The implementation of 2 bit magnitude comparator using EX-NOR and AND gates using the above expressions is shown in **Figure 2.34**.

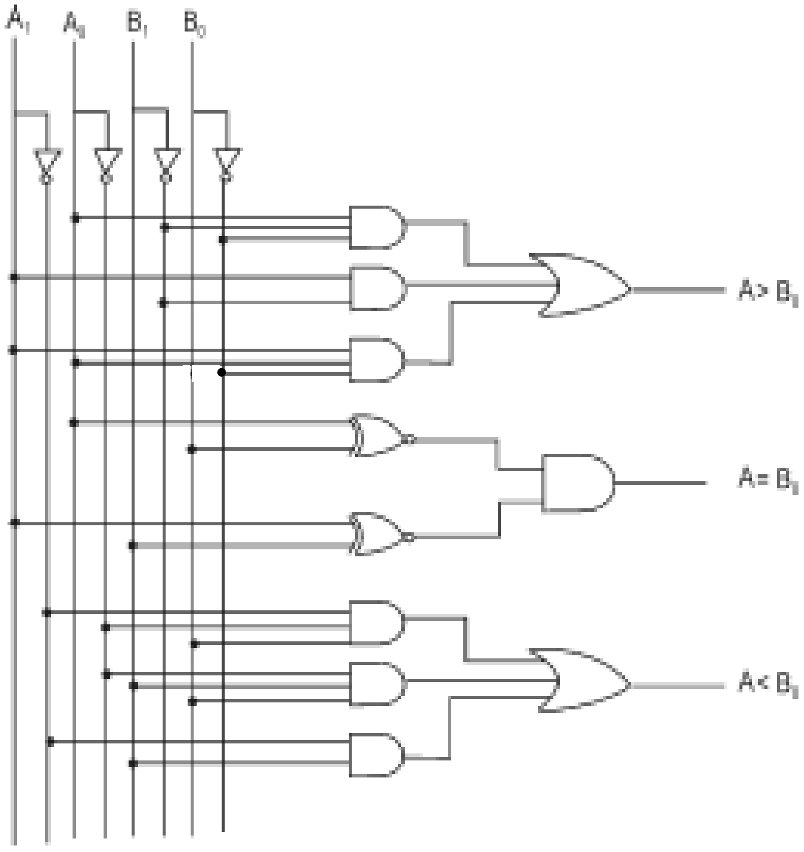


Fig. 2.34 : 2-bit Magnitude comparator

2.17 PARITY GENERATOR AND CHECKER

A parity bit is used for the purpose of detecting errors during transmission of binary information. A parity bit is an extra bit included with a binary message to make the number of 1s either odd or even. The message including the parity bit is transmitted and then checked at the receiving end for errors. An error is detected if the checked parity does not correspond with the one transmitted. The circuit that generates the parity bit in the transmitter is called a parity generator and the circuit that checks the parity bit in the transmitter is called a parity generator and the circuit that checks the parity in the receiver is called a parity checker.

Even parity means an ‘n’ bit input has an even number of 1s. For example, 110101 has even parity because it contains four 1s.

Odd parity means an ‘n’ bit input has an odd number of 1s. For example, 110100 has odd parity because it contains three 1s.

2.17.1 Parity Checker

Exclusive-OR gates are used for checking the parity of a binary number because they produce an output 1 when the input has an odd number of 1s. Therefore, an even parity input to an EX-OR gate produces a low output, while an odd parity input produces a high output. Remember the truth table for EX-OR gate as given below:

Inputs		*Outputs*
0	0	0
0	1	1
1	0	1
1	1	0

Figure 2.35shows a 4 bit parity checker. The output is 1 when the number of 1s in the inputs is odd and output is 0 when the number of 1s in the inputs is even. For example, when 1001 is the input given to the 4 bit parity generator, the output is 0, because the number has two 1s.

