Lab Code [10 points]
Filename: fsm.sv

```
`default_nettype none

module fsm
  (input logic [2:0] fsm_notif, patternSignal,
   input logic end_seq, len_reached,
   input logic ready,
   input logic clock, reset_N,
   output logic en_pc, cl_pc, re_p, re_s, en_wc, cl_wc,
                cl_lc, en_lc, done, found_it, error,
                cl_tmp, en_tmp, ld_tmp, sel_tmp);

   enum logic [5:0] {start = 6'b000000, readLetPat = 6'b000001,
                     checkPattern = 6'b000010, compTwoFirst = 6'b000011,
                     compTwoSec = 6'b000100, compThreeFirst = 6'b000101,
                     compThreeSec = 6'b000110, compThreeThird = 6'b000111,
                     oneMatchUpTo = 6'b001001, incPatFinish = 6'b010111,
                     doneOneLeft = 6'b001010, doneTwoLeft = 6'b001011,
                     zeroMatchUpTo = 6'b011100, oneMatchAll = 6'b011101,
                     incLetPat = 6'b001100, endNoGood = 6'b001101,
                     Error = 6'b001111, incPat21 = 6'b010000,
                     incPat22 = 6'b010001, incPat31 = 6'b010010,
                     incPat32 = 6'b010011, incPat33 = 6'b010100,
                     incPatUpTo = 6'b010101, incPatAll = 6'b010110,
                     good = 6'b011110, incWordAll = 6'b011111,
                     incWordZeroUp = 6'b001000, incWordOneUp = 6'b001110,
                     seenA2 = 6'b100001, seenC2 = 6'b100010,
                     seenT2 = 6'b100011, seenG2 = 6'b100100,
                     seenAC3 = 6'b100101, seenAT3 = 6'b100110,
                     seenAG3 = 6'b100111, seenCT3 = 6'b101000,
                     seenCG3 = 6'b101001, seenTG3 = 6'b101010,
                     seenA3 = 6'b101011, seenC3 = 6'b101100,
                     seenT3 = 6'b101101, seenG3 = 6'b101110,
                     check21 = 6'b111001, check31 = 6'b111010,
                     loadTmp2 = 6'b111011, loadTmp3 = 6'b111100}
                     state, nextState;

   // next state logic
   always_comb begin
     if (state != start && fsm_notif == 2)
       nextState = Error;
     else if (end_seq && fsm_notif != 7)
       nextState = endNoGood;
     unique case (state)
      start : nextState = (ready) ? readLetPat : start;
      readLetPat : nextState = checkPattern;
      checkPattern : begin
                       if (fsm_notif == 0)
                         nextState = incLetPat;
                       else if (fsm_notif == 1) // no matches
                         nextState = endNoGood;
                       else if (fsm_notif == 2)
                         nextState = Error;
                       else if (fsm_notif == 3)
                         nextState = incPat21;
                       else if (fsm_notif == 4)
                         nextState = incPat31;
                       else if (fsm_notif == 5)
                         nextState = incPatAll;
                       else if (fsm_notif == 6)
                         nextState = incPatUpTo;
                       else if (fsm_notif == 7)
                         nextState = good;
                     end
    compTwoFirst : begin
                     if (fsm_notif == 7)
                       nextState = Error;
                     else if (fsm_notif == 0)
                       nextState = doneOneLeft;
                     else
                       nextState = incPat22; //bad
```

```
 71                     end
 72
 73      compTwoSec : begin
 74                     if (fsm_notif == 7)
 75                       nextState = Error;
 76                     else if (fsm_notif == 0)
 77                       nextState = incLetPat;
 78                     else
 79                       nextState = endNoGood; //bad
 80                   end
 81
 82       compThreeFirst : begin
 83                          if (fsm_notif == 7)
 84                            nextState = Error;
 85                          else if (fsm_notif == 0)
 86                            nextState = doneTwoLeft;
 87                          else
 88                            nextState = incPat32; //bad
 89                        end
 90
 91       compThreeSec : begin
 92                          if (fsm_notif == 7)
 93                            nextState = Error;
 94                          else if (fsm_notif == 0)
 95                            nextState = doneOneLeft;
 96                          else
 97                            nextState = incPat33; //bad
 98                        end
 99
100       compThreeThird : begin
101                          if (fsm_notif == 7)
102                            nextState = Error;
103                          else if (fsm_notif == 0)
104                            nextState = incLetPat;
105                          else
106                            nextState = endNoGood; //bad
107                        end
108
109       incPatAll : nextState = oneMatchAll;
110       oneMatchAll : begin
111                       if (fsm_notif == 0 && len_reached == 0)
112                         nextState = incWordAll;
113                       else if (len_reached == 1)
114                         nextState = incPatFinish;
115                       else if (fsm_notif == 1)
116                         nextState = endNoGood; // bad
117                     end
118
119       incWordAll : nextState = oneMatchAll;
120
121       incPatUpTo : nextState = zeroMatchUpTo;
122       zeroMatchUpTo : nextState =
123                       (fsm_notif == 0) ? oneMatchUpTo : endNoGood; // bad
124       incWordZeroUp: nextState = oneMatchUpTo;
125       oneMatchUpTo : begin
126                          if (fsm_notif == 0 && len_reached == 0)
127                            nextState = incWordOneUp;
128                          else if (len_reached == 1)
129                            nextState = incPatFinish;
130                          else
131                            nextState = incPatFinish;
132                        end
133       incWordOneUp : nextState = oneMatchUpTo;
134
135       incPatFinish: nextState = readLetPat;
136
137       doneOneLeft : nextState = incLetPat;
138       doneTwoLeft : nextState = doneOneLeft;
139
140       incLetPat : nextState = readLetPat;
141
```

```
142       incPat21 : nextState = loadTmp2;
143       incPat22 : nextState = compTwoSec;
144       incPat31 : nextState = loadTmp3;
145       incPat32 : nextState = compThreeSec;
146       incPat33 : nextState = compThreeThird;
147
148       loadTmp2 : nextState = check21;
149       loadTmp3 : nextState = check31;
150
151       // A = 3'b000, C = 3'b001, T = 3'b010, G = 3'b011,
152       // other = 3'b111
153       check21 : begin
154                   if (patternSignal == 0) // A
155                     nextState = seenA2;
156                   else if (patternSignal == 1) // C
157                     nextState = seenC2;
158                   else if (patternSignal == 2) // T
159                     nextState = seenT2;
160                   else if (patternSignal == 3) // G
161                     nextState = seenG2;
162                   else
163                     nextState = Error;
164                 end
165
166       check31 : begin
167                   if (patternSignal == 0) // A
168                     nextState = seenA3;
169                   else if (patternSignal == 1) // C
170                     nextState = seenC3;
171                   else if (patternSignal == 2) // T
172                     nextState = seenT3;
173                   else if (patternSignal == 3) // G
174                     nextState = seenG3;
175                   else
176                     nextState = Error;
177                 end
178
179       seenA2 : begin
180                   if (patternSignal == 0) // A
181                     nextState = Error;
182                   else if (patternSignal == 1) // C
183                     nextState = compTwoFirst;
184                   else if (patternSignal == 2) // T
185                     nextState = compTwoFirst;
186                   else if (patternSignal == 3) // G
187                     nextState = compTwoFirst;
188                   else
189                     nextState = Error;
190                 end
191
192       seenC2 : begin
193                   if (patternSignal == 0) // A
194                     nextState = compTwoFirst;
195                   else if (patternSignal == 1) // C
196                     nextState = Error;
197                   else if (patternSignal == 2) // T
198                     nextState = compTwoFirst;
199                   else if (patternSignal == 3) // G
200                     nextState = compTwoFirst;
201                   else
202                     nextState = Error;
203                 end
204
205       seenT2 : begin
206                   if (patternSignal == 0) // A
207                     nextState = compTwoFirst;
208                   else if (patternSignal == 1) // C
209                     nextState = compTwoFirst;
210                   else if (patternSignal == 2) // T
211                     nextState = Error;
212                   else if (patternSignal == 3) // G
```

```
213                     nextState = compTwoFirst;
214                   else
215                     nextState = Error;
216                 end
217
218       seenG2 : begin
219                   if (patternSignal == 0) // A
220                     nextState = compTwoFirst;
221                   else if (patternSignal == 1) // C
222                     nextState = compTwoFirst;
223                   else if (patternSignal == 2) // T
224                     nextState = compTwoFirst;
225                   else if (patternSignal == 3) // G
226                     nextState = Error;
227                   else
228                     nextState = Error;
229                 end
230
231
232       seenA3 : begin
233                   if (patternSignal == 0) // A
234                     nextState = Error;
235                   else if (patternSignal == 1) // C
236                     nextState = seenAC3;
237                   else if (patternSignal == 2) // T
238                     nextState = seenAT3;
239                   else if (patternSignal == 3) // G
240                     nextState = seenAG3;
241                   else
242                     nextState = Error;
243                 end
244
245       seenC3 : begin
246                   if (patternSignal == 0) // A
247                     nextState = seenAC3;
248                   else if (patternSignal == 1) // C
249                     nextState = Error;
250                   else if (patternSignal == 2) // T
251                     nextState = seenCT3;
252                   else if (patternSignal == 3) // G
253                     nextState = seenCG3;
254                   else
255                     nextState = Error;
256                 end
257
258       seenT3 : begin
259                   if (patternSignal == 0) // A
260                     nextState = seenAT3;
261                   else if (patternSignal == 1) // C
262                     nextState = seenCT3;
263                   else if (patternSignal == 2) // T
264                     nextState = Error;
265                   else if (patternSignal == 3) // G
266                     nextState = seenTG3;
267                   else
268                     nextState = Error;
269                 end
270
271       seenG3 : begin
272                   if (patternSignal == 0) // A
273                     nextState = seenAG3;
274                   else if (patternSignal == 1) // C
275                     nextState = seenCG3;
276                   else if (patternSignal == 2) // T
277                     nextState = seenTG3;
278                   else if (patternSignal == 3) // G
279                     nextState = Error;
280                   else
281                     nextState = Error;
282                 end
283
```

```
284       seenAC3 : begin
285                   if (patternSignal == 0) // A
286                     nextState = Error;
287                   else if (patternSignal == 1) // C
288                     nextState = Error;
289                   else if (patternSignal == 2) // T
290                     nextState = compThreeFirst;
291                   else if (patternSignal == 3) // G
292                     nextState = compThreeFirst;
293                   else
294                     nextState = Error;
295                 end
296
297       seenAT3 : begin
298                   if (patternSignal == 0) // A
299                     nextState = Error;
300                   else if (patternSignal == 1) // C
301                     nextState = compThreeFirst;
302                   else if (patternSignal == 2) // T
303                     nextState = Error;
304                   else if (patternSignal == 3) // G
305                     nextState = compThreeFirst;
306                   else
307                     nextState = Error;
308                 end
309
310       seenAG3 : begin
311                   if (patternSignal == 0) // A
312                     nextState = Error;
313                   else if (patternSignal == 1) // C
314                     nextState = compThreeFirst;
315                   else if (patternSignal == 2) // T
316                     nextState = compThreeFirst;
317                   else if (patternSignal == 3) // G
318                     nextState = Error;
319                   else
320                     nextState = Error;
321                 end
322
323       seenCT3 : begin
324                   if (patternSignal == 0) // A
325                     nextState = compThreeFirst;
326                   else if (patternSignal == 1) // C
327                     nextState = Error;
328                   else if (patternSignal == 2) // T
329                     nextState = Error;
330                   else if (patternSignal == 3) // G
331                     nextState = compThreeFirst;
332                   else
333                     nextState = Error;
334                 end
335
336       seenCG3 : begin
337                   if (patternSignal == 0) // A
338                     nextState = compThreeFirst;
339                   else if (patternSignal == 1) // C
340                     nextState = Error;
341                   else if (patternSignal == 2) // T
342                     nextState = compThreeFirst;
343                   else if (patternSignal == 3) // G
344                     nextState = Error;
345                   else
346                     nextState = Error;
347                 end
348
349       seenTG3 : begin
350                   if (patternSignal == 0) // A
351                     nextState = compThreeFirst;
352                   else if (patternSignal == 1) // C
353                     nextState = compThreeFirst;
354                   else if (patternSignal == 2) // T
```

```
355                     nextState = Error;
356                   else if (patternSignal == 3) // G
357                     nextState = Error;
358                   else
359                     nextState = Error;
360                 end
361
362       endNoGood : nextState = (ready) ? readLetPat : endNoGood;
363       Error : nextState = (ready) ? readLetPat : Error;
364       good : nextState = (ready) ? readLetPat : good;
365       default : nextState = start;
366     endcase
367    end
368
369    always_ff @(posedge clock)
370     if (~reset_N)
371       state <= start;
372     else
373       state <= nextState;
374
375     // en_pc, cl_pc, re_p, re_s,
376     // en_wc, cl_wc, cl_lc, en_cl;
377    always_comb begin
378     done = 0;
379     found_it = 0;
380     error = 0;
381     en_wc = 0;
382     cl_wc = 0;
383     en_pc = 0;
384     cl_pc = 0;
385     en_lc = 0;
386     cl_lc = 0;
387     en_tmp = 0;
388     ld_tmp = 0;
389     cl_tmp = 0;
390     sel_tmp = 0;
391     //en_fc = 0;
392
393     if (state == start)
394       begin
395         en_wc = 1;
396         en_pc = 1;
397         cl_wc = 1;
398         cl_pc = 1;
399         en_lc = 1;
400         cl_lc = 1;
401         re_p = 0;
402         re_s = 0;
403       end
404
405     else if (state == readLetPat)
406       begin
407         re_s = 1;
408         re_p = 1;
409         en_wc = 0;
410         en_pc = 0;
411         cl_wc = 0;
412         cl_pc = 0;
413         en_lc = 0;
414         cl_lc = 0;
415      end
416
417     else if (state == checkPattern ||
418              state == compTwoFirst ||
419              state == compTwoSec ||
420              state == compThreeFirst ||
421              state == compThreeSec ||
422              state == compThreeThird)
423       begin
424         en_wc = 0;
425         cl_wc = 0;
```

```
426         en_pc = 0;
427         cl_pc = 0;
428         en_lc = 0;
429         cl_lc = 0;
430         re_p = 1;
431         re_s = 1;
432       end
433
434     else if (state == doneOneLeft || state == doneTwoLeft)
435       begin // skip one pattern
436         en_wc = 0;
437         cl_wc = 0;
438         en_pc = 1;
439         cl_pc = 0;
440         en_lc = 0;
441         cl_lc = 0;
442         re_p = 0;
443         re_s = 0;
444       end
445
446     else if (state == incLetPat)
447       begin
448         en_wc = 1;
449         cl_wc = 0;
450         en_pc = 1;
451         cl_pc = 0;
452         en_lc = 0;
453         cl_lc = 0;
454         re_p = 1;
455         re_s = 1;
456       end
457
458     else if (state == endNoGood)
459       begin
460         en_wc = 1;
461         cl_wc = 1;
462         en_pc = 1;
463         cl_pc = 1;
464         en_lc = 1;
465         cl_lc = 1;
466         re_p = 0;
467         re_s = 0;
468         done = 1;
469         found_it = 0;
470         error = 0;
471       end
472
473     else if (state == Error)
474       begin
475         en_wc = 1;
476         cl_wc = 1;
477         en_pc = 1;
478         cl_pc = 1;
479         en_lc = 1;
480         cl_lc = 1;
481         re_p = 0;
482         re_s = 0;
483         done = 1;
484         found_it = 0;
485         error = 1;
486       end
487
488     else if (state == incPat21 ||
489              state == incPat31)
490       begin
491         en_wc = 0;
492         cl_wc = 0;
493         en_pc = 1;
494         cl_pc = 0;
495         en_lc = 0;
496         cl_lc = 0;
```

```
497         re_p = 1;
498         re_s = 1;
499         ld_tmp = 1;
500         en_tmp = 1;
501         sel_tmp = 1;
502       end
503
504     else if (state == incPat22 ||
505               state == incPat32 ||
506               state == incPat32 ||
507               state == incPat33 ||
508               state == incPatFinish)
509       begin
510         en_wc = 0;
511         cl_wc = 0;
512         en_pc = 1;
513         cl_pc = 0;
514         en_lc = 0;
515         cl_lc = 0;
516         re_p = 1;
517         re_s = 1;
518       end
519
520     else if (state == incPatUpTo ||
521               state == incPatAll)
522       begin
523         en_wc = 0;
524         cl_wc = 0;
525         en_pc = 1;
526         cl_pc = 0;
527         en_lc = 1;
528         cl_lc = 1;
529         re_p = 1;
530         re_s = 1;
531       end
532
533     else if (state == oneMatchAll ||
534               state == zeroMatchUpTo ||
535               state == oneMatchUpTo)
536       begin
537         en_wc = 0;
538         cl_wc = 0;
539         en_pc = 0;
540         cl_pc = 0;
541         en_lc = 0;
542         cl_lc = 0;
543         re_p = 1;
544         re_s = 1;
545       end
546
547     else if (state == incWordOneUp ||
548               state == incWordZeroUp ||
549               state == incWordAll)
550       begin
551         en_wc = 1;
552         cl_wc = 0;
553         en_pc = 0;
554         cl_pc = 0;
555         en_lc = 1;
556         cl_lc = 0;
557         re_p = 1;
558         re_s = 1;
559       end
560
561     else if (state == good)
562       begin
563         en_wc = 1;
564         cl_wc = 1;
565         en_pc = 1;
566         cl_pc = 1;
567         en_lc = 1;
```

```
568         cl_lc = 1;
569         re_p = 0;
570         re_s = 0;
571         done = 1;
572         found_it = 1;
573         error = 0;
574       end
575
576     else if (state == check21 ||
577              state == check31)
578       begin
579         en_wc = 0;
580         cl_wc = 0;
581         en_pc = 0;
582         cl_pc = 0;
583         en_lc = 0;
584         cl_lc = 0;
585         re_p = 1;
586         re_s = 0;
587         en_tmp = 1;
588         ld_tmp = 0;
589         sel_tmp = 1;
590       end
591
592     else if (state == seenA2 ||
593              state == seenA3 ||
594              state == seenC2 ||
595              state == seenC3 ||
596              state == seenT2 ||
597              state == seenT3 ||
598              state == seenG2 ||
599              state == seenG2 ||
600              state == seenAC3 ||
601              state == seenAG3 ||
602              state == seenAT3 ||
603              state == seenTG3 ||
604              state == seenCG3 ||
605              state == seenCT3)
606        begin
607         en_wc = 0;
608         cl_wc = 0;
609         en_pc = 0;
610         cl_pc = 0;
611         en_lc = 0;
612         cl_lc = 0;
613         re_p = 1;
614         re_s = 0;
615         en_tmp = 1;
616         ld_tmp = 0;
617         sel_tmp = 1;
618       end
619
620     else if (state == loadTmp2 ||
621              state == loadTmp3)
622       begin
623         en_wc = 0;
624         cl_wc = 0;
625         en_pc = 0;
626         cl_pc = 0;
627         en_lc = 0;
628         cl_lc = 0;
629         re_p = 1;
630         re_s = 0;
631         en_tmp = 1;
632         ld_tmp = 1;
633         sel_tmp = 1;
634       end
635
636     end
637
638 endmodule: fsm
```

