4

Advanced Combinational Logic Circuits

Chapter Outline

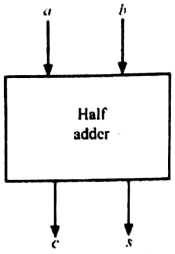
This chapter presents a few more advanced combinational circuits such as arithmetic circuits, multiplexers and comparators. It begins with a discussion on adders and subtractors along with their design. Multiplexers are presented with a host of illustrative examples on their cascading and on their amazing applications in implementation of Boolean functions. The chapter concludes with a treatment on comparators.

4.1 Arithmetic Circuits

Configuration of various types of adders and subtractors and their implementation is presented in this section.

4.1.1 Binary Half Adder

A binary half-adder adds two binary bits and generates a sum bit and a carry bit. The block diagram and the truth table are shown in Fig. 4.1, where a and b are the inputs, s is the sum output while c is the carry output.



(a) Block diagram

Cell no.	a	b	c	s
0	0	0	0	0
1	0	1	0	1
2	1	0	0	1
3	1	1	1	0

(b) Truth table

Fig. 4.1 Half adder

From the truth table in Fig. 4.1(a),

$$s = \Sigma(1, 2)$$

and

$$c = \Sigma(3)$$

It is easy to see that

$$s = \bar{a}b + a\bar{b} = a \oplus b$$

and

$$c = ab$$

The implementation of s and c is shown in Fig. 4.2.

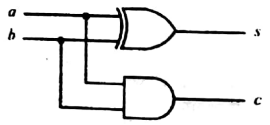
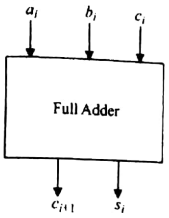


Fig. 4.2 Implementation of half adder

4.1.2 Binary Full Adder

A half adder adds two binary bits and generates the sum bit and the carry bit. In order to cascade several one-bit adders to configure multiple bit adders there must be a provision to add the carry

from the previous stage. Such an adder is called a *full adder*. The block diagram is shown in Fig. 4.3.



$$\begin{array}{r} c_i \\ +\ a_i \\ +\ b_i \\ \hline c_{i+1}\ s_i \end{array}$$

Fig. 4.3 Block diagram of Full adder

The subscript 'i' signifies that it is the i^{th} adder in a cascade of full adders. For example, if these adders are cascaded to form an adder to add two 4-bit numbers, the least significant adder would add c_0, a_0 and b_0 and give the result s_0 and c_1, while the most significant adder would add c_3, a_3 and b_3 and generate s_3 and c_4 as its outputs.

The truth table of the full adder is shown in Fig. 4.4.

Cell no.	a_i	b_i	c_i	c_{i+1}	s_i
0	0	0	0	0	0
1	0	0	1	0	1
2	0	1	0	0	1
3	0	1	1	1	0
4	1	0	0	0	1
5	1	0	1	1	0
6	1	1	0	1	0
7	1	1	1	1	1

Fig. 4.4 Truth table of full adder

$$s_i = \Sigma(1, 2, 4, 7)$$

and

$$c_{i+1} = \Sigma(3, 5, 6, 7)$$

Let us write the Karnaugh map for the two outputs in order to obtain a minimal Boolean expression for the sum and carry outputs as shown in Fig. 4.5.

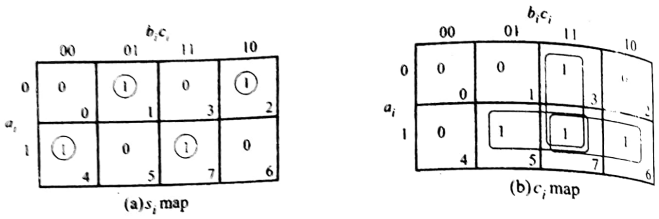


Fig. 4.5 Karnaugh map of Full adder outputs

From the subcubes in the s_i map, we get

$$s_i = \bar{a}_i \bar{b}_i c_i + \bar{a}_i b_i \bar{c}_i + a_i + \bar{b}_i \bar{c}_i + a_i b_i c_i$$
$$= \bar{a}_i (\bar{b}_i c_i + b_i \bar{c}_i) + a_i (\bar{b}_i \bar{c}_i + b_i \bar{c}_i) + a_i$$
$$= \bar{a}_i (b_i \oplus c_i) + a_i \overline{(b_i \oplus c_i)}$$
$$s_i = a_i \oplus b_i \oplus c_i \quad (4.1)$$

From the subcubes in the c_{i+1} map, we get

$$c_{i+1} = a_i b_i + b_i c_i + a_i c_i \quad (4.2)$$

The implementation is shown in Fig. 4.6

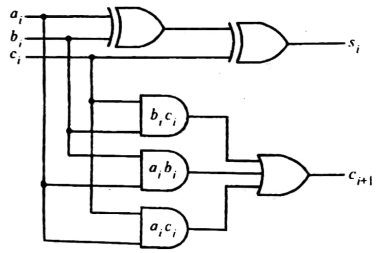


Fig. 4.6 Implementation of Full adder

More often there arises a need to add two 4 or 8-bit numbers. Let us look at methods to add two 4-bit numbers. One approach would be to construct a 8-variable truth table which would have 256 rows. Imagine the magnitude when the number of bits increase. Addition of two eight bit numbers would result in a truth table with 2^{16} rows. The other approach is to cascade full adders

to configure multiple-bit adders. Four full adders can be cascaded to configure a 4-bit adder. The arrangement is shown in Fig. 4.7.

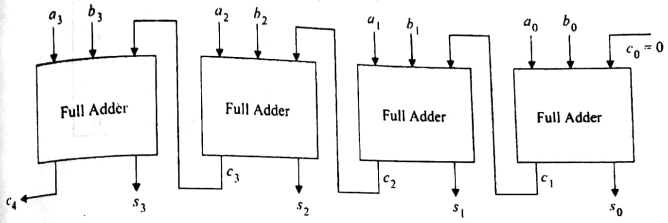


Fig. 4.7 Four-bit parallel adder using full adders

The arrangement in Fig. 4.7 is called a parallel adder since all the input bits are applied at the same time to the circuit. However, it is easy to see that the outputs do not appear at the same time. The correct result of the addition of a_3 and b_3 will occur only when the correct value of c_3 appears. This can happen only after the addition of all the previous stages have correctly taken place. The correct carry input to the n^{th} stage will only appear after the addition is completed in all previous n-1 stages.

This phenomenon arises due to the propagation delay in the gates in each stage. The carry output of each stage will appear only after the propagation delay associated with that stage. The carry thus ripples through each stage and hence this adder is referred to as a *ripple adder*.

Note that two half adders can be used to form a full adder as shown in Fig. 4.8.

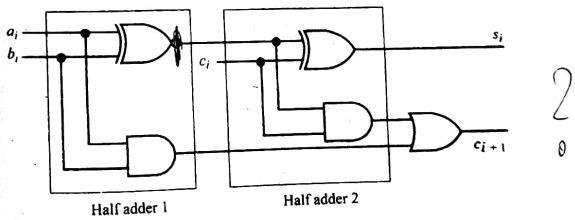
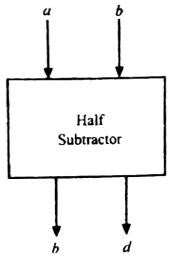


Fig 4.8 Full adder using two half adders

4.1.3 Binary Half Subtractor

A binary half subtractor subtracts one binary bit from another and generates the difference and the borrow outputs. The block diagram and the truth table of a half subtractor is shown in Fig. 4.9, where bit b is to be subtracted from bit a, d is the difference and b is the borrow.



(a) Block diagram

Cell no.	a	b	b	d
0	0	0	0	0
1	0	1	1	1
2	1	0	0	1
3	1	1	0	0

(b) Truth table

Fig. 4.9 Half subtractor

From the truth table of Fig. 4.9 (b),

$$d = \Sigma(1, 2)$$

and $$b = \Sigma(1)$$

It is easy to see that,

$$d = a\bar{b} + \bar{a}b = a \oplus b$$

and $$b = \bar{a}b$$

The implementation is shown in Fig. 4.10.

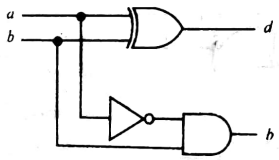


Fig. 4.10 Implementation of full adder

4.1.4 Binary Subtractors

Binary subtractors can be configured on the same lines as adders. The block diagram of a full subtractor is shown in Fig. 4.11.

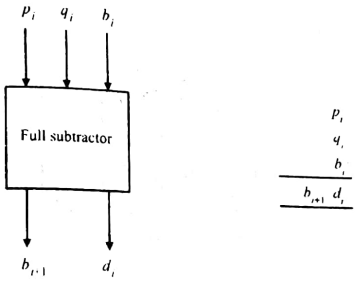


Fig. 4.11 Block diagram of Full subtractor

The truth table of the full subtractor is shown in Fig. 4.12.

Cell no.	p_i	q_i	b_i	b_{i+1}	d_i
0	0	0	0	0	0
1	0	0	1	1	1
2	0	1	0	1	1
3	0	1	1	1	0
4	1	0	0	0	1
5	1	0	1	0	0
6	1	1	0	0	0
7	1	1	1	1	1

Fig. 4.12 Truth table of full subtractor

$$d_i = \Sigma(1, 2, 4, 7)$$

$$b_{i+1} = \Sigma(1, 2, 3, 7)$$

Comparing Tables in Fig. 4.4 and 4.12, we see that the s_i and d_i columns are identical.

$$\therefore \quad d_i = p_i \oplus q_i \oplus b_i \tag{4.3}$$

Let us plot the Karnaugh map as shown in Fig. 4.13 for b_{i+1}.

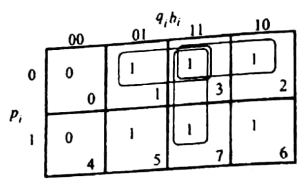


Fig. 4.13 Karnaugh map for b_{i+1}

From the subcubes, we can write

$$b_{i+1} = \bar{p}_i b_i + \bar{p}_i q_i + q_i b_i \qquad (4.4)$$

The implementation of a full subtractor is shown in Fig. 4.14.

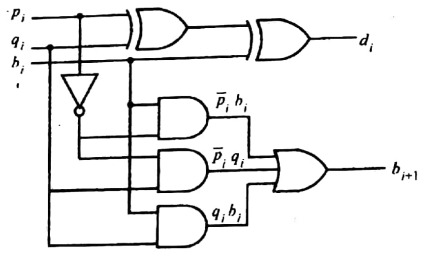


Fig. 4.14 Implementation of full subtractor

Such full subtractors can be cascaded to configure multiple-bit subtractors. A 4-bit parallel ripple subtractor is shown in Fig. 4.15.

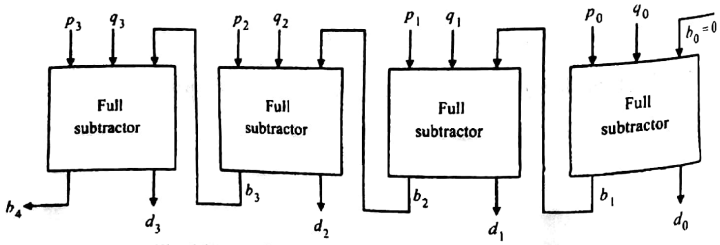


Fig. 4.15 Four-bit parallel subtractor using full subtractors

These cascaded forms are also ripple subtractors and the difference outputs do not appear simultaneously with the application of the inputs. The outputs at any stage will be valid only after the borrow of the previous stage has been generated.

We know that the subtraction $(p_i - q_i)$ can be performed through the following addition. p_i + (2s complement of q_i). Fig. 4.16 illustrates this with an example.

(a) Decimal	(b) Binary	(c) 2s complement	(d) Subtraction by addition
5	0 1 0 1	2s complement	0 1 0 1
− 3	− 0 0 1 1	of 0011 = 1101	+ 1 1 0 1
2	0 0 1 0	ignore–(1)	0 0 1 0

Fig. 4.16 Illustration of subtraction through addition

This concept is used to alter the inputs of the 4-bit full adder shown in Fig. 4.7 to perform subtraction. The arrangement is shown in Fig. 4.17. Inverters are used to obtain the 1s complement of the subtrahend and the carry at bit position 0; i.e., c_0 is set to 1 in order to add 1 to the 1s complement to obtain the 2s complement.

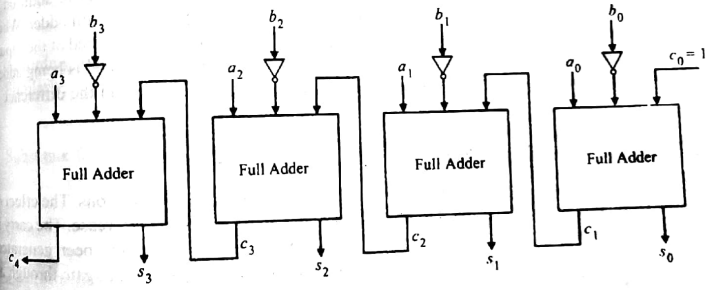


Fig. 4.17 Parallel adder performing parallel subtraction

An exclusive OR-gate can be used as a true complement gate depending on the logic level at one of its inputs designated as the control input. This property can be used to perform parallel addition as well as subtraction using the configuration shown in Fig. 4.18.

