Lab Code [15 points]
Filename: Lab3_abstract_and_explicit.sv
AndrewID: jtbell

```
// Testbench and code for MyExplicitFSM and original MYAbstractFSM


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

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

  // S0-S5 are the different states for the FSM in binary states  0000 - 0101
  enum logic [2:0] {S0, S1, S2,S3,S4,S5} currState, nextState;
  always_comb begin
    case (currState)
      S0: begin
        if(hMove == 4'h9)
            nextState = S1;
        else
          nextState=S0;
    end
      S1: begin
        if (hMove == 4'h4)
            nextState = S2;
        else if(hMove == 4'h1|hMove==4'h2|
                hMove==4'h3|hMove == 4'h7|hMove ==4'h8) begin
            nextState = S3;
            end
        else
            nextState = S1;
    end
      S2: begin
        if (hMove==4'h7|hMove == 4'h3|hMove == 4'h1)
            nextState = S4;
        else if (hMove==4'h8)
            nextState = S5;
        else
            nextState = S2;
    end
    S3: begin
      nextState = S3;
    end
    S4: begin
      nextState = S4;
    end
    S5: begin
      nextState = S5;
    end
    default: begin
      nextState = S0;
    end
  endcase
end
// Output logic defined here. You are basically transcribing
// the output column of the state transition table into a
// SystemVerilog case statement.
// Remember, if this is a Moore machine, this logic should only
// depend on the current state. Mealy also involves inputs.
always_comb begin
  cMove = 4'b0000; win = 1'b0;
```

```
 70   unique case (currState)
 71     S0: cMove = 4'b0101;
 72     S1: begin
 73           win = 1'b0;
 74           cMove = 4'b0110;
 75         end
 76     S2: begin
 77           win = 1'b0;
 78           cMove = 4'b0010;
 79         end
 80     S3: begin
 81           win = 1'b1;
 82           cMove = 4'b0100;
 83         end
 84     S4: begin
 85           win = 1'b1;
 86           cMove = 4'b1000;
 87         end
 88     S5: begin
 89           win = 1'b1;
 90           cMove = 4'b0111;
 91         end
 92     // ...
 93     // one case for each state in your FSM
 94     // ...
 95     // no default statement needed, due to unique case
 96   endcase
 97 end
 98 // Synchronous state update described here as an always_ff block.
 99 // In essence, these are your flip flops that will hold the state
100 // This doesn't do anything interesting except to capture the new
101 // state value on each clock edge. Also, synchronous reset.
102 always_ff @(posedge clock)
103   if (reset)
104     currState <= S0; // or whatever the reset state is
105   else
106     currState <= nextState;
107 endmodule: myAbstractFSM
108
109 module myExplicitFSM(
110   output logic [3:0] cMove,
111   output logic win,
112   output logic q0, q1, q2, // connect to FF outputs(add more if needed)
113   input logic [3:0] hmove,
114   input logic clock, reset);
115
116   logic d0, d1, d2; // connect to FF inputs (add more if needed)
117   // Example instantiation of D-flip-flop.
118   // Add more as necessary.
119   dFlipFlop ff0(.d(d0), .q(q0), .*),
120             ff1(.d(d1), .q(q1), .*),
121             ff2(.d(d2), .q(q2), .*);
122   // Next state logic goes here: combinational logic that drives
123   // next state (d0, etc) based upon input hMove and the
124   // current state (q0, q1, etc).
125
126   // Setting a NOt valid state
127   assign isnotvalid = ~hmove[3] &  ~hmove[2] & ~hmove[1] & ~hmove[0] | //0
128                       ~hmove[3] & hmove[2] & ~hmove[1] & hmove[0]|//5
129                       ~hmove[3] & hmove[2] & hmove[1] & ~hmove[0]|//6
130                       hmove[3] & ~hmove[2] & ~hmove[1] & hmove[0]|//9
131                       hmove[3] & ~hmove[2] & hmove[1] & ~hmove[0]|//10
132                       hmove[3] & ~hmove[2] & hmove[1] & hmove[0]|//11
133                       hmove[3] & hmove[2]; //12-15
134
135 // giving all the hmoves a number so it is easier to type later
136   assign one= ~hmove[3] & ~hmove[2] & ~hmove[1] & hmove[0];//1
137   assign two = ~hmove[3] & ~hmove[2] & hmove[1] & ~hmove[0];//2
138   assign three= ~hmove[3] & ~hmove[2] & hmove[1] & hmove[0];//3
139   assign four= ~hmove[3] & hmove[2] & ~hmove[1] & ~hmove[0];//4
140   assign five= ~hmove[3] & hmove[2] & ~hmove[1] & hmove[0];//5
```

```
141   assign six=  ~hmove[3] & hmove[2] & hmove[1] & ~hmove[0];//6
142   assign seven = ~hmove[3] & hmove[2] & hmove[1] & hmove[0];//7
143   assign eight = hmove[3] & ~hmove[2] & ~hmove[1] & ~hmove[0];//8
144   assign nine= hmove[3] & ~hmove[2] & ~hmove[1] & hmove[0];//9
145
146   // Transitions to different states
147   assign s0tos1 = ~q1&~q2&~q0 & (nine); // 6
148   assign s1tos2= ~q1&~q2&q0 & (four); // 2
149   assign s1tos3= ~q1&~q2&q0 & ~isnotvalid & ~(four); //4
150   assign s2tos5= ~q2&q1&~q0 & eight ; //  7
151   assign state2to4 = ~q2&q1&~q0 & (one | three | seven);//8
152
153  // Reset states where it resets within itself
154   assign s1tos1 = ~q2 & ~q1 & q0 & isnotvalid;
155   assign s2tos2 = ~q2&q1&~q0 & (isnotvalid | four) ;
156   assign s3tos3 = ~q2&q1&q0;
157   assign state4to4 =  q2&~q1&~q0;
158   assign state5to5 =  q2&~q1&q0;
159
160 // assigning the next states
161   assign d0= s0tos1|s1tos3|s2tos5|state5to5|s3tos3|s1tos1;
162   assign d1= s1tos2|s3tos3|s1tos3|s2tos2;
163   assign d2= state4to4|state5to5|s2tos5|state2to4;
164
165 // computer moves based on the states/ inputs
166   assign cMove[0] = ~q0&~q1&~q2 | q0&~q1&q2; //0 and 5
167   assign cMove[1] = q0&~q1&q2 | q0&~q1&~q2 | ~q0&q1&~q2; //,5,2,1
168   assign cMove[2] = q0&~q1&~q2 | q0&q1&~q2| ~q1&~q2&~q0| q0&~q1&q2; //0,3,5,1,
169   assign cMove[3] = q2&~q1&~q0;
170   assign win = (q2 & ~q1 & ~q0) | (q2 & ~q1 & q0) | (~q2 & q1 & q0);// 4,5,3
171
172   // Your output logic goes here: combinational logic that
173   // drives cMove and win based upon
174   // current state (q0, etc) and hMove.