639
640

Lab Code [10 points]
Filename: fsm2.sv

```
`default_nettype none

module fsm2
  (input logic [2:0] fsm_notif, patternSignal,
   input logic end_seq, len_reached,
   input logic ready,
   input logic clock, reset_N,
   output logic en_pc, cl_pc, re_p, re_s, en_wc, cl_wc,
                cl_lc, en_lc, done, found_it, error,
                cl_tmp, en_tmp, ld_tmp, sel_tmp, ld_wc,
                ld_pc);

   enum logic [5:0] {start = 6'b000000, readLetPat = 6'b000001,
                     checkPattern = 6'b000010, compTwoFirst = 6'b000011,
                     compTwoSec = 6'b000100, compThreeFirst = 6'b000101,
                     compThreeSec = 6'b000110, compThreeThird = 6'b000111,
                     oneMatchUpTo = 6'b001001, incPatFinish = 6'b010111,
                     doneOneLeft = 6'b001010, doneTwoLeft = 6'b001011,
                     zeroMatchUpTo = 6'b011100, oneMatchAll = 6'b011101,
                     incLetPat = 6'b001100, endNoGood = 6'b001101,
                     Error = 6'b001111, incPat21 = 6'b010000,
                     incPat22 = 6'b010001, incPat31 = 6'b010010,
                     incPat32 = 6'b010011, incPat33 = 6'b010100,
                     incPatUpTo = 6'b010101, incPatAll = 6'b010110,
                     good = 6'b011110, incWordAll = 6'b011111,
                     incWordZeroUp = 6'b001000, incWordOneUp = 6'b001110,
                     seenA2 = 6'b100001, seenC2 = 6'b100010,
                     seenT2 = 6'b100011, seenG2 = 6'b100100,
                     seenAC3 = 6'b100101, seenAT3 = 6'b100110,
                     seenAG3 = 6'b100111, seenCT3 = 6'b101000,
                     seenCG3 = 6'b101001, seenTG3 = 6'b101010,
                     seenA3 = 6'b101011, seenC3 = 6'b101100,
                     seenT3 = 6'b101101, seenG3 = 6'b101110,
                     check21 = 6'b111001, check31 = 6'b111010,
                     loadTmp2 = 6'b111011, loadTmp3 = 6'b111100}
                     state, nextState;


   // next state logic
   always_comb begin
     if (state != start && fsm_notif == 2)
       nextState = Error;
     else if (end_seq && fsm_notif != 7)
       nextState = endNoGood;
     unique case (state)
      start : nextState = (ready) ? readLetPat : start;
      readLetPat : nextState = checkPattern;
      checkPattern : begin
                       if (fsm_notif == 0)
                         nextState = incLetPat;
                       else if (fsm_notif == 1) // no matches
                         nextState = endNoGood;
                       else if (fsm_notif == 2)
                         nextState = Error;
                       else if (fsm_notif == 3)
                         nextState = incPat21;
                       else if (fsm_notif == 4)
                         nextState = incPat31;
                       else if (fsm_notif == 5)
                         nextState = incPatAll;
                       else if (fsm_notif == 6)
                         nextState = incPatUpTo;
                       else if (fsm_notif == 7)
                         nextState = good;
                     end
     compTwoFirst : begin
                       if (fsm_notif == 7)
                         nextState = Error;
                       else if (fsm_notif == 0)
                         nextState = doneOneLeft;
```

```
 71                      else
 72                        nextState = incPat22; //bad
 73                      end
 74
 75      compTwoSec : begin
 76                      if (fsm_notif == 7)
 77                        nextState = Error;
 78                      else if (fsm_notif == 0)
 79                        nextState = incLetPat;
 80                      else
 81                        nextState = endNoGood; //bad
 82                    end
 83
 84       compThreeFirst : begin
 85                          if (fsm_notif == 7)
 86                            nextState = Error;
 87                          else if (fsm_notif == 0)
 88                            nextState = doneTwoLeft;
 89                          else
 90                            nextState = incPat32; //bad
 91                        end
 92
 93       compThreeSec : begin
 94                          if (fsm_notif == 7)
 95                            nextState = Error;
 96                          else if (fsm_notif == 0)
 97                            nextState = doneOneLeft;
 98                          else
 99                            nextState = incPat33; //bad
100                        end
101
102       compThreeThird : begin
103                          if (fsm_notif == 7)
104                            nextState = Error;
105                          else if (fsm_notif == 0)
106                            nextState = incLetPat;
107                          else
108                            nextState = endNoGood; //bad
109                        end
110
111       incPatAll : nextState = oneMatchAll;
112       oneMatchAll : begin
113                       if (fsm_notif == 0 && len_reached == 0)
114                         nextState = incWordAll;
115                       else if (len_reached == 1)
116                         nextState = incPatFinish;
117                       else if (fsm_notif == 1)
118                         nextState = endNoGood; // bad
119                     end
120
121       incWordAll : nextState = oneMatchAll;
122
123       incPatUpTo : nextState = zeroMatchUpTo;
124       zeroMatchUpTo : nextState =
125                      (fsm_notif == 0) ? oneMatchUpTo : endNoGood; // bad
126       incWordZeroUp: nextState = oneMatchUpTo;
127       oneMatchUpTo : begin
128                          if (fsm_notif == 0 && len_reached == 0)
129                            nextState = incWordOneUp;
130                          else
131                            nextState = incPatFinish;
132                        end
133       incWordOneUp : nextState = oneMatchUpTo;
134
135       incPatFinish: nextState = readLetPat;
136
137       doneOneLeft : nextState = incLetPat;
138       doneTwoLeft : nextState = doneOneLeft;
139
140       incLetPat : nextState = readLetPat;
141
```

```
142       incPat21 : nextState = loadTmp2;
143       incPat22 : nextState = compTwoSec;
144       incPat31 : nextState = loadTmp3;
145       incPat32 : nextState = compThreeSec;
146       incPat33 : nextState = compThreeThird;
147
148       loadTmp2 : nextState = check21;
149       loadTmp3 : nextState = check31;
150
151       // A = 3'b000, C = 3'b001, T = 3'b010, G = 3'b011,
152       // other = 3'b111
153       check21 : begin
154                   if (patternSignal == 0) // A
155                     nextState = seenA2;
156                   else if (patternSignal == 1) // C
157                     nextState = seenC2;
158                   else if (patternSignal == 2) // T
159                     nextState = seenT2;
160                   else if (patternSignal == 3) // G
161                     nextState = seenG2;
162                   else
163                     nextState = Error;
164                 end
165
166       check31 : begin
167                   if (patternSignal == 0) // A
168                     nextState = seenA3;
169                   else if (patternSignal == 1) // C
170                     nextState = seenC3;
171                   else if (patternSignal == 2) // T
172                     nextState = seenT3;
173                   else if (patternSignal == 3) // G
174                     nextState = seenG3;
175                   else
176                     nextState = Error;
177                 end
178
179       seenA2 : begin
180                   if (patternSignal == 0) // A
181                     nextState = Error;
182                   else if (patternSignal == 1) // C
183                     nextState = compTwoFirst;
184                   else if (patternSignal == 2) // T
185                     nextState = compTwoFirst;
186                   else if (patternSignal == 3) // G
187                     nextState = compTwoFirst;
188                   else
189                     nextState = Error;
190                 end
191
192       seenC2 : begin
193                   if (patternSignal == 0) // A
194                     nextState = compTwoFirst;
195                   else if (patternSignal == 1) // C
196                     nextState = Error;
197                   else if (patternSignal == 2) // T
198                     nextState = compTwoFirst;
199                   else if (patternSignal == 3) // G
200                     nextState = compTwoFirst;
201                   else
202                     nextState = Error;
203                 end
204
205       seenT2 : begin
206                   if (patternSignal == 0) // A
207                     nextState = compTwoFirst;
208                   else if (patternSignal == 1) // C
209                     nextState = compTwoFirst;
210                   else if (patternSignal == 2) // T
211                     nextState = Error;
212                   else if (patternSignal == 3) // G
```

```
213                     nextState = compTwoFirst;
214                   else
215                     nextState = Error;
216                 end
217
218       seenG2 : begin
219                   if (patternSignal == 0) // A
220                     nextState = compTwoFirst;
221                   else if (patternSignal == 1) // C
222                     nextState = compTwoFirst;
223                   else if (patternSignal == 2) // T
224                     nextState = compTwoFirst;
225                   else if (patternSignal == 3) // G
226                     nextState = Error;
227                   else
228                     nextState = Error;
229                 end
230
231
232       seenA3 : begin
233                   if (patternSignal == 0) // A
234                     nextState = Error;
235                   else if (patternSignal == 1) // C
236                     nextState = seenAC3;
237                   else if (patternSignal == 2) // T
238                     nextState = seenAT3;
239                   else if (patternSignal == 3) // G
240                     nextState = seenAG3;
241                   else
242                     nextState = Error;
243                 end
244
245       seenC3 : begin
246                   if (patternSignal == 0) // A
247                     nextState = seenAC3;
248                   else if (patternSignal == 1) // C
249                     nextState = Error;
250                   else if (patternSignal == 2) // T
251                     nextState = seenCT3;
252                   else if (patternSignal == 3) // G
253                     nextState = seenCG3;
254                   else
255                     nextState = Error;
256                 end
257
258       seenT3 : begin
259                   if (patternSignal == 0) // A
260                     nextState = seenAT3;
261                   else if (patternSignal == 1) // C
262                     nextState = seenCT3;
263                   else if (patternSignal == 2) // T
264                     nextState = Error;
265                   else if (patternSignal == 3) // G
266                     nextState = seenTG3;
267                   else
268                     nextState = Error;
269                 end
270
271       seenG3 : begin
272                   if (patternSignal == 0) // A
273                     nextState = seenAG3;
274                   else if (patternSignal == 1) // C
275                     nextState = seenCG3;
276                   else if (patternSignal == 2) // T
277                     nextState = seenTG3;
278                   else if (patternSignal == 3) // G
279                     nextState = Error;
280                   else
281                     nextState = Error;
282                 end
283
```

```
284       seenAC3 : begin
285                   if (patternSignal == 0) // A
286                     nextState = Error;
287                   else if (patternSignal == 1) // C
288                     nextState = Error;
289                   else if (patternSignal == 2) // T
290                     nextState = compThreeFirst;
291                   else if (patternSignal == 3) // G
292                     nextState = compThreeFirst;
293                   else
294                     nextState = Error;
295                 end
296
297       seenAT3 : begin
298                   if (patternSignal == 0) // A
299                     nextState = Error;
300                   else if (patternSignal == 1) // C
301                     nextState = compThreeFirst;
302                   else if (patternSignal == 2) // T
303                     nextState = Error;
304                   else if (patternSignal == 3) // G
305                     nextState = compThreeFirst;
306                   else
307                     nextState = Error;
308                 end
309
310       seenAG3 : begin
311                   if (patternSignal == 0) // A
312                     nextState = Error;
313                   else if (patternSignal == 1) // C
314                     nextState = compThreeFirst;
315                   else if (patternSignal == 2) // T
316                     nextState = compThreeFirst;
317                   else if (patternSignal == 3) // G
318                     nextState = Error;
319                   else
320                     nextState = Error;
321                 end
322
323       seenCT3 : begin
324                   if (patternSignal == 0) // A
325                     nextState = compThreeFirst;
326                   else if (patternSignal == 1) // C
327                     nextState = Error;
328                   else if (patternSignal == 2) // T
329                     nextState = Error;
330                   else if (patternSignal == 3) // G
331                     nextState = compThreeFirst;
332                   else
333                     nextState = Error;
334                 end
335
336       seenCG3 : begin
337                   if (patternSignal == 0) // A
338                     nextState = compThreeFirst;
339                   else if (patternSignal == 1) // C
340                     nextState = Error;
341                   else if (patternSignal == 2) // T
342                     nextState = compThreeFirst;
343                   else if (patternSignal == 3) // G
344                     nextState = Error;
345                   else
346                     nextState = Error;
347                 end
348
349       seenTG3 : begin
350                   if (patternSignal == 0) // A
351                     nextState = compThreeFirst;
352                   else if (patternSignal == 1) // C
353                     nextState = compThreeFirst;
354                   else if (patternSignal == 2) // T
```

```
355                     nextState = Error;
356                   else if (patternSignal == 3) // G
357                     nextState = Error;
358                   else
359                     nextState = Error;
360                 end
361
362       endNoGood : nextState = (ready) ? readLetPat : endNoGood;
363       Error : nextState = (ready) ? readLetPat : Error;
364       good : nextState = (ready) ? readLetPat : good;
365       default : nextState = start;
366     endcase
367    end
368
369
370    always_ff @(posedge clock)
371     if (~reset_N)
372       state <= start;
373     else
374       state <= nextState;
375
376     // en_pc, cl_pc, re_p, re_s,
377     // en_wc, cl_wc, cl_lc, en_cl;
378    always_comb begin
379     done = 0;
380     found_it = 0;
381     error = 0;
382     en_wc = 0;
383     cl_wc = 0;
384     en_pc = 0;
385     cl_pc = 0;
386     en_lc = 0;
387     cl_lc = 0;
388     ld_wc = 0;
389     ld_pc = 0;
390     en_tmp = 0;
391     ld_tmp = 0;
392     cl_tmp = 0;
393     sel_tmp = 0;
394
395     if (state == start)
396       begin
397         en_wc = 1;
398         en_pc = 1;
399         cl_wc = 0;
400         cl_pc = 0;
401         en_lc = 1;
402         cl_lc = 1;
403         ld_wc = 1;
404         ld_pc = 1;
405         re_p = 0;
406         re_s = 0;
407       end
408
409     else if (state == readLetPat)
410       begin
411         re_s = 1;
412         re_p = 1;
413         en_wc = 0;
414         en_pc = 0;
415         cl_wc = 0;
416         cl_pc = 0;
417         en_lc = 0;
418         cl_lc = 0;
419      end
420
421     else if (state == checkPattern ||
422              state == compTwoFirst ||
423              state == compTwoSec ||
424              state == compThreeFirst ||
425              state == compThreeSec ||
```

```
426               start == compThreeThird)
427       begin
428         en_wc = 0;
429         cl_wc = 0;
430         en_pc = 0;
431         cl_pc = 0;
432         en_lc = 0;
433         cl_lc = 0;
434         re_p = 1;
435         re_s = 1;
436       end
437
438     else if (state == doneOneLeft || state == doneTwoLeft)
439       begin // skip one pattern
440         en_wc = 0;
441         cl_wc = 0;
442         en_pc = 1;
443         cl_pc = 0;
444         en_lc = 0;
445         cl_lc = 0;
446         re_p = 0;
447         re_s = 0;
448       end
449
450     else if (state == incLetPat)
451       begin
452         en_wc = 1;
453         cl_wc = 0;
454         en_pc = 1;
455         cl_pc = 0;
456         en_lc = 0;
457         cl_lc = 0;
458         re_p = 1;
459         re_s = 1;
460       end
461
462     else if (state == endNoGood)
463       begin
464         en_wc = 1;
465         cl_wc = 0;
466         en_pc = 1;
467         cl_pc = 0;
468         en_lc = 1;
469         cl_lc = 1;
470         ld_wc = 1;
471         ld_pc = 1;
472         re_p = 0;
473         re_s = 0;
474         done = 1;
475         found_it = 0;
476         error = 0;
477       end
478
479     else if (state == Error)
480       begin
481         en_wc = 1;
482         cl_wc = 0;
483         en_pc = 1;
484         cl_pc = 0;
485         en_lc = 1;
486         cl_lc = 1;
487         ld_wc = 1;
488         ld_pc = 1;
489         re_p = 0;
490         re_s = 0;
491         done = 1;
492         found_it = 0;
493         error = 1;
494       end
495
496    else if (state == check21 ||
```

```
497             state == check31)
498       begin
499         en_wc = 0;
500         cl_wc = 0;
501         en_pc = 0;
502         cl_pc = 0;
503         en_lc = 0;
504         cl_lc = 0;
505         re_p = 1;
506         re_s = 0;
507         ld_tmp = 0;
508         en_tmp = 1;
509         sel_tmp = 1;
510       end
511
512
513     else if (state == incPat21 ||
514              state == incPat31)
515       begin
516         en_wc = 0;
517         cl_wc = 0;
518         en_pc = 1;
519         cl_pc = 0;
520         en_lc = 0;
521         cl_lc = 0;
522         re_p = 1;
523         re_s = 1;
524         ld_tmp = 1;
525         en_tmp = 1;
526         sel_tmp = 1;
527       end
528
529     else if (state == incPat22 ||
530              state == incPat32 ||
531              state == incPat32 ||
532              state == incPat33 ||
533              state == incPatFinish)
534       begin
535         en_wc = 0;
536         cl_wc = 0;
537         en_pc = 1;
538         cl_pc = 0;
539         en_lc = 0;
540         cl_lc = 0;
541         re_p = 1;
542         re_s = 1;
543       end
544
545     else if (state == incPatUpTo ||
546              state == incPatAll)
547       begin
548         en_wc = 0;
549         cl_wc = 0;
550         en_pc = 1;
551         cl_pc = 0;
552         en_lc = 1;
553         cl_lc = 1;
554         re_p = 1;
555         re_s = 1;
556       end
557
558     else if (state == oneMatchAll ||
559              state == zeroMatchUpTo ||
560              state == oneMatchUpTo)
561       begin
562         en_wc = 0;
563         cl_wc = 0;
564         en_pc = 0;
565         cl_pc = 0;
566         en_lc = 0;
567         cl_lc = 0;
```

```
568         re_p = 1;
569         re_s = 1;
570       end
571
572     else if (state == incWordOneUp ||
573              state == incWordZeroUp ||
574              state == incWordAll)
575       begin
576         en_wc = 1;
577         cl_wc = 0;
578         en_pc = 0;
579         cl_pc = 0;
580         en_lc = 1;
581         cl_lc = 0;
582         re_p = 1;
583         re_s = 1;
584       end
585
586     else if (state == good)
587       begin
588         en_wc = 1;
589         cl_wc = 0;
590         en_pc = 1;
591         cl_pc = 0;
592         en_lc = 1;
593         cl_lc = 1;
594         ld_pc = 1;
595         ld_wc = 1;
596         re_p = 0;
597         re_s = 0;
598         done = 1;
599         found_it = 1;
600         error = 0;
601       end
602
603
604      else if (state == seenA2 ||
605               state == seenA3 ||
606               state == seenC2 ||
607               state == seenC3 ||
608               state == seenT2 ||
609               state == seenT3 ||
610               state == seenG2 ||
611               state == seenG2 ||
612               state == seenAC3 ||
613               state == seenAG3 ||
614               state == seenAT3 ||
615               state == seenTG3 ||
616               state == seenCG3 ||
617               state == seenCT3)
618        begin
619         en_wc = 0;
620         cl_wc = 0;
621         en_pc = 0;
622         cl_pc = 0;
623         en_lc = 0;
624         cl_lc = 0;
625         re_p = 1;
626         re_s = 0;
627         en_tmp = 1;
628         ld_tmp = 0;
629         sel_tmp = 1;
630       end
631
632     else if (state == loadTmp2 ||
633              state == loadTmp3)
634       begin
635         en_wc = 0;
636         cl_wc = 0;
637         en_pc = 0;
638         cl_pc = 0;
```

```
639         en_lc = 0;
640         cl_lc = 0;
641         re_p = 1;
642         re_s = 0;
643         en_tmp = 1;
644         ld_tmp = 1;
645         sel_tmp = 1;
646       end
647
648
649     end
650
651 endmodule: fsm2
652
653
```

