3

Introduction to Combinational Logic Design

Chapter Outline

Combinational circuits are generally built using the basic digital functional blocks of AND, OR and NOT and their combinations. Fairly complex circuits can be built. This chapter addresses the systematic procedure for the analysis and design of simple combinational circuits such as decoders and encoders.

3.1 Combinational Logic Design

In a combinational circuit, the outputs depend strictly on the present inputs. As in most design scenario, combinational logic design too begins with a verbal description of the problem which is then written in the form of a truth table is discussed in section 1.2. Depending on the number of variables involved, the function as represented in the truth table is minimized either by using

Karnaugh maps or by employing algorithmic procedures like the Quine-McCluskey technique to obtain the output function SOP in POS form. These simplified switching expressions are then implemented as required to arrive at the final schematic. This process will now be illustrated with a few interesting examples.

Design a combinational circuit to find the 9s complement of a *BCD* numbers.

Solution: Let us construct the truth table with the verbal description given in the problem. We know that the 9s complements of 1 is 8, of 2 is 7 and so on as shown in the table of Fig 3.1. Let *ABCD* be the *BCD* number and let *WXYZ* be it's 9s complement.

Cell no.	*A*	*B*	*C*	*D*	*W*	*X*	*Y*	*Z*
0	0	0	0	0	1	0	0	1
1	0	0	0	1	1	0	0	0
2	0	0	1	0	0	1	1	1
3	0	0	1	1	0	1	1	0
4	0	1	0	0	0	1	0	1
5	0	1	0	1	0	1	0	0
6	0	1	1	0	0	0	1	1
7	0	1	1	1	0	0	1	0
8	1	0	0	0	0	0	0	1
9	1	0	0	1	0	0	0	0
10	1	0	1	0	–	–	–	–
11	1	0	1	1	–	–	–	–
12	1	1	0	0	–	–	–	–
13	1	1	0	1	–	–	–	–
14	1	1	1	0	–	–	–	–
15	1	1	1	1	–	–	–	–

Fig. 3.1 Truth table for BCD to its 9s complement conversion

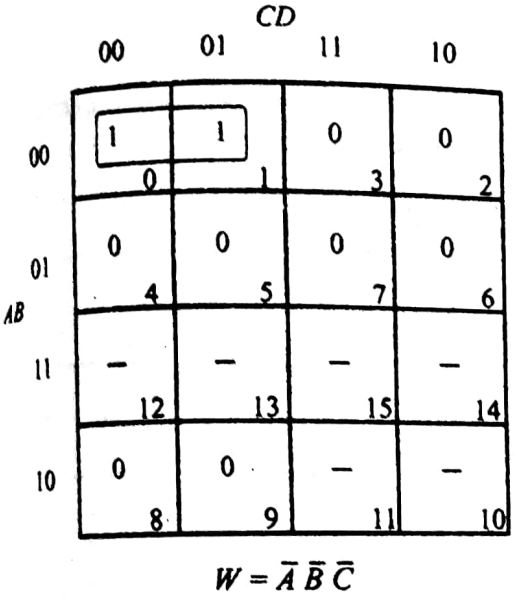
The minterm expressions are:

$$W = \Sigma(0, 1) + \Sigma d(10, 11, 12, 13, 14, 15)$$
$$X = \Sigma(2, 3, 4, 5) + \Sigma d(10, 11, 12, 13, 14, 15)$$

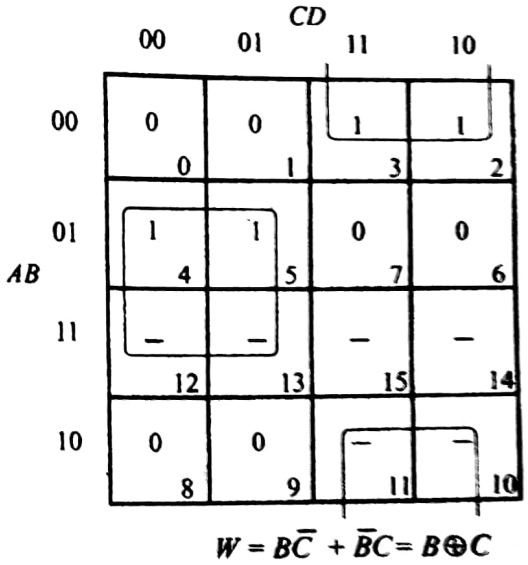
$$Y = \Sigma\,(2, 3, 6, 7) + \Sigma d\,(10, 11, 12, 13, 14, 15)$$

$$Z = \Sigma\,(0, 2, 4, 6, 8) + \Sigma d\,(10, 11, 12, 13, 14, 15)$$

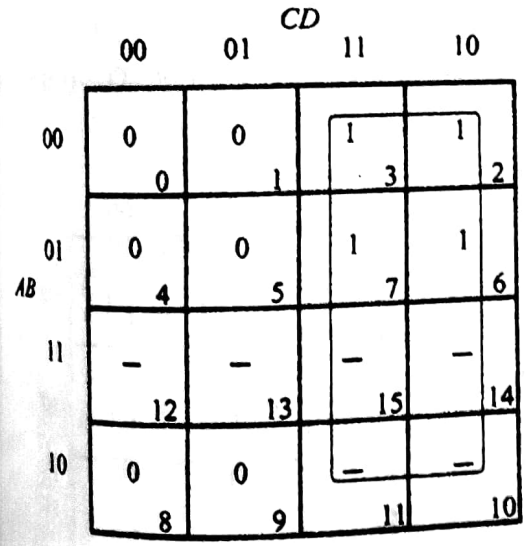
Let us proceed to draw the Karnaugh maps for the four outputs.



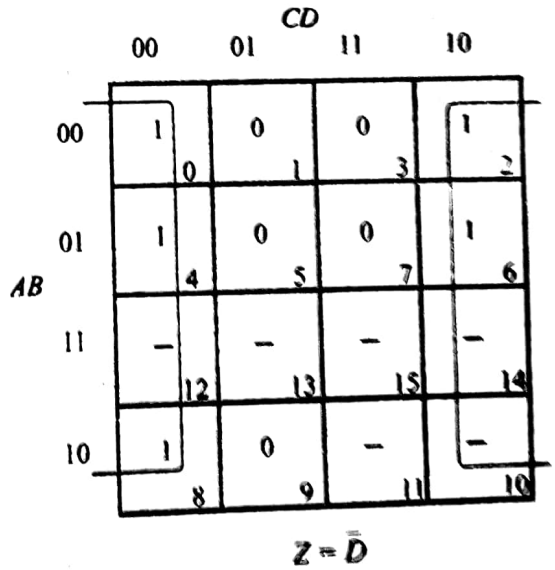
$W = \bar{A}\,\bar{B}\,\bar{C}$



$W = B\bar{C} + \bar{B}C = B \oplus C$



$Y = C$



$Z = \bar{D}$

The implementation is shown in Fig. 3.2.

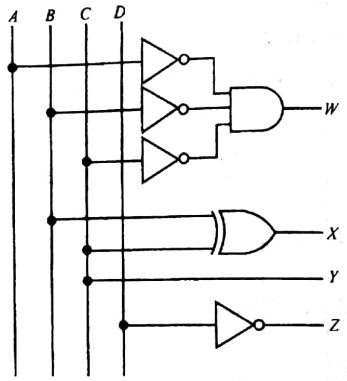


Fig. 3.2 *BCD* to 9s complement converter circuit

Example 3.1

Design a combinational circuit to convert *BCD* to excess-3.

Solution : Let *ABCD*, be the *BCD* input and *WXYZ* be the excess-3 output. The truth table is shown in Fig 3.3.

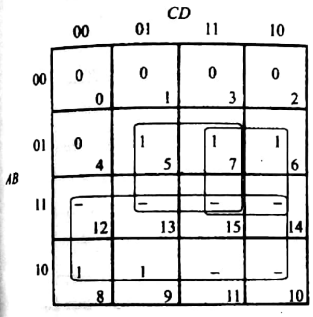
Cell no.	A	B	C	D	W	X	Y	Z
0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	0	1
3	0	0	1	1	0	1	1	0
4	0	1	0	0	0	1	1	1
5	0	1	0	1	1	0	0	0
6	0	1	1	0	1	0	0	1
7	0	1	1	1	1	0	1	0
8	1	0	0	0	1	0	1	1
9	1	0	0	1	1	1	0	0
10	1	0	1	0	–	–	–	–
11	1	0	1	1	–	–	–	–
12	1	1	0	0	–	–	–	–
13	1	1	0	1	–	–	–	–
14	1	1	1	0	–	–	–	–
15	1	1	1	1	–	–	–	–

Fig. 3.3 Truth table for *BCD* to excess-3 converter

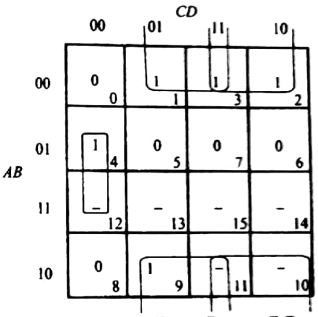
The output minterm expressions are:

$$W = \Sigma(5, 6, 7, 8, 9) + \Sigma d(10, 11, 12, 13, 14, 15)$$
$$X = \Sigma(1, 2, 3, 4, 9) + \Sigma d(10, 11, 12, 13, 14, 15)$$
$$Y = \Sigma(0, 3, 4, 7, 8) + \Sigma d(10, 11, 12, 13, 14, 15)$$
$$Z = \Sigma(0, 2, 4, 6, 8) + \Sigma d(10, 11, 12, 13, 14, 15)$$

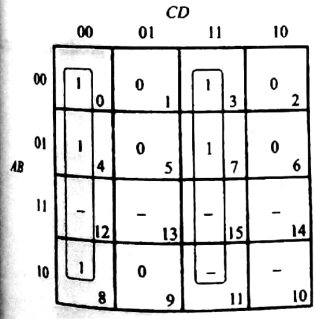
The Karnaugh maps are written for simplification of the expressions.



