Digital Electronics & Computer Organization (DECO)

1st Year CSE 2024-25 (Odd Semester)

Unit-1

Binary Logic, Basic Theorems and Properties of Boolean algebra, Boolean Functions, Digital Logic Gates, Introduction, the Map Method, Four-Variable Map, Don't-Care Conditions, NAND and NOR Implementation.

Study Material

Binary Number System:

Binary Number System: According to digital electronics and mathematics, a binary number is defined as a number that is expressed in the binary system or base 2 numeral system. It describes numeric values by two separate symbols; 1 (one) and 0 (zero). The base-2 system is the positional notation with 2 as a radix.
The binary system is applied internally by almost all latest computers and computer-based devices because of its direct implementation in electronic circuits using logic gates. Every digit is referred to as a **bit**.

Example: Convert 4 in binary.
Solution:
4 in binary is $(100)_2$.
Here, 4 is represented in the decimal number system, where we can represent the number using the digits from 0-9. However, in a binary number system, we use only two digits, such as 0 and 1.
Now, let's discuss how to convert 4 in binary number system. The following steps help to convert 4 in binary.

Step 1: First, divide the number 4 by 2. Use the integer quotient obtained in this step as the dividend for the next step. Continue this step, until the quotient becomes 0.

Dividend	**Remainder**
4/2 = 2	0
2/2 = 1	0
1/2 = 0	1

Step 2: Now, write the remainder in reverse chronological order. (i.e. from bottom to top).
Here, the Least Significant Bit (LSB) is 0 and the Most Significant Bit (MSB) is 1.
Hence, the decimal number 4 in binary is 100_2
So, if we want to find how many bits does 4 in binary have? We have to count the number of zeros and ones.
So, 4 in binary is 100_2. Here, there are 2 zeroes and 1 one. Hence, we have 3 bits.
Therefore, the number of bits does 4 in binary have is 3.
What is Bit in Binary Number?

A single binary digit is called a "**Bit".** A binary number consists of several bits. Examples are:

- 10101 is a five-bit binary number
- 101 is a three-bit binary number
- 100001 is a six-bit binary number

Facts to Remember:

- Binary numbers are made up of only 0's and 1's.
- A binary number is represented with a base-2
- A bit is a single binary digit.

Binary Numbers Table

Some of the binary notations of lists of decimal numbers from 1 to 30, are mentioned in the below list.

Number	Binary Number	Number	Binary Number	Number	Binary Number
1	1	11	1011	21	10101
2	10	12	1100	22	10110
3	11	13	1101	23	10111
4	100	14	1110	24	11000
5	101	15	1111	25	11001
6	110	16	10000	26	11010
7	111	17	10001	27	11011
8	1000	18	10010	28	11100
9	1001	19	10011	29	11101
10	1010	20	10100	30	11110

Binary logic:

Binary logic is the basis of electronic systems, such as computers and cell phones. It works on 0's and 1's. It involves addition, subtraction, multiplication, division of zeros and ones. It includes logic gate functions, AND, OR and NOT which translates input signals into specific output.

Boolean logic:

Boolean logic is a type of algebra in which results are calculated as either TRUE or FALSE (known as truth values or truth variables). Instead of using arithmetic operators like addition, subtraction, and multiplication, Boolean logic utilizes three basic logical operators: AND, OR, and NOT.

TRUE and FALSE: There can only be two
Behind Boolean logic are two very simple words: TRUE and FALSE.

For electronic systems Boolean logics are equivalent to binary logics as follows;

Boolean Logic		**Binary Logic**
TRUE	→	1
FALSE	→	0

So, all the mathematics related to binary logics is based on Boolean algebra.

Boolean algebra:

The logical symbol 0 and 1 are used for representing the digital input or output. The symbols "1" and "0" can also be used for a permanently open and closed digital circuit. The digital circuit can be made up of several logic gates. To perform the logical operation with minimum logic gates, a set of rules were invented, known as the **Laws of Boolean Algebra**. These rules are used to reduce the number of logic gates for performing logic operations.
The Boolean algebra is mainly used for simplifying and analyzing the complex Boolean expression. It is also known as **Binary algebra** because we only use binary numbers in this. **George Boole** developed the binary algebra in **1854**.

Rules in Boolean algebra
Only two values (1 for high/true and 0 for low/false) are possible for the variable used in Boolean algebra.

The overbar (-) or apostrophe (') is used for representing the complement variable. So, the complement of variable A is represented as $\overline{A}$ or A'.
The plus (+) operator is used to represent the **OR**ing of the variables.
The dot (.) operator is used to represent the **AND**ing of the variables.

Properties of Boolean algebra

These are the following properties of Boolean algebra:

Annulment Law
When the variable is AND with 0, it will give the result 0, and when the variable is OR with 1, it will give the result 1, i.e.,
B.0 = 0
B+1 = 1

Identity Law
When the variable is AND with 1 and OR with 0, the variable remains the same, i.e.,
B.1 = B
B+0 = B

Idempotent Law
When the variable is AND and OR with itself, the variable remains same or unchanged, i.e.,
B.B = B
B+B = B

Complement Law
When the variable is AND and OR with its complement, it will give the result 0 and 1 respectively.
B.B' = 0
B+B' = 1

Double Negation Law
This law states that, when the variable comes with two negations, the symbol gets removed and the original variable is obtained.
((A)')' = A

Commutative Law
This law states that no matter in which order we use the variables. It means that the order of variables doesn't matter in this law.
A.B = B.A
A+B = B+A

Associative Law
This law states that the operation can be performed in any order when the variables priority is of same as 'Multiply' and 'Divide'.
(A.B).C = A.(B.C)
(A+B)+C = A+(B+C)

Distributive Law
This law allows us to open up of brackets. Simply, we can open the brackets in the Boolean expressions.
A+(B.C) = (A+B).(A+C
A.(B+C) = (A.B)+(A.C)

Absorption Law
This law allows us for absorbing the similar variables.
B+(B.A) = B
B.(B+A) = B

De Morgan Law
The operation of an OR and AND logic circuit will remain same if we invert all the inputs, change operators from AND to OR and OR to AND, and invert the output.
(A.B)' = A'+B'
(A+B)' = A'.B'

Logic Gates:

A **logic gate** is an electronic circuit designed by using electronic components like diodes, transistors, resistors, and more. As the name implies, a logic gate is designed to perform logical operations in digital systems like computers, communication systems, etc.
Therefore, we can say that the building blocks of a digital circuit are logic gates, which execute numerous logical operations that are required by any digital circuit. A logic gate can take two or more inputs but only produce one output. The output of a logic gate depends on the combination of inputs and the logical operation that the logic gate performs.

Logic gates use Boolean algebra to execute logical processes. Logic gates are found in nearly every digital gadget we use on a regular basis. Logic gates are used in the architecture of our telephones, laptops, tablets, and memory devices.

Types of Logic Gates
A logic gate is a digital gate that allows data to be manipulated. Logic gates, use logic to determine whether or not to pass a signal. Logic gates, on the other hand, govern the flow of information based on a set of rules.
The logic gates can be classified into the following major types:

1. Basic Logic Gates
There are three basic logic gates:
1. AND Gate
2. OR Gate
3. NOT Gate

2. Universal Logic Gates
In digital electronics, the following two logic gates are considered as universal logic gates:
1. NOR Gate
2. NAND Gate

3. Derived Logic Gates
The following two are the derived logic gates used in digital systems:
1. XOR Gate
2. XNOR Gate

Let us now discuss each of these types of logic gates in detail one-by-one.

AND Gate:

In digital electronics, the AND gate is one of the basic logic gate that performs the logical multiplication of inputs applied to it. It generates a high or logic 1 output, only when all the inputs applied to it are high or logic 1. Otherwise, the output of the AND gate is low or logic 0.

Properties of AND Gate
The following are two main properties of the AND gate:
- AND gate can accept two or more than two input values at a time.
- When all of the inputs are logic 1, the output of this gate is logic 1.

The operation of an AND gate is described by a mathematical expression, which is called the Boolean expression of the AND gate.
For two-input AND gate, the Boolean expression is given by,

$$Z=A.B$$

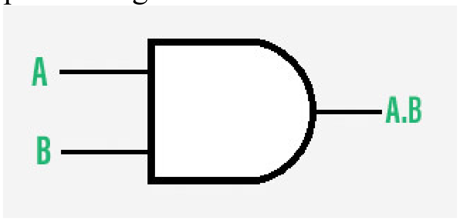
Where, A and B are inputs to the AND gate, while Z denotes the output of the AND gate.
We can extend this expression to any number of input variables, such as,

$$Z=A.B.C.D\ldots$$

Truth Table of AND Gate
The truth table of a two input AND gate is given below:

Input		Output
A	**B**	**A AND B**
0	0	0
0	1	0
1	0	0
1	1	1

Symbol of AND Gate
The logic symbol of a two input AND gate is shown in the following figure.