Lab Code [10 points]
Filename: fsm3.sv

```
`default_nettype none

module fsm3
  (input logic [2:0] fsm_notif, patternSignal,
   input logic end_seq, len_reached,
   input logic ready,
   input logic clock, reset_N,
   output logic en_pc, cl_pc, re_p, re_s, en_wc, cl_wc,
                cl_lc, en_lc, done, found_it, error, ld_wc,
                ld_pc, ld_fc, en_fc, start_sel, cl_tmp, en_tmp,
                ld_tmp, sel_tmp);

   enum logic [5:0] {start = 6'b000000, readLetPat = 6'b000001,
                     checkPattern = 6'b000010, compTwoFirst = 6'b000011,
                     compTwoSec = 6'b000100, compThreeFirst = 6'b000101,
                     compThreeSec = 6'b000110, compThreeThird = 6'b000111,
                     oneMatchUpTo = 6'b001001, incPatFinish = 6'b010111,
                     doneOneLeft = 6'b001010, doneTwoLeft = 6'b001011,
                     zeroMatchUpTo = 6'b011100, oneMatchAll = 6'b011101,
                     incLetPat = 6'b001100, endNoGood = 6'b001101,
                     Error = 6'b001111, incPat21 = 6'b010000,
                     incPat22 = 6'b010001, incPat31 = 6'b010010,
                     incPat32 = 6'b010011, incPat33 = 6'b010100,
                     incPatUpTo = 6'b010101, incPatAll = 6'b010110,
                     good = 6'b011110, incWordAll = 6'b011111,
                     incWordZeroUp = 6'b001000, incWordOneUp = 6'b001110,
                     seenA2 = 6'b100001, seenC2 = 6'b100010,
                     seenT2 = 6'b100011, seenG2 = 6'b100100,
                     seenAC3 = 6'b100101, seenAT3 = 6'b100110,
                     seenAG3 = 6'b100111, seenCT3 = 6'b101000,
                     seenCG3 = 6'b101001, seenTG3 = 6'b101010,
                     seenA3 = 6'b101011, seenC3 = 6'b101100,
                     seenT3 = 6'b101101, seenG3 = 6'b101110,
                     check21 = 6'b111001, check31 = 6'b111010,
                     loadTmp2 = 6'b111011, loadTmp3 = 6'b111100,
                     incFound = 6'b111101}
                     state, nextState;

   // next state logic
   always_comb begin
     if (state != start && fsm_notif == 2)
       nextState = Error;
     else if (end_seq && fsm_notif != 7)
       nextState = Error;
       unique case (state)
        start : nextState = (ready) ? readLetPat : start;
        readLetPat : nextState = checkPattern;
        checkPattern : begin
                         if (fsm_notif == 0)
                           nextState = incLetPat;
                         else if (fsm_notif == 1) // no matches
                           nextState = endNoGood;
                         else if (fsm_notif == 2)
                           nextState = Error;
                         else if (fsm_notif == 3)
                           nextState = incPat21;
                         else if (fsm_notif == 4)
                           nextState = incPat31;
                         else if (fsm_notif == 5)
                           nextState = incPatAll;
                         else if (fsm_notif == 6)
                           nextState = incPatUpTo;
                         else if (fsm_notif == 7)
                           nextState = good;
                       end
        compTwoFirst : begin
                         if (fsm_notif == 7)
                           nextState = Error;
                         else if (fsm_notif == 0)
                           nextState = doneOneLeft;
```

```
 71                        else
 72                          nextState = incPat22; //bad
 73                       end
 74
 75        compTwoSec : begin
 76                        if (fsm_notif == 7)
 77                          nextState = Error;
 78                        else if (fsm_notif == 0)
 79                          nextState = incLetPat;
 80                        else
 81                          nextState = endNoGood; //bad
 82                      end
 83
 84         compThreeFirst : begin
 85                            if (fsm_notif == 7)
 86                              nextState = Error;
 87                            else if (fsm_notif == 0)
 88                              nextState = doneTwoLeft;
 89                            else
 90                              nextState = incPat32; //bad
 91                          end
 92
 93         compThreeSec : begin
 94                            if (fsm_notif == 7)
 95                              nextState = Error;
 96                            else if (fsm_notif == 0)
 97                              nextState = doneOneLeft;
 98                            else
 99                              nextState = incPat33; //bad
100                          end
101
102         compThreeThird : begin
103                            if (fsm_notif == 7)
104                              nextState = Error;
105                            else if (fsm_notif == 0)
106                              nextState = incLetPat;
107                            else
108                              nextState = endNoGood; //bad
109                          end
110
111         incPatAll : nextState = oneMatchAll;
112         oneMatchAll : begin
113                         if (fsm_notif == 0 && len_reached == 0)
114                           nextState = incWordAll;
115                         else if (len_reached == 1)
116                           nextState = incPatFinish;
117                         else if (fsm_notif == 1)
118                           nextState = endNoGood; // bad
119                       end
120
121         incWordAll : nextState = oneMatchAll;
122
123         incPatUpTo : nextState = zeroMatchUpTo;
124         zeroMatchUpTo : nextState =
125                        (fsm_notif == 0) ? oneMatchUpTo : endNoGood; // bad
126         incWordZeroUp: nextState = oneMatchUpTo;
127         oneMatchUpTo : begin
128                          if (fsm_notif == 0 && len_reached == 0)
129                            nextState = incWordOneUp;
130                          else
131                            nextState = incPatFinish;
132                        end
133         incWordOneUp : nextState = oneMatchUpTo;
134
135         incPatFinish: nextState = readLetPat;
136
137         doneOneLeft : nextState = incLetPat;
138         doneTwoLeft : nextState = doneOneLeft;
139
140         incLetPat : nextState = (end_seq) ? Error : readLetPat;
141
```

```
142         incPat21 : nextState = loadTmp2;
143         incPat22 : nextState = compTwoSec;
144         incPat31 : nextState = loadTmp3;
145         incPat32 : nextState = compThreeSec;
146         incPat33 : nextState = compThreeThird;
147
148         incFound : nextState = readLetPat;
149
150         loadTmp2 : nextState = check21;
151         loadTmp3 : nextState = check31;
152
153         // A = 3'b000, C = 3'b001, T = 3'b010, G = 3'b011,
154         // other = 3'b111
155         check21 : begin
156                     if (patternSignal == 0) // A
157                       nextState = seenA2;
158                     else if (patternSignal == 1) // C
159                       nextState = seenC2;
160                     else if (patternSignal == 2) // T
161                       nextState = seenT2;
162                     else if (patternSignal == 3) // G
163                       nextState = seenG2;
164                     else
165                       nextState = Error;
166                   end
167
168         check31 : begin
169                     if (patternSignal == 0) // A
170                       nextState = seenA3;
171                     else if (patternSignal == 1) // C
172                       nextState = seenC3;
173                     else if (patternSignal == 2) // T
174                       nextState = seenT3;
175                     else if (patternSignal == 3) // G
176                       nextState = seenG3;
177                     else
178                       nextState = Error;
179                   end
180
181         seenA2 : begin
182                     if (patternSignal == 0) // A
183                       nextState = Error;
184                     else if (patternSignal == 1) // C
185                       nextState = compTwoFirst;
186                     else if (patternSignal == 2) // T
187                       nextState = compTwoFirst;
188                     else if (patternSignal == 3) // G
189                       nextState = compTwoFirst;
190                     else
191                       nextState = Error;
192                   end
193
194         seenC2 : begin
195                     if (patternSignal == 0) // A
196                       nextState = compTwoFirst;
197                     else if (patternSignal == 1) // C
198                       nextState = Error;
199                     else if (patternSignal == 2) // T
200                       nextState = compTwoFirst;
201                     else if (patternSignal == 3) // G
202                       nextState = compTwoFirst;
203                     else
204                       nextState = Error;
205                   end
206
207         seenT2 : begin
208                     if (patternSignal == 0) // A
209                       nextState = compTwoFirst;
210                     else if (patternSignal == 1) // C
211                       nextState = compTwoFirst;
212                     else if (patternSignal == 2) // T
```

```
213                       nextState = Error;
214                     else if (patternSignal == 3) // G
215                       nextState = compTwoFirst;
216                     else
217                       nextState = Error;
218                   end
219
220           seenG2 : begin
221                     if (patternSignal == 0) // A
222                       nextState = compTwoFirst;
223                     else if (patternSignal == 1) // C
224                       nextState = compTwoFirst;
225                     else if (patternSignal == 2) // T
226                       nextState = compTwoFirst;
227                     else if (patternSignal == 3) // G
228                       nextState = Error;
229                     else
230                       nextState = Error;
231                   end
232
233
234           seenA3 : begin
235                     if (patternSignal == 0) // A
236                       nextState = Error;
237                     else if (patternSignal == 1) // C
238                       nextState = seenAC3;
239                     else if (patternSignal == 2) // T
240                       nextState = seenAT3;
241                     else if (patternSignal == 3) // G
242                       nextState = seenAG3;
243                     else
244                       nextState = Error;
245                   end
246
247           seenC3 : begin
248                     if (patternSignal == 0) // A
249                       nextState = seenAC3;
250                     else if (patternSignal == 1) // C
251                       nextState = Error;
252                     else if (patternSignal == 2) // T
253                       nextState = seenCT3;
254                     else if (patternSignal == 3) // G
255                       nextState = seenCG3;
256                     else
257                       nextState = Error;
258                   end
259
260           seenT3 : begin
261                     if (patternSignal == 0) // A
262                       nextState = seenAT3;
263                     else if (patternSignal == 1) // C
264                       nextState = seenCT3;
265                     else if (patternSignal == 2) // T
266                       nextState = Error;
267                     else if (patternSignal == 3) // G
268                       nextState = seenTG3;
269                     else
