GRADAMIC

B.Tech–2nd Year Session 2023–24 Odd/Even Semester

Dr. A. P. J. Abdul Kalam Technical University

Lucknow, Uttar Pradesh

DIGITAL SYSTEM DESIGN / DIGITAL ELECTRONICS

(BOE 310 / BOE 310H / BOE 410 / BOE 410H)

app.gradamic.com

UNIT – 1st

UNIT-I

LOGIC SIMPLIFICATION AND COMBINATIONAL LOGIC DESIGN

- Number System
- Binary Arithmetic
- Signed Magnitude Representation
- Binary Codes
- Code Conversion
- Review Of Boolean Algebra and De-morgan's Theorem
- SOP & POS Forms
- Canonical Form
- Karnaugh Maps up to 5 Variables.

DIGITAL SYSTEMS

Any system which takes digital inputs and process the given data and gives the output are digital systems. Signals represented in the form of 1's and 0's are called digital signals.

These signals are also used to transfer information in encoded form, to prevent any casualities of mishappening with the information.

ADVANTAGES OF DIGITAL SYSTEMS

- Digital systems are more accurate and reliable than analog signals.
- Digital systems can be easily stored, processed, and transmitted.
- As they are easy to represent with 1's and 0's.
- Digital electronics are more precise and can perform more complex operations than analog electronics.
- Digital electronics are more efficient as they can perform same operations in less amount of power as compare to analog electronics.

APPLICATIONS OF DIGITAL ELECTRONICS

- Communication
- Business Transaction
- Traffic control
- Spacecraft Guidance
- Medical Treatment
- Military
- Generality
- Ability to represent & manipulate discrete elements of information

NUMBER SYSTEMS

There are 7 types of number systems:

1) **Decimal Number Systems**: The number having Base 10 are known as decimal number systems.

   Symbols that are used in this are 0 to 9.

2) **Binary Number System**: The number having Base 2 are known as decimal number systems.

   Symbols that are used in this are 0 and 1.

3) **Octal Number system**: The number having Base 8 are known as decimal number systems.

   Symbols that are used in this are 0 to 7.

4) **Hexadecimal Number System**: The number having Base 16 are known as decimal number systems.

   Symbols that are used in this are 0 to 9 and A to F & they are - 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F.

5) **Excess-3 Code:** Excess-3 is a binary coded decimal (BCD) code with unquestionable significance, seen for its work in enhancing number suffling task in early enlisting structures and smaller-than-expected PCs.

DECIMAL DIGIT	BCD CODES	EXCESS-3 CODE
0	0000	0011
1	0001	0100
2	0010	0101
3	0011	0110
4	0100	0111
5	0101	1000
6	0110	1001
7	0111	1010
8	1000	1011
9	1001	1100

6) **Gray Code:** Gray code is a form of binary that uses a different method of incrementing from one number to the next. With Gray Code, only one bit changes state from one position to another. This feature allows a system designer to perform some error checking (i.e., if more than 1 bit changes, the data must be incorrect).

TABLE 44.8 Gray Code

Decimal	Binary Code	Gray Code	Decimal	Binary Code	Gray Code
0	0000	0000	8	1000	1100
1	0001	0001	9	1001	1101
2	0010	0011	10	1010	1111
3	0011	0010	11	1011	1110
4	0100	0110	12	1100	1010
5	0101	0111	13	1101	1011
6	0110	0101	14	1110	1001
7	0111	0100	15	1111	1000

SIGNED MAGNITUDE REPRESENTATIONS

Signed magnitude is a convention in which we express a decimal number as a positive or negative binary number using its most significant bit.

The most significant bit is the left-most bit, which represents the sign of a binary digit. If it's ZERO (0) then it's positive, and if it's ONE (1) then it represents a negative number.

FOR EXAMPLE:

For a 4 bit representation of decimal number 4 we use 0100 (4-bits) but in signed magnitude representation we use 5 bits :-

1) If it is positive then we use the Most significant bit as a sign bit i.e. "00100". Here the left most bit (0) is known as the Sign bit.
2) If it is negative then we use the Most significant bit as a sign bit i.e. "10100". Here the left most bit (1) is known as the Sign bit.

BINARY TO DECIMAL CONVERSION

For converting any number from binary to decimal we have to multiply the digits of that binary number with the position of that number in the power of 2.

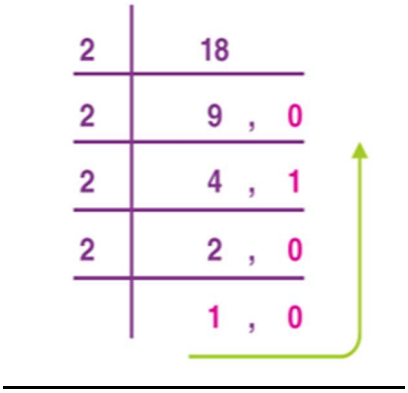
decimal = $d_0 \times 2^0 + d_1 \times 2^1 + d_2 \times 2^2 + \ldots$

FOR EXAMPLE:

1) $111001_2 = 1 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 57_{10}$
2) $1101_2 = 1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = 13_{10}$

More questions are in Practice Set.

DECIMAL TO BINARY CONVERSION



1) **Convert 13_{10} to binary**:

Division by 2	Quotient	Remainder	Bit #
13/2	6	1	0
6/2	3	0	1
3/2	1	1	2
1/2	0	1	3

So $(13)_{10} = (1101)_2$

2) **Convert 174_{10} to binary**:

Division by 2	Quotient	Remainder	Bit #
174/2	87	0	0
87/2	43	1	1
43/2	21	1	2
21/2	10	1	3
10/2	5	0	4
5/2	2	1	5
2/2	1	0	6
1/2	0	1	7

So $(174)_{10} = (10101110)_2$

DECIMAL TO OCTAL CONVERSION

1) **Convert 7562_{10} to octal:**

Division by 8	Quotient (integer)	Remainder (decimal)	Remainder (octal)	Digit #
7562/8	945	2	2	0
945/8	118	1	1	1
118/8	14	6	6	2
14/8	1	6	6	3
1/8	0	1	1	4

So $(7562)_{10} = (16612)_8$

2) **Convert 35631_{10} to octal:**

Division by 8	Quotient	Remainder (decimal)	Remainder (octal)	Digit #
35631/8	4453	7	7	0
4453/8	556	5	5	1
556/8	69	4	4	2
69/8	8	5	5	3
8/8	1	0	0	4
1/8	0	1	1	5

So $(35631)_{10} = (105457)_8$

BOOLEAN ALGEBRA & DE MORGAN'S THEOREM

It is a very powerful tool used in digital design. This theorem explains that the complements of the products of all the terms are equal to the sums of the complements of each and every term.