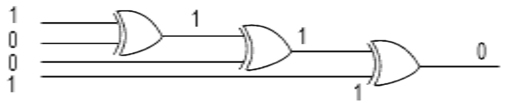


Fig. 2.35: 4 bit parity checker

Figure 2.36shows a 16 input EX-OR gate. A 16 bit number drives the input. The EX-OR gate produces an output 1 because the input (1010101010101000) has odd parity.

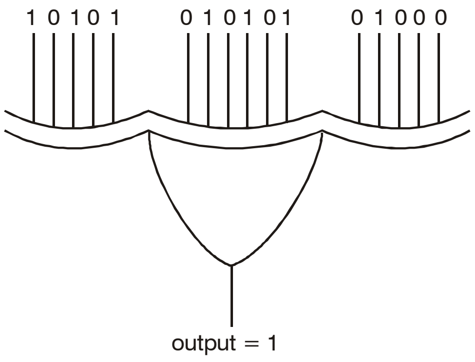


Fig. 2.36: 16 bit parity checker

2.17.2 Parity Generator

Figure 2.34 shows the odd-parity generator. Let the 8 bit binary number,

$$X_7 X_6 X_5 X_4 X_3 X_2 X_1 X_0 = 0100\ 0001$$

Then the number has even parity, which means the EX-OR gate produces an output of 0. Because of the inverter $X_8 = 1$ and the final 9-bit output is,

1 0100 0001

This 9 bit output has odd parity.

Suppose the input is 0110 0001. Now it has odd parity. The EX-OR produces an output 1. But the inverter produces a 0 (X_8= 0), so that the final 9 bit output is,

0 0100 0001.

Again the final output has odd parity,

Thus odd-parity generator produces a 9 bit output number with odd parity. If the 8 bit input has even parity, a 1 comes out of the inverter to produce a final output with odd parity. If the 8 bit input has odd parity, a 0 comes out of the inverter and the final 9 bit output again has odd parity.

To get **even parity generator** delete the inverter in the circuit shown in **Figure 2.37** .

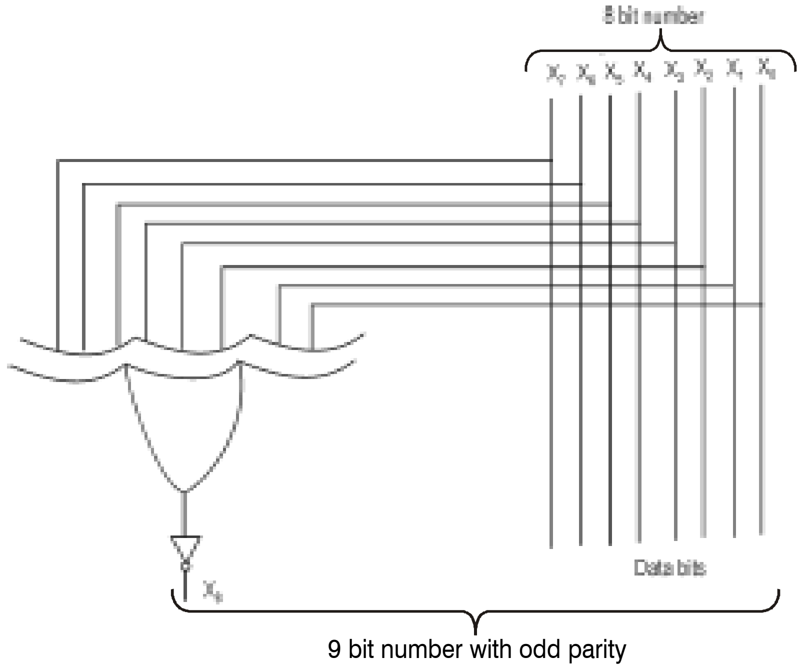


Fig. 2.37 : Odd-parity generator

2.18 CODE CONVERTERS

A code converter is a logic circuit that changes data presented in one type of binary code to another type of binary code. Now we will discuss about some code converters.

