BCD to Excess -3 Code Converter

A code converter combinational circuit is designed to convert BCD code to Excess - 3 code. The input code of code converter is BCD code. The output code of code converter is Excess-3 code. Fig 6.16 shows the logic diagram of BCP to excess-3 code converter

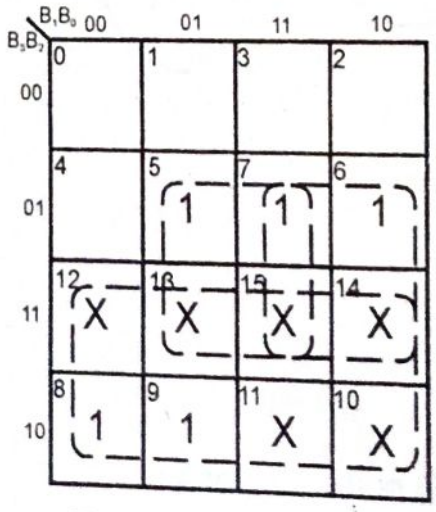
Truth Table

	BCD Code				Excess-3 Code			
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

The unused states are 1010, 1011, 1100, 1101, 1110 and 1111. So place X (Don't Care Condition) for the corresponding cells.

K-Map Simplification

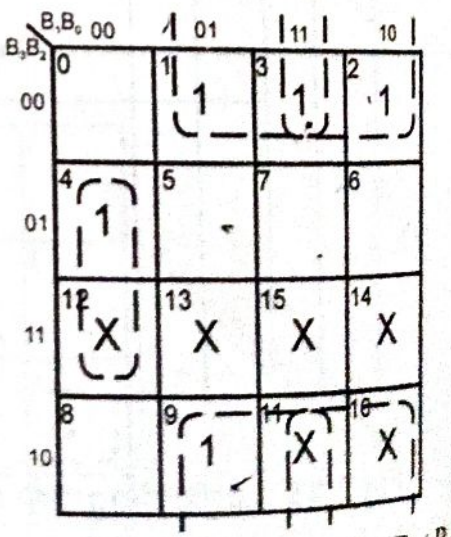
Expression for E_3



$E_3 = B_3 + B_2(B_0 + B_1)$

$E_3 = B_3 + B_2B_0 + B_2B_1$

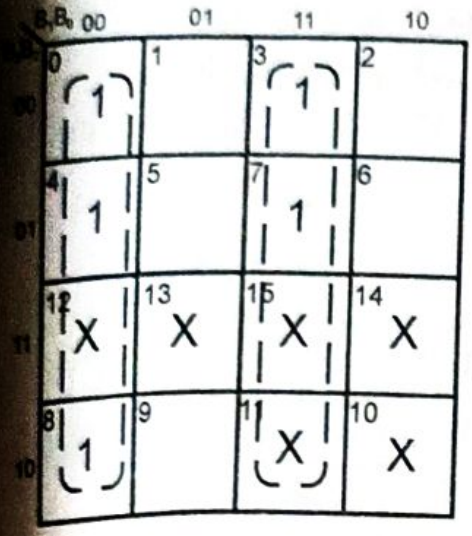
Expression for E_2



$E_2 = B_2B_1'B_0' + B_2'B_0 + B_2'B_1$

$= B_2B_1'B_0' + B_2'(B_0 + B_1)$

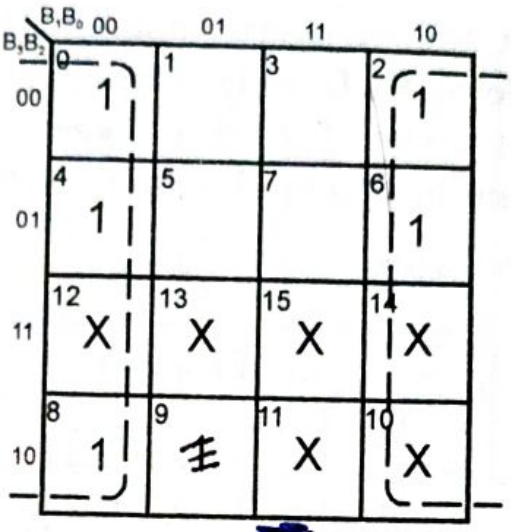
Expression for E_1



$E_1 = B_1' B_0' + B_1 B_0 = B_1 \odot B_0$

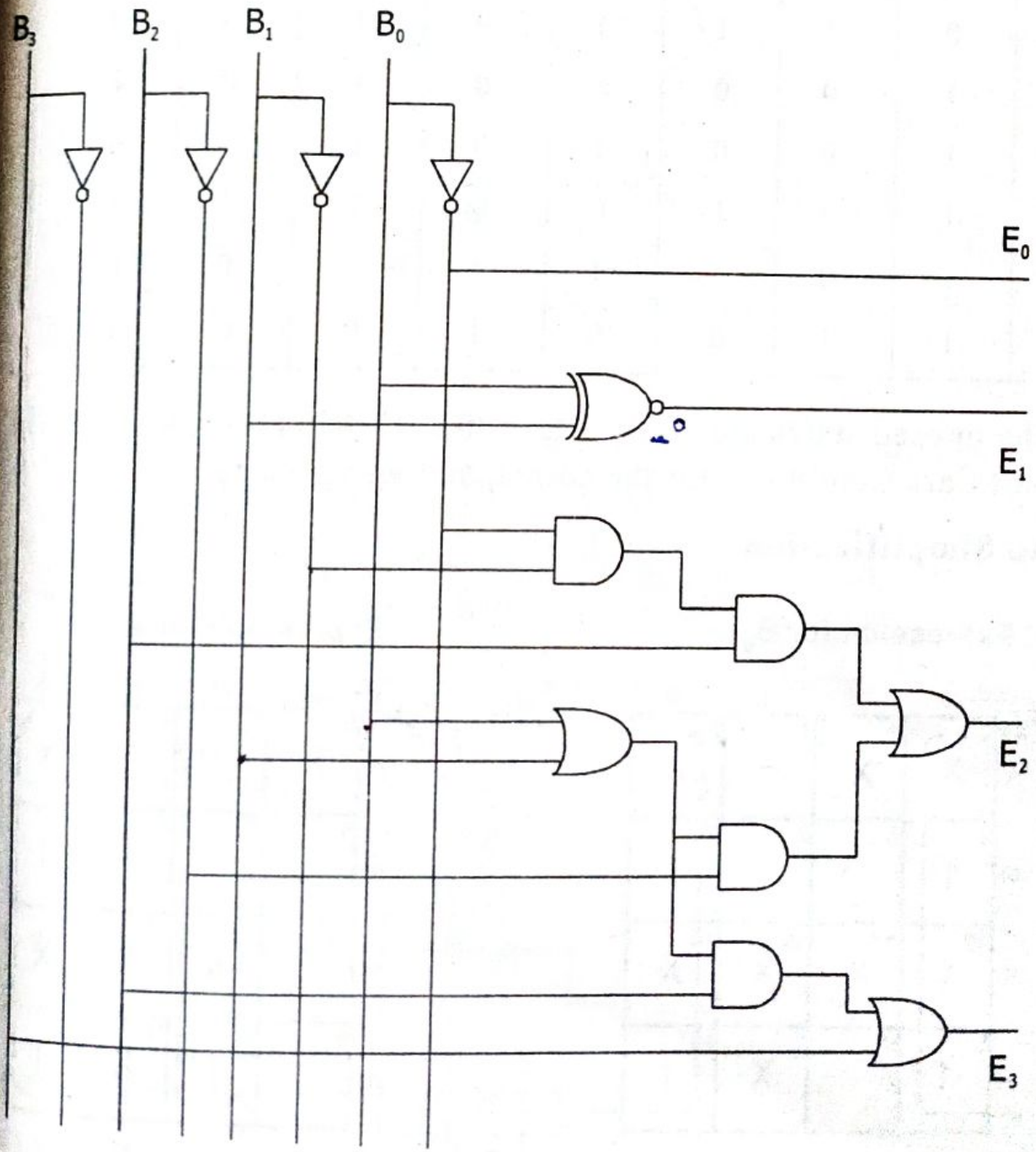
↳ Ex-NOR

Expression for E_0



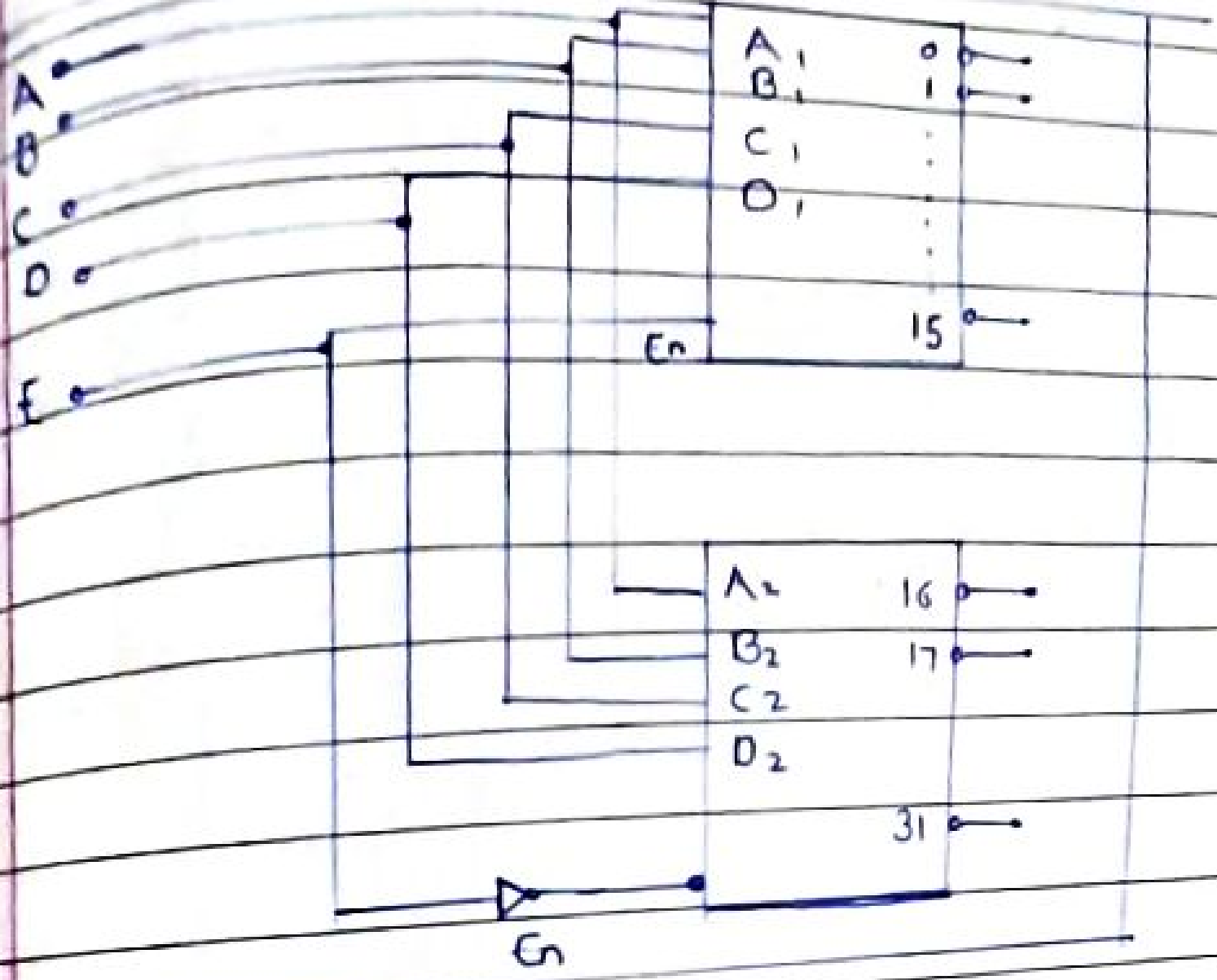
$E_0 = \overline{B_0}$

Logic Diagram



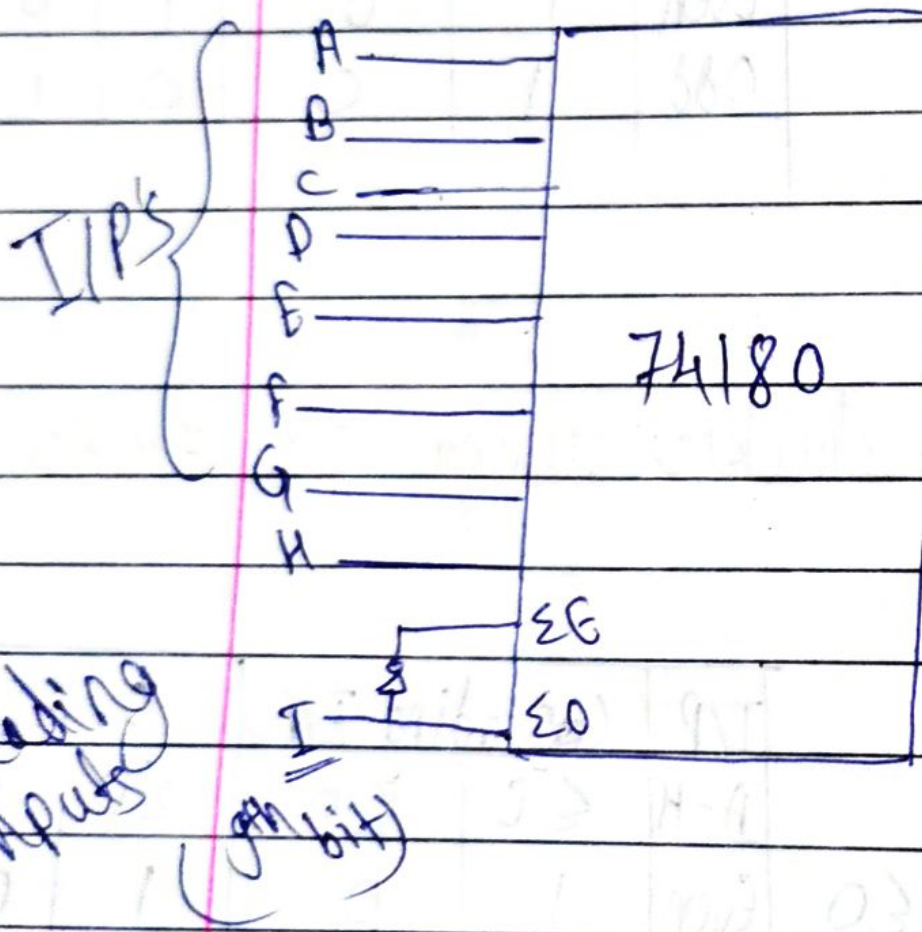
Date :