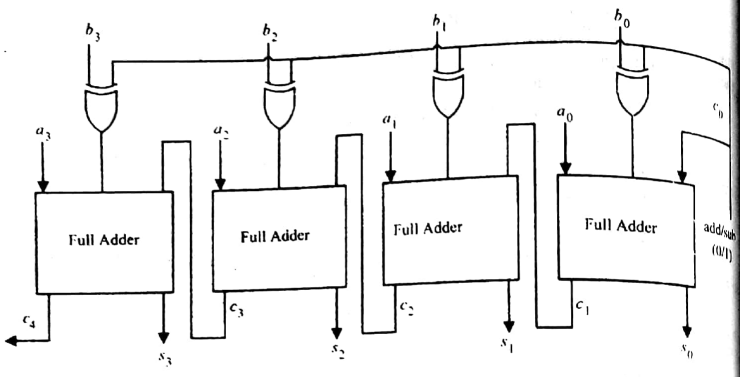


Fig. 4.18 Parallel adder and subtractor

The carry input c_0 to the bit-0 stage acts as a control input. When c_0 is set to 0, addition is performed because b_3, b_2, b_1 and b_0 appear uncomplemented at the inputs of the full adder. When c_0 is set to 1 subtraction is performed because b_3, b_2, b_1 and b_0 appear complemented at the inputs of the full adder. Further, since c_0 is set to 1, the 2s complement of the subtrahend is being added to the minuend. The outputs of the arrangement represent the sum if $c_0 = 0$ and the difference if $c_0 = 1$.

4.1.5 Look Ahead Carry Adder

The parallel adders and subtractors seen earlier are essentially ripple configurations. The effect of propagation delay would be more and more prominent as the number of bits increase. The carry at any given stage would be available only after the carry of the previous stage has been generated. The carry generated by the full adder in the least significant bit stage must propagate through all the intermediate adders till it reaches the most significant bit adder. Observe from Equation 4.1 and 4.2 that apart from the input bits to be added, the sum and carry out of any stage depends on the carry output of the previous stage. The carry out of the previous stage depends on the carry out of the stage prior to that and so on. If we could ensure that the sum and carry output of any stage is not dependent on the results of any previous stages, then the ripple effect is eliminated and speed of addition remarkably increases.

Consider the carry Equation 4.2

$$c_{i+1} = a_i b_i + b_i c_i + a_i c_i$$
$$c_{i+1} = a_i b_i + c_i (a_i + b_i) \quad (4.5)$$

The term $a_i b_i$ relates to the carry formed at the i^{th} stage and is referred to as the carry generate function g_i.

i.e., $$g_i = a_i b_i \quad (4.6)$$

The term $(a_i + b_i)$ relates to the carry c_i generated at the previous stage and thus $(a_i + b_i)$ is referred to as the carry propagate function p_i.

i.e., $$p_i = a_i + b_i \quad (4.7)$$

Observe that both p_i and g_i are functions of only the parallel inputs a_i and b_i. Comparing Equations 4.5, 4.6 and 4.7 we can write

$$c_{i+1} = g_i + p_i c_i \quad (4.8)$$

Now, let us look at the carry generated at every stage of the 4-bit parallel adder. Setting $i = 0$ in Equation 4.8, we get

$$c_1 = g_0 + p_0 c_0 \quad (4.9)$$

and setting $i = 1$, in Equation 4.8, we get

$$c_2 = g_1 + p_1 c_1$$

substitute for c_1 from Equation 4.9

$$\therefore \quad c_2 = g_1 + p_1 (g_0 + p_0 c_0)$$
$$c_2 = g_1 + p_1 g_0 + p_1 p_0 c_0 \quad (4.10)$$

According to Equations 4.9 and 4.10, c_1 and c_2 are functions of only the parallel inputs and c_0. Similarly,

$$c_3 = g_2 + p_2 c_2$$

Substitute for c_2 from Equation 4.10

$$c_3 = g_2 + p_2 (g_1 + p_1 g_0 + p_1 p_0 c_0)$$
$$c_3 = g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 c_0 \quad (4.11)$$

and

$$c_3 = g_2 + p_2 (g_1 + p_1 g_0 + p_1 p_0 c_0)$$
$$c_4 = g_3 + p_3 c_3$$
$$= g_3 + p_3 (g_2 + p_2 g_1 + p_2 p_1 g_0 + p_2 p_1 p_0 c_0)$$
$$c_4 = g_3 + p_3 g_2 + p_3 p_2 g_1 + p_3 p_2 p_1 g_0 + p_3 p_2 p_1 p_0 c_0 \quad (4.12)$$

and so on.

Equation 4.11 and 4.12 also indicate that c_3 and c_4 are functions of only the parallel inputs and c_0. The Boolean expression for c_1, c_2, c_3 etc., can themselves be implemented using gates and the carry required at each stage of the parallel adder can be made available simultaneously, thereby increasing the speed of addition. The look ahead carry generators are available in IC form in

several bit sizes. A four-bit fast parallel adder with lookahead carry is shown in Fig. 4.19. The lookahead carry block is an implementation of Equations 4.9, 4.10, 4.11 and 4.12 whereas the adder blocks are an implementation of Equations 4.1, 4.6 and 4.7.

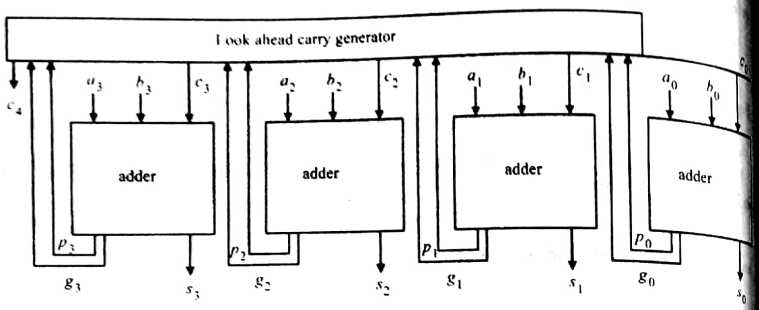


Fig. 4.19 Four-bit parallel fast look ahead carry adder

The implementation of the adder blocks of Fig. 4.19 is shown in Fig. 4.20.

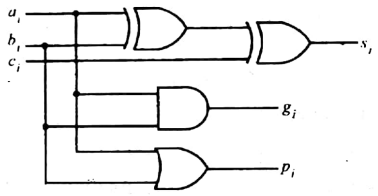
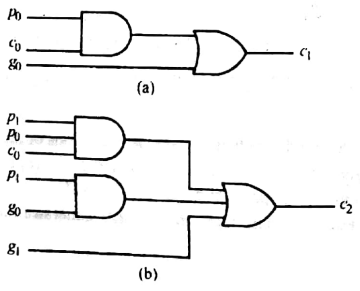


Fig. 4.20 Adder block implementation

The four-bit lookahead carry generator can be implemented as shown in Fig. 4.21.



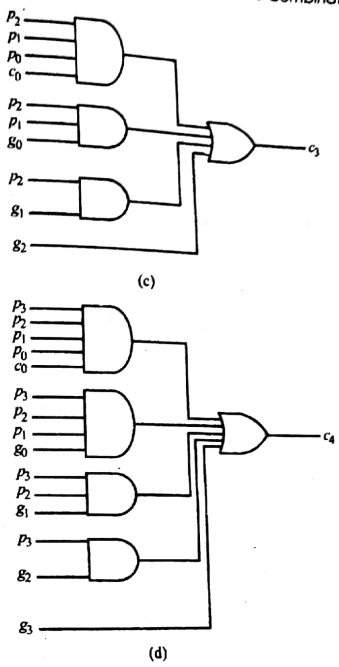


Fig. 4.21 Four-bit lookahead carry generator

We can see from Fig. 4.20 and 4.21 that g_i and p_i pass through four gating levels before the sum is produced at any stage.

They are,

one level of gating to produce g_i and p_i;

two levels of gating to produce c_i; and

one level exclusive-OR gating to produce s_i.

Design a full adder using a decoder.

Solution: The binary minterms equations of a full adder from section 4.1.2 are

$$s_i = \Sigma(1, 2, 4, 7)$$

and

$$c_{i+1} = \Sigma(3, 5, 6, 7)$$

These can be implemented using a 3-8 decoder 74138 as shown in Fig. 4.22.

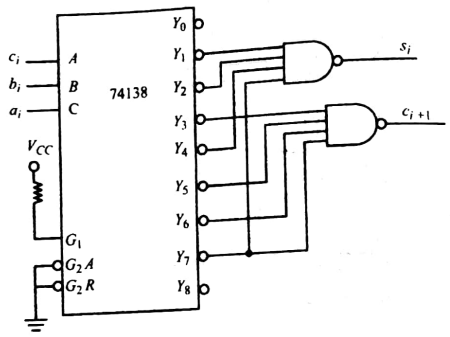


Fig. 4.22 Decoder implementation of full adder

Example 4.1

Design a full subtractor using a decoder.

Solution: The minterm expressions for a binary full subtractor from section 4.1.4 are

$$d_i = \Sigma(1, 2, 4, 7)$$

and

$$b_{i+1} = \Sigma(1, 2, 3, 7)$$

These can be implemented using a 3-8 decoders 74138 as shown in Fig. 4.23.

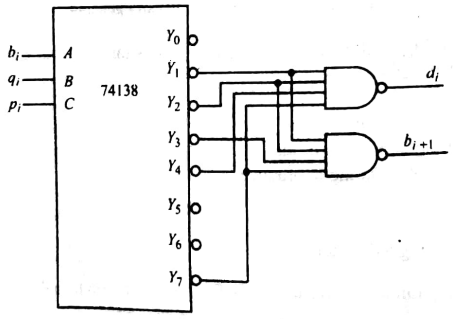


Fig. 4.23 Decoder implementation of full subtractor

Exercise 4.1

Implement a half adder and a half subtractor on a dual 2-4 decoder, 74139 with a control input x such that when $x = 0$ addition is performed and when $x = 1$ subtraction is performed.

Exercise 4.2

Show how a half adder could be used as a exclusive OR circuit.

4.2 Multiplexers

A multiplexer places data at one of its 2^n input lines selected by an n-bit address, on its output line. Just as in decoders, multiplexers also have one or more enable inputs. The symbol of a 8 to 1 multiplexer or 8 to 1 MUX as they are generally referred to is shown in Fig. 4.24.

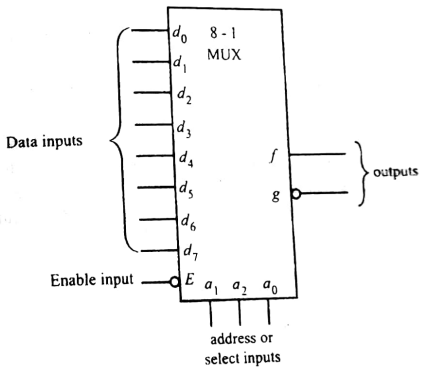
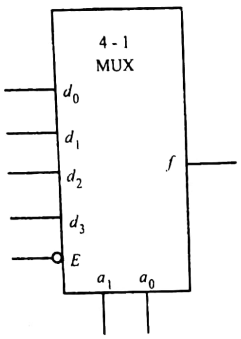


Fig. 4.24 Symbol of 8 to 1 MUX

The enable input is also called the strobe input.

4.2.1 4-1 Multiplexers

Let us now look at the design of a 4 to 1 MUX. The symbol and truth table of a 4 to 1 MUX is shown in Fig. 4.25.



Row no.	E	a_1	a_0	d_0	d_1	d_2	d_3	f
0	1	×	×	×	×	×	×	0
1	0	0	0	0	×	×	×	0
2	0	0	0	1	×	×	×	1
3	0	0	1	×	0	×	×	0
4	0	0	1	×	1	×	×	1
5	0	1	0	×	×	0	×	0
6	0	1	0	×	×	1	×	1
7	0	1	1	×	×	×	0	0
8	0	1	1	×	×	×	1	1

Fig. 4.25 Symbol and truth table of a 4 to 1 MUX

As seen in row 0, when $E = 1$, irrespective of other inputs $f = 0$. When $E = 0$, the data at the addressed or selected input appears at the output. For example, in row 3 the select inputs are 01 which selects data line d_1. The data on d_1 appears at the output. If $d_1 = 0$, $f = 0$ (row 3) and if $d_1 = 1, f = 1$ (row 4). This functional description can be represented in a functional table shown in Fig. 4.26.

E	a_1	a_0	f
1	×	×	0
0	0	0	d_0
0	0	1	d_1
0	1	0	d_2
0	1	1	d_3

Fig. 4.26 Functional Table of 4 to 1 MUX

It is now easy to write an expression for the output as follows.

$$f = \left(d_0 \bar{a}_1 \bar{a}_0 + d_1 \bar{a}_0 a_1 + d_2 a_1 \bar{a}_0 + d_3 a_1 a_0\right) \bar{E} \qquad (4.13)$$

Suppose, the MUX is enabled ($E = 0$) and the select lines are $a_1 a_0 = 01$ then the above equation becomes

$$f = d_1$$

If $d_1 = 0, f = 0$ and if $d_1 = 1, f = 1$.

The 4 to 1 MUX with reference to Equation 4.16 is shown in Fig. 4.27.

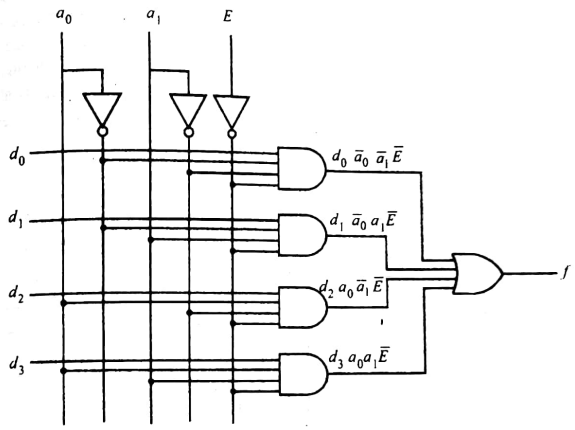


Fig. 4.27 Implementation of 4 to 1 MUX

Several multiplexers of the 2^n to 1 line are available in IC packages. These include 2 to 1 MUX, 4 to 1 MUX, 8 to 1 MUX and 16 to 1 MUX. Large number of inputs can be accommodated by cascading available multiplexers.