OR Gate:

In digital electronics, there is a type of basic logic gate which produces a low or logic 0 output only when it's all inputs are low or logic 0. For all other input combinations, the output of the OR gate is high or logic 1. This logic gate is termed as OR gate. An OR gate can be designed to have two or more inputs but only one output. The primary function of the OR gate is to perform the logical sum operation.

Properties of OR Gate
An OR gate have the following two properties:

- It can have two or more input lines at a time.
- When all of the inputs to the OR gate are low or logic 0, the output of it is low or logic 0.

The operation of an OR gate can be mathematically described through a mathematical expression called Boolean expression of the OR gate.
The Boolean expression for a two input OR gate is given by,

$$Z = A + B$$

The Boolean expression for a three-input OR gate is,

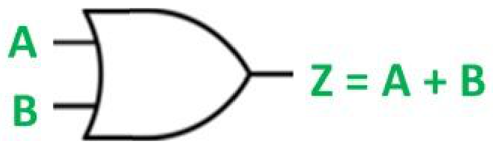
$$Z = A + B + C$$

Here, A, B, and C are inputs and Z is the output variables. We can extend this Boolean expression to any number of input variables.

Truth Table of OR Gate
The truth table of an OR gate describes the relationship between inputs and output. The following is the truth table for the two-input OR gate:

Input		**Output**
A	**B**	**A OR B**
0	0	0
0	1	1
1	0	1
1	1	1

Symbol of OR Gate:
The logic symbol of a two-input OR gate is shown in the following figure.



NOT Gate:

In digital electronics, the NOT gate is another basic logic gate used to perform **compliment of an input signal** applied to it. It takes only one input and one output. The output of the NOT gate is complement of the input applied to it. Therefore, if we apply a low or logic 0 output to the NOT gate is gives a high or logic 1 output and vice-versa. The NOT gate is also known as inverter, as it performs the inversion operation.

Properties of NOT Gate

- The output of a NOT gate is complement or inverse of the input applied to it.
- NOT gate takes only one output.

The logical operation of the NOT gate is described by its Boolean expression, which is given below.

$$Z = \overline{A}$$

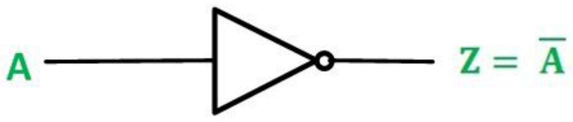
The bar over the input variable A represents the inversion operation.

Truth Table of OR Gate
The truth table describes the relationship between input and output. The following is the truth table for the NOT gate:

Input	**Output**
A	**NOT A**

Input	Output
0	1
1	0

Symbol of NOT Gate
The logic circuit symbol of a NOT gate is shown in the following figure. Here, A is the input line and Z is the output line.



NOR Gate:

The NOR gate is a type of universal logic gate that can take two or more inputs but one output. It is basically a combination of two basic logic gates i.e., OR gate and NOT gate. Thus, it can be expressed as,

NOR Gate = OR Gate + NOT Gate

In other words, a NOR gate is an OR gate followed by a NOT gate.

Properties of NOR Gate
The following are two important properties of NOR gate:

- A NOR gate can have two or more inputs and gives an output.
- A NOR gate gives a high or logic 1 output only when it's all inputs are low or logic 0.

Similar to basic logic gates, we can describe the operation of a NOR gate using a mathematical equation called Boolean expression of the NOR gate.
The Boolean expression of a two input NOR gate is given below:

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

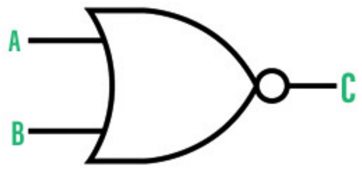
We can extend this expression to any number of input variables.
In the above Boolean expressions, the variables A and B are called input variables while the variable C is called the output variable.

Truth Table of NOR Gate
The following is the truth table of a two-input NOR gate showing the relationship between its inputs and output:

Input		Output
A	**B**	**A NOR B**
0	0	1
0	1	0

Input		**Output**
1	0	0
1	1	0

Symbol of NOR Gate



NAND Gate:

In digital electronics, the NAND gate is another type of universal logic gate used to perform logical operations. The NAND gate performs the inverted operation of the AND gate. Similar to NOR gate, the NAND gate can also have two or more input lines but only one output line.
The NAND gate is also represented as a combination of two basic logic gates namely, AND gate and NOT gate. Hence, it can be expressed as

NAND Gate = AND Gate + NOT Gate

Properties of NAND Gate
The following are the two key properties of NAND gate:

- NAND gate can take two or more inputs at a time and produces one output based on the combination of inputs applied.
- NAND gate produces a low or logic 0 output only when it's all inputs are high or logic 1.

We can describe the expression of NAND gate through a mathematical equation called its Boolean expression. Here is the Boolean expression of a two input NAND gate.

$$C = \overline{A.B}$$

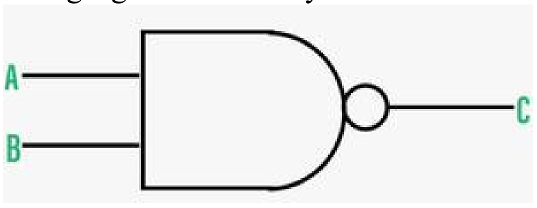
In this expression, A and B are the input variables and C is the output variable. We can extend this relation to any number of input variables like three, four, or more.

Truth Table of NAND Gate
The truth table is a table of inputs and output that describes the operation of the NAND gate and shows the logical relationship between them:

Input		**Output**
A	**B**	**A NAND B**
0	0	1

Input		Output
0	1	1
1	0	1
1	1	0

Symbol of NAND Gate
The logic symbol of a NAND gate is represented as a AND gate with a bubble on its output end as depicted in the following figure. It is the symbol of a two-input NAND gate.



XOR Gate:

In digital electronics, there is a specially designed logic gate named, XOR gate, which is used in digital circuits to perform **modulo sum**. It is also referred to as **Exclusive OR gate or Ex-OR gate**. The XOR gate can take only two inputs at a time and give an output. The output of the XOR gate is high or logic 1 only when its two inputs are dissimilar.

Properties of XOR Gate
The following two are the main properties of the XOR gate:

- It can accept only two inputs at a time. There is nothing like a three or more input XOR gate.
- The output of the XOR gate is logic 1 or high, when its inputs are dissimilar.

The operation of the XOR gate can be described through a mathematical equation called its Boolean expression. The following is the Boolean expression for the output of the XOR gate.

$$Z = A \oplus B$$

Here, Z is the output variable, and A and B are the input variables.
This expression can also be written as follows:

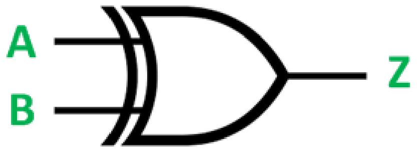
$$Z = \overline{A}B + A\overline{B}$$

Truth Table of XOR Gate
The truth table is a table of inputs and output that describe the relationship between them and the operation of the XOR gate for different input combinations. The truth table of the XOR gate is given below:

Input		Output
A	**B**	**A XOR B**

Input		**Output**
0	0	0
0	1	1
1	0	1
1	1	0

Symbol of XOR Gate:
The logic symbol of an XOR gate is shown in the following figure.



XNOR Gate:

The XNOR gate is another type of special purpose logic gate used to implement **exclusive operation in digital circuits**. It is used to implement the Exclusive NOR operation in digital circuits. It is also called the Ex-NOR or Exclusive NOR gate. It is a combination of two logic gates namely, XOR gate and NOT gate. Thus, it can be expressed as,

XNOR Gate = XOR Gate + NOT Gate

The output of an XNOR gate is high or logic 1 when it's both inputs are similar. Otherwise the output is low or logic 0. Hence, the XNOR gate is used as a similarity detector circuit.

Properties of XNOR Gate
The following are two key properties of XNOR gate:
- XNOR gate takes only two inputs and produces one output.
- The output of the XNOR gate is high or logic 1 only when it has similar inputs.

The operation of XNOR gate can be described through a mathematical equation called the Boolean expression of XNOR gate. Here is the Boolean expression of the XNOR gate.

$$Y=A\odot B$$

We can also write this expression as follows:

$$Z = \overline{A}\,\overline{B} + AB$$

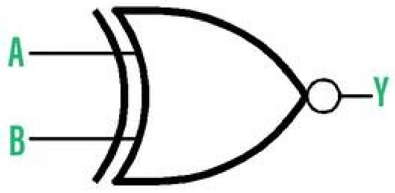
Here, the A and B are inputs and Y is the output.

