Christina Mai
Anne Lin

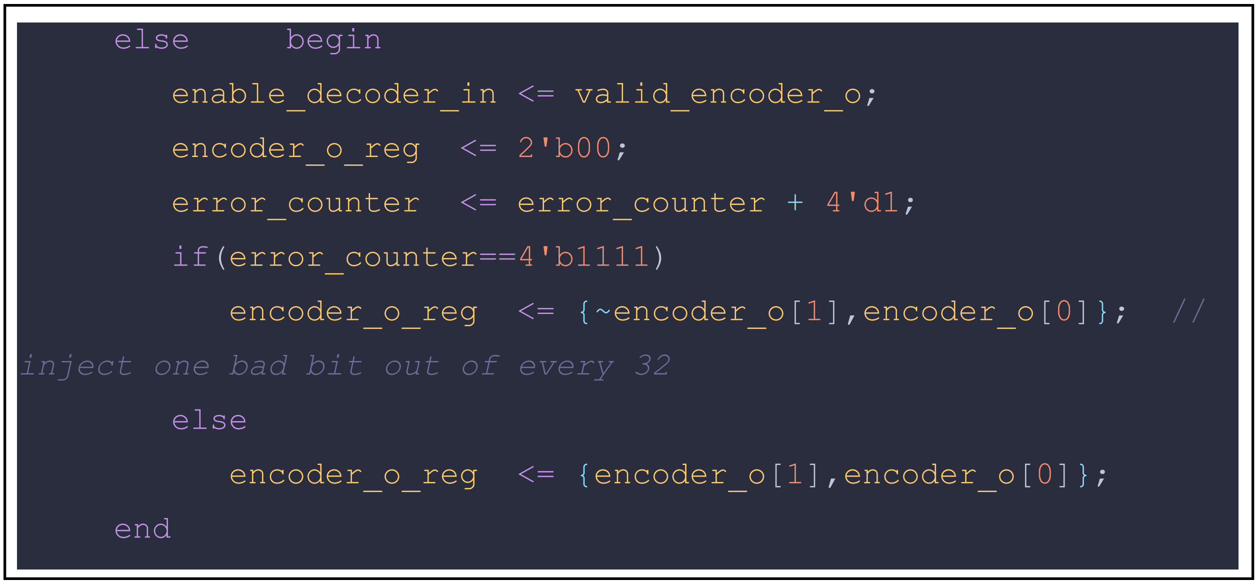
ECE 111: Homework 8 Viterbi Channel Simulation

Goal: The point of this exercise is to show how a bad ENCODER input bit can impact multiple outputs.

Highlighted Sections: Changes made in comparison to original code

Original:

Try five different decoder input error injection scenarios in addition to the 31 good - 1 bad - 31 good - 1 bad ... 3% error injection pattern I demonstrated in class.



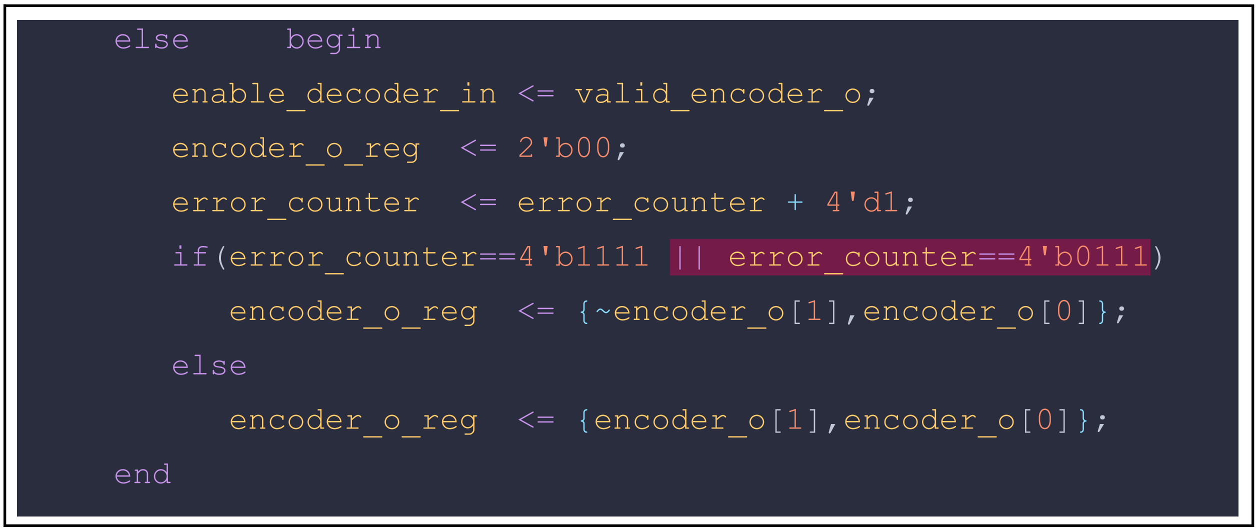
```
     else     begin
        enable_decoder_in <= valid_encoder_o;
        encoder_o_reg  <= 2'b00;
        error_counter  <= error_counter + 4'd1;
        if(error_counter==4'b1111)
           encoder_o_reg  <= {~encoder_o[1],encoder_o[0]};  //
inject one bad bit out of every 32
        else
           encoder_o_reg  <= {encoder_o[1],encoder_o[0]};
     end
```

```
Result: good =         256, bad =           0

Error Rate: 31 good - 1 bad

3% error
```