De Morgan's Theorem

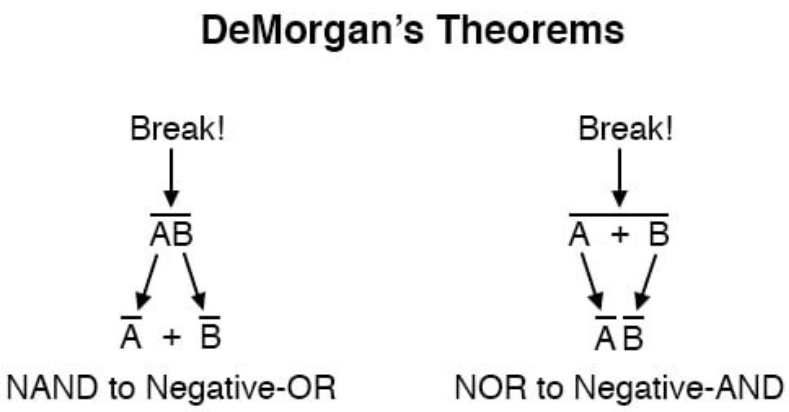
$$\neg(A \wedge B) = \neg A \vee \neg B$$

$$\neg(A \vee B) = \neg A \wedge \neg B$$

DeMorgan's Theorem

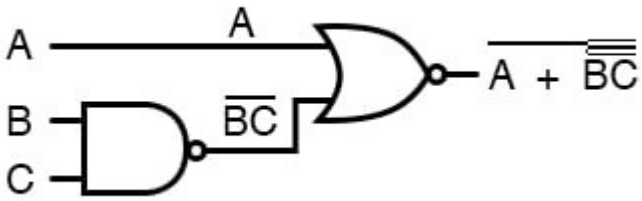
DeMorgan's theorem may be thought of in terms of *breaking* a long bar symbol.

When a long bar is broken, the operation directly underneath the break changes from addition to multiplication, or vice versa, and the broken bar pieces remain over the individual variables. To illustrate:



When multiple "layers" of bars exist in an expression, you may only break *one bar at a time*, and it is generally easier to begin simplification by breaking the longest (uppermost) bar first.

To illustrate, let's take the expression (A + (BC)')' and reduce it using DeMorgan's Theorems:



SOP vs POS in Digital Logic: Difference between SOP and POS in Digital Logic :-

The major difference between SOP and POS is that the SOP represents a Boolean expression through minterms, while POS defines a Boolean expression through max terms.

What is SOP?

SOP stands for Sum of Product. SOP form is a set of product(AND) terms that are summed(OR) together. When an expression or term is represented in a sum of binary terms known as minterms and sum of products.

What is POS?

POS stands for Product of Sum. A technique of explaining a Boolean expression through a set of max terms or sum terms, is known as POS(product of sum).

Difference between SOP & POS in Digital Logic

S.No.	SOP	POS
1	SOP stands for Sum of Products.	POS stands for Product of Sums.
2	It is a technique of defining the boolean terms as the sum of product terms.	It is a technique of defining boolean terms as a product of sum terms.
3	It prefers minterms.	It prefers maxterms.
4	In the case of SOP, the minterms are defined as ‘m’.	In the case of POS, the Maxterms are defined as ‘M’
5	It gives HIGH(1) output.	It gives LOW(0) output.
6	In SOP, we can get the final term by adding the product terms.	In POS, we can get the final term by multiplying the sum terms.

CANONICAL FORM

We will get four Boolean product terms by combining two variables x and y with logical AND operation.

These Boolean product terms are called as **min terms** or **standard product terms**.

The min terms are x’y’, x’y, xy’ and xy.

Similarly, we will get four Boolean sum terms by combining two variables x and y with logical OR operation.

These Boolean sum terms are called as **Max terms** or **standard sum terms**.

The Max terms are x + y, x + y’, x’ + y and x’ + y’.

The following table shows the representation of min terms and MAX terms for 2 variables.

x	y	Min terms	Max terms
0	0	$m_0=x'y'$	$M_0=x + y$
0	1	$m_1=x'y$	$M_1=x + y'$
1	0	$m_2=xy'$	$M_2=x' + y$
1	1	$m_3=xy$	$M_3=x' + y'$

If the binary variable is '0', then it is represented as complement of variable in min term and as the variable itself in Max term. Similarly, if the binary variable is '1', then it is represented as complement of variable in Max term and as the variable itself in min term.

From the above table, we can easily notice that min terms and Max terms are complement of each other. If there are 'n' Boolean variables, then there will be 2^n min terms and 2^n Max terms.

CANONICAL SOP & POS FORMS

A truth table consists of a set of inputs and outputs. If there are 'n' input variables, then there will be 2^n possible combinations with zeros and ones. So the value of each output variable depends on the combination of input variables. So, each output variable will have '1' for some combination of input variables and '0' for some other combination of input variables.

Therefore, we can express each output variable in following two ways.

- Canonical SoP form
- Canonical PoS form

CANONICAL SOP FORM

Canonical SoP form means Canonical Sum of Products form. In this form, each product term contains all literals. So, these product terms are nothing but the min terms.

Hence, canonical SoP form is also called as **sum of min terms** form.

First, identify the min terms for which, the output variable is one and then do the logical OR of those min terms in order to get the Boolean expression function $function$ corresponding to that output variable.

This Boolean function will be in the form of sum of min terms.

Follow the same procedure for other output variables also, if there is more than one output variable.

Consider the following **truth table**.

Inputs			Output
p	**q**	**r**	**f**
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Here, the output ff is '1' for four combinations of inputs. The corresponding min terms are p'qr, pq'r, pqr', pqr. By doing logical OR of these four min terms, we will get the Boolean function of output ff.

Therefore, the Boolean function of output is, f = p'qr + pq'r + pqr' + pqr. This is the **canonical SoP form** of output, f. We can also represent this function in following two notations.

$$f=m3+m5+m6+m7 \quad f=m3+m5+m6+m7$$

$$f=\sum m(3,5,6,7) \quad f=\sum m(3,5,6,7)$$

In one equation, we represented the function as sum of respective min terms. In other equation, we used the symbol for summation of those min terms.

CANONICAL POS FORM

Canonical PoS form means Canonical Product of Sums form. In this form, each sum term contains all literals. So, these sum terms are nothing but the Max terms. Hence, canonical PoS form is also called as **product of Max terms** form.

First, identify the Max terms for which, the output variable is zero and then do the logical AND of those Max terms in order to get the Boolean expression function$function$ corresponding to that output variable. This Boolean function will be in the form of product of Max terms.

Follow the same procedure for other output variables also, if there is more than one output variable.

Example

Consider the same truth table of previous example. Here, the output ff is '0' for four combinations of inputs. The corresponding Max terms are p + q + r, p + q + r', p + q' + r, p' + q + r. By doing logical AND of these four Max terms, we will get the Boolean function of output ff.

Therefore, the Boolean function of output is, f = p+q+r$p+q+r$.p+q+r'$p+q+r'$.p+q'+r$p+q'+r$.p'+q+r$p'+q+r$. This is the **canonical PoS form** of output, f. We can also represent this function in following two notations.

$$f=M0.M1.M2.M4 \quad f=M0.M1.M2.M4$$

$$f=\prod M(0,1,2,4) \quad f=\prod M(0,1,2,4)$$

In one equation, we represented the function as product of respective Max terms. In other equation, we used the symbol for multiplication of those Max terms.