175 endmodule: myExplicitFSM
176
177 // //// Explicit  and abstract FSM Test Bench Tests Everything*(transitions)
178
179 // update begining to this for Abstract FSM TEstbench
180 // logic [3:0] cMove;
181 //   logic win;
182 //   logic q2, q1, q0;
183 //   logic [3:0] hMove;
184 //   logic clock, reset;
185
186 //   myAbstractFSM f1(.*);
187
188 //   initial begin
189 //     clock = 0;
190 //     forever #5 clock = ~clock;
191 //   end
192 //   initial begin
193 //     $monitor($time,, "currState=%s, cMove=%d, hMove=%d, win=%b",
194 //                f1.currState.name, cMove, hMove, win);
195 //   // initialize values
196 //   hMove <= 4'h4; reset <= 1'b1;
197
198 // Explicit FSM Test Bench
199 // // module myFSM_test();
200 // //   logic [3:0] cMove;
201 // //   logic win;
202 // //   logic q2, q1, q0;
203 // //   logic [3:0] hmove;
204 // //   logic clock, reset;
205
206 // //   myExplicitFSM f1(.*);
207
208 // //   initial begin
209 // //     clock = 0;
210 // //     forever #5 clock = ~clock;
211 // //   end
```

```
212 // //   initial begin
213 // //     $monitor($time,, "state=%d, cMove=%d, hMove=%d, win=%b",
214 // //                {q2, q1, q0}, cMove, hmove, win);
215 // //   // initialize values
216 // //   hmove <= 4'h4; reset <= 1'b1;
217 // //   // reset the FSM
218 
219 // //   // win 9,4,7
220 // //   @(posedge clock); // wait for a positive clock edge
221 // //   @(posedge clock); // one edge is enough, but what the heck
222 // //   @(posedge clock);
223 // //   @(posedge clock); // begin cycle 0
224 // //   reset <= 1'b0; // release the reset
225 // //   // start an example sequence -- not meaningful for the lab
226 // //   hmove <= 4'h4; // these changes are after the clock edge
227 // //                  // which means the state change happens
228 // //                  // AFTER the next clock edge
229 // //   @(posedge clock); // begin cycle 1
230 // //   hmove <= 9'h9;
231 // //   @(posedge clock); // begin cycle 2
232 // //   hmove <= 4'h4;
233 // //   @(posedge clock)
234 // //   hmove <= 7'h7;
235 // //   // could check FSM outputs like so. Be careful about timing
236 // //   @(posedge clock);
237 
238 // // // win 9,1
239 // //   reset <= 1'b1;
240 // //   @(posedge clock); // wait for a positive clock edge
241 // //   @(posedge clock); // one edge is enough, but what the heck
242 // //   @(posedge clock);
243 
244 // //   @(posedge clock); // begin cycle 0
245 // //   reset <= 1'b0; // release the reset
246 // //   // start an example sequence -- not meaningful for the lab
247 // //   hmove <= 4'h4; // these changes are after the clock edge
248 // //                  // which means the state change happens
249 // //                  // AFTER the next clock edge
250 // //   @(posedge clock); // begin cycle 1
251 // //   hmove <= 4'h9;
252 // //   @(posedge clock); // begin cycle 2
253 // //   hmove <= 4'h1;
254 // //   // could check FSM outputs like so. Be careful about timing
255 // //   @(posedge clock);
256 
257 // // // stops working here
258 // // //Win 9,4,8
259 // //   reset <= 1'b1;
260 // //   @(posedge clock); // wait for a positive clock edge
261 // //   @(posedge clock); // one edge is enough, but what the heck
262 // //   @(posedge clock);
263 // //   @(posedge clock); // begin cycle 0
264 // //   reset <= 1'b0; // release the reset
265 // //   // start an example sequence -- not meaningful for the lab
266 // //   hmove <= 4'h4; // these changes are after the clock edge
267 // //                  // which means the state change happens
268 // //                  // AFTER the next clock edge
269 // //   @(posedge clock); // begin cycle 1
270 // //   hmove <= 4'h9;
271 // //   @(posedge clock); // begin cycle 2
272 // //   hmove <= 4'h4;
273 // //   @(posedge clock); // begin cycle 2
274 // //   hmove <= 4'h8;
275 // //   // could check FSM outputs like so. Be careful about timing
276 // //   @(posedge clock);
277 
278 // // //win 9,4,1
279 // //   reset<=1'b1;
280 // //   @(posedge clock); // wait for a positive clock edge
281 // //   @(posedge clock); // one edge is enough, but what the heck
282 // //   @(posedge clock);
```

```
283 // //   @(posedge clock); // begin cycle 0
284 // //   reset <= 1'b0; // release the reset
285 // //   // start an example sequence -- not meaningful for the lab
286 // //   hmove <= 4'h4; // these changes are after the clock edge
287 // //                  // which means the state change happens
288 // //                  // AFTER the next clock edge
289 // //   @(posedge clock); // begin cycle 1
290 // //   hmove <= 4'h9;
291 // //   @(posedge clock); // begin cycle 2
292 // //   hmove <= 4'h4;
293 // //   @(posedge clock); // begin cycle 2
294 // //   hmove <= 4'h1;
295 // //   // could check FSM outputs like so. Be careful about timing
296 // //   @(posedge clock);
297
298 // // //Win 9,4,2
299 // //   reset<=1'b1;
300 // //   @(posedge clock); // wait for a positive clock edge
301 // //   @(posedge clock); // one edge is enough, but what the heck
302 // //   @(posedge clock);
303 // //   @(posedge clock); // begin cycle 0
304 // //   reset <= 1'b0; // release the reset
305 // //   // start an example sequence -- not meaningful for the lab
