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

Viterbi_tx_rx

```
module viterbi_tx_rx #(parameter N=2) (
    input    clk,
    input    rst,
    input    encoder_i,
    input    enable_encoder_i,
    output   decoder_o);

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

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

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

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

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

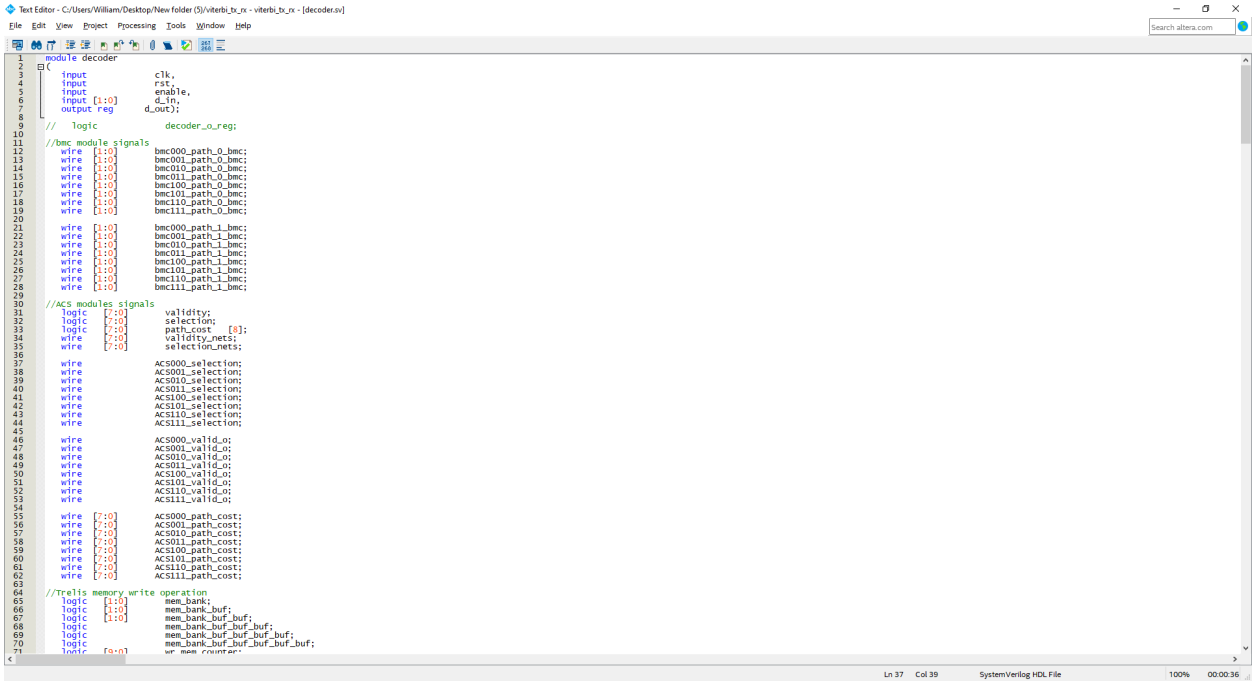
```
      end


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

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

endmodule
```

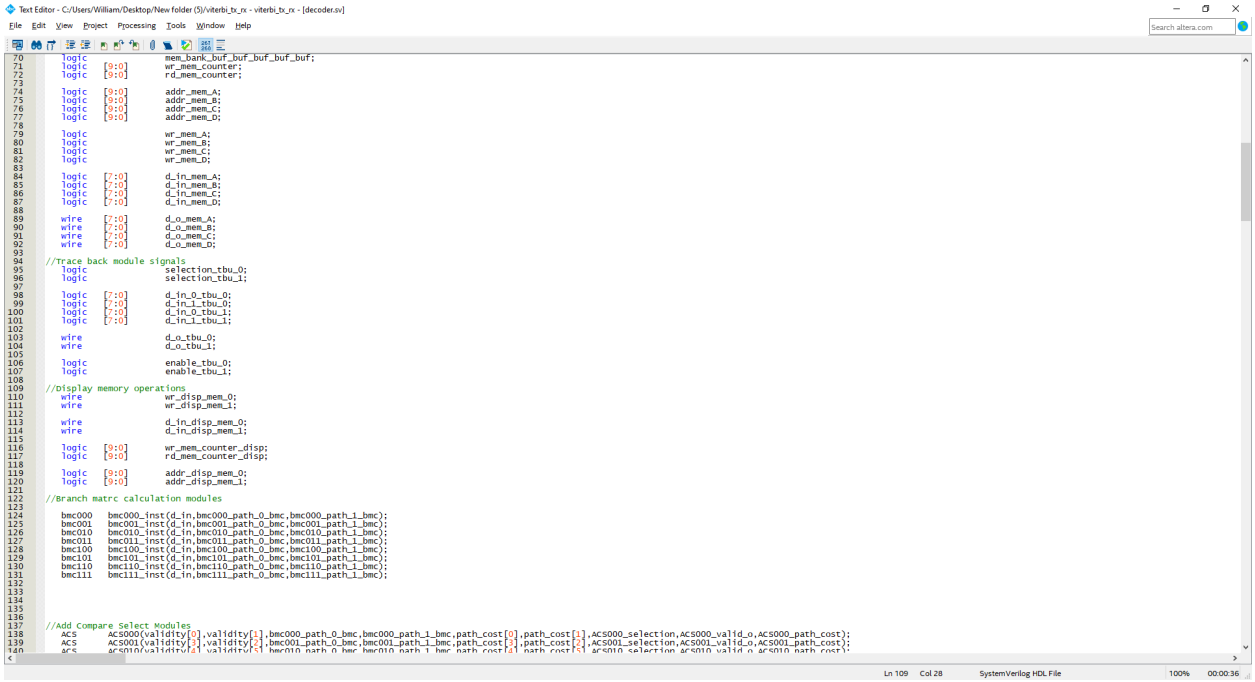
Encoder

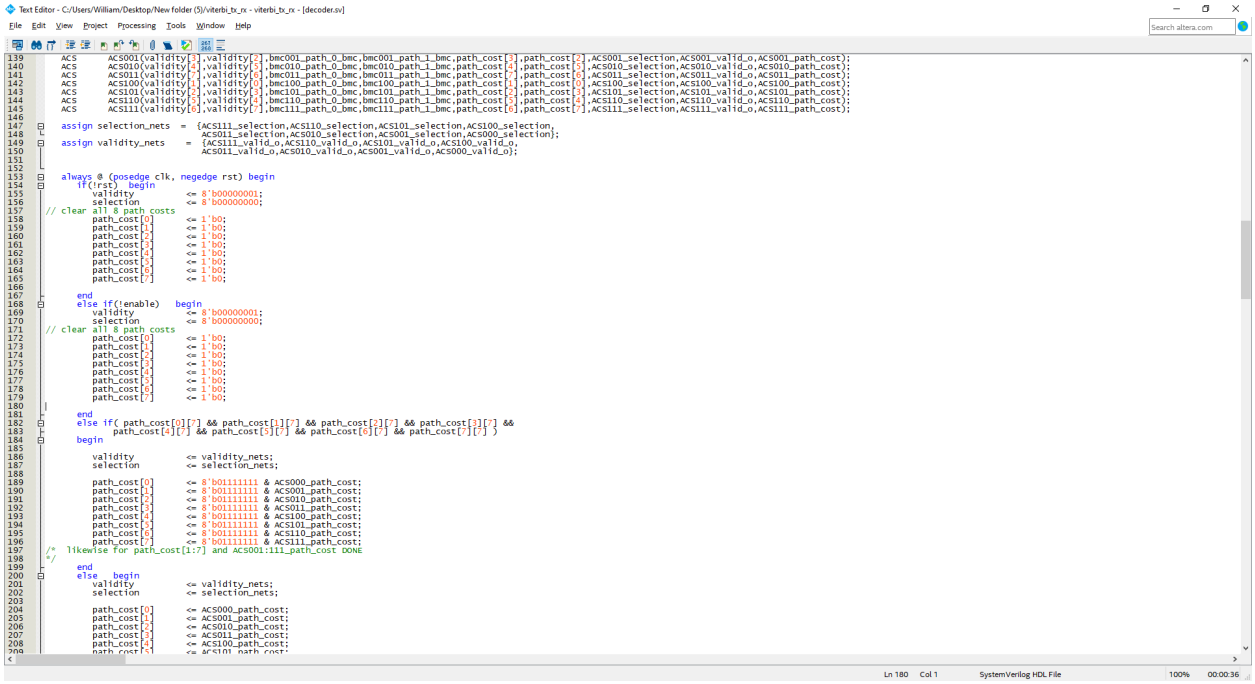
```
1   module encoder                          // use this one
2   (  input              clk,
3      input              rst,
4      input              enable_i,
5      input              d_in,
6      output logic       valid_o,
7      output logic [1:0] d_out);
8
9      logic          [2:0] cstate;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

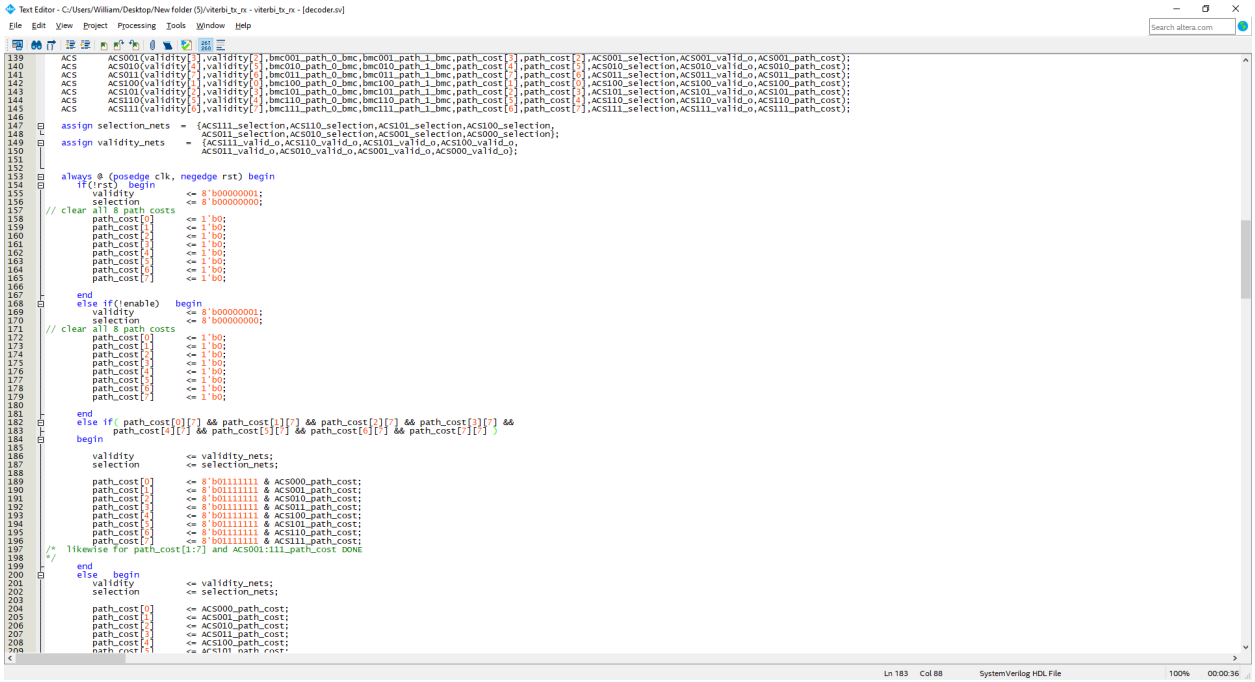
```
61                else       begin
62                   nstate    =  3'b010;
63                   d_out_reg =  2'b01;
64                end
65             3'b110:  if(!d_in)   begin
66                   nstate    =  3'b111;
67                   d_out_reg =  2'b00;
68                end
69                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
```

