

ECE 111: Advanced Digital Design

Project 2024

The Viterbi Algorithm

March 21, 2024

Andy Tu

PID: A17650683

Professor: John Eldon

Table of contents

Part 1: The Viterbi Algorithm.....	3
Viterbi_tx_rx.....	3
Encoder.....	4
Decoder.....	6
BMC Files.....	10
ACS.....	15
TBU.....	16
Simulation Transcript.....	16
Proof of synthesis.....	17
Encoder explanation.....	20
Decoder explanation.....	20
Part 2: Injecting the bad bits.....	21
Testbench.....	22
Part A.....	25
2.a.1.....	25
2.a.2.....	26
2.a.3.....	27
2.a.4.....	28
2.a.5.....	29
2.a.6.....	30
2.a.7.....	32
2.a.8.....	33
Part B.....	34
Testbench.....	34
B initial Parameter changes.....	35
Injecting 2 bad bits at [1] and [0] for N = 1.....	36
Injecting 2 bad bits at [1] and [0] for N = 2.....	36
Injecting 2 bad bits at [1] and [0] for N = 3.....	36
Injecting 2 bad bits at [1] and [0] for N = 4.....	36
2b1.....	37
2b2.....	38
2b3.....	39
2b4.....	40
2b5.....	41
2b6.....	42
2b7.....	43
2b8.....	44
Part C.....	45
Part D.....	46
Part E.....	47

Part 1: The Viterbi Algorithm

Viterbi_tx_rx

```
1 module viterbi_tx_rx #(parameter N=2) (
2     input  clk,
3     input  rst,
4     input  enable_encoder_i,
5     input  enable_decoder_i,
6     output decoder_o);
7
8     wire [1:0] encoder_o; // connects encoder to decoder
9
10    int
11        error_counter,
12        bad_bit_ct,
13        word_ct;
14    logic [1:0] encoder_o_reg;
15    logic [1:0] encoder_i_reg;
16    logic enable_decoder_in;
17    logic enable_encoder_i_reg;
18    wire [1:0] valid_encoder_o;
19    logic [1:0] err_inj;
20
21    always @ (posedge clk, negedge rst)
22    begin
23        if (rst) begin
24            error_counter <= 'd0;
25            encoder_o_reg <= 'b0;
26            enable_decoder_in <= 'b0;
27            enable_encoder_i_reg <= 'b0;
28            word_ct <= 'b0;
29        end
30        else begin
31            enable_encoder_i_reg <= enable_encoder_i;
32            enable_decoder_in <= enable_decoder_i;
33            encoder_o_reg <= 'b0;
34            //error_counter <= $random;
35            word_ct <= word_ct + 1;
36            // bit error injection in encoder_o_reg
37            encoder_i_reg <= encoder_i;
38            if (error_counter[N-1:0] == 1) begin
39                err_inj <= error_counter[29:28];
40                if (word_ct < 256)
41                    bad_bit_ct <= bad_bit_ct + error_counter[29] + error_counter[28];
42                $display("error_counter err_inj = %h %b %d", error_counter, err_inj, bad_bit_ct);
43                encoder_o_reg <= encoder_o ^ err_inj; // inject bad bits
44            end
45            else begin // clean version
46                encoder_o_reg <= {encoder_o[1], encoder_o[0]};
47                err_inj <= 2'b0;
48            end
49        end
50    end
51
52    // insert your convolutional encoder here
53    // change port names and module name as necessary/desired
54    encoder encoder1 (
55        .clk,
56        .rst,
57        .enable_i(enable_encoder_i), //_reg,
58        .d_in (encoder_i), //_reg,
59        .valid_o (valid_encoder_o),
60        .d_out (encoder_o));
61
62    // insert your term project code here
63    decoder decoder1 (
64        .clk,
65        .rst,
66        .enable (enable_decoder_in),
67        .d_in (encoder_o_reg),
68        .d_out (decoder_o));
69
70    endmodule
```

Encoder

```
1 module encoder // use this one
2 (
3     input      clk,
4     input      rst,
5     input      enable_i,
6     input      d_in,
7     output logic valid_o,
8     output logic [1:0] d_out);
9
10    logic [2:0] cstate;
11    logic [2:0] nstate;
12    logic valid_oq;
13    logic [1:0] d_out_reg;
14
15    always_comb begin
16        valid_oq = enable_i;
17        case (cstate)
18            3'b000: if(!d_in) begin
19                nstate = 3'b000;
20                d_out_reg = 2'b00;
21            end
22            else begin
23                nstate = 3'b100;
24                d_out_reg = 2'b11;
25            end
26            3'b001: if(!d_in) begin
27                nstate = 3'b100;
28                d_out_reg = 2'b00;
29            end
30            else begin
31                nstate = 3'b000;
32                d_out_reg = 2'b11;
33            end
34            3'b010: if(!d_in) begin
35                nstate = 3'b101;
36                d_out_reg = 2'b10;
37            end
38            else begin
39                nstate = 3'b001;
40                d_out_reg = 2'b01;
41            end
42            3'b011: if(!d_in) begin
43                nstate = 3'b001;
44                d_out_reg = 2'b10;
45            end
46            else begin
47                nstate = 3'b101;
48                d_out_reg = 2'b01;
49            end
50            3'b100: if(!d_in) begin
51                nstate = 3'b010;
52                d_out_reg = 2'b10;
53            end
54            else begin
55                nstate = 3'b110;
56                d_out_reg = 2'b01;
57            end
58            3'b101: if(!d_in) begin
59                nstate = 3'b110;
60                d_out_reg = 2'b10;
61            end
62            else begin
63                nstate = 3'b010;
64                d_out_reg = 2'b01;
65            end
66            3'b110: if(!d_in) begin
```

```

61         else begin
62             nstate = 3'b010;
63             d_out_reg = 2'b01;
64         end
65     3'b110: if(!d_in) begin
66         nstate = 3'b111;
67         d_out_reg = 2'b00;
68     end
69     else begin
70         nstate = 3'b011;
71         d_out_reg = 2'b11;
72     end
73     3'b111: if(!d_in) begin
74         nstate = 3'b011;
75         d_out_reg = 2'b00;
76     end
77     else begin
78         nstate = 3'b111;
79         d_out_reg = 2'b11;
80     end
81 endcase
82 end
83
84 always @ (posedge clk, negedge rst)
85 if(!rst)
86     cstate <= 'b0;
87 else
88     cstate <= enable_i? nstate : 'b0;
89 always @ (posedge clk) begin
90     d_out <= enable_i? d_out_reg : 'b0;
91     valid_o <= valid_oQ;
92 end
93
94 endmodule
95

```

Decoder

```
Text Editor - C:/Users/William/Desktop/New folder (5)/viterbi_tx_rx - viterbi_tx_rx - [decoder.v]
File Edit View Project Processing Tools Window Help
Search altera.com

1 module decoder
2
3     input        clk;
4     input        rst;
5     input        enable;
6     input [1:0]  d_in;
7     output reg   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;
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 //Trellis 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
72 // mem counter
```

Ln 37 Col 39 SystemVerilog HDL File 100% 00:00:36

```
Text Editor - C:\Users\William\Desktop\New folder (5)\viterbi_tx - viterbi_tx_n - [decoder.sv]
File Edit View Project Processing Tools Window Help
70 logic mem_bank_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;
90 wire [7:0] d_o_mem_B;
91 wire [7:0] d_o_mem_C;
92 wire [7:0] d_o_mem_D;
93
94 //Trace back module signals
95 logic selection_tbu_0;
96 logic selection_tbu_1;
97
98 logic [7:0] d_in_0_tbu_0;
99 logic [7:0] d_in_1_tbu_0;
100 logic [7:0] d_in_0_tbu_1;
101 logic [7:0] d_in_1_tbu_1;
102
103 wire d_o_tbu_0;
104 wire d_o_tbu_1;
105
106 logic enable_tbu_0;
107 logic enable_tbu_1;
108
109 //Display memory operations
110 wire wr_disp_mem_0;
111 wire wr_disp_mem_1;
112
113 wire d_in_disp_mem_0;
114 wire d_in_disp_mem_1;
115
116 logic [9:0] wr_mem_counter_disp;
117 logic [9:0] rd_mem_counter_disp;
118
119 logic [9:0] addr_disp_mem_0;
120 logic [9:0] addr_disp_mem_1;
121
122 //Branch matrc calculation modules
123
124 bmc000 bmc000_inst(d_in_bmc000_path_0_bmc,bmc000_path_1_bmc);
125 bmc001 bmc001_inst(d_in_bmc001_path_0_bmc,bmc001_path_1_bmc);
126 bmc010 bmc010_inst(d_in_bmc010_path_0_bmc,bmc010_path_1_bmc);
127 bmc011 bmc011_inst(d_in_bmc011_path_0_bmc,bmc011_path_1_bmc);
128 bmc100 bmc100_inst(d_in_bmc100_path_0_bmc,bmc100_path_1_bmc);
129 bmc101 bmc101_inst(d_in_bmc101_path_0_bmc,bmc101_path_1_bmc);
130 bmc110 bmc110_inst(d_in_bmc110_path_0_bmc,bmc110_path_1_bmc);
131 bmc111 bmc111_inst(d_in_bmc111_path_0_bmc,bmc111_path_1_bmc);
132
133
134
135
136
137 //Add Compare Select Modules
138 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);
139 ACS ACS001(validity[0],validity[1],bmc001_path_0_bmc,bmc001_path_1_bmc,path_cost[0],path_cost[1],ACS001_selection,ACS001_valid_o,ACS001_path_cost);
140 ACS ACS010(validity[0],validity[1],bmc010_path_0_bmc,bmc010_path_1_bmc,path_cost[0],path_cost[1],ACS010_selection,ACS010_valid_o,ACS010_path_cost);
141 ACS ACS011(validity[0],validity[1],bmc011_path_0_bmc,bmc011_path_1_bmc,path_cost[0],path_cost[1],ACS011_selection,ACS011_valid_o,ACS011_path_cost);
142 ACS ACS100(validity[0],validity[1],bmc100_path_0_bmc,bmc100_path_1_bmc,path_cost[0],path_cost[1],ACS100_selection,ACS100_valid_o,ACS100_path_cost);
143 ACS ACS101(validity[0],validity[1],bmc101_path_0_bmc,bmc101_path_1_bmc,path_cost[0],path_cost[1],ACS101_selection,ACS101_valid_o,ACS101_path_cost);
144 ACS ACS110(validity[0],validity[1],bmc110_path_0_bmc,bmc110_path_1_bmc,path_cost[0],path_cost[1],ACS110_selection,ACS110_valid_o,ACS110_path_cost);
145 ACS ACS111(validity[0],validity[1],bmc111_path_0_bmc,bmc111_path_1_bmc,path_cost[0],path_cost[1],ACS111_selection,ACS111_valid_o,ACS111_path_cost);
146
147 assign selection_nets = {ACS111_selection,ACS110_selection,ACS101_selection,ACS100_selection,
148 ACS011_selection,ACS010_selection,ACS001_selection,ACS000_selection};
149 assign validity_nets = {ACS111_valid_o,ACS110_valid_o,ACS101_valid_o,ACS100_valid_o,
150 ACS011_valid_o,ACS010_valid_o,ACS001_valid_o,ACS000_valid_o};
151
152
153 always @(posedge clk, negedge rst) begin
154 if(rst) begin
155 validity
156 selection
157 // clear all 8 path costs
158 path_cost[0] <= 1'b0;
159 path_cost[1] <= 1'b0;
160 path_cost[2] <= 1'b0;
161 path_cost[3] <= 1'b0;
162 path_cost[4] <= 1'b0;
163 path_cost[5] <= 1'b0;
164 path_cost[6] <= 1'b0;
165 path_cost[7] <= 1'b0;
166
167 end
168 else if(tenable) begin
169 validity
170 selection
171 // clear all 8 path costs
172 path_cost[0] <= 1'b0;
173 path_cost[1] <= 1'b0;
174 path_cost[2] <= 1'b0;
175 path_cost[3] <= 1'b0;
176 path_cost[4] <= 1'b0;
177 path_cost[5] <= 1'b0;
178 path_cost[6] <= 1'b0;
179 path_cost[7] <= 1'b0;
180
181 end
182 else if( path_cost[0][7] && path_cost[1][7] && path_cost[2][7] && path_cost[3][7] &&
183 path_cost[4][7] && path_cost[5][7] && path_cost[6][7] && path_cost[7][7] )
184 begin
185 validity_nets
186 selection_nets
187 path_cost[0] <= 8'b01111111 & ACS000_path_cost;
188 path_cost[1] <= 8'b01111111 & ACS001_path_cost;
189 path_cost[2] <= 8'b01111111 & ACS010_path_cost;
190 path_cost[3] <= 8'b01111111 & ACS011_path_cost;
191 path_cost[4] <= 8'b01111111 & ACS100_path_cost;
192 path_cost[5] <= 8'b01111111 & ACS101_path_cost;
193 path_cost[6] <= 8'b01111111 & ACS110_path_cost;
194 path_cost[7] <= 8'b01111111 & ACS111_path_cost;
195
196 /* likewise for path_cost[1:7] and ACS001:111_path_cost done
197 */
198 end
199 else begin
200 validity
201 selection
202 path_cost[0] <= ACS000_path_cost;
203 path_cost[1] <= ACS001_path_cost;
204 path_cost[2] <= ACS010_path_cost;
205 path_cost[3] <= ACS011_path_cost;
206 path_cost[4] <= ACS100_path_cost;
207 path_cost[5] <= ACS101_path_cost;
208 path_cost[6] <= ACS110_path_cost;
209 path_cost[7] <= ACS111_path_cost;
210
Ln 109 Col 28 SystemVerilog HDL File 100% 00:00:36
```

```
Text Editor - C:\Users\William\Desktop\New folder (5)\viterbi_tx - viterbi_tx_n - [decoder.sv]
File Edit View Project Processing Tools Window Help
138 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);
139 ACS ACS001(validity[0],validity[1],bmc001_path_0_bmc,bmc001_path_1_bmc,path_cost[0],path_cost[1],ACS001_selection,ACS001_valid_o,ACS001_path_cost);
140 ACS ACS010(validity[0],validity[1],bmc010_path_0_bmc,bmc010_path_1_bmc,path_cost[0],path_cost[1],ACS010_selection,ACS010_valid_o,ACS010_path_cost);
141 ACS ACS011(validity[0],validity[1],bmc011_path_0_bmc,bmc011_path_1_bmc,path_cost[0],path_cost[1],ACS011_selection,ACS011_valid_o,ACS011_path_cost);
142 ACS ACS100(validity[0],validity[1],bmc100_path_0_bmc,bmc100_path_1_bmc,path_cost[0],path_cost[1],ACS100_selection,ACS100_valid_o,ACS100_path_cost);
143 ACS ACS101(validity[0],validity[1],bmc101_path_0_bmc,bmc101_path_1_bmc,path_cost[0],path_cost[1],ACS101_selection,ACS101_valid_o,ACS101_path_cost);
144 ACS ACS110(validity[0],validity[1],bmc110_path_0_bmc,bmc110_path_1_bmc,path_cost[0],path_cost[1],ACS110_selection,ACS110_valid_o,ACS110_path_cost);
145 ACS ACS111(validity[0],validity[1],bmc111_path_0_bmc,bmc111_path_1_bmc,path_cost[0],path_cost[1],ACS111_selection,ACS111_valid_o,ACS111_path_cost);
146
147 assign selection_nets = {ACS111_selection,ACS110_selection,ACS101_selection,ACS100_selection,
148 ACS011_selection,ACS010_selection,ACS001_selection,ACS000_selection};
149 assign validity_nets = {ACS111_valid_o,ACS110_valid_o,ACS101_valid_o,ACS100_valid_o,
150 ACS011_valid_o,ACS010_valid_o,ACS001_valid_o,ACS000_valid_o};
151
152
153 always @(posedge clk, negedge rst) begin
154 if(rst) begin
155 validity
156 selection
157 // clear all 8 path costs
158 path_cost[0] <= 1'b0;
159 path_cost[1] <= 1'b0;
160 path_cost[2] <= 1'b0;
161 path_cost[3] <= 1'b0;
162 path_cost[4] <= 1'b0;
163 path_cost[5] <= 1'b0;
164 path_cost[6] <= 1'b0;
165 path_cost[7] <= 1'b0;
166
167 end
168 else if(tenable) begin
169 validity
170 selection
171 // clear all 8 path costs
172 path_cost[0] <= 1'b0;
173 path_cost[1] <= 1'b0;
174 path_cost[2] <= 1'b0;
175 path_cost[3] <= 1'b0;
176 path_cost[4] <= 1'b0;
177 path_cost[5] <= 1'b0;
178 path_cost[6] <= 1'b0;
179 path_cost[7] <= 1'b0;
180
181 end
182 else if( path_cost[0][7] && path_cost[1][7] && path_cost[2][7] && path_cost[3][7] &&
183 path_cost[4][7] && path_cost[5][7] && path_cost[6][7] && path_cost[7][7] )
184 begin
185 validity_nets
186 selection_nets
187 path_cost[0] <= 8'b01111111 & ACS000_path_cost;
188 path_cost[1] <= 8'b01111111 & ACS001_path_cost;
189 path_cost[2] <= 8'b01111111 & ACS010_path_cost;
190 path_cost[3] <= 8'b01111111 & ACS011_path_cost;
191 path_cost[4] <= 8'b01111111 & ACS100_path_cost;
192 path_cost[5] <= 8'b01111111 & ACS101_path_cost;
193 path_cost[6] <= 8'b01111111 & ACS110_path_cost;
194 path_cost[7] <= 8'b01111111 & ACS111_path_cost;
195
196 /* likewise for path_cost[1:7] and ACS001:111_path_cost done
197 */
198 end
199 else begin
200 validity
201 selection
202 path_cost[0] <= ACS000_path_cost;
203 path_cost[1] <= ACS001_path_cost;
204 path_cost[2] <= ACS010_path_cost;
205 path_cost[3] <= ACS011_path_cost;
206 path_cost[4] <= ACS100_path_cost;
207 path_cost[5] <= ACS101_path_cost;
208 path_cost[6] <= ACS110_path_cost;
209 path_cost[7] <= ACS111_path_cost;
210
Ln 180 Col 1 SystemVerilog HDL File 100% 00:00:36
```



```

File Edit View Project Processing Tools Window Help
Search altera.com

139 ACS AC5001(validity[0], validity[1], bmc001_path_0_bmc, bmc001_path_1_bmc, path_cost[0], path_cost[1], AC5001_selection, AC5001_valid_o, AC5001_path_cost);
140 ACS AC5010(validity[2], validity[3], bmc010_path_0_bmc, bmc010_path_1_bmc, path_cost[2], path_cost[3], AC5010_selection, AC5010_valid_o, AC5010_path_cost);
141 ACS AC5011(validity[4], validity[5], bmc011_path_0_bmc, bmc011_path_1_bmc, path_cost[4], path_cost[5], AC5011_selection, AC5011_valid_o, AC5011_path_cost);
142 ACS AC5100(validity[6], validity[7], bmc100_path_0_bmc, bmc100_path_1_bmc, path_cost[6], path_cost[7], AC5100_selection, AC5100_valid_o, AC5100_path_cost);
143 ACS AC5101(validity[8], validity[9], bmc101_path_0_bmc, bmc101_path_1_bmc, path_cost[8], path_cost[9], AC5101_selection, AC5101_valid_o, AC5101_path_cost);
144 ACS AC5110(validity[10], validity[11], bmc110_path_0_bmc, bmc110_path_1_bmc, path_cost[10], path_cost[11], AC5110_selection, AC5110_valid_o, AC5110_path_cost);
145 ACS AC5111(validity[12], validity[13], bmc111_path_0_bmc, bmc111_path_1_bmc, path_cost[12], path_cost[13], AC5111_selection, AC5111_valid_o, AC5111_path_cost);
146
147 assign selection_nets = (AC5111_selection, AC5110_selection, AC5101_selection, AC5100_selection,
148 AC5011_selection, AC5010_selection, AC5001_selection, AC5000_selection);
149 assign validity_nets = (AC5111_valid_o, AC5110_valid_o, AC5101_valid_o, AC5100_valid_o,
150 AC5011_valid_o, AC5010_valid_o, AC5001_valid_o, AC5000_valid_o);
151
152
153 always @(posedge clk, negedge rst) begin
154 if(rst) begin
155 validity
156 selection
157 // clear all 8 path costs
158 path_cost[0]
159 path_cost[1]
160 path_cost[2]
161 path_cost[3]
162 path_cost[4]
163 path_cost[5]
164 path_cost[6]
165 path_cost[7]
166
167 end
168 else if(enable) begin
169 validity
170 selection
171 // clear all 8 path costs
172 path_cost[0]
173 path_cost[1]
174 path_cost[2]
175 path_cost[3]
176 path_cost[4]
177 path_cost[5]
178 path_cost[6]
179 path_cost[7]
180
181 end
182 else if(path_cost[0][7] && path_cost[1][7] && path_cost[2][7] && path_cost[3][7] &&
183 path_cost[4][7] && path_cost[5][7] && path_cost[6][7] && path_cost[7][7])
184 begin
185 validity
186 selection
187
188 path_cost[0]
189 path_cost[1]
190 path_cost[2]
191 path_cost[3]
192 path_cost[4]
193 path_cost[5]
194 path_cost[6]
195 path_cost[7]
196
197 /* likewise for path_cost[1:7] and AC5001:111_path_cost DONE
198 */
199 end
200 else begin
201 validity
202 selection
203
204 path_cost[0]
205 path_cost[1]
206 path_cost[2]
207 path_cost[3]
208 path_cost[4]
209 path_cost[5]
210 path_cost[6]
211 path_cost[7]
212
213 /* likewise for 1:7 DONE
214 */
215 end
216
217
218 always @(posedge clk, negedge rst) begin
219 if(rst)
220 wr_mem_counter <= 10'd0;
221 else if(enable)
222 wr_mem_counter <= 10'd0;
223 else
224 wr_mem_counter <= wr_mem_counter + 10'd1;
225 end
226
227 always @(posedge clk, negedge rst) begin
228 if(rst)
229 rd_mem_counter <= 10'b1111111111; // -1 how do you handle this in 10 bit binary?
230 else if(enable)
231 rd_mem_counter <= rd_mem_counter - 10'd1;
232 end
233
234 always @(posedge clk, negedge rst)
235 if(rst)
236 mem_bank <= 2'b00;
237 else begin
238 if(wr_mem_counter==10'b1111111111)
239 mem_bank <= mem_bank + 2'b01;
240 end
241
242 always @(posedge clk) begin
243 d_in_mem_A <= selection;
244 d_in_mem_B <= selection;
245 d_in_mem_C <= selection;
246 d_in_mem_D <= selection;
247 end
248
249 always @(posedge clk) begin
250 case(mem_bank)
251 2'b01: begin
252 addr_mem_A <= wr_mem_counter;
253 addr_mem_B <= rd_mem_counter;
254 addr_mem_C <= 10'd0;
255 addr_mem_D <= rd_mem_counter;
256
257 wr_mem_A <= 1'b1;
258 wr_mem_B <= 1'b0;
259 wr_mem_C <= 1'b0;
260 wr_mem_D <= 1'b0;
261
262 /* other wr_mems = 0
263 */
264 end
265 2'b10: begin
266 addr_mem_A <= rd_mem_counter;
267 addr_mem_B <= wr_mem_counter;
268 addr_mem_C <= rd_mem_counter;
269 addr_mem_D <= 10'd0;
270
271 wr_mem_A <= 1'b1;
272 wr_mem_B <= 1'b0;
273 wr_mem_C <= 1'b0;
274 wr_mem_D <= 1'b0;
275
276 /* other wr_mems = 0
277 */
278 end
279 2'b10: begin
280 wr_mem_A <= 10'd0;

