UNIT - I

→ SIMPLIFICATION OF BOOLEAN FUNCTIONS

→ Introduction to HDL

→ Combination Logic

3.1 The map method

→ The complexity of the digital logic gates that implement a boolean function is directly related to the complexity of the algebraic expression from which the function is implemented

→ The map method provides a simple straight forward procedure for minimizing boolean functions.

→ The map is a diagram made up of squares.

→ Each square represents one minterm

→ Since any boolean function can be expressed as a sum of minterms, it follows that a boolean function is recognized graphically in the map from the area enclosed by those squares whose minterms are included in the function.

3.2. Two and three variable maps

→ A two variable map is shown below

m_0	m_1
m_2	m_3

	y	y'
x'	$x'y$	$x'y'$
x	xy	xy'

0	1
2	3

→ There are four minterms for two variables, hence the map consists of four squares., one for each minterm.

→ x appears primed in row 0 ($\bar{x}$) and unprimed in row 1 (x).

→ Similarly y appears primed in column 0 and unprimed in column 1.

Ex:

$$Y = \sum_{(A,B)} m(2,3)$$

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

Three Variable Maps:

→ A three variable map is shown in figure below

→ For eight combinations, the map consists of eight squares.

→ The characteristic of the sequence is that only one bit changes from 1 to 0 or from 0 to 1 in the listing sequence.

m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6

x \ yz	00	01	11	10
0	0	1	3	2
1	4	5	7	6

x \ yz	00	01	11	10
0	$x'y'z'$	$x'y'z$	$x'yz$	$x'yz'$
1	$xy'z'$	$xy'z$	xyz	xyz'

Four variable maps :

→ many digital computers and systems process 4 bit numbers

AB \ CD	00	01	11	10
00	0	1	3	2
01	4	5	7	6
11	12	13	15	14
10	8	9	11	10

	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$				
$\bar{A}B$				
AB				
$A\bar{B}$				

Pairs, Quad & octet :

→ The map contains a pair of 1's that are horizontally adjacent (next to each other)

→ Only one variable goes from uncomplemented to complemented form

→ we can eliminate that element. (4)

(1)

	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	0	0	0	0
$\bar{A}B$	0	0	0	0
AB	0	0	1	1
$A\bar{B}$	0	0	0	0

$Y = ABCD + ABC\bar{D}$

$= ABC(D + \bar{D})$

$= ABC.$

(2)

	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	0	0	0	0
$\bar{A}B$	0	0	0	0
AB	0	0	0	1
$A\bar{B}$	0	0	0	1

$Y = AC\bar{D}$

(3)

	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	0	0	1	0
$\bar{A}B$	0	0	1	0
AB	0	0	0	0
$A\bar{B}$	0	0	0	0

$Y = \bar{A}CD$

(4)

	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	0	0	0	0
$\bar{A}B$	0	0	0	0
AB	0	0	0	0
$A\bar{B}$	1	1	0	0

$Y = A\bar{B}\bar{C}$

(5)

	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	0	0	0	0
$\bar{A}B$	0	1	1	0
AB	1	0	0	0
$A\bar{B}$	1	0	0	0

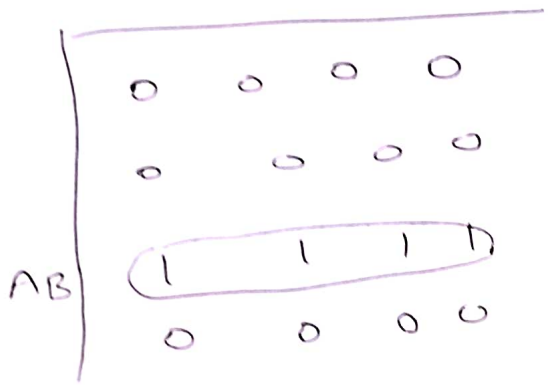
$Y = \bar{A}BD$

Quad :

→ A quad is a group of four 1's that are horizontally or vertically adjacent.

→ The 1's may be end to end or in the form of a square.

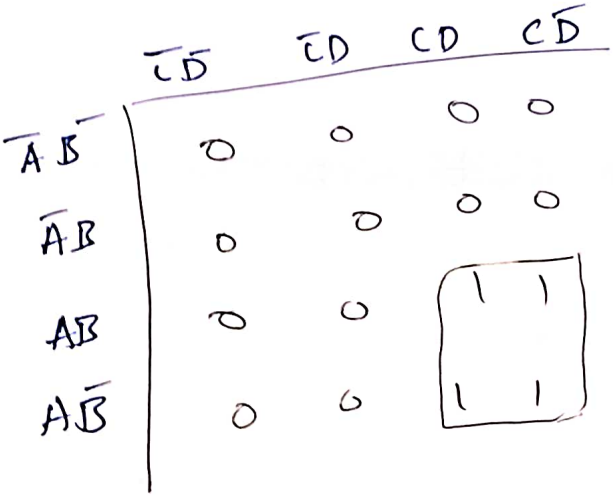
→ Quad eliminates two variables and their complements.



$Y = AB\bar{C} + ABC$

$= AB(\bar{C} + C)$

$= AB$

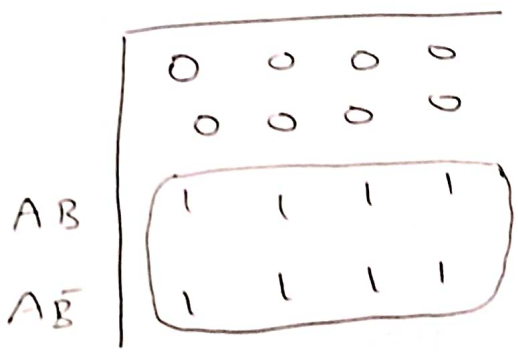


$Y = AC$

octet :

→ Group of eight

→ octet eliminates three variables



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

$= A$

Karnaugh Simplifications:

→ pair eliminates one variable & its complement.

→ Quad eliminates two variables & their complement.

→ octet eliminates 3 variables & their complement.

→ Because of this, after you draw a K-map encircle the octets first, quads second, and pairs last.

	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	0	1	1	1
$\bar{A}B$	0	0	0	1
AB	1	1	0	1
$A\bar{B}$	1	1	0	1

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

overlapping Groups

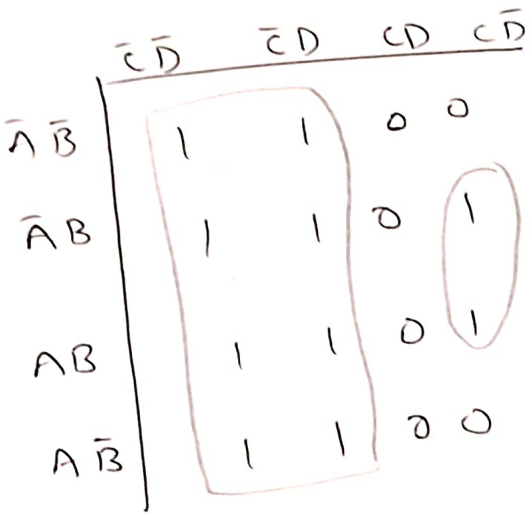
	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	0	0	0	0
$\bar{A}B$	0	1	0	0
AB	1	1	1	1
$A\bar{B}$	1	1	1	1

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

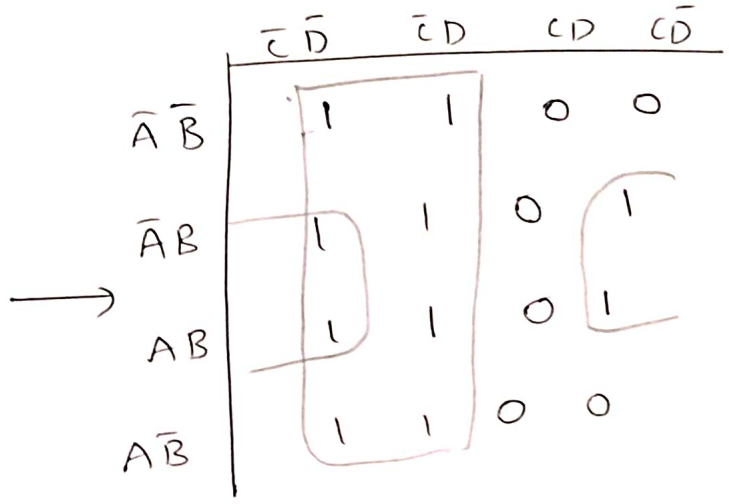
Rolling the map

	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	0	0	0	0
$\bar{A}B$	1	0	0	1
AB	1	0	0	1
$A\bar{B}$	0	0	0	0

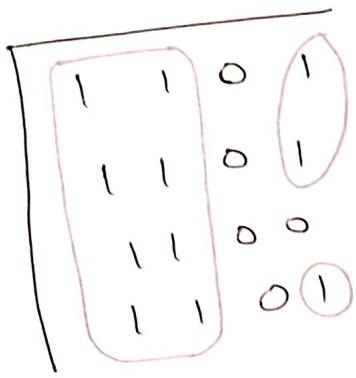
$Y = B\bar{D}$



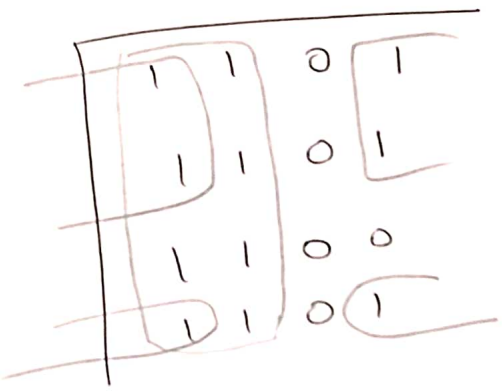
$Y = \bar{C} + BC\bar{D}$



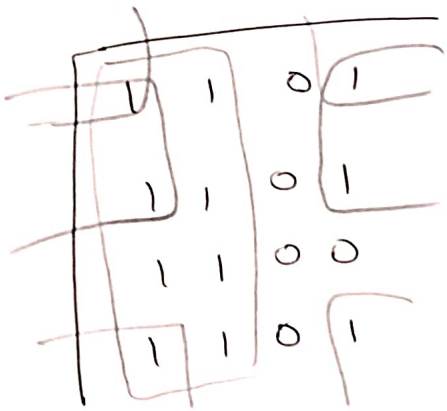
$Y = \bar{C} + BD$



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



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

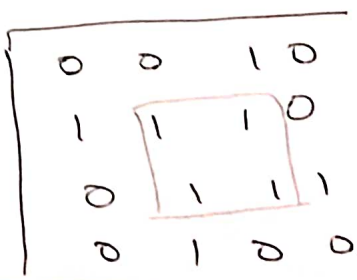


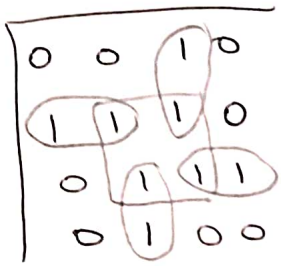
$Y = \bar{C} + \bar{A}\bar{D} + \bar{B}\bar{D}$

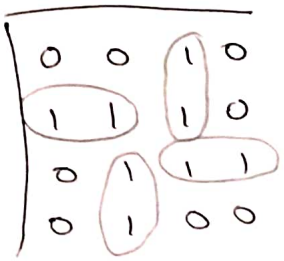
Eliminating Redundant Groups:

→ After encircling groups, we can eliminate redundant groups.

→ This is a group whose 1's are already used by other groups.







K-map method for Simplifying boolean equation:

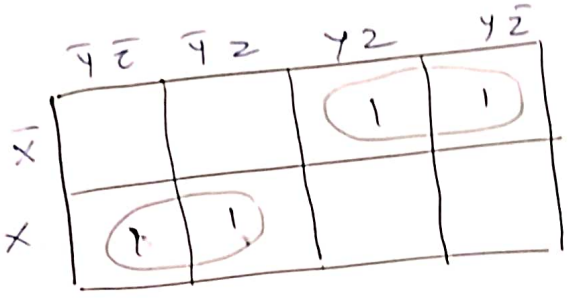
1. Enter a 1 on the k-map for each fundamental product that produces a 1 output in the truth table. Enter 0's elsewhere.
2. Encircle the octs, quads and pairs. Roll & overlap to get the largest group possible.
3. If any isolated 1's remain, encircle each.
4. Eliminate any redundant group
5. Write the boolean equation by ORing the products corresponding to the encircled groups.

~~Entered variable map~~

~~$Y = F(A, B, C) = \Sigma m(2, 6, 7)$~~

(3.1) Simplify the boolean function.

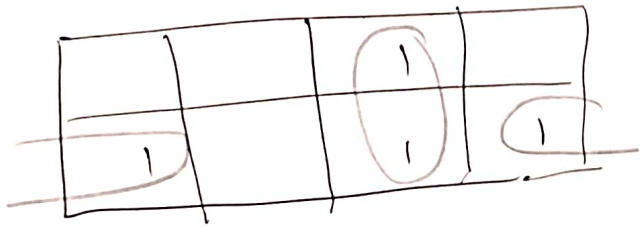
$$F = x'yz + x'yz' + xy'z' + xy'z$$



$$x'y + xy'$$

(3.2) Simplify the boolean function

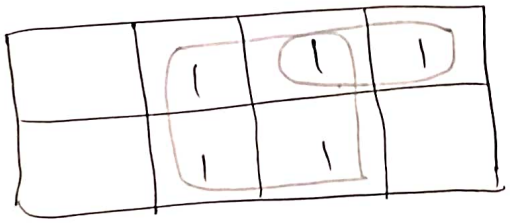
$$F = x'yz + xy'z' + xyz + xyz'$$



$$yz + xz'$$

(3.3) Simplify the boolean function.

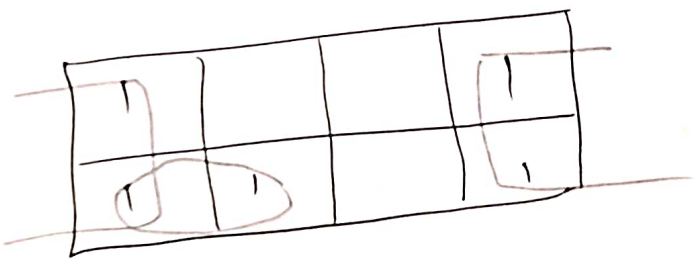
$$F = A'C + A'B + AB'C + BC$$



$$C + A'B$$

(3.4) Simplify the boolean function.

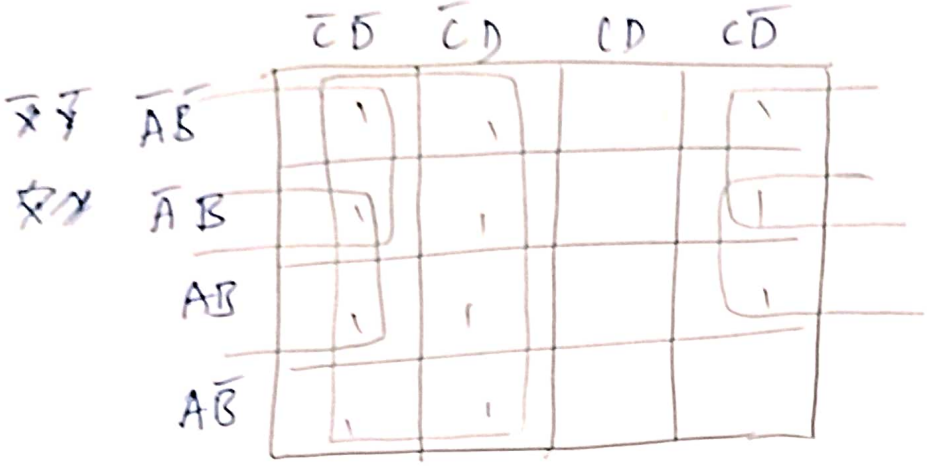
$$F(x, y, z) = \Sigma(0, 2, 4, 5, 6)$$



$$z' + xy'$$

(3.5) Simplify the boolean function.

