

## UNIT-IV

# Combinational Circuits

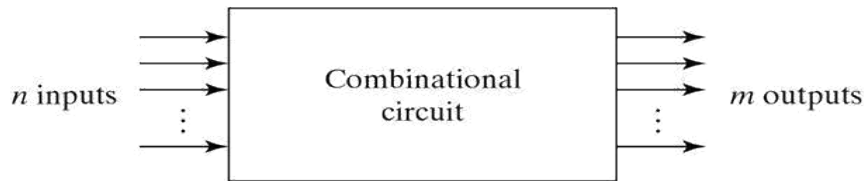


Fig. Block Diagram of Combinational Circuit

### Combinational Logic

- Logic circuits for digital systems may be combinational or sequential.
- A combinational circuit consists of input variables, logic gates, and output variables

### Analysis procedure

To obtain the output Boolean functions from a logic diagram, proceed as follows:

1. Label all gate outputs that are a function of input variables with arbitrary symbols. Determine the Boolean functions for each gate output.
2. Label the gates that are a function of input variables and previously labeled gates with other arbitrary symbols. Find the Boolean functions for these gates.
3. Repeat the process outlined in step 2 until the outputs of the circuit are obtained.
4. By repeated substitution of previously defined functions, obtain the output Boolean functions in terms of input variables.

### Design Procedure:

1. The problem is stated
2. The number of available input variables and required output variables is determined.
3. The input and output variables are assigned letter symbols.
4. The truth table that defines the required relationship between inputs and outputs is derived.
5. The simplified Boolean function for each output is obtained.
6. The logic diagram is drawn.

### BOOLEAN ALGEBRA :

#### Basic Definitions

- Axiomatic Definition of Boolean Algebra
- Basic Theorems and properties of Boolean Algebra
- Boolean Functions
- Canonical and Standard Forms, Other Logic Operations
- Digital Logic Gates
- Integrated Circuits

**Boolean Algebra:** Boolean algebra, like any other deductive mathematical system, may be defined with a set of elements, a set of operators, and a number of unproved axioms or postulates. A *set* of elements is any collection of objects having a common property. If  $S$  is a set and  $x$  and  $y$  are certain objects, then  $x \in S$  denotes that  $x$  is a member of the set  $S$ , and  $y \notin S$  denotes that  $y$  is not an element of  $S$ . A set with a denumerable number of elements is specified by braces:  $A = \{1, 2, 3, 4\}$ , i.e. the elements of set  $A$  are the numbers 1, 2, 3, and 4. A *binary operator* defined on a set  $S$  of elements is a rule that assigns to each pair of elements from  $S$  a unique element from  $S$ . Example: In  $a * b = c$ , we say that  $*$  is a binary operator if it specifies a rule for finding  $c$  from the pair  $(a, b)$  and also if  $a, b, c \in S$ .

**CLOSURE:** The Boolean system is *closed* with respect to a binary operator if for every pair of Boolean values, it produces a Boolean result. For example, logical AND is closed in the Boolean system because it accepts only Boolean operands and produces only Boolean results.

\_ A set  $S$  is closed with respect to a binary operator if, for every pair of elements of  $S$ , the binary operator specifies a rule for obtaining a unique element of  $S$ .

For example, the set of natural numbers  $N = \{1, 2, 3, 4, \dots, 9\}$  is closed with respect to the binary operator plus (+) by the rule of arithmetic addition, since for any  $a, b \in N$  we obtain a unique  $c \in N$  by the operation  $a + b = c$ .

**ASSOCIATIVE LAW:** A binary operator  $*$  on a set  $S$  is said to be associative whenever  $(x * y) * z = x * (y * z)$  for all  $x, y, z \in S$ , for all Boolean values  $x, y$  and  $z$ .

### COMMUTATIVE LAW:

A binary operator  $*$  on a set  $S$  is said to be commutative whenever  $x * y = y * x$  for all  $x, y, z \in S$ .

### IDENTITY ELEMENT:

A set  $S$  is said to have an identity element with respect to a binary operation  $*$  on  $S$  if there exists an element  $e \in S$  with the property  $e * x = x * e = x$  for every  $x \in S$ .

## BASIC IDENTITIES OF BOOLEAN ALGEBRA

*Postulate 1 (Definition):* A Boolean algebra is a closed algebraic system containing a set  $K$  of two or more elements and the two operators  $\cdot$  and  $+$  which refer to logical AND and logical OR

$$\blacksquare x + 0 = x$$

$$\blacksquare x \cdot 0 = 0$$

$$\blacksquare x + 1 = 1$$

$$\blacksquare x \cdot 1 = x$$

$$\blacksquare x + x = x$$

$$\blacksquare x \cdot x = x$$

$$\blacksquare x + x' = 1$$

$$\blacksquare x \cdot x' = 0$$

$$\blacksquare x + y = y + x$$

$$\blacksquare xy = yx$$

$$\blacksquare x + (y + z) = (x + y) + z$$

$$\blacksquare x(yz) = (xy)z$$

$$\blacksquare x(y + z) = xy + xz$$

$$\blacksquare x + yz = (x + y)(x + z)$$

$$\blacksquare (x + y)' = x' y'$$

$$\blacksquare (xy)' = x' + y'$$

$$\blacksquare (x')' = x$$

## DeMorgan's Theorem

(a)	$(a + b)' = a'b'$
(b)	$(ab)' = a' + b'$
Generalized DeMorgan's Theorem	
(a)	$(a + b + \dots z)' = a'b' \dots z'$
(b)	$(a.b \dots z)' = a' + b' + \dots z',,$

## AXIOMATIC DEFINITION OF BOOLEAN ALGEBRA:

1. Closure
  - a. Closure with respect to (wrt) OR (+)
  - b. Closure with respect to AND ( $\bullet$ )
2. Identity
  - a. Identity element wrt to OR : 0
  - b. Identity element wrt to AND : 1
- 3.

### Commutative Property

- a. Commutative Property wrt to OR :  $x + y = y + x$
- b. Commutative Property wrt to AND :  $x \cdot y = y \cdot x$

### 4. Distributive Property

- a.  $x \cdot (y + z) = (x \cdot y) + (x \cdot z)$
- b.  $x + (y \cdot z) = (x + y)(x + z)$

### Existence of Complement









- a.  $x + x'' = 1$
- b.  $x \cdot x'' = 0$

## LOGIC GATES

Formal logic: In formal logic, a statement (proposition) is a declarative sentence that is either true(1) or false (0). It is easier to communicate with computers using formal logic.

- Boolean variable: Takes only two values – either true (1) or false (0). They are used as basic units of formal logic.

- Boolean algebra: Deals with binary variables and logic operations operating on those variables.
- Logic diagram: Composed of graphic symbols for logic gates. A simple circuit sketch that represents inputs and outputs of Boolean functions.

Name	Graphic symbol	Algebraic function	Truth table															
AND		$F = x \cdot y$	<table><tr><th>x</th><th>y</th><th>F</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	x	y	F	0	0	0	0	1	0	1	0	0	1	1	1
x	y	F																
0	0	0																
0	1	0																
1	0	0																
1	1	1																
OR		$F = x + y$	<table><tr><th>x</th><th>y</th><th>F</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	x	y	F	0	0	0	0	1	1	1	0	1	1	1	1
x	y	F																
0	0	0																
0	1	1																
1	0	1																
1	1	1																
Inverter		$F = x'$	<table><tr><th>x</th><th>F</th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table>	x	F	0	1	1	0									
x	F																	
0	1																	
1	0																	
Buffer		$F = x$	<table><tr><th>x</th><th>F</th></tr><tr><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td></tr></table>	x	F	0	0	1	1									
x	F																	
0	0																	
1	1																	
NAND		$F = (xy)'$	<table><tr><th>x</th><th>y</th><th>F</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	x	y	F	0	0	1	0	1	1	1	0	1	1	1	0
x	y	F																
0	0	1																
0	1	1																
1	0	1																
1	1	0																
NOR		$F = (x + y)'$	<table><tr><th>x</th><th>y</th><th>F</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	x	y	F	0	0	1	0	1	0	1	0	0	1	1	0
x	y	F																
0	0	1																
0	1	0																
1	0	0																
1	1	0																
Exclusive-OR (XOR)		$F = xy' + x'y$ $= x \oplus y$	<table><tr><th>x</th><th>y</th><th>F</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	x	y	F	0	0	0	0	1	1	1	0	1	1	1	0
x	y	F																
0	0	0																
0	1	1																
1	0	1																
1	1	0																
Exclusive-NOR or equivalence		$F = xy + x'y'$ $= (x \oplus y)'$	<table><tr><th>x</th><th>y</th><th>F</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	x	y	F	0	0	1	0	1	0	1	0	0	1	1	1
x	y	F																
0	0	1																
0	1	0																
1	0	0																
1	1	1																

## Properties of XOR Gates

1. XOR (also  $\oplus$ ) : the “not-equal” function

$$2. \text{XOR}(X,Y) = X \oplus Y = X'Y + XY'$$

3. Identities:

- $X \oplus 0 = X$
- $X \oplus 1 = X'$
- $X \oplus X = 0$
- $X \oplus X' = 1$

4. Properties:

- $X \oplus Y = Y \oplus X$
- $(X \oplus Y) \oplus W = X \oplus (Y \oplus W)$

## Universal Logic Gates

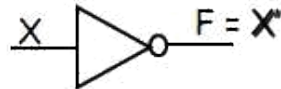
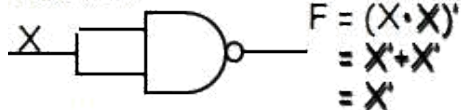
NAND and NOR gates are called Universal gates. All fundamental gates (NOT, AND, OR) can be realized by using either only NAND or only NOR gate. A universal gate provides flexibility and offers enormous advantage to logic designers.

### NAND as a Universal Gate

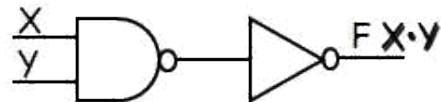
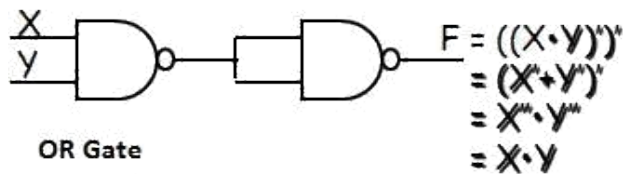
NAND Known as a “universal” gate because ANY digital circuit can be implemented with NAND gates alone.

To prove the above, it suffices to show that AND, OR, and NOT can be implemented using NAND gates only.

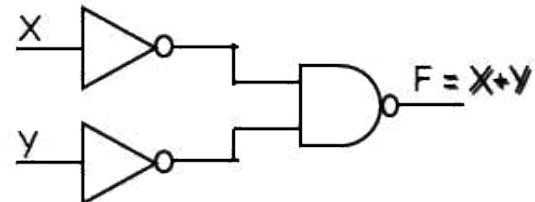
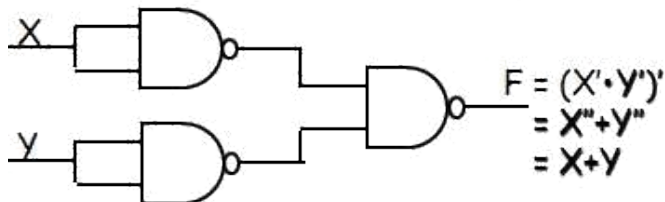
**NOT Gate**



**AND Gate**



**OR Gate**



## Canonical and Standard Forms

We need to consider formal techniques for the simplification of Boolean functions.

Identical functions will have exactly the same canonical form.