306 // //   hmove <= 4'h4; // these changes are after the clock edge
307 // //                  // which means the state change happens
308 // //                  // AFTER the next clock edge
309 // //   @(posedge clock); // begin cycle 1
310 // //   hmove <= 4'h9;
311 // //   @(posedge clock); // begin cycle 2
312 // //   hmove <= 4'h4;
313 // //   @(posedge clock); // begin cycle 2
314 // //   hmove <= 4'h2;
315 // //   // could check FSM outputs like so. Be careful about timing
316 // //   @(posedge clock);
317
318
319 // // // Win  9,4,3
320 // //   reset<=1'b1;
321 // //   @(posedge clock); // wait for a positive clock edge
322 // //   @(posedge clock); // one edge is enough, but what the heck
323 // //   @(posedge clock);
324 // //   @(posedge clock); // begin cycle 0
325 // //   reset <= 1'b0; // release the reset
326 // //   // start an example sequence -- not meaningful for the lab
327 // //   hmove <= 4'h4; // these changes are after the clock edge
328 // //                  // which means the state change happens
329 // //                  // AFTER the next clock edge
330 // //   @(posedge clock); // begin cycle 1
331 // //   hmove <= 4'h9;
332 // //   @(posedge clock); // begin cycle 2
333 // //   hmove <= 4'h4;
334 // //   @(posedge clock); // begin cycle 2
335 // //   hmove <= 4'h3;
336 // //   // could check FSM outputs like so. Be careful about timing
337 // //   @(posedge clock);
338
339 // // // win 9,4,4 and stay in place
340 // //   reset<=1'b1;
341 // //   @(posedge clock); // wait for a positive clock edge
342 // //   @(posedge clock); // one edge is enough, but what the heck
343 // //   @(posedge clock);
344 // //   @(posedge clock); // begin cycle 0
345 // //   reset <= 1'b0; // release the reset
346 // //   // start an example sequence -- not meaningful for the lab
347 // //   hmove <= 4'h4; // these changes are after the clock edge
348 // //                  // which means the state change happens
349 // //                  // AFTER the next clock edge
350 // //   @(posedge clock); // begin cycle 1
351 // //   hmove <= 4'h9;
352 // //   @(posedge clock); // begin cycle 2
353 // //   hmove <= 4'h4;
```

```
354 // //   @(posedge clock); // begin cycle 2
355 // //   hmove <= 4'h4;
356 // //   // could check FSM outputs like so. Be careful about timing
357 // //   @(posedge clock);
358
359
360 // // //Test 8 goes to Win value 9,4,5
361 // //   reset<=1'b1;
362 // //   @(posedge clock); // wait for a positive clock edge
363 // //   @(posedge clock); // one edge is enough, but what the heck
364 // //   @(posedge clock);
365 // //   @(posedge clock); // begin cycle 0
366 // //   reset <= 1'b0; // release the reset
367 // //   // start an example sequence -- not meaningful for the lab
368 // //   hmove <= 4'h4; // these changes are after the clock edge
369 // //                  // which means the state change happens
370 // //                  // AFTER the next clock edge
371 // //   @(posedge clock); // begin cycle 1
372 // //   hmove <= 4'h9;
373 // //   @(posedge clock); // begin cycle 2
374 // //   hmove <= 4'h4;
375 // //   @(posedge clock); // begin cycle 2
376 // //   hmove <= 4'h5;
377 // //   // could check FSM outputs like so. Be careful about timing
378 // //   @(posedge clock);
379
380
381 // //   //Win 9,4,8
382 // //   reset <= 1'b1;
383 // //   @(posedge clock); // wait for a positive clock edge
384 // //   @(posedge clock); // one edge is enough, but what the heck
385 // //   @(posedge clock);
386 // //   @(posedge clock); // begin cycle 0
387 // //   reset <= 1'b0; // release the reset
388 // //   // start an example sequence -- not meaningful for the lab
389 // //   hmove <= 4'h4; // these changes are after the clock edge
390 // //                  // which means the state change happens
391 // //                  // AFTER the next clock edge
392 // //   @(posedge clock); // begin cycle 1
393 // //   hmove <= 4'h1;
394 // //   // could check FSM outputs like so. Be careful about timing
395 // //   @(posedge clock); // begin cycle 1
396 // //   hmove <= 4'h9;
397 // //   @(posedge clock); // begin cycle 2
398 // //   hmove <= 4'h4;
399 // //   @(posedge clock); // begin cycle 3
400 // //   hmove <= 4'h8;
401 // //   // could check FSM outputs like so. Be careful about timing
402 // //   @(posedge clock);
403
404 // // //9,4,8,5 stay in place
405 // //   reset <= 1'b1;
406 // //   @(posedge clock); // wait for a positive clock edge
407 // //   @(posedge clock); // one edge is enough, but what the heck
408 // //   @(posedge clock);
409
410 // //   @(posedge clock); // begin cycle 0
411 // //   reset <= 1'b0; // release the reset
412 // //   // start an example sequence -- not meaningful for the lab
413 // //   hmove <= 4'h4; // these changes are after the clock edge
414 // //                  // which means the state change happens
415 // //                  // AFTER the next clock edge
416 // //   @(posedge clock); // begin cycle 1
417 // //   hmove <= 4'h1;
418 // //   // could check FSM outputs like so. Be careful about timing
419 // //   @(posedge clock); // begin cycle 1
420 // //   hmove <= 4'h9;
421 // //   @(posedge clock); // begin cycle 2