Such 4 to 1 multiplexers are packaged as dual 4 to 1 MUX in 74153 as shown in Fig. 4.28

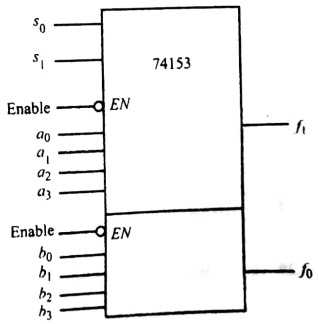


Fig. 4.28 Dual 4 to 1 MUX

Configure a 16 to 1 MUX using 4 to 1 MUX.

Solution: The arrangement is shown in Fig. 4.29.

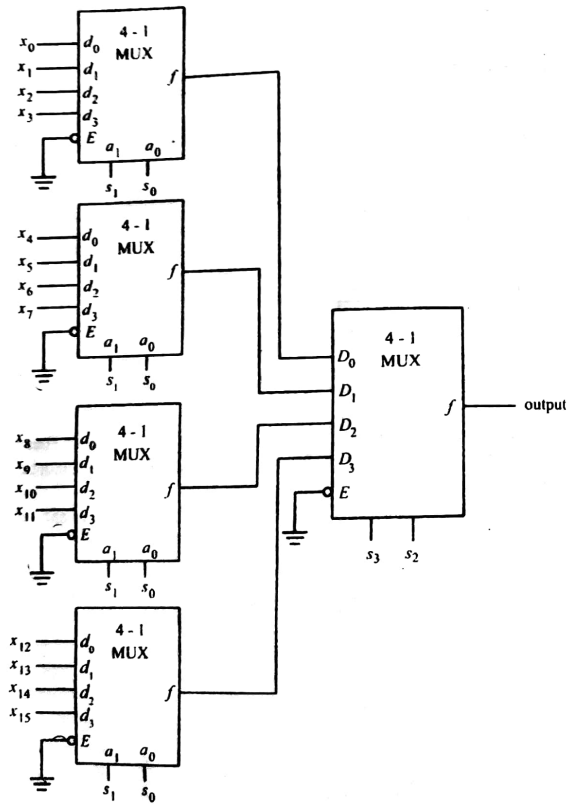


Fig. 4.29 16 to 1 MUX configured using 4 to 1 MUX

Let us say that address is $s_3 s_2 s_1 s_0 = 1101$ then $s_1 s_0 = 01$ places x_1 at D_0, x_5 at D_1, x_9 at D_2 and x_{13} at D_3. The address $s_3 s_2 = 11$ places D_3 *i.e.*, x_{13} at the final output.

As the word multiplexer implies, the basic application is in routing data on several lines on to one line. A communication channel can be realized by using a multiplexer at the source to route data on several lines on to one line and by using a decoder/demuliplexer at the destination to route the data from the single line channel to the desired output line. This single channel is often referred to as a one-bit bus. A typical arrangement is shown in Fig. 4.30.

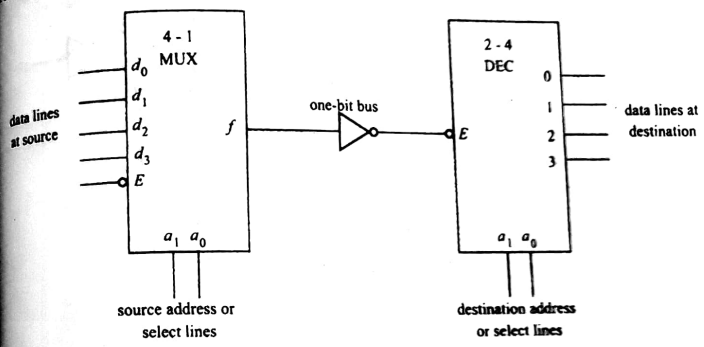


Fig. 4.30 MUX-DEMUX channel

Configure a 4-bit MUX using 74153.

Solution: One such implementation is shown in Fig. 4.31.

Let the four 4-bit inputs be

$d_3 d_2 d_1 d_0$,

$c_3 c_2 c_1 c_0$,

$b_3 b_2 b_1 b_0$ and

$a_3 a_2 a_1 a_0$

Let the select lines be $s_1 s_0$

When the select line $s_1 s_0$ are 01, d_1, c_1, b_1, a_1 will appear on f_3, f_2, f_1, f_0 respectively.

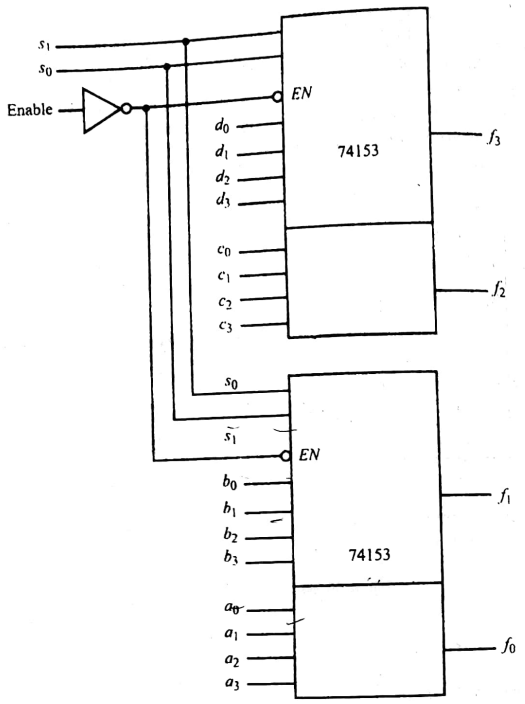


Fig. 4.31 Configuration for Example 4.4

4.2.2 8 to 1 Multiplexer

A 8 to 1 multiplexer will have 3 select lines, 8 data lines, output lines (true as well as complement some times) and enable lines. The 74151 is a 8 to 1 MUX as shown in Fig. 4.32.

The logic diagram of such a 8 to 1 MUX is shown in Fig. 4.33.

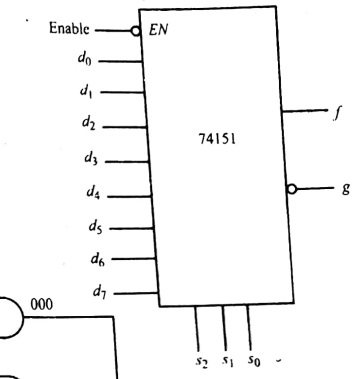


Fig. 4.32 An IC 8-1 MUX

Fig. 4.33 Logic diagram of 8 to 1 MUX

The truth table of 8 to 1 MUX, 74151 is shown in Fig. 4.34.

Inputs				Outputs	
Select			Enable		
s_2	s_1	s_0	EN	f	g
X	X	X	1	0	1
0	0	0	0	d_0	$\bar{d}_0$
0	0	1	0	d_1	$\bar{d}_1$
0	1	0	0	d_2	$\bar{d}_2$
0	1	1	0	d_3	$\bar{d}_3$
1	0	0	0	d_4	$\bar{d}_4$
1	0	1	0	d_5	$\bar{d}_5$
1	1	0	0	d_6	$\bar{d}_6$
1	1	1	0	d_7	$\bar{d}_7$

Fig. 4.34 Truth table of 8 to 1 MUX

Configure a 32 to 1 MUX using 74150, a 16 to 1 MUX.

Solution: 74150 is a 16 to 1 MUX with 16 input data lines, 4 address lines, one active low input enable line and one active low output line.

Two 74150 ICs can be used to configure a 32 to 1 MUX as shown in Fig. 4.35.

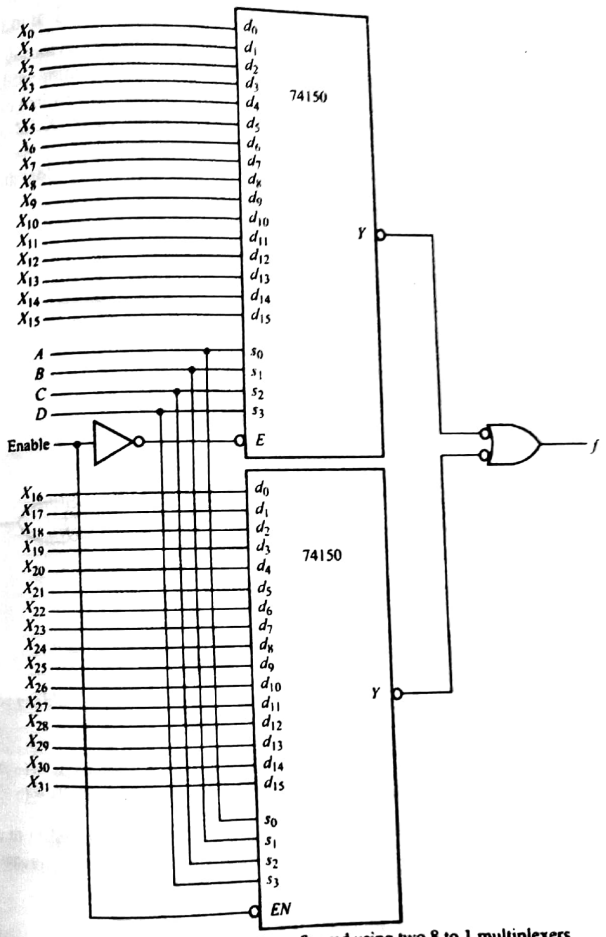


Fig. 4.35 A 32 to 1 MUX configured using two 8 to 1 multiplexers

Configure a 32 to 1 MUX using four 8 to 1 multiplexers.

Solution: A 2 to 4 decoder can be used to enable each of the four 8 to 1 MUX as shown in Fig. 4.36.

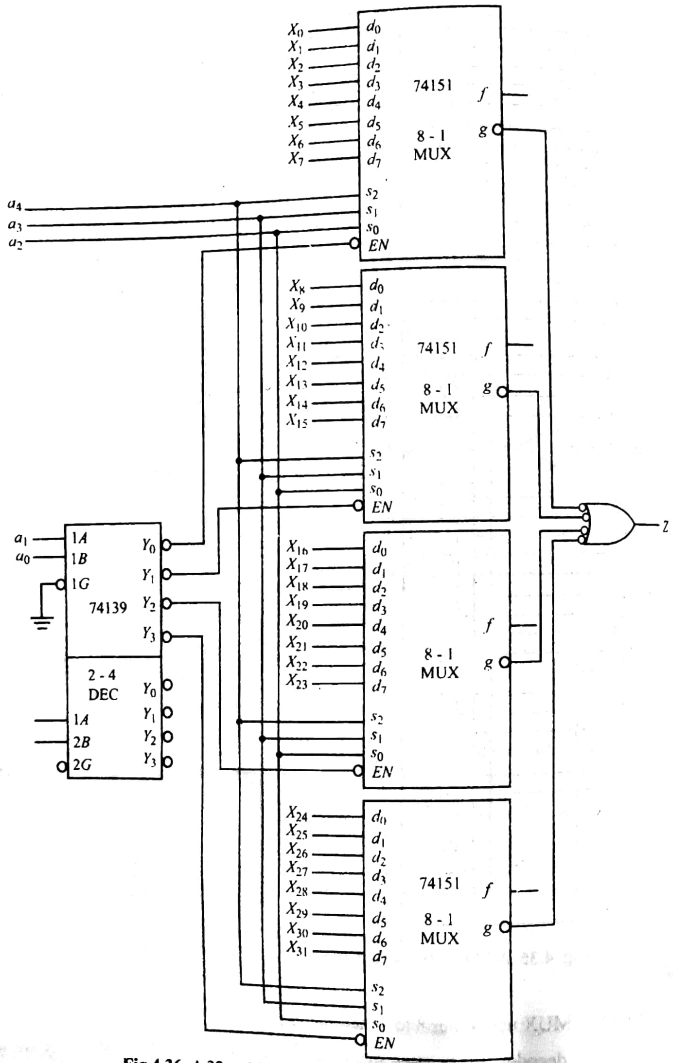


Fig 4.36 A 32 to 1 Mux configured using four 8-1 multiplexers

4.2.3 Logic Design Using Multiplexers

We found decoders convenient to implement Boolean expressions in minterm canonical form because all the minterms were available at the output terminals. We simply ORed the selected minterms to realize any function. Observe in Fig. 4.27 and Fig. 4.33 that the OR gate is also inbuilt in the MUX. Thus, the MUX is ideal to implement functions expressed as a sum of their minterms.

The truth table of a three variable function and implementation using a 8 to 1 MUX is shown in Fig. 4.37.

a	b	c	f
0	0	0	d_0
0	0	1	d_1
0	1	0	d_2
0	1	1	d_3
1	0	0	d_4
1	0	1	d_5
1	1	0	d_6
1	1	1	d_7

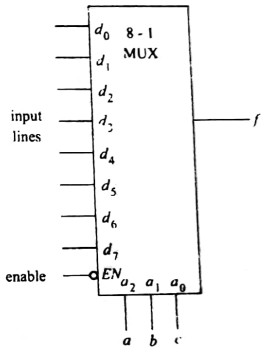


Fig. 4.37 Truth table and implementation of a three variable expression

Let the enable E be at logic 0.
Thus

$$f(a, b, c) = d_0 \cdot \bar{a}\,\bar{b}\,\bar{c} + d_1 \bar{a}\,\bar{b}\,c + d_2 \cdot \bar{a}\,b\bar{c} + d_3 \cdot \bar{a}\,bc + d_4 \cdot a\bar{b}\,\bar{c} + d_5 \cdot a\bar{b}c + d_6 \cdot ab\bar{c} + d_7 \cdot abc$$
$$f(a, b, c) = d_0 \cdot m_0 + d_1 \cdot m_1 + d_2 \cdot m_2 + d_3 \cdot m_3 + d_4 \cdot m_4 + d_5 \cdot m_5 + d_6 \cdot m_6 + d_7 \cdot m_7 \qquad (4.14)$$

Now, in order to implement a three variable function expressed in terms of its minterms are simply needed to set the data input lines as follows.