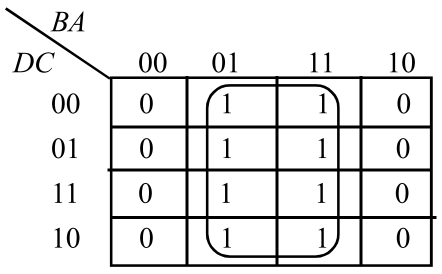
2.18.1 Binary to BCD Converter

Table 2.6 shows the binary codes and corresponding BCD codes. **(Dec. 2005)**

TABLE 2.6 : Truth Table for Binary to BCD converter

Binary Code				*BCD code*				
D	C	B	A	B_4	B_3	B_2	B_1	B_0
0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	1
0	0	1	0	0	0	0	1	0
0	0	1	1	0	0	0	1	1
0	1	0	0	0	0	1	0	0
0	1	0	1	0	0	1	0	1
0	1	1	0	0	0	1	1	0
0	1	1	1	0	0	1	1	1
1	0	0	0	0	1	0	0	0
1	0	0	1	0	1	0	0	1
1	0	1	0	1	0	0	0	0
1	0	1	1	1	0	0	0	1
1	1	0	0	1	0	0	1	0
1	1	0	1	1	0	0	1	1
1	1	1	0	1	0	1	0	0
1	1	1	1	1	0	1	0	1

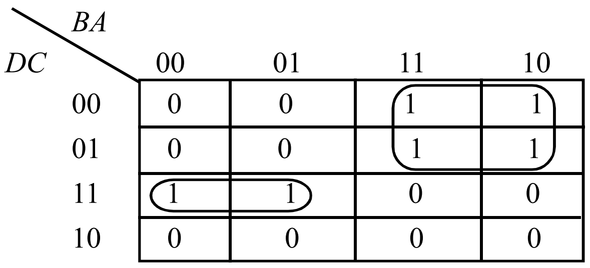
For B_0



$DC \backslash BA$	00	01	11	10
00	0	1	1	0
01	0	1	1	0
11	0	1	1	0
10	0	1	1	0

$B_0 = A$

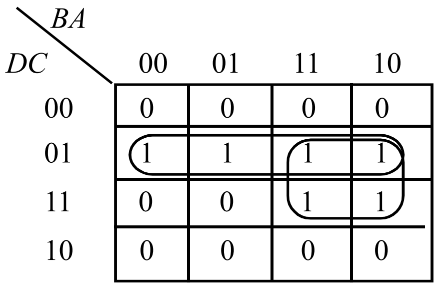
For B_1



$DC \backslash BA$	00	01	11	10
00	0	0	1	1
01	0	0	1	1
11	1	1	0	0
10	0	0	0	0

$B_1 = DC\overline{B} + \overline{D}B$

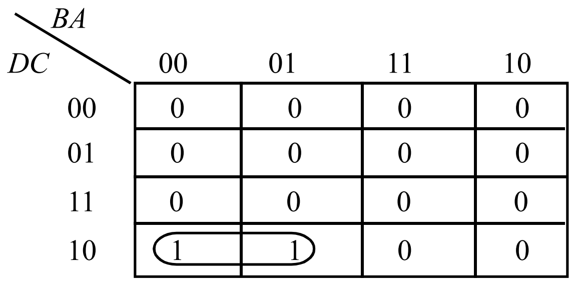
For B_2



$DC \backslash BA$	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	0	0	1	1
10	0	0	0	0

$B_2 = \overline{D}C + CB$

For B_3



$DC \backslash BA$	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	0	0	0	0
10	1	1	0	0

$B_3 = D\overline{C}\overline{B}$

For B_4

$DC \backslash BA$	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	1	1	1	1
10	0	0	1	1

