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

```
`default_nettype none

module dFlipFlop(
  output logic q,
  input  logic d, clock, reset);

  always_ff @(posedge clock)
    if (reset == 1'b1)
      q <= 0;
    else
      q <= d;

endmodule : dFlipFlop

//  ------------------------------------------------------
// |         |  Q2  Q1  Q0  |  desciption
// |  -------|--------------|-----------------------------------
// |  State1 |   0   0   0  |  computer #5:
// |  State2 |   0   0   1  |  computer #1, #5;
// |  State3 |   0   1   0  |  computer #1, #5, #9;     win
// |  State4 |   0   1   1  |  computer #1, #3, #5;     win
// |  State5 |   1   0   0  |  computer #1, #2, #3, #7; win
// |  State6 |   1   0   1  |  computer #1, #3, #5, #7; win
//  ------------------------------------------------------

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

  logic d0, d1, d2;

  // flip-flops instantiation
  dFlipFlop ff0(.d(d0),
                .q(q0),
                .clock(clock),
                .reset(reset)),
            ff1(.d(d1),
                .q(q1),
                .clock(clock),
                .reset(reset)),
            ff2(.d(d2),
                .q(q2),
                .clock(clock),
                .reset(reset));

  // next state generation
  assign d2 = ((~q2) & q1 & q0 & (~hMove[3]) & (~hMove[2]) & hMove[1]
  & (~hMove[0])) |
              ((~q2) & q1 & q0 & (~hMove[3]) & (~hMove[2]) & hMove[1]
              & (~hMove[0])) |
              ((~q2) & q1 & q0 & (~hMove[3]) & hMove[2] & hMove[1]
              & hMove[0]) |
              ((~q2) & q1 & q0 & hMove[3] & (~hMove[2]) & (~hMove[1])
              & (~hMove[0])) |

              (q2 & (~q1) & (~q0) & (~hMove[3]) & (~hMove[2]) & (~hMove[1])
               & hMove[0]) |
              (q2 & (~q1) & (~q0) & (~hMove[3]) & (~hMove[2]) & hMove[1]
               & (~hMove[0])) |
              (q2 & (~q1) & (~q0) & (~hMove[3]) & (~hMove[2]) & hMove[1]
              & hMove[0]) |
              (q2 & (~q1) & (~q0) & (~hMove[3]) & hMove[2] & (~hMove[1])
              & (~hMove[0])) |
              (q2 & (~q1) & (~q0) & (~hMove[3]) & hMove[2] & (~hMove[1])
              & hMove[0]) |
              (q2 & (~q1) & (~q0) & (~hMove[3]) & hMove[2] & hMove[1]
```

```
 70                & (~hMove[0])) |
 71                (q2 & (~q1) & (~q0) & (~hMove[3]) & hMove[2] & hMove[1]
 72                & hMove[0]) |
 73                (q2 & (~q1) & (~q0) & hMove[3] & (~hMove[2]) & (~hMove[1])
 74                & (~hMove[0])) |
 75                (q2 & (~q1) & (~q0) & hMove[3] & (~hMove[2]) & (~hMove[1])
 76                & hMove[0]) |
 77
 78                (q2 & (~q1) & q0 & (~hMove[3]) & (~hMove[2]) & (~hMove[1])
 79                 & hMove[0]) |
 80                (q2 & (~q1) & q0 & (~hMove[3]) & (~hMove[2]) & hMove[1]
 81                & (~hMove[0])) |