Design 5 line to 32 decoder using 4 line to 16 decoder



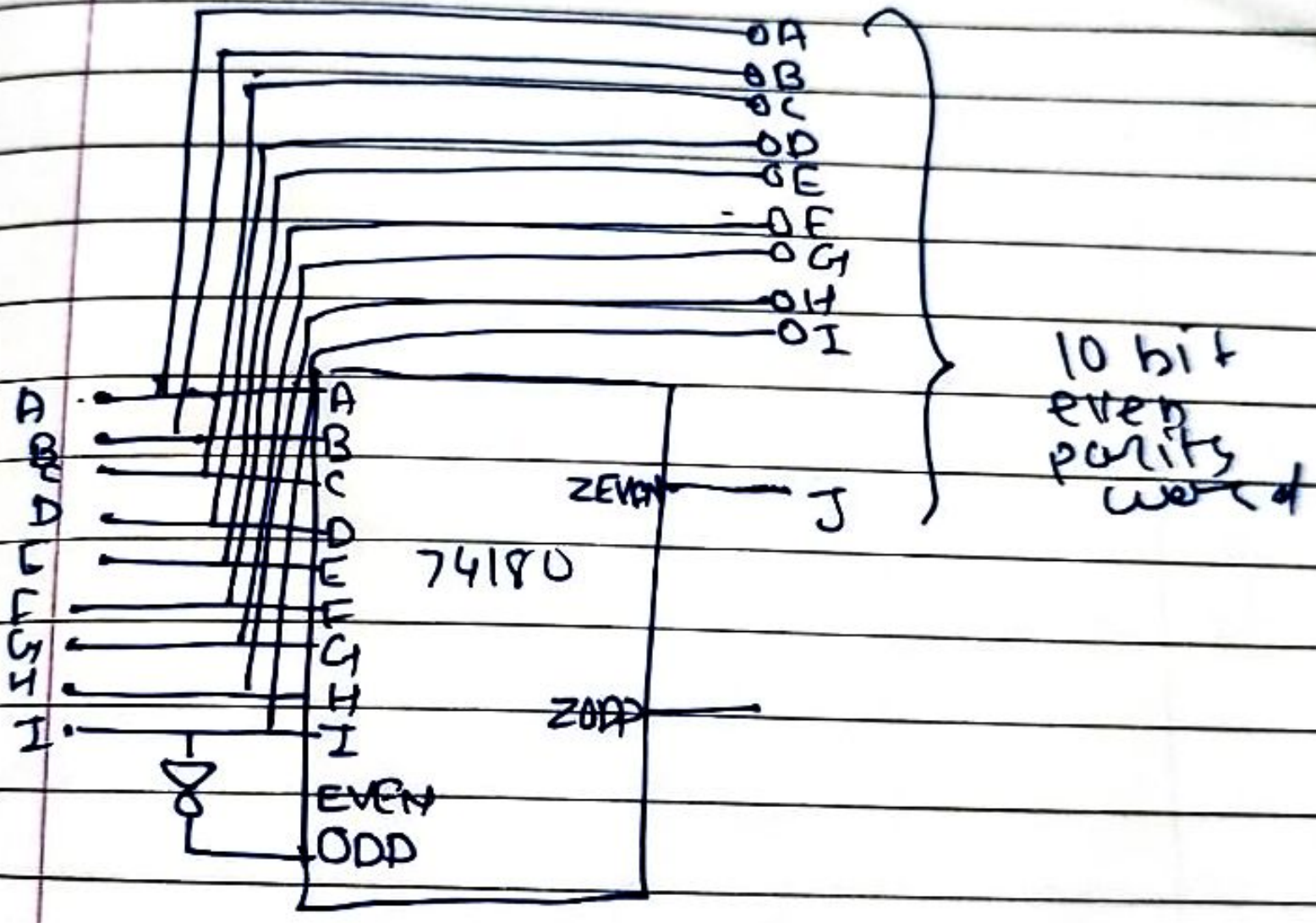
5 line to 32 line decoder

Design 9 bit odd parity checker using IC 74180



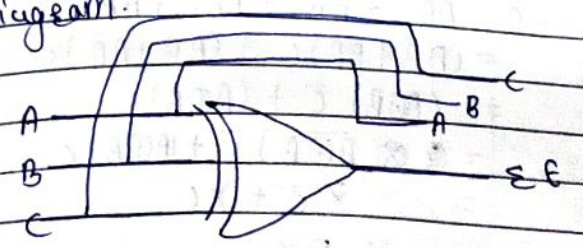
Input	Cascading I/P's		O/P's	
A-H	Σe	Σo	Σe	Σo
Even	1	0	1	0
Even	0	1	0	1
~~Even~~ odd	1	0	0	1
~~Even~~ odd	0	1	1	0

Q10- Design 10-bit even parity generator using single 74180 inverter.



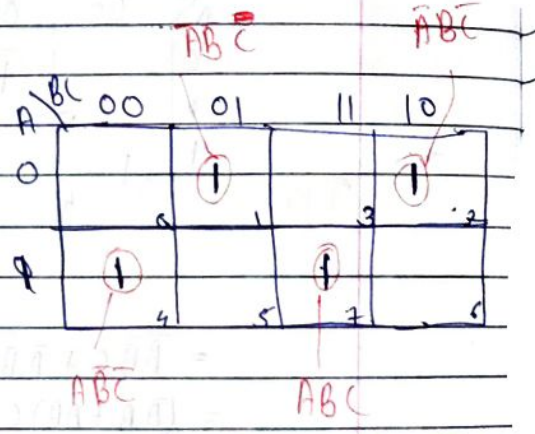
10-bit even parity generator

Circuit diagram.



* Design 3 bit even

I/P			O/P
A	B	C	EE
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1



$EE = \bar{A}\bar{B}C + ABC + A\bar{B}\bar{C} + \bar{A}B\bar{C}$

$= (\bar{A}B + A\bar{B})C + (\bar{A}B + A\bar{B})\bar{C}$

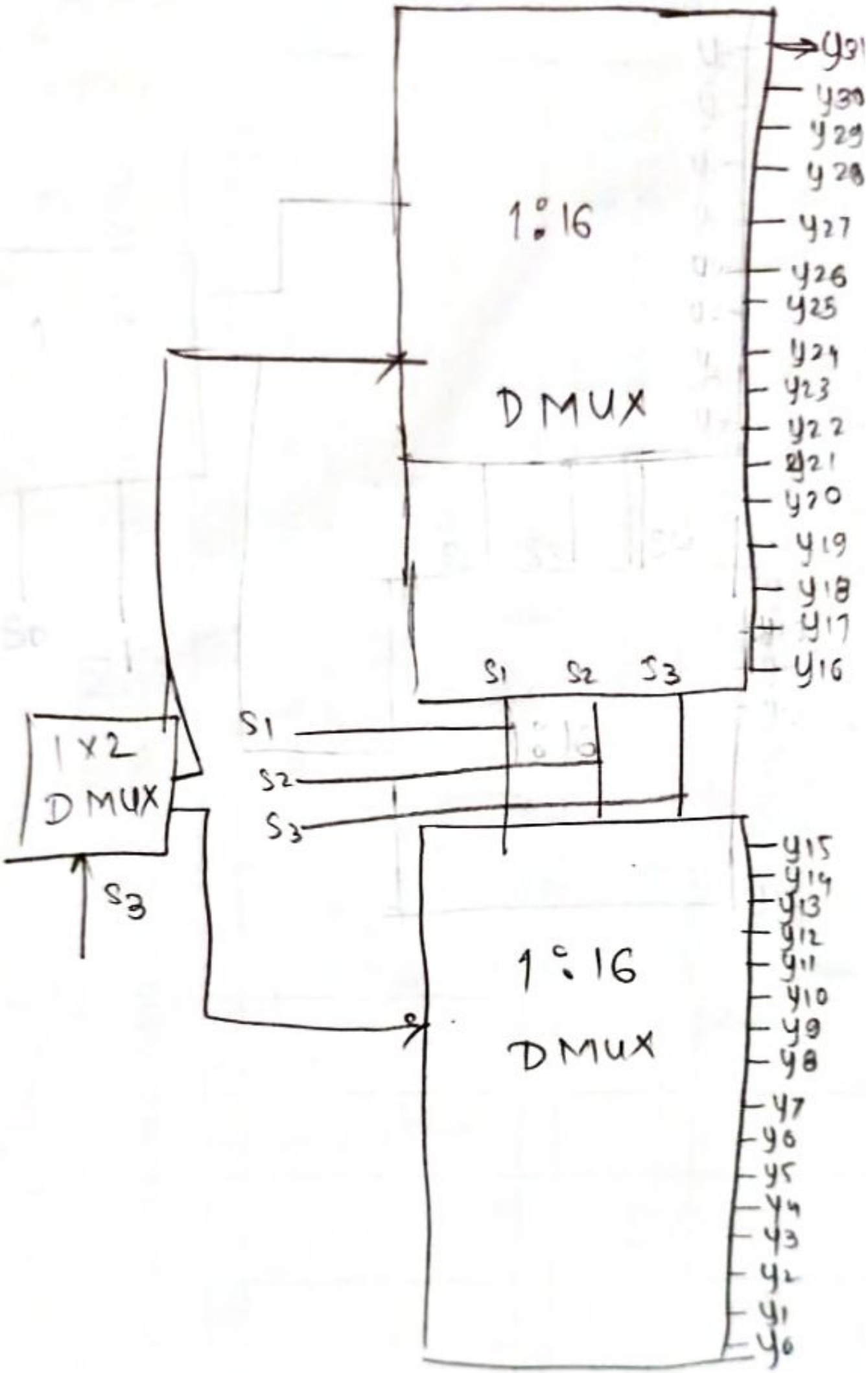
$= (A \oplus B)C + (A \oplus B)\bar{C}$

$= \bar{X}C + X\bar{C}$

$= X \oplus C = A \oplus B \oplus C$

Put $A \oplus B = X$

$A \odot B = \overline{A \oplus B}$


1:16
DMUX
y31
y30
y29
y28
y27
y26
y25
y24
y23
y22
y21
y20
y19
y18
y17
y16
S1
S2
S3
1x2
DMUX
S1
S2
S3
S3
1:16
DMUX
y15
y14
y13
y12
y11
y10
y9
y8
y7
y6
y5
y4
y3
y2
y1
y0

6.10 Decimal to BCD encoder

This type of encoder has ten puts - one for each decimal digit and our outputs corresponding to the BCD ode IC 74147 is used as a decimal to CD encoder. The logic symbol is hown in Fig. 6.11. Both inputs and utputs are active low. It is important o note that there is no input line for decimal zero. When this condition ccurs, all output lines are 1.