5. Minterms and Maxterms
6. Sum-of-Minterms and Product-of- Maxterms
7. Product and Sum terms
8. Sum-of-Products (SOP) and Product-of-Sums (POS)

## Definitions

**Literal:** A variable or its complement

**Product term:** literals connected by •

**Sum term:** literals connected by +

**Minterm:** a product term in which all the variables appear exactly once, either complemented or uncomplemented.

**Maxterm:** a sum term in which all the variables appear exactly once, either complemented or uncomplemented.

**Canonical form:** Boolean functions expressed as a sum of Minterms or product of Maxterms are said to be in canonical form.

### Minterm

3. Represents exactly one combination in the truth table.
4. Denoted by  $m_j$ , where  $j$  is the decimal equivalent of the minterm's corresponding binary combination ( $b_j$ ).
5. A variable in  $m_j$  is complemented if its value in  $b_j$  is 0, otherwise is uncomplemented.

Example: Assume 3 variables (A, B, C), and  $j=3$ . Then,  $b_j = 011$  and its corresponding minterm is denoted by  $m_j = A'BC$

### Maxterm

7. Represents exactly one combination in the truth table.
8. Denoted by  $M_j$ , where  $j$  is the decimal equivalent of the maxterm's corresponding binary combination ( $b_j$ ).
9. A variable in  $M_j$  is complemented if its value in  $b_j$  is 1, otherwise is uncomplemented.

Example: Assume 3 variables (A, B, C), and  $j=3$ . Then,  $b_j = 011$  and its corresponding maxterm is denoted by  $M_j = A+B'+C'$

### Truth Table notation for Minterms and Maxterms



Minterms and Maxterms are easy to denote using a truth table.

Example: Assume 3 variables x,y,z (order is fixed)

x	y	z	Minterm	Maxterm
0	0	0	$x'y'z' = m_0$	$x+y+z = M_0$
0	0	1	$x'y'z = m_1$	$x+y+z' = M_1$
0	1	0	$x'yz' = m_2$	$x+y'+z = M_2$
0	1	1	$x'yz = m_3$	$x+y'+z' = M_3$
1	0	0	$xy'z' = m_4$	$x'+y+z = M_4$
1	0	1	$xy'z = m_5$	$x'+y+z' = M_5$
1	1	0	$xyz' = m_6$	$x'+y'+z = M_6$
1	1	1	$xyz = m_7$	$x'+y'+z' = M_7$

### Canonical Forms



- ☐ Every function  $F()$  has two canonical forms:
- Canonical Sum-Of-Products (sum of minterms)
  - Canonical Product-Of-Sums (product of maxterms)

Canonical Sum-Of-Products:

The minterms included are those  $m_j$  such that  $F() = 1$  in row  $j$  of the truth table for  $F()$ .

Canonical Product-Of-Sums:

The maxterms included are those  $M_j$  such that  $F() = 0$  in row  $j$  of the truth table for  $F()$ .

## Example

Consider a Truth table for  $f_1(a,b,c)$  at right

The canonical sum-of-products form for  $f_1$

is  $f_1(a,b,c) = m_1 + m_2 + m_4 + m_6$

$$= a'b'c + a'bc' + ab'c' + abc'$$

The canonical product-of-sums form for  $f_1$

is  $f_1(a,b,c) = M_0 \cdot M_3 \cdot M_5 \cdot M_7$

$$(a+b+c) \cdot (a+b'+c') \cdot (a'+b+c') \cdot (a'+b'+c').$$

a	b	c	$f_1$
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

- ☐ Observe that:  $m_j = M_j'$

## Shorthand: $\sum$ and $\prod$

- $f_1(a,b,c) = \sum m(1,2,4,6)$ , where  $\sum$  indicates that this is a sum-of-products form, and  $m(1,2,4,6)$  indicates that the minterms to be included are  $m_1$ ,  $m_2$ ,  $m_4$ , and  $m_6$ .
- $f_1(a,b,c) = \prod M(0,3,5,7)$ , where  $\prod$  indicates that this is a product-of-sums form, and  $M(0,3,5,7)$  indicates that the maxterms to be included are  $M_0$ ,  $M_3$ ,  $M_5$ , and  $M_7$ .
- Since  $m_j = M_j'$  for any  $j$ ,  
 $m(1,2,4,6) = \prod M(0,3,5,7) = f_1(a,b,c)$

•

## Conversion between Canonical Forms

- Replace  $\sum$  with  $\prod$  (or *vice versa*) and replace those  $j$ 's that appeared in the original form with those that do not.

■ Example:

$$f_1(a,b,c) = a'b'c + a'bc' + ab'c' + abc'$$

$$m_1 + m_2 + m_4 + m_6$$

$$\sum(1,2,4,6)$$

$$\prod(0,3,5,7)$$

$$\blacksquare (a+b+c) \cdot (a+b'+c') \cdot (a'+b+c') \cdot (a'+b'+c')$$

## Standard Forms

Another way to express Boolean functions is in standard form. In this configuration, the terms that form the function may contain one, two, or any number of literals.

There are two types of standard forms: the sum of products and products of sums.

The sum of products is a Boolean expression containing AND terms, called product terms, with one or more literals each. The sum denotes the ORing of these terms. An example of a function expressed as a sum of products is

$$F_1 = y' + xy + x'yz'$$

The expression has three product terms, with one, two, and three literals. Their sum is, in effect, an OR operation.

A product of sums is a Boolean expression containing OR terms, called sum terms. Each term may have any number of literals. The product denotes the ANDing of these terms. An example of a function expressed as a product of sums is

$$F_2 = x(y' + z)(x' + y + z')$$

This expression has three sum terms, with one, two, and three literals. The product is an AND operation.

## Conversion of SOP from standard to canonical form

### Example-1.

Express the Boolean function  $F = A + B'C$  as a sum of minterms.

Solution: The function has three variables: A, B, and C. The first term A is missing two variables; therefore,  $A = A(B+B') = AB + AB'$

This function is still missing one variable, so

$$\begin{aligned} A &= AB(C + C') + AB'(C + C') \\ &= ABC + ABC' + AB'C + AB'C' \end{aligned}$$

The second term  $B'C$  is missing one variable; hence,

$$B'C = B'C(A + A') = AB'C + A'B'C$$

Combining all terms, we have

$$\begin{aligned} F &= A + B'C \\ &= ABC + ABC' + AB'C + AB'C' + A'B'C \end{aligned}$$

But  $AB'C$  appears twice, and according to theorem  $(x + x = x)$ , it is possible to remove one of those occurrences. Rearranging the minterms in ascending order, we finally obtain

$$\begin{aligned} F &= A'B'C + AB'C + AB'C' + ABC' + ABC \\ &= m_1 + m_4 + m_5 + m_6 + m_7 \end{aligned}$$

When a Boolean function is in its sum-of-minterms form, it is sometimes convenient to express the function in the following brief notation:

$$F(A, B, C) = \sum m(1, 4, 5, 6, 7)$$

### Example-2.

Express the Boolean function  $F = xy + x'z$  as a product of maxterms.

Solution: First, convert the function into OR terms by using the distributive law:

$$\begin{aligned} F &= xy + x'z = (xy + x')(xy + z) \\ &\quad (x + x')(y + x')(x + z)(y + z) \\ &= (x' + y)(x + z)(y + z) \end{aligned}$$

The function has three variables: x, y, and z. Each OR term is missing one variable;

$$\text{therefore, } x' + y = x' + y + zz' = (x' + y + z)(x' + y + z')$$

$$x + z = x + z + yy' = (x + y + z)(x + y' + z)$$

$$y + z = y + z + xx' = (x + y + z)(x' + y + z)$$

Combining all the terms and removing those which appear more than once, we finally obtain

$$\begin{aligned} F &= (x + y + z)(x + y' + z)(x' + y + z)(x' + y + z) \\ F &= M_0 M_2 M_4 M_5 \end{aligned}$$

A convenient way to express this function is as

$$\text{follows: } F(x, y, z) = \pi M(0, 2, 4, 5)$$

The product symbol,  $\pi$ , denotes the ANDing of maxterms; the numbers are the indices of the maxterms of the function.

## Minimization Techniques

### Two-variable k-map:

A two-variable k-map can have  $2^2=4$  possible combinations of the input variables A and B. Each of these combinations,  $\bar{A}\bar{B}$ ,  $\bar{A}B$ ,  $AB$  (in the SOP form) is called a minterm. The minterm may be represented in terms of their decimal designations – m0 for  $\bar{A}\bar{B}$ , m1 for  $\bar{A}B$ , m2 for  $AB$ , and m3 for  $AB$ , assuming that A represents the MSB. The letter m stands for minterm and the subscript represents the decimal designation of the minterm. The presence or absence of a minterm in the expression indicates that the output of the logic circuit assumes logic 1 or logic 0 level for that combination of input variables.

The expression  $F = \bar{A}\bar{B} + \bar{A}B + AB$ , it can be expressed using min

$$\text{term as } F = m_0 + m_2 + m_3 = \sum m(0, 2, 3)$$

Using Truth Table:

Minterm	Inputs		Output F
	A	B	
0	0	0	1
1	0	1	0
2	1	0	1
3	1	1	1

A 1 in the output contains that particular minterm in its sum and a 0 in that column indicates that the particular minterm does not appear in the expression for output. This information can also be indicated by a two-variable k-map.

### Mapping of SOP Expressions:

A two-variable k-map has  $2^2=4$  squares. These squares are called cells. Each square on the k-map represents a unique minterm. The minterm designation of the squares are placed in any square, indicates that the corresponding minterm does output expressions. And a 0 or no entry in any square indicates that the corresponding minterm does not appear in the expression for output.

		B	
		0	1
A	0	$\bar{A}\bar{B}$	$\bar{A}B$
	1	$AB$	$AB$

The minterms of a two-variable k-map

The mapping of the expressions  $=\sum m(0,2,3)$  is

	<b>B</b>	<b>0</b>	<b>1</b>
<b>A</b>			
<b>0</b>		<sup>0</sup> <b>1</b>	<sup>1</sup> <b>0</b>
<b>1</b>		<sup>2</sup> <b>1</b>	<sup>3</sup> <b>1</b>

k-map of  $\sum m(0,2,3)$

**EX:** Map the expressions  $f = B + A$

$F = m_1 + m_2 = \sum m(1,2)$  The k-map is

	<b>B</b>	<b>0</b>	<b>1</b>
<b>A</b>			
<b>0</b>	<b>0</b>	<sup>0</sup>	<sup>1</sup> <b>1</b>
<b>1</b>	<b>1</b>	<sup>2</sup>	<sup>3</sup> <b>0</b>

### Minimizations of SOP expressions:

To minimize Boolean expressions given in the SOP form by using the k-map, look for adjacent adjacent squares having 1's minterms adjacent to each other, and combine them to form larger squares to eliminate some variables. Two squares are said to be adjacent to each other, if their minterms differ in only one variable. (i.e, B & A differ only in one variable. so they may be combined to form a 2-square to eliminate the variable B. similarly all other.