 82                (q2 & (~q1) & q0 & (~hMove[3]) & (~hMove[2]) & hMove[1]
 83                & hMove[0]) |
 84                (q2 & (~q1) & q0 & (~hMove[3]) & hMove[2] & (~hMove[1])
 85                 & (~hMove[0])) |
 86                (q2 & (~q1) & q0 & (~hMove[3]) & hMove[2] & (~hMove[1])
 87                & hMove[0]) |
 88                (q2 & (~q1) & q0 & (~hMove[3]) & hMove[2] & hMove[1]
 89                & (~hMove[0])) |
 90                (q2 & (~q1) & q0 & (~hMove[3]) & hMove[2] & hMove[1]
 91                 & hMove[0]) |
 92                (q2 & (~q1) & q0 & hMove[3] & (~hMove[2]) & (~hMove[1])
 93                & (~hMove[0])) |
 94                (q2 & (~q1) & q0 & hMove[3] & (~hMove[2]) & (~hMove[1])
 95                & hMove[0]);
 96
 97   assign d1 = ((~q2) & (~q1) & q0 & (~hMove[3]) & (~hMove[2]) & hMove[1]
 98   & (~hMove[0])) |
 99                ((~q2) & (~q1) & q0 & (~hMove[3]) & (~hMove[2]) & hMove[1]
100                & hMove[0]) |
101                ((~q2) & (~q1) & q0 & (~hMove[3]) & hMove[2] & (~hMove[1])
102                & (~hMove[0])) |
103                ((~q2) & (~q1) & q0 & (~hMove[3]) & hMove[2] & hMove[1]
104                & hMove[0]) |
105                ((~q2) & (~q1) & q0 & hMove[3] & (~hMove[2]) & (~hMove[1])
106                & (~hMove[0])) |
107                ((~q2) & (~q1) & q0 & hMove[3] & (~hMove[2]) & (~hMove[1])
108                & hMove[0]) |
109
110                ((~q2) & q1 & (~q0) & (~hMove[3]) & (~hMove[2]) & (~hMove[1])
111                & hMove[0]) |
112                ((~q2) & q1 & (~q0) & (~hMove[3]) & (~hMove[2]) & hMove[1]
113                & (~hMove[0])) |
114                ((~q2) & q1 & (~q0) & (~hMove[3]) & (~hMove[2]) & hMove[1]
115                & hMove[0]) |
116                ((~q2) & q1 & (~q0) & (~hMove[3]) & hMove[2] & (~hMove[1])
117                 & (~hMove[0])) |
118                ((~q2) & q1 & (~q0) & (~hMove[3]) & hMove[2] & (~hMove[1])
119                & hMove[0]) |
120                ((~q2) & q1 & (~q0) & (~hMove[3]) & hMove[2] & hMove[1]
121                & (~hMove[0])) |
122                ((~q2) & q1 & (~q0) & (~hMove[3]) & hMove[2] & hMove[1]
123                 & hMove[0]) |
124                ((~q2) & q1 & (~q0) & q0 & hMove[3] & (~hMove[2]) & (~hMove[1])
125                & (~hMove[0])) |
126                ((~q2) & q1 & (~q0) & hMove[3] & (~hMove[2]) & (~hMove[1])
127                & hMove[0]) |
128
129                ((~q2) & q1 & q0 & (~hMove[3]) & (~hMove[2]) & (~hMove[1])
130                 & hMove[0]) |
131                ((~q2) & q1 & q0 & (~hMove[3]) & (~hMove[2]) & hMove[1]
132                 & hMove[0]) |
133                ((~q2) & q1 & q0 & (~hMove[3]) & hMove[2] & (~hMove[1])
134                & hMove[0]) |
135                ((~q2) & q1 & q0 & (~hMove[3]) & hMove[2] & hMove[1]
136                & (~hMove[0])) |
