7.9.1 Cascading IC Parallel Adders

The addition of large binary numbers can be accomplished by cascading two or more parallel adder chips. When two 74LS83 chips are cascaded to add two 8-bit numbers, the first adder adds the 4 LSBs of the numbers. The C_4 output of this adder is connected as the input carry to the first position of the second adder which adds the 4 MSBs of the numbers. The eight sum outputs represent the resultant sum of the two 8-bit numbers. The C_8 is the carry-out of the last position (MSB) of the second adder. The C_8 can be used as an overflow bit or as a carry into another adder stage if still larger binary numbers are to be handled.

7.10 2'S COMPLEMENT ADDITION AND SUBTRACTION USING PARALLEL ADDERS

Most modern computers use the 2's complement system to represent negative numbers and to perform subtraction. Both the addition and subtraction operations of signed numbers can be performed using only the addition operation, if we use the 2's complement form to represent negative numbers.

Figure 7.26 shows a complete circuit that can perform both addition and subtraction in the 2's complement. This adder/subtractor circuit is controlled by the control signal ADD/$\overline{\text{SUB}}$. When the ADD/$\overline{\text{SUB}}$ level is HIGH, the circuit performs the addition of the numbers stored in registers A and B. When the ADD/$\overline{\text{SUB}}$ level is LOW, the circuit subtracts the number in register B from the number in register A. The operation is described as follows:

When ADD/$\overline{\text{SUB}}$ is a 1:

1. AND gates 1, 3, 5, and 7 are enabled, allowing B_0, B_1, B_2, and B_3 to pass to the OR gates 9, 10, 11, and 12. AND gates 2, 4, 6, and 8 are disabled, blocking $\overline{B}_0$, $\overline{B}_1$, $\overline{B}_2$, and $\overline{B}_3$ from reaching the OR gates 9, 10, 11, and 12.
2. The levels B_0 to B_3 pass through the OR gates to the 4-bit parallel adder, to be added to the bits A_0 to A_3. The sum appears at the outputs S_0 to S_3.
3. ADD/$\overline{\text{SUB}}$ = 1 causes no carry into the adder.

When ADD/$\overline{\text{SUB}}$ is a 0:

1. AND gates 1, 3, 5, and 7 are disabled, blocking B_0, B_1, B_2, and B_3 from reaching the OR gates 9, 10, 11, and 12. AND gates 2, 4, 6, and 8 are enabled allowing $\overline{B}_0$, $\overline{B}_1$, $\overline{B}_2$, and $\overline{B}_3$ to pass to the OR gates.
2. The levels $\overline{B}_0$ to $\overline{B}_3$ pass through the OR gates into the 4-bit parallel adder, to be added to bits A_0 to A_3. The C_0 is now 1. Thus, the number in register B is converted to its 2's complement form.
3. The difference appears at the outputs S_0 to S_3.

Circuits like the adder/subtractor of Figure 7.26 are used in computers because they provide a relatively simple means for adding and subtracting signed binary numbers. In most computers, the output is usually transferred into the register A (accumulator) so that the results of the addition or subtraction always end up stored in the register A. This is accomplished by applying a transfer pulse to the CLK inputs of register A.

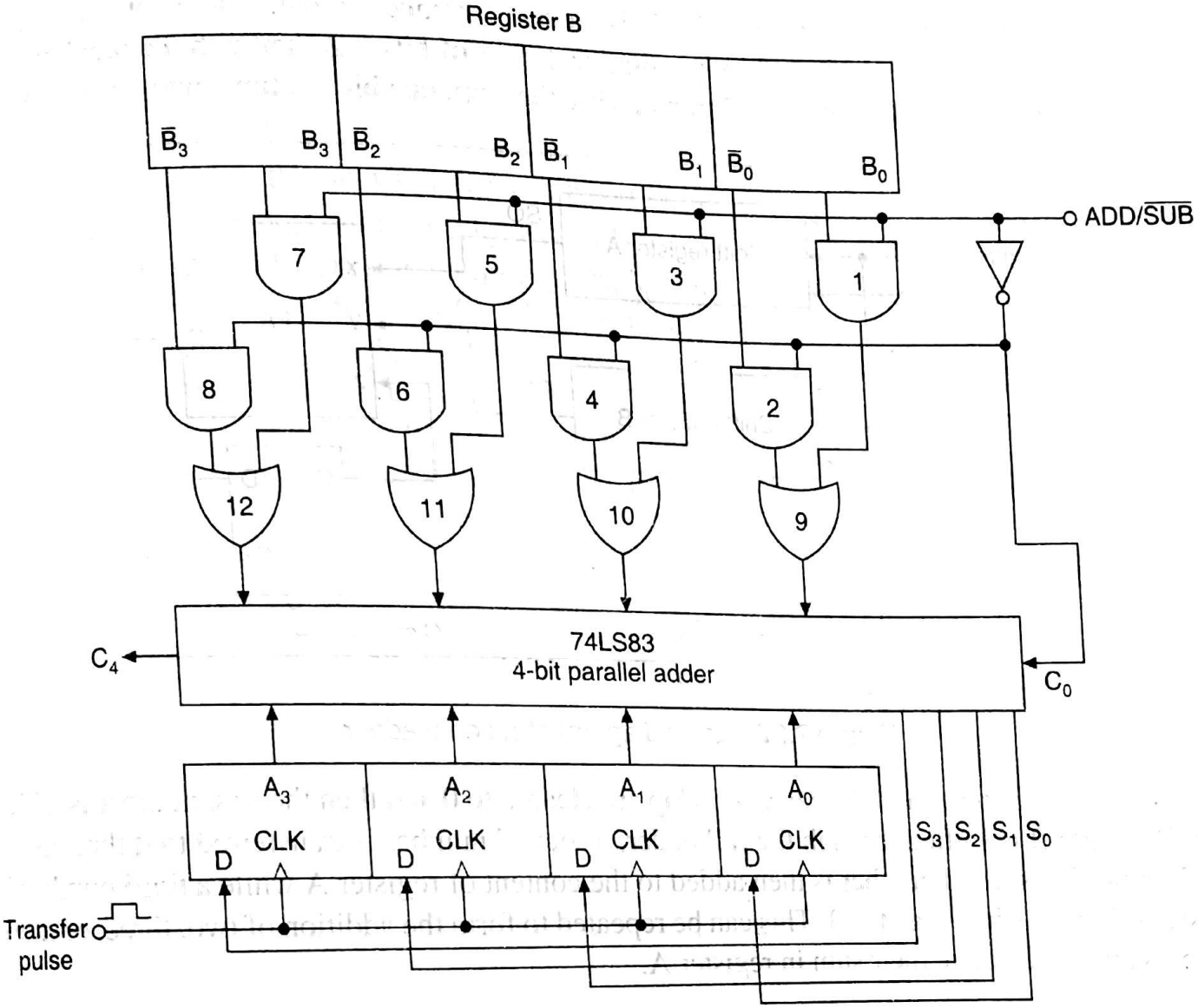


Figure 7.26 Logic diagram of a parallel adder/subtractor using 2's complement system.

the major components ... sion tables may be stored in the ROM. In some systems, conversions are accomplished by the computer itself, through execution of a specially designed program. This is called software conversion, as opposed to the hardware conversion performed by logic circuits.

7.16.1 Design of a 4-bit Binary-to-Gray Code Converter

The input to the 4-bit binary-to-Gray code converter circuit is a 4-bit binary and the output is a 4-bit Gray code. There are 16 possible combinations of 4-bit binary input and all of them are valid. Hence no don't cares. The 4-bit binary and the corresponding Gray code are shown in the conversion table (Figure 7.32a). From the conversion table, we observe that the expressions for the outputs G_4, G_3, G_2, and G_1 are as follows:

$$G_4 = \Sigma m(8, 9, 10, 11, 12, 13, 14, 15)$$
$$G_3 = \Sigma m(4, 5, 6, 7, 8, 9, 10, 11)$$
$$G_2 = \Sigma m(2, 3, 4, 5, 10, 11, 12, 13)$$
$$G_1 = \Sigma m(1, 2, 5, 6, 9, 10, 13, 14)$$

The K-maps for G_4, G_3, G_2, and G_1 and their minimization are shown in Figure 7.32b. The minimal expressions for the outputs obtained from the K-map are:

$$G_4 = B_4$$
$$G_3 = \bar{B}_4 B_3 + B_4 \bar{B}_3 = B_4 \oplus B_3$$
$$G_2 = \bar{B}_3 B_2 + B_3 \bar{B}_2 = B_3 \oplus B_2$$
$$G_1 = \bar{B}_2 B_1 + B_2 \bar{B}_1 = B_2 \oplus B_1$$

So, the conversion can be achieved by using three X-OR gates as shown in the logic diagram in Figure 7.32c.

4-bit binary				4-bit Gray			
B_4	B_3	B_2	B_1	G_4	G_3	G_2	G_1
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

(a) Conversion table

Figure 7.32 4-bit binary-to-Gray code converter *(Contd.)*

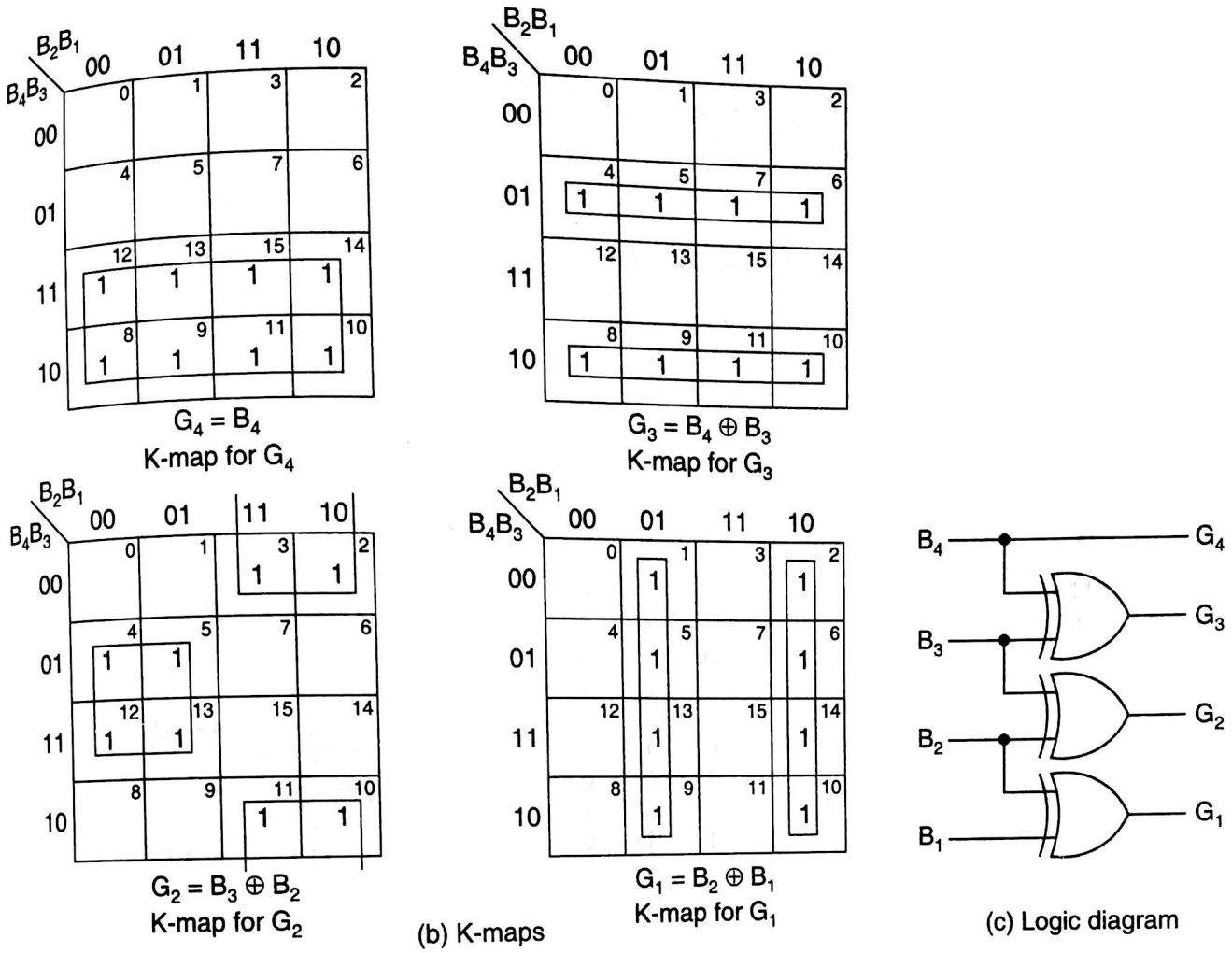


Figure 7.32 4-bit binary-to-Gray code converter.