422 // //   hmove <= 4'h4;
423 // //   @(posedge clock); // begin cycle 3
424 // //   hmove <= 4'h8;
```

```
425 // //   @(posedge clock); // begin cycle 4
426 // //   hmove <= 4'h5;
427 // //   // could check FSM outputs like so. Be careful about timing
428 // //   @(posedge clock);
429
430
431 // // //9,4,7,5 stay in place
432 // //   reset <= 1'b1;
433 // //   @(posedge clock); // wait for a positive clock edge
434 // //   @(posedge clock); // one edge is enough, but what the heck
435 // //   @(posedge clock);
436
437 // //   @(posedge clock); // begin cycle 0
438 // //   reset <= 1'b0; // release the reset
439 // //   // start an example sequence -- not meaningful for the lab
440 // //   hmove <= 4'h4; // these changes are after the clock edge
441 // //                  // which means the state change happens
442 // //                  // AFTER the next clock edge
443 // //   @(posedge clock); // begin cycle 1
444 // //   hmove <= 4'h1;
445 // //   // could check FSM outputs like so. Be careful about timing
446 // //   @(posedge clock); // begin cycle 1
447 // //   hmove <= 4'h9;
448 // //   @(posedge clock); // begin cycle 2
449 // //   hmove <= 4'h4;
450 // //   @(posedge clock); // begin cycle 3
451 // //   hmove <= 4'h7;
452 // //   @(posedge clock); // begin cycle 4
453 // //   hmove <= 4'h5;
454 // //   // could check FSM outputs like so. Be careful about timing
455 // //   @(posedge clock);
456
457
458 // //   // 9,4,7 Reset
459 // //   reset <= 1'b1;
460 // //   @(posedge clock); // wait for a positive clock edge
461 // //   @(posedge clock); // one edge is enough, but what the heck
462 // //   @(posedge clock);
463 // //   @(posedge clock); // begin cycle 0
464 // //   reset <= 1'b0; // release the reset
465 // //   // start an example sequence -- not meaningful for the lab
466 // //   hmove <= 4'h4; // these changes are after the clock edge
467 // //                  // which means the state change happens
468 // //                  // AFTER the next clock edge
469 // //   @(posedge clock); // begin cycle 1
470 // //   hmove <= 4'h1;
471 // //   // could check FSM outputs like so. Be careful about timing
472 // //   @(posedge clock); // begin cycle 1
473 // //   hmove <= 4'h9;
474 // //   @(posedge clock); // begin cycle 2
475 // //   hmove <= 4'h4;
476 // //   @(posedge clock); // begin cycle 3
477 // //   hmove <= 4'h7;
478 // //   @(posedge clock); // begin cycle 4
479 // //   hmove <= reset;
480 // //   // could check FSM outputs like so. Be careful about timing
481 // //   @(posedge clock);
482
483
484 // //   // 9,4 Reset
485 // //   reset <= 1'b1;
486 // //   @(posedge clock); // wait for a positive clock edge
487 // //   @(posedge clock); // one edge is enough, but what the heck
488 // //   @(posedge clock);
489 // //   @(posedge clock); // begin cycle 0
490 // //   reset <= 1'b0; // release the reset
491 // //   // start an example sequence -- not meaningful for the lab
492 // //   hmove <= 4'h4; // these changes are after the clock edge
493 // //                  // which means the state change happens
494 // //                  // AFTER the next clock edge
495 // //   @(posedge clock); // begin cycle 1
```

```
496 // //   hmove <= 4'h9;
497 // //   @(posedge clock); // begin cycle 2
498 // //   hmove <= 4'h4;
499 // //   @(posedge clock); // begin cycle 4
500 // //   hmove <= reset;
501 // //   // could check FSM outputs like so. Be careful about timing
502 // //   @(posedge clock);
503 
504 // //   // 9,4,8 Reset
505 // //   reset <= 1'b1;
506 // //   @(posedge clock); // wait for a positive clock edge
507 // //   @(posedge clock); // one edge is enough, but what the heck
508 // //   @(posedge clock);
509 // //   @(posedge clock); // begin cycle 0
510 // //   reset <= 1'b0; // release the reset
511 // //   // start an example sequence -- not meaningful for the lab
512 // //   hmove <= 4'h4; // these changes are after the clock edge
513 // //                  // which means the state change happens
514 // //                  // AFTER the next clock edge
515 // //   @(posedge clock); // begin cycle 1
516 // //   hmove <= 4'h9;
517 // //   @(posedge clock); // begin cycle 2
518 // //   hmove <= 4'h4;
519 // //   @(posedge clock);
520 // //   hmove<=4'h8;
521 // //   @(posedge clock); // begin cycle 4
522 // //   hmove <= reset;
523 // //   // could check FSM outputs like so. Be careful about timing
524 // //   @(posedge clock);
525 
526 // //   // 9,4,random reset
527 // //   reset <= 1'b1;
528 // //   @(posedge clock); // wait for a positive clock edge
529 // //   @(posedge clock); // one edge is enough, but what the heck
530 // //   @(posedge clock);
531 // //   @(posedge clock); // begin cycle 0
532 // //   reset <= 1'b0; // release the reset
533 // //   // start an example sequence -- not meaningful for the lab
534 // //   hmove <= 4'h4; // these changes are after the clock edge
535 // //                  // which means the state change happens
536 // //                  // AFTER the next clock edge
537 // //   @(posedge clock); // begin cycle 1
538 // //   hmove <= 4'h9;
539 // //   @(posedge clock); // begin cycle 2
540 // //   hmove <= 4'h4;
541 // //   @(posedge clock);
