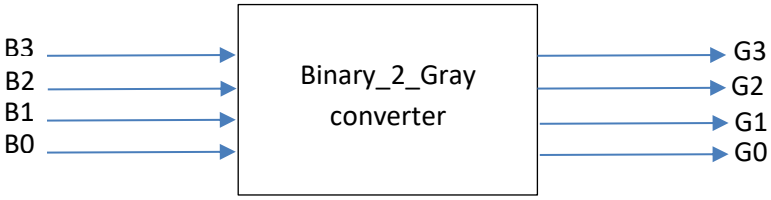
Design Example: Binary to Gray code conversion

In modern digital communications, **Gray codes** play an important role in error correction. For example, in a digital modulation scheme such as QAM where data is typically transmitted in symbols of 4 bits or more, the signal's constellation diagram is arranged so that the bit patterns conveyed by adjacent constellation points differ by only one bit. By combining this with forward error correction capable of correcting single-bit errors, it is possible for a receiver to correct any transmission errors that cause a constellation point to deviate into the area of an adjacent point. This makes the transmission system less susceptible to noise.

Decimal	Binary	Gray	Gray Decimal
0	0000	0000	0
1	0001	0001	1
2	0010	0011	3
3	0011	0010	2
4	0100	0110	6
5	0101	0111	7
6	0110	0101	5
7	0111	0100	4
8	1000	1100	12
9	1001	1101	13
10	1010	1111	15
11	1011	1110	14
12	1100	1010	10
13	1101	1011	11
14	1110	1001	9
15	1111	1000	8



(a) Use **Gate-level Verilog** code to design the combinational logic for the ***Binary_2_Gray*** converter module. Show your work including the Karnaugh maps for all output bits.

	B3B2 00	B3B2 01	B3B2 11	B3B2 10
B1Bo 00	0	0	1	1
B1Bo 01	0	0	1	1
B1Bo 11	0	0	1	1
B1Bo 10	0	0	1	1

	B3B2 00	B3B2 01	B3B2 11	B3B2 10
B1Bo 00	0	1	0	1
B1Bo 01	0	1	0	1
B1Bo 11	0	1	0	1
B1Bo 10	0	1	0	1

G3 = B3

G2 = (~B3)&B2 + B3&(~B2)

G2 = B2 xor B3

	B3B2 00	B3B2 01	B3B2 11	B3B2 10
B1Bo 00	0	1	1	0
B1Bo 01	0	1	1	0
B1Bo 11	1	0	0	1
B1Bo 10	1	0	0	1

	B3B2 00	B3B2 01	B3B2 11	B3B2 10
B1Bo 00	0	0	0	0
B1Bo 01	1	1	1	1
B1Bo 11	0	0	0	0
B1Bo 10	1	1	1	1

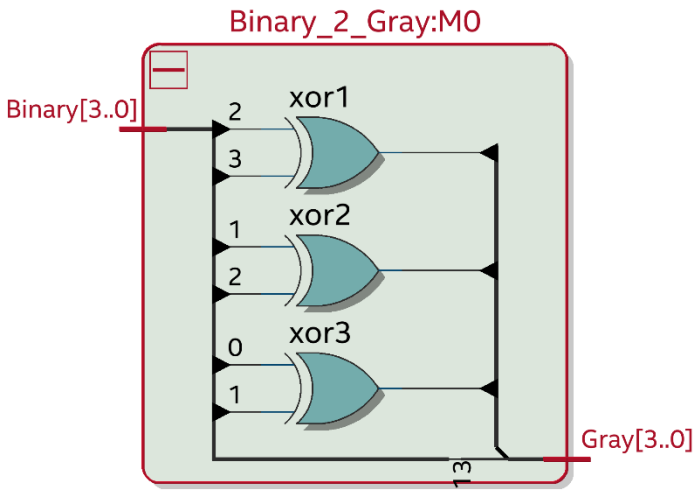
G1 = B2&(~B1) +(~B2)&B1

G1 = B1 xor B2

G0 = (~B1)&B0 + B1&(~B0)

