ECE-111 Advanced Digital Design Term Project Assignment Deliverables

All other necessary working code .sv (including synthesis and simulation) files are included in the zip folder on gradescope submissions "*term_project_v2.zip*".

PART 1 DELIVERABLES:

(1) Working decoder.sv code:

decoder.sv ×

decoder.sv > ...

```
1   module decoder
2   (
3      input             clk,
4      input             rst,
5      input             enable,
6      input [1:0]       d_in,
7      output logic      d_out);
8
9   //   logic             decoder_o_reg;
10
11  //bmc module signals
12     wire  [1:0]       bmc000_path_0_bmc;
13     wire  [1:0]       bmc001_path_0_bmc;
14     wire  [1:0]       bmc010_path_0_bmc;
15     wire  [1:0]       bmc011_path_0_bmc;
16     wire  [1:0]       bmc100_path_0_bmc;
17     wire  [1:0]       bmc101_path_0_bmc;
18     wire  [1:0]       bmc110_path_0_bmc;
19     wire  [1:0]       bmc111_path_0_bmc;
20
21     wire  [1:0]       bmc000_path_1_bmc;
22     wire  [1:0]       bmc001_path_1_bmc;
23     wire  [1:0]       bmc010_path_1_bmc;
24     wire  [1:0]       bmc011_path_1_bmc;
25     wire  [1:0]       bmc100_path_1_bmc;
26     wire  [1:0]       bmc101_path_1_bmc;
27     wire  [1:0]       bmc110_path_1_bmc;
28     wire  [1:0]       bmc111_path_1_bmc;
29
30  //ACS modules signals
31     logic   [7:0]       validity;
32     logic   [7:0]       selection;
33     logic   [7:0]       path_cost   [8];
34     wire    [7:0]       validity_nets;
35     wire    [7:0]       selection_nets;
36
37     wire                ACS000_selection;
38     wire                ACS001_selection;
39     wire                ACS010_selection;
40     wire                ACS011_selection;
41     wire                ACS100_selection;
42     wire                ACS101_selection;
43     wire                ACS110_selection;
44     wire                ACS111_selection;
45
46     wire                ACS000_valid_o;
47     wire                ACS001_valid_o;
48     wire                ACS010_valid_o;
49     wire                ACS011_valid_o;
50     wire                ACS100_valid_o;
51     wire                ACS101_valid_o;
52     wire                ACS110_valid_o;
53     wire                ACS111_valid_o;
54
55     wire  [7:0]       ACS000_path_cost;
56     wire  [7:0]       ACS001_path_cost;
57     wire  [7:0]       ACS010_path_cost;
58     wire  [7:0]       ACS011_path_cost;
59     wire  [7:0]       ACS100_path_cost;
60     wire  [7:0]       ACS101_path_cost;
61     wire  [7:0]       ACS110_path_cost;
62     wire  [7:0]       ACS111_path_cost;
63
64  //Trelis memory write operation
65     logic   [1:0]       mem_bank;
66     logic   [1:0]       mem_bank_buf;
67     logic   [1:0]       mem_bank_buf_buf;
68     logic               mem_bank_buf_buf_buf;
69     logic               mem_bank_buf_buf_buf_buf;
70     logic               mem_bank_buf_buf_buf_buf_buf;
71     logic   [9:0]       wr_mem_counter;
72     logic   [9:0]       rd_mem_counter;
73
74     logic   [9:0]       addr_mem_A;
75     logic   [9:0]       addr_mem_B;
76     logic   [9:0]       addr_mem_C;
77     logic   [9:0]       addr_mem_D;
78
79     logic               wr_mem_A;
80     logic               wr_mem_B;
81     logic               wr_mem_C;
82     logic               wr_mem_D;
83
84     logic   [7:0]       d_in_mem_A;
85     logic   [7:0]       d_in_mem_B;
86     logic   [7:0]       d_in_mem_C;
87     logic   [7:0]       d_in_mem_D;
88
89     wire    [7:0]       d_o_mem_A;
```

```
    wire    [7:0]       d_o_mem_B;
    wire    [7:0]       d_o_mem_C;
    wire    [7:0]       d_o_mem_D;

//Trace back module signals
    logic               selection_tbu_0;
    logic               selection_tbu_1;

    logic   [7:0]       d_in_0_tbu_0;
    logic   [7:0]       d_in_1_tbu_0;
    logic   [7:0]       d_in_0_tbu_1;
    logic   [7:0]       d_in_1_tbu_1;

    wire                d_o_tbu_0;
    wire                d_o_tbu_1;

    logic               enable_tbu_0;
    logic               enable_tbu_1;

//Display memory operations
    wire                wr_disp_mem_0;
    wire                wr_disp_mem_1;

    wire                d_in_disp_mem_0;
    wire                d_in_disp_mem_1;

    logic   [9:0]       wr_mem_counter_disp;
    logic   [9:0]       rd_mem_counter_disp;

    logic   [9:0]       addr_disp_mem_0;
    logic   [9:0]       addr_disp_mem_1;

//Branch matrc calculation modules

    bmc000   bmc000_inst(d_in,bmc000_path_0_bmc,bmc000_path_1_bmc);
/*  similarly for bmc001 through 111
*/
    bmc001   bmc001_inst(d_in,bmc001_path_0_bmc,bmc001_path_1_bmc);
    bmc010   bmc010_inst(d_in,bmc010_path_0_bmc,bmc010_path_1_bmc);
    bmc011   bmc011_inst(d_in,bmc011_path_0_bmc,bmc011_path_1_bmc);
    bmc100   bmc100_inst(d_in,bmc100_path_0_bmc,bmc100_path_1_bmc);
    bmc101   bmc101_inst(d_in,bmc101_path_0_bmc,bmc101_path_1_bmc);
    bmc110   bmc110_inst(d_in,bmc110_path_0_bmc,bmc110_path_1_bmc);
    bmc111   bmc111_inst(d_in,bmc111_path_0_bmc,bmc111_path_1_bmc);


//Add Compare Select Modules
    ACS      ACS000(validity[0],validity[1],bmc000_path_0_bmc,bmc000_path_1_bmc,path_cost[0],path_cost[1],ACS000_selection,ACS000_valid_o,ACS000_path_cost);
    ACS      ACS001(validity[3],validity[2],bmc001_path_0_bmc,bmc001_path_1_bmc,path_cost[3],path_cost[2],ACS001_selection,ACS001_valid_o,ACS001_path_cost);
    ACS      ACS010(validity[4],validity[5],bmc010_path_0_bmc,bmc010_path_1_bmc,path_cost[4],path_cost[5],ACS010_selection,ACS010_valid_o,ACS010_path_cost);
    ACS      ACS011(validity[7],validity[6],bmc011_path_0_bmc,bmc011_path_1_bmc,path_cost[7],path_cost[6],ACS011_selection,ACS011_valid_o,ACS011_path_cost);
    ACS      ACS100(validity[1],validity[0],bmc100_path_0_bmc,bmc100_path_1_bmc,path_cost[1],path_cost[0],ACS100_selection,ACS100_valid_o,ACS100_path_cost);
    ACS      ACS101(validity[2],validity[3],bmc101_path_0_bmc,bmc101_path_1_bmc,path_cost[2],path_cost[3],ACS101_selection,ACS101_valid_o,ACS101_path_cost);
    ACS      ACS110(validity[5],validity[4],bmc110_path_0_bmc,bmc110_path_1_bmc,path_cost[5],path_cost[4],ACS110_selection,ACS110_valid_o,ACS110_path_cost);
    ACS      ACS111(validity[6],validity[7],bmc111_path_0_bmc,bmc111_path_1_bmc,path_cost[6],path_cost[7],ACS111_selection,ACS111_valid_o,ACS111_path_cost);

    assign selection_nets  =  {ACS111_selection,ACS110_selection,ACS101_selection,ACS100_selection,
                               ACS011_selection,ACS010_selection,ACS001_selection,ACS000_selection};
    assign validity_nets   =  {ACS111_valid_o,ACS110_valid_o,ACS101_valid_o,ACS100_valid_o,
                               ACS011_valid_o,ACS010_valid_o,ACS001_valid_o,ACS000_valid_o};


    always @ (posedge clk, negedge rst) begin
       if(!rst)  begin
          validity          <= 8'b00000001;
          selection         <= 8'b00000000;
/* clear all 8 path costs
          path_cost[i]      <= 8'd0;
*/
          for (int i = 0; i < 8; i++) begin
             path_cost[i] <= 8'd0;
          end
       end
       else if(!enable)   begin
          validity          <= 8'b00000001;
          selection         <= 8'b00000000;
/* clear all 8 path costs
          path_cost[i]      <= 8'd0;
*/
          for (int i = 0; i < 8; i++) begin
             path_cost[i] <= 8'd0;
          end
       end
```

```
173         else if( path_cost[0][7] && path_cost[1][7] && path_cost[2][7] && path_cost[3][7] &&
174                  path_cost[4][7] && path_cost[5][7] && path_cost[6][7] && path_cost[7][7] )
175         begin
176
177           validity          <= validity_nets;
178           selection         <= selection_nets;
179
180           path_cost[0]      <= 8'b01111111 & ACS000_path_cost;
181   /*  likewise for path_cost[1:7] and ACS001:111_path_cost
182   */
183           path_cost[1]      <= 8'b01111111 & ACS001_path_cost;
184           path_cost[2]      <= 8'b01111111 & ACS010_path_cost;
185           path_cost[3]      <= 8'b01111111 & ACS011_path_cost;
186           path_cost[4]      <= 8'b01111111 & ACS100_path_cost;
187           path_cost[5]      <= 8'b01111111 & ACS101_path_cost;
188           path_cost[6]      <= 8'b01111111 & ACS110_path_cost;
189           path_cost[7]      <= 8'b01111111 & ACS111_path_cost;
190         end
191         else   begin
192           validity          <= validity_nets;
193           selection         <= selection_nets;
194
195           path_cost[0]      <= ACS000_path_cost;
196   /* likewise for 1:7
197   */
198           path_cost[1]      <= ACS001_path_cost;
199           path_cost[2]      <= ACS010_path_cost;
200           path_cost[3]      <= ACS011_path_cost;
201           path_cost[4]      <= ACS100_path_cost;
202           path_cost[5]      <= ACS101_path_cost;
203           path_cost[6]      <= ACS110_path_cost;
204           path_cost[7]      <= ACS111_path_cost;
205         end
206       end
207
208       always @ (posedge clk, negedge rst) begin
209         if(!rst)
210           wr_mem_counter <= 10'd0;
211         else if(!enable)
212           wr_mem_counter <= 10'd0;
213         else
214           wr_mem_counter <= wr_mem_counter + 10'd1;
215       end
```

```
216
217     always @ (posedge clk, negedge rst) begin
218         if(!rst)
219             rd_mem_counter <= 10'b1111111111;   // -1   how do you handle this in 10 bit binary?
220         else if(enable)
221             rd_mem_counter <= rd_mem_counter - 10'd1;
222     end
223
224     always @ (posedge clk, negedge rst)
225         if(!rst)
226             mem_bank <= 2'b00;
227         else begin
228             if(wr_mem_counter==10'b1111111111)
229                 mem_bank <= mem_bank + 2'b01;
230         end
231
232     always @ (posedge clk)    begin
233         d_in_mem_A  <= selection;
234         d_in_mem_B  <= selection;
235         d_in_mem_C  <= selection;
236         d_in_mem_D  <= selection;
237     end
238
239     always @ (posedge clk)     begin
240         case(mem_bank)
241             2'b00:          begin
242                 addr_mem_A          <= wr_mem_counter;
243                 addr_mem_B          <= rd_mem_counter;
244                 addr_mem_C          <= 10'd0;
245                 addr_mem_D          <= rd_mem_counter;
246
247                 wr_mem_A            <= 1'b1;
248     /* other wr_mems = 0
249     */
250                 wr_mem_B            <= 1'b0;
251                 wr_mem_C            <= 1'b0;
252                 wr_mem_D            <= 1'b0;
253             end
254             2'b01:          begin
255                 addr_mem_A          <= rd_mem_counter;
256                 addr_mem_B          <= wr_mem_counter;
257                 addr_mem_C          <= rd_mem_counter;
258                 addr_mem_D          <= 10'd0;
259
```

```
260             wr_mem_B          <= 1'b1;
261     /* other wr_mems = 0
262     */
263             wr_mem_A          <= 1'b0;
264             wr_mem_C          <= 1'b0;
265             wr_mem_D          <= 1'b0;
266           end
267           2'b10:    begin
268             addr_mem_A        <= 10'd0;
269             addr_mem_B        <= rd_mem_counter;
270             addr_mem_C        <= wr_mem_counter;
271             addr_mem_D        <= rd_mem_counter;
272
273             wr_mem_C       <= 1'b1;
274     /* other wr_mems = 0
275     */
276             wr_mem_A          <= 1'b0;
277             wr_mem_B          <= 1'b0;
278             wr_mem_D          <= 1'b0;
279           end
280           2'b11:     begin
281             addr_mem_A        <= rd_mem_counter;
282             addr_mem_B        <= 10'd0;
283             addr_mem_C        <= rd_mem_counter;
284             addr_mem_D        <= wr_mem_counter;
285
286             wr_mem_D       <= 1'b1;
287     /* other wr_mems = 0
288     */
289             wr_mem_A          <= 1'b0;
290             wr_mem_B          <= 1'b0;
291             wr_mem_C          <= 1'b0;
292           end
293         endcase
294       end
295
296     //Trelis memory module instantiation
297
298       mem   trelis_mem_A
299       (
300         .clk,
301         .wr(wr_mem_A),
302         .addr(addr_mem_A),
303         .d_i(d_in_mem_A),
```

```
304         .d_o(d_o_mem_A)
305       );
306     /* likewise for trelis_memB, C, D
307     */
308       mem   trelis_mem_B
309       (
310         .clk,
311         .wr(wr_mem_B),
312         .addr(addr_mem_B),
313         .d_i(d_in_mem_B),
314         .d_o(d_o_mem_B)
315       );
316
317       mem   trelis_mem_C
318       (
319         .clk,
320         .wr(wr_mem_C),
321         .addr(addr_mem_C),
322         .d_i(d_in_mem_C),
323         .d_o(d_o_mem_C)
324       );
325
326       mem   trelis_mem_D
327       (
328         .clk,
329         .wr(wr_mem_D),
330         .addr(addr_mem_D),
331         .d_i(d_in_mem_D),
332         .d_o(d_o_mem_D)
333       );
334     //Trace back module operation
335
336       always @(posedge clk)
337         mem_bank_buf    <= mem_bank;
338
339       always @(posedge clk)
340         mem_bank_buf_buf    <= mem_bank_buf;
341
342       always @ (posedge clk, negedge rst)
343         if(!rst)
344               enable_tbu_0    <= 1'b0;
345         else begin
346           if(mem_bank_buf_buf==2'b10)
347               enable_tbu_0    <= 1'b1;
```

```
348         else
349             enable_tbu_0   <= enable_tbu_0;
350       end
351
352     always @ (posedge clk, negedge rst)
353       if(!rst)
354             enable_tbu_1   <= 1'b0;
355       else begin
356         if(mem_bank_buf_buf==2'b11)
357             enable_tbu_1   <= 1'b1;
358         else
359             enable_tbu_1   <= enable_tbu_1;
360       end
361
362     always @ (posedge clk)
363       case(mem_bank_buf_buf)
364         2'b00:     begin
365             d_in_0_tbu_0   <= d_o_mem_D;
366             d_in_1_tbu_0   <= d_o_mem_C;
367
368             d_in_0_tbu_1   <= d_o_mem_C;
369             d_in_1_tbu_1   <= d_o_mem_B;
370
371             selection_tbu_0<= 1'b0;
372             selection_tbu_1<= 1'b1;
373
374         end
375         2'b01:      begin
376             d_in_0_tbu_0   <= d_o_mem_D;
377             d_in_1_tbu_0   <= d_o_mem_C;
378
379             d_in_0_tbu_1   <= d_o_mem_A;
380             d_in_1_tbu_1   <= d_o_mem_D;
381
382             selection_tbu_0<= 1'b1;
383             selection_tbu_1<= 1'b0;
384         end
385         2'b10:      begin
386             d_in_0_tbu_0   <= d_o_mem_B;
387             d_in_1_tbu_0   <= d_o_mem_A;
388
389             d_in_0_tbu_1   <= d_o_mem_A;
390             d_in_1_tbu_1   <= d_o_mem_D;
391
```

```
392             selection_tbu_0<= 1'b0;
393             selection_tbu_1<= 1'b1;
394           end
395           2'b11:     begin
396             d_in_0_tbu_0   <= d_o_mem_B;
397             d_in_1_tbu_0   <= d_o_mem_A;
398
399             d_in_0_tbu_1   <= d_o_mem_C;
400             d_in_1_tbu_1   <= d_o_mem_B;
401
402             selection_tbu_0<= 1'b1;
403             selection_tbu_1<= 1'b0;
404           end
405       endcase
406
407   //Trace-Back modules instantiation
408
409     tbu tbu_0   (
410       .clk,
411       .rst,
412       .enable(enable_tbu_0),
413       .selection(selection_tbu_0),
414       .d_in_0(d_in_0_tbu_0),
415       .d_in_1(d_in_1_tbu_0),
416       .d_o(d_o_tbu_0),
417       .wr_en(wr_disp_mem_0)
418     );
419
420   /* analogous for tbu_1
421   */
422     tbu tbu_1   (
423       .clk,
424       .rst,
425       .enable(enable_tbu_1),
426       .selection(selection_tbu_1),
427       .d_in_0(d_in_0_tbu_1),
428       .d_in_1(d_in_1_tbu_1),
429       .d_o(d_o_tbu_1),
430       .wr_en(wr_disp_mem_1)
431     );
432
433   //Display Memory modules Instantioation
434
435     assign   d_in_disp_mem_0   =  d_o_tbu_0;
```

```
436     assign   d_in_disp_mem_1   =  d_o_tbu_1;
437
438    mem_disp   disp_mem_0
439    (
440        .clk             ,
441        .wr(wr_disp_mem_0),
442        .addr(addr_disp_mem_0),
443        .d_i(d_in_disp_mem_0),
444        .d_o(d_o_disp_mem_0)
445     );
446   /* analogous for disp_mem_1
447   */
448    mem_disp   disp_mem_1
449    (
450        .clk             ,
451        .wr(wr_disp_mem_1),
452        .addr(addr_disp_mem_1),
453        .d_i(d_in_disp_mem_1),
454        .d_o(d_o_disp_mem_1)
455     );
456
457   // Display memory module operation
458     always @ (posedge clk)
459        mem_bank_buf_buf_buf <= mem_bank_buf_buf[0];
460
461     always @ (posedge clk)
462        if(!rst)
463           wr_mem_counter_disp  <= 10'b0000000010;
464        else if(!enable)
465           wr_mem_counter_disp  <= 10'b0000000010;
466        else
467           wr_mem_counter_disp  <= wr_mem_counter_disp - 10'd1;
468
469     always @ (posedge clk)
470        if(!rst)
471           rd_mem_counter_disp  <= 10'b1111111101;
472        else if(!enable)
473           rd_mem_counter_disp  <= 10'b1111111101;
474        else
475           rd_mem_counter_disp  <= rd_mem_counter_disp + 10'd1;
476
477     always @ (posedge clk)
478        case(mem_bank_buf_buf_buf)
479           1'b0:
```

```
480           begin
481              addr_disp_mem_0   <= rd_mem_counter_disp;
482              addr_disp_mem_1   <= wr_mem_counter_disp;
483           end
484           1'b1:  //swap rd and wr
485           begin
486              addr_disp_mem_0   <= wr_mem_counter_disp;
487              addr_disp_mem_1   <= rd_mem_counter_disp;
488           end
489        endcase
490
491     always @ (posedge clk) begin
492        mem_bank_buf_buf_buf_buf     <= mem_bank_buf_buf_buf;
493        mem_bank_buf_buf_buf_buf_buf <= mem_bank_buf_buf_buf_buf;
494     end
495
496     always @ (posedge clk)
497   /*  d_out = d_o_disp_mem_i
498     i = mem_bank_buf_buf_buf_buf_buf
499   */
500        // set d_out to d_o_disp_mem1 if mem_bank_buf_buf_buf_buf_buf is logic High and d_o_disp_mem0 otherwise
501        if (mem_bank_buf_buf_buf_buf_buf)
502           d_out <= d_o_disp_mem_1;
503        else
504           d_out <= d_o_disp_mem_0;
505   endmodule
506
```

(2) Bottom line of the Questa/ModelSim transcript (Transcript show our score / 256 trials):