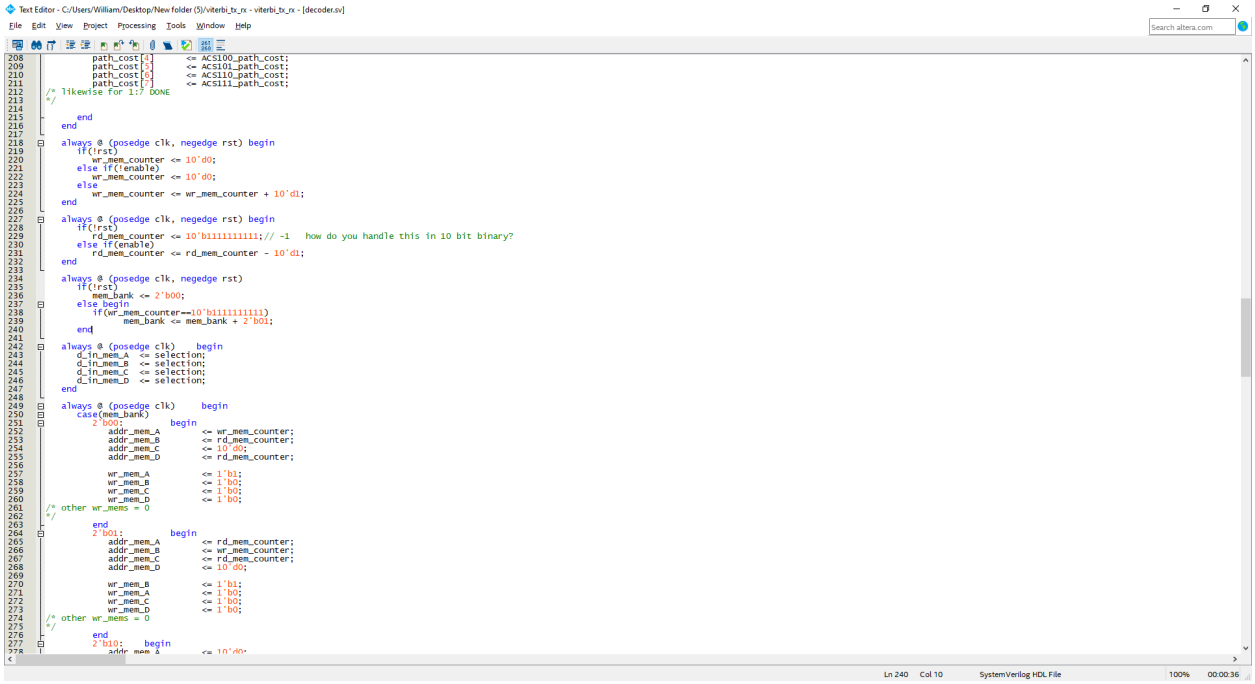
Decoder

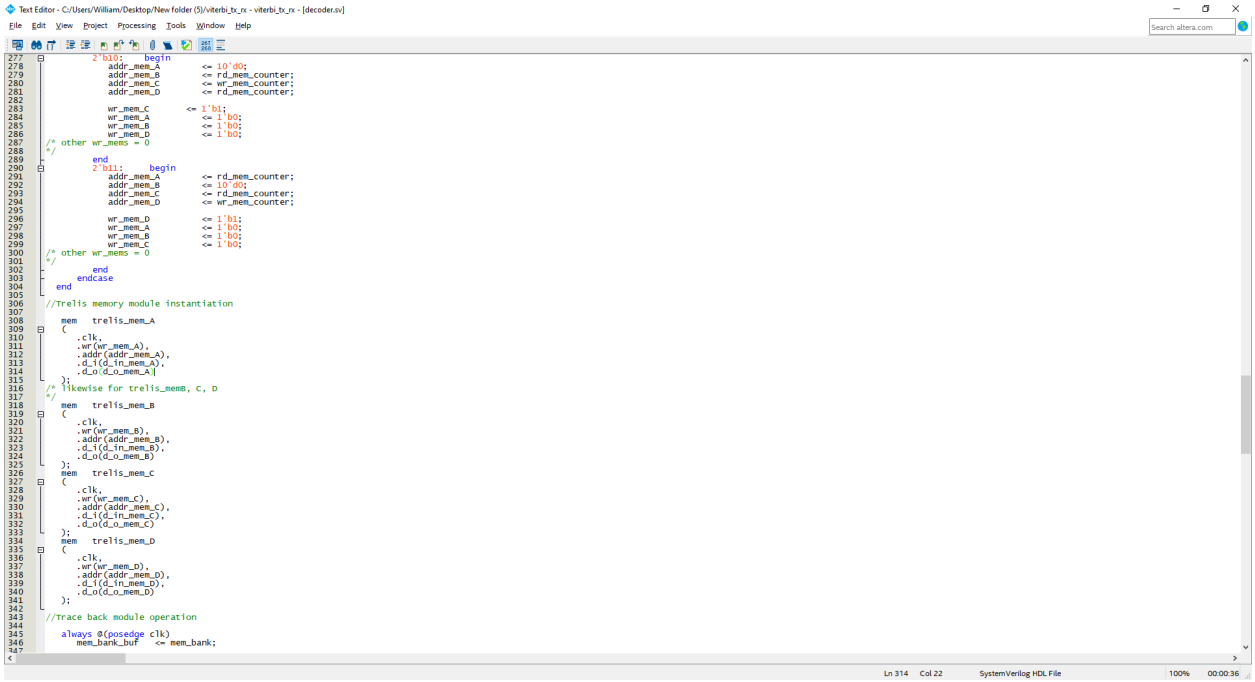

Text Editor - C:/Users/William/Desktop/New folder (5)/viterbi_tx_rx - viterbi_tx_rx - [decoder.sv]
File Edit View Project Processing Tools Window Help
Search altera.com
module decoder
(
input clk,
input rst,
input enable,
input [1:0] d_in,
output reg d_out);
// logic decoder_o_reg;
//bmc module signals
wire [1:0] bmc000_path_0_bmc;
wire [1:0] bmc001_path_0_bmc;
wire [1:0] bmc010_path_0_bmc;
wire [1:0] bmc011_path_0_bmc;
wire [1:0] bmc100_path_0_bmc;
wire [1:0] bmc101_path_0_bmc;
wire [1:0] bmc110_path_0_bmc;
wire [1:0] bmc111_path_0_bmc;
wire [1:0] bmc000_path_1_bmc;
wire [1:0] bmc001_path_1_bmc;
wire [1:0] bmc010_path_1_bmc;
wire [1:0] bmc011_path_1_bmc;
wire [1:0] bmc100_path_1_bmc;
wire [1:0] bmc101_path_1_bmc;
wire [1:0] bmc110_path_1_bmc;
wire [1:0] bmc111_path_1_bmc;
//ACS modules signals
logic [7:0] validity;
logic [7:0] selection;
logic [7:0] path_cost [8];
wire [7:0] validity_nets;
wire [7:0] selection_nets;
wire ACS000_selection;
wire ACS001_selection;
wire ACS010_selection;
wire ACS011_selection;
wire ACS100_selection;
wire ACS101_selection;
wire ACS110_selection;
wire ACS111_selection;
wire ACS000_valid_o;
wire ACS001_valid_o;
wire ACS010_valid_o;
wire ACS011_valid_o;
wire ACS100_valid_o;
wire ACS101_valid_o;
wire ACS110_valid_o;
wire ACS111_valid_o;
wire [7:0] ACS000_path_cost;
wire [7:0] ACS001_path_cost;
wire [7:0] ACS010_path_cost;
wire [7:0] ACS011_path_cost;
wire [7:0] ACS100_path_cost;
wire [7:0] ACS101_path_cost;
wire [7:0] ACS110_path_cost;
wire [7:0] ACS111_path_cost;
//Trellis memory write operation
logic [1:0] mem_bank;
logic [1:0] mem_bank_buf;
logic [1:0] mem_bank_buf_buf;
logic mem_bank_buf_buf_buf;
logic mem_bank_buf_buf_buf_buf;
logic mem_bank_buf_buf_buf_buf_buf;
Ln 37 Col 39
SystemVerilog HDL File
100%
00:00:36

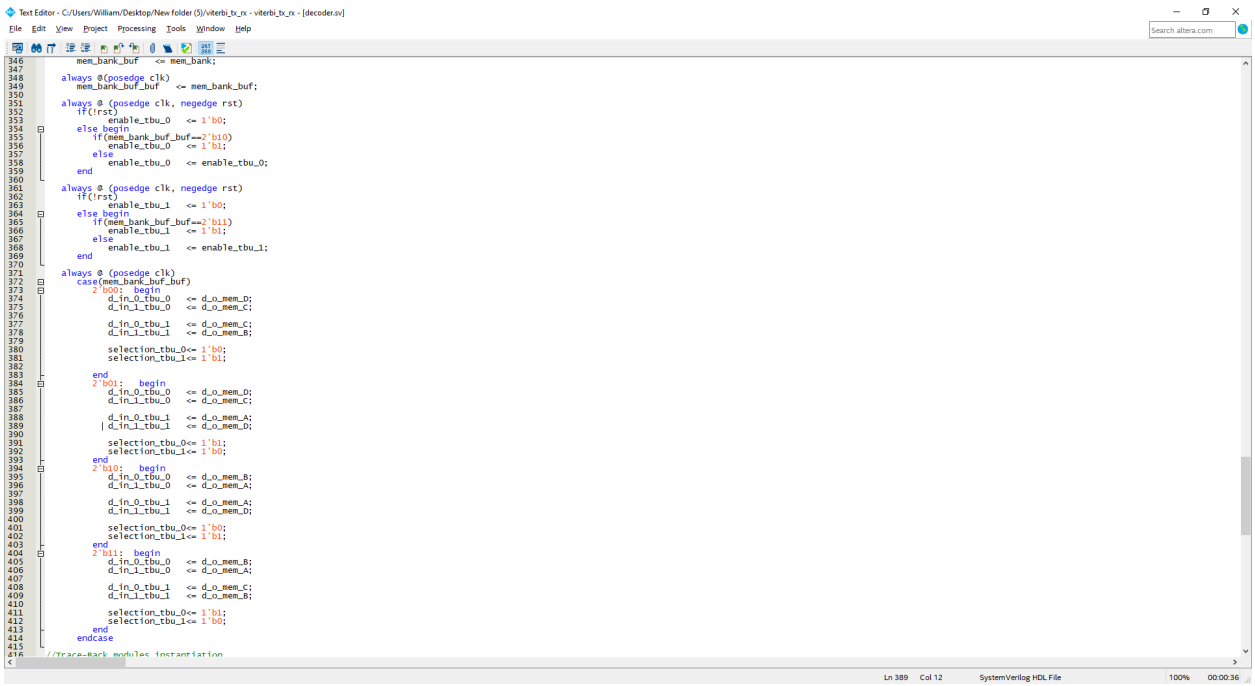

Text Editor - C:/Users/William/Desktop/New folder (5)/viterbi_tx_rx - viterbi_tx_rx - [decoder.sv]
File Edit View Project Processing Tools Window Help
Search altera.com
Ln 109 Col 28 SystemVerilog HDL File 100% 00:00:36

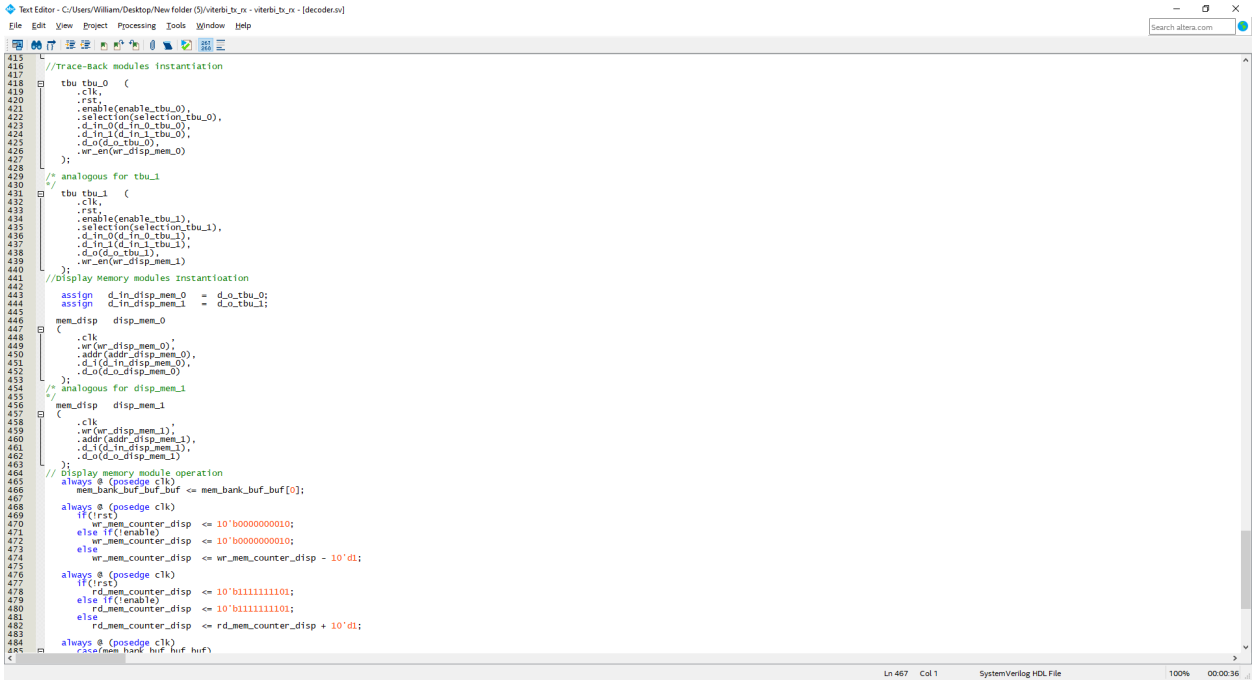

Text Editor - C:/Users/William/Desktop/New folder (5)/viterbi_tx_rx - viterbi_tx_rx - [decoder.sv]
File Edit View Project Processing Tools Window Help
Search altera.com
Ln 180 Col 1 SystemVerilog HDL File 100% 00:00:36