137                ((~q2) & q1 & q0 & hMove[3] & (~hMove[2]) & (~hMove[1])
138                & hMove[0]);
139
140   assign d0 = ((~q2) & (~q1) & (~q0) & (~hMove[3]) & hMove[2] & hMove[1]
```

```
141   & (~hMove[0])) |
142
143               ((~q2) & (~q1) & q0 & (~hMove[3]) & (~hMove[2]) & (~hMove[1])
144               & hMove[0]) |
145               ((~q2) & (~q1) & q0 & (~hMove[3]) & hMove[2] & (~hMove[1])
146               & hMove[0]) |
147               ((~q2) & (~q1) & q0 & (~hMove[3]) & (~hMove[2]) & (~hMove[1])
148               & hMove[0]) |
149
150
151               ((~q2) & q1 & q0 & (~hMove[3]) & (~hMove[2]) & (~hMove[1])
152               & hMove[0]) |
153               ((~q2) & q1 & q0 & (~hMove[3]) & (~hMove[2]) & hMove[1]
154                & (~hMove[0])) |
155               ((~q2) & q1 & q0 & (~hMove[3]) & (~hMove[2]) & hMove[1]
156                & hMove[0]) |
157               ((~q2) & q1 & q0 & (~hMove[3]) & hMove[2] & (~hMove[1])
158               & hMove[0]) |
159               ((~q2) & q1 & q0 & (~hMove[3]) & hMove[2] & hMove[1]
160                & (~hMove[0])) |
161               ((~q2) & q1 & q0 & hMove[3] & (~hMove[2]) & (~hMove[1])
162               & hMove[0]) |
163
164               (q2 & (~q1) & q0 & (~hMove[3]) & (~hMove[2]) & (~hMove[1])
165               & hMove[0]) |
166               (q2 & (~q1) & q0 & (~hMove[3]) & (~hMove[2]) & hMove[1]
167                & (~hMove[0])) |
168               (q2 & (~q1) & q0 & (~hMove[3]) & (~hMove[2]) & hMove[1]
169               & hMove[0]) |
170               (q2 & (~q1) & q0 & (~hMove[3]) & hMove[2] & (~hMove[1])
171               & (~hMove[0])) |
172               (q2 & (~q1) & q0 & (~hMove[3]) & hMove[2] & (~hMove[1])
173                & hMove[0]) |
174               (q2 & (~q1) & q0 & (~hMove[3]) & hMove[2] & hMove[1]
175                & (~hMove[0])) |
176               (q2 & (~q1) & q0 & (~hMove[3]) & hMove[2] & hMove[1]
177               & hMove[0]) |
178               (q2 & (~q1) & q0 & q0 & hMove[3] & (~hMove[2])
179               & (~hMove[1]) & (~hMove[0])) |
180               (q2 & (~q1) & q0 & hMove[3] & (~hMove[2]) & (~hMove[1])
181                & hMove[0]);
182
183   // output logic generation
184   assign cMove[3] = (~q2) & q1 & (~q0);
185
186   assign cMove[2] = ((~q2) & (~q1) & (~q0)) |
187                     (q2 & (~q1) & q0);
188
189   assign cMove[1] = ((~q2) & q1 & q0) |
190                     (q2 & (~q1) & (~q0)) |
191                     (q2 & (~q1) & q0);
192
193   assign cMove[0] = ((~q2) & (~q1) & (~q0)) |
194                     ((~q2) & (~q1) & q0) |
195                     ((~q2) & q1 & (~q0)) |
196                     ((~q2) & q1 & q0) |
197                     (q2 & (~q1) & q0);
198
199   assign win = ((~q2) & q1 & (~q0)) |
200                (q2 & (~q1) & (~q0)) |
201                (q2 & (~q1) & q0);
202
203 endmodule : myExplicitFSM
204
```