Table 6.5 shown the relation etween the decimal and BCD code. A_3 is most significant bit of the BCD code. A_3 is always 1 for decimal digit or 9. The expression for bit A_3 minterms of the decimal digits can written as.

$$A_3 = 8 + 9$$

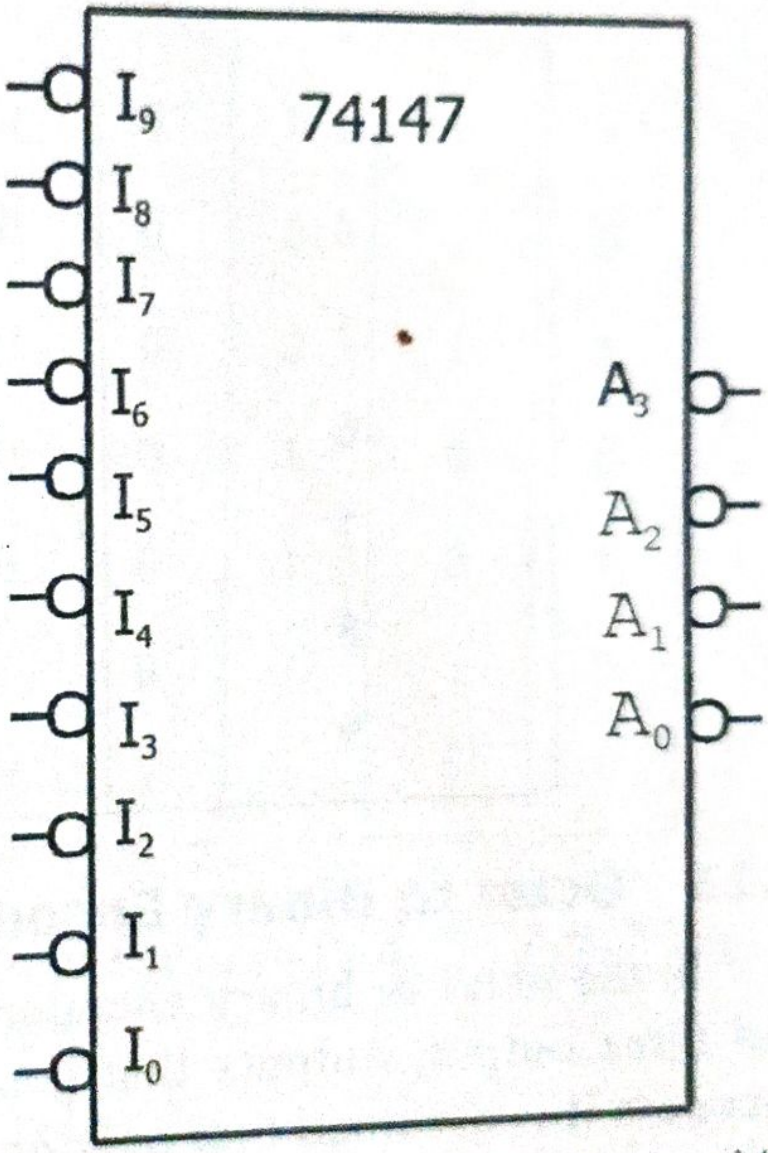


Fig.6.11 Logic symbol for decimal to BCD Encoder

Bit A_2 is always 1 for decimal digit 4,5,6 or 7 can be expressed as an OR function as follows.

$$A_2 = 4 + 5 + 6 + 7$$

Bit A_1 is always 1 for decimal digit 2, 3, 6 or 7 and can be expressed as

$$A_1 = 2 + 3 + 6 + 7$$

Bit A_0 is always 1 for decimal digit 1, 3, 5, 7 or 9 the expression for A_0 is

$$A_0 = 1 + 3 + 5 + 7 + 9$$

Now, we can draw the decimal to BCD encoder by using the above four expression. Figure 6.12 shows the decimal to BCD encoder.

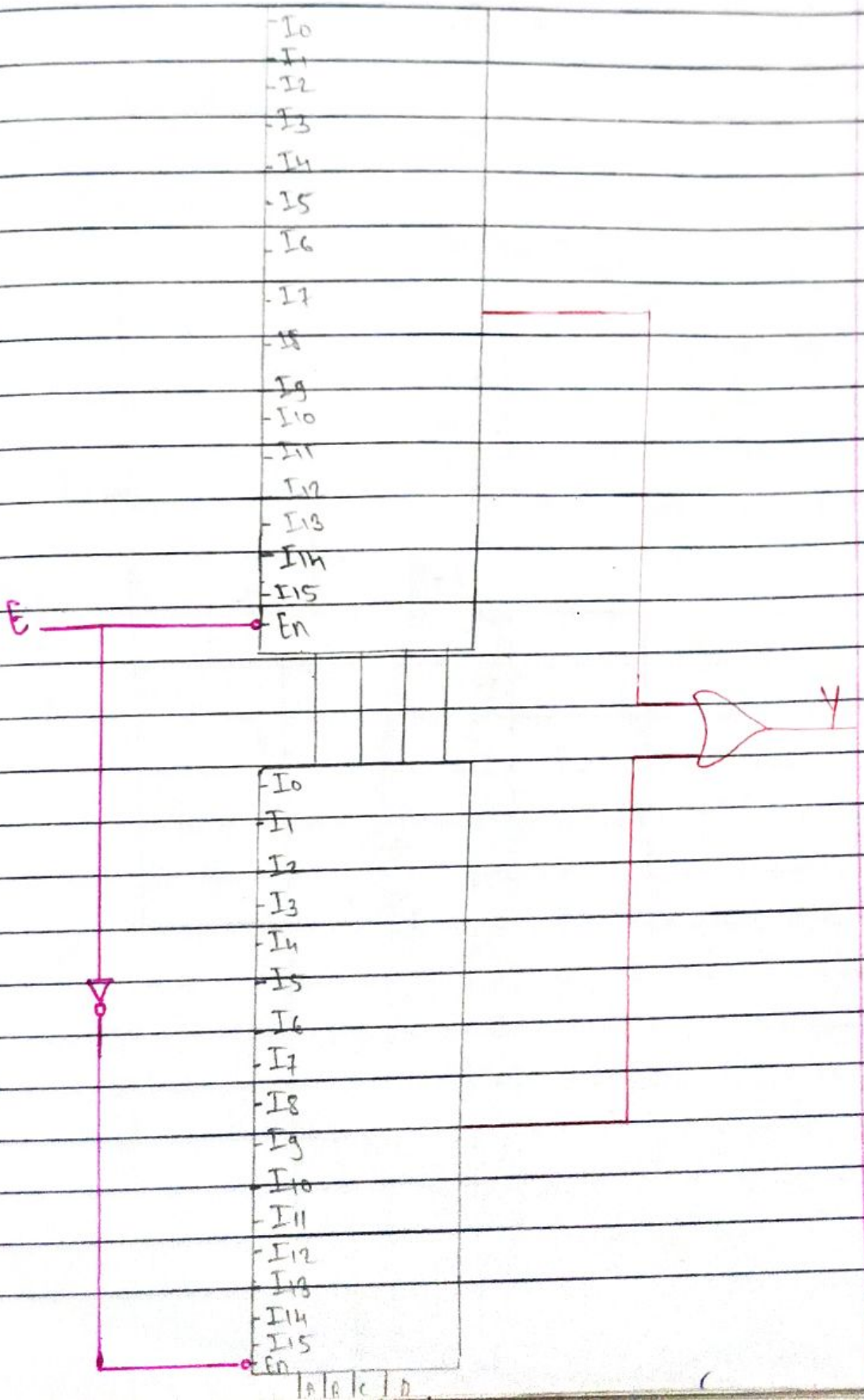
When a HIGH is appears on one of the decimal digit input lines, the appropriate levels occur on the four BCD output lines.