$B_4 = DC + DB$

$$\boxed{\begin{aligned} B_0 &= A \\ B_1 &= DC\overline{B} + \overline{D}B \\ B_2 &= \overline{D}C + CB \\ B_3 &= D\overline{C}\overline{B} \\ B_4 &= DC + DB \end{aligned}}$$

Logic Diagram

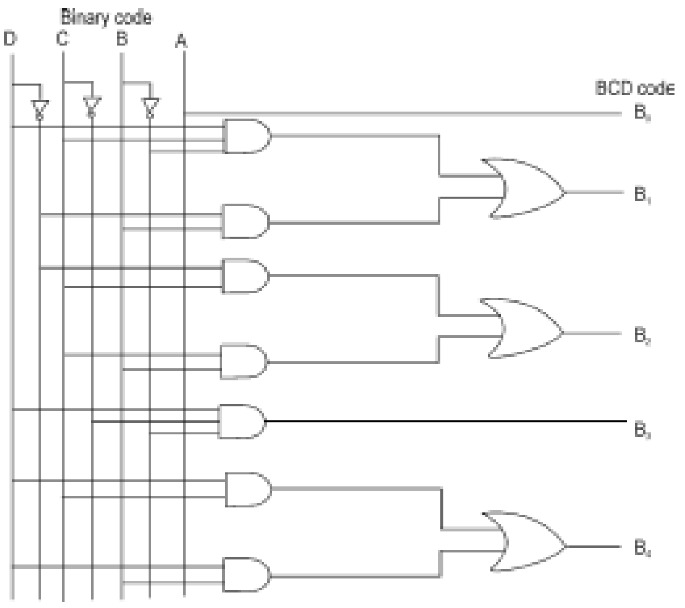


Fig. 2.38: Binary to BCD converter

2.18.2 BCD to Excess 3 Converter **(April 2004)**

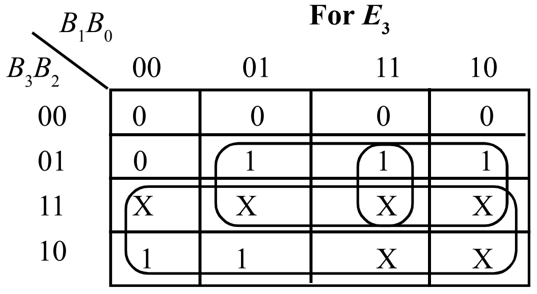
Excess-3 code is a modified form of a BCD number. The Excess-3 code can be derived from the natural BCD code by adding 3 to each coded number. The truth table for BCD to excess 3 code converter is shown in **Table 2.7**.

TABLE 2.7 : Truth Table for BCD to Excess 3 converter

Decimal	B_3	B_2	B_1	B_0	E_3	E_2	E_1	E_0
0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	0	1
3	0	0	1	1	0	1	1	0
4	0	1	0	0	0	1	1	1
5	0	1	0	1	1	0	0	0
6	0	1	1	0	1	0	0	1
7	0	1	1	1	1	0	1	0
8	1	0	0	0	1	0	1	1
9	1	0	0	1	1	1	0	0

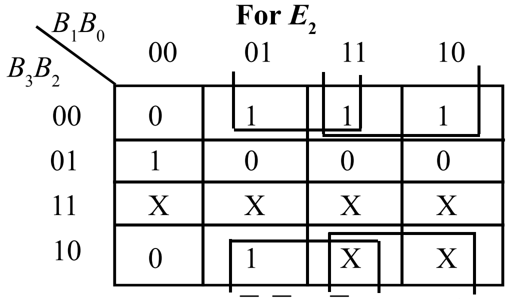
K-map Simplification

For E_3



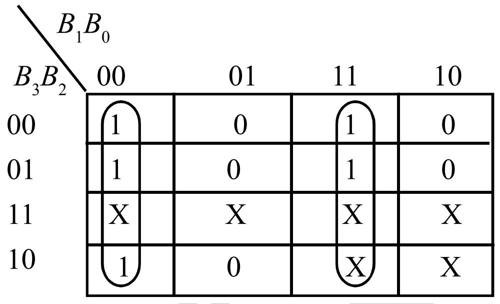
$$E_3 = B_3 + B_2\left(B_0 + B_1\right)$$