270                       nextState = Error;
271                   end
272
273           seenG3 : begin
274                     if (patternSignal == 0) // A
275                       nextState = seenAG3;
276                     else if (patternSignal == 1) // C
277                       nextState = seenCG3;
278                     else if (patternSignal == 2) // T
279                       nextState = seenTG3;
280                     else if (patternSignal == 3) // G
281                       nextState = Error;
282                     else
283                       nextState = Error;
```

```
284                     end
285
286             seenAC3 : begin
287                         if (patternSignal == 0) // A
288                           nextState = Error;
289                         else if (patternSignal == 1) // C
290                           nextState = Error;
291                         else if (patternSignal == 2) // T
292                           nextState = compThreeFirst;
293                         else if (patternSignal == 3) // G
294                           nextState = compThreeFirst;
295                         else
296                           nextState = Error;
297                       end
298
299             seenAT3 : begin
300                         if (patternSignal == 0) // A
301                           nextState = Error;
302                         else if (patternSignal == 1) // C
303                           nextState = compThreeFirst;
304                         else if (patternSignal == 2) // T
305                           nextState = Error;
306                         else if (patternSignal == 3) // G
307                           nextState = compThreeFirst;
308                         else
309                           nextState = Error;
310                       end
311
312             seenAG3 : begin
313                         if (patternSignal == 0) // A
314                           nextState = Error;
315                         else if (patternSignal == 1) // C
316                           nextState = compThreeFirst;
317                         else if (patternSignal == 2) // T
318                           nextState = compThreeFirst;
319                         else if (patternSignal == 3) // G
320                           nextState = Error;
321                         else
322                           nextState = Error;
323                       end
324
325             seenCT3 : begin
326                         if (patternSignal == 0) // A
327                           nextState = compThreeFirst;
328                         else if (patternSignal == 1) // C
329                           nextState = Error;
330                         else if (patternSignal == 2) // T
331                           nextState = Error;
332                         else if (patternSignal == 3) // G
333                           nextState = compThreeFirst;
334                         else
335                           nextState = Error;
336                       end
337
338             seenCG3 : begin
339                         if (patternSignal == 0) // A
340                           nextState = compThreeFirst;
341                         else if (patternSignal == 1) // C
342                           nextState = Error;
343                         else if (patternSignal == 2) // T
344                           nextState = compThreeFirst;
345                         else if (patternSignal == 3) // G
346                           nextState = Error;
347                         else
348                           nextState = Error;
349                       end
350
351             seenTG3 : begin
352                         if (patternSignal == 0) // A
353                           nextState = compThreeFirst;
354                         else if (patternSignal == 1) // C
```

```
355                       nextState = compThreeFirst;
356                     else if (patternSignal == 2) // T
357                       nextState = Error;
358                     else if (patternSignal == 3) // G
359                       nextState = Error;
360                     else
361                       nextState = Error;
362                   end
363
364         endNoGood : nextState = (end_seq) ? Error : incFound;
365         Error : nextState = Error;
366         good : nextState = (end_seq) ? Error : incFound;
367         default : nextState = start;
368       endcase
369     end
370
371    always_ff @(posedge clock)
372     if (~reset_N)
373       state <= start;
374     else
375       state <= nextState;
376
377     // en_pc, cl_pc, re_p, re_s,
378     // en_wc, cl_wc, cl_lc, en_cl;
379    always_comb begin
380     done = 0;
381     found_it = 0;
382     error = 0;
383     en_wc = 0;
384     cl_wc = 0;
385     en_pc = 0;
386     cl_pc = 0;
387     en_lc = 0;
388     cl_lc = 0;
389     ld_wc = 0;
390     ld_pc = 0;
391     ld_fc = 0;
392     start_sel = 0;
393     en_fc = 0;
394     en_tmp = 0;
395     ld_tmp = 0;
396     cl_tmp = 0;
397     sel_tmp = 0;
398
399
400     if (state == start)
401       begin
402         en_wc = 1;
403         en_pc = 1;
404         cl_wc = 0;
405         cl_pc = 0;
406         en_lc = 1;
407         cl_lc = 1;
408         ld_wc = 1;
409         ld_pc = 1;
410         start_sel = 1;
411         en_fc = 1;
412         ld_fc = 1;
413         re_p = 0;
414         re_s = 0;
415       end
416
417     else if (state == readLetPat)
418       begin
419         re_s = 1;
420         re_p = 1;
421         en_wc = 0;
422         en_pc = 0;
423         cl_wc = 0;
424         cl_pc = 0;
425         en_lc = 0;
```

```
426         cl_lc = 0;
427       end
428
429     else if (state == checkPattern ||
430              state == compTwoFirst ||
431              state == compTwoSec ||
432              state == compThreeFirst ||
433              state == compThreeSec ||
434              start == compThreeThird)
435       begin
436         en_wc = 0;
437         cl_wc = 0;
438         en_pc = 0;
439         cl_pc = 0;
440         en_lc = 0;
441         cl_lc = 0;
442         re_p = 1;
443         re_s = 1;
444       end
445
446     else if (state == doneOneLeft || state == doneTwoLeft)
447       begin // skip one pattern
448         en_wc = 0;
449         cl_wc = 0;
450         en_pc = 1;
451         cl_pc = 0;
452         en_lc = 0;
453         cl_lc = 0;
454         re_p = 0;
455         re_s = 0;
456       end
457
458       else if (state == check21 ||
459              state == check31)
460       begin
461         en_wc = 0;
462         cl_wc = 0;
463         en_pc = 0;
464         cl_pc = 0;
465         en_lc = 0;
466         cl_lc = 0;
467         re_p = 1;
468         re_s = 0;
469         ld_tmp = 0;
470         en_tmp = 1;
471         sel_tmp = 1;
472       end
473
474     else if (state == incLetPat)
475       begin
476         en_wc = 1;
477         cl_wc = 0;
478         en_pc = 1;
479         cl_pc = 0;
480         en_lc = 0;
481         cl_lc = 0;
482         re_p = 1;
483         re_s = 1;
484       end
485
486     else if (state == endNoGood)
487       begin
488         en_wc = 1;
489         cl_wc = 0;
490         en_pc = 1;
491         cl_pc = 0;
492         en_lc = 1;
493         cl_lc = 1;
494         ld_wc = 1;
495         ld_pc = 1;
496         ld_fc = 0;
```

```
497         en_fc = 1;
498         re_p = 1;
499         re_s = 1;
500         done = 0;
501         found_it = 0;
502         error = 0;
503       end
504
505     else if (state == Error)
506       begin
507         en_wc = 1;
508         cl_wc = 0;
509         en_pc = 1;
510         cl_pc = 0;
511         en_lc = 1;
512         cl_lc = 1;
513         ld_wc = 1;
514         ld_pc = 1;
515         re_p = 0;
516         re_s = 0;
517         done = 1;
518         found_it = 0;
519         error = 1;
520       end
521
522      else if (state == incPat21 ||
523                state == incPat31)
524       begin
525         en_wc = 0;
526         cl_wc = 0;
527         en_pc = 1;
528         cl_pc = 0;
529         en_lc = 0;
530         cl_lc = 0;
531         re_p = 1;
532         re_s = 1;
533         ld_tmp = 1;
534         en_tmp = 1;
535         sel_tmp = 1;
536       end
537
538     else if (state == incPat22 ||
539              state == incPat32 ||
540              state == incPat32 ||
541              state == incPat33 ||
542              state == incPatFinish)
543       begin
544         en_wc = 0;
545         cl_wc = 0;
546         en_pc = 1;
547         cl_pc = 0;
548         en_lc = 0;
549         cl_lc = 0;
550         re_p = 1;
551         re_s = 1;
552       end
553
554     else if (state == incPatUpTo ||
555              state == incPatAll)
556       begin
557         en_wc = 0;
558         cl_wc = 0;
559         en_pc = 1;
560         cl_pc = 0;
561         en_lc = 1;
562         cl_lc = 1;
563         re_p = 1;
564         re_s = 1;
565       end
566
567     else if (state == oneMatchAll ||
```

```
568               state == zeroMatchUpTo ||
569               state == oneMatchUpTo)
570         begin
571           en_wc = 0;
572           cl_wc = 0;
573           en_pc = 0;
574           cl_pc = 0;
575           en_lc = 0;
576           cl_lc = 0;
577           re_p = 1;
578           re_s = 1;
579         end
580
581       else if (state == incWordOneUp ||
582               state == incWordZeroUp ||
583               state == incWordAll)
584         begin
585           en_wc = 1;
586           cl_wc = 0;
587           en_pc = 0;
588           cl_pc = 0;
589           en_lc = 1;
590           cl_lc = 0;
591           re_p = 1;
592           re_s = 1;
593         end
594
595       else if (state == good)
596         begin
597           en_wc = 1;
598           cl_wc = 0;
599           en_pc = 1;
600           cl_pc = 0;
601           en_lc = 1;
602           cl_lc = 1;
603           ld_pc = 1;
604           ld_wc = 1;
605           ld_fc = 0;
606           re_p = 1;
607           re_s = 1;
608           en_fc = 1;
609           done = 0;
610           found_it = 1;
611           error = 0;
612         end
613
614       else if (state == incFound)
615         begin
616           en_wc = 0;
617           cl_wc = 0;
618           en_pc = 0;
619           cl_pc = 0;
620           en_lc = 0;
621           cl_lc = 0;
622           ld_pc = 0;
623           ld_wc = 0;
624           ld_fc = 0;
625           re_p = 1;
626           re_s = 1;
627           en_fc = 0;
628         end
629
630       else if (state == seenA2 ||
631               state == seenA3 ||
632               state == seenC2 ||
633               state == seenC3 ||
634               state == seenT2 ||
635               state == seenT3 ||
636               state == seenG2 ||
637               state == seenG2 ||
638               state == seenAC3 ||
```

```
639              state == seenAG3 ||
640              state == seenAT3 ||
641              state == seenTG3 ||
642              state == seenCG3 ||
643              state == seenCT3)
644         begin
645          en_wc = 0;
646          cl_wc = 0;
647          en_pc = 0;
648          cl_pc = 0;
649          en_lc = 0;
650          cl_lc = 0;
651          re_p = 1;
652          re_s = 0;
653          en_tmp = 1;
654          ld_tmp = 0;
655          sel_tmp = 1;
656        end
657
658     else if (state == loadTmp2 ||
659              state == loadTmp3)
660       begin
661         en_wc = 0;
662         cl_wc = 0;
663         en_pc = 0;
664         cl_pc = 0;
665         en_lc = 0;
666         cl_lc = 0;
667         re_p = 1;
668         re_s = 0;
669         en_tmp = 1;
670         ld_tmp = 1;
671         sel_tmp = 1;
672       end
673
674
675     end
676
677 endmodule: fsm3
678
679
```

