## Odd-Parity generator:

→ Consider the odd parity gerator using an XOR gate and a NOT gate, shown in fig 4(a) below. Let the 8-bit number $x_7 x_6 x_5 x_4 x_3 x_2 x_1 x_0$ be equal to 01000001. This number has even parity. It means that, when it is applied to an exclusive-OR (XOR) gate, it will produce an output of 0. Because of the inverter, $x_8 = 1$ and the final 9-bit output is 1 01000001. Now it has odd parity. Thus, an XNOR gate can be used to generate parity bits.

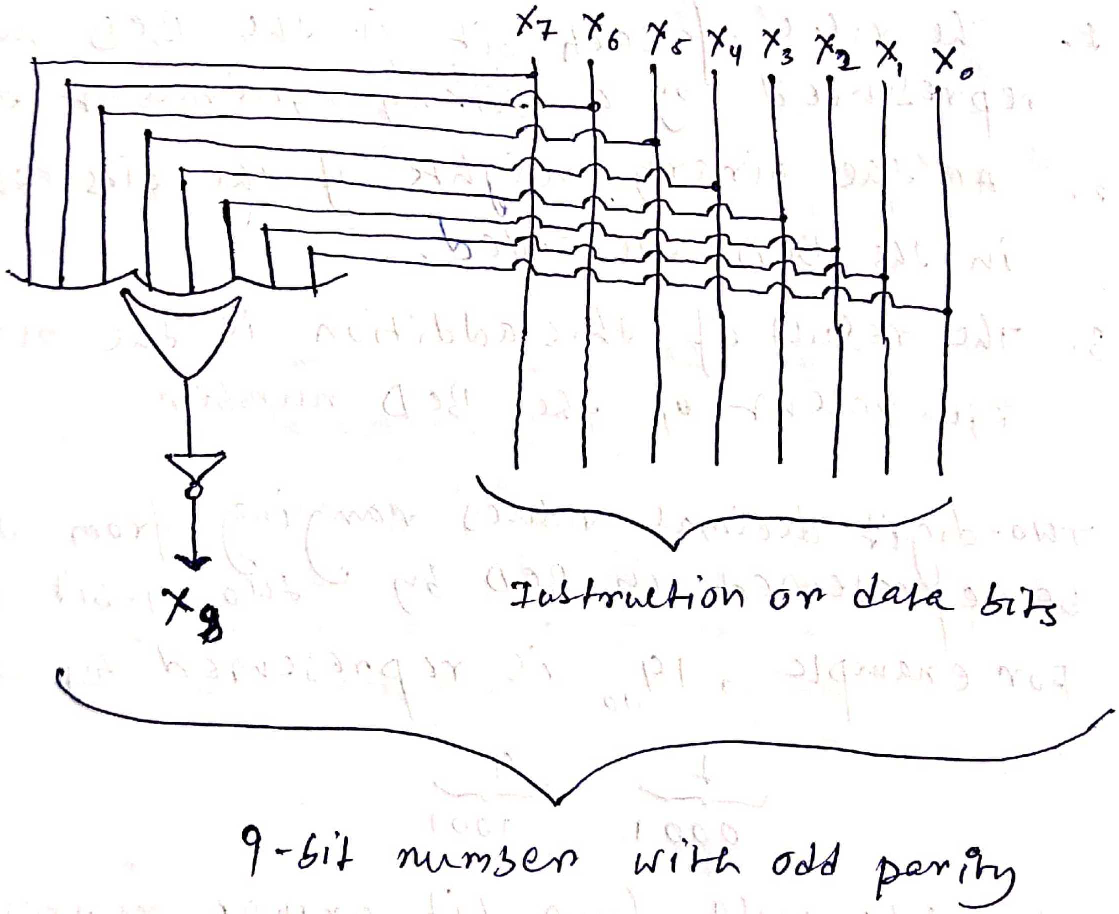



Fig 4(a): odd-parity generation.