The necessary condition for adjacency of minterms is that their decimal designations must differ by a power of 2. A minterm can be combined with any number of minterms adjacent to it to form larger squares. Two minterms which are adjacent to each other can be combined to form a bigger square called a 2-square or a pair. This eliminates one variable – the variable that is not common to both the minterms. For EX:

$m_0$  and  $m_1$  can be combined to yield,

$$f_1 = m_0 + m_1 = B +$$

$m_0$  and  $m_2$  can be combined to yield,

$$f = m_0 + m_2 = A$$

$m_1$  and  $m_3$  can be combined to yield,

$$f_3 = m_1 + m_3 = B + AB = B(1 + A) = B$$

$m_2$  and  $m_3$  can be combined to yield,

$$f_4 = m_2 + m_3 = A + AB = A(B + 1) = A$$

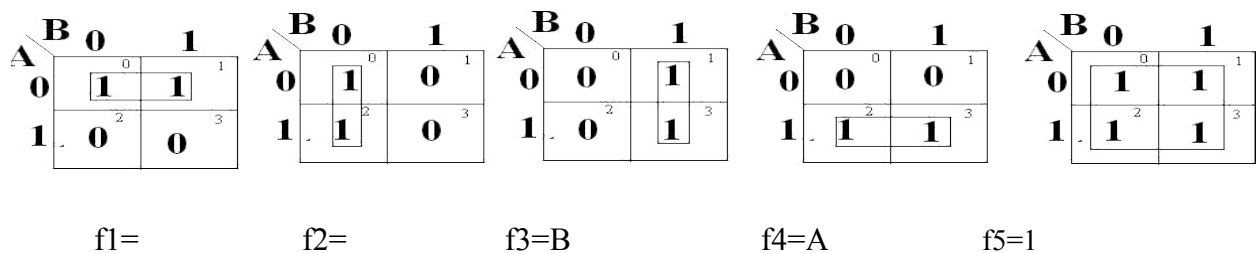
$m_0, m_1, m_2$  and  $m_3$  can be combined to yield,

$$f_1 = m_0 + m_1 = A + AB$$

$$f_2 = (B + 1) + A(B + 1)$$

$$f_5 = 1$$

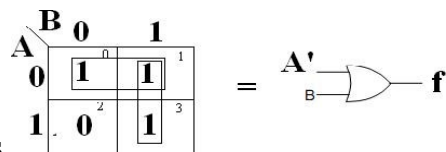
$$= 1$$



The possible minterm groupings in a two-variable k-map.

Two 2-squares adjacent to each other can be combined to form a 4-square. A 4-square eliminates 2 variables. A 4-square is called a quad. To read the squares on the map after minimization, consider only those variables which remain constant through the square, and ignore the variables which are varying. Write the non complemented variable if the variable is remaining constant as a 1, and the complemented variable if the variable is remaining constant as a 0, and write the variables as a product term. In the above figure  $f_1$  read as  $A'$ , because, along the square,  $A$  remains constant as a 0, that is, as  $A'$ , whereas  $B$  is changing from 0 to 1.

**EX:** Reduce the minterm  $f = A + AB$  using mapping Expressed in terms of minterms, the given expression is  $F = m_0 + m_1 + m_2 + m_3 = m \sum(0, 1, 2, 3)$  & the figure shows the k-map for  $f$  and its reduction. In one 2-square,  $A$  is constant as a 0 but  $B$  varies from a 0 to a 1, and in the other 2-square,  $B$  is constant as a 1 but  $A$  varies from a 0 to a 1. So, the reduced expressions is  $A' + B$ .



It requires two gate inputs for realization as

$f = A' + B$  (k-map in SOP form, and logic diagram.)

The main criterion in the design of a digital circuit is that its cost should be as low as possible. For that the expression used to realize that circuit must be minimal. Since the cost is proportional to number of gate inputs in the circuit, an expression is considered minimal only if it corresponds to the least possible number of gate inputs. & there is no guarantee for that k-map in SOP is the real minimal. To obtain real minimal expression, obtain the minimal expression both in SOP & POS form by using k-maps and take the minimal of these two minimals.

The 1's on the k-map indicate the presence of minterms in the output expressions, where as the 0s indicate the absence of minterms. Since the absence of a minterm in the SOP expression means the presence of the corresponding maxterm in the POS expression of the same. when a SOP expression is plotted on the k-map, 0s or no entries on the k-map represent the maxterms. To obtain the minimal expression in the POS form, consider the 0s on the k-map and follow the procedure used for combining 1s. Also, since the absence of a maxterm in the POS expression means the presence of the corresponding minterm in the SOP expression of the same, when a POS expression is plotted on the k-map, 1s or no entries on the k-map represent the minterms.

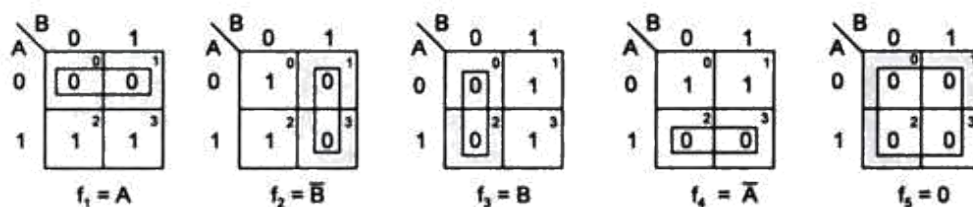
### Mapping of POS expressions:

Each sum term in the standard POS expression is called a maxterm. A function in two variables (A, B) has four possible maxterms,  $A+B$ ,  $A+\bar{B}$ ,  $\bar{A}+B$ ,  $\bar{A}+\bar{B}$ .

. They are represented as  $M_0$ ,  $M_1$ ,  $M_2$ , and  $M_3$  respectively. The uppercase letter M stands for maxterm and its subscript denotes the decimal designation of that maxterm obtained by treating the non-complemented variable as a 0 and the complemented variable as a 1 and putting them side by side for reading the decimal equivalent of the binary number so formed.

For mapping a POS expression on to the k-map, 0s are placed in the squares corresponding to the maxterms which are presented in the expression and 1s are placed in the squares corresponding to the maxterm which are not present in the expression. The decimal designation of the squares of the squares for maxterms is the same as that for the minterms. A two-variable k-map & the associated maxterms are as the maxterms of a two-variable k-map

The possible maxterm groupings in a two-variable k-map



### Minimization of POS Expressions:

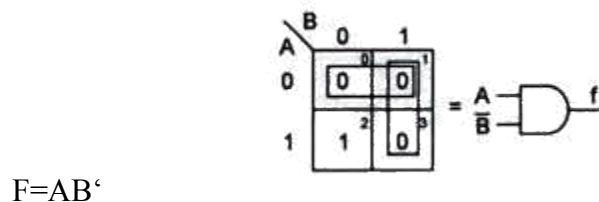
To obtain the minimal expression in POS form, map the given POS expression on to the K-map and combine the adjacent 0s into as large squares as possible. Read the squares putting the complemented variable if its value remains constant as a 1 and the non-complemented variable if its value remains constant as a 0 along the entire square ( ignoring the variables which do not remain constant throughout the square) and then write them as a sum term.

Various maxterm combinations and the corresponding reduced expressions are shown in figure. In this  $f_1$  read as  $A$  because  $A$  remains constant as a 0 throughout the square and  $B$  changes from a 0 to a 1.  $f_2$  is read as  $B'$  because  $B$  remains constant along the square as a 1 and  $A$  changes from a 0 to a 1.  $f_3$

Is read as a 0 because both the variables are changing along the square.

**Ex:** Reduce the expression  $f = (A+B)(A+B')(A'+B')$  using mapping.

The given expression in terms of maxterms is  $f = \pi M(0,1,3)$ . It requires two gates inputs for realization of the reduced expression as



K-map in POS form and logic diagram

In this given expression, the maxterm  $M_2$  is absent. This is indicated by a 1 on the k-map. The corresponding SOP expression is  $\sum m_2$  or  $AB'$ . This realization is the same as that for the POS form.

### Three-variable K-map:

A function in three variables ( $A, B, C$ ) expressed in the standard SOP form can have eight possible combinations:  $ABC$ ,  $AB\bar{C}$ ,  $A\bar{B}C$ ,  $A\bar{B}\bar{C}$ ,  $AB\bar{C}$ ,  $AB\bar{C}$ ,  $ABC$ , and  $ABC$ . Each one of these combinations designate d by  $m_0, m_1, m_2, m_3, m_4, m_5, m_6$ , and  $m_7$ , respectively, is called a minterm.  $A$  is the MSB of the minterm designator and  $C$  is the LSB.

In the standard POS form, the eight possible combinations are:  $A+B+C$ ,  $A+B+\bar{C}$ ,  $A+B+C$ ,  $A+B+\bar{C}$ ,  $A+B+C$ ,  $A+B+\bar{C}$ ,  $A+B+C$ , and  $A+B+\bar{C}$ . Each one of these combinations designated by  $M_0, M_1, M_2, M_3, M_4, M_5, M_6$ , and  $M_7$  respectively is called a maxterm.  $A$  is the MSB of the maxterm designator and  $C$  is the LSB.

A three-variable k-map has, therefore,  $8(=2^3)$  squares or cells, and each square on the map represents a minterm or maxterm as shown in figure. The small number on the top right corner of each cell indicates the min term or max term designation.



		BC			
		00	01	11	10
A	0	$\bar{A}\bar{B}\bar{C}^0$	$\bar{A}\bar{B}C^1$	$\bar{A}B\bar{C}^3$	$\bar{A}BC^2$
	1	$A\bar{B}\bar{C}^4$	$A\bar{B}C^5$	$AB\bar{C}^7$	$ABC^6$

(a) Minterms

		BC			
		00	01	11	10
A	0	$A+B+C^0$	$A+B+\bar{C}^1$	$A+\bar{B}+\bar{C}^3$	$A+\bar{B}+C^2$
	1	$\bar{A}+B+C^4$	$\bar{A}+B+\bar{C}^5$	$\bar{A}+\bar{B}+\bar{C}^7$	$\bar{A}+\bar{B}+C^6$

(b) Maxterms

The three-variable k-map.

The binary numbers along the top of the map indicate the condition of B and C for each column. The binary number along the left side of the map against each row indicates the condition of A for that row. For example, the binary number 01 on top of the second column in fig indicates that the variable B appears in complemented form and the variable C in non-complemented form in all the minterms in that column. The binary number 0 on the left of the first row indicates that the variable A appears in complemented form in all the minterms in that row, the binary numbers along the top of the k-map are not in normal binary order. They are, infact, in the Gray code. This is to ensure that twophysically adjacent squares are really adjacent, i.e., their minterms or maxterms differ by only one variable.

Ex: Map the expression  $f = C + \dots + \dots + ABC$

In the given expression , the minterms are :  $C=001=m_1$  ;  $=101=m_5$ ;  
 $=010=m_2$ ;

$=110=m_6$ ;  $ABC=111=m_7$ .

So the expression is  $f = \sum m(1,5,2,6,7) = \sum m(1,2,5,6,7)$ . The corresponding k-map is

		BC			
		00	01	11	10
A	0	0 <sup>0</sup>	1 <sup>1</sup>	0 <sup>3</sup>	1 <sup>2</sup>
	1	0 <sup>4</sup>	1 <sup>5</sup>	1 <sup>7</sup>	1 <sup>6</sup>

K-map in SOP form  
 $++)(++)(A++)(++)$

Ex: Map the expression  $f = (A+B+C)$ ,

(

In the given expression the maxterms are  
 $:A+B+C=000=M_0$ ;  $+++=101=M_5$ ;  $+++=111=M_7$ ;  $A+++=011=M_3$ ;  $+++=110=M_6$ .

So the expression is  $f = \pi M (0,5,7,3,6) = \pi M (0,3,5,6,7)$ . The mapping of the expression is

BC		00	01	11	10
		0	1	3	2
A	0	0 <sup>0</sup>	1 <sup>1</sup>	0 <sup>3</sup>	1 <sup>2</sup>
	1	1 <sup>4</sup>	0 <sup>5</sup>	0 <sup>7</sup>	0 <sup>6</sup>

K-map in POS form.

### Minimization of SOP and POS expressions:

For reducing the Boolean expressions in SOP (POS) form plotted on the k-map, look at the 1s (0s) present on the map. These represent the minterms (maxterms). Look for the minterms (maxterms) adjacent to each other, in order to combine them into larger squares. Combining of adjacent squares in a k-map containing 1s (or 0s) for the purpose of simplification of a SOP (or POS) expression is called *looping*. Some of the minterms (maxterms) may have many adjacencies. Always start with the minterms (maxterm) with the least number of adjacencies and try to form as large as large a square as possible. The larger must form a geometric square or rectangle. They can be formed even by wrapping around, but cannot be formed by using diagonal configurations. Next consider the minterm (maxterm) with next to the least number of adjacencies and form as large a square as possible. Continue this till all the minterms (maxterms) are taken care of . A minterm (maxterm) can be part of any number of squares if it is helpful in reduction. Read the minimal expression from the k-map, corresponding to the squares formed. There can be more than one minimal expression.