Christina Mai
Anne Lin

1) Decoder input errors spaced apart, but double their frequency of occurrence

Error_counter counts in cycles of 0-15. When error_coounter hits 7 or 15, an error bit is injected into the decoder input. The Viterbi decoder is still able to work in this scenario.



```
else     begin
   enable_decoder_in <= valid_encoder_o;
   encoder_o_reg  <= 2'b00;
   error_counter  <= error_counter + 4'd1;
   if(error_counter==4'b1111 || error_counter==4'b0111)
      encoder_o_reg  <= {~encoder_o[1],encoder_o[0]};
   else
      encoder_o_reg  <= {encoder_o[1],encoder_o[0]};
end
```

```
Result: good =         256, bad =           0
```

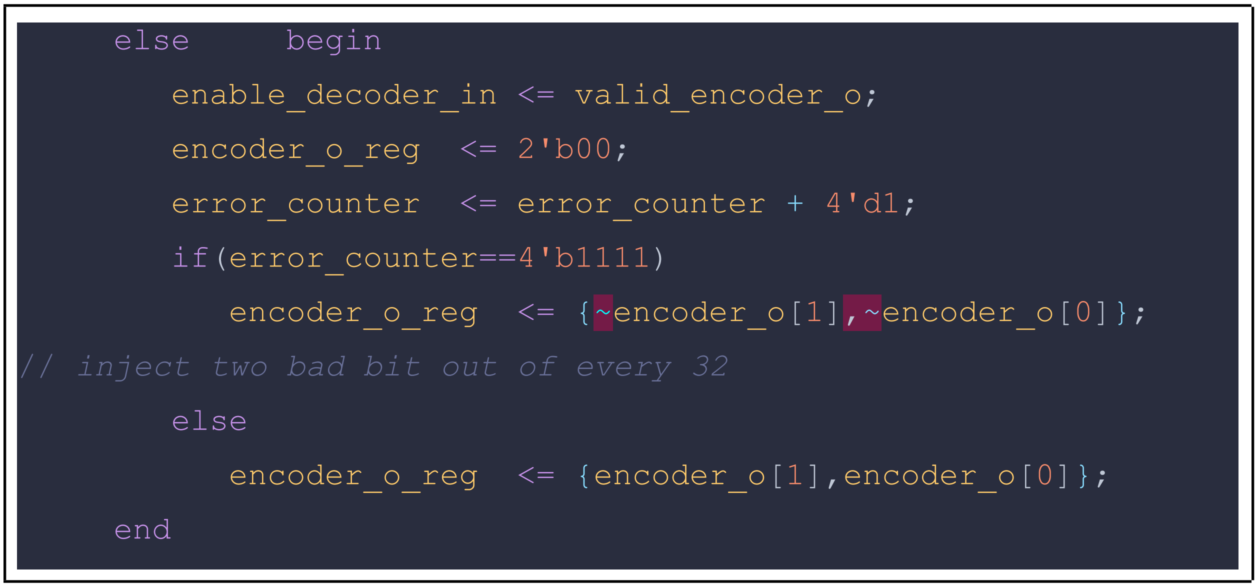
Error Rate: 15 good - 1 bad

6% error

Christina Mai
Anne Lin

2) Two adjacent errors into the decoder at a time, i.e., 30 good - 2 bad - 30 good- 2 bad

The same error rate as scenario 1 but with 2 consecutive error bits in a row. The Viterbi decoder is still able to handle this scenario.



```
     else     begin
        enable_decoder_in <= valid_encoder_o;
        encoder_o_reg  <= 2'b00;
        error_counter  <= error_counter + 4'd1;
        if(error_counter==4'b1111)
           encoder_o_reg  <= {~encoder_o[1],~encoder_o[0]};
// inject two bad bit out of every 32
        else
           encoder_o_reg  <= {encoder_o[1],encoder_o[0]};
     end
```

```
Result: good =         256, bad =           0

Error Rate: 30 good - 2 bad

6% error
```