Table 6.5 Truth table Decimal to BCD encoder

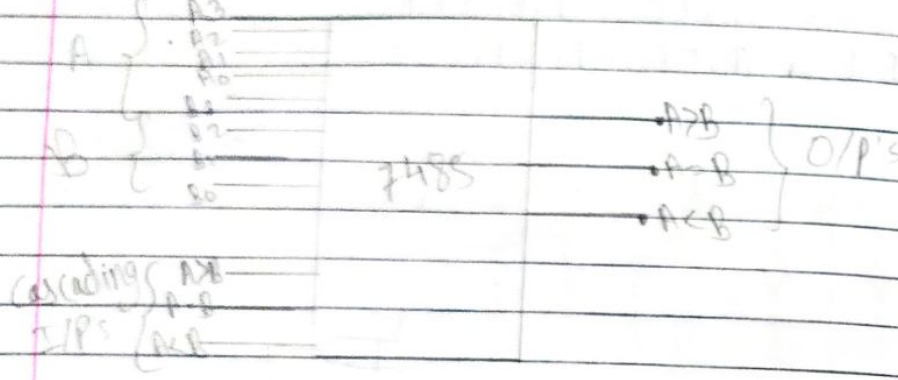
Decimal digit	BCD Code			
	A_3	A_2	A_1	A_0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1

* Design 32:1 Mux Using ① 16:1 Mux & OR gate
② 16:1 Mux & 2:1 Mux
③ 8:1 Mux & 4:1 Mux

⇒ ① 16:1 Mux & OR gate (En is 5th select line)

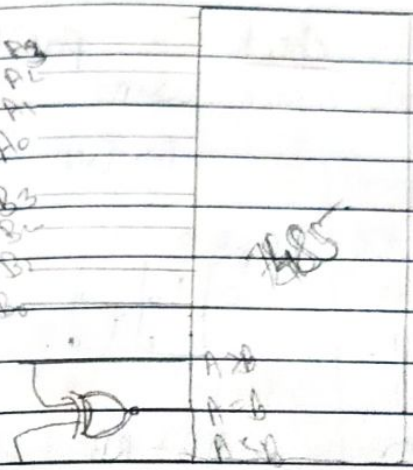


3) Design 4-bit Comparator



I/P's A,B	Cas. I/P's A>B	A=B	A<B	O/P's A>B	A=B	A<B
A>B	×	×	×	1	0	0
A=B	1	0	0	0	0	0
	0	1	0	0	1	0
	0	0	1	0	0	1
A<B	×	×	×	0	0	1

4) Design 5 bit comparator



I/P's A,B	Cas I/P's A>B	A=B	A<B	O/P's A>B	A=B	A<B
A>B	×	×	×	1	0	0
A=B	1	0	0	1	0	0
	0	1	0	0	1	0
	0	0	1	0	0	1
A<B	×	×	×	0	0	1

Design a Binary-to-Gray code converter.

Solution

The truth table of Binary-to-Gray code converter is given in Table 2.8. For each of the four outputs, K-maps are prepared and simplified. The K-maps are given in Fig. 5.34 and the simplified expressions are given by Eqs (5.50). The circuit is given in Fig. 5.35.

