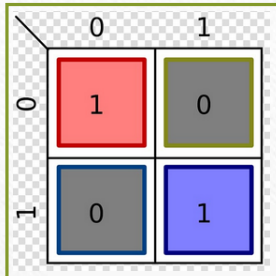
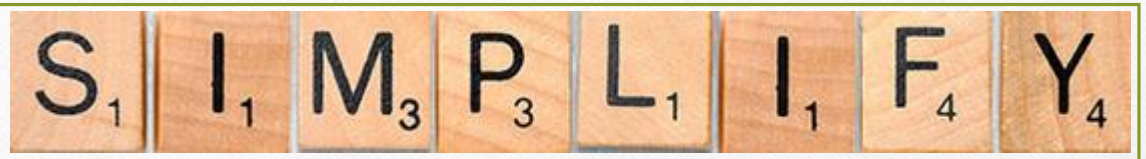


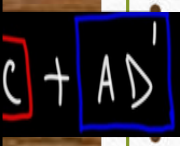


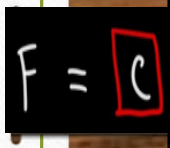
MINIMIZATION OF BOOLEAN FUNCTION USING KARNAUGH MAP METHOD

DEEPA MATHEWS

Minimization of Boolean Function

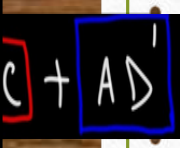
- Two methods for simplification of Boolean functions.
  - The **algebraic method** by using Identities
  - The **graphical method** by using Karnaugh Map method
- Algebraic procedures are difficult to apply in a systematic way. Also it is difficult to tell when you have arrived at a minimum solution.
- The K-map method is easy and straightforward. It is a systematic method of simplifying Boolean expressions and thereby simplifying logic circuits.

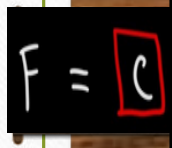




THE KARNAUGH MAP (K -MAP)

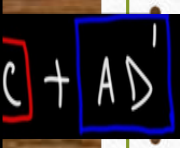
- A K-map is a matrix consisting of rows and columns that represent the output values of a Boolean function.
- One map cell corresponds to a minterm or a maxterm in the boolean expression or a row in the truth table. Multiple-cell areas of the map correspond to standard terms.
- K-map is directly applied to two-level networks composed of AND & OR gates.
  - Sum-of-Products (SOP)
  - Product-of-Sum (POS)

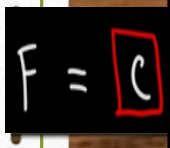




THE KARNAUGH MAP (K -MAP)

- K-map is a tool for simplifying combinational logic with 3 or 4 variables (usually limited to 6 variables);
- A K-map for a function of n variables consists of **2^n cells.** For 3 variables, 8 cells are required (2^3).
- Cells are usually labeled using 0's and 1's to represent the variable and its complement.
- The numbers are entered **in gray code**, to force adjacent cells to be different by only one variable.

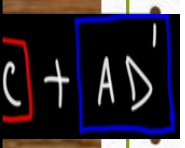


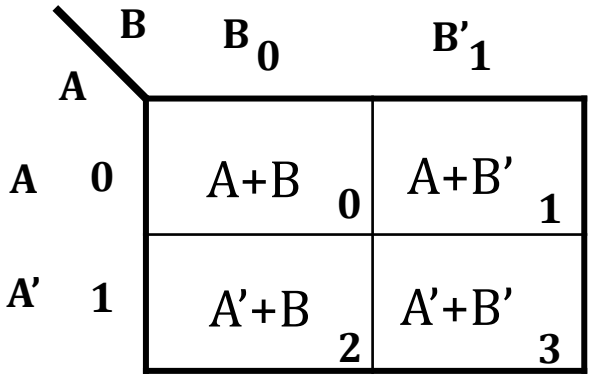


TWO VARIABLE KARNAUGH MAP (K -MAP)

Cell = 2^n ,where n is a number of variables

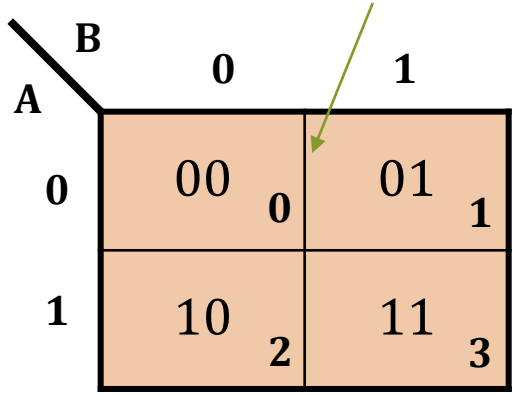
For the case of 2 variables, k-map consists of 2^2=4 cells

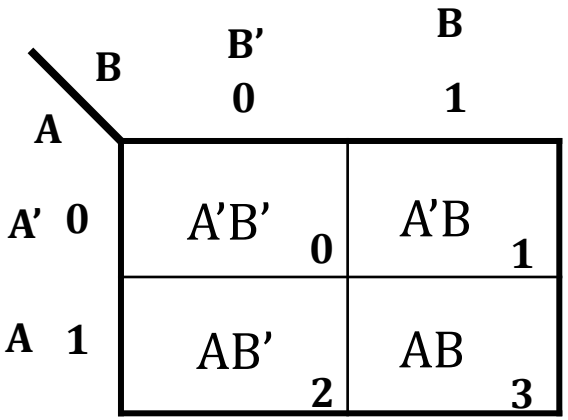




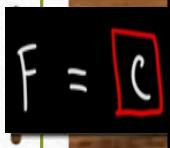
POS - Maxterm

Cell numbers are written in the cells.





SOP - Minterm



THREE VARIABLE KARNAUGH MAP (K -MAP)

For the case of 3 variables, k-map consists of 2^3=8 cells

Gray Code Numbering : 0**0**, **0**1,**1**1,1**0**

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

SOP: -

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

Cell numbers are written in the cells.

A \ B+C	B+C 00	B+$\bar{C}$ 01	$\bar{B}$+$\bar{C}$ 11	$\bar{B}$+C 10
A 0	A+B+C 0	A+B+$\bar{C}$ 1	A+$\bar{B}$+$\bar{C}$ 3	A+$\bar{B}$+C 2
$\bar{A}$ 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

POS: -

A+B \ C	C 0	$\bar{C}$ 1
A+B 0 0	A+B+C 0	A+B+$\bar{C}$ 1
A+$\bar{B}$ 0 1	A+$\bar{B}$+C 2	A+$\bar{B}$+$\bar{C}$ 3
$\bar{A}$+$\bar{B}$ 1 1	$\bar{A}$+$\bar{B}$+C 6	$\bar{A}$+$\bar{B}$+$\bar{C}$ 7
$\bar{A}$+B 1 0	$\bar{A}$+B+C 4	$\bar{A}$+B+$\bar{C}$ 5

FOUR VARIABLE KARNAUGH MAP (K -MAP)

For the case of 4 variables, k-map consists of 2^4=16 cells

Gray Code Numbering : 00**, **0**1,1**1**,1**0

SOP: -

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

POS: -

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

SOP Reduction

using k-Map

RULES FOR SIMPLIFYING THE SOP EXPRESSION USING K-MAPS

- **Select** the respective K-map based on the number of variables present in the given expression.
- **Mapping** – For the given expression, put 1 at respective minterm cells in the K-map for each minterm present in the expression.
- **Grouping** - Check for the possibilities of grouping maximum number of adjacent ones. It should be powers of two.
- **Simplifying** - Each grouping will give either a literal or one product term. It is known as **prime implicant**. The required prime implicants are added to generate the simplified Boolean function.

SOP Reduction – Mapping Minterms

Given an expression, put 1 at respective minterm cells in the K-map for each minterm present in the expression; **Given a truth table**, the map is filled in by placing 1's in squares where minterms lead to a 1 output; Put 0 in the other cells.

Map A'B'C' + A'B'C + ABC' + AB'C'

The binary – 000 + 001 + 110 + 100 and

the minterms are m0,m1,m6 and m4

AB \ C	0	1
00	**1** 0	**1** 1
01	0 2	0 3
11	**1** 6	0 7
10	**1** 4	0 5

SOP Reduction – Mapping Minterms

Map A'B'CD + AB'C'D' + AB'C'D + ABCD

Binary – 0011 + 1000 + 1001 + 1111

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

SOP Reduction – Mapping Minterms (Truth Table)

Given a truth table, the map is filled in by placing 1's in squares where minterms lead to a 1 output; Put 0 in the other cells.

A	B	C	X
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

=

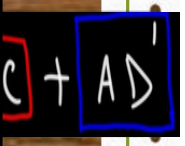
A \ BC	00	01	11	10
0	1	0	0	1
1	1	0	0	1

W	X	Y	Z	F_{WXYZ}
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	1
1	1	1	1	1

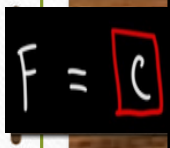
	$\bar{Y}\bar{Z}$	$\bar{Y}Z$	YZ	$Y\bar{Z}$
$\bar{W}\bar{X}$	0 (0)	1 (1)	0 (3)	1 (2)
$\bar{W}X$	1 (4)	1 (5)	1 (7)	0 (6)
WX	1 (12)	0 (13)	1 (15)	1 (14)
$W\bar{X}$	0 (8)	0 (9)	0 (11)	1 (10)

SOP Reduction – Grouping of 1's

For reducing the expression, adjacent 1's are grouped. The goal is to maximize the size of the groups and to minimize the number of groups.

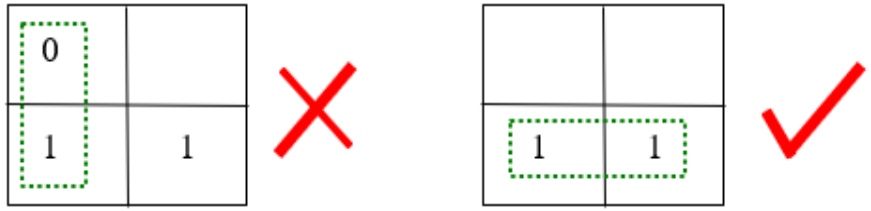


- Groups can be formed only at right angles; diagonal groups are not allowed.
- Number of 1s in a group must be a power of 2 (either 1,2,4,8 or 16 cells).
- The groups must be made as large as possible.
- Groups can overlap and wrap around the sides of the k-map.

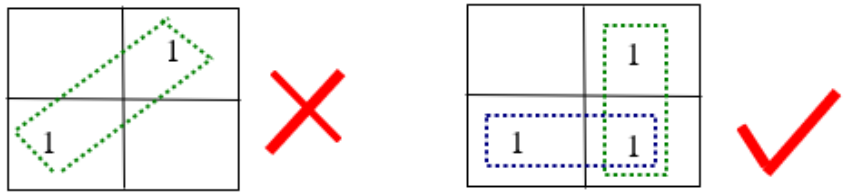


SOP Reduction – Grouping of 1's

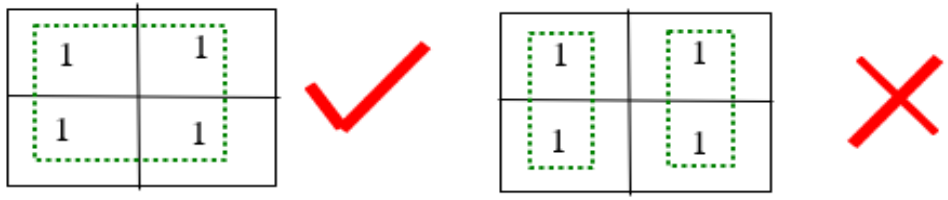
Groups may not include any cells containing a zero



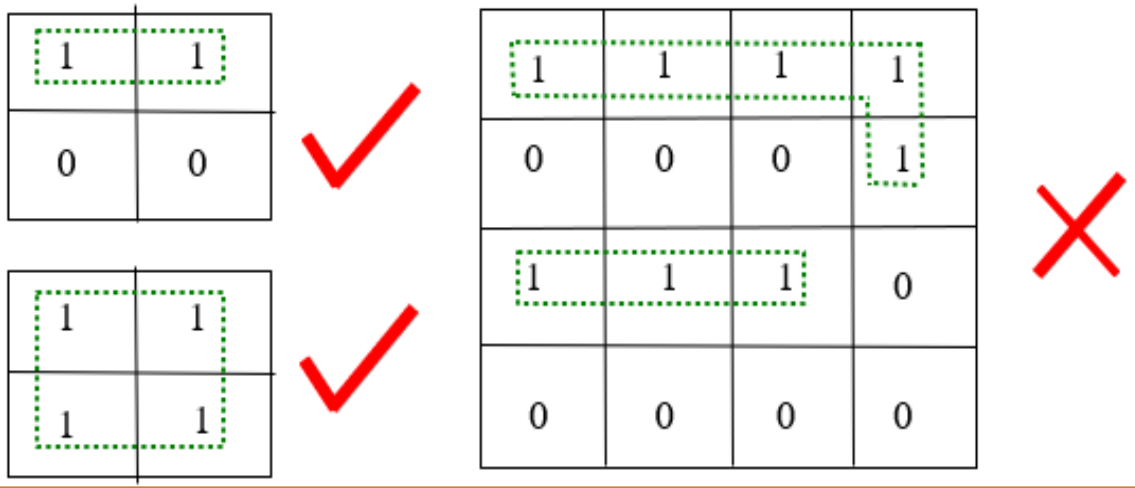
Groups may be horizontal or vertical, but not diagonal



Each group should be as large as possible



Groups must contain 1,2,4,8.. cells (ie. powers of 2)



SOP Reduction – Grouping of 1's

Each cell containing a one must be in atleast one group

1	1	0	0
0	0	0	1

Groups may overlap

1	0
1	0
1	1
1	1

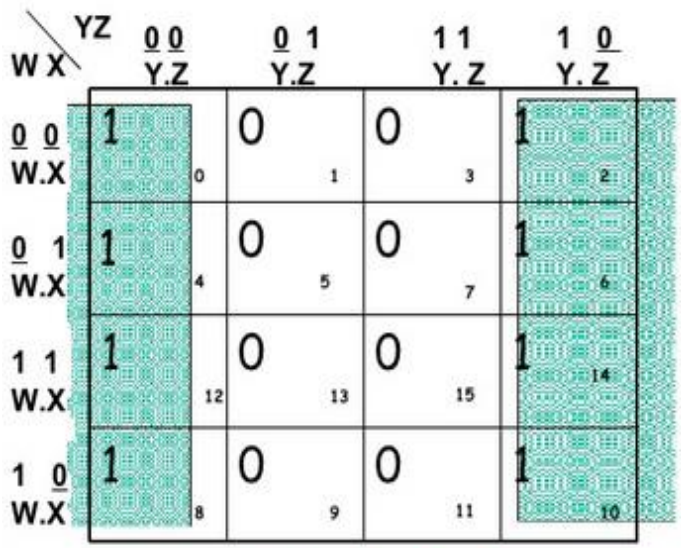
Groups may wrap around the table. The left most cell in a row may be grouped with the rightmost cell and the top cell in a column may be grouped with the bottom cell.