Two squares are said to be adjacent to each other (since the binary designations along the top of the map and those along the left side of the map are in Gray code), if they are physically adjacent to each other, or can be made adjacent to each other by wrapping around. For squares to be combinable into bigger squares it is essential but not sufficient that their minterm designations must differ by a power of two.

General procedure to simplify the Boolean expressions:

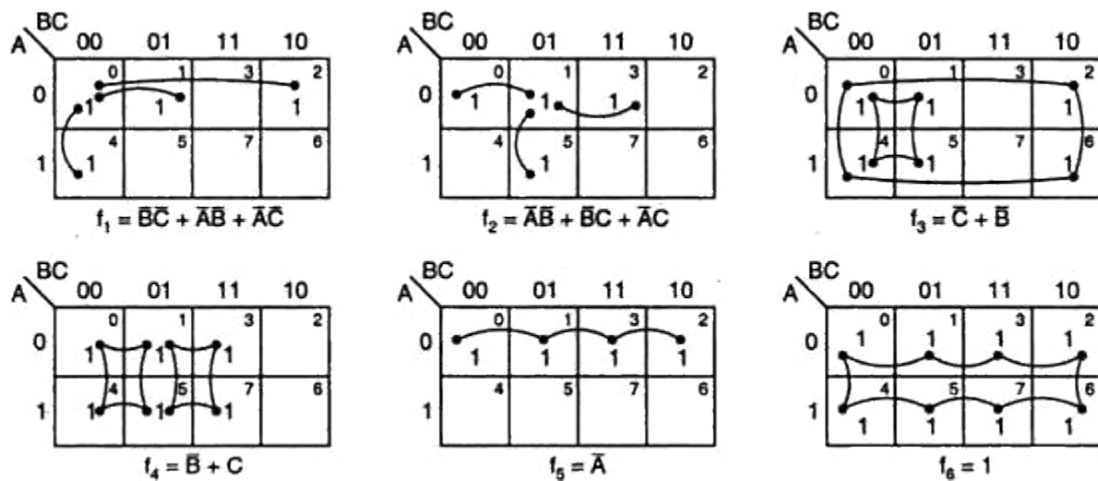
- T Plot the k-map and place 1s(0s) corresponding to the minterms (maxterms) of the SOP (POS) expression.
- U Check the k-map for 1s(0s) which are not adjacent to any other 1(0). They are isolated minterms(maxterms) . They are to be read as they are because they cannot be combined even into a 2-square.
- V Check for those 1s(0s) which are adjacent to only one other 1(0) and make them pairs (2 squares).
- W Check for quads (4 squares) and octets (8 squares) of adjacent 1s (0s) even if they contain some 1s(0s) which have already been combined. They must geometrically form a square or a rectangle.
- X Check for any 1s(0s) that have not been combined yet and combine them into bigger squares if possible.
- Y Form the minimal expression by summing (multiplying) the product the product (sum) terms of all the groups.

### Reading the K-maps:

While reading the reduced k-map in SOP (POS) form, the variable which remains constant as 0 along the square is written as the complemented (non-complemented) variable and the one which remains constant as 1 along the square is written as non-complemented (complemented) variable and the term as a product (sum) term. All the product (sum) terms are added (multiplied).

Some possible combinations of minterms and the corresponding minimal expressions read from the k-maps are shown in fig: Here  $f_6$  is read as 1, because along the 8-square no variable remains constant.  $f_5$  is read as  $\bar{A}$ , because, along the 4-square formed by  $m_0, m_1, m_2$  and  $m_3$ , the variables B and C are changing, and A remains constant as a 0. Algebraically,

$$\begin{aligned} f_5 &= m_0 + m_1 + m_2 + m_3 \\ &= \bar{A}\bar{B}\bar{C} + \bar{A}B\bar{C} + \bar{A}\bar{B}C + \bar{A}BC \\ &= \bar{A}(\bar{B}\bar{C} + B\bar{C} + \bar{B}C + BC) \\ &= \bar{A}(\bar{B}(\bar{C} + C) + B(\bar{C} + C)) \\ &= \bar{A}(\bar{B} + B) \\ &= \bar{A} \end{aligned}$$

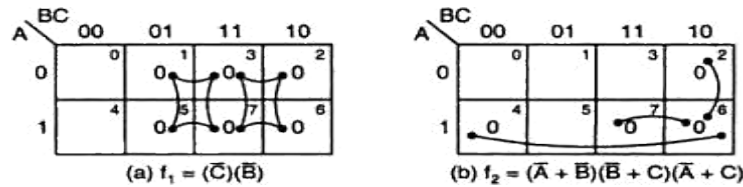


$f_3$  is read as  $\bar{A}$ , because in the 4-square formed by  $m_0, m_2, m_6$ , and  $m_4$ , the variable A and B are changing, whereas the variable C remains constant as a 0. So it is read as  $\bar{A}$ . In the 4-square formed by  $m_0, m_1, m_4, m_5$ , A and C are changing but B remains constant as a 0. So it is read as  $\bar{B}$ . So, the resultant expression for  $f_3$  is the sum of these two, i.e.,  $\bar{A} + \bar{B}$ .

$f_1$  is read as  $\bar{A}\bar{B}$ , because in the 2-square formed by  $m_0$  and  $m_4$ , A is changing from a 0 to a 1. Whereas B and C remain constant as a 0. So it is read as  $\bar{A}\bar{B}$ . In the 2-square formed by  $m_0$  and  $m_1$ , C is changing from a 0 to a 1, whereas A and B remain constant as a 0. So it is read as  $\bar{A}\bar{B}C$ . In the 2-square formed by  $m_0$  and  $m_2$ , B is changing from a 0 to a 1 whereas A and C remain constant as a 0. So, it is read as  $\bar{A}\bar{B}C$ . Therefore, the resultant SOP expression is

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

Some possible maxterm groupings and the corresponding minimal POS expressions read from the k-map are



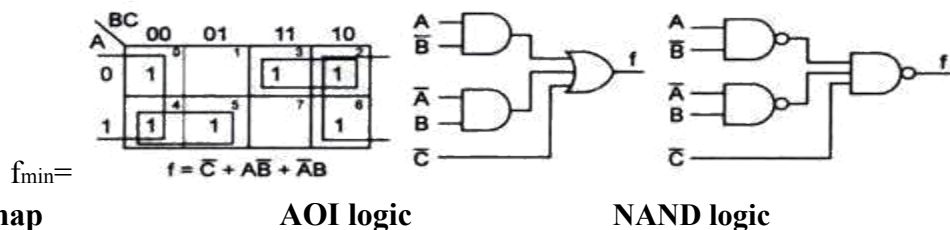
In this figure, along the 4-square formed by M<sub>1</sub>, M<sub>3</sub>, M<sub>7</sub>, M<sub>5</sub>, A and B are changing from a 0 to a 1, where as C remains constant as a 1. SO it is read as . Along the 4-squad formed by M<sub>3</sub>, M<sub>2</sub>, M<sub>7</sub>, and M<sub>6</sub>, variables A and C are changing from a 0 to a 1. But B remains constant as a 1. So it is read as . The minimal expression is the product of these two terms , i.e.,  $f_1 = ( ) ( )$ .also in this figure, along the 2-square formed by M<sub>4</sub> and M<sub>6</sub> , variable B is changing from a 0 to a 1, while variable A remains constant as a 1 and variable C remains constant as a 0. SO, read it as

+C. Similarly, the 2-square formed by M<sub>7</sub> and M<sub>6</sub> is read as + , while the 2-square formed by M<sub>2</sub> and M<sub>6</sub> is read as +C. The minimal expression is the product of these sum terms, i.e,  $f_2 = ( + ) ( + ) ( + C )$

**Ex:**Reduce the expression  $f = \sum m(0,2,3,4,5,6)$  using mapping and implement it in AOI logic as well as in NAND logic.The Sop k-map and its reduction , and the implementation of the minimal expression using AOI logic and the corresponding NAND logic are shown in figures below

In SOP k-map, the reduction is done as:

1. m<sub>5</sub> has only one adjacency m<sub>4</sub> , so combine m<sub>5</sub> and m<sub>4</sub> into a square. Along this 2-square A remains constant as 1 and B remains constant as 0 but C varies from 0 to 1. So read it as A .
2. m<sub>3</sub> has only one adjacency m<sub>2</sub> , so combine m<sub>3</sub> and m<sub>2</sub> into a square. Along this 2-square A remains constant as 0 and B remains constant as 1 but C varies from 1 to 0. So read it as B.
3. m<sub>6</sub> can form a 2-square with m<sub>2</sub> and m<sub>4</sub> can form a 2-square with m<sub>0</sub>, but observe that by wrapping the map from left to right m<sub>0</sub>, m<sub>4</sub> ,m<sub>2</sub> ,m<sub>6</sub> can form a 4-square. Out of these m<sub>2</sub> and m<sub>4</sub> have already been combined but they can be utilized again. So make it. Along this 4-square, A is changing from 0 to 1 and B is also changing from 0 to 1 but C is remaining constant as 0. so read it as .
4. Write all the product terms in SOP form. So the minimal SOP expression is



#### Four variable k-maps:

Four variable k-map expressions can have  $2^4=16$  possible combinations of input variables such as ,-----ABCD with minterm designations m<sub>0</sub>,m<sub>1</sub> ----- m<sub>15</sub> respectively in SOP form & A+B+C+D, A+B+C+ ,----- + + + with maxterms M<sub>0</sub>,M<sub>1</sub>, ----- -

-M15 respectively in POS form. It has  $2^4=16$  squares or cells. The binary number designations of rows & columns are in the gray code. Here follows 01 & 10 follows 11 called Adjacency ordering.

CD \ AB	00	01	11	10
00	0 $\bar{A}\bar{B}\bar{C}\bar{D}$	1 $\bar{A}\bar{B}\bar{C}D$	3 $\bar{A}\bar{B}C\bar{D}$	2 $\bar{A}\bar{B}CD$
01	4 $\bar{A}B\bar{C}\bar{D}$	5 $\bar{A}B\bar{C}D$	7 $\bar{A}BC\bar{D}$	6 $\bar{A}BCD$
11	12 $AB\bar{C}\bar{D}$	13 $AB\bar{C}D$	15 $ABCD$	14 $ABC\bar{D}$
10	8 $A\bar{B}\bar{C}\bar{D}$	9 $A\bar{B}\bar{C}D$	11 $A\bar{B}C\bar{D}$	10 $A\bar{B}CD$

SOP form

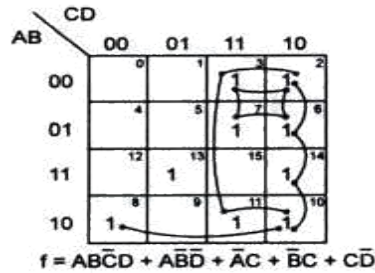
CD \ AB	00	01	11	10
00	0 $A+B+C+D$	1 $A+B+C+\bar{D}$	3 $A+B+\bar{C}+\bar{D}$	2 $A+B+\bar{C}+D$
01	4 $A+\bar{B}+C+D$	5 $A+\bar{B}+C+\bar{D}$	7 $A+\bar{B}+\bar{C}+\bar{D}$	6 $A+\bar{B}+\bar{C}+D$
11	12 $\bar{A}+\bar{B}+C+D$	13 $\bar{A}+\bar{B}+C+\bar{D}$	15 $\bar{A}+\bar{B}+\bar{C}+\bar{D}$	14 $\bar{A}+\bar{B}+\bar{C}+D$
10	8 $\bar{A}+B+C+D$	9 $\bar{A}+B+C+\bar{D}$	11 $\bar{A}+B+\bar{C}+\bar{D}$	10 $\bar{A}+B+\bar{C}+D$

POS form

EX: Reduce using mapping the expression  $\Sigma m(2, 3, 6, 7, 8, 10, 11, 13, 14)$ .