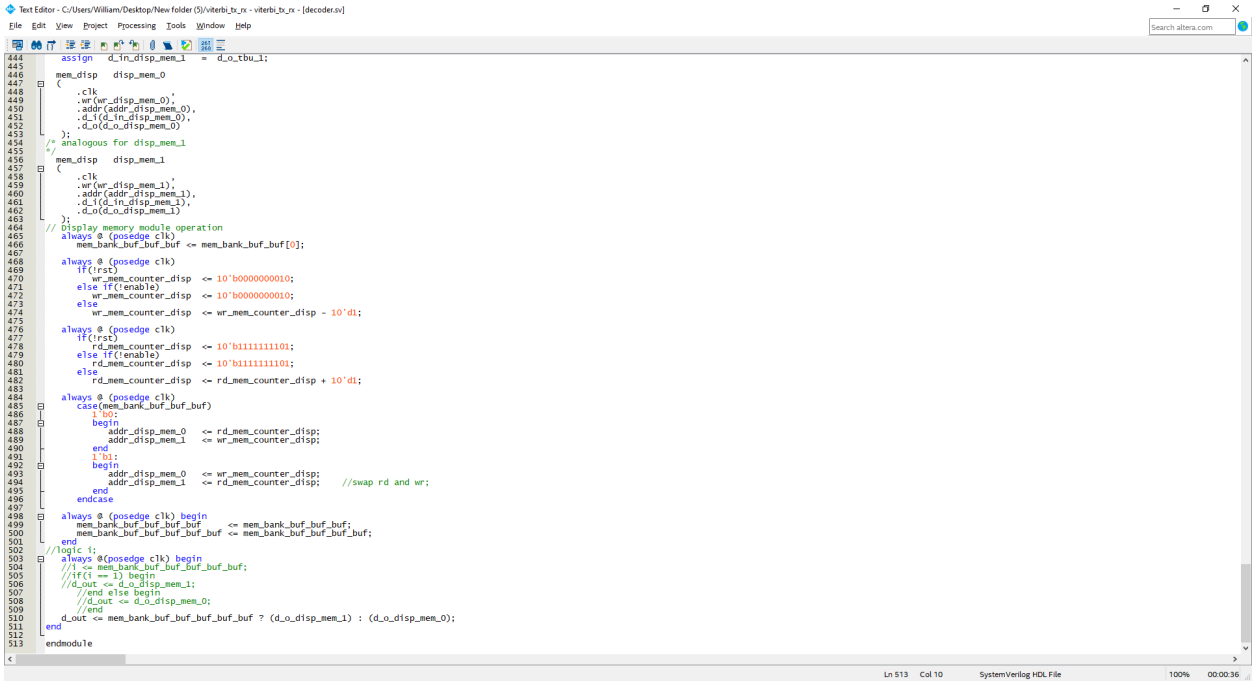

Text Editor - C:/Users/William/Desktop/New folder (5)/viterbi_tx_rx - viterbi_tx_rx - [decoder.sv]
File Edit View Project Processing Tools Window Help
Search altera.com
ACS ACS001(validity[2],validity[5],bmc001_path_0_bmc,bmc001_path_1_bmc,path_cost[3],path_cost[2],ACS001_selection,ACS001_valid_o,ACS001_path_cost);
ACS ACS010(validity[4],validity[5],bmc010_path_0_bmc,bmc010_path_1_bmc,path_cost[4],path_cost[5],ACS010_selection,ACS010_valid_o,ACS010_path_cost);
ACS ACS011(validity[7],validity[6],bmc011_path_0_bmc,bmc011_path_1_bmc,path_cost[7],path_cost[6],ACS011_selection,ACS011_valid_o,ACS011_path_cost);
ACS ACS100(validity[1],validity[0],bmc100_path_0_bmc,bmc100_path_1_bmc,path_cost[1],path_cost[0],ACS100_selection,ACS100_valid_o,ACS100_path_cost);
ACS ACS101(validity[2],validity[3],bmc101_path_0_bmc,bmc101_path_1_bmc,path_cost[2],path_cost[3],ACS101_selection,ACS101_valid_o,ACS101_path_cost);
ACS ACS110(validity[5],validity[4],bmc110_path_0_bmc,bmc110_path_1_bmc,path_cost[5],path_cost[4],ACS110_selection,ACS110_valid_o,ACS110_path_cost);
ACS ACS111(validity[0],validity[7],bmc111_path_0_bmc,bmc111_path_1_bmc,path_cost[0],path_cost[7],ACS111_selection,ACS111_valid_o,ACS111_path_cost);
assign selection_nets = {ACS111_selection,ACS110_selection,ACS101_selection,ACS100_selection,
ACS011_selection,ACS010_selection,ACS001_selection,ACS000_selection};
assign validity_nets = {ACS111_valid_o,ACS110_valid_o,ACS101_valid_o,ACS100_valid_o,
ACS011_valid_o,ACS010_valid_o,ACS001_valid_o,ACS000_valid_o};
always @ (posedge clk, negedge rst) begin
if(!rst) begin
validity <= 8'b00000001;
selection <= 8'b00000000;
// clear all 8 path costs
path_cost[0] <= 1'b0;
path_cost[1] <= 1'b0;
path_cost[2] <= 1'b0;
path_cost[3] <= 1'b0;
path_cost[4] <= 1'b0;
path_cost[5] <= 1'b0;
path_cost[6] <= 1'b0;
path_cost[7] <= 1'b0;
end
else if(!enable) begin
validity <= 8'b00000001;
selection <= 8'b00000000;
// clear all 8 path costs
path_cost[0] <= 1'b0;
path_cost[1] <= 1'b0;
path_cost[2] <= 1'b0;
path_cost[3] <= 1'b0;
path_cost[4] <= 1'b0;
path_cost[5] <= 1'b0;
path_cost[6] <= 1'b0;
path_cost[7] <= 1'b0;
end
else if(path_cost[0][7] && path_cost[1][7] && path_cost[2][7] && path_cost[3][7] &&
path_cost[4][7] && path_cost[5][7] && path_cost[6][7] && path_cost[7][7])
begin
validity <= validity_nets;
selection <= selection_nets;
path_cost[0] <= 8'b01111111 & ACS000_path_cost;
path_cost[1] <= 8'b01111111 & ACS001_path_cost;
path_cost[2] <= 8'b01111111 & ACS010_path_cost;
path_cost[3] <= 8'b01111111 & ACS011_path_cost;
path_cost[4] <= 8'b01111111 & ACS100_path_cost;
path_cost[5] <= 8'b01111111 & ACS101_path_cost;
path_cost[6] <= 8'b01111111 & ACS110_path_cost;
path_cost[7] <= 8'b01111111 & ACS111_path_cost;
/* likewise for path_cost[1:7] and ACS001:111_path_cost DONE
*/
end
else begin
validity <= validity_nets;
selection <= selection_nets;
path_cost[0] <= ACS000_path_cost;
path_cost[1] <= ACS001_path_cost;
path_cost[2] <= ACS010_path_cost;
path_cost[3] <= ACS011_path_cost;
path_cost[4] <= ACS100_path_cost;
path_cost[5] <= ACS101_path_cost;
Ln 183 Col 88 SystemVerilog HDL File 100% 00:00:36


Text Editor - C:/Users/William/Desktop/New folder (5)/viterbi_tx_rx - viterbi_tx_rx - [decoder.sv]
File Edit View Project Processing Tools Window Help
Search altera.com
path_cost[4] <= ACS100_path_cost;
path_cost[5] <= ACS101_path_cost;
path_cost[6] <= ACS110_path_cost;
path_cost[7] <= ACS111_path_cost;
/* likewise for 1:7 DONE
*/
end
end
always @ (posedge clk, negedge rst) begin
if(!rst)
wr_mem_counter <= 10'd0;
else if(!enable)
wr_mem_counter <= 10'd0;
else
wr_mem_counter <= wr_mem_counter + 10'd1;
end
always @ (posedge clk, negedge rst) begin
if(!rst)
rd_mem_counter <= 10'b1111111111;// -1 how do you handle this in 10 bit binary?
else if(enable)
rd_mem_counter <= rd_mem_counter - 10'd1;
end
always @ (posedge clk, negedge rst)
if(!rst)
mem_bank <= 2'b00;
else begin
if(wr_mem_counter==10'b1111111111)
mem_bank <= mem_bank + 2'b01;
end
always @ (posedge clk) begin
d_in_mem_A <= selection;
d_in_mem_B <= selection;
d_in_mem_C <= selection;
d_in_mem_D <= selection;
end
always @ (posedge clk) begin
case(mem_bank)
2'b00: begin
addr_mem_A <= wr_mem_counter;
addr_mem_B <= rd_mem_counter;
addr_mem_C <= 10'd0;
addr_mem_D <= rd_mem_counter;
wr_mem_A <= 1'b1;
wr_mem_B <= 1'b0;
wr_mem_C <= 1'b0;
wr_mem_D <= 1'b0;
/* other wr_mems = 0
*/
end
2'b01: begin
addr_mem_A <= rd_mem_counter;
addr_mem_B <= wr_mem_counter;
addr_mem_C <= rd_mem_counter;
addr_mem_D <= 10'd0;
wr_mem_B <= 1'b1;
wr_mem_A <= 1'b0;
wr_mem_C <= 1'b0;
wr_mem_D <= 1'b0;
/* other wr_mems = 0
*/
end
2'b10: begin
Ln 240 Col 10 SystemVerilog HDL File 100% 00:00:36


Text Editor - C:/Users/William/Desktop/New folder (5)/viterbi_tx_rx - viterbi_tx_rx - [decoder.sv]
File Edit View Project Processing Tools Window Help
Search altera.com
2'b10: begin
addr_mem_A <= 10'd0;
addr_mem_B <= rd_mem_counter;
addr_mem_C <= wr_mem_counter;
addr_mem_D <= rd_mem_counter;
wr_mem_C <= 1'b1;
wr_mem_A <= 1'b0;
wr_mem_B <= 1'b0;
wr_mem_D <= 1'b0;
/* other wr_mems = 0
*/
end
2'b11: begin
addr_mem_A <= rd_mem_counter;
addr_mem_B <= 10'd0;
addr_mem_C <= rd_mem_counter;
addr_mem_D <= wr_mem_counter;
wr_mem_D <= 1'b1;
wr_mem_A <= 1'b0;
wr_mem_B <= 1'b0;
wr_mem_C <= 1'b0;
/* other wr_mems = 0
*/
end
endcase
end
//Trelis memory module instantiation
mem trelis_mem_A
(
.clk,
.wr(wr_mem_A),
.addr(addr_mem_A),
.d_i(d_in_mem_A),
.d_o(d_o_mem_A)
);
/* likewise for trelis_memB, C, D
*/
mem trelis_mem_B
(
.clk,
.wr(wr_mem_B),
.addr(addr_mem_B),
.d_i(d_in_mem_B),
.d_o(d_o_mem_B)
);
mem trelis_mem_C
(
.clk,
.wr(wr_mem_C),
.addr(addr_mem_C),
.d_i(d_in_mem_C),
.d_o(d_o_mem_C)
);
mem trelis_mem_D
(
.clk,
.wr(wr_mem_D),
.addr(addr_mem_D),
.d_i(d_in_mem_D),
.d_o(d_o_mem_D)
);
//Trace back module operation
always @(posedge clk)
mem_bank_buf <= mem_bank;
Ln 314 Col 22 SystemVerilog HDL File 100% 00:00:36


Text Editor - C:/Users/William/Desktop/New folder (5)/viterbi_tx_rx - viterbi_tx_rx - [decoder.sv]
File Edit View Project Processing Tools Window Help
Search altera.com
mem_bank_buf <= mem_bank;
always @(posedge clk)
mem_bank_buf_buf <= mem_bank_buf;
always @ (posedge clk, negedge rst)
if(!rst)
enable_tbu_0 <= 1'b0;
else begin
if(mem_bank_buf_buf==2'b10)
enable_tbu_0 <= 1'b1;
else
enable_tbu_0 <= enable_tbu_0;
end
always @ (posedge clk, negedge rst)
if(!rst)
enable_tbu_1 <= 1'b0;
else begin
if(mem_bank_buf_buf==2'b11)
enable_tbu_1 <= 1'b1;
else
enable_tbu_1 <= enable_tbu_1;
end
always @ (posedge clk)
case(mem_bank_buf_buf)
2'b00: begin
d_in_0_tbu_0 <= d_o_mem_D;
d_in_1_tbu_0 <= d_o_mem_C;
d_in_0_tbu_1 <= d_o_mem_C;
d_in_1_tbu_1 <= d_o_mem_B;
selection_tbu_0<= 1'b0;
selection_tbu_1<= 1'b1;
end
2'b01: begin
d_in_0_tbu_0 <= d_o_mem_D;
d_in_1_tbu_0 <= d_o_mem_C;
d_in_0_tbu_1 <= d_o_mem_A;
d_in_1_tbu_1 <= d_o_mem_D;
selection_tbu_0<= 1'b1;
selection_tbu_1<= 1'b0;
end
2'b10: begin
d_in_0_tbu_0 <= d_o_mem_B;
d_in_1_tbu_0 <= d_o_mem_A;
d_in_0_tbu_1 <= d_o_mem_A;
d_in_1_tbu_1 <= d_o_mem_D;
selection_tbu_0<= 1'b0;
selection_tbu_1<= 1'b1;
end
2'b11: begin
d_in_0_tbu_0 <= d_o_mem_B;
d_in_1_tbu_0 <= d_o_mem_A;
d_in_0_tbu_1 <= d_o_mem_C;
d_in_1_tbu_1 <= d_o_mem_B;
selection_tbu_0<= 1'b1;
selection_tbu_1<= 1'b0;
end
endcase
//Trace-Back modules instantiation
Ln 389 Col 12 SystemVerilog HDL File 100% 00:00:36


Text Editor - C:/Users/William/Desktop/New folder (5)/viterbi_tx_rx - viterbi_tx_rx - [decoder.sv]
File Edit View Project Processing Tools Window Help
Search altera.com
//Trace-Back modules instantiation
tbu tbu_0 (
.clk,
.rst,
.enable(enable_tbu_0),
.selection(selection_tbu_0),
.d_in_0(d_in_0_tbu_0),
.d_in_1(d_in_1_tbu_0),
.d_o(d_o_tbu_0),
.wr_en(wr_disp_mem_0)
);
/* analogous for tbu_1
*/
tbu tbu_1 (
.clk,
.rst,
.enable(enable_tbu_1),
.selection(selection_tbu_1),
.d_in_0(d_in_0_tbu_1),
.d_in_1(d_in_1_tbu_1),
.d_o(d_o_tbu_1),
.wr_en(wr_disp_mem_1)
);
//Display Memory modules Instantioation
assign d_in_disp_mem_0 = d_o_tbu_0;
assign d_in_disp_mem_1 = d_o_tbu_1;
mem_disp disp_mem_0
(
.clk
.wr(wr_disp_mem_0),
.addr(addr_disp_mem_0),
.d_i(d_in_disp_mem_0),
.d_o(d_o_disp_mem_0)
);
/* analogous for disp_mem_1
*/
mem_disp disp_mem_1
(
.clk
.wr(wr_disp_mem_1),
.addr(addr_disp_mem_1),
.d_i(d_in_disp_mem_1),
.d_o(d_o_disp_mem_1)
);
// Display memory module operation
always @ (posedge clk)
mem_bank_buf_buf_buf <= mem_bank_buf_buf[0];
always @ (posedge clk)
if(!rst)
wr_mem_counter_disp <= 10'b0000000010;
else if(!enable)
wr_mem_counter_disp <= 10'b0000000010;
else
wr_mem_counter_disp <= wr_mem_counter_disp - 10'd1;
always @ (posedge clk)
if(!rst)
rd_mem_counter_disp <= 10'b1111111101;
else if(!enable)
rd_mem_counter_disp <= 10'b1111111101;
else
rd_mem_counter_disp <= rd_mem_counter_disp + 10'd1;
always @ (posedge clk)
case(mem_bank_buf_buf_buf)
Ln 467 Col 1
SystemVerilog HDL File
100%
00:00:36


Text Editor - C:/Users/William/Desktop/New folder (5)/viterbi_tx_rx - viterbi_tx_rx - [decoder.sv]
File Edit View Project Processing Tools Window Help
Search altera.com
assign d_in_disp_mem_1 = d_o_tbu_1;
mem_disp disp_mem_0
(
.clk
.wr(wr_disp_mem_0),
.addr(addr_disp_mem_0),
.d_i(d_in_disp_mem_0),
.d_o(d_o_disp_mem_0)
);
/* analogous for disp_mem_1
*/
mem_disp disp_mem_1
(
.clk
.wr(wr_disp_mem_1),
.addr(addr_disp_mem_1),
.d_i(d_in_disp_mem_1),
.d_o(d_o_disp_mem_1)
);
// Display memory module operation
always @ (posedge clk)
mem_bank_buf_buf_buf <= mem_bank_buf_buf[0];
always @ (posedge clk)
if(!rst)
wr_mem_counter_disp <= 10'b0000000010;
else if(!enable)
wr_mem_counter_disp <= 10'b0000000010;
else
wr_mem_counter_disp <= wr_mem_counter_disp - 10'd1;
always @ (posedge clk)
if(!rst)
rd_mem_counter_disp <= 10'b1111111101;
else if(!enable)
rd_mem_counter_disp <= 10'b1111111101;
else
rd_mem_counter_disp <= rd_mem_counter_disp + 10'd1;
always @ (posedge clk)
case(mem_bank_buf_buf_buf)
1'b0:
begin
addr_disp_mem_0 <= rd_mem_counter_disp;
addr_disp_mem_1 <= wr_mem_counter_disp;
end
1'b1:
begin
addr_disp_mem_0 <= wr_mem_counter_disp;
addr_disp_mem_1 <= rd_mem_counter_disp; //swap rd and wr;
end
endcase
always @ (posedge clk) begin
mem_bank_buf_buf_buf_buf <= mem_bank_buf_buf_buf;
mem_bank_buf_buf_buf_buf_buf <= mem_bank_buf_buf_buf_buf;
end
//logic 1;
always @(posedge clk) begin
//i <= mem_bank_buf_buf_buf_buf_buf;
//if(i == 1) begin
//d_out <= d_o_disp_mem_1;
//end else begin
//d_out <= d_o_disp_mem_0;
//end
d_out <= mem_bank_buf_buf_buf_buf_buf ? (d_o_disp_mem_1) : (d_o_disp_mem_0);
end
endmodule
Ln 513 Col 10
SystemVerilog HDL File
100%
00:00:36