542 // //   hmove<=4'h1;
543 // //   @(posedge clock); // begin cycle 4
544 // //   hmove <= reset;
545 // //   // could check FSM outputs like so. Be careful about timing
546 // //   @(posedge clock);
547 
548 // //   // 9,7 reset
549 // //   reset <= 1'b1;
550 // //   @(posedge clock); // wait for a positive clock edge
551 // //   @(posedge clock); // one edge is enough, but what the heck
552 // //   @(posedge clock);
553 // //   @(posedge clock); // begin cycle 0
554 // //   reset <= 1'b0; // release the reset
555 // //   // start an example sequence -- not meaningful for the lab
556 // //   hmove <= 4'h4; // these changes are after the clock edge
557 // //                  // which means the state change happens
558 // //                  // AFTER the next clock edge
559 // //   // could check FSM outputs like so. Be careful about timing
560 // //   @(posedge clock); // begin cycle 1
561 // //   hmove <= 4'h9;
562 // //   @(posedge clock); // begin cycle 2
563 // //   hmove <= 4'h7;
564 // //   @(posedge clock); // begin cycle 4
565 // //   hmove <= reset;
566 // //   // could check FSM outputs like so. Be careful about timing
```

```
567 // //   @(posedge clock);
568
569 // //   // 9 reset
570 // //   reset <= 1'b1;
571 // //   @(posedge clock); // wait for a positive clock edge
572 // //   @(posedge clock); // one edge is enough, but what the heck
573 // //   @(posedge clock);
574 // //   @(posedge clock); // begin cycle 0
575 // //   reset <= 1'b0; // release the reset
576 // //   // start an example sequence -- not meaningful for the lab
577 // //   hmove <= 4'h4; // these changes are after the clock edge
578 // //                  // which means the state change happens
579 // //                  // AFTER the next clock edge
580 // //   // could check FSM outputs like so. Be careful about timing
581 // //   @(posedge clock); // begin cycle 1
582 // //   hmove <= 4'h9;
583 // //   @(posedge clock); // begin cycle 4
584 // //   hmove <= reset;
585 // //   // could check FSM outputs like so. Be careful about timing
586 // //   @(posedge clock);
587
588 // //   // 1 stay in place
589 // //   reset <= 1'b1;
590 // //   @(posedge clock); // wait for a positive clock edge
591 // //   @(posedge clock); // one edge is enough, but what the heck
592 // //   @(posedge clock);
593 // //   @(posedge clock); // begin cycle 0
594 // //   reset <= 1'b0; // release the reset
595 // //   // start an example sequence -- not meaningful for the lab
596 // //   hmove <= 4'h4; // these changes are after the clock edge
597 // //                  // which means the state change happens
598 // //                  // AFTER the next clock edge
599
600 // //   @(posedge clock); // begin cycle 1
601 // //   hmove <= 4'h1;
602 // //   @(posedge clock);
603
604
605 // //   //9,9 stayin place
606 // //   reset <= 1'b1;
607 // //   @(posedge clock); // wait for a positive clock edge
608 // //   @(posedge clock); // one edge is enough, but what the heck
609 // //   @(posedge clock);
610 // //   @(posedge clock); // begin cycle 0
611 // //   reset <= 1'b0; // release the reset
612 // //   // start an example sequence -- not meaningful for the lab
613 // //   hmove <= 4'h4; // these changes are after the clock edge
614 // //                  // which means the state change happens
615 // //                  // AFTER the next clock edge
616 // //   // could check FSM outputs like so. Be careful about timing
617 // //   @(posedge clock); // begin cycle 1
618 // //   hmove <= 4'h9;
619 // //   @(posedge clock); // begin cycle 4
620 // //   hmove <= 4'h9;
621 // //   // could check FSM outputs like so. Be careful about timing
622 // //   @(posedge clock);
623
624
625 // //   //9,4,4 stayinplace
626 // //   reset <= 1'b1;
627 // //   @(posedge clock); // wait for a positive clock edge
628 // //   @(posedge clock); // one edge is enough, but what the heck
629 // //   @(posedge clock);
630 // //   @(posedge clock); // begin cycle 0
631 // //   reset <= 1'b0; // release the reset
632 // //   // start an example sequence -- not meaningful for the lab
633 // //   hmove <= 4'h4; // these changes are after the clock edge
634 // //                  // which means the state change happens
635 // //                  // AFTER the next clock edge
636 // //   // could check FSM outputs like so. Be careful about timing
637 // //   @(posedge clock); // begin cycle 1
```

```
638 // //   hmove <= 4'h9;
639 // //   @(posedge clock); // begin cycle 4
640 // //   hmove <= 4'h4;
641 // //   @(posedge clock); // begin cycle 4
642 // //   hmove <= 4'h4;
643 // //   // could check FSM outputs like so. Be careful about timing
644 // //   @(posedge clock);
645
646 // //   //9,3,3 stayinplace
647 // //   reset <= 1'b1;
648 // //   @(posedge clock); // wait for a positive clock edge
649 // //   @(posedge clock); // one edge is enough, but what the heck
650 // //   @(posedge clock);
651 // //   @(posedge clock); // begin cycle 0
652 // //   reset <= 1'b0; // release the reset
653 // //   // start an example sequence -- not meaningful for the lab
654 // //   hmove <= 4'h4; // these changes are after the clock edge
655 // //                  // which means the state change happens