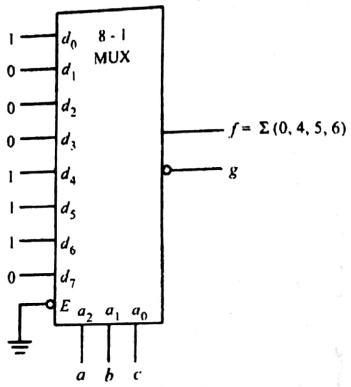
$$d_i = 1 \quad \text{for} \quad f = 1$$
$$d_i = 0 \quad \text{for} \quad f = 0$$

Implement the following function using a 8 to 1 MUX

$$f(a, b, c) = \Sigma\,(0, 4, 5, 6)$$

Solution: Let $d_0 = d_4 = d_5 = d_6 = 1$ and $d_1 = d_2 = d_3 = d_7 = 0$ in a 8 to 1 MUX.

The arrangement is shown below.



Implement the following function using a 4 to 1 MUX

$$f(a, b, c) = \Sigma(0, 1, 2, 7)$$

Solution: The function given is

$$f(a, b, c) = \Sigma(0, 1, 2, 7)$$

In algebraic form, the function becomes

$$f = \bar{a}\bar{b}\bar{c} + \bar{a}\bar{b}c + \bar{a}b\bar{c} + abc$$
$$= \bar{a}\bar{b}(\bar{c} + c) + \bar{a}b(\bar{c}) + ab(c)$$

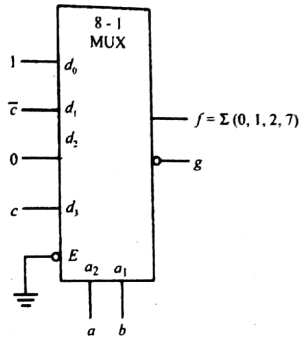
In terms of a two variable function, we can express this as

$$f = \bar{a}\bar{b}(1) + \bar{a}b(\bar{c}) + a\bar{b}(0) + ab(c)$$

Assuming enable is at logic 1, compare this with the output expression of a 4 to 1 MUX with d_0, d_1, d_2, d_3 as data lines and ab as the address select lines.

$$f = \bar{a}\bar{b}(d_0) + \bar{a}b(d_1) + a\bar{b}(d_2) + ab(d_3)$$

The implementation is shown below.

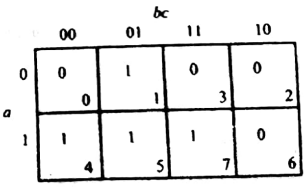


The procedure followed in Example 4.8 works fine if the address lines are assigned variables in order from the most significant position. In Example 4.8, a and b were assigned to a_0 and a_1. The situation changes when we would like to assign say b and c to the address or select lines of the multiplexer.

Consider the implementation of the function represented by the truth table in Fig. 4.38 (a).

Cell no.	*a*	*b*	*c*	*f*
0	0	0	0	0
1	0	0	1	1
2	0	1	0	0
3	0	1	1	0
4	1	0	0	1
5	1	0	1	1
6	1	1	0	0
7	1	1	1	1

(a)



(b)

Fig. 4.38 (a) Truth table and (b) Karnaugh map for $f = \Sigma(1, 4, 5, 7)$

The Karnaugh map is shown in Fig. 4.38 (b).
Let a be assigned to select line a_1 and b to select line a_0 in a 4 to 1 MUX.

The functional expression for a 4 to 1 MUX with address lines ab and enable E set to 0 is

$$f = \bar{a}\bar{b}(d_0) + \bar{a}b(d_1) + a\bar{b}(d_2) + ab(d_3)$$

Hence d_0 relates to cells with $a = 0, b = 0$, d_1 relates to cells with $a = 0, b = 1$, d_2 relates to cells with $a = 1, b = 0$ and d_3 relates to cells with $a = 1, b = 1$, as shown in Fig. 4.39.

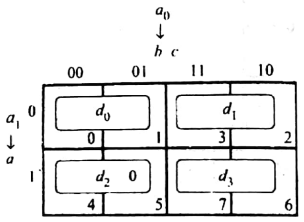


Fig. 4.39 Address and data lines mapping

Cells 0 to 1 correspond to $a = 0, b = 0$. Cell 0 corresponds to $c = 0$ and cell 1 to $c = 1$. On these lines, the Karnaugh maps for the sub-function d_0, d_1, d_2 and d_3 is shown in Fig. 4.40.

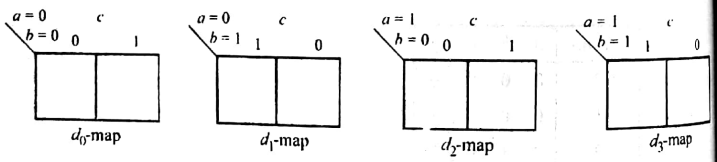


Fig. 4.40 Maps of sub-functions

Maps shown in Fig. 4.39 and 4.40, drawn for the function in Fig. 4.37 is shown in Fig. 4.41.

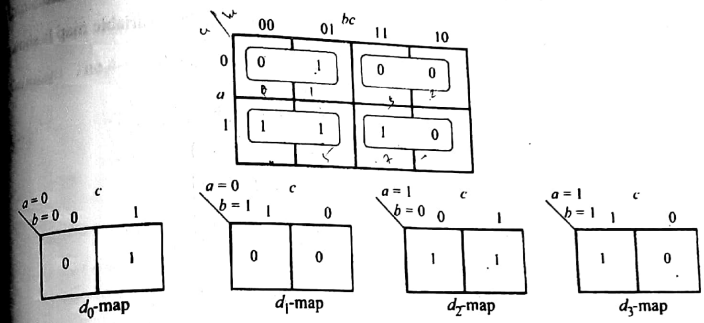


Fig. 4.41 Karnaugh map of the function and sub-function of Fig. 4.38

Now, we can see from the sub-function maps that

$$d_0 = c$$
$$d_1 = 0$$
$$d_2 = 1$$
$$d_3 = c$$

The implementation using 4 to 1 MUX is shown in Fig. 4.42.

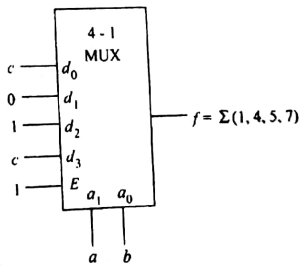
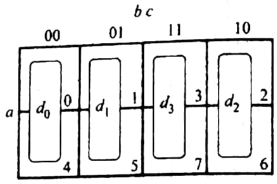


Fig. 4.42 Implementation of $f = \Sigma(1, 4, 5, 7)$ on 4 to 1 MUX

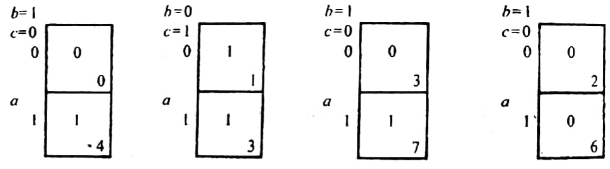
Implement $f(a, b, c) = \Sigma(1, 4, 5, 7)$ using a 4 to 1 MUX using b and c as the address lines.

***Solution*:** Sub-function mapping with b and c as selection lines on a three variable map is shown below.



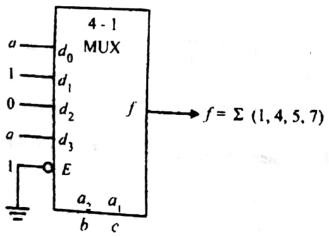
$b = 0, c = 0$	correspond to cells 0 and 4
$b = 0, c = 1$	correspond to cells 1 and 5
$b = 1, c = 0$	correspond to cells 2 and 6
$b = 1, c = 1$	correspond to cells 3 and 7

Filling in the function values we have



d_0-map $d_0 = a$ | d_1-map $d_1 = 1$ | d_3-map $d_3 = a$ | d_2-map $d_2 = 0$

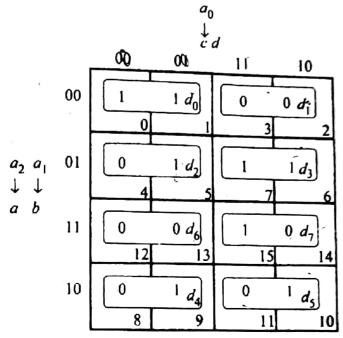
The implementation is shown below.



Implement the following function using a 8 to 1 MUX. Treat a, b and c as the select lines.

$$f(a, b, c, d) = \Sigma\,(0, 1, 5, 6, 7, 9, 10, 15)$$

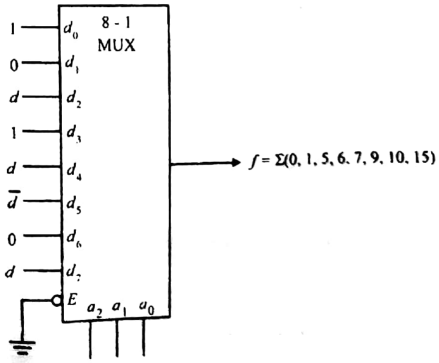
***Solution*:** The 4-variable map with sub-functions is shown below.



From the map

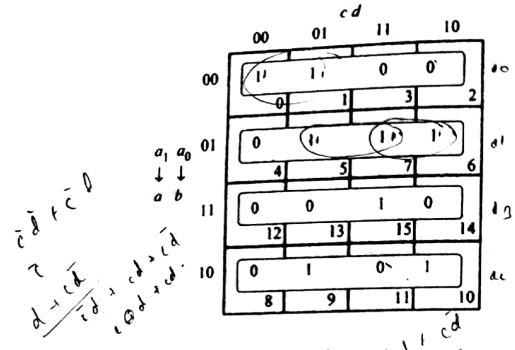
$$d_0 = 1 \quad d_4 = d$$
$$d_1 = 0 \quad d_5 = \bar{d}$$
$$d_2 = d \quad d_6 = 0$$
$$d_3 = 1 \quad d_7 = d$$

The implementation is shown below.



Implement the function in Example 4.10 using a 4 to 1 MUX with a and b as select lines.

Solution: The 4-variable map with sub-functions marked shown below.



$a = 0, b = 0$ relates to cells 0, 1, 2 and 3 (d_0)
$a = 0, b = 1$ relates to cells 4, 5, 6 and 7 (d_1)
$a = 1, b = 0$ relates to cells 8, 9, 10 and 11 (d_2)
$a = 1, b = 1$ relates to cells 12, 13, 14 and 15 (d_3)

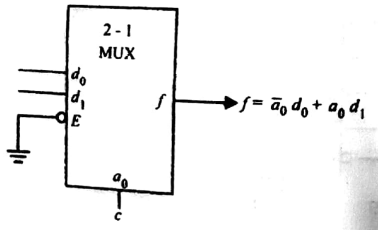
The sub-function maps are as follows:
From the submaps above,

$$d_0 = \bar{c} \qquad d_2 = \bar{c}\,d + c\,\bar{d} = c \oplus d$$
$$d_1 = c + d \qquad d_3 = cd$$

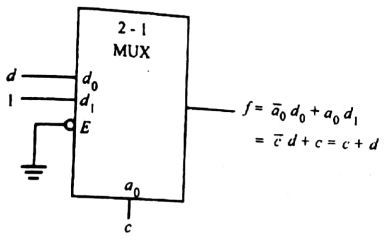
The implementation is shown below.

Implement the expressions for d_1, d_2 and d_3 using 2 to 1 MUX in Example 4.11.

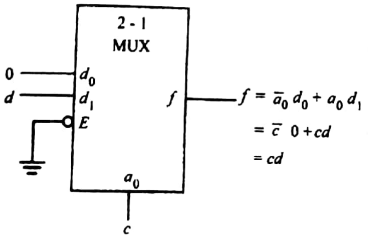
Solution: The symbol of a 2 to 1 MUX is shown below.



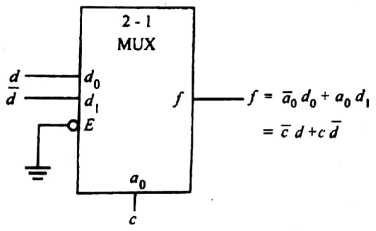
Let c be used as the adders. Then $c + d$ can be implemented as shown below.



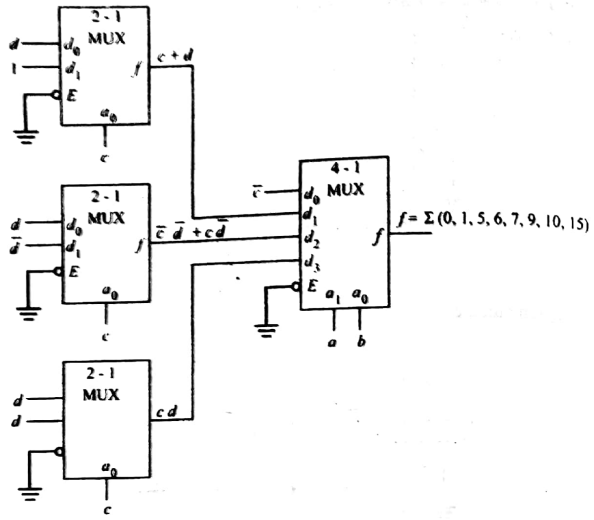
cd can be implemented as follows.