```

```

File Edit View Project Processing Tools Window Help
Search altera.com

208 path_cost[1]
209 path_cost[5]
210 path_cost[0]
211 path_cost[7]
212
213 /* likewise for 1:7 DONE
214 */
215
216
217
218 always @(posedge clk, negedge rst) begin
219 if(rst)
220 wr_mem_counter <= 10'd0;
221 else if(enable)
222 wr_mem_counter <= 10'd0;
223 else
224 wr_mem_counter <= wr_mem_counter + 10'd1;
225 end
226
227 always @(posedge clk, negedge rst) begin
228 if(rst)
229 rd_mem_counter <= 10'b1111111111; // -1 how do you handle this in 10 bit binary?
230 else if(enable)
231 rd_mem_counter <= rd_mem_counter - 10'd1;
232 end
233
234 always @(posedge clk, negedge rst)
235 if(rst)
236 mem_bank <= 2'b00;
237 else begin
238 if(wr_mem_counter==10'b1111111111)
239 mem_bank <= mem_bank + 2'b01;
240 end
241
242 always @(posedge clk) begin
243 d_in_mem_A <= selection;
244 d_in_mem_B <= selection;
245 d_in_mem_C <= selection;
246 d_in_mem_D <= selection;
247 end
248
249 always @(posedge clk) begin
250 case(mem_bank)
251 2'b01: begin
252 addr_mem_A <= wr_mem_counter;
253 addr_mem_B <= rd_mem_counter;
254 addr_mem_C <= 10'd0;
255 addr_mem_D <= rd_mem_counter;
256
257 wr_mem_A <= 1'b1;
258 wr_mem_B <= 1'b0;
259 wr_mem_C <= 1'b0;
260 wr_mem_D <= 1'b0;
261
262 /* other wr_mems = 0
263 */
264 end
265 2'b10: begin
266 addr_mem_A <= rd_mem_counter;
267 addr_mem_B <= wr_mem_counter;
268 addr_mem_C <= rd_mem_counter;
269 addr_mem_D <= 10'd0;
270
271 wr_mem_A <= 1'b1;
272 wr_mem_B <= 1'b0;
273 wr_mem_C <= 1'b0;
274 wr_mem_D <= 1'b0;
275
276 /* other wr_mems = 0
277 */
278 end
279 2'b10: begin
280 wr_mem_A <= 10'd0;

```

```

277   2'b10: begin
278       addr_mem_A <= 10'd0;
279       addr_mem_B <= rd_mem_counter;
280       addr_mem_C <= wr_mem_counter;
281       addr_mem_D <= rd_mem_counter;
282
283       wr_mem_C <= 1'b1;
284       wr_mem_A <= 1'b0;
285       wr_mem_B <= 1'b0;
286       wr_mem_D <= 1'b0;
287   /* other wr_mem = 0
288   */
289   end
290   2'b11: begin
291       addr_mem_A <= rd_mem_counter;
292       addr_mem_B <= 10'd0;
293       addr_mem_C <= rd_mem_counter;
294       addr_mem_D <= wr_mem_counter;
295
296       wr_mem_D <= 1'b1;
297       wr_mem_A <= 1'b0;
298       wr_mem_B <= 1'b0;
299       wr_mem_C <= 1'b0;
300   /* other wr_mem = 0
301   */
302   end
303 endcase
304 end
305
306 //trellis memory module instantiation
307
308 mem trellis_mem_A
309 (
310     .clk,
311     .wr(wr_mem_A),
312     .addr(addr_mem_A),
313     .d_i(d_o_in_mem_A),
314     .d_o(d_o_mem_A)
315 );
316 /* likewise for trellis_mem_B, C, D
317 */
318 mem trellis_mem_B
319 (
320     .clk,
321     .wr(wr_mem_B),
322     .addr(addr_mem_B),
323     .d_i(d_o_in_mem_B),
324     .d_o(d_o_mem_B)
325 );
326 mem trellis_mem_C
327 (
328     .clk,
329     .wr(wr_mem_C),
330     .addr(addr_mem_C),
331     .d_i(d_o_in_mem_C),
332     .d_o(d_o_mem_C)
333 );
334 mem trellis_mem_D
335 (
336     .clk,
337     .wr(wr_mem_D),
338     .addr(addr_mem_D),
339     .d_i(d_o_in_mem_D),
340     .d_o(d_o_mem_D)
341 );
342
343 //Trace back module operation
344 always @(posedge clk)
345     mem_bank_buf <= mem_bank;
346
347

```

```

346     mem_bank_buf <= mem_bank;
347
348     always @(posedge clk)
349         mem_bank_buf <= mem_bank;
350
351     always @(posedge clk, negedge rst)
352     ff1rst
353     enable_tbu_0 <= 1'b0;
354     else begin
355         if(mem_bank_buf==2'b10)
356             enable_tbu_0 <= 1'b1;
357     else
358         enable_tbu_0 <= enable_tbu_0;
359     end
360
361     always @(posedge clk, negedge rst)
362     ff1rst
363     enable_tbu_1 <= 1'b0;
364     else begin
365         if(mem_bank_buf==2'b11)
366             enable_tbu_1 <= 1'b1;
367     else
368         enable_tbu_1 <= enable_tbu_1;
369     end
370
371     always @(posedge clk)
372     case(mem_bank_buf)
373     2'b00: begin
374         d_in_0_tbu_0 <= d_o_mem_D;
375         d_in_1_tbu_0 <= d_o_mem_C;
376
377         d_in_0_tbu_1 <= d_o_mem_C;
378         d_in_1_tbu_1 <= d_o_mem_B;
379
380         selection_tbu_0 <= 1'b0;
381         selection_tbu_1 <= 1'b1;
382     end
383
384     2'b01: begin
385         d_in_0_tbu_0 <= d_o_mem_D;
386         d_in_1_tbu_0 <= d_o_mem_C;
387
388         d_in_0_tbu_1 <= d_o_mem_A;
389         d_in_1_tbu_1 <= d_o_mem_D;
390
391         selection_tbu_0 <= 1'b1;
392         selection_tbu_1 <= 1'b0;
393     end
394
395     2'b10: 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_A;
400         d_in_1_tbu_1 <= d_o_mem_D;
401
402         selection_tbu_0 <= 1'b0;
403         selection_tbu_1 <= 1'b1;
404     end
405
406     2'b11: begin
407         d_in_0_tbu_0 <= d_o_mem_B;
408         d_in_1_tbu_0 <= d_o_mem_A;
409
410         d_in_0_tbu_1 <= d_o_mem_C;
411         d_in_1_tbu_1 <= d_o_mem_B;
412
413         selection_tbu_0 <= 1'b1;
414         selection_tbu_1 <= 1'b0;
415     end
416 endcase
417
418 //Trace back module instantiation

```

```
Text Editor - C:\Users\William\Desktop\New folder (5)\viterbi_tx_n - viterbi_tx_n - [decoder.sv]
File Edit View Project Processing Tools Window Help
Search altera.com

415 //Trace-back modules instantiation
416
417 tbu_tbu_0 (
418     .clk(
419         .rst(
420             .enable(enable_tbu_0),
421             .selection(selection_tbu_0),
422             .d_in_0(d_in_0_tbu_0),
423             .d_in_1(d_in_1_tbu_0),
424             .d_o(d_o_tbu_0),
425             .wr_en(wr_disp_mem_0)
426         );
427     );
428
429 //analogous for tbu_1
430
431 tbu_tbu_1 (
432     .clk(
433         .rst(
434             .enable(enable_tbu_1),
435             .selection(selection_tbu_1),
436             .d_in_0(d_in_0_tbu_1),
437             .d_in_1(d_in_1_tbu_1),
438             .d_o(d_o_tbu_1),
439             .wr_en(wr_disp_mem_1)
440         );
441     );
442 //Display Memory modules instantiation
443 assign d_in_disp_mem_0 = d_o_tbu_0;
444 assign d_in_disp_mem_1 = d_o_tbu_1;
445
446 mem_disp disp_mem_0
447 (
448     .clk(
449         .wr(wr_disp_mem_0),
450         .addr(addr_disp_mem_0),
451         .d_i(d_in_disp_mem_0),
452         .d_o(d_o_disp_mem_0)
453     );
454 //analogous for disp_mem_1
455 //
456 mem_disp disp_mem_1
457 (
458     .clk(
459         .wr(wr_disp_mem_1),
460         .addr(addr_disp_mem_1),
461         .d_i(d_in_disp_mem_1),
462         .d_o(d_o_disp_mem_1)
463     );
464 // display memory module operation
465 always @ (posedge clk)
466     mem_bank_buf_buf_buf <= mem_bank_buf_buf[0];
467
468 always @ (posedge clk)
469     if (first)
470         wr_mem_counter_disp <= 10'b0000000010;
471     else if (enable)
472         wr_mem_counter_disp <= 10'b0000000010;
473     else
474         wr_mem_counter_disp <= wr_mem_counter_disp - 10'd1;
475
476 always @ (posedge clk)
477     if (first)
478         rd_mem_counter_disp <= 10'b1111111101;
479     else if (enable)
480         rd_mem_counter_disp <= 10'b1111111101;
481     else
482         rd_mem_counter_disp <= rd_mem_counter_disp + 10'd1;
483
484 always @ (posedge clk)
485     case (mem_bank_buf_buf_buf)
486         1'b0:
487             begin
488                 addr_disp_mem_0 <= rd_mem_counter_disp;
489                 addr_disp_mem_1 <= wr_mem_counter_disp;
490             end
491         1'b1:
492             begin
493                 addr_disp_mem_0 <= wr_mem_counter_disp;
494                 addr_disp_mem_1 <= rd_mem_counter_disp; //swap rd and wr;
495             end
496         endcase
497
498 always @ (posedge clk) begin
499     mem_bank_buf_buf_buf <= mem_bank_buf_buf_buf;
500     mem_bank_buf_buf_buf_buf <= mem_bank_buf_buf_buf_buf;
501 end
502 //logic 1;
503 always @ (posedge clk) begin
504     //if (i == 1) begin
505     //    d_out <= d_o_disp_mem_1;
506     //end else begin
507     //    d_out <= d_o_disp_mem_0;
508     //end
509     d_out <= mem_bank_buf_buf_buf_buf ? (d_o_disp_mem_1) : (d_o_disp_mem_0);
510 end
511
512 endmodule
```

```
Text Editor - C:\Users\William\Desktop\New folder (5)\viterbi_tx_n - viterbi_tx_n - [decoder.sv]
File Edit View Project Processing Tools Window Help
Search altera.com

444 assign d_in_disp_mem_1 = d_o_tbu_1;
445
446 mem_disp disp_mem_0
447 (
448     .clk(
449         .wr(wr_disp_mem_0),
450         .addr(addr_disp_mem_0),
451         .d_i(d_in_disp_mem_0),
452         .d_o(d_o_disp_mem_0)
453     );
454 //analogous for disp_mem_1
455 //
456 mem_disp disp_mem_1
457 (
458     .clk(
459         .wr(wr_disp_mem_1),
460         .addr(addr_disp_mem_1),
461         .d_i(d_in_disp_mem_1),
462         .d_o(d_o_disp_mem_1)
463     );
464 // display memory module operation
465 always @ (posedge clk)
466     mem_bank_buf_buf_buf <= mem_bank_buf_buf[0];
467
468 always @ (posedge clk)
469     if (first)
470         wr_mem_counter_disp <= 10'b0000000010;
471     else if (enable)
472         wr_mem_counter_disp <= 10'b0000000010;
473     else
474         wr_mem_counter_disp <= wr_mem_counter_disp - 10'd1;
475
476 always @ (posedge clk)
477     if (first)
478         rd_mem_counter_disp <= 10'b1111111101;
479     else if (enable)
480         rd_mem_counter_disp <= 10'b1111111101;
481     else
482         rd_mem_counter_disp <= rd_mem_counter_disp + 10'd1;
483
484 always @ (posedge clk)
485     case (mem_bank_buf_buf_buf)
486         1'b0:
487             begin
488                 addr_disp_mem_0 <= rd_mem_counter_disp;
489                 addr_disp_mem_1 <= wr_mem_counter_disp;
490             end
491         1'b1:
492             begin
493                 addr_disp_mem_0 <= wr_mem_counter_disp;
494                 addr_disp_mem_1 <= rd_mem_counter_disp; //swap rd and wr;
495             end
496         endcase
497
498 always @ (posedge clk) begin
499     mem_bank_buf_buf_buf <= mem_bank_buf_buf_buf;
500     mem_bank_buf_buf_buf_buf <= mem_bank_buf_buf_buf_buf;
501 end
502 //logic 1;
503 always @ (posedge clk) begin
504     //if (i == 1) begin
505     //    d_out <= d_o_disp_mem_1;
506     //end else begin
507     //    d_out <= d_o_disp_mem_0;
508     //end
509     d_out <= mem_bank_buf_buf_buf_buf ? (d_o_disp_mem_1) : (d_o_disp_mem_0);
510 end
511
512 endmodule
```