The Boolean function, f = p+q+r$p+q+r$.p+q+r'$p+q+r'$.p+q'+r$p+q'+r$.p'+q+r$p'+q+r$ is the dual of the Boolean function, f = p'qr + pq'r + pqr' + pqr.

Therefore, both canonical SoP and canonical PoS forms are **Dual** to each other. Functionally, these two forms are same. Based on the requirement, we can use one of these two forms.

STANDARD SOP & POS FORM

We discussed two canonical forms of representing the Boolean outputs$s$. Similarly, there are two standard forms of representing the Boolean outputs$s$. These are the simplified version of canonical forms.

- Standard SoP form
- Standard PoS form

We will discuss about Logic gates in later chapters. The main **advantage** of standard forms is that the number of inputs applied to logic gates can be minimized. Sometimes, there will be reduction in the total number of logic gates required.

STANDARD SOP FORM

Standard SoP form means **Standard Sum of Products** form.

In this form, each product term need not contain all literals. So, the product terms may or may not be the min terms. Therefore, the Standard SoP form is the simplified form of canonical SoP form.

We will get Standard SoP form of output variable in two steps.

- Get the canonical SoP form of output variable
- Simplify the above Boolean function, which is in canonical SoP form.

Follow the same procedure for other output variables also, if there is more than one output variable. Sometimes, it may not possible to simplify the canonical SoP form. In that case, both canonical and standard SoP forms are same.

Example

Convert the following Boolean function into Standard SoP form.

$$f = p'qr + pq'r + pqr' + pqr$$

The given Boolean function is in canonical SoP form. Now, we have to simplify this Boolean function in order to get standard SoP form.

Step 1 − Use the **Boolean postulate**, x + x = x. That means, the Logical OR operation with any Boolean variable 'n' times will be equal to the same variable. So, we can write the last term pqr two more times.

$$\Rightarrow f = p'qr + pq'r + pqr' + pqr + pqr + pqr$$

Step 2 − Use **Distributive law** for 1st and 4th terms, 2nd and 5th terms, 3rd and 6th terms.

$$\Rightarrow f = qrp'+pp'+p + prq'+qq'+q + pqr'+rr'+r$$

Step 3 − Use **Boolean postulate**, x + x' = 1 for simplifying the terms present in each parenthesis.

$$\Rightarrow f = qr11 + pr11 + pq11$$

Step 4 − Use **Boolean postulate**, x.1 = x for simplifying above three terms.

$$\Rightarrow f = qr + pr + pq$$

$$\Rightarrow f = pq + qr + pr$$

This is the simplified Boolean function. Therefore, the **standard SoP form** corresponding to given canonical SoP form is **f = pq + qr + pr**

STANDARD POS FORM

Standard PoS form means **Standard Product of Sums** form.

In this form, each sum term need not contain all literals. So, the sum terms may or may not be the Max terms. Therefore, the Standard PoS form is the simplified form of canonical PoS form.

We will get Standard PoS form of output variable in two steps.

- Get the canonical PoS form of output variable
- Simplify the above Boolean function, which is in canonical PoS form.

Follow the same procedure for other output variables also, if there is more than one output variable. Sometimes, it may not possible to simplify the canonical PoS form.

In that case, both canonical and standard PoS forms are same.

Example

Convert the following Boolean function into Standard PoS form.

$$f = p+q+rp+q+r.p+q+r'p+q+r'.p+q'+rp+q'+r.p'+q+rp'+q+r$$

The given Boolean function is in canonical PoS form. Now, we have to simplify this Boolean function in order to get standard PoS form.

Step 1 − Use the **Boolean postulate**, x.x = x. That means, the Logical AND operation with any Boolean variable 'n' times will be equal to the same variable. So, we can write the first term p+q+r two more times.

$$\Rightarrow f = p+q+rp+q+r.p+q+rp+q+r.p+q+rp+q+r.p+q+r'p+q+r'.p+q'+rp+q'+r.p'+q+rp'+q+r$$

Step 2 − Use **Distributive law,** x + y.z$y.z$ = x+yx+y.x+zx+z for 1st and 4th parenthesis, 2nd and 5th parenthesis, 3rd and 6th parenthesis.

$$\Rightarrow f = p+q+rr'p+q+rr'.p+r+qq'p+r+qq'.q+r+pp'q+r+pp'$$

Step 3 − Use **Boolean postulate**, x.x'=0 for simplifying the terms present in each parenthesis.

$$\Rightarrow f = p+q+0p+q+0.p+r+0p+r+0.q+r+0q+r+0$$

Step 4 − Use **Boolean postulate**, x + 0 = x for simplifying the terms present in each parenthesis

$$\Rightarrow f = p+qp+q.p+rp+r.q+rq+r$$

$$\Rightarrow f = p+qp+q.q+rq+r.p+rp+r$$

This is the simplified Boolean function.

Therefore, the **standard PoS form** corresponding to given canonical PoS form is **f =** p+q$p+q$**.**q+r$q+r$**.**p+r$p+r$.

This is the **dual** of the Boolean function, f = pq + qr + pr.

Therefore, both Standard SoP and Standard PoS forms are Dual to each other.

KARNAUGH MAP (K MAP)

In numerous digital circuits and other practical problems, finding expressions that have minimum variables becomes a prerequisite. In such cases, minimisation of Boolean expressions is possible that have 3, 4 variables. It can be done using the Karnaugh map without using any theorems of Boolean algebra.

The K-map can easily take two forms, namely, Sum of Product or SOP and Product of Sum or POS, according to what we need in the problem. K-map is a representation that is table-like, but it gives more data than the TRUTH TABLE. Fill a grid of K-map with 1s and 0s, then solve it by creating various groups.

Solving an Expression Using K-Map

Here are the steps that are used to solve an expression using the K-map method:

1. Select a K-map according to the total number of variables.

2. Identify maxterms or minterms as given in the problem.

3. For SOP, put the 1's in the blocks of the K-map with respect to the minterms (elsewhere 0's).

4. For POS, putting 0's in the blocks of the K-map with respect to the maxterms (elsewhere 1's).

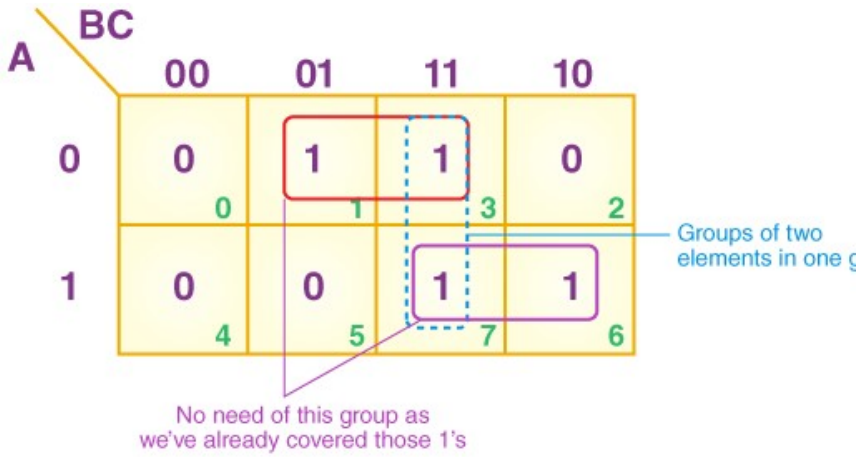
5. Making rectangular groups that contain the total terms in the power of two, such as 2,4,8 ..(except 1) and trying to cover as many numbers of elements as we can in a single group.