$\bar{c}\,d + c\,\bar{d}$ can be implemented as follows.



The implementation of $f = \Sigma\,(0, 1, 5, 6, 7, 9, 10, 15)$ is as follows:



Example

The following are the allocations to the address or select lines $a_2\ a_1\ a_0$ of an 8 to 1 MUX. Indicate the location of the submaps d_0 to d_7 on a 4-variable Karnaugh map with variables a, b, c, d.

a. b, c, d to $a_2\ a_1\ a_0$

b. a, c, d to $a_2\ a_1\ a_0$

c. c, b, a to $a_2\ a_1\ a_0$

***Solution*:**

a. b, c, d to $a_2\ a_1\ a_0$

Let us draw the 4-variable map and then locate d_0 to d_7

$b = 0, c = 0, d = 0$ *i.e.*, cells 0, 8 relate to d_0

$b = 0, c = 0, d = 1$ *i.e.*, cells 1, 9 relate to d_1

$b = 0, c = 1, d = 0$ *i.e.*, cells 2, 10 relate to d_2

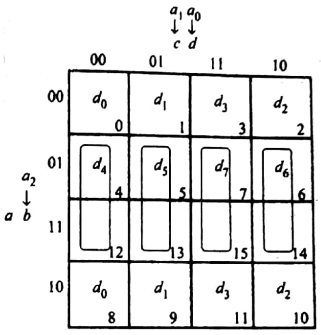
$b = 0, c = 1, d = 1$ *i.e.*, cells 3, 11 relate to d_3

$b = 1, c = 0, d = 0$ *i.e.*, cells 4, 12 relate to d_4

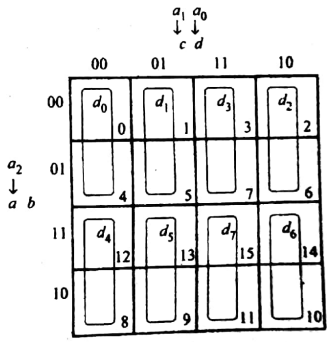
$b = 1, c = 0, d = 1$ *i.e.*, cells 5, 13 relate to d_5

$b = 1, c = 1, d = 0$ *i.e.* cells 6, 14 relate to d_6

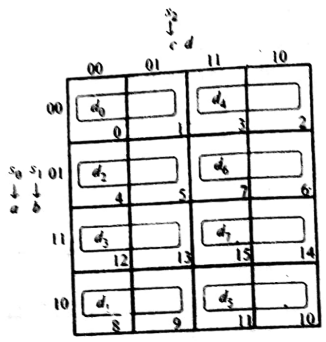
$b = 1, c = 1, d = 1$ *i.e.*, cells 7, 15 relate to d_7



b. a, c, d to $a_2\ a_1\ a_0$



c. c, b, a to a_2, a_1, a_0

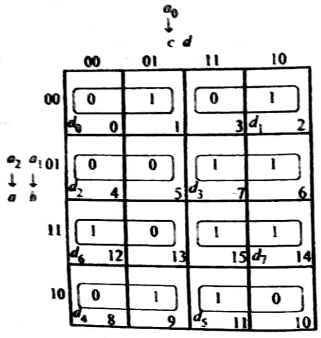


Implement the following Boolean functions using an 8 to 1 line multiplexer where a, b, c are connected to address lines a_2, a_1 and a_0 respectively.

a. $f(a, b, c, d) = \Sigma(1, 2, 6, 7, 9, 11, 12, 14, 15)$
b. $f(a, b, c, d) = \Sigma(2, 5, 6, 7, 9, 12, 13, 15)$

Solution:

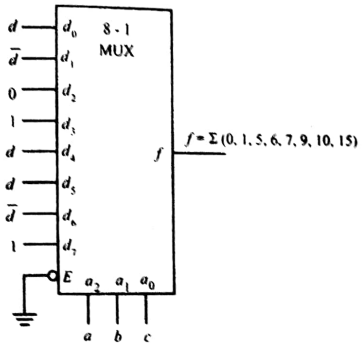
a. $f(a, b, c, d) = \Sigma(1, 2, 6, 7, 9, 11, 12, 14, 15)$



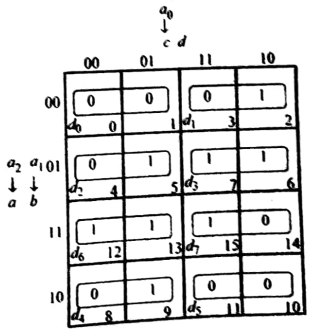
From the map

$d_0 = d$ $\quad d_4 = d$
$d_1 = \bar{d}$ $\quad d_5 = d$
$d_2 = 0$ $\quad d_6 = \bar{d}$
$d_3 = 1$ $\quad d_7 = 1$

The implementation is shown below.



b. $f = \Sigma(2, 5, 6, 7, 9, 12, 13, 15)$



From the map,

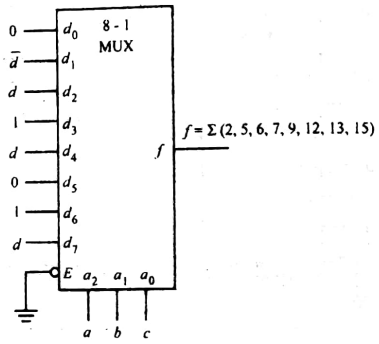
$d_0 = 0$ $\quad d_4 = d$

$d_1 = \bar{d}$ $\quad d_5 = 0$

$d_2 = d$ $\quad d_6 = 1$

$d_3 = 1$ $\quad d_7 = d$

The implementation is shown below.



Implement the Boolean function $f(a, b, c, d) = \Sigma(4, 5, 7, 8, 10, 12, 15)$ using a 4 to 1 line MUX and external gates if,

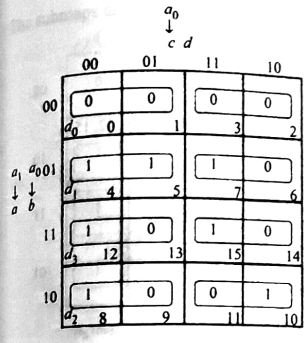
a. a and b are connected to select lines a_1 and a_0 respectively.

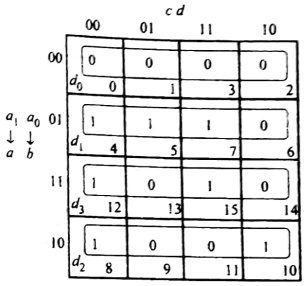
b. c and d are connected to select lines a_1 and a_0 respectively.

***Solution*:**

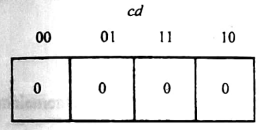
$$f(a, b, c, d) = \Sigma(4, 5, 7, 8, 10, 12, 15)$$

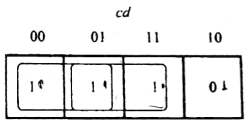
Let us plot the Karnaugh map with the sub-functions.

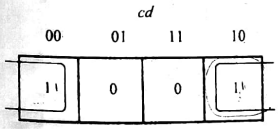


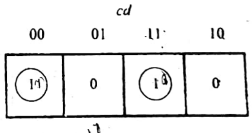


From the map, the sub-maps are







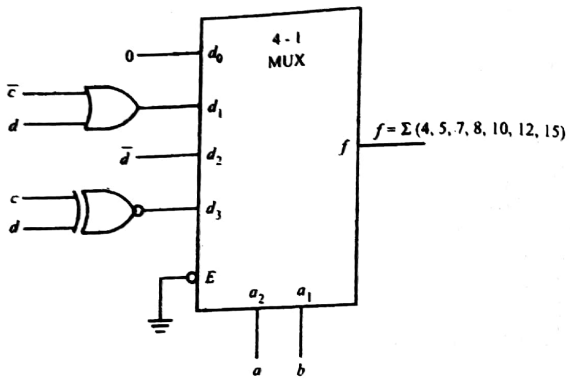


From the above submaps,

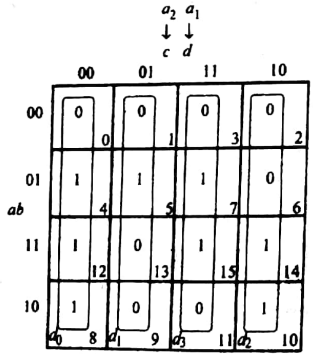
$d_0 = 0$ $\quad d_2 = \bar{d}$

$d_1 = \bar{c} + d$ $\quad d_3 = \bar{c}\,\bar{d} + cd = \overline{c \oplus d}$

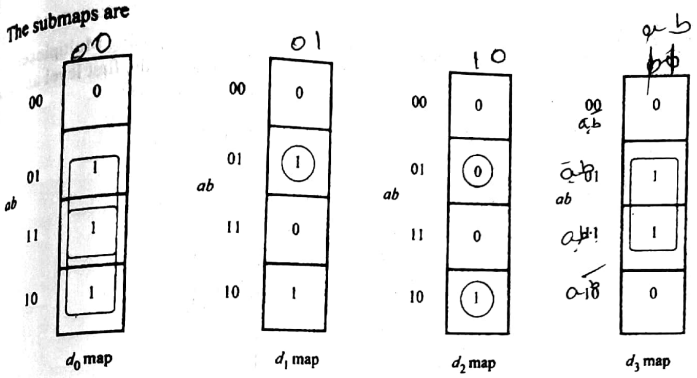
The implementation is shown below.



b. Now c and d are connected to a_2a_1. The Karnaugh map with subfunction is shown below.



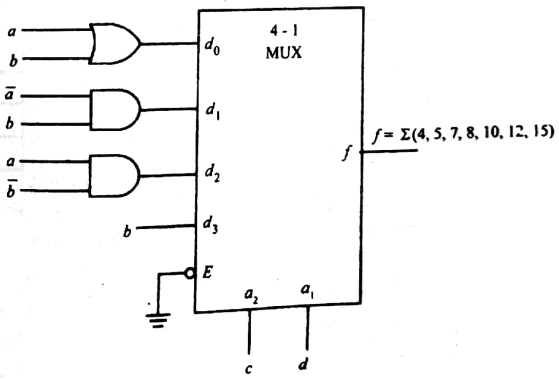
The submaps are



From the submaps

$$d_0 = a + b \qquad d_2 = a\bar{b}$$
$$d_1 = \bar{a}\,b \qquad d_3 = b$$

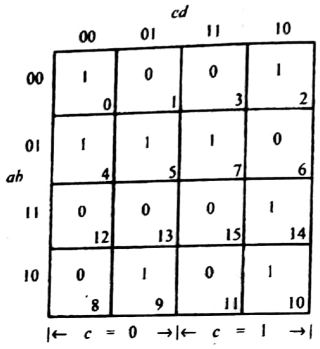
The implementation is shown below.



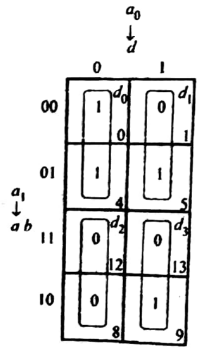
Implement the Boolean function $f(a, b, c, d) = \Sigma(0, 2, 4, 5, 7, 9, 10, 14)$ using Multiplexers with two 4 to 1 line MUX with variables ad connected to their select lines in the first level and one 2 to 1 line MUX with variable c connected to its select line in the second level.

Solution: $f(a, b, c, d) = \Sigma(0, 2, 4, 5, 7, 9, 10, 14)$

The Karnaugh map is plotted as follows.



Let us consider the $c = 0$ map for the final 2 to 1 line MUX.



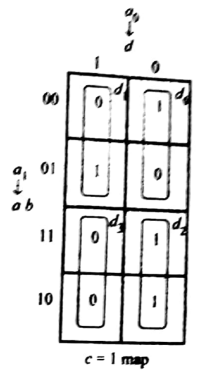
$(a = 0,\ d = 0)$ represents d_0
$(a = 0,\ d = 1)$ represents d_1
$(a = 1,\ d = 0)$ represents d_2
$(a = 1,\ d = 1)$ represents d_3

Let us write d_0, d_1, d_2 and d_3 in terms of variable b.

$d_0 = 1$ $\qquad$ $d_2 = 0$

$d_1 = b$ $\qquad$ $d_3 = \overline{b}$

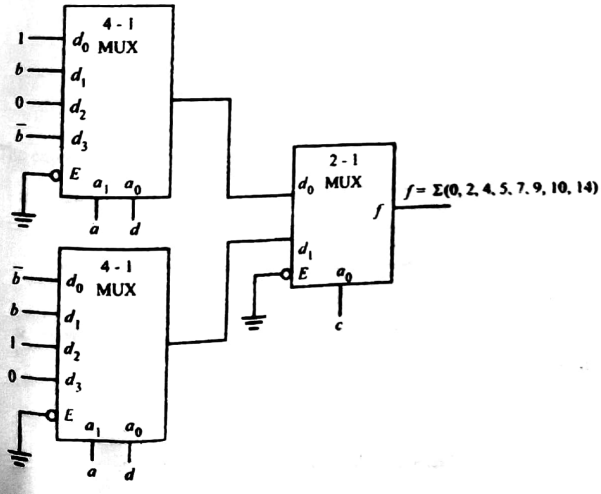
Now, let us look at the $c = 1$ map.