For E_2



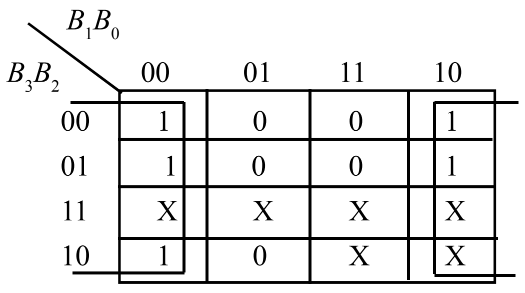
$$E_2 = B_2\,\overline{B_1}\,\overline{B_0} + \overline{B_2}\left(B_0 + B_1\right)$$

For E_1



$$E_1 = \overline{B_1}\,\overline{B_0} + B_1 B_0 = \overline{B_1 \oplus B_0}$$

For E_0



$$E_0 = \overline{B_0}$$

$$\boxed{\begin{aligned} E_3 &= B_3 + B_2\left(B_0 + B_1\right) \\ E_2 &= B_2\,\overline{B_1}\,\overline{B_0} + \overline{B_2}\left(B_0 + B_1\right) \\ E_1 &= \overline{B_1 \oplus B_0} \\ E_0 &= \overline{B_0} \end{aligned}}$$

BCD code

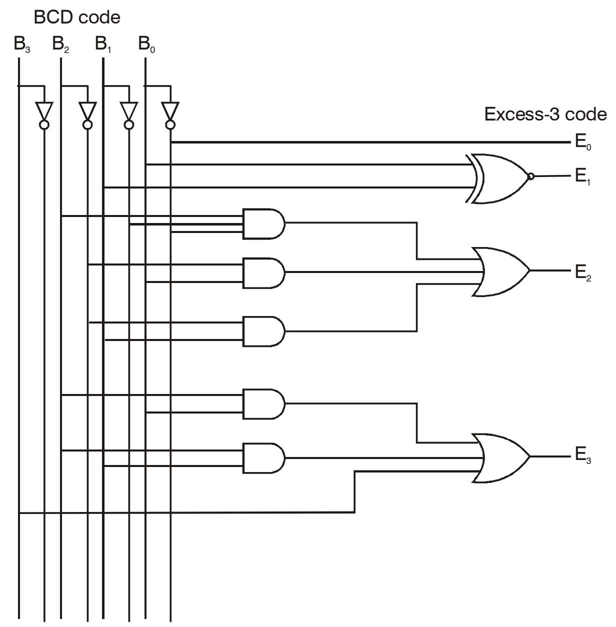


Fig. 2.39: BCD to Excess-3 code converter

2.18.3 Binary to Gray Code Converter **(Dec. 2005)**

The Gray code is often used in digital systems because it has the advantage that only one bit in the numerical representation changes between successive numbers. **Table 2.8** shows decimal and Binary codes and corresponding Gray code.

TABLE 2.8

Decimal	*Binary code*				*Gray code*			
	D	***C***	***B***	***A***	G_3	G_2	G_1	G_0
0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1
2	0	0	1	0	0	0	1	1
3	0	0	1	1	0	0	1	0
4	0	1	0	0	0	1	1	0
5	0	1	0	1	0	1	1	1
6	0	1	1	0	0	1	0	1
7	0	1	1	1	0	1	0	0
8	1	0	0	0	1	1	0	0
9	1	0	0	1	1	1	0	1
10	1	0	1	0	1	1	1	1
11	1	0	1	1	1	1	1	0
12	1	1	0	0	1	0	1	0
13	1	1	0	1	1	0	1	1
14	1	1	1	0	1	0	0	1
15	1	1	1	1	1	0	0	0