BMC Files

BMC 000

```
1  module bmc000                      // branch metric computation
2  ⊞ (
3      input    [1:0] rx_pair,
4      output   [1:0] path_0_bmc,
5      output   [1:0] path_1_bmc);
6
7      wire tmp00, tmp01, tmp10, tmp11;
8
9
10     assign tmp00 = rx_pair[0];
11     assign tmp01 = rx_pair[1];
12
13     assign tmp10 = ~tmp00;
14     assign tmp11 = ~tmp01;
15
16     assign path_0_bmc[0] = tmp00 ^ tmp01;
17     assign path_0_bmc[1] = tmp00 & tmp01;
18
19     assign path_1_bmc[0] = tmp10 ^ tmp11;
20     assign path_1_bmc[1] = tmp10 & tmp11;
21 endmodule
22
```

BMC 001

```
1  module bmc001
2  ⊞ (
3      input    [1:0] rx_pair,
4      output   [1:0] path_0_bmc,
5      output   [1:0] path_1_bmc);
6
7
8      wire tmp00, tmp01, tmp10, tmp11;
9
10
11     assign tmp00 = rx_pair[0];
12     assign tmp01 = ~rx_pair[1];
13
14     assign tmp10 = ~tmp00;
15     assign tmp11 = ~tmp01;
16
17     assign path_0_bmc[0] = tmp00 ^ tmp01;
18     assign path_0_bmc[1] = tmp00 & tmp01;
19
20     assign path_1_bmc[0] = tmp10 ^ tmp11;
21     assign path_1_bmc[1] = tmp10 & tmp11;
22
23
24 endmodule
25
```

BMC 010

```
1 module bmc010
2   (
3     input    [1:0] rx_pair,
4     output   [1:0] path_0_bmc,
5     output   [1:0] path_1_bmc);
6
7   wire tmp00, tmp01, tmp10, tmp11;
8
9
10  assign tmp00 = rx_pair[0];
11  assign tmp01 = ~rx_pair[1];
12
13  assign tmp10 = ~tmp00;
14  assign tmp11 = ~tmp01;
15
16  assign path_0_bmc[0] = tmp00 ^ tmp01;
17  assign path_0_bmc[1] = tmp00 & tmp01;
18
19  assign path_1_bmc[0] = tmp10 ^ tmp11;
20  assign path_1_bmc[1] = tmp10 & tmp11;
21 endmodule
22
```

BMC011

```
1 module bmc011
2   (
3     input    [1:0] rx_pair,
4     output   [1:0] path_0_bmc,
5     output   [1:0] path_1_bmc);
6
7   wire tmp00, tmp01, tmp10, tmp11;
8
9
10  assign tmp00 = rx_pair[0];
11  assign tmp01 = rx_pair[1];
12
13  assign tmp10 = ~tmp00;
14  assign tmp11 = ~tmp01;
15
16  assign path_0_bmc[0] = tmp00 ^ tmp01;
17  assign path_0_bmc[1] = tmp00 & tmp01;
18
19  assign path_1_bmc[0] = tmp10 ^ tmp11;
20  assign path_1_bmc[1] = tmp10 & tmp11;
21 endmodule
22
```

BMC100

```
1 module bmc100
2   (
3     input    [1:0] rx_pair,
4     output   [1:0] path_0_bmc,
5     output   [1:0] path_1_bmc);
6
7
8     wire tmp00, tmp01, tmp10, tmp11;
9
10
11     assign tmp00 = rx_pair[0];
12     assign tmp01 = rx_pair[1];
13
14     assign tmp10 = ~tmp00;
15     assign tmp11 = ~tmp01;
16
17     assign path_0_bmc[0] = tmp00 ^ tmp01;
18     assign path_0_bmc[1] = tmp00 & tmp01;
19
20     assign path_1_bmc[0] = tmp10 ^ tmp11;
21     assign path_1_bmc[1] = tmp10 & tmp11;
22 endmodule
23
```

BMC101

```
1 module bmc101
2   (
3     input    [1:0] rx_pair,
4     output   [1:0] path_0_bmc,
5     output   [1:0] path_1_bmc);
6
7
8     wire tmp00, tmp01, tmp10, tmp11;
9
10
11     assign tmp00 = rx_pair[0];
12     assign tmp01 = ~rx_pair[1];
13
14     assign tmp10 = ~tmp00;
15     assign tmp11 = ~tmp01;
16
17     assign path_0_bmc[0] = tmp00 ^ tmp01;
18     assign path_0_bmc[1] = tmp00 & tmp01;
19
20     assign path_1_bmc[0] = tmp10 ^ tmp11;
21     assign path_1_bmc[1] = tmp10 & tmp11;
22 endmodule
```

BMC110

```
1 module bmc110
2   (
3     input    [1:0] rx_pair,
4     output   [1:0] path_0_bmc,
5     output   [1:0] path_1_bmc);
6
7
8     wire tmp00, tmp01, tmp10, tmp11;
9
10
11     assign tmp00 = rx_pair[0];
12     assign tmp01 = ~rx_pair[1];
13
14     assign tmp10 = ~tmp00;
15     assign tmp11 = ~tmp01;
16
17     assign path_0_bmc[0] = tmp00 ^ tmp01;
18     assign path_0_bmc[1] = tmp00 & tmp01;
19
20     assign path_1_bmc[0] = tmp10 ^ tmp11;
21     assign path_1_bmc[1] = tmp10 & tmp11;
22 endmodule
23
```

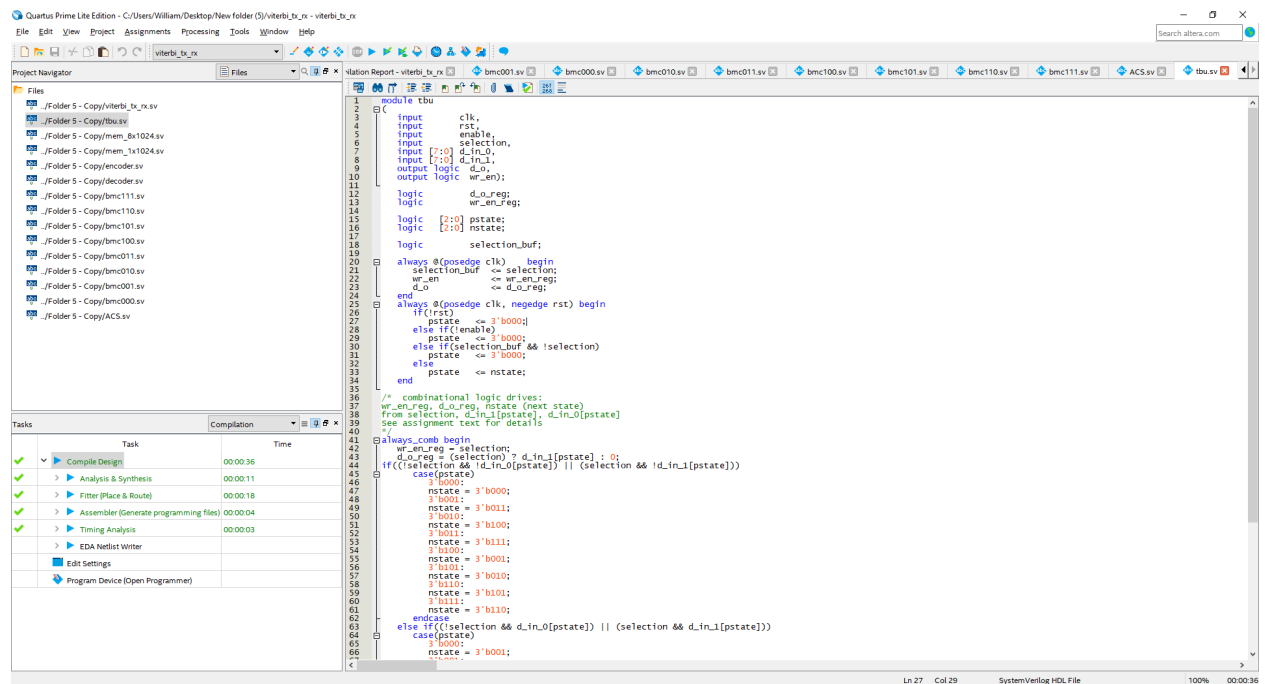
BMC111

```
1 module bmc111 (
2   input    [1:0] rx_pair,
3   output   [1:0] path_0_bmc,
4   output   [1:0] path_1_bmc);
5
6
7     wire tmp00, tmp01, tmp10, tmp11;
8
9
10     assign tmp00 = rx_pair[0];
11     assign tmp01 = rx_pair[1];
12
13     assign tmp10 = ~tmp00;
14     assign tmp11 = ~tmp01;
15
16     assign path_0_bmc[0] = tmp00 ^ tmp01;
17     assign path_0_bmc[1] = tmp00 & tmp01;
18
19     assign path_1_bmc[0] = tmp10 ^ tmp11;
20     assign path_1_bmc[1] = tmp10 & tmp11;
21 endmodule
22
```

ACS

```
1  module ACS                                     // add-compare-select
2  (
3      input      path_0_valid,
4      input      path_1_valid,
5      input [1:0] path_0_bmc,                    // branch metric computation
6      input [1:0] path_1_bmc,
7      input [7:0] path_0_pmc,                    // path metric computation
8      input [7:0] path_1_pmc,
9
10     output logic      selection,
11     output logic      valid_o,
12     output [7:0] path_cost);
13
14     logic [7:0] path_cost_0;                    // branch metric + path metric
15     logic [7:0] path_cost_1;
16
17
18     assign path_cost = (valid_o?(selection?path_cost_1:path_cost_0):8'd0);
19     assign path_cost_0 = path_0_bmc + path_0_pmc;
20     assign path_cost_1 = path_1_bmc + path_1_pmc;
21
22     always_comb begin
23         valid_o = (!path_0_valid && !path_1_valid) ? 1'b0 : 1'b1;
24
25         if(path_0_valid == 0 && path_1_valid == 0) begin
26             selection = 0;
27             valid_o = 0;
28
29         end
30
31         else if(path_0_valid == 0 && path_1_valid == 1) begin
32             selection = 1;
33             valid_o = 1;
34
35         end
36
37         else if(path_0_valid == 1 && path_1_valid == 0) begin
38             selection = 0;
39             valid_o = 1;
40
41         end
42
43         else if(path_cost_0 > path_cost_1) begin
44             selection = 1;
45             valid_o = 1;
46
47         end
48
49         else begin
50             selection = 0;
51             valid_o = 1;
52
53         end
54
55     end
56 end
57 endmodule
58
59
```


TBU

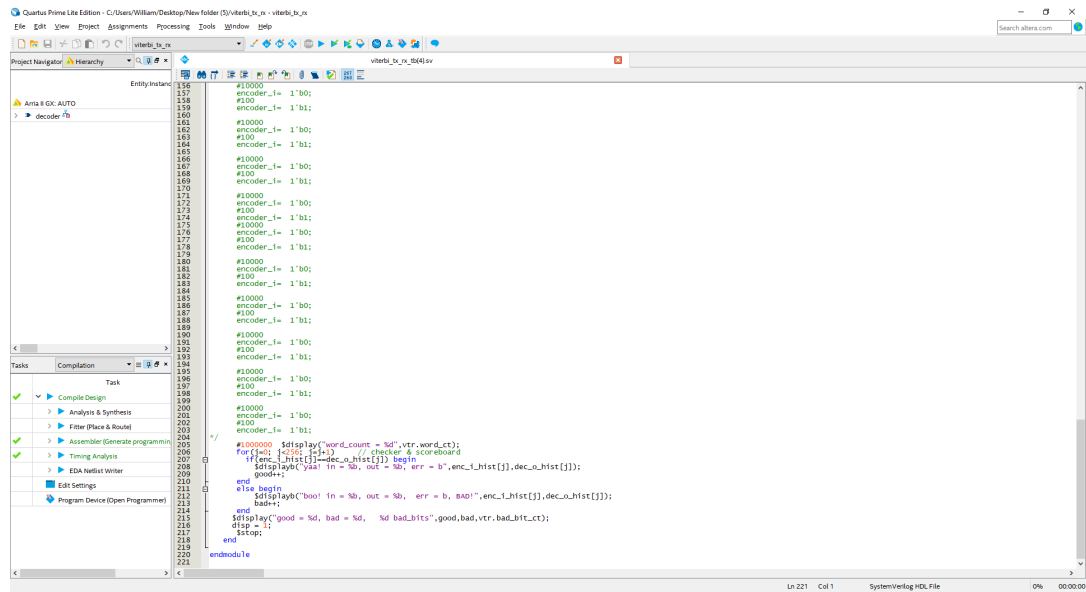


```

63 else if(!selection && d_in_0[pstate]) || (selection && d_in_i[pstate]))
64     case(pstate)
65         3'b000:
66             nstate = 3'b001;
67         3'b001:
68             nstate = 3'b010;
69         3'b010:
70             nstate = 3'b101;
71         3'b011:
72             nstate = 3'b110;
73         3'b100:
74             nstate = 3'b000;
75         3'b101:
76             nstate = 3'b011;
77         3'b110:
78             nstate = 3'b100;
79         3'b111:
80             nstate = 3'b111;
81     endcase
82
83     else
84         nstate = pstate;
85     end
86 endmodule

```

Testbench:

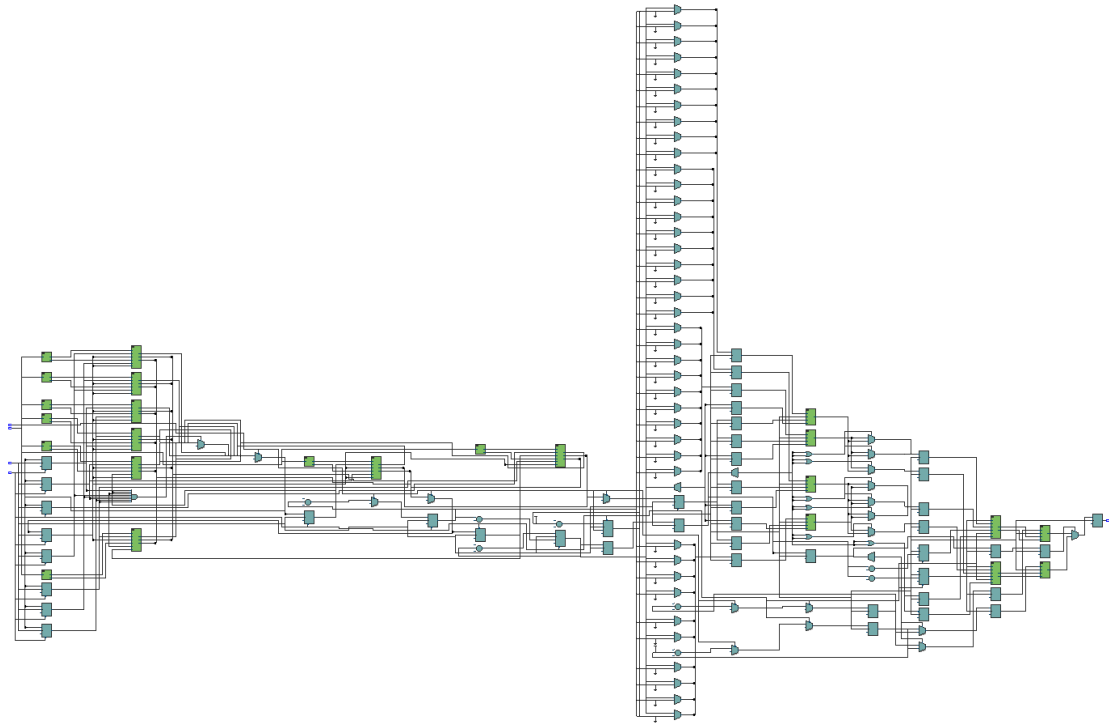


Simulation Transcript

```
# yaa! in = 1, out = 1, err = b
# good = 256, bad = 0, 0 bad_bits
** Note: $stop : C:/Users/William/Desktop/New folder (7)/viterbi_tx_rx_tb(4).sv(217)
# Time: 1045 ns Iteration: 0 Instance: /viterbi_tx_rx_tb
# Break in Module viterbi_tx_rx_tb at C:/Users/William/Desktop/New folder (7)/viterbi_tx_rx_tb(4).sv line 217
VSIM 3>
```