6. From the groups that have been created in step 5, find the product terms and then sum them up for the SOP form.

SOP FORM

1. 3 variables K-map:

Z = ∑P, Q, R (1, 3, 6, 7)



From the red group, the product term would be —

P'R

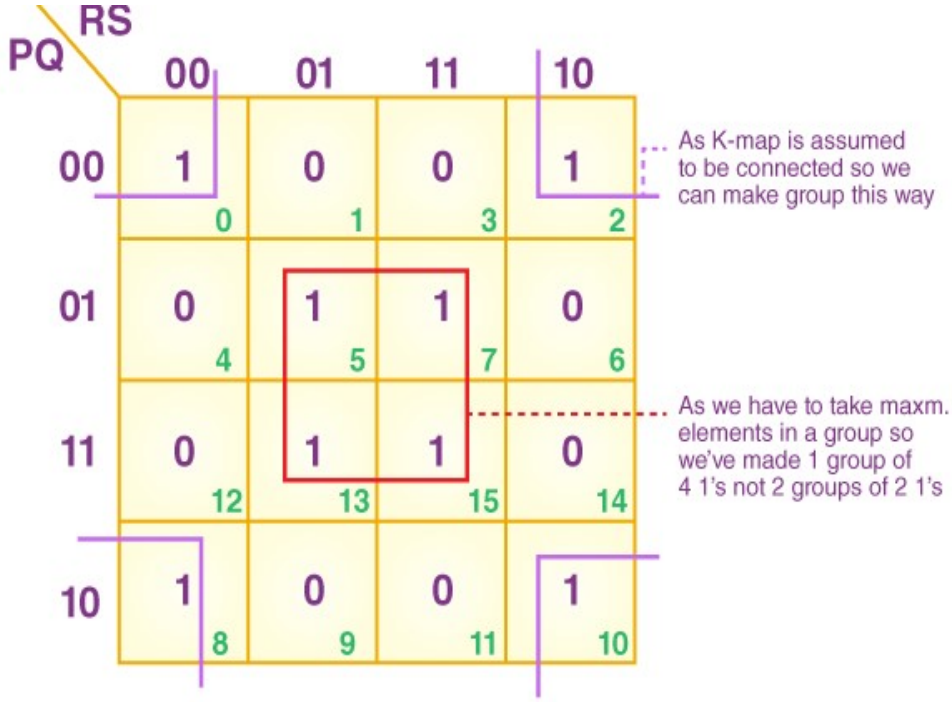
From the green group, the product term would be —

PQ

If we sum these product terms, then we will get this final expression (P'R + PQ)

2. 4 variables K-map:

F (A, B, C, D) = ∑(0, 2, 5, 7, 8, 10, 13, 15)



From the red group, the product term would be —

BD

From the lilac group, the product term would be —

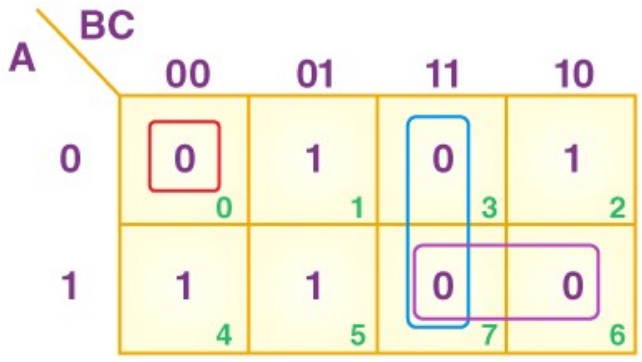
B'D'

If we sum these product terms, then we will get this final expression (BD + B'D')

POS FORM

1. 3 variables K-map

F (P, Q, R) = π(0,3,6,7)



From the lilac group, the terms would be

P Q

If we take the complement of these two

P’ Q’

And then sum up them

(P’ + Q’)

From the blue group, the terms would be

B R

When we take the complement of these terms

B’ R’

And then sum them up

(B’ + R’)

From the red group, the terms would be

P’ Q’ R’

If we take the complement of the two terms

P Q R

And then sum them up

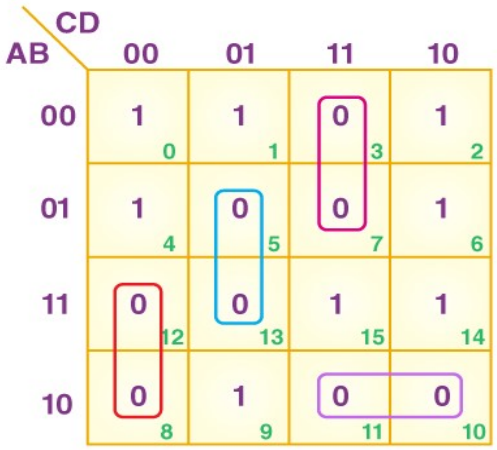
(P + Q + R)

If we take the product of these three terms, then we will get this final expression –

(P' + Q') (P' + R') (P + Q + R)

2. 4 variables K-map

F (P, Q, R, S) = π (3, 5, 7, 8, 10, 11, 12, 13)



From the blue group, the terms would be

R' S Q

We take their complement and then sum them

(R + S'+ Q')

From the purple group, the terms would be

R S P'

We take their complement and then sum them

(R' + S'+ P) S

From the red group, the terms would be

P R' S'

We take their complement and then sum them

(P' + R + S)

From the lilac group, the terms would be

P Q' R

We take their complement and then sum them

(P' + Q + R')

Finally, we will express these in the form of the product –

(R + S'+ Q').(R' + S'+A).(P'+ R + S).(P'+ Q + R')

Pitfall – Always remember that *POS ≠ (SOP)'*

*Here, the correct form would be (POS of F) = (SOP of F')'

Creating 5 Variable K Map

Rules to be followed while creating 5 variable K map

- If a function is given in compact canonical SOP (Sum of Products) form, we write "1" in the corresponding cell numbers for each minterm (provided in the question). For example, for summation of (0, 1, 5, 7, 30, 31), we will write "1" for cell numbers (0, 1, 5, 7, 30, and 31).
- If a function is given in compact canonical POS (Product of Sums) form, we write "0" in the corresponding cell numbers for each maxterm (provided in the question). For example, for products of (0, 1, 5, 7, 30, 31), we will write "0" for cell numbers (0, 1, 5, 7, 30, and 31).