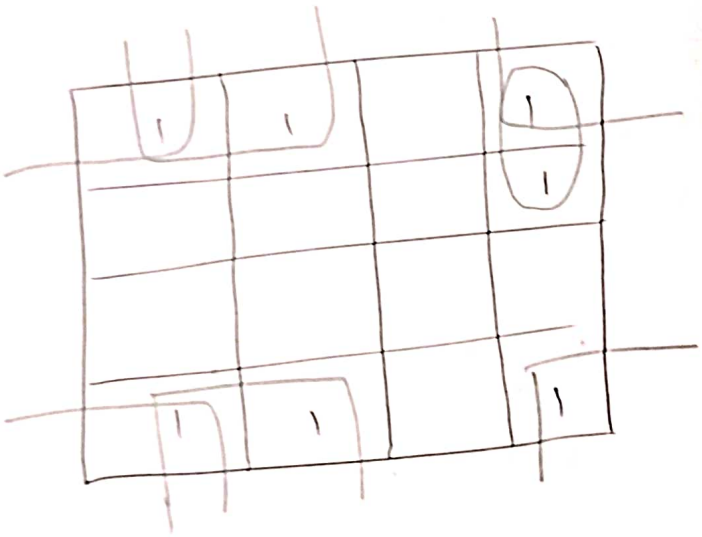
$F(w,x,y,z) = \Sigma(0,1,2,4,5,6,8,9,12,13,14)$



$F = y' + w'z' + xz'$

(3.6) simplify the boolean function

$F = A'B'C' + B'CD' + A'BCD' + AB'C'$



$F = B'D' + B'C' + A'CD'$

3.8 Dont care condition

→ The 1's and 0's in the map signify the combination of variables that makes the function equal to 1 or 0.

→ A four-bit decimal code, for example has six combinations which are not used.

→ Any digital circuit using this code operates under the assumption that these unused combinations will never occur as long as the system is working properly.

→ As a result, we dont care what the function output is to be for these combinations of the variables because they are guaranteed never to occur.

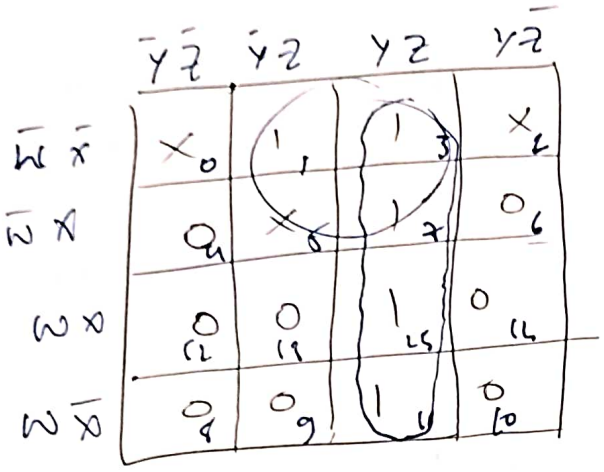
→ when choosing adjacent squares to simplify the function in the map, the x's may be assumed to be either 0 or 1, whichever gives the simplest expression.

→ In addition an x need not be used at all if it does not contribute to covering a larger area.

Ex : 3.12 simplify the boolean function

$F(w,x,y,z) = \Sigma(1,3,7,11,15)$

$d(w,x,y,z) = \Sigma(0,2,5)$



$F = w'z + yz$ → combining 1's

combining 0's → $F = z(w' + y)$

Tutorial-3

Codes

① 2.14 – 80

② 2.15 – 80

③ 2.16 – 82

④ 2.17 – 83

⑤ 2.18 – 84

⑥ 2.19 – 86

⑦ 2.20 – 87

⑧ 2.22 – 92

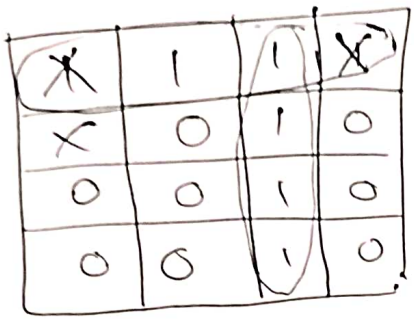
⑨ 2.23 – 93

⑩ 2.24 – 94

⑪ 2.27 – 97

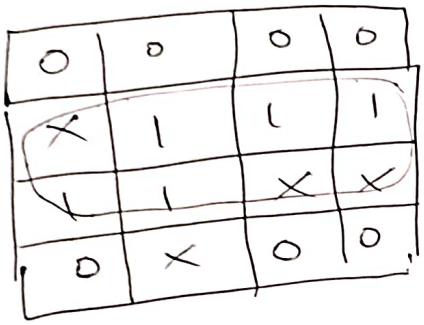
⑫ 2.28 – 99

① $f(A,B,C,D) = \sum m(1,3,7,11,15) + \sum d(0,2,4)$



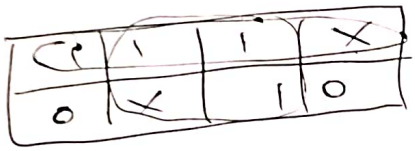
$Y = \bar{A}\bar{B} + CD$

② $f(A,B,C,D) = \sum m(5,6,7,12,13) + \sum d(4,9,14,15)$



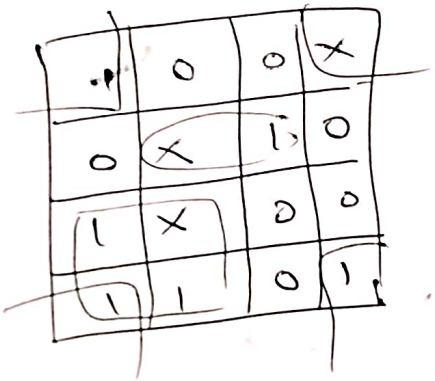
$Y = B$

③ $f(A,B,C) = \sum(0,1,3,7) + \sum d(2,5)$



$Y = \bar{A} + C$

④ $F(w,x,y,z) = \sum m(0,7,8,9,10,12) + \sum d(2,5,13)$



$F(w,x,y,z) = \bar{x}\bar{z} + \bar{w}xz + w\bar{y}$

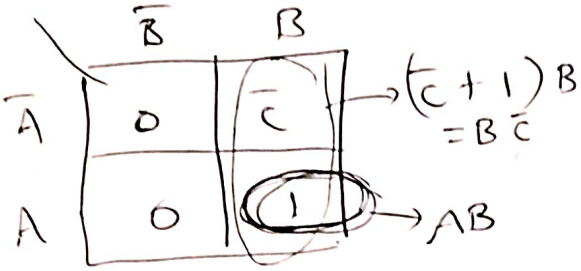
Entered Variable Maps:-

Ex :- $Y = F(A,B,C) = \Sigma m\,(2,6,7)$

A	B	C	Y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

① Consider C as MEV

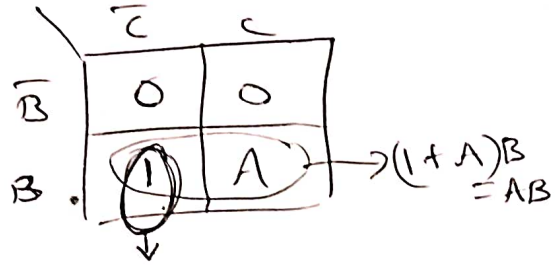
AB	Y
0 0	0
0 1	$\bar{C}$
1 0	0
1 1	1



$Y = B\bar{C} + AB$

② Consider A as MEV

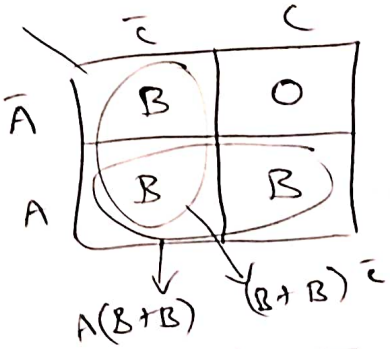
B	C	Y
0	0	0
0	1	0
1	0	1
1	1	A



$Y = B\bar{C} + AB$

③

A	C	Y
0	0	B
0	1	0
1	0	B
1	1	B



$Y = AB + B\bar{C}$

B as MEV

Canonical and Standard Forms

minterms & maxterms:

→ A binary variable may appear either in its normal form (x) or in its complement form (x').

→ Consider two binary variables x and y combined with an AND operation.

→ Since each variable may appear in either form, there are four possible combinations $x'y'$, $x'y$, xy', xy.

→ Each of the four AND terms represents one of the distinct areas in the venn diagram and is called a minterm or a standard product.

→ In a similar manner, n variables can be combined to form 2^n minterms.

→ The binary numbers from 0 to $2^n - 1$ are listed under the n variables.

→ In a similar fashion, n variables forming an OR term, with each variable being primed or unprimed provide 2^n possible combinations called maxterms or standard sums

→ The eight maxterms for three variables together with their symbolic designation are listed in table below.

	minterms		maxterms	
xyz	Term	Designation	Term	Designation
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

→ A boolean function may be expressed algebraically from a given truth table by forming a minterm for each combination of the variables which produces a 1 in the function and then taking the OR of all those terms.

→ For example, the function f_1.

$$f_1 = x'y'z + xy'z' + xyz$$
$$= m_1 + m_4 + m_7$$
$$001 + 100 + 111$$

$$f_2 = x'yz + xy'z + xyz' + xyz$$
$$= m_3 + m_5 + m_6 + m_7$$

→ These examples demonstrate an important property of Boolean Algebra.

→ Any boolean function can be expressed as a sum of minterms.

→ Now consider the complement of a Boolean function.

→ The complement of f_1 is read as:

$$f_1' = x'y'z' + x'yz' + x'yz + xy'z + xyz'$$

→ If we take the complement of f_1' we obtain the function f_1:

$$f_1 = (x+y+z)(x+y'+z)(x+y'+z')$$
$$(x'+y+z')(x'+y'+z)$$
$$= M_0 \cdot M_2 \cdot M_3 \cdot M_5 \cdot M_6$$

$$f_2 = (x+y+z)(x+y+z')(x+y'+z)(x'+y+z)$$
$$= M_0 M_1 M_2 M_4$$

→ These examples demonstrate a second important property of Boolean algebra:

→ Any Boolean function can be expressed as a product of maxterms.

→ Boolean functions expressed as a sum of minterms or product of maxterms are said to be in Canonical form.

Sum of Minterms:

→ For n binary variables, one can obtain 2^n distinct minterms

→ Any boolean function can be expressed as a sum of minterms.

→ The minterms whose sum defines the boolean function are those that give the 1's of the function in a truth table.

→ It is sometimes convinient to express the boolean function in its sum-of-minterms form.

→ Each term in the equation is inspected to see if it contains all the variables.

→ If it misses one or more variables it is ANDed with an expression such as $x + x'$ where x is one of the missing variables.

Example 2.4 :

Express the boolean function $F = A + B'C$ in a sum of minterms. The function has three variables A, B and C.

→ The first term A is missing two variables

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

→ This is still missing one variable.

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

→ The second term $B'C$ is missing one variable :

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

→ Combining all terms,

$$F = A + B'C$$
$$= ABC + ABC' + AB'C +$$
$$AB'C' + AB'C + A'B'C$$

→ Rearranging the minterms in ascending order, we obtain

$$F = A'B'C + AB'C' + AB'C + ABC' + ABC$$
$$= m_1 + m_4 + m_5 + m_6 + m_7.$$

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

Product of maxterms:

→ To express the boolean function as a product of maxterms, it must first be brought into a form of OR terms.

→ This may be done by using the distributive law

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

→ Then any missing variable x in each OR term is ORed with xx'.

Ex 2.5 : Express the boolean function $F = xy + x'z$ in a product of maxterm form.

→ First convert the function into OR terms using the distributive law:

$$F = xy + x'z$$
$$= (xy + x')(xy + z)$$
$$= (x + x')(y + x')(x + z)(y + z)$$
$$= (x' + y)(x + z)(y + z)$$

→ The function has three variables x, y and z

→ Each OR term is missing one variable.

$x'+y = x'+y+zz'$

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

$x+z = x+z+yy'$

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

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

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

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

$F = (x+y+z)(x+y'+z)(x'+y+z)$
$(x'+y+z')$ ↳ EX 2.13

$= M_0 M_2 M_4 M_5$

$F(x,y,z) = \Pi(0,2,4,5)$

2.5

	x y z	F
m_0	0 0 0	0
m_1	0 0 1	1
m_2	0 1 0	0
m_3	0 1 1	1
m_4	1 0 0	0
m_5	1 0 1	0
m_6	1 1 0	1
m_7	1 1 1	1

M-0

2.4

	A B C	F
m_0	0 0 0	0
m_1	0 0 1	1
m_2	0 1 0	0
m_3	0 1 1	0
m_4	1 0 0	1
m_5	1 0 1	1
m_6	1 1 0	1
m_7	1 1 1	1

M-1

Conversion between canonical forms:

→ The complement of a function expressed as the sum of minterms equals the sum of minterms missing from the original function.

→ This is because the original function is expressed by those minterms that make the function equal to 1, while its complement is a 1 for those minterms that the function is a 0.

→ As an example consider the function:

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

→ This has a complement that can be expressed as

$$F'(A,B,C) = \Sigma\,(0,2,3)$$
$$= m_0 + m_2 + m_3$$

→ Now, if we take the complement of F' by De Morgan's theorem, we obtain F in a different form:

$$F = (m_0 + m_2 + m_3)'$$
$$= m_0' \cdot m_2' \cdot m_3' = M_0 M_2 M_3$$
$$= \Pi\,(0,2,3)$$

→ From the table, it is clear that the following relation holds true.

$$m_j' = M_j$$

→ To convert from one canonical form to another, interchange the symbols Σ and Π and list those numbers missing from the original form.

$$F(x, y, z) = \Pi(0, 2, 4, 5)$$

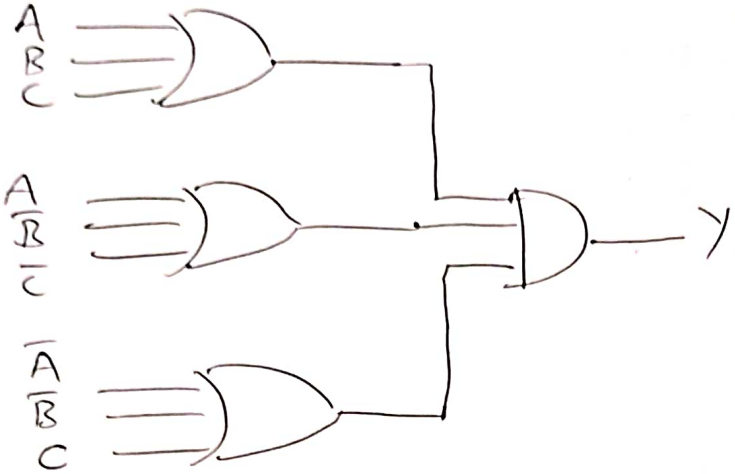
$$F(x, y, z) = \Sigma(1, 3, 6, 7)$$

→ The total number of minterms or maxterms is 2^n, where n is the number of binary variables in the function.