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

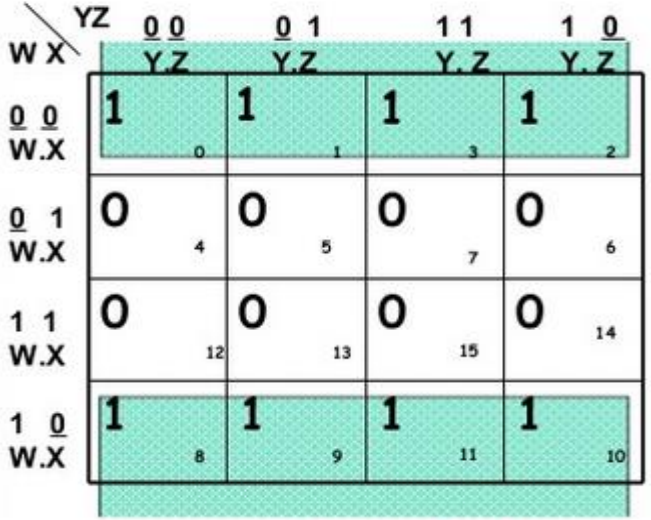
1			1
1			1
1			1
1			1

SOP Reduction – Grouping of 1's

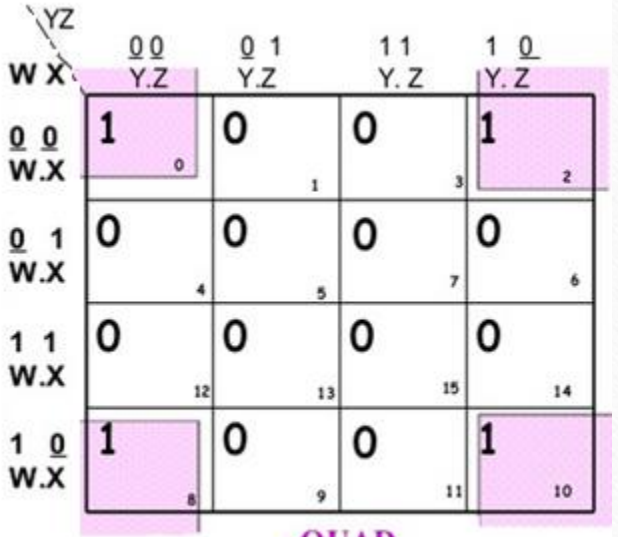
Map Rolling – consider the map as if its left & right edges are touching and the top & bottom edges are touching. While marking the pairs, quads, and octets map must be rolled.



(m0,m2,m4,m6,
m8, m10, m12,m14)



(m0,m1,m2,m3
m8,m9,m10,m11)



QUAD
(m0,m2,m8,m10)

SOP Reduction – Grouping of 1's

Overlapping Groups : Overlapping means same 1 can be encircled more than once. Overlapping always leads to simpler expressions.

Fig 2

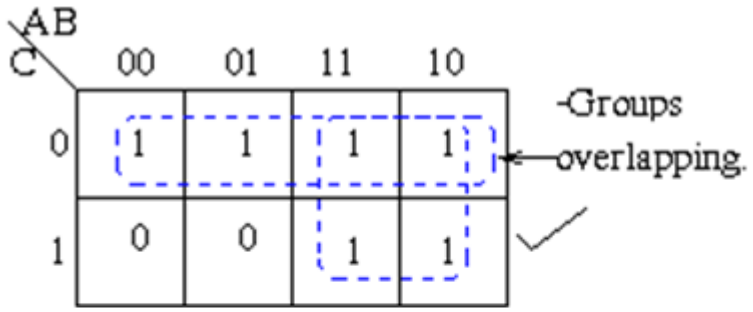


Fig 1

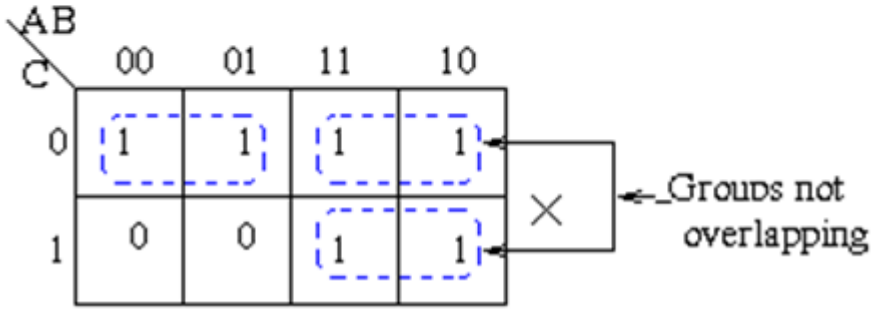
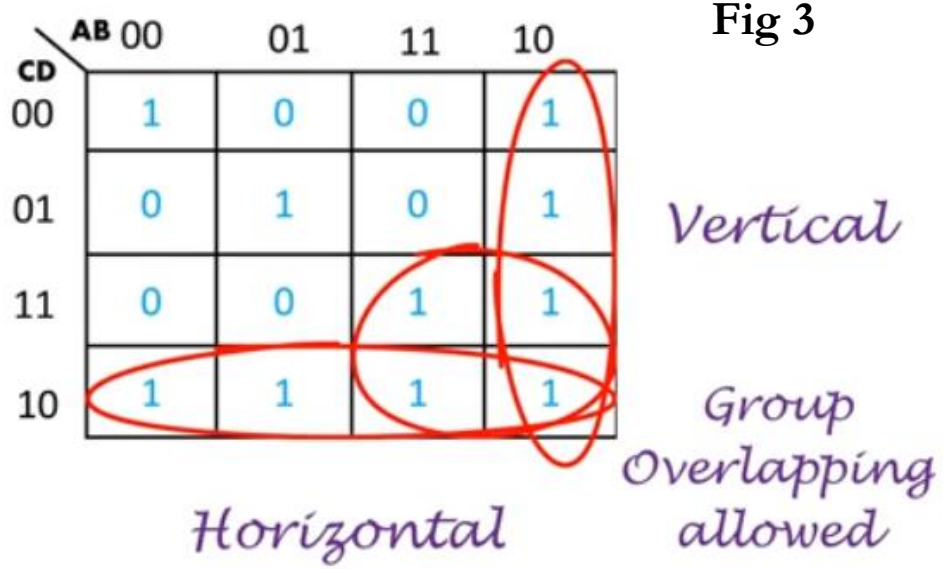
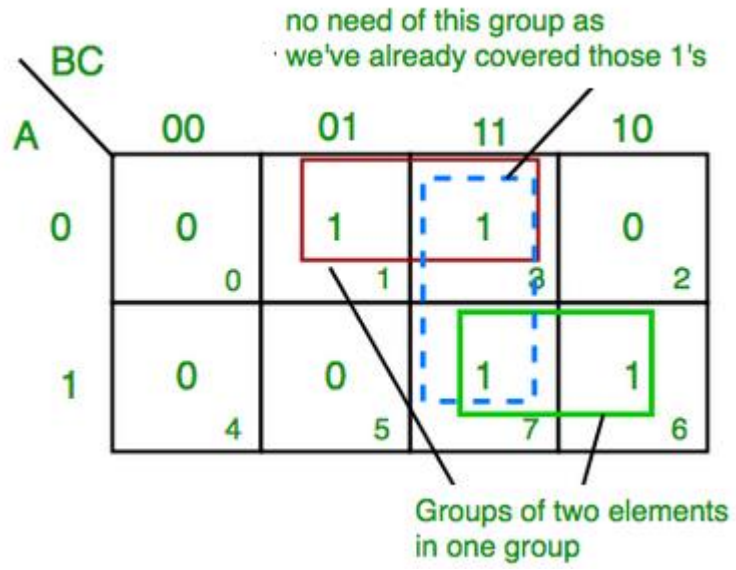


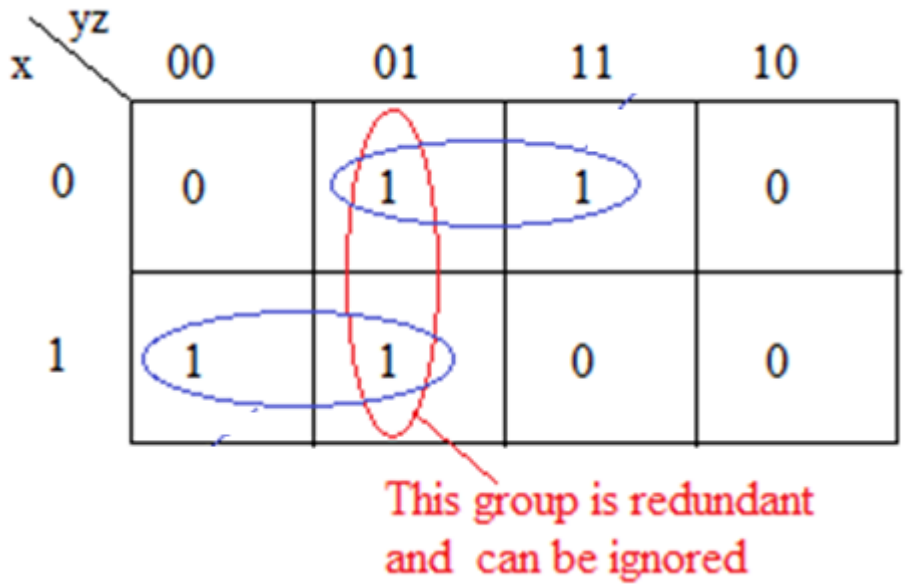
Fig 3



SOP Reduction – Grouping of 1's

Redundant Group: A redundant group is one whose all the 1's have been consumed by other groups. So, there is no need to form such group.