BMC Files

BMC 000

```
module bmc000              // branch metric computation
(
    input    [1:0] rx_pair,
    output   [1:0] path_0_bmc,
    output   [1:0] path_1_bmc);

     wire tmp00, tmp01, tmp10, tmp11;


    assign tmp00 = rx_pair[0];
    assign tmp01 = rx_pair[1];

    assign tmp10 = ~tmp00;
    assign tmp11 = ~tmp01;

    assign path_0_bmc[0] = tmp00 ^ tmp01;
    assign path_0_bmc[1] = tmp00 & tmp01;

    assign path_1_bmc[0] = tmp10 ^ tmp11;
    assign path_1_bmc[1] = tmp10 & tmp11;
endmodule
```

BMC 001

```
module bmc001
(
    input    [1:0] rx_pair,
    output   [1:0] path_0_bmc,
    output   [1:0] path_1_bmc);


    wire tmp00, tmp01, tmp10, tmp11;


    assign tmp00 = rx_pair[0];
    assign tmp01 = ~rx_pair[1];

    assign tmp10 = ~tmp00;
    assign tmp11 = ~tmp01;

    assign path_0_bmc[0] = tmp00 ^ tmp01;
    assign path_0_bmc[1] = tmp00 & tmp01;

    assign path_1_bmc[0] = tmp10 ^ tmp11;
    assign path_1_bmc[1] = tmp10 & tmp11;


endmodule

```

BMC 010

```
module bmc010
(
    input    [1:0] rx_pair,
    output   [1:0] path_0_bmc,
    output   [1:0] path_1_bmc);

   wire tmp00, tmp01, tmp10, tmp11;


    assign tmp00 = rx_pair[0];
    assign tmp01 = ~rx_pair[1];

    assign tmp10 = ~tmp00;
    assign tmp11 = ~tmp01;

    assign path_0_bmc[0] = tmp00 ^ tmp01;
    assign path_0_bmc[1] = tmp00 & tmp01;

    assign path_1_bmc[0] = tmp10 ^ tmp11;
    assign path_1_bmc[1] = tmp10 & tmp11;
endmodule

```

BMC011

```
module bmc011
(
    input    [1:0] rx_pair,
    output   [1:0] path_0_bmc,
    output   [1:0] path_1_bmc);

   wire tmp00, tmp01, tmp10, tmp11;


    assign tmp00 = rx_pair[0];
    assign tmp01 = rx_pair[1];

    assign tmp10 = ~tmp00;
    assign tmp11 = ~tmp01;

    assign path_0_bmc[0] = tmp00 ^ tmp01;
    assign path_0_bmc[1] = tmp00 & tmp01;

    assign path_1_bmc[0] = tmp10 ^ tmp11;
    assign path_1_bmc[1] = tmp10 & tmp11;
endmodule

```

BMC100

```
module bmc100
(
    input    [1:0] rx_pair,
    output   [1:0] path_0_bmc,
    output   [1:0] path_1_bmc);


     wire tmp00, tmp01, tmp10, tmp11;


    assign tmp00 = rx_pair[0];
    assign tmp01 = rx_pair[1];

    assign tmp10 = ~tmp00;
    assign tmp11 = ~tmp01;

    assign path_0_bmc[0] = tmp00 ^ tmp01;
    assign path_0_bmc[1] = tmp00 & tmp01;

    assign path_1_bmc[0] = tmp10 ^ tmp11;
    assign path_1_bmc[1] = tmp10 & tmp11;
endmodule

```

BMC101

```
module bmc101          (
    input    [1:0] rx_pair,
    output   [1:0] path_0_bmc,
    output   [1:0] path_1_bmc);


    wire tmp00, tmp01, tmp10, tmp11;


    assign tmp00 = rx_pair[0];
    assign tmp01 = ~rx_pair[1];

    assign tmp10 = ~tmp00;
    assign tmp11 = ~tmp01;

    assign path_0_bmc[0] = tmp00 ^ tmp01;
    assign path_0_bmc[1] = tmp00 & tmp01;

    assign path_1_bmc[0] = tmp10 ^ tmp11;
    assign path_1_bmc[1] = tmp10 & tmp11;
endmodule

```

BMC110

```
module bmc110
(
    input    [1:0] rx_pair,
    output   [1:0] path_0_bmc,
    output   [1:0] path_1_bmc);


   wire tmp00, tmp01, tmp10, tmp11;


    assign tmp00 = rx_pair[0];
    assign tmp01 = ~rx_pair[1];

    assign tmp10 = ~tmp00;
    assign tmp11 = ~tmp01;

    assign path_0_bmc[0] = tmp00 ^ tmp01;
    assign path_0_bmc[1] = tmp00 & tmp01;

    assign path_1_bmc[0] = tmp10 ^ tmp11;
    assign path_1_bmc[1] = tmp10 & tmp11;
endmodule

```

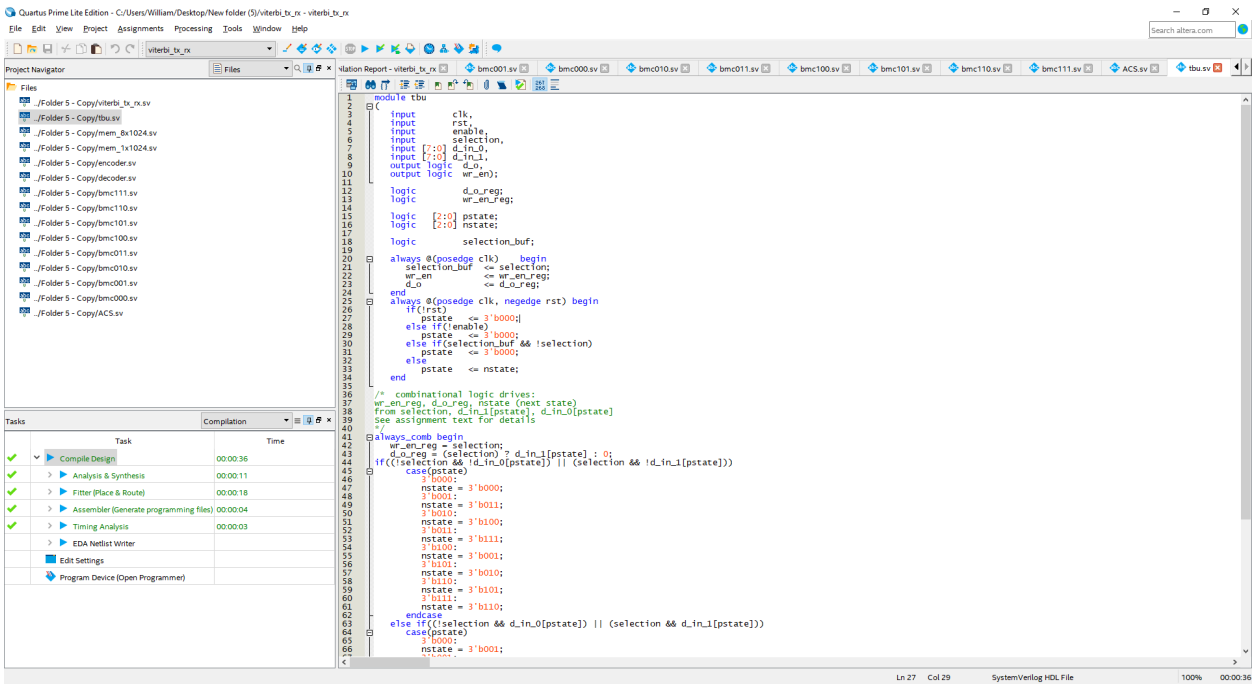
BMC111

```
module bmc111 (
    input    [1:0] rx_pair,
    output   [1:0] path_0_bmc,
    output   [1:0] path_1_bmc);


    wire tmp00, tmp01, tmp10, tmp11;


    assign tmp00 = rx_pair[0];
    assign tmp01 = rx_pair[1];

    assign tmp10 = ~tmp00;
    assign tmp11 = ~tmp01;

    assign path_0_bmc[0] = tmp00 ^ tmp01;
    assign path_0_bmc[1] = tmp00 & tmp01;

    assign path_1_bmc[0] = tmp10 ^ tmp11;
    assign path_1_bmc[1] = tmp10 & tmp11;
endmodule
```

ACS

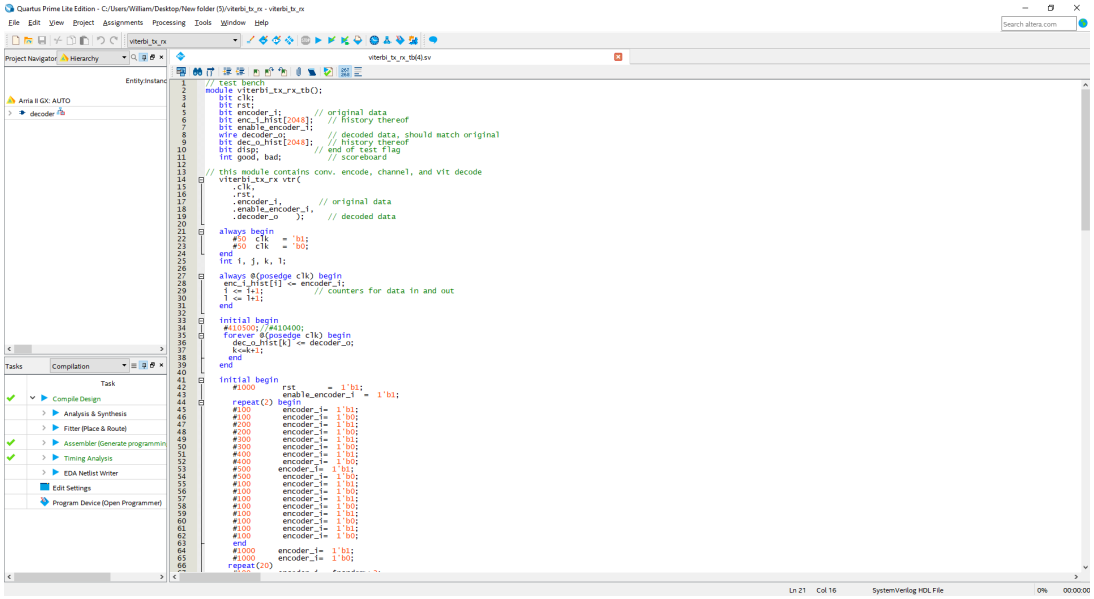
```
1   module ACS                                  // add-compare-select
2   (
3       input       path_0_valid,
4       input       path_1_valid,
5       input [1:0] path_0_bmc,                 // branch metric computation
6       input [1:0] path_1_bmc,
7       input [7:0] path_0_pmc,              // 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
56      end
57
58  endmodule
59
```