7.16.2 Design of a 4-bit Gray-to-Binary Code Converter

The input to the 4-bit Gray-to-binary code converter circuit is a 4-bit Gray code and the output is a 4-bit binary. There are 16 possible combinations of 4-bit Gray input and all of them are valid. Hence no don't cares. The 4-bit input Gray code and the corresponding output binary numbers are shown in the conversion table of Figure 7.33a. From the conversion table we observe that the expressions for the outputs B_4, B_3, B_2 and B_1 are:

$$B_4 = \Sigma\, m(12, 13, 15, 14, 10, 11, 9, 8) = \Sigma\, m(8, 9, 10, 11, 12, 13, 14, 15)$$
$$B_3 = \Sigma\, m(6, 7, 5, 4, 10, 11, 9, 8) = \Sigma\, m(4, 5, 6, 7, 8, 9, 10, 11)$$
$$B_2 = \Sigma\, m(3, 2, 5, 4, 15, 14, 9, 8) = \Sigma\, m(2, 3, 4, 5, 8, 9, 14, 15)$$
$$B_1 = \Sigma\, m(1, 2, 7, 4, 13, 14, 11, 8) = \Sigma\, m(1, 2, 4, 7, 8, 11, 13, 14)$$

Drawing the K-maps for B_4, B_3, B_2 and B_1 in terms of G_4, G_3, G_2, and G_1 as shown in Figure 7.33b and simplifying them, the minimal expressions for the outputs are as follows:

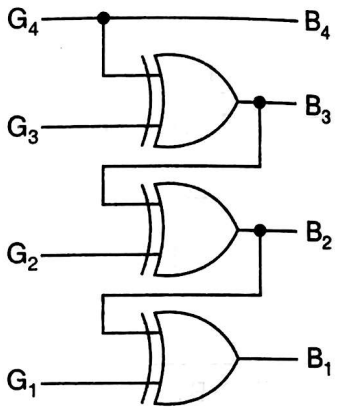
$$B_4 = G_4$$
$$B_3 = \bar{G}_4 G_3 + G_4 \bar{G}_3 = G_4 \oplus G_3$$
$$B_2 = \bar{G}_4 G_3 \bar{G}_2 + \bar{G}_4 \bar{G}_3 G_2 + G_4 \bar{G}_3 \bar{G}_2 + G_4 G_3 G_2$$
$$= \bar{G}_4 (G_3 \oplus G_2) + G_4 (\overline{G_3 \oplus G_2}) = G_4 \oplus G_3 \oplus G_2 = B_3 \oplus G_2$$
$$B_1 = \bar{G}_4 \bar{G}_3 \bar{G}_2 G_1 + \bar{G}_4 \bar{G}_3 G_2 \bar{G}_1 + \bar{G}_4 G_3 G_2 G_1 + \bar{G}_4 G_3 \bar{G}_2 \bar{G}_1 + G_4 G_3 \bar{G}_2 G_1 + G_4 \bar{G}_3 G_2 G_1 + G_4 \bar{G}_3 \bar{G}_2 \bar{G}_1 + G_4 G_3 G_2 \bar{G}_1$$

$$= \bar{G}_4\bar{G}_3(G_2 \oplus G_1) + G_4G_3(G_2 \oplus G_1) + \bar{G}_4G_3\overline{(G_2 \oplus G_1)} + G_4\bar{G}_3\overline{(G_2 \oplus G_1)}$$
$$= (G_2 \oplus G_1)\overline{(G_4 \oplus G_3)} + \overline{(G_2 \oplus G_1)}(G_4 \oplus G_3)$$
$$= G_4 \oplus G_3 \oplus G_2 \oplus G_1$$
$$= B_2 \oplus G_1$$

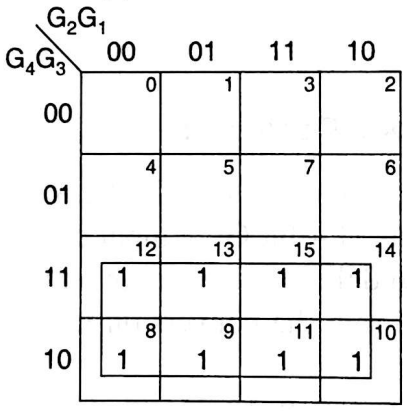
Based on the above expressions, a logic circuit can be drawn as shown in Figure 7.33c.

4-bit Gray				4-bit binary			
G_4	G_3	G_2	G_1	B_4	B_3	B_2	B_1
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

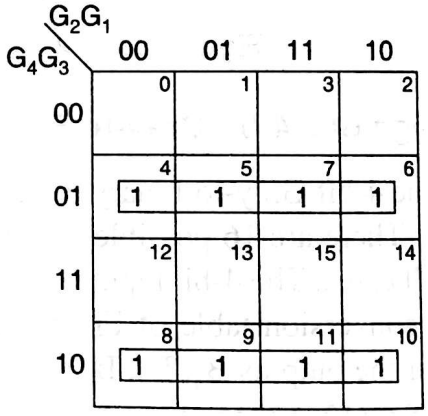
(a) Conversion table



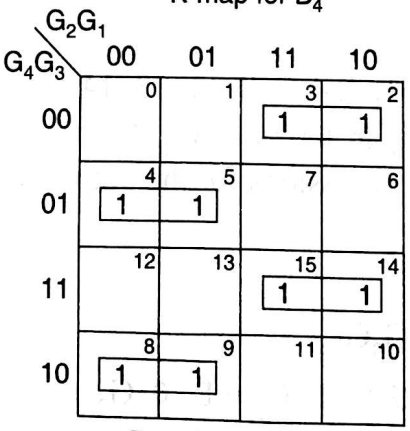
(c) Logic diagram



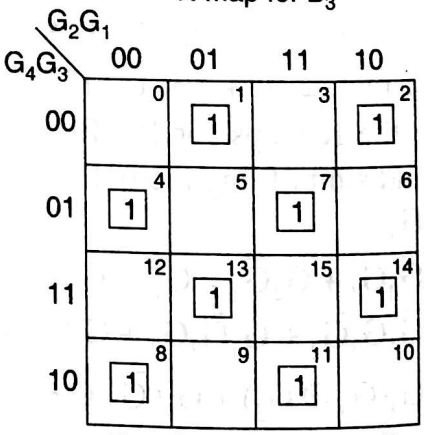
$B_4 = G_4$
K-map for B_4



$B_3 = G_4 \oplus G_3$
K-map for B_3



$B_2 = G_4 \oplus G_3 \oplus G_2$
K-map for B_2



$B_1 = G_4 \oplus G_3 \oplus G_2 \oplus G_1$
K-map for B_1

(b) K-maps

Figure 7.33 4-bit Gray-to-binary code converter.

7.16.3 Design of a 4-bit Binary-to-BCD Code Converter

The input is a 4-bit binary. There are 16 possible combinations of 4-bit binary inputs (representing 0–15) and all are valid. Hence there are no don't cares. Since the input is of 4 bits (i.e. a maximum of 2 decimal digits), the output has to be an 8-bit one; but since the first three bits will all be a 0 for all combinations of inputs, the output can be treated as a 5-bit one. The conversion is shown in the conversion table in Figure 7.34a. From the conversion table, we observe that the expressions for BCD outputs are as follows:

$$A = \Sigma\, m(10, 11, 12, 13, 14, 15)$$
$$B = \Sigma\, m(8, 9)$$
$$C = \Sigma\, m(4, 5, 6, 7, 14, 15)$$
$$D = \Sigma\, m(2, 3, 6, 7, 12, 13)$$
$$E = \Sigma\, m(1, 3, 5, 7, 9, 11, 13, 15)$$

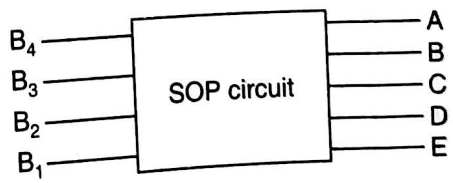
Drawing the K-maps for the outputs and minimizing them as shown in Figure 7.34c the minimal expressions for the BCD outputs A, B, C, D, and E in terms of the 4-bit binary inputs B_4, B_3, B_2, and B_1 are as follows:

$$A = B_4B_3 + B_4B_2$$
$$B = B_4\bar{B}_3\bar{B}_2$$
$$C = \bar{B}_4B_3 + B_3B_2$$
$$D = B_4B_3\bar{B}_2 + \bar{B}_4B_2$$
$$E = B_1$$

A logic diagram can be drawn based on the above minimal expressions.

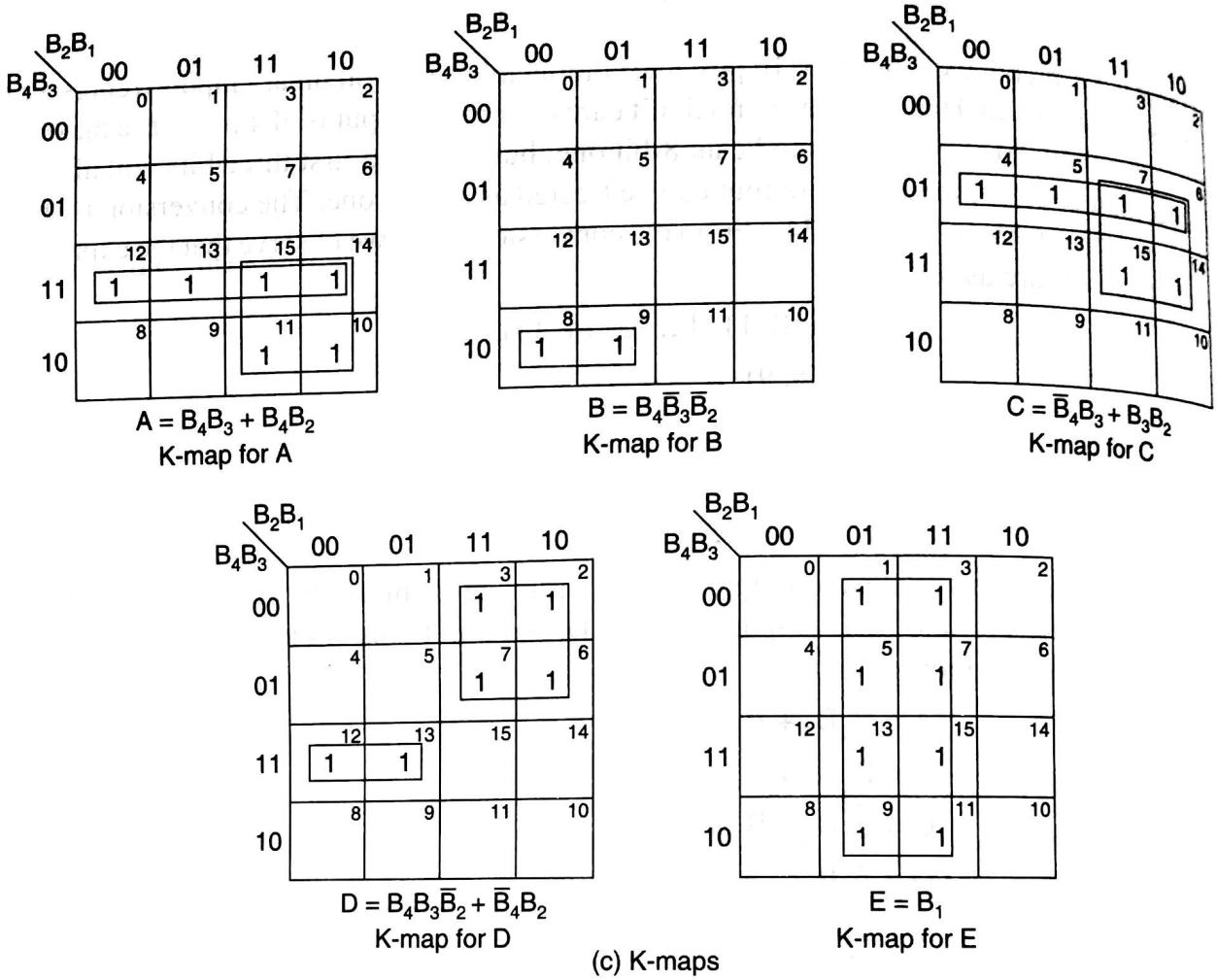
Decimal	4-bit binary				BCD output				
	B_4	B_3	B_2	B_1	A	B	C	D	E
0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	0	1
2	0	0	1	0	0	0	0	1	0
3	0	0	1	1	0	0	0	1	1
4	0	1	0	0	0	0	1	0	0
5	0	1	0	1	0	0	1	0	1
6	0	1	1	0	0	0	1	1	0
7	0	1	1	1	0	0	1	1	1
8	1	0	0	0	0	1	0	0	0
9	1	0	0	1	0	1	0	0	1
10	1	0	1	0	1	0	0	0	0
11	1	0	1	1	1	0	0	0	1
12	1	1	0	0	1	0	0	1	0
13	1	1	0	1	1	0	0	1	1
14	1	1	1	0	1	0	1	0	0
15	1	1	1	1	1	0	1	0	1

(a) Conversion table



(b) Block diagram

Figure 7.34 4-bit binary-to-BCD code converter *(Contd.)*



(c) K-maps

Figure 7.34 4-bit binary-to-BCD code converter.

7.16.4 Design of a 4-bit BCD-to-XS-3 Code Converter

BCD means 8421 BCD. The 4-bit input BCD code ($B_4\ B_3\ B_2\ B_1$) and the corresponding output XS-3 code($X_4\ X_3\ X_2\ X_1$) numbers are shown in the conversion table in Figure 7.35a. The input combinations 1010, 1011, 1100, 1101, 1110, and 1111 are invalid in BCD. So they are treated as don't cares.

8421 code				XS-3 code			
B_4	B_3	B_2	B_1	X_4	X_3	X_2	X_1
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0

(a) Conversion table

$X_4 = \Sigma\, m(5, 6, 7, 8, 9) + d(10, 11, 12, 13, 14, 15)$
$X_3 = \Sigma\, m(1, 2, 3, 4, 9) + d(10, 11, 12, 13, 14, 15)$
$X_2 = \Sigma\, m(0, 3, 4, 7, 8) + d(10, 11, 12, 13, 14, 15)$
$X_1 = \Sigma\, m(0, 2, 4, 6, 8) + d(10, 11, 12, 13, 14, 15)$

The minimal expressions are

$X_4 = B_4 + B_3B_2 + B_3B_1$
$X_3 = B_3\bar{B}_2\bar{B}_1 + \bar{B}_3B_1 + \bar{B}_3B_2$
$X_2 = \bar{B}_2\bar{B}_1 + B_2B_1$
$X_1 = \bar{B}_1$

(b) Minimal expressions

Figure 7.35 4-bit BCD-to-XS-3 code converter *(Contd.)*

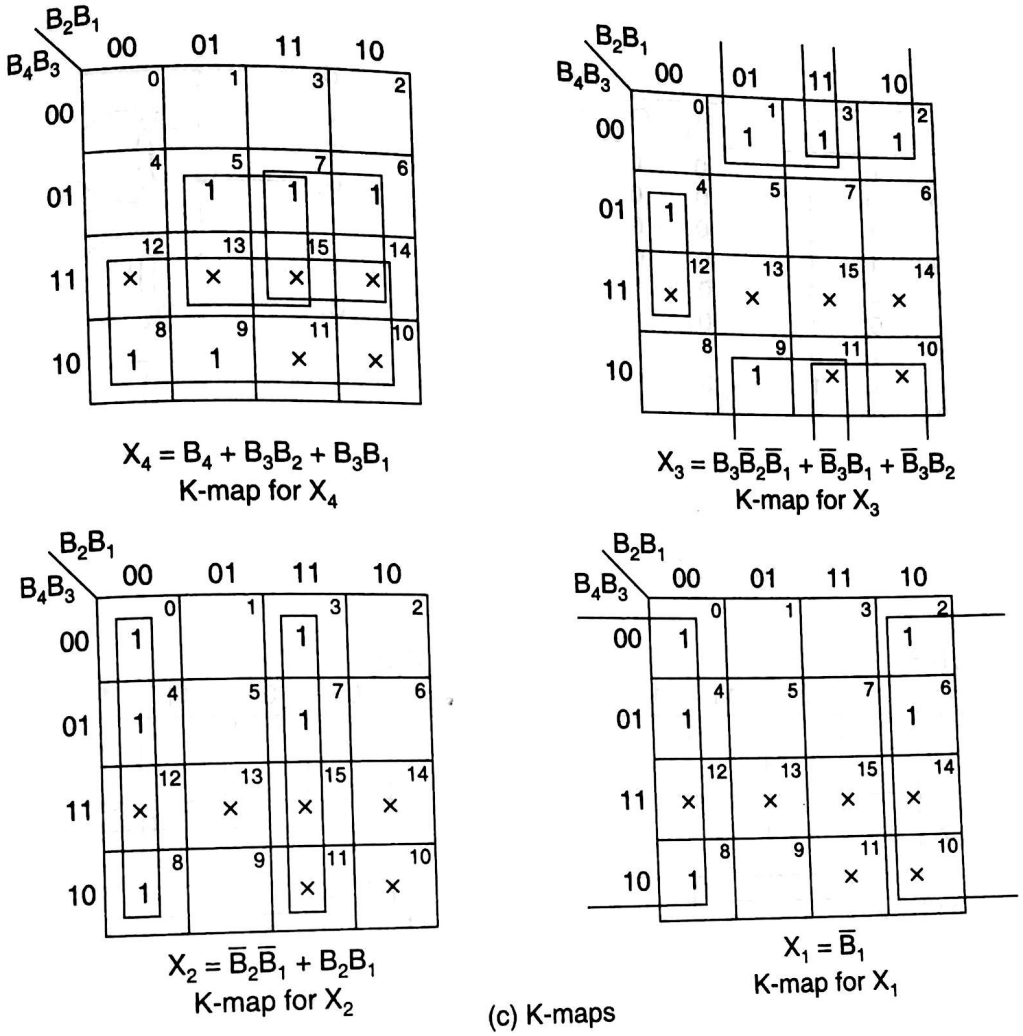


Figure 7.35 4-bit BCD-to-XS-3 code converter.