Lab Code [15 points]
Filename: 01_testbench.sv
AndrewID: ocarroll

```
`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: ocarroll

```
`default_nettype none

module myAbstractFSM (
  output logic [3:0] cMove,
  output logic       win,
  input  logic [3:0] hMove,
  input  logic       clock, reset);

  enum logic [2:0] {S0 = 3'b000, S1 = 3'b001,
                    S2 = 3'b010, S3 = 3'b011,
                    S4 = 3'b100, S5 = 3'b101} currState, nextState;

  // next state generation
  always_comb
    unique case (currState)
      S0:
        nextState = (hMove == 4'h6) ? S1 : S0;
      S1: begin
        if (hMove == 4'h1 || hMove == 4'h5 || hMove == 4'h6)
          nextState = S1;
        else
          nextState = (hMove == 4'h9) ? S3 : S2;
      end
      S2:
        nextState = S2;
      S3: begin
        if (hMove == 4'h1 || hMove == 4'h4 || hMove == 4'h5 || hMove == 4'h6
        || hMove == 4'h9)
          nextState = S3;
        else
            nextState = (hMove == 4'h2) ? S5 : S4;
      end
      S4:
        nextState = S4;
      S5:
        nextState = S5;
    endcase


  // output generation
  always_comb begin
    cMove = 4'b0000;
    win = 1'b1;
    if (currState == S0) begin
      cMove = 4'h5;
      win = 0;
    end
    if (currState == S1) begin
      cMove = 4'h1;
      win = 0;
    end
    if (currState == S2) begin
      cMove = 4'h9;
      win = 1;
    end
    if (currState == S3) begin
      cMove = 4'h3;
      win = 0;
    end
    if (currState == S4) begin
      cMove = 4'h2;
      win = 1;
    end
    if (currState == S5) begin
      cMove = 4'h7;
      win = 1;
    end
  end

```

```
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: 03_testbench.sv
AndrewID: ocarroll

```
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: ocarroll

```
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),
11   .win(ww));
12   assign LEDG = {ww, ww, ww, ww, ww, ww, ww, ww};
13   logic [7:0] blank;
14   assign blank = 8'b00000000;
15
16   SevenSegmentDisplay DUT2 (.BCX0(c),
17                             .blank(blank),
18                             .HEX0(HEX0));
19
20 endmodule: chipInterface
```

Lab Code [15 points]
Filename: 05_HexDisplay.sv
AndrewID: ocarroll

```
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: ocarroll

```
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    input logic CLOCK_50,
6    output logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, HEX6, HEX7,
7    output logic [7:0] LEDG,
8    output logic [3:0] LEDR);
9    logic [3:0] h_3, h_2, h_1, h_0, c_3, c_2, c_1, c_0;
10
11   logic ww;
12
13   task5(.newGame_L(KEY[0]),
14          .clock(CLOCK_50),
15          .h3(h_3),
16          .h2(h_2),
17          .h1(h_1),
18          .h0(h_0),
19          .c3(c_3),
20          .c2(c_2),
21          .c1(c_1),
22          .c0(c_0),
23          .enter_L(KEY[3]),
24          .reset(SW[17]),
25          .hMove(SW[3:0]),
26          .cMove(LEDR[3:0]),
27          .win(ww));
28
29   assign LEDG = {ww, ww, ww, ww, ww, ww, ww, ww};
30   logic [7:0] blank;
31   assign blank = 8'b00000000;
32
33   SevenSegmentDisplay(.BCX0(h_3), .blank(blank), .HEX0(HEX7));
34   SevenSegmentDisplay(.BCX0(h_2), .blank(blank), .HEX0(HEX6));
35   SevenSegmentDisplay(.BCX0(h_1), .blank(blank), .HEX0(HEX5));
36   SevenSegmentDisplay(.BCX0(h_0), .blank(blank), .HEX0(HEX4));
37   SevenSegmentDisplay(.BCX0(c_3), .blank(blank), .HEX0(HEX3));
38   SevenSegmentDisplay(.BCX0(c_2), .blank(blank), .HEX0(HEX2));
39   SevenSegmentDisplay(.BCX0(c_1), .blank(blank), .HEX0(HEX1));
40   SevenSegmentDisplay(.BCX0(c_0), .blank(blank), .HEX0(HEX0));
41
42
43 endmodule: chipInterface
44
45
```

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

```
`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: ocarroll

```
`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
          nextState = C_5;
      end
      C_1_5_H_6: begin
        if (~enter_L)
          unique case (hMove)
```