Truth Table of XNOR Gate
The truth table of the XNOR gate is given below. This truth table is describing the relationship between inputs and output of the XNOR gate.

Input	**Output**

Input		Output
A	**B**	**A XNOR B**
0	0	1
0	1	0
1	0	0
1	1	1

Symbol of XNOR Gate
The logic symbol of XNOR gate is shown in the following figure. Here, A and B are inputs and Y is the output.



Applications of Logic Gates:
Logic gates are the fundamental building blocks of all digital circuits and devices like computers. Here are some key digital devices in which logic gates are utilized to design their circuits:

- Computers
- Microprocessors
- Microcontrollers
- Digital and smart watches
- Smartphones, etc.

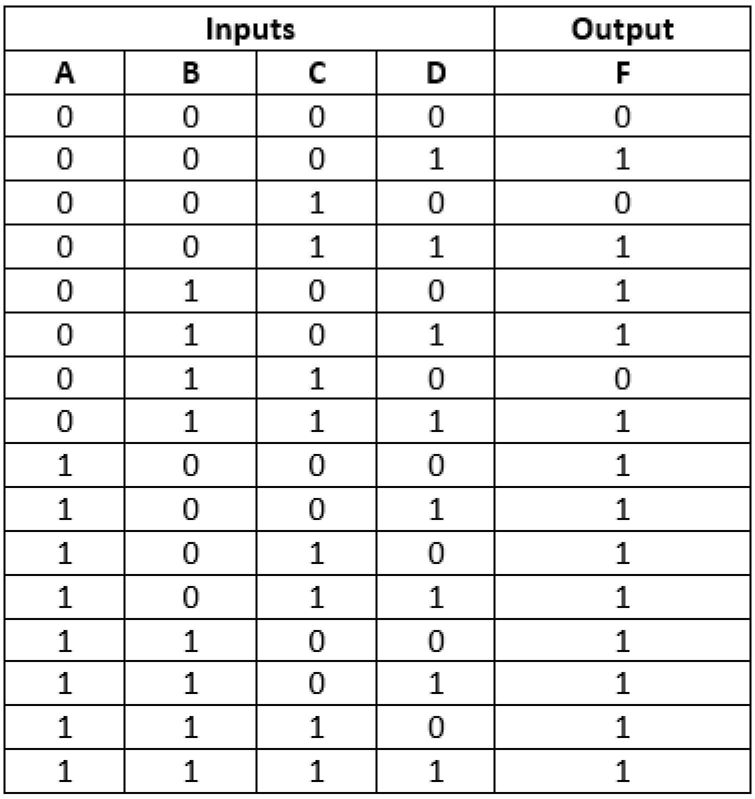
Boolean Functions:

The binary variables and logic operations are used in Boolean algebra. The algebraic expression is known as **Boolean Expression**, is used to describe the **Boolean Function**. The Boolean expression consists of the constant value 1 and 0, logical operation symbols, and binary variables.

Example 1: F=xy'z+p
The Boolean function **F=xy'z+p** in terms of four binary variables x, y, z, and p. This function will be equal to 1 when x=1, y=0, z=1 or p=1.

Example 2: F(A,B,C,D)=A+BC'+D

The output will be high when A=1 or BC'=1 or D=1 or all are set to 1. The truth table of the above example is given below. The 2^n is the number of rows in the truth table. The n defines the number of input variables. So the possible input combinations are $2^4=16$.



Inputs				Output
A	B	C	D	F
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Methods of simplification (K-map):

Minterm and Maxterm
Minterms and Maxterms are important parts of Boolean algebra. Minterm is the product of N distinct literals where each literal occurs exactly once. The output of the minterm functions is 1. Minterm is represented by m. Maxterm is the sum of N distinct literals where each literals occurs exactly once. The output of the maxterm functions is 0. Maxterm is represented by M.

What is Minterm?
Minterm is the product of various different literals in which each literal occurs exactly once. The output result of the minterm functions is 1. It is represented by m. To represent a function, we perform a sum of minterms also called the **Sum of Products (SOP)**.

Example of SOP:

A’B + AC + BC

Table for Two-Variable Minterm

Variable		Minterm	
A	B	Term	Representation

Variable		**Minterm**	
0	0	A'B'	m_0
0	1	A'B	m_1
1	0	AB'	m_2
1	1	AB	m_3

Minterms for Values
Minterms for values is the minterms obtained by the values of the Boolean variable.

Steps for Obtaining Minterms from Values
1. If the value of the Boolean variable is 1 then we will take the variable without complementing it.
2. If the value of the Boolean variable is 0 then we will take the variable by complementing it.

Minterm Example:
If there are four Boolean variables A, B, C, D with the values A = 1, B = 0, C = 0 and D = 1. Find the minterms for values given.

Solution:
Given the values of the Boolean variables:
A = 1, B = 0, C = 0 and D = 1
The required minterm is given by = AB'C'D
We complemented B and C as its value is 0.

What is Maxterm?
Maxterm is the sum of various different literals in which each literal occurs exactly once. The output result of the maxterm functions is 0. It is represented by M. To represent a function, we perform product of maxterms also called **Product of Sum (POS).**

Example of POS:

(A+B).(C+D)

Table for Two Variable Maxterm

Variable		**Maxterm**	
A	B	Term	Representation
0	0	A+B	M_0
0	1	A+B'	M_1

Variable		Maxterm	
1	0	A'+B	M_2
1	1	A'+B'	M_3

Maxterms for Values
Maxterms for values is the maxterms obtained by the values of the Boolean variable.

Steps for obtaining maxterms from values:
1. If the value of the Boolean variable is 0 then we will take the variable without complementing it.
2. If the value of the Boolean variable is 1 then we will take the variable by complementing it.

Maxterm Example:
If there are four Boolean variables A, B, C, D with the values A = 1, B = 0, C = 0 and D = 1. Find the maxterm for values given.

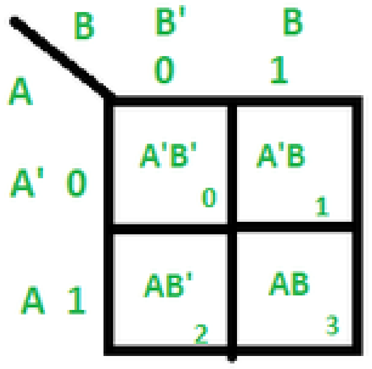
Solution:
Given the values of the Boolean variables:
A = 1, B = 0, C = 0 and D = 1
The required maxterm is given by = A'+ B + C + D'
We complemented A and D as its value is 1 and B and C are 0 therefore there are no change.

Minterm vs Maxterm

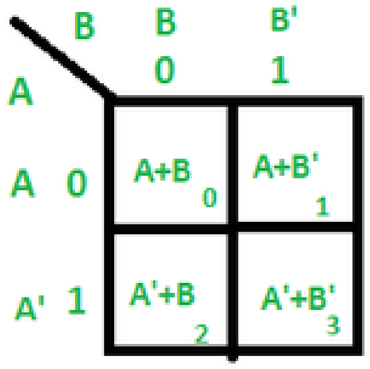
Minterm	Maxterm
Minterm is the term with the product of N literals occurring exactly once.	Maxterm is the term with the sum of N literals occurring exactly once.
It is represented by m.	It is represented by M.
It is logical AND of distinct literals.	It is logical OR of distinct literals.
The sum of minterms forms SOP (Sum of Product) functions.	The product of maxterms forms POS (Product of Sum) functions.
The output result of minterm function is 1.	The output result of maxterm function is 0.
It works on active high.	It works on active low.
Example: AB + A'B'	Example: (A+B).(A'+B')