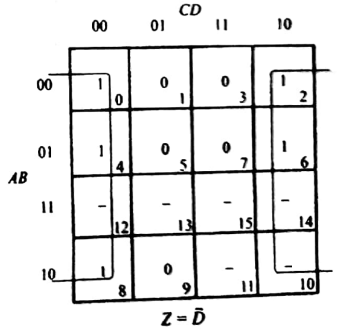
$W = BD + BC + A$



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



$Y = \bar{C}\bar{D} + CD = \overline{C \oplus D}$



$Z = \bar{D}$

The implementation is shown in Fig. 3.4.

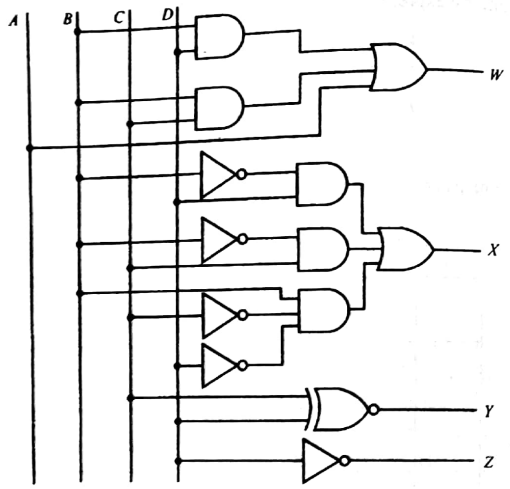


Fig. 3.4 Implementation of *BCD* to excess-3 converter

Design a combinational circuit that will multiply two 2-bit numbers.

Solution: Let the two 2-bit numbers be AB and CD and let $WXYZ$ be their product. The truth table is shown in Fig. 3.5.

Cell no.	A	B	C	D	W	X	Y	Z
0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	0
2	0	0	1	0	0	0	0	0
3	0	0	1	1	0	0	0	0
4	0	1	0	0	0	0	0	0
5	0	1	0	1	0	0	0	1
6	0	1	1	0	0	0	1	0
7	0	1	1	1	0	0	1	1
8	1	0	0	0	0	0	0	0
9	1	0	0	1	0	0	1	0
10	1	0	1	0	0	1	0	0
11	1	0	1	1	0	1	1	0
12	1	1	0	0	0	0	0	0
13	1	1	0	1	0	0	1	1
14	1	1	1	0	0	1	1	0
15	1	1	1	1	1	0	0	1

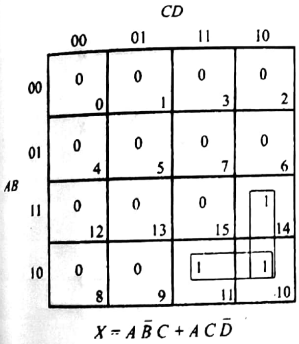
Fig. 3.5 Truth table of 2-bit multiplier

The minterm expressions are:

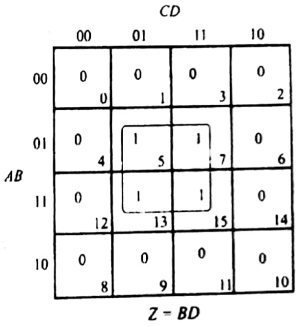
$$W = \Sigma(15)$$
$$X = \Sigma(10, 11, 14)$$
$$Y = \Sigma(6, 7, 9, 11, 13, 14)$$
$$Z = \Sigma(5, 7, 13, 15)$$

We have, $W = ABCD$

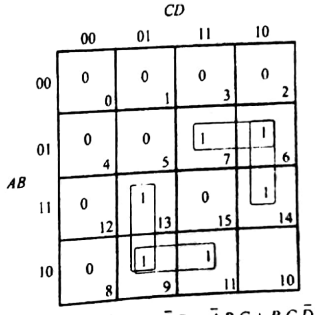
The maps for the other switching expressions are now drawn.



$X = A\bar{B}C + AC\bar{D}$



$Z = BD$



$Y = A\bar{C}D + A\bar{B}D + \bar{A}BC + BC\bar{D}$

The implementation is shown in Fig 3.6.

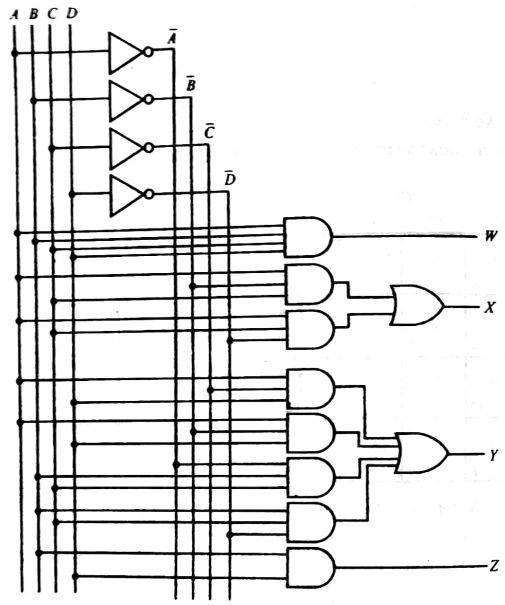
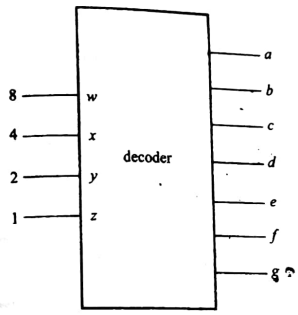


Fig. 3.6 Implementation of 2-bit multiplier

Design a combination circuit to drive a common cathode seven segment display with *BC*
inputs.

Solution: The block diagram and truth table for such a system is shown in Fig 3.7. Observe th
since the display is of common cathode type, a 1 is required to light up a particular segment.



(a) Block diagram

Cell no.	*w*	*x*	*y*	*z*	*a*	*b*	*c*	*d*	*e*	*f*	*g*
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	1	0	1	1
10	1	0	1	0	–	–	–	–	–	–	–
11	1	0	1	1	–	–	–	–	–	–	–
12	1	1	0	0	–	–	–	–	–	–	–
13	1	1	0	1	–	–	–	–	–	–	–
14	1	1	1	0	–	–	–	–	–	–	–
15	1	1	1	1	–	–	–	–	–	–	–

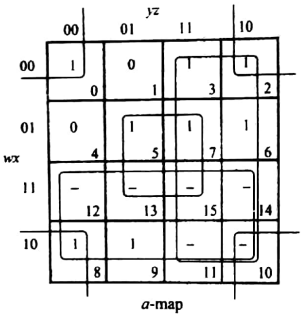
(b) Truth table of a *BCD* to 7-segment decoder

Fig. 3.7 (a) Block diagram and (b) Truth table of a *BCD* to 7-segment decoder

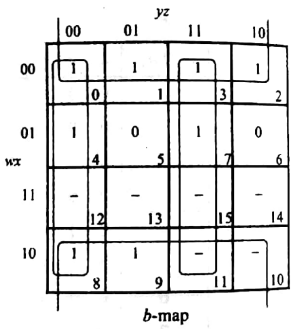
The minterm expressions are

$$a = \Sigma(0, 2, 3, 5, 6, 7, 8, 9) + \Sigma d\,(10 \text{ to } 15)$$
$$b = \Sigma(0, 1, 2, 3, 4, 7, 8, 9) + \Sigma d\,(10 \text{ to } 15)$$
$$c = \Sigma(0, 1, 3, 4, 5, 6, 7, 8, 9) + \Sigma d\,(10 \text{ to } 15)$$
$$d = \Sigma(0, 2, 3, 5, 6, 8, 9) + \Sigma d\,(10 \text{ to } 15)$$
$$e = \Sigma(0, 2, 6, 8) + \Sigma d\,(10 \text{ to } 15)$$
$$f = \Sigma(0, 4, 5, 6, 8, 9) + \Sigma d\,(10 \text{ to } 15)$$
$$g = \Sigma(2, 3, 4, 5, 6, 8, 9) + \Sigma d\,(10 \text{ to } 15)$$