```
70              4'd1,
71              4'd5,
72              4'd6: nextState = C_1_5_H_6_I;
73              4'd2: nextState = C_1_5_H_2_6_E;
74              4'd3: nextState = C_1_5_H_3_6_E;
75              4'd4: nextState = C_1_5_H_4_6_E;
76              4'd7: nextState = C_1_5_H_6_7_E;
77              4'd8: nextState = C_1_5_H_6_8_E;
78              4'd9: nextState = C_1_5_H_6_9_E;
79            endcase
80          else
81            nextState = C_1_5_H_6;
82        end
83        C_1_3_5_H_6_9:
84          if (~enter_L)
85            unique case (hMove)
86              4'd1,
87              4'd3,
88              4'd5,
89              4'd6,
90              4'd9: nextState = C_1_3_5_H_6_9_I;
91              4'd2: nextState = C_1_3_5_H_2_6_9_E;
92              4'd4: nextState = C_1_3_5_H_4_6_9_E;
93              4'd7: nextState = C_1_3_5_H_6_7_9_E;
94              4'd8: nextState = C_1_3_5_H_6_8_9_E;
95            endcase
96          else
97            nextState = C_1_3_5_H_6_9;
98        C_5_H_6_E:
99          if (enter_L)
100           nextState = C_1_5_H_6;
101         else
102           nextState = C_5_H_6_E;
103       C_1_5_H_2_6_E:
104         if (enter_L)
105           nextState = C_1_5_9_H_2_6_W;
106         else
107           nextState = C_1_5_H_2_6_E;
108       C_1_5_H_3_6_E:
109         if (enter_L)
110           nextState = C_1_5_9_H_3_6_W;
111         else
112           nextState = C_1_5_H_3_6_E;
113       C_1_5_H_4_6_E:
114         if (enter_L)
115           nextState = C_1_5_9_H_4_6_W;
116         else
117           nextState = C_1_5_H_4_6_E;
118       C_1_5_H_6_7_E:
119         if (enter_L)
120           nextState = C_1_5_9_H_6_7_W;
121         else
122           nextState = C_1_5_H_6_7_E;
123       C_1_5_H_6_8_E:
124         if (enter_L)
125           nextState = C_1_5_9_H_6_8_W;
126         else
127           nextState = C_1_5_H_6_8_E;
128       C_1_5_H_6_9_E:
129         if (enter_L)
130           nextState = C_1_3_5_H_6_9;
131         else
132           nextState = C_1_5_H_6_9_E;
133       C_1_3_5_H_2_6_9_E:
134         if (enter_L)
135           nextState = C_1_3_5_7_H_2_6_9_W;
136         else
137           nextState = C_1_3_5_H_2_6_9_E;
138       C_1_3_5_H_4_6_9_E:
139         if (enter_L)
140           nextState = C_1_2_3_5_H_4_6_9_W;
```

```
141         else
142           nextState = C_1_3_5_H_4_6_9_E;
143       C_1_3_5_H_6_7_9_E:
144         if (enter_L)
145           nextState = C_1_2_3_5_H_6_7_9_W;
146         else
147           nextState = C_1_3_5_H_6_7_9_E;
148       C_1_3_5_H_6_8_9_E:
149         if (enter_L)
150           nextState = C_1_2_3_5_H_6_8_9_W;
151         else
152           nextState = C_1_3_5_H_6_8_9_E;
153
154       C_1_5_9_H_2_6_W:
155         if (~newGame_L)
156           nextState = C_1_5_9_H_2_6_W_N;
157         else
158           nextState = C_1_5_9_H_2_6_W;
159       C_1_5_9_H_3_6_W:
160         if (~newGame_L)
161           nextState = C_1_5_9_H_3_6_W_N;
162         else
163           nextState = C_1_5_9_H_3_6_W;
164       C_1_5_9_H_4_6_W:
165         if (~newGame_L)
166           nextState = C_1_5_9_H_4_6_W_N;
167         else
168           nextState = C_1_5_9_H_4_6_W;
169       C_1_5_9_H_6_7_W:
170         if (~newGame_L)
171           nextState = C_1_5_9_H_6_7_W_N;
172         else
173           nextState = C_1_5_9_H_6_7_W;
174       C_1_5_9_H_6_8_W:
175         if (~newGame_L)
176           nextState = C_1_5_9_H_6_8_W_N;
177         else
178           nextState = C_1_5_9_H_6_8_W;
179       C_1_3_5_7_H_2_6_9_W:
180         if (~newGame_L)
181           nextState = C_1_3_5_7_H_2_6_9_W_N;
182         else
183           nextState = C_1_3_5_7_H_2_6_9_W;
184       C_1_2_3_5_H_4_6_9_W:
185         if (~newGame_L)
186           nextState = C_1_2_3_5_H_4_6_9_W_N;
187         else
188           nextState = C_1_2_3_5_H_4_6_9_W;
189       C_1_2_3_5_H_6_7_9_W:
190         if (~newGame_L)
191           nextState = C_1_2_3_5_H_6_7_9_W_N;
192         else
193           nextState = C_1_2_3_5_H_6_7_9_W;
194       C_1_2_3_5_H_6_8_9_W:
195         if (~newGame_L)
196           nextState = C_1_2_3_5_H_6_8_9_W_N;
197         else
198           nextState = C_1_2_3_5_H_6_8_9_W;
199
200       C_5_I:
201         if (enter_L)
202           nextState = C_5;
203         else
204           nextState = C_5_I;
205       C_1_5_H_6_I:
206         if (enter_L)
207           nextState = C_1_5_H_6;
208         else
209           nextState = C_1_5_H_6_I;
210       C_1_3_5_H_6_9_I:
211         if (enter_L)
```