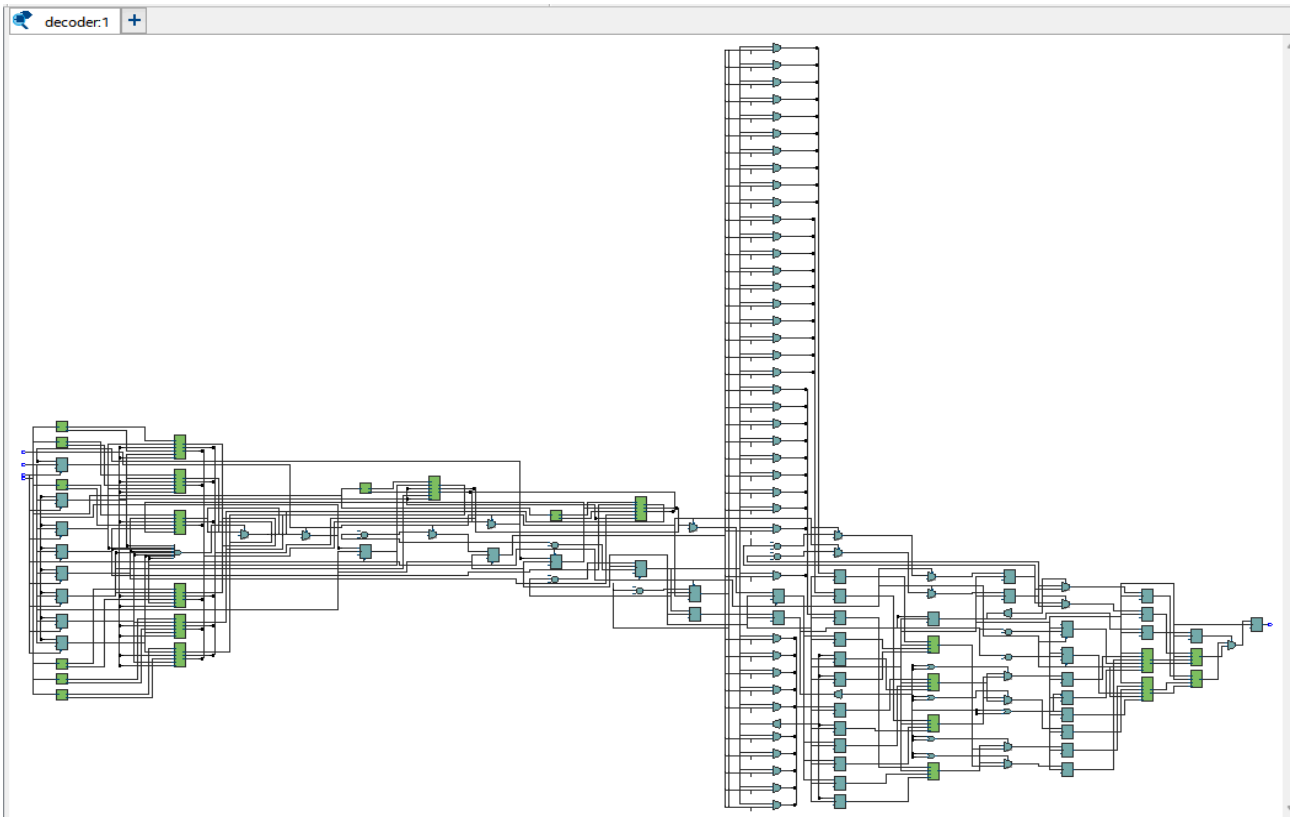
```
# yaa! in = 0, out = 0, w_ct =   0, err = 00
# yaa! in = 0, out = 0, w_ct =   1, err = 00
# yaa! in = 0, out = 0, w_ct =   2, err = 00
# yaa! in = 0, out = 0, w_ct =   3, err = 00
# yaa! in = 0, out = 0, w_ct =   4, err = 00
# yaa! in = 0, out = 0, w_ct =   5, err = 00
# yaa! in = 0, out = 0, w_ct =   6, err = 00
# yaa! in = 0, out = 0, w_ct =   7, err = 00
# yaa! in = 0, out = 0, w_ct =   8, err = 00
# yaa! in = 0, out = 0, w_ct =   9, err = 00
# yaa! in = 0, out = 0, w_ct =  10, err = 00
# yaa! in = 1, out = 1, w_ct =  11, err = 00
# yaa! in = 0, out = 0, w_ct =  12, err = 00
# yaa! in = 0, out = 0, w_ct =  13, err = 00
# yaa! in = 1, out = 1, w_ct =  14, err = 00
# yaa! in = 1, out = 1, w_ct =  15, err = 00
# yaa! in = 0, out = 0, w_ct =  16, err = 00
# yaa! in = 0, out = 0, w_ct =  17, err = 00
# yaa! in = 0, out = 0, w_ct =  18, err = 00
# yaa! in = 1, out = 1, w_ct =  19, err = 00
# yaa! in = 1, out = 1, w_ct =  20, err = 00
# yaa! in = 1, out = 1, w_ct =  21, err = 00
# yaa! in = 0, out = 0, w_ct =  22, err = 00
# yaa! in = 0, out = 0, w_ct =  23, err = 00
# yaa! in = 0, out = 0, w_ct =  24, err = 00
# yaa! in = 0, out = 0, w_ct =  25, err = 00
# yaa! in = 1, out = 1, w_ct =  26, err = 00
# yaa! in = 1, out = 1, w_ct =  27, err = 00
# yaa! in = 1, out = 1, w_ct =  28, err = 00
# yaa! in = 1, out = 1, w_ct =  29, err = 00
# yaa! in = 0, out = 0, w_ct =  30, err = 00
# yaa! in = 0, out = 0, w_ct =  31, err = 00
# yaa! in = 0, out = 0, w_ct =  32, err = 00
# yaa! in = 0, out = 0, w_ct =  33, err = 00
# yaa! in = 0, out = 0, w_ct =  34, err = 00
# yaa! in = 1, out = 1, w_ct =  35, err = 00
# yaa! in = 1, out = 1, w_ct =  36, err = 00
# yaa! in = 1, out = 1, w_ct =  37, err = 00
# yaa! in = 1, out = 1, w_ct =  38, err = 00
# yaa! in = 1, out = 1, w_ct =  39, err = 00
# yaa! in = 0, out = 0, w_ct =  40, err = 00
# yaa! in = 1, out = 1, w_ct =  41, err = 00
# yaa! in = 0, out = 0, w_ct =  42, err = 00
# yaa! in = 1, out = 1, w_ct =  43, err = 00
# yaa! in = 0, out = 0, w_ct =  44, err = 00
# yaa! in = 1, out = 1, w_ct =  45, err = 00
# yaa! in = 0, out = 0, w_ct =  46, err = 00
# yaa! in = 1, out = 1, w_ct =  47, err = 00
# yaa! in = 0, out = 0, w_ct =  48, err = 00
# yaa! in = 1, out = 1, w_ct =  49, err = 00
# yaa! in = 0, out = 0, w_ct =  50, err = 00
# yaa! in = 0, out = 0, w_ct =  51, err = 00
# yaa! in = 1, out = 1, w_ct =  52, err = 00
# yaa! in = 1, out = 1, w_ct =  53, err = 00
# yaa! in = 0, out = 0, w_ct =  54, err = 00
# yaa! in = 0, out = 0, w_ct =  55, err = 00
# yaa! in = 0, out = 0, w_ct =  56, err = 00
# yaa! in = 1, out = 1, w_ct =  57, err = 00
# yaa! in = 1, out = 1, w_ct =  58, err = 00
# yaa! in = 1, out = 1, w_ct =  59, err = 00
# yaa! in = 0, out = 0, w_ct =  60, err = 00
# yaa! in = 0, out = 0, w_ct =  61, err = 00
# yaa! in = 0, out = 0, w_ct =  62, err = 00
# yaa! in = 0, out = 0, w_ct =  63, err = 00
# yaa! in = 1, out = 1, w_ct =  64, err = 00
# yaa! in = 1, out = 1, w_ct =  65, err = 00
# yaa! in = 1, out = 1, w_ct =  66, err = 00
# yaa! in = 1, out = 1, w_ct =  67, err = 00
# yaa! in = 0, out = 0, w_ct =  68, err = 00
# yaa! in = 0, out = 0, w_ct =  69, err = 00
# yaa! in = 0, out = 0, w_ct =  70, err = 00
# yaa! in = 0, out = 0, w_ct =  71, err = 00
# yaa! in = 0, out = 0, w_ct =  72, err = 00
# yaa! in = 1, out = 1, w_ct =  73, err = 00
# yaa! in = 1, out = 1, w_ct =  74, err = 00
# yaa! in = 1, out = 1, w_ct =  75, err = 00
# yaa! in = 1, out = 1, w_ct =  76, err = 00
# yaa! in = 1, out = 1, w_ct =  77, err = 00
# yaa! in = 0, out = 0, w_ct =  78, err = 00
# yaa! in = 1, out = 1, w_ct =  79, err = 00
# yaa! in = 0, out = 0, w_ct =  80, err = 00
# yaa! in = 1, out = 1, w_ct =  81, err = 00
# yaa! in = 0, out = 0, w_ct =  82, err = 00
# yaa! in = 1, out = 1, w_ct =  83, err = 00
# yaa! in = 0, out = 0, w_ct =  84, err = 00
# yaa! in = 1, out = 1, w_ct =  85, err = 00
# yaa! in = 0, out = 0, w_ct =  86, err = 00
# yaa! in = 0, out = 0, w_ct =  87, err = 00
# yaa! in = 0, out = 0, w_ct =  88, err = 00
# yaa! in = 0, out = 0, w_ct =  89, err = 00
# yaa! in = 0, out = 0, w_ct =  90, err = 00
# yaa! in = 0, out = 0, w_ct =  91, err = 00
# yaa! in = 0, out = 0, w_ct =  92, err = 00
# yaa! in = 0, out = 0, w_ct =  93, err = 00
# yaa! in = 0, out = 0, w_ct =  94, err = 00
# yaa! in = 0, out = 0, w_ct =  95, err = 00
# yaa! in = 1, out = 1, w_ct =  96, err = 00
# yaa! in = 1, out = 1, w_ct =  97, err = 00
# yaa! in = 1, out = 1, w_ct =  98, err = 00
# yaa! in = 1, out = 1, w_ct =  99, err = 00
# yaa! in = 1, out = 1, w_ct = 100, err = 00
# yaa! in = 1, out = 1, w_ct = 101, err = 00
# yaa! in = 1, out = 1, w_ct = 102, err = 00
# yaa! in = 1, out = 1, w_ct = 103, err = 00
# yaa! in = 1, out = 1, w_ct = 104, err = 00
# yaa! in = 1, out = 1, w_ct = 105, err = 00
# yaa! in = 0, out = 0, w_ct = 106, err = 00
# yaa! in = 1, out = 1, w_ct = 107, err = 00
# yaa! in = 0, out = 0, w_ct = 108, err = 00
# yaa! in = 1, out = 1, w_ct = 109, err = 00
# yaa! in = 1, out = 1, w_ct = 110, err = 00
# yaa! in = 1, out = 1, w_ct = 111, err = 00
# yaa! in = 1, out = 1, w_ct = 112, err = 00
# yaa! in = 0, out = 0, w_ct = 113, err = 00
# yaa! in = 1, out = 1, w_ct = 114, err = 00
# yaa! in = 1, out = 1, w_ct = 115, err = 00
# yaa! in = 1, out = 1, w_ct = 116, err = 00
# yaa! in = 0, out = 0, w_ct = 117, err = 00
# yaa! in = 1, out = 1, w_ct = 118, err = 00
# yaa! in = 1, out = 1, w_ct = 119, err = 00
# yaa! in = 1, out = 1, w_ct = 120, err = 00
# yaa! in = 0, out = 0, w_ct = 121, err = 00
# yaa! in = 1, out = 1, w_ct = 122, err = 00
# yaa! in = 0, out = 0, w_ct = 123, err = 00
# yaa! in = 1, out = 1, w_ct = 124, err = 00
# yaa! in = 1, out = 1, w_ct = 125, err = 00
# yaa! in = 0, out = 0, w_ct = 126, err = 00
# yaa! in = 0, out = 0, w_ct = 127, err = 00
# yaa! in = 0, out = 0, w_ct = 128, err = 00
# yaa! in = 0, out = 0, w_ct = 129, err = 00
# yaa! in = 0, out = 0, w_ct = 130, err = 00
# yaa! in = 0, out = 0, w_ct = 131, err = 00
# yaa! in = 0, out = 0, w_ct = 132, err = 00
# yaa! in = 0, out = 0, w_ct = 133, err = 00
# yaa! in = 0, out = 0, w_ct = 134, err = 00
# yaa! in = 0, out = 0, w_ct = 135, err = 00
# yaa! in = 0, out = 0, w_ct = 136, err = 00
# yaa! in = 1, out = 1, w_ct = 137, err = 00
# yaa! in = 1, out = 1, w_ct = 138, err = 00
# yaa! in = 1, out = 1, w_ct = 139, err = 00
# yaa! in = 1, out = 1, w_ct = 140, err = 00
# yaa! in = 1, out = 1, w_ct = 141, err = 00
# yaa! in = 1, out = 1, w_ct = 142, err = 00
# yaa! in = 1, out = 1, w_ct = 143, err = 00
# yaa! in = 1, out = 1, w_ct = 144, err = 00
# yaa! in = 1, out = 1, w_ct = 145, err = 00
# yaa! in = 1, out = 1, w_ct = 146, err = 00
# yaa! in = 0, out = 0, w_ct = 147, err = 00
# yaa! in = 1, out = 1, w_ct = 148, err = 00
# yaa! in = 1, out = 1, w_ct = 149, err = 00
# yaa! in = 1, out = 1, w_ct = 150, err = 00
# yaa! in = 1, out = 1, w_ct = 151, err = 00
# yaa! in = 1, out = 1, w_ct = 152, err = 00
# yaa! in = 1, out = 1, w_ct = 153, err = 00
# yaa! in = 1, out = 1, w_ct = 154, err = 00
# yaa! in = 1, out = 1, w_ct = 155, err = 00
# yaa! in = 1, out = 1, w_ct = 156, err = 00
# yaa! in = 1, out = 1, w_ct = 157, err = 00
# yaa! in = 1, out = 1, w_ct = 158, err = 00
# yaa! in = 1, out = 1, w_ct = 159, err = 00
# yaa! in = 1, out = 1, w_ct = 160, err = 00
# yaa! in = 1, out = 1, w_ct = 161, err = 00
# yaa! in = 1, out = 1, w_ct = 162, err = 00
# yaa! in = 1, out = 1, w_ct = 163, err = 00
# yaa! in = 1, out = 1, w_ct = 164, err = 00
# yaa! in = 1, out = 1, w_ct = 165, err = 00
# yaa! in = 1, out = 1, w_ct = 166, err = 00
# yaa! in = 1, out = 1, w_ct = 167, err = 00
# yaa! in = 1, out = 1, w_ct = 168, err = 00
# yaa! in = 1, out = 1, w_ct = 169, err = 00
# yaa! in = 1, out = 1, w_ct = 170, err = 00
# yaa! in = 1, out = 1, w_ct = 171, err = 00
# yaa! in = 1, out = 1, w_ct = 172, err = 00
# yaa! in = 1, out = 1, w_ct = 173, err = 00
# yaa! in = 1, out = 1, w_ct = 174, err = 00
# yaa! in = 1, out = 1, w_ct = 175, err = 00
# yaa! in = 1, out = 1, w_ct = 176, err = 00
# yaa! in = 1, out = 1, w_ct = 177, err = 00
# yaa! in = 1, out = 1, w_ct = 178, err = 00
# yaa! in = 1, out = 1, w_ct = 179, err = 00
# yaa! in = 1, out = 1, w_ct = 180, err = 00
# yaa! in = 1, out = 1, w_ct = 181, err = 00
# yaa! in = 1, out = 1, w_ct = 182, err = 00
# yaa! in = 1, out = 1, w_ct = 183, err = 00
# yaa! in = 1, out = 1, w_ct = 184, err = 00
# yaa! in = 1, out = 1, w_ct = 185, err = 00
# yaa! in = 1, out = 1, w_ct = 186, err = 00
# yaa! in = 1, out = 1, w_ct = 187, err = 00
# yaa! in = 1, out = 1, w_ct = 188, err = 00
# yaa! in = 1, out = 1, w_ct = 189, err = 00
# yaa! in = 1, out = 1, w_ct = 190, err = 00
# yaa! in = 1, out = 1, w_ct = 191, err = 00
# yaa! in = 1, out = 1, w_ct = 192, err = 00
# yaa! in = 1, out = 1, w_ct = 193, err = 00
# yaa! in = 1, out = 1, w_ct = 194, err = 00
# yaa! in = 1, out = 1, w_ct = 195, err = 00
# yaa! in = 1, out = 1, w_ct = 196, err = 00
# yaa! in = 1, out = 1, w_ct = 197, err = 00
# yaa! in = 1, out = 1, w_ct = 198, err = 00
# yaa! in = 1, out = 1, w_ct = 199, err = 00
# yaa! in = 1, out = 1, w_ct = 200, err = 00
# yaa! in = 1, out = 1, w_ct = 201, err = 00
# yaa! in = 1, out = 1, w_ct = 202, err = 00
# yaa! in = 1, out = 1, w_ct = 203, err = 00
# yaa! in = 1, out = 1, w_ct = 204, err = 00
# yaa! in = 1, out = 1, w_ct = 205, err = 00
# yaa! in = 1, out = 1, w_ct = 206, err = 00
# yaa! in = 1, out = 1, w_ct = 207, err = 00
```

```
# yaa! in = 1, out = 1, w_ct =         208, err = 00
# yaa! in = 1, out = 1, w_ct =         209, err = 00
# yaa! in = 1, out = 1, w_ct =         210, err = 00
# yaa! in = 1, out = 1, w_ct =         211, err = 00
# yaa! in = 1, out = 1, w_ct =         212, err = 00
# yaa! in = 1, out = 1, w_ct =         213, err = 00
# yaa! in = 1, out = 1, w_ct =         214, err = 00
# yaa! in = 1, out = 1, w_ct =         215, err = 00
# yaa! in = 1, out = 1, w_ct =         216, err = 00
# yaa! in = 1, out = 1, w_ct =         217, err = 00
# yaa! in = 1, out = 1, w_ct =         218, err = 00
# yaa! in = 1, out = 1, w_ct =         219, err = 00
# yaa! in = 1, out = 1, w_ct =         220, err = 00
# yaa! in = 1, out = 1, w_ct =         221, err = 00
# yaa! in = 1, out = 1, w_ct =         222, err = 00
# yaa! in = 1, out = 1, w_ct =         223, err = 00
# yaa! in = 1, out = 1, w_ct =         224, err = 00
# yaa! in = 1, out = 1, w_ct =         225, err = 00
# yaa! in = 1, out = 1, w_ct =         226, err = 00
# yaa! in = 1, out = 1, w_ct =         227, err = 00
# yaa! in = 1, out = 1, w_ct =         228, err = 00
# yaa! in = 1, out = 1, w_ct =         229, err = 00
# yaa! in = 1, out = 1, w_ct =         230, err = 00
# yaa! in = 1, out = 1, w_ct =         231, err = 00
# yaa! in = 1, out = 1, w_ct =         232, err = 00
# yaa! in = 1, out = 1, w_ct =         233, err = 00
# yaa! in = 1, out = 1, w_ct =         234, err = 00
# yaa! in = 1, out = 1, w_ct =         235, err = 00
# yaa! in = 1, out = 1, w_ct =         236, err = 00
# yaa! in = 1, out = 1, w_ct =         237, err = 00
# yaa! in = 1, out = 1, w_ct =         238, err = 00
# yaa! in = 1, out = 1, w_ct =         239, err = 00
# yaa! in = 1, out = 1, w_ct =         240, err = 00
# yaa! in = 1, out = 1, w_ct =         241, err = 00
# yaa! in = 1, out = 1, w_ct =         242, err = 00
# yaa! in = 1, out = 1, w_ct =         243, err = 00
# yaa! in = 1, out = 1, w_ct =         244, err = 00
# yaa! in = 1, out = 1, w_ct =         245, err = 00
# yaa! in = 1, out = 1, w_ct =         246, err = 00
# yaa! in = 1, out = 1, w_ct =         247, err = 00
# yaa! in = 0, out = 0, w_ct =         248, err = 00
# yaa! in = 1, out = 1, w_ct =         249, err = 00
# yaa! in = 1, out = 1, w_ct =         250, err = 00
# yaa! in = 1, out = 1, w_ct =         251, err = 00
# yaa! in = 1, out = 1, w_ct =         252, err = 00
# yaa! in = 1, out = 1, w_ct =         253, err = 00
# yaa! in = 1, out = 1, w_ct =         254, err = 00
# yaa! in = 1, out = 1, w_ct =         255, err = 00
# good =          256, bad =          0,             0 bad_bits
# ** Note: $stop    : E:/damei/term_project_v2/viterbi_tx_rx_tb.sv(89)
#    Time: 1045 ns  Iteration: 0  Instance: /viterbi_tx_rx_tb
# Break in Module viterbi_tx_rx_tb at E:/damei/term_project_v2/viterbi_tx_rx_tb.sv line 89
```

As shown above in the Modelsim transcript snapshot, the message prints that “good = _256_, bad = _0_, _0_ bad bits” as expected.

(3) Proof of successful synthesis of decoder.sv (including, of course, its subfiles) -- either Quartus RTL viewer diagram or Mentor Precision netlist text, utilization report, or just a shot of transcript/log showing proof of synthesis.

- *Schematic - Quartus RTL Simulation Netlist Viewer (synthesis netlist):*



- *Schematic - Analysis & Synthesis Resource Usage Summary AND resource utilization report and provide total flipflops and combination Aluts ["Flow Summary" shows DFF count as "total registers"]: Total flipflops = 239 & Combinational ALUTs = 482:*

Analysis & Synthesis Resource Usage Summary

<<Filter>>

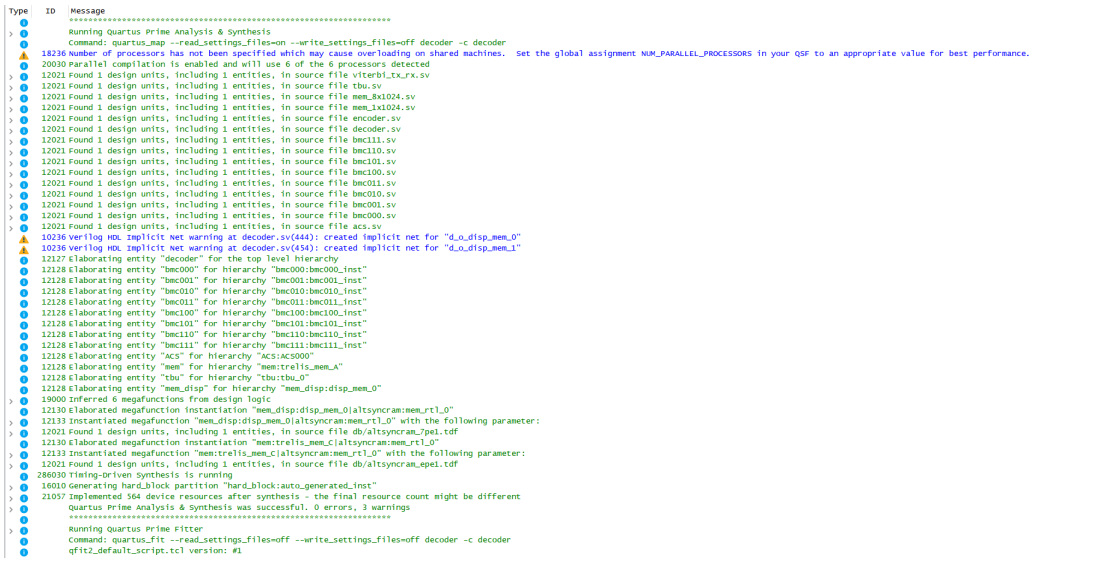
	Resource	Usage
1	Estimated ALUTs Used	356
1	-- Combinational ALUTs	356
2	-- Memory ALUTs	0
3	-- LUT_REGs	0
2	Dedicated logic registers	239
3		
4	Estimated ALUTs Unavailable	52
1	-- Due to unpartnered combinational logic	52
2	-- Due to Memory ALUTs	0
5		
6	Total combinational functions	356
7	Combinational ALUT usage by number of inputs	
1	-- 7 input functions	8
2	-- 6 input functions	54
3	-- 5 input functions	16
4	-- 4 input functions	23
5	-- <=3 input functions	255
8		
9	Combinational ALUTs by mode	
1	-- normal mode	180
2	-- extended LUT mode	8
3	-- arithmetic mode	168
4	-- shared arithmetic mode	0
10		
11	Estimated ALUT/register pairs used	482
12		
13	Total registers	239
1	-- Dedicated logic registers	239
2	-- I/O registers	0
3	-- LUT_REGs	0
14		
15		
16	I/O pins	6
17	Total MLAB memory bits	0
18	Total block memory bits	34816
19		
20	DSP block 18-bit elements	0
21		
22	Maximum fan-out node	clk~input
23	Maximum fan-out	273
24	Total fan-out	2539
25	Average fan-out	3.96

Flow Summary

<<Filter>>

Flow Status	Successful - Fri Mar 22 11:30:23 2024
Quartus Prime Version	20.1.1 Build 720 11/11/2020 SJ Lite Edition
Revision Name	decoder
Top-level Entity Name	decoder
Family	Arria II GX
Logic utilization	1 %
Total registers	239
Total pins	6 / 176 (3 %)
Total virtual pins	0
Total block memory bits	34,816 / 2,939,904 (1 %)
DSP block 18-bit elements	0 / 232 (0 %)
Total GXB Receiver Channel PCS	0 / 4 (0 %)
Total GXB Receiver Channel PMA	0 / 4 (0 %)
Total GXB Transmitter Channel PCS	0 / 4 (0 %)
Total GXB Transmitter Channel PMA	0 / 4 (0 %)
Total PLLs	0 / 4 (0 %)
Total DLLs	0 / 2 (0 %)
Device	EP2AGX45CU17I3
Timing Models	Final

- Schematics - A shot of transcript/log showing proof of synthesis:

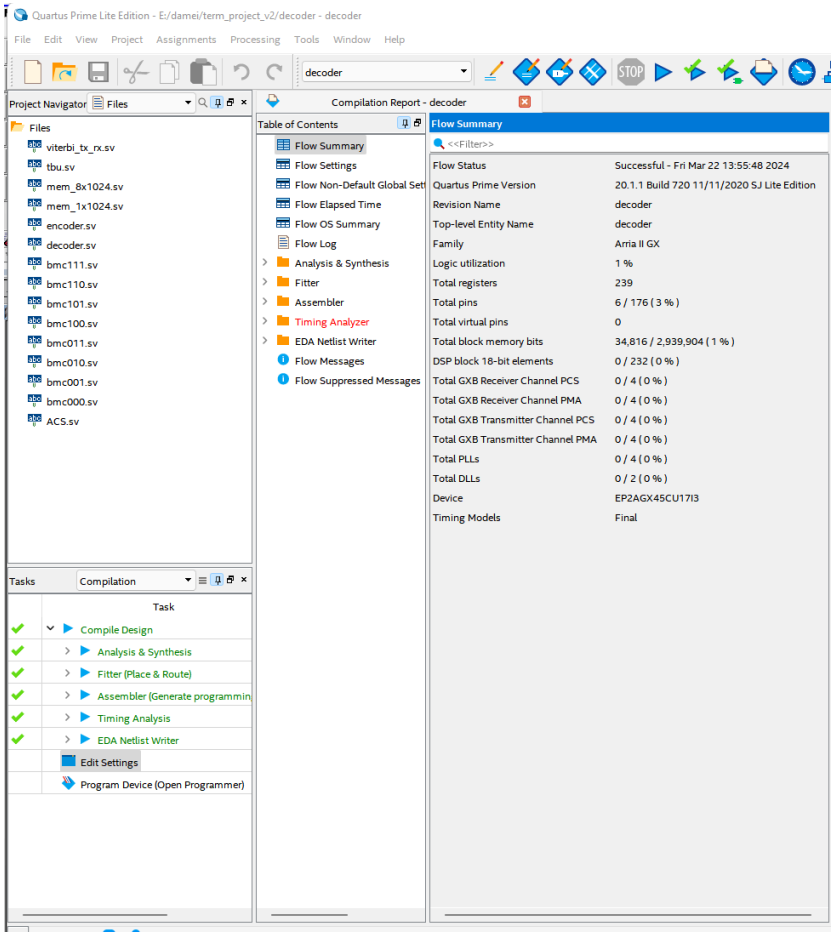


```
Type  ID    Message
      ********************************************************************
      Running Quartus Prime Analysis & Synthesis
      Command: quartus_map --read_settings_files=on --write_settings_files=off decoder -c decoder
 18236 Number of processors has not been specified which may cause overloading on shared machines.  Set the global assignment NUM_PARALLEL_PROCESSORS in your QSF to an appropriate value for best performance.
 20030 Parallel compilation is enabled and will use 6 of the 6 processors detected
 12021 Found 1 design units, including 1 entities, in source file viterbi_tx_rx.sv
 12021 Found 1 design units, including 1 entities, in source file tbu.sv
 12021 Found 1 design units, including 1 entities, in source file mem_8x1024.sv
 12021 Found 1 design units, including 1 entities, in source file mem_1x1024.sv
 12021 Found 1 design units, including 1 entities, in source file encoder.sv
 12021 Found 1 design units, including 1 entities, in source file decoder.sv
 12021 Found 1 design units, including 1 entities, in source file bmc111.sv
 12021 Found 1 design units, including 1 entities, in source file bmc110.sv
 12021 Found 1 design units, including 1 entities, in source file bmc101.sv
 12021 Found 1 design units, including 1 entities, in source file bmc100.sv
 12021 Found 1 design units, including 1 entities, in source file bmc011.sv
 12021 Found 1 design units, including 1 entities, in source file bmc010.sv
 12021 Found 1 design units, including 1 entities, in source file bmc001.sv
 12021 Found 1 design units, including 1 entities, in source file bmc000.sv
 12021 Found 1 design units, including 1 entities, in source file acs.sv
 10236 Verilog HDL Implicit Net warning at decoder.sv(444): created implicit net for "d_o_disp_mem_0"
 10236 Verilog HDL Implicit Net warning at decoder.sv(454): created implicit net for "d_o_disp_mem_1"
 12127 Elaborating entity "decoder" for the top level hierarchy
 12128 Elaborating entity "bmc000" for hierarchy "bmc000:bmc000_inst"
 12128 Elaborating entity "bmc001" for hierarchy "bmc001:bmc001_inst"
 12128 Elaborating entity "bmc010" for hierarchy "bmc010:bmc010_inst"
 12128 Elaborating entity "bmc011" for hierarchy "bmc011:bmc011_inst"
 12128 Elaborating entity "bmc100" for hierarchy "bmc100:bmc100_inst"
 12128 Elaborating entity "bmc101" for hierarchy "bmc101:bmc101_inst"
 12128 Elaborating entity "bmc110" for hierarchy "bmc110:bmc110_inst"
 12128 Elaborating entity "bmc111" for hierarchy "bmc111:bmc111_inst"
 12128 Elaborating entity "ACS" for hierarchy "ACS:ACS000"
 12128 Elaborating entity "mem" for hierarchy "mem:trelis_mem_A"
 12128 Elaborating entity "tbu" for hierarchy "tbu:tbu_0"
 12128 Elaborating entity "mem_disp" for hierarchy "mem_disp:disp_mem_0"
 19000 Inferred 6 megafunctions from design logic
 12130 Elaborated megafunction instantiation "mem_disp:disp_mem_0|altsyncram:mem_rtl_0"
 12133 Instantiated megafunction "mem_disp:disp_mem_0|altsyncram:mem_rtl_0" with the following parameter:
 12021 Found 1 design units, including 1 entities, in source file db/altsyncram_7pe1.tdf
 12130 Elaborated megafunction instantiation "mem:trelis_mem_C|altsyncram:mem_rtl_0"
 12133 Instantiated megafunction "mem:trelis_mem_C|altsyncram:mem_rtl_0" with the following parameter:
 12021 Found 1 design units, including 1 entities, in source file db/altsyncram_epe1.tdf
286030 Timing-Driven Synthesis is running
 16010 Generating hard_block partition "hard_block:auto_generated_inst"
 21057 Implemented 564 device resources after synthesis - the final resource count might be different
      Quartus Prime Analysis & Synthesis was successful. 0 errors, 3 warnings
      ********************************************************************
      Running Quartus Prime Fitter
      Command: quartus_fit --read_settings_files=off --write_settings_files=off decoder -c decoder
      qfit2_default_script.tcl version: #1
```

```
      Project  = decoder
      Revision = decoder
 18236 Number of processors has not been specified which may cause overloading on shared machines.  Set the global assignment NUM_PARALLEL_PROCESSORS in your QSF to an appropriate value for best performance.
 20030 Parallel compilation is enabled and will use 6 of the 6 processors detected
119004 Automatically selected device EP2AGX45CU17I3 for design decoder
 21076 High junction temperature operating condition is not set. Assuming a default value of '100'.
 21076 Low junction temperature operating condition is not set. Assuming a default value of '-40'.
171003 Fitter is performing an Auto Fit compilation, which may decrease Fitter effort to reduce compilation time
292013 Feature LogicLock is only available with a valid subscription license. You can purchase a software subscription to gain full access to this feature.
176444 Device migration not selected. If you intend to use device migration later, you may need to change the pin assignments as they may be incompatible with other devices
169124 Fitter converted 1 user pins into dedicated programming pins
 15714 Some pins have incomplete I/O assignments. Refer to the I/O Assignment Warnings report for details
176045 Design uses memory blocks. Violating setup or hold times of memory block address registers for either read or write operations could cause memory contents to be corrupted. Make sure that all memory block address registers meet the setup and hold time requiremen
169085 No exact pin location assignment(s) for 6 pins of 6 total pins. For the list of pins please refer to the I/O Assignment warnings table in the fitter report.
332012 Synopsys Design Constraints File file not found: 'decoder.sdc'. A Synopsys Design Constraints File is required by the Timing Analyzer to get proper timing constraints. Without it, the Compiler will not properly optimize the design.
332144 No user constrained base clocks found in the design
332143 No user constrained clock uncertainty found in the design. Calling "derive_clock_uncertainty"
332154 The derive_clock_uncertainty command did not apply clock uncertainty to any clock-to-clock transfers.
332130 Timing requirements not specified -- quality metrics such as performance may be sacrificed to reduce compilation time.
176353 Automatically promoted node clk~input (placed in PIN T1D (CLK6, DIFFCLK_0p))
176353 Automatically promoted node rst~input (placed in PIN U1D (CLK4, DIFFCLK_0n))
176233 Starting register packing
176235 Finished register packing
176214 Statistics of I/O pins that need to be placed that use the same VCCIO and VREF, before I/O pin placement
176215 I/O bank details before I/O pin placement
171121 Fitter preparation operations ending: elapsed time is 00:00:01
 14896 Fitter has disabled Advanced Physical Optimization because it is not supported for the current family.
170189 Fitter placement preparation operations beginning
170190 Fitter placement preparation operations ending: elapsed time is 00:00:00
170191 Fitter placement operations beginning
170137 Fitter placement was successful
170192 Fitter placement operations ending: elapsed time is 00:00:02
170193 Fitter routing operations beginning
170195 Router estimated average interconnect usage is 0% of the available device resources
170199 The Fitter performed an Auto Fit compilation.  Optimizations were skipped to reduce compilation time.
170194 Fitter routing operations ending: elapsed time is 00:00:01
 11888 Total time spent on timing analysis during the Fitter is 0.40 seconds.
334003 Started post-fitting delay annotation
334004 Delay annotation completed successfully
334003 Started post-fitting delay annotation
334004 Delay annotation completed successfully
 11218 Fitter post-fit operations ending: elapsed time is 00:00:01
185021 Detected external clock signal directly connected to the RAM "mem_disp:disp_mem_1|altsyncram:mem_rtl_0|altsyncram_7pe1:auto_generated|ram_block1a0" CLK0 port. This connection may cause unexpected memory behavior if your external clock signal violates the minimu
185021 Detected external clock signal directly connected to the RAM "mem_disp:disp_mem_0|altsyncram:mem_rtl_0|altsyncram_7pe1:auto_generated|ram_block1a0" CLK0 port. This connection may cause unexpected memory behavior if your external clock signal violates the minimu
185021 Detected external clock signal directly connected to the RAM "mem:trelis_mem_B|altsyncram:mem_rtl_0|altsyncram_epe1:auto_generated|ram_block1a0" CLK0 port. This connection may cause unexpected memory behavior if your external clock signal violates the minimum p
185021 Detected external clock signal directly connected to the RAM "mem:trelis_mem_D|altsyncram:mem_rtl_0|altsyncram_epe1:auto_generated|ram_block1a0" CLK0 port. This connection may cause unexpected memory behavior if your external clock signal violates the minimum p
185021 Detected external clock signal directly connected to the RAM "mem:trelis_mem_A|altsyncram:mem_rtl_0|altsyncram_epe1:auto_generated|ram_block1a0" CLK0 port. This connection may cause unexpected memory behavior if your external clock signal violates the minimum p
185021 Detected external clock signal directly connected to the RAM "mem:trelis_mem_C|altsyncram:mem_rtl_0|altsyncram_epe1:auto_generated|ram_block1a0" CLK0 port. This connection may cause unexpected memory behavior if your external clock signal violates the minimum p
```

```
144001 Generated suppressed messages file E:/damei/term_project_v2/output_files/decoder.fit.smsg
      Quartus Prime Fitter was successful. 0 errors, 11 warnings
      ********************************************************************
      Running Quartus Prime Assembler
      Command: quartus_asm --read_settings_files=off --write_settings_files=off decoder -c decoder
 18236 Number of processors has not been specified which may cause overloading on shared machines.  Set the global assignment NUM_PARALLEL_PROCESSORS in your QSF to an appropriate value for best performance.
115031 Writing out detailed assembly data for power analysis
115030 Assembler is generating device programming files
      Quartus Prime Assembler was successful. 0 errors, 1 warning
293026 Skipped module Power Analyzer due to the assignment FLOW_ENABLE_POWER_ANALYZER
      ********************************************************************
      Running Quartus Prime Timing Analyzer
      Command: quartus_sta decoder -c decoder
      qsta_default_script.tcl version: #1
 18236 Number of processors has not been specified which may cause overloading on shared machines.  Set the global assignment NUM_PARALLEL_PROCESSORS in your QSF to an appropriate value for best performance.
 20030 Parallel compilation is enabled and will use 6 of the 6 processors detected
 21076 High junction temperature operating condition is not set. Assuming a default value of '100'.
 21076 Low junction temperature operating condition is not set. Assuming a default value of '-40'.
332012 Synopsys Design Constraints File file not found: 'decoder.sdc'. A Synopsys Design Constraints File is required by the Timing Analyzer to get proper timing constraints. Without it, the Compiler will not properly optimize the design.
332142 No user constrained base clocks found in the design. Calling "derive_clocks -period 1.0"
332105 Deriving Clocks
332143 No user constrained clock uncertainty found in the design. Calling "derive_clock_uncertainty"
332123 Deriving Clock Uncertainty. Please refer to report_sdc in the Timing Analyzer to see clock uncertainties.
      Found TIMING_ANALYZER_REPORT_SCRIPT_INCLUDE_DEFAULT_ANALYSIS = ON
      Analyzing Slow 900mV 100C Model
332148 Timing requirements not met
332146 Worst-case setup slack is -2.517
332146 Worst-case hold slack is 0.246
332140 No Recovery paths to report
332140 No Removal paths to report
332146 Worst-case minimum pulse width slack is -3.000
332114 Report Metastability: Found 8 synchronizer chains.
      Analyzing Slow 900mV -40C Model
334003 Started post-fitting delay annotation
334004 Delay annotation completed successfully
332123 Deriving Clock Uncertainty. Please refer to report_sdc in the Timing Analyzer to see clock uncertainties.
332148 Timing requirements not met
332146 Worst-case setup slack is -2.335
332146 Worst-case hold slack is 0.209
332140 No Recovery paths to report
332140 No Removal paths to report
332146 Worst-case minimum pulse width slack is -3.000
332114 Report Metastability: Found 8 synchronizer chains.
      Analyzing Fast 900mV -40C Model
332123 Deriving Clock Uncertainty. Please refer to report_sdc in the Timing Analyzer to see clock uncertainties.
332148 Timing requirements not met
332146 Worst-case setup slack is -0.766
332146 Worst-case hold slack is 0.109
```

```
332140 No Recovery paths to report
332140 No Removal paths to report
332146 Worst-case minimum pulse width slack is -2.846
332114 Report Metastability: Found 8 synchronizer chains.
 21076 High junction temperature operating condition is not set. Assuming a default value of '100'.
 21076 Low junction temperature operating condition is not set. Assuming a default value of '-40'.
332123 Deriving Clock Uncertainty. Please refer to report_sdc in the Timing Analyzer to see clock uncertainties.
332102 Design is not fully constrained for setup requirements
332102 Design is not fully constrained for hold requirements
      Quartus Prime Timing Analyzer was successful. 0 errors, 5 warnings
      ********************************************************************
      Running Quartus Prime EDA Netlist Writer
      Command: quartus_eda --read_settings_files=off --write_settings_files=off decoder -c decoder
 18236 Number of processors has not been specified which may cause overloading on shared machines.  Set the global assignment NUM_PARALLEL_PROCESSORS in your QSF to an appropriate value for best performance.
204019 Generated file decoder.svo in folder "E:/damei/term_project_v2/simulation/modelsim/" for EDA simulation tool
      Quartus Prime EDA Netlist Writer was successful. 0 errors, 1 warning
293000 Quartus Prime Full Compilation was successful. 0 errors, 21 warnings
```