Start with the minterm with the least number of adjacencies. The minterm  $m_{13}$  has no adjacency. Keep it as it is. The  $m_8$  has only one adjacency,  $m_{10}$ . Expand  $m_8$  into a 2-square with  $m_{10}$ . The  $m_7$  has two adjacencies,  $m_6$  and  $m_3$ . Hence  $m_7$  can be expanded into a 4-square with  $m_6$ ,  $m_3$  and  $m_2$ . Observe that,  $m_7$ ,  $m_6$ ,  $m_2$ , and  $m_3$  form a geometric square. The  $m_{11}$  has 2 adjacencies,  $m_{10}$  and  $m_3$ . Observe that,  $m_{11}$ ,  $m_{10}$ ,  $m_3$ , and  $m_2$  form a geometric square on wrapping the K-map. So expand  $m_{11}$  into a 4-square with  $m_{10}$ ,  $m_3$  and  $m_2$ . Note that,  $m_2$  and  $m_3$ , have already become a part of the 4-square  $m_7$ ,  $m_6$ ,  $m_2$ , and  $m_3$ . But if  $m_{11}$  is expanded only into a 2-square with  $m_{10}$ , only one variable is eliminated. So  $m_2$  and  $m_3$  are used again to make another 4-square with  $m_{11}$  and  $m_{10}$  to eliminate two variables. Now only  $m_6$  and  $m_{14}$  are left uncovered. They can form a 2-square that eliminates only one variable. Don't do that. See whether they can be expanded into a larger square. Observe that,  $m_2$ ,  $m_6$ ,  $m_{14}$ , and  $m_{10}$  form a rectangle. So  $m_6$  and  $m_{14}$  can be expanded into a 4-square with  $m_2$  and  $m_{10}$ . This eliminates two variables.





### Five variable k-map:

Five variable k-map can have  $2^5 = 32$  possible combinations of input variable as  $E, \dots, ABCDE$  with minterms  $m_0, m_1, \dots, m_{31}$  respectively in SOP &  $A+B+C+D+E, A+B+C+\dots$  with maxterms  $M_0, M_1, \dots, M_{31}$  respectively in POS form. It has  $2^5 = 32$  squares or cells of the k-map are divided into 2 blocks of

16 squares each. The left block represents minterms from  $m_0$  to  $m_{15}$  in which  $A$  is a 0, and the right block represents minterms from  $m_{16}$  to  $m_{31}$  in which  $A$  is 1. The 5-variable k-map may contain 2-squares, 4-squares, 8-squares, 16-squares or 32-squares involving these two blocks. Squares are also considered adjacent in these two blocks, if when superimposing one block on top of another, the squares coincide with one another.

Some possible 2-squares in a five-variable map are  $m_0, m_{16}; m_2, m_{18}; m_5, m_{21}; m_{15}, m_{31}; m_{11}, m_{27}$ .

Some possible 4-squares are  $m_0, m_2, m_{16}, m_{18}; m_0, m_1, m_{16}, m_{17}; m_0, m_4, m_{16}, m_{20}; m_{13}, m_{15}, m_{29}, m_{31}; m_5, m_{13}, m_{21}, m_{29}$ .

Some possible 8-squares are  $m_0, m_1, m_3, m_2, m_{16}, m_{17}, m_{19}, m_{18}; m_0, m_4, m_{12}, m_8, m_{16}, m_{20}, m_{28}, m_{24}; m_5, m_7, m_{13}, m_{15}, m_{21}, m_{23}, m_{29}, m_{31}$ .

The squares are read by dropping out the variables which change. Some possible

Grouping is

(a)  $m_0, m_{16} = \overline{B}\overline{C}\overline{D}\overline{E}$

(b)  $m_2, m_{18} = \overline{B}\overline{C}D\overline{E}$

(c)  $m_4, m_6, m_{20}, m_{22} = \overline{B}C\overline{E}$

(d)  $m_5, m_7, m_{13}, m_{15}, m_{21}, m_{23}, m_{29}, m_{31} = CE$

(e)  $m_8, m_9, m_{10}, m_{11}, m_{24}, m_{25}, m_{26}, m_{27} = B\overline{C}$

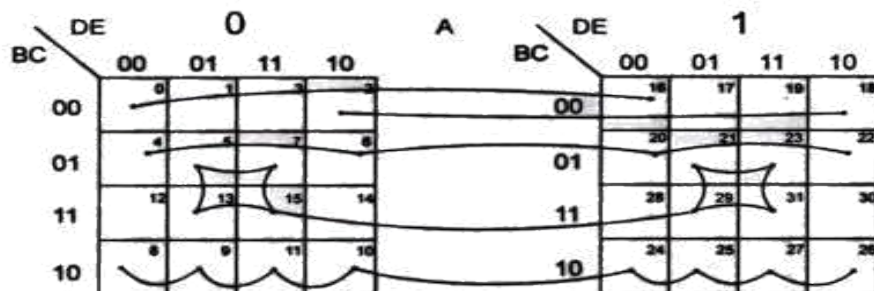
$M_0, M_{16} = B + C + D + E$

$M_2, M_{18} = B + C + \overline{D} + E$

$M_4, M_6, M_{20}, M_{22} = B + \overline{C} + E$

$M_5, M_7, M_{13}, M_{15}, M_{21}, M_{23}, M_{29}, M_{31} = \overline{C} + \overline{E}$

$M_8, M_9, M_{10}, M_{11}, M_{24}, M_{25}, M_{26}, M_{27} = \overline{B} + C$



Ex:  $F = \sum m(0,1,4,5,6,13,14,15,22,24,25,28,29,30,31)$  is SOP

POS is  $F = \pi M(2,3,7,8,9,10,11,12,16,17,18,19,20,21,23,26,27)$

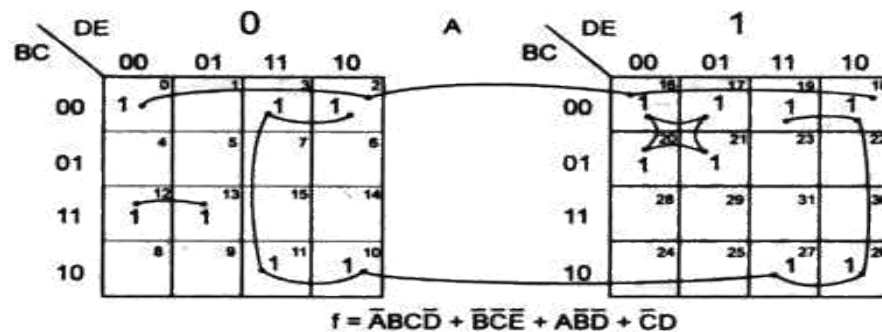
The real minimal expression is the minimal of the SOP and POS forms.

The reduction is done as

- There is no isolated 1s
- $M_{12}$  can go only with  $m_{13}$ . Form a 2-square which is read as  $A'B'CD'$
- $M_0$  can go with  $m_2, m_{16}$  and  $m_{18}$ . so form a 4-square which is read as  $B'C'E'$
- $M_{20}, m_{21}, m_{17}$  and  $m_{16}$  form a 4-square which is read as  $AB'D'$
- $M_2, m_3, m_{18}, m_{19}, m_{10}, m_{11}, m_{26}$  and  $m_{27}$  form an 8-square which is read as  $C'd$
- Write all the product terms in SOP form.

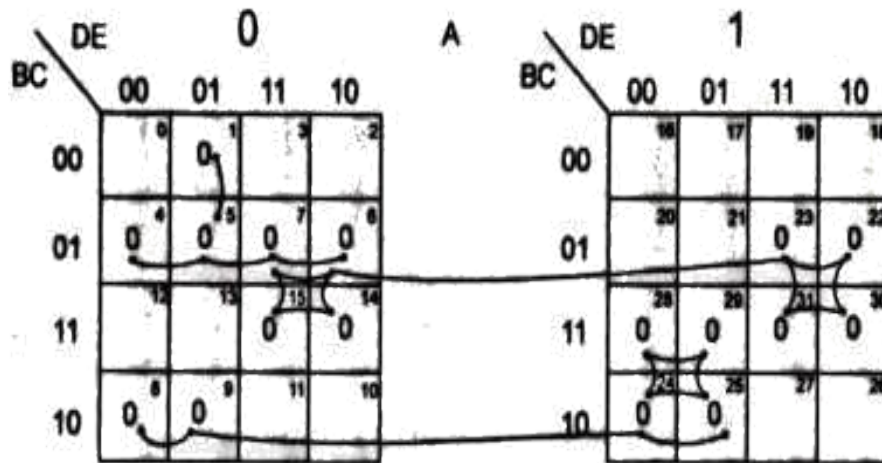
So the minimal expression is

$F_{\min} = A'B'CD' + B'C'E' + AB'D' + C'D$  (16 inputs)



In the POS k-map, the reduction is done as:

1. There are no isolated 0s
2.  $M_1$  can go only with  $M_5$ . So, make a 2-square, which is read as  $(A + B + D + \bar{E})$ .
3.  $M_4$  can go with  $M_5, M_7$ , and  $M_6$  to form a 4-square, which is read as  $(A + B + \bar{C})$ .
4.  $M_8$
- \*  $M_{28}$
6.  $M_{30}$
7. Sum terms in POS form. So the minimal expression in POS is  $F_{\min} = A'BcD' + B'C'E' + AB'D' + C'D$



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

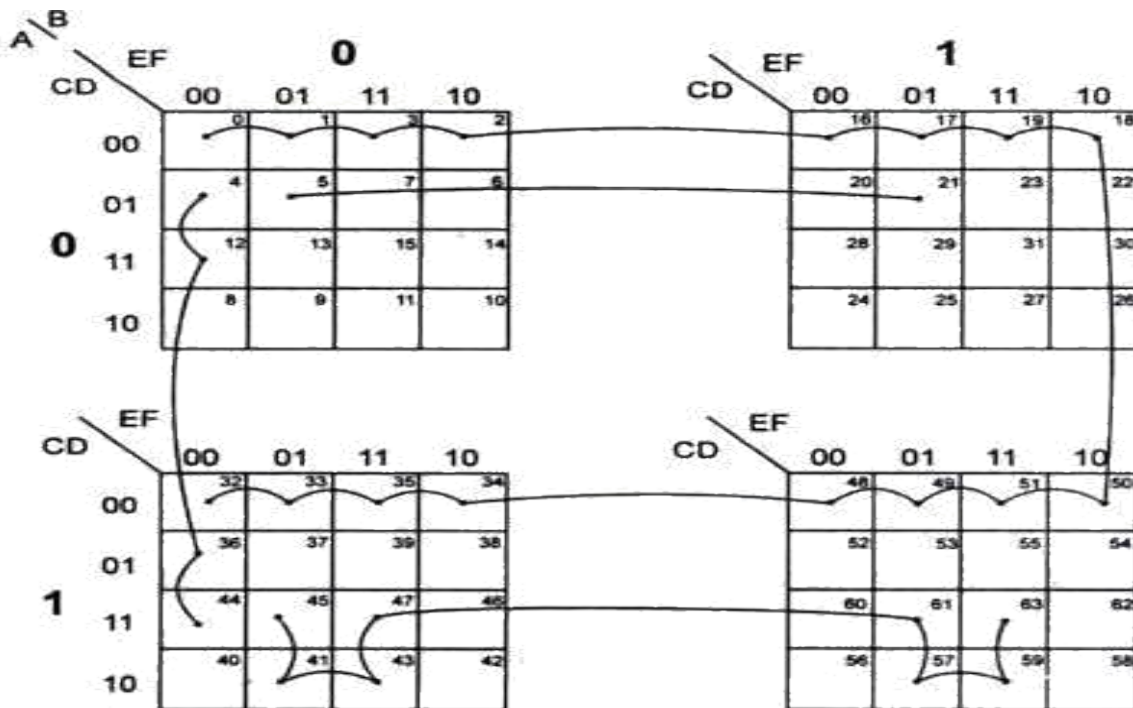
Six variable k-map:

Six variable k-map can have  $2^6 = 64$  combinations as

---ABCDEF with minterms  $m_0, m_1, \dots, m_{63}$  respectively in SOP &  $(A+B+C+D+E+F)$ ,  
 ----- (

----- ) with maxterms  $M_0, M_1, \dots, M_{63}$  respectively in POS form. It has

$2^6 = 64$  squares or cells of the k-map are divided into 4 blocks of 16 squares each.



