Problem 2: (15 points)

In communication links comma symbols are used to correctly align words or mark the beginning of valid data. It is required to design a module that can detect a comma symbol of 3'b101 in an input word starting from the LSB side. If the symbol is found, then the index (bit order) of the MSB of the comma symbol should be given at the output. If the code is not found then an index of **zero**, should appear at the output.

```
module comma_index #(parameter word_size = 16, index_size = 4)
                    (output reg    [index_size -1:0]  index_out,
                     input         [word_size -1:0]   word_in,
                     input                            trigger);
always @ (posedge trigger)
    begin : search
        for (index_out = 2, index_out < word_size, index_out = index_out + 1)
            if (word_in [index_out : index_out-2] == 3'b101)
                disable search;
    end
endmodule
```

(a) Is this code satisfying the behavioral specifications of the design? Are there any bugs in the code? Explain and suggest a way to fix the code.

- **Syntax error: A ';' should separate the initial value setting, the loop condition, and the increment.**
  `for (index_out = 2 ; index_out < word_size ; index_out = index_out+1)`
  * To fix this error, the semicolon replaced the commas (2)

- **Runtime error: The loop may never be disabled. With the index_out being 4 bits, the count would go from 2 to 15, and then it will reset back to 0, so the index_out register will remain less than 16. Some synthesizers would resize the register to avoid this run time error. This is what Quartus synthesizer does. To avoid that, a disabling condition needs to be added based on the functional description.**
  * When the comma_code is not found and index_out == (word_size-1), the search loop is disabled. (2)

- **Functional error: The functional description requires that the index_out equals 4'b0000 when the code is not found. This is not satisfied in the code. This code would set the code at 16 (after resizing the index_out) if the comma is not found.**
  * An if statement was added to set the index_out to zero when the code is not found. (3)

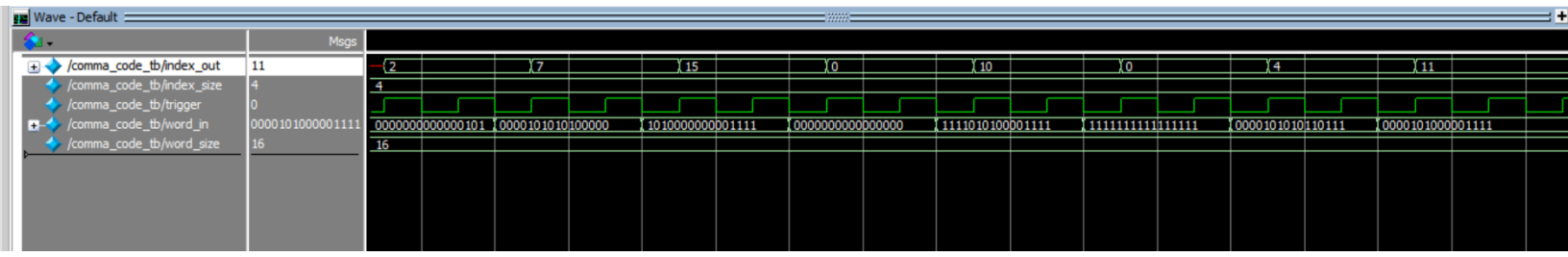
- **Syntax error: Quartus gives an error, indicating that 'index_out is not a constant', pointing to the statement:** `(word_in [index_out : index_out-2] == 3'b101)`. **Subtracting a 2 from index_out to identify the 'moving' 3 bits to be compared to the comma code was causing the error.**
  * A temporary register was used to save the word_in and shift its contents to the right and always compare the last 3 bits to the comma code.