## Code converters:

→ A code converter is a logic circuit that changes data presented in one types of binary code to another type of binary code. The following are some of most commonly used code converters:

(i) BCD to binary
(ii) Binary to BCD
(iii) Binary to Gray code
(iv) Gray code to binary

## BCD to binary converters:

The steps involved in the BCD-to-binary conversion process are as follows:

1. The value of each bit in the BCD number is represented by a ~~binary~~ decimal equivalent or weight.
2. All the binary weights of the bits that are 1s in the BCD are added.
3. The result of this addition is the binary equivalent of the BCD number

→ Two-digit decimal values ranging from 00 to 09 can be represented in BCD by two 4-bit code groups.
For example, $19_{10}$ is represented ~~by~~ as

$$\underbrace{0001}_{1} \quad \underbrace{1001}_{9}$$

The left-most four bit groups represents 10 and right-most fours-bit group represents 9.
That is, the left-most group has a weight of $10^1 = 10$

and the right-most group has a weight of $10^0 = 1$. The straight binary representation for decimal 19 is $19_{10} = 10011_2$. Fig 5(a) shows the block diagram of a two-digit BCD-to-binary converter.

The inputs to the converter are the two 4-bit code groups $D_0C_0B_0A_0$ representing the $10^0$ or units digit and $D_1C_1B_1A_1$ representing $10^1$ or tens digit of the decimal values.

→ Let $b_6b_5b_4b_3b_2b_1b_0$ be the outputs of the converter, the 7 bits of the binary equivalent of the decimal values. The binary equivalent of each BCD bit is a binary number representing the weight of that bit within the total BCD number. This is given in table-1(a)

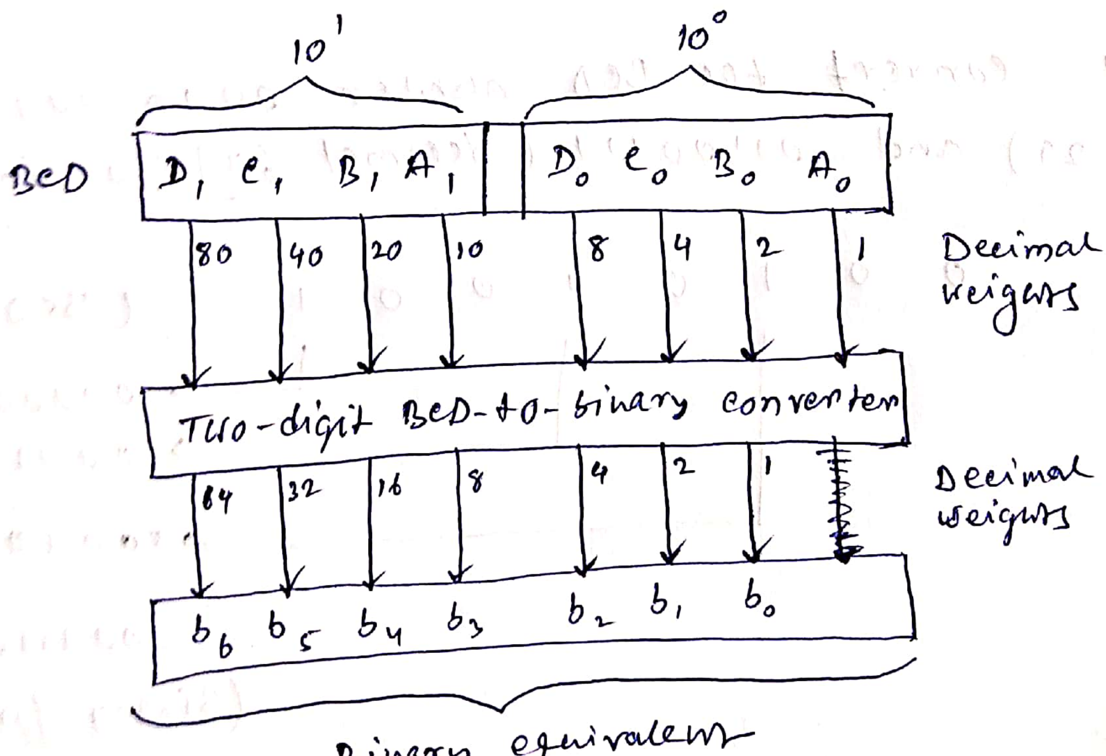



Fig: 5(a) Block diagram of a 2-digit BCD-to-binary converter.