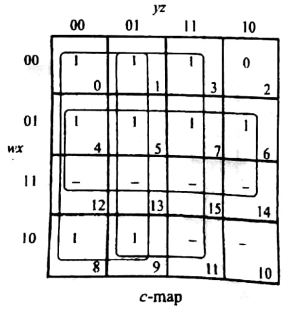
Let us now plot the seven maps



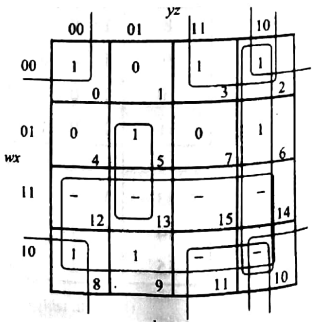
a-map



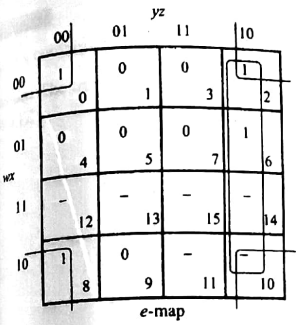
b-map



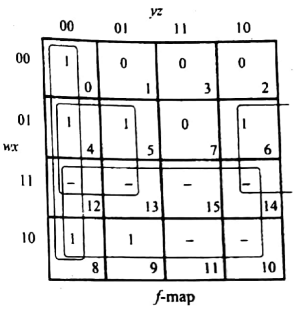
c-map



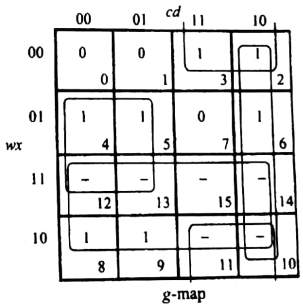
d-map



e-map



f-map



g-map

$$a = w + y + xz + \bar{x}\bar{z}$$
$$b = \bar{x} + \bar{y}\bar{z} + yz$$
$$c = x + \bar{y} + z$$
$$d = w + y\bar{z} + \bar{x}\bar{z} + \bar{x}y + x\bar{y}z$$
$$e = y\bar{z} + \bar{x}\bar{z}$$
$$f = w + \bar{y}\bar{z} + x\bar{y} + x\bar{z}$$
$$g = w + x\bar{y} + \bar{x}y + y\bar{z}$$

Design a combinational circuit to output a 1 when an illegal *BCD* code occurs.

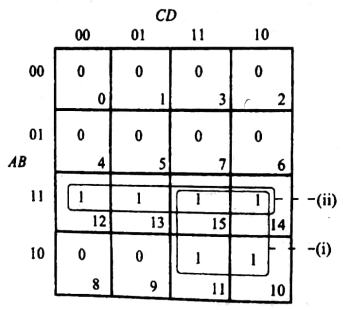
Solution: Let us write the truth table with output 1 for rows 10 to 15 as shown in Fig. 3.8.

Cell no.	A	B	C	D	Y
0	0	0	0	0	0
1	0	0	0	1	0
2	0	0	1	0	0
3	0	0	1	1	0
4	0	1	0	0	0
5	0	1	0	1	0
6	0	1	1	0	0
7	0	1	1	1	0
8	1	0	0	0	0
9	1	0	0	1	0
10	1	0	1	0	1
11	1	0	1	1	1
12	1	1	0	0	1
13	1	1	0	1	1
14	1	1	1	0	1
15	1	1	1	1	1

Fig. 3.8 Truth table for Example 3.5

$$Y = \Sigma (10, 11, 12, 13, 14, 15)$$

The Karnaugh map is shown below



$$Y = AC + AB$$

The implementation is shown in Fig 3.9.

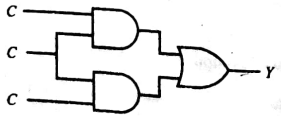


Fig 3.9 Implementation of illegal code defects

Design a combinational circuit to output the 2s complement of a 4-bit binary number.

Solution: The truth table is shown in Fig. 3.10. Let the number be $ABCD$ and its 2s complement be $WXYZ$.

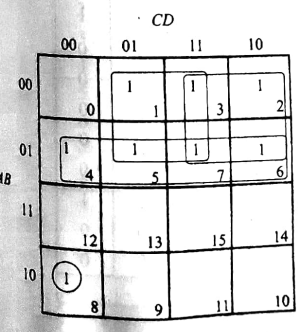
Cell no.	A	B	C	D	W	X	Y	Z
0	0	0	0	0	0	0	0	0
1	0	0	0	1	1	1	1	1
2	0	0	1	0	1	1	1	0
3	0	0	1	1	1	1	0	1
4	0	1	0	0	1	1	0	0
5	0	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	0
7	0	1	1	1	1	0	0	1
8	1	0	0	0	1	0	0	0
9	1	0	0	1	0	1	1	1
10	1	0	1	0	0	1	1	0
11	1	0	1	1	0	1	0	1
12	1	1	0	0	0	1	0	0
13	1	1	0	1	0	0	1	1
14	1	1	1	0	0	0	1	0
15	1	1	1	1	0	0	0	1

Fig. 3.10 Truth table of Binary to 2s complement conveter

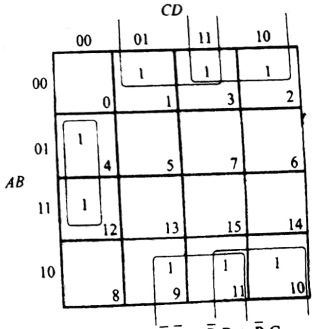
The minterm expressions are

$$W = \Sigma (1, 2, 3, 4, 5, 6, 7, 8)$$
$$X = \Sigma (1, 2, 3, 4, 9, 10, 11, 12)$$
$$Y = \Sigma (1, 2, 5, 6, 9, 10, 13, 14)$$
$$Z = \Sigma (1, 3, 5, 7, 9, 11, 13, 15)$$

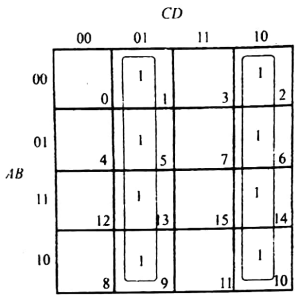
The Karnaugh maps can now be drawn.



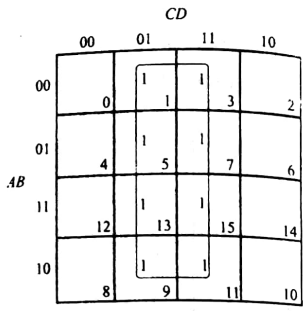
$$W = A\bar{B}\bar{C}\bar{D} + \bar{A}B + \bar{A}D + \bar{A}C$$



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



$Y = \bar{C}D + C\bar{D} = C \oplus D$



$Z = D$

Implementation is left as an exercise to the reader.

Example 3.7

Design a combinational circuit to drive a common anode seven segment display with a excess -3 *BCD* code.

Solution: Segments in a common anode seven segment display are lit by a 0. The segment table for display from 0 to 9 is shown in Fig. 3.11.

Display digit	*Segment outputs*						
	a	*b*	*c*	*d*	*e*	*f*	*g*
0	0	0	0	0	0	0	1
1	1	0	0	1	1	1	1
2	0	0	1	0	0	1	0
3	0	0	0	0	1	1	0
4	1	0	0	1	1	0	0
5	0	1	0	0	1	0	0
6	0	1	0	0	0	0	0
7	0	0	0	1	1	1	1
8	0	0	0	0	0	0	0
9	0	0	0	0	1	0	0

Fig. 3.11 Segment display table

The BCD to excess-3 conversion table is shown in Fig. 3.12.

Digit	*BCD*				*Excess-3*			
	A	*B*	*C*	*D*	*W*	*X*	*Y*	*Z*
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

Fig. 3.12 *BCD* to Excess-3 conversion

Now, using the tables in Fig. 3.11 and Fig. 3.12, the truth table for the Excess-3 to seven segment driver can be written as shown in Fig. 3.13.