Below mentioned is the image of **Two and Four Variables K-Map.**

Two Variable K-Maps

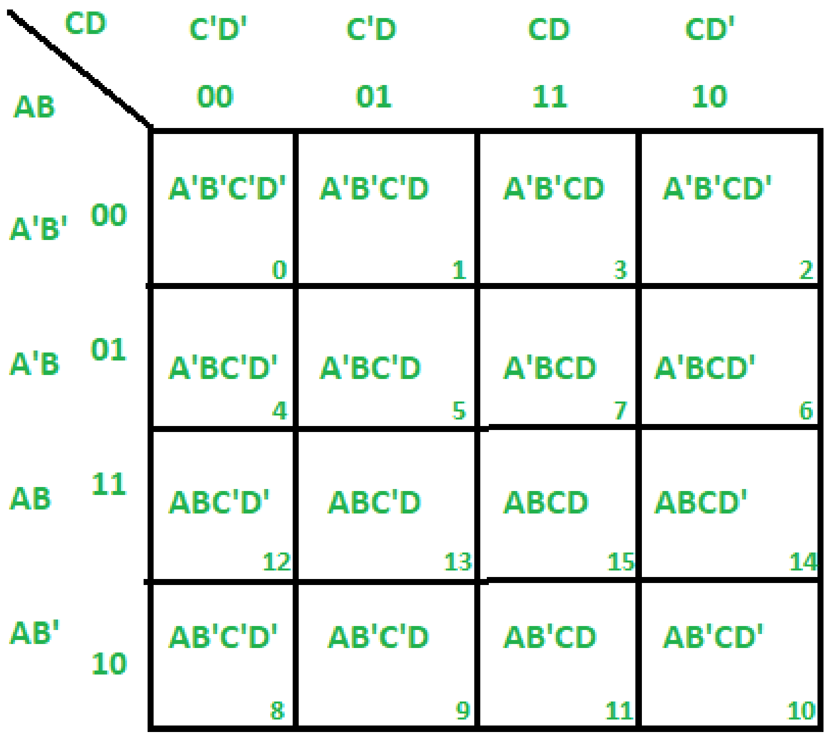


SOP(MINTERMS)

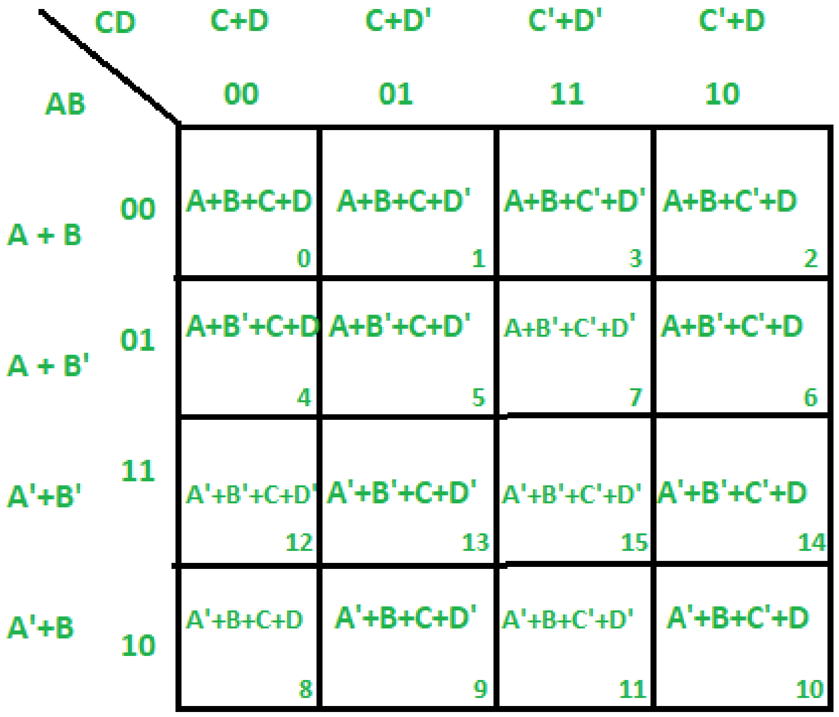


POS(MAXTERMS)

Four Variable K-Maps



SOP(MINTERMS)



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

POS(MAXTERMS)

Introduction of K-Map (Karnaugh-Map)

In many digital circuits and practical problems, we need to find expressions with minimum variables. We can minimize Boolean expressions of 3, 4 variables very easily using K-map without using any Boolean algebra theorems.

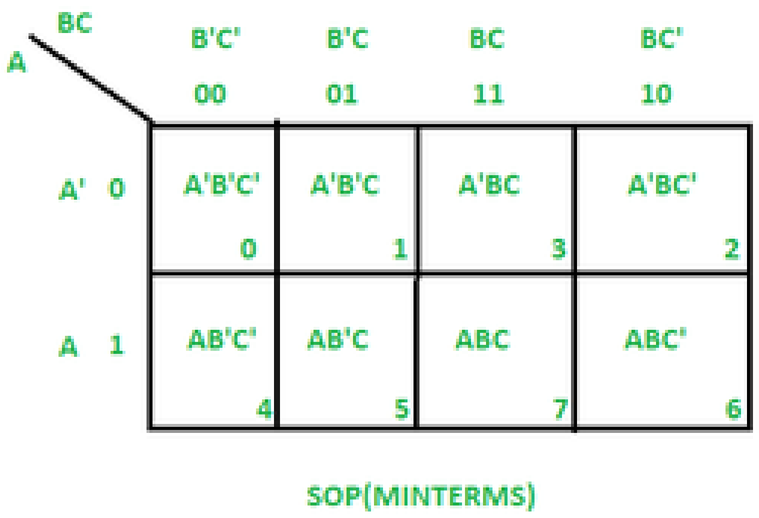
K-map can take two forms:
1. Sum of product (SOP)
2. Product of Sum (POS)

According to the need of problem. K-map is a table-like representation, but it gives more information than the TABLE. We fill a grid of the K-map with 0's and 1's then solve it by making groups.

Steps to Solve Expression using K-map
1. Select the K-map according to the number of variables.
2. Identify minterms or maxterms as given in the problem.
3. For SOP put 1's in blocks of K-map respective to the minterms (0's elsewhere).
4. For POS put 0's in blocks of K-map respective to the max terms (1's elsewhere).
5. Make rectangular groups containing total terms in power of two like 2,4,8 .. (except 1) and try to cover as many elements as you can in one group.
6. From the groups made in step 5 find the product terms and sum them up for SOP form.

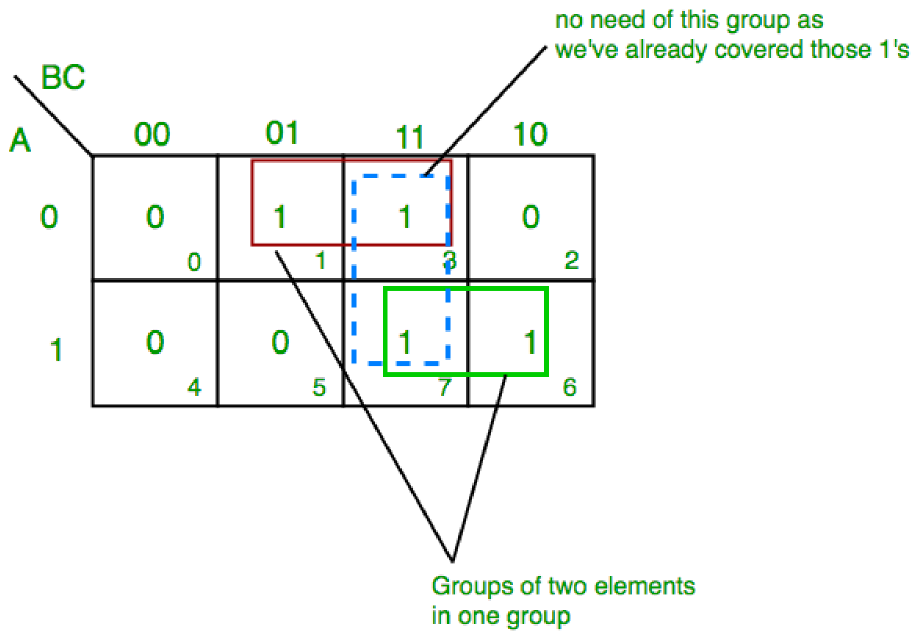
SOP FORM:

1. K-map of 3 variables



8 Blocks = 1
4 Blocks = 1 variable term
2 Blocks = 2 variable term
1 Block = 3 variable term

Z(A,B,C)=Σm(1,3,6,7)