The expressions for the outputs X_4, X_3, X_2 and X_1 are shown in Figure 7.35b. Drawing K-maps for the outputs X_4, X_3, X_2 and X_1 in terms of the inputs B_4, B_3, B_2, and B_1 and simplifying them, as shown in Figure 7.35c the minimal expressions for X_4, X_3, X_2, and X_1 are as shown in Figure 7.35b. A logic diagram can be drawn based on those minimal expressions.

7.16.5 Design of a BCD-to-Gray Code Converter

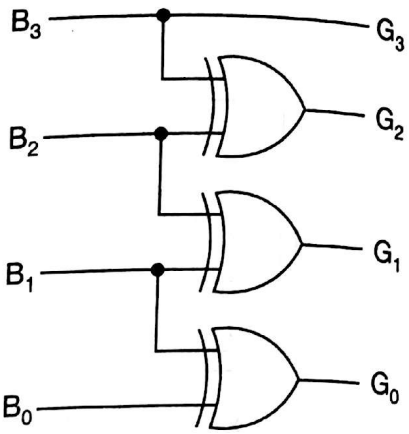
The BCD to Gray code conversion table is shown in Figure 7.36a. For a 4-bit BCD code minterms 10, 11, 12, 13, 14, and 15 are don't cares. So the expressions for the Gray code outputs in terms of BCD inputs are as follows:

$$G_3 = \Sigma\, m(8, 9) + d\,(10, 11, 12, 13, 14, 15)$$
$$G_2 = \Sigma\, m(4, 5, 6, 7, 8, 9) + d\,(10, 11, 12, 13, 14, 15)$$
$$G_1 = \Sigma\, m(2, 3, 4, 5) + d\,(10, 11, 12, 13, 14, 15)$$
$$G_0 = \Sigma\, m(1, 2, 5, 6, 9) + d\,(10, 11, 12, 13, 14, 15)$$

The K-maps for G_3, G_2, G_1, and G_0, their minimization, and the minimal expressions obtained from them are shown in Figure 7.37. The logic diagram of the BCD to Gray code converter based on those minimal expressions is shown in Figure 7.36b.

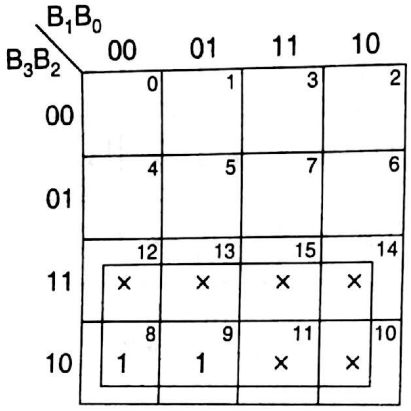
BCD code				Gray code			
B_3	B_2	B_1	B_0	G_3	G_2	G_1	G_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1

(a) BCD-to-Gray code conversion table

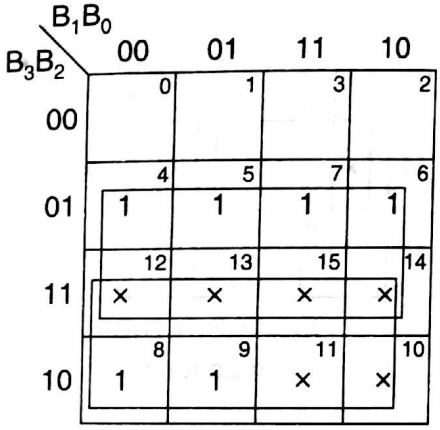


(b) Logic diagram

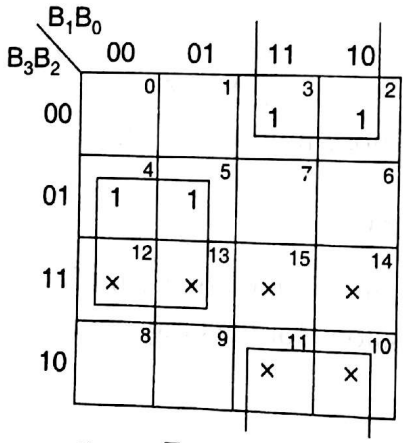
Figure 7.36 BCD-to-Gray code converter.



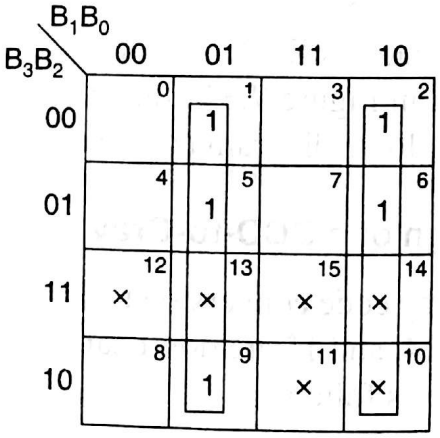
$G_3 = B_3$



$G_2 = B_2 + B_3$



$G_1 = B_2\overline{B}_1 + \overline{B}_2B_1 = B_2 \oplus B_1$



$G_0 = \overline{B}_1B_0 + B_1 \cdot \overline{B}_0 = B_1 \oplus B_0$

Figure 7.37 K-maps for a BCD-to-Gray code converter.

7.16.6 Design of an SOP Circuit to Detect the Decimal Numbers 5 through 12 in a 4-bit Gray Code Input

The input to the SOP circuit is a 4-bit Gray code. Let the input Gray code be ABCD. There are 16 possible combinations of 4-bit Gray code. All of them are valid and hence there are no don't cares.

The truth table of the SOP circuit is shown in Figure 7.38a. Looking at the truth table of the SOP circuit, we observe that the output is 1 for the input combinations corresponding to minterms 7, 5, 4, 12, 13, 15, 14, and 10 (i.e. corresponding to the Gray code of decimal numbers 5, 6, 7, 8, 9, 10, 11 and 12). So the expression for the output is

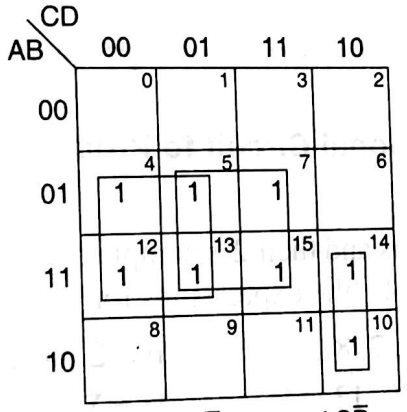
$$f = \Sigma\, m(7, 5, 4, 12, 13, 15, 14, 10) = \Sigma\, m(4, 5, 7, 10, 12, 13, 14, 15)$$

The K-map for f, its minimization, the minimal expression obtained from it and its realization in NAND logic are shown in Figures 7.38b and c respectively.

$$f_{min} = B\bar{C} + BD + AC\bar{D} = \overline{\overline{B\bar{C}} \cdot \overline{BD} \cdot \overline{AC\bar{D}}}$$

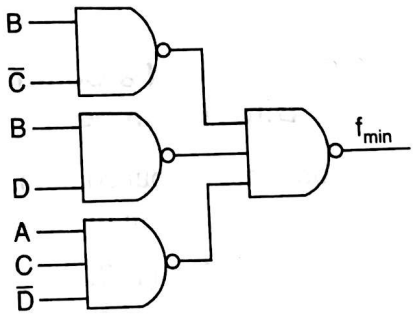
Decimal number	4-bit Gray code A	B	C	D	Output f
0	0	0	0	0	0
1	0	0	0	1	0
2	0	0	1	1	0
3	0	0	1	0	0
4	0	1	1	0	0
5	0	1	1	1	1
6	0	1	0	1	1
7	0	1	0	0	1
8	1	1	0	0	1
9	1	1	0	1	1
10	1	1	1	1	1
11	1	1	1	0	1
12	1	0	1	0	1
13	1	0	1	1	0
14	1	0	0	1	0
15	1	0	0	0	0

(a) Truth table



$f_{min} = B\bar{C} + BD + AC\bar{D}$

(b) K-map



(c) NAND logic

Figure 7.38 Truth table, K-map and logic diagram for the SOP circuit.

7.16.7 Design of an SOP Circuit to Detect the Decimal Numbers 0, 2, 4, 6, and 8 in a 4-bit 5211 BCD Code Input

The input to the SOP circuit is a 5211 BCD code. Let it be ABCD. It is a 4-bit input. Therefore, there are 16 possible combinations of inputs, out of which only the 10 combinations shown in the truth table of Figure 7.39a are used to code the decimal digits in 5211 code. The remaining 6 combinations 0010, 0100, 0110, 1001, 1011, and 1101 are invalid. So, the corresponding outputs are don't cares (i.e. minterms 2, 4, 6, 9, 11, and 13 are don't cares). Looking at the truth table of the SOP circuit shown in Figure 7.39a we observe that the output is 1 for the input combinations corresponding to minterms 0, 3, 7, 10, 14 (i.e. corresponding to 5211 code of decimal numbers 0, 2, 4, 6, and 8).

So the problem may be stated as

$$F = \Sigma\, m(0, 3, 7, 10, 14) + d(2, 4, 6, 9, 11, 13)$$

Figure 7.54

7.18 COMPARATORS

A comparator is a logic circuit used to compare the magnitudes of two binary numbers. Dependi
on the design, it may either simply provide an output that is active (goes HIGH for example) wh
the two numbers are equal, or additionally provide outputs that signify which of the numbers
greater when equality does not hold.

The X-NOR gate (coincidence gate) is a basic comparator, because its output is a 1 only if i
two input bits are equal, i.e. the output is a 1 if and only if the input bits coincide.

Two binary numbers are equal, if and only if all their corresponding bits coincide. For example, two 4-bit binary numbers, $A_3A_2A_1A_0$ and $B_3B_2B_1B_0$ are equal, if and only if, $A_3 = B_3$, $A_2 = B_2$, $A_1 = B_1$ and $A_0 = B_0$. Thus, equality holds when A_3 coincides with B_3, A_2 coincides with B_2, A_1 coincides with B_1, and A_0 coincides with B_0. The implementation of this logic,

$$\text{EQUALITY} = (A_3 \odot B_3)(A_2 \odot B_2)(A_1 \odot B_1)(A_0 \odot B_0)$$

is straightforward. It is obvious that this circuit can be expanded or compressed to accommodate binary numbers with any other number of bits.

The block diagram of a 1-bit comparator which can be used as a module for comparison of larger numbers is shown in Figure 7.55.

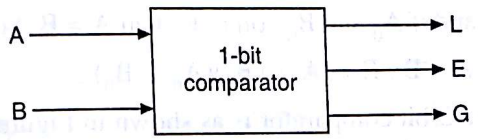


Figure 7.55 Block diagram of a 1-bit comparator.

7.18.1 1-bit Magnitude Comparator

The logic for a 1-bit magnitude comparator: Let the 1-bit numbers be $A = A_0$ and $B = B_0$.
If $A_0 = 1$ and $B_0 = 0$, then $A > B$.
Therefore,

$$A > B: G = A_0\bar{B}_0$$

If $A_0 = 0$ and $B_0 = 1$, then $A < B$.
Therefore,

$$A < B: L = \bar{A}_0B_0$$

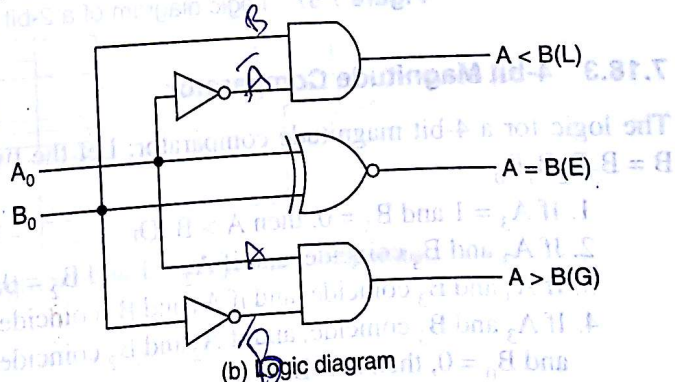
If A_0 and B_0 coincide, i.e. $A_0 = B_0 = 0$ or if $A_0 = B_0 = 1$, then $A = B$.
Therefore,

$$A = B : E = A_0 \odot B_0$$

The truth table and the logic diagram for the 1-bit comparator are shown in Figure 7.56. The logic expressions for G, L, and E can also be obtained from the truth table.

A_0	B_0	L	E	G
0	0	0	1	0
0	1	1	0	0
1	0	0	0	1
1	1	0	1	0

(a) Truth table



(b) Logic diagram

Figure 7.56 1-bit comparator.

7.18.2 2-bit Magnitude Comparator

The logic for a 2-bit magnitude comparator: Let the two 2-bit numbers be $A = A_1 A_0$ and $B = B_1 B_0$.