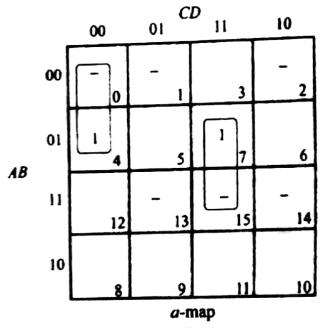
Digit	*Cell no.*	*Binary sequence*				*Seven segment code*						
		A	*B*	*C*	*D*	*a*	*b*	*c*	*d*	*e*	*f*	*g*
–	0	0	0	0	0	–	–	–	–	–	–	–
–	1	0	0	0	1	–	–	–	–	–	–	–
–	2	0	0	1	0	–	–	–	–	–	–	–
0	3	0	0	1	1	0	0	0	0	0	0	1
1	4	0	1	0	0	1	0	0	1	1	1	1
2	5	0	1	0	1	0	0	1	0	0	1	0
3	6	0	1	1	0	0	0	0	0	1	1	0
4	7	0	1	1	1	1	0	0	1	1	0	0
5	8	1	0	0	0	0	1	0	0	1	0	0
6	9	1	0	0	1	0	1	0	0	0	0	0
7	10	1	0	1	0	0	0	0	1	1	1	1
8	11	1	0	1	1	0	0	0	0	0	0	0
9	12	1	1	0	0	0	0	0	0	1	0	0
–	13	1	1	0	1	–	–	–	–	–	–	–
–	14	1	1	1	0	–	–	–	–	–	–	–
–	15	1	1	1	1	–	–	–	–	–	–	–

Fig. 3.13 Truth table for implementation

From the table in Fig. 3.13, The minterm expressions are,

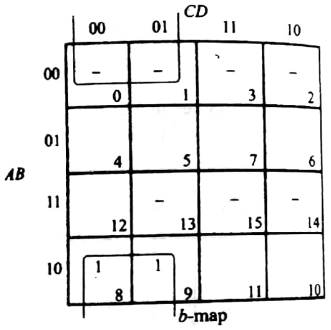
$$a = \Sigma(4, 7) + \Sigma d\,(0, 1, 2, 13, 14, 15)$$
$$b = \Sigma(8, 9) + \Sigma d\,(0, 1, 2, 13, 14, 15)$$
$$c = \Sigma(5) + \Sigma d\,(0, 1, 2, 13, 14, 15)$$
$$d = \Sigma(4, 7, 10) + \Sigma d\,(0, 1, 2, 13, 14, 15)$$
$$e = \Sigma(4, 6, 7, 8, 10, 12) + \Sigma d\,(0, 1, 2, 13, 14, 15)$$
$$f = \Sigma(4, 5, 6, 10) + \Sigma d\,(0, 1, 2, 13, 14, 15)$$
$$g = \Sigma(3, 4, 10) + \Sigma d\,(0, 1, 2, 13, 14, 15)$$

Karnaugh maps for the segment outputs can now be written from these minterm expressions



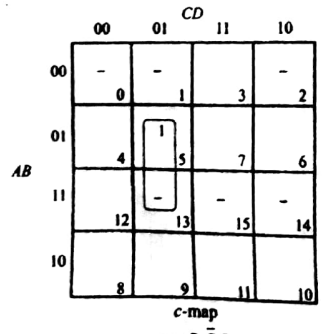
a-map

$a = \bar{A}\bar{C}\bar{D} + BCD$



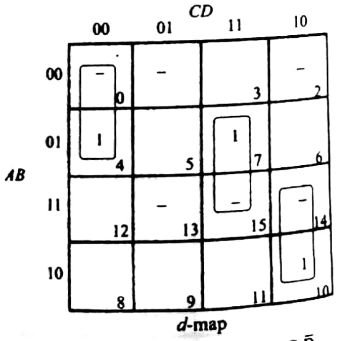
b-map

$b = \bar{B}\bar{C}$



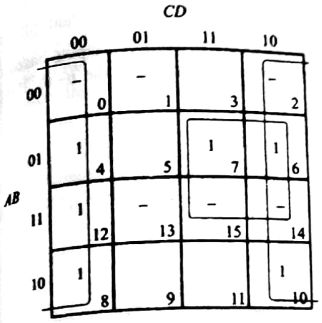
c-map

$c = B\bar{C}D$



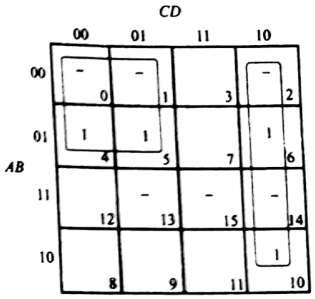
d-map

$d = \bar{A}\bar{C}\bar{D} + BCD + AC\bar{D}$



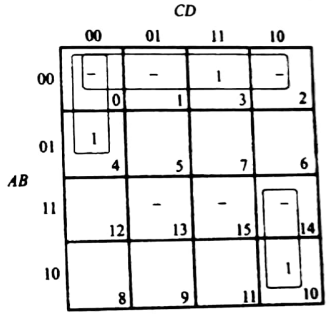
e-map

$e = BC + \bar{D}$



f-map

$f = \bar{A}\bar{C} + C\bar{D}$



g-map

$g = \bar{A}\bar{C}\bar{D} + AC\bar{D} + \bar{A}\bar{B}$

The expressions can be implemented easily using gates.

Exercise 3.1

Design a system to accept a five bit input and generate an active low output whenever the number of 1s are odd.

Exercise 3.2

Design a system to accept a five bit input and generate an active low output whenever the input code is either divisible by 3 or 7.

Exercise 3.3

Design a system to perform the following conversions on 4 bit numbers:

(a) Binary to gray (b) Excess-4 to *BCD*

3.2 Decoders

A decoder is a device which when activated, selects one of its possible 2^n outputs based on n-bit digit. There are several types of decoders, the most common being the n- to 2^n type decoders as shown in Fig 3.14.

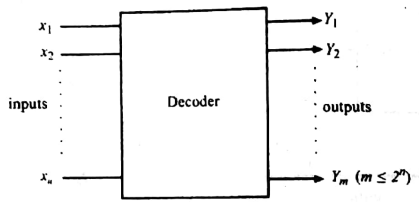


Fig. 3.14 General representation of a decoder

Let us now look at a few typical commercially available decoders.

3.2.1 2 to 4 line Decoders

Let us look at the logic of a 2 to 4 line decoder in its simplest form before looking at the commercial configurations. Figure 3.15 shows the truth table.

Inputs		*Outputs*			
A	B	Y_0	Y_1	Y_2	Y_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

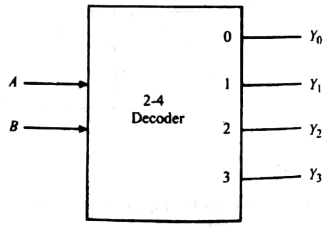


Fig 3.15 Truth table and block schematic of a 2 to 4 line decoder

Observe that there are 2 input lines and 2^2 output lines.
It is easy to see from Fig. 3.15 that

$Y_0 = \overline{A}\,\overline{B}$
$Y_1 = \overline{A}\,B$
$Y_2 = A\,\overline{B}$
$Y_3 = A\,B$,

signifying that all the possible minterms of a two variable expression are available at the output of a 2 line to 4 line decoder. Does this not sound a bell? Yes, these outputs can be used to realize functions of two variables.
Consider the two variable function

$$f = \overline{A} + B$$

The Karnaugh map for this can be plotted as shown in Fig. 3.16 entering 1 in every cell corresponding to A and every cell corresponding to B.

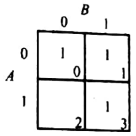


Fig. 3.16 Karnaugh map for $f = \overline{A} + B$

$$\therefore \qquad f = \Sigma(0, 1, 3)$$

This can be implemented using a 2 to 4 line decoder as shown in Fig 3.17.

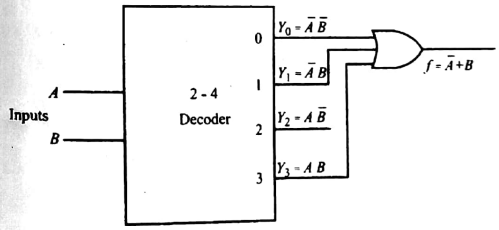


Fig. 3.17 Implementation of $f = \overline{A} + B$ using a 2-4 line decoder

Besides the n input lines and 2^n output lines these decoders also have enable inputs, some active high and some active low which act to select the chip to make it functional as well as to cascade to form decoders with large number of inputs. Now 74139 is a dual 2 to 4 decoder integrated circuit (IC) with an enable input for each of the decoders as shown in Fig. 3.18.

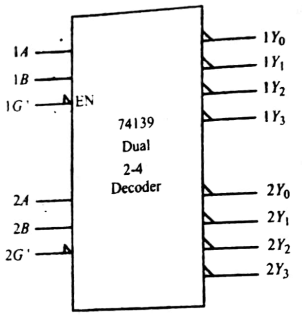


Fig. 3.18 Dual 2-4 line decoder

Fig. 3.18 shows the IEEE symbol for the 74139 IC. The wedges at the output indicate a bubble output or active low outputs. This means that generally the outputs are high and go low when activated by an appropriate input code. The wedge at the input indicates that that section of the device is activated by a low on that input. These operations are captured in the functional table of 74139 shown in Fig. 3.19.