| BCD bit | BCD weight or decimal weight | binary equivalent or weight | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | $b_6$ (=64) | $b_5$ (=32) | $b_4$ (=16) | $b_3$ (=8) | $b_2$ (=4) | $b_1$ (=2) | $b_0$ (=1) |
| $A_0$ | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| $B_0$ | 2 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| $C_0$ | 4 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| $D_0$ | 8 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| $A_1$ | 10 | 0 | 0 | 0 | 1 | 0 | 1 | 0 |
| $B_1$ | 20 | 0 | 0 | 1 | 0 | 1 | 0 | 0 |
| $C_1$ | 40 | 0 | 1 | 0 | 1 | 0 | 0 | 0 |
| $D_1$ | 80 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |

→ Using the weights, the BCD to binary conversion can be done easily by computing the binary sum of the binary equivalents or weights of those bits in the BCD representation that are 1s.

<u>Ex.-1</u> Convert the BCD number 0010 1001 (decimal 29) and 01100111 (decimal 67) to binary.

⇒ 0 0 1 0 1 0 0 1 (BCD)

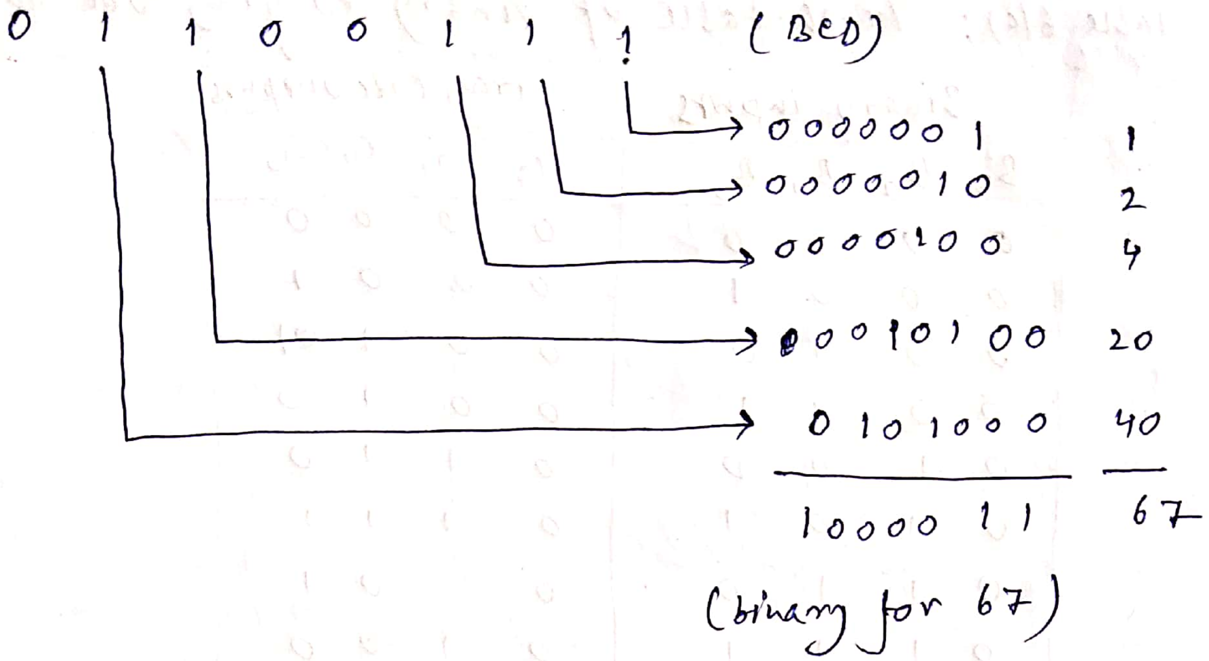
→ 00000001 1

→ 00010000 8

→ 00010100 20

00 11101 29

(Binary for 29)