Proof of synthesis

RTL Viewer



Transcript Log

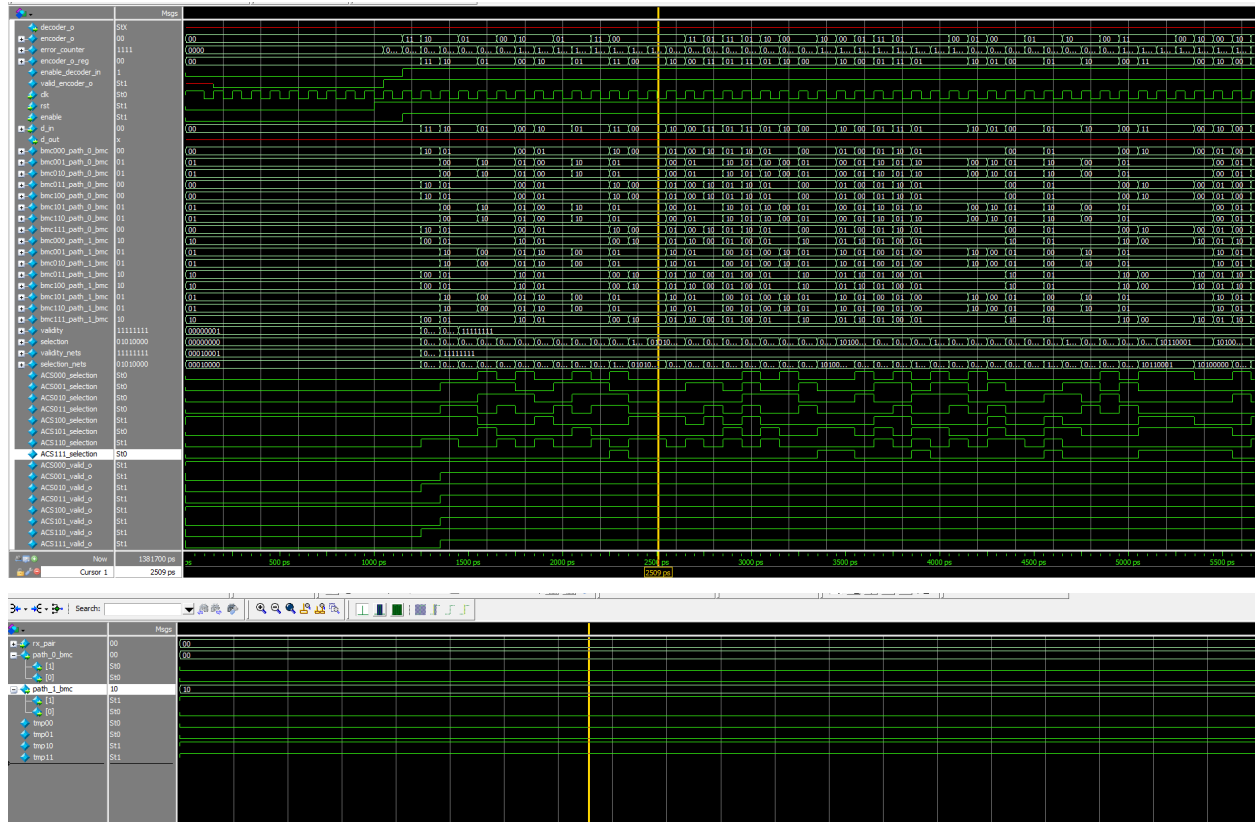
```
# Reading pref.tcl
# Loading project Part 2
# Compile of ACS.sv was successful.
# Compile of bmc000.sv was successful.
# Compile of bmc001.sv was successful.
# Compile of bmc010.sv was successful.
# Compile of bmc011.sv was successful.
# Compile of bmc100.sv was successful.
# Compile of bmc101.sv was successful.
# Compile of bmc110.sv was successful.
# Compile of bmc111.sv was successful.
# Compile of decoder.sv was successful.
# Compile of encoder.sv was successful.
# Compile of mem_1x1024.sv was successful.
# Compile of mem_8x1024.sv was successful.
# Compile of tbu.sv was successful.
# Compile of viterbi_tx_rx_tb(4).sv was successful.
# Compile of viterbi_tx_rx1(1).sv was successful.
# 16 compiles, 0 failed with no errors.
# Compile of ACS.sv was successful.
# Compile of bmc000.sv was successful.
# Compile of bmc001.sv was successful.
# Compile of bmc010.sv was successful.
# Compile of bmc011.sv was successful.
# Compile of bmc100.sv was successful.
# Compile of bmc101.sv was successful.
# Compile of bmc110.sv was successful.
# Compile of bmc111.sv was successful.
# Compile of decoder.sv was successful.
# Compile of encoder.sv was successful.
# Compile of mem_1x1024.sv was successful.
# Compile of mem_8x1024.sv was successful.
# Compile of tbu.sv was successful.
# Compile of viterbi_tx_rx_tb(4).sv was successful.
# Compile of viterbi_tx_rx1(1).sv was successful.
# 16 compiles, 0 failed with no errors.
ModelSim> vsim -gui work.viterbi_tx_rx_tb
# vsim -gui work.viterbi_tx_rx_tb
# Start time: 16:18:31 on Mar 22, 2024
# Loading sv_std.std
# Loading work.viterbi_tx_rx_tb
# Loading work.viterbi_tx_rx
# Loading work.encoder
# Loading work.decoder
# Loading work.bmc000
# Loading work.bmc001
# Loading work.bmc010
# Loading work.bmc011
# Loading work.bmc100
# Loading work.bmc101
# Loading work.bmc110
# Loading work.bmc111
# Loading work.ACS
# Loading work.mem
# Loading work.tbu
# Loading work.mem_disp
# WARNING: No extended dataflow license exists
VSIM 2> run -all
# word count = 10440
```

Encoder explanation

In the term project encoder file, the encoded output, `d_out`, is determined by the current input, `d_in`. The current state, `cstate`, and the next state are 3 bits wide and the next state is dependent on the `d_in`. In the homework 7 file, there is a different structure where it is a little more flexible. With the use of two different mask patterns, `mask0` and `mask1`, and the inclusion of the shift register, the output will produce more encrypted codes based on the loaded mask. Unlike the encoder file, the `data_out` will be controlled by the two mask patterns instead of just the input. The main reason why the decoder does not work with the homework 7 encoder is because the Viterbi decoder is created to work with specific convolutional code. As we mentioned before, the homework 7 encoder can create a variety of convolutional codes and that makes the decoder unable to process it. Whereas, the `encoder.sv` file is a fixed encoder which is tailored to this specific decoder. The state transition logic is matched and for the decoder to process. To elaborate further on these codes, the homework file is also non-systematic and non-recursive because there is no feedback output in the shift registers. Additionally, the shift register does not copy the input bits to the output bits that come from the XOR operation and position the shift register uniquely. The encoder file is systematic and recursive because it has feedback and due to `d_out_reg[0] = d_in` proves that there is functionality behind that. This is important because systematic codes are easier for decoders to handle because the input data has a correlation with the original data whereas non systematic codes have no direct reference to the actual data bit.

Decoder explanation

The decoder receives the encoded data from the encoder and will begin the path metric calculation within the BMC that determines the path cost of each state. Next the ACS, Add-Compare-Select, will add each possible transition into the states and compare the metrics of the two paths of which one has a higher chance of a lower cost which is dependent on the amount of error. The ACS also selects the better path, lower path cost, and sets the path as a more cost effective path. In the trellis memory module, it represents the survivor path selection that will determine the most cost effective path leading to each state. Throughout this process, the decoder eliminates other less likely paths and continuously updates the path metrics which will prepare the best route for the traceback unit. In the traceback unit, the sequence should be entirely processed in which the decoder should find the best ending state with the least path cost. The Trellis module should already have highlighted the most likely path and reconstructed the original data sequence. Finally, the `d_out` should output the most likely path of the unencoded data bit.



When no errors present, the minimum branch metric = 0 due to the hamming distance of the input. In BMC, the input, rx_pair, is set to the temp variables and assigned to either path 0 or path 1 which is then ANDed and XORed. You can see that if the BMC output on a path was 00 that meant most likely that it had the least path cost which would point out to the decoder that it would be the most likely path.

There are various steps on how the decoder determines the sequence of bits that was sent from the encoder. The branch metric, BMC, computes the transition between the states at each clock cycle. Each BMC will measure the difference between the received encoder output, d_in. In ACS, it updates the path's metric cost and points out which is more optimal. The ACS is able to point out the optimal path by comparing the path_cost_0 or path_cost_1 and selection will choose one or the other by assigning itself to 1 or 0. Once the entire sequence has been processed the traceback unit tries to reconstruct the original sequence from each nstate by setting a temp variable pstate to either the nstate or 0 when reset is 0, enable is 0, or when selection_buf && !selection passes through. This traceback unit is hard coded as depending on the whether the selection and pstate, it outputs a specific nstate. The nstate value will be outputted dependent on the memory module operation that will decide which nstate path is correct.

Part 2: Injecting the bad bits

Disclaimer: We used a different testbench from the one that was provided because we worked on it earlier. All data and results were confirmed with the TA and it worked just as expected.

Testbench:

```
1 // test bench
2 module viterbi_tx_rx_tb();
3   bit clk;
4   bit rst;
5   bit encoder_i;
6   bit enc_i_hist[2048]; // original data
7   bit enable_encoder_i; // history thereof
8   wire decoder_o;
9   bit dec_o_hist[2048]; // decoded data
10  bit disp;
11  int good, bad; // scoreboard
12
13 // this module contains conv. encode and vit decode
14 viterbi_tx_rx vtr(
15   .clk,
16   .rst,
17   .encoder_i, // original data
18   .enable_encoder_i, // history thereof
19   .decoder_o); // decoded data
20
21 always begin
22   #50 clk = 1;
23   #50 clk = 0;
24 end
25 int i, j, k, l;
26
27 always @(posedge clk) begin
28   enc_i_hist[i] <= encoder_i;
29   i <= i+1;
30   l <= l+1;
31 end
32
33 initial begin
34   #410500; // #410400;
35   forever @(posedge clk) begin
36     dec_o_hist[k] <= decoder_o;
37     k <= k+1;
38   end
39 end
40
41 initial begin
42   clk = 1'b1;
43   #1000
44   rst = 1'b1;
45   enable_encoder_i = 1'b1;
46   #100
47   encoder_i = 1'b1;
48   #100
49   encoder_i = 1'b0;
50   #100
51   encoder_i = 1'b0;
52   #100
53   encoder_i = 1'b1;
54   #100
55   encoder_i = 1'b1;
56   #100
57   encoder_i = 1'b0;
58   #100
59   encoder_i = 1'b0;
60   #100
61   encoder_i = 1'b0;
62   #100
63   encoder_i = 1'b1;
64   #100
65   encoder_i = 1'b1;
66   encoder_i = 1'b1;
67   encoder_i = 1'b1;
68   #100
69   encoder_i = 1'b0;
70   #100
71   encoder_i = 1'b0;
72   #100
73   encoder_i = 1'b0;
74   #100
75   encoder_i = 1'b0;
76   #100
77   encoder_i = 1'b1;
78   #100
79   encoder_i = 1'b1;
80   #100
81   encoder_i = 1'b1;
82   #100
83   encoder_i = 1'b1;
84   #100
85   encoder_i = 1'b0;
86   #100
87   encoder_i = 1'b0;
88   #100
89   encoder_i = 1'b0;
90   #100
91   encoder_i = 1'b0;
92   #100
93   encoder_i = 1'b0;
94   #100
95   encoder_i = 1'b1;
96   #100
97   encoder_i = 1'b1;
98   #100
99   encoder_i = 1'b1;
100  #100
101  encoder_i = 1'b1;
102  #100
103  encoder_i = 1'b1;
104  #100
105  encoder_i = 1'b0;
106  #100
107  encoder_i = 1'b1;
108  #100
109  encoder_i = 1'b0;
110  #100
111  encoder_i = 1'b0;
112  #100
113  encoder_i = 1'b1;
114  #100
115  encoder_i = 1'b1;
116  #100
117  encoder_i = 1'b0;
118  #100
119  encoder_i = 1'b0;
120  #100
121  encoder_i = 1'b0;
122  #100
123  encoder_i = 1'b1;
124  #100
125  encoder_i = 1'b1;
126  #100
127  encoder_i = 1'b1;
128  #100
129  encoder_i = 1'b0;
130  #100
131  encoder_i = 1'b0;
```



```
131 encoder_i= 1'b0;
132 #100
133 encoder_i= 1'b0;
134 #100
135 encoder_i= 1'b0;
136 #100
137 encoder_i= 1'b1;
138 #100
139 encoder_i= 1'b1;
140 #100
141 encoder_i= 1'b1;
142 #100
143 encoder_i= 1'b1;
144 #100
145 encoder_i= 1'b0;
146 #100
147 encoder_i= 1'b0;
148 #100
149 encoder_i= 1'b0;
150 #100
151 encoder_i= 1'b0;
152 #100
153 encoder_i= 1'b0;
154 #100
155 encoder_i= 1'b1;
156 #100
157 encoder_i= 1'b1;
158 #100
159 encoder_i= 1'b1;
160 #100
161 encoder_i= 1'b1;
162 #100
163 encoder_i= 1'b1;
164
165 #10000
166 encoder_i= 1'b0;
167 #100
168 encoder_i= 1'b1;
169 #10000
170 encoder_i= 1'b0;
171 #100
172 encoder_i= 1'b1;
173
174 #10000
175 encoder_i= 1'b0;
176 #100
177 encoder_i= 1'b1;
178
179
180 #10000
181 encoder_i= 1'b0;
182 #100
183 encoder_i= 1'b1;
184
185 #10000
186 encoder_i= 1'b0;
187 #100
188 encoder_i= 1'b1;
189
190 #10000
191 encoder_i= 1'b0;
192 #100
193 encoder_i= 1'b1;
194
195 #10000
196 encoder_i= 1'b0;
```

```
196 encoder_i= 1'b0;
197 #100
198 encoder_i= 1'b1;
199
200
201 #10000
202 encoder_i= 1'b0;
203 #100
204 encoder_i= 1'b1;
205
206 #10000
207 encoder_i= 1'b0;
208 #100
209 encoder_i= 1'b1;
210
211 #10000
212 encoder_i= 1'b0;
213 #100
214 encoder_i= 1'b1;
215
216 #10000
217 encoder_i= 1'b0;
218 #100
219 encoder_i= 1'b1;
220
221
222 #10000
223 encoder_i= 1'b0;
224 #100
225 encoder_i= 1'b1;
226
227 #10000
228 encoder_i= 1'b0;
229 #100
230 encoder_i= 1'b1;
231
232 #10000
233 encoder_i= 1'b0;
234 #100
235 encoder_i= 1'b1;
236
237 #10000
238 encoder_i= 1'b0;
239 #100
240 encoder_i= 1'b1;
241
242 #10000
243 encoder_i= 1'b0;
244 #100
245 encoder_i= 1'b1;
246
247
248 #10000
249 encoder_i= 1'b0;
250 #100
251 encoder_i= 1'b1;
252
253 #10000
254 encoder_i= 1'b0;
255 #100
256 encoder_i= 1'b1;
257
258 #10000
259 encoder_i= 1'b0;
260 #100
261 encoder_i= 1'b1;
```

```

261 encoder_i= 1'b1;
262
263 #10000
264 encoder_i= 1'b0;
265 #100
266 encoder_i= 1'b1;
267
268 #10000
269 encoder_i= 1'b0;
270 #100
271 encoder_i= 1'b1;
272
273 #10000
274 encoder_i= 1'b0;
275 #100
276 encoder_i= 1'b1;
277
278 #10000
279 encoder_i= 1'b0;
280 #100
281 encoder_i= 1'b1;
282
283 #10000
284 encoder_i= 1'b0;
285 #100
286 encoder_i= 1'b1;
287
288 #10000
289 encoder_i= 1'b0;
290 #100
291 encoder_i= 1'b1;
292
293 #10000
294 encoder_i= 1'b0;
295 #100
296 encoder_i= 1'b1;
297
298
299 #10000
300 encoder_i= 1'b0;
301 #100
302 encoder_i= 1'b1;
303
304 #10000
305 encoder_i= 1'b0;
306 #100
307 encoder_i= 1'b1;
308
309 #10000
310 encoder_i= 1'b0;
311 #100
312 encoder_i= 1'b1;
313
314 #10000
315 encoder_i= 1'b0;
316 #100
317 encoder_i= 1'b1;
318
319 #10000
320 encoder_i= 1'b0;
321 #100
322 encoder_i= 1'b1;
323
324 #10000
325 encoder_i= 1'b0;
326 #100
327 . . . . .
328
329 #100
330 encoder_i= 1'b1;
331 #10000
332 encoder_i= 1'b0;
333 #100
334 encoder_i= 1'b1;
335
336 #10000
337 encoder_i= 1'b0;
338 #100
339 encoder_i= 1'b1;
340
341 #10000
342 encoder_i= 1'b0;
343 #100
344 encoder_i= 1'b1;
345
346 #10000
347 encoder_i= 1'b0;
348 #100
349 encoder_i= 1'b1;
350
351 #10000
352 encoder_i= 1'b0;
353 #100
354 encoder_i= 1'b1;
355
356 #10000
357 encoder_i= 1'b0;
358 #100
359 encoder_i= 1'b1;
360
361 #1000000
362 for(j=0; j<256; j=j+1) // checker & scoreboard
363   if(enc_i_hist[j]==dec_o_hist[j]) begin
364     $displayb("Yaa! in = %b, out = %b",enc_i_hist[j],dec_o_hist[j]);
365     good++;
366   end
367   else begin
368     $displayb("boo! in = %b, out = %b",enc_i_hist[j],dec_o_hist[j]);
369     bad++;
370   end
371   $display("good = %d, bad = %d",good,bad);
372   disp = 1;
373   $stop;
374 end
375 endmodule