1. If $A_1 = 1$ and $B_1 = 0$, then $A > B$ or
2. If A_1 and B_1 coincide and $A_0 = 1$ and $B_0 = 0$, then $A > B$. So the logic expression for $A > B$ is

$$A > B : G = A_1\bar{B}_1 + (A_1 \odot B_1)A_0\bar{B}_0$$

1. If $A_1 = 0$ and $B_1 = 1$, then $A < B$ or
2. If A_1 and B_1 coincide and $A_0 = 0$ and $B_0 = 1$, then $A < B$. So the expression for $A < B$ is

$$A < B : L = \bar{A}_1 B_1 + (A_1 \odot B_1)\bar{A}_0 B_0$$

If A_1 and B_1 coincide and if A_0 and B_0 coincide then $A = B$. So the expression for $A = B$ is

$$A = B : E = (A_1 \odot B_1)(A_0 \odot B_0)$$

The logic diagram for a 2-bit comparator is as shown in Figure 7.57.

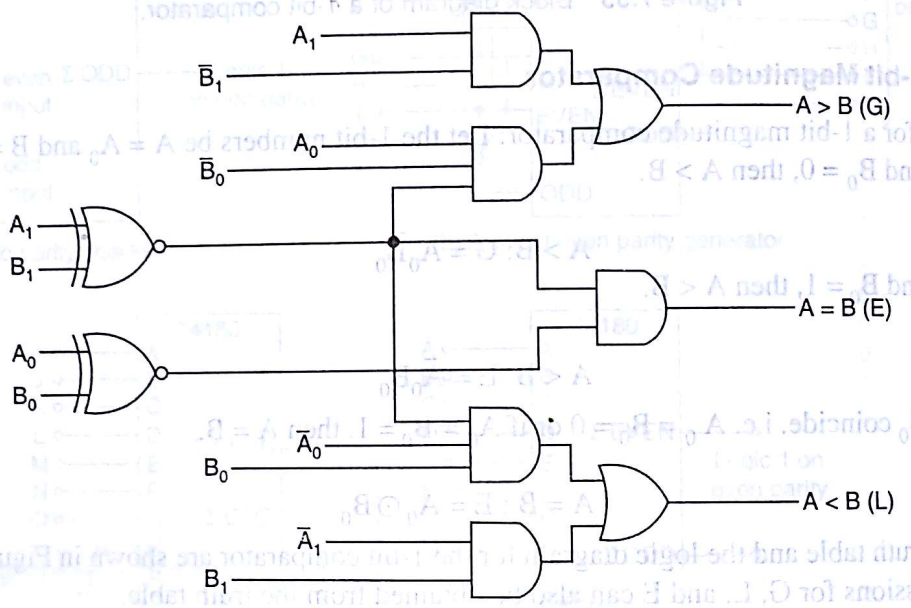


Figure 7.57 Logic diagram of a 2-bit magnitude comparator.

7.18.3 4-bit Magnitude Comparator

The logic for a 4-bit magnitude comparator: Let the two 4-bit numbers be $A = A_3A_2A_1A_0$ and $B = B_3B_2B_1B_0$.

1. If $A_3 = 1$ and $B_3 = 0$, then $A > B$. Or
2. If A_3 and B_3 coincide, and if $A_2 = 1$ and $B_2 = 0$, then $A > B$. Or
3. If A_3 and B_3 coincide, and if A_2 and B_2 coincide, and if $A_1 = 1$ and $B_1 = 0$, then $A > B$. Or
4. If A_3 and B_3 coincide, and if A_2 and B_2 coincide, and if A_1 and B_1 coincide, and if $A_0 = 1$ and $B_0 = 0$, then $A > B$.

From these statements, we see that the logic expression for $A > B$ can be written as

$$(A > B) : G = A_3\bar{B}_3 + (A_3 \odot B_3)A_2\bar{B}_2 + (A_3 \odot B_3)(A_2 \odot B_2)A_1\bar{B}_1 + (A_3 \odot B_3)(A_2 \odot B_2)(A_1 \odot B_1)A_0\bar{B}_0$$

Similarly, the logic expression for A < B can be written as

$$(A < B): L = \bar{A}_3B_3 + (A_3 \odot B_3)\bar{A}_2B_2 + (A_3 \odot B_3)(A_2 \odot B_2)\bar{A}_1B_1 + (A_3 \odot B_3)(A_2 \odot B_2)(A_1 \odot B_1)\bar{A}_0B_0$$

If A_3 and B_3 coincide and if A_2 and B_2 coincide and if A_1 and B_1 coincide and if A_0 and B_0 coincide, then A = B.

So the expression for A = B can be written as

$$(A = B): E = (A_3 \odot B_3)(A_2 \odot B_2)(A_1 \odot B_1)(A_0 \odot B_0)$$

Figure 7.58 shows the logic diagram of a comparator that implements the logic we have described. Note that, it provides three active-HIGH outputs: A > B, A < B, and A = B.

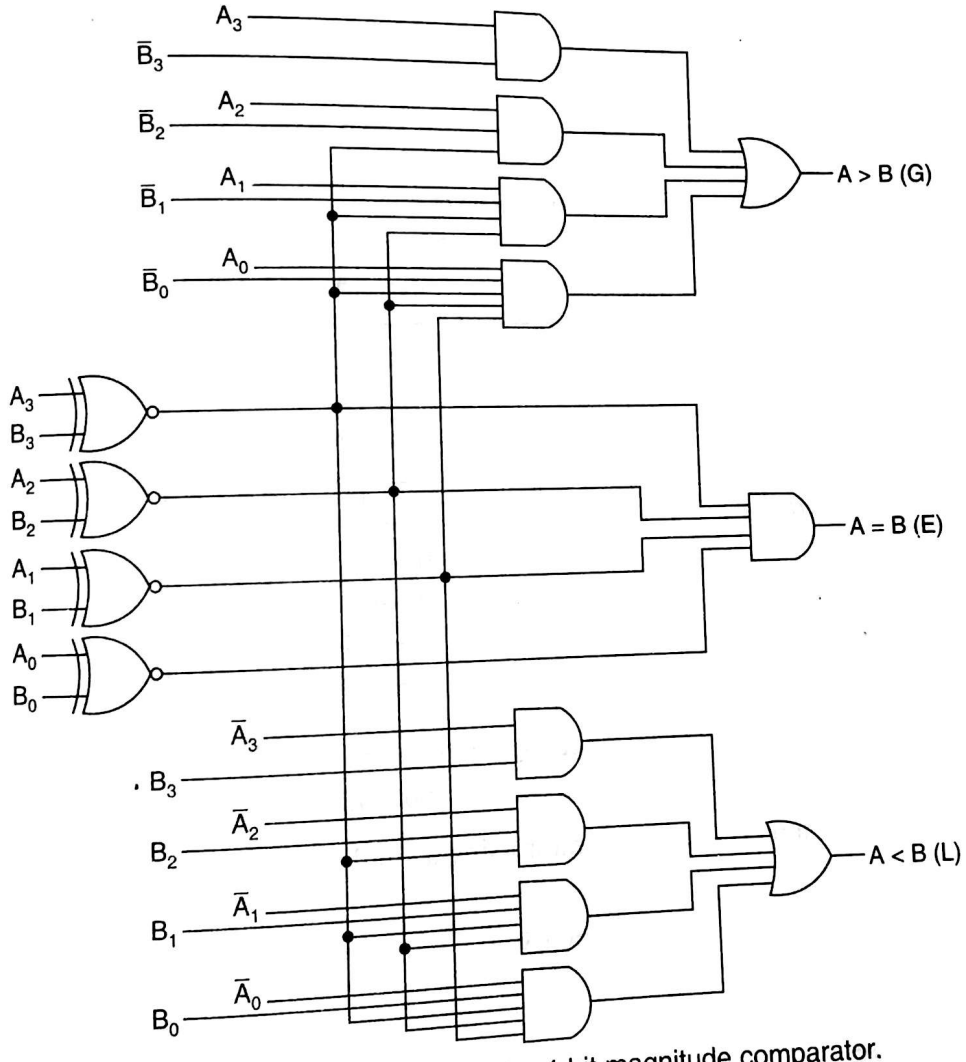


Figure 7.58 Logic diagram of a 4-bit magnitude comparator.

7.19 IC COMPARATOR

Figure 7.59a shows the pin diagram of IC 7485, a 4-bit comparator. Pins labelled $(A < B)_{IN}$, $(A = B)_{IN}$, and $(A > B)_{IN}$ are used for cascading. Figure 7.59b shows how two 4-bit comparators

Solution

The circuit is shown in Figure 7.60. The two 5-bit numbers to be compared are $X_4X_3X_2X_1$ and $Y_4Y_3Y_2Y_1Y_0$.

7.20 ENCODERS

An encoder is a device whose inputs are decimal digits and/or alphabetic characters and w outputs are the coded representation of those inputs, i.e. an encoder is a device which con familiar numbers or symbols into coded format. In other words, an encoder may be said to combinational logic circuit that performs the 'reverse' operation of the decoder. The opposi the decoding process is called encoding, i.e. encoding is a process of converting familiar num

or symbols into a coded format. An encoder has a number of input lines, only one of which is activated at a given time, and produces an N-bit output code depending on which input is activated. Figure 7.61 shows the block diagram of an encoder with M inputs and N outputs. Here the inputs are active HIGH, which means they are normally LOW.

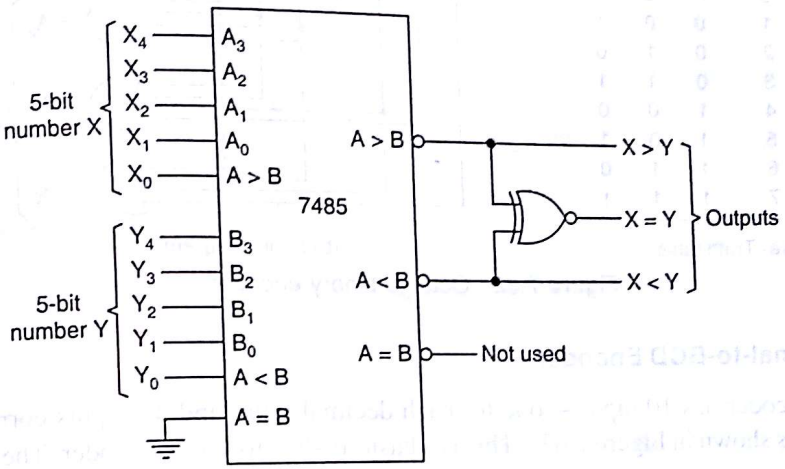


Figure 7.60 Example 7.9: Use of 7485 as a 5-bit comparator.

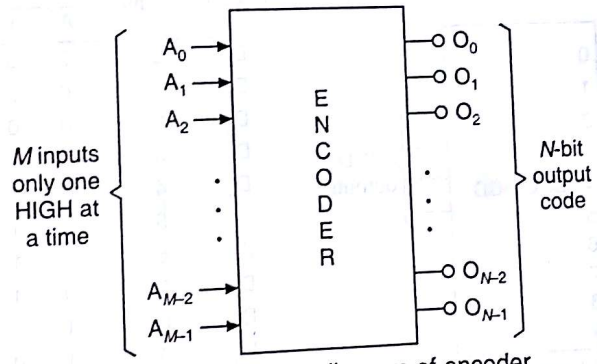


Figure 7.61 Block diagram of encoder.

7.20.1 Octal-to-Binary Encoder

An octal-to-binary encoder (8-line to 3-line encoder) accepts 8 input lines and produces a 3-bit output code corresponding to the activated input. Figure 7.62 shows the truth table and the logic circuit for an octal-to-binary encoder with active HIGH inputs.

From the truth table, we see that A_2 is a 1 if any of the digits D_4 or D_5 or D_6 or D_7 is a 1. Therefore,

$$A_2 = D_4 + D_5 + D_6 + D_7$$

Similarly,

$$A_1 = D_2 + D_3 + D_6 + D_7$$

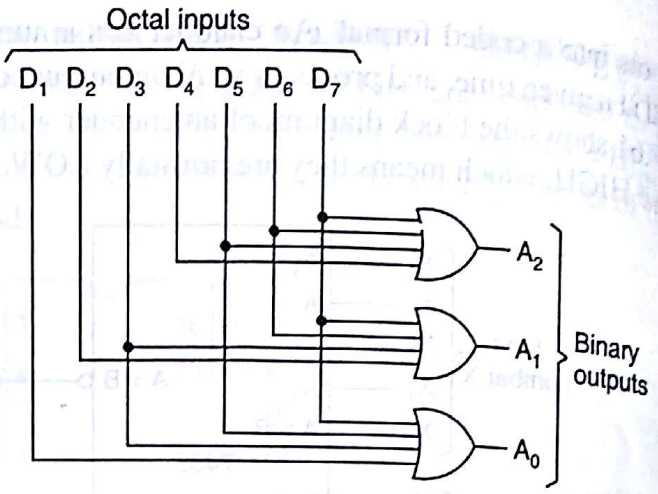
and

$$A_0 = D_1 + D_3 + D_5 + D_7$$

We see that D_0 is not present in any of the expressions. So, D_0 is a don't care.

Octal digits		Binary		
		A_2	A_1	A_0
D_0	0	0	0	0
D_1	1	0	0	1
D_2	2	0	1	0
D_3	3	0	1	1
D_4	4	1	0	0
D_5	5	1	0	1
D_6	6	1	1	0
D_7	7	1	1	1

(a) Truth table

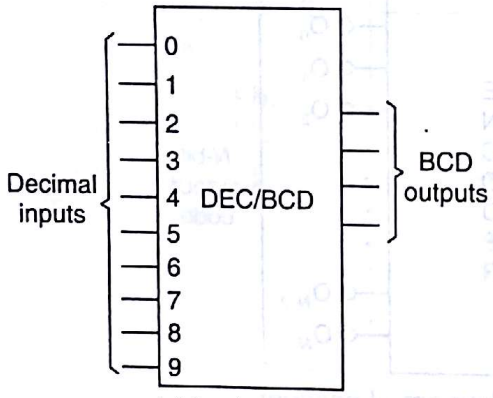


(b) Logic diagram

Figure 7.62 Octal-to-binary encoder.

