = 4'd4;
519       h2 = 4'd6;
520       h1 = 4'd9;
521     end
522
523     if (currState == C_1_2_3_5_H_4_6_9_W_N) begin
524       cMove = 4'h2;
525       win = 1;
526       c3 = 4'd1;
527       c2 = 4'd2;
528       c1 = 4'd3;
529       c0 = 4'd5;
530       h3 = 4'd4;
531       h2 = 4'd6;
532       h1 = 4'd9;
533     end
534
535     if (currState == C_1_2_3_5_H_6_7_9_W) begin
536       cMove = 4'h2;
537       win = 1;
538       c3 = 4'd1;
539       c2 = 4'd2;
540       c1 = 4'd3;
541       c0 = 4'd5;
542       h3 = 4'd6;
543       h2 = 4'd7;
544       h1 = 4'd9;
545     end
546
547     if (currState == C_1_2_3_5_H_6_7_9_W_N) begin
548       cMove = 4'h2;
549       win = 1;
550       c3 = 4'd1;
551       c2 = 4'd2;
552       c1 = 4'd3;
553       c0 = 4'd5;
554       h3 = 4'd6;
555       h2 = 4'd7;
556       h1 = 4'd9;
557
558     end
559
```

```
560     if (currState == C_1_2_3_5_H_6_8_9_W) begin
561       cMove = 4'h2;
562       win = 1;
563       c3 = 4'd1;
564       c2 = 4'd2;
565       c1 = 4'd3;
566       c0 = 4'd5;
567       h3 = 4'd6;
568       h2 = 4'd8;
569       h1 = 4'd9;
570     end
571
572     if (currState == C_1_2_3_5_H_6_8_9_W_N) begin
573       cMove = 4'h2;
574       win = 1;
575       c3 = 4'd1;
576       c2 = 4'd2;
577       c1 = 4'd3;
578       c0 = 4'd5;
579       h3 = 4'd6;
580       h2 = 4'd8;
581       h1 = 4'd9;
582
583     end
584
585     if (currState == C_1_3_5_7_H_2_6_9_W) begin
586       cMove = 4'h7;
587       win = 1;
588       c3 = 4'd1;
589       c2 = 4'd3;
590       c1 = 4'd5;
591       c0 = 4'd7;
592       h3 = 4'd2;
593       h2 = 4'd6;
594       h1 = 4'd9;
595     end
596
597     if (currState == C_1_3_5_7_H_2_6_9_W_N) begin
598       cMove = 4'h7;
599       win = 1;
600       c3 = 4'd1;
601       c2 = 4'd3;
602       c1 = 4'd5;
603       c0 = 4'd7;
604       h3 = 4'd2;
605       h2 = 4'd6;
606       h1 = 4'd9;
607     end
608   end
609
610     // register
611   always_ff @(posedge clock, posedge reset)
612     if (reset)
613       currState <= C_5;
614     else
615       currState <= nextState;
616
617 endmodule: task5
618
```

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

```
`default_nettype none // Required in every sv file
module chipInterface
  (input logic [3:0] KEY,
  input logic [17:0] SW,
  input logic CLOCK_50,
  output logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, HEX6, HEX7,
  output logic [7:0] LEDG,
  output logic [3:0] LEDR);
  logic [3:0] h_3, h_2, h_1, h_0, c_3, c_2, c_1, c_0;

  logic ww;

  task5(.newGame_L(KEY[0]),
            .clock(CLOCK_50),
        .h3(h_3),
Line contains tabs (each tab replaced by 2 spaces in this print)
        .h2(h_2),
Line contains tabs (each tab replaced by 2 spaces in this print)
        .h1(h_1),
Line contains tabs (each tab replaced by 2 spaces in this print)
            .h0(h_0),
        .c3(c_3),
Line contains tabs (each tab replaced by 2 spaces in this print)
        .c2(c_2),
Line contains tabs (each tab replaced by 2 spaces in this print)
        .c1(c_1),
Line contains tabs (each tab replaced by 2 spaces in this print)
        .c0(c_0),
Line contains tabs (each tab replaced by 2 spaces in this print)
        .enter_L(KEY[3]),
Line contains tabs (each tab replaced by 2 spaces in this print)
        .reset(SW[17]),
Line contains tabs (each tab replaced by 2 spaces in this print)
        .hMove(SW[3:0]),
Line contains tabs (each tab replaced by 2 spaces in this print)
        .cMove(LEDR[3:0]),
Line contains tabs (each tab replaced by 2 spaces in this print)
        .win(ww));
Line contains tabs (each tab replaced by 2 spaces in this print)

  assign LEDG = {ww, ww, ww, ww, ww, ww, ww, ww};
  logic [7:0] blank;
  assign blank = 8'b00000000;

  SevenSegmentDisplay(.BCX0(h_3), .blank(blank), .HEX0(HEX7));
  SevenSegmentDisplay(.BCX0(h_2), .blank(blank), .HEX0(HEX6));
  SevenSegmentDisplay(.BCX0(h_1), .blank(blank), .HEX0(HEX5));
  SevenSegmentDisplay(.BCX0(h_0), .blank(blank), .HEX0(HEX4));
  SevenSegmentDisplay(.BCX0(c_3), .blank(blank), .HEX0(HEX3));
  SevenSegmentDisplay(.BCX0(c_2), .blank(blank), .HEX0(HEX2));
  SevenSegmentDisplay(.BCX0(c_1), .blank(blank), .HEX0(HEX1));
  SevenSegmentDisplay(.BCX0(c_0), .blank(blank), .HEX0(HEX0));


endmodule: chipInterface


```