↳ Ex 2-14

Problems on SOP, POS, minterm & maxterm

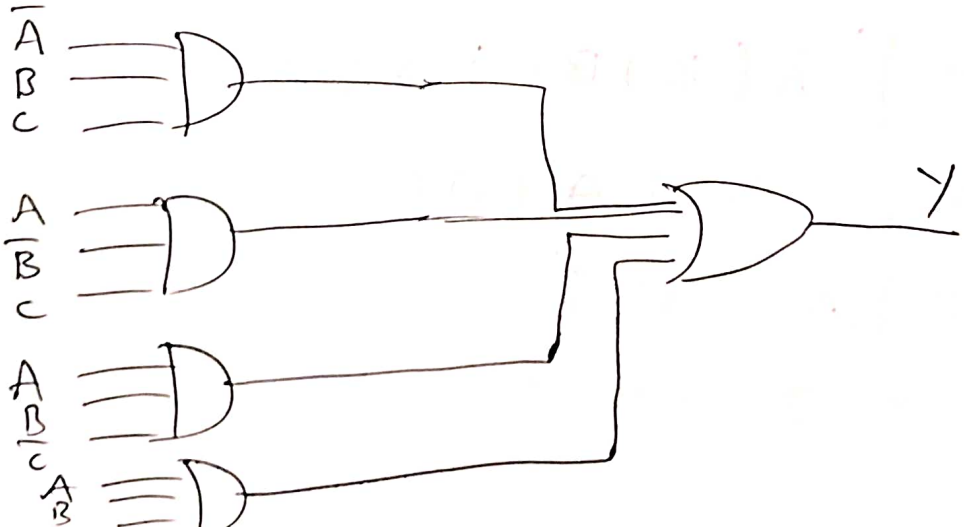
① Write the logic circuit for the POS expression $ABC + A\bar{B}\bar{C} + \bar{A}\bar{B}C$



② Suppose a truth table has a low output for the first three input conditions 000, 001 and 010. If all other outputs are high, what is the POS equation.

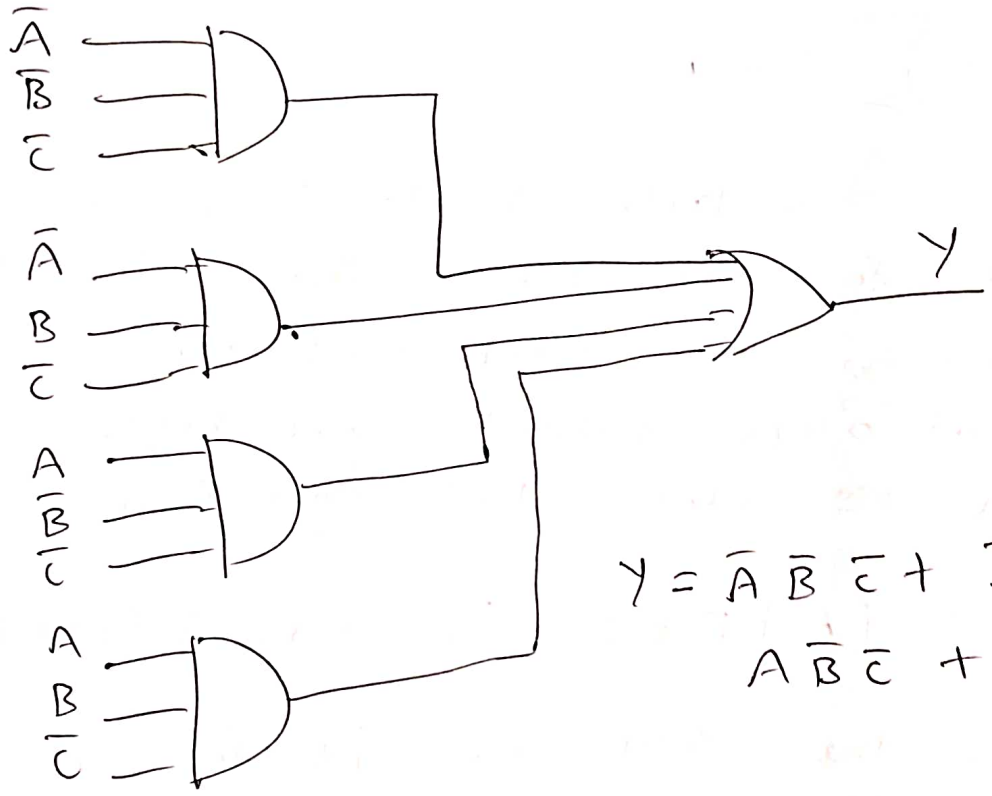
$$Y = (A+B+C)(A+B+\bar{C})(A+\bar{B}+C)$$

③ Write the SOP circuit for the equation $Y = \bar{A}BC + A\bar{B}C + AB\bar{C} + ABC$



(4) write the logic circuit for the three valuable truth table that has a high output for the input conditions 000 010 100 110. simplify the same

000 – $\bar{A}\bar{B}\bar{C}$
010 – $\bar{A}B\bar{C}$
100 – $A\bar{B}\bar{C}$
110 – $AB\bar{C}$



$Y = \bar{A}\bar{B}\bar{C} + \bar{A}B\bar{C} + A\bar{B}\bar{C} + AB\bar{C}$

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

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

$= [\bar{A}(1) + A(1)]\bar{C}$

$= [\bar{A} + A]\bar{C}$

$= \bar{C}$

(5) convert the given expression to standard SOP form.

$Y = AC + AB + BC$

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

$= ABC + A\bar{B}C + ABC + AB\bar{C} + ABC + \bar{A}BC$

$= ABC + A\bar{B}C + AB\bar{C} + \bar{A}BC$

(6) convert the given expression to standard SOP format.

$Y = A + AB + ABC$

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

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

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

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

(7) convert the given equation to standard POS form

$Y = (A+B)(B+C)(A+C)$

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

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

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

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

(8) convert the given expression in standard POS form.

$Y = A(A+B)(A+B+C)$

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

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

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

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

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

(9) simplify the following three variable expression using boolean algebra

$Y = \Sigma m(1, 3, 5, 7)$

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

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

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

$= \bar{A}C + AC$

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

$= C$

(10) convert the given expression into minterms using complementary property and simplify the expression.

$Y = \Pi M(3, 5, 7)$

$Y = \Sigma m(0, 1, 2, 4, 6)$

$Y = \bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}C + \bar{A}B\bar{C} + A\bar{B}\bar{C} + AB\bar{C}$

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

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

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

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

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

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

Standard form :

→ The two canonical forms of boolean algebra are basic forms that one obtains from reading a function from the truth table.

→ These forms are very seldom the ones with the least number of literals because each minterm or maxterm must contain, by definition, all the variables either complemented or uncomplemented.

→ 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.
sum of products & product of sums

→ The sum of products is a boolean expression containing AND terms, called Product terms, of one or more literals each.

→ The sum denotes the ORing of these terms

Ex:- $F_1 = y' + xy + x'yz'$

→ The expression has three product terms of one, two and three literals each.

→ 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.

→ Their product denotes the ANDing of these terms.

→ Ex: $F_2 = x(y' + z)(x' + y + z' + w)$

→ This expression has three sum terms of one, two, and four literals each.

→ The product denotes the ANDing of these terms.

→ ~~Ex~~: The use of the words product and sum stems from the similarity of the AND operation to the arithmetic product (multiplicat and the similarity of the OR operation to the arithmetic sum (additio

Sum of products equation:

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

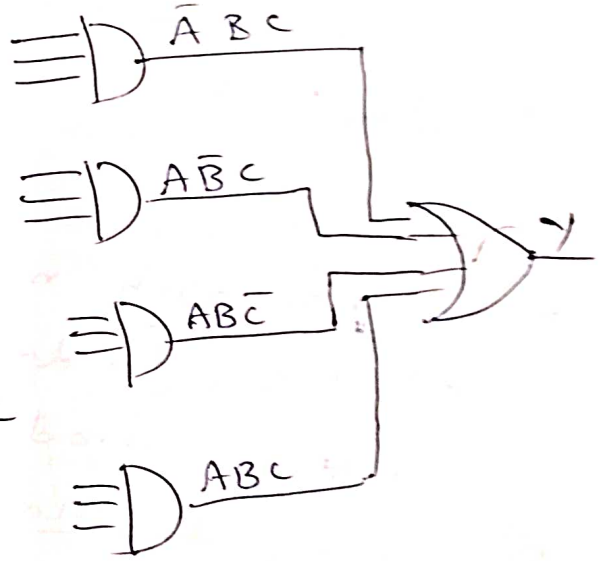
$$Y = F(A, B, C) = \Sigma m\,(3, 5, 6, 7)$$

→ Σ symbolizes summation or logical OR operation that is performed on corresponding minterms.

→ $Y = F(A, B, C)$ means Y is a function of three Boolean variables A, B, C.

Truth Table

	A	B	C	Y	
0	0	0	0	0	
1	0	0	1	0	
2	0	1	0	0	
3	0	1	1	1	→ $\bar{A}BC$
4	1	0	0	0	
5	1	0	1	1	→ $A\bar{B}C$
6	1	1	0	1	→ $AB\bar{C}$
7	1	1	1	1	→ ABC



↓

Diagram 3.7
(page - 85)

→ We can draw the logic circuit using an AND-OR network ⟷ Nand-Nand

→ Since a 4 I/P OR Gate is not available as a TTL chip, a 4 I/P Nand can be

3.7 Product of sums method:

→ With sum of products method the design starts with a truth table that summarizes the desired i/p o/p conditions.

→ The next step is to convert the truth table into an equivalent SOP eqn.

→ The final step is to draw the AND-OR n/w or its NAND-NAND equivalent.

→ The product of sums method is similar.

→ Given the truth table, identify the fundamental sums needed for a logic design.

→ By ANDing these sums, you get the POS eqn corresponding to the truth table.

converting a truth table into an equation.

			Y	
0	0	0	0	M_0
0	0	1	1	M_1
0	1	0	1	M_2
0	1	1	0	M_3
1	0	0	1	M_4
1	0	1	1	M_5
1	1	0	0	M_6
1	1	1	1	M_7

$Y = F(A, B, C)$

$= \Pi M(0, 3, 6)$

$Y = (A + B + C)$

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

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

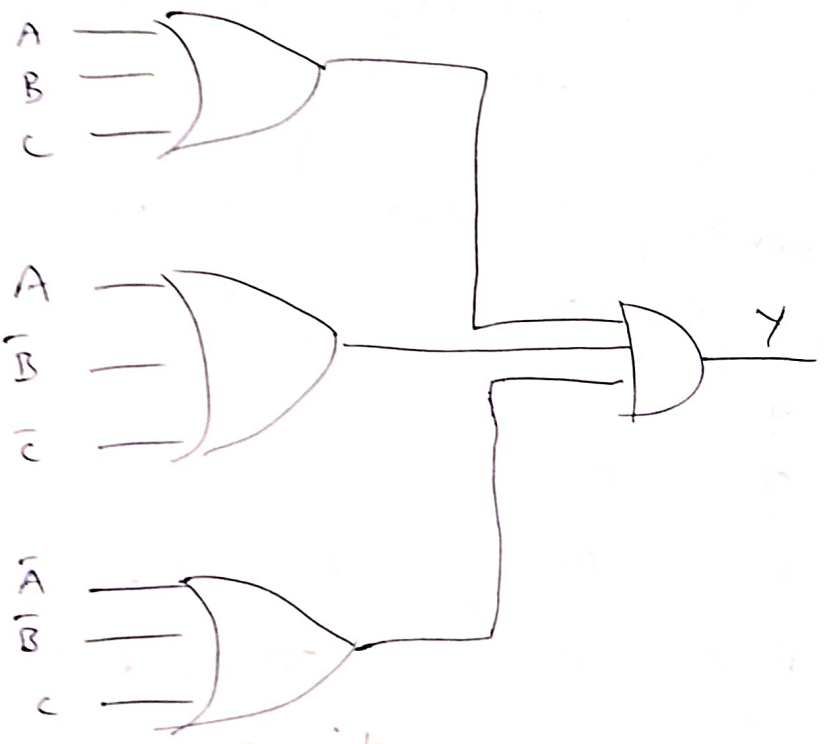
Π symbolizes Product i.e. AND operation

Logic circuit

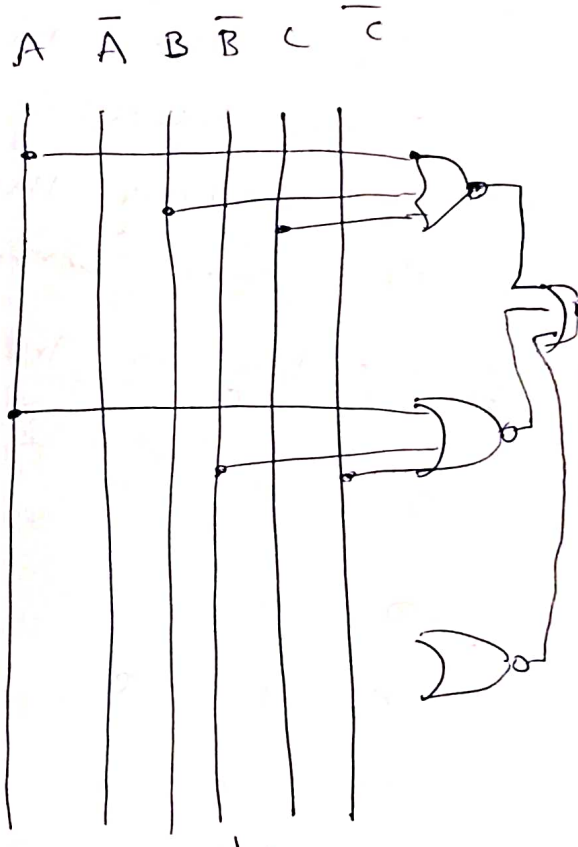
→ After we get a POS eqn, we can get the logic circuit by drawing an OR-AND network (or) a NOR-NOR n/w.

→ A 3 input OR gate is not available as a TTL chip.

→ With De morgans first theorem, we can replace OR-AND circuit by NOR-NOR circuit.



POS circuit



(NOR-NOR)

Conversion between SOP & POS

→ SOP → considering ones in a truth table

→ POS → considering zeros in truth table.

SOP → each gives a AND term, finally ORed

POS → each gives a OR term, finally ANDed

→ Thus SOP and POS occupy complementary locations in a truth table.