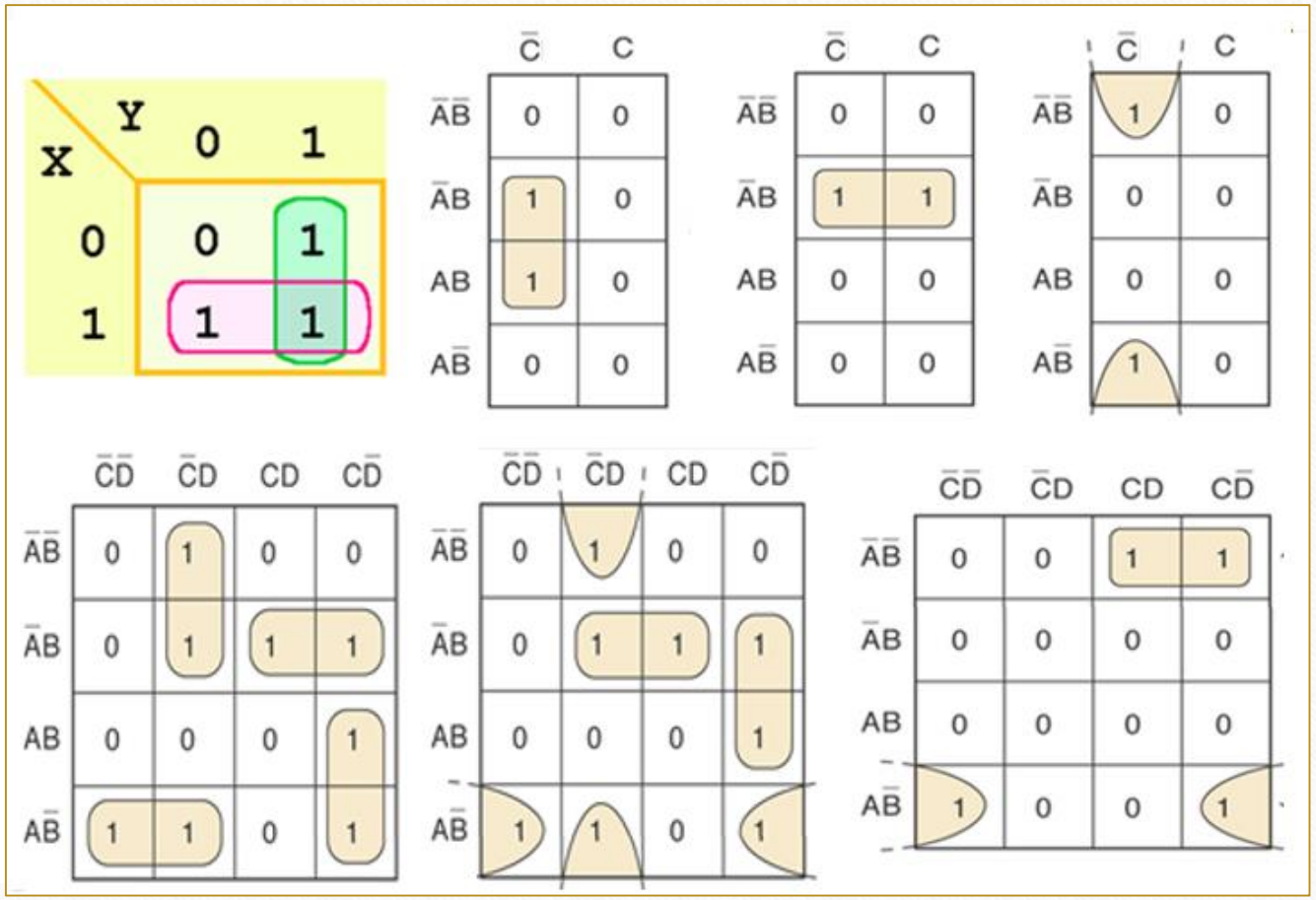




SOP Reduction – Grouping of 1's

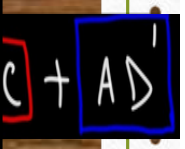
If 2 adjacent 1's are encircled,

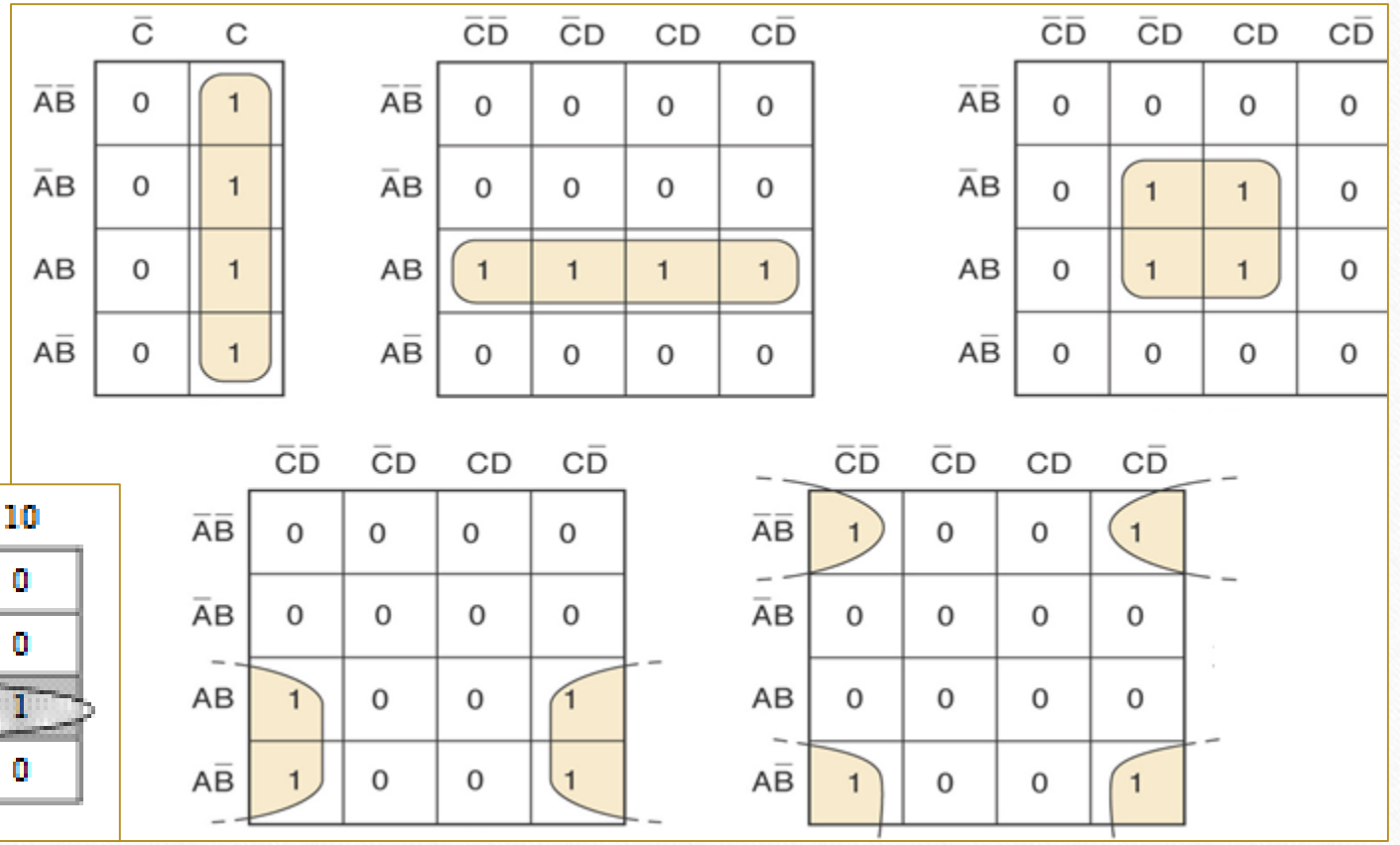
it makes a **pair**;

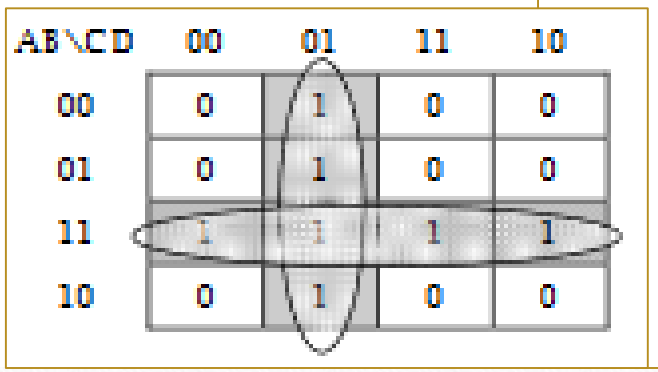


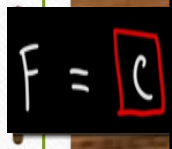
SOP Reduction – Grouping of 1's

If 4 adjacent 1's are encircled, it makes a **quad**;

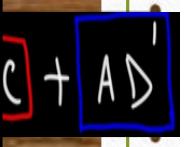






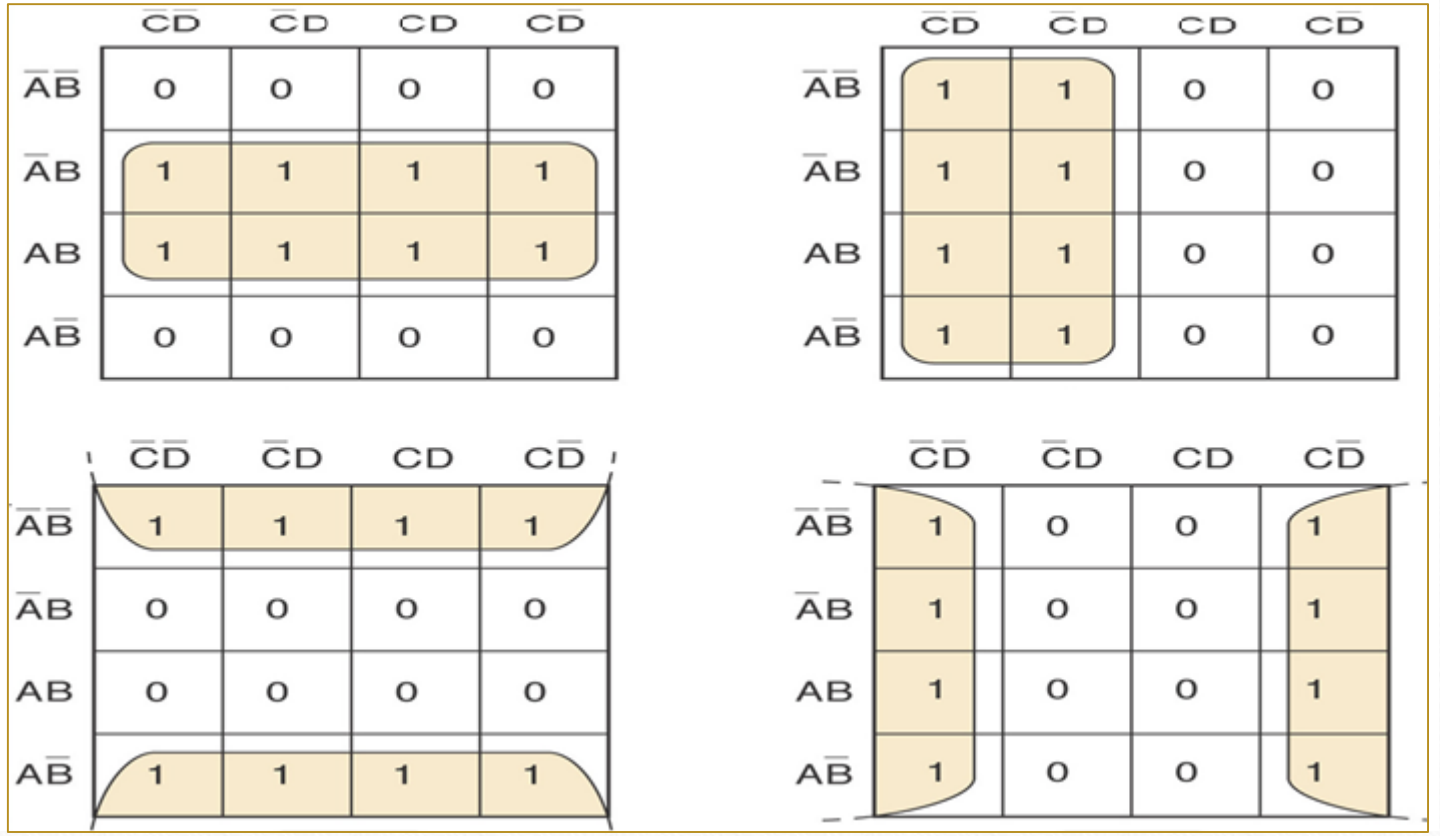


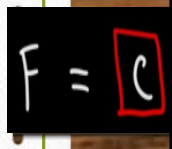
SOP Reduction – Grouping of 1's



If 8 adjacent 1's are encircled,

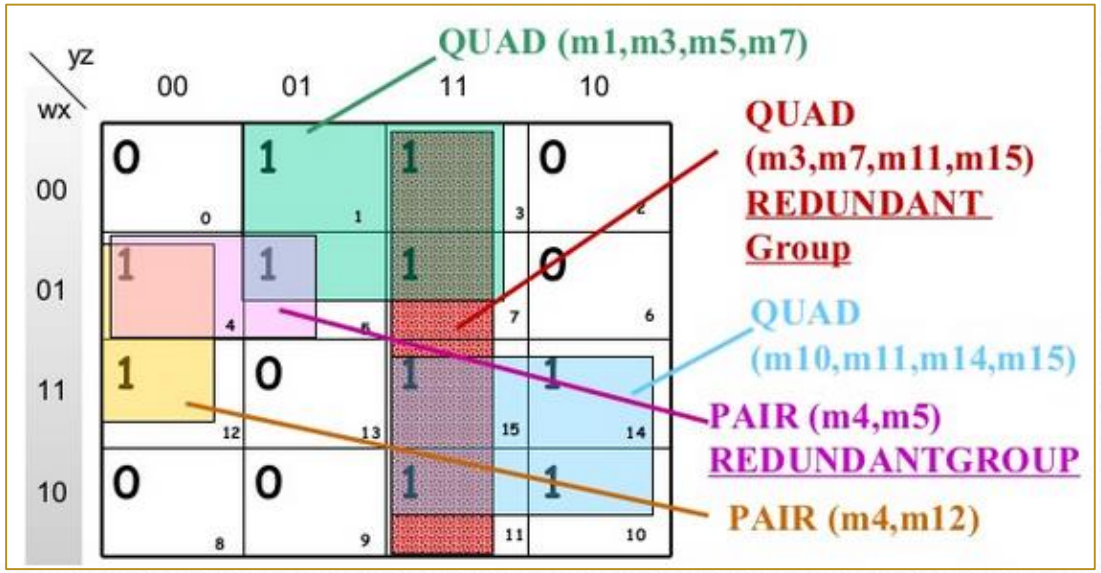
it makes an **Octet**;





SOP Reduction – Grouping of 1's

1. No zeros should be grouped.
2. No diagonals.
3. Only power of 2 number of cells in each group.
4. Groups should be as large as possible.
5. Less number of groups
6. Every **1** must be in at least one group.
7. Overlapping allowed.
8. Wrap around allowed.
9. Remove Redundant Groups

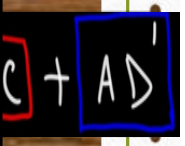


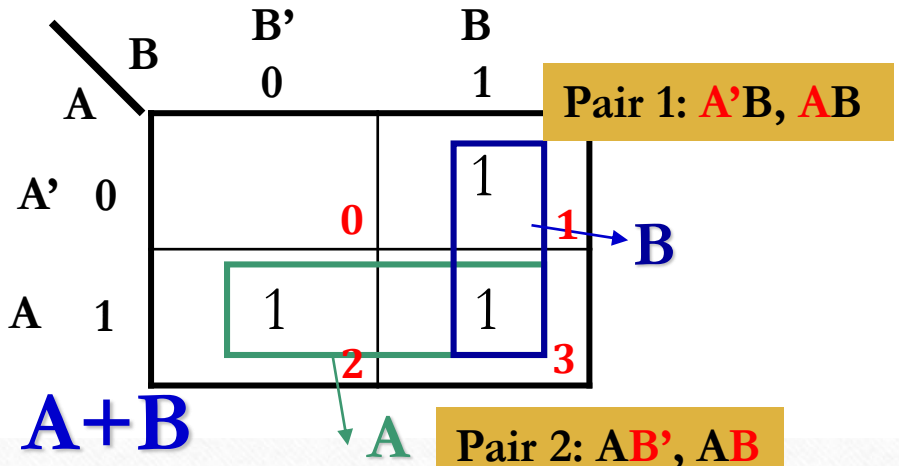
SOP Reduction – Grouping of 1's & Simplifying

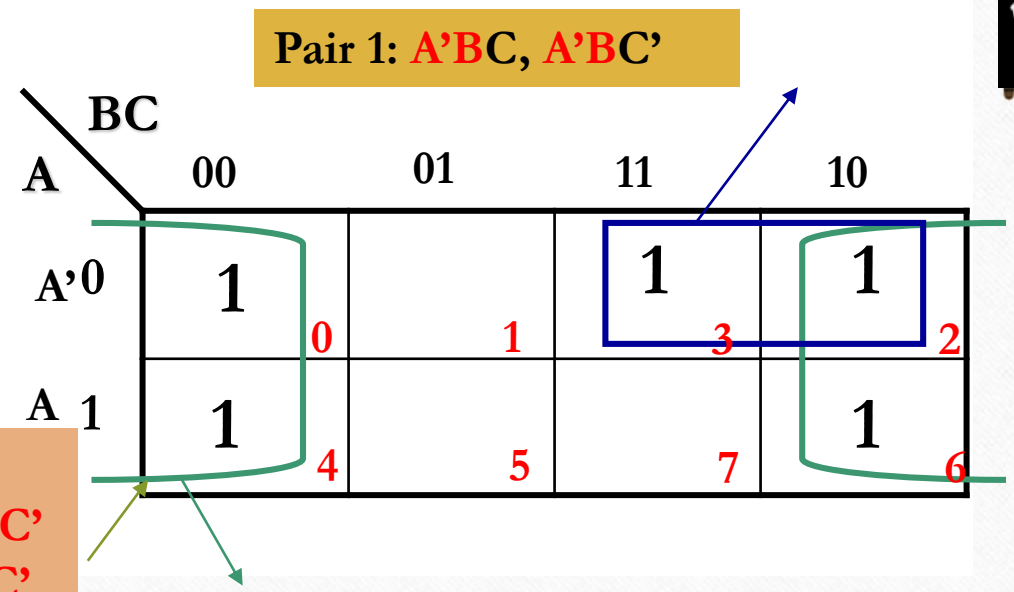
Remove the variable which changes its state in the group.