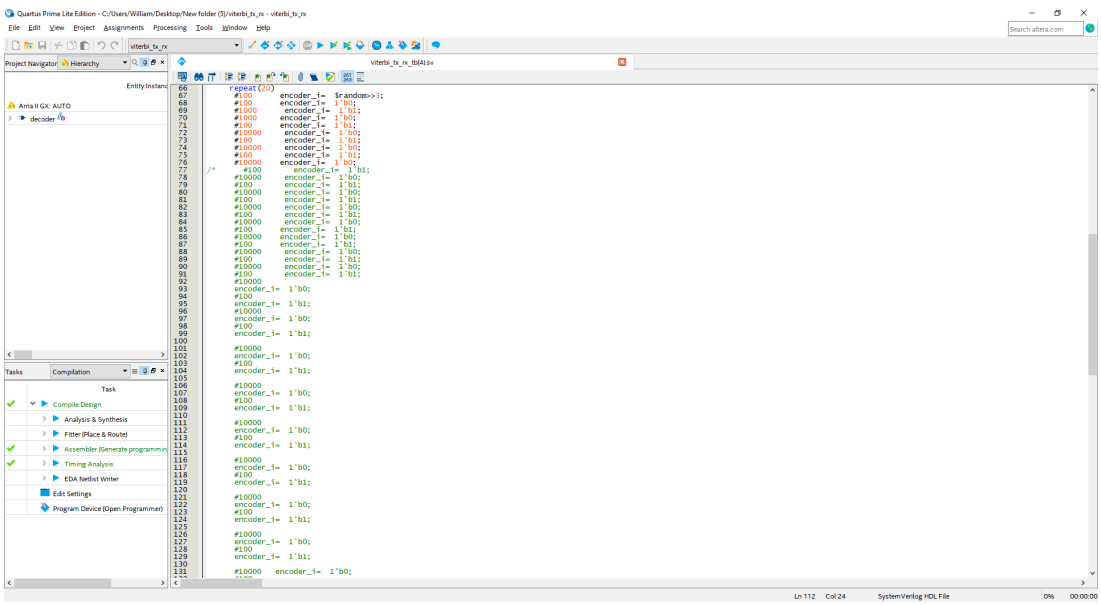
TBU

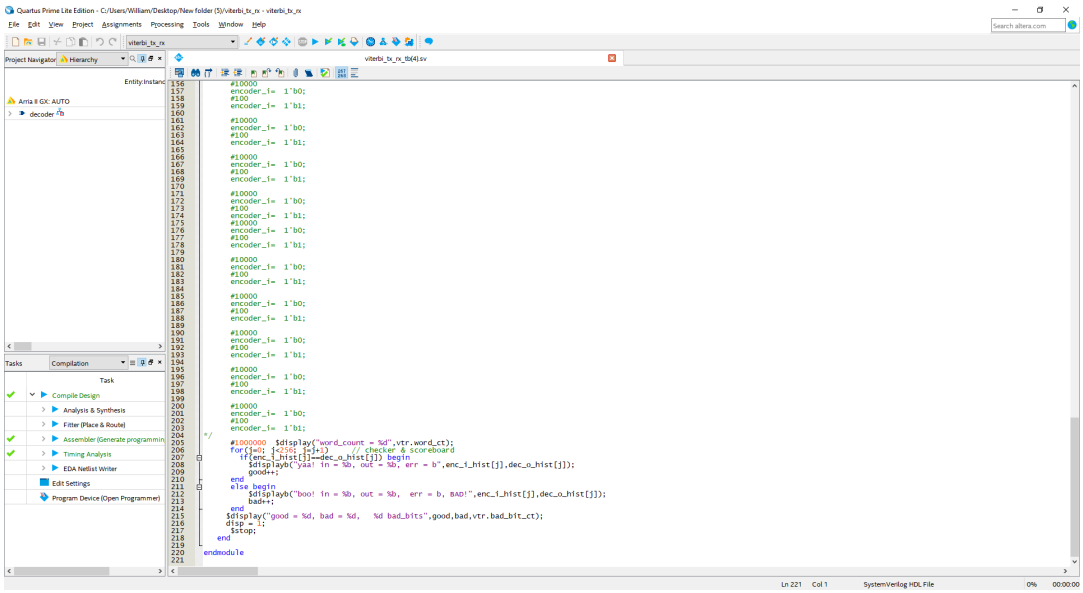



```
63      else if((!selection && d_in_0[pstate]) || (selection && d_in_1[pstate]))
64          case(pstate)
65              3'b000:
66                  nstate = 3'b001;
67              3'b001:
68                  nstate = 3'b010;
69              3'b010:
70                  nstate = 3'b101;
71              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:


Quartus Prime Lite Edition - C:/Users/William/Desktop/New folder (5)/viterbi_tx_rx - viterbi_tx_rx
File Edit View Project Assignments Processing Tools Window Help
viterbi_tx_rx
viterbi_tx_rx_tb(4).sv
Project Navigator Hierarchy
Arria II GX: AUTO
decoder
Tasks Compilation
Task
Compile Design
Analysis & Synthesis
Fitter (Place & Route)
Assembler (Generate programmin
Timing Analysis
EDA Netlist Writer
Edit Settings
Program Device (Open Programmer)
// test bench
module viterbi_tx_rx_tb();
bit clk;
bit rst;
bit encoder_i;
bit enc_i_hist[2048];
bit enable_encoder_i;
wire decoder_o;
bit dec_o_hist[2048];
bit disp;
int good, bad;
// this module contains conv. encode, channel, and vit decode
viterbi_tx_rx vtr(
.clk,
.rst,
.encoder_i,
.enable_encoder_i,
.decoder_o);
always begin
#50 clk = 'b1;
#50 clk = 'b0;
end
int i, j, k, l;
always @(posedge clk) begin
enc_i_hist[i] <= encoder_i;
i <= i+1;
l <= l+1;
end
initial begin
#410500;//#410400;
forever @(posedge clk) begin
dec_o_hist[k] <= decoder_o;
k<=k+1;
end
end
initial begin
#1000 rst = 1'b1;
enable_encoder_i = 1'b1;
repeat(2) begin
#100 encoder_i= 1'b1;
#100 encoder_i= 1'b0;
#200 encoder_i= 1'b1;
#200 encoder_i= 1'b0;
#300 encoder_i= 1'b1;
#300 encoder_i= 1'b0;
#400 encoder_i= 1'b1;
#400 encoder_i= 1'b0;
#500 encoder_i= 1'b1;
#100 encoder_i= 1'b0;
#100 encoder_i= 1'b1;
#100 encoder_i= 1'b0;
#100 encoder_i= 1'b1;
#100 encoder_i= 1'b0;
#100 encoder_i= 1'b1;
#100 encoder_i= 1'b0;
#100 encoder_i= 1'b1;
#100 encoder_i= 1'b0;
end
#1000 encoder_i= 1'b1;
#1000 encoder_i= 1'b0;
repeat(20)
Ln 21 Col 16
SystemVerilog HDL File
0%
00:00:00


Quartus Prime Lite Edition - C:/Users/William/Desktop/New folder (5)/viterbi_tx_rx - viterbi_tx_rx
File Edit View Project Assignments Processing Tools Window Help
viterbi_tx_rx
viterbi_tx_rx_tb(4).sv
Project Navigator Hierarchy
Arria II GX: AUTO
decoder
Tasks Compilation
Task
Compile Design
Analysis & Synthesis
Fitter (Place & Route)
Assembler (Generate programmin
Timing Analysis
EDA Netlist Writer
Edit Settings
Program Device (Open Programmer)
repeat(20)
#100 encoder_i= $random>>3;
#100 encoder_i= 1'b0;
#1000 encoder_i= 1'b1;
#1000 encoder_i= 1'b0;
#100 encoder_i= 1'b1;
#10000 encoder_i= 1'b0;
#100 encoder_i= 1'b1;
#10000 encoder_i= 1'b0;
#100 encoder_i= 1'b1;
#10000 encoder_i= 1'b0;
/* #100 encoder_i= 1'b1;
#10000 encoder_i= 1'b0;
#100 encoder_i= 1'b1;
#10000 encoder_i= 1'b0;
#100 encoder_i= 1'b1;
#10000 encoder_i= 1'b0;
#100 encoder_i= 1'b1;
#10000 encoder_i= 1'b0;
#100 encoder_i= 1'b1;
#10000 encoder_i= 1'b0;
#100 encoder_i= 1'b1;
#10000 encoder_i= 1'b0;
#100 encoder_i= 1'b1;
#10000
encoder_i= 1'b0;
#100
encoder_i= 1'b1;
#10000
encoder_i= 1'b0;
#100
encoder_i= 1'b1;
#10000
encoder_i= 1'b0;
#100
encoder_i= 1'b1;
#10000
encoder_i= 1'b0;
#100
encoder_i= 1'b1;
#10000
encoder_i= 1'b0;
#100
encoder_i= 1'b1;
#10000
encoder_i= 1'b0;
#100
encoder_i= 1'b1;
#10000
encoder_i= 1'b0;
#100
encoder_i= 1'b1;
#10000 encoder_i= 1'b0;
Ln 112 Col 24
SystemVerilog HDL File
0%
00:00:00

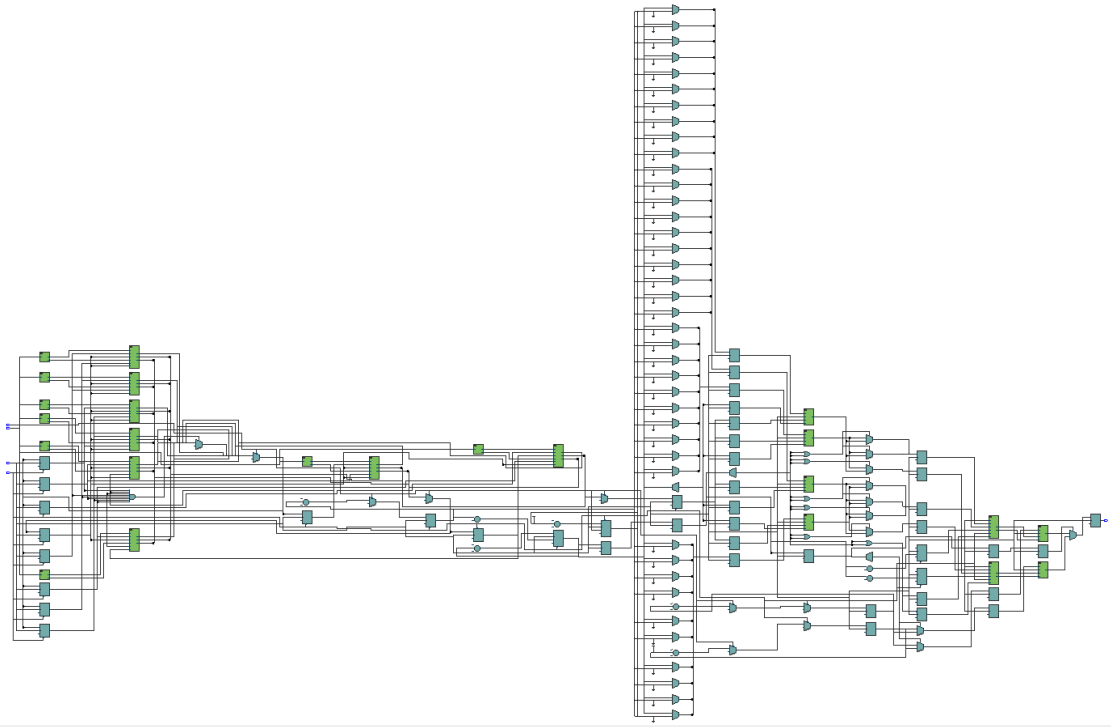


Simulation Transcript

```
# 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

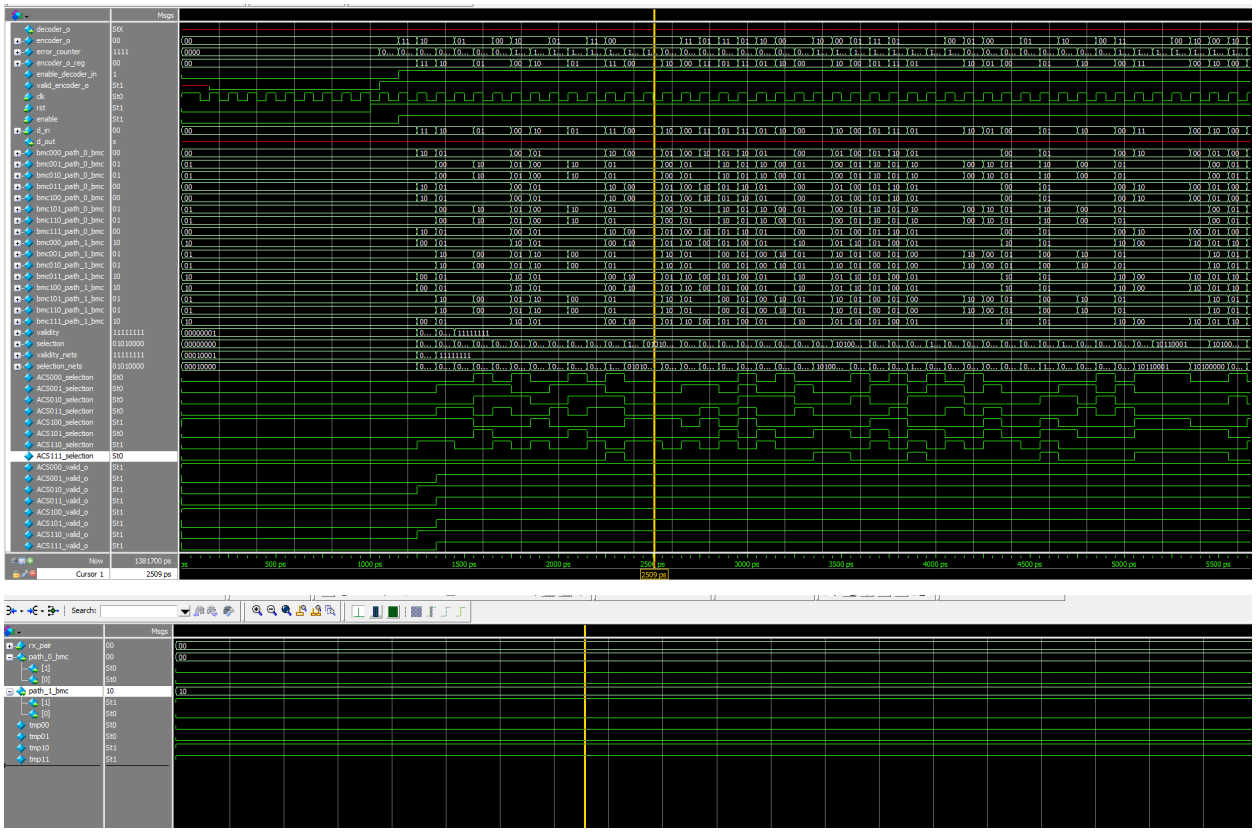
```
# Reading pref.tcl
# Loading project Part 2
# Compile of ACS.sv was successful.
# Compile of bmc000.sv was successful.
# Compile of bmc001.sv was successful.
# Compile of bmc010.sv was successful.
# Compile of bmc011.sv was successful.
# Compile of bmc100.sv was successful.
# Compile of bmc101.sv 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:

```
// test bench
module viterbi_tx_rx_tb();
   bit clk;
   bit rst;
   bit encoder_i;        // original data
   bit enc_i_hist[2048];   // history thereof
   bit enable_encoder_i;
   wire decoder_o;         // decoded data
   bit dec_o_hist[2048];   // history thereof
   bit disp;
   int good, bad;          // scoreboard

// this module contains conv. encode and vit decode
   viterbi_tx_rx vtr(
      .clk,
      .rst,
      .encoder_i,        // original data
      .enable_encoder_i,
      .decoder_o     );     // decoded data

   always begin
      #50   clk    = 1;
      #50  clk    = 0;
   end
   int i, j, k, l;

   always @(posedge clk) begin
    enc_i_hist[i] <= encoder_i;
    i <= i+1;
    l <= l+1;
   end
      initial begin
  #410500;//#410400;
  forever @(posedge clk) begin
    dec_o_hist[k] <= decoder_o;
    k<=k+1;
     end
   end

   initial begin
      clk      =  1'b1;
      #1000
      rst      =  1'b1;
      enable_encoder_i  =  1'b1;
      #100
      encoder_i=  1'b1;
      #100
      encoder_i=  1'b0;
      #100
      encoder_i=  1'b0;
      #100
      encoder_i=  1'b1;
      #100
      encoder_i=  1'b1;
      #100
      encoder_i=  1'b0;
      #100
      encoder_i=  1'b0;
      #100
      encoder_i=  1'b0;
      #100
      encoder_i=  1'b1;
      #100
      encoder_i=  1'b1;
      #100
      encoder_i=  1'b1;
      #100
      encoder_i=  1'b1;
      #100
      encoder_i=  1'b1;
      #100
      encoder_i=  1'b0;
      #100
      encoder_i=  1'b0;
      #100
      encoder_i=  1'b0;
      #100
      encoder_i=  1'b1;
      #100
      encoder_i=  1'b1;
      #100
      encoder_i=  1'b1;
      #100
      encoder_i=  1'b1;
      #100
      encoder_i=  1'b1;
      #100
      encoder_i=  1'b1;
      #100
      encoder_i=  1'b1;
      #100
      encoder_i=  1'b1;
      #100
      encoder_i=  1'b1;
      #100
      encoder_i=  1'b0;
      #100
      encoder_i=  1'b1;
      #100
      encoder_i=  1'b0;
      #100
      encoder_i=  1'b0;
      #100
      encoder_i=  1'b1;
      #100
      encoder_i=  1'b1;
      #100
      encoder_i=  1'b0;
      #100
      encoder_i=  1'b0;
      #100
      encoder_i=  1'b0;
      #100
      encoder_i=  1'b1;
      #100
      encoder_i=  1'b1;
      #100
      encoder_i=  1'b1;
      #100
      encoder_i=  1'b0;
      #100
      encoder_i=  1'b0;
```