Inputs			*Outputs*			
Enable	*Select*					
G'	B	A	Y_0	Y_1	Y_2	Y_3
H	×	×	H	H	H	H
L	L	L	L	H	H	H
L	L	H	H	L	H	H
L	H	L	H	H	L	H
L	H	H	H	H	H	L

Fig. 3.19 Functional table of 74139

When the G' line is high and the outputs are high irrespective of the inputs and that section of the device is said to be disabled. The logic diagram for one of the 2-4 decoders in the IC is shown in Fig. 3.20.

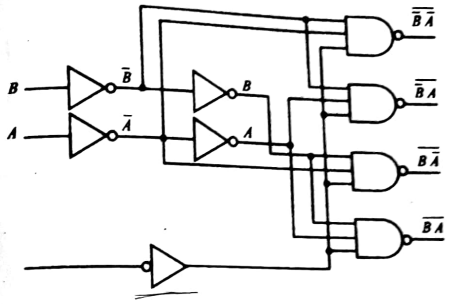


Fig. 3.20 Logic diagram of one section of 74139

The enable input can be used to configure a 3-8 decoder using a dual 2-4 decoder as shown in Fig. 3.21.

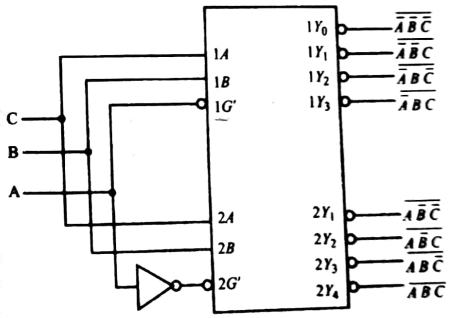


Fig. 3.21 Dual 2-4 decoder configured as 3-8 decoder

Realize the following Boolean functions using 74139.

(a) $f_1(a, b) = \Sigma(0, 2)$
(b) $f_2(a, b, c) = \Sigma(1, 3, 6, 7)$

Solution
(a) $f_1(a, b) = \Sigma(0, 2)$

Since only two variables are involved, one section of dual 2-4 decoder 74139 is sufficient for the implementation as shown in Fig. 3.22.

$$f_1(a, b) = \Sigma(0, 2) = \bar{a}\,\bar{b} + a\bar{b}$$

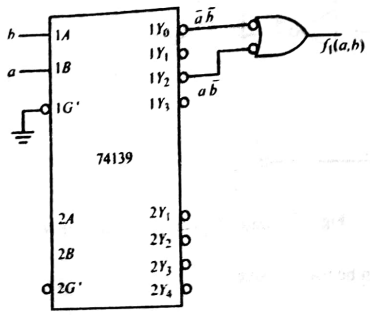


Fig. 3.22 Implementation of Example 3.8(a)

(b) $f_2(a, b, c) = \Sigma(1, 3, 6, 7)$

The enable line is used to cascade the dual section to form a 3-8 decoder as shown in Fig.3.23.

Now, a brief note on the gate symbol used in Fig. 3.22.

Consider the truth table of a NAND gate as shown.

a	b	y
0	0	1
0	1	1
1	0	1
1	1	0

We can read the table as output being high when input a or input b or both are high or as output is low if input a and input b are low. The gate symbol shown in Fig. 3.22 fits the first description though it is simply a 2-input NAND gate.

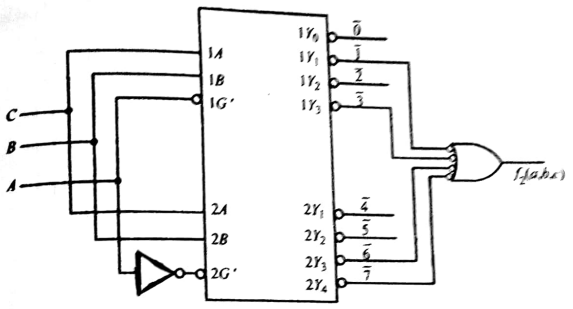


Fig. 3.23 Implementation of Example 3.8(b)

3.2.2 3 to 8 Line Decoders

As in the case of 2 to 4 line decoders, 3 to 8 lines decoders are also available in IC form as 74138. The logic diagram can be developed on similar lines as that for a 2 to 4 line decoder and is left as an exercise to the reader.

The functional symbol of 74138 is shown in Fig. 3.24.

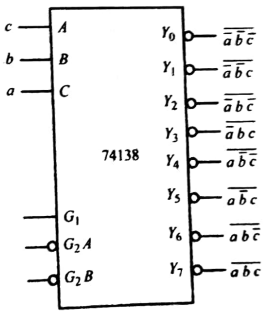


Fig. 3.24 Functional symbol of 74138

The device functions as a decoder only when $G_1 = 1$ and $G_2A = G_2B = 0$ and under other conditions all the outputs will be high. The functional table of 74138 is shown in Fig. 3.25.

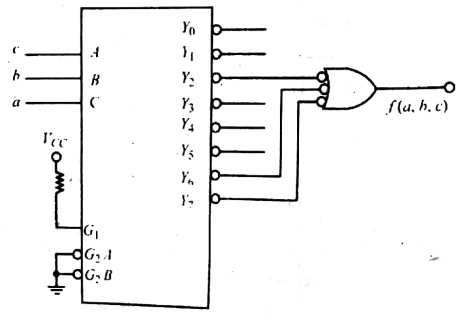
Inputs						Outputs							
Enable			Select										
G_1	G_2A	G_2B	C	B	A	Y_0	Y_1	Y_2	Y_3	Y_4	Y_5	Y_6	Y_7
0	×	×	×	×	×	1	1	1	1	1	1	1	1
×	1	×	×	×	×	1	1	1	1	1	1	1	1
×	×	1	×	×	×	1	1	1	1	1	1	1	1
1	0	0	0	0	0	0	1	1	1	1	1	1	1
1	0	0	0	0	1	1	0	1	1	1	1	1	1
1	0	0	0	1	0	1	1	0	1	1	1	1	1
1	0	0	0	1	1	1	1	1	0	1	1	1	1
1	0	0	1	0	0	1	1	1	1	0	1	1	1
1	0	0	1	0	1	1	1	1	1	1	0	1	1
1	0	0	1	1	0	1	1	1	1	1	1	0	1
1	0	0	1	1	1	1	1	1	1	1	1	1	0

Fig 3.25 Functional table of 74138

A 3 to 8 decoder can be used straight away for the implementation of logic expressions with three variables as illustrated in the examples below.

Implement the following function using 74138.

$f(a, b, c) = \Sigma\,(2, 6, 7)$



Suggest a scheme to construct a 4 to 16 line decoder using 74138.

Solution: The enable inputs are effectively used to achieve this. The most significant bit which is 0 for the first half of the table and 1 for the remaining rows is used to enable the correct 74138 in order to configure a 4 to 16 line decoder as shown in Fig. 3.26.

Let *abcd* be the four inputs, with *d* being the least significant bit.

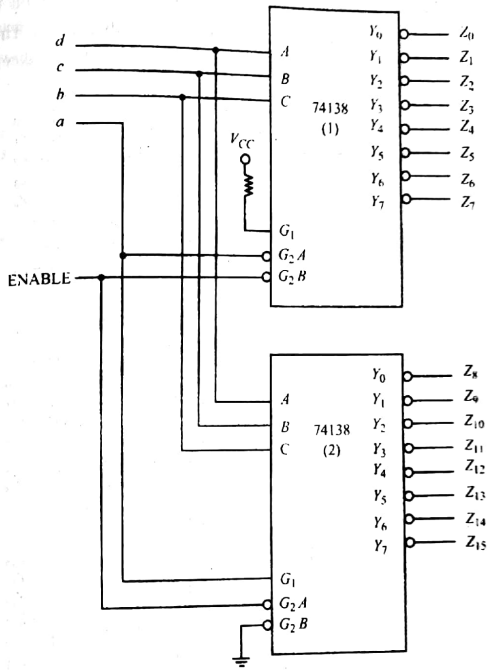


Fig 3.26 4-6 Decoder using two 3-8 decoders

G_1 of 74138 (1) and G_2B of 74138 (2) are permanently enabled. G_2B of 74138 (1) and G_2A of 74138 (2) are used together as enable inputs. When the ENABLE input is 1, both the devices are disabled and all the outputs are at 1. When the ENABLE input is 0, the devices are enabled.

Observe that for the upper half of the four input table, the most significant bit is 0. This enables 74138 (1) and disables 74138 (2). Hence for inputs *abcd* = 0000 to 0111, $a = 0$ and one of the outputs Y_0 to Y_7 of 74138 (1) designated Z_0 to Z_7 becomes 0 depending on the input code.

For inputs *abcd* = 1000 to 1111, $a = 1$ and 74138 (1) is disabled, while 74138 (2) is enabled. Thus for inputs from *abcd* = 1000 to 1111 one of the outputs Y_0 to Y_7 of 74138 (2) designated Z_8 to Z_{15} becomes 0 depending on the input code. Thereby two 74138 ICs can be configured to function as a 4 to 16 line decoder.