→ One representation can be obtained from the other by

(1) Identifying complementary locations

(2) changing minterm to maxterm or reverse

(3) changing summation by product or reverse

$$Y = F(A,B,C) = \Pi M(0,3,6) = \Sigma m(1,2,4,5,7)$$

$$Y = F(A,B,C) = \Sigma m(3,5,6,7) = \Pi M(0,1,2,4)$$

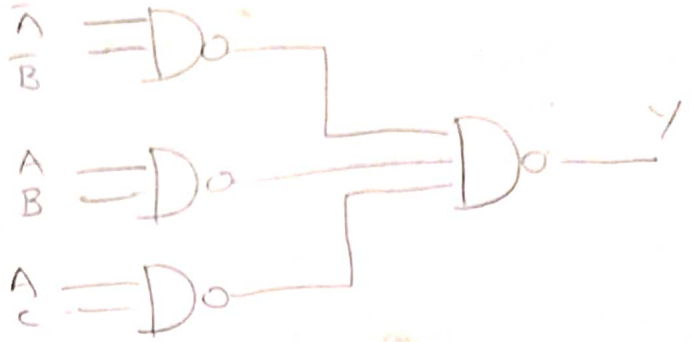
3.8 product of sums simplification :-

A	B	C	D
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	0	1
1	0	0	0
1	0	0	1
1	0	1	0
1	0	1	1
1	1	0	0
1	1	0	1
1	1	1	0
1	1	1	1

Y	
1	
1	
1	
1	
0	4
0	5
0	6
0	7
1	8
1	9
1	
1	
1	
1	
1	
1	

M(4,5,6,7, 8,9)

	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	1	1	1	1
$\bar{A}B$	0	0	0	0
AB	1	1	1	1
$A\bar{B}$	0	0	1	1

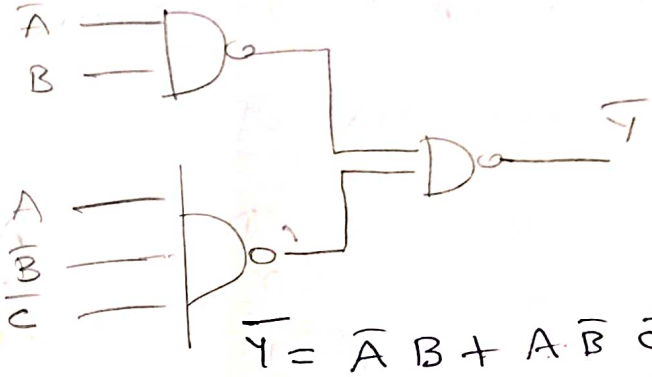


$$Y = \bar{A}\bar{B} + AB + AC$$

Complementary circuit:

→ To get pos circuit begin by complementing each 0 and 1 on the K-map.

	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	0	0	0	0
$\bar{A}B$	1	1	1	1
AB	0	0	0	0
$A\bar{B}$	1	1	0	0



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

→ The above figure shows the corresponding NAND-NAND circuit for $\bar{Y}$.

→ This circuit does not produce the desired output, it produces complement of the desired output.

Finding NOR-NOR ckt:

→ De Morgans first theorem tells us NAND gates can be replaced by bubbled OR gate.

2.5 Introduction to HDL :

→ Textual description of hardware design. is required

→ It is very important that it should be machine readable.

→ The advantages when we use a language are

(i) Describe a large complex design requiring hundreds of logic gates in a convinient manner in a smaller space.

(ii) use s/w Test bench to detect error. (simulation)

(iii) get hardware implementation details. (synthesis)

→ HDL (Hardware description language) is the solution

→ currently there are two widely used HDL s - Verilog & VHDL (very high speed integrated circuit Hardware description Language).

→ Verilog is considered simpler of the two, However both share lot of common features.

Verilog HDL :

→ Introduced in 1988 as a simulation and verification tool by Gateway Design automation.

→ Later acquired by Cadence Data systems.

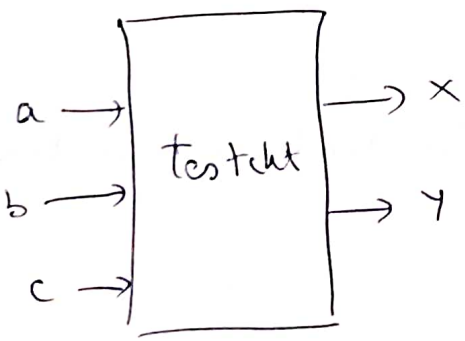
→ Put to public domain in 1990

→ Now it is controlled by a group of companies open verilog international.

Describing input output :

→ In any digital circuit, there are a set of inputs and set of outputs.

→ often termed as ports.




```
module testckt (x, y, a, b, c);
input a, b, c;
output x, y;
//module body
end module.
```

// → put comments

module, endmodule → written in bold.

writing module body:

Ex Two i/p or gate

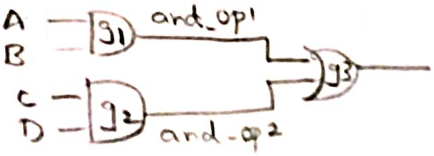
```
module or_gate(A, B, Y);
input A,B ; // defines two input port
output Y ; // defines two output port
or g1(Y,A, B);/* Gate declaration with
                 Predefined keyword or
                 representing logic OR */

end module
```

→ verilog supports predefined gate level primitives such as and, or, not, nand, nor, xor, xnor.

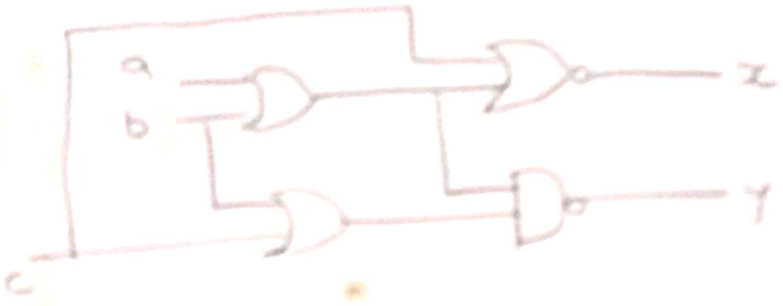
→ verilog can take upto 12 i/p's for logic gates.

→ wire is used when we need to represent intermediate variables

```
module ex_a(A, B, C, D, Y);
input A,B,C,D ;
output Y ;
wire and_op1, and_op2 ; //internal connections
and g1(and_op1, A, B)
and g2(and_op2, C, D);
or g3(Y, and_op1, and_op2);
```



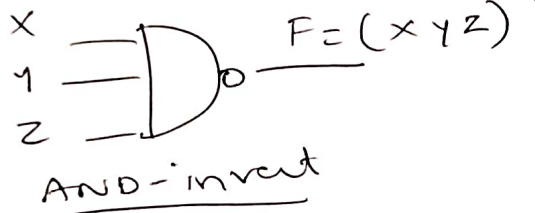
(Example 2.16) (page-66)



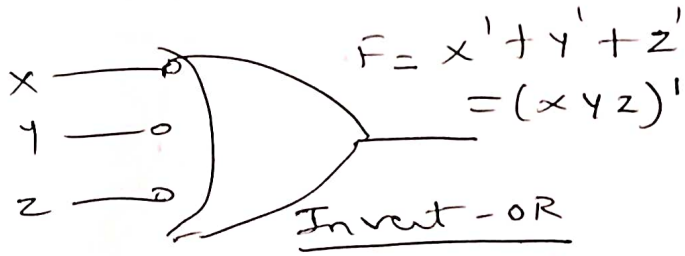
3.6 Nand & NOR implementation

→ Digital circuits are more frequently constructed with NAND or NOR gates.

→ To facilitate conversion to NAND and NOR logic, it is convinient to define two other graphic symbols for these gates.

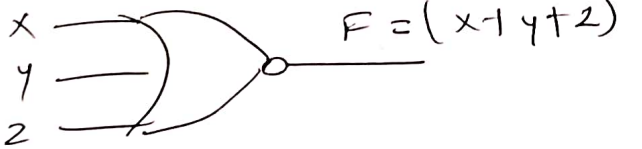


AND-invert

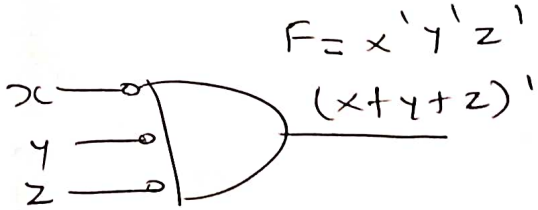


Invert-OR

(a) Two graphic symbols for NAND gates



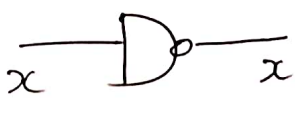
OR-invert



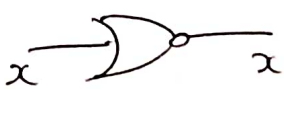
Invert-AND

(b) Two graphic symbols for NOR gate.

Buffer-invert

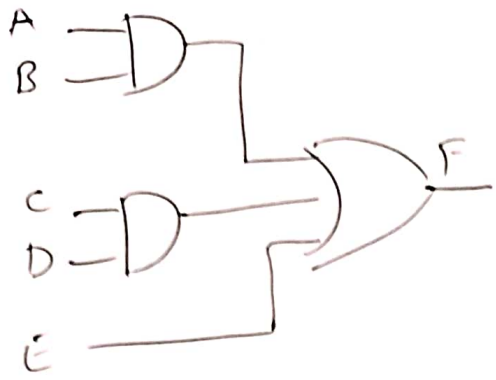


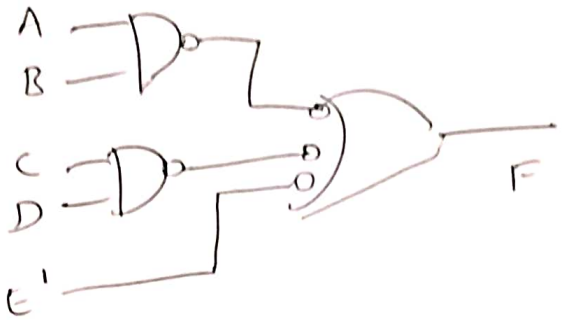
AND-invert

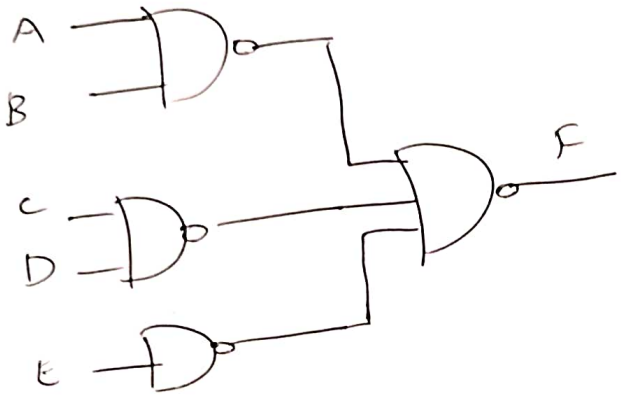


OR-invert

(c) Three graphic symbols for inverter.







Three ways to implement $F = AB + CD + E$

→ The rule for obtaining the NAND logic diagram from a boolean function is as follows:

① Simplify the function and express it in sum of products.

② Draw a NAND gate for each product term of the function that has at least two literals. The inputs to each NAND gate are the literals of the term. This constitutes a group of first level gates.

(3) Draw a single NAND gate (using the AND-invert or invert-OR graphic symbol) in the second level, with inputs coming from outputs of first-level gates.

(4) A term with a single literal requires an inverter in the first level or may be complemented and applied as an input to the second-level NAND gate.

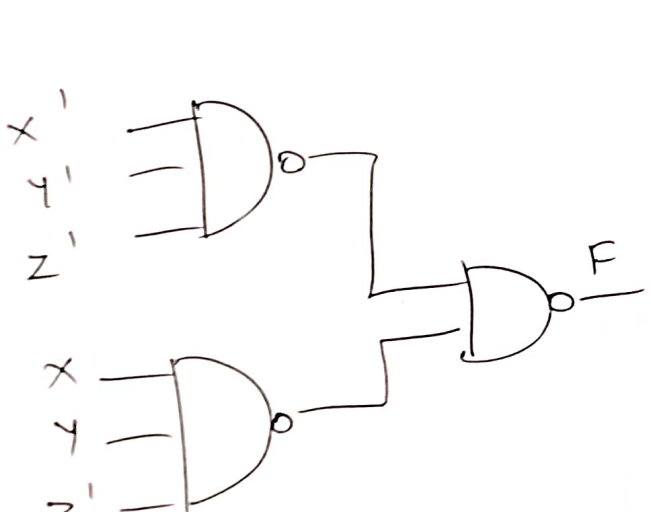
Ex: 3.9 Implement the following function with NAND gates.

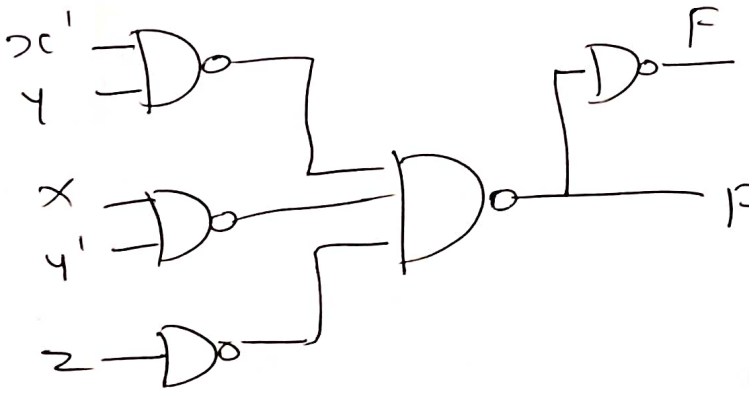
$$F(x,y,z) = \Sigma(0,6)$$

1	0	0	0
0	0	0	1

$$F = x'y'z' + xyz'$$

$$F' = x'y + xy' + z$$

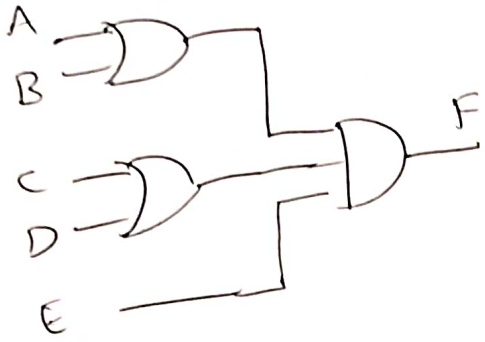


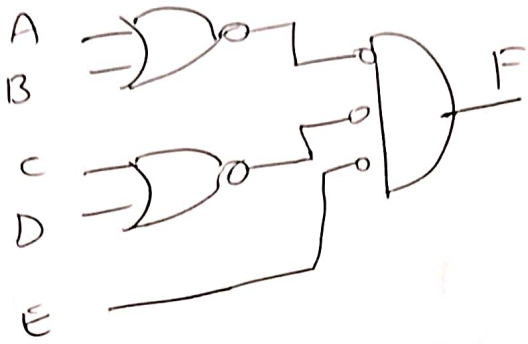


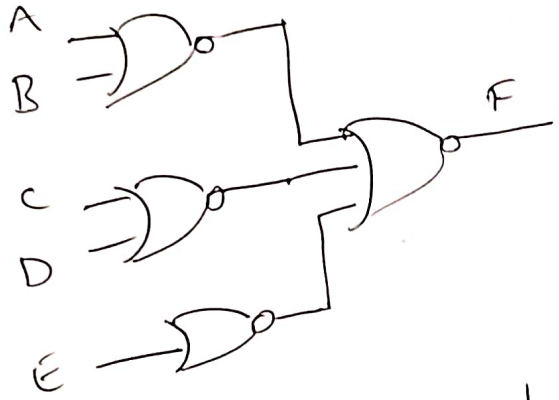
NOR implementation:

→ The NOR function is the dual of the NAND function.