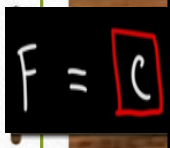
Based on Complementation Law, A + A' = 1

Pair removes one variable only. **Quad** removes 2 and **Octet** removes three variables. So, while grouping first search for octets, then for quads, and lastly go for pairs as the **bigger group removes more variables**.



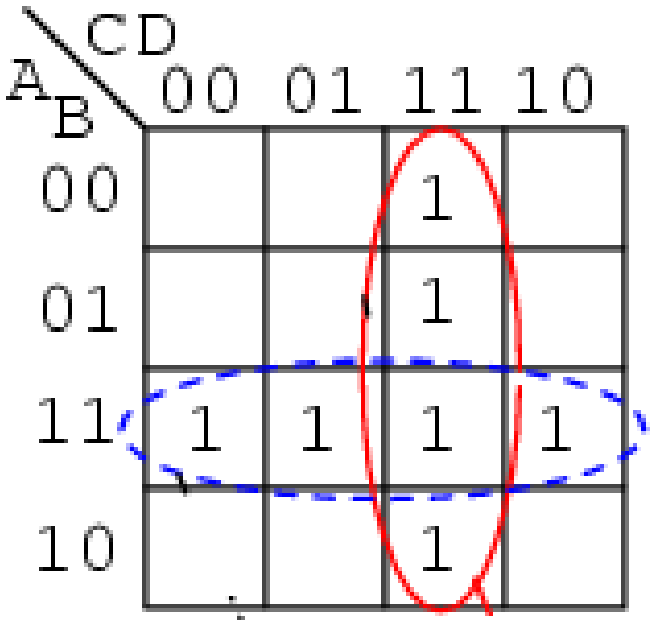






Simplification Using K-Maps - Exercises

$F = \overline{A}\,\overline{B}\,C\,D + \overline{A}\,B\,C\,D + A\,B\,C\,D + A\,\overline{B}\,C\,D + A\,B\,\overline{C}\,\overline{D} + A\,B\,\overline{C}\,D + A\,B\,C\,\overline{D}$



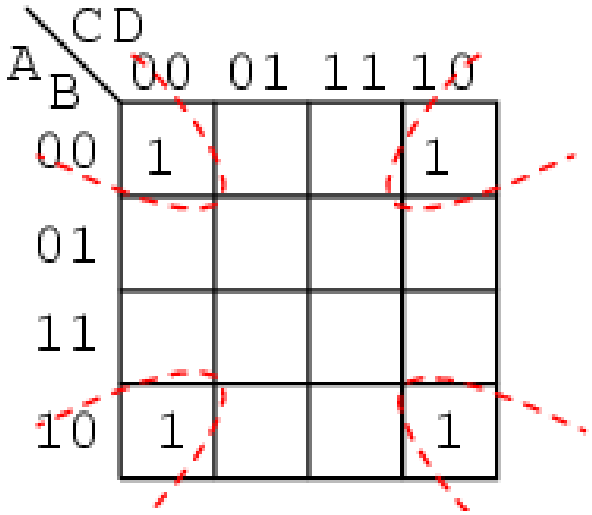
Quad 1: **ABC'D'**, **ABC'D**, **ABCD**, **ABCD'** → **AB**

Quad 2: A'B'CD, A'BCD, ABCD, AB'CD → **CD**

So, F = AB + CD

Simplification Using K-Maps - Exercises

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



Quad 1:

A'B'C'D'

A'B'CD'

AB'C'D'

AB'CD'

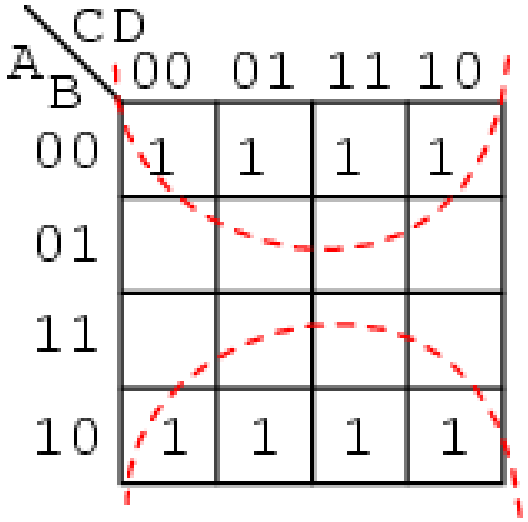
$F(A,B,C,D) = \bar{B}\bar{D}$

Exercise:

$F(A,B,C,D) = \Sigma(1,2,5,7,8,10,12,13,14)$

Simplification Using K-Maps - Exercises

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



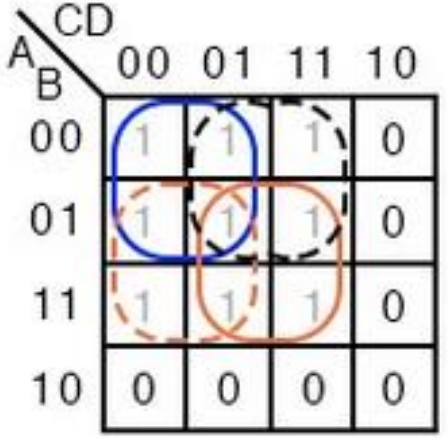
$F(A,B,C,D) = \bar{B}$

Octet

A'B'C'D', A'B'C'D, A'B'CD, A'B'CD'

AB'C'D', AB'C'D, AB'CD, AB'CD'

$f(A,B,C,D) = \Sigma m(0,1,3,4,5,7,12,13,15)$

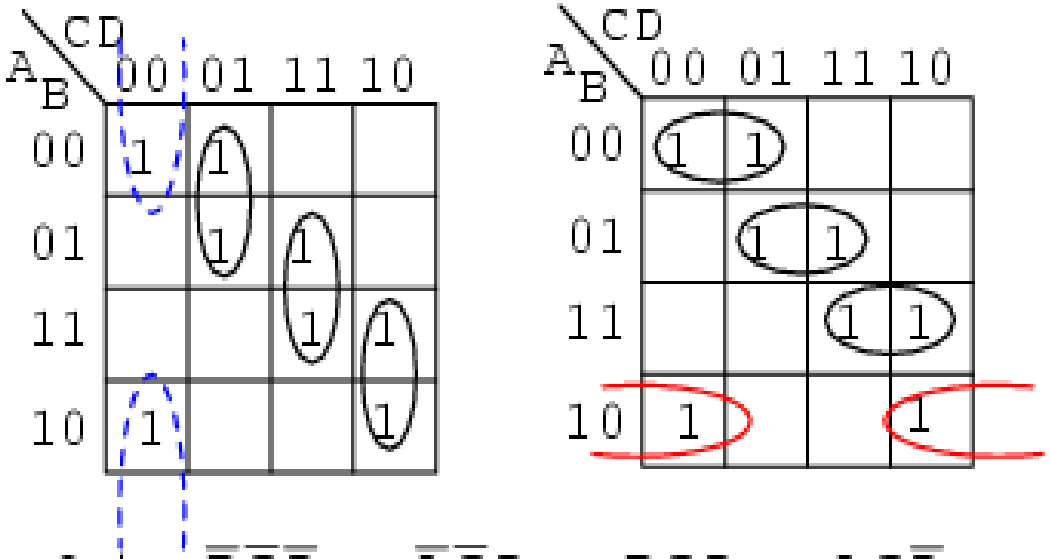


Quads 1: A'B'C'D', A'B'C'D, A'BC'D', A'BC'D
Quads 2: A'B'C'D, A'B'CD, A'BCD', A'BCD
Quads 3: A'BC'D', A'BC'D, ABC'D', ABC'D
Quads 4: A'BC'D, A'BCD, ABC'D, ABCD

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

Simplification Using K-Maps - Exercises

$$Out = \overline{A}\,\overline{B}\,\overline{C}\,\overline{D} + \overline{A}\,\overline{B}\,\overline{C}D + \overline{A}B\overline{C}D + \overline{A}BCD + ABCD + ABC\overline{D} + A\overline{B}\,\overline{C}\,\overline{D} + A\overline{B}C\overline{D}$$



$$Out = \overline{B}\,\overline{C}\,\overline{D} + \overline{A}\,\overline{C}D + BCD + AC\overline{D}$$

$$Out = \overline{A}\,\overline{B}\,\overline{C} + \overline{A}BD + ABC + A\overline{B}\,\overline{D}$$

Often times there is more than one minimum cost solution to a simplification problem.

Both results have four product terms of three Boolean variable each.

Both are equally **valid *minimal cost*** (minimum number of gates with the minimum number of inputs) **solutions.**

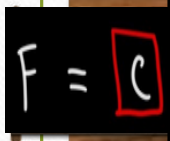
The difference in the final solution is due to how the cells are grouped.

POS Reduction

using k-Map

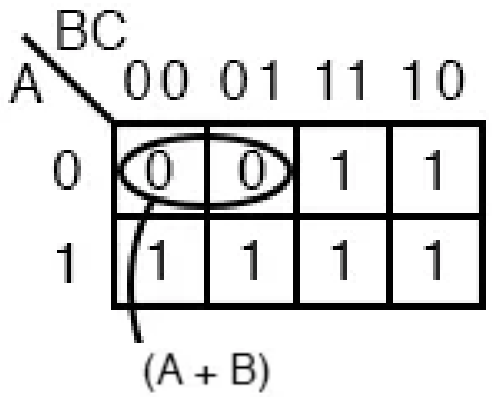
RULES FOR SIMPLIFYING THE POS EXPRESSION USING K-MAPS

- **Select** the respective K-map based on the number of variables present in the given expression or in the truth table.
- **Mapping** – Enter **0** at respective maxterm cells in the K-map for each maxterm present in the expression or for the maxterm with **LOW** outputs in the truth table.
- **Grouping** - Check for the possibilities of grouping maximum number of adjacent **Zeroes**. It should be powers of two. Do not forget to roll the map and overlap
- **Simplifying** - Each grouping will give either a literal or one sum term. The reduced expressions for all the groups are then ANDed.



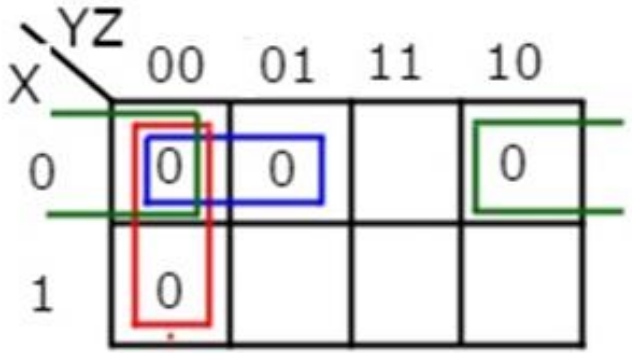
EXERCISES

$F = (A + B + C)(A + B + \overline{C})$



F = (A+B)

$f(X,Y,Z) = \prod M(0,1,2,4)$

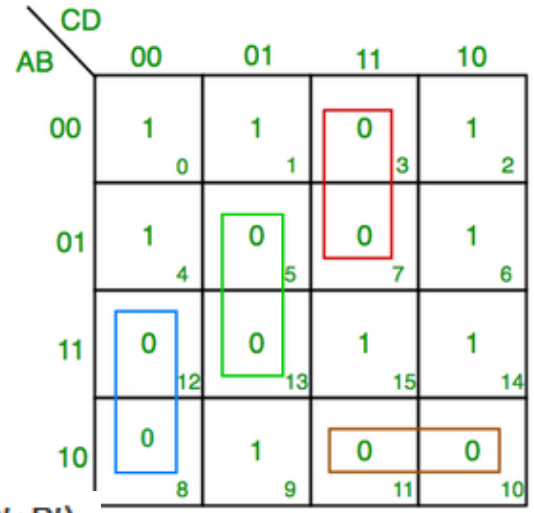


Pair 1: X+Y+Z, X+Y+Z' = X+Y

Pair 2: X+Y+Z, X'+Y+Z = Y+Z

Pair 3: X+Y+Z, X+Y'+Z = X+Z

f = (X+Y).(Y+Z).(X+Z)



green (C+D'+B')

red (C'+D'+A)

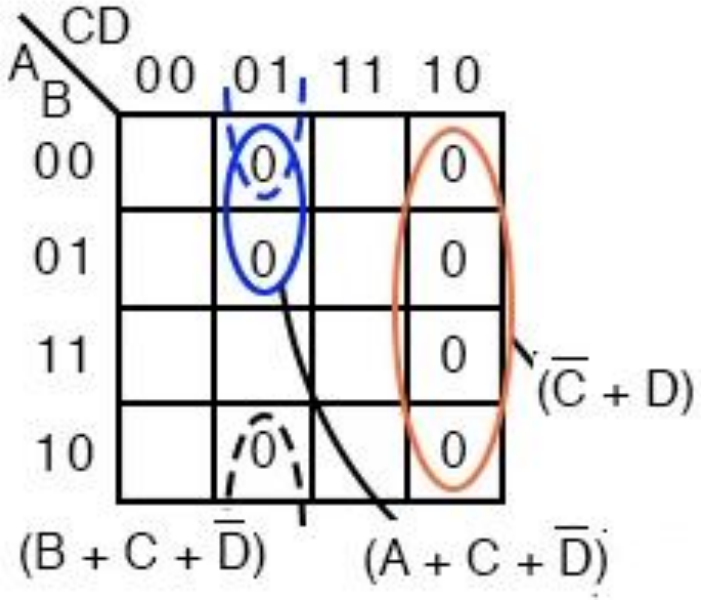
blue (A'+C+D)

brown (A'+B+C')