656 // //                  // AFTER the next clock edge
657 // //   // could check FSM outputs like so. Be careful about timing
658 // //   @(posedge clock); // begin cycle 1
659 // //   hmove <= 4'h9;
660 // //   @(posedge clock); // begin cycle 4
661 // //   hmove <= 4'h3;
662 // //   @(posedge clock); // begin cycle 4
663 // //   hmove <= 4'h3;
664 // //   // could check FSM outputs like so. Be careful about timing
665 // //   @(posedge clock);
666
667 // //   //reset
668 // //   reset <= 1'b1;
669 // //   @(posedge clock); // wait for a positive clock edge
670 // //   @(posedge clock); // one edge is enough, but what the heck
671 // //   @(posedge clock);
672 // //   @(posedge clock); // begin cycle 0
673 // //   reset <= 1'b0; // release the reset
674 // //   // start an example sequence -- not meaningful for the lab
675 // //   hmove <= 4'h4; // these changes are after the clock edge
676 // //                  // which means the state change happens
677 // //                  // AFTER the next clock edge
678 // //   // could check FSM outputs like so. Be careful about timing
679 // //   @(posedge clock); // begin cycle 4
680 // //   hmove <= reset;
681 // //   // could check FSM outputs like so. Be careful about timing
682 // //   @(posedge clock);
683 // //   #1 $finish;
684
685 // // end
686 // // endmodule: myFSM_test
687
688
```

Lab Code [15 points]
Filename: Lab3_task5.sv
AndrewID: jtbell

```
`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

module myAbstractFSM (
  output logic [3:0] cMove,
  output logic [15:0] cHis, hHis,
  output logic win,
  input logic [3:0] hMove,
  input logic clock, reset, nenter, new_game);
  //not state 4 and not used and we use default encoding
  enum logic [4:0] {S0, S1, S2, S3, S4, S5, S6, S7,
  S8, S9, S10, S11, S12, S13, S14, S15, S16, buff1,
  buff2, buff3, buff4, buff5, buff6, buff7,buff8,buff9,
  Swin, Swin1}
  currState, nextState;
  logic isnotvalid;
  logic not_valid_when_win;
  logic one,two,three,four,five,six,seven,eight,nine;
  //defines what is not valid
  assign isnotvalid =   ~hMove[3] &  ~hMove[2] & ~hMove[1] & ~hMove[0] | //0
                        hMove[3] & ~hMove[2] & hMove[1] & ~hMove[0]|//10
                        hMove[3] & ~hMove[2] & hMove[1] & hMove[0]|//11
                        hMove[3] & hMove[2]; //12-15
  //defines one-nine to hMove
  assign one = ~hMove[3] & ~hMove[2] & ~hMove[1] & hMove[0]; //1
  assign two = ~hMove[3] & ~hMove[2] & hMove[1] & ~hMove[0]; //2
  assign three = ~hMove[3] & ~hMove[2] & hMove[1] & hMove[0];//3
  assign four = ~hMove[3] & hMove[2] & ~hMove[1] & ~hMove[0]; //4
  assign five = ~hMove[3] & hMove[2] & ~hMove[1] & hMove[0]; //5
  assign six =  ~hMove[3] & hMove[2] & hMove[1] & ~hMove[0]; //6
  assign seven = ~hMove[3] & hMove[2] & hMove[1] & hMove[0]; //7
  assign eight = hMove[3] & ~hMove[2] & ~hMove[1] & ~hMove[0]; //8
  assign nine =  hMove[3] & ~hMove[2] & ~hMove[1] & hMove[0];//9
  assign not_valid_when_win = (isnotvalid | one | two | three | four |
  five | six | seven | eight | nine);

  always_comb begin
  //currState = S0;
    nextState = currState;
    case(currState)
      S0:begin //S0, output=5, next_state = S6
        if (nenter) nextState = S6;
        else if (nenter & isnotvalid) nextState = S0;
        else nextState = S0;
      end
      S1:begin //S1, output=6, next_state = S7
        if(nenter) nextState=S7;
        else if (nenter & isnotvalid|five|nine|six) nextState = S1;
        else nextState = S1;
      end
      S2: begin //S2, output=2, next_state = S8
        if (nenter) nextState = S8;
        else if (nenter & isnotvalid|five|six|nine|four|two) nextState = S2;
        else nextState = S2;
      end
      S4: begin //S4, output=8, next_state = sWin
        if (new_game) nextState = buff8;
        else if (nenter & not_valid_when_win) nextState = S4;
        else nextState = S4;
      end
```

```
70        S5: begin //S5, output=7, next_state = sWin
71          if (new_game) nextState = buff9;
72          else if (nenter & not_valid_when_win) nextState = S5;
73          else nextState = S5;
74        end
75        //These states are the intermediate states
76        S6:begin
77          //between state 0 and state 1
78          if (~nenter & hMove == 4'h9) nextState = S1;
79          else if ( ~nenter & isnotvalid) nextState = S0; //S6
80          else nextState = S0;
81        end
82        S7:begin
83          //between state 1 and 2 cases on hMove
84          if (~nenter & hMove == 4'h4) nextState= S2;
85          else if ( ~nenter & hMove == 4'h1) nextState=S9;
86          else if ( ~nenter & hMove == 4'h2) nextState=S10;
87          else if ( ~nenter & hMove == 4'h3) nextState=S11;
88          else if ( ~nenter & hMove == 4'h7) nextState=S12;
89          else if ( ~nenter & hMove == 4'h8) nextState=S13;
90          else if ( ~nenter & isnotvalid|five|six|nine|four) nextState= S1; //S1
91          else nextState = S1;
92        end
93        S8:begin
94          //between states 2, 7 and 8 cases on hMove
95          if (~nenter & hMove == 4'h7) nextState = S4;
96          else if (~nenter & hMove == 4'h8) nextState = S5;