Christina Mai
Anne Lin

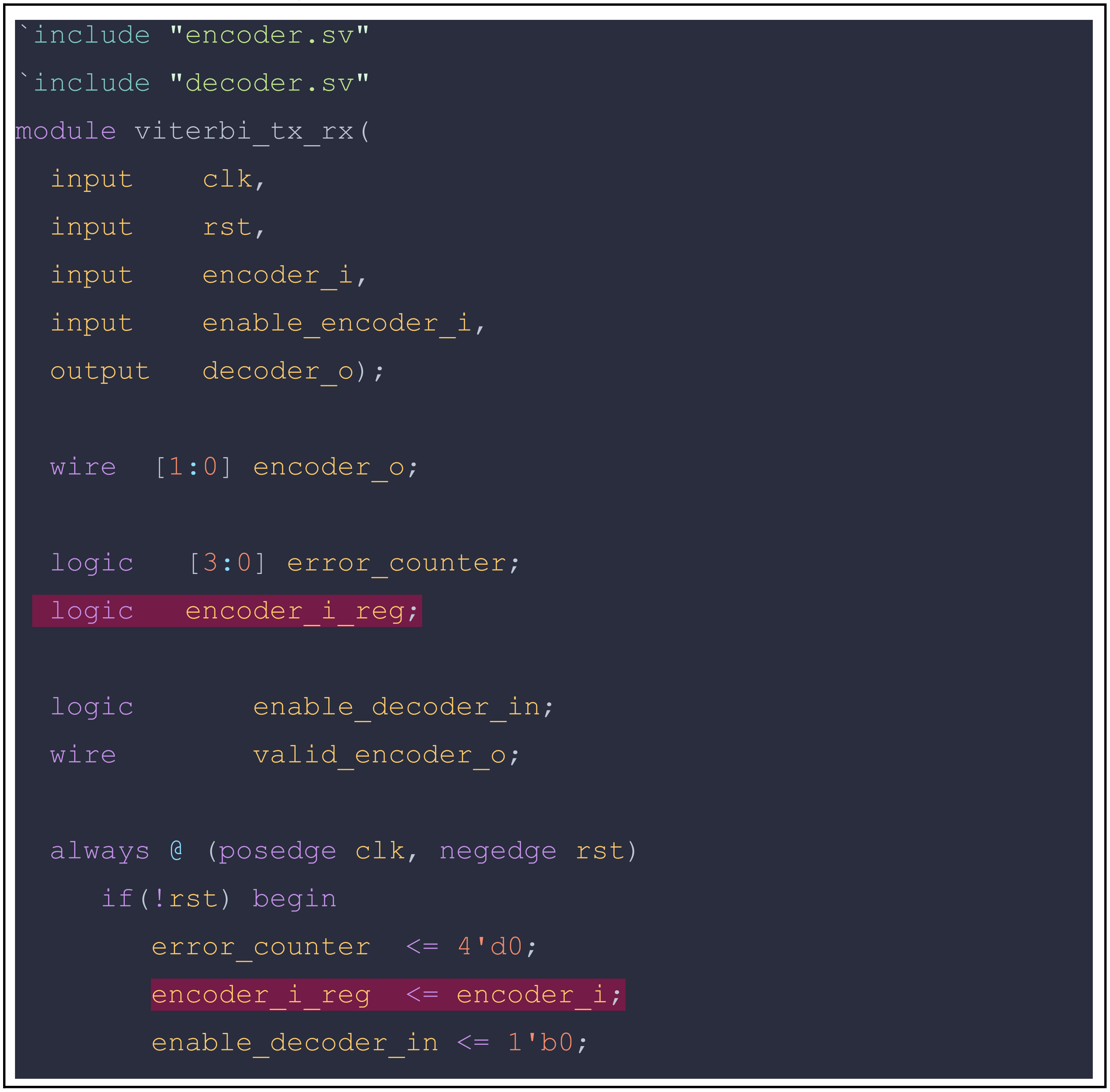
3) Decoder output score if we inject one bad bit into the encoder input

Note: The Viterbi decoder is designed for corruption between perfect transmission and the receiver, so this trial violates that, just for study purposes.

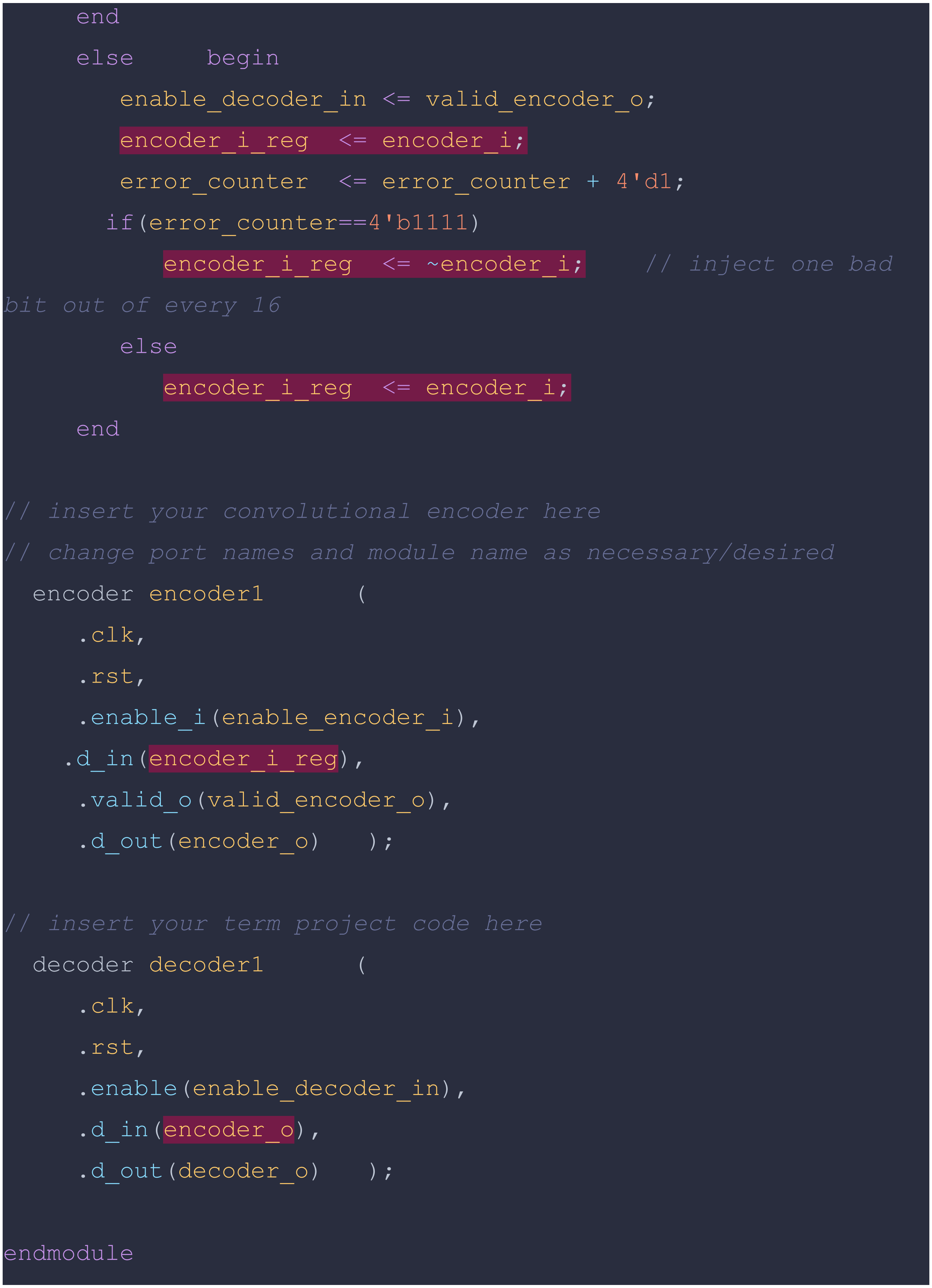
When a bad bit is injected into the encoder input, the decoder output is inverted so this will be a bad bit. The Viterbi decoder cannot handle this type of corruption.