Some possible groupings in a six variable k-map



## Binary Adder

The most basic arithmetic operation is addition. The circuit, which performs the addition of two binary numbers is known as **Binary adder**. First, let us implement an adder, which performs the addition of two bits.

### Half Adder

Half adder is a combinational circuit, which performs the addition of two binary numbers A and B are of **single bit**. It produces two outputs sum, S & carry, C.

The **Truth table** of Half adder is shown below.

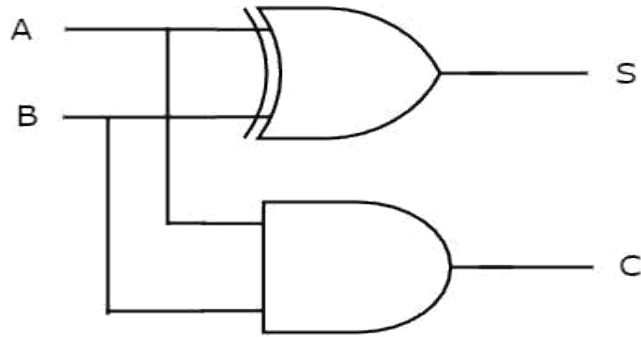
Inputs		Outputs	
A	B	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Let, sum, S is the Least significant bit and carry, C is the Most significant bit of the resultant sum. For first three combinations of inputs, carry, C is zero and the value of S will be either zero or one based on the **number of ones** present at the inputs. But, for last combination of inputs, carry, C is one and sum, S is zero, since the resultant sum is two.

From Truth table, we can directly write the **Boolean functions** for each output as  $S=A \oplus B$

$$C=AB$$

We can implement the above functions with 2-input Ex-OR gate & 2-input AND gate. The **circuit diagram** of Half adder is shown in the following figure.



In the above circuit, a two input Ex-OR gate & two input AND gate produces sum, S & carry, C respectively. Therefore, Half-adder performs the addition of two bits.

### Full Adder

Full adder is a combinational circuit, which performs the **addition of three bits** A, B and  $C_{in}$ . Where, A & B are the two parallel significant bits and  $C_{in}$  is the carry bit, which is generated from previous stage. This Full adder also produces two outputs sum, S & carry,  $C_{out}$ , which are similar to Half adder.

The **Truth table** of Full adder is shown below.

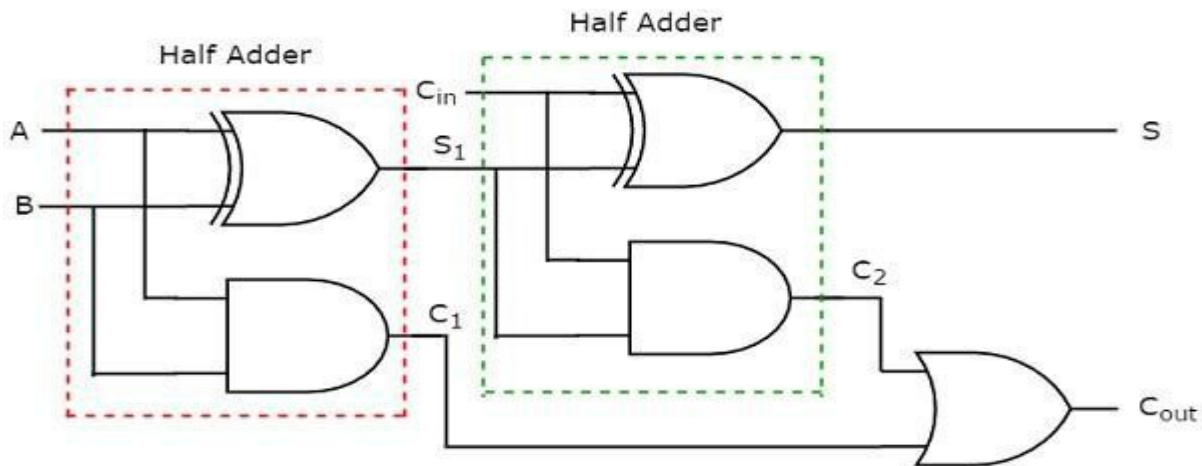
Inputs			Outputs	
A	B	$C_{in}$	$C_{out}$	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0

1	1	0	1	0
1	1	1	1	1

Let, sum, S is the Least significant bit and carry,  $C_{out}$  is the Most significant bit of resultant sum. It is easy to fill the values of outputs for all combinations of inputs in the truth table. Just count the **number of ones** present at the inputs and write the equivalent binary number at outputs. If  $C_{in}$  is equal to zero, then Full adder truth table is same as that of Half adder truth table.

We will get the following **Boolean functions** for each output after simplification.  $S = A \oplus B \oplus C_{in}$   
 $C_{out} = AB + (A \oplus B)C_{in}$

he sum, S is equal to one, when odd number of ones present at the inputs. We know that Ex-OR gate produces an output, which is an odd function. So, we can use either two 2input Ex-OR gates or one 3-input Ex-OR gate in order to produce sum, S. We can implement carry,  $C_{out}$  using two 2-input AND gates & one OR gate. The **circuit diagram** of Full adder is shown in the following figure.



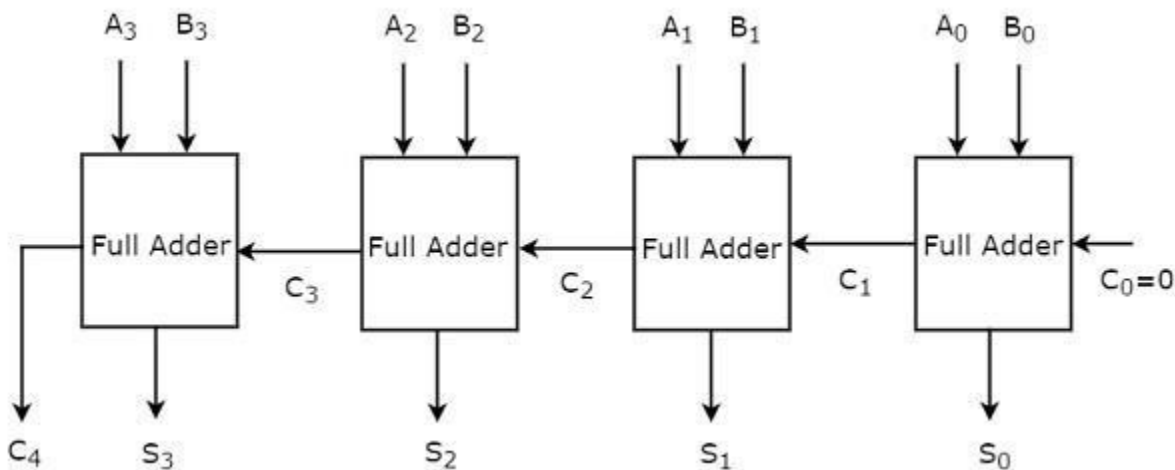
This adder is called as **Full adder** because for implementing one Full adder, we require two Half adders and one OR gate. If  $C_{in}$  is zero, then Full adder becomes Half adder. We can verify it easily from the above circuit diagram or from the Boolean functions of outputs of Full adder.

## 4-bit Binary Adder

The 4-bit binary adder performs the **addition of two 4-bit numbers**. Let the 4-bit binary numbers,  $A=A_3A_2A_1A_0$  and  $B=B_3B_2B_1B_0$ . We can implement 4-bit binary adder in one of the two following ways.

- Use one Half adder for doing the addition of two Least significant bits and three Full adders for doing the addition of three higher significant bits.
- Use four Full adders for uniformity. Since, initial carry  $C_{in}$  is zero, the Full adder which is used for adding the least significant bits becomes Half adder.

For the time being, we considered second approach. The **block diagram** of 4-bit binary adder is shown in the following figure.



Here, the 4 Full adders are cascaded. Each Full adder is getting the respective bits of two parallel inputs A & B. The carry output of one Full adder will be the carry input of subsequent higher order Full adder. This 4-bit binary adder produces the resultant sum having at most 5 bits. So, carry out of last stage Full adder will be the MSB.

In this way, we can implement any higher order binary adder just by cascading the required number of Full adders. This binary adder is also called as **ripple carry (binary) adder** because the carry propagates (ripples) from one stage to the next stage.

## Binary Subtractor

The circuit, which performs the subtraction of two binary numbers is known as **Binary subtractor**. We can implement Binary subtractor in following two methods.

- Cascade Full subtractors

## ■ 2's complement method

In first method, we will get an n-bit binary subtractor by cascading „n“ Full subtractors. So, first you can implement Half subtractor and Full subtractor, similar to Half adder & Full adder. Then, you can implement an n-bit binary subtractor, by cascading „n“ Full subtractors. So, we will be having two separate circuits for binary addition and subtraction of two binary numbers.

In second method, we can use same binary adder for subtracting two binary numbers just by doing some modifications in the second input. So, internally binary addition operation takes place but, the output is resultant subtraction.

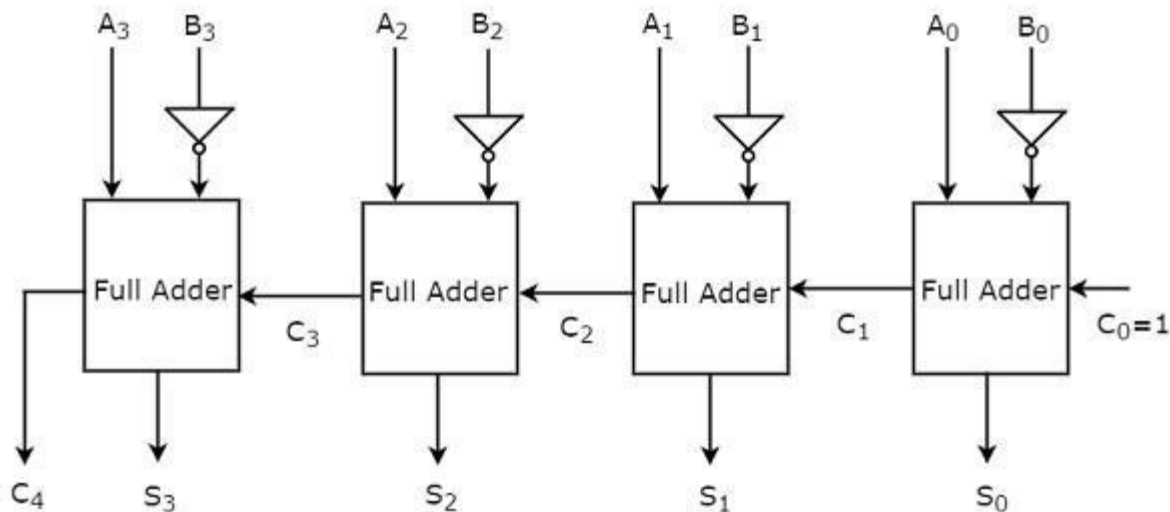
We know that the subtraction of two binary numbers A & B can be written as,

$$A - B = A + (2' \text{ s complement of } B)$$

$$\Rightarrow A - B = A + (1' \text{ s complement of } B) + 1$$

### 4-bit Binary Subtractor:

The 4-bit binary subtractor produces the **subtraction of two 4-bit numbers**. Let the 4bit binary numbers,  $A = A_3A_2A_1A_0$  and  $B = B_3B_2B_1B_0$ . Internally, the operation of 4-bit Binary subtractor is similar to that of 4-bit Binary adder. If the normal bits of binary number A, complemented bits of binary number B and initial carry (borrow),  $C_{in}$  as one are applied to 4-bit Binary adder, then it becomes 4-bit Binary subtractor. The **block diagram** of 4-bit binary subtractor is shown in the following figure.