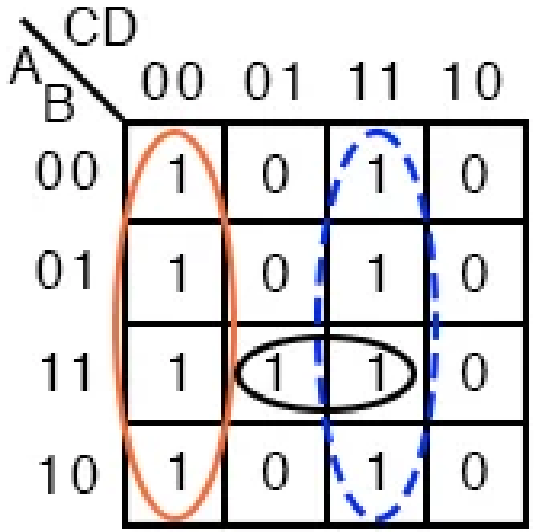
F = (C+D'+B').(C'+D'+A).(A'+C+D).(A'+B+C')

EXERCISES

$$F = (A+B+C+\overline{D})(A+B+\overline{C}+D)(A+\overline{B}+C+\overline{D})(A+\overline{B}+\overline{C}+D)$$
$$(\overline{A}+\overline{B}+\overline{C}+D)(\overline{A}+B+C+\overline{D})(\overline{A}+B+\overline{C}+D)$$



F = (B+C+D').(A+C+D').(C'+D)



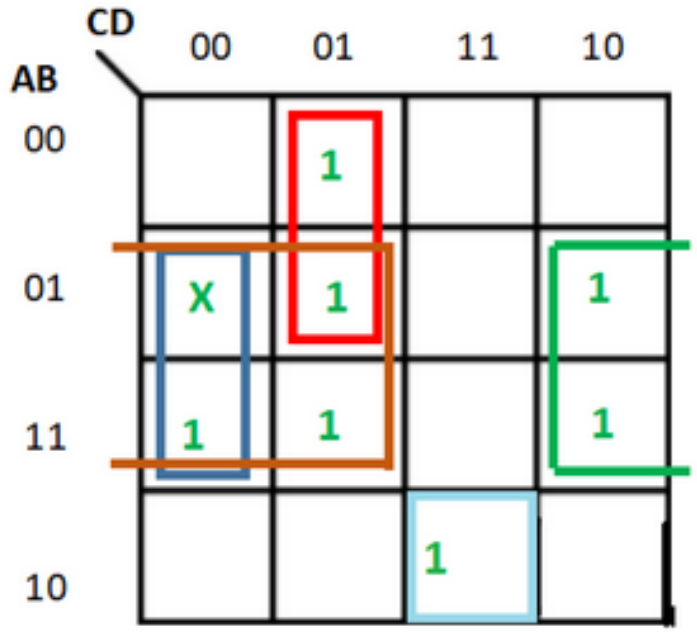
F = ABD+CD+C'D'

Don't Care Conditions

- Some times, not all values of a function are defined
  - Some input conditions will never occur (For Example, in BCD code, input states 1001, 1010, 1011, 1100, 1101, 1110 and 1111 are invalid)
  - We don't care what the output is for that input condition
- Don't Care cell can be represented by a cross(X) in K-Maps representing a invalid combination.
- While simplifying, the don't care minterms may be assumed to be either 0 or 1. when simplifying the function, we can choose to include each don't care minterm with either the 1's or the 0's, depending on which combination gives the simplest expression.

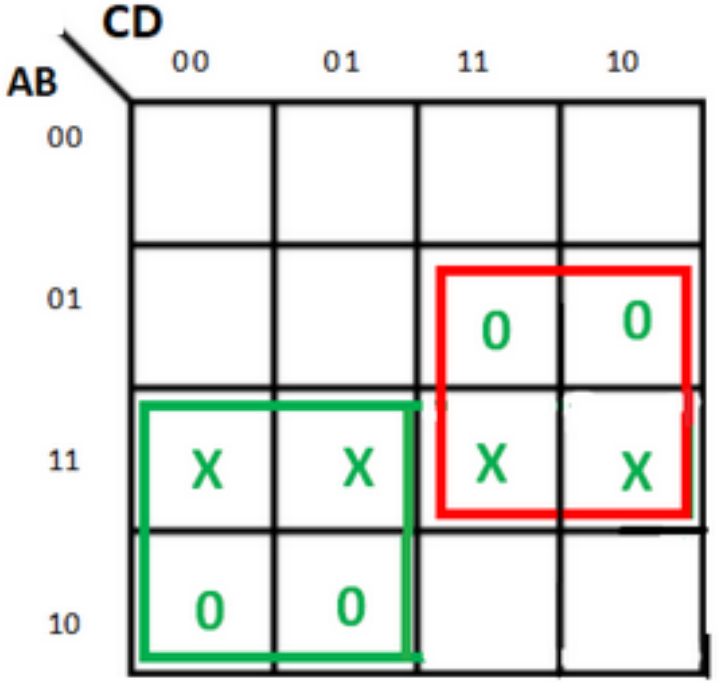
EXERCISES:-

f = m(1, 5, 6, 11, 12, 13, 14) + d(4)



f = BC' + BD' + A'C'D + AB'CD

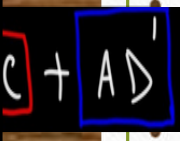
F(A, B, C, D) = M(6, 7, 8, 9) + d(12, 13, 14, 15)

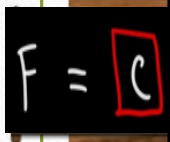


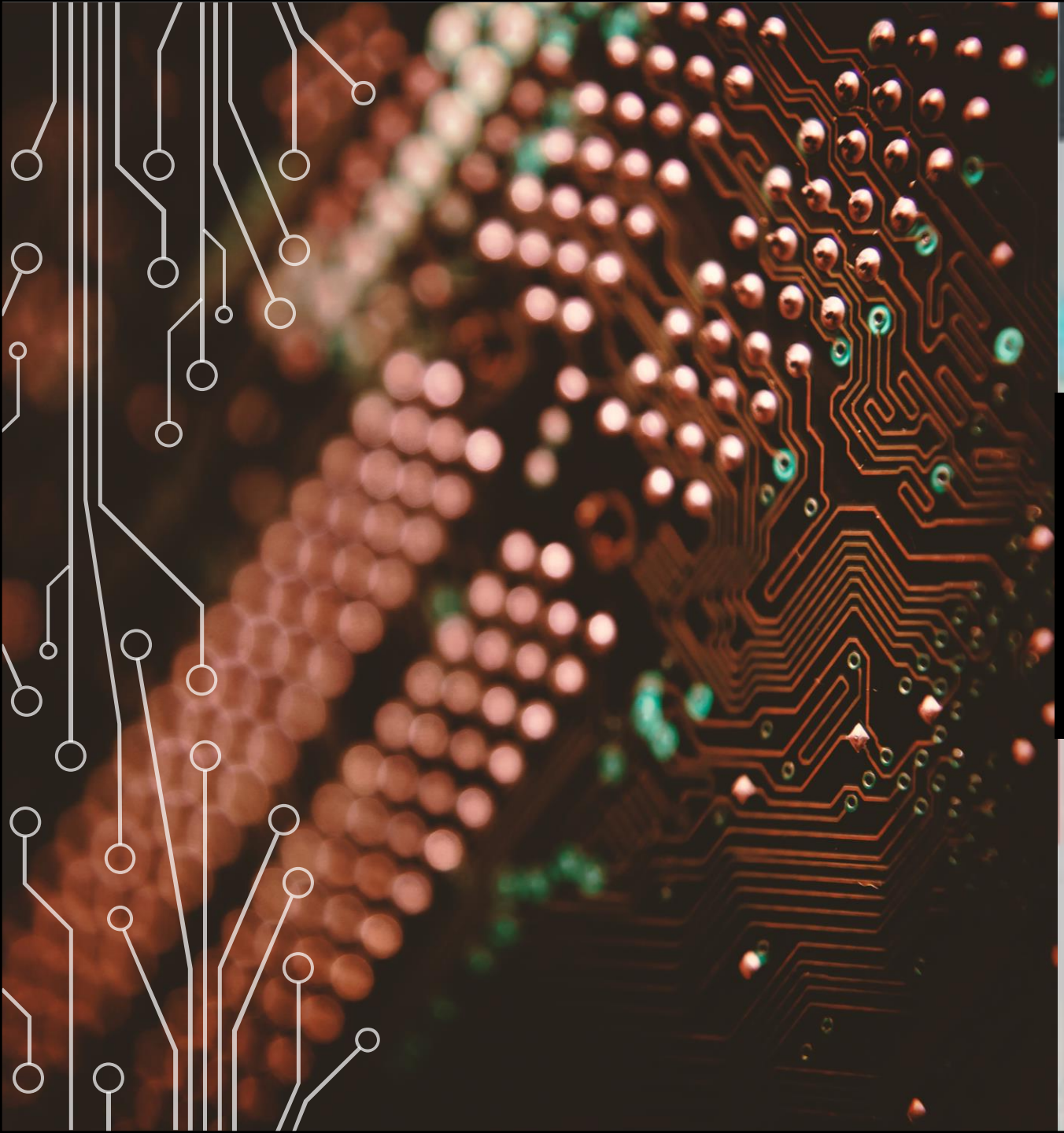
F = (A'+ C)(B' + C')

EXERCICES

1. Minimize $F(A,B,C,D) = \sum m(1,4,7,10,13) + \sum d(5,14,15)$
2. Minimize $F(A,B,C,D) = \sum m(4,5,7,12,14,15) + \sum d(3,8,10)$
3. Minimize $F(A,B,C,D) = \sum m(1,3,7,11,15) + \sum d(0,2,5)$



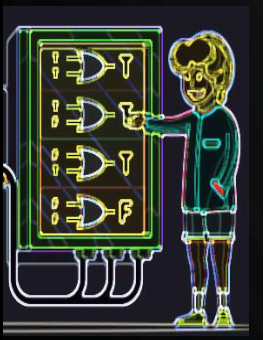




Realization Using Logic Gates

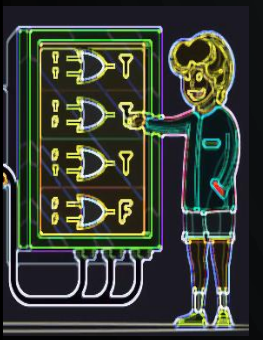
Deepa Mathews

Logic Circuits Using AND, OR & NOT Gates



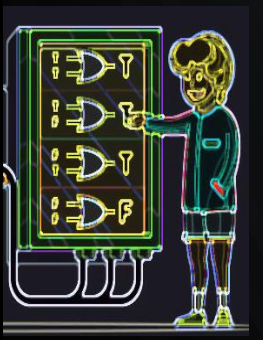
- Boolean functions in either SOP or POS forms can be implemented using 2-Level implementations.
- For SOP forms AND gates will be in the first level and a single OR gate will be in the second level.
- For POS forms OR gates will be in the first level and a single AND gate will be in the second level.

Designing AOI SOP logic Circuits

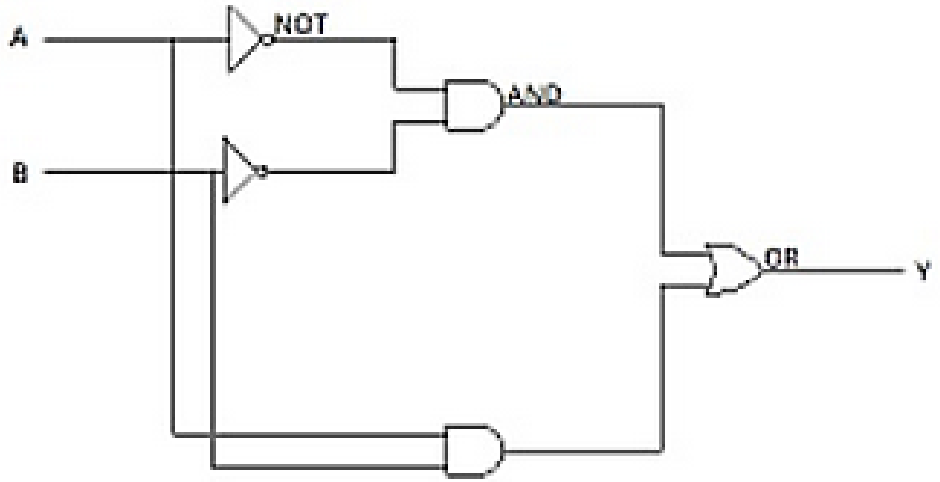


1. Implement each Minterm in the logic expression with an **AND** gate with the **same number** of inputs as there are variables in the Minterm. (i.e., AB = **2** input gate, ABC = **3** input gate, etc.)
2. **OR** together the outputs of the AND gates to produce the logic expression.
3. If necessary, gates can be **cascaded** to create gates with more inputs.