```
        encoder_i=  1'b0;
        #100
        encoder_i=  1'b0;
        #100
        encoder_i=  1'b0;
        #100
        encoder_i=  1'b1;
        #100
        encoder_i=  1'b1;
        #100
        encoder_i=  1'b1;
        #100
        encoder_i=  1'b1;
        #100
        encoder_i=  1'b0;
        #100
        encoder_i=  1'b0;
        #100
        encoder_i=  1'b0;
        #100
        encoder_i=  1'b0;
        #100
        encoder_i=  1'b0;
        #100
        encoder_i=  1'b1;
        #100
        encoder_i=  1'b1;
        #100
        encoder_i=  1'b1;
        #100
        encoder_i=  1'b1;
        #100
        encoder_i=  1'b1;

        #10000
        encoder_i=  1'b0;
        #100
        encoder_i=  1'b1;
        #1000
        encoder_i=  1'b0;
        #100
        encoder_i=  1'b1;

        #10000
        encoder_i=  1'b0;
        #100
        encoder_i=  1'b1;


        #10000
        encoder_i=  1'b0;
        #100
        encoder_i=  1'b1;

        #10000
        encoder_i=  1'b0;
        #100
        encoder_i=  1'b1;

        #10000
        encoder_i=  1'b0;
        #100
        encoder_i=  1'b1;

        #10000
        encoder_i=  1'b0;
```

```
        encoder_i=  1'b0;
        #100
        encoder_i=  1'b1;


        #10000
        encoder_i=  1'b0;
        #100
        encoder_i=  1'b1;

        #10000
        encoder_i=  1'b0;
        #100
        encoder_i=  1'b1;

        #10000
        encoder_i=  1'b0;
        #100
        encoder_i=  1'b1;

        #10000
        encoder_i=  1'b0;
        #100
        encoder_i=  1'b1;


        #10000
        encoder_i=  1'b0;
        #100
        encoder_i=  1'b1;

        #10000
        encoder_i=  1'b0;
        #100
        encoder_i=  1'b1;

        #10000
        encoder_i=  1'b0;
        #100
        encoder_i=  1'b1;

        #10000
        encoder_i=  1'b0;
        #100
        encoder_i=  1'b1;


        #10000
        encoder_i=  1'b0;
        #100
        encoder_i=  1'b1;

        #10000
        encoder_i=  1'b0;
        #100
        encoder_i=  1'b1;
```

```
        encoder_i=  1'b1;

        #10000
        encoder_i=  1'b0;
        #100
        encoder_i=  1'b1;

        #10000
        encoder_i=  1'b0;
        #100
        encoder_i=  1'b1;

        #10000
        encoder_i=  1'b0;
        #100
        encoder_i=  1'b1;

        #10000
        encoder_i=  1'b0;
        #100
        encoder_i=  1'b1;

        #10000
        encoder_i=  1'b0;
        #100
        encoder_i=  1'b1;

        #10000
        encoder_i=  1'b0;
        #100
        encoder_i=  1'b1;

        #10000
        encoder_i=  1'b0;
        #100
        encoder_i=  1'b1;


        #10000
        encoder_i=  1'b0;
        #100
        encoder_i=  1'b1;

        #10000
        encoder_i=  1'b0;
        #100
        encoder_i=  1'b1;

        #10000
        encoder_i=  1'b0;
        #100
        encoder_i=  1'b1;

        #10000
        encoder_i=  1'b0;
        #100
        encoder_i=  1'b1;

        #10000
        encoder_i=  1'b0;
        #100
        encoder_i=  1'b1;

        #10000
        encoder_i=  1'b0;
        #100
        encoder_i=  1'b1;
        #10000
        encoder_i=  1'b0;
        #100
        encoder_i=  1'b1;

        #10000
        encoder_i=  1'b0;
        #100
        encoder_i=  1'b1;

        #10000
        encoder_i=  1'b0;
        #100
        encoder_i=  1'b1;

        #10000
        encoder_i=  1'b0;
        #100
        encoder_i=  1'b1;

        #10000
        encoder_i=  1'b0;
        #100
        encoder_i=  1'b1;

        #10000
        encoder_i=  1'b0;
        #100
        encoder_i=  1'b1;

        #1000000
        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",enc_i_hist[j],dec_o_hist[j]);
            good++;
        end
        else begin
            $displayb("boo! in = %b, out = %b",enc_i_hist[j],dec_o_hist[j]);
            bad++;
        end
       $display("good = %d, bad = %d",good,bad);
       disp = 1;

        $stop;
    end

endmodule
```

Part A

2.a.1

```
module viterbi_tx_rx(
    input    clk,
    input    rst,
    input    encoder_i,
    input    enable_encoder_i,
    output   decoder_o);

    wire  [1:0] encoder_o;

    logic   [3:0] error_counter;
    logic   [1:0] encoder_o_reg;

    logic       enable_decoder_in;
    wire        valid_encoder_o;

    //Logic for error for bit 0 and 1
    logic [2:0] error_bit8;
    logic [4:0] error_bit32;
    int bad_bit_ct = 1;
    int word_ct;


    always @ (posedge clk, negedge rst)
       if(!rst) begin
           error_counter  <= 4'd0;
           encoder_o_reg  <= 2'b00;
           enable_decoder_in <= 1'b0;
           error_bit8 <= 3'd0;
           error_bit32 <= 5'd0;
            word_ct = 0;
       end
       else     begin
           enable_decoder_in <= valid_encoder_o;
           encoder_o_reg  <= 2'b00;
           error_counter  <= error_counter + 4'd1;
           error_bit8 <= error_bit8 + 3'd1;
           error_bit32 <= error_bit32 + 5'd1;
            word_ct           <= word_ct + 1;

         if(error_bit8==3'b111)begin //2a1 # good =         256, bad =           0 , Injected : 32
             encoder_o_reg  <= {encoder_o[1],~encoder_o[0]};  // inject one bad bit out of every 8
                if(word_ct<256)begin
                bad_bit_ct      <= bad_bit_ct + 1;
                $display("error_counter = %d",bad_bit_ct);
              end
           end

           else
               encoder_o_reg  <= {encoder_o[1],encoder_o[0]};
       end

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

```
# 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
```

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

endmodule
```

2.a.2

```
module viterbi_tx_rx(
    input    clk,
    input    rst,
    input    encoder_i,
    input    enable_encoder_i,
    output   decoder_o);

    wire  [1:0] encoder_o;

    logic   [3:0] error_counter;
    logic   [1:0] encoder_o_reg;

    logic       enable_decoder_in;
    wire        valid_encoder_o;

    //Logic for error for bit 0 and 1
    logic [2:0] error_bit8;
    logic [4:0] error_bit32;
    int bad_bit_ct = 1;
    int word_ct;


    always @ (posedge clk, negedge rst)
       if(!rst) begin
          error_counter  <= 4'd0;
          encoder_o_reg  <= 2'b00;
          enable_decoder_in <= 1'b0;
          error_bit8 <= 3'd0;
          error_bit32 <= 5'd0;
           word_ct = 0;
       end
       else     begin
          enable_decoder_in <= valid_encoder_o;
          encoder_o_reg  <= 2'b00;
          error_counter  <= error_counter + 4'd1;
          error_bit8 <= error_bit8 + 3'd1;
          error_bit32 <= error_bit32 + 5'd1;
           word_ct              <= word_ct + 1;

          if(error_bit8==3'b111)begin //2a2 # good =         256, bad =           0 , Injected : 32
             encoder_o_reg  <= {~encoder_o[1],encoder_o[0]};  // inject one bad bit out of every 8
              if(word_ct<256)begin
                bad_bit_ct      <= bad_bit_ct + 1;
                $display("error_counter = %d",bad_bit_ct);
             end
          end

          else
             encoder_o_reg  <= {encoder_o[1],encoder_o[0]};
       end

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

```
       end

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

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

 endmodule
```

```
# 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

```
module viterbi_tx_rx(
    input    clk,
    input    rst,
    input    encoder_i,
    input    enable_encoder_i,
    output   decoder_o);

    wire  [1:0] encoder_o;

    logic   [3:0] error_counter;
    logic   [1:0] encoder_o_reg;

    logic       enable_decoder_in;
    wire        valid_encoder_o;

    //Logic for error for bit 0 and 1
    logic [2:0] error_bit8;
    logic [4:0] error_bit32;
    int bad_bit_ct = 2;
    int word_ct;


    always @ (posedge clk, negedge rst)
       if(!rst) begin
           error_counter  <= 4'd0;
           encoder_o_reg  <= 2'b00;
           enable_decoder_in <= 1'b0;
           error_bit8 <= 3'd0;
           error_bit32 <= 5'd0;
            word_ct = 0;
       end
       else     begin
           enable_decoder_in <= valid_encoder_o;
           encoder_o_reg  <= 2'b00;
           error_counter  <= error_counter + 4'd1;
           error_bit8 <= error_bit8 + 3'd1;
           error_bit32 <= error_bit32 + 5'd1;
            word_ct             <= word_ct + 1;

           if(error_bit8==3'b111)begin //2a3 # good =         205, bad =          51 , Injected = 64
              encoder_o_reg  <= {~encoder_o[1],~encoder_o[0]};    // inject one bad bit for 1 and 0 out of every 8
               if(word_ct<256)begin
                   bad_bit_ct      <= bad_bit_ct + 2;
                   $display("error_counter = %d",bad_bit_ct);
               end
           end

           else
              encoder_o_reg  <= {encoder_o[1],encoder_o[0]};
       end

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

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

  endmodule
```

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

```
module viterbi_tx_rx(
    input    clk,
    input    rst,
    input    encoder_i,
    input    enable_encoder_i,
    output   decoder_o);

    wire  [1:0] encoder_o;

    logic   [3:0] error_counter;
    logic   [1:0] encoder_o_reg;

    logic       enable_decoder_in;
    wire        valid_encoder_o;

    //Logic for error for bit 0 and 1
    logic [2:0] error_bit8;
    logic [4:0] error_bit32;
    int bad_bit_ct = 1;
    int word_ct;


    always @ (posedge clk, negedge rst)
        if(!rst) begin
            error_counter  <= 4'd0;
            encoder_o_reg  <= 2'b00;
            enable_decoder_in <= 1'b0;
            error_bit8 <= 3'd0;
            error_bit32 <= 5'd0;
              word_ct = 0;
        end
        else     begin
            enable_decoder_in <= valid_encoder_o;
            encoder_o_reg  <= 2'b00;
            error_counter  <= error_counter + 4'd1;
            error_bit8 <= error_bit8 + 3'd1;
            error_bit32 <= error_bit32 + 5'd1;
              word_ct           <= word_ct + 1;

          if(error_counter == 4'b1001 || error_counter == 4'b1000 ) begin  //2a4 # good =     256, bad =      0 , Injected : 32
              encoder_o_reg  <= {encoder_o[1],~encoder_o[0]};  // inject two bad bit out of every 16
                  if(word_ct<256)begin
                      bad_bit_ct      <= bad_bit_ct + 1;
                      $display("error_counter = %d",bad_bit_ct);
                end
          end

           else
              encoder_o_reg  <= {encoder_o[1],encoder_o[0]};
        end

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

```
# error_counter =           29
# error_counter =           30
# error_counter =           31
# error_counter =           32
# yaa! in = 0, out = 0
# yaa! 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

```
module viterbi_tx_rx(
    input    clk,
    input    rst,
    input    encoder_i,
    input    enable_encoder_i,
    output   decoder_o);

    wire  [1:0] encoder_o;

    logic   [3:0] error_counter;
    logic   [1:0] encoder_o_reg;

    logic       enable_decoder_in;
    wire        valid_encoder_o;

    //Logic for error for bit 0 and 1
    logic [2:0] error_bit8;
    logic [4:0] error_bit32;
    int bad_bit_ct = 1;
    int word_ct;


    always @ (posedge clk, negedge rst)
       if(!rst) begin
           error_counter  <= 4'd0;
           encoder_o_reg  <= 2'b00;
           enable_decoder_in <= 1'b0;
           error_bit8 <= 3'd0;
           error_bit32 <= 5'd0;
            word_ct = 0;
       end
       else     begin
           enable_decoder_in <= valid_encoder_o;
           encoder_o_reg  <= 2'b00;
           error_counter  <= error_counter + 4'd1;
           error_bit8 <= error_bit8 + 3'd1;
           error_bit32 <= error_bit32 + 5'd1;
            word_ct                 <= word_ct + 1;

             if(error_counter == 4'b1001 || error_counter == 4'b1000 ) begin  //2a5 # good =     256, bad =      0 , Injected : 32
              encoder_o_reg  <= {~encoder_o[1],encoder_o[0]};  // inject two bad bit out of every 16
                 if(word_ct<256)begin
                    bad_bit_ct      <= bad_bit_ct + 1;
                    $display("error_counter = %d",bad_bit_ct);
               end
          end

           else
              encoder_o_reg  <= {encoder_o[1],encoder_o[0]};
       end

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

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

endmodule
```

```
# error_counter =           29
# error_counter =           30
# error_counter =           31
# error_counter =           32
# yaa! in = 0, out = 0
# yaa! 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

```
module viterbi_tx_rx(
    input    clk,
    input    rst,
    input    encoder_i,
    input    enable_encoder_i,
    output   decoder_o);

    wire  [1:0] encoder_o;

    logic   [3:0] error_counter;
    logic   [1:0] encoder_o_reg;

    logic       enable_decoder_in;
    wire        valid_encoder_o;

    //Logic for error for bit 0 and 1
    logic [2:0] error_bit8;
    logic [4:0] error_bit32;
    int bad_bit_ct = 1;
    int word_ct;


    always @ (posedge clk, negedge rst)
       if(!rst) begin
           error_counter  <= 4'd0;
           encoder_o_reg  <= 2'b00;
           enable_decoder_in <= 1'b0;
           error_bit8 <= 3'd0;
           error_bit32 <= 5'd0;
            word_ct = 0;
       end
       else     begin
           enable_decoder_in <= valid_encoder_o;
           encoder_o_reg  <= 2'b00;
           error_counter  <= error_counter + 4'd1;
           error_bit8 <= error_bit8 + 3'd1;
           error_bit32 <= error_bit32 + 5'd1;
            word_ct                 <= word_ct + 1;

           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
              encoder_o_reg  <= {encoder_o[1],~encoder_o[0]};  // inject 4 in a row bad bit out of every 32
                  if(word_ct<256)begin
                    bad_bit_ct     <= bad_bit_ct + 1;
                     $display("error_counter = %d",bad_bit_ct);
                   end
              end
           else
              encoder_o_reg  <= {encoder_o[1],encoder_o[0]};
       end

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


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

  endmodule
```

```
# 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