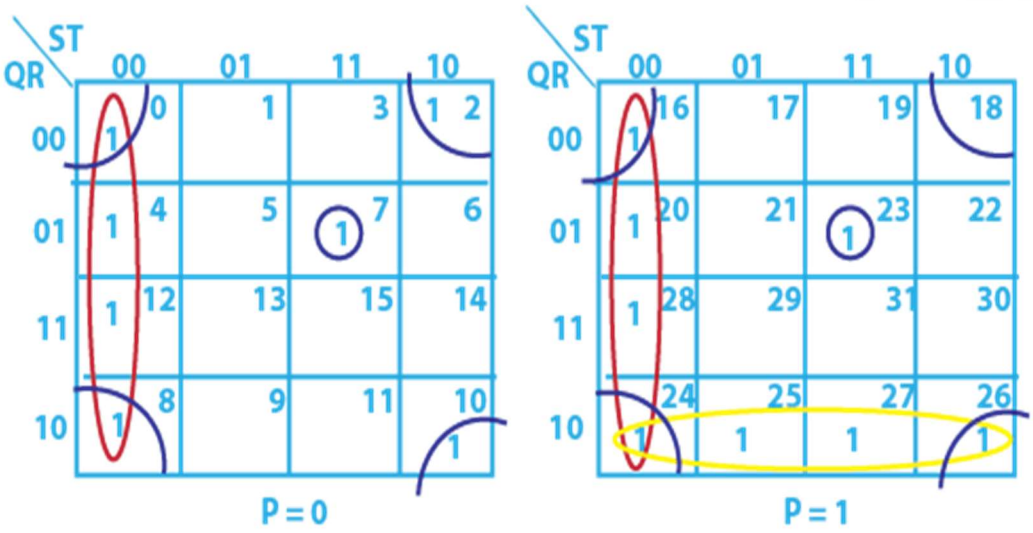
Steps to be followed while creating 5 variable K map

- In the K-Map, create the largest possible size subcube that covers all the marked 1's in the case of SOP or all the marked 0's in the case of POS. It should be noted that each subcube can only contain terms with powers of two. A subcube of 2^m cells is also possible if and only if each cell in that subcube has a "m" number of adjacent cells.
- All Essential Prime Implicants (EPIs) must be present in the minimal expressions.

Solving the SOP function

For a clear understanding, let us solve the example of SOP function minimization of5 variable K Map using the following expression : summation of (0, 2, 4, 7, 8, 10, 12, 16, 18, 20, 23, 24, 25, 26, 27, 28) In the above K-Map we have 4 subcubes:

- Subcube 1: The one marked in red comprises cells (0, 4, 8, 12, 16, 20, 24, 28)
- Subcube 2: The one marked in blue comprises cells (7, 23)
- Subcube 3: The one marked in pink comprises cells (0, 2, 8, 10, 16, 18, 24, 26)
- Subcube 4: The one marked in yellow comprises cells (24, 25, 26, 27)



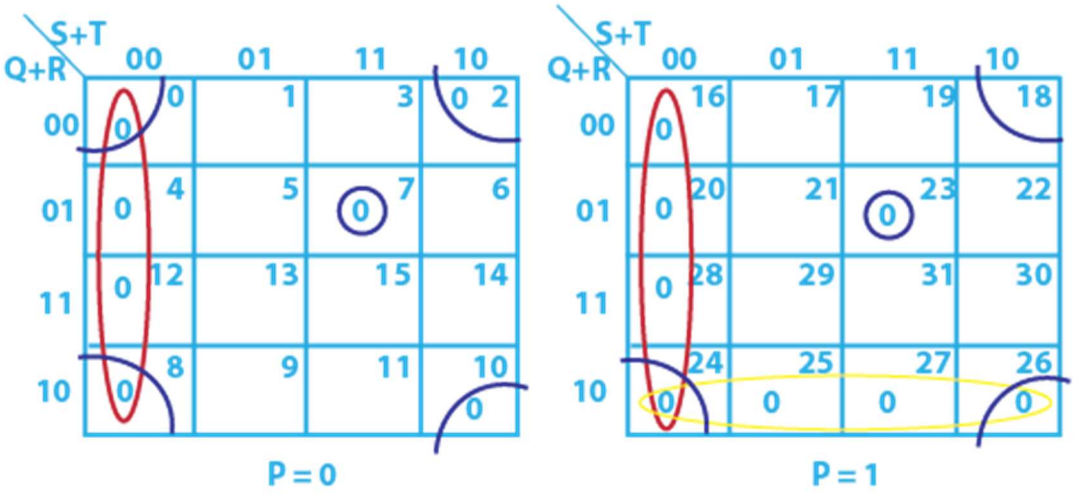
Now, while writing the minimal expression of each of the subcubes, we will search for the literal that is common to all the cells present in that subcube.

Finally, the minimal expression of the given boolean Function can be expressed as follows:

$$f(PQRST) = (S+T)(Q+\bar{R}+\bar{S}+\bar{T})(R+T)(\bar{P}+\bar{Q}+R)$$

Solving the POS function

Now, let us solve the example of POS function minimization of a5 variable K Map using the following expression: prod. of (0, 2, 4, 7, 8, 10, 12, 16, 18, 20, 23, 24, 25, 26, 27, 28) In the above K-Map we have 4 subcubes:



Now, while writing the minimal expression of each of the subcubes, we will search for the literal that is common to all the cells present in that subcube.

Finally, the minimal expression of the given boolean Function can be expressed as follows:

NOTE:

For the5 variable K Map, the Range of the cell numbers will be from 0 to 2^5 -1 i.e., 0 to 31.
The above-mentioned term “Adjacent Cells” means “any two cells that differ in only one variable”.

Grouping and Simplification

Grouping is a critical step in K-Map simplification. The goal is to identify adjacent cells with the same output value and form groups that cover as many cells as possible, preferably in powers of two (1, 2, 4, 8, etc.). The groups should be rectangular in shape and can wrap around the edges of the K-Map if necessary.

After forming groups, we express the simplified Boolean expression by combining the variables that remain constant within each group. We use the Boolean OR operation (+) to combine these groups, and the result represents the simplified expression.

HANDLING DON’T CARE CONDITIONS

Using 5-Variable K Maps in Circuit Design

The ultimate goal of K-Map simplification is to design more efficient digital circuits. The simplified Boolean expression obtained from the K-Map helps in creating a logic circuit with fewer gates and lower power consumption. This leads to more cost-effective and reliable designs.

To implement the simplified expression, we use logic gates like AND, OR, and NOT gates to build the circuit. Utilizing the minimized expression reduces propagation delays and minimizes the chances of glitches or errors in the circuit’s output.

Conclusion

Five Variable K Maps are a valuable tool in digital circuit design, allowing designers to simplify complex Boolean expressions and create more efficient logic circuits. By understanding the principles of K Maps, how to create them, and how to group cells for simplification, you can effectively design digital circuits with fewer gates, lower power consumption, and improved reliability. Mastering K Maps is a crucial skill for digital circuit designers and engineers, as it enables them to optimize circuit designs and create high-performance systems.