7.20.2 Decimal-to-BCD Encoder

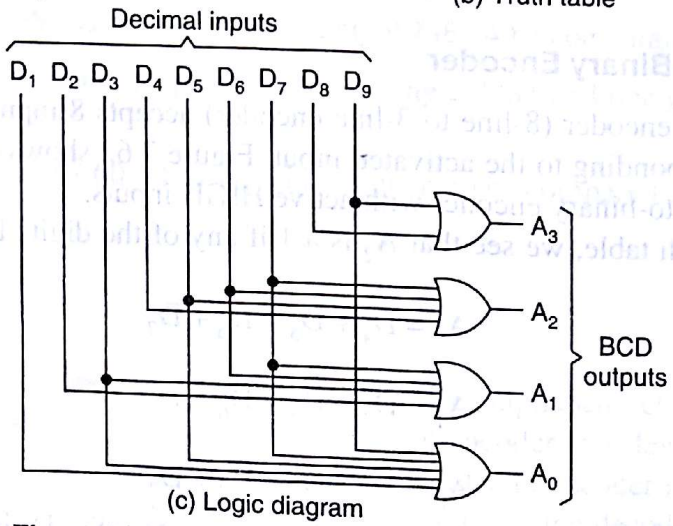
This type of encoder has 10 inputs—one for each decimal digit, and 4 outputs corresponding to the BCD code as shown in Figure 7.63a. This is a basic 10-line to 4-line encoder. The BCD code is



(a) Logic symbol

Decimal inputs		Binary			
		A_3	A_2	A_1	A_0
D_0	0	0	0	0	0
D_1	1	0	0	0	1
D_2	2	0	0	1	0
D_3	3	0	0	1	1
D_4	4	0	1	0	0
D_5	5	0	1	0	1
D_6	6	0	1	1	0
D_7	7	0	1	1	1
D_8	8	1	0	0	0
D_9	9	1	0	0	1

(b) Truth table



(c) Logic diagram

Figure 7.63 Decimal-to-BCD encoder.

or resets, as necessary to produce the 4-bit code corresponding to the decimal digit. For example, when the key 7 is pressed, the diodes connected to the S inputs of Q_4, Q_2, and Q_1 are forward biased, as is that connected to the R input of Q_8. Thus, the output is 0111. Note that, the diode configuration at each S and R input is essentially a diode OR gate. Diode matrix encoders are found on printed circuit boards of many devices having a keyboard as the means of data entry.

7.22 PRIORITY ENCODERS

The encoders discussed so far will operate correctly, provided that one and only one decimal input is HIGH at any given time. In some practical systems, two or more decimal inputs may inadvertently become HIGH at the same time. For example, a person operating a keyboard might press a second key before releasing the first. Let us say he presses key 3 before releasing key 4. In such a case the output will be 7_{10} (0111) instead of being 4_{10} or 3_{10}.

A priority encoder is a logic circuit that responds to just one input in accordance with some priority system, among all those that may be simultaneously HIGH. The most common priority system is based on the relative magnitudes of the inputs; whichever decimal input is the largest, is the one that is encoded. Thus, in the above example, a priority encoder would encode decimal 4 if both 3 and 4 are simultaneously HIGH.

In some practical applications, priority encoders may have several inputs that are routinely HIGH at the same time, and the principal function of the encoder in those cases is to select the input with the highest priority. This function is called *arbitration*. A common example is found in computer systems, where there are numerous input devices and several of which may attempt to supply data to the computer at the same time. A priority encoder is used to enable that input device which has the highest priority among those competing for access to the computer at the same time.

7.22.1 4-Input Priority Encoder

The truth table of a 4-input priority encoder is given in Figure 7.65a. In addition to the outputs A and B, the circuit has a third output designated by V. This is a valid bit indicator that is set to 1 when one or more inputs are equal to 1. If all inputs are 0, there is no valid input and V is equal to 0. The other two outputs are not inspected when V equals 0 and are specified as don't care conditions. According to the truth table, the higher the subscript number, the higher the priority of the input. Input D_3 has the highest priority. So regardless of the values of other inputs, when this input is 1, the output for AB is 11 (binary 3). D_2 has the next priority level. The output is 10 if $D_2 = 1$ provided that $D_3 = 0$ regardless 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 levels.

From the truth table we observe that

$$A = D_3 + \bar{D}_3 D_2 = D_3 + D_2$$

and

$$B = D_3 + \bar{D}_3 \bar{D}_2 D_1 = D_3 + \bar{D}_2 D_1$$

$$V = D_3 + D_2 + D_1 + D_0$$

The condition for output V is an OR function of all the input variables.

The maps for simplifying outputs A and B and the corresponding logic diagram are shown in Figures 7.65b, c and d. Although the table has only five rows, when each × in a row is first replaced

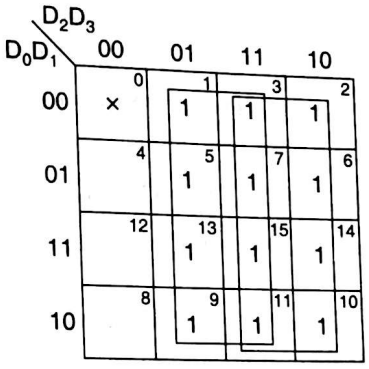
by 0 and then by 1, we obtain all 16 possible input combinations. The minterms for the two functions A and B are derived from the table as

$$A = \Sigma\, m(1, 2, 3, 5, 6, 7, 9, 10, 11, 13, 14, 15)$$
$$B = \Sigma\, m(1, 3, 4, 5, 7, 9, 11, 12, 13, 15)$$

The same values of A and B mentioned above are obtained from the K-map.

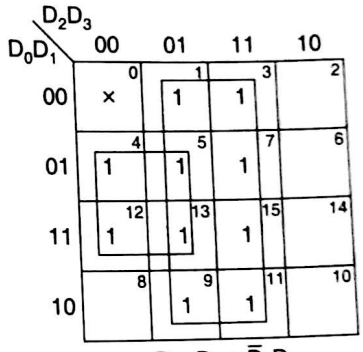
Inputs				Outputs		
D_0	D_1	D_2	D_3	A	B	V
0	0	0	0	×	×	0
1	0	0	0	0	0	1
×	1	0	0	0	1	1
×	×	1	0	1	0	1
×	×	×	1	1	1	1

(a) Truth table



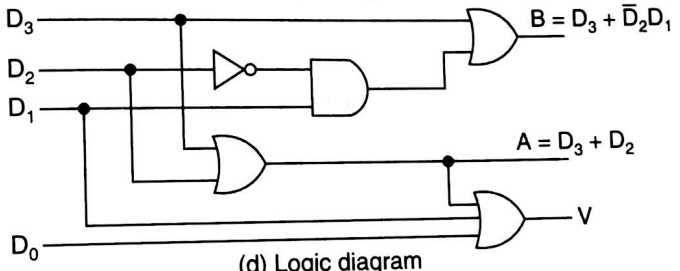
$A = D_3 + D_2$

(b) K-map for A



$B = D_3 + \bar{D}_2D_1$

(c) K-map for B



(d) Logic diagram

Figure 7.65 4-bit priority encoder.

7.22.2 Decimal-to-BCD Priority Encoder

This type of encoder performs the same basic function of encoding the decimal digits into 4-bit BCD outputs, as that performed by a normal decimal-to-BCD encoder. It, however, offers the additional facility of providing priority. That is, it produces a BCD output corresponding to the highest order decimal digit appearing on the inputs and ignores all others.

Now, let us look at the requirements for the priority detection logic. The purpose of this logic circuitry is to prevent a lower-order digit input from disrupting the encoding of a higher-order digit. This is accomplished by using inhibit gates. Referring to the truth table and logic diagram of the decimal-to-BCD encoder of Figure 7.63, note that A_0 is HIGH when D_1, D_3, D_5, D_7, or D_9 is HIGH.

In the priority encoder, input digit 1 will be allowed to activate the A_0 output, only if no higher order digits other than those that also activate A_0 are HIGH, i.e. D_2, D_4, D_6, and D_8 must be LOW. If any of those are HIGH, then A_0 will be LOW. This can be stated as follows:

A_0 is HIGH if:

D_1 is HIGH and D_2, D_4, D_6, and D_8 are LOW. Or
D_3 is HIGH and D_4, D_6, and D_8 are LOW. Or
D_5 is HIGH and D_6 and D_8 are LOW. Or
D_7 is HIGH and D_8 is LOW. Or
D_9 is HIGH.

Thus,

$$A_0 = D_1\bar{D}_2\bar{D}_4\bar{D}_6\bar{D}_8 + D_3\bar{D}_4\bar{D}_6\bar{D}_8 + D_5\bar{D}_6\bar{D}_8 + D_7\bar{D}_8 + D_9$$

Similarly, A_1 is HIGH when D_2, D_3, D_6, or D_7 is HIGH. So, in the priority encoder A_1 will be HIGH if:

D_2 is HIGH and D_4, D_5, D_8, and D_9 are LOW. Or
D_3 is HIGH and D_4, D_5, D_8, and D_9 are LOW. Or
D_6 is HIGH and D_8 and D_9 are LOW. Or
D_7 is HIGH and D_8 and D_9 are LOW.

Thus,

$$A_1 = D_2\bar{D}_4\bar{D}_5\bar{D}_8\bar{D}_9 + D_3\bar{D}_4\bar{D}_5\bar{D}_8\bar{D}_9 + D_6\bar{D}_8\bar{D}_9 + D_7\bar{D}_8\bar{D}_9$$

Also, A_2 is HIGH when D_4, D_5, D_6, or D_7 is HIGH. So, in the priority encoder A_2 will become HIGH if:

D_4 is HIGH and D_8 and D_9 are LOW. Or
D_5 is HIGH and D_8 and D_9 are LOW. Or
D_6 is HIGH and D_8 and D_9 are LOW. Or
D_7 is HIGH and D_8 and D_9 are LOW.

Thus,

$$A_2 = D_4\bar{D}_8\bar{D}_9 + D_5\bar{D}_8\bar{D}_9 + D_6\bar{D}_8\bar{D}_9 + D_7\bar{D}_8\bar{D}_9$$

Finally, A_3 is HIGH if D_8 is HIGH or if D_9 is HIGH. So, in the priority encoder A_3 will be HIGH if D_8 is HIGH OR D_9 is HIGH.

Thus,

$$A_3 = D_8 + D_9$$

The truth table operation of the priority encoder is shown in Table 7.2. The inputs and outputs are active low. The truth table clearly shows that the magnitudes of the decimal inputs determine their priorities. If any decimal input is active, it is encoded provided all higher value inputs are inactive regardless of the states of all lower value inputs.

Table 7.2 Truth table of a decimal-to-BCD priority encoder

Decimal inputs (× = Don't care)									BCD outputs			
D_1	D_2	D_3	D_4	D_5	D_6	D_7	D_8	D_9	A_3	A_2	A_1	A_0
1	1	1	1	1	1	1	1	1	1	1	1	1
×	×	×	×	×	×	×	×	0	0	1	1	0
×	×	×	×	×	×	×	0	1	0	1	1	1
×	×	×	×	×	×	0	1	1	1	0	0	0
×	×	×	×	×	0	1	1	1	1	0	0	1
×	×	×	×	0	1	1	1	1	1	0	1	0
×	×	×	0	1	1	1	1	1	1	0	1	1
×	×	0	1	1	1	1	1	1	1	1	0	0
×	0	1	1	1	1	1	1	1	1	1	0	1
0	1	1	1	1	1	1	1	1	1	1	1	0

7.22.3 Octal-to-Binary Priority Encoder

The octal code is often used at the inputs of digital circuits that require manual entering of long binary words. Priority encoder 74148 IC has been designed to achieve this operation. Its pin diagram is given in Figure 7.66. This circuit has active LOW inputs and active LOW outputs. The enable input and the Gray outputs which are also active LOW are used to cascade circuits to handle more inputs. A hexadecimal-to-binary encoder which is also a very useful circuit because of the widespread use of the hexadecimal code in computers, microprocessors, etc. can be designed using this facility.

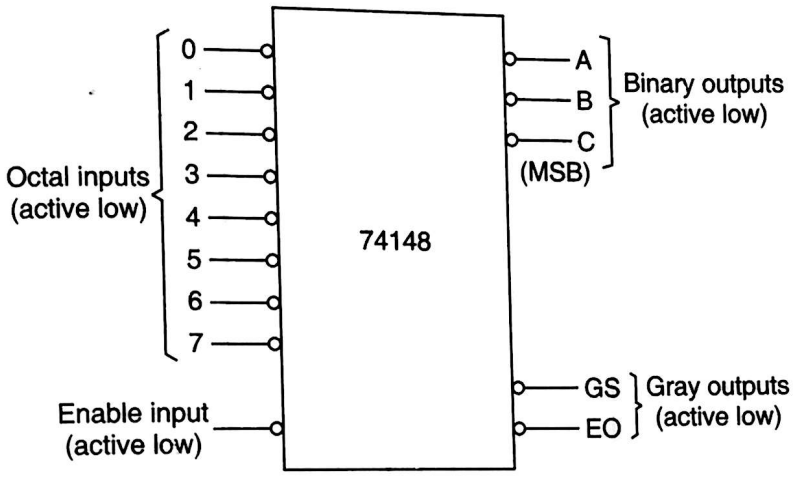


Figure 7.66 Pin diagram of an octal-to-binary priority encoder (IC 74148).

EXAMPLE 7.10 Design a hexadecimal-to-binary encoder using 74148 encoders and 74157 multiplexer.

Solution

Since there are 16 symbols (0–F) in the hexadecimal number system, two 74148 encoders are required. Hexadecimal inputs 0–7 are applied to IC1 input lines and hexadecimal inputs 8–F to IC2 input lines. Whenever one of the inputs of IC2 is active (LOW), IC1 must be disabled. On the other hand, if all the inputs of IC2 are HIGH, then IC1 must be enabled. This is achieved by connecting the EO line of IC2 to the EI line of IC1. A quad 2:1 multiplexer is required to get the proper 4-bit binary outputs. The complete circuit is shown in Figure 7.67. The GS output of 74148 goes LOW whenever one of its inputs is active. Therefore, the GS of IC2 is connected to the SELECT input of 74157. The 74157 selects its A inputs if the SELECT input is LOW, otherwise B inputs are selected. The outputs of the multiplexer are the required binary outputs and are active LOW. This circuit is also a priority encoder.