This 4-bit binary subtractor produces an output, which is having at most 5 bits. If Binary number A is greater than Binary number B, then MSB of the output is zero and the remaining bits hold the magnitude of A-B. If Binary number A is less than Binary number B, then MSB of the output is one. So, take the 2's complement of output in order to get the magnitude of A-B.

### Binary Adder / Subtractor

The circuit, which can be used to perform either addition or subtraction of two binary numbers at any time is known as **Binary Adder / subtractor**. Both, Binary adder and Binary subtractor contain a set of Full adders, which are cascaded. The input bits of binary number A are directly applied in both Binary adder and Binary subtractor.

There are two differences in the inputs of Full adders that are present in Binary adder and Binary subtractor.

- The input bits of binary number B are directly applied to Full adders in Binary adder, whereas the complemented bits of binary number B are applied to Full adders in Binary subtractor.
- The initial carry,  $C_0 = 0$  is applied in 4-bit Binary adder, whereas the initial carry (borrow),  $C_0 = 1$  is applied in 4-bit Binary subtractor.

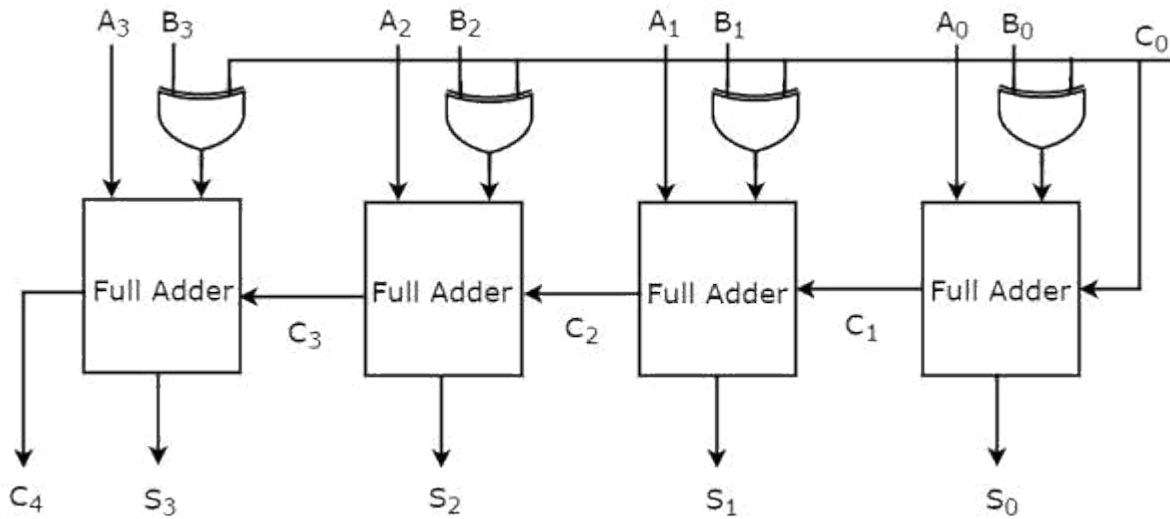
We know that a **2-input Ex-OR gate** produces an output, which is same as that of first input when other input is zero. Similarly, it produces an output, which is complement of first input when other input is one.

Therefore, we can apply the input bits of binary number B, to 2-input Ex-OR gates. The other input to all these Ex-OR gates is  $C_0$ . So, based on the value of  $C_0$ , the Ex-OR gates produce either the normal or complemented bits of binary number B.

### 4-bit Binary Adder / Subtractor

The 4-bit binary adder / subtractor produces either the addition or the subtraction of two 4-bit numbers based on the value of initial carry or borrow,  $C_0$ . Let the 4-bit binary numbers,  $A=A_3A_2A_1A_0$  and  $B=B_3B_2B_1B_0$ . The operation of 4-bit Binary adder / subtractor is similar to that of 4-bit Binary adder and 4-bit Binary subtractor.

Apply the normal bits of binary numbers A and B & initial carry or borrow,  $C_0$  from externally to a 4-bit binary adder. The **block diagram** of 4-bit binary adder / subtractor is shown in the following figure.



If initial carry,  $C_0$  is zero, then each full adder gets the normal bits of binary numbers A & B. So, the 4-bit binary adder / subtractor produces an output, which is the **addition of two binary numbers** A & B.

If initial borrow,  $b_0$  is one, then each full adder gets the normal bits of binary number A & complemented bits of binary number B. So, the 4-bit binary adder / subtractor produces an output, which is the **subtraction of two binary numbers** A & B.

## Binary Multipliers:

In binary multiplication by the paper and pencil method, is modified somewhat in digital machines because a binary adder can add only two binary numbers at a time.

In a binary multiplier, instead of adding all the partial products at the end, they are added two at a time and their sum accumulated in a register (the accumulator register). In addition, when the multiplier bit is a 0, 0s are not written down and added because it does not affect the final result. Instead, the multiplicand is shifted left by one bit.

The multiplication of 1110 by 1001 using this process is

Multiplicand 1110

Multiplier 1001

1110

1110

0 0)

The LSB of the multiplier is a 1; write down the multiplicand; shift the multiplicand one position to the left (11100)

The second multiplier bit is a 0; write down the previous result 1110; shift the multiplicand to the left again (1 1 1 0



+1110000

The fourth multiplier bit is a 1 write down the new multiplicand add it to the first partial product to obtain the final product.

1111110

This multiplication process can be performed by the serial multiplier circuit , which multiplies two 4-bit numbers to produce an 8-bit product. The circuit consists of following elements

**X register:** A 4-bit shift register that stores the multiplier --- it will shift right on the falling edge of the clock. Note that 0s are shifted in from the left.

**B register:** An 8-bit register that stores the multiplicand; it will shift left on the falling edge of the clock. Note that 0s are shifted in from the right.

**A register:** An 8-bit register, i.e, the accumulator that accumulates the partial products.

**Adder:** An 8-bit parallel adder that produces the sum of A and B registers. The adder outputs S<sub>7</sub> through S<sub>0</sub> are connected to the D inputs of the accumulator so that the sum can be transferred to the accumulator only when a clock pulse gets through the AND gate.

The circuit operation can be described by going through each step in the multiplication of 1110 by 1001. The complete process requires 4 clock cycles.

**1. Before the first clock pulse:** Prior to the occurrence of the first clock pulse, the register A is loaded with 00000000, the register B with the multiplicand 00001110, and the register X with the multiplier 1001. Assume that each of these registers is loaded using its asynchronous inputs(i.e., PRESET and CLEAR). The output of the adder will be the sum of A and B,i.e., 00001110.

**2. First Clock pulse:** Since the LSB of the multiplier (X<sub>0</sub>) is a 1, the first clock pulse gets through the AND gate and its positive going transition transfers the sum outputs into the accumulator. The subsequent negative going transition causes the X and B registers to shift right and left, respectively. This produces a new sum of A and B.

**3. Second Clock Pulse:** The second bit of the original multiplier is now in X<sub>0</sub> . Since this bit is a 0, the second clock pulse is inhibited from reaching the accumulator. Thus, the sum outputs are not transferred into the accumulator and the number in the accumulator does not change. The negative going transition of the clock pulse will again shift the X and B registers. Again a new sum is produced.

**4. Third Clock Pulse:** The third bit of the original multiplier is now in X<sub>0</sub>; since this bit is a 0, the third clock pulse is inhibited from reaching the accumulator. Thus, the sum outputs are not transferred into the accumulator and the number in the accumulator does not change. The negative going transition of the clock pulse will again shift the X and B registers. Again a new sum is produced.

**5. Fourth Clock Pulse:** The last bit of the original multiplier is now in X<sub>0</sub> , and since it is a 1, the positive going transition of the fourth pulse transfers the sum into the accumulator. The accumulator now holds the final product. The negative going transition of the clock pulse shifts X and B again. Note that, X is now 0000, since all the multiplier bits have been shifted out.





## 1. Magnitude Comparator:

### 1- bit Magnitude Comparator:

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

1. If  $A_1 = 1$  and  $B_1 = 0$ , then  $A > B$  or

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

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

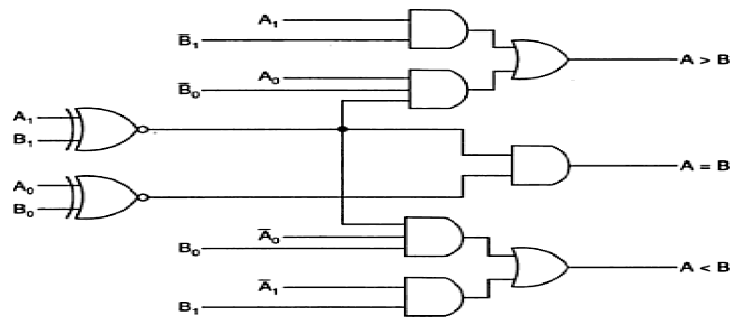
1. If  $A_1 = 0$  and  $B_1 = 1$ , then  $A < B$  or

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

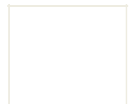
$$A < B : L = \bar{A}_1 B_1 + (A_1 \odot B_1) \bar{A}_0 B_0$$

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

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



Logic diagram of a 2-bit magnitude comparator.



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

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

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

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

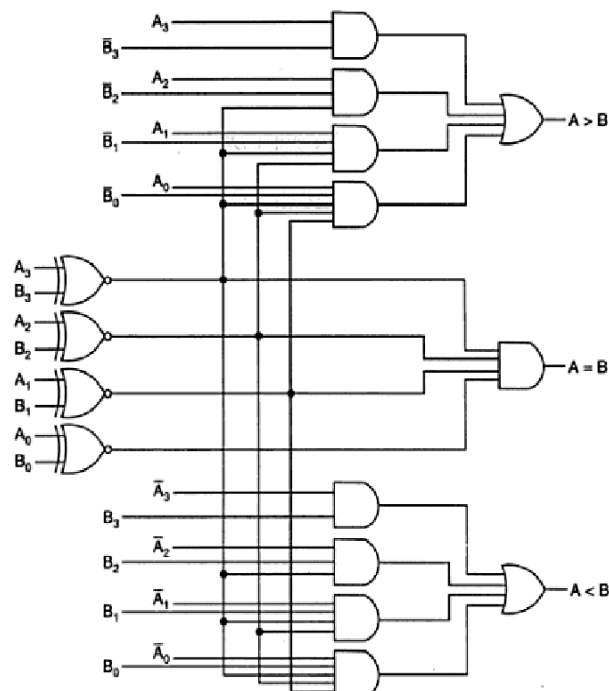
Similarly, the logic expression for  $A < B$  can be written as

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

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

So the expression for  $A = B$  can be written as

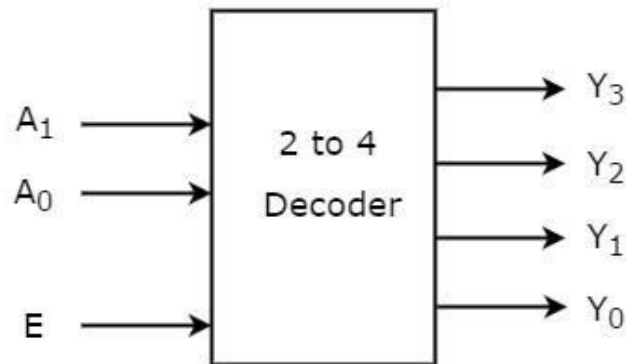
$$(A = B) = (A_3 \odot B_3)(A_2 \odot B_2)(A_1 \odot B_1)(A_0 \odot B_0)$$



**Decoder** is a combinational circuit that has „n“ input lines and maximum of  $2^n$  output lines. One of these outputs will be active High based on the combination of inputs present, when the decoder is enabled. That means decoder detects a particular code. The outputs of the decoder are nothing but the **min terms** of „n“ input variables (lines), when it is enabled.