$$F = (A+B)(C+D)E$$







Three ways to implement $F = (A+B)(C+D)E$

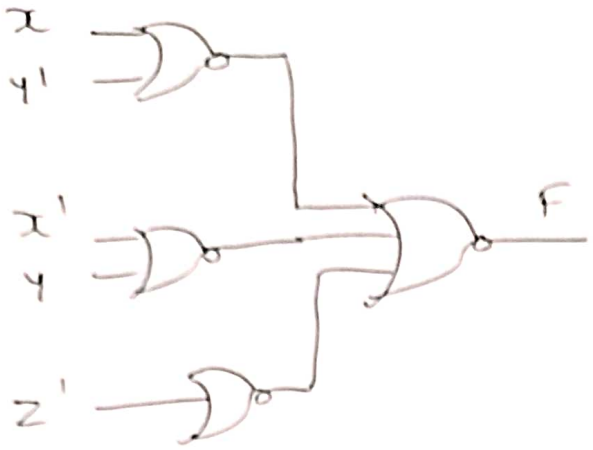
Ex: 3.10 Implement the function

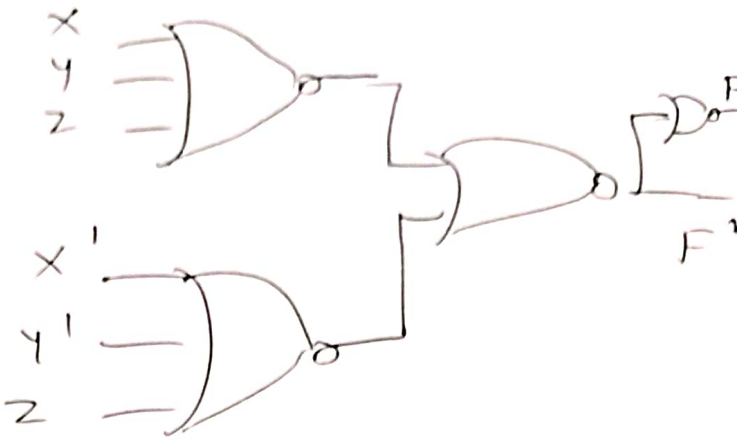
$$F(x, y, z) = \Sigma(0, 6)$$

combining 0's

$$F' = x'y + xy' + z$$

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





Rules for Nand & NOR implementation

case	Function to simplify	standard form to use	How to derive	Implement with
(a)	F	SOP	combine 1's in map	NAND
(b)	F'	SOP	combine 0's in map	NAND
(c)	F	POS	complement F' in (b)	NOR
(d)	F'	POS	complement F in (a)	NOR

3.11 HDL Implementation Models

Dataflow modeling

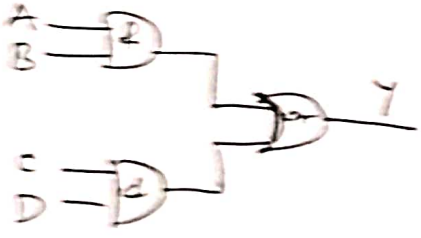
- Verilog provides a keyword assign and a set of operators to describe a circuit through its behaviour or function.

⇒ Here we do not explicitly need to define any gate structure using and, or etc. and it is not necessary to use intermediate variables through wire showing gate level interconnections.

- All assign statements are concurrent. i.e. order in which they appear do not matter.

- Any change in a variable in the right hand side will immediately affect left hand side.

```
module EX2 (A, B, C, D, Y);
input A, B, C, D;
output Y;
assign Y = (A & B) | (C & D);
end module
```



3.11 HDL Implementation Models

Dataflow modeling :

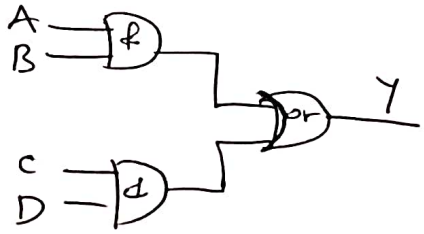
→ verilog provides a keyword assign and a set of operators to describe a circuit through its behaviour or function.

→ Here we do not explicitly need to define any gate structure using and, or etc. and it is not necessary to use intermediate variables through wire showing gate level interconnections.

→ All assign statements are concurrent. ie. order in which they appear do not matter

→ Any change in a variable in the right hand side will immediately effect left hand side.

```
module ex2 (A, B, C, D, Y);
input A, B, C, D ;
output Y ;
assign Y = (A & B) | (C & D);
end module
```



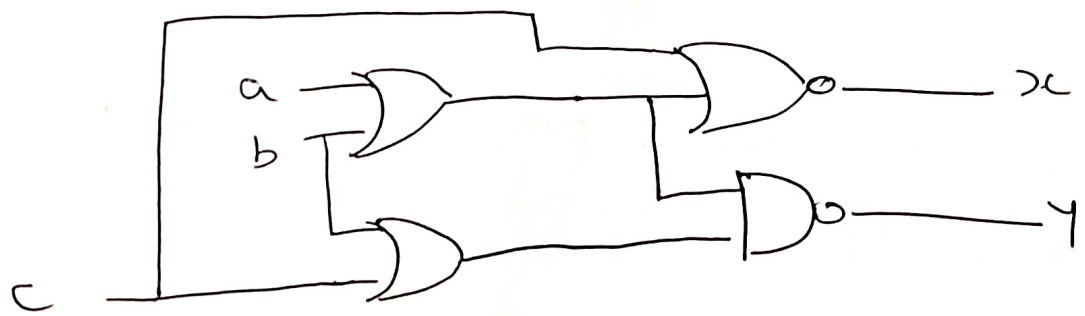
Relational operation	symbol
less than	<
less than or equal	<=
Greater than	>
Equal to	==
Not equal to	!=

Bitwise operation	symbol
Bitwise NOT	~
Bitwise AND	&
Bitwise OR	|
Bitwise EX-OR	^

logical operation	symbol
Logical NOT	!
Logical AND	&&
Logical OR	||

Arithmetic operation	symbol
Binary addition	+
Binary subtraction	-
Binary multiplication	*
Binary division	/

```
module testckt (a, b, c, x, y);
input a, b, c;
output x, y;
assign x = ~ ((a | b) | c);
assign y = ~ ((a | b) & (b | c));
end module.
```



Behavioral Modeling :

→ In a behavioral model, statements are executed sequentially following arithmetic description.

→ It always uses always keyword followed by sensitivity list.

→ The procedural statements following always is executed only if any variable within sensitivity list changes its value.

→ procedure assignment or output variables within always must be of register type, defined by reg which unlike wire is not continuously updated but only after a new value is assigned to it.

Ex : $Y = \bar{C}' + \bar{A}'\bar{D}' + A\,B'\,D'$

```
module ex_1 (A, B, C, D, Y);
input A, B, C, D;
output Y;
assign Y = ~C | (~A & ~D) | (~B & ~D);
End module.
```

↓ Dataflow modeling

Ex 2 : $Y = AB + CD$

$Y = 1$ if $AB = 11$ or if $CD = 11$

otherwise $Y = 0$

```
module ex_2 (A, B, C, D, Y);
input A, B, C, D;
output Y;
reg Y;
always @ (A or B or C or D)
  if ((A==1) && (B==1))
    Y = 1;
  elseif ((C==1) && (D==1))
    Y = 1;
  else
    Y = 0;
endmodule
```

Simplification by Quine-Mc clusky method

→ Reduction of logic equation by K-map method though very simple, becomes difficult to adapt for simplification of 5 or more variables.

→ Quine Mc-clusky method involves Preparation of two tables.

→ one determines prime implicants and the other selects essential Prime implicants to get the minimal expression.

→ prime implicants are expressions with least number of literals that represents all the terms given in the truth table.

→ In stage-1 of the process, we find out all the terms that gives output 1 from truth table and put them in different groups depending on how many 1 input variable combinations have.

→ For example, first group has no 1 in the input combination, second group has only one 1, third two 1's, fourth three 1's, fifth four 1's.

→ In stage 2 we first try to combine first and second group of stage 1.

→ The rule is to see if only one binary digit is differing between two members and we mark that position by '–'.

→ All members of a particular stage, which finds itself in atleast one combination of next stage are tick (✓) marked.

Selection of prime implicants:

→ After finishing combining with all stages (Further stage not possible) we have the prime implicants at the last stage.

→ we have to select the essential Prime implicants.

→ For this we have to prepare a table that along the row lists all the prime implicants and along columns lists all minterms.

→ The crosspoint of a row and column is ticked if the term is covered by corresponding Prime implicant.

→ Selection of essential prime implicants from this table is done in the following way.

→ we find minimum number of prime implicants ~~from~~ that covers all the minterms.

→ we find that two prime implicants cover the same terms, one can be neglected.

Ex: (Malvino) Give the simplified logic equation by Quine - Mc-cluskey method.

$Y(A,B,C) = \Sigma\,(2, 6, 7)$

Stage - 1	Stage - 2
ABC	ABC
010 (2)	- 10 (2,6)
↑10 (6)	
111 (7)	11 - (6, 7)

	2	6	7
BC'	✓	✓	
AB		✓	✓

$Y = AB + BC'$

Ex 2 : $F = \Sigma(0,1,2,3,10,11,12,13,14,15)$

(Mahino)

Stage-1

A B C D	
0 0 0 0	(0) ✓
0 0 0 1	(1) ✓
0 0 1 0	(2) ✓
0 0 1 1	(3) ✓
1 0 1 0	(10) ✓
1 1 0 0	(12) ✓
1 0 1 1	(11) ✓
1 1 0 1	(13) ✓
1 1 1 0	(14) ✓
1 1 1 1	(15) ✓

Stage-2

A B C D	
0 0 0 -	(0,1) ✓
0 0 - 0	(0,2) ✓
0 0 - 1	(1,3) ✓
0 0 1 -	(2,3) ✓
- 0 1 0	(2,10) ✓
- 0 1 1	(3,11) ✓
1 0 1 -	(10,11) ✓
1 - 1 0	(10,14) ✓
1 1 0 -	(12,13) ✓
1 1 - 0	(12,14) ✓
1 - 1 1	(11,15) ✓
1 1 - 1	(13,15) ✓
1 1 1 -	(14,15) ✓

Stage-3

A B C D	
0 0 - -	(0123)
0 0 - -	(0213)
- 0 1 -	(2,10,3,11)
1 - 1 -	(10,11,14,15)
1 - 1 -	(10,14,11,15)
1 1 - -	(12,13,14,15)
1 1 - -	(12,14,13,15)

	0	1	2	3	10	11	12	13	14	15
$A'B'$ (0,1,2,3)	✓	✓	✓	✓						
$B'C$ (2,3,10,11)			✓	✓	✓	✓				
AC (10,11,14,15)					✓	✓			✓	✓
AB (12,13,14,15)							✓	✓	✓	✓

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

$Y = A'B' + AC + AB$

Godse 2.29

(3) Simplify the following boolean function by using a Quine Mc clusky method.

$F(A,B,C,D) = \Sigma m(0,2,3,6,7,8,10,12,13)$

Stage - 1

0000 (0) ✓

0010 (2) ✓
1000 (8) ✓

0011 (3) ✓
0110 (6) ✓
1010 (10) ✓
1100 (12) ✓

0111 (7) ✓
1101 (13) ✓

Stage - 2

00-0 (0,2) ✓
-000 (0,8) ✓

001- (2,3) ✓
0-10 (2,6) ✓
-010 (2,10) ✓
10-0 (8,10) ✓
1-00 (8,12)

0-11 (3,7) ✓
011- (6,7) ✓
110- (12,13)

Stage - 3

-0-0 (0,2,8,10)
~~-0-0 (0,8,2,10)~~
0-1- (2,3,6,7)
0-1- (2,6,3,7)

	0	2	3	6	7	8	10	12	13
$A\bar{C}\bar{D}$						✓		✓	
ABC'								✓	✓
$B'D'$	✓	✓				✓	✓		
$A'C$		✓	✓	✓	✓				

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

(4) Y = $\bar{A}'B\,c'\,D' + A'B\,C'\,D + A\,B\,C'\,D' + A\,B\,C'\,D$
$+ A\,B'\,C'\,D + A'\,B'\,C\,D'$

Wodse 2.30

Stage - 1

0 0 1 0 (2)
0 1 0 0 (4) ✓
0 1 0 1 (5) ✓
1 0 0 1 (9) ✓
1 1 0 0 (12) ✓
1 1 0 1 (13) ✓

Stage - 2

0 1 0 - (4,5) ✓
- 1 0 0 (4,12) ✓
- 1 0 1 (5,13) ✓
1 - 0 1 (9,13)
1 1 0 - (12,13) ✓

Stage - 3

- 1 0 -
(4,5,12,13)

Prime implicants

0 0 1 0 (2) → $A'B'C\,D'$
1 - 0 1 (9,13) → $A\,C'D$
- 1 0 - (4,5,12,13) → $B\,C'$

Prime implicants	2	4	5	9	12	13
$A'B'C\,D'$	✓					
$A\,C'D$				✓		✓
$B\,C'$		✓	✓		✓	✓

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

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

Codec 2.31

Stage - 1

0001 (1) ✓
0010 (2) ✓
0100 (4) ✓
1000 (8) ✓

0011 (3) ✓
0101 (5) ✓
1001 (9) ✓
1100 (12) ✓

1011 (11) ✓
1110 (14) ✓

1111 (15) ✓

Stage 2

00 - 1 (1, 3) ✓
0 - 01 (1, 5)
- 001 (1, 9) ✓
00 - (2, 3)
010 - (4, 5)
- 100 (4, 12)
100 - (8, 9)
1 - 00 (8, 12)

- 011 (3, 11) ✓
1 0 - 1 (9, 11) ✓
11 - 0 (12, 14)

1 - 11 (11, 15)
111 - (14, 15)

Stage - 3

- 0 - 1
(1, 3, 9, 11)

List of prime implicants

0 - 01 (1, 5)
~~0 0 1 -~~ (2, 3)
0 0 1 - (4, 5)
0 1 0 - (4, 12)
- 100 (8, 9)
1 0 0 - (8, 12)
1 - 00 (12, 14)
11 - 0 (11, 15)
1 - 11 (14, 15)

Prime implicants		1	2	3	4	5	8	9	11	12	14	15
1, 5	$w'y'z$	✓				✓						
2, 3	$w'x'y'$		✓	✓								
4, 5	$w'xy'$				✓	✓						
4, 12	$xy'z'$				✓					✓		
8, 9	$wx'y'$						✓	✓				
8, 12	$wy'z'$						✓			✓		
12, 14	wxz'									✓	✓	
11, 15	wyz								✓			✓
14, 15	wxy										✓	✓
1, 3, 9, 11	$x'z$	✓		✓				✓		✓		
		x	x	x	x	x			x	x		x

$w'x'y' + wyz + w'y'z + xy'z' +$

$Y = w'y'z + w'x'y + wxz' + wxy + x'z$

(6) Simplify the following 5 variable boolean expression using Quine Mcclusky method

Ans

$$F = \Sigma m(0,1,9,15,24,29,30) + d(8,11,31)$$

Stage - 1

00000 (0) ✓
00001 (1) ✓
01000 (8) ✓
01001 (9) ✓
11000 (24) ✓
0110- (11) ✓
01111 (15) ✓
11101 (29) ✓
11110 (30) ✓
11111 (31) ✓

Stage - 2

0000- (0,1) ✓
0-000 (0,8) ✓
0-001 (1,9) ✓
01 00- (8,9) ✓
-1000 (8,24)
010-1 (9,11)
01-11 (11,15)
-1111 (15,31)
111-1 (29,31)
1111- (30,31)

Stage - 3

(0189)
0--00-

Prime implicants.	0	1	X 8	9	X 11	15	24	29	30	X 31
BC'D'E'			✓				✓			
A'BC'E				✓	✓					
A'BDE					✓	✓				
BCDE						✓				✓
ABCE								✓		✓
ABCD									✓	✓
A'C'D'	✓	✓	✓	✓						

$Y = BC'D'E' + BCDE + ABCE + ABCD + A'C'D'$

Ex 3.13 Morris mano

Simplify the following boolean function by using tabulation method.