Configure a 5 to 32 decoder using four 3 to 8 decoder ICs and 1 to 4 decoder IC.

Solution: Pins G_1 and G_2B of all the 3-8 decoders are permanently enabled. The outputs of the 2-4 decoder are used to enable one of the 3-8 decoders. The arrangement is shown in Fig. 3.27.

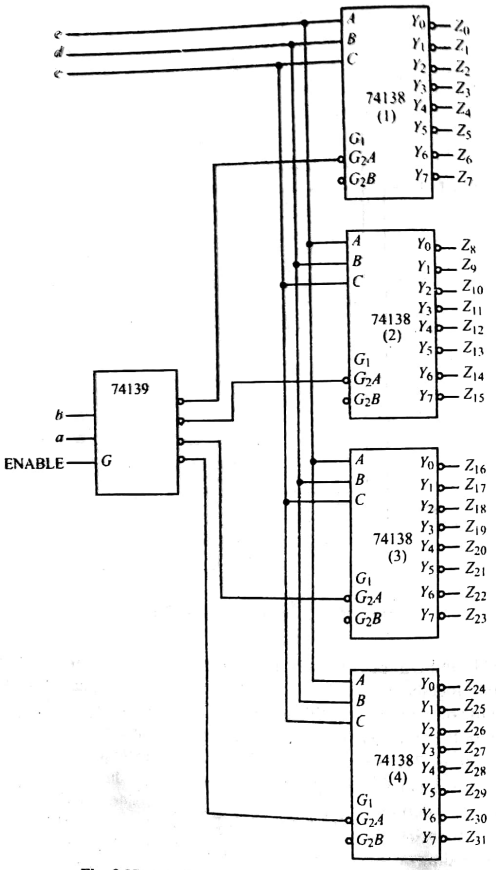


Fig. 3.27 5-32 decoder from 3-8 and 2-4 decoders

Implement the multiple function $f_1(a, b, c, d) = \Sigma(0, 4, 8, 10, 14, 15)$ and $f_2(a, b, c, d) = \Sigma(3, 7, 9, 13)$ using two 74138 (3-8) decoders

Solution: The implementation is shown in Fig. 3.28.

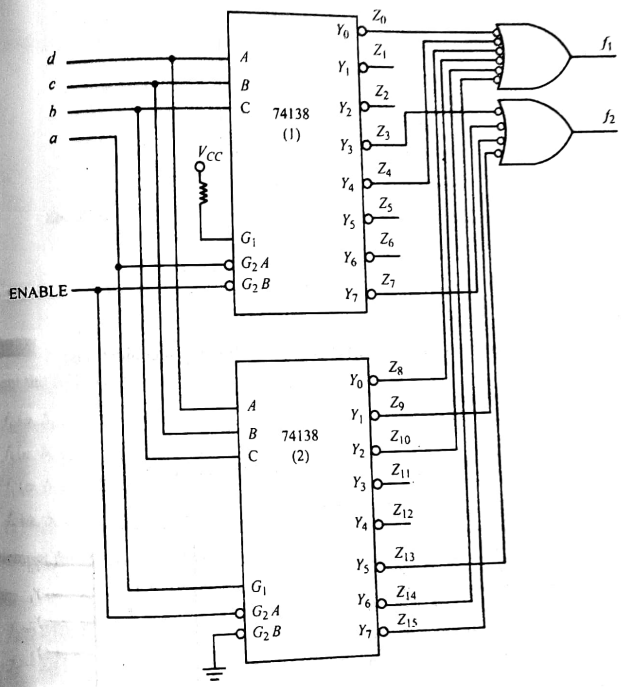


Fig. 3.28 Implementation of Example 3.12

Implement $f(x, y, z) = \Pi\ (1, 2, 4, 6)$ on a 3-8 decoder (74138).

Solution: The implementation is shown in Fig. 3.29.

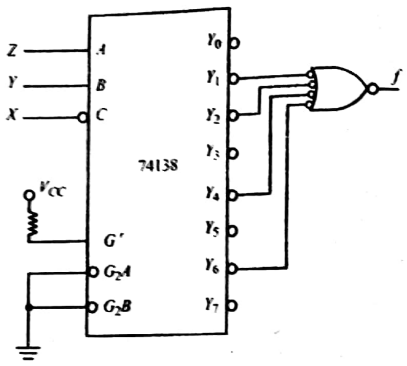


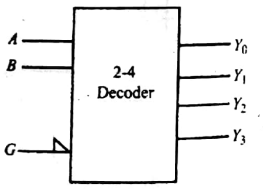
Fig. 3.29 Implementation of Example 3.12

Develop the logic diagram of a 2 to 4 decoder with the following specifications:
(i) active low enable input
(ii) active high encoded outputs.
Draw the IEEE symbol.

Solution: The truth table and IEEE symbol are shown in Fig. 3.30.

Inputs			*Outputs*			
Enable	*Select*					
G	*B*	*A*	Y_0	Y_1	Y_2	Y_3
H	×	×	0	0	0	0
L	0	0	1	0	0	0
L	0	1	0	1	0	0
L	1	0	0	0	1	0
L	1	1	0	0	0	1

(a) Truth table



(b) IEEE symbol

Fig 3.30 Truth table and logic symbol for Example 3.13

The combinational circuit to realize the logic of Fig. 3.30 (a) is shown in Fig. 3.31. The minterm generated is also indicated.

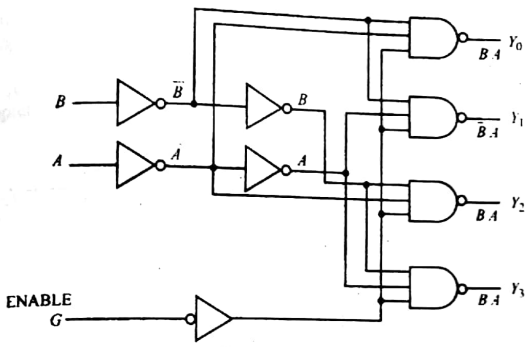


Fig. 3.31 Logic diagram of 2-4 decoder of Example 3.13

Implement the following function pairs using 74138 decoder.

a. $f_1(a, b, c) = \Sigma(0, 1, 5, 6, 7)$
   $f_2(a, b, c) = \Sigma(1, 2, 3, 6, 7)$
b. $f_1(a, b, c) = \Sigma(0, 2, 4)$
   $f_2(a, b, c) = \Sigma(1, 2, 4, 5, 7)$

Implement for minimum gate inputs.

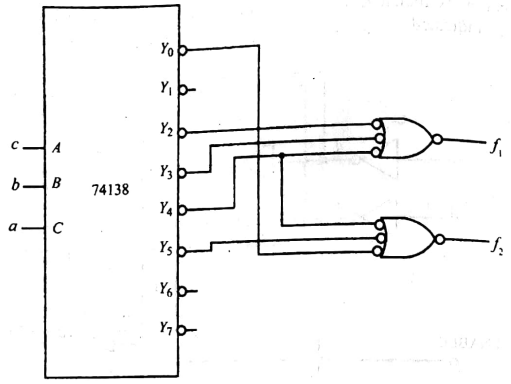
Solution:

a. $f_1 = \Sigma(0, 1, 5, 6, 7)$
   $f_2 = \Sigma(1, 2, 3, 6, 7)$

Here we need to implement $\bar{f}_1$ and $\bar{f}_2$ minimize the number of gate inputs.

$\therefore$ $\bar{f}_1 = \Sigma(2, 3, 4)$

$\bar{f}_2 = \Sigma(0, 4, 5)$



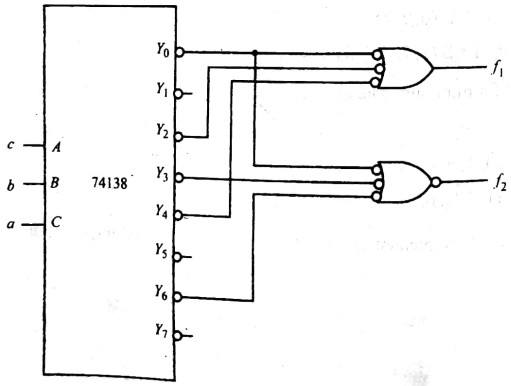
b. $f_1 = \Sigma\,(0, 2, 4)$

$f_2 = \Sigma\,(1, 2, 4, 5, 7)$

f_1 can be directly implemented

f_2 is realized by implementing $\overline{f}_2$ and inverting the output.

$$\overline{f}_2 = \Sigma\,(0, 3, 6)$$



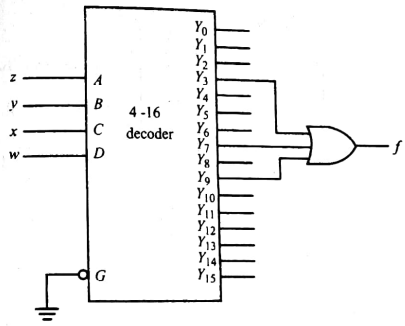
Implement the following with a suitable decoder with active low enable input and active high output:

(a) $f(w, x, y, z) = \Sigma\,(3, 7, 9)$

(b) $g(a, b, c) = \Pi\,(2, 4, 7)$

Solution:

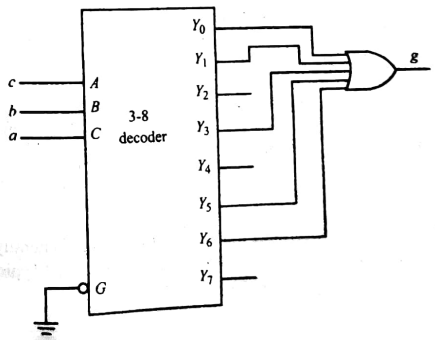
(a) The implementation is shown below with a 4-16 decoder



(b) The implementation is shown below with a 3-8 decoder.