(4) Explain briefly how this encoder works, since it differs somewhat from the Homework 7 unit, although similar in principle. Use this encoder with the decoder – your Homework 7 encoder will provide decoder input vectors that won't work with this design.

Term project's encoder uses a recursive, systematic encoder instead of HW7 unit's simpler non-recursive, non-systematic version. In the term project, it has a recursive generator for the encoder and differs a bit from Homework 7. The encoder takes a one bit input and encodes it into 2-bit *d_out* based on the dynamics of a Mealy FSM. Because this is a systematic code for the term project encoder, one of the two bits put out is the original data, meaning that we get to see the data. However, unlike Lab7's nonsystematic encoder, it brings in data and we cannot see the data on the output string, meaning neither sides of the two bits coming out look like the original. Also, its nonrecursive property for Homework 7 encoder is very simple in which the next state takes the current state, left shifts it (doing a shift register), and brings output with no feedback (filled in with input).

The *nstate* (next state of the FSM used to encode next *d_in*) is determined on each clock cycle based on *d_in* and *cstate*. The FSM's current state is stored in *cstate*, and the next state it should transition to based on the current state and input, is calculated and stored in *nstate*. The FSM uses a 3-bit representation for these states, allowing for up to eight distinct states. In addition, the encoder uses a low signal asynchronous reset and is enabled to reset the state. The state transition table is also given by the instruction writeup. [d_out is a function of two of the state variables where d_out_reg[0] is always equal to d_in while d_out_reg[1] has two possibilities (d_in or not d_in) and what it determines is based on what is in current state by d_in^cstate[2]^cstate[1]; if bringing in d_in = 0, the only two possible outputs are 00 and 10 or if bringing in d_in = 1, the only two possible outputs are 01 and 11] The convolutional encoder also uses a rate of 1/2 in homework 7. However, it doesn't use a FSM approach and instead relies on a shift register combined with mask patterns which involves bitwise AND operations between the shift register and the mask patterns, followed by a bitwise XOR to reduce the result to a single bit. The output is a function of multiple bits from the input history, and the masks define the specific relationship.

Because we can run our decoder across each line and down across each column. Then, we decode using an iterative process to get decode values of the line and column in turn (we don't have quite as much horsepower per line since they're shorter). This works much better if we have systematic coding so we can see the original data and correct it as needed. That explains why we need to use the updated encoder2.sv rather than conv_enc.sv in Lab 7 due to its two important properties of recursive and

systematic that would be useful when working with the design of the decoder in our final term project, and also briefly explains why we cannot swap them without modifying the term project decoder.

cstate	d_in = 0		d_in = 1	
	nstate	d_out_reg	nstate	d_out_reg
0	0	00	4	11
1	4	00	0	11
2	5	10	1	01
3	1	10	5	01
4	2	10	6	01
5	6	10	2	01
6	7	00	3	11
7	3	00	7	11

$$p_i[n] = \left(\sum_{j=0}^{K-1} g_i[j] x[n-j] \right) \bmod 2$$

(5) Explain how the decoder works and show that the minimum branch metric = 0 when no errors are present. How does the decoder determine what sequence of bits was sent to the encoder?

Given each encoded signal (could have some errors in transmission), the decoder calculate the hamming distance between the received sequence and what the encoder would have to output for each state transition (there is two possible transition for each state of encoder either 1 or 0) through the BMC modules.Since there are 8 possible states for our encoder, the 8 corresponding BMC modules in our decoder are used to consider each possible state the encoder could have been in.

The branch metrics as well as the accumulated cost (initialized to 0) for each possible transition at state is then used by the ACS modules to evaluate which of the one is more likely. It makes the selection which is used by the TBU to reconstruct the original message as well as updating the accumulated pathcost for each based on the previous decisions made by ACS at previous clock cycle of the and records which paths is chosen through the ACS_xxx_selection and ACS_xxx_pathcost wires. For each state, the decoder keeps track of the "survivor" path, which is the path with the lowest cumulative metric up to the current point. This information is stored in a memory structure often visualized as a trellis diagram. Once the entire sequence has been processed, the decoder finds the ending state with the lowest path metric and traces back through the survivor paths to determine the most likely sequence of states (and hence the sequence of bits) that were transmitted. This traceback starts from the last state in the sequence and works backward to the beginning.

When no errors are present in the received symbols,the received symbols match exactly what was transmitted. Consequently, the branch metrics calculated for the correct path (the hamming distance) would be 0. Therefore, for each step in the decoding process, the correct path's branch metric contributes 0 to the cumulative path metric.

PART 2 DELIVERABLES (with different test files screenshots && transcripts snapshots each included in report):

All necessary working code .sv (viterbi_tx_rx_2a1.sv, viterbi_tx_rx_2a2.sv, …) files are included in the zip folder on gradescope submissions *"part2_viterbi_tx_rx.zip"*.

Test the robustness of your design. Create a table of the number of errors injected, patterns of errors injected, and number of errors uncorrected.

2.a) Try various channel bit error patterns of rate of 1/16:

2.a.1) Invert bit[0] of the channel once every 8 samples.

2.a.2) Repeat for bit[1].

2.a.3) Repeat with bit 0 and 1 both inverted once every 16 samples.

2.a.4) Invert bit[0] twice in a row out of every 16 samples: 14 good, 2 bad, 14 good, 2 bad, … .

2.a.5) Repeat for bit[1]

2.a.6) Invert bit[0] four times in a row out of every 32 samples.

2.a.7) Repeat for bit[1]

2.a.8) Invert bits 1 and 0 twice in a row out of every 32 samples.

2.b) Using the $random function, devise randomized versions of the above tests.

2.c) How long a string of consecutive bad bit[0]s is needed to produce output errors?

2.d) Repeat for bad bit[1].

2.e) Repeat for bit [1] and [0] both.

NonRandom Tests						
Test Number	***ViterbiTxRx***	***BER***	***SymbolPattern***	***BitPattern***	***BadBitsIn***	***BadBitsOut***
2.a.1	*viterbi_tx_rx_2a1.sv*	*1/16*	*1 in a row*	*01*	*31*	*0*
2.a.2	*viterbi_tx_rx_2a2.sv*	*1/16*	*1 in a row*	*10*	*31*	*0*
2.a.3	*viterbi_tx_rx_2a3.sv*	*1/16*	*2 in a row*	*11*	*30*	*0*
2.a.4	*viterbi_tx_rx_2a4.sv*	*1/16*	*1 in a row*	*0101*	*31*	*0*
2.a.5	*viterbi_tx_rx_2a5.sv*	*1/16*	*1 in a row*	*1010*	*31*	*0*
2.a.6	*viterbi_tx_rx_2a6.sv*	*1/16*	*1 in a row*	*01010101*	*31*	*24*
2.a.7	*viterbi_tx_rx_2a7.sv*	*1/16*	*1 in a row*	*10101010*	*31*	*13*
2.a.8	*viterbi_tx_rx_2a8.sv*	*1/16*	*2 in a row*	*1111*	*30*	*18*
Random Tests						
Test Number	***ViterbiTxRx***	***Avg. BER***	***SymbolPattern***	***BitPattern***	***BadBitsIn***	***BadBitsOut***
2.b.1	*viterbi_tx_rx_2b1.sv*	*1/16*	*1 in a row*	*01*	*31*	*0*

2.b.2	*viterbi_tx_rx_2b2.sv*	*1/16*	*1 in a row*	*10*	*37*	*3*
2.b.3	*viterbi_tx_rx_2b3.sv*	*1/16*	*2 in a row*	*11*	*30*	*14*
2.b.4	*viterbi_tx_rx_2b4.sv*	*1/16*	*1 in a row*	*0101*	*32*	*23*
2.b.5	*viterbi_tx_rx_2b5.sv*	*1/16*	*1 in a row*	*1010*	*30*	*8*
2.b.6	*viterbi_tx_rx_2b6.sv*	*1/16*	*1 in a row*	*01010101*	*24*	*24*
2.b.7	*viterbi_tx_rx_2b7.sv*	*1/16*	*1 in a row*	*10101010*	*32*	*16*
2.b.8	*viterbi_tx_rx_2b8.sv*	*1/16*	*2 in a row*	*1111*	*40*	*29*
Length Tests						
Test Number	***ViterbiTxRx***	***Avg. BER***	***SymbolPattern***	***BitPattern***	***BadBitsIn***	***BadBitsOut***
2.c.1	*viterbi_tx_rx_2c1.sv*	*1/16*	*1 in a row*	*01*	*32*	*0*
2.c.2	*viterbi_tx_rx_2c2.sv*	*1/8*	*2 in a row*	*0101*	*64*	*0*
2.c.3	*viterbi_tx_rx_2c3.sv*	*3/16*	*3 in a row*	*010101*	*96*	*121*
2.d.1	*viterbi_tx_rx_2d1.sv*	*1/16*	*1 in a row*	*10*	*32*	*0*
2.d.2	*viterbi_tx_rx_2d2.sv*	*1/8*	*2 in a row*	*1010*	*64*	*0*
2.d.3	*viterbi_tx_rx_2d3.sv*	*3/16*	*3 in a row*	*101010*	*96*	*39*
2.e.1	*viterbi_tx_rx_2e1.sv*	*1/16*	*1 in a row*	*11*	*32*	*0*
2.e.2	*viterbi_tx_rx_2e2.sv*	*1/8*	*2 in a row*	*1111*	*64*	*44*

Part 2 Codes & Transcripts for each case

Part 2 A

2.a.1

```
module viterbi_tx_rx #(parameter N=3) (
   input    clk,
   input    rst,
   input    encoder_i,
   input    enable_encoder_i,
   output   decoder_o);

   wire  [1:0] encoder_o;  // connects encoder to decoder

   int           error_counter,
                 error_counterQ,
                 bad_bit_ct,
                 word_ct;
   logic   [1:0] encoder_o_reg0,
                 encoder_o_reg;
   logic         encoder_i_reg;
   logic         enable_decoder_in;
   logic         enable_encoder_i_reg;
   wire          valid_encoder_o;
   logic   [1:0] err_inj;

   int sample_counter;

   always @ (posedge clk, negedge rst)
      if(!rst) begin
         error_counter        <= 'd0;
//       error_counterQ       <= 'd0;
         encoder_o_reg        <= 'b0;
        encoder_o_reg0        <= 'b0;
         enable_decoder_in    <= 'b0;
        enable_encoder_i_reg <= 'b0;
        word_ct              <= 'b0;
        sample_counter <= 'd0;
      end
      else begin
         enable_encoder_i_reg <= enable_encoder_i;
         enable_decoder_in    <= valid_encoder_o;
         encoder_o_reg        <= 'b0;
         error_counter <= error_counter + 1;
         word_ct              <= word_ct + 1;
         encoder_i_reg     <= encoder_i;
         encoder_o_reg0    <= encoder_o;
         // bit error injection in encoder_o_reg
         if(error_counter[N-1:0]=='1) begin
            encoder_o_reg <= {encoder_o[1], ~encoder_o[0]};    // inject bad bits
            err_inj <= 2'b01;
         end
         else begin              // clean version
            encoder_o_reg  <= encoder_o;
            err_inj        <= 2'b0;
         end

         if(word_ct<256) begin
           bad_bit_ct  <= bad_bit_ct + (encoder_o_reg0[1]^encoder_o_reg[1])
                                     + (encoder_o_reg0[0]^encoder_o_reg[0]);
         $display("error_counter,err_inj = %h %b %d %d",
                 error_counter,err_inj,bad_bit_ct,word_ct);
         end
      end


// insert your convolutional encoder here
// change port names and module name as necessary/desired
   encoder encoder1       (
      .clk,
      .rst,
      .enable_i(enable_encoder_i), //_reg),
      .d_in    (encoder_i), //_reg),
      .valid_o (valid_encoder_o),
      .d_out   (encoder_o)   );

// insert your term project code here
   decoder decoder1       (
      .clk,
      .rst,
      .enable (enable_decoder_in),
      .d_in   (encoder_o_reg),
      .d_out  (decoder_o)   );

endmodule
```

```
VSIM 37> run
# error_counter,err_inj = 00000000 xx          0           0
# error_counter,err_inj = 00000001 00          0           1
# error_counter,err_inj = 00000002 00          0           2
# error_counter,err_inj = 00000003 00          0           3
# error_counter,err_inj = 00000004 00          0           4
# error_counter,err_inj = 00000005 00          0           5
# error_counter,err_inj = 00000006 00          0           6
# error_counter,err_inj = 00000007 00          0           7
# error_counter,err_inj = 00000008 01          0           8
# error_counter,err_inj = 00000009 00          1           9
# error_counter,err_inj = 0000000a 00          1          10
# error_counter,err_inj = 0000000b 00          1          11
# error_counter,err_inj = 0000000c 00          1          12
# error_counter,err_inj = 0000000d 00          1          13
# error_counter,err_inj = 0000000e 00          1          14
# error_counter,err_inj = 0000000f 00          1          15
# error_counter,err_inj = 00000010 01          1          16
# error_counter,err_inj = 00000011 00          2          17
# error_counter,err_inj = 00000012 00          2          18
# error_counter,err_inj = 00000013 00          2          19
# error_counter,err_inj = 00000014 00          2          20
# error_counter,err_inj = 00000015 00          2          21
# error_counter,err_inj = 00000016 00          2          22
# error_counter,err_inj = 00000017 00          2          23
# error_counter,err_inj = 00000018 01          2          24
```

Skipping the repetition.

```
# error_counter,err_inj = 000000f7 00          30          247
# error_counter,err_inj = 000000f8 01          30          248
# error_counter,err_inj = 000000f9 00          31          249
# error_counter,err_inj = 000000fa 00          31          250
# error_counter,err_inj = 000000fb 00          31          251
# error_counter,err_inj = 000000fc 00          31          252
# error_counter,err_inj = 000000fd 00          31          253
# error_counter,err_inj = 000000fe 00          31          254
# error_counter,err_inj = 000000ff 00          31          255
# word_count =       10440
# yaa! in = 0, out = 0, err = 01
# yaa! in = 0, out = 0, err = 01
# yaa! in = 0, out = 0, err = 01
# yaa! in = 0, out = 0, err = 01
# yaa! in = 0, out = 0, err = 01
# yaa! in = 0, out = 0, err = 01
```

Skipping the repetition.

```
# yaa! in = 1, out = 1, err = 01
# yaa! in = 1, out = 1, err = 01
# yaa! in = 1, out = 1, err = 01
# yaa! in = 1, out = 1, err = 01
# yaa! in = 1, out = 1, err = 01
# yaa! in = 1, out = 1, err = 01
# yaa! in = 1, out = 1, err = 01
# yaa! in = 1, out = 1, err = 01
# yaa! in = 1, out = 1, err = 01
# yaa! in = 1, out = 1, err = 01
# yaa! in = 1, out = 1, err = 01
# yaa! in = 1, out = 1, err = 01
# yaa! in = 1, out = 1, err = 01
# yaa! in = 1, out = 1, err = 01
# yaa! in = 1, out = 1, err = 01
# yaa! in = 1, out = 1, err = 01
# yaa! in = 0, out = 0, err = 01
# yaa! in = 1, out = 1, err = 01
# yaa! in = 1, out = 1, err = 01
# yaa! in = 1, out = 1, err = 01
# yaa! in = 1, out = 1, err = 01
# yaa! in = 1, out = 1, err = 01
# yaa! in = 1, out = 1, err = 01
# yaa! in = 1, out = 1, err = 01
# good =         256, bad =           0,            31 bad_bits
# ** Note: $stop    : C:/Users/Charlotte_Qi/Desktop/final_project2/viterbi_tx_rx_tb2.sv(217)
#    Time: 1045 ns  Iteration: 0  Instance: /viterbi_tx_rx_tb
# Break in Module viterbi_tx_rx_tb at C:/Users/Charlotte_Qi/Desktop/final_project2/viterbi_t
```

2.a.2

```
module viterbi_tx_rx #(parameter N=3) (
   input    clk,
   input    rst,
   input    encoder_i,
   input    enable_encoder_i,
   output   decoder_o);

   wire  [1:0] encoder_o;  // connects encoder to decoder

   int           error_counter,
                 error_counterQ,
                 bad_bit_ct,
                 word_ct;
   logic   [1:0] encoder_o_reg0,
                 encoder_o_reg;
   logic         encoder_i_reg;
   logic         enable_decoder_in;
   logic         enable_encoder_i_reg;
   wire          valid_encoder_o;
   logic   [1:0] err_inj;

   int sample_counter;

   always @ (posedge clk, negedge rst)
      if(!rst) begin
         error_counter        <= 'd0;
//       error_counterQ       <= 'd0;
         encoder_o_reg        <= 'b0;
       encoder_o_reg0       <= 'b0;
         enable_decoder_in    <= 'b0;
       enable_encoder_i_reg <= 'b0;
       word_ct              <= 'b0;
       sample_counter <= 'd0;
      end
      else begin
         enable_encoder_i_reg <= enable_encoder_i;
         enable_decoder_in    <= valid_encoder_o;
         encoder_o_reg        <= 'b0;
         error_counter <= error_counter + 1;
         word_ct              <= word_ct + 1;
         encoder_i_reg     <= encoder_i;
         encoder_o_reg0    <= encoder_o;
         // bit error injection in encoder_o_reg
         if(error_counter[N-1:0]=='1) begin
            encoder_o_reg <= {~encoder_o[1], encoder_o[0]};    // inject bad bits
            err_inj <= 2'b10;
         end
         else begin              // clean version
            encoder_o_reg  <= encoder_o;
            err_inj        <= 2'b0;
         end

         if(word_ct<256) begin
          bad_bit_ct  <= bad_bit_ct + (encoder_o_reg0[1]^encoder_o_reg[1])
                                    + (encoder_o_reg0[0]^encoder_o_reg[0]);
         $display("error_counter,err_inj = %h %b %d %d",
                  error_counter,err_inj,bad_bit_ct,word_ct);
         end
      end


   // insert your convolutional encoder here
   // change port names and module name as necessary/desired
      encoder encoder1       (
         .clk,
         .rst,
         .enable_i(enable_encoder_i), //_reg),
         .d_in    (encoder_i), //_reg),
         .valid_o (valid_encoder_o),
         .d_out   (encoder_o)   );

   // insert your term project code here
      decoder decoder1       (
         .clk,
         .rst,
         .enable (enable_decoder_in),
         .d_in   (encoder_o_reg),
         .d_out  (decoder_o)   );

   endmodule
```

```
VSIM 43> run
# error_counter,err_inj = 00000000 xx          0          0
# error_counter,err_inj = 00000001 00          0          1
# error_counter,err_inj = 00000002 00          0          2
# error_counter,err_inj = 00000003 00          0          3
# error_counter,err_inj = 00000004 00          0          4
# error_counter,err_inj = 00000005 00          0          5
# error_counter,err_inj = 00000006 00          0          6
# error_counter,err_inj = 00000007 00          0          7
# error_counter,err_inj = 00000008 10          0          8
# error_counter,err_inj = 00000009 00          1          9
# error_counter,err_inj = 0000000a 00          1         10
# error_counter,err_inj = 0000000b 00          1         11
# error_counter,err_inj = 0000000c 00          1         12
# error_counter,err_inj = 0000000d 00          1         13
# error_counter,err_inj = 0000000e 00          1         14
# error_counter,err_inj = 0000000f 00          1         15
# error_counter,err_inj = 00000010 10          1         16
# error_counter,err_inj = 00000011 00          2         17
# error_counter,err_inj = 00000012 00          2         18
# error_counter,err_inj = 00000013 00          2         19
# error_counter,err_inj = 00000014 00          2         20
# error_counter,err_inj = 00000015 00          2         21
# error_counter,err_inj = 00000016 00          2         22
# error_counter,err_inj = 00000017 00          2         23
# error_counter,err_inj = 00000018 10          2         24
```

Skipping the repetition.

```
# error_counter,err_inj = 000000f7 00          30          247
# error_counter,err_inj = 000000f8 10          30          248
# error_counter,err_inj = 000000f9 00          31          249
# error_counter,err_inj = 000000fa 00          31          250
# error_counter,err_inj = 000000fb 00          31          251
# error_counter,err_inj = 000000fc 00          31          252
# error_counter,err_inj = 000000fd 00          31          253
# error_counter,err_inj = 000000fe 00          31          254
# error_counter,err_inj = 000000ff 00          31          255
# word_count =       10440
# yaa! in = 0, out = 0, err = 10
# yaa! in = 0, out = 0, err = 10
# yaa! in = 0, out = 0, err = 10
# yaa! in = 0, out = 0, err = 10
# yaa! in = 0, out = 0, err = 10
# yaa! in = 0, out = 0, err = 10
# yaa! in = 0, out = 0, err = 10
# yaa! in = 0, out = 0, err = 10
```