$F = \Sigma(0, 1, 2, 8, 10, 11, 14, 15)$

Stage-1

0000 (0) ✓
0001 (1) ✓
0010 (2) ✓
1000 (8) ✓
1010 (10) ✓
1011 (11) ✓
1110 (14) ✓
1111 (15) ✓

Stage-2

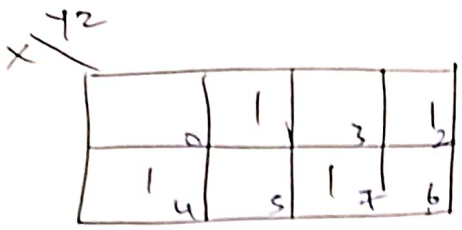
000 - (0, 1) ✓
00 - 0 (0, 2) ✓
- 000 (0, 8) ✓
- 010 (2, 10) ✓
1 0 - 0 (8, 10) ✓
101 - (10, 11) ✓
1 - 10 (10, 14) ✓
1 - 11 (11, 15) ✓
111 - (14, 15) ✓

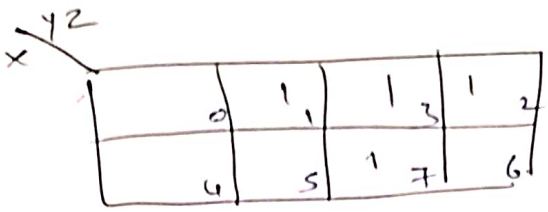
Stage-3

- 0 - 0 (0, 2, 8, 10)
- 0 - 0 (0, 8, 2, 10)
1 - 1 - (10, 11, 14, 15)
1 - 1 - (10, 14, 11, 15)

	1	4	6	7	8	9	10	11	15
$x'y'z$ (1, 9)	X					X			
$w'xz'$ (4, 6)		X	X						
$w'xy$ (6, 7)			X	X					X
xyz (7, 15)				X					
wyz (11, 15)								X	X
wx' (8, 9, 10, 11)					X	X	X	X	

$F = x'y'z + w'xz' + wx' + xyz$





$D = x'y'z + x'yz + xy'z' + xyz$

$B = x'y + x'z + yz$

4.5 Code conversion

→ The availability of a large variety of codes for the same discrete elements of information results in the use of different codes by different digital systems.

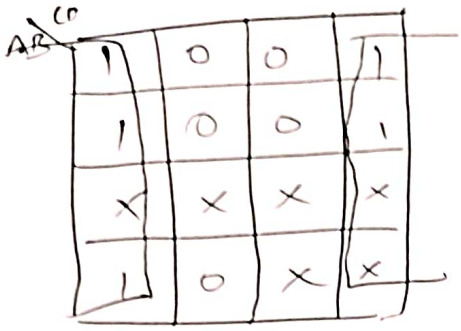
→ It is sometimes necessary to use the output of one system as the input to another.

→ A code converter is a circuit that makes the two systems compatible even though each uses a different binary code.

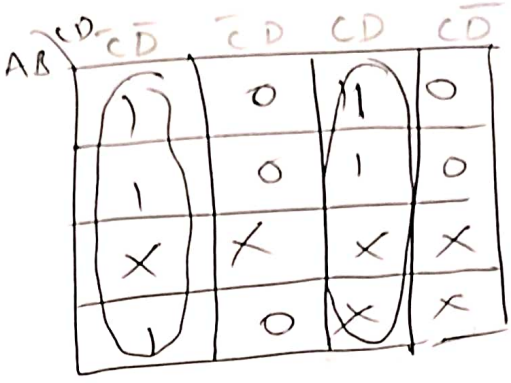
→ To convert from binary code A to binary code B, the input lines must supply the bit combination of elements as specified by code A and output lines must generate the corresponding bit combination of code B

Truth table for code-conversion

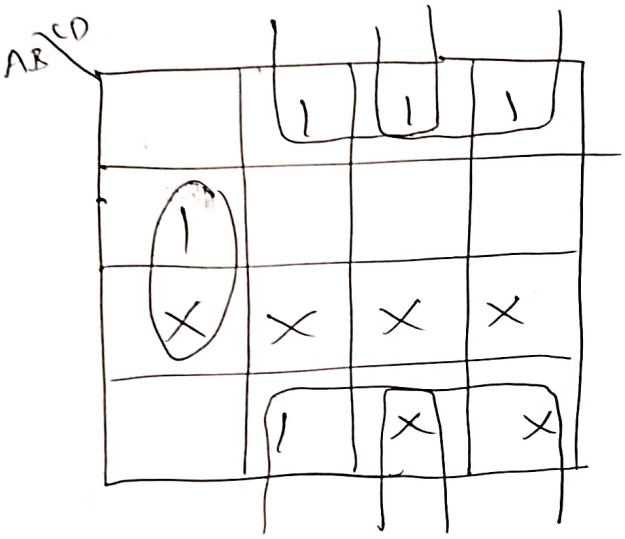
Input BCD				output Excess-3			
A	B	C	D	W	X	Y	Z
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0



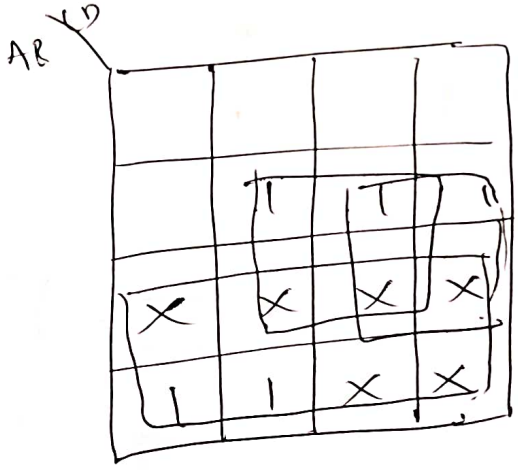
$Z = D'$



$Y = CD + C'D'$



$X = B'C + B'D + BC'D'$



$W = A + BC + BD$

$Z = D'$

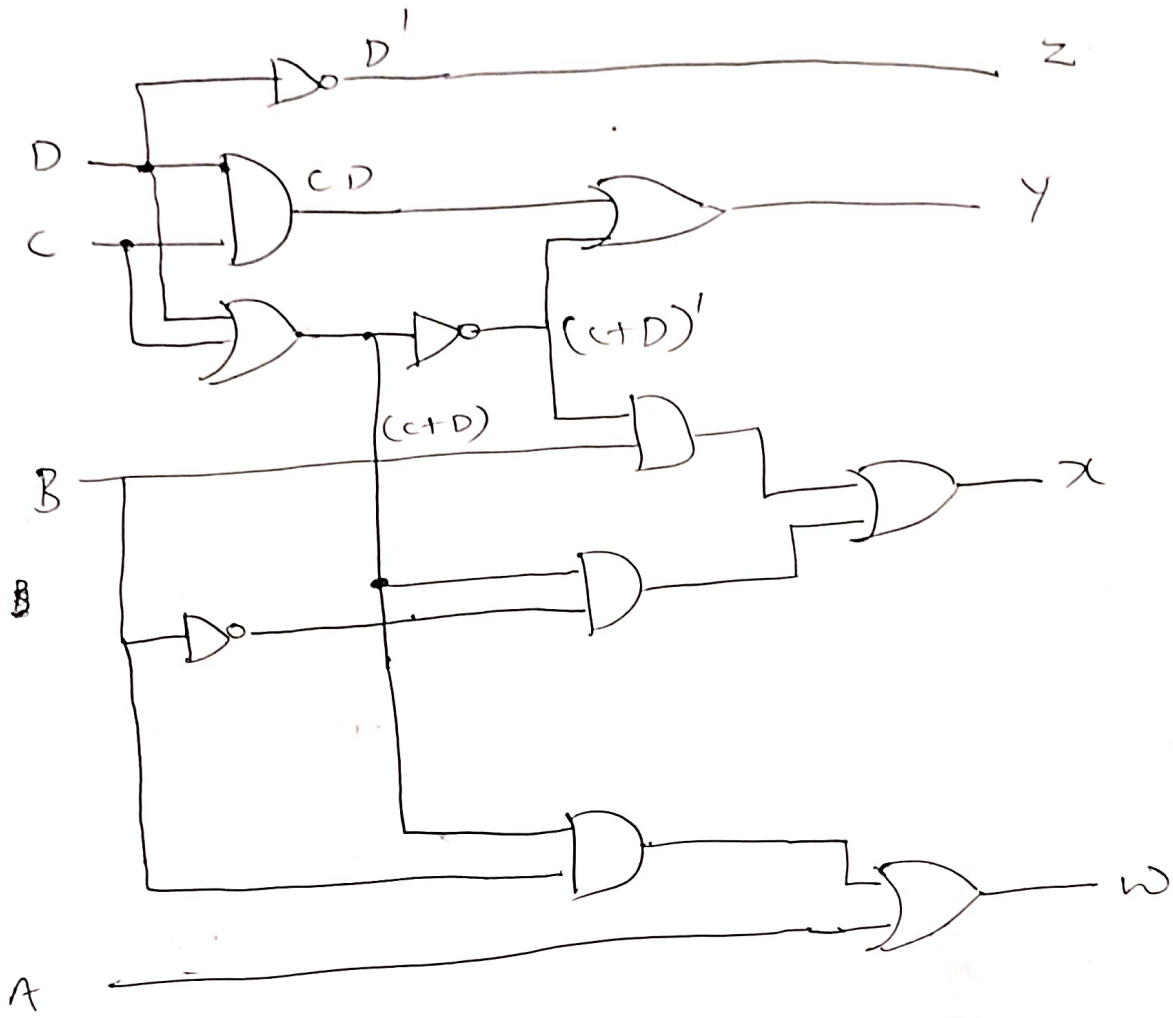
$Y = CD + C'D' = CD + (C+D)'$

$X = B'C + B'D + BC'D'$

$= B'(C+D) + BC'D'$

$= B'(C+D) + B(C+D)'$

$W = A + BC + BD = A + B(C+D)$



Tutorial - 5

1) Design a code converter that converts the given ~~BCD~~ excess-3 number to BCD number (4 bits each)

2) Convert the given binary number (4 bit) to 4 bit gray / reflected code. Design a circuit for the same.

4.6 Analysis Procedure

→ The design of a combinational circuit starts from verbal specifications of a required function and culminates with a set of o/p boolean functions or logic diagram.

→ The analysis of a combinational circuit the reverse process.

→ It starts with a given logic diagram and culminates with a set of boolean functions.

→ The first step in the analysis is to make sure that the given circuit is combinational and not sequential.

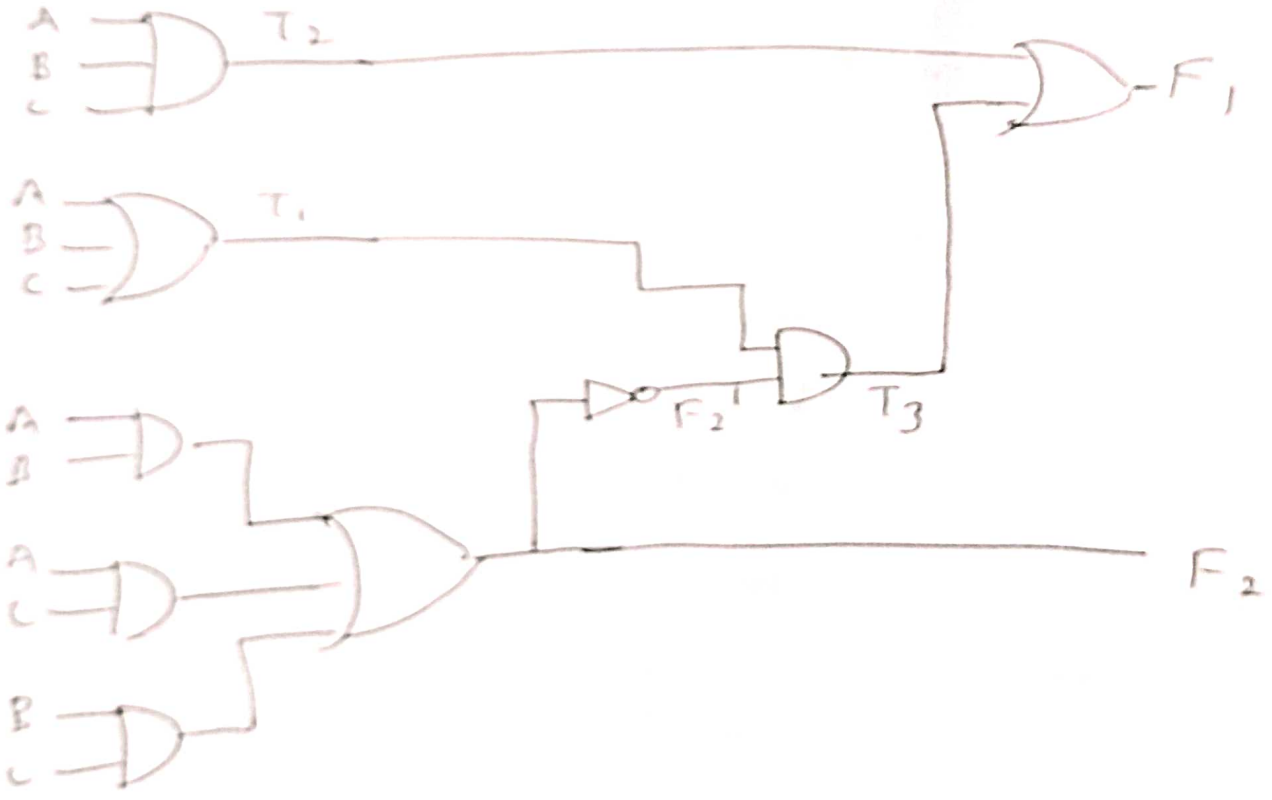
→ To obtain the output boolean functions from a logic diagram, Proceed as follows:

① Label with arbitrary symbols all gate outputs that are a function of the input variables. obtain the boolean functions for each gate.