```

Part A

2.a.1

```
1 module viterbi_tx_rx(
2     input clk,
3     input rst,
4     input encoder_i,
5     input enable_encoder_i,
6     output decoder_o);
7
8     wire [1:0] encoder_o;
9
10    logic [3:0] error_counter;
11    logic [1:0] encoder_o_reg;
12
13    logic enable_decoder_in;
14    wire valid_encoder_o;
15
16    //Logic for error for bit 0 and 1
17    logic [2:0] error_bit8;
18    logic [4:0] error_bit32;
19    int bad_bit_ct = 1;
20    int word_ct;
21
22
23    always @ (posedge clk, negedge rst)
24    if(!rst) begin
25        error_counter <= 4'd0;
26        encoder_o_reg <= 2'b00;
27        enable_decoder_in <= 1'b0;
28        error_bits <= 3'd0;
29        error_bit32 <= 5'd0;
30        word_ct <= 0;
31    end
32    else begin
33        enable_decoder_in <= valid_encoder_o;
34        encoder_o_reg <= 2'b00;
35        error_counter <= error_counter + 4'd1;
36        error_bits <= error_bits + 3'd1;
37        error_bit32 <= error_bit32 + 5'd1;
38        word_ct <= word_ct + 1;
39
40        if(error_bit8==3'b111)begin //2a1 # good = 256, bad = 0 , Injected : 32
41            encoder_o_reg <= {encoder_o[1],~encoder_o[0]}; // inject one bad bit out of every 8
42            if(word_ct<256)begin
43                bad_bit_ct <= bad_bit_ct + 1;
44                $display("error_counter = %d",bad_bit_ct);
45            end
46        end
47
48        else
49            encoder_o_reg <= {encoder_o[1],encoder_o[0]};
50        end
51    end
52
53    // insert your convolutional encoder here
54    // change port names and module name as necessary/desired
55    encoder_encoder1 (
56        .clk(clk),
57        .rst(rst),
58        .enable_i(enable_encoder_i),
59        .d_in(encoder_i),
60        .valid_o(valid_encoder_o),
61        .d_out(decoder_o)
62    );
63 endmodule
```

```
# error_counter = 30
# error_counter = 31
# error_counter = 32
```

```
- # good = 256, bad = 0
# ** Note: $stop : H:/Folder 5 - Copy/viterbi_tx_rx_tb.sv(371)
# Time: 1381700 ps Iteration: 0 Instance: /viterbi_tx_rx_tb
```

```
61 // insert your term project code here
62 decoder_decoder1 (
63     .clk(clk),
64     .rst(rst),
65     .enable(enable_decoder_in),
66     .d_in(decoder_i),
67     .d_out(decoder_o)
68 );
69
70 endmodule
71
72
73
```

2.a.2

```

1 module viterbi_tx_rx(
2     input  clk,
3     input  rst,
4     input  encoder_i,
5     input  enable_encoder_i,
6     output decoder_o);
7
8     wire [1:0] encoder_o;
9
10    logic [3:0] error_counter;
11    logic [1:0] encoder_o_reg;
12
13    logic enable_decoder_in;
14    wire valid_encoder_o;
15
16    //Logic for error for bit 0 and 1
17    logic [2:0] error_bit8;
18    logic [4:0] error_bit32;
19    int bad_bit_ct = 1;
20    int word_ct;
21
22    always @(posedge clk, negedge rst)
23    begin
24        if(!rst) begin
25            error_counter <= 4'd0;
26            encoder_o_reg <= 2'b00;
27            enable_decoder_in <= 1'b0;
28            error_bit8 <= 3'd0;
29            error_bit32 <= 5'd0;
30            word_ct = 0;
31        end
32        else begin
33            enable_decoder_in <= valid_encoder_o;
34            encoder_o_reg <= 2'b00;
35            error_counter <= error_counter + 4'd1;
36            error_bit8 <= error_bit8 + 3'd1;
37            error_bit32 <= error_bit32 + 5'd1;
38            word_ct <= word_ct + 1;
39
40            if(error_bit8==3'b111)begin //2a2 # good = 256, bad = 0, Injected : 32
41                encoder_o_reg <= {1-encoder_o[1],encoder_o[0]}; // inject one bad bit out of every 8
42                if(word_ct<256)begin
43                    bad_bit_ct <= bad_bit_ct + 1;
44                    $display("error_counter = %d",bad_bit_ct);
45                end
46            end
47            else
48                encoder_o_reg <= {encoder_o[1],encoder_o[0]};
49        end
50    end
51
52    //insert your convolutional encoder here
53    //change port names and module name as necessary/desired
54    encoder encoder1 (
55        .clk(clk),
56        .rst(rst),
57        .enable_i(enable_encoder_i),
58        .d_in(encoder_i),
59        .valid_o(valid_encoder_o),
60        .d_out(encoder_o) );
61
62    //insert your term project code here
63    decoder decoder1 (
64        .clk(clk),
65        .rst(rst),
66        .enable(enable_decoder_in),
67        .d_in(encoder_o_reg),
68        .d_out(decoder_o) );
69
70 endmodule
71
72
73
74
75

```

```

# error_counter = 30
# error_counter = 31
# error_counter = 32

```

```

# yaa! in = 1, out = 1
# good = 256, bad = 0
# ** Note: $stop : H:/Folder 5 - Copy/viterbi_tx_rx_tb.sv(371)
# Time: 1381700 ps Iteration: 0 Instance: /viterbi_tx_rx_tb
# Break in Module viterbi_tx_rx_tb at H:/Folder 5 - Copy/viterbi_tx_rx_tb.sv line 371

```

2.a.3

```

1 module viterbi_tx_rx(
2     input  clk,
3     input  rst,
4     input  encoder_i,
5     input  enable_encoder_i,
6     output decoder_o);
7
8     wire [1:0] encoder_o;
9
10    logic [3:0] error_counter;
11    logic [1:0] encoder_o_reg;
12
13    logic enable_decoder_in;
14    wire valid_encoder_o;
15
16    //Logic for error for bit 0 and 1
17    logic [2:0] error_bit8;
18    logic [4:0] error_bit32;
19    int bad_bit_ct = 2;
20    int word_ct;
21
22    always @(posedge clk, negedge rst)
23    begin
24        if(!rst) begin
25            error_counter <= 4'd0;
26            encoder_o_reg <= 2'b00;
27            enable_decoder_in <= 1'b0;
28            error_bit8 <= 3'd0;
29            error_bit32 <= 5'd0;
30            word_ct = 0;
31        end
32        else begin
33            enable_decoder_in <= valid_encoder_o;
34            encoder_o_reg <= 2'b00;
35            error_counter <= error_counter + 4'd1;
36            error_bit8 <= error_bit8 + 3'd1;
37            error_bit32 <= error_bit32 + 5'd1;
38            word_ct <= word_ct + 1;
39
40            if(error_bit8==3'b111)begin //2a3 # good = 205, bad = 51, Injected = 64
41                encoder_o_reg <= {encoder_o[1],encoder_o[0]}; // inject one bad bit for 1 and 0 out of every 8
42                if(word_ct<256)begin
43                    bad_bit_ct <= bad_bit_ct + 2;
44                    $display("error_counter = %d",bad_bit_ct);
45                end
46            end
47            else
48                encoder_o_reg <= {encoder_o[1],encoder_o[0]};
49            end
50        end
51
52        // insert your convolutional encoder here
53        // change port names and module name as necessary/desired
54        encoder encoder1 (
55            .clk(clk),
56            .rst(rst),
57            .enable_i(enable_encoder_i),
58            .d_in(encoder_i),
59            .valid_o(valid_encoder_o),
60            .d_out(decoder_o));
61    end
62
63    // insert your term project code here
64    decoder decoder1 (
65        .clk(clk),
66        .rst(rst),
67        .enable(enable_decoder_in),
68        .d_in(encoder_o_reg),
69        .d_out(decoder_o));
70
71 endmodule
72
73
74
75

```

```

# error_counter = 58
# error_counter = 60
# error_counter = 62
# error_counter = 64
# yaa! in = 0, out = 0
# vaa! in = 0. out = 0

```

```

# boo! in = 1, out = 0
# good = 205, bad = 51
# ** Note: $stop : H:/Folder 5 - Copy/viterbi_tx_rx_tb.sv(371)
# Time: 1381700 ps Iteration: 0 Instance: /viterbi_tx_rx_tb
# Break in Module viterbi_tx_rx_tb at H:/Folder 5 - Copy/viterbi_tx_rx_tb.sv line 371

```

2.a.4

```

1 module viterbi_tx_rx(
2     input  clk,
3     input  rst,
4     input  encoder_i,
5     input  enable_encoder_i,
6     output decoder_o);
7
8     wire [1:0] encoder_o;
9
10    logic [3:0] error_counter;
11    logic [1:0] encoder_o_reg;
12
13    logic enable_decoder_in;
14    wire  valid_encoder_o;
15
16    //Logic for error for bit 0 and 1
17    logic [2:0] error_bits;
18    logic [4:0] error_bit32;
19    int bad_bit_ct = 1;
20    int word_ct;
21
22
23    always @(posedge clk, negedge rst)
24    if(!rst) begin
25        error_counter <= 4'd0;
26        encoder_o_reg <= 2'b00;
27        enable_decoder_in <= 1'b0;
28        error_bits <= 3'd0;
29        error_bit32 <= 5'd0;
30        word_ct <= 0;
31    end
32    else begin
33        enable_decoder_in <= valid_encoder_o;
34        encoder_o_reg <= 2'b00;
35        error_counter <= error_counter + 4'd1;
36        error_bits <= error_bits + 3'd1;
37        error_bit32 <= error_bit32 + 5'd1;
38        word_ct <= word_ct + 1;
39    end
40
41    if(error_counter == 4'b1001 || error_counter == 4'b1000) begin //244 # good = 256, bad = 0, Injected : 32
42        encoder_o_reg <= {encoder_o[1],~encoder_o[0]}; // inject two bad bit out of every 16
43        if(word_ct<256)begin
44            bad_bit_ct <= bad_bit_ct + 1;
45            $display("error_counter = %d",bad_bit_ct);
46        end
47    end
48    else
49        encoder_o_reg <= {encoder_o[1],encoder_o[0]};
50    end
51
52    // Insert your convolutional encoder here
53    // change port names and module name as necessary/desired
54    encoder_encoder1 (
55        .clk(clk),
56        .rst(rst),
57        .enable_i(enable_encoder_i),
58        .d_in(encoder_i),
59        .valid_o(valid_encoder_o),
60        .d_out(encoder_o));
61

```

```

# error_counter =          29
# error_counter =          30
# error_counter =          31
# error_counter =          32
# yaa! in = 0, out = 0
# vaa! in = 0, out = 0
# yaa! in = 1, out = 1
# good =          256, bad =          0
# ** Note: $stop      : H:/Folder 5 - Copy/viterbi_tx_rx_tb.sv(371)
#   Time: 1381700 ps  Iteration: 0  Instance: /viterbi_tx_rx_tb
# Break in Module viterbi_tx_rx_tb at H:/Folder 5 - Copy/viterbi_tx_rx_tb.sv line 371

```

2.a.5

```

1 module viterbi_tx_rx(
2     input clk,
3     input rst,
4     input encoder_i,
5     input enable_encoder_i,
6     output decoder_o);
7
8     wire [1:0] encoder_o;
9
10    logic [3:0] error_counter;
11    logic [1:0] encoder_o_reg;
12
13    logic enable_decoder_in;
14    wire valid_encoder_o;
15
16    //Logic for error for bit 0 and 1
17    logic [2:0] error_bit8;
18    logic [4:0] error_bit32;
19    int bad_bit_ct = 1;
20    int word_ct;
21
22
23    always @(posedge clk, negedge rst)
24    if(!rst) begin
25        error_counter <= 4'd0;
26        encoder_o_reg <= 2'b00;
27        enable_decoder_in <= 1'b0;
28        error_bit8 <= 3'd0;
29        error_bit32 <= 5'd0;
30        word_ct <= 0;
31    end
32    else begin
33        enable_decoder_in <= valid_encoder_o;
34        encoder_o_reg <= 2'b00;
35        error_counter <= error_counter + 4'd1;
36        error_bit8 <= error_bit8 + 3'd1;
37        error_bit32 <= error_bit32 + 5'd1;
38        word_ct <= word_ct + 1;
39
40        if(error_counter == 4'b1000 || error_counter == 4'b1000) begin //2a5 # good = 256, bad = 0, Injected : 32
41            encoder_o_reg <= {~encoder_o[1], encoder_o[0]}; // inject two bad bit out of every 16
42            if(word_ct < 256) begin
43                bad_bit_ct <= bad_bit_ct + 1;
44                $display("error_counter = %d", bad_bit_ct);
45            end
46        end
47        else
48            encoder_o_reg <= {encoder_o[1], encoder_o[0]};
49        end
50    end
51
52    // insert your convolutional encoder here
53    // change port names and module name as necessary/desired
54    encoder_encoder1 (
55        .clk(clk),
56        .rst(rst),
57        .enable_i(enable_encoder_i),
58        .d_in(encoder_i),
59        .valid_o(valid_encoder_o),
60        .d_out(encoder_o));
61
62    // insert your term project code here
63    decoder_decoder1 (
64        .clk(clk),
65        .rst(rst),
66        .enable_i(enable_decoder_in),
67        .d_in(encoder_o_reg),
68        .d_out(decoder_o));
69
70 endmodule
71
72

```

```

# error_counter = 49
# error_counter = 30
# error_counter = 31
# error_counter = 32
# yaa! in = 0, out = 0
# vaa! in = 0. out = 0
# yaa! in = 1, out = 1
# good = 256, bad = 0
# ** Note: $stop : H:/Folder 5 - Copy/viterbi_tx_rx_tb.sv(371)
# Time: 1381700 ps Iteration: 0 Instance: /viterbi_tx_rx_tb
# Break in Module viterbi_tx_rx_tb at H:/Folder 5 - Copy/viterbi_tx_rx_tb.sv line 371

```

2.a.6

```

1 module viterbi_tx_rx(
2     input  clk,
3     input  rst,
4     input  encoder_i,
5     input  enable_encoder_i,
6     output decoder_o);
7
8     wire [1:0] encoder_o;
9
10    logic [3:0] error_counter;
11    logic [1:0] encoder_o_reg;
12
13    logic enable_decoder_in;
14    wire valid_encoder_o;
15
16    //Logic for error for bit 0 and 1
17    logic [2:0] error_bit8;
18    logic [4:0] error_bit32;
19    int bad_bit_ct = 1;
20    int word_ct;
21
22    always @(posedge clk, negedge rst)
23    begin
24        if(!rst) begin
25            error_counter <= 4'd0;
26            encoder_o_reg <= 2'b00;
27            enable_decoder_in <= 1'b0;
28            error_bit8 <= 3'd0;
29            error_bit32 <= 5'd0;
30            word_ct <= 0;
31        end
32        else begin
33            enable_decoder_in <= valid_encoder_o;
34            encoder_o_reg <= 2'b00;
35            error_counter <= error_counter + 4'd1;
36            error_bit8 <= error_bit8 + 3'd1;
37            error_bit32 <= error_bit32 + 5'd1;
38            word_ct <= word_ct + 1;
39
40            if(error_bit32 == 5'b01010 || error_bit32 == 5'b01011 || error_bit32 == 5'b01100 || error_bit32 == 5'b01101) begin //2a6 // # good = 223, bad = 33 Injected :32
41                encoder_o_reg <= {encoder_o[1],~encoder_o[0]}; // inject 4 in a row bad bit out of every 32
42                if(word_ct<256)begin
43                    bad_bit_ct <= bad_bit_ct + 1;
44                    $display("error_counter = %d",bad_bit_ct);
45                end
46            end
47            else
48                encoder_o_reg <= {encoder_o[1],encoder_o[0]};
49        end
50
51        // Insert your convolutional encoder here
52        // change port names and module name as necessary/desired
53        encoder_encoder1 (
54            .clk(clk),
55            .rst(rst),
56            .enable_i(enable_encoder_i),
57            .d_in(encoder_i),
58            .valid_o(valid_encoder_o),
59            .d_out(decoder_o) );
60    end
61
62    // Insert your term project code here
63    decoder_decoder1 (
64        .clk(clk),
65        .rst(rst),
66        .enable(enable_decoder_in),
67        .d_in(encoder_o_reg),
68        .d_out(decoder_o) );
69
70 endmodule
71

```

```

# error_counter =      28
# error_counter =      29
# error_counter =      30
# error_counter =      31
# error_counter =      32

```

```
# yaa! in = 0, out = 0
```

```
# yaa! in = 1, out = 1
```

```
# good =      223, bad =      33
```

```
# ** Note: $stop      : H:/Folder 5 - Copy/viterbi_tx_rx_tb.sv(371)
```

```
# Time: 1381700 ps Iteration: 0 Instance: /viterbi_tx_rx_tb
```

```
# Break in Module viterbi_tx_rx_tb at H:/Folder 5 - Copy/viterbi_tx_rx_tb.sv line 371
```