```
212           nextState = C_1_3_5_H_6_9;
213         else
214           nextState = C_1_3_5_H_6_9_I;
215
216       C_1_5_9_H_2_6_W_N:
217         if (newGame_L)
218           nextState = C_5;
219         else
220           nextState = C_1_5_9_H_2_6_W_N;
221       C_1_5_9_H_3_6_W_N:
222         if (newGame_L)
223           nextState = C_5;
224         else
225           nextState = C_1_5_9_H_3_6_W_N;
226       C_1_5_9_H_4_6_W_N:
227         if (newGame_L)
228           nextState = C_5;
229         else
230           nextState = C_1_5_9_H_4_6_W_N;
231       C_1_5_9_H_6_7_W_N:
232         if (newGame_L)
233           nextState = C_5;
234         else
235           nextState = C_1_5_9_H_6_7_W_N;
236       C_1_5_9_H_6_8_W_N:
237         if (newGame_L)
238           nextState = C_5;
239         else
240           nextState = C_1_5_9_H_6_8_W_N;
241       C_1_2_3_5_H_4_6_9_W_N:
242         if (newGame_L)
243           nextState = C_5;
244         else
245           nextState = C_1_2_3_5_H_4_6_9_W_N;
246       C_1_2_3_5_H_6_7_9_W_N:
247         if (newGame_L)
248           nextState = C_5;
249         else
250           nextState = C_1_2_3_5_H_6_7_9_W_N;
251       C_1_2_3_5_H_6_8_9_W_N:
252         if (newGame_L)
253           nextState = C_5;
254         else
255           nextState = C_1_2_3_5_H_6_8_9_W_N;
256       C_1_3_5_7_H_2_6_9_W_N:
257         if (newGame_L)
258           nextState = C_5;
259         else
260           nextState = C_1_3_5_7_H_2_6_9_W_N;
261     endcase
262   end
263
264
265
266 always_comb begin
267     cMove = 4'b0000;
268     win = 1'b1;
269     {c3, c2, c1, c0, h3, h2, h1, h0} = 32'b0;
270     if (currState == C_5) begin
271       cMove = 4'h5;
272       win = 0;
273       c3 = 4'd5;
274     end
275
276     if (currState == C_5_I) begin
277       cMove = 4'h5;
278       win = 0;
279       c3 = 4'd5;
280     end
281
282     if (currState == C_5_H_6_E) begin
```

```
283       cMove = 4'h5;
284       win = 0;
285       c3 = 4'd5;
286     end
287
288     if (currState == C_1_5_H_6) begin
289       cMove = 4'h1;
290       win = 0;
291       c3 = 4'd1;
292       c2 = 4'd5;
293       h3 = 4'd6;
294     end
295
296     if (currState == C_1_5_H_6_I) begin
297       cMove = 4'h1;
298       win = 0;
299       c3 = 4'd1;
300       c2 = 4'd5;
301       h3 = 4'd6;
302     end
303
304
305     if (currState == C_1_5_H_2_6_E) begin
306       cMove = 4'h1;
307       win = 0;
308       c3 = 4'd1;
309       c2 = 4'd5;
310       h3 = 4'd6;
311     end
312     if (currState == C_1_5_H_3_6_E) begin
313       cMove = 4'h1;
314       win = 0;
315       c3 = 4'd1;
316       c2 = 4'd5;
317       h3 = 4'd6;
318     end
319     if (currState == C_1_5_H_4_6_E) begin
320       cMove = 4'h1;
321       win = 0;
322       c3 = 4'd1;
323       c2 = 4'd5;
324       h3 = 4'd6;
325     end
326     if (currState == C_1_5_H_6_7_E) begin
327       cMove = 4'h1;
328       win = 0;
329       c3 = 4'd1;
330       c2 = 4'd5;
331       h3 = 4'd6;
332     end
333     if (currState == C_1_5_H_6_8_E) begin
334       cMove = 4'h1;
335       win = 0;
336       c3 = 4'd1;
337       c2 = 4'd5;
338       h3 = 4'd6;
339     end
340     if (currState == C_1_5_H_6_9_E) begin
341       cMove = 4'h1;
342       win = 0;
343       c3 = 4'd1;
344       c2 = 4'd5;
345       h3 = 4'd6;
346     end
347
348
349     if (currState == C_1_5_9_H_2_6_W) begin
350       cMove = 4'h7;
351       win = 1;
352       c3 = 4'd1;
353       c2 = 4'd5;
```