**2 to 4 Decoder:**

Let 2 to 4 Decoder has two inputs  $A_1$  &  $A_0$  and four outputs  $Y_3$ ,  $Y_2$ ,  $Y_1$  &  $Y_0$ . The **block diagram** of 2 to 4 decoder is shown in the following figure.



One of these four outputs will be „1“ for each combination of inputs when enable, E is „1“. The **Truth table** of 2 to 4 decoder is shown below.

Enable	Inputs		Outputs			
E	A <sub>1</sub>	A <sub>0</sub>	Y <sub>3</sub>	Y <sub>2</sub>	Y <sub>1</sub>	Y <sub>0</sub>
0	x	x	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

From Truth table, we can write the **Boolean functions** for each output as

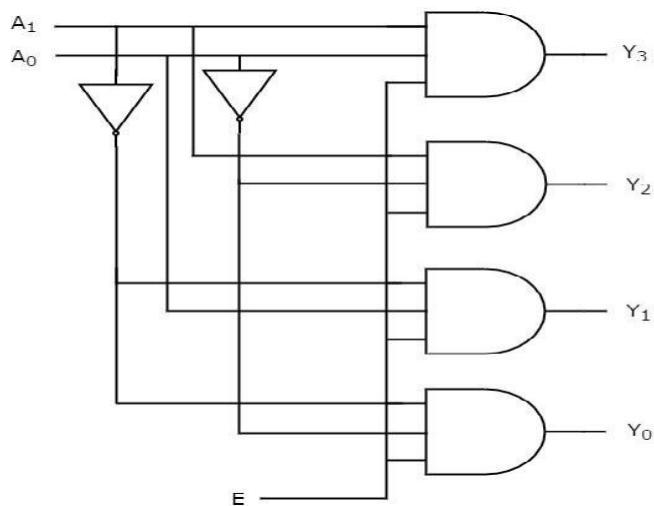
$$Y_3 = E \cdot A_1 \cdot A_0 \quad Y_3 = E \cdot A_1 \cdot A_0$$

$$Y_2 = E \cdot A_1 \cdot A_0' \quad Y_2 = E \cdot A_1 \cdot A_0'$$

$$Y_1 = E \cdot A_1' \cdot A_0 \quad Y_1 = E \cdot A_1' \cdot A_0$$

$$Y_0 = E \cdot A_1' \cdot A_0'$$

The **circuit diagram** of 2 to 4 decoder is shown in the following figure.



Therefore, the outputs of 2 to 4 decoder are nothing but the **min terms** of two input variables  $A_1$  &  $A_0$ , when enable, E is equal to one. If enable, E is zero, then all the outputs of decoder will be equal to zero.

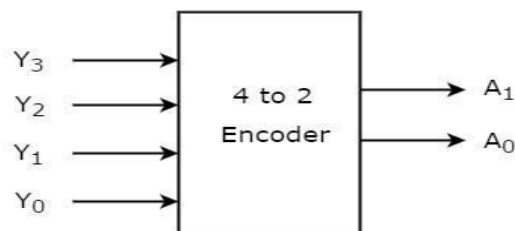
Similarly, 3 to 8 decoder produces eight min terms of three input variables  $A_2$ ,  $A_1$  &  $A_0$  and 4 to 16 decoder produces sixteen min terms of four input variables  $A_3$ ,  $A_2$ ,  $A_1$  &  $A_0$ .

### Encoder:

An **Encoder** is a combinational circuit that performs the reverse operation of Decoder. It has maximum of  $2^n$  input lines and „n“ output lines. It will produce a binary code equivalent to the input, which is active High. Therefore, the encoder encodes  $2^n$  input lines with „n“ bits. It is optional to represent the enable signal in encoders.

### 4 to 2 Encoder

Let 4 to 2 Encoder has four inputs  $Y_3$ ,  $Y_2$ ,  $Y_1$  &  $Y_0$  and two outputs  $A_1$  &  $A_0$ . The **block diagram** of 4 to 2 Encoder is shown in the following figure.



The **Truth table** of 4 to 2 encoder is shown below.

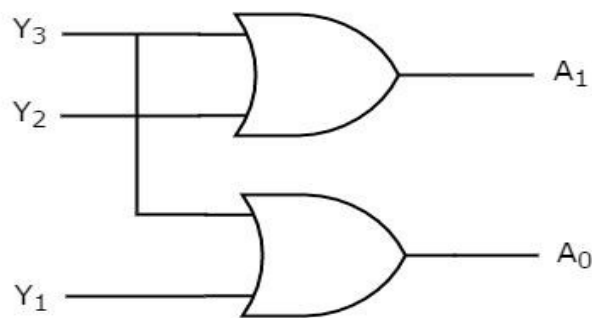
Inputs				Outputs	
$Y_3$	$Y_2$	$Y_1$	$Y_0$	$A_1$	$A_0$
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

From Truth table, we can write the **Boolean functions** for each output as

$$A_1 = Y_3 + Y_2$$

$$A_0 = Y_3 + Y_1$$

We can implement the above two Boolean functions by using two input OR gates. The **circuit diagram** of 4 to 2 encoder is shown in the following figure.



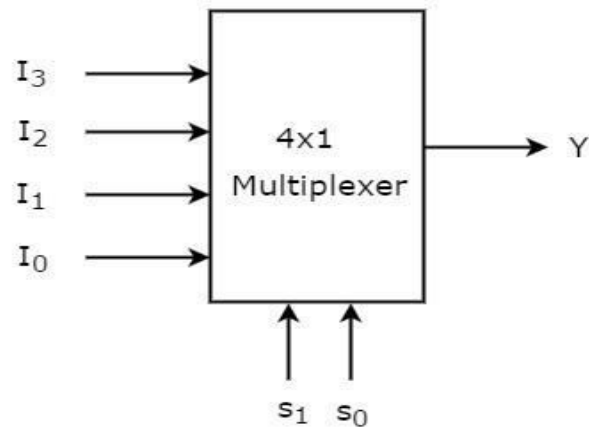
**Multiplexer** is a combinational circuit that has maximum of  $2^n$  data inputs, „n“ selection lines and single output line. One of these data inputs will be connected to the output based on the values of selection lines.

Since there are „n“ selection lines, there will be  $2^n$  possible combinations of zeros and ones. So, each combination will select only one data input.

### 4x1 Multiplexer:

4x1 Multiplexer has four data inputs  $I_3$ ,  $I_2$ ,  $I_1$  &  $I_0$ , two selection lines  $s_1$  &  $s_0$  and one output  $Y$ .

The **block diagram** of 4x1 Multiplexer is shown in the following figure.



One of these 4 inputs will be connected to the output based on the combination of inputs present at these two selection lines. **Truth table** of 4x1 Multiplexer is shown below.

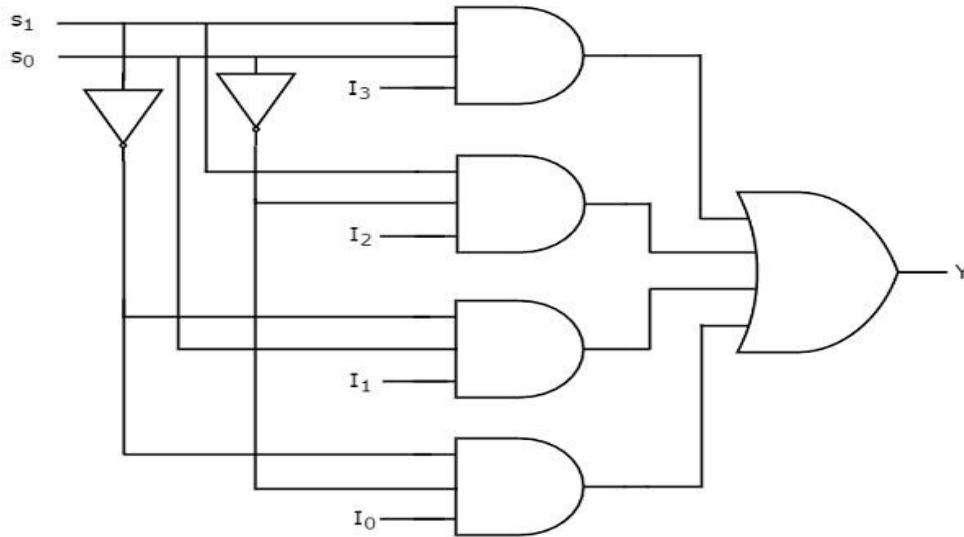
Selection Lines		Output
$S_1$	$S_0$	$Y$
0	0	$I_0$
0	1	$I_1$
1	0	$I_2$
1	1	$I_3$

From Truth table, we can directly write the **Boolean function** for output,  $Y$  as

$$Y = S_1'S_0'I_0 + S_1'S_0I_1 + S_1S_0'I_2 + S_1S_0I_3$$

We can implement this Boolean function using Inverters, AND gates & OR gate.

The **circuit diagram** of 4x1 multiplexer is shown in the following figure.

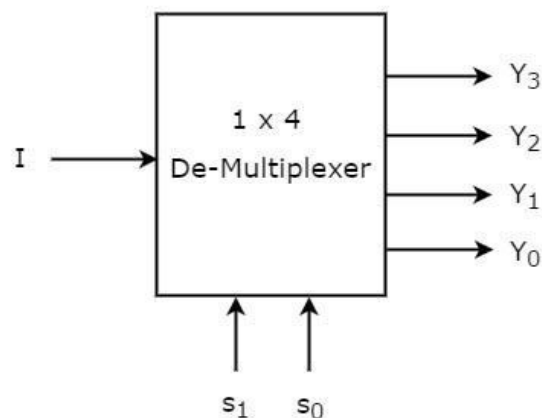


**De-Multiplexer** is a combinational circuit that performs the reverse operation of Multiplexer. It has single input, „n“ selection lines and maximum of  $2^n$  outputs. The input will be connected to one of these outputs based on the values of selection lines.

Since there are „n“ selection lines, there will be  $2^n$  possible combinations of zeros and ones. So, each combination can select only one output. De-Multiplexer is also called as **De-Mux**.

#### 1x4 De-Multiplexer :

1x4 De-Multiplexer has one input I, two selection lines,  $s_1$  &  $s_0$  and four outputs  $Y_3$ ,  $Y_2$ ,  $Y_1$  &  $Y_0$ . The **block diagram** of 1x4 De-Multiplexer is shown in the following figure.



The single input „I“ will be connected to one of the four outputs,  $Y_3$  to  $Y_0$  based on the values of selection lines  $s_1$  &  $s_0$ . The **Truth table** of 1x4 De-Multiplexer is shown below.



Selection Inputs		Outputs			
S <sub>1</sub>	S <sub>0</sub>	Y <sub>3</sub>	Y <sub>2</sub>	Y <sub>1</sub>	Y <sub>0</sub>
0	0	0	0	0	I
0	1	0	0	I	0
1	0	0	I	0	0
1	1	I	0	0	0

From the above Truth table, we can directly write the **Boolean functions** for each output

$$\text{as } Y_3 = s_1 s_0 I \quad Y_3 = s_1 s_0 I$$

$$Y_2 = s_1 s_0' I \quad Y_2 = s_1 s_0' I$$

$$Y_1 = s_1' s_0 I \quad Y_1 = s_1' s_0 I$$

$$Y_0 = s_1' s_0' I \quad Y_0 = s_1' s_0' I$$

We can implement these Boolean functions using Inverters & 3-input AND gates. The **circuit diagram** of 1x4 De-Multiplexer is shown in the following figure.