Skipping the repetition.

```
# yaa! in = 1, out = 1, err = 10
# yaa! in = 1, out = 1, err = 10
# yaa! in = 1, out = 1, err = 10
# yaa! in = 1, out = 1, err = 10
# yaa! in = 1, out = 1, err = 10
# yaa! in = 1, out = 1, err = 10
# yaa! in = 1, out = 1, err = 10
# yaa! in = 1, out = 1, err = 10
# yaa! in = 1, out = 1, err = 10
# yaa! in = 1, out = 1, err = 10
# yaa! in = 1, out = 1, err = 10
# yaa! in = 1, out = 1, err = 10
# yaa! in = 1, out = 1, err = 10
# yaa! in = 1, out = 1, err = 10
# yaa! in = 1, out = 1, err = 10
# yaa! in = 1, out = 1, err = 10
# yaa! in = 1, out = 1, err = 10
# yaa! in = 1, out = 1, err = 10
# yaa! in = 1, out = 1, err = 10
# yaa! in = 1, out = 1, err = 10
# yaa! in = 0, out = 0, err = 10
# yaa! in = 1, out = 1, err = 10
# yaa! in = 1, out = 1, err = 10
# yaa! in = 1, out = 1, err = 10
# yaa! in = 1, out = 1, err = 10
# yaa! in = 1, out = 1, err = 10
# yaa! in = 1, out = 1, err = 10
# yaa! in = 1, out = 1, err = 10
# good =         256, bad =           0,           31 bad_bits
# ** Note: $stop    : C:/Users/Charlotte_Qi/Desktop/final_project2/viterbi_tx_rx_tb2.sv(217)
#    Time: 1045 ns  Iteration: 0  Instance: /viterbi_tx_rx_tb
```

2.a.3

```
module viterbi_tx_rx #(parameter N=4) (
   input    clk,
   input    rst,
   input    encoder_i,
   input    enable_encoder_i,
   output   decoder_o);

   wire  [1:0] encoder_o;  // connects encoder to decoder

   int           error_counter,
                 error_counterQ,
                 bad_bit_ct,
                 word_ct;
   logic   [1:0] encoder_o_reg0,
                 encoder_o_reg;
   logic         encoder_i_reg;
   logic         enable_decoder_in;
   logic         enable_encoder_i_reg;
   wire          valid_encoder_o;
   logic   [1:0] err_inj;

   int sample_counter;

   always @ (posedge clk, negedge rst)
      if(!rst) begin
         error_counter        <= 'd0;
//       error_counterQ       <= 'd0;
         encoder_o_reg        <= 'b0;
        encoder_o_reg0        <= 'b0;
         enable_decoder_in    <= 'b0;
        enable_encoder_i_reg <= 'b0;
        word_ct               <= 'b0;
        sample_counter <= 'd0;
      end
      else begin
         enable_encoder_i_reg <= enable_encoder_i;
         enable_decoder_in    <= valid_encoder_o;
         encoder_o_reg        <= 'b0;
         error_counter <= error_counter + 1;
         word_ct               <= word_ct + 1;
         encoder_i_reg     <= encoder_i;
         encoder_o_reg0    <= encoder_o;
         // bit error injection in encoder_o_reg
         if(error_counter[N-1:0]=='1) begin
            encoder_o_reg <= {~encoder_o[1], ~encoder_o[0]};    // inject bad bits
            err_inj <= 2'b11;
         end
         else begin              // clean version
            encoder_o_reg  <= encoder_o;
            err_inj        <= 2'b00;
         end

        if(word_ct<256) begin
          bad_bit_ct  <= bad_bit_ct + (encoder_o_reg0[1]^encoder_o_reg[1])
                                    + (encoder_o_reg0[0]^encoder_o_reg[0]);
        $display("error_counter,err_inj = %h %b %d %d",
                 error_counter,err_inj,bad_bit_ct,word_ct);
        end
      end


   // insert your convolutional encoder here
   // change port names and module name as necessary/desired
      encoder encoder1       (
         .clk,
         .rst,
         .enable_i(enable_encoder_i), //_reg),
         .d_in    (encoder_i), //_reg),
         .valid_o (valid_encoder_o),
         .d_out   (encoder_o)   );

   // insert your term project code here
      decoder decoder1       (
         .clk,
         .rst,
         .enable (enable_decoder_in),
         .d_in   (encoder_o_reg),
         .d_out  (decoder_o)   );

   endmodule
```

```
VSIM 51> run
# error_counter,err_inj = 00000000 xx           0           0
# error_counter,err_inj = 00000001 00           0           1
# error_counter,err_inj = 00000002 00           0           2
# error_counter,err_inj = 00000003 00           0           3
# error_counter,err_inj = 00000004 00           0           4
# error_counter,err_inj = 00000005 00           0           5
# error_counter,err_inj = 00000006 00           0           6
# error_counter,err_inj = 00000007 00           0           7
# error_counter,err_inj = 00000008 00           0           8
# error_counter,err_inj = 00000009 00           0           9
# error_counter,err_inj = 0000000a 00           0          10
# error_counter,err_inj = 0000000b 00           0          11
# error_counter,err_inj = 0000000c 00           0          12
# error_counter,err_inj = 0000000d 00           0          13
# error_counter,err_inj = 0000000e 00           0          14
# error_counter,err_inj = 0000000f 00           0          15
# error_counter,err_inj = 00000010 11           0          16
# error_counter,err_inj = 00000011 00           2          17
# error_counter,err_inj = 00000012 00           2          18
# error_counter,err_inj = 00000013 00           2          19
# error_counter,err_inj = 00000014 00           2          20
# error_counter,err_inj = 00000015 00           2          21
# error_counter,err_inj = 00000016 00           2          22
# error_counter,err_inj = 00000017 00           2          23
# error_counter,err_inj = 00000018 00           2          24
# error_counter,err_inj = 00000019 00           2          25
# error_counter,err_inj = 0000001a 00           2          26
# error_counter,err_inj = 0000001b 00           2          27
# error_counter,err_inj = 0000001c 00           2          28
# error_counter,err_inj = 0000001d 00           2          29
# error_counter,err_inj = 0000001e 00           2          30
# error_counter,err_inj = 0000001f 00           2          31
# error_counter,err_inj = 00000020 11           2          32
```

Skipping the repetition.

```
# error_counter,err_inj = 000000ef 00          28          239
# error_counter,err_inj = 000000f0 11          28          240
# error_counter,err_inj = 000000f1 00          30          241
# error_counter,err_inj = 000000f2 00          30          242
# error_counter,err_inj = 000000f3 00          30          243
# error_counter,err_inj = 000000f4 00          30          244
# error_counter,err_inj = 000000f5 00          30          245
# error_counter,err_inj = 000000f6 00          30          246
# error_counter,err_inj = 000000f7 00          30          247
# error_counter,err_inj = 000000f8 00          30          248
# error_counter,err_inj = 000000f9 00          30          249
# error_counter,err_inj = 000000fa 00          30          250
# error_counter,err_inj = 000000fb 00          30          251
# error_counter,err_inj = 000000fc 00          30          252
# error_counter,err_inj = 000000fd 00          30          253
# error_counter,err_inj = 000000fe 00          30          254
# error_counter,err_inj = 000000ff 00          30          255
# word_count =       10440
# yaa! in = 0, out = 0, err = 00
# yaa! in = 0, out = 0, err = 00
# yaa! in = 0, out = 0, err = 00
# yaa! in = 0, out = 0, err = 00
# yaa! in = 0, out = 0, err = 00
# yaa! in = 0, out = 0, err = 00
# yaa! in = 0, out = 0, err = 00
# yaa! in = 0, out = 0, err = 00
# yaa! in = 0, out = 0, err = 00
# yaa! in = 0, out = 0, err = 00
# yaa! in = 0, out = 0, err = 00
# yaa! in = 0, out = 0, err = 00
# yaa! in = 1, out = 1, err = 00
# yaa! in = 0, out = 0, err = 00
# yaa! in = 0, out = 0, err = 00
# yaa! in = 1, out = 1, err = 00
```

Skipping the repetition.

```
# yaa! in = 1, out = 1, err = 00
# yaa! in = 1, out = 1, err = 00
# yaa! in = 1, out = 1, err = 00
# yaa! in = 1, out = 1, err = 00
# yaa! in = 1, out = 1, err = 00
# yaa! in = 1, out = 1, err = 00
# yaa! in = 1, out = 1, err = 00
# yaa! in = 1, out = 1, err = 00
# yaa! in = 0, out = 0, err = 00
# yaa! in = 1, out = 1, err = 00
# yaa! in = 1, out = 1, err = 00
# yaa! in = 1, out = 1, err = 00
# yaa! in = 1, out = 1, err = 00
# yaa! in = 1, out = 1, err = 00
# yaa! in = 1, out = 1, err = 00
# yaa! in = 1, out = 1, err = 00
# good =         256, bad =           0,            30 bad_bits
# ** Note: $stop    : C:/Users/Charlotte_Qi/Desktop/final_project2/viterbi_tx_rx_tb2.sv(217)
#    Time: 1045 ns  Iteration: 0  Instance: /viterbi_tx_rx_tb
```

2.a.4

```
module viterbi_tx_rx #(parameter N=4) (
   input    clk,
   input    rst,
   input    encoder_i,
   input    enable_encoder_i,
   output   decoder_o);

   wire  [1:0] encoder_o;  // connects encoder to decoder

   int           error_counter,
                 error_counterQ,
                 bad_bit_ct,
                 word_ct;
   logic   [1:0] encoder_o_reg0,
                 encoder_o_reg;
   logic         encoder_i_reg;
   logic         enable_decoder_in;
   logic         enable_encoder_i_reg;
   wire          valid_encoder_o;
   logic   [1:0] err_inj;

   int sample_counter;

   always @ (posedge clk, negedge rst)
      if(!rst) begin
         error_counter        <= 'd0;
//       error_counterQ       <= 'd0;
         encoder_o_reg        <= 'b0;
       encoder_o_reg0       <= 'b0;
         enable_decoder_in    <= 'b0;
       enable_encoder_i_reg <= 'b0;
       word_ct              <= 'b0;
       sample_counter <= 'd0;
      end
      else begin
         enable_encoder_i_reg <= enable_encoder_i;
         enable_decoder_in    <= valid_encoder_o;
         encoder_o_reg        <= 'b0;
         error_counter <= error_counter + 1;
         word_ct              <= word_ct + 1;
         encoder_i_reg     <= encoder_i;
         encoder_o_reg0    <= encoder_o;
         // bit error injection in encoder_o_reg
         if(error_counter[N-1:1]=='1) begin
            encoder_o_reg <= {encoder_o[1], ~encoder_o[0]};    // inject bad bits
            err_inj <= 2'b01;
         end
          else begin              // clean version
             encoder_o_reg  <= encoder_o;
             err_inj        <= 2'b00;
          end

         if(word_ct<256) begin
           bad_bit_ct  <= bad_bit_ct + (encoder_o_reg0[1]^encoder_o_reg[1])
                                     + (encoder_o_reg0[0]^encoder_o_reg[0]);
         $display("error_counter,err_inj = %h %b %d %d",
                  error_counter,err_inj,bad_bit_ct,word_ct);
         end
       end


  // insert your convolutional encoder here
  // change port names and module name as necessary/desired
     encoder encoder1       (
        .clk,
        .rst,
        .enable_i(enable_encoder_i), //_reg),
        .d_in    (encoder_i), //_reg),
        .valid_o (valid_encoder_o),
        .d_out   (encoder_o)   );

  // insert your term project code here
     decoder decoder1       (
        .clk,
        .rst,
        .enable (enable_decoder_in),
        .d_in   (encoder_o_reg),
        .d_out  (decoder_o)   );

  endmodule
```

```
VSIM 186> run
# error_counter,err_inj = 00000000 xx          0          0
# error_counter,err_inj = 00000001 00          0          1
# error_counter,err_inj = 00000002 00          0          2
# error_counter,err_inj = 00000003 00          0          3
# error_counter,err_inj = 00000004 00          0          4
# error_counter,err_inj = 00000005 00          0          5
# error_counter,err_inj = 00000006 00          0          6
# error_counter,err_inj = 00000007 00          0          7
# error_counter,err_inj = 00000008 00          0          8
# error_counter,err_inj = 00000009 00          0          9
# error_counter,err_inj = 0000000a 00          0         10
# error_counter,err_inj = 0000000b 00          0         11
# error_counter,err_inj = 0000000c 00          0         12
# error_counter,err_inj = 0000000d 00          0         13
# error_counter,err_inj = 0000000e 00          0         14
# error_counter,err_inj = 0000000f 01          0         15
# error_counter,err_inj = 00000010 01          1         16
# error_counter,err_inj = 00000011 00          2         17
```

Skipping the repetition.

```
# error_counter,err_inj = 000000ee 00          28          238
# error_counter,err_inj = 000000ef 01          28          239
# error_counter,err_inj = 000000f0 01          29          240
# error_counter,err_inj = 000000f1 00          30          241
# error_counter,err_inj = 000000f2 00          30          242
# error_counter,err_inj = 000000f3 00          30          243
# error_counter,err_inj = 000000f4 00          30          244
# error_counter,err_inj = 000000f5 00          30          245
# error_counter,err_inj = 000000f6 00          30          246
# error_counter,err_inj = 000000f7 00          30          247
# error_counter,err_inj = 000000f8 00          30          248
# error_counter,err_inj = 000000f9 00          30          249
# error_counter,err_inj = 000000fa 00          30          250
# error_counter,err_inj = 000000fb 00          30          251
# error_counter,err_inj = 000000fc 00          30          252
# error_counter,err_inj = 000000fd 00          30          253
# error_counter,err_inj = 000000fe 00          30          254
# error_counter,err_inj = 000000ff 01          30          255
# word_count =       10440
# yaa! in = 0, out = 0, w_ct =           0, err = 00
# yaa! in = 0, out = 0, w_ct =           1, err = 00
# yaa! in = 0, out = 0, w_ct =           2, err = 00
# yaa! in = 0, out = 0, w_ct =           3, err = 00
# yaa! in = 0, out = 0, w_ct =           4, err = 00
# yaa! in = 0, out = 0, w_ct =           5, err = 00
# yaa! in = 0, out = 0, w_ct =           6, err = 00
# yaa! in = 0, out = 0, w_ct =           7, err = 00
# yaa! in = 0, out = 0, w_ct =           8, err = 00
# yaa! in = 0, out = 0, w_ct =           9, err = 00
# yaa! in = 0, out = 0, w_ct =          10, err = 00
```

Skipping the repetition.

```
# yaa! in = 1, out = 1, w_ct =         244, err = 00
# yaa! in = 1, out = 1, w_ct =         245, err = 00
# yaa! in = 1, out = 1, w_ct =         246, err = 00
# yaa! in = 1, out = 1, w_ct =         247, err = 00
# yaa! in = 0, out = 0, w_ct =         248, err = 00
# yaa! in = 1, out = 1, w_ct =         249, err = 00
# yaa! in = 1, out = 1, w_ct =         250, err = 00
# yaa! in = 1, out = 1, w_ct =         251, err = 00
# yaa! in = 1, out = 1, w_ct =         252, err = 00
# yaa! in = 1, out = 1, w_ct =         253, err = 00
# yaa! in = 1, out = 1, w_ct =         254, err = 00
# yaa! in = 1, out = 1, w_ct =         255, err = 00
# good =         256, bad =           0,            31 bad_bits
# ** Note: $stop    : C:/Users/Charlotte_Qi/Desktop/final_project2/viterbi_tx_rx_tb2.sv(89)
#    Time: 1045 ns  Iteration: 0  Instance: /viterbi_tx_rx_tb
```

2.a.5

```
module viterbi_tx_rx #(parameter N=4) (
   input    clk,
   input    rst,
   input    encoder_i,
   input    enable_encoder_i,
   output   decoder_o);

   wire  [1:0] encoder_o;  // connects encoder to decoder

   int          error_counter,
                error_counterQ,
                bad_bit_ct,
                word_ct;
   logic   [1:0] encoder_o_reg0,
                encoder_o_reg;
   logic         encoder_i_reg;
   logic         enable_decoder_in;
   logic         enable_encoder_i_reg;
   wire          valid_encoder_o;
   logic   [1:0] err_inj;

   int sample_counter;

   always @ (posedge clk, negedge rst)
      if(!rst) begin
         error_counter        <= 'd0;
//       error_counterQ       <= 'd0;
         encoder_o_reg        <= 'b0;
        encoder_o_reg0       <= 'b0;
         enable_decoder_in    <= 'b0;
        enable_encoder_i_reg <= 'b0;
        word_ct              <= 'b0;
        sample_counter <= 'd0;
      end
      else begin
         enable_encoder_i_reg <= enable_encoder_i;
         enable_decoder_in    <= valid_encoder_o;
         encoder_o_reg        <= 'b0;
         error_counter <= error_counter + 1;
         word_ct              <= word_ct + 1;
         encoder_i_reg     <= encoder_i;
         encoder_o_reg0    <= encoder_o;
         // bit error injection in encoder_o_reg
         if(error_counter[N-1:1]=='1) begin
            encoder_o_reg <= {~encoder_o[1], encoder_o[0]};    // inject bad bits
            err_inj <= 2'b10;
         end
         else begin              // clean version
            encoder_o_reg  <= encoder_o;
            err_inj        <= 2'b00;
         end

         if(word_ct<256) begin
          bad_bit_ct  <= bad_bit_ct + (encoder_o_reg0[1]^encoder_o_reg[1])
                                    + (encoder_o_reg0[0]^encoder_o_reg[0]);
         $display("error_counter,err_inj = %h %b %d %d",
                 error_counter,err_inj,bad_bit_ct,word_ct);
         end
      end


   // insert your convolutional encoder here
   // change port names and module name as necessary/desired
      encoder encoder1       (
         .clk,
         .rst,
         .enable_i(enable_encoder_i), //_reg),
         .d_in    (encoder_i), //_reg),
         .valid_o (valid_encoder_o),
         .d_out   (encoder_o)   );

   // insert your term project code here
      decoder decoder1       (
         .clk,
         .rst,
         .enable (enable_decoder_in),
         .d_in   (encoder_o_reg),
         .d_out  (decoder_o)   );

   endmodule
```

```
VSIM 188> run
# error_counter,err_inj = 00000000 xx          0          0
# error_counter,err_inj = 00000001 00          0          1
# error_counter,err_inj = 00000002 00          0          2
# error_counter,err_inj = 00000003 00          0          3
# error_counter,err_inj = 00000004 00          0          4
# error_counter,err_inj = 00000005 00          0          5
# error_counter,err_inj = 00000006 00          0          6
# error_counter,err_inj = 00000007 00          0          7
# error_counter,err_inj = 00000008 00          0          8
# error_counter,err_inj = 00000009 00          0          9
# error_counter,err_inj = 0000000a 00          0         10
# error_counter,err_inj = 0000000b 00          0         11
# error_counter,err_inj = 0000000c 00          0         12
# error_counter,err_inj = 0000000d 00          0         13
# error_counter,err_inj = 0000000e 00          0         14
# error_counter,err_inj = 0000000f 10          0         15
# error_counter,err_inj = 00000010 10          1         16
# error_counter,err_inj = 00000011 00          2         17
```

Skipping the repetition.

```
# error_counter,err_inj = 000000ef 10          28          239
# error_counter,err_inj = 000000f0 10          29          240
# error_counter,err_inj = 000000f1 00          30          241
# error_counter,err_inj = 000000f2 00          30          242
# error_counter,err_inj = 000000f3 00          30          243
# error_counter,err_inj = 000000f4 00          30          244
# error_counter,err_inj = 000000f5 00          30          245
# error_counter,err_inj = 000000f6 00          30          246
# error_counter,err_inj = 000000f7 00          30          247
# error_counter,err_inj = 000000f8 00          30          248
# error_counter,err_inj = 000000f9 00          30          249
# error_counter,err_inj = 000000fa 00          30          250
# error_counter,err_inj = 000000fb 00          30          251
# error_counter,err_inj = 000000fc 00          30          252
# error_counter,err_inj = 000000fd 00          30          253
# error_counter,err_inj = 000000fe 00          30          254
# error_counter,err_inj = 000000ff 10          30          255
# word_count =       10440
# yaa! in = 0, out = 0, w_ct =           0, err = 00
# yaa! in = 0, out = 0, w_ct =           1, err = 00
# yaa! in = 0, out = 0, w_ct =           2, err = 00
# yaa! in = 0, out = 0, w_ct =           3, err = 00
# yaa! in = 0, out = 0, w_ct =           4, err = 00
# yaa! in = 0, out = 0, w_ct =           5, err = 00
# yaa! in = 0, out = 0, w_ct =           6, err = 00
# yaa! in = 0, out = 0, w_ct =           7, err = 00
# yaa! in = 0, out = 0, w_ct =           8, err = 00
```

Skipping the repetition.

```
# yaa! in = 1, out = 1, w_ct =         246, err = 00
# yaa! in = 1, out = 1, w_ct =         247, err = 00
# yaa! in = 0, out = 0, w_ct =         248, err = 00
# yaa! in = 1, out = 1, w_ct =         249, err = 00
# yaa! in = 1, out = 1, w_ct =         250, err = 00
# yaa! in = 1, out = 1, w_ct =         251, err = 00
# yaa! in = 1, out = 1, w_ct =         252, err = 00
# yaa! in = 1, out = 1, w_ct =         253, err = 00
# yaa! in = 1, out = 1, w_ct =         254, err = 00
# yaa! in = 1, out = 1, w_ct =         255, err = 00
# good =         256, bad =           0,            31 bad_bits
# ** Note: $stop    : C:/Users/Charlotte_Qi/Desktop/final_project2/viterbi_tx_rx_tb2.sv(89)
#    Time: 1045 ns  Iteration: 0  Instance: /viterbi_tx_rx_tb
```

2.a.6

```
module viterbi_tx_rx #(parameter N=5) (
   input    clk,
   input    rst,
   input    encoder_i,
   input    enable_encoder_i,
   output   decoder_o);

   wire  [1:0] encoder_o;  // connects encoder to decoder

   int           error_counter,
                 error_counterQ,
                 bad_bit_ct,
                 word_ct;
   logic   [1:0] encoder_o_reg0,
                 encoder_o_reg;
   logic         encoder_i_reg;
   logic         enable_decoder_in;
   logic         enable_encoder_i_reg;
   wire          valid_encoder_o;
   logic   [1:0] err_inj;

   int sample_counter;

   always @ (posedge clk, negedge rst)
      if(!rst) begin
         error_counter        <= 'd0;
//       error_counterQ       <= 'd0;
         encoder_o_reg        <= 'b0;
       encoder_o_reg0         <= 'b0;
         enable_decoder_in    <= 'b0;
       enable_encoder_i_reg <= 'b0;
       word_ct              <= 'b0;
       sample_counter <= 'd0;
      end
      else begin
         enable_encoder_i_reg <= enable_encoder_i;
         enable_decoder_in    <= valid_encoder_o;
         encoder_o_reg        <= 'b0;
         error_counter <= error_counter + 1;
         word_ct              <= word_ct + 1;
         encoder_i_reg     <= encoder_i;
         encoder_o_reg0    <= encoder_o;
         // bit error injection in encoder_o_reg
         if(error_counter[N-1:2]=='1) begin
            encoder_o_reg <= {encoder_o[1], ~encoder_o[0]};    // inject bad bits
            err_inj <= 2'b01;
         end
         else begin              // clean version
            encoder_o_reg  <= encoder_o;
            err_inj        <= 2'b00;
         end

        if(word_ct<256) begin
          bad_bit_ct  <= bad_bit_ct + (encoder_o_reg0[1]^encoder_o_reg[1])
                                    + (encoder_o_reg0[0]^encoder_o_reg[0]);
        $display("error_counter,err_inj = %h %b %d %d",
                 error_counter,err_inj,bad_bit_ct,word_ct);
        end
      end


// insert your convolutional encoder here
// change port names and module name as necessary/desired
   encoder encoder1       (
      .clk,
      .rst,
      .enable_i(enable_encoder_i), //_reg),
      .d_in    (encoder_i), //_reg),
      .valid_o (valid_encoder_o),
      .d_out   (encoder_o)   );

// insert your term project code here
   decoder decoder1       (
      .clk,
      .rst,
      .enable (enable_decoder_in),
      .d_in   (encoder_o_reg),
      .d_out  (decoder_o)   );

endmodule

```

```
VSIM 190> run
# error_counter,err_inj = 00000000 xx          0          0
# error_counter,err_inj = 00000001 00          0          1
# error_counter,err_inj = 00000002 00          0          2
# error_counter,err_inj = 00000003 00          0          3
# error_counter,err_inj = 00000004 00          0          4
# error_counter,err_inj = 00000005 00          0          5
# error_counter,err_inj = 00000006 00          0          6
# error_counter,err_inj = 00000007 00          0          7
# error_counter,err_inj = 00000008 00          0          8
# error_counter,err_inj = 00000009 00          0          9
# error_counter,err_inj = 0000000a 00          0         10
# error_counter,err_inj = 0000000b 00          0         11
# error_counter,err_inj = 0000000c 00          0         12
# error_counter,err_inj = 0000000d 00          0         13
# error_counter,err_inj = 0000000e 00          0         14
# error_counter,err_inj = 0000000f 00          0         15
# error_counter,err_inj = 00000010 00          0         16
# error_counter,err_inj = 00000011 00          0         17
# error_counter,err_inj = 00000012 00          0         18
# error_counter,err_inj = 00000013 00          0         19
# error_counter,err_inj = 00000014 00          0         20
# error_counter,err_inj = 00000015 00          0         21
# error_counter,err_inj = 00000016 00          0         22
# error_counter,err_inj = 00000017 00          0         23
# error_counter,err_inj = 00000018 00          0         24
# error_counter,err_inj = 00000019 00          0         25
# error_counter,err_inj = 0000001a 00          0         26
# error_counter,err_inj = 0000001b 00          0         27
# error_counter,err_inj = 0000001c 00          0         28
# error_counter,err_inj = 0000001d 01          0         29
# error_counter,err_inj = 0000001e 01          1         30
# error_counter,err_inj = 0000001f 01          2         31
# error_counter,err_inj = 00000020 01          3         32
# error_counter,err_inj = 00000021 00          4         33
```

Skipping the repetition.

```
# yaa! in = 0, out = 0, w_ct =          90, err = 00
# yaa! in = 0, out = 0, w_ct =          91, err = 00
# yaa! in = 0, out = 0, w_ct =          92, err = 00
# yaa! in = 0, out = 0, w_ct =          93, err = 00
# yaa! in = 0, out = 0, w_ct =          94, err = 00
# yaa! in = 0, out = 0, w_ct =          95, err = 00
# yaa! in = 1, out = 1, w_ct =          96, err = 00
# boo! in = 1, out = 0, w_ct =          97, err = 00,              1045000, BAD!
# boo! in = 1, out = 0, w_ct =          98, err = 00,              1045000, BAD!
# yaa! in = 1, out = 1, w_ct =          99, err = 00
# yaa! in = 1, out = 1, w_ct =         100, err = 00
# boo! in = 1, out = 0, w_ct =         101, err = 00,              1045000, BAD!
# boo! in = 1, out = 0, w_ct =         102, err = 00,              1045000, BAD!
# boo! in = 1, out = 0, w_ct =         103, err = 00,              1045000, BAD!
# boo! in = 1, out = 0, w_ct =         104, err = 00,              1045000, BAD!
# yaa! in = 1, out = 1, w_ct =         105, err = 00
# yaa! in = 0, out = 0, w_ct =         106, err = 00
# yaa! in = 0, out = 0, w_ct =         107, err = 00
# yaa! in = 0, out = 0, w_ct =         108, err = 00
# yaa! in = 1, out = 1, w_ct =         109, err = 00
# yaa! in = 0, out = 0, w_ct =         110, err = 00
# yaa! in = 1, out = 1, w_ct =         111, err = 00
# yaa! in = 1, out = 1, w_ct =         112, err = 00
```