```
354       c1 = 4'd9;
355       h3 = 4'd2;
356       h2 = 4'd6;
357     end
358     if (currState == C_1_5_9_H_2_6_W_N) begin
359       cMove = 4'h9;
360       win = 1;
361       c3 = 4'd1;
362       c2 = 4'd5;
363       c1 = 4'd9;
364       h3 = 4'd2;
365       h2 = 4'd6;
366     end
367
368     if (currState == C_1_5_9_H_3_6_W) begin
369       cMove = 4'h9;
370       win = 1;
371       c3 = 4'd1;
372       c2 = 4'd5;
373       c1 = 4'd7;
374       h3 = 4'd3;
375       h2 = 4'd6;
376     end
377     if (currState == C_1_5_9_H_3_6_W_N) begin
378       cMove = 4'h9;
379       win = 1;
380       c3 = 4'd1;
381       c2 = 4'd5;
382       c1 = 4'd9;
383       h3 = 4'd3;
384       h2 = 4'd6;
385     end
386
387     if (currState == C_1_5_9_H_4_6_W) begin
388       cMove = 4'h9;
389       win = 1;
390       c3 = 4'd1;
391       c2 = 4'd5;
392       c1 = 4'd9;
393       h3 = 4'd4;
394       h2 = 4'd6;
395     end
396     if (currState == C_1_5_9_H_4_6_W_N) begin
397       cMove = 4'h9;
398       win = 1;
399       c3 = 4'd1;
400       c2 = 4'd5;
401       c1 = 4'd9;
402       h3 = 4'd4;
403       h2 = 4'd6;
404     end
405
406     if (currState == C_1_5_9_H_6_7_W) begin
407       cMove = 4'h9;
408       win = 1;
409       c3 = 4'd1;
410       c2 = 4'd5;
411       c1 = 4'd9;
412       h3 = 4'd6;
413       h2 = 4'd7;
414     end
415     if (currState == C_1_5_9_H_6_7_W_N) begin
416       cMove = 4'h9;
417       win = 1;
418       c3 = 4'd1;
419       c2 = 4'd5;
420       c1 = 4'd9;
421       h3 = 4'd6;
422       h2 = 4'd7;
423     end
424
```

```
425
426
427     if (currState == C_1_5_9_H_6_8_W) begin
428       cMove = 4'h9;
429       win = 1;
430       c3 = 4'd1;
431       c2 = 4'd5;
432       c1 = 4'd9;
433       h3 = 4'd6;
434       h2 = 4'd8;
435     end
436     if (currState == C_1_5_9_H_6_8_W_N) begin
437       cMove = 4'h9;
438       win = 1;
439       c3 = 4'd1;
440       c2 = 4'd5;
441       c1 = 4'd9;
442       h3 = 4'd6;
443       h2 = 4'd8;
444     end
445
446
447     if (currState == C_1_3_5_H_6_9) begin
448       cMove = 4'h3;
449       win = 0;
450       c3 = 4'd1;
451       c2 = 4'd3;
452       c1 = 4'd5;
453       h3 = 4'd6;
454       h2 = 4'd9;
455     end
456     if (currState == C_1_3_5_H_6_9_I) begin
457       cMove = 4'h3;
458       win = 0;
459       c3 = 4'd1;
460       c2 = 4'd3;
461       c1 = 4'd5;
462       h3 = 4'd6;
463       h2 = 4'd9;
464     end
465
466     if (currState == C_1_3_5_H_2_6_9_E) begin
467       cMove = 4'h3;
468       win = 0;
469       c3 = 4'd1;
470       c2 = 4'd3;
471       c1 = 4'd5;
472       h3 = 4'd6;
473       h2 = 4'd9;
474     end
475
476     if (currState == C_1_3_5_H_4_6_9_E) begin
477       cMove = 4'h3;
478       win = 0;
479       c3 = 4'd1;
480       c2 = 4'd3;
481       c1 = 4'd5;
482       h3 = 4'd6;
483       h2 = 4'd9;
484     end
485
486     if (currState == C_1_3_5_H_6_7_9_E) begin
487       cMove = 4'h3;
488       win = 0;
489       c3 = 4'd1;
490       c2 = 4'd3;
491       c1 = 4'd5;
492       h3 = 4'd6;
493       h2 = 4'd9;
494     end
495
```