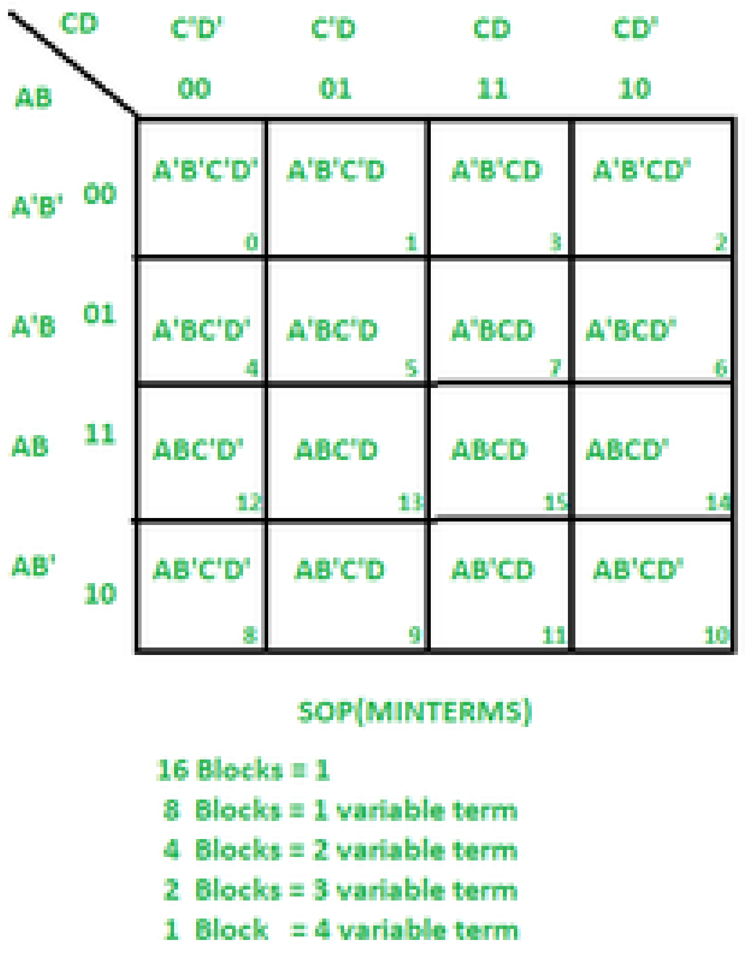

no need of this group as we've already covered those 1's
BC
A
00
01
11
10
0
1
Groups of two elements in one group

From **red** group we get product term-
A’C

From **green** group we get product term-
AB

Summing these product terms we get- **Final expression Z=(A’C+AB)**

2. K-map for 4 variables



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

SOP(MINTERMS)

16 Blocks	= 1
8 Blocks	= 1 variable term
4 Blocks	= 2 variable term
2 Blocks	= 3 variable term
1 Block	= 4 variable term

F(P,Q,R,S) = Σm(0,2,5,7,8,10,13,15)

From **red** group we get product term-
QS

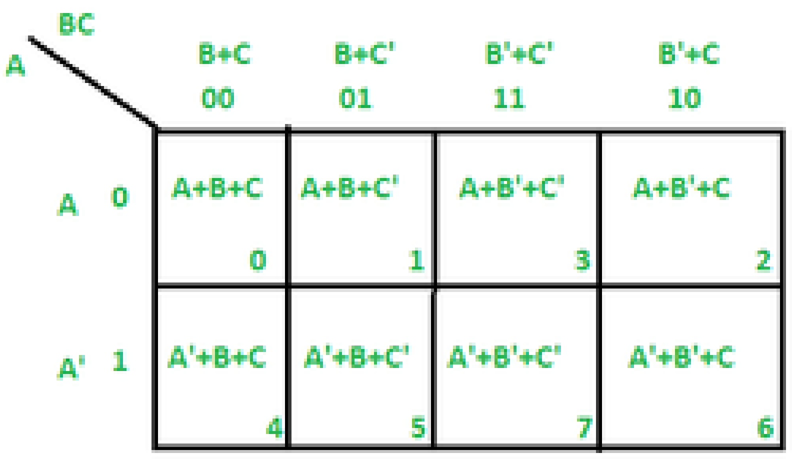
From **green** group we get product term-
Q'S'

Summing these product terms we get- **Final expression F = (QS+Q'S')**.

POS FORM

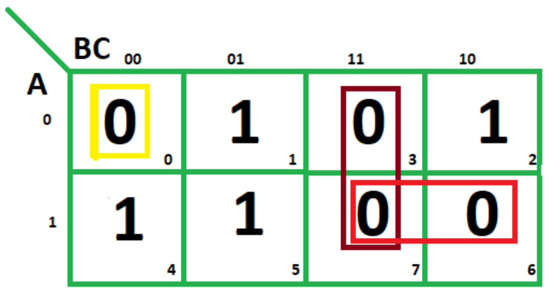
1. K-map of 3 variables

F(A,B,C)=∏M(0,3,6,7)



POS (MAXTERMS)

8 Blocks = 0
4 Blocks = 1 variable term
2 Blocks = 2 variable term
1 Block = 3 variable term



From **red** group we find terms

AB

Taking complement of these two

A'B'

Now **sum** up them

(A'+B')

From **brown** group we find terms

BC

Taking complement of these two terms

B’C’

Now sum up them

(B’+C’)

From **yellow** group we find terms

A'B'C'

Taking complement of these two

ABC

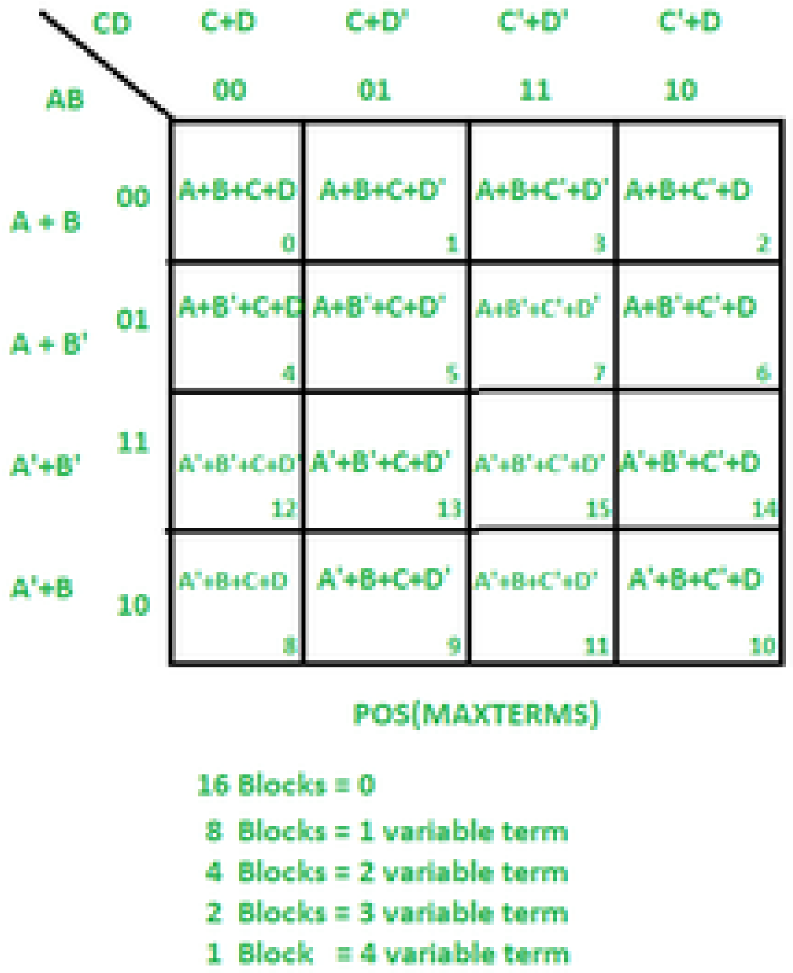
Now **sum** up them

(A+B+C)

We will take product of these three terms: **Final expression-**

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

2. K-map of 4 variables



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

POS(MAXTERMS)

16 Blocks = 0

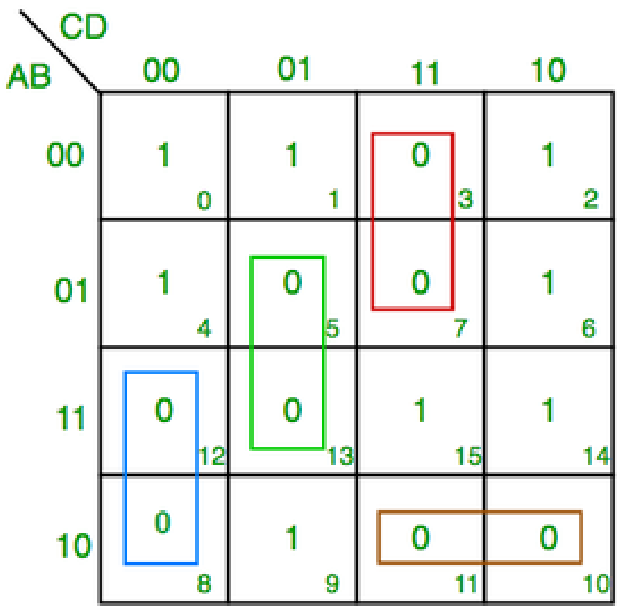
8 Blocks = 1 variable term

4 Blocks = 2 variable term

2 Blocks = 3 variable term

1 Block = 4 variable term

F(A,B,C,D)= ∏M(3,5,7,8,10,11,12,13)



From **green** group we find terms
C’DB

Taking their complement and summing them
(C+D’+B’)

From **red** group we find terms
CDA’

Taking their complement and summing them
(C’+D’+A)

From **blue** group we find terms
AC’D’

Taking their complement and summing them
(A’+C+D)

From **brown** group we find terms
AB’C

Taking their complement and summing them
(A’+B+C’)