Doubling the encoder's bad bit injection frequency as seen from 1/16 to ⅛ below doubles the error bits that are output from the Viterbi decoder and scales proportionally.

Inject one bad bit into the encoder input 1/16



```
`include "encoder.sv"
`include "decoder.sv"
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   encoder_i_reg;

  logic       enable_decoder_in;
  wire        valid_encoder_o;

  always @ (posedge clk, negedge rst)
     if(!rst) begin
        error_counter  <= 4'd0;
        encoder_i_reg  <= encoder_i;
        enable_decoder_in <= 1'b0;
```



```
     end
     else     begin
        enable_decoder_in <= valid_encoder_o;
        encoder_i_reg  <= encoder_i;
        error_counter  <= error_counter + 4'd1;
       if(error_counter==4'b1111)
           encoder_i_reg  <= ~encoder_i;    // inject one bad
bit out of every 16
        else
           encoder_i_reg  <= encoder_i;
     end

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

endmodule
```

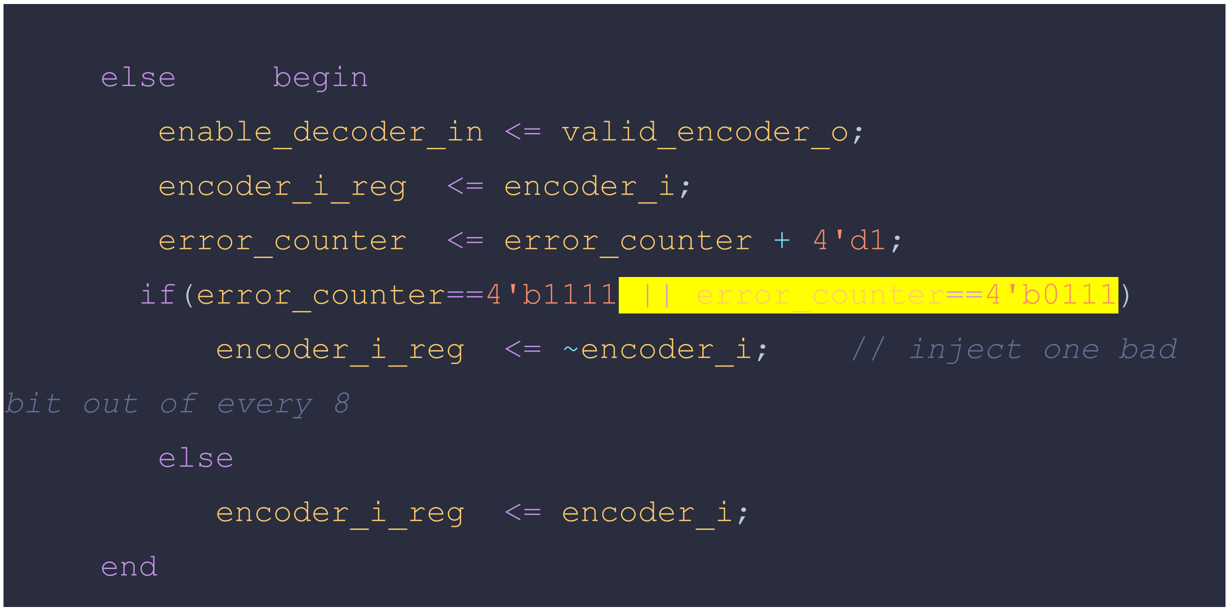
Christina Mai
Anne Lin

```
Result: good =         241, bad =          15
```

Error Rate: 15 good - 1 bad

6% error

Inject one bad bit into the encoder input 1/8



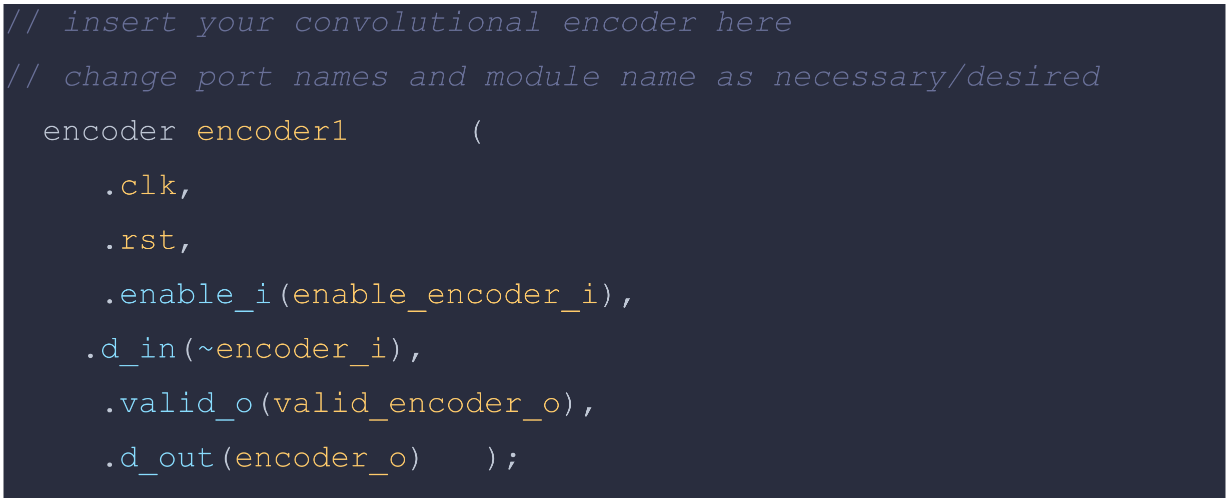
```
     else     begin
        enable_decoder_in <= valid_encoder_o;
        encoder_i_reg  <= encoder_i;
        error_counter  <= error_counter + 4'd1;
       if(error_counter==4'b1111 || error_counter==4'b0111)
           encoder_i_reg  <= ~encoder_i;    // inject one bad
bit out of every 8
        else
           encoder_i_reg  <= encoder_i;
     end
```

```
Result: good =         226, bad =          30
```