$$G_3 = B_3 \tag{5.50a}$$

$$G_2 = B_2 \oplus B_3 \tag{5.50b}$$

$$G_1 = B_1 \oplus B_2 \tag{5.50c}$$

$$G_0 = B_0 \oplus B_1 \tag{5.50d}$$

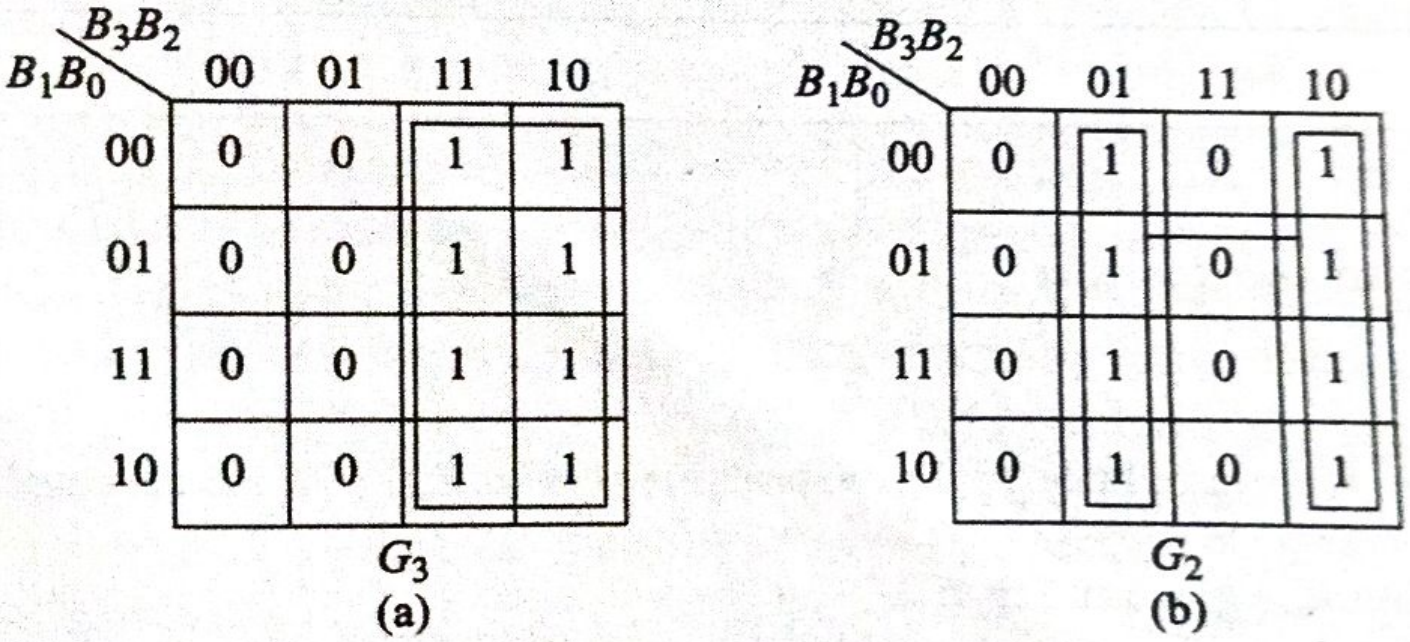


Fig. 5.34 ***K-maps of Ex. 5.19***

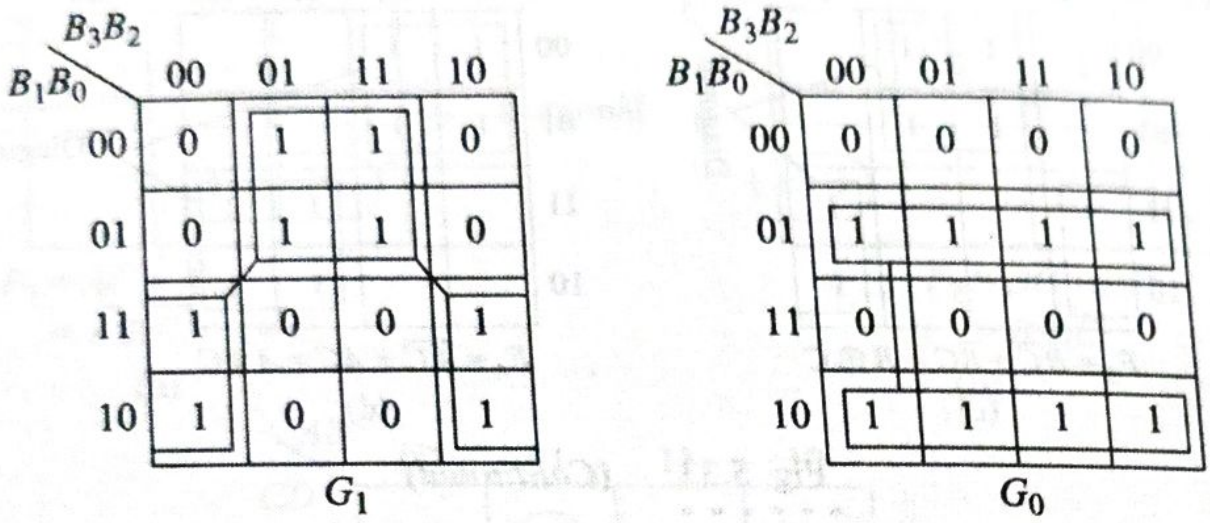


Fig. 5.34 ***(Continued)***

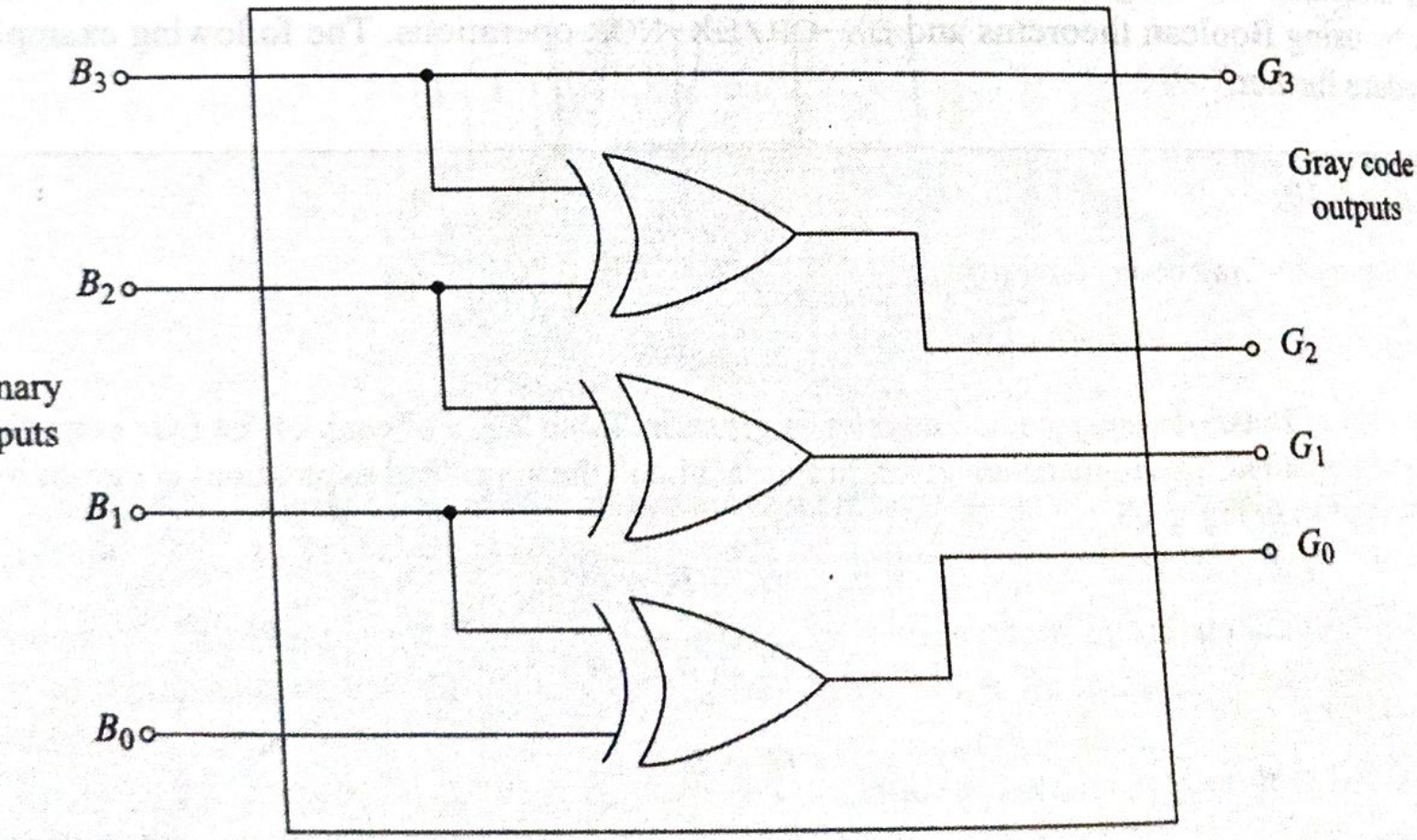


Fig. 5.35 ***Binary-to-Gray Code Converter***

6.4.1 Adder with Look-Ahead Carry

In the half-adder circuit of Fig. 5.19 and full-adder circuits of Fig. 5.21 and Prob. 5.19*b*, it was assumed that the inputs (augend and addend) and carry input are simultaneously present and the sum and carry outputs are generated instantaneously. However, in Prob. 5.20 it has been shown that even if the augend, addend, and carry inputs are present simultaneously, then the sum and carry outputs will be delayed due to the propagation delays of gates through which the signals are passing. Now, if we consider the *n*-bit parallel adder of Fig. 6.12*a*, we observe that the sum (S_0) and carry (C_0) outputs are delayed because of the propagation delays of the gates involved in FA0. The S_0 output, however, is the LSB of the final result and it is not required to be passed through other gates and hence does not get further delayed. The carry output C_0 acts as carry input of the full-adder FA1 and therefore, the outputs of FA1 S_1 and C_1, will reach steady-state only after arrival of C_0 and propagation delay introduced by FA1. This means the total delay upto FA1 will be the sum of delays introduced by FA0 and FA1. Similarly going further in the chain of adders towards MSB, the carry has to ripple through all the stages, thereby reducing the speed of the adder as the number of adder stages are increased (Prob. 6.11).

To design fast operating parallel adders, we can use gates with lower propagation delay time. Even with this, the delay time of the adder will increase with increasing number of bits to be added. Another approach most commonly used is the concept of *look-ahead carry*. Although it requires additional circuitry but the speed of the adder becomes independent of the number of bits.

Let us consider a full-adder circuit in block diagram form and its EX-OR realisation (Prob. 5.19(b)) shown in Fig. 6.13. Here,

$$P_i = A_i \oplus B_i \tag{6.3a}$$
$$G_i = A_i B_i \tag{6.3b}$$
$$S_i = P_i \oplus C_{i-1} = A_i \oplus B_i \oplus C_{i-1} \tag{6.3c}$$
$$C_i = G_i + P_i C_{i-1} \tag{6.3d}$$

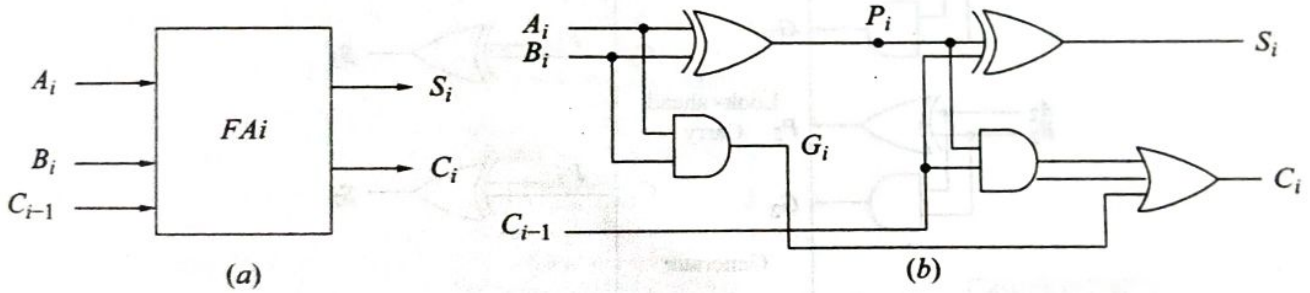


Fig. 6.13 ***(a) Block Diagram of ith Stage of a Full-Adder***
(b) EX-OR ***Implementation of Full-Adder***

The output G_i of the first half-adder is 1 if A_i and B_i both are 1 and a carry is generated. The variable G_i is known as a *carry generate*. Its value is independent of the input carry. The variable P_i is known as a *carry propagate* because this is the term associated with the propagation of the carry from C_{i-1} to C_i. Using Eq. (6.3d), we write Boolean expression for carry output of each stage:

$$C_{i+1} = G_{i+1} + P_{i+1} \cdot C_i$$

Substituting the value of C_i from Eq. (6.3d), we obtain,

$$C_{i+1} = G_{i+1} + P_{i+1}\, G_i + P_{i+1} P_i C_{i-1} \tag{6.4}$$

Similarly, for an *n*-stage adder, the final carry C_{n-1} can be determined. For clear understanding of the advantage of this approach, we consider a 4-bit adder and formulate Boolean expressions for the carry outputs C_0, C_1, C_2, and C_3. We obtain,

$$P_0 = A_0 \oplus B_0 \quad \text{and} \quad G_0 = A_0 B_0$$

$$C_0 = G_0 + P_0 C_{-1} \tag{6.5a}$$

$$C_1 = G_1 + P_1 G_0 + P_1 P_0 C_{-1} \tag{6.5b}$$

$$C_2 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_{-1} \tag{6.5c}$$

$$C_3 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 C_{-1} \tag{6.5d}$$

Now, let us observe the logic variables involved in Eq. (6.5), these are,

$$G_0, G_1, G_2, G_3, P_0, P_1, P_2, P_3, \text{ and } C_{-1}$$

The G variables can be generated directly from A and B inputs using AND gates (Eq. 6.3b), the P variables are obtained again directly from A and B inputs using EX-OR gates (Eq. 6.3a). C_{-1} is the carry input. If Gs, Ps, and C_{-1} are available simultaneously, the carry outputs C_0, C_1, C_2, and C_3 are produced by using 2-level realisation (AND-OR or NAND-NAND) since Eq. 6.5 gives these outputs in SOP form. Therefore, for the generation of these carry outputs, propagation delay time of two gates only will be there. These carry outputs are connected to the carry inputs of the succeeding stages, thereby eliminating the problem of carry rippling through all the stages.