7.23 DECODERS

A decoder is a logic circuit that converts an N-bit binary input code into M output lines such that only one output line is activated for each one of the possible combinations of inputs. In other words, we can say that a decoder identifies or recognizes or detects a particular code. Figure 7.68 shows the general decoder diagram with N inputs and M outputs. Since each of the N inputs can be a 0 or a 1, there are 2^N possible input combinations or codes. For each of these input combinations,

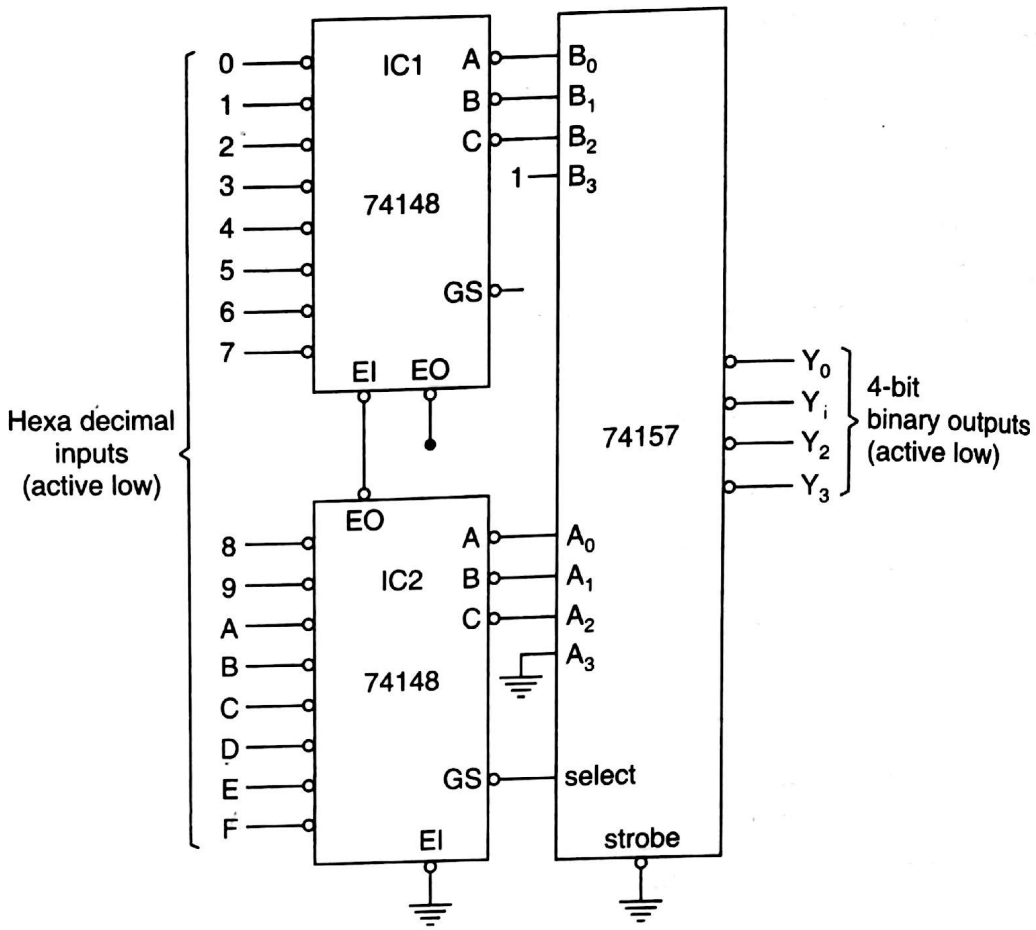


Figure 7.67 Hexadecimal-to-binary encoder using 74148s and 74157.

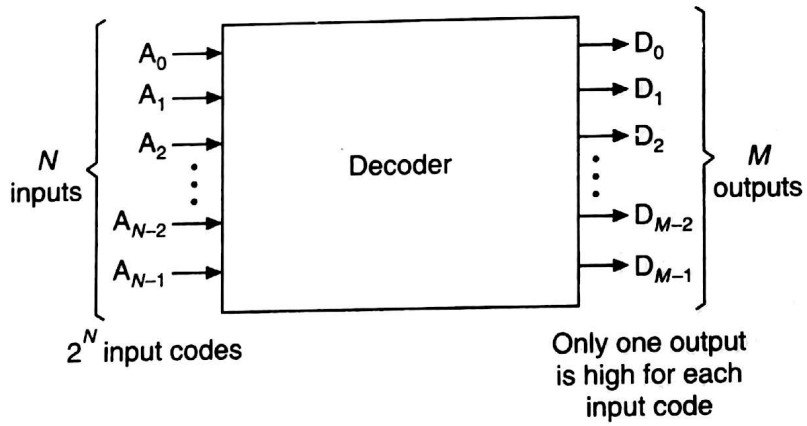


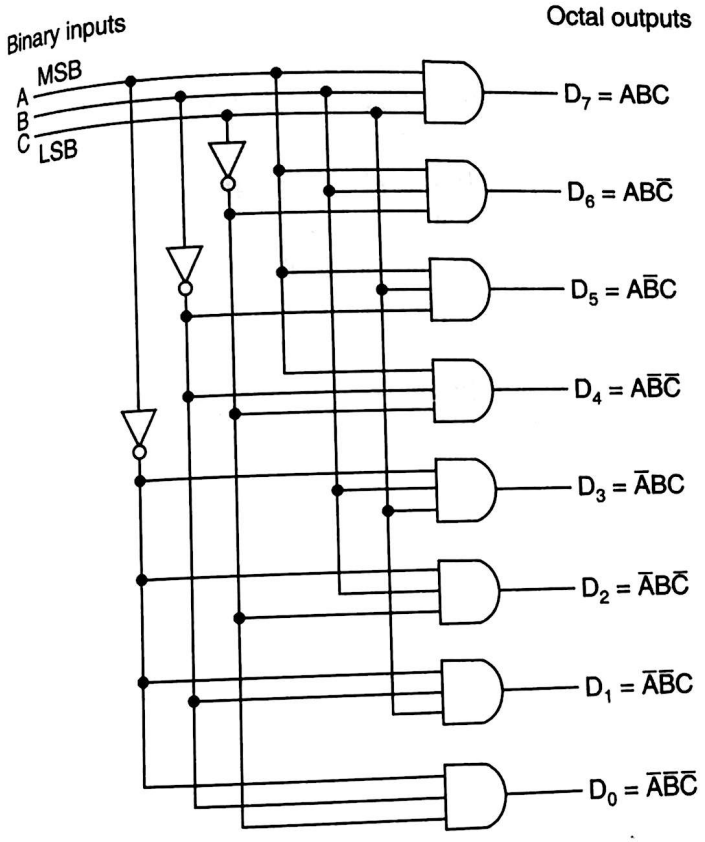
Figure 7.68 General block diagram of a decoder.

only one of the M outputs will be active (HIGH), all the other outputs will remain inactive (LOW). Some decoders are designed to produce active LOW output, while all the other outputs remain HIGH.

Some decoders do not utilize all of the 2^N possible input codes. For example, a BCD to decimal decoder has a 4-bit input code and 10 output lines that correspond to the 10 BCD code groups 0000 through 1001. Decoders of this type are often designed so that if any of the unused codes are applied to the input, none of the outputs will be activated.

7.23.1 3-Line-to-8-Line Decoder

Figure 7.69a shows the circuitry for a decoder with three inputs and eight outputs. It uses all AND gates, and therefore, the outputs are active-HIGH. For active-LOW outputs, NAND gates are used. The truth table of the decoder is shown in Figure 7.69b. This decoder can be referred to in several ways. It can be called a 3-line to 8-line decoder because it has three input lines and eight output lines. It is also called a binary-to-octal decoder because it takes a 3-bit binary input code and activates one of the eight (octal) outputs corresponding to that code. It is also referred to as a *1-of-8 decoder* because only one of the eight outputs is activated at one time.



(a) Logic diagram

Inputs			Outputs							
A	B	C	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

(b) Truth table

Figure 7.69 3-line to 8-line decoder.

7.23.2 Enable Inputs

Some decoders have one or more ENABLE inputs that are used to control the operation of the decoder. For example, in the 3-line to 8-line decoder, if a common ENABLE line is connected to the fourth input of each gate, a particular output as determined by the A, B, C input code will go HIGH only when the ENABLE line is held HIGH. When the ENABLE is held LOW, however, all the outputs will be forced to the LOW state regardless of the levels at the A, B, and C inputs. Thus, the decoder is enabled only when the ENABLE is HIGH. IC74 LS138 is a 3-line to 8-line decoder.

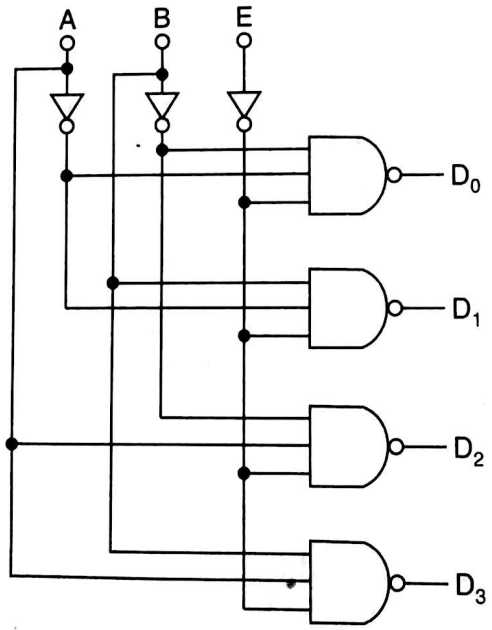
7.23.3 BCD-to-Decimal Decoder (7442)

The BCD to-decimal Decoder is also called a 4-line to 10-line or 4-to-10 decoder or 1-of-10 decoder. It has 4-input lines (for A_3, A_2, A_1, A_0) and 10 output lines (for D_0, D_1, D_2, D_3, D_4, D_5,

D_6, D_7, D_8, D_9). Only one output line is active at time. 6 of the 16 input combinations are invalid and for input combinations that are invalid for BCD none of the outputs will be activated. The inputs and outputs can be active high or active low. IC 7442 is a BCD to decimal decoder with active low inputs and outputs. The TTL 7445 IC is a BCD-to-decimal decoder/driver. The term driver is added to its description because this IC has open collector outputs that can operate at higher current and voltage limits than a normal TTL output. It makes 7445 suitable for directly driving loads such as indicator LEDs or lamps, relays or DC motors.

7.23.4 2-Line-to-4-Line Decoder with NAND Gates

Some decoders are constructed with NAND gates. Since a NAND gate produces the AND operation with an inverted output, it becomes more economical to generate the decoder minterms in their complemented form. A 2-to-4 line decoder with an enable input is constructed with NAND gates as shown in Figure 7.70.The circuit operates with complemented outputs and complemented enable input. The decoder is enabled when E = 0. The inputs are active high and outputs active low. As indicated by the truth table, only one output can be equal to 0 at any given time, all other outputs are equal to 1. The output whose value is equal to 0 represents the minterm selected by inputs A and B. The circuit is disabled when E = 1, regardless of the values of the other two inputs. When the circuit is disabled, none of the outputs are equal to 0 and none of the minterms are selected. In general a decoder may operate with complemented or un complemented outputs. The enable input may be activated with a 0 or a 1 signal. Some decoders have two or more enable inputs that must satisfy a given logic condition in order to enable the circuits.



(a) Logic diagram

E	A	B	D_0	D_1	D_2	D_3
1	×	×	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	0

(b) Truth table

Figure 7.70 2 line-to-4 line decoder with NAND gates.

A decoder with enable input can function as a demultiplexer. A demultiplexer is a circuit that receives information from a single line and directs it to one of the 2^n possible output lines. The selection of a specific output is controlled by the bit combination of n selection lines.

The decoder of Figure 7.70 can function as a 1-to-4 line demultiplexer when E is taken as a data input line and A and B are taken as the selection inputs. The single input variable E has a path

to all 4 outputs, but the input information is directed to only one of the output lines, as specified by the binary combination of two selection lines A and B. This can be verified from the truth table of the circuit. For example, if the selection lines AB = 10, the output D_2 will be same as the input value, while all other outputs are maintained at a 1. Because decoder and demultiplexer operations are obtained from the same circuit, a decoder with an enable input is referred to as a decoder/demultiplexer.

7.23.5 Combinational Logic Implementation

A decoder provides 2^n minterms of n input variables. Since any Boolean function can be expressed in sum of minterms, one can use a decoder to generate the minterms and an external OR gate to form the logic sum. In this way any combinational circuit with n inputs and m outputs can be implemented with an n-to-2^n decoder and m OR gates.

The procedure for implementing a combinational circuit by means of a decoder and OR gates requires that the Boolean function for the circuit is expressed in sum of minterms. A decoder that generates all the minterms of the input variables is then chosen. The inputs to each OR gate are selected from the decoder outputs according to the list of minterms of each function. This procedure will be illustrated by an example that implements a full adder.

From the truth table of the full adder, we obtain the functions for the combinational circuit in sum of minterms.

$$S = \bar{A}\bar{B}C_{in} + \bar{A}B\bar{C}_{in} + A\bar{B}\bar{C}_{in} + ABC_{in} = A \oplus B \oplus C_{in} = \Sigma\, m(1, 2, 4, 7)$$

and $$C_{out} = \bar{A}BC_{in} + A\bar{B}C_{in} + AB\bar{C}_{in} + ABC_{in} = AB + (A \oplus B)C_{in} = \Sigma\, m(3, 5, 6, 7)$$

Since there are 3 inputs and a total of 8 minterms, we need a 3-to-8 line decoder. The implementation is shown in Figure 7.71. The decoder generates the 8 minterms for A, B, C_{in}. The OR gate for output S forms the logical sum of minterms 1, 2, 4 and 7. The OR gate for C_{out} forms the logical sum of the minterms 4, 5, 6 and 7.

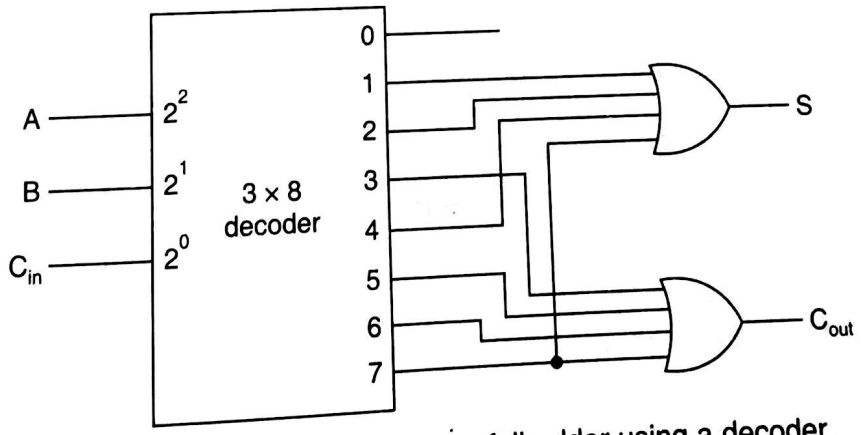


Figure 7.71 Logic diagram of a full adder using a decoder.