UNIVERSAL LOGIC GATES – NAND & NOR GATES

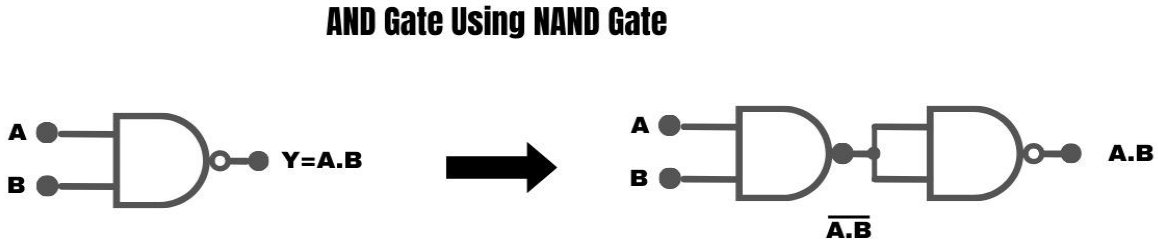
NAND Universal Logic Gate

The NAND is a universal gate. It can be used to create any other type of logic gate. Here are the techniques for implementing fundamental gates using NAND gates:

AND Gate Using NAND Gate

You require two NAND gates to create an AND Gate.

- In the first NAND gate, the outcome is the inverse of the logical AND operation between the two inputs.
- The second NAND gate then inverts the output from the first gate, thus producing the original AND logic.

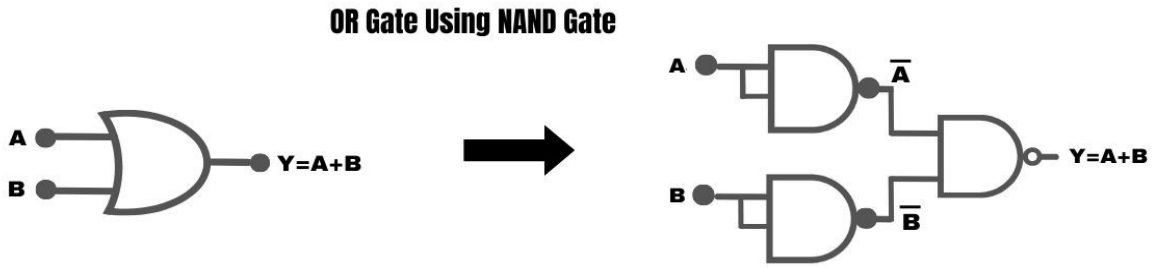


A NAND gate is employed as an AND gate by inverting its inputs and output. The outcome replicates the behavior of an AND gate, where only when both inputs are 1 does the output become 0 due to the inversion.

OR Gate Using NAND Gate

Constructing an OR gate involves three NAND gates.

- The first NAND gate produces the inverse of the first input.
- The second NAND gate generates the inverse of the second input.
- The third NAND gate computes the logical OR of the outputs from the first two NAND gates.



A NAND gate is used as an OR gate by inverting its inputs and output. The result is that when both inputs are 0, the NAND gate outputs 1, simulating an OR gate's behavior.

NOR Gate Using NAND Gate

A NOR gate with inputs A and B can be constructed using three NAND gates as follows:

First NAND Gate

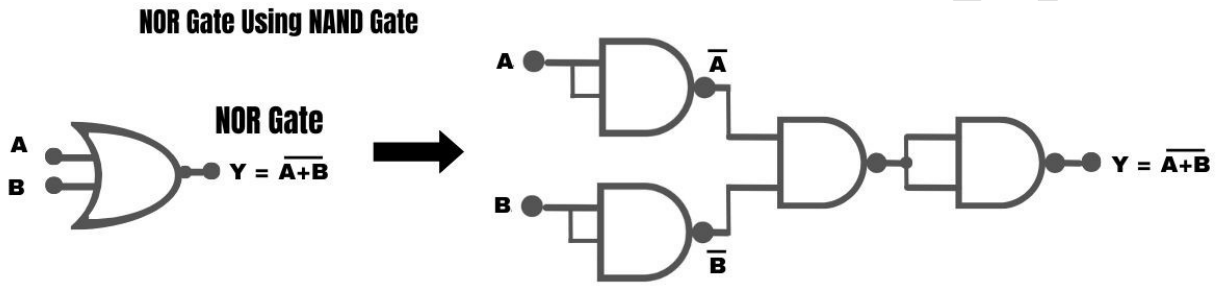
- Connect inputs A and B to the two inputs of the first NAND gate.
- The output of the first NAND gate will be A NAND B (A.B) since NAND is the complement of AND.

Second NAND Gate

- Connect the output of the first NAND gate (A.B) to both inputs of the second NAND gate.
- The output of the second NAND gate will be the complement of A.B, which is ¬(A.B).

Third NAND Gate

- Connect the outputs of the first and second NAND gates to the two inputs of the third NAND gate.
- The output of the third NAND gate will be ¬(A.B) NAND ¬(A.B), which simplifies to ¬(A.B).

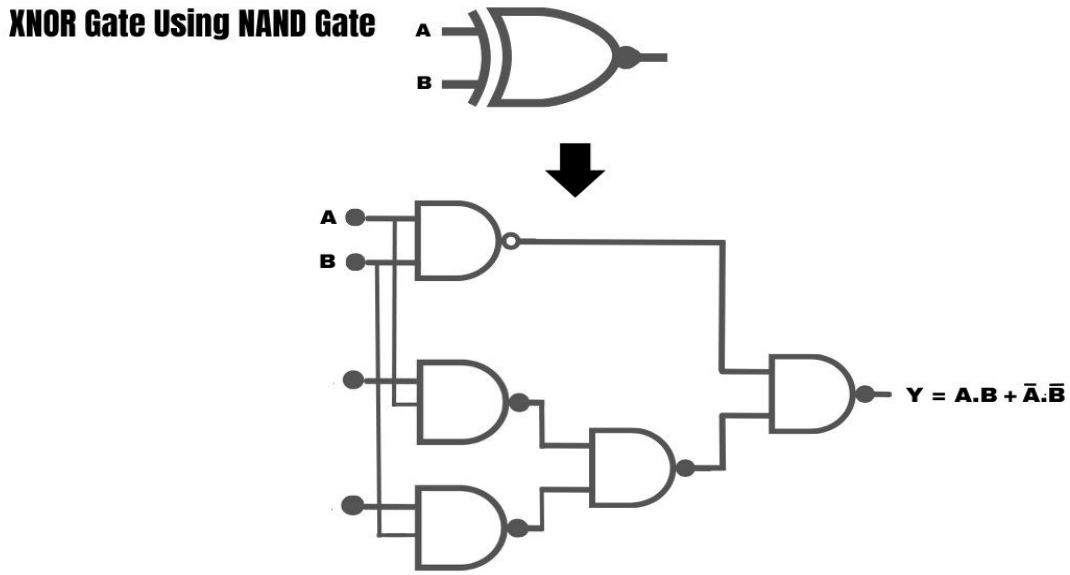


An XNOR gate can be implemented using NAND gates –

XNOR Gate Using NAND Gate

- The first NAND gate takes the inputs A and B and inverts them.
- The second NAND gate takes the inverted inputs and performs an OR operation.
- The third NAND gate takes the output of the second NAND gate and inverts it again.