Logic circuit of a look-ahead carry generator can be prepared using Eqs. 6.5. A 4-bit adder with look-ahead carry is shown in Fig. 6.14. Thus we see that by generating all the individual carry terms needed by each full-adder in a 2-level circuit, the propagation delay time through the adder is considerably reduced and it becomes independent of the number of full-adders in the circuit.

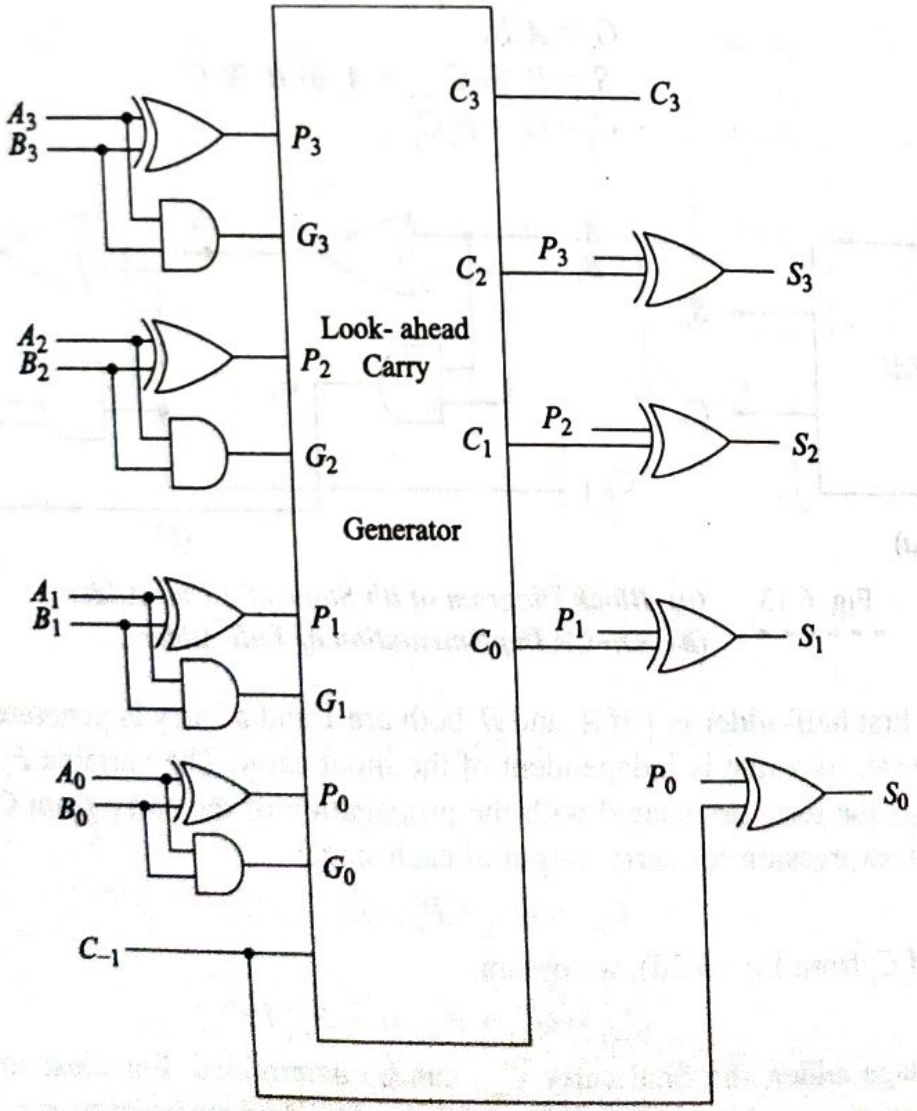


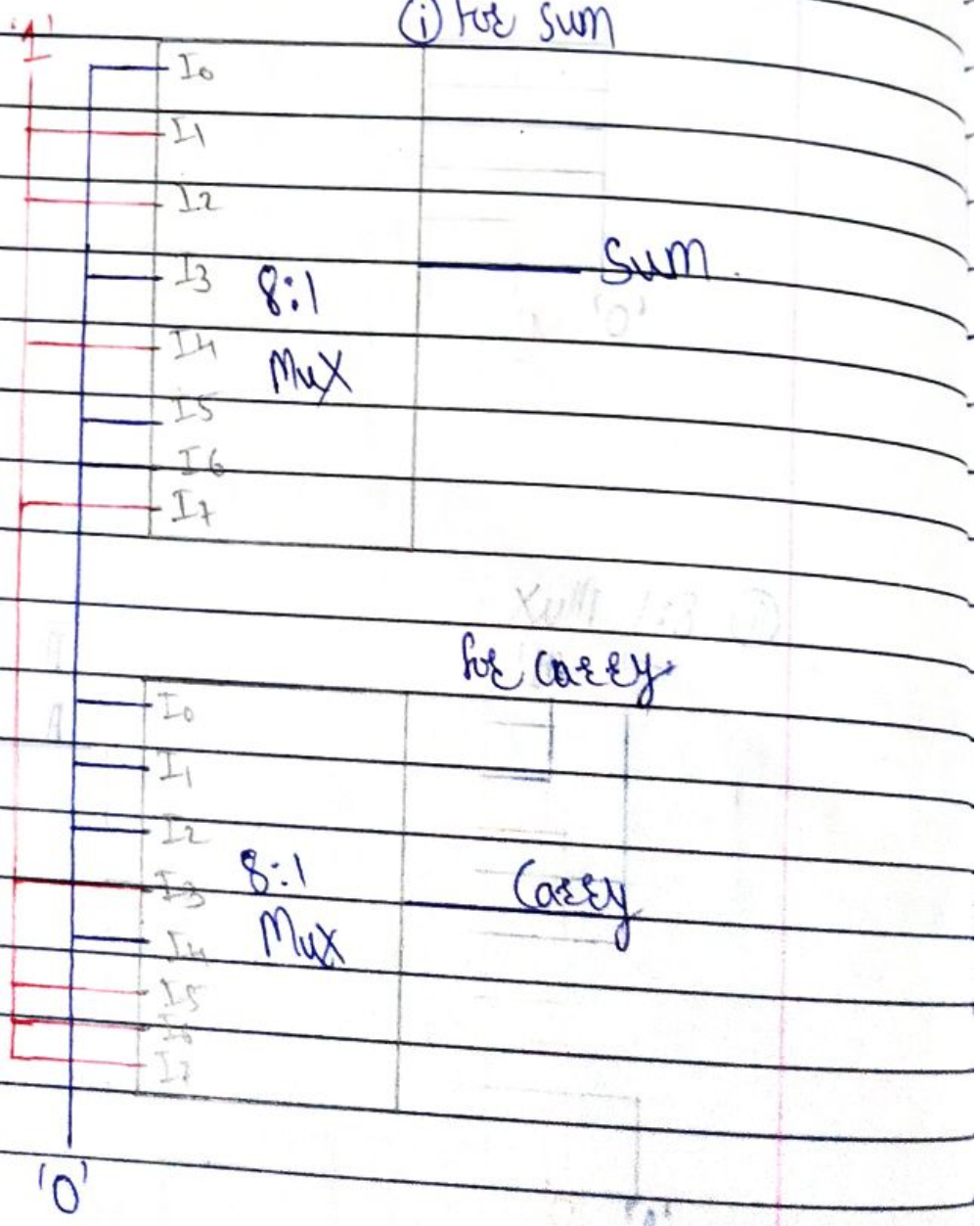
Fig. 6.14 *A 4-bit Adder with Look-Ahead Carry*

Implement Full Adder using 8:1 Mux

I/P			O/P	
A	B	C	S	C
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$S = \Sigma m (1,2,4,7)$

$C = \Sigma m (3,5,6,7)$



Implement the following multi-output combinational logic circuit using a 4-to-16-line decoder.