Error Rate: 7 good - 1 bad

12% error

Inject bad bit for all encoder inputs



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

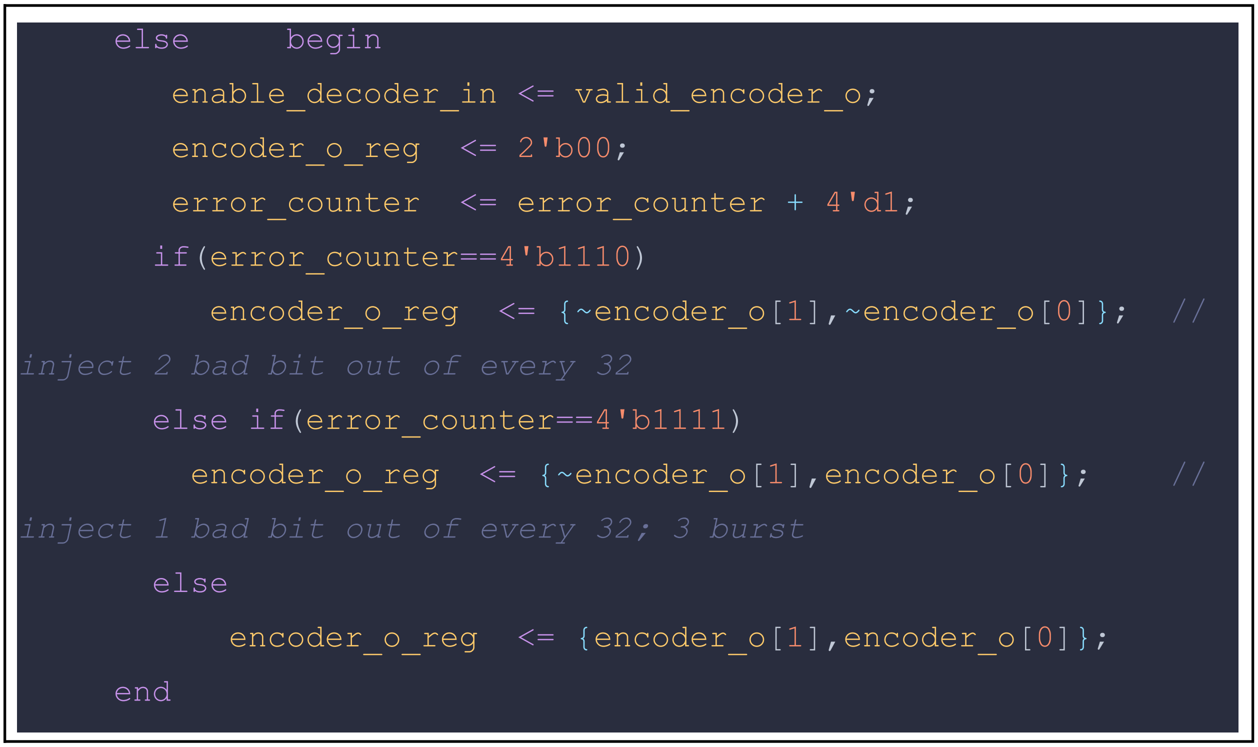
Christina Mai
Anne Lin

```
Result: good =         10, bad =        246
Error Rate: 256 bad /256 outs
100% error
```

When “encoder_i” (encoder input) is inverted, “out” is the inversion of “in” when “rst” is 1. “Out” is 0 when “rst” is 1, which occurs for 10 “outs” and are also the occurrences that are labeled as good. For each bad bit injected into the encoder input, the decoder output score becomes bad for when “rst" is 1.

4) Burst of three consecutive errors to the decoder input, i.e., 29 good - 3 bad - 29 good - 3 bad ... ?

Once the Viterbi decoder starts getting 3 consecutive errors to the decoder input, the decoder breaks and will have bad bits as seen from the progression of scenario 1, 2 , to 4.



```
     else     begin
        enable_decoder_in <= valid_encoder_o;
        encoder_o_reg  <= 2'b00;
        error_counter  <= error_counter + 4'd1;
       if(error_counter==4'b1110)
          encoder_o_reg  <= {~encoder_o[1],~encoder_o[0]};  //
inject 2 bad bit out of every 32
       else if(error_counter==4'b1111)
         encoder_o_reg  <= {~encoder_o[1],encoder_o[0]};    //
inject 1 bad bit out of every 32; 3 burst
       else
           encoder_o_reg  <= {encoder_o[1],encoder_o[0]};
     end
```

```
Result: good =         214, bad =          42

Error Rate: 29 good - 3 bad

9% error

# boo! # = 72, rst = 1, in = 1, out = 0

# yaa! # = 73, rst = 1, in = 1, out = 1

# boo! # = 74, rst = 1, in = 1, out = 0

# boo! # = 75, rst = 1, in = 1, out = 0
```

Produces error in patterns of “boo, yaa, boo, boo”.