2.a.7

```

1 module viterbi_tx_rx(
2     input    clk,
3     input    rst,
4     input    encoder_i,
5     input    enable_encoder_i,
6     output   decoder_o);
7
8     wire [1:0] encoder_o;
9
10    logic [3:0] error_counter;
11    logic [1:0] encoder_o_reg;
12
13    logic enable_decoder_in;
14    wire   valid_encoder_o;
15
16    //Logic for error for bit 0 and 1
17    logic [2:0] error_bit8;
18    logic [4:0] error_bit32;
19    int bad_bit_ct = 1;
20    int word_ct;
21
22    always @(posedge clk, negedge rst)
23    if(!rst) begin
24        error_counter <= 4'd0;
25        encoder_o_reg <= 2'b00;
26        enable_decoder_in <= 1'b0;
27        error_bit8 <= 3'd0;
28        error_bit32 <= 5'd0;
29        word_ct = 0;
30    end
31    else begin
32        enable_decoder_in <= valid_encoder_o;
33        encoder_o_reg <= 2'b00;
34        error_counter <= error_counter + 4'd1;
35        error_bit8 <= error_bit8 + 3'd1;
36        error_bit32 <= error_bit32 + 5'd1;
37        word_ct <= word_ct + 1;
38
39        if(error_bit32 == 5'b01010 || error_bit32 == 5'b01011 || error_bit32 == 5'b01100 || error_bit32 == 5'b01101 )begin //2a7 // # good = 240, bad = 16, Injected :32
40            encoder_o_reg <= {~encoder_o[1],encoder_o[0]}; // inject 4 in a row bad bit out of every 32
41            if(word_ct<36)begin
42                bad_bit_ct <= bad_bit_ct + 1;
43                $display("error_counter = %d",bad_bit_ct);
44            end
45        end
46        else
47            encoder_o_reg <= {encoder_o[1],encoder_o[0]};
48        end
49    end
50
51    // insert your convolutional encoder here
52    // change port names and module name as necessary/desired
53    encoder encoder1 (
54        .clk(clk),
55        .rst(rst),
56        .enable(enable_encoder_i),
57        .d_in(encoder_i),
58        .valid_o(valid_encoder_o),
59        .d_out(decoder_o) );
60
61    // insert your term project code here
62    decoder decoder1 (
63        .clk(clk),
64        .rst(rst),
65        .enable(enable_decoder_in),
66        .d_in(decoder_o),
67        .d_out(decoder_o) );
68
69 endmodule

```

```

# error_counter =          31
# error_counter =          32
# good =          240, bad =          16
# ** Note: $stop      : H:/Folder 5 - Copy/viterbi_tx_rx_tb.sv(371)
# Time: 1381700 ps Iteration: 0 Instance: /viterbi_tx_rx_tb
# Break in Module viterbi_tx_rx_tb at H:/Folder 5 - Copy/viterbi_tx_rx_tb.sv line 371

```

2.a.8

```

1 module viterbi_tx_rx(
2     input clk,
3     input rst,
4     input encoder_i,
5     input enable_encoder_i,
6     output decoder_o);
7
8     wire [1:0] encoder_o;
9
10    logic [3:0] error_counter;
11    logic [1:0] encoder_o_reg;
12
13    logic enable_decoder_in;
14    wire valid_encoder_o;
15
16    //Logic for error for bit 0 and 1
17    logic [2:0] error_bit8;
18    logic [4:0] error_bit32;
19    int bad_bit_ct = 0;
20    int word_ct;
21
22    always @(posedge clk, negedge rst)
23    begin
24        if(!rst) begin
25            error_counter <= 4'd0;
26            encoder_o_reg <= 2'b00;
27            enable_decoder_in <= 1'b0;
28            error_bit8 <= 3'd0;
29            error_bit32 <= 5'd0;
30            word_ct = 0;
31        end
32        else begin
33            enable_decoder_in <= valid_encoder_o;
34            encoder_o_reg <= 2'b00;
35            error_counter <= error_counter + 4'd1;
36            error_bit8 <= error_bit8 + 3'd1;
37            error_bit32 <= error_bit32 + 5'd1;
38            word_ct <= word_ct + 1;
39
40            if(error_bit32 == 5'b01010 || error_bit32 == 5'b01011) begin //2a8 # good = 232, bad = 24 Injected: 32
41                encoder_o_reg <= {encoder_o[1], encoder_o[0]}; // inject 2 in a row bad bit out of every 32
42                if(word_ct < 256) begin
43                    bad_bit_ct <= bad_bit_ct + 2;
44                    $display("error_counter = %d", bad_bit_ct);
45                end
46            end
47        end
48        else
49            encoder_o_reg <= {encoder_o[1], encoder_o[0]};
50        end
51    end
52
53    // insert your convolutional encoder here
54    // change port names and module name as necessary/desired
55    encoder_encoder1 (
56        .clk(clk),
57        .rst(rst),
58        .enable(enable_encoder_i),
59        .d_in(encoder_i),
60        .valid_o(valid_encoder_o),
61        .d_out(decoder_o));
62
63    // insert your term project code here
64    decoder_decoder1 (
65        .clk(clk),
66        .rst(rst),
67        .enable(enable_decoder_in),
68        .d_in(encoder_o_reg),
69        .d_out(decoder_o));
70
71 endmodule

```

```

# error_counter = 28
# error_counter = 30
# error_counter = 32
# yaa! in = 0, out = 0

```

```

# good = 232, bad = 24
# ** Note: $stop : H:/Folder 5 - Copy/viterbi_tx_rx_tb.sv(371)
# Time: 1381700 ps Iteration: 0 Instance: /viterbi_tx_rx_tb
# Break in Module viterbi_tx_rx_tb at H:/Folder 5 - Copy/viterbi_tx_rx_tb.sv line 371

```

Part B

Testbench

```
1 // test bench
2 module viterbi_tx_rx_tb();
3   bit clk;
4   bit rst;
5   bit encoder_i; // original data
6   bit enc_i_hist[2048]; // history thereof
7   bit enable_encoder_i;
8   wire decoder_o; // decoded data, should match original
9   bit dec_o_hist[2048]; // history thereof
10  bit disp; // end of test flag
11  int good, bad; // scoreboard
12
13 // this module contains conv. encode, channel, and vit decode
14 viterbi_tx_rx vtr(
15   .clk,
16   .rst,
17   .encoder_i, // original data
18   .enable_encoder_i,
19   .decoder_o // decoded data
20 );
21
22 always begin
23   #50 clk = 'b1;
24   #50 clk = 'b0;
25 end
26
27 int i, j, k, l;
28
29 always @(posedge clk) begin
30   enc_i_hist[i] <= encoder_i;
31   i <= i+1;
32   l <= l+1; // counters for data in and out
33 end
34
35 initial begin
36   #410500; // #410400;
37   forever @(posedge clk) begin
38     dec_o_hist[k] <= decoder_o;
39     k <= k+1;
40   end
41 end
42
43 initial begin
44   #1000 rst = 'b1;
45   #1000 enable_encoder_i = 'b1;
46   repeat(2) begin
47     #100 encoder_i = 'b1;
48     #100 encoder_i = 'b0;
49     #200 encoder_i = 'b1;
50     #200 encoder_i = 'b0;
51     #300 encoder_i = 'b1;
52     #300 encoder_i = 'b0;
53     #400 encoder_i = 'b1;
54     #400 encoder_i = 'b0;
55     #500 encoder_i = 'b1;
56     #500 encoder_i = 'b0;
57     #100 encoder_i = 'b1;
58     #100 encoder_i = 'b0;
59     #100 encoder_i = 'b1;
60     #100 encoder_i = 'b0;
61     #100 encoder_i = 'b1;
62     #100 encoder_i = 'b0;
63   end
64   #1000 encoder_i = 'b1;
65   #1000 encoder_i = 'b0;
66   repeat(20) begin
```

```
66   repeat(20)
67     #100 encoder_i = $random>>3;
68     #100 encoder_i = 'b0;
69     #1000 encoder_i = 'b1;
70     #1000 encoder_i = 'b0;
71     #100 encoder_i = 'b1;
72     #1000 encoder_i = 'b0;
73     #100 encoder_i = 'b1;
74     #1000 encoder_i = 'b0;
75     #100 encoder_i = 'b1;
76     #1000 encoder_i = 'b0;
77     /*
78     #100 encoder_i = 'b1;
79     #1000 encoder_i = 'b0;
80     #100 encoder_i = 'b1;
81     #1000 encoder_i = 'b0;
82     #1000 encoder_i = 'b1;
83     #100 encoder_i = 'b0;
84     #1000 encoder_i = 'b1;
85     #1000 encoder_i = 'b0;
86     #1000 encoder_i = 'b1;
87     #100 encoder_i = 'b0;
88     #1000 encoder_i = 'b1;
89     #100 encoder_i = 'b0;
90     #1000 encoder_i = 'b1;
91     #100 encoder_i = 'b0;
92     #10000 encoder_i = 'b0;
93     #100
94     encoder_i = 'b1;
95     #10000
96     encoder_i = 'b0;
97     #100
98     encoder_i = 'b1;
99     #100
100    encoder_i = 'b0;
101    #10000
102    encoder_i = 'b1;
103    #100
104    encoder_i = 'b0;
105    #10000
106    encoder_i = 'b1;
107    #100
108    encoder_i = 'b0;
109    #10000
110    encoder_i = 'b1;
111    #10000
112    encoder_i = 'b0;
113    #100
114    encoder_i = 'b1;
115    #10000
116    encoder_i = 'b0;
117    #100
118    encoder_i = 'b1;
119    #10000
120    encoder_i = 'b0;
121    #100
122    encoder_i = 'b1;
123    #10000
124    encoder_i = 'b0;
125    #100
126    encoder_i = 'b1;
127    #10000
128    encoder_i = 'b0;
129    #100
130    encoder_i = 'b1;
131    #10000 encoder_i = 'b0;
```

```

    encoder_i= 1'b1;
*/
#1000000 $display("word_count = %d",vtr.word_ct);
for(j=0; j<256; j=j+1) // checker & scoreboard
    if(enc_i_hist[j]==dec_o_hist[j]) begin
        $displayb("yaa! in = %b, out = %b, err = %b",enc_i_hist[j],dec_o_hist[j],vtr.err_inj);
        good++;
    end
    else begin
        $displayb("boo! in = %b, out = %b, err = %b, BAD!",enc_i_hist[j],dec_o_hist[j],vtr.err_inj);
        bad++;
    end
    $display("good = %d, bad = %d, %d bad_bits",good,bad,vtr.bad_bit_ct);
    disp = 1;
    $stop;
end
endmodule

```

B initial Parameter changes

Varun told us to add this

Injecting 2 bad bits at [1] and [0] for N = 1

```

# good =      187, bad =      69,      108 bad_bits
# ** Note: $stop : H:/Folder 5 - Copy/viterbi_tx_rx_tb (2b).sv(217)
# Time: 1045 ns Iteration: 0 Instance: /viterbi_tx_rx_tb
# Break in Module viterbi_tx_rx_tb at H:/Folder 5 - Copy/viterbi_tx_rx_tb (2b).sv line 217

```

Injecting 2 bad bits at [1] and [0] for N = 2

```

# good =      246, bad =      10,      41 bad_bits
# ** Note: $stop : H:/Folder 5 - Copy/viterbi_tx_rx_tb (2b).sv(217)
# Time: 1045 ns Iteration: 0 Instance: /viterbi_tx_rx_tb
# Break in Module viterbi_tx_rx_tb at H:/Folder 5 - Copy/viterbi_tx_rx_tb (2b).sv line 217

```

Injecting 2 bad bits at [1] and [0] for N = 3

```

# good =      252, bad =       4,      14 bad_bits
# ** Note: $stop : H:/Folder 5 - Copy/viterbi_tx_rx_tb (2b).sv(217)
# Time: 1045 ns Iteration: 0 Instance: /viterbi_tx_rx_tb
# Break in Module viterbi_tx_rx_tb at H:/Folder 5 - Copy/viterbi_tx_rx_tb (2b).sv line 217

```

Injecting 2 bad bits at [1] and [0] for N = 4

```

# good =      256, bad =       0,       5 bad_bits
# ** Note: $stop : H:/Folder 5 - Copy/viterbi_tx_rx_tb (2b).sv(217)
# Time: 1045 ns Iteration: 0 Instance: /viterbi_tx_rx_tb
# Break in Module viterbi_tx_rx_tb at H:/Folder 5 - Copy/viterbi_tx_rx_tb (2b).sv line 217

```

2b1.

```

1 // contains convolutional encoder,
2 // (possibly corrupting) channel,
3 // and viterbi decoder
4 // parameter N sets the channel bit error rate
5 // this time, two errors in a row
6 module viterbi_tx_rx #(parameter N=3) (
7     input clk,
8     input rst,
9     input encoder_i,
10    input enable_encoder_i,
11    output decoder_o);
12
13    wire [1:0] encoder_o; // connects encoder to decoder
14
15    int
16        error_counter,
17        error_counterq,
18        bad_bit_ct,
19        word_ct;
20    logic [1:0]
21        encoder_o_reg,
22        encoder_i_reg,
23        enable_decoder_in,
24        enable_encoder_i_reg;
25    wire
26        valid_encoder_o;
27    logic [1:0] err_inj;
28
29    always @(posedge clk, negedge rst)
30    begin
31        if(rst) begin
32            error_counter <= 'd0;
33            error_counterq <= 'd0;
34            encoder_o_reg <= 'b0;
35            encoder_o_reg <= 'b0;
36            enable_decoder_in <= 'b0;
37            enable_encoder_i_reg <= 'b0;
38            word_ct <= 'b0;
39        end
40        else begin
41            enable_encoder_i_reg <= enable_encoder_i;
42            enable_decoder_in <= valid_encoder_o;
43            encoder_o_reg <= 'b0;
44            if(word_ct[0]) error_counter <= $random;
45            word_ct <= word_ct + 1;
46            // bit error injection in encoder_o_reg
47            encoder_i_reg <= encoder_i;
48            encoder_o_reg <= encoder_o;
49            error_counterq <= error_counter;
50            if(error_counter[N-1:0] == 1) begin //
51                err_inj[0] <= error_counter[28];
52                encoder_o_reg <= encoder_o^err_inj; // inject bad bits
53            end
54            else begin
55                // clean version
56                encoder_o_reg <= encoder_o;
57                err_inj <= 'b0;
58            end
59            if(word_ct < 256) begin
60                bad_bit_ct <= bad_bit_ct + (encoder_o_reg[1]^encoder_o_reg[1]);
61                bad_bit_ct <= bad_bit_ct + (encoder_o_reg[0]^encoder_o_reg[0]);
62                $display("error_counter,err_inj = %d %d %d",
63                    error_counter,err_inj,bad_bit_ct,word_ct);
64            end
65        end
66    end
67 end
68
69
70
71
72
73
74 // insert your term project code here
75 decoder decoder1 (
76     .clk,
77     .rst,
78     .enable (enable_decoder_in),
79     .d_in (encoder_o_reg),
80     .valid_o (valid_encoder_o),
81     .d_out (decoder_o));
82
83 endmodule

```

```

# good =          256, bad =          0,          6 bad_bits
# ** Note: $stop   : H:/Folder 5 - Copy/viterbi_tx_rx_tb (2b).sv(217)
# Time: 1045 ns Iteration: 0 Instance: /viterbi_tx_rx_tb
# Break in Module viterbi_tx_rx_tb at H:/Folder 5 - Copy/viterbi_tx_rx_tb (2b).sv line 217

```

2b2.

```

1 // contains convolutional encoder,
2 // (possibly corrupting) channel,
3 // and viterbi decoder
4 parameter N sets the channel bit error rate
5 this time, two errors in a row
6 module viterbi_tx_rx #(parameter N=3) (
7     input clk,
8     input rst,
9     input enable_encoder_i,
10    input enable_decoder_i,
11    output decoder_o);
12
13    wire [1:0] encoder_o; // connects encoder to decoder
14
15    int error_counter,
16        error_counterq,
17        bad_bit_ct,
18        word_ct;
19    logic [1:0] encoder_o_reg0,
20            encoder_o_reg;
21    logic enable_decoder_in;
22    logic enable_encoder_i_reg;
23    logic valid_encoder_o;
24    wire [1:0] err_inj;
25
26    always @ (posedge clk, negedge rst)
27    begin
28        if(!rst) begin
29            error_counter <= 'd0;
30            error_counterq <= 'd0;
31            encoder_o_reg <= 'b0;
32            encoder_o_reg0 <= 'b0;
33            enable_decoder_in <= 'b0;
34            enable_encoder_i_reg <= 'b0;
35            word_ct <= 'b0;
36        end
37        else begin
38            enable_encoder_i_reg <= enable_encoder_i;
39            enable_decoder_in <= valid_encoder_o;
40            //
41            if(word_ct[0]) error_counter <= $random;
42            word_ct <= word_ct + 1;
43            // bit error injection in encoder_o_reg
44            encoder_i_reg <= encoder_i;
45            encoder_o_reg0 <= encoder_o;
46            error_counterq <= error_counter;
47            if(error_counter[N-1:0]==1) begin //
48                err_inj[1] <= error_counter[20]; // inject bad bits
49                encoder_o_reg <= encoder_o^err_inj;
50            end
51            else begin // clean version
52                encoder_o_reg <= encoder_o;
53                err_inj <= 2'b0;
54            end
55            if(word_ct<256) begin
56                bad_bit_ct <= bad_bit_ct + (encoder_o_reg0[1]^encoder_o_reg[1])
57                    * (encoder_o_reg0[0]^encoder_o_reg[0]);
58                $display("error_counter,err_inj = %h %b %d %d",
59                    error_counter,err_inj,bad_bit_ct,word_ct);
60            end
61        end
62    end
63
64 // ===== Insert your convolutional encoder here =====
65 // change port names and module name as necessary/desired
66 encoder encoder1 (
67     .clk,
68     .rst,
69     .enable_i(enable_encoder_i), //_reg,
70     .d_in (encoder_i),
71     .valid_o (valid_encoder_o), //_reg,
72     .d_out (encoder_o) );
73
74 // insert your term project code here
75 decoder decoder1 (
76     .clk,
77     .rst,
78     .enable (enable_decoder_in),
79     .d_in (encoder_o_reg),
80     .d_out (decoder_o) );
81
82 endmodule
83