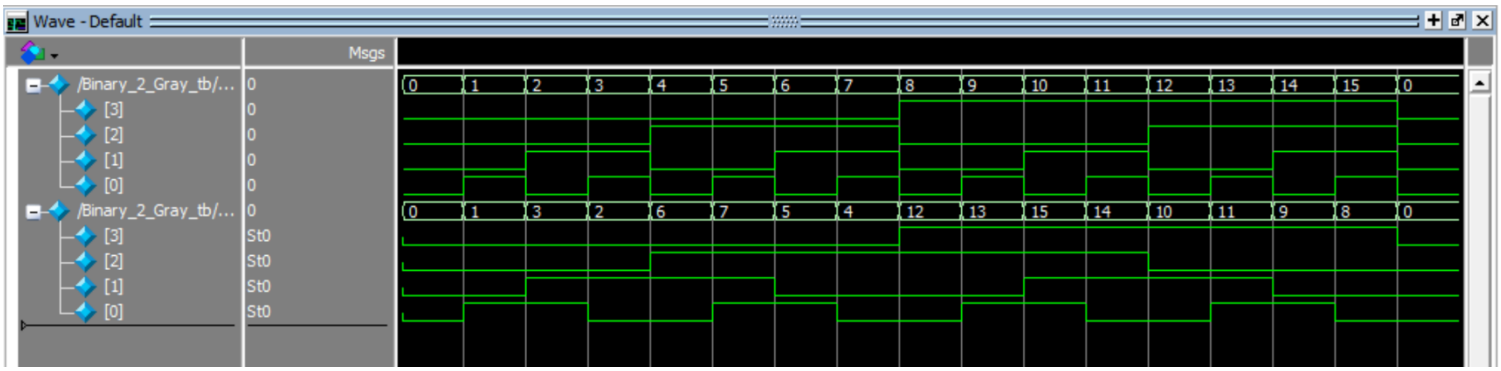
G0 = B0 xor B1

```
module Binary_2_Gray (input  [3:0] Binary,
                      output [3:0] Gray     );

xor xor1   (Gray[2], Binary[3], Binary[2]);
xor xor2   (Gray[1], Binary[2], Binary[1]);
xor xor3   (Gray[0], Binary[1], Binary[0]);

assign   Gray[3] = Binary [3];

endmodule
```



(b) Create a test-bench ***Binary_2_Gray_tb*** to test your design. The testbench should display all the possible input binary combinations along with the corresponding Gray code.

```
module Binary_2_Gray_tb ();

wire [3:0]   Gray;
reg  [3:0]   Binary;

Binary_2_Gray    UUT   (Binary, Gray);

initial
begin
        Binary = 4'b0000;
#10     Binary = 4'b0001;
#10     Binary = 4'b0010;
#10     Binary = 4'b0011;
#10     Binary = 4'b0100;
#10     Binary = 4'b0101;
#10     Binary = 4'b0110;
#10     Binary = 4'b0111;
#10     Binary = 4'b1000;
#10     Binary = 4'b1001;
#10     Binary = 4'b1010;
#10     Binary = 4'b1011;
#10     Binary = 4'b1100;
#10     Binary = 4'b1101;
#10     Binary = 4'b1110;
#10     Binary = 4'b1111;
#10     Binary = 4'b0000;
end

initial
begin
$monitor($time,, "  Binary = %b = %d :   Gray = %b = %d ",
                    Binary, Binary,     Gray, Gray );
end


endmodule
```



```
VSIM 9> run
#                    0   Binary = 0000 =  0 :  Gray = 0000 =  0
#                   10   Binary = 0001 =  1 :  Gray = 0001 =  1
#                   20   Binary = 0010 =  2 :  Gray = 0011 =  3
#                   30   Binary = 0011 =  3 :  Gray = 0010 =  2
#                   40   Binary = 0100 =  4 :  Gray = 0110 =  6
#                   50   Binary = 0101 =  5 :  Gray = 0111 =  7
#                   60   Binary = 0110 =  6 :  Gray = 0101 =  5
#                   70   Binary = 0111 =  7 :  Gray = 0100 =  4
#                   80   Binary = 1000 =  8 :  Gray = 1100 = 12
#                   90   Binary = 1001 =  9 :  Gray = 1101 = 13
VSIM 10> run
#                  100   Binary = 1010 = 10 :  Gray = 1111 = 15
#                  110   Binary = 1011 = 11 :  Gray = 1110 = 14
#                  120   Binary = 1100 = 12 :  Gray = 1010 = 10
#                  130   Binary = 1101 = 13 :  Gray = 1011 = 11
#                  140   Binary = 1110 = 14 :  Gray = 1001 =  9
#                  150   Binary = 1111 = 15 :  Gray = 1000 =  8
#                  160   Binary = 0000 =  0 :  Gray = 0000 =  0
```