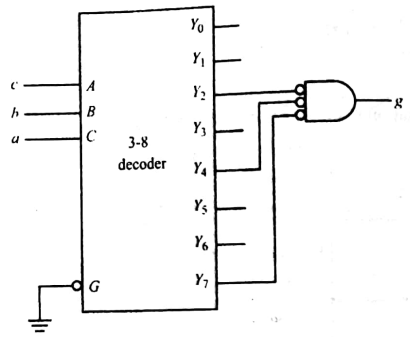
$$g = \Pi\,(2, 4, 7)$$

$$\therefore \quad g = \Sigma\,(0, 1, 3, 5, 6)$$



or

it can be directly implemented as shown below.



◆ **Exercise 3.4**

Implement the following using 2-4 decoders:

(a) $f_1 = \Sigma\,(0, 4, 5)$
(b) $f_1 = \Pi\,(6, 7)$

◆ **Exercise 3.5**

Implement the following using 3-8 decoders:

(a) $f_1 = \Sigma\,(2, 3, 5, 6)$
(b) $f_2 = \Sigma\,(4, 10, 12)$
(c) $f_3 = \Pi\,(5, 7, 13, 15)$

◆ **Exercise 3.6**

Suggest a scheme for 8-256 decoders using 74138.

3.3 Encoders

Encoders too like decoders that convert one binary code to another. Generally in an encoder, the number of input lines are much more than the number of output lines. Typically, if there are 2^n input lines, there could be n output lines as shown in Fig. 3.32.

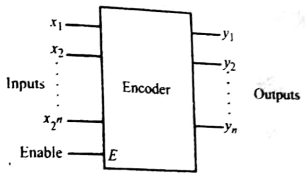


Fig. 3.32 A general encoder

Some examples of encoders are 4 to 2 line encoders, 8 to 3 line encoders, decimal to *BCD* encoders and so on.

The truth table of a 8 to 3 line encoder is shown in Fig. 3.33.

Inputs								*Outputs*		
x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	y_2	y_1	y_0
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

Fig. 3.33 Truth table of 8 to 3 line encoder

The Boolean expression for the outputs can be written looking at the table in Fig. 3.33 as

$$y_2 = x_4 + x_5 + x_6 + x_7$$
$$y_1 = x_2 + x_3 + x_6 + x_7$$
$$y_0 = x_1 + x_3 + x_5 + x_7$$

The block diagram and implementation are shown in Fig. 3.34.

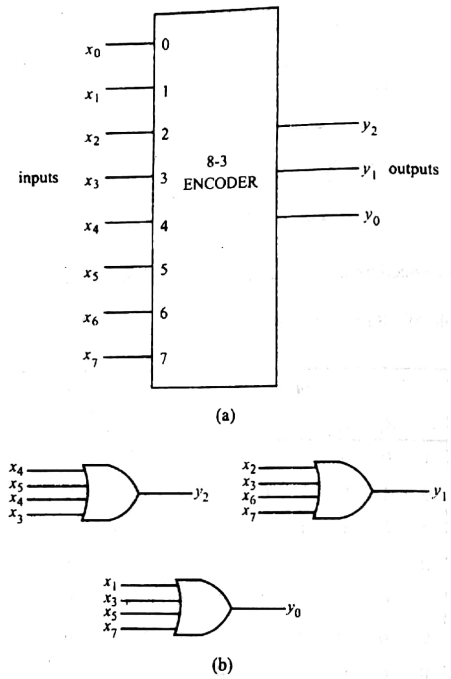


Fig. 3.34 (a) Block diagram of a 8 to 3 line encoder (b) Implementation of a 8 to 3 line encoder

There are some problems associated with this implementation. It is possible that when more than one inputs are at logic 1 there may be an error in the output code. For example, if $x_3 = x_4 = 1$ then $y_2 y_1 y_0 = 111$, whereas this output should have resulted for only $x_7 = 1$. The other problem is that the output is 000 when all the inputs are at logic 0 as well as when $x_0 = 1$. The schematic in Fig. 3.34 does not distinguish between these two conditions of inputs.

These problems can be overcome by the priority encoder with a validity indicator. The truth table of a 8 to 3 line priority encoder is shown in Fig. 3.35.

Row no.	Inputs								Outputs			
	x_0	x_1	x_2	x_3	x_4	x_5	x_6	x_7	y_2	y_1	y_0	Valid
0	0	0	0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	0	0	0	0	0	0	0	1
2	×	1	0	0	0	0	0	0	0	0	1	1
3	×	×	1	0	0	0	0	0	0	1	0	1
4	×	×	×	1	0	0	0	0	0	1	1	1
5	×	×	×	×	1	0	0	0	1	0	0	1
6	×	×	×	×	×	1	0	0	1	0	1	1
7	×	×	×	×	×	×	1	0	1	1	0	1
8	×	×	×	×	×	×	×	1	1	1	1	1

Fig. 3.35 Truth table of 8 to 3 line priority encoder

Observe in the table that higher order bits have higher priority. For example, if x_3 is at logic 1, then irrespective of the logic values of x_0, x_1 and x_2 the output code will be 011. The valid output being 1 indicates that the input code is valid. The outputs at row 0 and row 1 are now distinguished by the logic level of the 'Valid' output.

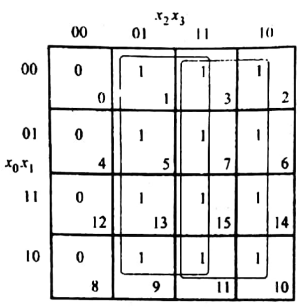
The truth table as the one shown in Fig. 3.35 is referred to as the condensed truth table as all don't care conditions of the input are not explicitly listed.

Write the condensed truth table for a 4 to 2 line priority encoder with a valid output where the highest priority is given to the highest bit position or input with highest index and obtain the minimal sum expressions for the outputs.

Solution: The truth table of a 4 to 2 line priority encoder with priority to higher order bits is shown below.

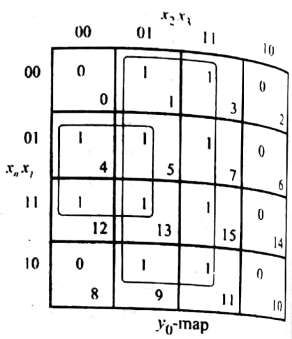
Cell no.	x_0	x_1	x_2	x_3	y_1	y_0	Valid
0	0	0	0	0	0	0	0
8	1	0	0	0	0	0	1
4, 12	×	1	0	0	0	1	1
2, 6, 10, 14	×	×	1	0	1	0	1
1, 3, 5, 7, 9 11, 13, 15	×	×	×	1	1	1	1

Let us now construct the Karnaugh maps for the outputs.



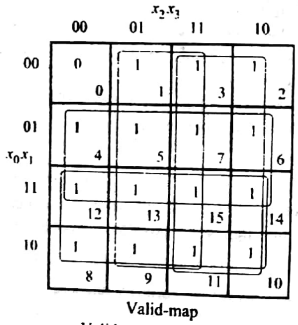
y_1-map

$y_1 = x_2 + x_3$



y_0-map

$y_0 = x_3 + x_1\bar{x}_2$



Valid-map

Valid $= x_0 + x_1 + x_2 + x_3$

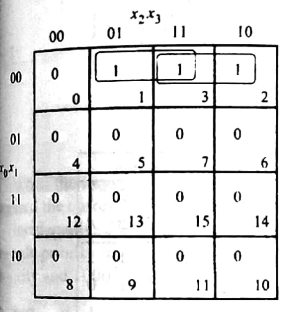
EXAMPLE 3.17

Repeat Example 3.16 assigning highest priority to the least significant input or input with lowest index.