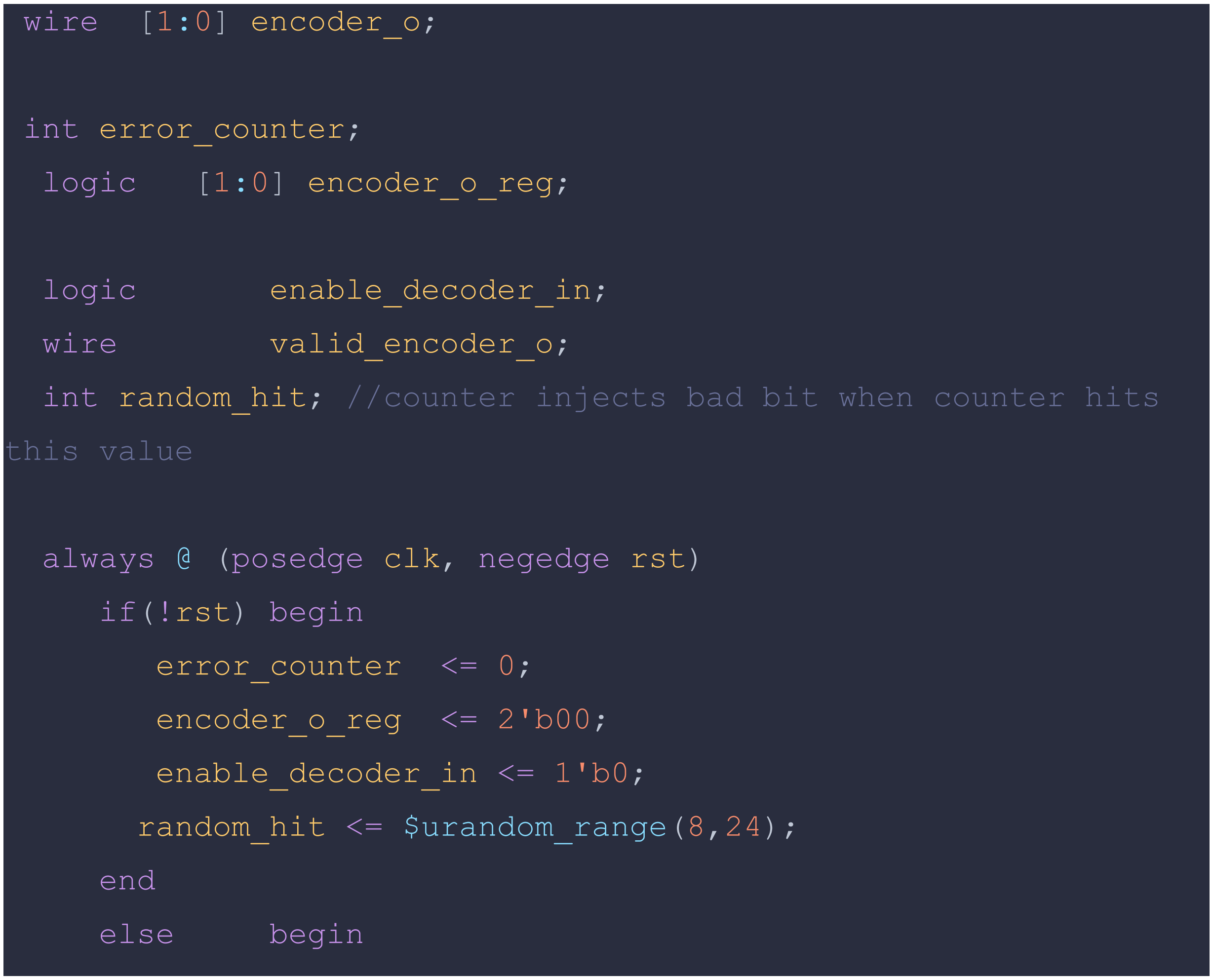
Christina Mai
Anne Lin

5) Randomly spaced errors into the decoder at a 6% average rate.