Skipping the repetition.

```
# yaa! in = 1, out = 1, w_ct =         247, err = 00
# yaa! in = 0, out = 0, w_ct =         248, err = 00
# yaa! in = 1, out = 1, w_ct =         249, err = 00
# yaa! in = 1, out = 1, w_ct =         250, err = 00
# yaa! in = 1, out = 1, w_ct =         251, err = 00
# yaa! in = 1, out = 1, w_ct =         252, err = 00
# yaa! in = 1, out = 1, w_ct =         253, err = 00
# yaa! in = 1, out = 1, w_ct =         254, err = 00
# yaa! in = 1, out = 1, w_ct =         255, err = 00
# good =         232, bad =          24,            31 bad_bits
# ** Note: $stop    : C:/Users/Charlotte_Qi/Desktop/final_project2/viterbi_tx_rx_tb2.sv(89)
#    Time: 1045 ns  Iteration: 0  Instance: /viterbi_tx_rx_tb
# Break in Module viterbi_tx_rx_tb at C:/Users/Charlotte_Qi/Desktop/final_project2/viterbi_tx
```

2.a.7

```
module viterbi_tx_rx #(parameter N=5) (
   input    clk,
   input    rst,
   input    encoder_i,
   input    enable_encoder_i,
   output   decoder_o);

   wire  [1:0] encoder_o;  // connects encoder to decoder

   int           error_counter,
                 error_counterQ,
                 bad_bit_ct,
                 word_ct;
   logic   [1:0] encoder_o_reg0,
                 encoder_o_reg;
   logic         encoder_i_reg;
   logic         enable_decoder_in;
   logic         enable_encoder_i_reg;
   wire          valid_encoder_o;
   logic   [1:0] err_inj;

   int sample_counter;

   always @ (posedge clk, negedge rst)
      if(!rst) begin
         error_counter        <= 'd0;
//       error_counterQ       <= 'd0;
         encoder_o_reg        <= 'b0;
        encoder_o_reg0       <= 'b0;
         enable_decoder_in    <= 'b0;
        enable_encoder_i_reg <= 'b0;
        word_ct              <= 'b0;
        sample_counter <= 'd0;
      end
      else begin
         enable_encoder_i_reg <= enable_encoder_i;
         enable_decoder_in    <= valid_encoder_o;
         encoder_o_reg        <= 'b0;
         error_counter <= error_counter + 1;
         word_ct              <= word_ct + 1;
         encoder_i_reg     <= encoder_i;
         encoder_o_reg0    <= encoder_o;
         // bit error injection in encoder_o_reg
         if(error_counter[N-1:2]=='1) begin
            encoder_o_reg <= {~encoder_o[1], encoder_o[0]};    // inject bad bits
            err_inj <= 2'b10;
         end
         else begin              // clean version
            encoder_o_reg  <= encoder_o;
            err_inj        <= 2'b00;
         end

         if(word_ct<256) begin
          bad_bit_ct  <= bad_bit_ct + (encoder_o_reg0[1]^encoder_o_reg[1])
                                    + (encoder_o_reg0[0]^encoder_o_reg[0]);
         $display("error_counter,err_inj = %h %b %d %d",
                error_counter,err_inj,bad_bit_ct,word_ct);
         end
      end


// insert your convolutional encoder here
// change port names and module name as necessary/desired
   encoder encoder1       (
      .clk,
      .rst,
      .enable_i(enable_encoder_i), //_reg),
      .d_in    (encoder_i), //_reg),
      .valid_o (valid_encoder_o),
      .d_out   (encoder_o)   );

// insert your term project code here
   decoder decoder1       (
      .clk,
      .rst,
      .enable (enable_decoder_in),
      .d_in   (encoder_o_reg),
      .d_out  (decoder_o)   );

endmodule
```

```
VSIM 192> run
# error_counter,err_inj = 00000000 xx           0           0
# error_counter,err_inj = 00000001 00           0           1
# error_counter,err_inj = 00000002 00           0           2
# error_counter,err_inj = 00000003 00           0           3
# error_counter,err_inj = 00000004 00           0           4
# error_counter,err_inj = 00000005 00           0           5
# error_counter,err_inj = 00000006 00           0           6
# error_counter,err_inj = 00000007 00           0           7
# error_counter,err_inj = 00000008 00           0           8
# error_counter,err_inj = 00000009 00           0           9
# error_counter,err_inj = 0000000a 00           0          10
# error_counter,err_inj = 0000000b 00           0          11
# error_counter,err_inj = 0000000c 00           0          12
# error_counter,err_inj = 0000000d 00           0          13
# error_counter,err_inj = 0000000e 00           0          14
# error_counter,err_inj = 0000000f 00           0          15
# error_counter,err_inj = 00000010 00           0          16
# error_counter,err_inj = 00000011 00           0          17
# error_counter,err_inj = 00000012 00           0          18
# error_counter,err_inj = 00000013 00           0          19
# error_counter,err_inj = 00000014 00           0          20
# error_counter,err_inj = 00000015 00           0          21
# error_counter,err_inj = 00000016 00           0          22
# error_counter,err_inj = 00000017 00           0          23
# error_counter,err_inj = 00000018 00           0          24
# error_counter,err_inj = 00000019 00           0          25
# error_counter,err_inj = 0000001a 00           0          26
# error_counter,err_inj = 0000001b 00           0          27
# error_counter,err_inj = 0000001c 00           0          28
# error_counter,err_inj = 0000001d 10           0          29
# error_counter,err_inj = 0000001e 10           1          30
# error_counter,err_inj = 0000001f 10           2          31
# error_counter,err_inj = 00000020 10           3          32
# error_counter,err_inj = 00000021 00           4          33
```

Skipping the repetition.

```
# yaa! in = 1, out = 1, w_ct =          67, err = 00
# yaa! in = 0, out = 0, w_ct =          68, err = 00
# boo! in = 0, out = 1, w_ct =          69, err = 00,              1045000, BAD!
# yaa! in = 0, out = 0, w_ct =          70, err = 00
# yaa! in = 0, out = 0, w_ct =          71, err = 00
# yaa! in = 0, out = 0, w_ct =          72, err = 00
# yaa! in = 1, out = 1, w_ct =          73, err = 00
# yaa! in = 1, out = 1, w_ct =          74, err = 00
# yaa! in = 1, out = 1, w_ct =          75, err = 00
# boo! in = 1, out = 0, w_ct =          76, err = 00,              1045000, BAD!
# yaa! in = 1, out = 1, w_ct =          77, err = 00
# yaa! in = 0, out = 0, w_ct =          78, err = 00
# yaa! in = 1, out = 1, w_ct =          79, err = 00
# yaa! in = 0, out = 0, w_ct =          80, err = 00
# yaa! in = 1, out = 1, w_ct =          81, err = 00
# yaa! in = 0, out = 0, w_ct =          82, err = 00
# yaa! in = 1, out = 1, w_ct =          83, err = 00
# yaa! in = 0, out = 0, w_ct =          84, err = 00
# yaa! in = 1, out = 1, w_ct =          85, err = 00
# yaa! in = 0, out = 0, w_ct =          86, err = 00
# yaa! in = 0, out = 0, w_ct =          87, err = 00
# yaa! in = 0, out = 0, w_ct =          88, err = 00
# yaa! in = 0, out = 0, w_ct =          89, err = 00
# yaa! in = 0, out = 0, w_ct =          90, err = 00
# yaa! in = 0, out = 0, w_ct =          91, err = 00
# yaa! in = 0, out = 0, w_ct =          92, err = 00
# yaa! in = 0, out = 0, w_ct =          93, err = 00
# yaa! in = 0, out = 0, w_ct =          94, err = 00
# yaa! in = 0, out = 0, w_ct =          95, err = 00
# yaa! in = 1, out = 1, w_ct =          96, err = 00
# yaa! in = 1, out = 1, w_ct =          97, err = 00
# yaa! in = 1, out = 1, w_ct =          98, err = 00
# yaa! in = 1, out = 1, w_ct =          99, err = 00
# yaa! in = 1, out = 1, w_ct =         100, err = 00
# boo! in = 1, out = 0, w_ct =         101, err = 00,              1045000, BAD!
# yaa! in = 1, out = 1, w_ct =         102, err = 00
# boo! in = 1, out = 0, w_ct =         103, err = 00,              1045000, BAD!
# boo! in = 1, out = 0, w_ct =         104, err = 00,              1045000, BAD!
# yaa! in = 1, out = 1, w_ct =         105, err = 00
# yaa! in = 0, out = 0, w_ct =         106, err = 00
```

Skipping the repetition.

```
# yaa! in = 1, out = 1, w_ct =         244, err = 00
# yaa! in = 1, out = 1, w_ct =         245, err = 00
# yaa! in = 1, out = 1, w_ct =         246, err = 00
# yaa! in = 1, out = 1, w_ct =         247, err = 00
# yaa! in = 0, out = 0, w_ct =         248, err = 00
# yaa! in = 1, out = 1, w_ct =         249, err = 00
# yaa! in = 1, out = 1, w_ct =         250, err = 00
# yaa! in = 1, out = 1, w_ct =         251, err = 00
# yaa! in = 1, out = 1, w_ct =         252, err = 00
# yaa! in = 1, out = 1, w_ct =         253, err = 00
# yaa! in = 1, out = 1, w_ct =         254, err = 00
# yaa! in = 1, out = 1, w_ct =         255, err = 00
# good =         243, bad =          13,            31 bad_bits
# ** Note: $stop    : C:/Users/Charlotte_Qi/Desktop/final_project2/viterbi_tx_rx_tb2.sv(89)
#    Time: 1045 ns  Iteration: 0  Instance: /viterbi_tx_rx_tb
# Break in Module viterbi_tx_rx_tb at C:/Users/Charlotte_Qi/Desktop/final_project2/viterbi
```

2.a.8

```
module viterbi_tx_rx #(parameter N=5) (
   input    clk,
   input    rst,
   input    encoder_i,
   input    enable_encoder_i,
   output   decoder_o);

   wire  [1:0] encoder_o;  // connects encoder to decoder

   int          error_counter,
                error_counterQ,
                bad_bit_ct,
                word_ct;
   logic  [1:0] encoder_o_reg0,
                encoder_o_reg;
   logic        encoder_i_reg;
   logic        enable_decoder_in;
   logic        enable_encoder_i_reg;
   wire         valid_encoder_o;
   logic  [1:0] err_inj;

   int sample_counter;

   always @ (posedge clk, negedge rst)
      if(!rst) begin
         error_counter        <= 'd0;
//       error_counterQ       <= 'd0;
         encoder_o_reg        <= 'b0;
       encoder_o_reg0       <= 'b0;
         enable_decoder_in    <= 'b0;
       enable_encoder_i_reg <= 'b0;
       word_ct              <= 'b0;
       sample_counter <= 'd0;
      end
      else begin
         enable_encoder_i_reg <= enable_encoder_i;
         enable_decoder_in    <= valid_encoder_o;
         encoder_o_reg        <= 'b0;
         error_counter <= error_counter + 1;
         word_ct              <= word_ct + 1;
         encoder_i_reg     <= encoder_i;
         encoder_o_reg0    <= encoder_o;
         // bit error injection in encoder_o_reg
         if(error_counter[N-1:1]=='1) begin
            encoder_o_reg <= {~encoder_o[1], ~encoder_o[0]};    // inject bad bits
            err_inj <= 2'b11;
         end
         else begin              // clean version
            encoder_o_reg  <= encoder_o;
            err_inj        <= 2'b00;
         end

         if(word_ct<256) begin
          bad_bit_ct  <= bad_bit_ct + (encoder_o_reg0[1]^encoder_o_reg[1])
                                    + (encoder_o_reg0[0]^encoder_o_reg[0]);
         $display("error_counter,err_inj = %h %b %d %d",
                  error_counter,err_inj,bad_bit_ct,word_ct);
         end
      end


// insert your convolutional encoder here
// change port names and module name as necessary/desired
   encoder encoder1       (
      .clk,
      .rst,
      .enable_i(enable_encoder_i), //_reg),
      .d_in    (encoder_i), //_reg),
      .valid_o (valid_encoder_o),
      .d_out   (encoder_o)   );

// insert your term project code here
   decoder decoder1       (
      .clk,
      .rst,
      .enable (enable_decoder_in),
      .d_in   (encoder_o_reg),
      .d_out  (decoder_o)   );

endmodule

```

```
VSIM 196> run
# error_counter,err_inj = 00000000 xx           0           0
# error_counter,err_inj = 00000001 00           0           1
# error_counter,err_inj = 00000002 00           0           2
# error_counter,err_inj = 00000003 00           0           3
# error_counter,err_inj = 00000004 00           0           4
# error_counter,err_inj = 00000005 00           0           5
# error_counter,err_inj = 00000006 00           0           6
# error_counter,err_inj = 00000007 00           0           7
# error_counter,err_inj = 00000008 00           0           8
# error_counter,err_inj = 00000009 00           0           9
# error_counter,err_inj = 0000000a 00           0          10
# error_counter,err_inj = 0000000b 00           0          11
# error_counter,err_inj = 0000000c 00           0          12
# error_counter,err_inj = 0000000d 00           0          13
# error_counter,err_inj = 0000000e 00           0          14
# error_counter,err_inj = 0000000f 00           0          15
# error_counter,err_inj = 00000010 00           0          16
# error_counter,err_inj = 00000011 00           0          17
# error_counter,err_inj = 00000012 00           0          18
# error_counter,err_inj = 00000013 00           0          19
# error_counter,err_inj = 00000014 00           0          20
# error_counter,err_inj = 00000015 00           0          21
# error_counter,err_inj = 00000016 00           0          22
# error_counter,err_inj = 00000017 00           0          23
# error_counter,err_inj = 00000018 00           0          24
# error_counter,err_inj = 00000019 00           0          25
# error_counter,err_inj = 0000001a 00           0          26
# error_counter,err_inj = 0000001b 00           0          27
# error_counter,err_inj = 0000001c 00           0          28
# error_counter,err_inj = 0000001d 00           0          29
# error_counter,err_inj = 0000001e 00           0          30
# error_counter,err_inj = 0000001f 11           0          31
# error_counter,err_inj = 00000020 11           2          32
# error_counter,err_inj = 00000021 00           4          33
```

Skipping the repetition.

```
# error_counter,err_inj = 000000f5 00          28          245
# error_counter,err_inj = 000000f6 00          28          246
# error_counter,err_inj = 000000f7 00          28          247
# error_counter,err_inj = 000000f8 00          28          248
# error_counter,err_inj = 000000f9 00          28          249
# error_counter,err_inj = 000000fa 00          28          250
# error_counter,err_inj = 000000fb 00          28          251
# error_counter,err_inj = 000000fc 00          28          252
# error_counter,err_inj = 000000fd 00          28          253
# error_counter,err_inj = 000000fe 00          28          254
# error_counter,err_inj = 000000ff 11          28          255
# word_count =       10440
# yaa! in = 0, out = 0, w_ct =           0, err = 00
# yaa! in = 0, out = 0, w_ct =           1, err = 00
# yaa! in = 0, out = 0, w_ct =           2, err = 00
# yaa! in = 0, out = 0, w_ct =           3, err = 00
# yaa! in = 0, out = 0, w_ct =           4, err = 00
# yaa! in = 0, out = 0, w_ct =           5, err = 00
# yaa! in = 0, out = 0, w_ct =           6, err = 00
# yaa! in = 0, out = 0, w_ct =           7, err = 00
# yaa! in = 0, out = 0, w_ct =           8, err = 00
```

Skipping the repetition.

```
# yaa! in = 1, out = 1, w_ct =          99, err = 00
# yaa! in = 1, out = 1, w_ct =         100, err = 00
# boo! in = 1, out = 0, w_ct =         101, err = 00,             1045000, BAD!
# yaa! in = 1, out = 1, w_ct =         102, err = 00
# boo! in = 1, out = 0, w_ct =         103, err = 00,             1045000, BAD!
# boo! in = 1, out = 0, w_ct =         104, err = 00,             1045000, BAD!
# yaa! in = 1, out = 1, w_ct =         105, err = 00
# yaa! in = 0, out = 0, w_ct =         106, err = 00
# yaa! in = 0, out = 0, w_ct =         107, err = 00
# yaa! in = 0, out = 0, w_ct =         108, err = 00
# yaa! in = 1, out = 1, w_ct =         109, err = 00
# yaa! in = 0, out = 0, w_ct =         110, err = 00
# yaa! in = 1, out = 1, w_ct =         111, err = 00
# yaa! in = 1, out = 1, w_ct =         112, err = 00
# yaa! in = 0, out = 0, w_ct =         113, err = 00
# yaa! in = 0, out = 0, w_ct =         114, err = 00
# yaa! in = 0, out = 0, w_ct =         115, err = 00
# yaa! in = 1, out = 1, w_ct =         116, err = 00
# yaa! in = 0, out = 0, w_ct =         117, err = 00
# yaa! in = 1, out = 1, w_ct =         118, err = 00
# yaa! in = 1, out = 1, w_ct =         119, err = 00
# yaa! in = 1, out = 1, w_ct =         120, err = 00
# yaa! in = 1, out = 1, w_ct =         121, err = 00
# yaa! in = 0, out = 0, w_ct =         122, err = 00
# yaa! in = 0, out = 0, w_ct =         123, err = 00
# yaa! in = 1, out = 1, w_ct =         124, err = 00
# yaa! in = 0, out = 0, w_ct =         125, err = 00
# yaa! in = 0, out = 0, w_ct =         126, err = 00
# yaa! in = 0, out = 0, w_ct =         127, err = 00
# yaa! in = 0, out = 0, w_ct =         128, err = 00
# yaa! in = 0, out = 0, w_ct =         129, err = 00
# yaa! in = 0, out = 0, w_ct =         130, err = 00
# yaa! in = 0, out = 0, w_ct =         131, err = 00
# yaa! in = 0, out = 0, w_ct =         132, err = 00
# yaa! in = 0, out = 0, w_ct =         133, err = 00
# yaa! in = 0, out = 0, w_ct =         134, err = 00
# boo! in = 0, out = 1, w_ct =         135, err = 00,             1045000, BAD!
# boo! in = 0, out = 1, w_ct =         136, err = 00,             1045000, BAD!
# yaa! in = 1, out = 1, w_ct =         137, err = 00
# yaa! in = 1, out = 1, w_ct =         138, err = 00
# yaa! in = 1, out = 1, w_ct =         139, err = 00
# boo! in = 1, out = 0, w_ct =         140, err = 00,             1045000, BAD!
# yaa! in = 1, out = 1, w_ct =         141, err = 00
```

Skipping the repetition.

```
# yaa! in = 1, out = 1, w_ct =         250, err = 00
# yaa! in = 1, out = 1, w_ct =         251, err = 00
# yaa! in = 1, out = 1, w_ct =         252, err = 00
# yaa! in = 1, out = 1, w_ct =         253, err = 00
# yaa! in = 1, out = 1, w_ct =         254, err = 00
# yaa! in = 1, out = 1, w_ct =         255, err = 00
# good =         238, bad =          18,            30 bad_bits
# ** Note: $stop    : C:/Users/Charlotte_Qi/Desktop/final_project2/viterbi_tx_rx_tb2.sv(89)
#    Time: 1045 ns  Iteration: 0  Instance: /viterbi_tx_rx_tb
```

Part 2 B

2.b.1

```
module viterbi_tx_rx #(parameter N=3) (
   input    clk,
   input    rst,
   input    encoder_i,
   input    enable_encoder_i,
   output   decoder_o);

   wire  [1:0] encoder_o;  // connects encoder to decoder

   int           error_counter,
                 error_counterQ,
                 bad_bit_ct,
                 word_ct;
   logic   [1:0] encoder_o_reg0,
                 encoder_o_reg;
   logic         encoder_i_reg;
   logic         enable_decoder_in;
   logic         enable_encoder_i_reg;
   wire          valid_encoder_o;
   logic   [1:0] err_inj;

   always @ (posedge clk, negedge rst)
      if(!rst) begin
         error_counter        <= 'd0;
//       error_counterQ       <= 'd0;
         encoder_o_reg        <= 'b0;
        encoder_o_reg0        <= 'b0;
         enable_decoder_in    <= 'b0;
        enable_encoder_i_reg  <= 'b0;
        word_ct               <= 'b0;
      end
      else begin
         enable_encoder_i_reg <= enable_encoder_i;
         enable_decoder_in    <= valid_encoder_o;
//          encoder_o_reg        <= 'b0;
        // if(word_ct % 8 == 0) begin
           error_counter <= $random >> 1;
           // error_counterQ <= error_counter;
        // end
        word_ct               <= word_ct + 1;
// bit error injection in encoder_o_reg
        encoder_i_reg     <= encoder_i;
        encoder_o_reg0    <= encoder_o;
        if(error_counter[N-1:0]=='1) begin //
           encoder_o_reg <= {encoder_o[1], ~encoder_o[0]};    // inject bad bits
           err_inj <= 2'b01;
        end
         else begin              // clean version
            encoder_o_reg  <= encoder_o;
            err_inj        <= 2'b0;
        end
        if(word_ct<256) begin
          bad_bit_ct  <= bad_bit_ct + (encoder_o_reg0[1]^encoder_o_reg[1])
                                    + (encoder_o_reg0[0]^encoder_o_reg[0]);
         $display("error_counter,err_inj = %h %b %d %d",
                 error_counter,err_inj,bad_bit_ct,word_ct);
         end
      end


// insert your convolutional encoder here
// change port names and module name as necessary/desired
   encoder encoder1       (
      .clk,
      .rst,
      .enable_i(enable_encoder_i), //_reg),
      .d_in    (encoder_i),        //_reg),
      .valid_o (valid_encoder_o),
      .d_out   (encoder_o)   );

// insert your term project code here
   decoder decoder1       (
      .clk,
      .rst,
      .enable (enable_decoder_in),
      .d_in   (encoder_o_reg),
      .d_out  (decoder_o)   );

endmodule
```

```
VSIM 198> run
# error_counter,err_inj = 00000000 xx           0           0
# error_counter,err_inj = 090a9a92 00           0           1
# error_counter,err_inj = 6044af40 00           0           2
# error_counter,err_inj = 42426b04 00           0           3
# error_counter,err_inj = 58f82b31 00           0           4
# error_counter,err_inj = 035cbd86 00           0           5
# error_counter,err_inj = 236fccc6 00           0           6
# error_counter,err_inj = 59614232 00           0           7
# error_counter,err_inj = 449ba909 00           0           8
# error_counter,err_inj = 0079f180 00           0           9
# error_counter,err_inj = 036be686 00           0          10
# error_counter,err_inj = 1d91f8bb 00           0          11
# error_counter,err_inj = 0f46e69e 00           0          12
# error_counter,err_inj = 3b6a2bf6 00           0          13
# error_counter,err_inj = 2316fbc6 00           0          14
# error_counter,err_inj = 3e7ef4fc 00           0          15
# error_counter,err_inj = 719b9263 00           0          16
# error_counter,err_inj = 717bc262 00           0          17
# error_counter,err_inj = 6a89e955 00           0          18
# error_counter,err_inj = 3957fbf2 00           0          19
# error_counter,err_inj = 5de9393b 00           0          20
# error_counter,err_inj = 44996b09 00           0          21
# error_counter,err_inj = 23f66dc7 00           0          22
# error_counter,err_inj = 3c9834f9 01           0          23
# error_counter,err_inj = 73bb4b67 00           1          24
# error_counter,err_inj = 7a003d74 01           1          25
# error_counter,err_inj = 71652762 00           2          26
# error_counter,err_inj = 172c24ae 00           2          27
# error_counter,err_inj = 6f47145e 00           2          28
# error_counter,err_inj = 4b55ac16 00           2          29
# error_counter,err_inj = 59539332 00           2          30
# error_counter,err_inj = 58f7b131 00           2          31
# error_counter,err_inj = 02b9c385 00           2          32
```

```
# good =         256, bad =           0,            31 bad_bits
# ** Note: $stop    : C:/Users/Charlotte_Qi/Desktop/final_project2/viterbi_tx_rx_tb2.sv(89)
#    Time: 1045 ns  Iteration: 0  Instance: /viterbi_tx_rx_tb
```

2.b.2

```
module viterbi_tx_rx #(parameter N=3) (
   input    clk,
   input    rst,
   input    encoder_i,
   input    enable_encoder_i,
   output   decoder_o);

   wire  [1:0] encoder_o;  // connects encoder to decoder

   int          error_counter,
                error_counterQ,
                bad_bit_ct,
                word_ct;
   logic   [1:0] encoder_o_reg0,
                encoder_o_reg;
   logic        encoder_i_reg;
   logic        enable_decoder_in;
   logic        enable_encoder_i_reg;
   wire         valid_encoder_o;
   logic   [1:0] err_inj;

   always @ (posedge clk, negedge rst)
      if(!rst) begin
         error_counter        <= 'd0;
//       error_counterQ       <= 'd0;
         encoder_o_reg        <= 'b0;
        encoder_o_reg0       <= 'b0;
         enable_decoder_in    <= 'b0;
        enable_encoder_i_reg <= 'b0;
        word_ct              <= 'b0;
      end
      else begin
         enable_encoder_i_reg <= enable_encoder_i;
         enable_decoder_in    <= valid_encoder_o;
//          encoder_o_reg        <= 'b0;
         error_counter        <= $random >> 4;
         word_ct              <= word_ct + 1;
// bit error injection in encoder_o_reg
         encoder_i_reg     <= encoder_i;
         encoder_o_reg0    <= encoder_o;
//       error_counterQ    <= error_counter;
         if(error_counter[N-1:0]=='1) begin //
         //  err_inj        <= error_counter[29:28];
         //    encoder_o_reg  <= encoder_o^err_inj;
            encoder_o_reg <= {~encoder_o[1], encoder_o[0]};   // inject bad bits
            err_inj <= 2'b10;
         end
         else begin              // clean version
            encoder_o_reg  <= encoder_o;
            err_inj        <= 2'b0;
       end
        if(word_ct<256) begin
         bad_bit_ct  <= bad_bit_ct + (encoder_o_reg0[1]^encoder_o_reg[1])
                                   + (encoder_o_reg0[0]^encoder_o_reg[0]);
        $display("error_counter,err_inj = %h %b %d %d",
               error_counter,err_inj,bad_bit_ct,word_ct);
        end
      end


// insert your convolutional encoder here
// change port names and module name as necessary/desired
   encoder encoder1       (
      .clk,
      .rst,
      .enable_i(enable_encoder_i), //_reg),
      .d_in    (encoder_i),        //_reg),
      .valid_o (valid_encoder_o),
      .d_out   (encoder_o)   );

// insert your term project code here
   decoder decoder1       (
      .clk,
      .rst,
      .enable (enable_decoder_in),
      .d_in   (encoder_o_reg),
      .d_out  (decoder_o)   );

endmodule