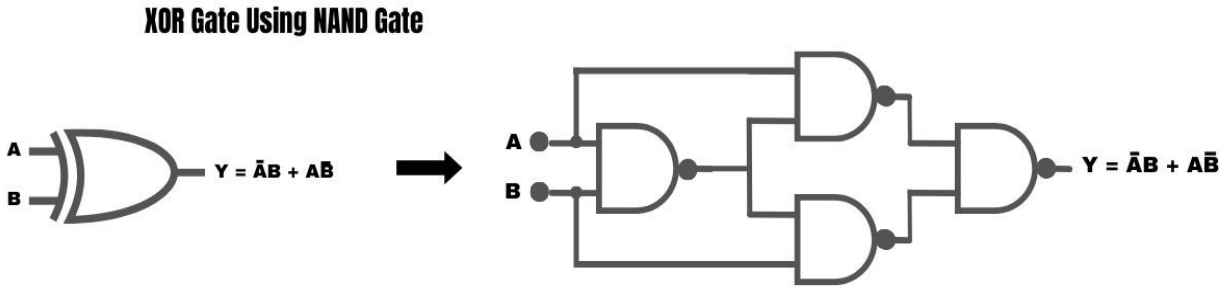
The output of the third NAND gate is the output of the XNOR gate.



XOR Gate Using NAND Gate

Building an XOR gate requires four NAND gates.

- The first two NAND gates produce the inverse of the respective inputs.
- The third NAND gate computes the logical AND of the outputs from the first two NAND gates.
- The fourth NAND gate inverts the output of the third NAND gate, resulting in the XOR operation.



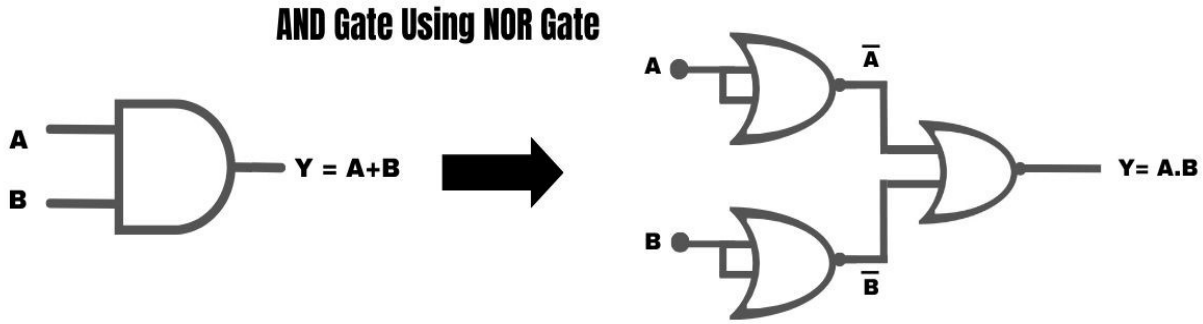
NOR Universal Logic Gate

The NOR gate is also a universal gate and can be used to create any other kind of logic gate. Below are the methodologies for implementing fundamental gates using NOR gates:

AND Gate Using NOR Gate

Three NOR gates are required to construct an AND gate.

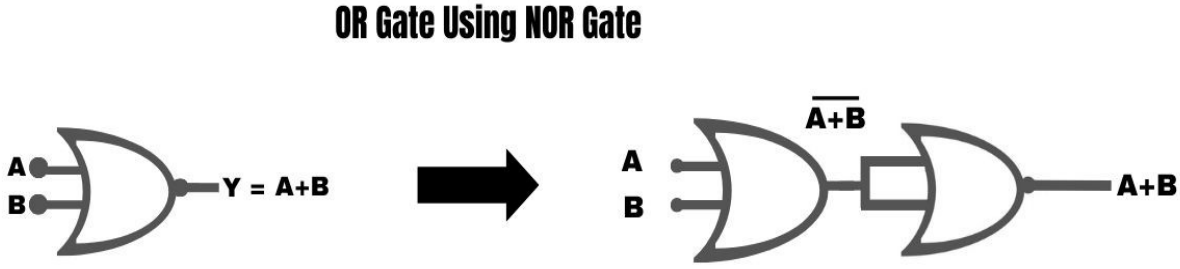
- The first NOR gate's output is the inverse of the logical OR operation between the two inputs.
- The second NOR gate inverts the output of the first NOR gate, resulting in the original AND logic.



OR Gate Using NOR Gate

An OR gate can be created using two NOR gates, as demonstrated.

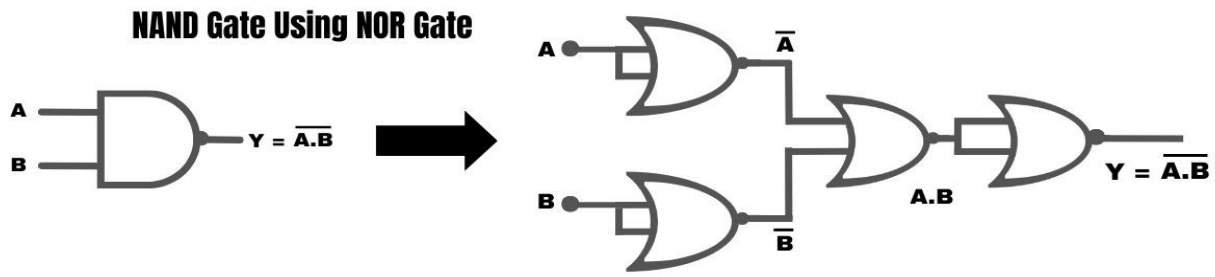
- The output of the first NOR gate is the complement of each input.
- The second NOR gate computes the logical OR of the outputs from the first NOR gate.



NAND Gate Using NOR Gate

A NAND gate can be implemented using NOR gates.

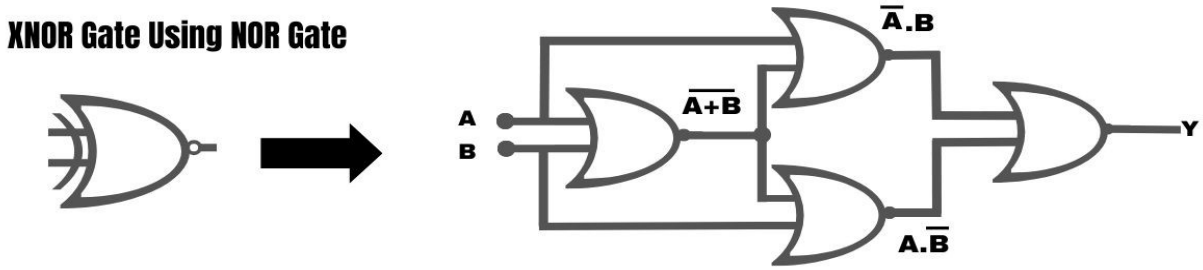
1. The first NOR gate takes the input A and inverts it.
2. The second NOR gate takes the input B and inverts it.
3. The third NOR gate takes the outputs of the first two NOR gates and performs an OR operation.
4. The output of the third NOR gate is the output of the NAND gate.



XNOR Gate using NOR Gate

- The first NOR gate takes the inputs A and B and inverts them.
- The second NOR gate takes the inverted inputs and performs an OR operation.
- The third NOR gate takes the output of the second NOR gate and inverts it again.