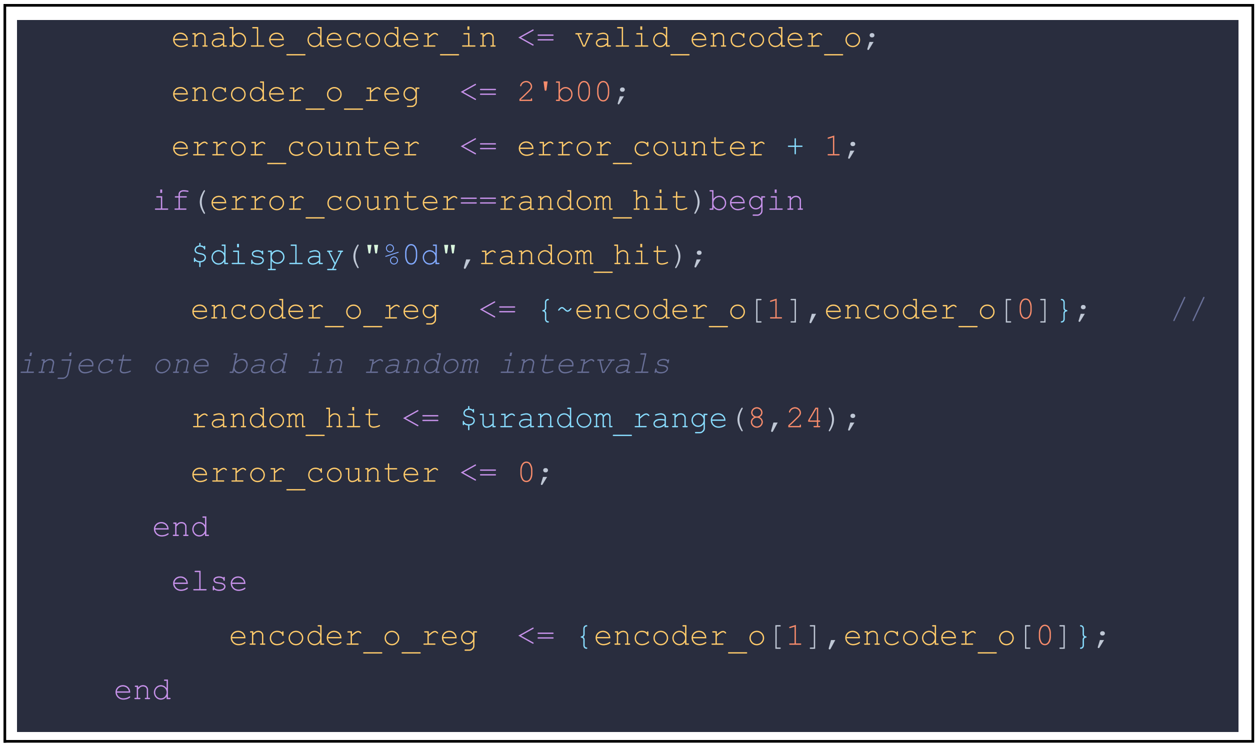
Consecutive Random can produce bursts of four based on my test case will cause corruption; however randomly spaced errors that are still within the limits of the Viterbi decoder, one or two consecutive errors into the decoder input, will not break the decoder.

$urandom_range generates a random number between 8 and 24 to have an average of 16 spaced errors. This number, random_hit, is first generated during reset and then everytime the counter hits random_hit to generate a bad bit to inject into the decoder input, the error_counter resets to 0 and another random number is generated for random_hit.

Non-consecutive Random



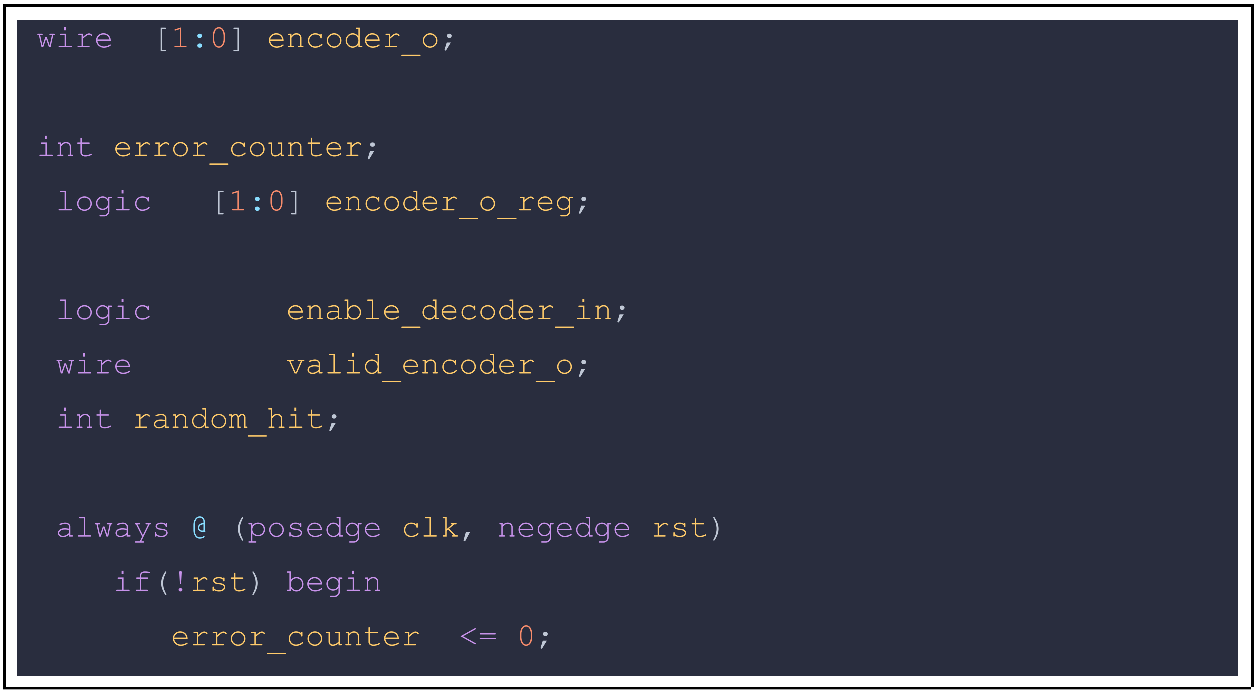
```
 wire  [1:0] encoder_o;

 int error_counter;
  logic   [1:0] encoder_o_reg;

  logic       enable_decoder_in;
  wire        valid_encoder_o;
  int random_hit; //counter injects bad bit when counter hits
this value

  always @ (posedge clk, negedge rst)
     if(!rst) begin
        error_counter  <= 0;
        encoder_o_reg  <= 2'b00;
        enable_decoder_in <= 1'b0;
       random_hit <= $urandom_range(8,24);
     end
     else     begin
```



```
        enable_decoder_in <= valid_encoder_o;
        encoder_o_reg  <= 2'b00;
        error_counter  <= error_counter + 1;
       if(error_counter==random_hit)begin
         $display("%0d",random_hit);
         encoder_o_reg  <= {~encoder_o[1],encoder_o[0]};    //
inject one bad in random intervals
         random_hit <= $urandom_range(8,24);
         error_counter <= 0;
       end
        else
           encoder_o_reg  <= {encoder_o[1],encoder_o[0]};
     end
```