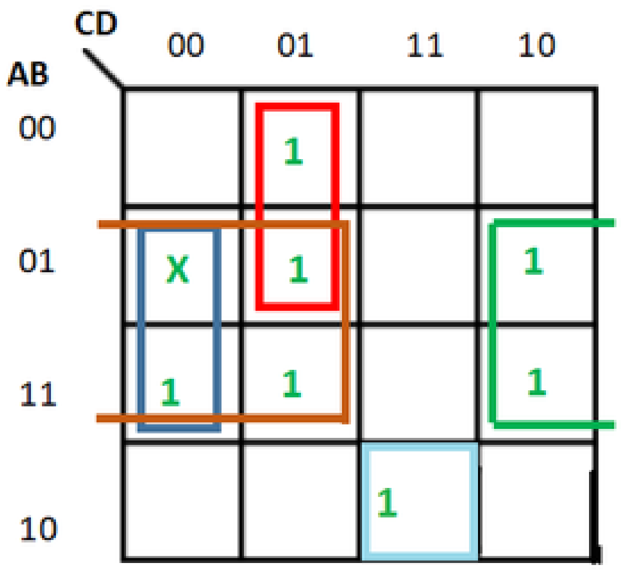
Finally we express these as product-
F=(C+D’+B’).(C’+D’+A).(A’+C+D).(A’+B+C’)

Don’t Care (X) Conditions in K-Maps:

One of the very significant and useful concepts in simplifying the output expression using K-Map is the concept of "Don't Care". The "Don't Care" conditions allow us to replace the empty cell of a K-Map to form a grouping of the variables which is larger than that of forming groups without don't care. While forming groups of cells, we can consider a "Don't Care" cell as 1 or 0 or we can also ignore that cell. Therefore, the "Don't Care" condition can help us to form a larger group of cells.
A Don't Care cell can be represented by a cross (X) or minus (-) or phi (Φ) in K-Maps representing an invalid combination. For example, in the Excess-3 code system, the states 0000, 0001, 0010, 1101, 1110, and 1111 are invalid or unspecified. These states are called don't cares.
A standard SOP function having don't cares can be converted into a POS expression by keeping don't cares as they are, and writing the missing minterms of the SOP form as the maxterm of POS form. Similarly, a POS function having don't cares can be converted to SOP form keeping the don't cares as they are and writing the missing maxterms of the POS expression as the minterms of SOP expression.

Example-1:
Minimise the following function in SOP minimal form using K-Maps:
f = Σm(1, 5, 6, 11, 12, 13, 14) + d(4)

Explanation:
The SOP K-map for the given expression is:

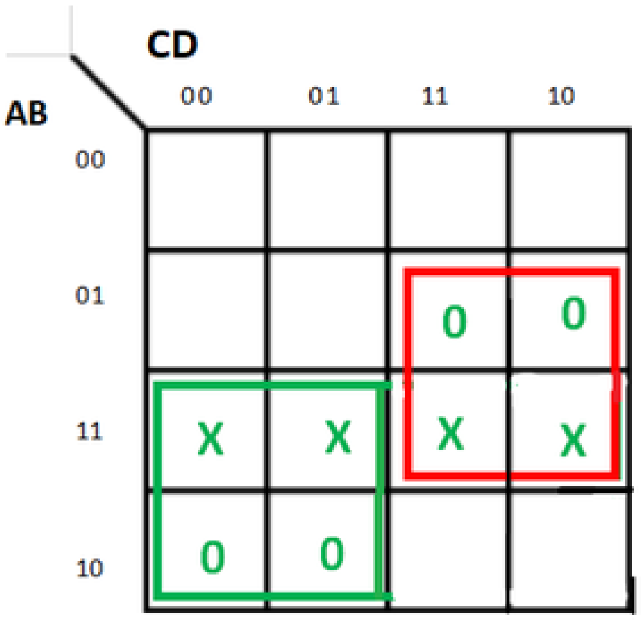


Therefore, SOP minimal is,
f = BC' + BD' + A'C'D + AB'CD

Example-2:
Minimise the following function in POS minimal form using K-Maps:
F(A, B, C, D) = Σm(0, 1, 2, 3, 4, 5) + d(10, 11, 12, 13, 14, 15)

Explanation:
Writing the given expression in POS form:
F(A, B, C, D) = M(6, 7, 8, 9) + d(12, 13, 14, 15)

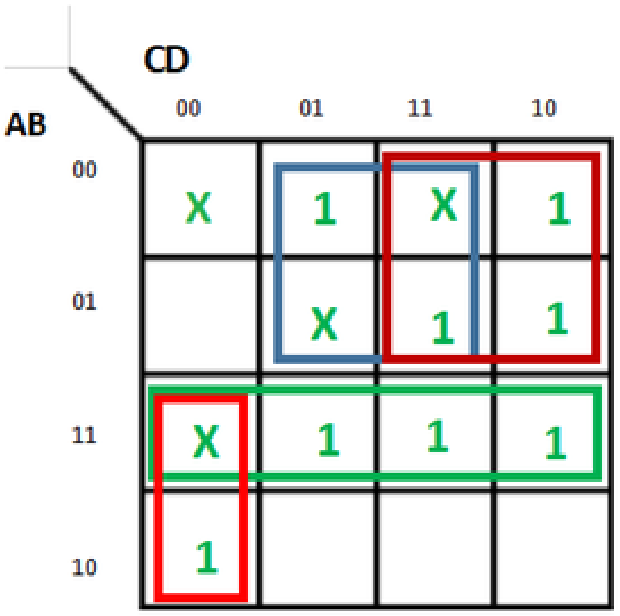
The POS K-map for the given expression is:



Therefore, POS minimal is,
F = (A'+ C)(B' + C')

Example-3:
Minimise the following function in SOP minimal form using K-Maps:
F(A, B, C, D) = Σm(1, 2, 6, 7, 8, 13, 14, 15) + d(0, 3, 5, 12)

Explanation:
The SOP K-map for the given expression is:



Therefore,
f = AC'D' + A'D + A'C + AB

Significance of "Don't Care" Conditions:
Don't Care conditions have the following significance in designing of the digital circuits.

1. **Simplification of the output:** These conditions denote inputs that are invalid for a given digital circuit. Thus, they can used to further simplify the Boolean output expression of a digital circuit.
2. **Reduction in number of gates required:** Simplification of the expression reduces the number of gates to be used for implementing the given expression. Therefore, don't cares make the digital circuit design more economical.
3. **Reduced Power Consumption:** While grouping the terms along with don't cares reduces switching of the states. This decreases the memory space that is required to represent a given digital circuit which in turn results in less power consumption.
4. **Represent Invalid States in Code Converters:** These are used in code converters. For example- In design of 4-bit BCD-to-EXCESS-3 code converter, the input combinations 1010, 1011, 1100, 1101, 1110, and 1111 are don't cares.
5. **Prevention of Hazards in Digital Circuits:** Don't cares also prevent hazards in digital systems.

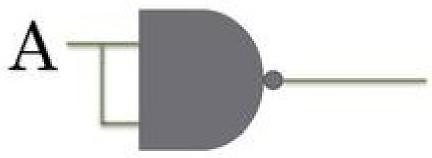
Implementing Any Circuit Using NAND Gate Only:

A universal gate is such a gate that we can implement any Boolean function, no matter how complex, using a circuit that consists of **only** that particular universal gate. The NAND & NOR gates are the most commonly encountered universal gates in digital logic.
We will take a look at how to convert any circuit into a circuit that consists only of NAND gates. Since the NAND gate is a universal gate, we can convert any circuit into a circuit consisting only of NAND gates. We first start by showing how other gates (AND, OR, Inverter) can be implemented using only NAND gates, then we use this knowledge to discuss how to convert any circuit into only a NAND circuit.

Objective:
Given a circuit, our task is to implement a circuit that is equivalent to the given circuit and consists of only **NAND** gates.

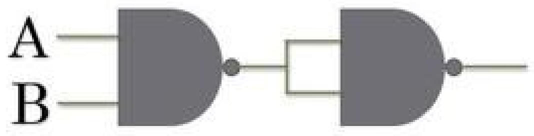
Before we get to how to convert any circuit to a NAND-only circuit, we will take a look at how to implement Complement, AND & OR operation using NAND gates. We also will need to take a look at how to implement NAND operation using OR gate.