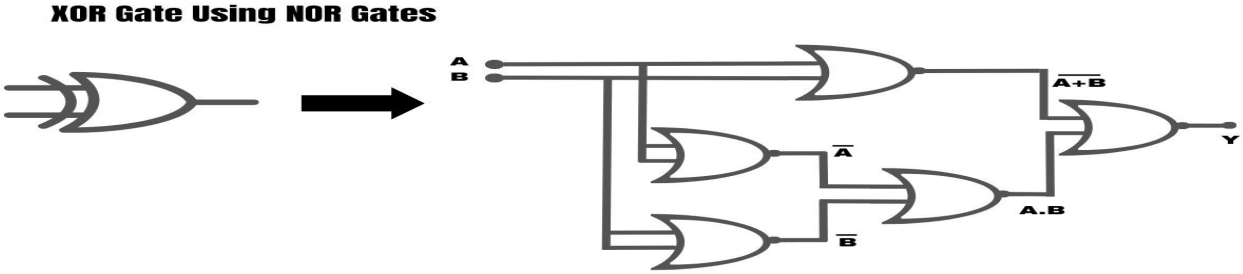
The output of the third NOR gate is the output of the XNOR gate.



XOR Gate Using NOR Gate

An XOR gate can be implemented using 3 NOR gates –

- The 1st NOR gate takes the inputs A and B and inverts them.
- The 2nd NOR gate takes the inverted inputs and inverts them again.
- The 3rd NOR gate takes the outputs of the first two NOR gates and performs a NOR operation.



The output of the third NOR gate is the output of the XOR gate.

QUINE – McCLUSKEY TABULAR METHOD

Procedure of Quine-McCluskey Tabular Method

Follow these steps for simplifying Boolean functions using Quine-McClukey tabular method.

Step 1 − Arrange the given min terms in an **ascending order** and make the groups based on the number of ones present in their binary representations. So, there will be **at most 'n+1' groups** if there are 'n' Boolean variables in a Boolean function or 'n' bits in the binary equivalent of min terms.

Step 2 − Compare the min terms present in **successive groups**. If there is a change in only one-bit position, then take the pair of those two min terms. Place this symbol '_' in the differed bit position and keep the remaining bits as it is.

Step 3 − Repeat step2 with newly formed terms till we get all **prime implicants**.

Step 4 − Formulate the **prime implicant table**. It consists of set of rows and columns. Prime implicants can be placed in row wise and min terms can be placed in column wise. Place '1' in the cells corresponding to the min terms that are covered in each prime implicant.

Step 5 − Find the essential prime implicants by observing each column. If the min term is covered only by one prime implicant, then it is **essential prime implicant**. Those essential prime implicants will be part of the simplified Boolean function.

Step 6 − Reduce the prime implicant table by removing the row of each essential prime implicant and the columns corresponding to the min terms that are covered in that essential prime implicant. Repeat step 5 for Reduced prime implicant table. Stop this process when all min terms of given Boolean function are over.

For Example:

Let us **simplify** the following Boolean function,

f(W,X,Y,Z)=∑m(2,6,8,9,10,11,14,15)

$f(W,X,Y,Z)=\sum m(2,6,8,9,10,11,14,15)$

using Quine-McClukey tabular method.

The given Boolean function is in **sum of min terms** form. It is having 4 variables W, X, Y & Z.

The given min terms are 2, 6, 8, 9, 10, 11, 14 and 15. The ascending order of these min terms based on the number of ones present in their binary equivalent is 2, 8, 6, 9, 10, 11, 14 and 15.

The following table shows these **min terms and their equivalent binary** representations.

Group Name	Min terms	W	X	Y	Z
GA1	2	0	0	1	0
	8	1	0	0	0
GA2	6	0	1	1	0
	9	1	0	0	1
	10	1	0	1	0
GA3	11	1	0	1	1
	14	1	1	1	0
GA4	15	1	1	1	1

The given min terms are arranged into 4 groups based on the number of ones present in their binary equivalents.

The following table shows the possible **merging of min terms** from adjacent groups.

Group Name	Min terms	W	X	Y	Z
	2,6	0	-	1	0
	2,10	-	0	1	0
GB1	8,9	1	0	0	-
	8,10	1	0	-	0
	6,14	-	1	1	0
GB2	9,11	1	0	-	1

	10,11	1	0	1	-
	10,14	1	-	1	0
GB3	11,15	1	-	1	1
	14,15	1	1	1	-

The min terms, which are differed in only one-bit position from adjacent groups are merged.

That differed bit is represented with this symbol, '-'.

In this case, there are three groups and each group contains combinations of two min terms.

The following table shows the possible **merging of min term pairs** from adjacent groups.

Group Name	Min terms	W	X	Y	Z
GB1	2,6,10,14	-	-	1	0
	2,10,6,14	-	-	1	0
	8,9,10,11	1	0	-	-
	8,10,9,11	1	0	-	-
GB2	10,11,14,15	1	-	1	-
	10,14,11,15	1	-	1	-

The successive groups of min term pairs, which are differed in only one-bit position are merged. That differed bit is represented with this symbol, '-'.

In this case, there are two groups and each group contains combinations of four min terms.

Here, these combinations of 4 min terms are available in two rows.

So, we can remove the repeated rows. The reduced table after removing the redundant rows is shown below.

Group Name	Min terms	W	X	Y	Z
GC1	2,6,10,14	-	-	1	0
	8,9,10,11	1	0	-	-
GC2	10,11,14,15	1	-	1	-

Further merging of the combinations of min terms from adjacent groups is not possible, since they are differed in more than one-bit position.

There are three rows in the above table. So, each row will give one prime implicant. Therefore, the **prime implicants** are YZ', WX' & WY.

*The **prime implicant table** is shown below.*

Min terms / Prime Implicants	2	6	8	9	10	11	14	15
YZ'	1	1			1		1	
WX'			1	1	1	1		
WY					1	1	1	1

The prime implicants are placed in row wise and min terms are placed in column wise. 1s are placed in the common cells of prime implicant rows and the corresponding min term columns.The min terms 2 and 6 are covered only by one prime implicant **YZ'**. So, it is an **essential prime implicant**. This will be part of simplified Boolean function.

Now, remove this prime implicant row and the corresponding min term columns. The reduced prime implicant table is shown below.

Min terms / Prime Implicants	8	9	11	15
WX'	1	1	1	
WY			1	1

The min terms 8 and 9 are covered only by one prime implicant **WX'**. So, it is an **essential prime implicant**. This will be part of simplified Boolean function. Now, remove this prime implicant row and the corresponding min term columns. The reduced prime implicant table is shown below.

Min terms / Prime Implicants		**15**
WY	1	

The min term 15 is covered only by one prime implicant **WY**. So, it is an **essential prime implicant**. This will be part of simplified Boolean function.

In this example problem, we got three prime implicants and all the three are essential. Therefore, the **simplified Boolean function** is

fW,X,Y,ZW,X,Y,Z **= YZ' + WX' + WY.**