```
module viterbi_tx_rx(
    input    clk,
    input    rst,
    input    encoder_i,
    input    enable_encoder_i,
    output   decoder_o);

    wire  [1:0] encoder_o;

    logic   [3:0] error_counter;
    logic   [1:0] encoder_o_reg;

    logic       enable_decoder_in;
    wire        valid_encoder_o;

    //Logic for error for bit 0 and 1
    logic [2:0] error_bit8;
    logic [4:0] error_bit32;
    int bad_bit_ct = 1;
    int word_ct;


    always @ (posedge clk, negedge rst)
        if(!rst) begin
            error_counter  <= 4'd0;
            encoder_o_reg  <= 2'b00;
            enable_decoder_in <= 1'b0;
            error_bit8 <= 3'd0;
            error_bit32 <= 5'd0;
             word_ct = 0;
        end
        else     begin
            enable_decoder_in <= valid_encoder_o;
            encoder_o_reg  <= 2'b00;
            error_counter  <= error_counter + 4'd1;
            error_bit8 <= error_bit8 + 3'd1;
            error_bit32 <= error_bit32 + 5'd1;
             word_ct              <= word_ct + 1;

            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
               encoder_o_reg  <= {~encoder_o[1],encoder_o[0]};  // inject 4 in a row bad bit out of every 32
                if(word_ct<256)begin
                  bad_bit_ct     <= bad_bit_ct + 1;
                  $display("error_counter = %d",bad_bit_ct);
                end
            end

            else
                encoder_o_reg  <= {encoder_o[1],encoder_o[0]};
        end

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

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

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

```
module viterbi_tx_rx(
    input    clk,
    input    rst,
    input    encoder_i,
    input    enable_encoder_i,
    output   decoder_o);

    wire  [1:0] encoder_o;

    logic   [3:0] error_counter;
    logic   [1:0] encoder_o_reg;

    logic       enable_decoder_in;
    wire        valid_encoder_o;

    //Logic for error for bit 0 and 1
    logic [2:0] error_bit8;
    logic [4:0] error_bit32;
    int bad_bit_ct = 2;
    int word_ct;


    always @ (posedge clk, negedge rst)
       if(!rst) begin
           error_counter  <= 4'd0;
           encoder_o_reg  <= 2'b00;
           enable_decoder_in <= 1'b0;
           error_bit8 <= 3'd0;
           error_bit32 <= 5'd0;
            word_ct = 0;
       end
       else     begin
           enable_decoder_in <= valid_encoder_o;
           encoder_o_reg  <= 2'b00;
           error_counter  <= error_counter + 4'd1;
           error_bit8 <= error_bit8 + 3'd1;
           error_bit32 <= error_bit32 + 5'd1;
            word_ct                <= word_ct + 1;

           if(error_bit32 == 5'b01010 || error_bit32 == 5'b01011  ) begin //2a8 # good =         232, bad =          24 Injected: 32
              encoder_o_reg  <= {~encoder_o[1],~encoder_o[0]};    // inject 2 in a row bad bit out of every 32
                  if(word_ct<256)begin
                    bad_bit_ct     <= bad_bit_ct + 2;
                     $display("error_counter = %d",bad_bit_ct);
                   end
             end

           else
              encoder_o_reg  <= {encoder_o[1],encoder_o[0]};
       end

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

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

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

```
// test bench
module viterbi_tx_rx_tb();
    bit clk;
    bit rst;
    bit encoder_i;          // original data
    bit enc_i_hist[2048];   // history thereof
    bit enable_encoder_i;
    wire decoder_o;         // decoded data, should match original
    bit dec_o_hist[2048];   // history thereof
    bit disp;               // end of test flag
    int good, bad;          // scoreboard

// this module contains conv. encode, channel, and vit decode
    viterbi_tx_rx vtr(
        .clk,
        .rst,
        .encoder_i,         // original data
        .enable_encoder_i,
        .decoder_o     );   // decoded data

    always begin
        #50  clk   = 'b1;
        #50  clk   = 'b0;
    end
    int i, j, k, l;

    always @(posedge clk) begin
     enc_i_hist[i] <= encoder_i;
      i <= i+1;             // counters for data in and out
      l <= l+1;
    end

    initial begin
     #410500;//#410400;
     forever @(posedge clk) begin
        dec_o_hist[k] <= decoder_o;
        k<=k+1;
      end
    end

    initial begin
        #1000       rst       =  1'b1;
                    enable_encoder_i   =  1'b1;
        repeat(2) begin
        #100        encoder_i=  1'b1;
        #100        encoder_i=  1'b0;
        #200        encoder_i=  1'b1;
        #200        encoder_i=  1'b0;
        #300        encoder_i=  1'b1;
        #300        encoder_i=  1'b0;
        #400        encoder_i=  1'b1;
        #400        encoder_i=  1'b0;
        #500       encoder_i=  1'b1;
        #500        encoder_i=  1'b0;
        #100        encoder_i=  1'b1;
        #100        encoder_i=  1'b0;
        #100        encoder_i=  1'b1;
        #100        encoder_i=  1'b0;
        #100        encoder_i=  1'b1;
        #100        encoder_i=  1'b0;
        #100        encoder_i=  1'b1;
        #100        encoder_i=  1'b0;
        end
        #1000       encoder_i=  1'b1;
        #1000       encoder_i=  1'b0;
      repeat(20)
```

```
      repeat(20)
        #100        encoder_i=  $random>>3;
        #100        encoder_i=  1'b0;
        #1000        encoder_i=  1'b1;
        #1000       encoder_i=  1'b0;
        #100        encoder_i=  1'b1;
        #10000       encoder_i=  1'b0;
        #100         encoder_i=  1'b1;
        #10000       encoder_i=  1'b0;
        #100         encoder_i=  1'b1;
        #10000      encoder_i=  1'b0;
/*          #100        encoder_i=  1'b1;
        #10000       encoder_i=  1'b0;
        #100         encoder_i=  1'b1;
        #10000       encoder_i=  1'b0;
        #100         encoder_i=  1'b1;
        #10000       encoder_i=  1'b0;
        #100         encoder_i=  1'b1;
        #10000       encoder_i=  1'b0;
        #100        encoder_i=  1'b1;
        #10000      encoder_i=  1'b0;
        #100        encoder_i=  1'b1;
        #10000       encoder_i=  1'b0;
        #100         encoder_i=  1'b1;
        #10000       encoder_i=  1'b0;
        #100         encoder_i=  1'b1;
        #10000
        encoder_i=  1'b0;
        #100
        encoder_i=  1'b1;
        #10000
        encoder_i=  1'b0;
        #100
        encoder_i=  1'b1;

        #10000
        encoder_i=  1'b0;
        #100
        encoder_i=  1'b1;

        #10000
        encoder_i=  1'b0;
        #100
        encoder_i=  1'b1;

        #10000
        encoder_i=  1'b0;
        #100
        encoder_i=  1'b1;

        #10000
        encoder_i=  1'b0;
        #100
        encoder_i=  1'b1;

        #10000
        encoder_i=  1'b0;
        #100
        encoder_i=  1'b1;

        #10000
        encoder_i=  1'b0;
        #100
        encoder_i=  1'b1;

        #10000
        encoder_i=  1'b0;
        #100
        encoder_i=  1'b1;

        #10000   encoder_i=  1'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.

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

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

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

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

```

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

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

  endmodule
```

```
# 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.

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

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

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

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

```

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

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

  endmodule

```

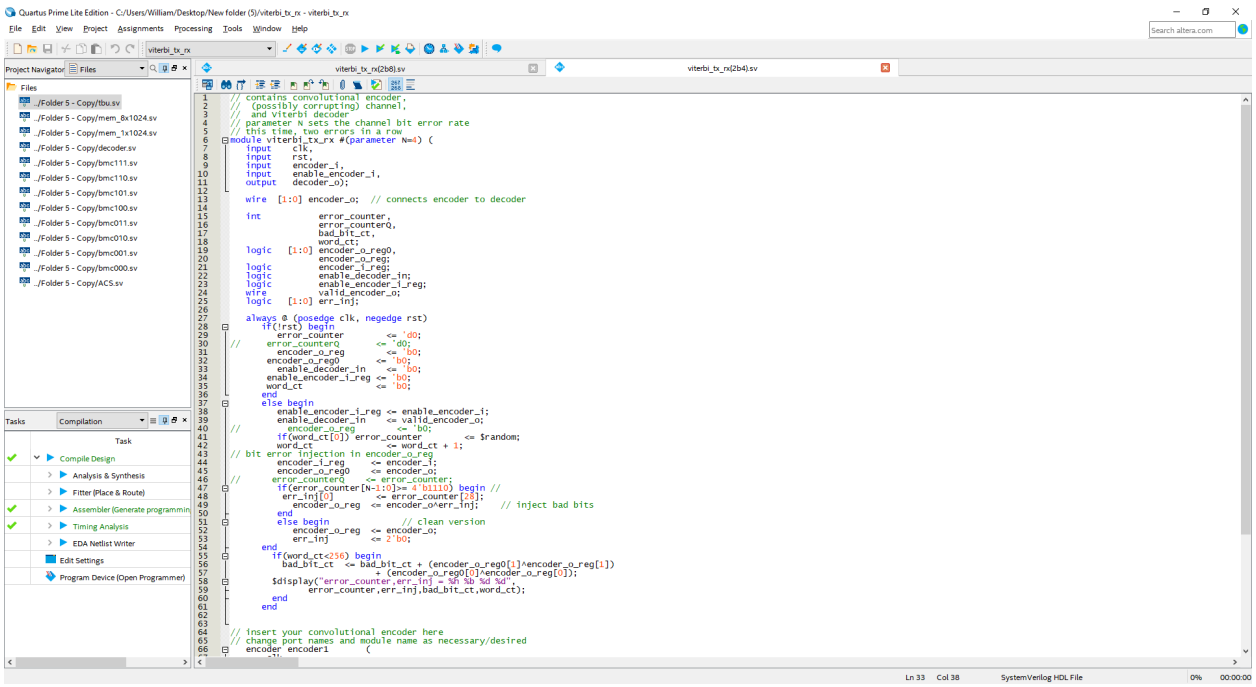
```
# good =          256, bad =            0,         8 bad_bits
# ** Note: $stop    : H:/Folder 5 - Copy/viterbi_tx_rx_tb (2b).sv(217)
#    Time: 1045 ns  Iteration: 0  Instance: /viterbi_tx_rx_tb
# Break in Module viterbi_tx_rx_tb at H:/Folder 5 - Copy/viterbi_tx_rx_tb (2b).sv line 217
```

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
```

```
// contains convolutional encoder,
//  (possibly corrupting) channel,
//  and viterbi decoder
// parameter N sets the channel bit error rate
// this time, two errors in a row
module viterbi_tx_rx #(parameter N=4) (
    input    clk,
    input    rst,
    input    encoder_i,
    input    enable_encoder_i,
    output   decoder_o);

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

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

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

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

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

endmodule
```

2b4.




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

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

endmodule

```

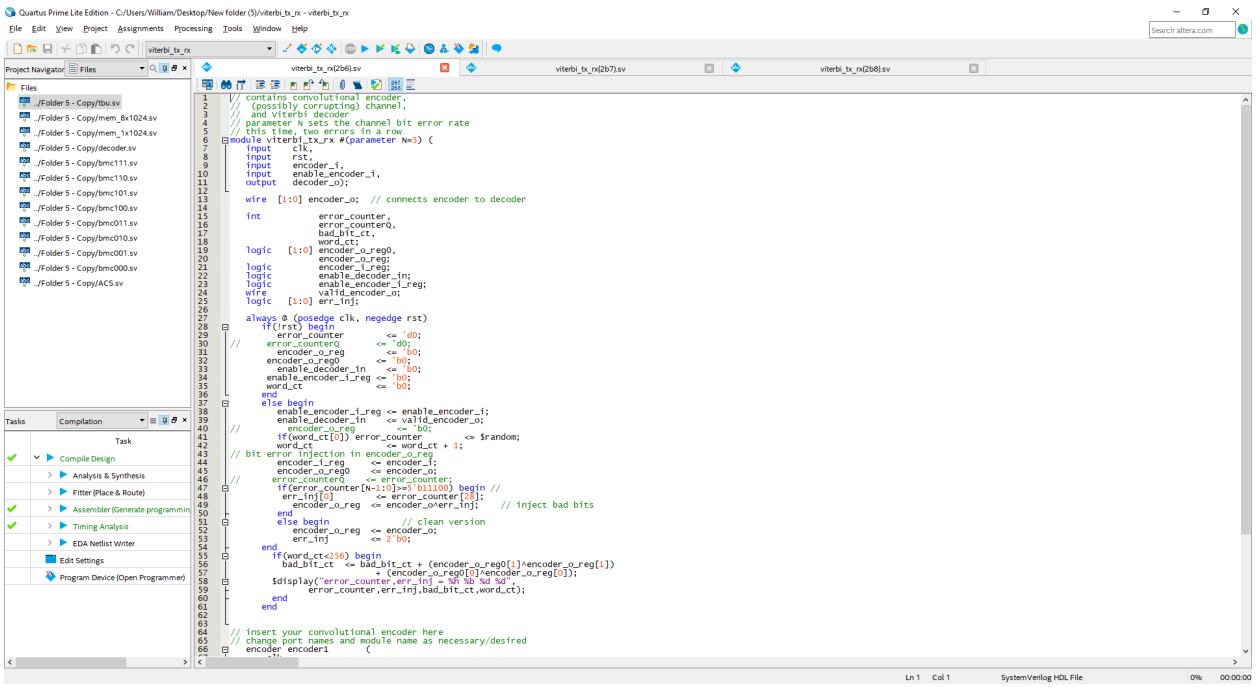
```
# good =          256, bad =            0,             4 bad_bits
# ** Note: $stop    : H:/Folder 5 - Copy/viterbi_tx_rx_tb (2b).sv(217)
#    Time: 1045 ns  Iteration: 0  Instance: /viterbi_tx_rx_tb
# Break in Module viterbi_tx_rx_tb at H:/Folder 5 - Copy/viterbi_tx_rx_tb (2b).sv line 217
```

2b5.