97          else if (~nenter & hMove == 4'h1) nextState = S14;
98          else if (~nenter & hMove == 4'h3) nextState = S15;
99          else if( ~nenter & isnotvalid|five|six|nine|two|four) nextState = S2;
100         // S2
101         else nextState = S2;
102       end
103     //if we are in these win states stay in these win states
104     //also if we are in win states then check if new_game is pressed
105     //and win
106       S9: begin
107         if (new_game && win)nextState = buff1;
108         else nextState = S9;
109       end
110       S10: begin
111         if (new_game && win)nextState = buff2;
112         else nextState = S10;
113       end
114       S11: begin
115         if (new_game && win)nextState = buff3;
116         else nextState  = S11;
117       end
118       S12: begin
119        if (new_game && win)nextState = buff4;
120        else nextState = S12;
121       end
122       S13: begin
123         if (new_game && win)nextState = buff5;
124         else nextState = S13;
125       end
126       S14: begin
127         if (new_game && win)nextState = buff6;
128         else nextState = S14;
129       end
130       S15: begin
131         if (new_game && win) nextState = buff7;
132         else nextState = S15;
133       end
134       //check if buffer and new_game released and nenter released
135       //then reset to S0
136       buff1,buff2,buff3,buff4,buff5,buff6,buff7,buff8,buff9:
137       if (~new_game && ~nenter) nextState = S0;
138   endcase
139   // Output logic defined here
140   win = 1'b0;
```

```
141   cMove = 4'h1; //default case for the hex on fpga
142   cHis = 16'd0;
143   hHis = 16'd0;
144   unique case (currState)
145     //This shows states S0-S15 and outputs cMove,win,cHis,hHis
146     S0: begin
147         cMove = 4'h5;
148         win = 1'b0;
149         cHis={4'h5,4'h0,4'h0,4'h0};
150         hHis={4'h0,4'h0,4'h0,4'h0};
151     end
152     S1: begin
153         cMove = 4'h6;
154         win = 1'b0;
155         cHis={4'h5,4'h6,4'h0,4'h0};
156         hHis={4'h9,4'h0,4'h0,4'h0};
157         end
158     S2: begin
159         cMove = 4'h2;
160         win = 1'b0;
161         cHis={4'h2,4'h5,4'h6,4'h0};
162         hHis={4'h4,4'h9,4'h0,4'h0};
163     end
164     S4: begin
165         win = 1'b1;
166         cMove = 4'h8;
167         cHis = {4'h2, 4'h5, 4'h6, 4'h8};
168         hHis = {4'h4, 4'h7, 4'h9, 4'h0};
169     end
170     S5: begin
171         win = 1'b1;
172         cMove = 4'h7;
173         cHis = {4'h2, 4'h5, 4'h6, 4'h7};
174         hHis = {4'h4, 4'h8,4'h9, 4'h0};
175     end
176     S6: begin
177         win = 1'b0;
178         cMove = 4'h5;
179         cHis = {4'h5,4'h0,4'h0,4'h0};
180         hHis = {4'h0,4'h0,4'h0,4'h0};
181     end
182     S7: begin
183         win = 1'b0;
184         cMove = 4'h6;
185         cHis={4'h5,4'h6,4'h0,4'h0};
186         hHis={4'h9,4'h0,4'h0,4'h0};
187     end
188     S8: begin
189         win = 1'b0;
190         cMove = 4'h2;
191         cHis = {4'h2,4'h5,4'h6,4'h0};
192         hHis = {4'h4,4'h9,4'h0,4'h0};
193     end
194     //win_condition states start here
195     //outputs cHis,cMove,win,and cMove
196     S9: begin
197         win = 1'b1;
198         cMove = 4'h4;
199         cHis = {4'h4,4'h5,4'h6,4'h0};
200         hHis = {4'h1,4'h9,4'h0,4'h0};
201     end
202     S10: begin
203         win = 1'b1;
204         cMove = 4'h4;
205         cHis = {4'h4,4'h5,4'h6,4'h0};
206         hHis = {4'h2,4'h9,4'h0,4'h0};
207     end
208     S11: begin
209         win = 1'b1;
210         cMove = 4'h4;
211         cHis = {4'h4,4'h5,4'h6,4'h0};
```

```
212         hHis = {4'h3,4'h9,4'h0,4'h0};
213     end
214     S12: begin
215         win = 1'b1;
216         cMove = 4'h4;
217         cHis = {4'h4,4'h5,4'h6,4'h0};
218         hHis = {4'h7,4'h9,4'h0,4'h0};
219     end
220     S13: begin
221         win = 1'b1;
222         cMove = 4'h4;
223         cHis = {4'h4,4'h5,4'h6,4'h0};
224         hHis = {4'h8,4'h9,4'h0,4'h0};
225     end
226     S14: begin
227         win = 1'b1;
228         cMove = 4'h8;
229         cHis = {4'h2,4'h5,4'h6,4'h8};
230         hHis = {4'h1,4'h4,4'h9,4'h0};
231     end
232     S15: begin
233         win = 1'b1;
234         cMove = 4'h8;
235         cHis = {4'h2,4'h5,4'h6,4'h8};
236         hHis = {4'h3,4'h4,4'h9,4'h0};
237     end
238   //buffer states between newgame and an end state
239   //These states output a win and cHis, and hHis,
240   //cHis and hHis represent the computer history
241   //hHis represents the human history
242     buff1: begin
243         win = 1'b1;
244         cMove = 4'h4;
245         cHis = {4'h4,4'h5,4'h6,4'h0};
246         hHis = {4'h1,4'h9,4'h0,4'h0};
247     end
248     buff2: begin
249         win = 1'b1;
250         cMove = 4'h4;
251         cHis = {4'h4,4'h5,4'h6,4'h0};
252         hHis = {4'h2,4'h9,4'h0,4'h0};
253     end
254     buff3: begin
255         win = 1'b1;
256         cMove = 4'h4;
257         cHis = {4'h4,4'h5,4'h6,4'h0};
258         hHis = {4'h3,4'h9,4'h0,4'h0};
259     end
260     buff4: begin
261         win = 1'b1;
262         cMove = 4'h4;
263         cHis = {4'h4,4'h5,4'h6,4'h0};
264         hHis = {4'h7,4'h9,4'h0,4'h0};
265     end
266     buff5: begin
267         win = 1'b1;