$\therefore$ $\qquad d_0 = \overline{b}$ $\qquad d_2 = 1$

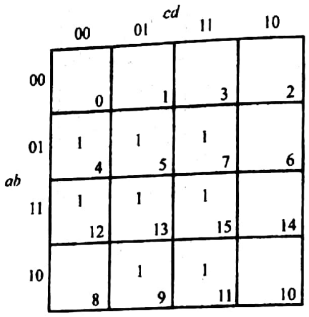
$d_1 = b$ $\qquad d_3 = 0$

The implementation is shown below.



Implement $u = ad + b\bar{c} + bd$ using (i) 16-1 MUX (ii) 8-1 MUX (iii) 4-1 MUX

Solution: Let us first draw the Karnaugh map for the function



$u = \Sigma(4, 5, 7, 9, 11, 12, 13, 15)$

(i) Implementation using a 16-1 MUX is shown in Fig 4.43.

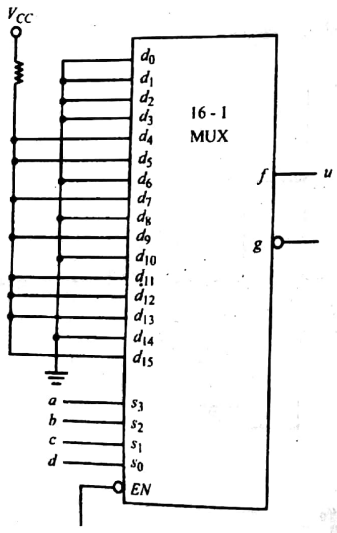


Fig. 4.43 Implementation with 16-1 MUX

(ii) From the map, we can write,

$$u = \underline{\bar{a}b\bar{c}\bar{d} + \bar{a}b\bar{c}d} + \bar{a}bcd + \underline{ab\bar{c}\bar{d} + ab\bar{c}d} + abcd + a\bar{b}\bar{c}d + a\bar{b}cd$$

$$= \bar{a}b\bar{c}(d) + \bar{a}bc(d) + ab\bar{c}(1) + abc(d) + a\bar{b}\bar{c}d + a\bar{b}cd$$

$$= d_2(1) + d_3(d) + d_6(1) + d_7(d) + d_4(d) + d_5(d)$$

Implementation with 8-1 MUX is shown in Fig 4.44.

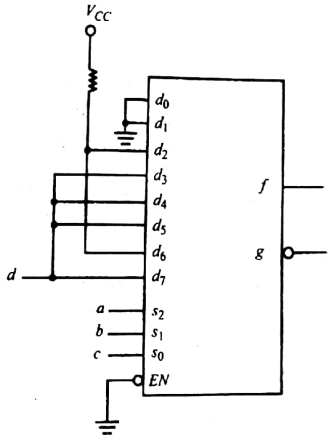


Fig 4.44 Implementation using 8-1 MUX

(iii) The expression in part (ii) can also be written as

$$u = \bar{a}b(\bar{c}\bar{d} + \bar{c}d + cd) + ab(\bar{c}\bar{d} + \bar{c}d + cd) + a\bar{b}(\bar{c}d + cd)$$

$$= \bar{a}b(\bar{c} + cd) + ab(\bar{c} + cd) + a\bar{b}d$$

$$= \bar{a}b(\bar{c} + d) + ab(\bar{c} + d) + a\bar{b}d$$

or

$$u = \bar{a}\bar{b}(o) + \bar{a}b(\bar{c} + d) + a\bar{b}(d) + ab(\bar{c} + d)$$

The implementation using 4-1 Mux is shown in Fig 4.45

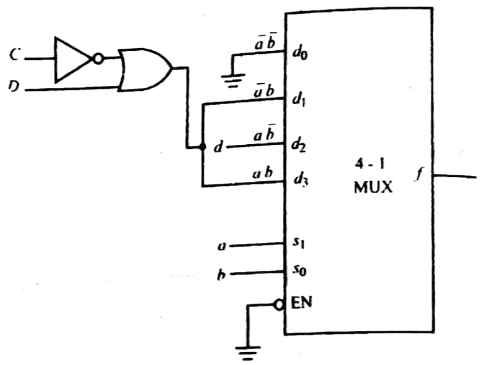


Fig 4.45 Implementation using 4-1 MUX

Example 4.18 Implement the following using multiplexers whose data inputs could accept only either $a, 0$ or $a, 1$. A 16 to 1 MUX and 8 to 1 MUX with active low enable and with active high outputs are available.

(a) $f(a, b, c) = \Sigma(0, 2, 3, 6, 7)$

(b) $f(a, b, c, d) = \Sigma(2, 5, 6, 9, 11, 14)$

(c) $f(a, b, c, d, e) = \Sigma(1, 5, 8, 12, 19, 21, 27)$

Solution:

(a) $f(a, b, c) = \Sigma(0, 2, 3, 6, 7)$.

The implementation using 8-1 MUX is shown in Fig 4.46 (a).

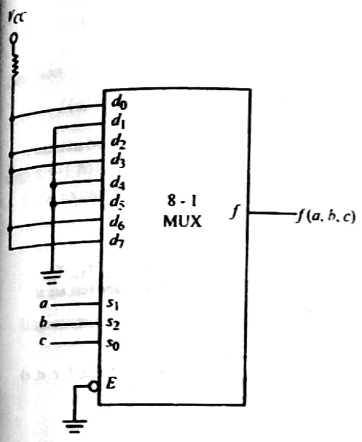


Fig 4.46 (a) Implementation of Example 4.18(a)

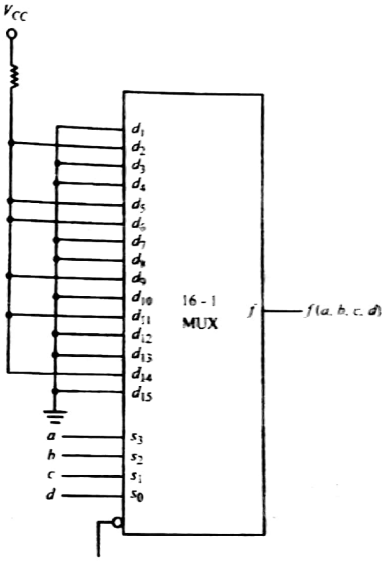


Fig 4.46 (b) Implementation of Example 4.18 (b)

(b) $f(a, b, c, d) = \Sigma(2, 5, 6, 9, 11, 14)$

The implementation using 16-1 MUX is shown in Fig 4.46 (b)

(c) $f(a, b, c, d, e) = \Sigma(1, 5, 8, 12, 19, 21, 27)$

The implementation using two 16-1 MUX is shown in Fig 4.46 (c).

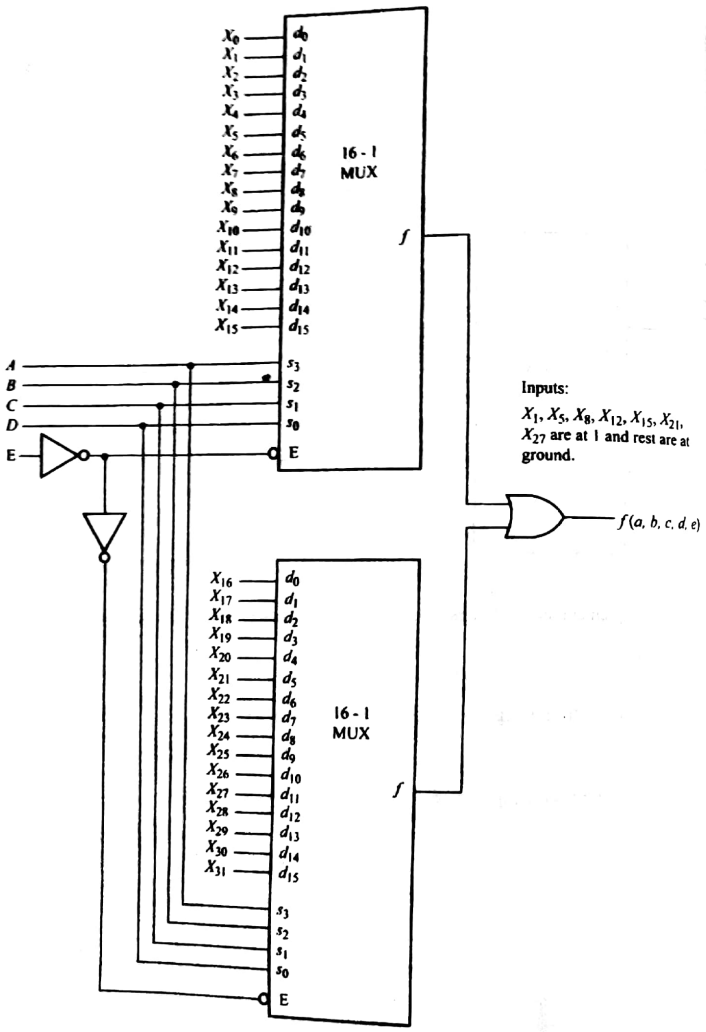


Fig 4.46 (c) Implementation of Example 4.18 (c)

Implement the following expressions using suitable MUX single variable MEV can appear at the inputs of the MUX.

(a) $f(a, b, c) = \Sigma(0, 4, 5, 7)$
(b) $f(a, b, c, d) = \Sigma(2, 3, 4, 5, 8, 12, 13, 15)$
(c) $f(a, b, c, d, e) = \Sigma(0, 1, 2, 6, 7, 13, 24, 25)$

Solution:

(a) $f(a, b, c) = \Sigma(0, 4, 5, 7)$

This can be directly implemented with a 8-1 MUX. A single variable appears when implemented with a 4-1 MUX.

Now expanding the compact notation we have,

$$f(a, b, c) = \bar{a}\bar{b}\bar{c} + a\bar{b}\bar{c} + a\bar{b}c + abc$$
$$= \bar{a}\bar{b}(\bar{c}) + a\bar{b}(1) + ab(c) + \bar{a}b(0)$$

Implementation using 4-1 MUX is shown in Fig 4.47 (a).

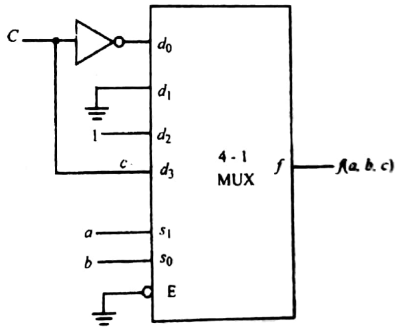


Fig 4.47 (a) Implementation of Example 4.19 (a)

(b) $f(a, b, c, d) = \Sigma(2, 3, 4, 5, 8, 12, 13, 15)$

This can be implemented directly using a 16-1 MUX and can be implemented using a 8-1 MUX when one *MEV* is allowed as shown in Fig 4.47 (b).

Expanding the compact notation we have,

$$f = \bar{a}\bar{b}c\bar{d} + \bar{a}\bar{b}cd + \bar{a}b\bar{c}\bar{d} + \bar{a}b\bar{c}d + a\bar{b}\bar{c}\bar{d} + ab\bar{c}\bar{d} + ab\bar{c}d + abcd$$
$$= \bar{a}\bar{b}c(1) + \bar{a}b\bar{c}(1) + a\bar{b}\bar{c}(\bar{d}) + ab\bar{c}(1) + abc(d)$$

Implementation using 8-1 MUX is shown in Fig 4.47 (b).

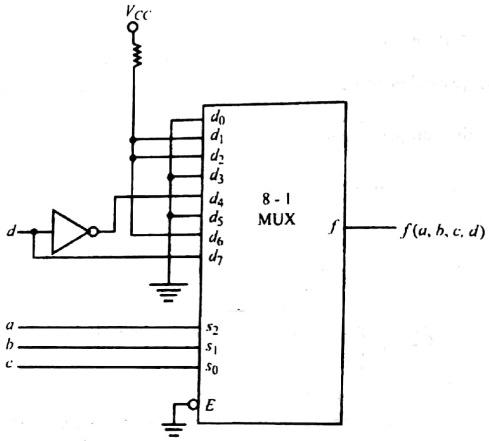


Fig 4.47 (b) Implementation of Example 4.19 (b)

(c) $f(a, b, c, d, e) = \Sigma(0, 1, 2, 6, 7, 13, 24, 25)$

Expanding the compact notation, we have,

$$f = \bar{a}\bar{b}\bar{c}\bar{d}\bar{e} + \bar{a}\bar{b}\bar{c}\bar{d}e + \bar{a}\bar{b}\bar{c}d\bar{e} + \bar{a}\bar{b}cd\bar{e} + \bar{a}\bar{b}cde$$
$$+ \bar{a}b\bar{c}de + abcd\bar{e} + abcde$$

$$\therefore \quad f = \bar{a}\bar{b}\bar{c}\bar{d}(1) + \bar{a}\bar{b}\bar{c}d(\bar{e}) + \bar{a}\bar{b}cd(1) + \bar{a}b\bar{c}d(e) + abcd(1)$$

or $$f = d_0(1) + d_1(\bar{e}) + d_3(1) + d_5(e) + d_{14}(1)$$

The implementation is shown in Fig 4.47(c).