## Binary-to-Gray code converters:

The block diagram of a 4-bit binary-to-gray code converter is shown in fig 6(a). It has four inputs ($B_3 B_2 B_1 B_0$) representing 4-bit binary numbers and four outputs ($G_3 G_2 G_1 G_0$) representing 4-bit gray code. The truth table for the binary-to-gray code converter is shown in table 6(a)

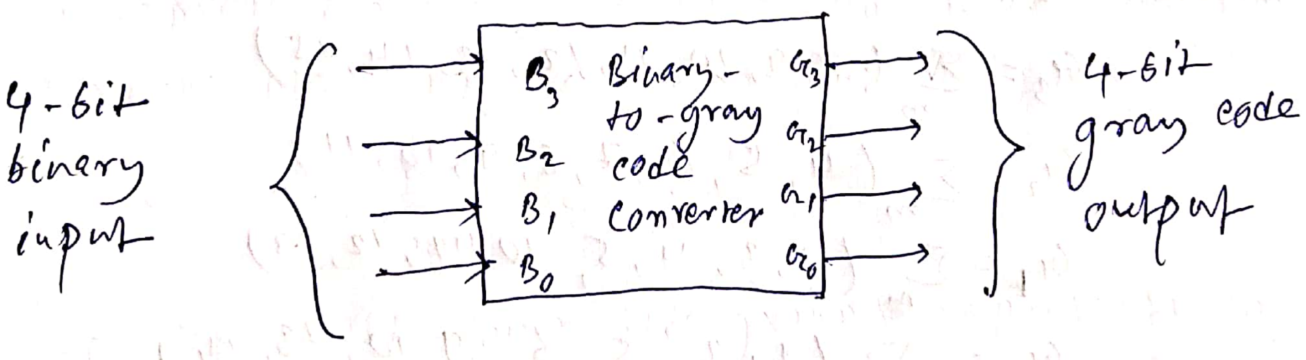



Fig 6(a): Block diagram of 4-bit binary-to-gray code converter.

Table 6(a): Truth-table of binary to gray code converters.

| Binary inputs | | | | Gray code outputs | | | |
|---|---|---|---|---|---|---|---|
| $B_3$ | $B_2$ | $B_1$ | $B_0$ | $G_3$ | $G_2$ | $G_1$ | $G_0$ |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 | 1 | 1 | 1 |
| 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 1 | 1 | 1 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |

From the truth table shown in table 6(a), the logic expressions for the gray code outputs can be written as:

$$G_3 = \sum_m (8, 9, 10, 11, 12, 13, 14, 15)$$

$$G_2 = \sum_m (4, 5, 6, 7, 8, 9, 10, 11)$$

$$G_1 = \sum_m (2, 3, 4, 5, 10, 11, 12, 13)$$

$$G_0 = \sum_m (1, 2, 5, 6, 9, 10, 13, 14)$$

The above expressions can be simplified using K-map method as shown in Table 6(b) – 6(e).

| $B_1B_0$ \ $B_3B_2$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 0 | 0 | 1 | 1 |
| 01 | 0 | 0 | 1 | 1 |
| 11 | 0 | 0 | 1 | 1 |
| 10 | 0 | 0 | 1 | 1 |

6(b): K-map of $G_3$

| $B_1B_0$ \ $B_3B_2$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 0 | 1 | 0 | 1 |
| 01 | 0 | 1 | 0 | 1 |
| 11 | 0 | 1 | 0 | 1 |
| 10 | 0 | 1 | 0 | 1 |