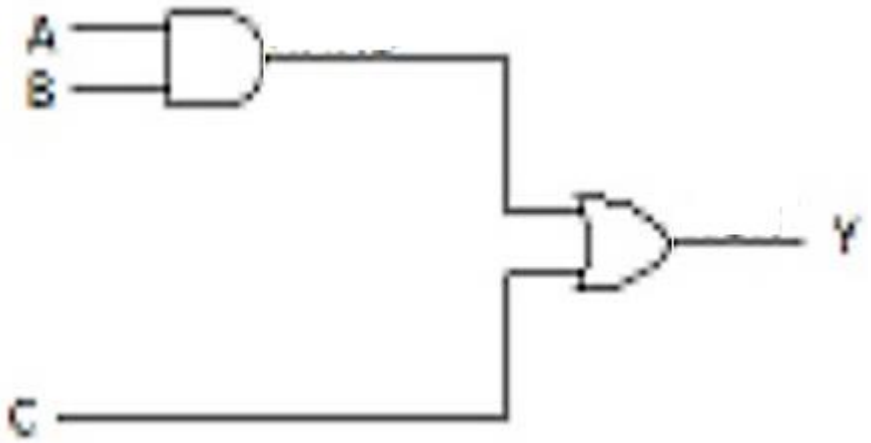
Implementation using AOI Logic



F(A,B) = AB+A'B'

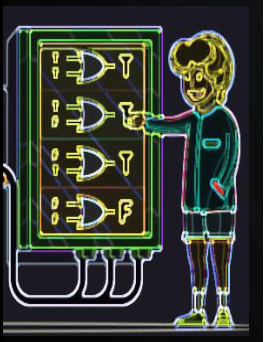


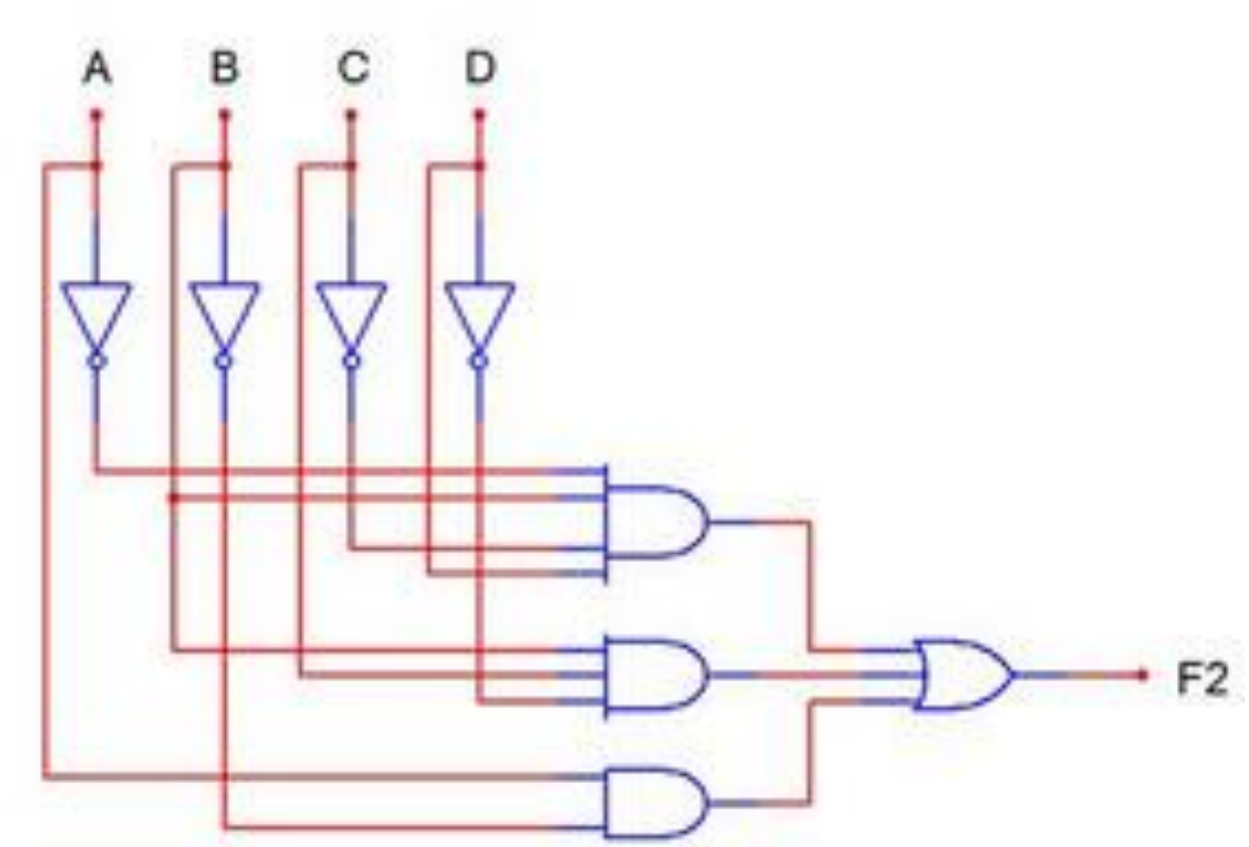
F(A,B,C) = (AB+C)



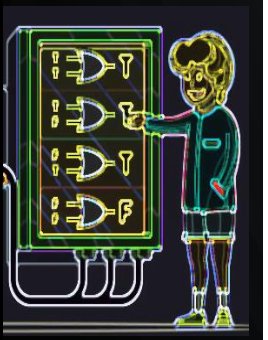
Implementation using AOI Logic

$$F_2 = \bar{A}B\bar{C}D + BC\bar{D} + A\bar{B}$$

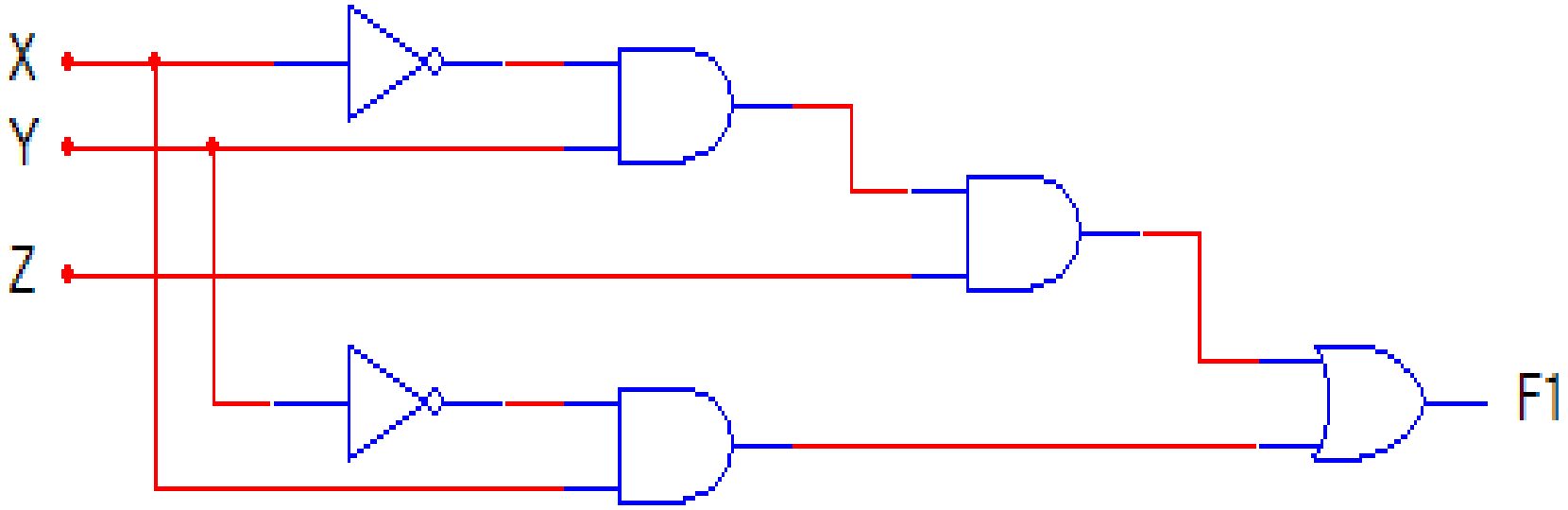




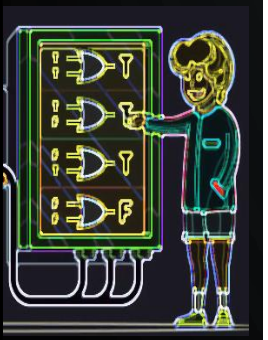
Implementation using AOI Logic



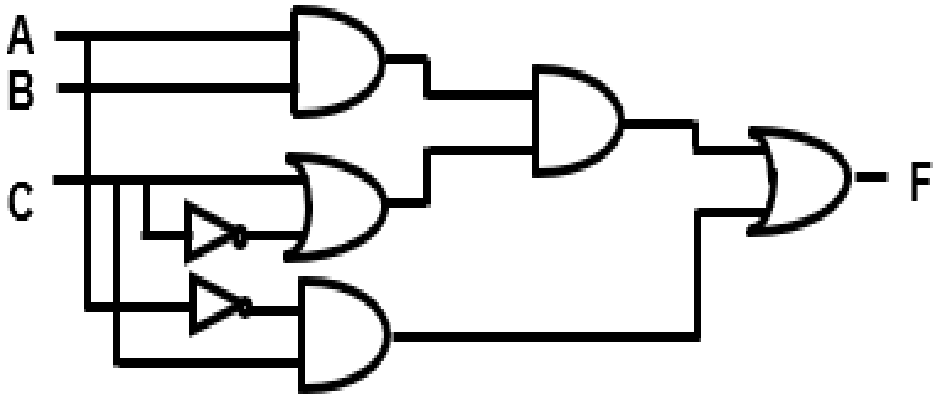
$$F_1 = \overline{X}\,Y\,Z + X\,\overline{Y}$$



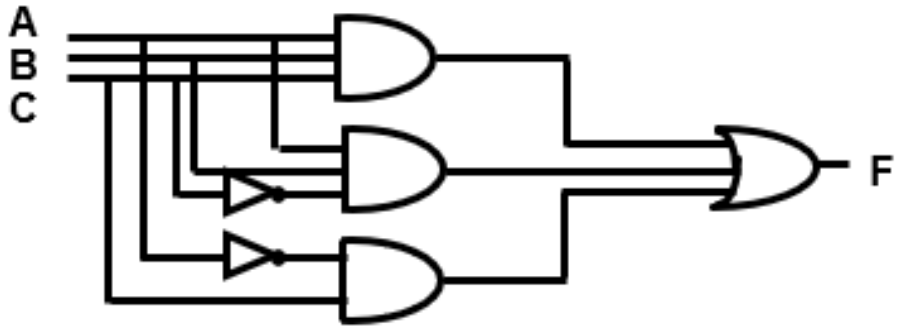
Implementation using AOI Logic



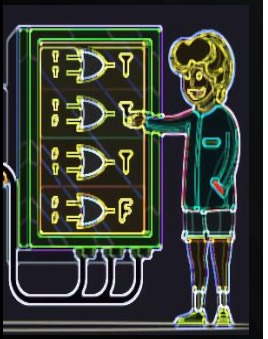
$F = AB(C + C') + A'C$



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



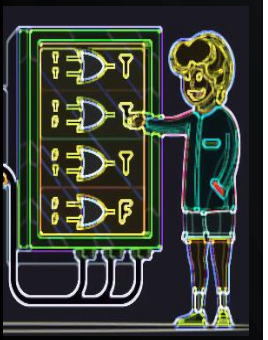
Designing OAI POS logic Circuits

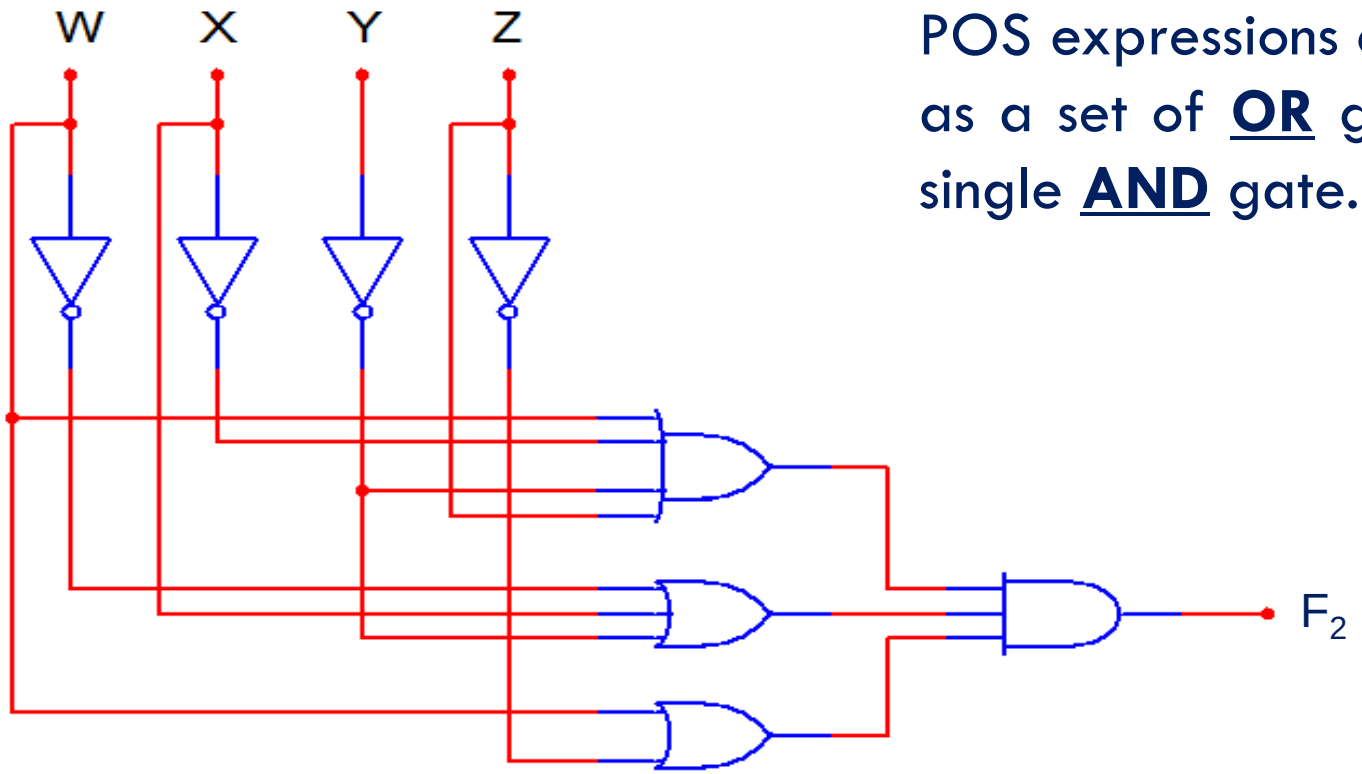


1. Implement each Maxterm in the logic expression with an **OR** gate
2. **AND** together the outputs of the **OR** gates to produce the logic expression.
3. If necessary, gates can be **cascaded** to create gates with more inputs.