② Label with other arbitrary symbols those gates which are a function of input variables and/or previously labeled gates. 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 only



$F_2 = AB + Ac + Bc$

$T_1 = A + B + c$

$T_2 = ABC$

$T_3 = F_2' T_1$

$F_1 = T_3 + T_2$

$F_1 = T_3 + T_2$

$= F_2' T_1 + ABC$

$= (AB + Ac + Bc)' (A + B + c) + ABC$

$= (A' + B')(A' + c')(B' + c')(A + B + c) + ABC$

Distributive law

$= (A' + B'c')(AB' + Ac' + Bc' + B'c) + ABC$

$= A'Bc' + A'B'C + AB'c' + ABC.$

Truth table

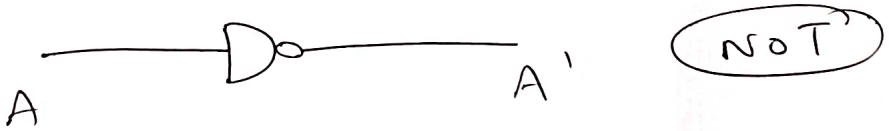
A	B	C	F_2	F_2'	T_1	T_2	T_3	F_1
0	0	0	0	1	0	0	0	0
0	0	1	0	1	1	0	1	1
0	1	0	0	1	1	0	1	1
0	1	1	1	0	1	0	0	0
1	0	0	0	1	1	0	1	1
1	0	1	1	0	1	0	0	0
1	1	0	1	0	1	0	0	0
1	1	1	1	0	1	1	0	1

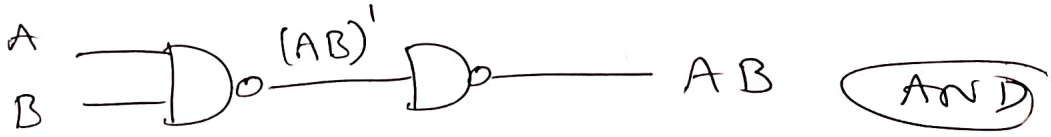
4.7 Multilevel NAND circuits

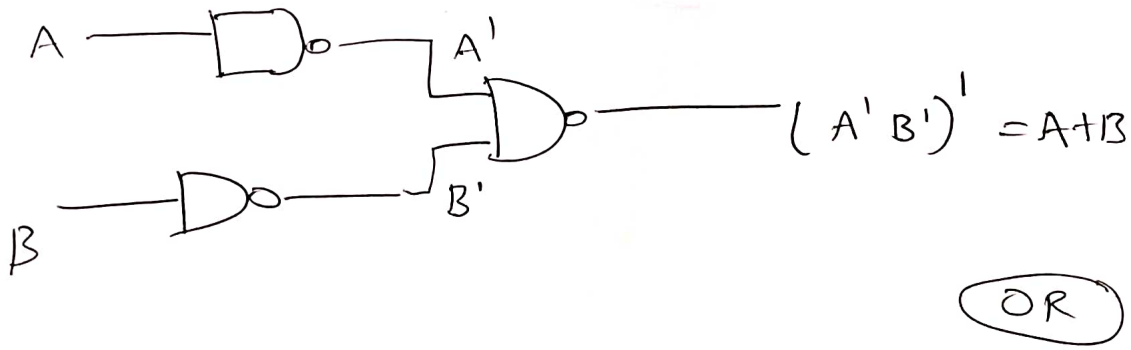
→ Combinational circuits are more frequently constructed with NAND or NOR gates. rather than AND and OR gates.

Universal gate:

→ The NAND gate is said to be a universal gate because any digital system can be implemented with it.







Implementation of NOT, AND or OR by NAND gate

Boolean function implementation

Block diagram method

① From the given algebraic expression, draw the logic diagram with AND, OR and NOT gates. Assume that both the normal & complement inputs are available.

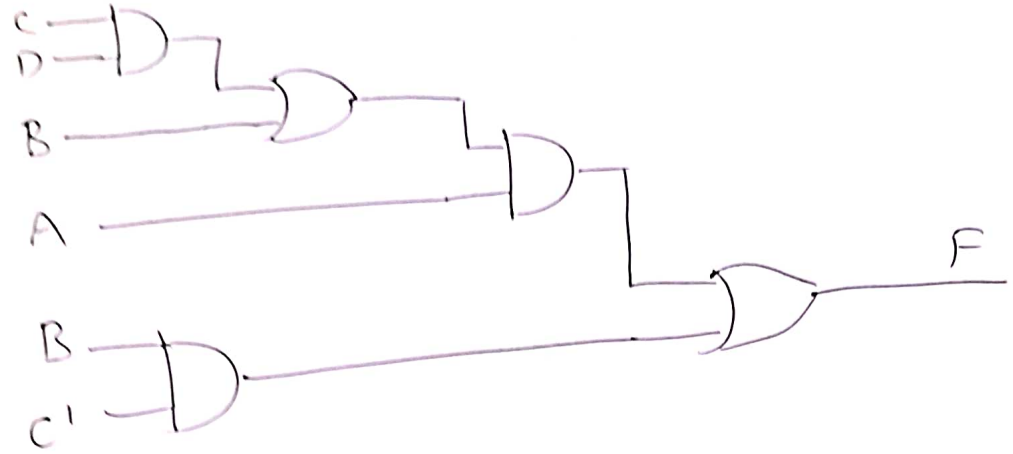
② Draw a second logic diagram with the equivalent NAND logic.

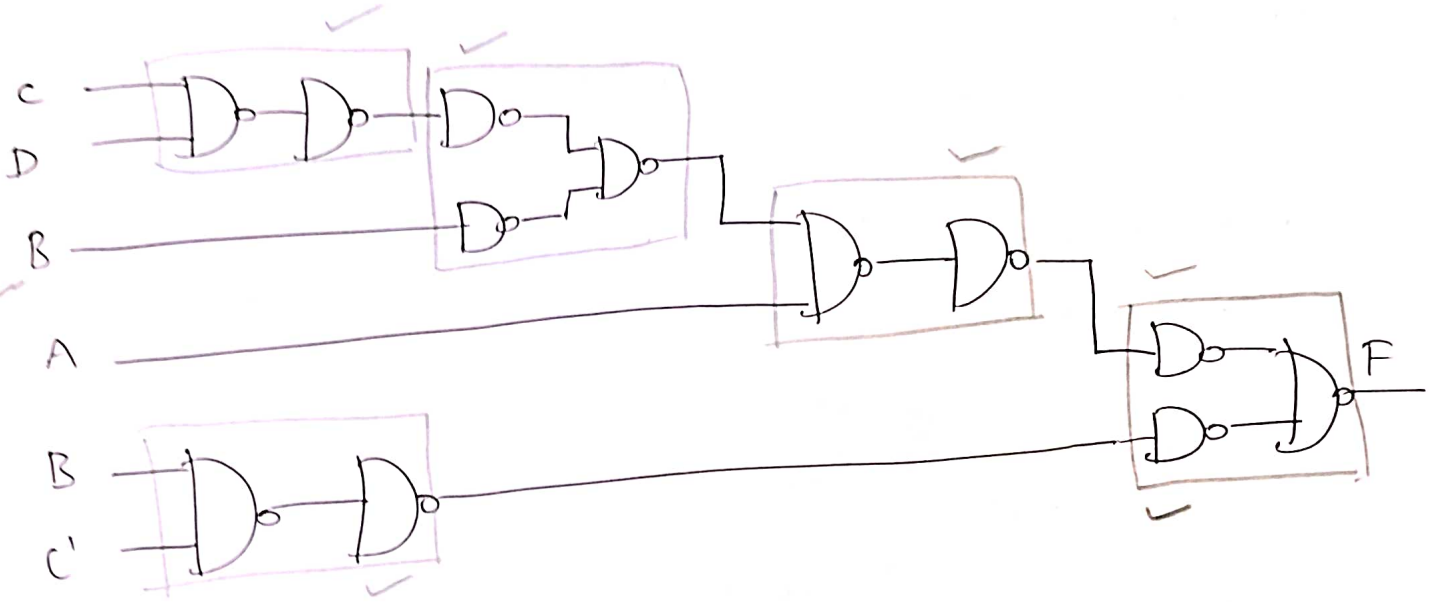
③ • Remove any two cascaded inverters from the diagram, since double inversion does not perform a logic function.

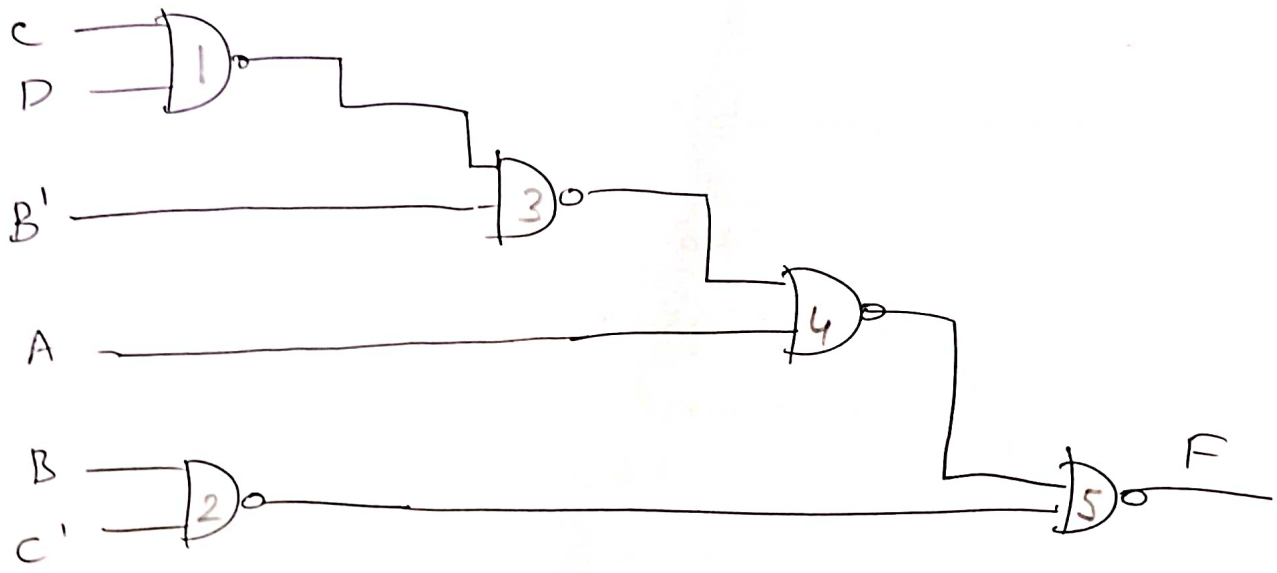
• Remove inverters connected to single external inputs and complement the corresponding input variable.

• The new logic diagram obtained is the required NAND gate implementation.

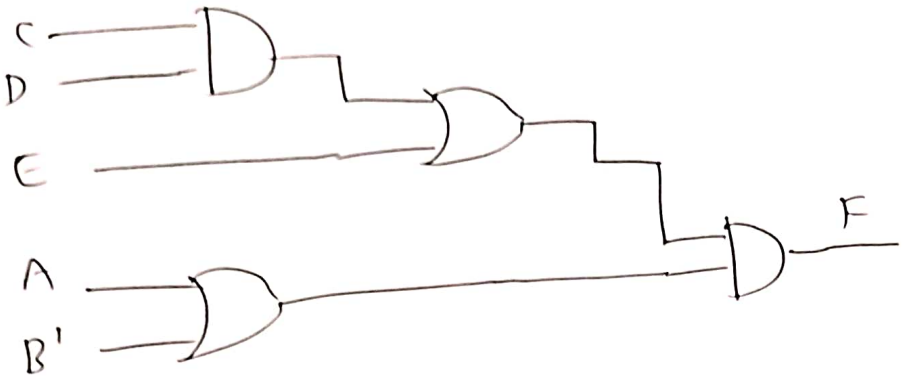
Ex 1 : $F = A(B \cdot C \cdot D) + BC'$

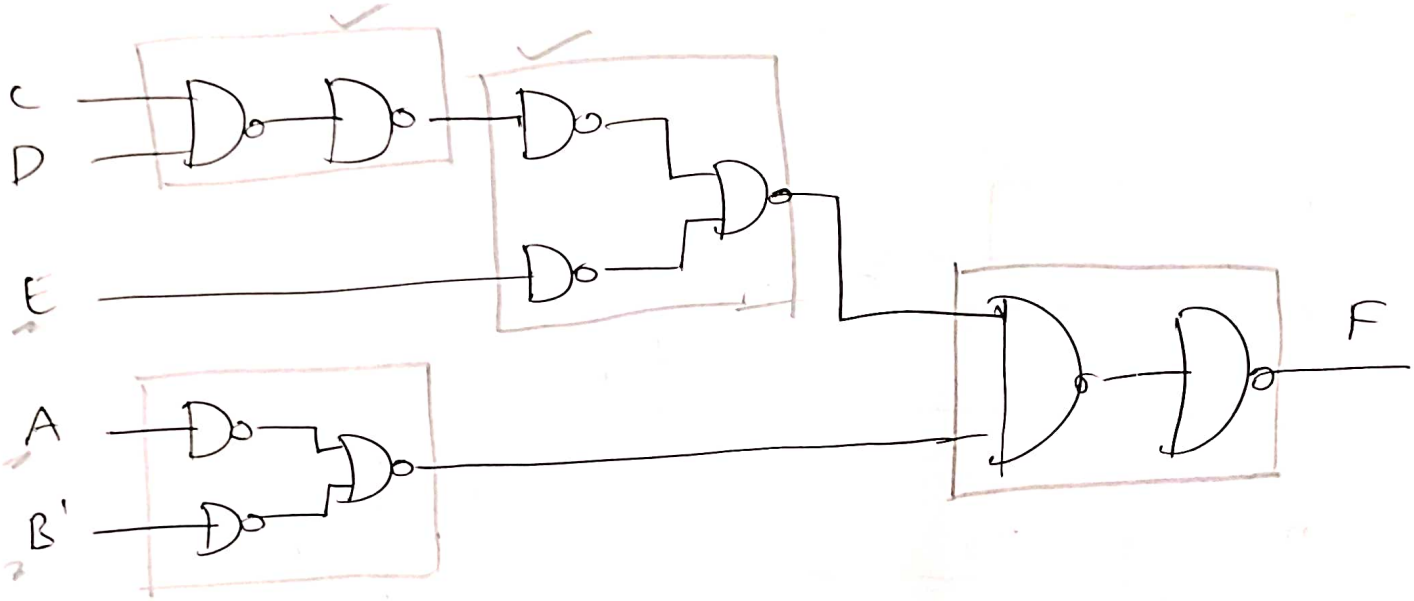


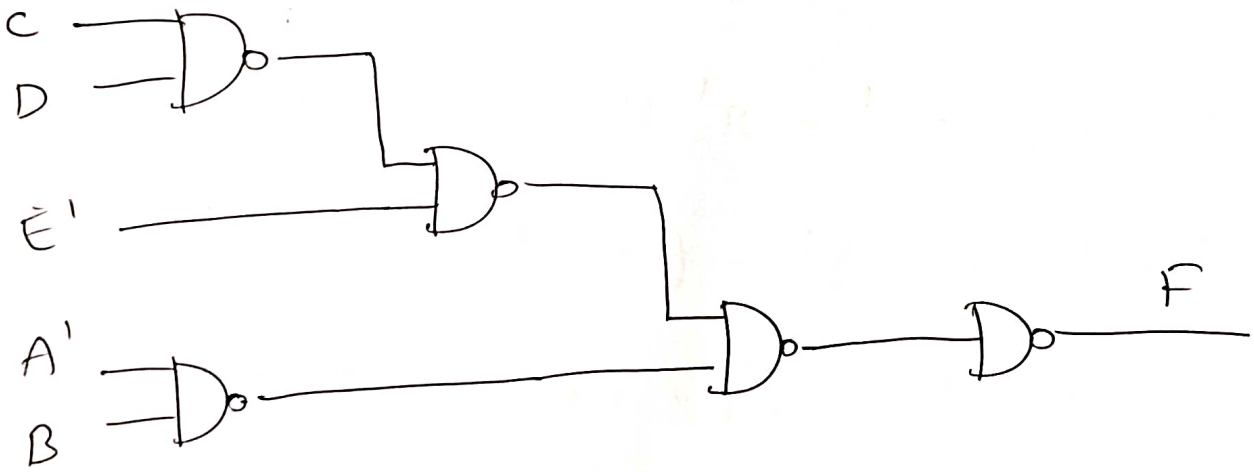




EX 2 $(A + B')(CD + E)$



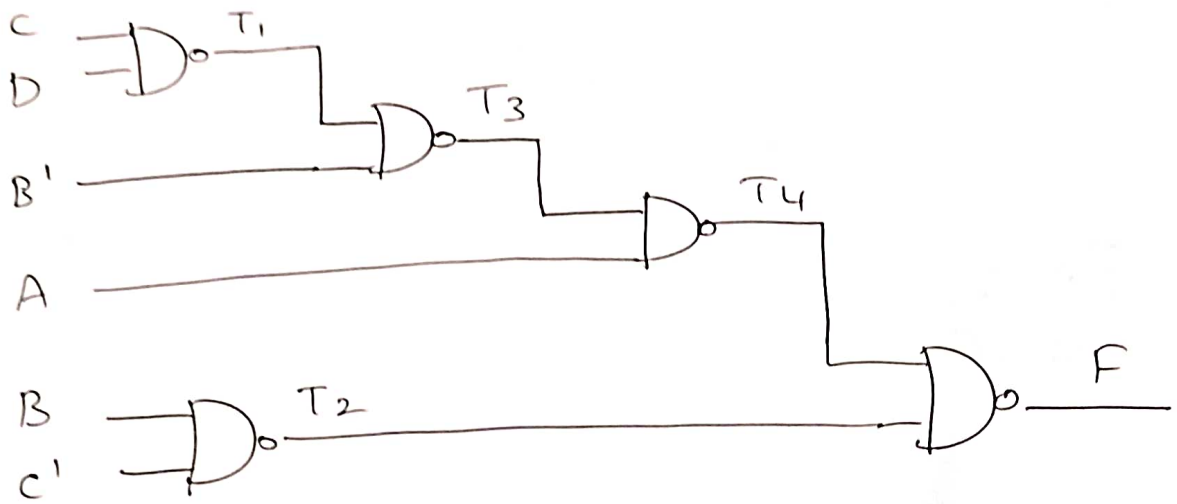




Analysis procedure

→ The foregoing procedure considered the problem of deriving a NAND logic diagram from a given Boolean function.