Lab Code [10 points]
Filename: lab5p1.sv

```
  1 `default_nettype none
  2
  3 module lab5p1
  4   (input logic ready,
  5    input logic [15:0] dna_length,
  6    input logic clock, reset_N,
  7    output logic done, found_it, error);
  8
  9    logic [7:0] pattern;
 10    logic [1:0] nuc;
 11    logic [2:0] fsm_notif;
 12    logic [3:0] how_much;
 13    logic end_seq, len_reached, en_pc, cl_pc, re_p,
 14          re_s, en_wc, cl_wc, cl_lc, en_lc;
 15    logic seqLt, seqEq, seqGt;
 16    logic lenLt, lenEq, lenGt;
 17    logic [15:0] WordCount;
 18    logic [11:0] PatternCount;
 19    logic [3:0] LetterCount;
 20    logic [2:0] patternSignal;
 21    logic [11:0] checkPatAhead, MuxedPatCount, regCheckAhead;
 22    logic sel_tmp, ld_tmp, cl_tmp, en_tmp;
 23
 24
 25    fsm FSM (.*);
 26
 27    MemoryNucs Seq_Mem (.re(re_s), .we(1'b0), .clock,
 28                                      .Addr(WordCount), .Data(nuc));
 29    MemoryPattern Pattern_Mem (.re(re_p), .we(1'b0), .clock,
 30                        .Addr(MuxedPatCount), .Data(pattern));
 31
 32    PatternChecker PatCheck (.*);
 33
 34    Counter #(16) WordCounter (.en(en_wc), .clear(cl_wc), .load(1'b0),
 35                               .up(1'b1), .clock,
 36                               .D(WordCount), .Q(WordCount));
 37
 38    Counter #(12) PatternCounter (.en(en_pc), .clear(cl_pc), .load(1'b0),
 39                                  .up(1'b1), .clock,
 40                                  .D(PatternCount), .Q(PatternCount));
 41
 42    Counter #(4) LetterCounter (.en(en_lc), .clear(cl_lc), .load(1'b0),
 43                                .up(1'b1), .clock,
 44                                .D(LetterCount), .Q(LetterCount));
 45
 46    Counter #(12) CheckAheadCounter (.en(en_tmp), .clear(cl_tmp), .load(ld_tmp),
 47                                     .up(1'b1), .clock, .D(regCheckAhead),
 48                                     .Q(checkPatAhead));
 49
 50    //Register #(12) PatCountReg (.en(1'b1), .clear(1'b0), .D(regCheckAhead), ...
Line length of 96 (max is 80)
 51
 52
 53    MagComp #(16) WordComp (.A(WordCount), .B(dna_length),
 54                            .AltB(seqLt), .AeqB(seqEq), .AgtB(seqGt));
 55
 56    MagComp #(4) LetterComp (.A(LetterCount), .B(how_much),
 57                             .AltB(lenLt), .AeqB(lenEq), .AgtB(lenGt));
 58
 59    Mux2to1 #(12) PatCountLoadMux (.S(ld_tmp), .I0(checkPatAhead), .I1(Pattern...
Line length of 84 (max is 80)
 60                                .Y(regCheckAhead));
 61
 62    Mux2to1 #(12) PatCountMux (.S(sel_tmp), .I0(PatternCount), .I1(checkPatAhe...
Line length of 81 (max is 80)
 63                                .Y(MuxedPatCount));
 64
 65
 66
 67
```

```
 68    assign len_reached = (lenEq) ? 1'b1 : 1'b0;
 69    assign end_seq = (seqEq) ? 1'b1 : 1'b0;
 70
 71 endmodule: lab5p1
 72
 73
 74
```