```

```
VSIM 200> run
# error_counter,err_inj = 00000000 xx          0           0
# error_counter,err_inj = 01215352 00          0           1
# error_counter,err_inj = 0c0895e8 00          0           2
# error_counter,err_inj = 08484d60 00          0           3
# error_counter,err_inj = 0b1f0566 00          0           4
# error_counter,err_inj = 006b97b0 00          0           5
# error_counter,err_inj = 046df998 00          0           6
# error_counter,err_inj = 0b2c2846 00          0           7
# error_counter,err_inj = 08937521 00          0           8
# error_counter,err_inj = 000f3e30 00          0           9
# error_counter,err_inj = 006d7cd0 00          0          10
# error_counter,err_inj = 03b23f17 00          0          11
# error_counter,err_inj = 01e8dcd3 10          0          12
# error_counter,err_inj = 076d457e 00          1          13
# error_counter,err_inj = 0462df78 00          1          14
# error_counter,err_inj = 07cfde9f 00          1          15
# error_counter,err_inj = 0e33724c 10          1          16
# error_counter,err_inj = 0e2f784c 00          2          17
# error_counter,err_inj = 0d513d2a 00          2          18
# error_counter,err_inj = 072aff7e 00          2          19
# error_counter,err_inj = 0bbd2727 00          2          20
# error_counter,err_inj = 08932d61 10          2          21
# error_counter,err_inj = 047ecdb8 00          3          22
# error_counter,err_inj = 0793069f 00          3          23
# error_counter,err_inj = 0e77696c 10          3          24
# error_counter,err_inj = 0f4007ae 00          4          25
# error_counter,err_inj = 0e2ca4ec 00          4          26
# error_counter,err_inj = 02e58495 00          4          27
# error_counter,err_inj = 0de8e28b 00          4          28
# error_counter,err_inj = 096ab582 00          4          29
# error_counter,err_inj = 0b2a7266 00          4          30
# error_counter,err_inj = 0b1ef626 00          4          31
# error_counter,err_inj = 00573870 00          4          32
```

```
# good =         253, bad =           3,            37 bad_bits
# ** Note: $stop    : C:/Users/Charlotte_Qi/Desktop/final_project2/viterbi_tx_rx_tb2.sv(89)
#    Time: 1045 ns  Iteration: 0  Instance: /viterbi_tx_rx_tb
```

2.b.3

```
module viterbi_tx_rx #(parameter N=4) (
   input    clk,
   input    rst,
   input    encoder_i,
   input    enable_encoder_i,
   output   decoder_o);

   wire  [1:0] encoder_o;  // connects encoder to decoder

   int           error_counter,
                 error_counterQ,
                 bad_bit_ct,
                 word_ct;
   logic   [1:0] encoder_o_reg0,
                 encoder_o_reg;
   logic         encoder_i_reg;
   logic         enable_decoder_in;
   logic         enable_encoder_i_reg;
   wire          valid_encoder_o;
   logic   [1:0] err_inj;

   always @ (posedge clk, negedge rst)
      if(!rst) begin
         error_counter        <= 'd0;
//     error_counterQ       <= 'd0;
         encoder_o_reg        <= 'b0;
        encoder_o_reg0       <= 'b0;
         enable_decoder_in    <= 'b0;
        enable_encoder_i_reg <= 'b0;
        word_ct              <= 'b0;
      end
      else begin
         enable_encoder_i_reg <= enable_encoder_i;
         enable_decoder_in    <= valid_encoder_o;
//          encoder_o_reg        <= 'b0;
         error_counter        <= $random >> 5;
         word_ct              <= word_ct + 1;
// bit error injection in encoder_o_reg
         encoder_i_reg     <= encoder_i;
         encoder_o_reg0    <= encoder_o;
//       error_counterQ    <= error_counter;
         if(error_counter[N-1:0]=='1) begin //
         //  err_inj        <= error_counter[29:28];
         //    encoder_o_reg  <= encoder_o^err_inj;
            encoder_o_reg <= {~encoder_o[1], ~encoder_o[0]};   // inject bad bits
            err_inj <= 2'b11;
         end
         else begin              // clean version
            encoder_o_reg  <= encoder_o;
            err_inj        <= 2'b0;
      end
        if(word_ct<256) begin
         bad_bit_ct  <= bad_bit_ct + (encoder_o_reg0[1]^encoder_o_reg[1])
                                   + (encoder_o_reg0[0]^encoder_o_reg[0]);
        $display("error_counter,err_inj = %h %b %d %d",
              error_counter,err_inj,bad_bit_ct,word_ct);
        end
      end


// insert your convolutional encoder here
// change port names and module name as necessary/desired
   encoder encoder1       (
      .clk,
      .rst,
      .enable_i(enable_encoder_i), //_reg),
      .d_in    (encoder_i),        //_reg),
      .valid_o (valid_encoder_o),
      .d_out   (encoder_o)   );

// insert your term project code here
   decoder decoder1       (
      .clk,
      .rst,
      .enable (enable_decoder_in),
      .d_in   (encoder_o_reg),
      .d_out  (decoder_o)   );

endmodule
```

```
VSIM 202> run
# error_counter,err_inj = 00000000 xx           0           0
# error_counter,err_inj = 0090a9a9 00           0           1
# error_counter,err_inj = 06044af4 00           0           2
# error_counter,err_inj = 042426b0 00           0           3
# error_counter,err_inj = 058f82b3 00           0           4
# error_counter,err_inj = 0035cbd8 00           0           5
# error_counter,err_inj = 0236fccc 00           0           6
# error_counter,err_inj = 05961423 00           0           7
# error_counter,err_inj = 0449ba90 00           0           8
# error_counter,err_inj = 00079f18 00           0           9
# error_counter,err_inj = 0036be68 00           0          10
# error_counter,err_inj = 01d91f8b 00           0          11
# error_counter,err_inj = 00f46e69 00           0          12
# error_counter,err_inj = 03b6a2bf 00           0          13
# error_counter,err_inj = 02316fbc 11           0          14
# error_counter,err_inj = 03e7ef4f 00           2          15
# error_counter,err_inj = 0719b926 11           2          16
# error_counter,err_inj = 0717bc26 00           4          17
# error_counter,err_inj = 06a89e95 00           4          18
# error_counter,err_inj = 03957fbf 00           4          19
# error_counter,err_inj = 05de9393 11           4          20
# error_counter,err_inj = 044996b0 00           6          21
# error_counter,err_inj = 023f66dc 00           6          22
# error_counter,err_inj = 03c9834f 00           6          23
# error_counter,err_inj = 073bb4b6 11           6          24
# error_counter,err_inj = 07a003d7 00           8          25
# error_counter,err_inj = 07165276 00           8          26
# error_counter,err_inj = 0172c24a 00           8          27
# error_counter,err_inj = 06f47145 00           8          28
# error_counter,err_inj = 04b55ac1 00           8          29
# error_counter,err_inj = 05953933 00           8          30
# error_counter,err_inj = 058f7b13 00           8          31
# error_counter,err_inj = 002b9c38 00           8          32
```

```
# good =          242, bad =          14,            30 bad_bits
# ** Note: $stop    : C:/Users/Charlotte_Qi/Desktop/final_project2/viterbi_tx_rx_tb2.sv(89)
#    Time: 1045 ns  Iteration: 0  Instance: /viterbi_tx_rx_tb
```

2.b.4

```
module viterbi_tx_rx #(parameter N=4) (
   input    clk,
   input    rst,
   input    encoder_i,
   input    enable_encoder_i,
   output   decoder_o);

   wire  [1:0] encoder_o;  // connects encoder to decoder

   int           error_counter,
                 error_counterQ,
                 bad_bit_ct,
                 word_ct;
   logic   [1:0] encoder_o_reg0,
                 encoder_o_reg;
   logic         encoder_i_reg;
   logic         enable_decoder_in;
   logic         enable_encoder_i_reg;
   wire          valid_encoder_o;
   logic   [1:0] err_inj;

   always @ (posedge clk, negedge rst)
      if(!rst) begin
         error_counter        <= 'd0;
//       error_counterQ       <= 'd0;
         encoder_o_reg        <= 'b0;
        encoder_o_reg0        <= 'b0;
         enable_decoder_in    <= 'b0;
        enable_encoder_i_reg <= 'b0;
        word_ct               <= 'b0;
      end
      else begin
         enable_encoder_i_reg <= enable_encoder_i;
         enable_decoder_in    <= valid_encoder_o;
//         encoder_o_reg        <= 'b0;
         if (word_ct[0] == '1) error_counter        <= $random >> 7;
         word_ct              <= word_ct + 1;
// bit error injection in encoder_o_reg
         encoder_i_reg     <= encoder_i;
         encoder_o_reg0    <= encoder_o;
//       error_counterQ    <= error_counter;
         if(error_counter[N-1:1]=='1) begin //
         //  err_inj        <= error_counter[29:28];
         //    encoder_o_reg  <= encoder_o^err_inj;
            encoder_o_reg <= {encoder_o[1], ~encoder_o[0]};    // inject bad bits
            err_inj <= 2'b01;
          end
          else begin              // clean version
            encoder_o_reg  <= encoder_o;
            err_inj        <= 2'b0;
       end
         if(word_ct<256) begin
          bad_bit_ct  <= bad_bit_ct + (encoder_o_reg0[1]^encoder_o_reg[1])
                                    + (encoder_o_reg0[0]^encoder_o_reg[0]);
         $display("error_counter,err_inj = %h %b %d %d",
                 error_counter,err_inj,bad_bit_ct,word_ct);
         end
       end


// insert your convolutional encoder here
// change port names and module name as necessary/desired
   encoder encoder1       (
      .clk,
      .rst,
      .enable_i(enable_encoder_i), //_reg),
      .d_in    (encoder_i),        //_reg),
      .valid_o (valid_encoder_o),
      .d_out   (encoder_o)   );

// insert your term project code here
   decoder decoder1       (
      .clk,
      .rst,
      .enable (enable_decoder_in),
      .d_in   (encoder_o_reg),
      .d_out  (decoder_o)   );

endmodule
```

```
VSIM 204> run
# error_counter,err_inj = 00000000 xx          0          0
# error_counter,err_inj = 00000000 00          0          1
# error_counter,err_inj = 00242a6a 00          0          2
# error_counter,err_inj = 00242a6a 00          0          3
# error_counter,err_inj = 018112bd 00          0          4
# error_counter,err_inj = 018112bd 00          0          5
# error_counter,err_inj = 010909ac 00          0          6
# error_counter,err_inj = 010909ac 00          0          7
# error_counter,err_inj = 0163e0ac 00          0          8
# error_counter,err_inj = 0163e0ac 00          0          9
# error_counter,err_inj = 000d72f6 00          0         10
# error_counter,err_inj = 000d72f6 00          0         11
# error_counter,err_inj = 008dbf33 00          0         12
# error_counter,err_inj = 008dbf33 00          0         13
# error_counter,err_inj = 01658508 00          0         14
# error_counter,err_inj = 01658508 00          0         15
# error_counter,err_inj = 01126ea4 00          0         16
# error_counter,err_inj = 01126ea4 00          0         17
# error_counter,err_inj = 0001e7c6 00          0         18
# error_counter,err_inj = 0001e7c6 00          0         19
# error_counter,err_inj = 000daf9a 00          0         20
# error_counter,err_inj = 000daf9a 00          0         21
# error_counter,err_inj = 007647e2 00          0         22
# error_counter,err_inj = 007647e2 00          0         23
# error_counter,err_inj = 003d1b9a 00          0         24
# error_counter,err_inj = 003d1b9a 00          0         25
# error_counter,err_inj = 00eda8af 00          0         26
# error_counter,err_inj = 00eda8af 01          0         27
# error_counter,err_inj = 008c5bef 01          1         28
# error_counter,err_inj = 008c5bef 01          2         29
# error_counter,err_inj = 00f9fbd3 01          3         30
# error_counter,err_inj = 00f9fbd3 00          4         31
# error_counter,err_inj = 01c66e49 00          4         32
# error_counter,err_inj = 01c66e49 00          4         33
# error_counter,err_inj = 01c5ef09 00          4         34
# error_counter,err_inj = 01c5ef09 00          4         35
# error_counter,err_inj = 01aa27a5 00          4         36
# error_counter,err_inj = 01aa27a5 00          4         37
# error_counter,err_inj = 00e55fef 00          4         38
# error_counter,err_inj = 00e55fef 01          4         39
# error_counter,err_inj = 0177a4e4 01          5         40
# error_counter,err_inj = 0177a4e4 00          6         41
# error_counter,err_inj = 011265ac 00          6         42

# good =         233, bad =          23,            32 bad_bits
# ** Note: $stop    : C:/Users/Charlotte_Qi/Desktop/final_project2/viterbi_tx_rx_tb2.sv(89)
#    Time: 1045 ns  Iteration: 0  Instance: /viterbi_tx_rx_tb
```

2.b.5

```
module viterbi_tx_rx #(parameter N=4) (
   input    clk,
   input    rst,
   input    encoder_i,
   input    enable_encoder_i,
   output   decoder_o);

   wire  [1:0] encoder_o;  // connects encoder to decoder

   int           error_counter,
                 error_counterQ,
                 bad_bit_ct,
                 word_ct;
   logic   [1:0] encoder_o_reg0,
                 encoder_o_reg;
   logic         encoder_i_reg;
   logic         enable_decoder_in;
   logic         enable_encoder_i_reg;
   wire          valid_encoder_o;
   logic   [1:0] err_inj;

   always @ (posedge clk, negedge rst)
      if(!rst) begin
        error_counter        <= 'd0;
//      error_counterQ       <= 'd0;
        encoder_o_reg        <= 'b0;
       encoder_o_reg0        <= 'b0;
        enable_decoder_in    <= 'b0;
       enable_encoder_i_reg <= 'b0;
       word_ct               <= 'b0;
      end
      else begin
        enable_encoder_i_reg <= enable_encoder_i;
        enable_decoder_in    <= valid_encoder_o;
//        encoder_o_reg        <= 'b0;
        if (word_ct[0] == '1) error_counter        <= $random >> 8;
        word_ct              <= word_ct + 1;
// bit error injection in encoder_o_reg
        encoder_i_reg     <= encoder_i;
        encoder_o_reg0    <= encoder_o;
//      error_counterQ    <= error_counter;
        if(error_counter[N-1:1]=='1) begin //
        //  err_inj        <= error_counter[29:28];
        //    encoder_o_reg  <= encoder_o^err_inj;
           encoder_o_reg <= {~encoder_o[1], encoder_o[0]};    // inject bad bits
           err_inj <= 2'b10;
        end
        else begin              // clean version
           encoder_o_reg  <= encoder_o;
           err_inj        <= 2'b0;
      end
        if(word_ct<256) begin
         bad_bit_ct  <= bad_bit_ct + (encoder_o_reg0[1]^encoder_o_reg[1])
                                   + (encoder_o_reg0[0]^encoder_o_reg[0]);
        $display("error_counter,err_inj = %h %b %d %d",
              error_counter,err_inj,bad_bit_ct,word_ct);
        end
      end


// insert your convolutional encoder here
// change port names and module name as necessary/desired
   encoder encoder1       (
      .clk,
      .rst,
      .enable_i(enable_encoder_i), //_reg),
      .d_in    (encoder_i),        //_reg),
      .valid_o (valid_encoder_o),
      .d_out   (encoder_o)   );

// insert your term project code here
   decoder decoder1       (
      .clk,
      .rst,
      .enable (enable_decoder_in),
      .d_in   (encoder_o_reg),
      .d_out  (decoder_o)   );

endmodule

```

```
VSIM 206> run
# error_counter,err_inj = 00000000 xx          0          0
# error_counter,err_inj = 00000000 00          0          1
# error_counter,err_inj = 00121535 00          0          2
# error_counter,err_inj = 00121535 00          0          3
# error_counter,err_inj = 00c0895e 00          0          4
# error_counter,err_inj = 00c0895e 10          0          5
# error_counter,err_inj = 008484d6 10          1          6
# error_counter,err_inj = 008484d6 00          2          7
# error_counter,err_inj = 00b1f056 00          2          8
# error_counter,err_inj = 00b1f056 00          2          9
# error_counter,err_inj = 0006b97b 00          2         10
# error_counter,err_inj = 0006b97b 00          2         11
# error_counter,err_inj = 0046df99 00          2         12
# error_counter,err_inj = 0046df99 00          2         13
# error_counter,err_inj = 00b2c284 00          2         14
# error_counter,err_inj = 00b2c284 00          2         15
# error_counter,err_inj = 00893752 00          2         16
# error_counter,err_inj = 00893752 00          2         17
# error_counter,err_inj = 0000f3e3 00          2         18
# error_counter,err_inj = 0000f3e3 00          2         19
# error_counter,err_inj = 0006d7cd 00          2         20
# error_counter,err_inj = 0006d7cd 00          2         21
# error_counter,err_inj = 003b23f1 00          2         22
# error_counter,err_inj = 003b23f1 00          2         23
# error_counter,err_inj = 001e8dcd 00          2         24
# error_counter,err_inj = 001e8dcd 00          2         25
# error_counter,err_inj = 0076d457 00          2         26
# error_counter,err_inj = 0076d457 00          2         27
# error_counter,err_inj = 00462df7 00          2         28
# error_counter,err_inj = 00462df7 00          2         29
# error_counter,err_inj = 007cfde9 00          2         30
# error_counter,err_inj = 007cfde9 00          2         31
# error_counter,err_inj = 00e33724 00          2         32
```

```
# good =          248, bad =            8,             30 bad_bits
# ** Note: $stop    : C:/Users/Charlotte_Qi/Desktop/final_project2/viterbi_tx_rx_tb2.sv(89)
#    Time: 1045 ns  Iteration: 0  Instance: /viterbi_tx_rx_tb
```

2.b.6

```
module viterbi_tx_rx #(parameter N=5) (
   input    clk,
   input    rst,
   input    encoder_i,
   input    enable_encoder_i,
   output   decoder_o);

   wire  [1:0] encoder_o;  // connects encoder to decoder

   int           error_counter,
                 error_counterQ,
                 bad_bit_ct,
                 word_ct;
   logic   [1:0] encoder_o_reg0,
                 encoder_o_reg;
   logic         encoder_i_reg;
   logic         enable_decoder_in;
   logic         enable_encoder_i_reg;
   wire          valid_encoder_o;
   logic   [1:0] err_inj;

   always @ (posedge clk, negedge rst)
      if(!rst) begin
         error_counter        <= 'd0;
//     error_counterQ       <= 'd0;
         encoder_o_reg        <= 'b0;
        encoder_o_reg0       <= 'b0;
         enable_decoder_in    <= 'b0;
        enable_encoder_i_reg <= 'b0;
        word_ct              <= 'b0;
      end
      else begin
         enable_encoder_i_reg <= enable_encoder_i;
         enable_decoder_in    <= valid_encoder_o;
//         encoder_o_reg        <= 'b0;
         if(word_ct[1:0] == '1) error_counter        <= $random >> 9;
         word_ct              <= word_ct + 1;
// bit error injection in encoder_o_reg
         encoder_i_reg     <= encoder_i;
         encoder_o_reg0    <= encoder_o;
//      error_counterQ    <= error_counter;
         if(error_counter[N-1:2]=='1) begin //
         //  err_inj        <= error_counter[29:28];
         //    encoder_o_reg  <= encoder_o^err_inj;
            encoder_o_reg <= {encoder_o[1], ~encoder_o[0]};    // inject bad bits
            err_inj <= 2'b01;
         end
         else begin              // clean version
            encoder_o_reg  <= encoder_o;
            err_inj        <= 2'b0;
      end
        if(word_ct<256) begin
          bad_bit_ct  <= bad_bit_ct + (encoder_o_reg0[1]^encoder_o_reg[1])
                                    + (encoder_o_reg0[0]^encoder_o_reg[0]);
         $display("error_counter,err_inj = %h %b %d %d",
                error_counter,err_inj,bad_bit_ct,word_ct);
        end
      end


// insert your convolutional encoder here
// change port names and module name as necessary/desired
   encoder encoder1       (
      .clk,
      .rst,
      .enable_i(enable_encoder_i), //_reg),
      .d_in    (encoder_i),        //_reg),
      .valid_o (valid_encoder_o),
      .d_out   (encoder_o)   );

// insert your term project code here
   decoder decoder1       (
      .clk,
      .rst,
      .enable (enable_decoder_in),
      .d_in   (encoder_o_reg),
      .d_out  (decoder_o)   );

endmodule

```

```
VSIM 208> run
# error_counter,err_inj = 00000000 xx          0          0
# error_counter,err_inj = 00000000 00          0          1
# error_counter,err_inj = 00000000 00          0          2
# error_counter,err_inj = 00000000 00          0          3
# error_counter,err_inj = 00090a9a 00          0          4
# error_counter,err_inj = 00090a9a 00          0          5
# error_counter,err_inj = 00090a9a 00          0          6
# error_counter,err_inj = 00090a9a 00          0          7
# error_counter,err_inj = 006044af 00          0          8
# error_counter,err_inj = 006044af 00          0          9
# error_counter,err_inj = 006044af 00          0         10
# error_counter,err_inj = 006044af 00          0         11
# error_counter,err_inj = 0042426b 00          0         12
# error_counter,err_inj = 0042426b 00          0         13
# error_counter,err_inj = 0042426b 00          0         14
# error_counter,err_inj = 0042426b 00          0         15
# error_counter,err_inj = 0058f82b 00          0         16
# error_counter,err_inj = 0058f82b 00          0         17
# error_counter,err_inj = 0058f82b 00          0         18
# error_counter,err_inj = 0058f82b 00          0         19
# error_counter,err_inj = 00035cbd 00          0         20
# error_counter,err_inj = 00035cbd 01          0         21
# error_counter,err_inj = 00035cbd 01          1         22
# error_counter,err_inj = 00035cbd 01          2         23
# error_counter,err_inj = 00236fcc 01          3         24
# error_counter,err_inj = 00236fcc 00          4         25
# error_counter,err_inj = 00236fcc 00          4         26
# error_counter,err_inj = 00236fcc 00          4         27
# error_counter,err_inj = 00596142 00          4         28
# error_counter,err_inj = 00596142 00          4         29
# error_counter,err_inj = 00596142 00          4         30
# error_counter,err_inj = 00596142 00          4         31
# error_counter,err_inj = 00449ba9 00          4         32

# good =         232, bad =          24,             24 bad_bits
# ** Note: $stop    : C:/Users/Charlotte_Qi/Desktop/final_project2/viterbi_tx_rx_tb2.sv(89)
#    Time: 1045 ns  Iteration: 0  Instance: /viterbi_tx_rx_tb
```

2.b.7

```
module viterbi_tx_rx #(parameter N=5) (
   input    clk,
   input    rst,
   input    encoder_i,
   input    enable_encoder_i,
   output   decoder_o);

   wire  [1:0] encoder_o;  // connects encoder to decoder

   int           error_counter,
                 error_counterQ,
                 bad_bit_ct,
                 word_ct;
   logic   [1:0] encoder_o_reg0,
                 encoder_o_reg;
   logic         encoder_i_reg;
   logic         enable_decoder_in;
   logic         enable_encoder_i_reg;
   wire          valid_encoder_o;
   logic   [1:0] err_inj;

   always @ (posedge clk, negedge rst)
      if(!rst) begin
        error_counter        <= 'd0;
//     error_counterQ       <= 'd0;
        encoder_o_reg        <= 'b0;
       encoder_o_reg0        <= 'b0;
        enable_decoder_in    <= 'b0;
       enable_encoder_i_reg <= 'b0;
       word_ct               <= 'b0;
      end
      else begin
        enable_encoder_i_reg <= enable_encoder_i;
        enable_decoder_in    <= valid_encoder_o;
//        encoder_o_reg        <= 'b0;
        if(word_ct[1:0] == '1) error_counter        <= $random >> 10;
        word_ct              <= word_ct + 1;
// bit error injection in encoder_o_reg
        encoder_i_reg     <= encoder_i;
        encoder_o_reg0    <= encoder_o;
//      error_counterQ    <= error_counter;
        if(error_counter[N-1:2]=='1) begin //
        //  err_inj        <= error_counter[29:28];
        //    encoder_o_reg  <= encoder_o^err_inj;
           encoder_o_reg <= {~encoder_o[1], encoder_o[0]};    // inject bad bits
           err_inj <= 2'b10;
        end
        else begin              // clean version
           encoder_o_reg  <= encoder_o;
           err_inj        <= 2'b0;
      end
        if(word_ct<256) begin
          bad_bit_ct  <= bad_bit_ct + (encoder_o_reg0[1]^encoder_o_reg[1])
                                    + (encoder_o_reg0[0]^encoder_o_reg[0]);
        $display("error_counter,err_inj = %h %b %d %d",
              error_counter,err_inj,bad_bit_ct,word_ct);
        end
      end


// insert your convolutional encoder here
// change port names and module name as necessary/desired
   encoder encoder1       (
      .clk,
      .rst,
      .enable_i(enable_encoder_i), //_reg),
      .d_in    (encoder_i),        //_reg),
      .valid_o (valid_encoder_o),
      .d_out   (encoder_o)   );

// insert your term project code here
   decoder decoder1       (
      .clk,
      .rst,
      .enable (enable_decoder_in),
      .d_in   (encoder_o_reg),
      .d_out  (decoder_o)   );

endmodule

```

```
VSIM 210> run
# error_counter,err_inj = 00000000 xx          0          0
# error_counter,err_inj = 00000000 00          0          1
# error_counter,err_inj = 00000000 00          0          2
# error_counter,err_inj = 00000000 00          0          3
# error_counter,err_inj = 0004854d 00          0          4
# error_counter,err_inj = 0004854d 00          0          5
# error_counter,err_inj = 0004854d 00          0          6
# error_counter,err_inj = 0004854d 00          0          7
# error_counter,err_inj = 00302257 00          0          8
# error_counter,err_inj = 00302257 00          0          9
# error_counter,err_inj = 00302257 00          0         10
# error_counter,err_inj = 00302257 00          0         11
# error_counter,err_inj = 00212135 00          0         12
# error_counter,err_inj = 00212135 00          0         13
# error_counter,err_inj = 00212135 00          0         14
# error_counter,err_inj = 00212135 00          0         15
# error_counter,err_inj = 002c7c15 00          0         16
# error_counter,err_inj = 002c7c15 00          0         17
# error_counter,err_inj = 002c7c15 00          0         18
# error_counter,err_inj = 002c7c15 00          0         19
# error_counter,err_inj = 0001ae5e 00          0         20
# error_counter,err_inj = 0001ae5e 10          0         21
# error_counter,err_inj = 0001ae5e 10          1         22
# error_counter,err_inj = 0001ae5e 10          2         23
# error_counter,err_inj = 0011b7e6 10          3         24
# error_counter,err_inj = 0011b7e6 00          4         25
# error_counter,err_inj = 0011b7e6 00          4         26
# error_counter,err_inj = 0011b7e6 00          4         27
# error_counter,err_inj = 002cb0a1 00          4         28
# error_counter,err_inj = 002cb0a1 00          4         29
# error_counter,err_inj = 002cb0a1 00          4         30
# error_counter,err_inj = 002cb0a1 00          4         31
# error_counter,err_inj = 00224dd4 00          4         32
```

```
# good =         240, bad =          16,            32 bad_bits
# ** Note: $stop    : C:/Users/Charlotte_Qi/Desktop/final_project2/viterbi_tx_rx_tb2.sv(89)
#    Time: 1045 ns  Iteration: 0  Instance: /viterbi_tx_rx_tb
```