```

```

# good =      256, bad =      0,      |      8 bad_bits
# ** Note: $stop      : H:/Folder 5 - Copy/viterbi_tx_rx_tb (2b).sv(217)
# Time: 1045 ns Iteration: 0 Instance: /viterbi_tx_rx_tb
# Break in Module viterbi_tx_rx_tb at H:/Folder 5 - Copy/viterbi_tx_rx_tb (2b).sv line 217

```

2b3.

```
# good = 256, bad = 0, 5 bad_bits
# ** Note: $stop : H:/Folder 5 - Copy/viterbi_tx_rx_tb (2b).sv(217)
# Time: 1045 ns Iteration: 0 Instance: /viterbi_tx_rx_tb
# Break in Module viterbi_tx_rx_tb at H:/Folder 5 - Copy/viterbi_tx_rx_tb (2b).sv line 217
```

```
1 // contains convolutional encoder,
2 // (possibly corrupting) channel,
3 // and viterbi decoder
4 // parameter N sets the channel bit error rate
5 // this time, two errors in a row
6 module viterbi_tx_rx #(parameter N=4) (
7     input clk,
8     input rst,
9     input encoder_i,
10    input enable_encoder_i,
11    output decoder_o);
12
13    wire [1:0] encoder_o; // connects encoder to decoder
14
15    int
16        error_counter,
17        error_counterQ,
18        bad_bit_ct,
19        word_ct;
20    logic [1:0] encoder_o_reg0,
21              encoder_o_reg;
22    logic encoder_i_reg;
23    logic enable_decoder_in;
24    logic enable_encoder_i_reg;
25    wire valid_encoder_o;
26    logic [1:0] err_inj;
27
28    always @ (posedge clk, negedge rst)
29    if (rst) begin
30        error_counter <= 'd0;
31        error_counterQ <= 'd0;
32        encoder_o_reg <= 'b0;
33        encoder_o_reg0 <= 'b0;
34        enable_decoder_in <= 'b0;
35        enable_encoder_i_reg <= 'b0;
36        word_ct <= 'b0;
37    end
38    else begin
39        enable_encoder_i_reg <= enable_encoder_i;
40        enable_decoder_in <= valid_encoder_o;
41        encoder_o_reg <= 'b0;
42        if (word_ct[0]) error_counter <= $random;
43        word_ct <= word_ct + 1;
44        // bit error injection in encoder_o_reg
45        encoder_i_reg <= encoder_i;
46        encoder_o_reg0 <= encoder_o;
47        error_counterQ <= error_counter;
48        // if (error_counter[N-1:0] == '1) begin //
49            err_inj <= error_counter[29:28];
50            encoder_o_reg <= encoder_o^err_inj; // inject bad bits
51        end
52        else begin // clean version
53            encoder_o_reg <= encoder_o;
54            err_inj <= 'b0;
55        end
56        if (word_ct < 256) begin
57            bad_bit_ct <= bad_bit_ct + (encoder_o_reg0[1]^encoder_o_reg[1])
58                          + (encoder_o_reg0[0]^encoder_o_reg[0]);
59            $display("error_counter,err_inj = %d %d %d %d",
60                    error_counter,err_inj,bad_bit_ct,word_ct);
61        end
62    end
63 end
64
65 // insert your convolutional encoder here
66 encoder encoder1 (
67     .clk,
68     .rst,
69     .enable_i(enable_encoder_i), //_reg,
70     .d_in (encoder_i), //_reg,
71     .valid_o (valid_encoder_o), //_reg,
72     .d_out (encoder_o) );
73
74 // insert your term project code here
75 decoder decoder1 (
76     .clk,
77     .rst,
78     .enable (enable_decoder_in),
79     .d_in (encoder_o_reg),
80     .d_out (decoder_o) );
81
82 endmodule
83
```

2b4.

```

1 // contains convolutional encoder,
2 // (possibly corrupting) channel,
3 // and viterbi decoder
4 // parameter N sets the channel bit error rate
5 // this time, two errors in a row
6 module viterbi_tx_rx_tb #(parameter N=4) (
7     input clk,
8     input rst,
9     input enable_encoder_i,
10    output decoder_o);
11
12    wire [1:0] encoder_o; // connects encoder to decoder
13
14    int
15        error_counter,
16        error_counter_q,
17        bad_bit_ct;
18    word_ct;
19    logic [1:0] encoder_o_reg;
20    logic encoder_o_reg;
21    logic enable_decoder_in;
22    logic enable_encoder_i_reg;
23    wire valid_encoder_o;
24    logic [1:0] err_in;
25
26    always @ (posedge clk, negedge rst)
27    if(rst) begin
28        error_counter
29        error_counter_q
30        encoder_o_reg
31        encoder_o_reg
32        enable_decoder_in
33        enable_encoder_i_reg
34        word_ct
35    end
36    else begin
37        enable_encoder_i_reg <= enable_encoder_i;
38        enable_decoder_in <= valid_encoder_o;
39        encoder_o_reg <= b0;
40        if(word_ct[0]) error_counter
41        word_ct <= word_ct + 1; <= $random;
42        // bit error injection in encoder_o_reg
43        encoder_o_reg <= encoder_i;
44        error_counter_q
45        error_counter <= error_counter;
46        if(error_counter[N-1:0] == 4'b1110) begin //
47            err_in[0] <= error_counter[3];
48            encoder_o_reg <= encoder_o_err_in; // inject bad bits
49        end
50        else begin // clean version
51            encoder_o_reg <= encoder_o;
52            err_in <= b0;
53        end
54        if(word_ct<256) begin
55            bad_bit_ct <= bad_bit_ct + (encoder_o_reg[1]&encoder_o_reg[1])
56            + (encoder_o_reg[0]&encoder_o_reg[1]);
57            $display("error_counter,err_in]= %d %d %d %d",
58                error_counter,err_in,bad_bit_ct,word_ct);
59        end
60    end
61
62
63 // insert your convolutional encoder here
64 change port names and module name as necessary/desired
65 encoder encoder1 (
66     .clk,
67     .rst,
68     .enable_i(enable_encoder_i), // _reg,
69     .d_in (encoder_i), // _reg,
70     .valid_o (valid_encoder_o),
71     .d_out (encoder_o) );
72
73 // insert your term project code here
74 decoder decoder1 (
75     .clk,
76     .rst,
77     .enable (enable_decoder_in),
78     .d_in (encoder_o_reg),
79     .d_out (decoder_o) );
80
81 endmodule
82
83

```

```

63 L
64 // insert your convolutional encoder here
65 change port names and module name as necessary/desired
66 encoder encoder1 (
67     .clk,
68     .rst,
69     .enable_i(enable_encoder_i), // _reg,
70     .d_in (encoder_i), // _reg,
71     .valid_o (valid_encoder_o),
72     .d_out (encoder_o) );
73
74 // insert your term project code here
75 decoder decoder1 (
76     .clk,
77     .rst,
78     .enable (enable_decoder_in),
79     .d_in (encoder_o_reg),
80     .d_out (decoder_o) );
81
82 endmodule
83

```

```

# good = 256, bad = 0, 4 bad_bits
# ** Note: $stop : H:/Folder 5 - Copy/viterbi_tx_rx_tb (2b).sv(217)
# Time: 1045 ns Iteration: 0 Instance: /viterbi_tx_rx_tb
# Break in Module viterbi_tx_rx_tb at H:/Folder 5 - Copy/viterbi_tx_rx_tb (2b).sv line 217

```


2b5.

```

1 // contains convolutional encoder,
2 // (possibly corrupting) channel,
3 // and viterbi decoder
4 // parameter N sets the channel bit error rate
5 // this time, two errors in a row
6 module viterbi_tx_rx #(parameter N=4) (
7     input clk,
8     input rst,
9     input enable_encoder_i,
10    input enable_decoder_i,
11    output decoder_o);
12
13    wire [1:0] encoder_o; // connects encoder to decoder
14
15    int error_counter,
16        error_counter0,
17        bad_bit_ct;
18    logic [1:0] encoder_o_reg0;
19    logic encoder_o_reg;
20    logic enable_decoder_in;
21    logic enable_encoder_i_reg;
22    wire valid_encoder_o;
23    logic [1:0] err_inj;
24
25    always @(posedge clk, negedge rst)
26    if(!rst) begin
27        error_counter <= 'd0;
28        error_counter0 <= 'd0;
29        encoder_o_reg <= 'b0;
30        enable_decoder_in <= 'b0;
31        enable_encoder_i_reg <= 'b0;
32        valid_encoder_o <= 'b0;
33        word_ct <= 'b0;
34    end
35    else begin
36        enable_encoder_i_reg <= enable_encoder_i;
37        enable_decoder_in <= enable_decoder_i;
38        encoder_o_reg <= encoder_o;
39        if(word_ct[0]) error_counter <= $random;
40        word_ct <= word_ct + 1;
41        // bit error injection in encoder_o_reg
42        encoder_i_reg <= encoder_i;
43        encoder_o_reg0 <= encoder_o;
44        error_counter0 <= error_counter;
45        if(error_counter[N-1:0] >= 4'b1110) begin //
46            err_inj[1] <= error_counter[2:0];
47            encoder_o_reg <= encoder_o ^ err_inj; // inject bad bits
48        end
49        else begin // clean version
50            encoder_o_reg <= encoder_o;
51            err_inj <= 2'b0;
52        end
53        if(word_ct < 256) begin
54            bad_bit_ct <= bad_bit_ct + (encoder_o_reg0[1] ^ encoder_o_reg[1])
55                + (encoder_o_reg0[0] ^ encoder_o_reg[0]);
56            $display("error_counter, err_inj = %h %b %d %d",
57                error_counter, err_inj, bad_bit_ct, word_ct);
58        end
59    end
60 end
61
62
63
64 // insert your convolutional encoder here
65 // change port names and module name as necessary/desired
66 encoder1 (
67     .clk,
68     .rst,
69     .enable_i(enable_encoder_i), //reg,
70     .d_in (encoder_i), //reg,
71     .valid_o (valid_encoder_o),
72     .d_out (encoder_o));
73
74 // insert your term project code here
75 decoder1 (
76     .clk,
77     .rst,
78     .enable (enable_decoder_in),
79     .d_in (encoder_o_reg),
80     .d_out (decoder_o));
81 endmodule

```

```

# good = 256, bad = 0, 6 bad_bits
# ** Note: $stop : H:/Folder 5 - Copy/viterbi_tx_rx_tb (2b).sv(217)
# Time: 1045 ns Iteration: 0 Instance: /viterbi_tx_rx_tb
# Break in Module viterbi_tx_rx_tb at H:/Folder 5 - Copy/viterbi_tx_rx_tb (2b).sv line 217

```

2b6.

```

1 // contains: convolutional encoder,
2 // (possibly corrupting) channel,
3 // and viterbi decoder
4 // parameter N sets the channel bit error rate
5 // this time, two errors in a row
6 module viterbi_tx_rx_tb #(parameter N=5) (
7     input clk,
8     input rst,
9     input enable_encoder_i,
10    input enable_decoder_i,
11    output decoder_o);
12
13    wire [1:0] encoder_o; // connects encoder to decoder
14
15    int error_counter,
16        error_counterq,
17        bad_bit_ct;
18    word_ct encoder_o_reg,
19    word_ct encoder_i_reg;
20    logic [1:0] encoder_o_reg;
21    logic [1:0] encoder_i_reg;
22    logic enable_decoder_in;
23    logic enable_encoder_i_reg;
24    wire valid_encoder_o;
25    logic [1:0] err_inj;
26
27    always @ (posedge clk, negedge rst)
28    if(rst) begin
29        error_counter
30        // error_counterq
31        encoder_o_reg
32        encoder_i_reg
33        enable_decoder_in
34        enable_encoder_i_reg
35        word_ct
36    end
37    else begin
38        enable_encoder_i_reg <= enable_encoder_i;
39        enable_decoder_in <= valid_encoder_o;
40        encoder_o_reg <= b0;
41        if(word_ct[0]) error_counter
42        word_ct <= word_ct + 1; <= $random;
43        // bit error injection in encoder_o_reg
44        encoder_i_reg <= encoder_i;
45        encoder_o_reg <= encoder_o;
46        error_counterq <= error_counter;
47        if(error_counter[N-1:0]==5'b11100) begin //
48            err_inj[0] <= error_counter[28];
49            encoder_o_reg <= encoder_o^err_inj; // inject bad bits
50        end
51        // clean version
52        encoder_o_reg <= encoder_o;
53        err_inj <= b0;
54    end
55    if(word_ct<256) begin
56        bad_bit_ct <= bad_bit_ct + (encoder_o_reg[1]^encoder_o_reg[1])
57        + (encoder_o_reg[0]^encoder_o_reg[0]);
58        $display("error_counter,err_inj = %d %d %d %d",
59            error_counter,err_inj,bad_bit_ct,word_ct);
60    end
61    end
62    end
63    end
64    // insert your convolutional encoder here
65    change port names and module name as necessary/desired
66    encoder encoder1 (
67        .clk,
68        .rst,
69        .enable_i(enable_encoder_i), //_reg,
70        .d_in (encoder_i), //_reg,
71        .valid_o (valid_encoder_o), //_reg,
72        .d_out (encoder_o) );
73
74    // insert your term project code here
75    decoder decoder1 (
76        .clk,
77        .rst,
78        .enable (enable_decoder_in),
79        .d_in (encoder_o_reg),
80        .d_out (decoder_o));
81
82 endmodule
83

```

```

64 // insert your convolutional encoder here
65 change port names and module name as necessary/desired
66 encoder encoder1 (
67     .clk,
68     .rst,
69     .enable_i(enable_encoder_i), //_reg,
70     .d_in (encoder_i), //_reg,
71     .valid_o (valid_encoder_o), //_reg,
72     .d_out (encoder_o) );
73
74 // insert your term project code here
75 decoder decoder1 (
76     .clk,
77     .rst,
78     .enable (enable_decoder_in),
79     .d_in (encoder_o_reg),
80     .d_out (decoder_o));
81
82 endmodule
83

```

```

# good = 256, bad = 0, 7 bad_bits
# ** Note: $stop : H:/Folder 5 - Copy/viterbi_tx_rx_tb (2b).sv(217)
# Time: 1045 ns Iteration: 0 Instance: /viterbi_tx_rx_tb
# Break in Module viterbi_tx_rx_tb at H:/Folder 5 - Copy/viterbi_tx_rx_tb (2b).sv line 217

```

2b7.

```

1 // contains convolutional encoder,
2 // (possibly corrupting) channel,
3 // and viterbi decoder
4 // parameter N sets the channel bit error rate
5 // this time, two errors in a row
6 module viterbi_tx_rx_tb2b7 (parameter N=5) (
7     input clk,
8     input rst,
9     input enable_encoder_i,
10    output decoder_o);
11
12    wire [1:0] encoder_o; // connects encoder to decoder
13
14    int error_counter,
15        error_counter_q,
16        bad_bit_ct;
17    word_ct encoder_o_reg,
18    word_ct encoder_i_reg;
19    logic [1:0] encoder_o_reg,
20    encoder_i_reg;
21    logic enable_decoder_in;
22    logic enable_encoder_i_reg;
23    wire valid_encoder_o;
24    logic [1:0] err_in;
25
26    always @ (posedge clk, negedge rst)
27    if (rst) begin
28        error_counter
29        error_counter_q
30        encoder_o_reg
31        encoder_i_reg
32        enable_decoder_in
33        enable_encoder_i_reg
34        word_ct
35    end
36    else begin
37        enable_encoder_i_reg <= enable_encoder_i;
38        enable_decoder_in <= valid_encoder_o;
39        encoder_o_reg <= b0;
40        if (word_ct[0]) error_counter
41        word_ct <= word_ct + 1; <= $random;
42        // bit error injection in encoder_o_reg
43        encoder_o_reg <= encoder_i;
44        encoder_i_reg <= error_counter;
45        if (error_counter[N-1:0] == 5'b11100) begin //
46            err_in[1] <= error_counter[29];
47            encoder_o_reg <= encoder_o_err_in; // inject bad bits
48        end
49        else begin // clean version
50            encoder_o_reg <= encoder_o;
51            err_in <= b0;
52        end
53        if (word_ct < 256) begin
54            bad_bit_ct <= bad_bit_ct + (encoder_o_reg[1] < encoder_o_reg[1])
55            + (encoder_o_reg[0] < encoder_o_reg[0]);
56            $display("error_counter: %0d, err_in: %0d, bad_bit_ct: %0d, word_ct: %0d",
57                error_counter, err_in, bad_bit_ct, word_ct);
58        end
59    end
60
61    // insert your convolutional encoder here
62    // change port names and module name as necessary/desired
63    encoder encoder1 (
64        .clk,
65        .rst,
66        .enable_i (enable_encoder_i), //_reg,
67        .d_in (encoder_i), //_reg,
68        .valid_o (valid_encoder_o),
69        .d_out (encoder_o) );
70
71    // insert your term project code here
72    decoder decoder1 (
73        .clk,
74        .rst,
75        .enable (enable_decoder_in),
76        .d_in (encoder_o_reg),
77        .d_out (decoder_o) );
78
79 endmodule

