George Suarez

CSE 310

Homework 3

1. Write a Verilog program that simulates the outputs of gray-code-to-binary-code converter. First write a module named **GBC** which takes *w, x, y, z* as inputs and *a, b, c, d* as outputs. Then write a test bench module named **bcd_tb** to test the GBC module. Compile and run your program and show the outputs for all 16 combinations of *w, x, y, z*.

   gbc.v:

```
module GBC ( output a, b, c, d, input w, x, y, z );
   assign a = w;
   assign b = w ^ x;
   assign c = b ^ y;
   assign d = c ^ z;
endmodule
```

   bcd_tb.v:

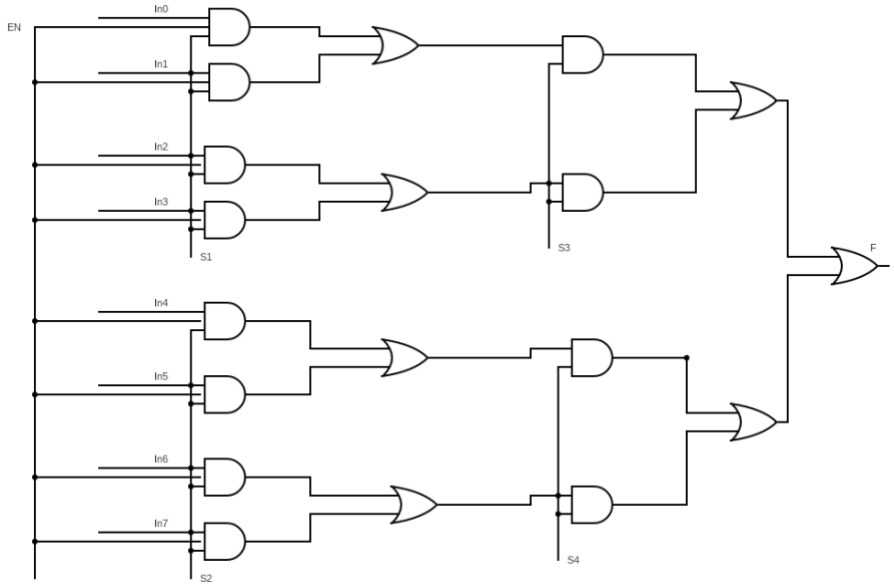
```
`timescale 1ns / 1ps

module bcd_tb();

   reg w, x, y, z;
   output a, b, c, d;

   initial
      begin

         $display("Time\t w    x  y  z |  a b c d");
         $monitor("%g\t %b    %b  %b  %b |  %b %b %b %b",
         $time, w, x, y, z, a, b, c, d);
   end

   initial
      begin
```

```
        w = 1'b0;
        x = 1'b0;
        y = 1'b0;
        z = 1'b0;

        #150 $finish;
    end


    always #10 z = ~z;
    always #20 y = ~y;
    always #40 x = ~x;
    always #80 w = ~w;


    GBC run(a, b, c, d, w, x, y, z);
endmodule
```

Output:

```
Time w   x  y  z |  a b c d
0        0   0  0  0 |  0 0 0 0
10       0   0  0  1 |  0 0 0 1
20       0   0  1  0 |  0 0 1 1
30       0   0  1  1 |  0 0 1 0
40       0   1  0  0 |  0 1 1 1
50       0   1  0  1 |  0 1 1 0
60       0   1  1  0 |  0 1 0 0
70       0   1  1  1 |  0 1 0 1
80       1   0  0  0 |  1 1 1 1
90       1   0  0  1 |  1 1 1 0
100      1   0  1  0 |  1 1 0 0
110      1   0  1  1 |  1 1 0 1
120      1   1  0  0 |  1 0 0 0
130      1   1  0  1 |  1 0 0 1
140      1   1  1  0 |  1 0 1 1
```

2. First write a Verilog module that implements a 2x1 MUX (multiplexer) and a module that implements 4x1 MUX using 2x1 MUX. Add an extra input port *enable* to the 4x1 MUX and 2x1 MUX.
   Then write another module that implements 8x1 MUX using two 4x1 MUX with *enable* ports and an *or* gate. Draw a diagram to show how this is done. Also write test bench programs to test all the multiplexer modules.



mux.v:

```
module mux2x1 ( output reg F, input en, input in1, in0 );
   always @ ( en or in1 or in0 )
   begin
      if ( en == 1'b0 )
         F = in0;
      else
         F = in1;
   end
endmodule

module mux4x1 ( output reg F, input en, input in3, in2, in1, in0 );
```

```
  wire a, b, c;
  mux2x1 m1 ( a, en, in1, in0 );
  mux2x1 m2 ( b, en, in3, in2 );
  mux2x1 m3 ( c, en, b,  a );
  always @ ( a or b or c )
    F = c;
endmodule

module mux8x1 ( F, in, en );
    input[7:0] in;
    input[1:0] en;
    output F;
    wire mux1, mux2;
    mux4x1 x1( mux1, en[0], in[0], in[1], in[2], in[3] );
    mux4x1 x2( mux2, en[1], in[4], in[5], in[6], in[7] );
    or ( F, mux1, mux2 );
endmodule
```

mux2x1_tb.v:

```
module mux2x1_tb();

    reg en, in0, in1;

    wire F;

    mux2x1 uut ( F, en, in1, in0 );

    always
        #5 in0 = ~in0;
    always
        #10 in1 = ~in1;
    always
        #20 en = ~en;

    initial begin
```

```
    in0 = 0;
    in1 = 0;
    en = 0;

        $display("Time \t   en   in1    in0     F");
        $monitor("%g   \t   %d     %d      %d      %d ", $time, en, in1, in0, F);

        #30 $finish;
 end
endmodule
```