6(c): K-map of $G_2$

| $B_1B_0$ \ $B_3B_2$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 0 | 1 | 1 | 0 |
| 01 | 0 | 1 | 1 | 0 |
| 11 | 1 | 0 | 0 | 1 |
| 10 | 1 | 0 | 0 | 1 |

6(d): K-map of $G_1$

| $B_1B_0$ \ $B_3B_2$ | 00 | 01 | 11 | 10 |
|---|---|---|---|---|
| 00 | 0 | 0 | 0 | 0 |
| 01 | 1 | 1 | 1 | 1 |
| 11 | 0 | 0 | 0 | 0 |
| 10 | 1 | 1 | 1 | 1 |

6(e): K-map of $G_0$

→ From table 6(b), $G_3 = B_3$ —— (i)

From table 6(c), $G_2 = B_2\overline{B}_3 + B_3\overline{B}_2$ —— (ii)

From table 6(d), $G_1 = B_2\overline{B}_1 + \overline{B}_2B_1$ —— (iii)

From table 6(e), $G_0 = \overline{B}_1B_0 + B_1\overline{B}_0$ —— (iv)

→ Now, the above expressions can be implemented using

XOR gates as shown in Fig 6(b) below.

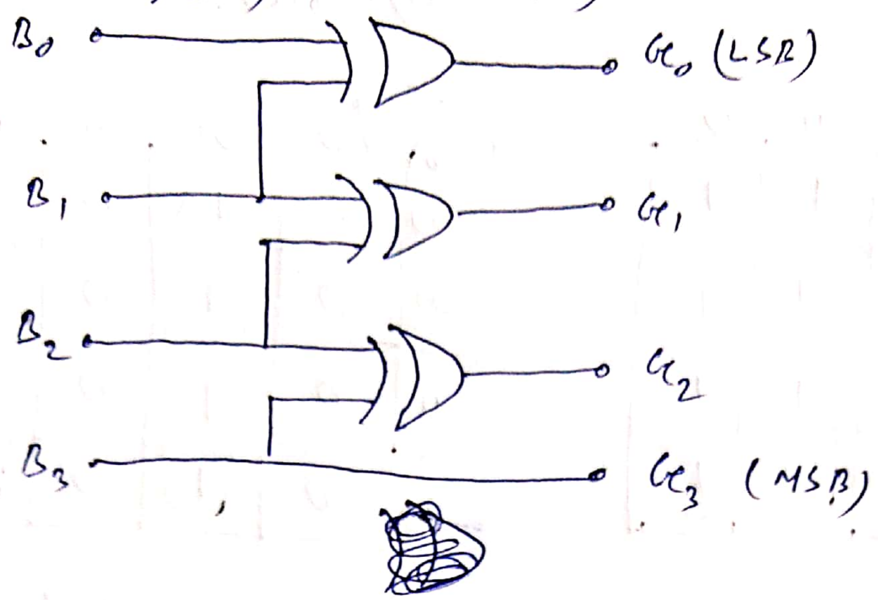



Fig 6(b): Logic diagram of 4-bit binary-to-gray code converter.

## Gray-to-binary converters:

→ The block diagram of a 4-bit gray-to-binary converter is shown in fig 6(a). It has four inputs ($G_3 G_2 G_1 G_0$) representing 4-bit gray code and four outputs ($B_3 B_2 B_1 B_0$) representing 4-bit binary number. The truth table for the gray code-to-binary converter is shown in table 6(a).