K-map Simplification

For G_0

DC \ BA	00	01	11	10
00	0	1	0	1
01	0	1	0	1
11	0	1	0	1
10	0	1	0	1

$$G_0 = \overline{B}A + B\overline{A}$$
$$= B \oplus A$$

For G_1

DC \ BA	00	01	11	10
00	0	0	1	1
01	1	1	0	0
11	1	1	0	0
10	0	0	1	1

$$G_1 = C\overline{B} + \overline{C}B$$
$$= C \oplus B$$

For G_2

DC \ BA	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	0	0	0	0
10	1	1	1	1

$$G_2 = \overline{D}C + D\overline{C} = D \oplus C$$

For G_3

DC \ BA	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	1	1	1	1
10	1	1	1	1

$$G_3 = D$$

$$\boxed{\begin{aligned} G_0 &= B \oplus A \\ G_1 &= C \oplus B \\ G_2 &= D \oplus C \\ G_3 &= D \end{aligned}}$$

Logic Diagram

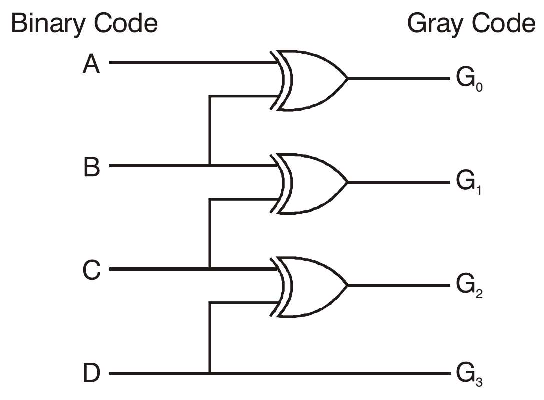


Fig. 2.40 : Binary to gray code converter

2.18.4 Gray Code to Binary Code Converter

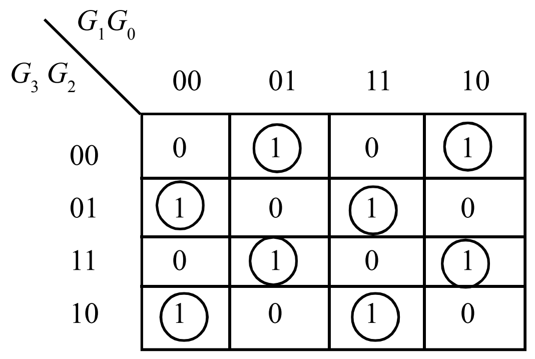
Table 2.9 shows the truth table for gray code to binary code converter.

TABLE 2.9 : Truth Table for gray code to binary code converter

Gray code				Binary code			
G_3	G_2	G_1	G_0	D	C	B	A
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	1	0	0	1	0
0	0	1	0	0	0	1	1
0	1	1	0	0	1	0	0
0	1	1	1	0	1	0	1
0	1	0	1	0	1	1	0
0	1	0	0	0	1	1	1
1	1	0	0	1	0	0	0
1	1	0	1	1	0	0	1
1	1	1	1	1	0	1	0
1	1	1	0	1	0	1	1
1	0	1	0	1	1	0	0
1	0	1	1	1	1	0	1
1	0	0	1	1	1	1	0
1	0	0	0	1	1	1	1

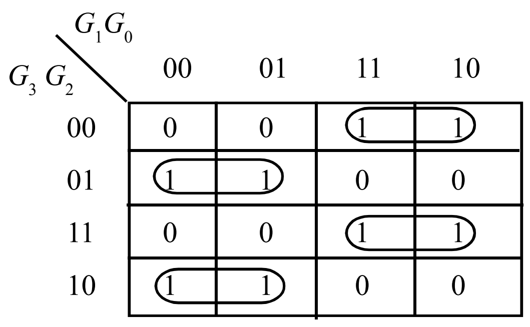
K-map Simplification

For A



$G_3G_2 \backslash G_1G_0$	00	01	11	10
00	0	1	0	1
01	1	0	1	0
11	0	1	0	1
10	1	0	1	0