2.b.8

```
module viterbi_tx_rx #(parameter N=5) (
   input    clk,
   input    rst,
   input    encoder_i,
   input    enable_encoder_i,
   output   decoder_o);

   wire  [1:0] encoder_o;  // connects encoder to decoder

   int           error_counter,
                 error_counterQ,
                 bad_bit_ct,
                 word_ct;
   logic   [1:0] encoder_o_reg0,
                 encoder_o_reg;
   logic         encoder_i_reg;
   logic         enable_decoder_in;
   logic         enable_encoder_i_reg;
   wire          valid_encoder_o;
   logic   [1:0] err_inj;

   always @ (posedge clk, negedge rst)
      if(!rst) begin
         error_counter        <= 'd0;
//       error_counterQ       <= 'd0;
         encoder_o_reg        <= 'b0;
        encoder_o_reg0        <= 'b0;
         enable_decoder_in    <= 'b0;
        enable_encoder_i_reg <= 'b0;
        word_ct               <= 'b0;
      end
      else begin
         enable_encoder_i_reg <= enable_encoder_i;
         enable_decoder_in    <= valid_encoder_o;
//          encoder_o_reg        <= 'b0;
         if(word_ct[0] == '1) error_counter        <= $random >> 11;
         word_ct              <= word_ct + 1;
// bit error injection in encoder_o_reg
         encoder_i_reg     <= encoder_i;
         encoder_o_reg0    <= encoder_o;
//       error_counterQ    <= error_counter;
         if(error_counter[N-1:1]=='1) begin //
         //  err_inj        <= error_counter[29:28];
         //    encoder_o_reg  <= encoder_o^err_inj;
            encoder_o_reg <= {~encoder_o[1], ~encoder_o[0]};    // inject bad bits
            err_inj <= 2'b11;
         end
         else begin              // clean version
            encoder_o_reg  <= encoder_o;
            err_inj        <= 2'b0;
      end
        if(word_ct<256) begin
          bad_bit_ct  <= bad_bit_ct + (encoder_o_reg0[1]^encoder_o_reg[1])
                                    + (encoder_o_reg0[0]^encoder_o_reg[0]);
         $display("error_counter,err_inj = %h %b %d %d",
                  error_counter,err_inj,bad_bit_ct,word_ct);
        end
      end


// insert your convolutional encoder here
// change port names and module name as necessary/desired
   encoder encoder1       (
      .clk,
      .rst,
      .enable_i(enable_encoder_i), //_reg),
      .d_in    (encoder_i),        //_reg),
      .valid_o (valid_encoder_o),
      .d_out   (encoder_o)   );

// insert your term project code here
   decoder decoder1       (
      .clk,
      .rst,
      .enable (enable_decoder_in),
      .d_in   (encoder_o_reg),
      .d_out  (decoder_o)   );

endmodule
```

```
VSIM 212> run
# error_counter,err_inj = 00000000 xx          0          0
# error_counter,err_inj = 00000000 00          0          1
# error_counter,err_inj = 000242a6 00          0          2
# error_counter,err_inj = 000242a6 00          0          3
# error_counter,err_inj = 0018112b 00          0          4
# error_counter,err_inj = 0018112b 00          0          5
# error_counter,err_inj = 0010909a 00          0          6
# error_counter,err_inj = 0010909a 00          0          7
# error_counter,err_inj = 00163e0a 00          0          8
# error_counter,err_inj = 00163e0a 00          0          9
# error_counter,err_inj = 0000d72f 00          0         10
# error_counter,err_inj = 0000d72f 00          0         11
# error_counter,err_inj = 0008dbf3 00          0         12
# error_counter,err_inj = 0008dbf3 00          0         13
# error_counter,err_inj = 00165850 00          0         14
# error_counter,err_inj = 00165850 00          0         15
# error_counter,err_inj = 001126ea 00          0         16
# error_counter,err_inj = 001126ea 00          0         17
# error_counter,err_inj = 00001e7c 00          0         18
# error_counter,err_inj = 00001e7c 00          0         19
# error_counter,err_inj = 0000daf9 00          0         20
# error_counter,err_inj = 0000daf9 00          0         21
# error_counter,err_inj = 0007647e 00          0         22
# error_counter,err_inj = 0007647e 11          0         23
# error_counter,err_inj = 0003d1b9 11          2         24
# error_counter,err_inj = 0003d1b9 00          4         25
# error_counter,err_inj = 000eda8a 00          4         26
# error_counter,err_inj = 000eda8a 00          4         27
# error_counter,err_inj = 0008c5be 00          4         28
# error_counter,err_inj = 0008c5be 11          4         29
# error_counter,err_inj = 000f9fbd 11          6         30
# error_counter,err_inj = 000f9fbd 00          8         31
# error_counter,err_inj = 001c66e4 00          8         32
```

```
# good =         227, bad =          29,            40 bad_bits
# ** Note: $stop    : C:/Users/Charlotte_Qi/Desktop/final_project2/viterbi_tx_rx_tb2.sv(89)
#    Time: 1045 ns  Iteration: 0  Instance: /viterbi_tx_rx_tb
```

2.c

We found that a string of 3 consecutive bad bit[0]s is needed to produce output errors.

1. With a string of 1 consecutive bad bit[0]s, no output errors are produced.

```
// contains convolutional encoder,
//  (possibly corrupting) channel,
//  and Viterbi decoder
// parameter N sets the channel bit error rate
// this time, two errors in a row
`include "encoder.sv"
`include "decoder.sv"
module viterbi_tx_rx #(parameter N=3) (
   input    clk,
   input    rst,
   input    encoder_i,
   input    enable_encoder_i,
   output   decoder_o);

   wire  [1:0] encoder_o;  // connects encoder to decoder

   int           error_counter,
                 bad_bit_ct,
                 word_ct;
   logic   [1:0] encoder_o_reg0,
                 encoder_o_reg;
   logic         encoder_i_reg;
   logic         enable_decoder_in;
   logic         enable_encoder_i_reg;
   wire          valid_encoder_o;
   logic   [1:0] err_inj;

   always @ (posedge clk, negedge rst)
      if(!rst) begin
         error_counter        <= 'd0;
         encoder_o_reg        <= 'b0;
         encoder_o_reg0       <= 'b0;
         enable_decoder_in    <= 'b0;
         enable_encoder_i_reg <= 'b0;
         word_ct              <= 'b0;
      end
      else begin
         enable_encoder_i_reg <= enable_encoder_i;
         enable_decoder_in    <= valid_encoder_o;
         error_counter        <= error_counter + 1;
         word_ct              <= word_ct + 1;
// bit error injection in encoder_o_reg
         encoder_i_reg     <= encoder_i;
         encoder_o_reg0    <= encoder_o;
//       error_counterQ    <= error_counter;
        if(error_counter[N-1:0]== 3'b001) begin //
          //  err_inj        <= error_counter[29:28];
          //    encoder_o_reg  <= encoder_o^err_inj;
            encoder_o_reg <= {encoder_o[1], ~encoder_o[0]};    // inject bad bits
            err_inj <= 2'b01;
         end
         else begin                // clean version
            encoder_o_reg  <= encoder_o;
            err_inj        <= 2'b0;
        end
        if(word_ct<256) begin
          bad_bit_ct  <= bad_bit_ct + (encoder_o_reg0[1]^encoder_o_reg[1])
                                    + (encoder_o_reg0[0]^encoder_o_reg[0]);
          $display("error_counter,err_inj = %h %b %d %d",
                 error_counter,err_inj,bad_bit_ct,word_ct);
        end
      end


// insert your convolutional encoder here
// change port names and module name as necessary/desired
   encoder encoder1      (
      .clk,
      .rst,
      .enable_i(enable_encoder_i), //_reg),
      .d_in    (encoder_i),        //_reg),
      .valid_o (valid_encoder_o),
      .d_out   (encoder_o)   );

// insert your term project code here
   decoder decoder1      (
      .clk,
      .rst,
      .enable (enable_decoder_in),
      .d_in   (encoder_o_reg),
      .d_out  (decoder_o)   );

endmodule
```

```
# run -all
# error_counter,err_inj = 00000000 xx          0          0
# error_counter,err_inj = 00000001 00          0          1
# error_counter,err_inj = 00000002 01          0          2
# error_counter,err_inj = 00000003 00          1          3
# error_counter,err_inj = 00000004 00          1          4
# error_counter,err_inj = 00000005 00          1          5
# error_counter,err_inj = 00000006 00          1          6
# error_counter,err_inj = 00000007 00          1          7
# error_counter,err_inj = 00000008 00          1          8
# error_counter,err_inj = 00000009 00          1          9
# error_counter,err_inj = 0000000a 01          1         10
# error_counter,err_inj = 0000000b 00          2         11
# error_counter,err_inj = 0000000c 00          2         12
# error_counter,err_inj = 0000000d 00          2         13
# error_counter,err_inj = 0000000e 00          2         14
# error_counter,err_inj = 0000000f 00          2         15
# error_counter,err_inj = 00000010 00          2         16
# error_counter,err_inj = 00000011 00          2         17
# error_counter,err_inj = 00000012 01          2         18
# error_counter,err_inj = 00000013 00          3         19
```

Skipping the repetition.

```
# error_counter,err_inj = 000000f1 00          30         241
# error_counter,err_inj = 000000f2 01          30         242
# error_counter,err_inj = 000000f3 00          31         243
# error_counter,err_inj = 000000f4 00          31         244
# error_counter,err_inj = 000000f5 00          31         245
# error_counter,err_inj = 000000f6 00          31         246
# error_counter,err_inj = 000000f7 00          31         247
# error_counter,err_inj = 000000f8 00          31         248
# error_counter,err_inj = 000000f9 00          31         249
# error_counter,err_inj = 000000fa 01          31         250
# error_counter,err_inj = 000000fb 00          32         251
# error_counter,err_inj = 000000fc 00          32         252
# error_counter,err_inj = 000000fd 00          32         253
# error_counter,err_inj = 000000fe 00          32         254
# error_counter,err_inj = 000000ff 00          32         255
# word_count =       10440
# yaa! in = 0, out = 0, w_ct =           0, err = 00
# yaa! in = 0, out = 0, w_ct =           1, err = 00
# yaa! in = 0, out = 0, w_ct =           2, err = 00
# yaa! in = 0, out = 0, w_ct =           3, err = 00
# yaa! in = 0, out = 0, w_ct =           4, err = 00
# yaa! in = 0, out = 0, w_ct =           5, err = 00
# yaa! in = 0, out = 0, w_ct =           6, err = 00
# yaa! in = 0, out = 0, w_ct =           7, err = 00
```

Skipping the repetition.

```
# yaa! in = 1, out = 1, w_ct =         235, err = 00
# yaa! in = 1, out = 1, w_ct =         236, err = 00
# yaa! in = 1, out = 1, w_ct =         237, err = 00
# yaa! in = 1, out = 1, w_ct =         238, err = 00
# yaa! in = 1, out = 1, w_ct =         239, err = 00
# yaa! in = 1, out = 1, w_ct =         240, err = 00
# yaa! in = 1, out = 1, w_ct =         241, err = 00
# yaa! in = 1, out = 1, w_ct =         242, err = 00
# yaa! in = 1, out = 1, w_ct =         243, err = 00
# yaa! in = 1, out = 1, w_ct =         244, err = 00
# yaa! in = 1, out = 1, w_ct =         245, err = 00
# yaa! in = 1, out = 1, w_ct =         246, err = 00
# yaa! in = 1, out = 1, w_ct =         247, err = 00
# yaa! in = 0, out = 0, w_ct =         248, err = 00
# yaa! in = 1, out = 1, w_ct =         249, err = 00
# yaa! in = 1, out = 1, w_ct =         250, err = 00
# yaa! in = 1, out = 1, w_ct =         251, err = 00
# yaa! in = 1, out = 1, w_ct =         252, err = 00
# yaa! in = 1, out = 1, w_ct =         253, err = 00
# yaa! in = 1, out = 1, w_ct =         254, err = 00
# yaa! in = 1, out = 1, w_ct =         255, err = 00
# good =         256, bad =           0,            32 bad_bits
# ** Note: $stop    : viterbi_tx_rx_tb.sv(89)
#    Time: 1045 us  Iteration: 0  Instance: /viterbi_tx_rx_tb
```

2. With a string of 2 consecutive bad bit[0]s, no output errors are produced.

```
`include "encoder.sv"
`include "decoder.sv"
module viterbi_tx_rx #(parameter N=3) (
   input    clk,
   input    rst,
   input    encoder_i,
   input    enable_encoder_i,
   output   decoder_o);

   wire  [1:0] encoder_o;  // connects encoder to decoder

   int           error_counter,
                 bad_bit_ct,
                 word_ct;
   logic   [1:0] encoder_o_reg0,
                 encoder_o_reg;
   logic         encoder_i_reg;
   logic         enable_decoder_in;
   logic         enable_encoder_i_reg;
   wire          valid_encoder_o;
   logic   [1:0] err_inj;

   always @ (posedge clk, negedge rst)
      if(!rst) begin
         error_counter        <= 'd0;
         encoder_o_reg        <= 'b0;
         encoder_o_reg0       <= 'b0;
         enable_decoder_in    <= 'b0;
         enable_encoder_i_reg <= 'b0;
         word_ct              <= 'b0;
      end
      else begin
         enable_encoder_i_reg <= enable_encoder_i;
         enable_decoder_in    <= valid_encoder_o;
         error_counter        <= error_counter + 1;
         word_ct              <= word_ct + 1;
// bit error injection in encoder_o_reg
         encoder_i_reg     <= encoder_i;
         encoder_o_reg0    <= encoder_o;
//       error_counterQ    <= error_counter;
        if(error_counter[N-1:0]== 3'b001
           ||error_counter[N-1:0]== 3'b010) begin //
          //  err_inj        <= error_counter[29:28];
        //    encoder_o_reg  <= encoder_o^err_inj;
           encoder_o_reg <= {encoder_o[1], ~encoder_o[0]};    // inject b
           err_inj <= 2'b01;
        end
        else begin                // clean version
           encoder_o_reg  <= encoder_o;
           err_inj        <= 2'b0;
        end
        if(word_ct<256) begin
          bad_bit_ct  <= bad_bit_ct + (encoder_o_reg0[1]^encoder_o_reg[1])
                                    + (encoder_o_reg0[0]^encoder_o_reg[0]);
          $display("error_counter,err_inj = %h %b %d %d",
                error_counter,err_inj,bad_bit_ct,word_ct);
        end
      end


// insert your convolutional encoder here
// change port names and module name as necessary/desired
   encoder encoder1      (
      .clk,
      .rst,
      .enable_i(enable_encoder_i), //_reg),
      .d_in    (encoder_i),        //_reg),
      .valid_o (valid_encoder_o),
      .d_out   (encoder_o)    );

// insert your term project code here
   decoder decoder1      (
      .clk,
      .rst,
      .enable (enable_decoder_in),
      .d_in   (encoder_o_reg),
      .d_out  (decoder_o)    );

endmodule
```

```
# run -all
# error_counter,err_inj = 00000000 xx          0          0
# error_counter,err_inj = 00000001 00          0          1
# error_counter,err_inj = 00000002 01          0          2
# error_counter,err_inj = 00000003 01          1          3
# error_counter,err_inj = 00000004 00          2          4
# error_counter,err_inj = 00000005 00          2          5
# error_counter,err_inj = 00000006 00          2          6
# error_counter,err_inj = 00000007 00          2          7
# error_counter,err_inj = 00000008 00          2          8
# error_counter,err_inj = 00000009 00          2          9
# error_counter,err_inj = 0000000a 01          2         10
# error_counter,err_inj = 0000000b 01          3         11
# error_counter,err_inj = 0000000c 00          4         12
# error_counter,err_inj = 0000000d 00          4         13
# error_counter,err_inj = 0000000e 00          4         14
# error_counter,err_inj = 0000000f 00          4         15
# error_counter,err_inj = 00000010 00          4         16
# error_counter,err_inj = 00000011 00          4         17
# error_counter,err_inj = 00000012 01          4         18
# error_counter,err_inj = 00000013 01          5         19
# error_counter,err_inj = 00000014 00          6         20
```

Skipping the repetition.

```
# error_counter,err_inj = 000000f2 01          60         242
# error_counter,err_inj = 000000f3 01          61         243
# error_counter,err_inj = 000000f4 00          62         244
# error_counter,err_inj = 000000f5 00          62         245
# error_counter,err_inj = 000000f6 00          62         246
# error_counter,err_inj = 000000f7 00          62         247
# error_counter,err_inj = 000000f8 00          62         248
# error_counter,err_inj = 000000f9 00          62         249
# error_counter,err_inj = 000000fa 01          62         250
# error_counter,err_inj = 000000fb 01          63         251
# error_counter,err_inj = 000000fc 00          64         252
# error_counter,err_inj = 000000fd 00          64         253
# error_counter,err_inj = 000000fe 00          64         254
# error_counter,err_inj = 000000ff 00          64         255
# word_count =       10440
# yaa! in = 0, out = 0, w_ct =           0, err = 00
# yaa! in = 0, out = 0, w_ct =           1, err = 00
# yaa! in = 0, out = 0, w_ct =           2, err = 00
# yaa! in = 0, out = 0, w_ct =           3, err = 00
# yaa! in = 0, out = 0, w_ct =           4, err = 00
# yaa! in = 0, out = 0, w_ct =           5, err = 00
# yaa! in = 0, out = 0, w_ct =           6, err = 00
# yaa! in = 0, out = 0, w_ct =           7, err = 00
# yaa! in = 0, out = 0, w_ct =           8, err = 00
```

Skipping the repetition.

```
# yaa! in = 1, out = 1, w_ct =         234, err = 00
# yaa! in = 1, out = 1, w_ct =         235, err = 00
# yaa! in = 1, out = 1, w_ct =         236, err = 00
# yaa! in = 1, out = 1, w_ct =         237, err = 00
# yaa! in = 1, out = 1, w_ct =         238, err = 00
# yaa! in = 1, out = 1, w_ct =         239, err = 00
# yaa! in = 1, out = 1, w_ct =         240, err = 00
# yaa! in = 1, out = 1, w_ct =         241, err = 00
# yaa! in = 1, out = 1, w_ct =         242, err = 00
# yaa! in = 1, out = 1, w_ct =         243, err = 00
# yaa! in = 1, out = 1, w_ct =         244, err = 00
# yaa! in = 1, out = 1, w_ct =         245, err = 00
# yaa! in = 1, out = 1, w_ct =         246, err = 00
# yaa! in = 1, out = 1, w_ct =         247, err = 00
# yaa! in = 0, out = 0, w_ct =         248, err = 00
# yaa! in = 1, out = 1, w_ct =         249, err = 00
# yaa! in = 1, out = 1, w_ct =         250, err = 00
# yaa! in = 1, out = 1, w_ct =         251, err = 00
# yaa! in = 1, out = 1, w_ct =         252, err = 00
# yaa! in = 1, out = 1, w_ct =         253, err = 00
# yaa! in = 1, out = 1, w_ct =         254, err = 00
# yaa! in = 1, out = 1, w_ct =         255, err = 00
# good =         256, bad =           0,            64 bad_bits
# ** Note: $stop    : viterbi_tx_rx_tb.sv(89)
```

3. With a string of 1 consecutive bad bit[0]s, output errors are produced.

```
module viterbi_tx_rx #(parameter N=3) (
   input    clk,
   input    rst,
   input    encoder_i,
   input    enable_encoder_i,
   output   decoder_o);

   wire  [1:0] encoder_o;  // connects encoder to decoder

   int           error_counter,
                 bad_bit_ct,
                 word_ct;
   logic   [1:0] encoder_o_reg0,
                 encoder_o_reg;
   logic         encoder_i_reg;
   logic         enable_decoder_in;
   logic         enable_encoder_i_reg;
   wire          valid_encoder_o;
   logic   [1:0] err_inj;

   always @ (posedge clk, negedge rst)
      if(!rst) begin
         error_counter        <= 'd0;
         encoder_o_reg        <= 'b0;
         encoder_o_reg0       <= 'b0;
         enable_decoder_in    <= 'b0;
         enable_encoder_i_reg <= 'b0;
         word_ct              <= 'b0;
      end
      else begin
         enable_encoder_i_reg <= enable_encoder_i;
         enable_decoder_in    <= valid_encoder_o;
         error_counter        <= error_counter + 1;
         word_ct              <= word_ct + 1;
```

```
// bit error injection in encoder_o_reg
         encoder_i_reg     <= encoder_i;
         encoder_o_reg0    <= encoder_o;
//       error_counterQ    <= error_counter;
        if(error_counter[N-1:0]== 3'b001
          ||error_counter[N-1:0]== 3'b010
          ||error_counter[N-1:0]== 3'b011) begin //
          //  err_inj        <= error_counter[29:28];
         //    encoder_o_reg  <= encoder_o^err_inj;
           encoder_o_reg <= {encoder_o[1], ~encoder_o[0]};    // inject ba
           err_inj <= 2'b01;
        end
        else begin                // clean version
           encoder_o_reg  <= encoder_o;
           err_inj        <= 2'b0;
        end
        if(word_ct<256) begin
          bad_bit_ct  <= bad_bit_ct + (encoder_o_reg0[1]^encoder_o_reg[1])
                                    + (encoder_o_reg0[0]^encoder_o_reg[0]);
          $display("error_counter,err_inj = %h %b %d %d",
                error_counter,err_inj,bad_bit_ct,word_ct);
        end
      end


// insert your convolutional encoder here
// change port names and module name as necessary/desired
   encoder encoder1      (
      .clk,
      .rst,
      .enable_i(enable_encoder_i), //_reg),
      .d_in    (encoder_i),        //_reg),
      .valid_o (valid_encoder_o),
      .d_out   (encoder_o)   );
```

```

// insert your term project code here
   decoder decoder1      (
      .clk,
      .rst,
      .enable (enable_decoder_in),
      .d_in   (encoder_o_reg),
      .d_out  (decoder_o)   );

endmodule
```

```
# run -all
# error_counter,err_inj = 00000000 xx          0          0
# error_counter,err_inj = 00000001 00          0          1
# error_counter,err_inj = 00000002 01          0          2
# error_counter,err_inj = 00000003 01          1          3
# error_counter,err_inj = 00000004 01          2          4
# error_counter,err_inj = 00000005 00          3          5
# error_counter,err_inj = 00000006 00          3          6
# error_counter,err_inj = 00000007 00          3          7
# error_counter,err_inj = 00000008 00          3          8
# error_counter,err_inj = 00000009 00          3          9
# error_counter,err_inj = 0000000a 01          3         10
# error_counter,err_inj = 0000000b 01          4         11
# error_counter,err_inj = 0000000c 01          5         12
# error_counter,err_inj = 0000000d 00          6         13
# error_counter,err_inj = 0000000e 00          6         14
# error_counter,err_inj = 0000000f 00          6         15
# error_counter,err_inj = 00000010 00          6         16
# error_counter,err_inj = 00000011 00          6         17
# error_counter,err_inj = 00000012 01          6         18
# error_counter,err_inj = 00000013 01          7         19
# error_counter,err_inj = 00000014 01          8         20
```

Skipping the repetition.

```
# error_counter,err_inj = 000000f8 00          93         248
# error_counter,err_inj = 000000f9 00          93         249
# error_counter,err_inj = 000000fa 01          93         250
# error_counter,err_inj = 000000fb 01          94         251
# error_counter,err_inj = 000000fc 01          95         252
# error_counter,err_inj = 000000fd 00          96         253
# error_counter,err_inj = 000000fe 00          96         254
# error_counter,err_inj = 000000ff 00          96         255
# word_count =       10440
# yaa! in = 0, out = 0, w_ct =           0, err = 00
# yaa! in = 0, out = 0, w_ct =           1, err = 00
# yaa! in = 0, out = 0, w_ct =           2, err = 00
# yaa! in = 0, out = 0, w_ct =           3, err = 00
# yaa! in = 0, out = 0, w_ct =           4, err = 00
# yaa! in = 0, out = 0, w_ct =           5, err = 00
# yaa! in = 0, out = 0, w_ct =           6, err = 00
# yaa! in = 0, out = 0, w_ct =           7, err = 00
# yaa! in = 0, out = 0, w_ct =           8, err = 00
# yaa! in = 0, out = 0, w_ct =           9, err = 00
# boo! in = 0, out = 1, w_ct =          10, err = 00,              1045000, BAD!
# boo! in = 1, out = 0, w_ct =          11, err = 00,              1045000, BAD!
# boo! in = 0, out = 1, w_ct =          12, err = 00,              1045000, BAD!
```

Skipping the repetition.

```
# boo! in = 1, out = 0, w_ct =         242, err = 00,              1045000, BAD!
# boo! in = 1, out = 0, w_ct =         243, err = 00,              1045000, BAD!
# boo! in = 1, out = 0, w_ct =         244, err = 00,              1045000, BAD!
# yaa! in = 1, out = 1, w_ct =         245, err = 00
# yaa! in = 1, out = 1, w_ct =         246, err = 00
# yaa! in = 1, out = 1, w_ct =         247, err = 00
# yaa! in = 0, out = 0, w_ct =         248, err = 00
# yaa! in = 1, out = 1, w_ct =         249, err = 00
# boo! in = 1, out = 0, w_ct =         250, err = 00,              1045000, BAD!
# boo! in = 1, out = 0, w_ct =         251, err = 00,              1045000, BAD!
# boo! in = 1, out = 0, w_ct =         252, err = 00,              1045000, BAD!
# yaa! in = 1, out = 1, w_ct =         253, err = 00
# yaa! in = 1, out = 1, w_ct =         254, err = 00
# boo! in = 1, out = 0, w_ct =         255, err = 00,              1045000, BAD!
# good =         135, bad =         121,            96 bad_bits
# ** Note: $stop    : viterbi_tx_rx_tb.sv(89)
#    Time: 1045 us  Iteration: 0  Instance: /viterbi_tx_rx_tb
```

2.d

We found that a string of 3 consecutive bad bit[1]s is needed to produce output errors.