```

```

64 // insert your convolutional encoder here
65 // change port names and module name as necessary/desired
66 encoder encoder1 (
67     .clk,
68     .rst,
69     .enable_i (enable_encoder_i), //_reg,
70     .d_in (encoder_i), //_reg,
71     .valid_o (valid_encoder_o),
72     .d_out (encoder_o) );
73
74 // insert your term project code here
75 decoder decoder1 (
76     .clk,
77     .rst,
78     .enable (enable_decoder_in),
79     .d_in (encoder_o_reg),
80     .d_out (decoder_o) );
81
82 endmodule
83

```

```

# good = 256, bad = 0, 8 bad_bits
# ** Note: $stop : H:/Folder 5 - Copy/viterbi_tx_rx_tb (2b).sv(217)
# Time: 1045 ns Iteration: 0 Instance: /viterbi_tx_rx_tb
# Break in Module viterbi_tx_rx_tb at H:/Folder 5 - Copy/viterbi_tx_rx_tb (2b).sv line 217

```

2b8.

```

1 // contains convolutional encoder,
2 // (possibly corrupting) channel,
3 // and viterbi decoder
4 // parameter N sets the channel bit error rate
5 // this time, two errors in a row
6 module viterbi_tx_rx_tb #(parameter N=5) (
7     input clk,
8     input rst,
9     input enable_encoder_i,
10    output decoder_o);
11
12    wire [1:0] encoder_o; // connects encoder to decoder
13
14    int error_counter,
15        error_counter0,
16        bad_bit_ct;
17    word_ct encoder_o_reg0,
18    word_ct encoder_o_reg;
19    logic [1:0] encoder_o_reg;
20    logic enable_decoder_in;
21    logic enable_encoder_i_reg;
22    wire valid_encoder_o;
23    logic [1:0] err_in;
24
25    always @ (posedge clk, negedge rst)
26    if(rst) begin
27        error_counter <= 'd0;
28        error_counter0 <= 'd0;
29        encoder_o_reg <= 'b0;
30        encoder_o_reg0 <= 'b0;
31        enable_decoder_in <= 'b0;
32        enable_encoder_i_reg <= 'b0;
33        word_ct <= 'b0;
34    end
35    else begin
36        enable_encoder_i_reg <= enable_encoder_i;
37        enable_decoder_in <= valid_encoder_o;
38        encoder_o_reg <= 'b0;
39        if(word_ct[0]) error_counter <= $random;
40        word_ct <= word_ct + 1;
41        // bit error injection in encoder_o_reg
42        encoder_o_reg0 <= encoder_o;
43        encoder_o_reg <= encoder_i;
44        if(error_counter[N-1:0] == 5'b11110) begin //
45            error_counter0 <= error_counter;
46            encoder_o_reg <= encoder_o_err_in; // inject bad bits
47        end
48        else begin // clean version
49            encoder_o_reg <= encoder_o;
50            err_in <= 'b0;
51        end
52        if(word_ct < 256) begin
53            bad_bit_ct <= bad_bit_ct + (encoder_o_reg0[1] ^ encoder_o_reg[1])
54            + (encoder_o_reg0[0] ^ encoder_o_reg[0]);
55            $display("error_counter: %d", error_counter);
56            $display("bad_bit_ct: %d", bad_bit_ct);
57        end
58    end
59
60    // insert your convolutional encoder here
61    change port names and module name as necessary/desired
62    encoder encoder1 (
63        .clk,
64        .rst,
65        .enable_i(enable_encoder_i), //reg,
66        .d_in (encoder_i), //reg,
67        .valid_o (valid_encoder_o),
68        .d_out (encoder_o) );
69
70    // insert your term project code here
71    decoder decoder1 (
72        .clk,
73        .rst,
74        .enable (enable_decoder_in),
75        .d_in (encoder_o_reg),
76        .d_out (decoder_o));
77
78 endmodule

```

```

63 // insert your convolutional encoder here
64 change port names and module name as necessary/desired
65 encoder encoder1 (
66     .clk,
67     .rst,
68     .enable_i(enable_encoder_i), //reg,
69     .d_in (encoder_i), //reg,
70     .valid_o (valid_encoder_o),
71     .d_out (encoder_o) );
72
73 // insert your term project code here
74 decoder decoder1 (
75     .clk,
76     .rst,
77     .enable (enable_decoder_in),
78     .d_in (encoder_o_reg),
79     .d_out (decoder_o));
80
81 endmodule
82
83

```

```

# good = 256, bad = 0, 5 bad_bits
# ** Note: $stop : H:/Folder 5 - Copy/viterbi_tx_rx_tb (2b).sv(217)
# Time: 1045 ns Iteration: 0 Instance: /viterbi_tx_rx_tb
# Break in Module viterbi_tx_rx_tb at H:/Folder 5 - Copy/viterbi_tx_rx_tb (2b).sv line 217

```

Part C

Testbench: same as part A

```
1 module viterbi_tx_rx(
2     input clk,
3     input rst,
4     input encoder_i,
5     input enable_encoder_i,
6     output decoder_o);
7
8     wire [1:0] encoder_o;
9
10    logic [3:0] error_counter;
11    logic [1:0] encoder_o_reg;
12
13    logic enable_decoder_in;
14    wire valid_encoder_o;
15
16    //Logic for error for bit 0 and 1
17    logic [2:0] error_bit8;
18    logic [4:0] error_bit32;
19    int bad_bit_ct = 1;
20    int word_ct;
21
22
23    always @(posedge clk, negedge rst)
24    if(!rst) begin
25        error_counter <= 4'd0;
26        encoder_o_reg <= 2'b00;
27        enable_decoder_in <= 1'b0;
28        error_bit8 <= 3'd0;
29        error_bit32 <= 5'd0;
30        word_ct <= 0;
31    end
32    else begin
33        enable_decoder_in <= valid_encoder_o;
34        encoder_o_reg <= 2'b00;
35        error_counter <= error_counter + 4'd1;
36        error_bit8 <= error_bit8 + 3'd1;
37        error_bit32 <= error_bit32 + 5'd1;
38        word_ct <= word_ct + 1;
39
40        if(error_counter == 4'b1001 || error_counter == 4'b1000 || error_counter == 4'b0111 || error_counter == 4'b0110) begin
41            //2: //with 3 consecutive # good =>56, bad=0 // with 4 consecutive good = 181, bad = 75 BROKE THE CODE
42            encoder_o_reg <= {encoder_o[1],encoder_o[0]}; // inject 4 bad bit out of every 16
43            if(word_ct<256)begin
44                bad_bit_ct <= bad_bit_ct + 1;
45                $display("error_counter = %d",bad_bit_ct);
46            end
47        end
48    end
49    else
50        encoder_o_reg <= {encoder_o[1],encoder_o[0]};
51    end
52
53    // insert your convolutional encoder here
54    // change port names and module name as necessary/desired
55    encoder encoder1 (
56        .clk(clk),
57        .rst(rst),
58        .enable(enable_encoder_i),
59        .d_in(encoder_i),
60        .valid_o(valid_encoder_o),
61        .d_out(decoder_o));
62
63
64
```

With 3 consecutive, we got 0 bad bits and 4 consecutive broke our code.

```
65 // insert your term project code here
66 decoder decoder1 (
67     .clk(clk),
68     .rst(rst),
69     .enable(enable_decoder_in),
70     .d_in(encoder_o_reg),
71     .d_out(decoder_o)
72 );
73
74 endmodule
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
2455
2456
2457
2458
2459
2460
2461
2462
2463
2464
2465
2466
2467
2468
2469
2470
2471
2472
2473
2474
2475
2476
2477
2478
2479
2480
2481
2482
2483
2484
2485
2486
2487
2488
2489
2490
2491
2492
2493
2494
2495
2496
2497
2498
2499
2500
2501
2502
2503
2504
2505
2506
2507
2508
2509
2510
2511
2512
2513
2514
2515
2516
2517
2518
2519
2520
2521
2522
2523
2524
2525
2526
2527
2528
2529
2530
2531
2532
2533
2534
2535
2536
253
```

Part D

Testbench: same as part A

```
1 module viterbi_tx_rx(
2     input clk,
3     input rst,
4     input enable_encoder_i,
5     output decoder_o);
6
7     wire [1:0] encoder_o;
8
9
10    logic [3:0] error_counter;
11    logic [1:0] encoder_o_reg;
12
13    logic enable_decoder_in;
14    wire valid_encoder_o;
15
16    //Logic for error for bit 0 and 1
17    logic [2:0] error_bit8;
18    logic [4:0] error_bit32;
19    int bad_bit_ct = 1;
20    int word_ct;
21
22
23    always @ (posedge clk, negedge rst)
24    if(rst) begin
25        error_counter <= 4'd0;
26        encoder_o_reg <= 2'b00;
27        enable_decoder_in <= 1'b0;
28        error_bit8 <= 3'd0;
29        error_bit32 <= 5'd0;
30        word_ct = 0;
31    end
32    else begin
33        enable_decoder_in <= valid_encoder_o;
34        encoder_o_reg <= 2'b00;
35        error_counter <= error_counter + 4'd1;
36        error_bit8 <= error_bit8 + 3'd1;
37        error_bit32 <= error_bit32 + 5'd1;
38        word_ct <= word_ct + 1;
39    end
40
41    if(error_counter == 4'b1001 || error_counter == 4'b1000 || error_counter == 4'b0111 || error_counter == 4'b0110 ) begin
42        //2d //with 3 consecutive # good ~256, bad=0 // with 4 consecutive good = 226, bad = 30 BROKE THE CODE
43        encoder_o_reg <= {encoder_o[1],encoder_o[0]}; // inject 4 bits every 16
44        if(word_ct>0)begin
45            bad_bit_ct <= bad_bit_ct + 1;
46            $display("error_counter = %d",bad_bit_ct);
47        end
48    end
49
50    else
51        encoder_o_reg <= {encoder_o[1],encoder_o[0]};
52    end
53
54    // Insert your convolutional encoder here
55    // change port names and module name as necessary/desired
56    encoder_encoder1 (
57        .clk(clk),
58        .rst(rst),
59        .enable_i(enable_encoder_i),
60        .d_in(encoder_o_reg),
61        .valid_o(valid_encoder_o),
62        .d_out(encoder_o) );
63
64 endmodule
```

With 3 consecutive we got 0 bad bits, and with 4 consecutive, it broke our code.

```
64 // insert your term project code here
65 decoder_decoder1 (
66     .clk(clk),
67     .rst(rst),
68     .enable(enable_decoder_in),
69     .d_in(encoder_o_reg),
70     .d_out(decoder_o)
71 );
72 endmodule
73
74
75
76 # error_counter = 61
77 # error_counter = 62
78 # error_counter = 63
79 # error_counter = 64
80
81
82 # boo! in = 1, out = 0
83 # good = 226, bad = 30
84 # ** Note: $stop : C:/Users/William/Desktop/Folder 5/viterbi_tx_rx_tb(4).sv(371)
85 # Time: 1381700 ps Iteration: 0 Instance: /viterbi_tx_rx_tb
86 # Break in Module viterbi_tx_rx_tb at C:/Users/William/Desktop/Folder 5/viterbi_tx_rx_tb(4).sv line 371
```

Part E

Testbench: same as part A

```
1 module viterbi_tx_rx(
2     input clk,
3     input rst,
4     input enable_encoder_1,
5     input enable_decoder_1,
6     output decoder_o);
7
8     wire [1:0] encoder_o;
9
10    logic [3:0] error_counter;
11    logic [1:0] encoder_o_reg;
12
13    logic enable_decoder_in;
14    wire valid_encoder_o;
15
16    //Logic for error for bit 0 and 1
17    logic [2:0] error_bit8;
18    logic [2:0] error_bit32;
19    int bad_bit_ct = 2;
20    int word_ct;
21
22    always @(posedge clk, negedge rst)
23    begin
24        if(rst) begin
25            error_counter <= 4'd0;
26            encoder_o_reg <= 2'b00;
27            enable_decoder_in <= 1'b0;
28            error_bit8 <= 3'd0;
29            error_bit32 <= 5'd0;
30            word_ct = 0;
31        end
32        else begin
33            enable_decoder_in <= valid_encoder_o;
34            encoder_o_reg <= 2'b00;
35            error_counter <= error_counter + 4'd1;
36            error_bit8 <= error_bit8 + 3'd1;
37            error_bit32 <= error_bit32 + 5'd1;
38            word_ct <= word_ct + 1;
39        end
40    end
41
42    if(error_counter == 4'b1001 || error_counter == 4'b1000) begin
43        //2e # with 2 injected every 16 good = 256, bad = 0, Injected = 32 // with 4 consecutive, good = 211, bad = 45 BROKE THE CODE
44        encoder_o_reg <= {encoder_o[1], encoder_o[0]};
45        if(word_ct < 16) begin
46            bad_bit_ct <= bad_bit_ct + 2;
47            $display("error_counter = %d", bad_bit_ct);
48        end
49    end
50    else
51        encoder_o_reg <= {encoder_o[1], encoder_o[0]};
52    end
53
54    // Insert your convolutional encoder here
55    change port names and module name as necessary/desired
56    encoder_encoder1 (
57        .clk(clk),
58        .rst(rst),
59        .enable_1(enable_encoder_1),
60        .d_in(encoder_o_reg),
61        .valid_o(valid_encoder_o),
62        .d_out(encoder_o));
63
64    // Insert your term project code here
65 endmodule
```

With 2 bits injected every 16, we got 0 bad bits. With 4 bad bits injected in a row, it broke the code.

```
64 // Insert your term project code here
65 decoder_decoder1 (
66     .clk(clk),
67     .rst(rst),
68     .enable(enable_decoder_in),
69     .d_in(encoder_o_reg),
70     .d_out(decoder_o)
71 );
72 endmodule
73
74
75 # error_counter = 22
76 # error_counter = 24
77 # error_counter = 26
78 # error_counter = 28
79 # error_counter = 30
80 # error_counter = 32
81 # error_counter = 34
82 # error_counter = 36
83 # error_counter = 38
84 # error_counter = 40
85 # error_counter = 42
86 # error_counter = 44
87 # error_counter = 46
88 # error_counter = 48
89 # error_counter = 50
90 # error_counter = 52
91 # error_counter = 54
92 # error_counter = 56
93 # error_counter = 58
94 # error_counter = 60
95 # error_counter = 62
96 # error_counter = 64
97
98 # yaa! in = 1, out = 1
99 # good = 211, bad = 45
100 # ** Note: $stop : C:/Users/William/Downloads/viterbi_tx_rx_tb(4).sv(371)
101 # Time: 1381700 ps Iteration: 0 Instance: /viterbi_tx_rx_tb
102 # Break in Module viterbi_tx_rx_tb at C:/Users/William/Downloads/viterbi_tx_rx_tb(4).sv line 371
103
104 VSIM 12>
```

Final Results of all Test

2	NonRandom Tests							
3	Test Number	ViterbiTxRx	BER	SymbolPattern	BitPattern	BadBitsIn	BadBitsOut	
4	2.a.1	viterbi_tx_rx(a1).sv	1/8	1 in a row	01	32	0	
5	2.a.2	viterbi_tx_rx(a2).sv	1/8	1 in a row	10	32	0	
6	2.a.3	viterbi_tx_rx(a3).sv	2/8	1 in a row	11	64	51	
7	2.a.4	viterbi_tx_rx(a4).sv	2/16	2 in a row	01	32	0	
8	2.a.5	viterbi_tx_rx(a5).sv	2/16	2 in a row	10	32	0	
9	2.a.6	viterbi_tx_rx(a6).sv	4/32	4 in a row	01	32	33	
10	2.a.7	viterbi_tx_rx(a7).sv	4/32	4 in a row	10	32	16	
11	2.a.8	viterbi_tx_rx(a8).sv	2/32	2 in a row	11	32	24	
12	2.C.	viterbi_tx_rx(2c).sv	2/16	1 in a row	01	64	75	
13	2.D	viterbi_tx_rx(2d).sv	4/16	1 in a row	10	64	30	
14	2.E	viterbi_tx_rx(2e).sv	2/16	2 in a row	11	64	45	
15								
16								
17								
18								
19								
20								
21								
22								
23								
24								
25								
26	Random Tests							
27	Test Number	ViterbiTxRx	Avg. BER	SymbolPattern	BitPattern	BadBitsIn	BadBitsOut	
28	2.b.1	viterbi_tx_rx(2b1).sv	3/128	1 in a row	01	6	0	
29	2.b.2	viterbi_tx_rx(2b2).sv	1/32	1 in a row	10	8	0	
30	2.b.3	viterbi_tx_rx(2b3).sv	5/256	1 in a row	11	5	0	
31	2.b.4	viterbi_tx_rx(2b4).sv	1/64	2 in a row	01	4	0	
32	2.b.5	viterbi_tx_rx(2b5).sv	3/128	2 in a row	10	6	0	
33	2.b.6	viterbi_tx_rx(2b6).sv	7/256	4 in a row	01	7	0	
34	2.b.7	viterbi_tx_rx(2b7).sv	1/32	4 in a row	10	8	0	
35	2.b.8	viterbi_tx_rx(2b8).sv	5/256	2 in a row	11	5	0	

The decoder is performing well because we expect badbitsout to be significantly lower than badbitsin.

The tests are categorized into nonrandom and random, each presenting different conditions and bit patterns. Variations in performance across tests suggest the decoder's error correction efficiency.