***Solution*:** The truth table is shown below.

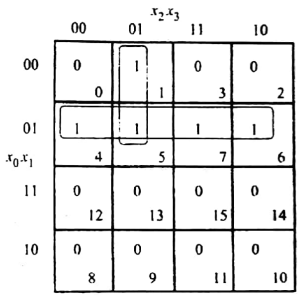
Cell no.	x_0	x_1	x_2	x_3	y_1	y_0	*Valid*
0	0	0	0	0	0	0	0
8, 9, 10, 11, 12, 13, 14, 15	1	×	×	×	0	0	1
4, 5, 6, 7	0	1	×	×	0	1	1
2, 3	0	0	1	×	1	0	1
1	0	0	0	1	1	1	1

The Karnaugh maps for the outputs are shown below.



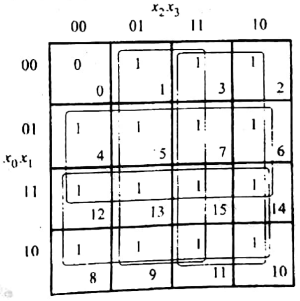
y_1-map

$y_1 = \bar{x}_0\,\bar{x}_1\,x_2 + \bar{x}_0\,\bar{x}_1 x_3$



y_0-map

$y_0 = \bar{x}_0 x_1 + \bar{x}_0 x_2 x_3$

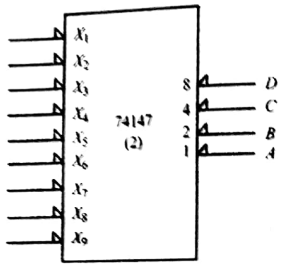


Valid-map

Valid $= x_0 + x_1 + x_2 + x_3$

3.3.1 Decimal to BCD Encoder

A decimal to *BCD* encoder produces the 8421 output code depending on which of its nine input lines are active. The 74147 is a decimal to *BCD* encoder with both inputs and outputs active low. The logic symbol and truth table of 74147 is shown in Fig. 3.36.



(a) Logic symbol

	Input									Output			
										8	4	2	1
Decimal	X_1	X_2	X_3	X_4	X_5	X_6	X_7	X_8	X_9	D	C	B	A
0	1	1	1	1	1	1	1	1	1	1	1	1	1
1	0	1	1	1	1	1	1	1	1	1	1	1	0
2	×	0	1	1	1	1	1	1	1	1	1	0	1
3	×	×	0	1	1	1	1	1	1	1	1	0	0
4	×	×	×	0	1	1	1	1	1	1	0	1	1
5	×	×	×	×	0	1	1	1	1	1	0	1	0
6	×	×	×	×	×	0	1	1	1	1	0	0	1
7	×	×	×	×	×	×	0	1	1	1	0	0	0
8	×	×	×	×	×	×	×	0	1	0	1	1	1
9	×	×	×	×	×	×	×	1	0	0	1	1	0

(b) Truth table

Fig 3.36 Logic symbol and truth table of 74147

For example, when the input 7 goes low, the outputs C, B and A go low. Such devices are generally used to interface a numeric keypad to a microcontroller or a computer.

Fig. 3.37 shows a keypad interface configured using 74147.

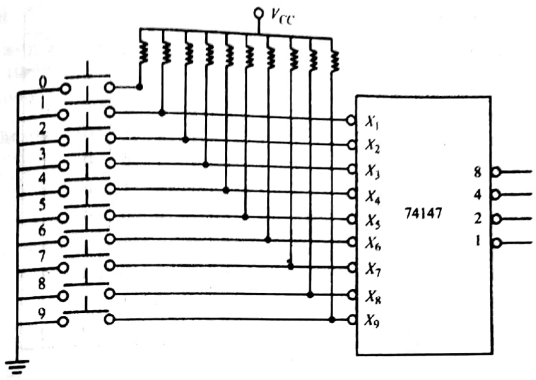


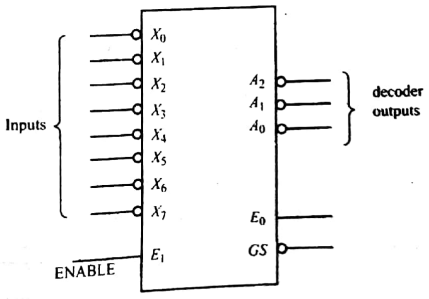
Fig 3.37 Keypad interface using 74147

When the keys are open, all the inputs are at 1 and all the outputs are at 1. When any key is pressed the corresponding active low code appear at the output.

Incidentally, the 74147 is also referred to as a priority encoder. This is because the device awards priority to the highest order input. For example if X_3, X_5 and X_7 are pressed, X_7 is given priority and 1000 is output. If X_1, X_2 and X_5 are pressed, X_5 is given priority and 1010 is output.

3.3.2 8 line to 3 Line Priority Encoder

A 8 line to 3 line priority encoder can be used to set priority to eight events and is available as 74148 in IC form. The logic design and truth table is shown in Fig. 3.38.



(a) Logic diagram

Inputs								Outputs				
E_1	X_0	X_1	X_2	X_3	X_4	X_5	X_6	X_7	A_2	A_1	A_0	GS	E_0
1	×	×	×	×	×	×	×	×	1	1	1	1	1
0	1	1	1	1	1	1	1	1	1	1	1	1	0
0	0	1	1	1	1	1	1	1	1	1	1	0	1
0	×	0	1	1	1	1	1	1	1	1	0	0	1
0	×	×	1	1	1	1	1	1	1	0	1	0	1
0	×	×	×	0	1	1	1	1	1	0	0	0	1
0	×	×	×	×	0	1	1	1	0	1	1	0	1
0	×	×	×	×	×	0	1	1	0	1	0	0	1
0	×	×	×	×	×	×	0	1	0	0	1	0	1
0	×	×	×	×	×	×	×	0	0	0	0	0	1

(b) Truth table

Fig. 3.38 Logic design and truth table of 8 to 3 priority encoder

When the enable input E_1, is active (0), the output lines are high and enable output is high. The enable output E_0 being low indicates that none of the priority inputs have been activated. GS goes low to indicate that one of the inputs have been activated.

Suggest a scheme to transmit 8 line data over 3 lines.

Solution: This could be done using a 8-3 line encoder such as 74148 cascaded with a 3-8 line decoder such as 74138 as shown in Fig. 3.39.

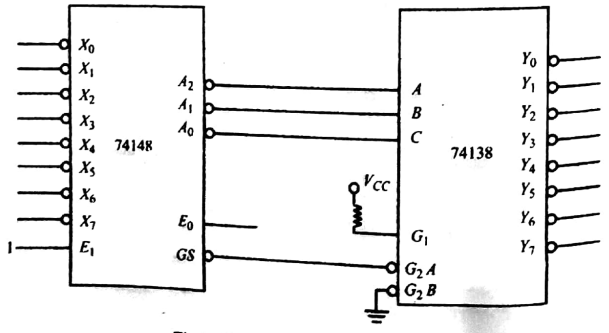


Fig. 3.39 Scheme for Example 3.18

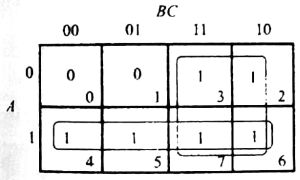
Design a priority encoder for a system with three inputs, with the middle bit with highest priority encoding to 10, the MSB with the next priority encoding to 11, while the LSB with the least priority encoding to 01.

Solution: The truth table of the system is shown in Fig. 3.40.

Cell no.	Inputs			Outputs	
	A	B	C	Y	Z
0	0	0	0	0	0
1	0	0	1	0	1
2	0	1	0	1	0
3	0	1	1	1	0
4	1	0	0	1	1
5	1	0	1	1	1
6	1	1	0	1	0
7	1	1	1	1	0

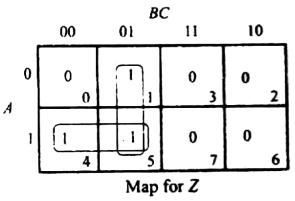
Fig. 3.40 Truth table for Example 3.19

Let us draw the Karnaugh maps for Y and Z.



Map for Y

$Y = A + B$



Map for Z

$Z = A\bar{B} + \bar{B}C$

The encoders is shown in Fig. 3.41.

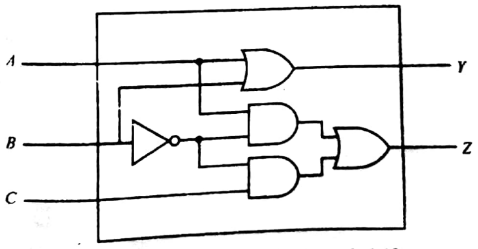


Fig. 3.41 Implementation for Example 3.19

◆ Exercise 3.7

Design a five line to three line priority encoder with the following specifications:

(i) Bit 3 with the highest priority to encode to 111
(ii) Bit 1 with the next priority to encode to 100
(iii) Bit 4 with the next priority to encode to 001
(iv) Bit 0 with the next priority to encode to 011 and
(v) Bit 2 with the least priority to encode to 101

◆ Exercise 3.8

Design a 16-4 line HEX keypad encoder to output 0000 when key 0 is pressed, 0001 when ke is pressed, 1010 when key *A* is pressed, 1011 when key *B* is pressed and so on.