Derivation of the boolean function by algebraic manipulation :



$$T_1 = (CD)' = C' + D'$$

$$T_2 = (BC')' = B' + C$$

$$T_3 = (B'T_1)' = (B'C' + B'D')'$$

$$= (B + C)(B + D) = B + CD$$

$$T_4 = (AT_3)' = [A(B + CD)]'$$

$$F = (T_2 T_4)' = \{(BC')' [A(B + CD)]'\}'$$

$$= BC' + A(B + CD)$$

Derivation of truth table

A	B	C	D	T_1	T_2	T_3	T_4	F
0	0	0	0	1	1	0	1	0
0	0	0	1	1	1	0	1	0
0	0	1	0	1	1	0	1	0
0	0	1	1	0	1	1	1	0
0	1	0	0	1	0	1	1	1
0	1	0	1	1	0	1	1	1
0	1	1	0	1	1	1	1	0
0	1	1	1	0	1	1	1	0
1	0	0	0	1	1	0	1	0
1	0	0	1	1	1	0	1	0
1	0	1	0	1	1	0	1	0
1	0	1	1	0	1	1	0	1
1	1	0	0	1	0	1	0	1
1	1	0	1	1	0	1	0	1
1	1	1	0	1	1	1	0	1
1	1	1	1	0	1	1	0	1

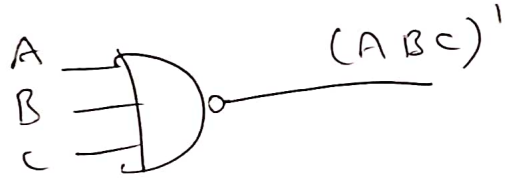
AB \ CD	C'D'	C'D	CD	CD'
A'B'	0	0	0	0
A'B	1	1	0	0
AB	1	1	1	1
AB'	0	0	1	0

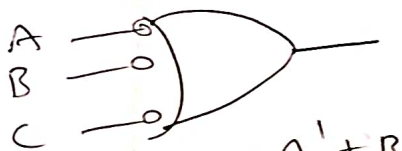
$F = AB + BC' + ACD$

Block diagram Transformation

→ It is sometimes convinient to convert a NAND logic diagram to its equivalent AND-OR logic diagram to facilitate the analysis

Procedure.

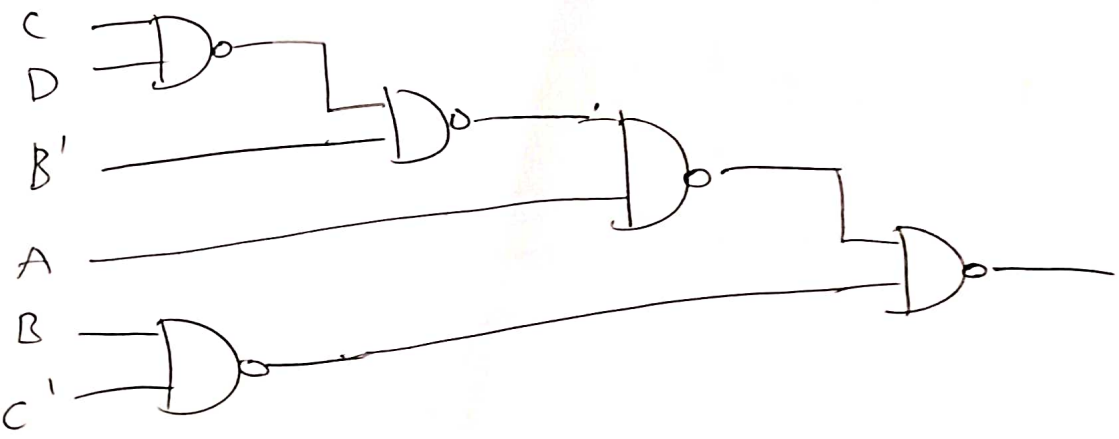


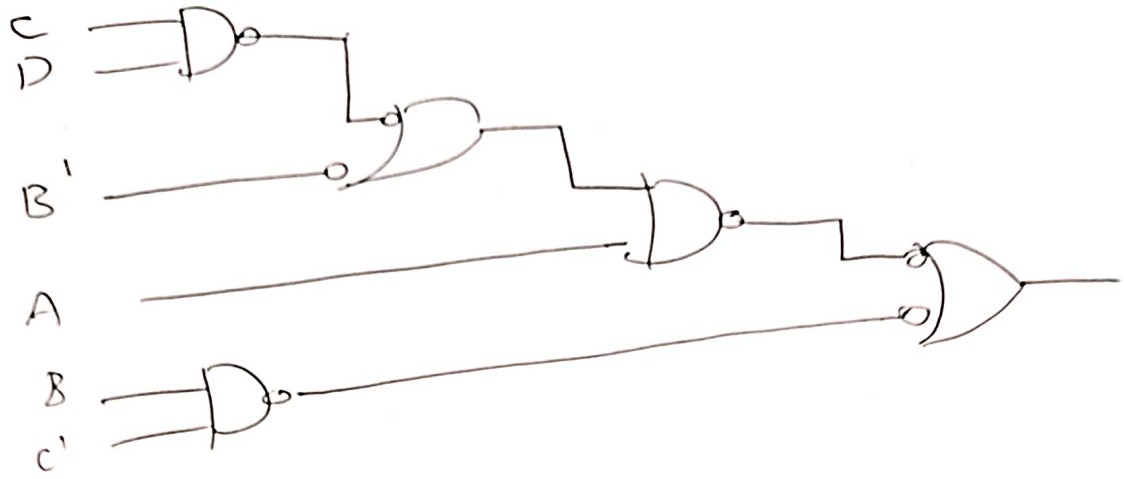


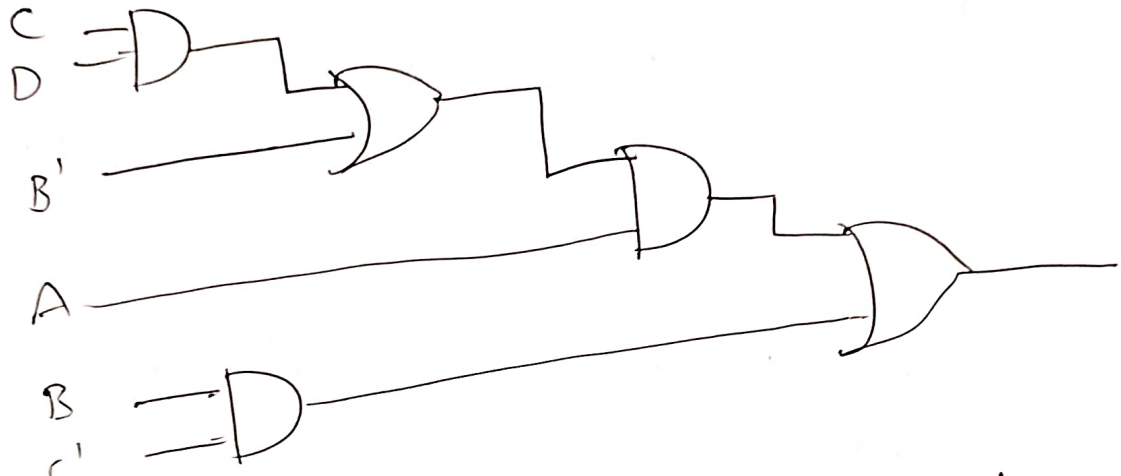
$A' + B' + C'$

$= (ABC)'$

EX :



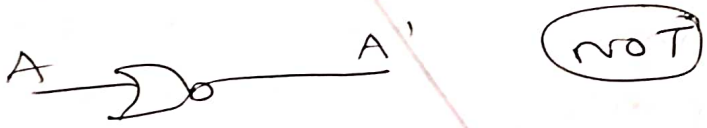


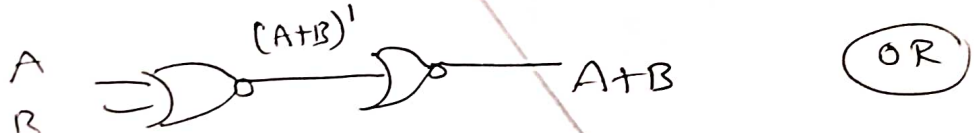


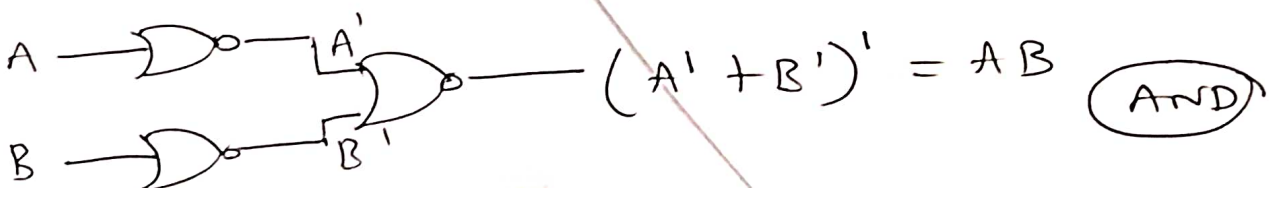
4.8 multilevel NOR circuit

→ The NOR is the dual of the NAND function.

universal gate:







The Reflected code : (Xtra Data useful for Problems)

→ Digital systems can be designed to Process data in discrete form only.

→ Many physical systems supply continuous output data.

→ These data must be converted into digital or discrete form before they are applied to a digital system.

→ continuous or analog information is converted into digital form by means of an analog-to digital converter.

→ It is sometimes convinient to use the reflected code.

→ The advantage of the reflected code over pure binary numbers is that a number in the reflected code changes by only one bit as it Proceeds from one number to the next.

→ A typical application of the reflected code occurs when the analog data are represented by a continuous change of a shaft position.

(a) Message	P(odd)
0000	1
0001	0
0010	0
0011	1
0100	0
0101	1
0110	1
0111	0
1000	0
1001	1
1010	1
1011	0
1100	1
1101	0
1110	0
1111	1

(b) Message	P(even)
0000	0
0001	1
0010	1
0011	0
0100	1
0101	0
0110	0
0111	1
1000	1
1001	0
1010	0
1011	1
1100	0
1101	1
1110	1
1111	0

→ The parity method detects the presence of one, three or any odd combination of errors.

→ An even combination of errors is undetable.

→ If errors occur so often as to distort the meaning of the received information the system is checked for malfunction.

→ A parity bit is an extra bit included with a message to make the total number of 1's either odd or even

→ P is chosen so that the sum of all 1's is odd. for checking odd parity.

→ P is chosen so that the sum of all 1's is even for checking. even parity.

→ In the sending end the message is applied to a "parity generation" network where the required P bit is generated.

→ The message including the parity bit is transferred to its destination.

→ In the receiving end, all the incoming bits are applied to a "parity check" network to check the proper parity adopted.

→ An error is detected if the checked parity does not correspond to the adopted one.

→ one or more bits may change value
→ A circuit in the receiving side can detect the presence of more than two 1's and if the received combination of bits does not agree with the allowable combination, an error is detected

Error detection codes:

→ Binary information, be it pulse modulated signals or digital computer input or output, may be transmitted through some form of communication medium such as wires or radio waves.

→ Any external noise introduced into a physical communication medium changes bit values from 0 to 1 or vice versa.

→ An error detection code can be used to detect errors during transmission.

→ The detected error cannot be corrected but its presence is indicated.

→ The usual procedure is to observe the frequency of errors.

→ If error occurs only once in a while, at random, and without a pronounced effect on the overall information transmitted, then either nothing is done or the particular erroneous message is transmitted again.

→ The BCD code for 13 is

0001 0011

→ The Binary conversion for the same is 1101

→ The excess-3, 2421, 8-4-2-1 are self complementary codes

→ That is 9's complement of the decimal number is easily obtained by changing 1's to 0's & 0's to 1's

→ Ex: 395 in 2421 code is

0 0 1 1 1 1 1 1 1 0 1 1

Its 9's complement 604 is

1 1 0 0 0 0 0 0 0 1 0 0

→ Biquinary code is an example of a seven bit code with error detection properties.

→ Each decimal digit consists of five 0's & two 1's placed in the corresponding weighted columns.

→ During transmission of signals from one location to another, an error may occur.

→ It is also possible to assign negative weights to a decimal code as shown by 8, 4, -2, -1 code.

Decimal digit	8 4 -2 -1	2421
0	0000	0000
1	0111	0001
2	0110	0010
3	0101	0011
4	0100	0100
5	1011	1011
6	1010	1100
7	1001	1101
8	1000	1110
9	1111	1111

→ A decimal code that has been used in some old computers is the excess - 3 code.

→ This is an unweighted code.

→ Its code assignment is obtained from the corresponding value of BCD after the addition of 3.

Decimal digit	Excess - 3
0	0011
1	0100
2	0101
3	0110
4	0111
5	1000
6	1001
7	1010
8	1011
9	1100

→ The examples show that the distinct bit combinations of an n-bit code can be found by counting in binary from 0 to $(2^n - 1)$.

Decimal codes:

→ Binary codes for decimal digits require a minimum of four bits.

Decimal digit	BCD (8421)
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001

→ The BCD (Binary coded decimal) is a straight assignment of the binary equivalent.

$$0110 = 0 \times 8 + 1 \times 4 + 1 \times 2 + 0 \times 1$$

$$= 6$$

395 in binary = 110001011,

→ Two other weighted codes are 2421 and 5043210.

→ 395 in BCD → $\underset{3}{0011}\ \underset{9}{1001}\ \underset{5}{0101}$

1.6 Binary Codes :

→ Electronic digital systems use signals that have two distinct values and circuit elements that have two stable states.

→ A bit by definition is a binary digit.

→ When used in conjunction with a binary code, we can denote it using either 0 or 1.

→ To represent a group of 2^n distinct elements in a binary code requires a minimum of n bits.

→ This is because it is possible to arrange n bits in 2^n distinct ways.

→ For example a group of four distinct quantities can be represented by a two bit code 00, 01, 10, 11.

→ A group of eight elements requires a three bit code, with each element assigned to one and only one of the following : 000, 001, 010, 011, 100, 101, 110, 111.