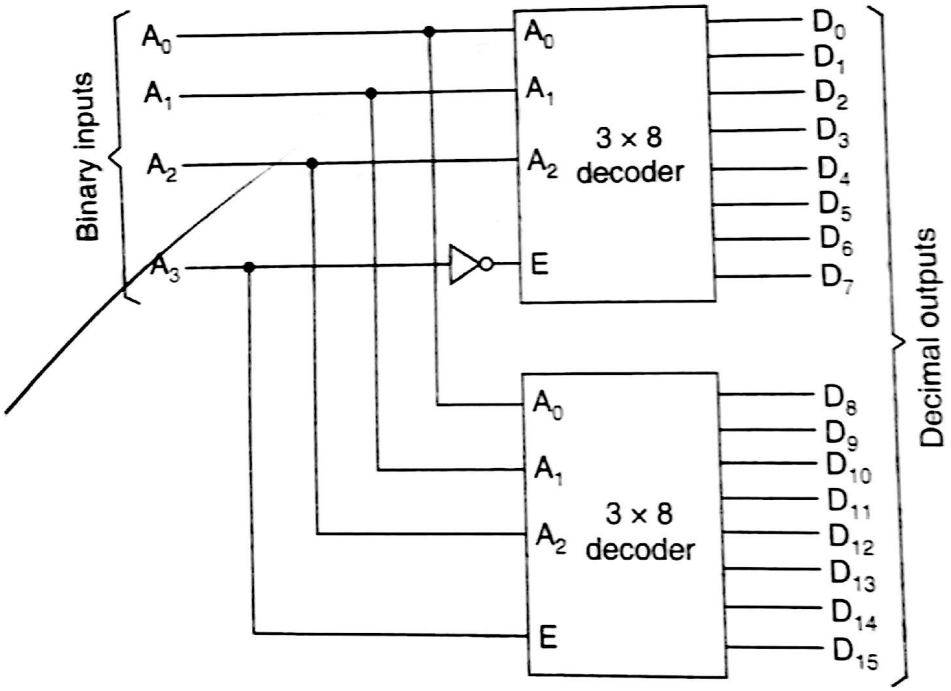
A function with a long list of minterms requires an OR gate with a large number of inputs. A function having a list of K minterms can be expressed in its complemented form $\bar{F}$ with 2^n–K terms. If the number of minterms in a function is greater than $2^n/2$, then $\bar{F}$ can be expressed with fewer minterms. In such a case, it is advantageous to use a NOR gate to sum the minterms of $\bar{F}$.

The output of the NOR gate complements this sum and generates the normal output F. If NAND gates are used for the decoder as in Figure 7.68 then the external gates must be NAND gates instead of OR gates. This is because a two-level NAND gate circuit implements a sum of minterms function and is equivalent to a 2-level AND-OR circuit.

7.23.6 4-to-16 Decoder from Two 3-to-8 Decoders

Decoders with enable inputs can be connected together to form a larger decoder circuit. Figure 7.72 shows the arrangement for using two 74138s, 3-to-8 decoders, to obtain a 4-to-16 decoder. The most significant input bit A_3 is connected through an inverter to $\overline{E}$ on the upper decoder (for D_0 through D_7) and directly to E on the lower decoder (for D_8 through D_{15}). Thus, when A_3 is LOW the upper decoder is enabled and the lower decoder is disabled. The bottom decoder outputs all 0s and top 8 outputs generate minterms. When A_3 is HIGH, the lower decoder is enabled and the upper decoder is disabled. The bottom decoder outputs generate minterms 1000 to 1111 while the outputs of the top decoder are all 0s.

This example demonstrates the usefulness of enable inputs in decoders and other combinational logic components. In general, enable inputs are a convenient feature for interconnecting two or more standard components for the purpose of expanding the component into a similar function with more inputs and outputs.



(a) Logic diagram

	Binary inputs				Decimal output (active low)
	A_3	A_2	A_1	A_0	
Upper decoder	0	0	0	0	D_0
	0	0	0	1	D_1
	0	0	1	0	D_2
	0	0	1	1	D_3
	0	1	0	0	D_4
	0	1	0	1	D_5
	0	1	1	0	D_6
	0	1	1	1	D_7
Lower decoder	1	0	0	0	D_8
	1	0	0	1	D_9
	1	0	1	0	D_{10}
	1	0	1	1	D_{11}
	1	1	0	0	D_{12}
	1	1	0	1	D_{13}
	1	1	1	0	D_{14}
	1	1	1	1	D_{15}

(b) Function table

Figure 7.72 Connecting two 74138 3-to-8 decoders to obtain a 4-to-16 decoder.

7.23.7 Decoder Applications

Decoders are used whenever an output or a group of outputs is to be activated only on the occurrence of a specific combination of input levels. These input levels are often provided by the outputs of a counter or register. When the decoder inputs come from a counter that is being continually pulsed, the decoder outputs will be activated sequentially, and they can be used as timing or sequencing signals to turn devices on or off at specific times.

decode the selection input lines. In general, a 2^n- to-1 line multiplexer resemble a decoder circuit and, indeed, they are applied to a single OR gate. The size of multiplexer is specified by the number 2^n of its data n-to-2^n decoder by adding to it 2^n input lines, one to each AND gate. The outputs of the AND gates lines and the single output line. The n selection lines are implied from the 2^n data lines. As in decoders multiplexers may have an enable input to control the operation of the unit. When the enable input is in the inactive state, the outputs are disabled and when it is in the active state, the circuit functions as a normal multiplexer.

7.24.3 The 16-Input Multiplexer from Two 8-Input Multiplexers

Figure 7.78 shows an arrangement to use two 8-input multiplexers (74151A) to get a 16-input multiplexer. One OR gate and one inverter are also required. The four select inputs S_3, S_2, S_1, and S_0 will select one of the 16 inputs to pass through to X. The S_3 input determines which multiplexer is enabled. When $S_3 = 0$, the left multiplexer is enabled and S_2, S_1, and S_0 inputs determine which of its data inputs will appear at its output and pass through the OR gate to X. When $S_3 = 1$, the right multiplexer is enabled and S_2, S_1, and S_0 inputs select one of its data inputs for passage to output X. This arrangement is also called multiplexer tree. Figure 7.94 shows an arrangement to obtain a 32×1 mux using two 16×1 muxes and one 2×1 mux.

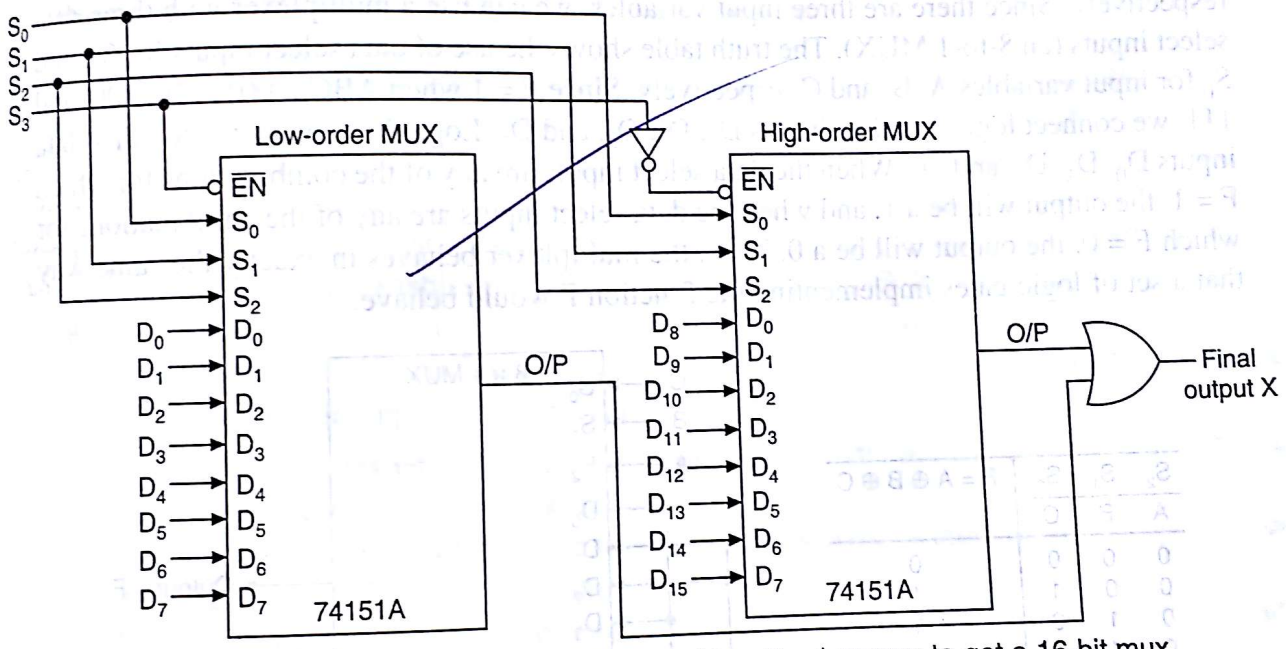


Figure 7.78 Logic diagram for cascading of two 8 × 1 muxes to get a 16-bit mux.

7.25 APPLICATIONS OF MULTIPLEXERS

Multiplexers find numerous and varied applications in digital systems of all types. These applications include data selection, data routing, operation sequencing, parallel-to-serial conversion, waveform generation, and logic function generation.

7.26 DEMULTIPLEXERS (DATA DISTRIBUTORS)

A multiplexer takes several inputs and transmits one of them to the output. A demultiplexer performs the reverse operation; it takes a single input and distributes it over several outputs. So a demultiplexer can be thought of as a 'distributor', since it transmits the same data to different destinations. Thus whereas a multiplexer is an N-to-1 device, a demultiplexer is a 1-to-N (or 2^n) device. Figure 7.88 shows the functional diagram for a demultiplexer (DEMUX). The large arrows for inputs and outputs can represent one or more lines. The 'select' input code determines the output line to which the input data will be transmitted. In other words, the demultiplexer takes one input data source and selectively distributes it to 1-of-N output channels just like a multi-position switch.

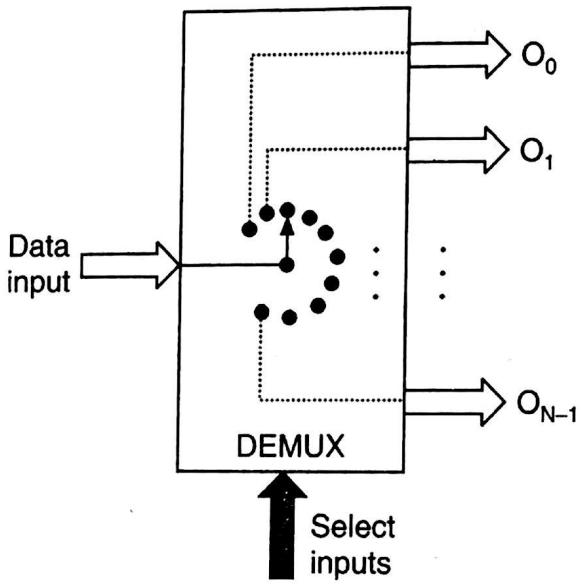


Figure 7.88 Functional diagram of a general demultiplexer.

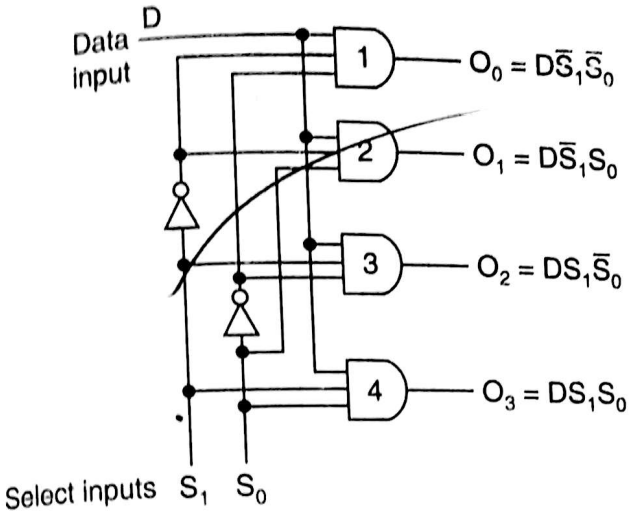
7.26.1 1-Line to 4-Line Demultiplexer

Figure 7.89 shows a 1-line to 4-line demultiplexer circuit. The input data line goes to all of the AND gates. The two select lines S_0 and S_1 enable only one gate at a time, and the data appearing on the input line will pass through the selected gate to the associated output line.

7.26.2 1-Line to 8-Line Demultiplexer

Figure 7.90a shows the logic diagram for a demultiplexer that distributes one input line to eight output lines. The single data input line D is connected to all eight AND gates, but only one of these gates will be enabled by the select input lines. For example, with $S_2\ S_1\ S_0 = 000$, only the AND

gate O_0 will be enabled, and the data input D will appear at output O_0. Other select codes cause input D to reach the other outputs. The truth table in Figure 7.90b summarizes the operation.

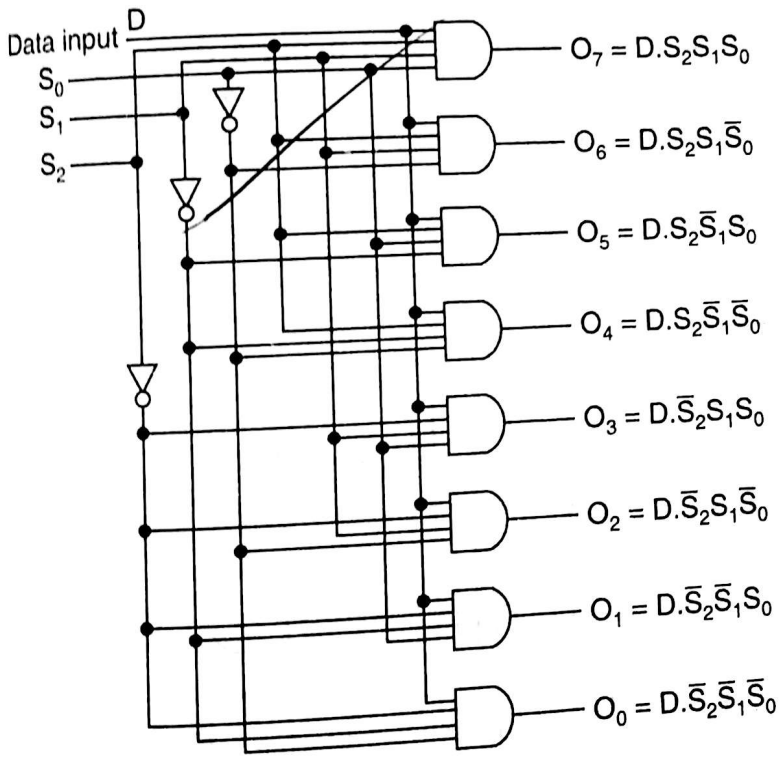


(a) Logic diagram

Select code		Outputs			
S_1	S_0	O_3	O_2	O_1	O_0
0	0	0	0	0	D
0	1	0	0	D	0
1	0	0	D	0	0
1	1	D	0	0	0

(b) Truth table

Figure 7.89 1-line to 4-line demultiplexer.