```
// contains convolutional encoder,
//  (possibly corrupting) channel,
//  and viterbi decoder
// parameter N sets the channel bit error rate
// this time, two errors in a row
module viterbi_tx_rx #(parameter N=4) (
    input    clk,
    input    rst,
    input    encoder_i,
    input    enable_encoder_i,
    output   decoder_o);

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

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

    always @ (posedge clk, negedge rst)
       if(!rst) begin
          error_counter        <= 'd0;
//        error_counterQ       <= 'd0;
          encoder_o_reg        <= 'b0;
        encoder_o_reg0         <= 'b0;
          enable_decoder_in    <= 'b0;
        enable_encoder_i_reg <= 'b0;
        word_ct                <= 'b0;
       end
       else begin
          enable_encoder_i_reg <= enable_encoder_i;
          enable_decoder_in    <= valid_encoder_o;
//          encoder_o_reg       <= 'b0;
          if(word_ct[0]) error_counter        <= $random;
          word_ct                <= word_ct + 1;
// bit error injection in encoder_o_reg
          encoder_i_reg     <= encoder_i;
          encoder_o_reg0    <= encoder_o;
//        error_counterQ    <= error_counter;
          if(error_counter[N-1:0]=='b1110) begin //
            err_inj[1]       <= error_counter[29];
              encoder_o_reg  <= encoder_o^err_inj;    // inject bad bits
          end
          else begin              // clean version
              encoder_o_reg  <= encoder_o;
              err_inj        <= 2'b0;
       end
          if(word_ct<256) begin
            bad_bit_ct  <= bad_bit_ct + (encoder_o_reg0[1]^encoder_o_reg[1])
                              + (encoder_o_reg0[0]^encoder_o_reg[0]);
         $display("error_counter,err_inj = %h %b %d %d",
                  error_counter,err_inj,bad_bit_ct,word_ct);
          end
       end

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

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

endmodule
```

```
# 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
```

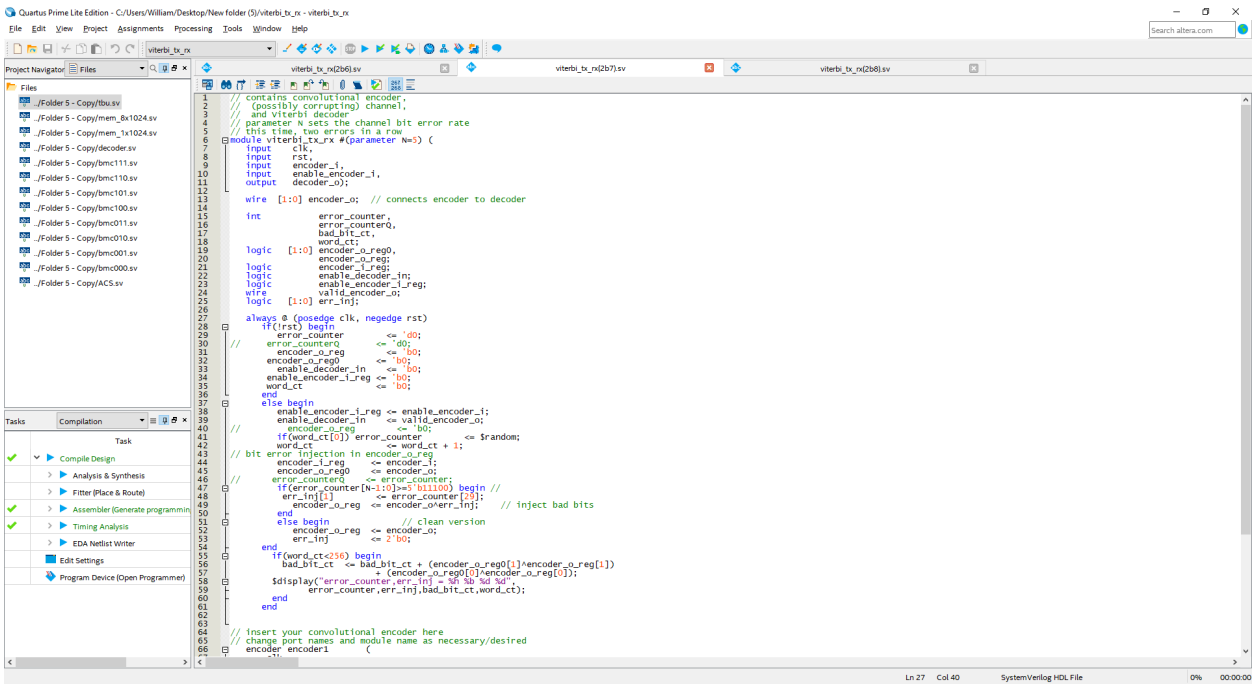
2b6.




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

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

endmodule
```

```
# 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
```

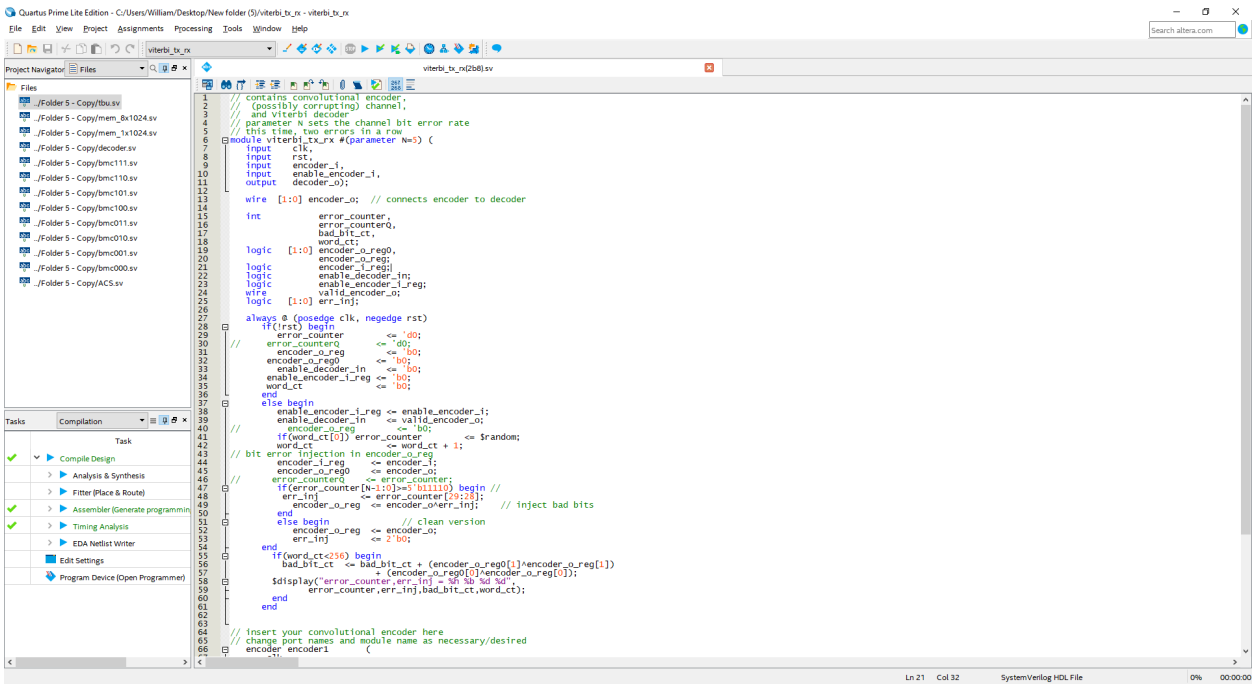
2b7.




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

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

endmodule
```

```
# 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.




```
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),        //_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,              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

```
module viterbi_tx_rx(
    input    clk,
    input    rst,
    input    encoder_i,
    input    enable_encoder_i,
    output   decoder_o);

    wire  [1:0] encoder_o;

    logic   [3:0] error_counter;
    logic   [1:0] encoder_o_reg;

    logic       enable_decoder_in;
    wire        valid_encoder_o;

    //Logic for error for bit 0 and 1
    logic [2:0] error_bit8;
    logic [4:0] error_bit32;
    int bad_bit_ct = 1;
    int word_ct;


    always @ (posedge clk, negedge rst)
        if(!rst) begin
            error_counter  <= 4'd0;
            encoder_o_reg  <= 2'b00;
            enable_decoder_in <= 1'b0;
            error_bit8 <= 3'd0;
            error_bit32 <= 5'd0;
             word_ct = 0;
        end
        else     begin
            enable_decoder_in <= valid_encoder_o;
            encoder_o_reg  <= 2'b00;
            error_counter  <= error_counter + 4'd1;
            error_bit8 <= error_bit8 + 3'd1;
            error_bit32 <= error_bit32 + 5'd1;
             word_ct              <= word_ct + 1;


            if(error_counter == 4'b1001 || error_counter == 4'b1000 || error_counter == 4'b0111 || error_counter == 4'b0110 ) begin
            //2c //with 3 consecutive # good =256, bad=0 // with 4 consecutive  good =     181, bad =          75 BROKE THE CODE
                encoder_o_reg  <= {encoder_o[1],~encoder_o[0]};  // inject 4 bad bit out of every 16
                    if(word_ct<256)begin
                        bad_bit_ct      <= bad_bit_ct + 1;
                        $display("error_counter = %d",bad_bit_ct);
                 end
            end


            else
                encoder_o_reg  <= {encoder_o[1],encoder_o[0]};
        end

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

```

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

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

endmodule
```

```
# error_counter =           62
# error_counter =           63
# error_counter =           64
```

```
# boo! in = 1, out = 0
# good =         181, bad =          75
# ** Note: $stop    : C:/Users/William/Desktop/Folder 5/viterbi_tx_rx_tb(4).sv(371)
#    Time: 1381700 ps  Iteration: 0  Instance: /viterbi_tx_rx_tb
# Break in Module viterbi_tx_rx_tb at C:/Users/William/Desktop/Folder 5/viterbi_tx_rx_tb(4).sv line 371

VSIM 6>
```

Part D

Testbench: same as part A

```
module viterbi_tx_rx(
    input    clk,
    input    rst,
    input    encoder_i,
    input    enable_encoder_i,
    output   decoder_o);

    wire  [1:0] encoder_o;

    logic   [3:0] error_counter;
    logic   [1:0] encoder_o_reg;

    logic       enable_decoder_in;
    wire        valid_encoder_o;

    //Logic for error for bit 0 and 1
    logic [2:0] error_bit8;
    logic [4:0] error_bit32;
    int bad_bit_ct = 1;
    int word_ct;


    always @ (posedge clk, negedge rst)
       if(!rst) begin
           error_counter  <= 4'd0;
           encoder_o_reg  <= 2'b00;
           enable_decoder_in <= 1'b0;
           error_bit8 <= 3'd0;
           error_bit32 <= 5'd0;
            word_ct = 0;
       end
       else     begin
           enable_decoder_in <= valid_encoder_o;
           encoder_o_reg  <= 2'b00;
           error_counter  <= error_counter + 4'd1;
           error_bit8 <= error_bit8 + 3'd1;
           error_bit32 <= error_bit32 + 5'd1;
            word_ct             <= word_ct + 1;


        if(error_counter == 4'b1001 || error_counter == 4'b1000 || error_counter == 4'b0111 || error_counter == 4'b0110 ) begin
         //2d //with 3 consecutive # good =256, bad=0 // with 4 consecutive  good =     226, bad =     30 BROKE THE CODE
            encoder_o_reg  <= {~encoder_o[1],encoder_o[0]};  // inject 4 bits every 16
                if(word_ct<256)begin
                    bad_bit_ct      <= bad_bit_ct + 1;
                    $display("error_counter = %d",bad_bit_ct);
             end
        end

         else
            encoder_o_reg  <= {encoder_o[1],encoder_o[0]};
       end

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

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

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

endmodule
```

```
# error_counter =           61
# error_counter =           62
# error_counter =           63
# error_counter =           64
```

```
# boo! in = 1, out = 0
# good =         226, bad =          30
# ** Note: $stop    : C:/Users/William/Desktop/Folder 5/viterbi_tx_rx_tb(4).sv(371)
#    Time: 1381700 ps  Iteration: 0  Instance: /viterbi_tx_rx_tb
# 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

```
module viterbi_tx_rx(
    input    clk,
    input    rst,
    input    encoder_i,
    input    enable_encoder_i,
    output   decoder_o);

    wire  [1:0] encoder_o;

    logic   [3:0] error_counter;
    logic   [1:0] encoder_o_reg;

    logic       enable_decoder_in;
    wire        valid_encoder_o;

    //Logic for error for bit 0 and 1
    logic [2:0] error_bit8;
    logic [4:0] error_bit32;
    int bad_bit_ct = 2;
    int word_ct;


    always @ (posedge clk, negedge rst)
        if(!rst) begin
            error_counter  <= 4'd0;
            encoder_o_reg  <= 2'b00;
            enable_decoder_in <= 1'b0;
            error_bit8 <= 3'd0;
            error_bit32 <= 5'd0;
             word_ct = 0;
        end
        else     begin
            enable_decoder_in <= valid_encoder_o;
            encoder_o_reg  <= 2'b00;
            error_counter  <= error_counter + 4'd1;
            error_bit8 <= error_bit8 + 3'd1;
            error_bit32 <= error_bit32 + 5'd1;
             word_ct              <= word_ct + 1;


                if(error_counter == 4'b1001 || error_counter == 4'b1000 )begin
                //2e # with 2 injected every 16 good =     256, bad =     0 , Injected = 32 // With 4 consecutive,  good =   211, bad =    45   BROKE THE CODE
              encoder_o_reg  <= {~encoder_o[1],~encoder_o[0]};
                if(word_ct<256)begin
                    bad_bit_ct     <= bad_bit_ct + 2;
                    $display("error_counter = %d",bad_bit_ct);
                end
            end

            else
                encoder_o_reg  <= {encoder_o[1],encoder_o[0]};
        end

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

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

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

endmodule
```

```
# error_counter =           22
# error_counter =           24
# error_counter =           26
# error_counter =           28
# error_counter =           30
# error_counter =           32
# error_counter =           34
# error_counter =           36
# error_counter =           38
# error_counter =           40
# error_counter =           42
# error_counter =           44
# error_counter =           46
# error_counter =           48
# error_counter =           50
# error_counter =           52
# error_counter =           54
# error_counter =           56
# error_counter =           58
# error_counter =           60
# error_counter =           62
# error_counter =           64
```

```
# yaa! in = 1, out = 1
# good =          211, bad =           45
# ** Note: $stop    : C:/Users/William/Downloads/viterbi_tx_rx_tb(4).sv(371)
#    Time: 1381700 ps  Iteration: 0  Instance: /viterbi_tx_rx_tb
# Break in Module viterbi_tx_rx_tb at C:/Users/William/Downloads/viterbi_tx_rx_tb(4).sv line 371

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.