COMPLEMENT Using NAND



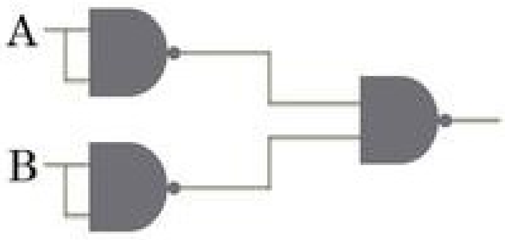
(AA)' = A'

AND Using NAND



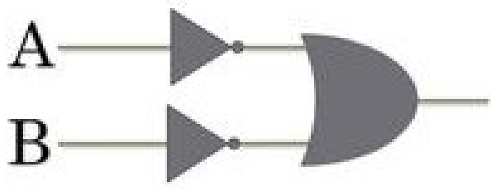
This is quite straightforward, we wish to obtain AB but the NAND gate gives an output (AB)’ so we complement the output of the NAND gate using another NAND gate to obtain ((AB)’)’ which is AB.

OR Using NAND



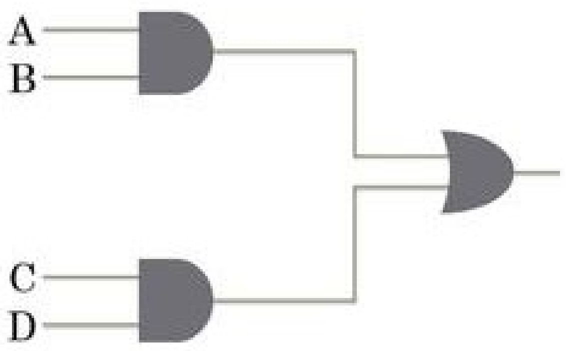
We first complement the inputs A and B. Then we perform the NAND operation on these complemented inputs. We get (A’B’)’.
Using De-Morgan’s law we can show that (A’B’)’ = A + B.

NAND using OR & Inverter

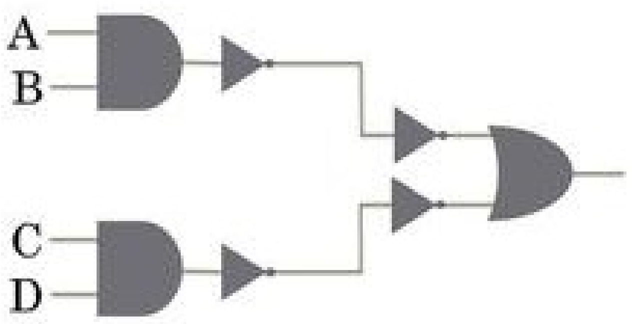


To implement NAND operation using OR gate, we first complement the inputs and then perform OR on the complemented inputs.
We get A’ + B’.
A’ + B’ is equivalent to (AB)’ which can be shown to be true by using De-Morgan’s law.

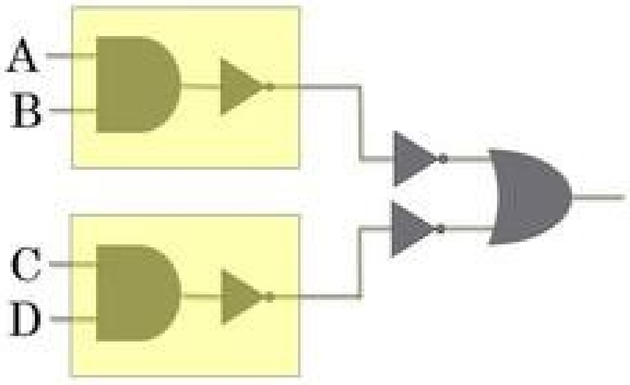
Procedure for Conversion
Consider the following circuit.
We have:



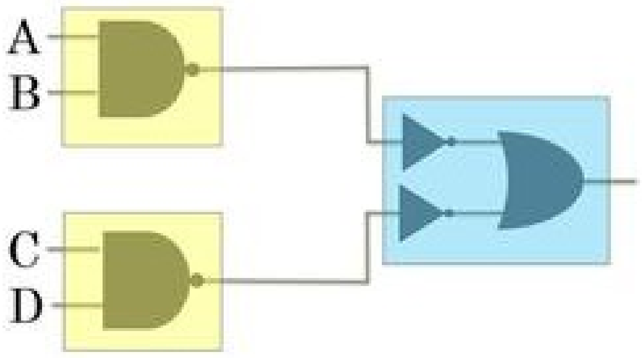
We have to somehow re-implement this circuit using only NAND gates.
Suppose what happens if we insert two inverters between each AND gate and OR gate.



This circuit is completely equivalent to the original one as the output of each AND gate is being complemented twice before the signal reaches the OR gate. These inverters will play a key role in the process of conversion.
Now take a look at the highlighted areas.

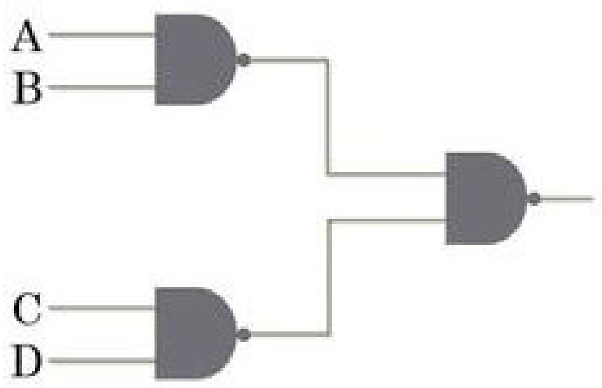


The gates in each highlighted area can be easily combined into one NAND gate since it is basically just an AND gate followed by an inverter.
We get:



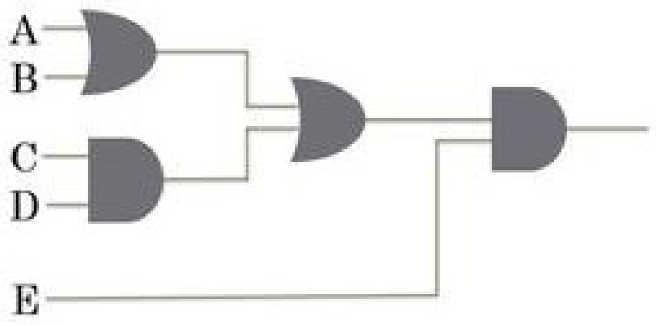
Now we only have to take care of the blue highlighted area. Recall the implementation of a NAND operation using OR & inverter gates we have covered above, the gates in the blue area implement a NAND operation. Therefore we can just replace all the gates in the blue area with a single NAND gate!

Our Final Result:



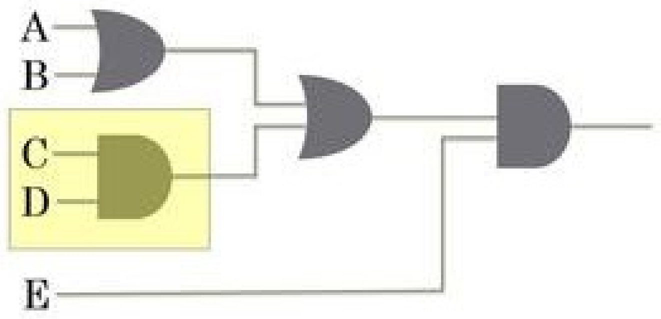
You can verify for yourself that this circuit implements the function AB + CD, same as that of the original circuit.

Another Example:
Original Circuit:

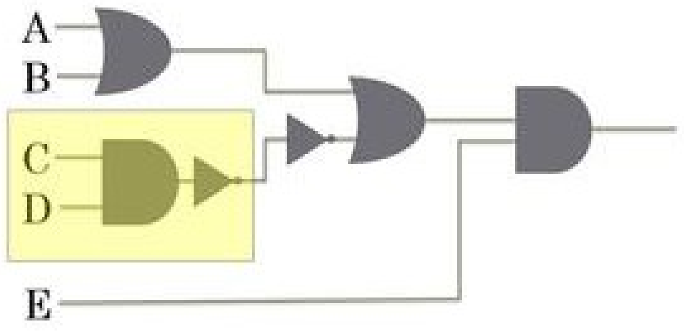


(A+B + CD)E

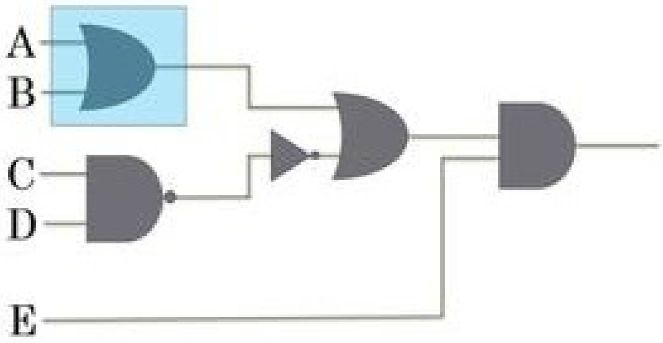
Method 1;



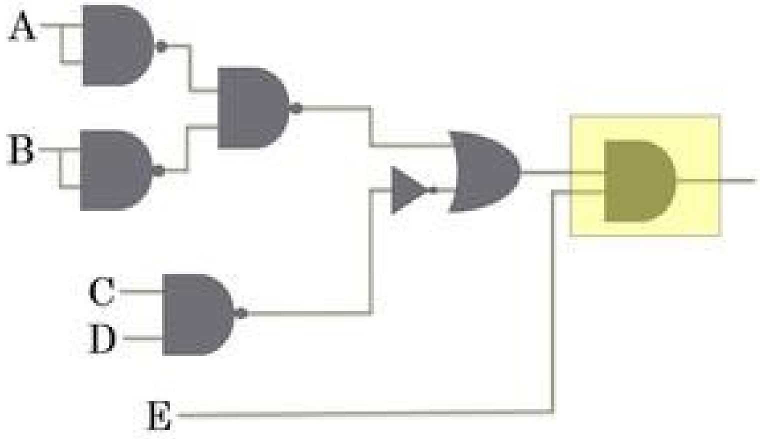
First, we start by replacing the first AND gate (highlighted yellow) with a NAND gate. To do this we insert two inverters after this AND gate.