(a) Logic diagram

Select code			Outputs							
S_2	S_1	S_0	O_7	O_6	O_5	O_4	O_3	O_2	O_1	O_0
0	0	0	0	0	0	0	0	0	0	D
0	0	1	0	0	0	0	0	0	D	0
0	1	0	0	0	0	0	0	D	0	0
0	1	1	0	0	0	0	D	0	0	0
1	0	0	0	0	0	D	0	0	0	0
1	0	1	0	0	D	0	0	0	0	0
1	1	0	0	D	0	0	0	0	0	0
1	1	1	D	0	0	0	0	0	0	0

(b) Truth table

Figure 7.90 1-line to 8-line demultiplexer.

The demultiplexer circuit of Figure 7.90a is very similar to the 3-line to 8-line decoder circuit of Figure 7.69a, except that a fourth input D has been added to each gate. The inputs ABC of Figure 7.69b are here labelled S_2 S_1 S_0 and become the data select inputs.

In the 3-to-8 IC decoder, there are three input lines and eight output lines. The enable input $\bar{E}$ is used to enable or disable the decoding process. This 3-to-8 decoder can be used as a 1-to-8 demultiplexer as follows.

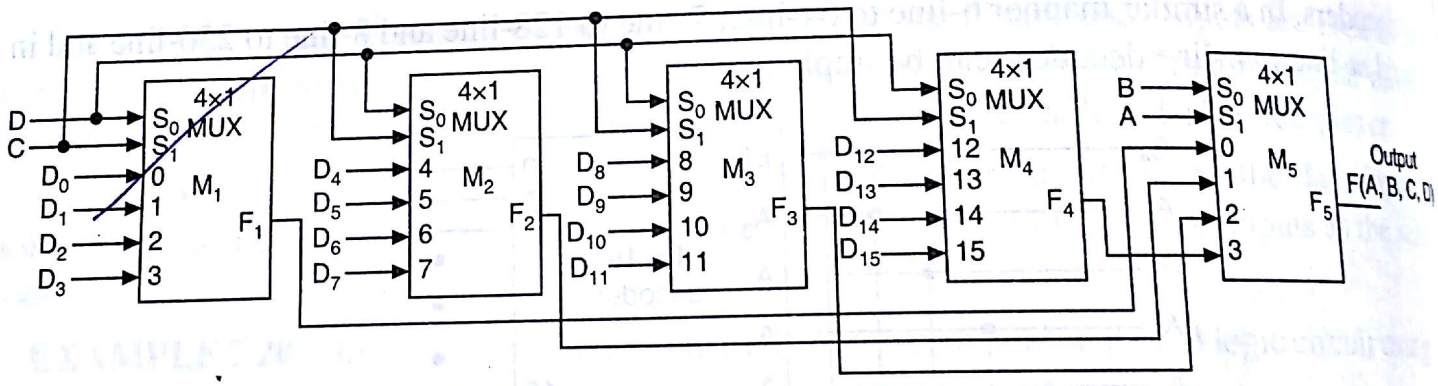
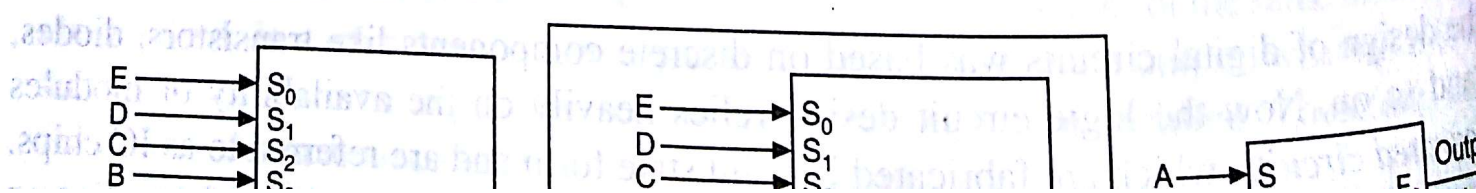


Figure 7.93 16:1 mux using 4:1 muxs.

7.27.2 Design of a 32:1 Mux Using Two 16:1 Muxs and One 2:1 Mux Modules

The arrangement to obtain a 32:1 mux using two 16:1 muxes and one 2:1 mux is shown Figure 7.94. A 32:1 mux has 32 data inputs. So it requires five data select lines. Since a 16:1 has only four data select lines, the inputs B,C,D,E are connected to the data select lines of both 16:1 muxes and the most significant input A is connected to the single data select line of the mux. For the values of BCDE = 0000 to 1111, inputs 0 to 15 will appear at the input terminal the 2:1 mux through the output F_1 of the first 16:1 mux and inputs 16 to 31 will appear at the terminal 1 of the 2:1 mux through the output F_2 of the second 16:1 mux. For A = 0, output For A = 1, output $F = F_2$.



chip with A_{10}. Finally, the two outputs have to be mixed with an output OR gate. The additional hardware in this case is thus one inverter and one OR gate. A similar scheme is commonly used in expanding the size of memory by connecting additional modules.

7.27.5 Design of a 2-bit Comparator Using Two 1-bit Comparator Modules

Let $A = A_1\ A_0$ and $B = B_1\ B_0$ be the 2-bit numbers to be compared. A_1 and B_1 and A_0 and B_0 are compared separately using two 1-bit comparator modules. Then we know

$$A > B \text{ if } A_1 > B_1 \quad \text{or} \quad A_1 = B_1 \text{ and } A_0 > B_0$$

$$\therefore \quad G = G_1 + E_1 \cdot G_0$$

$$A = B \text{ if } A_1 = B_1 \text{ and } A_0 = B_0$$

$$\therefore \quad E = E_1 \cdot E_0$$

$$A < B \text{ if } A_1 < B_1 \quad \text{or} \quad A_1 = B_1 \text{ and } A_0 < B_0$$

$$\therefore \quad L = L_1 + E_1 \cdot L_0$$

So the 2-bit comparator using 1-bit modules is as shown in Figure 7.97.

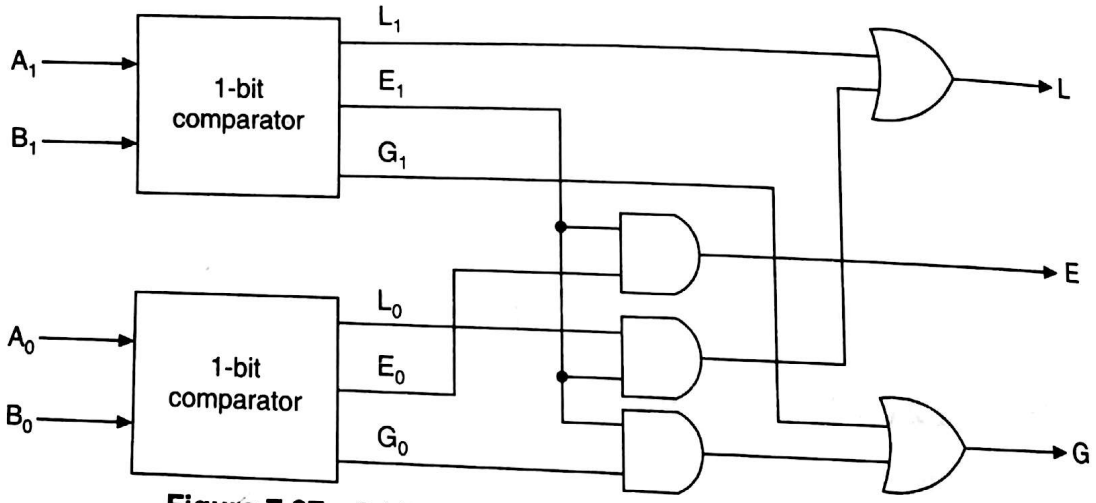


Figure 7.97 2-bit comparator using two 1-bit modules.

7.27.6 Design of a 4-bit Comparator Using Four 1-bit Comparator Modules

Let $A = A_3\ A_2\ A_1\ A_0$ and $B = B_3\ B_2\ B_1\ B_0$ be the two 4-bit numbers to be compared. A_3 and B_3, A_2 and B_2, A_1 and B_1, and A_0 and B_0 are compared separately using four 1-bit comparator modules. Then we know

$$A > B \text{ if } A_3 > B_3 \text{ or } A_3 = B_3 \text{ and } A_2 > B_2 \text{ or } A_3 = B_3 \text{ and } A_2 = B_2 \text{ and}$$
$$A_1 > B_1 \text{ or } A_3 = B_3 \text{ and } A_2 = B_2 \text{ and } A_1 = B_1 \text{ and } A_0 > B_0$$

Therefore, $A > B$: $G = G_3 + E_3G_2 + E_3E_2G_1 + E_3E_2E_1G_0$

$$A = B \text{ if } A_3 = B_3 \text{ and } A_2 = B_2 \text{ and } A_1 = B_1 \text{ and } A_0 = B_0$$

$$\therefore \quad A = B\text{: } E = E_3 \cdot E_2 \cdot E_1 \cdot E_0$$

$$A < B \text{ if } A_3 < B_3 \text{ or } A_3 = B_3 \text{ and } A_2 < B_2 \text{ or } A_3 = B_3 \text{ and } A_2 = B_2 \text{ and } A_1 < B_1$$

or $A_3 = B_3$ and $A_2 = B_2$ and $A_1 = B_1$ and $A_0 < B_0$

$$\therefore \quad A < B\text{: } L = L_3 + E_3\,L_2 + E_3E_2\,L_1 + E_3E_2E_1\,L_0$$

So the 4-bit comparator using four 1-bit modules is shown in Figure 7.98.

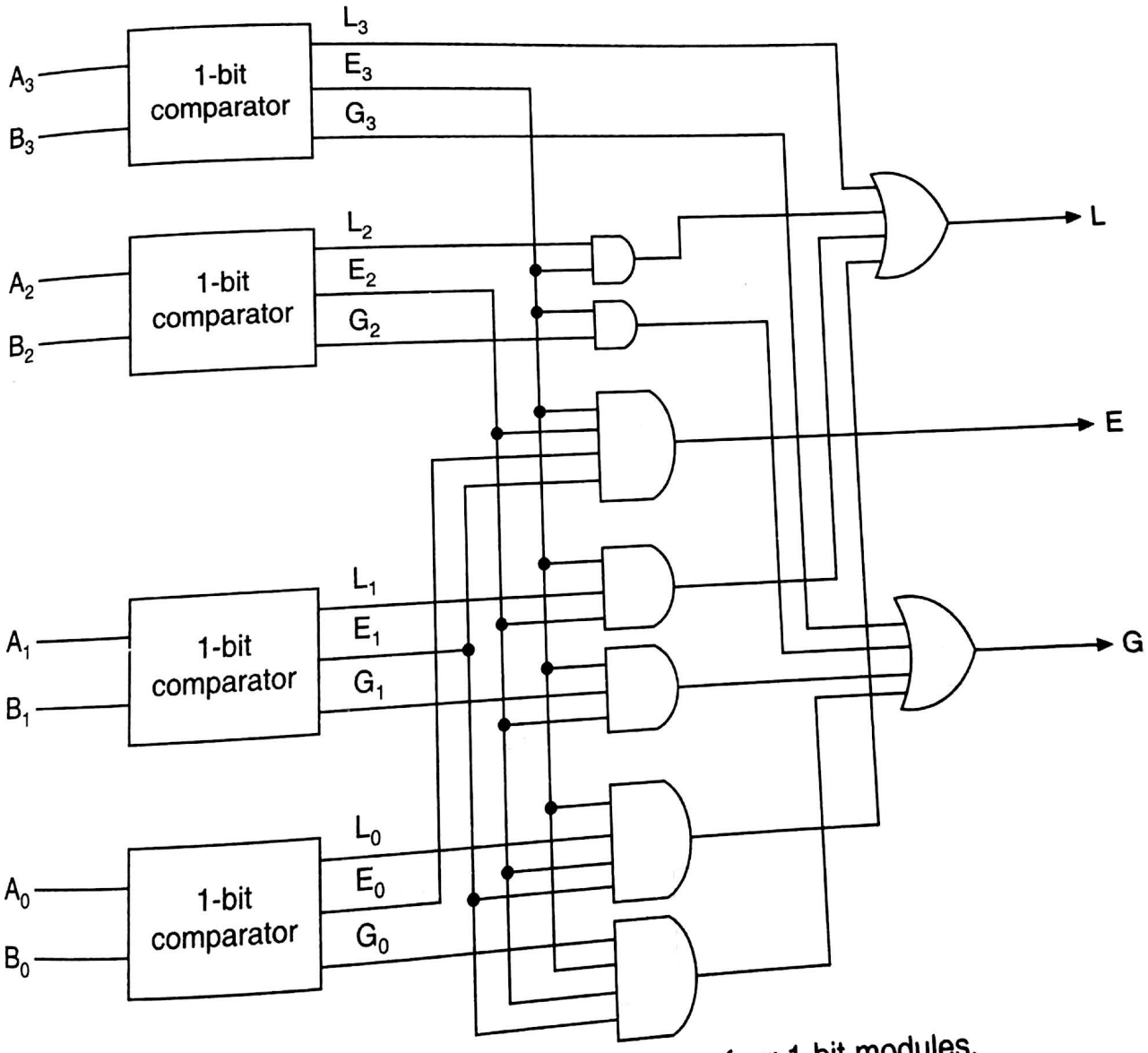


Figure 7.98 4-bit comparator using four 1-bit modules.

7.28 HAZARDS AND HAZARD-FREE REALIZATIONS

Hazards are ... switching transients that may appear at the output of a circuit because ... transient is also called a *glitch* or a