Lab Code [10 points]
Filename: lab5p1_test.sv

```
 1 `default_nettype none
 2
 3 module lab5p1_test ();
 4   logic ready;
 5   logic [15:0] dna_length;
 6   logic clock, reset_N;
 7   logic done, found_it, error;
 8
 9   lab5p1 psm(.*);
10   initial begin
11     $monitor($time,, "ready=%b, reset_N = %b, done=%b, error=%b\
12                       state=%s, found_it=%b, dna_length=%d \n\
13                       nuc=%b, pat=%h, wordCount=%d, wordEq=%b\
14                       fsm_notif=%b, nextState=%s, patCount=%d\n\
15                       en_wc=%b, en_pc=%b, cl_wc=%b, cl_pc=%b\
16                       re_p=%b, re_s=%b, en_tmp=%b, ld_tmp=%b\n\
17                       muxedPatCount=%d, checkAhead=%d\
18                       patternSignal=%d",
19                       ready, reset_N, done, error, psm.FSM.state.name,
20                       found_it, dna_length, psm.nuc, psm.pattern,
21                       psm.WordCount, psm.seqEq, psm.fsm_notif,
22                       psm.FSM.nextState.name, psm.PatternCount,
23                       psm.en_wc, psm.en_pc, psm.cl_wc, psm.cl_pc,
24                       psm.re_p, psm.re_s, psm.en_tmp, psm.ld_tmp,
25                       psm.MuxedPatCount,
26                       psm.checkPatAhead, psm.patternSignal);
27     clock = 0;
28     forever #5 clock = ~clock;
29   end
30
31   initial begin
32     reset_N = 0;
33     dna_length = 4;
34     ready = 0;
35     @(posedge clock);
36     reset_N = 1;
37     @(posedge clock);
38     ready = 1;
39     @(posedge clock);
40     ready = 0;
41     wait(done);
42     @(posedge clock);
43     @(posedge clock);
44     @(posedge clock);
45     $finish;
46    end
47
48 endmodule: lab5p1_test
49
```

Lab Code [10 points]
Filename: lab5p2.sv

```
  1 `default_nettype none
  2
  3 module lab5p2
  4   (input logic ready,
  5    input logic [15:0] dna_length,
  6    input logic clock, reset_N,
  7    input logic [15:0] dna_start,
  8    input logic [11:0] pattern_start,
  9    output logic done, found_it, error);
 10
 11    logic [7:0] pattern;
 12    logic [1:0] nuc;
 13    logic [2:0] fsm_notif;
 14    logic [3:0] how_much;
 15    logic end_seq, len_reached, en_pc, cl_pc, re_p,
 16          re_s, en_wc, cl_wc, cl_lc, en_lc, ld_wc, ld_pc;
 17    logic seqLt, seqEq, seqGt;
 18    logic lenLt, lenEq, lenGt;
 19    logic [15:0] WordCount, MuxedWord;
 20    logic [11:0] PatternCount, MuxedPat;
 21    logic [3:0] LetterCount;
 22    logic [2:0] patternSignal;
 23    logic [11:0] checkPatAhead, MuxedPatCount, regCheckAhead;
 24    logic sel_tmp, ld_tmp, cl_tmp, en_tmp;
 25
 26    fsm2 FSM (.*);
 27
 28    MemoryNucs Seq_Mem (.re(re_s), .we(1'b0), .clock,
 29                                      .Addr(WordCount), .Data(nuc));
 30    MemoryPattern Pattern_Mem (.re(re_p), .we(1'b0), .clock,
 31                        .Addr(MuxedPat), .Data(pattern));
 32
 33    PatternChecker PatCheck (.*);
 34
 35    Counter #(16) WordCounter (.en(en_wc), .clear(cl_wc), .load(ld_wc),
 36                               .up(1'b1), .clock,
 37                               .D(MuxedWord), .Q(WordCount));
 38
 39    Counter #(12) PatternCounter (.en(en_pc), .clear(cl_pc), .load(ld_pc),
 40                                  .up(1'b1), .clock,
 41                                  .D(MuxedPat), .Q(PatternCount));
 42
 43    Counter #(4) LetterCounter (.en(en_lc), .clear(cl_lc), .load(1'b0),
 44                                .up(1'b1), .clock,
 45                                .D(LetterCount), .Q(LetterCount));
 46
 47    Counter #(12) CheckAheadCounter (.en(en_tmp), .clear(cl_tmp), .load(ld_tmp),
 48                                     .up(1'b1), .clock, .D(regCheckAhead),
 49                                     .Q(checkPatAhead));
 50
 51    MagComp #(16) WordComp (.A(WordCount), .B(dna_length),
 52                            .AltB(seqLt), .AeqB(seqEq), .AgtB(seqGt));
 53
 54    MagComp #(4) LetterComp (.A(LetterCount), .B(how_much),
 55                             .AltB(lenLt), .AeqB(lenEq), .AgtB(lenGt));
 56
 57    Mux2to1 #(16) WordMux (.S(ld_wc), .I0(WordCount),
 58                           .I1(dna_start), .Y(MuxedWord));
 59
 60
 61
 62    Mux2to1 #(12) PatMux (.S(ld_pc), .I0(MuxedPatCount),
 63                          .I1(pattern_start), .Y(MuxedPat));
 64
 65    Mux2to1 #(12) PatCountLoadMux (.S(ld_tmp), .I0(checkPatAhead), .I1(Pattern...
Line length of 84 (max is 80)
 66                                .Y(regCheckAhead));
 67
 68    Mux2to1 #(12) PatCountMux (.S(sel_tmp), .I0(PatternCount), .I1(checkPatAhe...
Line length of 81 (max is 80)
```

```
 69                                .Y(MuxedPatCount));
 70
 71
 72    assign len_reached = (lenEq) ? 1'b1 : 1'b0;
 73    assign end_seq = (seqEq) ? 1'b1 : 1'b0;
 74
 75 endmodule: lab5p2
 76
 77
 78
```

Lab Code [10 points]
Filename: lab5p2_test.sv

```
 1 `default_nettype none
 2
 3 module lab5p2_test ();
 4   logic ready;
 5   logic [15:0] dna_length, dna_start;
 6   logic [11:0] pattern_start;
 7   logic clock, reset_N;
 8   logic done, found_it, error;
 9
10   lab5p2 psm(.*);
11   initial begin
12     $monitor($time,, "ready=%b, reset_N = %b, done=%b, error=%b\
13                       state=%s, found_it=%b, dna_length=%d \n\
14                       nuc=%b, pat=%h, wordCount=%d, wordEq=%b\
15                       fsm_notif=%b, nextState=%s, muxedpatCount=%d\n\
16                       en_wc=%b, en_pc=%b, cl_wc=%b, cl_pc=%b\
17                       re_p=%b, re_s=%b, en_tmp=%b, ld_tmp=%b\n\
18                       muxedPatCount=%d, checkAhead=%d\
19                       patternSignal=%d, ld_wc=%b, ld_pc=%b\
20                       patCount=%d",
21                       ready, reset_N, done, error, psm.FSM.state.name,
22                       found_it, dna_length, psm.nuc, psm.pattern,
23                       psm.WordCount, psm.seqEq, psm.fsm_notif,
24                       psm.FSM.nextState.name, psm.MuxedPat,
25                       psm.en_wc, psm.en_pc, psm.cl_wc, psm.cl_pc,
26                       psm.re_p, psm.re_s, psm.en_tmp, psm.ld_tmp,
27                       psm.MuxedPatCount, psm.checkPatAhead,
28                       psm.patternSignal, psm.ld_wc, psm.ld_pc,
29                       psm.PatternCount);
30     clock = 0;
31     forever #5 clock = ~clock;
32   end
33
34   initial begin
35     reset_N = 0;
36     dna_start = 5;
37     pattern_start = 104;
38     dna_length = 45;
39     ready = 0;
40     @(posedge clock);
41     reset_N = 1;
42     @(posedge clock);
43     ready = 1;
44     @(posedge clock);
45     ready = 0;
46     wait(done);
47     @(posedge clock);
48     @(posedge clock);
49     @(posedge clock);
50     $finish;
51    end
52
53 endmodule: lab5p2_test
54
```

Lab Code [10 points]
Filename: lab5p3.sv

```
 1 `default_nettype none
 2
 3 module lab5p3
 4   (input logic ready,
 5    input logic [15:0] dna_length,
 6    input logic clock, reset_N,
 7    input logic [15:0] dna_start,
 8    input logic [11:0] pattern_start,
 9    output logic done, found_it, error,
10    output logic [15:0] found_location);
11
12    logic [7:0] pattern;
13    logic [1:0] nuc;
14    logic [2:0] fsm_notif;
15    logic [3:0] how_much;
16    logic end_seq, len_reached, en_pc, cl_pc, re_p,
17          re_s, en_wc, cl_wc, cl_lc, en_lc, ld_wc, ld_pc,
18          en_fc, ld_fc;
19    logic seqLt, seqEq, seqGt;
20    logic lenLt, lenEq, lenGt;
21    logic [15:0] WordCount, MuxedWord;
22    logic [11:0] PatternCount, MuxedPat;
23    logic [15:0] MuxedStart, next_start, RegStart;
24    logic [3:0] LetterCount;
25    logic [2:0] patternSignal;
26    logic [11:0] checkPatAhead, MuxedPatCount, regCheckAhead;
27    logic sel_tmp, ld_tmp, cl_tmp, en_tmp;
28    logic start_sel;
29
30
31    fsm3 FSM (.*);
32
33    MemoryNucs Seq_Mem (.re(re_s), .we(1'b0), .clock,
34                                       .Addr(WordCount), .Data(nuc));
35
36    MemoryPattern Pattern_Mem (.re(re_p), .we(1'b0), .clock,
37                         .Addr(MuxedPat), .Data(pattern));
38
39    PatternChecker PatCheck (.*);
40
41    Counter #(16) WordCounter (.en(en_wc), .clear(cl_wc), .load(ld_wc),
42                               .up(1'b1), .clock,
43                               .D(MuxedWord), .Q(WordCount));
44
45    Counter #(12) PatternCounter (.en(en_pc), .clear(cl_pc), .load(ld_pc),
46                                  .up(1'b1), .clock,
47                                  .D(MuxedPat), .Q(PatternCount));
48
49    Counter #(16) FoundCounter (.en(en_fc), .clear(1'b0), .load(ld_fc),
50                                .up(1'b1), .clock,
51                                .D(MuxedStart), .Q(next_start));
52
53    Register #(16) NextStartRef (.en(1'b1), .clear(1'b0),
54                                 .clock, .D(next_start), .Q(RegStart));
55
56    Counter #(4) LetterCounter (.en(en_lc), .clear(cl_lc), .load(1'b0),
57                                .up(1'b1), .clock,
58                                .D(LetterCount), .Q(LetterCount));
59
60    Counter #(12) CheckAheadCounter (.en(en_tmp), .clear(cl_tmp), .load(ld_tmp),
61                                     .up(1'b1), .clock, .D(regCheckAhead),
62                                     .Q(checkPatAhead));
63
64    MagComp #(16) WordComp (.A(WordCount), .B(dna_length),
65                            .AltB(seqLt), .AeqB(seqEq), .AgtB(seqGt));
66
67    MagComp #(4) LetterComp (.A(LetterCount), .B(how_much),
68                             .AltB(lenLt), .AeqB(lenEq), .AgtB(lenGt));
69
70    Mux2to1 #(12) PatCountLoadMux (.S(ld_tmp), .I0(checkPatAhead), .I1(Pattern...
Line length of 84 (max is 80)
```

```
 71                                .Y(regCheckAhead));
 72
 73    Mux2to1 #(12) PatCountMux (.S(sel_tmp), .I0(PatternCount), .I1(checkPatAhe...
Line length of 81 (max is 80)
 74                                .Y(MuxedPatCount));
 75
 76    Mux2to1 #(12) PatMux (.S(ld_pc), .I0(MuxedPatCount),
 77                          .I1(pattern_start), .Y(MuxedPat));
 78
 79    Mux2to1 #(16) WordMux (.S(ld_wc), .I0(WordCount),
 80                           .I1(MuxedStart), .Y(MuxedWord));
 81
 82    Mux2to1 #(16) StartMux (.S(start_sel), .I0(RegStart),
 83                            .I1(dna_start), .Y(MuxedStart));
 84
 85    always_comb begin
 86         if (found_it && next_start > 0)
 87             found_location = next_start-1;
 88         else if (found_it)
 89             found_location = next_start;
 90         else
 91             found_location = 16'bx;
 92      end
 93    assign len_reached = (lenEq) ? 1'b1 : 1'b0;
 94    assign end_seq = (next_start - 1 == dna_length) ? 1'b1 : 1'b0;
 95
 96 endmodule: lab5p3
 97
 98
 99
```