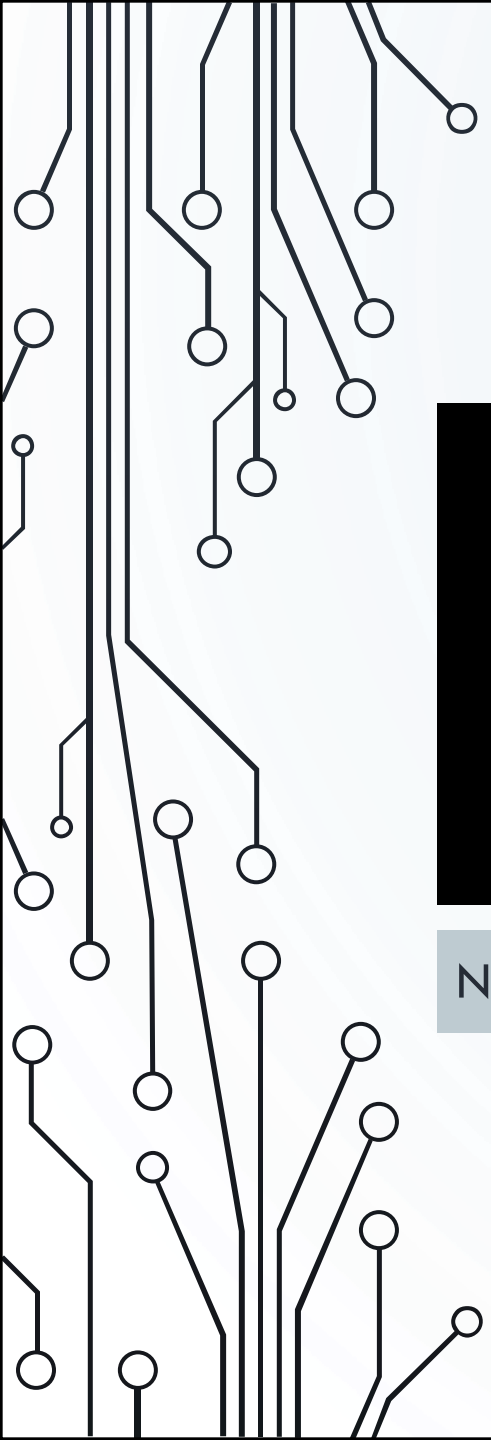
Implementation using OAI Logic

$$F_2 = (W + \overline{X} + \overline{Y} + Z)(\overline{W} + X + \overline{Y})(W + \overline{Z})$$





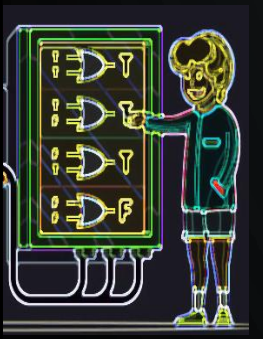
POS expressions can be implemented as a set of **OR** gates feeding into a single **AND** gate.



Circuit Realization using Universal Gates

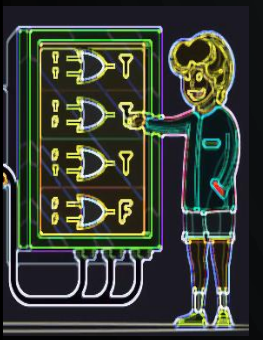
NAND, NOR

Universal Gates



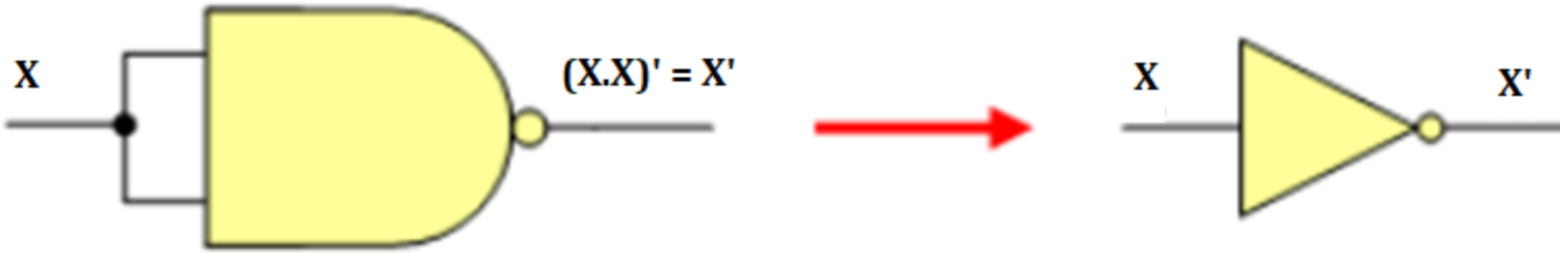
- A **universal gate** is a **gate** which can implement any Boolean function without need to use any other **gate** type.
- The NAND and NOR **gates** are **universal gates**. They are said to be universal gates because all the other gates can be created using these gates.
- In practice, these gates are advantageous since NAND and NOR **gates** are economical and easier to fabricate and are the basic **gates** used in all IC digital logic families.

NAND Gate is a Universal Gate?



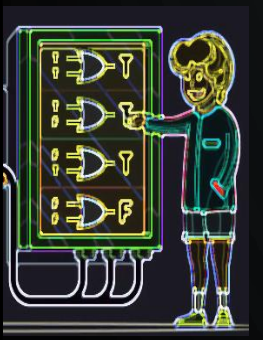
Implementing an Inverter

All NAND input pins connect to the input signal X gives an output X'.



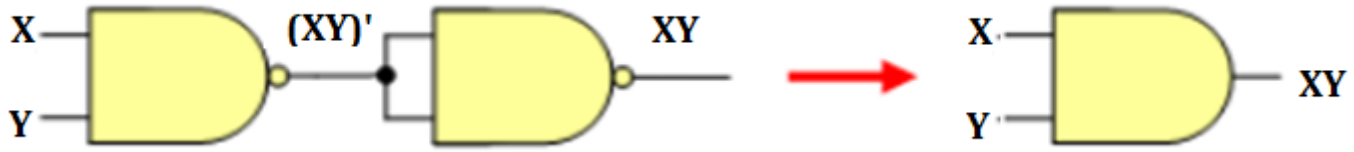
Input	Output	Rule
(X.X)'	= X'	Idempotent

NAND Gate is a Universal Gate?



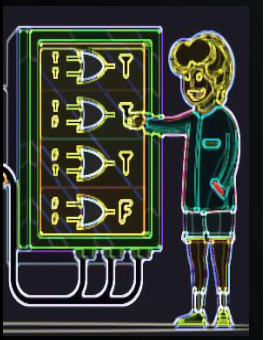
Implementing AND

AND is replaced by a NAND gate with its output complemented by a NAND gate inverter



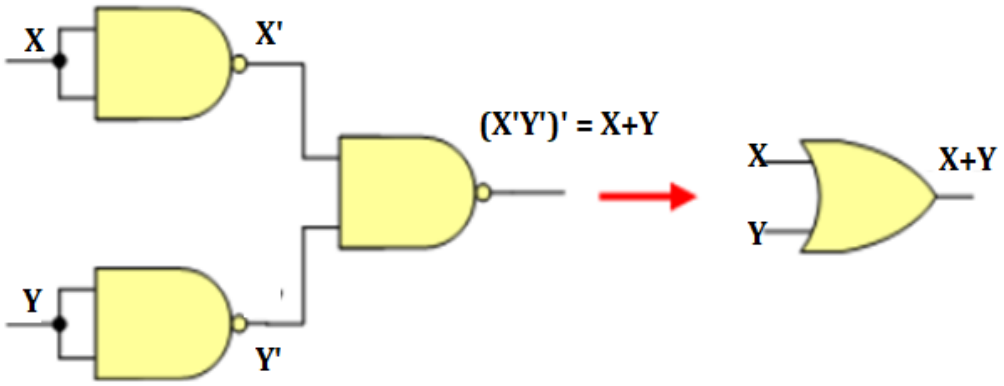
Input	Output	Rule
((XY)'(XY)')'	= ((XY)')'	Idempotent
	= (XY)	Involution

NAND Gate is a Universal Gate?



Implementing OR

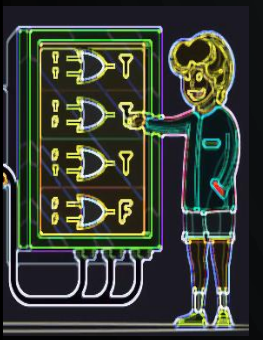
OR gate is replaced by a NAND gate with all its inputs complemented by NAND gate



Input	Output	Rule
((XX)'(YY)')'	= (X'Y')'	Idempotent
	= X"+Y"	DeMorgan
	= X+Y	Involution

Thus, the NAND gate is a universal gate since it can be used to implement the AND, OR and NOT functions.

NOR Gate is a Universal Gate?

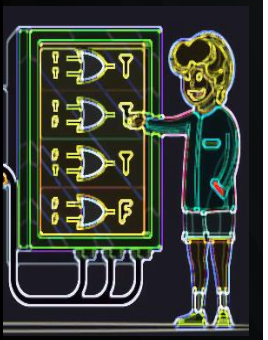


Implementing an Inverter

All NOR input pins connect to the input signal X gives an output X'.

Input	Output	Rule
(X+X)'	= X'	Idempotent

NOR Gate is a Universal Gate?

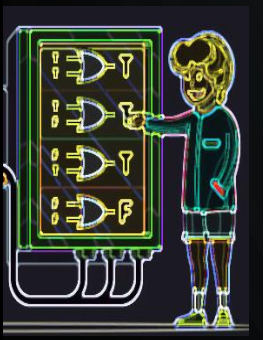


Implementing OR

OR gate is replaced by a NOR gate with all its inputs complemented by NOR gate inverters

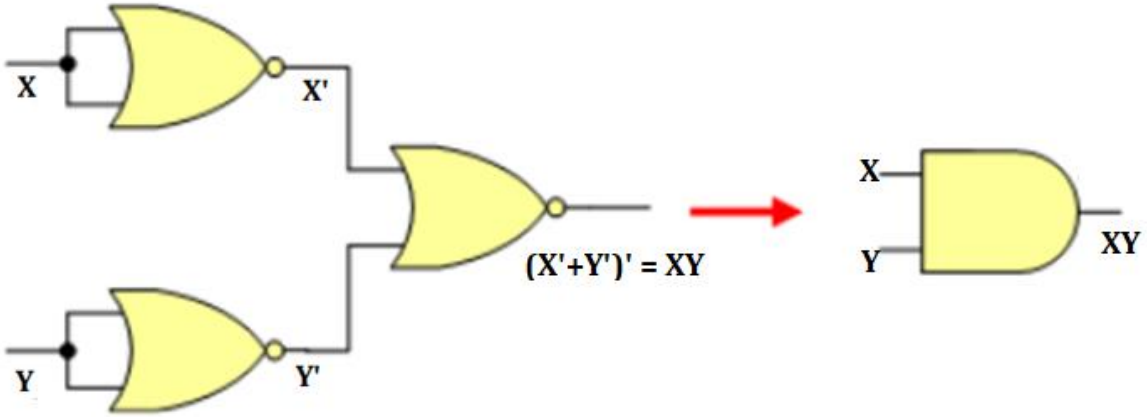
Input	Output	Rule
((X+Y)'+(X+Y)')'	= ((X+Y)')'	Idempotent
	= X+Y	Involution

NOR Gate is a Universal Gate?



Implementing AND

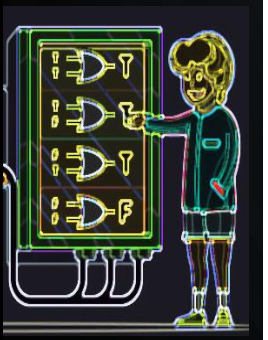
The AND gate is replaced by a NOR gate with all its inputs complemented by NOR gate inverters



Input	Output	Rule
((X+X)'+(Y+Y)')'	=(X'+Y')'	Idempotent
	= X''.Y''	DeMorgan
	= (X.Y)	Involution

Thus, the NOR gate is a universal gate since it can be used to implement the AND, OR and NOT functions.

Circuit Realization Using Universal Gates



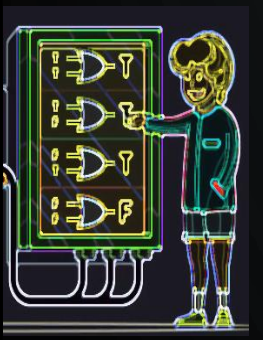
1. Draw the Circuit for the given function using AND, OR & NOT gates.
2. Select the type of Universal gate (NAND/NOR)
3. Draw a bubble at output of each AND gate & bubble at each input of OR gate, if **NAND**.