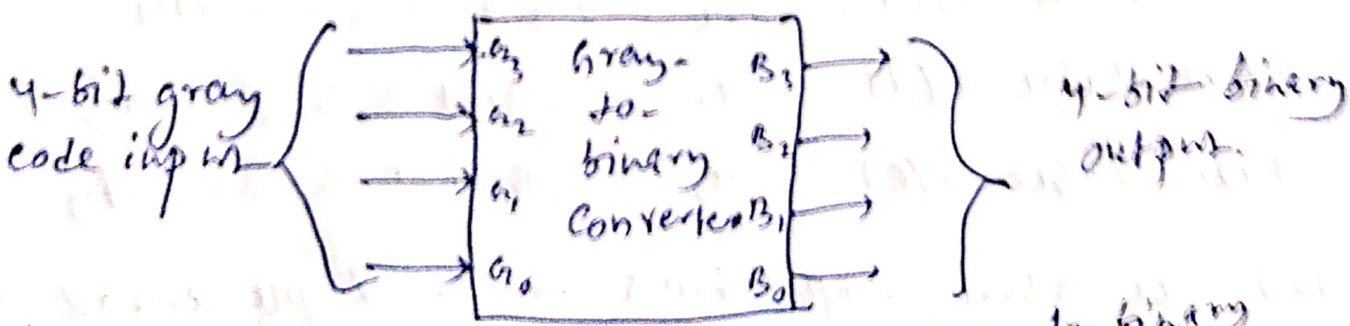



Fig 6(a): Block diagram of 4-bit gray-to-binary converter.

→ The truth table of the gray code-to-binary converter is shown in table 6(a).

Table 6(a): Truth table of gray code-to-binary converter.

| Gray code output | | | | Binary outputs | | | |
|---|---|---|---|---|---|---|---|
| $G_3$ | $G_2$ | $G_1$ | $G_0$ | $B_3$ | $B_2$ | $B_1$ | $B_0$ |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 0 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 1 | 1 | 0 | 1 | 0 |

→ From the above truth table, the logic expressions for the binary outputs can be written as:

$$B_3 = \sum_m (8, 9, 10, 11, 12, 13, 14, 15)$$

$$B_2 = \sum_m (4, 5, 6, 7, 8, 9, 10, 11)$$

$$B_1 = \sum_m (2, 3, 4, 5, 8, 9, 14, 15)$$

$$B_0 = \sum_m (1, 2, 4, 7, 8, 11, 13, 14)$$

The above expressions can be simplified using K-map method as shown in table 6(b) - 6(e).

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

Table: 6(b)

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

Table: 6(c)

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

Table: 6(d)

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

Table: 6(e)

Scanned by CamScanner

→ From table 6(b), $B_3 = G_3$ — (i)

From table 6(c), $B_2 = \overline{G_3} G_2 + G_3 \overline{G_2} = G_3 \oplus G_2$ — (ii)

From table 6(d), $B_1 = \overline{G_3}\,\overline{G_2} G_1 + \overline{G_3} G_2 \overline{G_1} + G_3 G_2 G_1 + G_3 \overline{G_2}\,\overline{G_1}$

~~From table 6(e),~~

or $B_1 = \overline{G_3}(\overline{G_2} G_1 + G_2 \overline{G_1}) + G_3 (G_2 G_1 + \overline{G_2}\,\overline{G_1})$

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

$= G_3 \oplus G_2 \oplus G_1$

$= G_3 \oplus B_2$ — (iii) $[\because B_2 = G_3 \oplus G_2]$

From table 6(e),

$B_0 = \overline{G_3}\,\overline{G_2}\,\overline{G_1} G_0 + \overline{G_3}\,\overline{G_2} G_1 \overline{G_0} + \overline{G_3} G_2 \overline{G_1}\,\overline{G_0} + \overline{G_3} G_2 G_1 G_0$
$+ G_3 G_2 \overline{G_1} G_0 + G_3 G_2 G_1 \overline{G_0} + G_3 \overline{G_2}\,\overline{G_1}\,\overline{G_0} + G_3 \overline{G_2} G_1 G_0$

$= \overline{G_3}\,\overline{G_2}(\overline{G_1} G_0 + G_1 \overline{G_0}) + G_3 G_2 (\overline{G_1} G_0 + G_1 \overline{G_0})$
$+ \overline{G_1}\,\overline{G_0}(\overline{G_3} G_2 + G_3 \overline{G_2}) + G_1 G_0 (\overline{G_3} G_2 + G_3 \overline{G_2})$

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

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

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

$= G_0 \oplus G_1 \oplus G_2 \oplus G_3$

$= G_0 \oplus B_1$ — (iv)

Now the above expressions can be implemented using ~~XNOR~~ XOR gates as shown in fig 6(b).