Lab Code [10 points]
Filename: lab5p3_test.sv

```
 1 `default_nettype none
 2
 3 module lab5p3_test ();
 4   logic ready;
 5   logic [15:0] dna_length, dna_start;
 6   logic [11:0] pattern_start;
 7   logic clock, reset_N;
 8   logic done, found_it, error;
 9   logic [15:0] found_location;
10
11   lab5p3 psm (.*);
12
13   initial begin
14     $monitor($time,, "ready=%b, reset_N = %b, done=%b, error=%b\
15                       state=%s, found_it=%b, dna_length=%d\n\
16                       nuc=%b, pat=%h, wordCount=%d, wordEq=%b\
17                       fsm_notif=%b, nextState=%s, patCount=%d\n\
18                       en_wc=%b, en_pc=%b, ld_pc=%b, ld_wc=%b\n\
19                       ld_fc=%b, next_start=%d, dna_start=%d,\
20                       start_sel=%b, found_loc=%d, muxedStart=%b\
21                       en_fc=%b, checkAhead=%d, ld_tmp=%b, en_tmp = %b\
22                       muxedPatCount=%d, patternSignal=%d",
23                       ready, reset_N, done, error, psm.FSM.state.name,
24                       found_it, dna_length, psm.nuc, psm.pattern,
25                       psm.WordCount, psm.seqEq, psm.fsm_notif,
26                       psm.FSM.nextState.name, psm.PatternCount,
27                       psm.en_wc, psm.en_pc, psm.ld_pc, psm.ld_wc,
28                       psm.ld_fc, psm.next_start, dna_start,
29                       psm.start_sel, found_location, psm.MuxedStart,
30                       psm.en_fc, psm.checkPatAhead, psm.ld_tmp, psm.en_tmp,
31                       psm.MuxedPatCount, psm.patternSignal);
32     clock = 0;
33     forever #5 clock = ~clock;
34   end
35
36   initial begin
37     reset_N = 0;
38     dna_start = 1;
39     pattern_start = 0;
40     dna_length = 19;
41     ready = 0;
42     @(posedge clock);
43     reset_N = 1;
44     @(posedge clock);
45     ready = 1;
46     @(posedge clock);
47     ready = 0;
48     wait(done);
49     @(posedge clock);
50 //    wait(done);
51 //    @(posedge clock);
52 //    wait(done);
53 //    @(posedge clock);
54 //    wait(done);
55 //    @(posedge clock);
56 //    wait(done);
57 //    @(posedge clock);
58 //    wait(done)
59 //    @(posedge clock);
60 //    wait(done);
61 //    @(posedge clock);
62     $finish;
63    end
64
65 endmodule: lab5p3_test
66
```

Lab Code [10 points]
Filename: library.sv

```
`default_nettype none

module MagComp
  # (parameter WIDTH = 1)
  (input logic [WIDTH-1:0] A, B,
   output logic AltB, AeqB, AgtB);

  always_comb begin
    if (A < B)
      begin
        AeqB = 1'b0;
        AltB = 1'b1;
        AgtB = 1'b0;
      end
    else if (A == B)
      begin
        AeqB = 1'b1;
        AltB = 1'b0;
        AgtB = 1'b0;
      end
    else
      begin
        AgtB = 1'b1;
        AltB = 1'b0;
        AeqB = 1'b0;
      end
  end

endmodule: MagComp


module Multiplexer
  # (parameter WIDTH = 2)
  // parameter S_WIDTH = $clog2(WIDTH);
  (input logic [WIDTH-1:0] I,
   input logic [$clog2(WIDTH)-1:0] S,
   output logic Y);

  always_comb begin
    if (S < WIDTH)
      Y = I[S];
  end

endmodule: Multiplexer

module Mux2to1
  // input width
  # (parameter WIDTH = 2)
  (input logic [WIDTH-1:0] I0, I1,
   input logic S,
   output logic [WIDTH-1:0] Y);

  assign Y = S ? I1 : I0;

endmodule: Mux2to1

module Decoder
    # (parameter WIDTH = 1)
    // parameter I_WIDTH = $clog2(WIDTH);
    (input logic [$clog2(WIDTH)-1:0] I,
     input logic en,
     output logic [WIDTH-1:0] D);

    always_comb begin
      D = 0;
      if (en)
        begin
          if (I < WIDTH)
            D[I] = 1'b1;
        end
```

```
 71       else
 72         D = I;
 73     end
 74
 75 endmodule: Decoder
 76
 77 module Adder
 78   # (parameter WIDTH = 1)
 79   (input logic [WIDTH-1:0] A, B,
 80    input logic Cin,
 81    output logic [WIDTH-1:0] S,
 82    output logic Cout);
 83
 84   assign {Cout, S} = A + B + Cin;
 85
 86 endmodule: Adder
 87
 88 module Register
 89   # (parameter WIDTH = 1)
 90   (input logic [WIDTH-1:0] D,
 91    input logic en, clear,
 92    input logic clock,
 93    output logic [WIDTH-1:0] Q);
 94
 95    always_ff @(posedge clock)
 96      if (en)
 97        Q <= clear ? 0 : D;
 98
 99 endmodule: Register
100
101 module Counter
102   #(parameter WIDTH = 1)
103    (input logic en, clear, load, up,
104     input logic clock,
105     input logic [WIDTH-1:0] D,
106     output logic [WIDTH-1:0] Q);
107
108   always_ff @(posedge clock)
109     if (clear && en)
110       Q <= 0;
111     else if (load && en)
112       Q <= D;
113     else if (up && en)
114       Q <= Q + 1;
115 endmodule: Counter
116
117 module ShiftRegister
118   #(parameter WIDTH = 1)
119    (input logic en, left, load,
120     input logic clock,
121     input logic [WIDTH-1:0] D,
122     output logic [WIDTH-1:0] Q);
123
124   always_ff @(posedge clock)
125     if (load)
126       Q <= D;
127     else if (en && left)
128       Q <= (Q << 1);
129     else if (en && ~left)
130       Q <= (Q >> 1);
131
132 endmodule: ShiftRegister
133
134 module BarrelShiftRegister
135   #(parameter WIDTH = 1)
136    (input logic load, en,
137     input logic [1:0] by,
138     input logic [WIDTH-1:0] D,
139     input logic clock,
140     output logic [WIDTH-1:0] Q);
141
```

```
142   always_ff @(posedge clock)
143     if (load)
144       Q <= D;
145     else if (en)
146       Q <= (Q << by);
147 endmodule: BarrelShiftRegister
148
149 module MemoryNucs
150   #(parameter DW = 2,
151               W = 65536,
152               AW = $clog2(W))
153    (input logic re, we, clock,
154     input logic [AW-1:0] Addr,
155     output logic [DW-1:0] Data);
156
157   logic [DW-1:0] M[W];
158   logic [DW-1:0] out;
159
160   assign Data = (re) ? out : 16'b0;
161
162   always @(posedge clock)
163     if (we)
164       M[Addr] <= Data;
165
166   initial
167     $readmemb("task2nuc.mem", M);
168
169   always_comb
170     out = M[Addr];
171
172 endmodule: MemoryNucs
173
174 module MemoryPattern
175   #(parameter DW = 8,
176               W = 4096,
177               AW = $clog2(W))
178    (input logic re, we, clock,
179     input logic [AW-1:0] Addr,
180     output logic [DW-1:0] Data);
181
182   logic [DW-1:0] M[W];
183   logic [DW-1:0] out;
184
185   assign Data = (re) ? out : 16'b0;
186
187   always @(posedge clock)
188     if (we)
189       M[Addr] <= Data;
190
191   initial
192     $readmemh("task2patts.mem", M);
193
194   always_comb
195     out = M[Addr];
196
197 endmodule: MemoryPattern
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
```

213
214
215
216
217

Lab Code [10 points]
Filename: library1.sv

```
 1 `default_nettype none
 2
 3 module MagComp
 4   # (parameter WIDTH = 1)
 5   (input logic [WIDTH-1:0] A, B,
 6    output logic AltB, AeqB, AgtB);
 7
 8   always_comb begin
 9     if (A < B)
10       begin
11         AeqB = 1'b0;
12         AltB = 1'b1;
13         AgtB = 1'b0;
14       end
15     else if (A == B)
16       begin
17         AeqB = 1'b1;
18         AltB = 1'b0;
19         AgtB = 1'b0;
20       end
21     else
22       begin
23         AgtB = 1'b1;
24         AltB = 1'b0;
25         AeqB = 1'b0;
26       end
27   end
28
29 endmodule: MagComp
30
31
32 module Multiplexer
33   # (parameter WIDTH = 2)
34   // parameter S_WIDTH = $clog2(WIDTH);
35   (input logic [WIDTH-1:0] I,
36    input logic [$clog2(WIDTH)-1:0] S,
37    output logic Y);
38
39   always_comb begin
40     if (S < WIDTH)
41       Y = I[S];
42   end
43
44 endmodule: Multiplexer
45
46 module Mux2to1
47   // input width
48   # (parameter WIDTH = 2)
49   (input logic [WIDTH-1:0] I0, I1,
50    input logic S,
51    output logic [WIDTH-1:0] Y);
52
53   assign Y = S ? I1 : I0;
54
55 endmodule: Mux2to1
56
57 module Decoder
58     # (parameter WIDTH = 1)
59     // parameter I_WIDTH = $clog2(WIDTH);
60     (input logic [$clog2(WIDTH)-1:0] I,
61      input logic en,
62      output logic [WIDTH-1:0] D);
63
64     always_comb begin
65       D = 0;
66       if (en)
67         begin
68           if (I < WIDTH)
69             D[I] = 1'b1;
70         end
```

```
 71       else
 72         D = I;
 73     end
 74
 75 endmodule: Decoder
 76
 77 module Adder
 78   # (parameter WIDTH = 1)
 79   (input logic [WIDTH-1:0] A, B,
 80    input logic Cin,
 81    output logic [WIDTH-1:0] S,
 82    output logic Cout);
 83
 84   assign {Cout, S} = A + B + Cin;
 85
 86 endmodule: Adder
 87
 88 module Register
 89   # (parameter WIDTH = 1)
 90   (input logic [WIDTH-1:0] D,
 91    input logic en, clear,
 92    input logic clock,
 93    output logic [WIDTH-1:0] Q);
 94
 95    always_ff @(posedge clock)
 96      if (en)
 97        Q <= clear ? 0 : D;
 98
 99 endmodule: Register
100
101 module Counter
102   #(parameter WIDTH = 1)
103    (input logic en, clear, load, up,
104     input logic clock,
105     input logic [WIDTH-1:0] D,
106     output logic [WIDTH-1:0] Q);
107
108   always_ff @(posedge clock)
109     if (clear && en)
110       Q <= 0;
111     else if (load && en)
112       Q <= D;
113     else if (up && en)
114       Q <= Q + 1;
115 endmodule: Counter
116
117 module ShiftRegister
118   #(parameter WIDTH = 1)
119    (input logic en, left, load,
120     input logic clock,
121     input logic [WIDTH-1:0] D,
122     output logic [WIDTH-1:0] Q);
123
124   always_ff @(posedge clock)
125     if (load)
126       Q <= D;
127     else if (en && left)
128       Q <= (Q << 1);
129     else if (en && ~left)
130       Q <= (Q >> 1);
131
132 endmodule: ShiftRegister
133
134 module BarrelShiftRegister
135   #(parameter WIDTH = 1)
136    (input logic load, en,
137     input logic [1:0] by,
138     input logic [WIDTH-1:0] D,
139     input logic clock,
140     output logic [WIDTH-1:0] Q);
141
```

```
142   always_ff @(posedge clock)
143     if (load)
144       Q <= D;
145     else if (en)
146       Q <= (Q << by);
147 endmodule: BarrelShiftRegister
148
149 module MemoryNucs
150   #(parameter DW = 2,
151               W = 65536,
152               AW = $clog2(W))
153    (input logic re, we, clock,
154     input logic [AW-1:0] Addr,
155     output logic [DW-1:0] Data);
156
157   logic [DW-1:0] M[W];
158   logic [DW-1:0] out;
159
160   assign Data = (re) ? out : 16'b0;
161
162   always @(posedge clock)
163     if (we)
164       M[Addr] <= Data;
165
166   initial
167     $readmemb("task1checkoffnuc.mem", M);
168
169   always_comb
170     out = M[Addr];
171
172 endmodule: MemoryNucs
173
174 module MemoryPattern
175   #(parameter DW = 8,
176               W = 4096,
177               AW = $clog2(W))
178    (input logic re, we, clock,
179     input logic [AW-1:0] Addr,
180     output logic [DW-1:0] Data);
181
182   logic [DW-1:0] M[W];
183   logic [DW-1:0] out;
184
185   assign Data = (re) ? out : 16'b0;
186
187   always @(posedge clock)
188     if (we)
189       M[Addr] <= Data;
190
191   initial
192     $readmemh("task1test5.mem", M);
193
194   always_comb
195     out = M[Addr];
196
197 endmodule: MemoryPattern
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
```

