Assignment 3

Probelm 1:

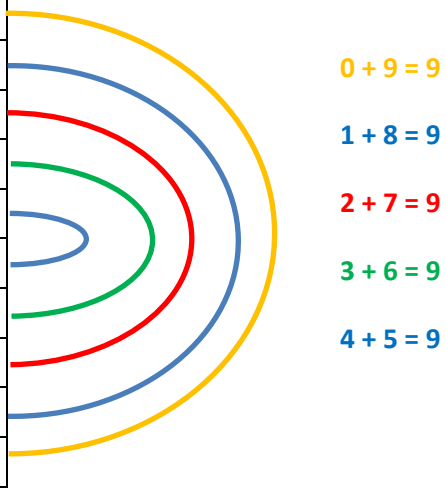
Excess-3 codes are unweighted and can be obtained by adding 3 to each decimal digit. Excess-3 code is a self complementary code. The 1's complement of a binary number can be obtained by replacing 0's with 1's and 1's with 0's. For self-complementary codes, the sum of a binary number and its complement is always equal to decimal 9. This means that the 1's complement of an excess-3 code is the excess-3 code for the 9's complement of the corresponding decimal number.

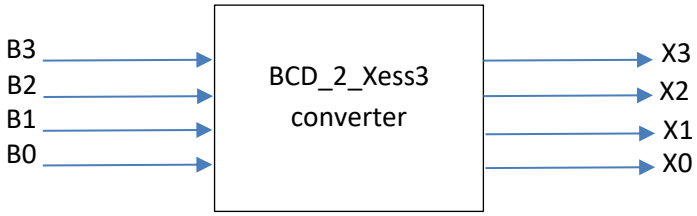
For example, the excess-3 code for decimal number 5 is 1000 (5+3 = 8) and 1's complement of 1000 is 0111, which is excess-3 code for decimal number 4 (4+3 = 7), and it is the 9's complement of number 5.

The primary advantage of excess-3 coding is that a decimal number can be nines' complemented (for subtraction) just by inverting all bits. Also, when the sum of two excess-3 digits is greater than 9, the carry bit of a 4-bit adder will be set high. This works because, after adding two digits, an "excess" value of 6 results in the sum. Because a 4-bit integer can only hold values 0 to 15, an excess of 6 means that any sum over 9 will overflow (produce a carry out).

Another advantage is that the codes 0000 and 1111 are not used for any digit. A fault in a memory or basic transmission line may result in these codes, as they stay at the same logic level without changing.

Decimal value	BCD – 8421 weighted code	Excess-3 code
0	0000	0011
1	0001	0100
2	0010	0101
3	0011	0110
4	0100	0111
5	0101	1000
6	0110	1001
7	0111	1010
8	1000	1011
9	1001	1100





(a) Use **Gate-level Verilog** code to design the combinational logic for the ***BCD_2_Xess3*** converter module. Show your work including the Karnaugh maps for all output bits. Assume that the inputs will only have digits from 0 to 9. Values 10 to 15 can be considered "don't care" cases.