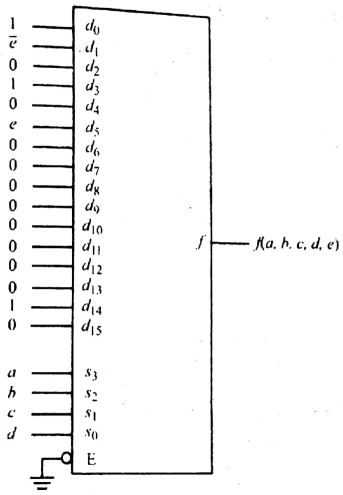


Fig 4.47 (c) Implementation of Example 4.19 (c)

4.3 Comparators

Comparators are designed to compare the magnitude of two binary numbers and indicate whether one is greater than (gt), less than (lt) or equal (eq) to the other. We will consider 1-bit, 2-bit as well as iterative comparators in this section.

4.3.1 One-bit Comparators

One-bit comparators compare two one-bit numbers a and b and produce three outputs $a = b$, $a > b$ and $a < b$. The truth table of a 1-bit comparator is shown in Fig 4.48.

Inputs		*Outputs*		
a	b	a eq b	a gt b	a lt b
0	0	1	0	0
0	1	0	0	1
1	0	0	1	0
1	1	1	0	0

Fig. 4.48 Truth table of a 1-bit comparator

$$a \text{ eq } b = \bar{a}\,\bar{b} + ab = a \odot b$$

$$a \text{ gt } b = a\,\bar{b}$$

and $$a \text{ lt } b = \bar{a}\,b$$

The implementation is shown in Fig 4.49.

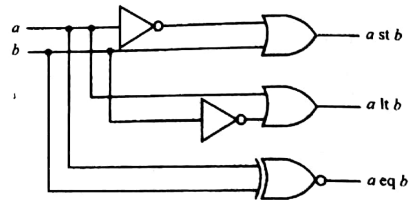


Fig 4.49 Implementation of 1-bit comparator

Implement a 1-bit comparator using a 2-4 decoder, 74139.

Solution: From Fig 4.48,

$$a \text{ eq } b = \Sigma\,(0,3)$$

$$a \text{ gt } b = \Sigma\,(2)$$

$$a \text{ lt } b = \Sigma\,(1)$$

Implementation using 2-4 decoder is shown in Fig 4.50.

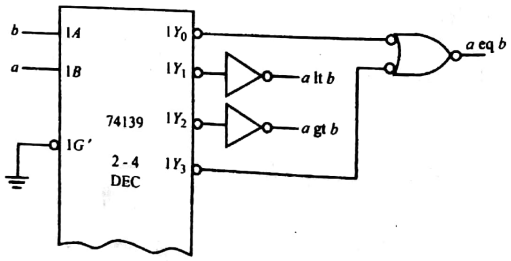


Fig 4.50 Implementation of 1-bit comparator using 2-4 decoder

4.3.2 Two-bit Comparators

A two bit magnitude comparator would accept two 2-bit numbers and generate three outputs. The block diagram of such a comparator is shown in Fig. 4.51.

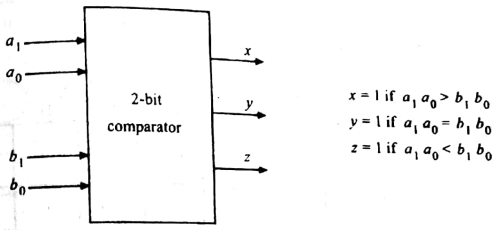


Fig. 4.51 Block diagram of 2-bit magnitude comparator

We now need to obtain expressions for x, y and z. The truth table for the 2-bit magnitude comparator is shown Fig. 4.52.

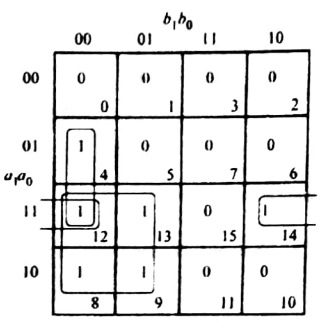
Cell no.	a_1	a_0	b_1	b_0	x	y	z
0	0	0	0	0	0	1	0
1	0	0	0	1	0	0	1
2	0	0	1	0	0	0	1
3	0	0	1	1	0	0	1
4	0	1	0	0	1	0	0
5	0	1	0	1	0	1	0
6	0	1	1	0	0	0	1
7	0	1	1	1	0	0	1
8	1	0	0	0	1	0	0
9	1	0	0	1	1	0	0
10	1	0	1	0	0	1	0
11	1	0	1	1	0	0	1
12	1	1	0	0	1	0	0
13	1	1	0	1	1	0	0
14	1	1	1	0	1	0	0
15	1	1	1	1	0	1	0

Fig. 4.52 Truth table for 2-bit magnitude comparator

We have

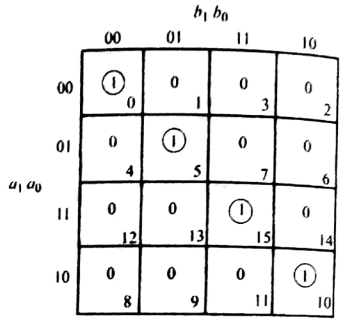
$$x = \Sigma(4, 8, 9, 12, 13, 14)$$
$$y = \Sigma(0, 5, 10, 15)$$
and $$z = \Sigma(1, 2, 3, 6, 7, 11)$$

Let us now write the Karnaugh maps for each output.



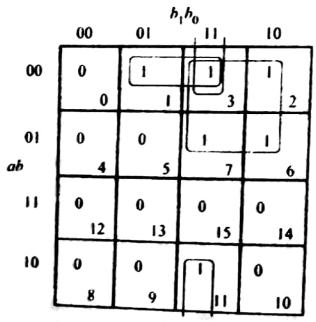
x-map

$x = a_1\bar{b}_1 + a_0\bar{b}_1\bar{b}_0 + a_1a_0\bar{b}_0$



y-map

$y = \bar{a}_1\bar{a}_0\bar{b}_1\bar{b}_0 + \bar{a}_1a_0\bar{b}_1b_0 + \bar{a}_1a_0\bar{b}_1b_0 + a_1a_0b_1b_0$



z-map

$z = \bar{a}_1b_1 + \bar{a}_0b_1b_0 + \bar{a}_1\bar{a}_0b_0$

Once we have x and z, we can write y as

$$y = \bar{x} \cdot \bar{z} = \overline{x + z}$$

The implementation is shown in Fig. 4.53 and y can also be implemented using its Boolean expression which is rather costly.

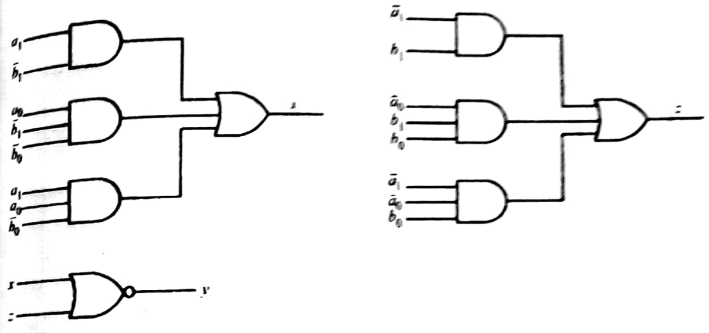


Fig. 4.53 Implementation of 2-bit comparator

Implement a 2-bit comparator using a 4-16 decoder.

Solution: We have from section 4.3.2

$$x = \Sigma(4, 8, 9, 12, 13, 14)$$
$$y = \Sigma(0, 5, 10, 15)$$
and $$z = \Sigma(1, 2, 3, 6, 7, 11)$$

The implementation using 4-16 decoder is shown in Fig 4.54.

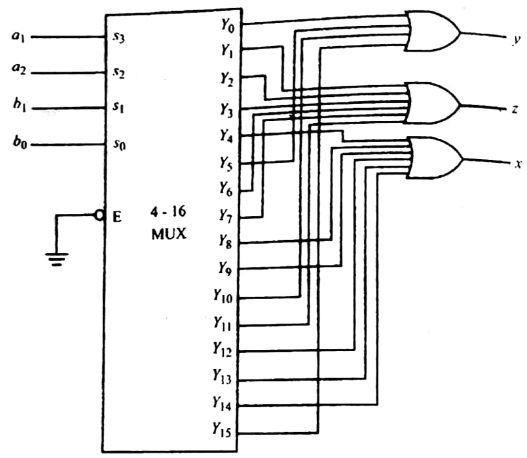


Fig 4.54 Implementation of 2-bit comparator using 4-16 decoder

◆ **Exercise 4.5**

Implement a 2-bit comparator using (a) two 3-8 decoders (b) four 2-4 decoders.

4.3.3 Iterative Comparators

Most often there is a requirement to compare two 4-bit, 8-bit or higher bit binary numbers. It is worthwhile to look at a 1-bit comparator with inputs and outputs which can be used to cascade several of these to configure multiple-bit comparators. The block diagram of the 1-bit comparator at the i^{th} stage, of such a cascade is shown in Fig. 4.55.

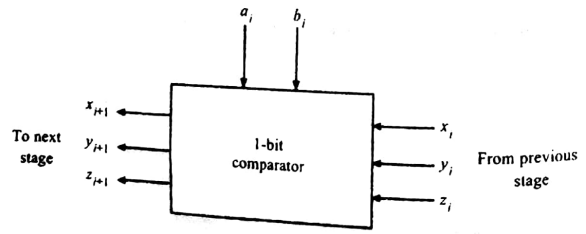


Fig. 4.55 Block diagram of 1-bit comparator

The truth table of the cascadable 1-bit comparator is shown in Fig. 4.56.

Cell no.		a_i	b_i	x_i	y_i	z_i	x_{i+1}	y_{i+1}	z_{i+1}
0	0	0	0	0	0	0	–	–	–
	1	0	0	0	0	1	0	0	1
1	2	0	0	0	1	0	0	1	0
	3	0	0	0	1	1	–	–	–
2	4	0	0	1	0	0	1	0	0
	5	0	0	1	0	1	–	–	–
3	6	0	0	1	1	0	–	–	–
	7	0	0	1	1	1	–	–	–
4	8	0	1	0	0	0	–	–	–
	9	0	1	0	0	1	0	0	1
5	10	0	1	0	1	0	0	0	1
	11	0	1	0	1	1	–	–	–
6	12	0	1	1	0	0	0	0	1
	13	0	1	1	0	1	–	–	–
7	14	0	1	1	1	0	–	–	–
	15	0	1	1	1	1	–	–	–
8	16	1	0	0	0	0	–	–	–
	17	1	0	0	0	1	1	0	0
9	18	1	0	0	1	0	1	0	0
	19	1	0	0	1	1	–	–	–
10	20	1	0	1	0	0	1	0	0
	21	1	0	1	0	1	–	–	–
11	22	1	0	1	1	0	–	–	–
	23	1	0	1	1	1	–	–	–
12	24	1	1	0	0	0	–	–	–
	25	1	1	0	0	1	0	0	1
13	26	1	1	0	1	0	0	1	0
	27	1	1	0	1	1	–	–	–
14	28	1	1	1	0	0	1	0	0
	29	1	1	1	0	1	–	–	–
15	30	1	1	1	1	0	–	–	–
	31	1	1	1	1	1	–	–	–

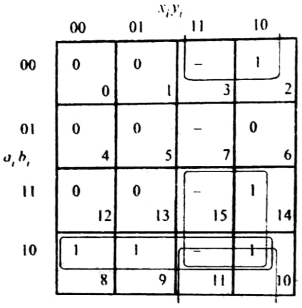
Fig. 4.56 Truth table of 1-bit cascade comparator

In row 0, $x_i = y_i = z_i = 0$. This will not occur because one of them will be set to 1 by the comparator at the previous stage. The output corresponding to the row can be don't cares. In row 1, $z_i = 1$ implies that in the previous stage a_i was less than b_i. The situation is shown in Fig. 4.57.

$a_i\ a_{i-1}$ / $b_i\ b_{i-1}$	0 0	0 1
	$a_i = b_i$	$a_{i-1} < b_{i-1}$

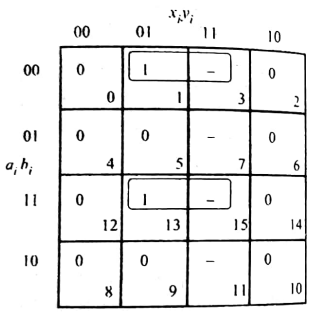
Fig. 4.57 Situation in row 1 of Fig. 4.56

$z_i = 1$ implies that in the previous stage a_{i-1} is less than b_{i-1}. In other words, a_{i-1} must ha been 0 and b_{i-1} must have been 1. Under these circumstances, when $a_i = 0$, $b_i = 0$ then the bi under comparison are 00 and 01. Thus, the output to be generated is $z_{i+1} = 1$. In row 2, y_i implies $a_{i-1} = b_{i-1} = 1$ or $a_{i-1}\ b_{i-1} = 0$. The possible bits under comparison at i^{th} stage wi $a_i = b_i = 0$ are 00 and 00 or 01 and 01. In either case, $y_{i+1} = 1$ needs to be generated. By simil reasoning the entire table can be filled. Don't cares are placed in rows where $x_i = y_i = z_i$ as we as in rows where more than one 1s appear under x_i, y_i and z_i. In any row if $a_i > b_i$, then x_{i+1} = irrespective of x_i, y_i and z_i and if $a_i < b_i$, then $z_{i+1} = 1$ irrespective of x_i, y_i and z_i.