213
214
215
216
217

Lab Code [10 points]
Filename: library2.sv

```
`default_nettype none

module MagComp
  # (parameter WIDTH = 1)
  (input logic [WIDTH-1:0] A, B,
   output logic AltB, AeqB, AgtB);

  always_comb begin
    if (A < B)
      begin
        AeqB = 1'b0;
        AltB = 1'b1;
        AgtB = 1'b0;
      end
    else if (A == B)
      begin
        AeqB = 1'b1;
        AltB = 1'b0;
        AgtB = 1'b0;
      end
    else
      begin
        AgtB = 1'b1;
        AltB = 1'b0;
        AeqB = 1'b0;
      end
  end

endmodule: MagComp


module Multiplexer
  # (parameter WIDTH = 2)
  // parameter S_WIDTH = $clog2(WIDTH);
  (input logic [WIDTH-1:0] I,
   input logic [$clog2(WIDTH)-1:0] S,
   output logic Y);

  always_comb begin
    if (S < WIDTH)
      Y = I[S];
  end

endmodule: Multiplexer

module Mux2to1
  // input width
  # (parameter WIDTH = 2)
  (input logic [WIDTH-1:0] I0, I1,
   input logic S,
   output logic [WIDTH-1:0] Y);

  assign Y = S ? I1 : I0;

endmodule: Mux2to1

module Decoder
    # (parameter WIDTH = 1)
    // parameter I_WIDTH = $clog2(WIDTH);
    (input logic [$clog2(WIDTH)-1:0] I,
     input logic en,
     output logic [WIDTH-1:0] D);

    always_comb begin
      D = 0;
      if (en)
        begin
          if (I < WIDTH)
            D[I] = 1'b1;
        end
```

```
 71       else
 72         D = I;
 73     end
 74
 75 endmodule: Decoder
 76
 77 module Adder
 78   # (parameter WIDTH = 1)
 79   (input logic [WIDTH-1:0] A, B,
 80    input logic Cin,
 81    output logic [WIDTH-1:0] S,
 82    output logic Cout);
 83
 84   assign {Cout, S} = A + B + Cin;
 85
 86 endmodule: Adder
 87
 88 module Register
 89   # (parameter WIDTH = 1)
 90   (input logic [WIDTH-1:0] D,
 91    input logic en, clear,
 92    input logic clock,
 93    output logic [WIDTH-1:0] Q);
 94
 95    always_ff @(posedge clock)
 96      if (en)
 97        Q <= clear ? 0 : D;
 98
 99 endmodule: Register
100
101 module Counter
102   #(parameter WIDTH = 1)
103    (input logic en, clear, load, up,
104     input logic clock,
105     input logic [WIDTH-1:0] D,
106     output logic [WIDTH-1:0] Q);
107
108   always_ff @(posedge clock)
109     if (clear && en)
110       Q <= 0;
111     else if (load && en)
112       Q <= D;
113     else if (up && en)
114       Q <= Q + 1;
115 endmodule: Counter
116
117 module ShiftRegister
118   #(parameter WIDTH = 1)
119    (input logic en, left, load,
120     input logic clock,
121     input logic [WIDTH-1:0] D,
122     output logic [WIDTH-1:0] Q);
123
124   always_ff @(posedge clock)
125     if (load)
126       Q <= D;
127     else if (en && left)
128       Q <= (Q << 1);
129     else if (en && ~left)
130       Q <= (Q >> 1);
131
132 endmodule: ShiftRegister
133
134 module BarrelShiftRegister
135   #(parameter WIDTH = 1)
136    (input logic load, en,
137     input logic [1:0] by,
138     input logic [WIDTH-1:0] D,
139     input logic clock,
140     output logic [WIDTH-1:0] Q);
141
```

```
142   always_ff @(posedge clock)
143     if (load)
144       Q <= D;
145     else if (en)
146       Q <= (Q << by);
147 endmodule: BarrelShiftRegister
148
149 module MemoryNucs
150   #(parameter DW = 2,
151               W = 65536,
152               AW = $clog2(W))
153    (input logic re, we, clock,
154     input logic [AW-1:0] Addr,
155     output logic [DW-1:0] Data);
156
157   logic [DW-1:0] M[W];
158   logic [DW-1:0] out;
159
160   assign Data = (re) ? out : 16'b0;
161
162   always @(posedge clock)
163     if (we)
164       M[Addr] <= Data;
165
166   initial
167     $readmemb("T1_1_Y.nuc", M);
168
169   always_comb
170     out = M[Addr];
171
172 endmodule: MemoryNucs
173
174 module MemoryPattern
175   #(parameter DW = 8,
176               W = 4096,
177               AW = $clog2(W))
178    (input logic re, we, clock,
179     input logic [AW-1:0] Addr,
180     output logic [DW-1:0] Data);
181
182   logic [DW-1:0] M[W];
183   logic [DW-1:0] out;
184
185   assign Data = (re) ? out : 16'b0;
186
187   always @(posedge clock)
188     if (we)
189       M[Addr] <= Data;
190
191   initial
192     $readmemh("T1_1.gpatt", M);
193
194   always_comb
195     out = M[Addr];
196
197 endmodule: MemoryPattern
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
```

213
214
215
216
217

Lab Code [10 points]
Filename: library3.sv

```
`default_nettype none

module MagComp
  # (parameter WIDTH = 1)
  (input logic [WIDTH-1:0] A, B,
   output logic AltB, AeqB, AgtB);

  always_comb begin
    if (A < B)
      begin
        AeqB = 1'b0;
        AltB = 1'b1;
        AgtB = 1'b0;
      end
    else if (A == B)
      begin
        AeqB = 1'b1;
        AltB = 1'b0;
        AgtB = 1'b0;
      end
    else
      begin
        AgtB = 1'b1;
        AltB = 1'b0;
        AeqB = 1'b0;
      end
  end

endmodule: MagComp


module Multiplexer
  # (parameter WIDTH = 2)
  // parameter S_WIDTH = $clog2(WIDTH);
  (input logic [WIDTH-1:0] I,
   input logic [$clog2(WIDTH)-1:0] S,
   output logic Y);

  always_comb begin
    if (S < WIDTH)
      Y = I[S];
  end

endmodule: Multiplexer

module Mux2to1
  // input width
  # (parameter WIDTH = 2)
  (input logic [WIDTH-1:0] I0, I1,
   input logic S,
   output logic [WIDTH-1:0] Y);

  assign Y = S ? I1 : I0;

endmodule: Mux2to1

module Decoder
    # (parameter WIDTH = 1)
    // parameter I_WIDTH = $clog2(WIDTH);
    (input logic [$clog2(WIDTH)-1:0] I,
     input logic en,
     output logic [WIDTH-1:0] D);

    always_comb begin
      D = 0;
      if (en)
        begin
          if (I < WIDTH)
            D[I] = 1'b1;
        end
```

```
 71       else
 72         D = I;
 73     end
 74
 75 endmodule: Decoder
 76
 77 module Adder
 78   # (parameter WIDTH = 1)
 79   (input logic [WIDTH-1:0] A, B,
 80    input logic Cin,
 81    output logic [WIDTH-1:0] S,
 82    output logic Cout);
 83
 84   assign {Cout, S} = A + B + Cin;
 85
 86 endmodule: Adder
 87
 88 module Register
 89   # (parameter WIDTH = 1)
 90   (input logic [WIDTH-1:0] D,
 91    input logic en, clear,
 92    input logic clock,
 93    output logic [WIDTH-1:0] Q);
 94
 95    always_ff @(posedge clock)
 96      if (en)
 97        Q <= clear ? 0 : D;
 98
 99 endmodule: Register
100
101 module Counter
102   #(parameter WIDTH = 1)
103    (input logic en, clear, load, up,
104     input logic clock,
105     input logic [WIDTH-1:0] D,
106     output logic [WIDTH-1:0] Q);
107
108   always_ff @(posedge clock)
109     if (clear && en)
110       Q <= 0;
111     else if (load && en)
112       Q <= D;
113     else if (up && en)
114       Q <= Q + 1;
115 endmodule: Counter
116
117 module ShiftRegister
118   #(parameter WIDTH = 1)
119    (input logic en, left, load,
120     input logic clock,
121     input logic [WIDTH-1:0] D,
122     output logic [WIDTH-1:0] Q);
123
124   always_ff @(posedge clock)
125     if (load)
126       Q <= D;
127     else if (en && left)
128       Q <= (Q << 1);
129     else if (en && ~left)
130       Q <= (Q >> 1);
131
132 endmodule: ShiftRegister
133
134 module BarrelShiftRegister
135   #(parameter WIDTH = 1)
136    (input logic load, en,
137     input logic [1:0] by,
138     input logic [WIDTH-1:0] D,
139     input logic clock,
140     output logic [WIDTH-1:0] Q);
141
```

```
142   always_ff @(posedge clock)
143     if (load)
144       Q <= D;
145     else if (en)
146       Q <= (Q << by);
147 endmodule: BarrelShiftRegister
148
149 module MemoryNucs
150   #(parameter DW = 2,
151               W = 65536,
152               AW = $clog2(W))
153    (input logic re, we, clock,
154     input logic [AW-1:0] Addr,
155     output logic [DW-1:0] Data);
156
157   logic [DW-1:0] M[W];
158   logic [DW-1:0] out;
159
160   assign Data = (re) ? out : 16'b0;
161
162   always @(posedge clock)
163     if (we)
164       M[Addr] <= Data;
165
166   initial
167     $readmemb("attempt1.nuc", M);
168
169   always_comb
170     out = M[Addr];
171
172 endmodule: MemoryNucs
173
174 module MemoryPattern
175   #(parameter DW = 8,
176               W = 4096,
177               AW = $clog2(W))
178    (input logic re, we, clock,
179     input logic [AW-1:0] Addr,
180     output logic [DW-1:0] Data);
181
182   logic [DW-1:0] M[W];
183   logic [DW-1:0] out;
184
185   assign Data = (re) ? out : 16'b0;
186
187   always @(posedge clock)
188     if (we)
189       M[Addr] <= Data;
190
191   initial
192     $readmemh("attempt1.gpatt", M);
193
194   always_comb
195     out = M[Addr];
196
197 endmodule: MemoryPattern
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
```

213
214
215
216
217

Lab Code [10 points]
Filename: pattern_checker.sv

```
 1 `default_nettype none
 2
 3 module PatternChecker
 4   #(parameter NW = 2,
 5               PW = 8)
 6   (input logic [NW-1:0] nuc,
 7    input logic [PW-1:0] pattern,
 8    output logic [2:0] fsm_notif,
 9    output logic [3:0] how_much,
10    output logic [2:0] patternSignal);
11    //patternSignal:
12    // A = 3'b000, C = 3'b001, T = 3'b010, G = 3'b011,
13    // other = 3'b111
14
15    // fsm_notif: match = 3'b000, no_match = 3'b001,
16    //                   error = 3'b010, next2 = 3'b011,
17    //                  next3 = 3'b100, exactlyN = 3'b101,
18    //                  upTo = 3'b110, end = 3'b111;
19    // how_much only active for exactlyN, upTo
20
21   always_comb begin
22     patternSignal = 3'b111;
23     casez (pattern)
24       6'h00 : fsm_notif = 3'b111;
25       6'h10 : begin
26                 fsm_notif = (nuc == 2'b00) ? 3'b000 : 3'b001;
27                 patternSignal = 3'b001;
28               end
29       6'h11 : begin
30                 fsm_notif = (nuc == 2'b01) ? 3'b000 : 3'b001;
31                 patternSignal = 3'b010;
32               end
33       6'h12 : begin
34                 fsm_notif = (nuc == 2'b10) ? 3'b000 : 3'b001;
35                 patternSignal = 3'b000;
36               end
37       6'h13 : begin
38                 fsm_notif = (nuc == 2'b11) ? 3'b000 : 3'b001;
39                 patternSignal = 3'b011;
40               end
41       6'h20 : fsm_notif = 3'b000;
42       6'h21 : fsm_notif = 3'b011;
43       6'h22 : fsm_notif = 3'b100;
44       6'h0? : begin
45                 fsm_notif = 3'b101;
46                 how_much = pattern[3:0];
47               end
48       6'h3? : begin
49                 fsm_notif = 3'b110;
50                 how_much = 16 - pattern[3:0];
51               end
52       default : fsm_notif = 3'b010;
53     endcase
54   end
55
56 endmodule: PatternChecker
57
58
59
60
```