1. With a string of 1 consecutive bad bit[1]s, no output errors are produced.

```
module viterbi_tx_rx #(parameter N=3) (
   input    clk,
   input    rst,
   input    encoder_i,
   input    enable_encoder_i,
   output   decoder_o);

   wire  [1:0] encoder_o;  // connects encoder to decoder

   int           error_counter,
                 bad_bit_ct,
                 word_ct;
   logic   [1:0] encoder_o_reg0,
                 encoder_o_reg;
   logic         encoder_i_reg;
   logic         enable_decoder_in;
   logic         enable_encoder_i_reg;
   wire          valid_encoder_o;
   logic   [1:0] err_inj;

   always @ (posedge clk, negedge rst)
      if(!rst) begin
         error_counter        <= 'd0;
         encoder_o_reg        <= 'b0;
         encoder_o_reg0       <= 'b0;
         enable_decoder_in    <= 'b0;
         enable_encoder_i_reg <= 'b0;
         word_ct              <= 'b0;
      end
      else begin
         enable_encoder_i_reg <= enable_encoder_i;
         enable_decoder_in    <= valid_encoder_o;
         error_counter        <= error_counter + 1;
         word_ct              <= word_ct + 1;
```

```
// bit error injection in encoder_o_reg
         encoder_i_reg     <= encoder_i;
         encoder_o_reg0    <= encoder_o;
        if(error_counter[N-1:0]== 3'b001) begin //
            encoder_o_reg <= {~encoder_o[1], encoder_o[0]};    // inject ba
            err_inj <= 2'b01;
         end
         else begin                // clean version
            encoder_o_reg  <= encoder_o;
            err_inj        <= 2'b0;
        end
        if(word_ct<256) begin
          bad_bit_ct  <= bad_bit_ct + (encoder_o_reg0[1]^encoder_o_reg[1])
                                    + (encoder_o_reg0[0]^encoder_o_reg[0]);
           $display("error_counter,err_inj = %h %b %d %d",
                 error_counter,err_inj,bad_bit_ct,word_ct);
        end
      end

// insert your convolutional encoder here
// change port names and module name as necessary/desired
   encoder encoder1      (
      .clk,
      .rst,
      .enable_i(enable_encoder_i), //_reg),
      .d_in    (encoder_i),        //_reg),
      .valid_o (valid_encoder_o),
      .d_out   (encoder_o)   );

// insert your term project code here
   decoder decoder1      (
      .clk,
      .rst,
      .enable (enable_decoder_in),
```

```
// insert your term project code here
   decoder decoder1      (
      .clk,
      .rst,
      .enable (enable_decoder_in),
      .d_in   (encoder_o_reg),
      .d_out  (decoder_o)   );

endmodule
```

```
# run -all
# error_counter,err_inj = 00000000 xx          0          0
# error_counter,err_inj = 00000001 00          0          1
# error_counter,err_inj = 00000002 01          0          2
# error_counter,err_inj = 00000003 00          1          3
# error_counter,err_inj = 00000004 00          1          4
# error_counter,err_inj = 00000005 00          1          5
# error_counter,err_inj = 00000006 00          1          6
# error_counter,err_inj = 00000007 00          1          7
# error_counter,err_inj = 00000008 00          1          8
# error_counter,err_inj = 00000009 00          1          9
# error_counter,err_inj = 0000000a 01          1         10
# error_counter,err_inj = 0000000b 00          2         11
# error_counter,err_inj = 0000000c 00          2         12
# error_counter,err_inj = 0000000d 00          2         13
# error_counter,err_inj = 0000000e 00          2         14
# error_counter,err_inj = 0000000f 00          2         15
# error_counter,err_inj = 00000010 00          2         16
# error_counter,err_inj = 00000011 00          2         17
# error_counter,err_inj = 00000012 01          2         18
# error_counter,err_inj = 00000013 00          3         19
```

Skipping the repetition.

```
# error_counter,err_inj = 000000f7 00          31         247
# error_counter,err_inj = 000000f8 00          31         248
# error_counter,err_inj = 000000f9 00          31         249
# error_counter,err_inj = 000000fa 01          31         250
# error_counter,err_inj = 000000fb 00          32         251
# error_counter,err_inj = 000000fc 00          32         252
# error_counter,err_inj = 000000fd 00          32         253
# error_counter,err_inj = 000000fe 00          32         254
# error_counter,err_inj = 000000ff 00          32         255
# word_count =       10440
# yaa! in = 0, out = 0, w_ct =           0, err = 00
# yaa! in = 0, out = 0, w_ct =           1, err = 00
# yaa! in = 0, out = 0, w_ct =           2, err = 00
# yaa! in = 0, out = 0, w_ct =           3, err = 00
# yaa! in = 0, out = 0, w_ct =           4, err = 00
# yaa! in = 0, out = 0, w_ct =           5, err = 00
# yaa! in = 0, out = 0, w_ct =           6, err = 00
# yaa! in = 0, out = 0, w_ct =           7, err = 00
# yaa! in = 0, out = 0, w_ct =           8, err = 00
```

Skipping the repetition.

```
# yaa! in = 1, out = 1, w_ct =         246, err = 00
# yaa! in = 1, out = 1, w_ct =         247, err = 00
# yaa! in = 0, out = 0, w_ct =         248, err = 00
# yaa! in = 1, out = 1, w_ct =         249, err = 00
# yaa! in = 1, out = 1, w_ct =         250, err = 00
# yaa! in = 1, out = 1, w_ct =         251, err = 00
# yaa! in = 1, out = 1, w_ct =         252, err = 00
# yaa! in = 1, out = 1, w_ct =         253, err = 00
# yaa! in = 1, out = 1, w_ct =         254, err = 00
# yaa! in = 1, out = 1, w_ct =         255, err = 00
# good =         256, bad =           0,            32 bad_bits
# ** Note: $stop    : viterbi_tx_rx_tb.sv(89)
#    Time: 1045 us  Iteration: 0  Instance: /viterbi_tx_rx_tb
# Break in Module viterbi_tx_rx_tb at viterbi_tx_rx_tb.sv line 89
# exit
# End time: 01:49:52 on Mar 24,2024, Elapsed time: 0:00:01
# Errors: 0, Warnings: 0
```

2. With a string of 2 consecutive bad bit[1]s, no output errors are produced.

```
module viterbi_tx_rx #(parameter N=3) (
   input    clk,
   input    rst,
   input    encoder_i,
   input    enable_encoder_i,
   output   decoder_o);

   wire  [1:0] encoder_o;  // connects encoder to decoder

   int           error_counter,
                 bad_bit_ct,
                 word_ct;
   logic   [1:0] encoder_o_reg0,
                 encoder_o_reg;
   logic         encoder_i_reg;
   logic         enable_decoder_in;
   logic         enable_encoder_i_reg;
   wire          valid_encoder_o;
   logic   [1:0] err_inj;

   always @ (posedge clk, negedge rst)
      if(!rst) begin
         error_counter        <= 'd0;
         encoder_o_reg        <= 'b0;
         encoder_o_reg0       <= 'b0;
         enable_decoder_in    <= 'b0;
         enable_encoder_i_reg <= 'b0;
         word_ct              <= 'b0;
      end
      else begin
         enable_encoder_i_reg <= enable_encoder_i;
         enable_decoder_in    <= valid_encoder_o;
         error_counter        <= error_counter + 1;
         word_ct              <= word_ct + 1;
```

```
// bit error injection in encoder_o_reg
         encoder_i_reg     <= encoder_i;
         encoder_o_reg0    <= encoder_o;
        if(error_counter[N-1:0]== 3'b001
           ||error_counter[N-1:0]== 3'b010) begin //
           encoder_o_reg <= {~encoder_o[1], encoder_o[0]};    // inject b
           err_inj <= 2'b01;
         end
         else begin               // clean version
           encoder_o_reg  <= encoder_o;
           err_inj        <= 2'b0;
        end
        if(word_ct<256) begin
          bad_bit_ct  <= bad_bit_ct + (encoder_o_reg0[1]^encoder_o_reg[1])
                                    + (encoder_o_reg0[0]^encoder_o_reg[0]);
          $display("error_counter,err_inj = %h %b %d %d",
                 error_counter,err_inj,bad_bit_ct,word_ct);
        end
      end

// insert your convolutional encoder here
// change port names and module name as necessary/desired
   encoder encoder1      (
      .clk,
      .rst,
      .enable_i(enable_encoder_i), //_reg),
      .d_in    (encoder_i),        //_reg),
      .valid_o (valid_encoder_o),
      .d_out   (encoder_o)   );

// insert your term project code here
   decoder decoder1      (
      .clk,
      .rst,
```

```
76        .enable (enable_decoder_in),
77        .d_in   (encoder_o_reg),
78        .d_out  (decoder_o)   );
79
80  endmodule
```

```
# run -all
# error_counter,err_inj = 00000000 xx           0           0
# error_counter,err_inj = 00000001 00           0           1
# error_counter,err_inj = 00000002 01           0           2
# error_counter,err_inj = 00000003 01           1           3
# error_counter,err_inj = 00000004 00           2           4
# error_counter,err_inj = 00000005 00           2           5
# error_counter,err_inj = 00000006 00           2           6
# error_counter,err_inj = 00000007 00           2           7
# error_counter,err_inj = 00000008 00           2           8
# error_counter,err_inj = 00000009 00           2           9
# error_counter,err_inj = 0000000a 01           2          10
# error_counter,err_inj = 0000000b 01           3          11
# error_counter,err_inj = 0000000c 00           4          12
# error_counter,err_inj = 0000000d 00           4          13
# error_counter,err_inj = 0000000e 00           4          14
# error_counter,err_inj = 0000000f 00           4          15
# error_counter,err_inj = 00000010 00           4          16
# error_counter,err_inj = 00000011 00           4          17
```

Skipping the repetition.

```
# error_counter,err_inj = 000000f8 00          62         248
# error_counter,err_inj = 000000f9 00          62         249
# error_counter,err_inj = 000000fa 01          62         250
# error_counter,err_inj = 000000fb 01          63         251
# error_counter,err_inj = 000000fc 00          64         252
# error_counter,err_inj = 000000fd 00          64         253
# error_counter,err_inj = 000000fe 00          64         254
# error_counter,err_inj = 000000ff 00          64         255
# word_count =       10440
# yaa! in = 0, out = 0, w_ct =           0, err = 00
# yaa! in = 0, out = 0, w_ct =           1, err = 00
# yaa! in = 0, out = 0, w_ct =           2, err = 00
# yaa! in = 0, out = 0, w_ct =           3, err = 00
# yaa! in = 0, out = 0, w_ct =           4, err = 00
# yaa! in = 0, out = 0, w_ct =           5, err = 00
# yaa! in = 0, out = 0, w_ct =           6, err = 00
```

Skipping the repetition.

```
# yaa! in = 1, out = 1, w_ct =         246, err = 00
# yaa! in = 1, out = 1, w_ct =         247, err = 00
# yaa! in = 0, out = 0, w_ct =         248, err = 00
# yaa! in = 1, out = 1, w_ct =         249, err = 00
# yaa! in = 1, out = 1, w_ct =         250, err = 00
# yaa! in = 1, out = 1, w_ct =         251, err = 00
# yaa! in = 1, out = 1, w_ct =         252, err = 00
# yaa! in = 1, out = 1, w_ct =         253, err = 00
# yaa! in = 1, out = 1, w_ct =         254, err = 00
# yaa! in = 1, out = 1, w_ct =         255, err = 00
# good =         256, bad =           0,            64 bad_bits
# ** Note: $stop    : viterbi_tx_rx_tb.sv(89)
#    Time: 1045 us  Iteration: 0  Instance: /viterbi_tx_rx_tb
# Break in Module viterbi_tx_rx_tb at viterbi_tx_rx_tb.sv line 89
# exit
# End time: 01:39:18 on Mar 24,2024, Elapsed time: 0:00:01
```

3. With a string of 3 consecutive bad bit[1]s, output errors are produced.

```
module viterbi_tx_rx #(parameter N=3) (
   input    clk,
   input    rst,
   input    encoder_i,
   input    enable_encoder_i,
   output   decoder_o);

   wire  [1:0] encoder_o;  // connects encoder to decoder

   int           error_counter,
                 bad_bit_ct,
                 word_ct;
   logic   [1:0] encoder_o_reg0,
                 encoder_o_reg;
   logic         encoder_i_reg;
   logic         enable_decoder_in;
   logic         enable_encoder_i_reg;
   wire          valid_encoder_o;
   logic   [1:0] err_inj;

   always @ (posedge clk, negedge rst)
      if(!rst) begin
         error_counter        <= 'd0;
         encoder_o_reg        <= 'b0;
         encoder_o_reg0       <= 'b0;
         enable_decoder_in    <= 'b0;
         enable_encoder_i_reg <= 'b0;
         word_ct              <= 'b0;
      end
      else begin
         enable_encoder_i_reg <= enable_encoder_i;
         enable_decoder_in    <= valid_encoder_o;
         error_counter        <= error_counter + 1;
         word_ct              <= word_ct + 1;
```

```
// bit error injection in encoder_o_reg
         encoder_i_reg     <= encoder_i;
         encoder_o_reg0    <= encoder_o;
        if(error_counter[N-1:0]== 3'b001
           ||error_counter[N-1:0]== 3'b010
           ||error_counter[N-1:0]== 3'b011) begin //
            encoder_o_reg <= {~encoder_o[1], encoder_o[0]};    // inject ba
            err_inj <= 2'b01;
         end
         else begin                // clean version
            encoder_o_reg  <= encoder_o;
            err_inj        <= 2'b0;
        end
        if(word_ct<256) begin
          bad_bit_ct  <= bad_bit_ct + (encoder_o_reg0[1]^encoder_o_reg[1])
                                    + (encoder_o_reg0[0]^encoder_o_reg[0]);
          $display("error_counter,err_inj = %h %b %d %d",
                 error_counter,err_inj,bad_bit_ct,word_ct);
        end
      end

// insert your convolutional encoder here
// change port names and module name as necessary/desired
   encoder encoder1      (
      .clk,
      .rst,
      .enable_i(enable_encoder_i), //_reg),
      .d_in    (encoder_i),        //_reg),
      .valid_o (valid_encoder_o),
      .d_out   (encoder_o)   );

// insert your term project code here
   decoder decoder1      (
      .clk,
```

```
      .rst,
      .enable (enable_decoder_in),
      .d_in   (encoder_o_reg),
      .d_out  (decoder_o)   );

endmodule
```

```
# run -all
# error_counter,err_inj = 00000000 xx          0          0
# error_counter,err_inj = 00000001 00          0          1
# error_counter,err_inj = 00000002 01          0          2
# error_counter,err_inj = 00000003 01          1          3
# error_counter,err_inj = 00000004 01          2          4
# error_counter,err_inj = 00000005 00          3          5
# error_counter,err_inj = 00000006 00          3          6
# error_counter,err_inj = 00000007 00          3          7
# error_counter,err_inj = 00000008 00          3          8
# error_counter,err_inj = 00000009 00          3          9
# error_counter,err_inj = 0000000a 01          3         10
# error_counter,err_inj = 0000000b 01          4         11
# error_counter,err_inj = 0000000c 01          5         12
# error_counter,err_inj = 0000000d 00          6         13
# error_counter,err_inj = 0000000e 00          6         14
# error_counter,err_inj = 0000000f 00          6         15
# error_counter,err_inj = 00000010 00          6         16
# error_counter,err_inj = 00000011 00          6         17
# error_counter,err_inj = 00000012 01          6         18
# error_counter,err_inj = 00000013 01          7         19
# error_counter,err_inj = 00000014 01          8         20
# error_counter,err_inj = 00000015 00          9         21
```

Skipping the repetition.

```
# error_counter,err_inj = 000000f6 00          93         246
# error_counter,err_inj = 000000f7 00          93         247
# error_counter,err_inj = 000000f8 00          93         248
# error_counter,err_inj = 000000f9 00          93         249
# error_counter,err_inj = 000000fa 01          93         250
# error_counter,err_inj = 000000fb 01          94         251
# error_counter,err_inj = 000000fc 01          95         252
# error_counter,err_inj = 000000fd 00          96         253
# error_counter,err_inj = 000000fe 00          96         254
# error_counter,err_inj = 000000ff 00          96         255
# word_count =       10440
# yaa! in = 0, out = 0, w_ct =           0, err = 00
# yaa! in = 0, out = 0, w_ct =           1, err = 00
# yaa! in = 0, out = 0, w_ct =           2, err = 00
# yaa! in = 0, out = 0, w_ct =           3, err = 00
# yaa! in = 0, out = 0, w_ct =           4, err = 00
# yaa! in = 0, out = 0, w_ct =           5, err = 00
# yaa! in = 0, out = 0, w_ct =           6, err = 00
# yaa! in = 0, out = 0, w_ct =           7, err = 00
# yaa! in = 0, out = 0, w_ct =           8, err = 00
# yaa! in = 0, out = 0, w_ct =           9, err = 00
# boo! in = 0, out = 1, w_ct =          10, err = 00,              1045000, BAD!
```

Skipping the repetition.

```
# boo! in = 1, out = 0, w_ct =         244, err = 00,              1045000, BAD!
# yaa! in = 1, out = 1, w_ct =         245, err = 00
# yaa! in = 1, out = 1, w_ct =         246, err = 00
# yaa! in = 1, out = 1, w_ct =         247, err = 00
# yaa! in = 0, out = 0, w_ct =         248, err = 00
# yaa! in = 1, out = 1, w_ct =         249, err = 00
# boo! in = 1, out = 0, w_ct =         250, err = 00,              1045000, BAD!
# yaa! in = 1, out = 1, w_ct =         251, err = 00
# yaa! in = 1, out = 1, w_ct =         252, err = 00
# yaa! in = 1, out = 1, w_ct =         253, err = 00
# yaa! in = 1, out = 1, w_ct =         254, err = 00
# yaa! in = 1, out = 1, w_ct =         255, err = 00
# good =         217, bad =          39,            96 bad_bits
# ** Note: $stop    : viterbi_tx_rx_tb.sv(89)
#    Time: 1045 us  Iteration: 0  Instance: /viterbi_tx_rx_tb
```

2.e

We found that a string of 2 consecutive bad bit[1]bit[0]s is needed to produce output errors.

1. With a string of 1 consecutive bad bit[1]bit[0]s, no output errors are produced.

```
module viterbi_tx_rx #(parameter N=4) (
   input    clk,
   input    rst,
   input    encoder_i,
   input    enable_encoder_i,
   output   decoder_o);

   wire  [1:0] encoder_o;  // connects encoder to decoder

   int           error_counter,
                 bad_bit_ct,
                 word_ct;
   logic   [1:0] encoder_o_reg0,
                 encoder_o_reg;
   logic         encoder_i_reg;
   logic         enable_decoder_in;
   logic         enable_encoder_i_reg;
   wire          valid_encoder_o;
   logic   [1:0] err_inj;

   always @ (posedge clk, negedge rst)
      if(!rst) begin
         error_counter        <= 'd0;
         encoder_o_reg        <= 'b0;
         encoder_o_reg0       <= 'b0;
         enable_decoder_in    <= 'b0;
         enable_encoder_i_reg <= 'b0;
         word_ct              <= 'b0;
      end
      else begin
         enable_encoder_i_reg <= enable_encoder_i;
         enable_decoder_in    <= valid_encoder_o;
         error_counter        <= error_counter + 1;
         word_ct              <= word_ct + 1;
```

```
// bit error injection in encoder_o_reg
         encoder_i_reg     <= encoder_i;
         encoder_o_reg0    <= encoder_o;
        if(error_counter[N-1:0]== 4'b0001) begin //
          encoder_o_reg <= {~encoder_o[1], ~encoder_o[0]};    // inject ba
            err_inj <= 2'b01;
         end
         else begin                // clean version
            encoder_o_reg  <= encoder_o;
            err_inj        <= 2'b0;
        end
        if(word_ct<256) begin
          bad_bit_ct  <= bad_bit_ct + (encoder_o_reg0[1]^encoder_o_reg[1])
                                    + (encoder_o_reg0[0]^encoder_o_reg[0]);
          $display("error_counter,err_inj = %h %b %d %d",
                 error_counter,err_inj,bad_bit_ct,word_ct);
        end
      end

// insert your convolutional encoder here
// change port names and module name as necessary/desired
   encoder encoder1      (
      .clk,
      .rst,
      .enable_i(enable_encoder_i), //_reg),
      .d_in    (encoder_i),        //_reg),
      .valid_o (valid_encoder_o),
      .d_out   (encoder_o)   );

// insert your term project code here
   decoder decoder1      (
      .clk,
      .rst,
      .enable (enable_decoder_in),
```

```
      .d_in   (encoder_o_reg),
      .d_out  (decoder_o)   );

endmodule
```

```
# run -all
# error_counter,err_inj = 00000000 xx          0           0
# error_counter,err_inj = 00000001 00          0           1
# error_counter,err_inj = 00000002 01          0           2
# error_counter,err_inj = 00000003 00          2           3
# error_counter,err_inj = 00000004 00          2           4
# error_counter,err_inj = 00000005 00          2           5
# error_counter,err_inj = 00000006 00          2           6
# error_counter,err_inj = 00000007 00          2           7
# error_counter,err_inj = 00000008 00          2           8
# error_counter,err_inj = 00000009 00          2           9
# error_counter,err_inj = 0000000a 00          2          10
# error_counter,err_inj = 0000000b 00          2          11
# error_counter,err_inj = 0000000c 00          2          12
# error_counter,err_inj = 0000000d 00          2          13
# error_counter,err_inj = 0000000e 00          2          14
# error_counter,err_inj = 0000000f 00          2          15
# error_counter,err_inj = 00000010 00          2          16
# error_counter,err_inj = 00000011 00          2          17
# error_counter,err_inj = 00000012 01          2          18
# error_counter,err_inj = 00000013 00          4          19
# error_counter,err_inj = 00000014 00          4          20
```

Skipping the repetition.

```
# error_counter,err_inj = 000000f8 00          32         248
# error_counter,err_inj = 000000f9 00          32         249
# error_counter,err_inj = 000000fa 00          32         250
# error_counter,err_inj = 000000fb 00          32         251
# error_counter,err_inj = 000000fc 00          32         252
# error_counter,err_inj = 000000fd 00          32         253
# error_counter,err_inj = 000000fe 00          32         254
# error_counter,err_inj = 000000ff 00          32         255
# word_count =       10440
# yaa! in = 0, out = 0, w_ct =           0, err = 00
# yaa! in = 0, out = 0, w_ct =           1, err = 00
# yaa! in = 0, out = 0, w_ct =           2, err = 00
# yaa! in = 0, out = 0, w_ct =           3, err = 00
# yaa! in = 0, out = 0, w_ct =           4, err = 00
# yaa! in = 0, out = 0, w_ct =           5, err = 00
# yaa! in = 0, out = 0, w_ct =           6, err = 00
# yaa! in = 0, out = 0, w_ct =           7, err = 00
```

Skipping the repetition.

```
# yaa! in = 1, out = 1, w_ct =         245, err = 00
# yaa! in = 1, out = 1, w_ct =         246, err = 00
# yaa! in = 1, out = 1, w_ct =         247, err = 00
# yaa! in = 0, out = 0, w_ct =         248, err = 00
# yaa! in = 1, out = 1, w_ct =         249, err = 00
# yaa! in = 1, out = 1, w_ct =         250, err = 00
# yaa! in = 1, out = 1, w_ct =         251, err = 00
# yaa! in = 1, out = 1, w_ct =         252, err = 00
# yaa! in = 1, out = 1, w_ct =         253, err = 00
# yaa! in = 1, out = 1, w_ct =         254, err = 00
# yaa! in = 1, out = 1, w_ct =         255, err = 00
# good =         256, bad =           0,            32 bad_bits
# ** Note: $stop    : viterbi_tx_rx_tb.sv(89)
```

2. With a string of 2 consecutive bad bit[1]bit[0]s, output errors are produced.

```
module viterbi_tx_rx #(parameter N=4) (
   input    clk,
   input    rst,
   input    encoder_i,
   input    enable_encoder_i,
   output   decoder_o);

   wire  [1:0] encoder_o;  // connects encoder to decoder

   int           error_counter,
                 bad_bit_ct,
                 word_ct;
   logic   [1:0] encoder_o_reg0,
                 encoder_o_reg;
   logic         encoder_i_reg;
   logic         enable_decoder_in;
   logic         enable_encoder_i_reg;
   wire          valid_encoder_o;
   logic   [1:0] err_inj;

   always @ (posedge clk, negedge rst)
      if(!rst) begin
         error_counter        <= 'd0;
         encoder_o_reg        <= 'b0;
         encoder_o_reg0       <= 'b0;
         enable_decoder_in    <= 'b0;
         enable_encoder_i_reg <= 'b0;
         word_ct              <= 'b0;
      end
      else begin
         enable_encoder_i_reg <= enable_encoder_i;
         enable_decoder_in    <= valid_encoder_o;
         error_counter        <= error_counter + 1;
         word_ct              <= word_ct + 1;
```

```
// bit error injection in encoder_o_reg
         encoder_i_reg     <= encoder_i;
         encoder_o_reg0    <= encoder_o;
        if(error_counter[N-1:0]== 4'b0001
           ||error_counter[N-1:0]== 4'b0010) begin //
          encoder_o_reg <= {~encoder_o[1], ~encoder_o[0]};    // inject bad
            err_inj <= 2'b01;
         end
         else begin                // clean version
            encoder_o_reg  <= encoder_o;
            err_inj        <= 2'b0;
        end
        if(word_ct<256) begin
          bad_bit_ct  <= bad_bit_ct + (encoder_o_reg0[1]^encoder_o_reg[1])
                                    + (encoder_o_reg0[0]^encoder_o_reg[0]);
          $display("error_counter,err_inj = %h %b %d %d",
                 error_counter,err_inj,bad_bit_ct,word_ct);
        end
      end

// insert your convolutional encoder here
// change port names and module name as necessary/desired
   encoder encoder1      (
      .clk,
      .rst,
      .enable_i(enable_encoder_i), //_reg),
      .d_in    (encoder_i),        //_reg),
      .valid_o (valid_encoder_o),
      .d_out   (encoder_o)   );

// insert your term project code here
   decoder decoder1      (
      .clk,
      .rst,
```

```
      .enable (enable_decoder_in),
      .d_in   (encoder_o_reg),
      .d_out  (decoder_o)    );

endmodule
```

```
# run -all
# error_counter,err_inj = 00000000 xx          0          0
# error_counter,err_inj = 00000001 00          0          1
# error_counter,err_inj = 00000002 01          0          2
# error_counter,err_inj = 00000003 01          2          3
# error_counter,err_inj = 00000004 00          4          4
# error_counter,err_inj = 00000005 00          4          5
# error_counter,err_inj = 00000006 00          4          6
# error_counter,err_inj = 00000007 00          4          7
# error_counter,err_inj = 00000008 00          4          8
# error_counter,err_inj = 00000009 00          4          9
# error_counter,err_inj = 0000000a 00          4         10
# error_counter,err_inj = 0000000b 00          4         11
# error_counter,err_inj = 0000000c 00          4         12
# error_counter,err_inj = 0000000d 00          4         13
# error_counter,err_inj = 0000000e 00          4         14
# error_counter,err_inj = 0000000f 00          4         15
# error_counter,err_inj = 00000010 00          4         16
# error_counter,err_inj = 00000011 00          4         17
# error_counter,err_inj = 00000012 01          4         18
# error_counter,err_inj = 00000013 01          6         19
```

Skipping the repetition.

```
# error_counter,err_inj = 000000fa 00          64         250
# error_counter,err_inj = 000000fb 00          64         251
# error_counter,err_inj = 000000fc 00          64         252
# error_counter,err_inj = 000000fd 00          64         253
# error_counter,err_inj = 000000fe 00          64         254
# error_counter,err_inj = 000000ff 00          64         255
# word_count =       10440
# yaa! in = 0, out = 0, w_ct =           0, err = 00
# yaa! in = 0, out = 0, w_ct =           1, err = 00
# yaa! in = 0, out = 0, w_ct =           2, err = 00
# yaa! in = 0, out = 0, w_ct =           3, err = 00
# yaa! in = 0, out = 0, w_ct =           4, err = 00
# yaa! in = 0, out = 0, w_ct =           5, err = 00
# yaa! in = 0, out = 0, w_ct =           6, err = 00
# yaa! in = 0, out = 0, w_ct =           7, err = 00
# yaa! in = 0, out = 0, w_ct =           8, err = 00
# yaa! in = 0, out = 0, w_ct =           9, err = 00
# boo! in = 0, out = 1, w_ct =          10, err = 00,              1045000, BAD!
# boo! in = 1, out = 0, w_ct =          11, err = 00,              1045000, BAD!
```

Skipping the repetition.

```
# yaa! in = 1, out = 1, w_ct =         247, err = 00
# yaa! in = 0, out = 0, w_ct =         248, err = 00
# yaa! in = 1, out = 1, w_ct =         249, err = 00
# boo! in = 1, out = 0, w_ct =         250, err = 00,              1045000, BAD!
# boo! in = 1, out = 0, w_ct =         251, err = 00,              1045000, BAD!
# yaa! in = 1, out = 1, w_ct =         252, err = 00
# yaa! in = 1, out = 1, w_ct =         253, err = 00
# yaa! in = 1, out = 1, w_ct =         254, err = 00
# boo! in = 1, out = 0, w_ct =         255, err = 00,              1045000, BAD!
# good =         212, bad =          44,            64 bad_bits
# ** Note: $stop    : viterbi_tx_rx_tb.sv(89)
#    Time: 1045 us  Iteration: 0  Instance: /viterbi_tx_rx_tb
# Break in Module viterbi_tx_rx_tb at viterbi_tx_rx_tb.sv line 89
# exit
# End time: 02:14:54 on Mar 24,2024, Elapsed time: 0:00:01
# Errors: 0, Warnings: 0
```