```
496     if (currState == C_1_3_5_H_6_8_9_E) begin
497       cMove = 4'h3;
498       win = 0;
499       c3 = 4'd1;
500       c2 = 4'd3;
501       c1 = 4'd5;
502       h3 = 4'd6;
503       h2 = 4'd9;
504     end
505
506     if (currState == C_1_2_3_5_H_4_6_9_W) begin
507       cMove = 4'h2;
508       win = 1;
509       c3 = 4'd1;
510       c2 = 4'd2;
511       c1 = 4'd3;
512       c0 = 4'd5;
513       h3 = 4'd4;
514       h2 = 4'd6;
515       h1 = 4'd9;
516     end
517
518     if (currState == C_1_2_3_5_H_4_6_9_W_N) begin
519       cMove = 4'h2;
520       win = 1;
521       c3 = 4'd1;
522       c2 = 4'd2;
523       c1 = 4'd3;
524       c0 = 4'd5;
525       h3 = 4'd4;
526       h2 = 4'd6;
527       h1 = 4'd9;
528     end
529
530     if (currState == C_1_2_3_5_H_6_7_9_W) begin
531       cMove = 4'h2;
532       win = 1;
533       c3 = 4'd1;
534       c2 = 4'd2;
535       c1 = 4'd3;
536       c0 = 4'd5;
537       h3 = 4'd6;
538       h2 = 4'd7;
539       h1 = 4'd9;
540     end
541
542     if (currState == C_1_2_3_5_H_6_7_9_W_N) begin
543       cMove = 4'h2;
544       win = 1;
545       c3 = 4'd1;
546       c2 = 4'd2;
547       c1 = 4'd3;
548       c0 = 4'd5;
549       h3 = 4'd6;
550       h2 = 4'd7;
551       h1 = 4'd9;
552
553     end
554
555     if (currState == C_1_2_3_5_H_6_8_9_W) begin
556       cMove = 4'h2;
557       win = 1;
558       c3 = 4'd1;
559       c2 = 4'd2;
560       c1 = 4'd3;
561       c0 = 4'd5;
562       h3 = 4'd6;
563       h2 = 4'd8;
564       h1 = 4'd9;
565     end
566
```

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

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

```
`default_nettype none // Required in every sv file
module dFlipFlopcI(
  output logic q,
  input  logic d, clock, reset);

  always_ff @(posedge clock)
    if (reset == 1'b1)
      q <= 0;
    else
      q <= d;

endmodule : dFlipFlopcI

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),
        .h2(h_2),
        .h1(h_1),
        .h0(h_0),
        .c3(c_3),
        .c2(c_2),
        .c1(c_1),
        .c0(c_0),
        .enter_L(KEY[3]),
        .reset(SW[17]),
        .hMove(SW[3:0]),
        .cMove(LEDR[3:0]),
        .win(ww));

  assign LEDG = {ww, ww, ww, ww, ww, ww, ww, ww};
  logic [7:0] blank;
  assign blank = 8'b00000000;
  dFlipFlopcI ff0(.d(KEY[3]),
                  .q(enter_L),
                  .clock(CLOCK_50),
                  .reset(SW[17])),
              ff1(.d(KEY[0]),
                  .q(newGame_L),
                  .clock(CLOCK_50),
                  .reset(SW[17]));



  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


```