Output:

Time	en	in1	in0	F
0	0	0	0	0
5	0	0	1	1
10	0	1	0	0
15	0	1	1	1
20	1	0	0	0
25	1	0	1	0
30	1	1	0	1

mux4x1_tb.v:

```
    module mux4x1_tb();

        reg in0, in1, in2, in3;
        reg en;

        wire F;

        mux4x1 uut ( F, en, in3, in2, in1, in0 );

        always
            #5 en = ~en;
        always
            #10 in3 = ~in3;
```

```
    always
      #20 in2 = ~in2;
    always
      #40 in1 = ~in1;
    always
      #80 in0 = ~in0;

    initial begin
      in0 = 0;
      in1 = 0;
      in2 = 0;
      in3 = 0;
      en = 0;

      $display("Time \t en in0 in1 in2 in3  F");
      $monitor("%g   \t %d   %d   %d   %d   %d   %d",
      $time, en, in0, in1, in2, in3, F);

      #100 $finish;
    end
endmodule
```

Output:

Time	en	in0	in1	in2	in3	F
0	0	0	0	0	0	0
5	1	0	0	0	0	0
10	0	0	0	0	1	0
15	1	0	0	0	1	1
20	0	0	0	1	0	0
25	1	0	0	1	0	0
30	0	0	0	1	1	0

```
35      1   0   0   1   1   1
40      0   0   1   0   0   0
45      1   0   1   0   0   0
50      0   0   1   0   1   0
55      1   0   1   0   1   1
60      0   0   1   1   0   0
65      1   0   1   1   0   0
70      0   0   1   1   1   0
75      1   0   1   1   1   1
80      0   1   0   0   0   1
85      1   1   0   0   0   0
90      0   1   0   0   1   1
95      1   1   0   0   1   1
100   0   1   0   1   0   1
```

mux8x1_tb.v:

```
`timescale 1ns / 1ps

module mux8x1_tb();

   reg [0:7] in;
   reg [1:0] en;

   wire F;

   mux8x1 uut(F, in, en);

   initial begin

      $display ("Time\ten\t   in  \t\tF");
      $monitor ("%g\t%b\t%b\t%b", $time, en, in, F);
      #50 $finish;

   end
```

```
  initial begin
    en=2'b00; in=8'b00000000;
    #10 en=2'b10; in=8'b10000000;
    #10 en=2'b01; in=8'b11111111;
    #10 en=2'b01; in=8'b00000100;
    #10 en=2'b10; in=8'b10100000;
    #10 en=2'b01; in=8'b11100100;
    #10 $finish;
  end
endmodule
```

Output:

Time	en	in	F
0	00	00000000	0
10	10	10000000	0
20	01	11111111	1
30	01	00000100	0
40	10	10100000	0
50	01	11100100	1

3.

a. The following syntax is correct. True or false? Why?

```
wire a, b;
always @ ( b )
  a = b;
```

False because it is missing a `begin` keyword after the `always @` statement.

b. The following syntax is correct. True or false? Why?

```
reg a, b;
```

```
assign a = b;
```

False, a `reg` element cannot be used on the left side of an `assign` statement.

c. Wire elements can only be used to model combinational circuits. True or false? Why?

True because wires are used for connecting different elements as they are treated as physical wires, and they cannot store values.

d. What are the values of *a*, *b*, *c* after execution of the following code segment? Why?

```
reg [3:0] a;
reg [3:0] b;
reg [3:0] c;

initial begin
a = 1;
b = 2;
c = 0;
a = b;
c = a + 1;
end
```

a = 2, b = 2, c = 3; reg *a* gets changed from 1 to 2 from reg *b* and reg *c* is 3 since it takes the result of adding the value in reg *a* with the value 1.

e. What are the values of *a*, *b*, *c* after execution of the following code segment? Why?

```
reg [3:0] a;
reg [3:0] b;
reg [3:0] c;

initial begin
a = 1;
b = 2;
c = 0;
a <= b;
c <= a + 1;
end
```

a = 2, b = 2, c = 2; The registers are assigned by using the non-blocking assignment which means that the registers are assigned in parallel, and any evaluations that are done to them are deferred until it reaches the `end` statement.

4. AB + CD + AC' + DE' = A(B + C') + D(C + E') =
   (A+D)(B+C'+D)(A+C+E')

5. 0110, 1, 0101 (A, C, E)
6. 1100, 1 (A, E)
7. If the output carry = 1, then add 0011 (C)
8. WX'YZ' + W'XYZ' + WX'Y'Z + W'XY'Z
   = WX'(YZ' + Y'Z) + W'X(YZ' + Y'Z)
   = WX' + W'X(Y®Z)
   = (W ® X)(Y®Z) (D)