X0 = ~B0

	B3 B2 00	B3 B2 01	B3 B2 11	B3 B2 10
B1 B0 00	1	1	X	1
B1 B0 01	0	0	X	0
B1 B0 11	0	0	X	X
B1 B0 10	1	1	X	X

X1 = (~B1~B0) | B1B0

	B3 B2 00	B3 B2 01	B3 B2 11	B3 B2 10
B1 B0 00	1	1	X	1
B1 B0 01	0	0	X	0
B1 B0 11	1	1	X	X
B1 B0 10	0	0	X	X

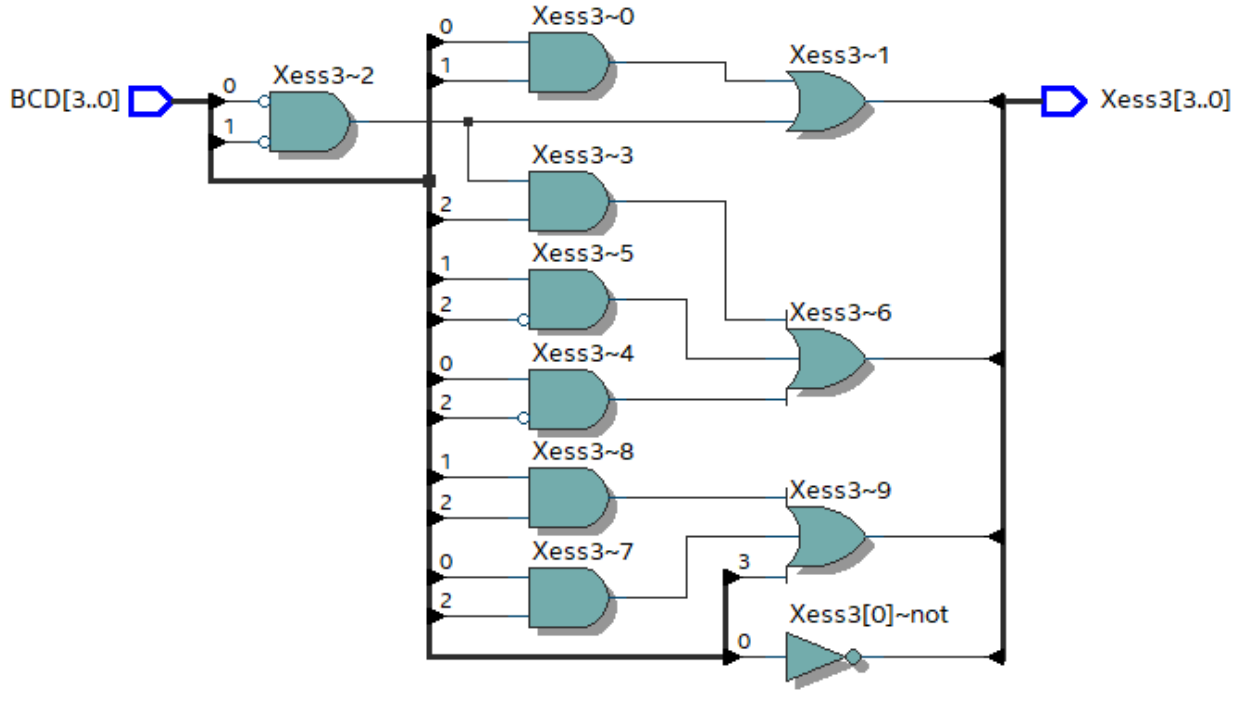
X2 = (~B1~B0B2) | (B0~B2) | (B1~B2)