For B



$G_3G_2 \backslash G_1G_0$	00	01	11	10
00	0	0	1	1
01	1	1	0	0
11	0	0	1	1
10	1	1	0	0

$$A=\left(\overline{G}_3G_2+G_3\overline{G}_2\right)\overline{G}_1\overline{G}_0+\left(\overline{G}_3\overline{G}_2+G_3G_2\right)\overline{G}_1G_0+\left(\overline{G}_3G_2+G_3\overline{G}_2\right)G_1G_0+\left(\overline{G}_3\overline{G}_2+G_3G_2\right)G_1\overline{G}_0$$

$$=\left(G_3\oplus G_2\right)\overline{G}_1\overline{G}_0+\left(G_3\odot G_2\right)\overline{G}_1G_0+\left(G_3\oplus G_2\right)G_1G_0+\left(G_3\odot G_2\right)G_1\overline{G}_0$$

$$=\left(G_3\oplus G_2\right)\left(\overline{G}_1\overline{G}_0+G_1G_0\right)+\left(G_3\odot G_2\right)\left(\overline{G}_1G_0+G_1\overline{G}_0\right)$$

$$=\left(G_3\oplus G_2\right)\left(G_2\odot G_0\right)+\left(G_3\odot G_2\right)\left(G_1\oplus G_0\right)$$

$$=\left(G_3\oplus G_2\right)\left(\overline{G_1\oplus G_0}\right)+\left(G_3\oplus G_2\right)\left(G_1\oplus G_0\right)$$

$$=\left(G_3\oplus G_2\right)\oplus\left(G_1\oplus G_0\right)$$

$$B=\left(\overline{G}_3\overline{G}_2+G_3G_2\right)G_1+\left(\overline{G}_3G_2+G_3\overline{G}_2\right)\overline{G}_1$$

$$=\left(G_3\odot G_2\right)G_1\oplus\left(G_3\oplus G_2\right)\overline{G}_1$$

$$=\left(\overline{G_3\oplus G_2}\right)G_1+\left(G_3\oplus G_2\right)\overline{G}_1=G_3\oplus G_2\oplus G_1$$

$$\boxed{\begin{aligned}&A=\left(G_3\oplus G_2\right)\oplus\left(G_1\oplus G_0\right)\\&B=G_3\oplus G_2\oplus G_1\\&C=G_3\oplus G_2\\&D=G_3\end{aligned}}$$

For *C*

$G_3G_2 \backslash G_1G_0$	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	0	0	0	0
10	1	1	1	1

$C=\overline{G}_3G_2+G_3\overline{G}_2=G_3\oplus G_2$

For *D*

$G_3G_2 \backslash G_1G_0$	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	1	1	1	1
10	1	1	1	1

$D=G_3$

Logic Diagram

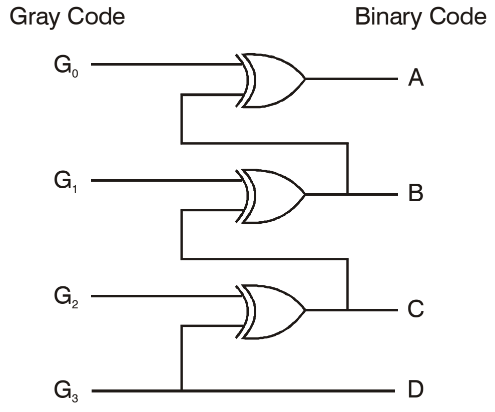


Fig. 2.41 : Gray code to binary code converter