(+2)

```
module Comma_code #(parameter word_size = 16,
                    index_size = 4)(
                output      reg       [index_size-1:0]   index_out,
                input                 [word_size -1:0]   word_in,
                input                                    trigger);
                                                                                    (+4)
                reg [word_size-1 :0]    temp_reg;     // Temporary register to locate the comma code
                                                      // Using shifting, we can test the 3LSB instead
                                                      // of moving the index
                                                      // Avoid the error that index_out is not a constant word

   always @ (posedge trigger)

         begin: search_code

               temp_reg = word_in;          // copying the word into the temporary register

               for (index_out=2; index_out<(word_size); index_out=(index_out+1))
                 begin
                   if (temp_reg[2:0] == 3'b101) disable search_code;     // Always checking the 3 LSBs
                   else begin
                           temp_reg = temp_reg >>1;                                 // Shift the temp_reg to the right
                           if ((temp_reg == 0) | (index_out == (word_size-1)))  // The temp_reg is all 0s,
                                   begin                                        // or the MSB was reached
                                   index_out = 4'b0000;                         // The code was not found
                                   disable search_code;
                                   end
                         end
                   end
          end
endmodule
```

```
    module comma_code_tb ();

    parameter                           word_size  = 16;
    parameter                           index_size = 4 ;
    wire          [index_size-1:0]      index_out;
    reg           [word_size -1:0]      word_in;
    reg                                 trigger;


    Comma_code UUT ( index_out, word_in, trigger);

    initial  begin
             trigger    =  1'b0;
             forever
                 #10    trigger = ~trigger; end

    initial fork
            word_in = 16'b_0000_0000_0000_0101;
       #40  word_in = 16'b_0000_1010_1010_0000;
       #80  word_in = 16'b_1010_0000_0000_1111;
       #120 word_in = 16'b_0000_0000_0000_0000;
       #160 word_in = 16'b_1111_0101_0000_1111;
       #200 word_in = 16'b_1111_1111_1111_1111;
       #240 word_in = 16'b_0000_1010_1011_0111;
       #280 word_in = 16'b_0000_1010_0000_1111;
            join

    initial
    $monitor ($time,, "word_in = %b : index_out = %d",
                       word_in, index_out);

    endmodule
```

```
  0 word_in = 0000000000000101 : index_out =  x
 10 word_in = 0000000000000101 : index_out =  2
 40 word_in = 0000101010100000 : index_out =  2
 50 word_in = 0000101010100000 : index_out =  7
 80 word_in = 1010000000001111 : index_out =  7
 90 word_in = 1010000000001111 : index_out = 15

120 word_in = 0000000000000000 : index_out = 15
130 word_in = 0000000000000000 : index_out =  0
160 word_in = 1111010100001111 : index_out =  0
170 word_in = 1111010100001111 : index_out = 10

200 word_in = 1111111111111111 : index_out = 10
210 word_in = 1111111111111111 : index_out =  0
240 word_in = 0000101010110111 : index_out =  0
250 word_in = 0000101010110111 : index_out =  4
280 word_in = 0000101000001111 : index_out =  4
290 word_in = 0000101000001111 : index_out = 11
```



(b) Redesign the fixed module in part (a) by replacing the **for** loop with a **while** loop. The module should satisfy the functionality and design specifications of the comma detector.

```
module comma_index_while   #(parameter word_size = 16,
                                       index_size = 4)(
                output     reg      [index_size-1:0]   index_out,
                input               [word_size -1:0]   word_in,
                input                                  trigger);

                reg [word_size-1 :0]   temp_reg;    // Temporary register to locate the comma code
                                                    // Using shifting, we can test the 3LSB instead
                                                    // of moving the index
                                                    // Avoid the error that index_out is not a constant word

   always @ (posedge trigger)

          begin: search_code

           (2)  temp_reg = word_in;          // copying the word into the temporary register
                index_out = 1;

                while (index_out<(word_size))          (2)
                  begin
            (2)     index_out = index_out+1;                          // Increment index_out to match the code MSB
                    if (temp_reg[2:0] == 3'b101) disable search_code;     // Always checking the 3 LSBs
                    else begin                                                (2)
                            temp_reg = temp_reg >>1;                          // Shift the temp_reg to the right
                            if ((temp_reg == 0) | (index_out == (word_size-1)))  // The temp_reg is all Os,
                                    begin                                        // or the MSB was reached
                                    index_out = 4'b0000;                         // The code was not found
                                    disable search_code;
                                    end
                          end
                  end
          end
endmodule
```

```
module comma_code_tb ();

parameter                          word_size  = 16;
parameter                          index_size = 4 ;
wire          [index_size-1:0]     index_out;
reg           [word_size -1:0]     word_in;
reg                                trigger;


//Comma_code UUT ( index_out, word_in, trigger);
comma_index_while UUT ( index_out, word_in, trigger);

initial  begin
         trigger   =  1'b0;
         forever
             #10    trigger = ~trigger; end

initial fork
        word_in = 16'b_0000_0000_0000_0101;
   #40  word_in = 16'b_0000_1010_1010_0000;
   #80  word_in = 16'b_1010_0000_0000_1111;
   #120 word_in = 16'b_0000_0000_0000_0000;
   #160 word_in = 16'b_1111_0101_0000_1111;
   #200 word_in = 16'b_1111_1111_1111_1111;
   #240 word_in = 16'b_0000_1010_1011_0111;
   #280 word_in = 16'b_0000_1010_0000_1111;
        join

initial
$monitor ($time,, "word_in = %b : index_out = %d",
                   word_in, index_out);

endmodule
```

```
  0 word_in = 0000000000000101 : index_out =  x
 10 word_in = 0000000000000101 : index_out =  2
 40 word_in = 0000101010100000 : index_out =  2
 50 word_in = 0000101010100000 : index_out =  7
 80 word_in = 1010000000001111 : index_out =  7
 90 word_in = 1010000000001111 : index_out = 15

120 word_in = 0000000000000000 : index_out = 15
130 word_in = 0000000000000000 : index_out =  0
160 word_in = 1111010100001111 : index_out =  0
170 word_in = 1111010100001111 : index_out = 10

200 word_in = 1111111111111111 : index_out = 10
210 word_in = 1111111111111111 : index_out =  0
240 word_in = 0000101010110111 : index_out =  0
250 word_in = 0000101010110111 : index_out =  4
280 word_in = 0000101000001111 : index_out =  4
290 word_in = 0000101000001111 : index_out = 11
```