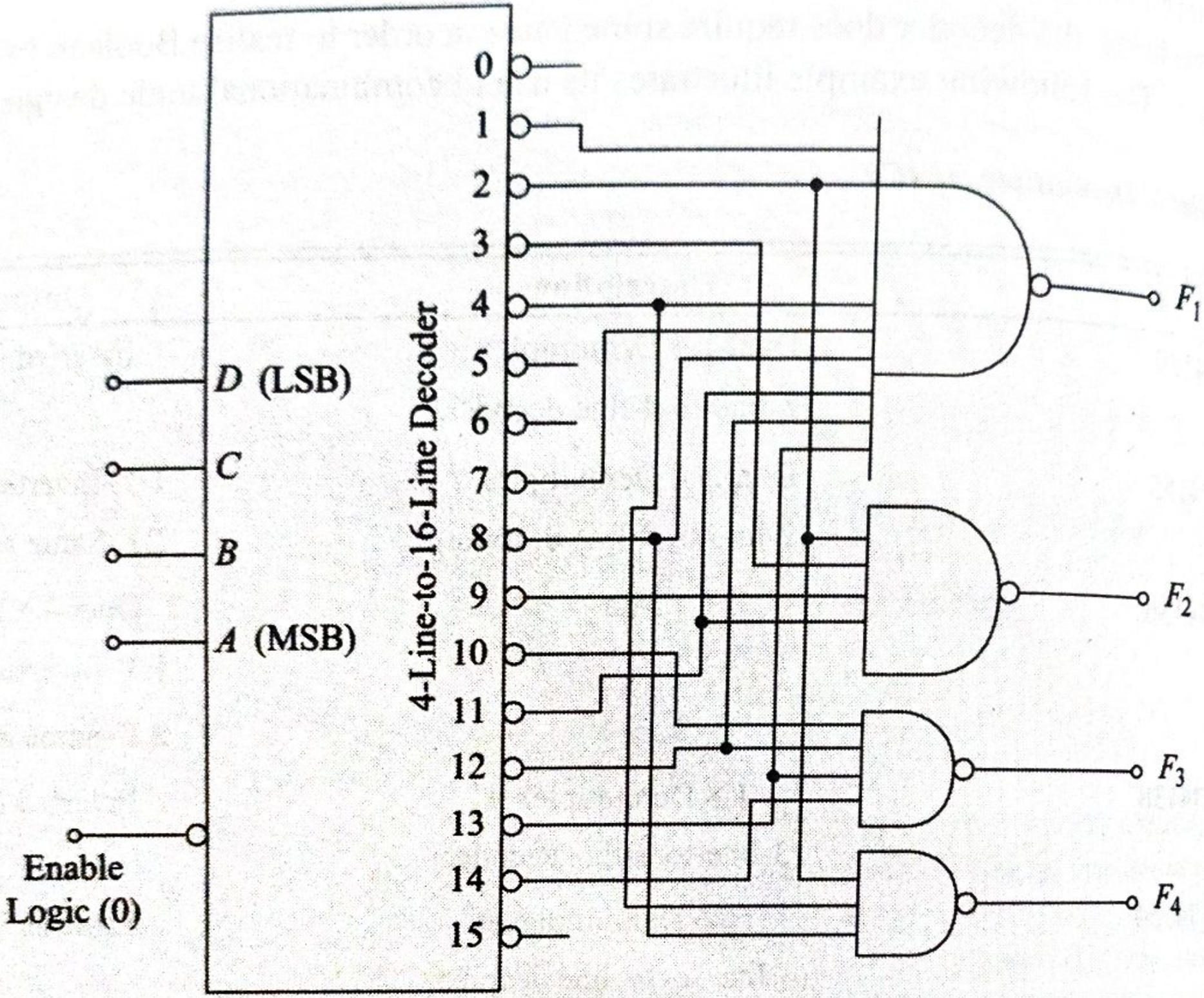
$$F_1 = \Sigma m\,(1, 2, 4, 7, 8, 11, 12, 13)$$
$$F_2 = \Sigma m\,(2, 3, 9, 11)$$
$$F_3 = \Sigma m\,(10, 12, 13, 14)$$
$$F_4 = \Sigma m\,(2, 4, 8)$$

Solution

The realisation is shown in Fig. 6.8.

The four-bit input $ABCD$ is applied at the Select input terminals S_3, S_2, S_1, and S_0. The output F_1 is required to be 1 for minterms 1, 2, 4, 7, 8, 11, 12, and 13. Therefore, a NAND gate is connected as shown. Similarly NAND gates are used for the outputs F_2, F_3, and F_4. Here, the decoder's outputs are active-low, therefore a NAND gate is required for every output of the combinational circuit.

In the combinational logic design using multiplexer, additional gates are not required, whereas design using demultiplexer requires additional gates. However, even with this disadvantage, the decoder is more economical in cases where nontrivial, multiple-output expressions of the same input variables are required. In such cases, one multiplexer is required for each output whereas it is likely that only one decoder will be required, supported with a few gates. Therefore, using a decoder could have advantages over using a multiplexer.


4-Line-to-16-Line Decoder
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
D (LSB)
C
B
A (MSB)
Enable
Logic (0)
F1
F2
F3
F4

Implement the following boolean expression using multiplexer

(a) $Y = \bar{A}B\bar{C} + AB\bar{C}D + \bar{A}\bar{B}D + \bar{A}CD$

$= \bar{A}B\bar{C}(D+\bar{D}) + AB\bar{C}D + \bar{A}\bar{B}D(C+\bar{C}) + \bar{A}CD(B+\bar{B})$

$= \bar{A}B\bar{C}D + \bar{A}B\bar{C}\bar{D} + A\bar{B}\bar{C}D + \bar{A}\bar{B}DC + \bar{A}\bar{B}D\bar{C} + \bar{A}CDB + \bar{A}CD\bar{B}$

$= \bar{A}B\bar{C}D + \bar{A}B\bar{C}\bar{D} + A\bar{B}\bar{C}D + \bar{A}\bar{B}DC + \bar{A}BCD + \bar{A}\bar{B}CD + \bar{A}\bar{B}\bar{C}D$

in SOP

$\bar{A}=0$

$A=1$

$\bar{A}B\bar{C}D = 0101 = 5$

$\bar{A}B\bar{C}\bar{D} = 0100 = 4$

$A\bar{B}\bar{C}D = 1001 = 9$

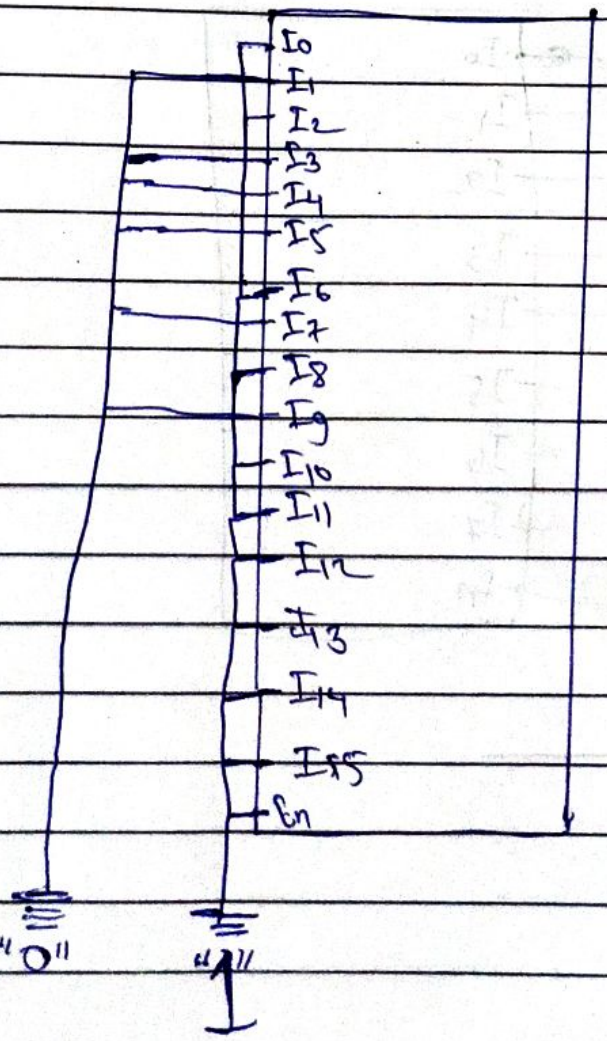
$\bar{A}\bar{B}CD = 0011 = 3$

$\bar{A}BCD = 0111 = 7$

$\bar{A}\bar{B}\bar{C}D = 0001 = 1$

$Y = \sum m(1,3,4,5,7,9)$

$Y = M(0,2,6,8,10,11,12,13,14,15)$



(2) $Y = (A+B)(\bar{A}+B+C)(A+\bar{B})$

$Y = [(A+B)+C.\bar{C}][\bar{A}+B+C].[(A+B)+C.\bar{C}]$

in POS

$= (A+B+C).(A+B+\bar{C}).(\bar{A}+B+C).(A+\bar{B}+C)(A+\bar{B}+\bar{C})$

$\bar{A}=1$

$A=0$

$A+B+C = 000 = 0$

$A+B+\bar{C} = 001 = 1$

$\bar{A}+B+C = 100 = 4$

$A+\bar{B}+C = 010 = 2$

$A+\bar{B}+\bar{C} = 011 = 3$

$Y = \Pi M(0,1,2,3,4)$

$Y = \Sigma M(5,6,7)$