	B3 B2 00	B3 B2 01	B3 B2 11	B3 B2 10
B1 B0 00	0	1	X	0
B1 B0 01	1	0	X	1
B1 B0 11	1	0	X	X
B1 B0 10	1	0	X	X

X3 = B3 | B0B2 | B1B2

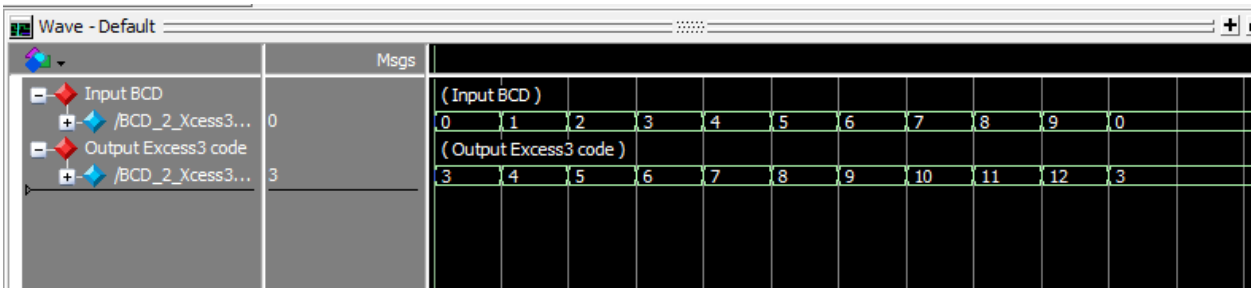
	B3 B2 00	B3 B2 01	B3 B2 11	B3 B2 10
B1 B0 00	0	0	X	1
B1 B0 01	0	1	X	1
B1 B0 11	0	1	X	X
B1 B0 10	0	1	X	X

```
/*=============================================================================================================

   BCD to Excess3 code converter:
   ------------------------------

   Excess-3 codes are unweighted and can be obtained by adding 3 to each decimal digit.
   Excess-3 code is a self complementary code. The 1's complement of a binary number can be obtained
   by replacing 0's with 1's and 1's with 0's. For self-complementary codes, the sum of a binary
   number and its complement is always equal to decimal 9. This means that the 1's complement of
   an excess-3 code is the excess-3 code for the 9's complement of the corresponding decimal number.

   For example, the excess-3 code for decimal number 5 is 1000 (5+3 = 8) and 1's complement of
   1000 is 0111, which is excess-3 code for decimal number 4 (4+3 = 7), and it is the 9's complement
   of number 5.
   The primary advantage of excess-3 coding is that a decimal number can be nines' complemented
   (for subtraction) just by inverting all bits. Also, when the sum of two excess-3 digits is greater
   than 9, the carry bit of a 4-bit adder will be set high. This works because, after adding two
   digits, an "excess" value of 6 results in the sum. Because a 4-bit integer can only hold values
   0 to 15, an excess of 6 means that any sum over 9 will overflow (produce a carry out).

   Another advantage is that the codes 0000 and 1111 are not used for any digit. A fault in a memory
   or basic transmission line may result in these codes, as they stay at the same logic level without
   changing.

=============================================================================================================*/


   module BCD_2_Xcess3 (BCD, Xess3);

   input    [3:0]   BCD;
   output   [3:0]   Xess3;

   assign Xess3[0] =    ~BCD[0]                                                                                ;
   assign Xess3[1] =  ( ~BCD[1] &  ~BCD[0] )          | ( BCD[1] &  BCD[0] )                                  ;
   assign Xess3[2] =  ( ~BCD[1] &  ~BCD[0] & BCD[2] ) | ( BCD[0] & ~BCD[2] ) | (BCD[1] & ~BCD[2])             ;
   assign Xess3[3] =    BCD[3] | ( BCD[0] & BCD[2] ) | ( BCD[1] &  BCD[2] )                                   ;

   endmodule
```



(b) Create a test-bench ***BCD_2_Xess3_tb*** to test your design. The testbench should display all the possible input binary combinations along with the corresponding Excess-3 code.

```
module BCD_2_Xcess3_tb ();

reg    [3:0]   BCD;
wire   [3:0]   Xess3;

BCD_2_Xcess3 UUT (BCD, Xess3);

initial
begin
      BCD = 4'b0000;
#10   BCD = 4'b0001;
#10   BCD = 4'b0010;
#10   BCD = 4'b0011;
#10   BCD = 4'b0100;
#10   BCD = 4'b0101;
#10   BCD = 4'b0110;
#10   BCD = 4'b0111;
#10   BCD = 4'b1000;
#10   BCD = 4'b1001;
#10   BCD = 4'b0000;
end

initial
$monitor ($time,, "BCD = %b = %d    :    Excess3 = %b = %d" , BCD, BCD, Xess3, Xess3);

endmodule
```




```
Transcript
add wave -position end  sim:/BCD_2_Xcess3_tb/BCD
add wave -position end  sim:/BCD_2_Xcess3_tb/Xess3
VSIM 6> run
#                    0 BCD = 0000 =  0   :   Excess3 = 0011 =  3
#                   10 BCD = 0001 =  1   :   Excess3 = 0100 =  4
#                   20 BCD = 0010 =  2   :   Excess3 = 0101 =  5
#                   30 BCD = 0011 =  3   :   Excess3 = 0110 =  6
#                   40 BCD = 0100 =  4   :   Excess3 = 0111 =  7
#                   50 BCD = 0101 =  5   :   Excess3 = 1000 =  8
#                   60 BCD = 0110 =  6   :   Excess3 = 1001 =  9
#                   70 BCD = 0111 =  7   :   Excess3 = 1010 = 10
#                   80 BCD = 1000 =  8   :   Excess3 = 1011 = 11
#                   90 BCD = 1001 =  9   :   Excess3 = 1100 = 12
VSIM 7> run
#                  100 BCD = 0000 =  0   :   Excess3 = 0011 =  3
```

From the simulation results, we see that the combinational logic has correctly added a three to the input values to convert them to the exccess3 code equivalent values.