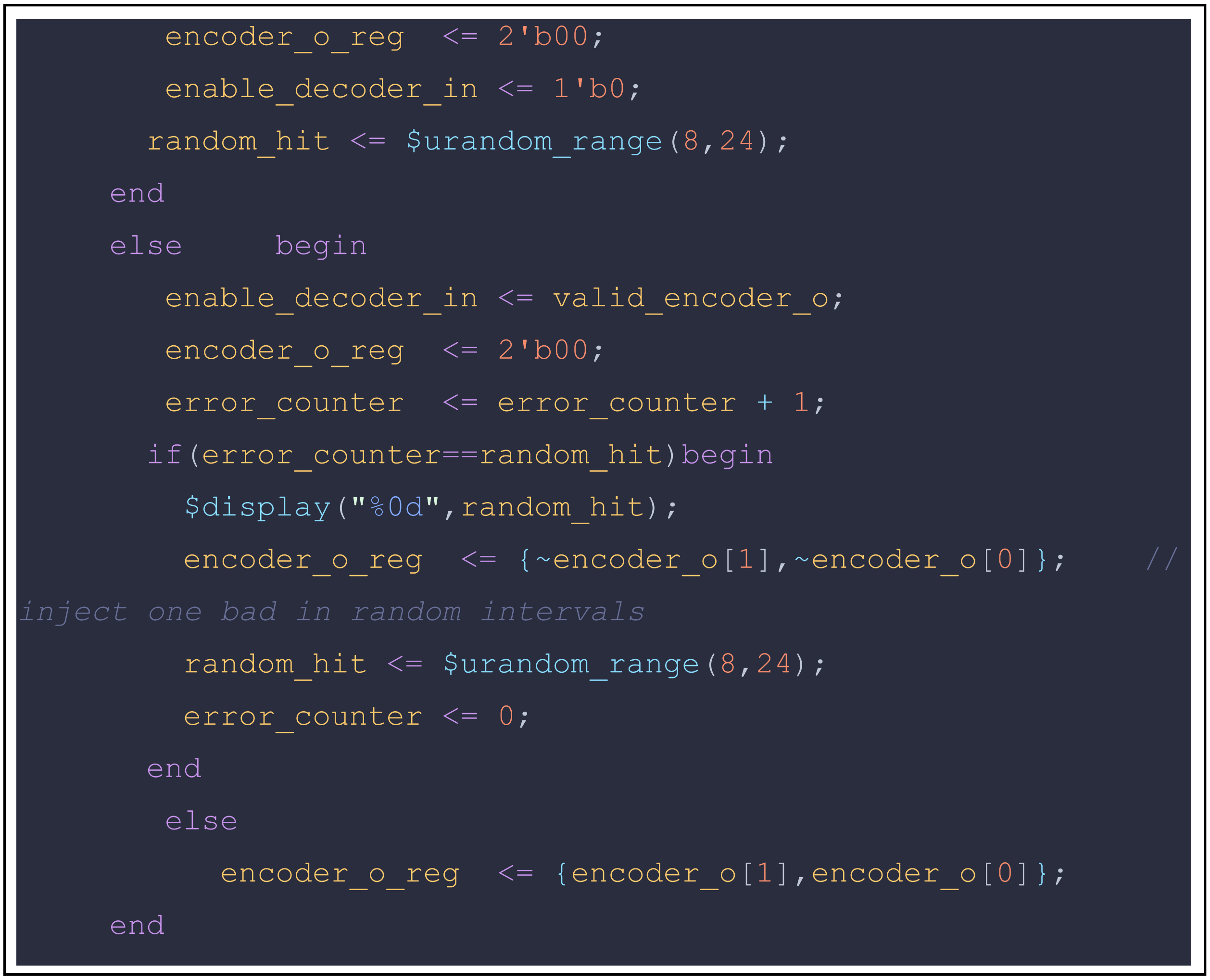
```
Result: good =          256, bad =           0
```

Consecutive Random



```
 wire  [1:0] encoder_o;

 int error_counter;
  logic   [1:0] encoder_o_reg;

  logic       enable_decoder_in;
  wire        valid_encoder_o;
  int random_hit;

  always @ (posedge clk, negedge rst)
     if(!rst) begin
        error_counter  <= 0;
```

Christina Mai
Anne Lin



```
        encoder_o_reg  <= 2'b00;
        enable_decoder_in <= 1'b0;
       random_hit <= $urandom_range(8,24);
     end
     else     begin
        enable_decoder_in <= valid_encoder_o;
        encoder_o_reg  <= 2'b00;
        error_counter  <= error_counter + 1;
       if(error_counter==random_hit)begin
         $display("%0d",random_hit);
         encoder_o_reg  <= {~encoder_o[1],~encoder_o[0]};    // inject one bad in random intervals
         random_hit <= $urandom_range(8,24);
         error_counter <= 0;
       end
        else
           encoder_o_reg  <= {encoder_o[1],encoder_o[0]};
     end
```

```
Result: good =         252, bad =           4
```