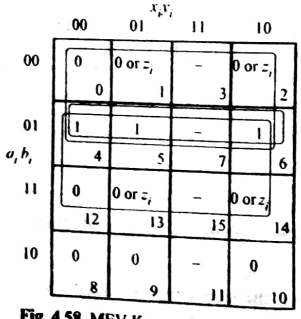
x_{i+1} map

$x_{i+1} = a_i \bar{b}_i + a_i x_i + \bar{b}_i x_i$



y_{i+1} map

$y_{i+1} = \bar{a}_i\ \bar{b}_i\ y_i + a_i\ b_i\ y_i$



z_{i+1} map

$z_{i+1} = \bar{a}_i b_i + \bar{a}_i z_i + b_i z_i$

Fig. 4.58 MEV Karnaugh map for Table 4.9

Let us now develop expressions for x_{i+1}, y_{i+1} and z_{i+1}. The maps for each output is shown in Fig. 4.58 taking z_i as the MEV and applying the rules of table.
The implementation of three expressions is shown in Fig. 4.59.

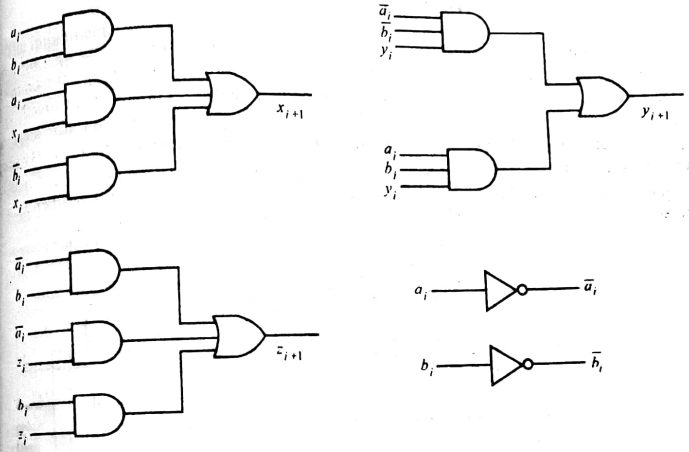


Fig. 4.59 Implementaion of 1-bit cascadable comparator

Such 1-bit comparators can be cascaded to configure multiple-bit comparators. Fig. 4.60 shows a 4-bit comparator configured using four stages of 1-bit comparators.

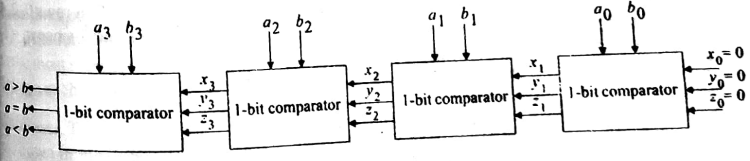


Fig. 4.60 4-bit comparator

Design a comparator to check if two n-bit numbers are equal. Configure this using cascaded stages of 1-bit comparators.

Solution: **The block diagram of the 1-bit comparator is shown in Fig. 4.61.**

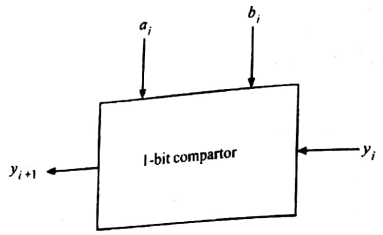


Fig. 4.61 Block diagram of 1-bit equality comparator

The truth table for this 1-bit equality comparator is shown in Fig. 4.62.

Cell no.	a_i	b_i	y_i	y_{i+1}
0	0	0	0	0
1	0	0	1	1
2	0	1	0	0
3	0	1	1	0
4	1	0	0	0
5	1	0	1	0
6	1	1	0	0
7	1	1	1	1

Fig. 4.62 Truth table of 1-bit equality comparator

The very fact that $y_i = 0$ indicates inequality. Thus, $y_{i+1} = 0$, whenever $y_i = 0$ in rows 0, 2, 4 and 6. Further, $y_{i+1} = 0$ whenever $a_i \neq b_i$, in rows 3 and 5. $y_{i+1} = 1$ in rows 1 and 7 since $a_i = b_i$ and $y_i = 1$ implies all previous bits were also equal.

The Karnaugh map for the above truth table is shown in Fig. 4.63.

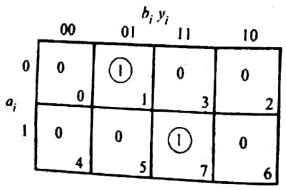


Fig. 4.63 Karnaugh map for Fig. 4.62

$$y = \bar{a}_i \bar{b}_i y_i + a_i b_i y_i$$
$$y_{i+1} = \bar{a}_i \bar{b}_i y_i + a_i b_i y_i$$
$$= (\overline{a_i \oplus b_i}) y_i$$

The implementation is shown in Fig. 4.64.

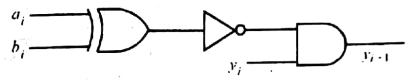


Fig. 4.64 Implementation of 1-bit equality comparator

n-such stages can be cascaded to obtain an n-bit equality comparator as shown in Fig. 4.65.

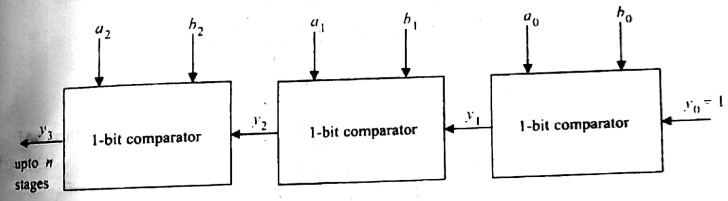


Fig. 4.65 Multiple stage equality comparator

The 1-bit comparator had 3 outputs corresponding to $a > b$, $a = b$ and $a < b$. It is possible to code these three outputs using two bits pq such as $pq = 10$ for $a > b$, $pq = 00$ for $a = b$ and $pq = 01$ for $a < b$. This reduces the number of output lines of each 1-bit comparator to 2. The 1-bit comparator at the most significant position, however, should have a converter to convert back to three outputs. Design such a 1-bit comparator as well as the output converter network.

Solution: The block diagram of the 1-bit comparator and the output network with three outputs is shown in Fig. 4.66.

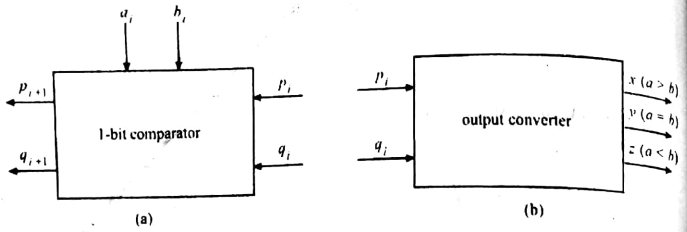


Fig. 4.66 Block diagram of (a) 1-bit comparator (b) output converter network

The truth table is shown in Fig. 4.67.

Cell no.	a_i	b_i	p_i	q_i	p_{i+1}	q_{i+1}
0	0	0	0	0	0	0
1	0	0	0	1	0	1
2	0	0	1	0	1	0
3	0	0	1	1	–	–
4	0	1	0	0	0	1
5	0	1	0	1	0	1
6	0	1	1	0	0	1
7	0	1	1	1	–	–
8	1	0	0	0	1	0
9	1	0	0	1	1	0
10	1	0	1	0	1	0
11	1	0	1	1	–	–
12	1	1	0	0	0	0
13	1	1	0	1	0	1
14	1	1	1	0	1	0
15	1	1	1	1	–	–

Fig. 4.67 Truth table of 1-bit comparator of Example 4.23

For $a_i = b_i$, $p_i q_i = 00$ implies $p_{i+1} q_{i+1} = 00$
For $a_i = 0$, $b_i = 1$, $p_{i+1} q_{i+1} = 01$
For $a_i = 1$, $b_i = 0$, $p_{i+1} q_{i+1} = 10$
For $a_i = b_i$, $p_i q_i = 01$ implies $p_{i+1} q_{i+1} = 01$ and
$p_i q_i = 10$ implies $p_{i+1} q_{i+1} = 10$

The Karnaugh map for the outputs are shown in Fig. 4.68.

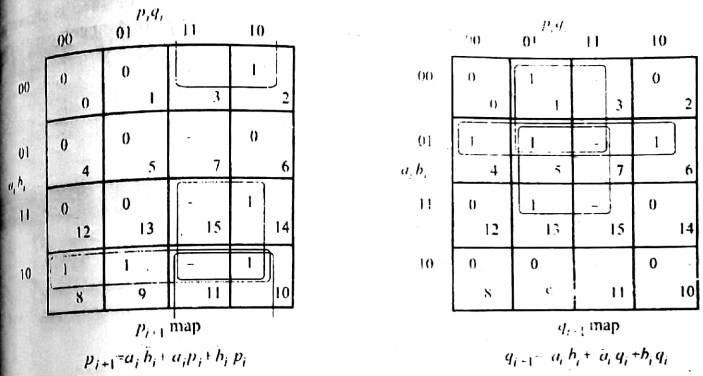


Fig. 4.68 Karnaugh map of Fig. 4.67

The truth table, Karnaugh map and implementation of the output converter is shown in Fig. 4.69.

Cell no.	p_i	q_i	$a > b$ x	$a = b$ y	$a < b$ z
0	0	0	0	1	0
1	0	1	0	0	1
2	1	0	1	0	0
3	1	1	–	–	–

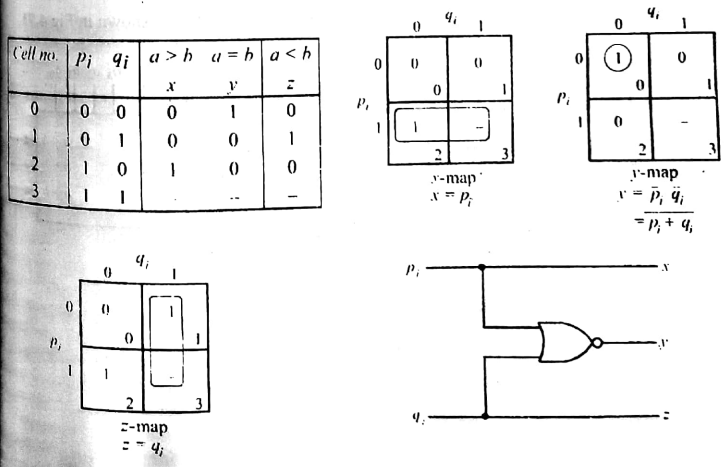


Fig. 4.69 Design of output converter

4.3.4 IC Comparator

Four bit comparators are available in IC form as 7485. The logic design of 7485 is shown in Fig. 4.70. It has boundary inputs $a < b$, $a = b$ and $a > b$ to allow cascading of 4 bit comparators to configure $4n$ bit comparators.

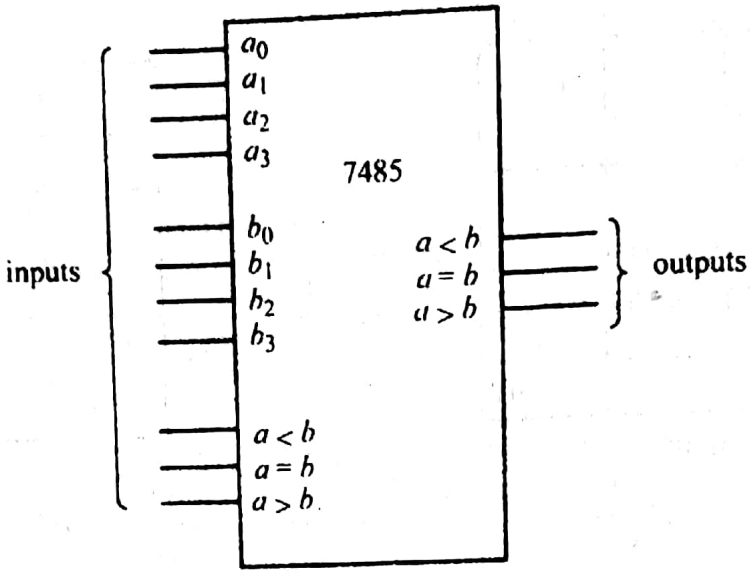


Fig. 4.71 4-bit IC comparator

Example 4.25

Implement a 12-bit comparator using IC 7485.

Solution: The boundary inputs and outputs are used to cascade the ICs as shown in Fig 4.71.

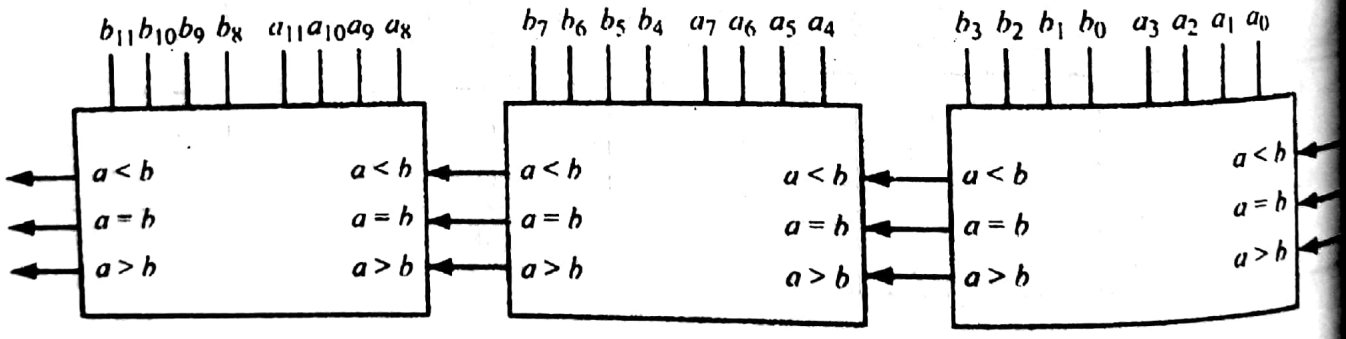


Fig. 4.71 A 12-bit comparator using four-7485 ICs