   Draw a bubble at output of each OR gate & bubble at each input of AND gate, if **NOR**.
4. For each bubble that is put in step 3, **put one more bubble** on the same line.
5. Input bubbled OR is replaced by Output bubbled AND gate, if NAND (De Morgan's Law)

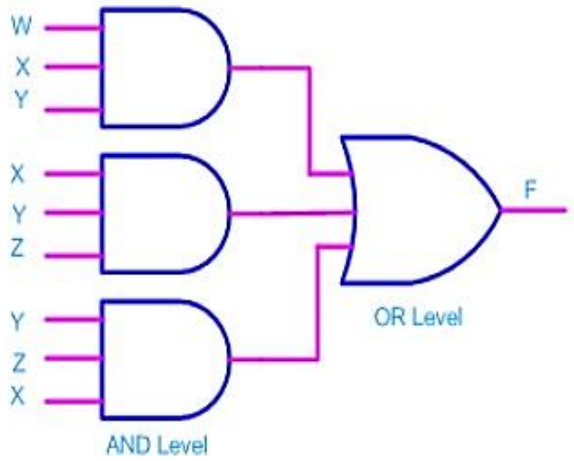
   Input bubbled AND is replaced by Output bubbled OR gate, if NOR (De Morgan's Law)
6. Cancel double bubbles in a line if there **(Apply Double Inversion Law)**
7. Replace NOT gates with NAND/NOR

Realization Using Universal Gates – Exercises

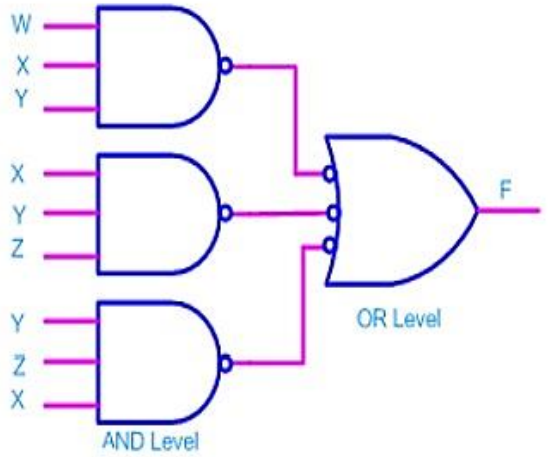
Q1. Implement F = W.X.Y + X.Y.Z + Y.Z.W using NAND gates



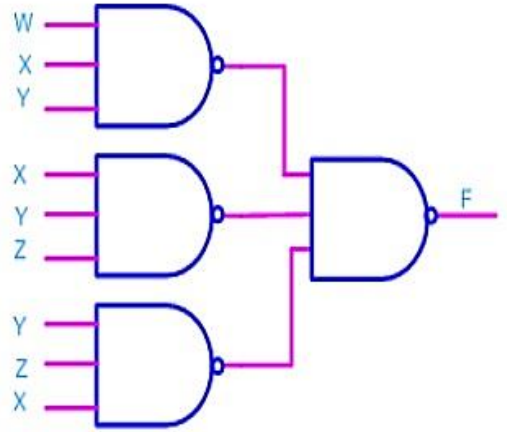
1. **Implemented with three AND gates in first stage and one OR gate**



2. **Put bubbles at the output line of AND gate and at the input lines of the OR gate.**

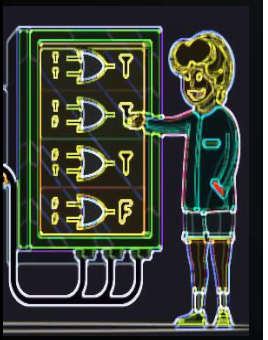


3. **Replace OR gate with input bubble with the NAND gate.**

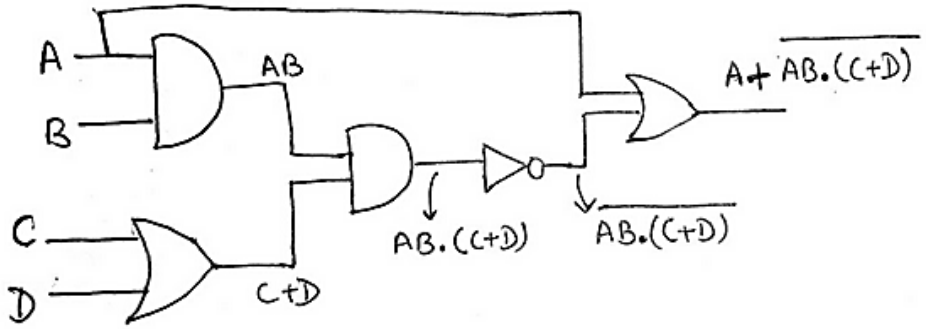


Realization Using Universal Gates – Exercises

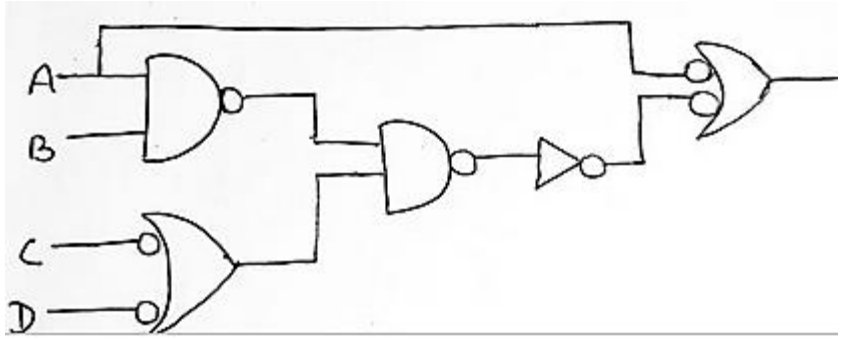
Q2. Implement Y = A + (AB.(C + D))' using NAND gates



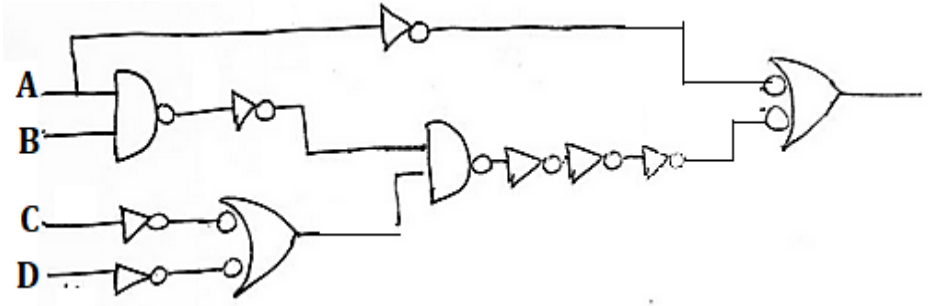
1. Implement the function using AOI



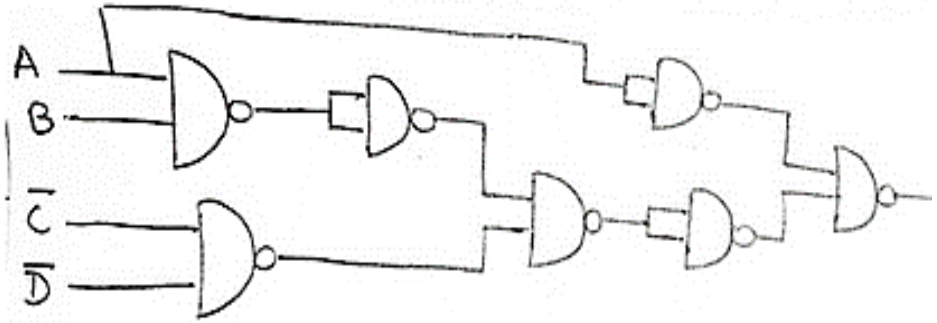
2. Replace AND with NAND, OR with input bubbled OR



3. For each bubble that is put in step 2, put one more bubble (inverter) on same line.



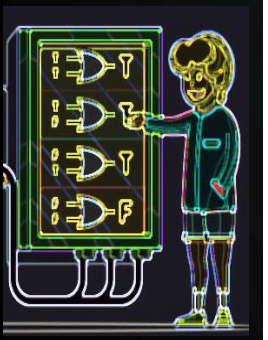
4. Replace input bubbled OR gate with NAND gate, inverters with NAND

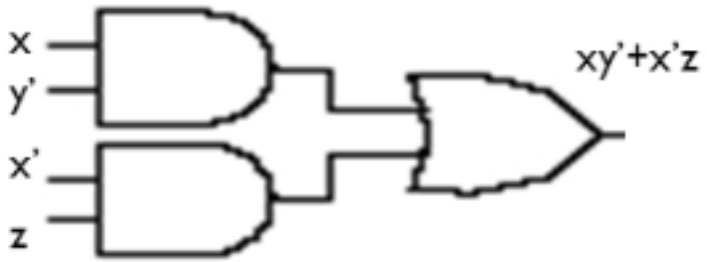


Realization Using Universal Gates – Exercises

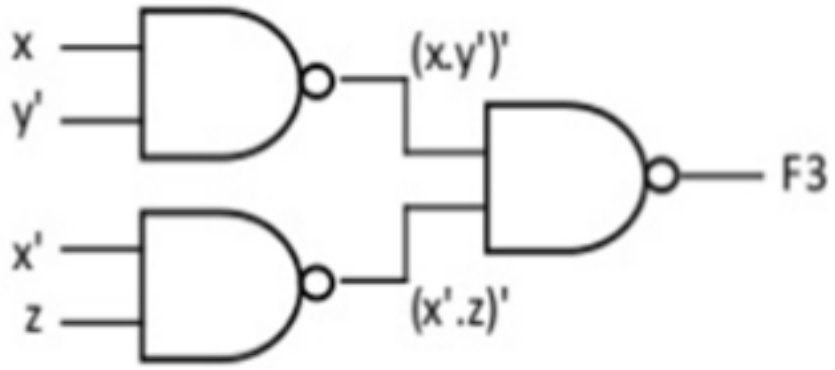
Q3. Implement F using mínimum number of NAND gates

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





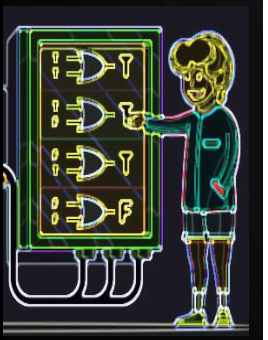
AND-OR CIRCUIT



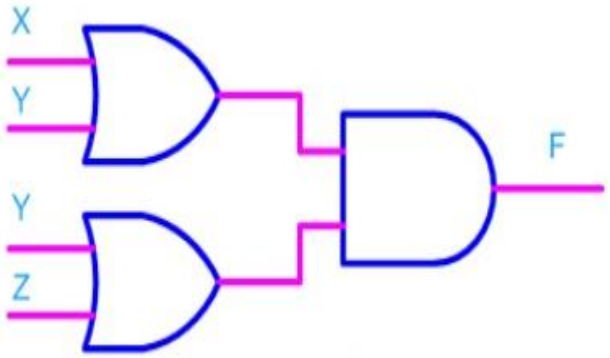
NAND-NAND

Realization Using Universal Gates – Exercises

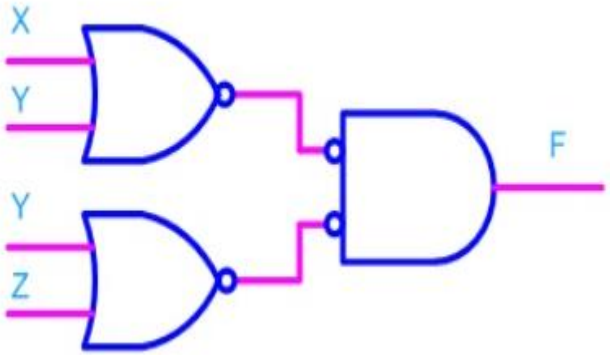
Q4. Implement F = (X+Y) . (Y+Z) using NOR gates



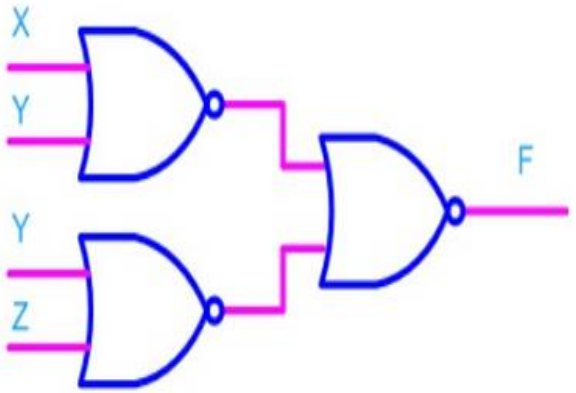
1. Implemented with two OR gates and one AND gate



2. Put bubbles at the output line of OR gate and at the input lines of AND gate.

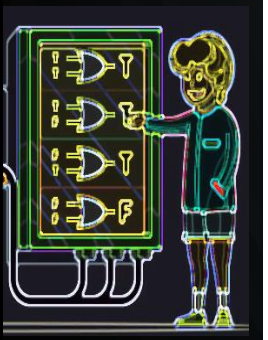


3. For each bubble put, draw 1 more bubble on same line; apply Double inversion; Replace input bubbled AND with NOR.



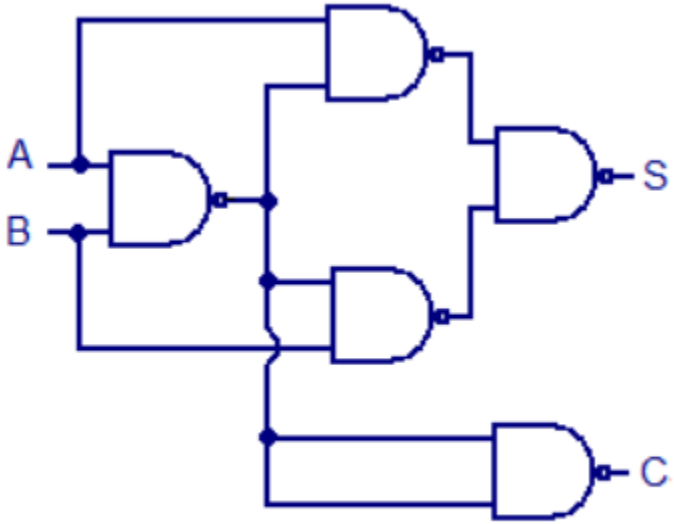
Realization Using Universal Gates – Exercises

Q5. Implement Half Adder using mínimum number of NAND & NOR gates

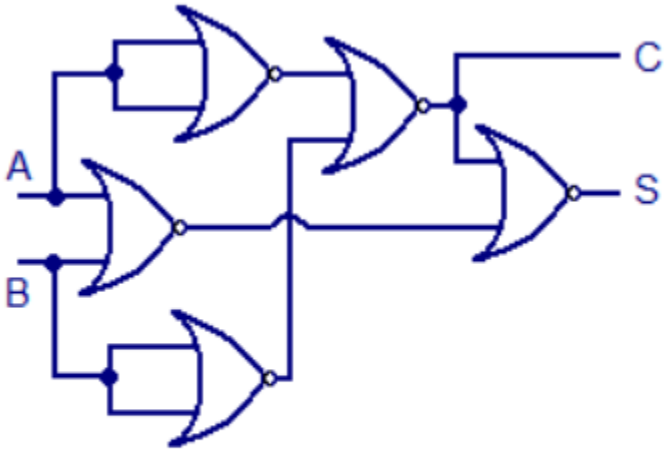


$$S = A'B + AB' = A \oplus B$$

$$C = A \cdot B$$



Half adder using NAND logic



Half adder using NOR logic

REALIZATION USING LOGIC GATES

Deepa Mathews