268         cMove = 4'h4;
269         cHis = {4'h4,4'h5,4'h6,4'h0};
270         hHis = {4'h8,4'h9,4'h0,4'h0};
271     end
272     buff6: begin
273         win = 1'b1;
274         cMove = 4'h8;
275         cHis = {4'h2,4'h5,4'h6,4'h8};
276         hHis = {4'h1,4'h4,4'h9,4'h0};
277     end
278     buff7: begin
279         win = 1'b1;
280         cMove = 4'h8;
281         cHis = {4'h2,4'h5,4'h6,4'h8};
282         hHis = {4'h3,4'h4,4'h9,4'h0};
```

```
283     end
284     buff8: begin
285         win = 1'b1;
286         cMove = 4'h8;
287         cHis = {4'h2, 4'h5, 4'h6, 4'h8};
288         hHis = {4'h4, 4'h7, 4'h9, 4'h0};
289     end
290     buff9: begin
291         win = 1'b1;
292         cMove = 4'h7;
293         cHis = {4'h2, 4'h5, 4'h6, 4'h7};
294         hHis = {4'h4, 4'h8,4'h9, 4'h0};
295   end
296   endcase
297  end
298
299 always_ff @(posedge clock, posedge reset)
300   if (reset)
301     currState <= S0; // or whatever the reset state is
302   else
303     currState <= nextState;
304 endmodule: myAbstractFSM
```

Lab Code [15 points]
Filename: chipInterface_task4.sv
AndrewID: jtbell

```
// Original Chip Interface to go back to if we mess up task 5
//chipinterface from task 1-4

`default_nettype none
module chipInterface
    (output logic [6:0]HEX0, // for the cMove
     output logic [16:0]LEDG, // win
     input logic [3:0] KEY, // clock
     input logic [17:0] SW); // reset

    logic [3:0] cMove;
    logic win;
    logic q2, q1, q0;
    logic [3:0] hMove;
    logic clock, reset;

  myAbstractFSM Af(.reset(SW[17]),.clock(KEY[0]),.hMove(SW[3:0]), .*);


  always_comb begin
    if (win == 1'b1)begin
        LEDG[0] = 1'b1;
    end else begin
        LEDG[0] = 1'b0;
    end
  end
  BCDtoSevenSegment B(.bcd(cMove[3:0]), .segment(HEX0));

endmodule : chipInterface

module BCDtoSevenSegment
    (input logic [3:0] bcd,
     output logic [6:0] segment);
     always_comb begin
        case(bcd)
            4'd0: segment = 7'b100_0000;
            4'd1: segment = 7'b111_1001;
            4'd2: segment = 7'b010_0100;
            4'd3: segment = 7'b101_0000;
            4'd4: segment = 7'b001_1001;
            4'd5: segment = 7'b001_0010;
            4'd6: segment = 7'b000_0010;
            4'd7: segment = 7'b111_1000;
            4'd8: segment = 7'b000_0000;
            4'd9: segment = 7'b001_1000;
            default: segment = 7'b111_1111;
        endcase
     end
 endmodule: BCDtoSevenSegment
```

Lab Code [15 points]
Filename: chipinterface_lab3.sv
AndrewID: jtbell

```
module chipinterface_lab3
    (output logic [17:0]LEDR, // for the cMove
     output logic [7:0]LEDG, // win
     output logic [6:0] HEX0,//cmove
     output logic [6:0] HEX1,//cmove
     output logic [6:0] HEX2,//cmove
     output logic [6:0] HEX3,//cmove
     output logic [6:0] HEX4,//hmove
     output logic [6:0] HEX5,//hmove
     output logic [6:0] HEX6,//hmove
     output logic [6:0] HEX7,//hmove
     input logic CLOCK_50, // clock
     input logic [17:0]SW,// hmove and reset
     input logic [3:0]KEY);//enter, new_game

  logic [3:0] cMove;
  logic win;
  logic q2, q1, q0;
  logic [3:0] hMove;
  logic clock, reset, new_game, enter;
  logic currState;
  logic [15:0] hHis, cHis;


  logic newgame, nenter;
  logic intermediate_1, intermediate_2;

  //Synchronizes with key3 and key0
  dFlipFlop d (.d(KEY[3]), .clock(CLOCK_50),.reset(SW[17]), .q(intermediate_2));
  dFlipFlop d0 (.d(intermediate_2), .clock(CLOCK_50),
  .reset(SW[17]), .q(nenter));

  dFlipFlop d2 (.d(KEY[0]), .clock(CLOCK_50),.reset(SW[17]),
  .q(intermediate_1));
  dFlipFlop d3 (.d(intermediate_1), .clock(CLOCK_50),.reset(SW[17]),
  .q(newgame));

  myAbstractFSM Af(.reset(SW[17]),.clock(CLOCK_50), .hMove(SW[3:0]),
                   .new_game(~newgame), .nenter(~nenter), .cMove(LEDR[3:0]),
                   .*) ;


  always_comb begin
    if (win == 1'b1)begin
        LEDG[0] = 1'b1;
    end else begin
        LEDG[0] = 1'b0;
    end
  end

  //assigning cHis and hHis to HEX
  assign LEDR[17:14] = cHis[3:0];
  BCDtoSevenSegment B1(.bcd(cHis[3:0]), .segment(HEX0));
  BCDtoSevenSegment B2(.bcd(cHis[7:4]), .segment(HEX1));
  BCDtoSevenSegment B3(.bcd(cHis[11:8]), .segment(HEX2));
  BCDtoSevenSegment B4(.bcd(cHis[15:12]), .segment(HEX3));
  BCDtoSevenSegment B5(.bcd(hHis[3:0]), .segment(HEX4));
  BCDtoSevenSegment B6(.bcd(hHis[7:4]), .segment(HEX5));
  BCDtoSevenSegment B7(.bcd(hHis[11:8]), .segment(HEX6));
  BCDtoSevenSegment B8(.bcd(hHis[15:12]), .segment(HEX7));
endmodule

module BCDtoSevenSegment
    (input logic [3:0] bcd,
     output logic [6:0] segment);
     always_comb begin
        case(bcd)
            4'd0: segment = 7'b111_1111;
            4'd1: segment = 7'b111_1001;
```

```
70              4'd2: segment = 7'b010_0100;
71              4'd3: segment = 7'b011_0000;
72              4'd4: segment = 7'b001_1001;
73              4'd5: segment = 7'b001_0010;
74              4'd6: segment = 7'b000_0010;
75              4'd7: segment = 7'b111_1000;
76              4'd8: segment = 7'b000_0000;
77              4'd9: segment = 7'b001_1000;
78              default: segment = 7'b111_1111;
79          endcase
80      end
81  endmodule: BCDtoSevenSegment
```