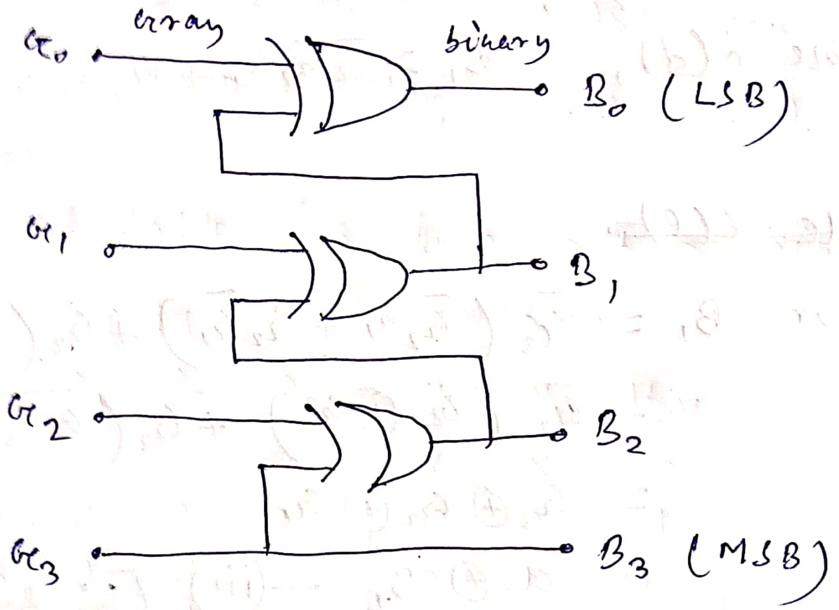



Fig 6(b): Logic diagram of 4-bit gray code-to-binary converter.

---

## ☐ Priority Encoder:

→ A priority encoder is an encoder that includes the priority function. The operation of the priority encoder is such that if two or more inputs are equals to 1 at the same time, the input having the highest priority will take precedence.

→ The truth table of four-input priority encoder is given in table below.

→ The Xs are don't care conditions that designate the fact that the binary values they represent may be equal to 0 or 1.

→ Input $D_3$ has the highest priority; so regardless of the values of the other inputs, when this input is 1, the output $Y_2Y_1$ is 11 (i.e. 3).

Table: Truth table of a four-input priority encoder

| Inputs | | | | Outputs | | |
|---|---|---|---|---|---|---|
| $D_0$ | $D_1$ | $D_2$ | $D_3$ | $Y_2$ | $Y_1$ | V |
| 0 | 0 | 0 | 0 | X | X | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 |
| X | 1 | 0 | 0 | 0 | 1 | 1 |
| X | X | 1 | 0 | 1 | 0 | 1 |
| X | X | X | 1 | 1 | 1 | 1 |

→ $D_2$ has the next priority level. The output is 10 if $D_2=1$ and $D_3=0$, irrespective of the values of the other two lower priority inputs.

The output for $D_1$ is generated only if higher priority inputs are 0, and so on down the priority level. A valid output indicator, V is ~~equal to 0, and the other two outputs of the circuit are not used.~~ set to 1 only when one or more of the inputs are equal to 1. If all the inputs are 0, V is equal to 0, and the other two outputs of the circuit are not used.

IC 74147 and IC 74148 are some of the priority encoder ICs.