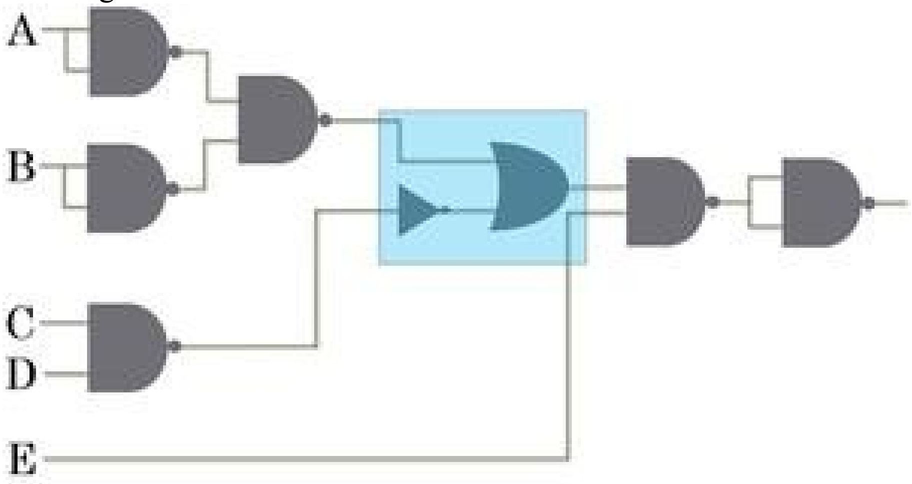
Remember that this circuit is the same as two complement operations resulting in the original signal. The AND gate and the inverter that follows it can be combined into a single NAND gate (see yellow highlighted area).



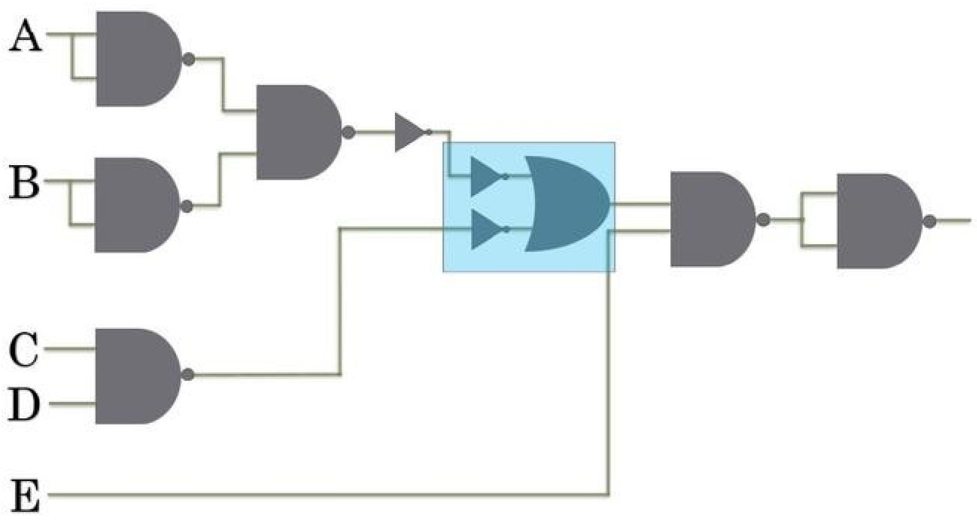
Next, we replace the OR gate in the blue highlighted area with NAND gates. We have seen how to implement OR operation using NAND gates, we put that knowledge to use now.



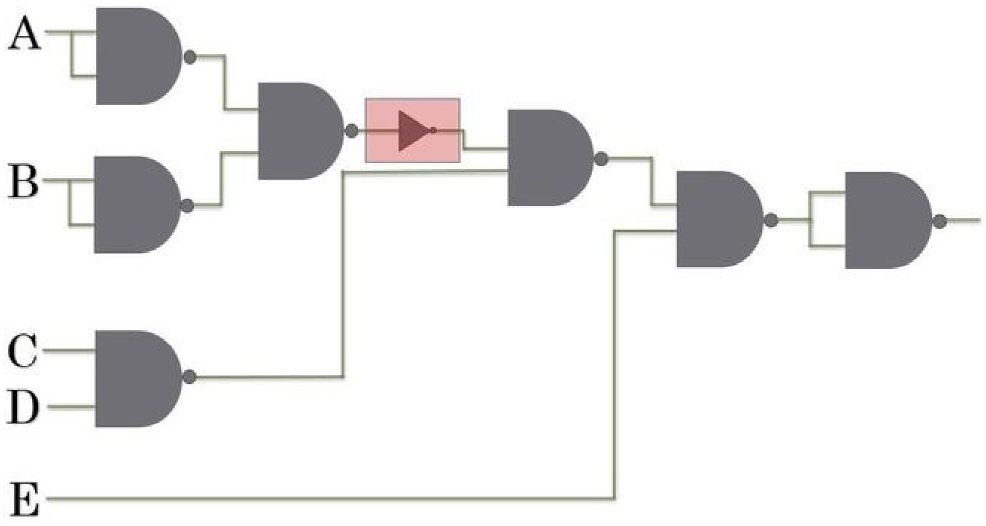
We then proceed to replace the last AND gate (highlighted yellow) with NAND gates. Just like we replaced the OR gate in the previous step, we replace the AND gate with its equivalent NAND gate circuit.



We now are left with an OR Gate and an Inverter Gate. A NAND gate can be implemented by an OR gate with complemented inputs. Here we have only one complemented input to the OR gate. To meet the condition that both the inputs are complemented, we insert two inverters between the highlighted OR gate and the preceding NAND gate.
We get:

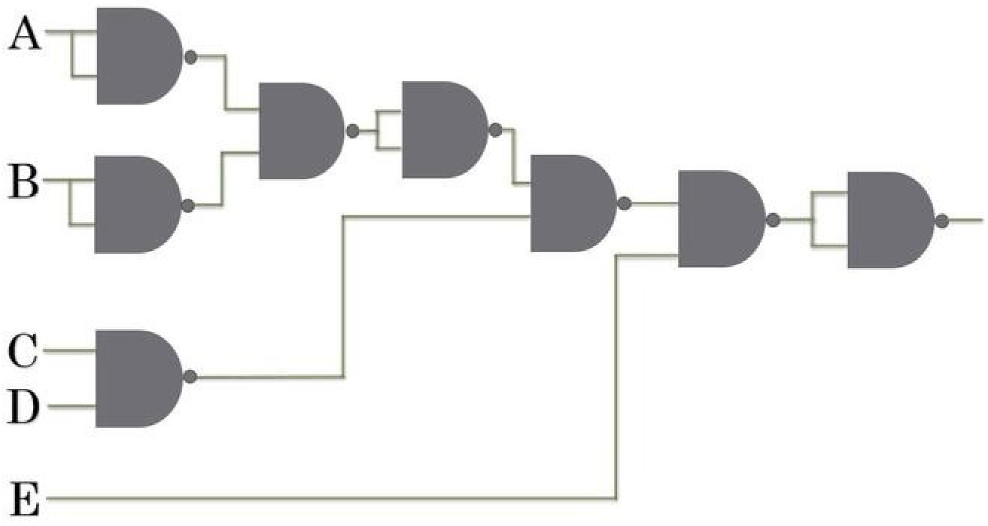


Now both the inputs to the OR gate are complemented. The gates in the blue area represent a NAND operation so we can replace them with a NAND gate.



We are now only left with an Inverter. An Inverter gate is basically a one-input NAND gate. We make the necessary replacement and obtain our final circuit.

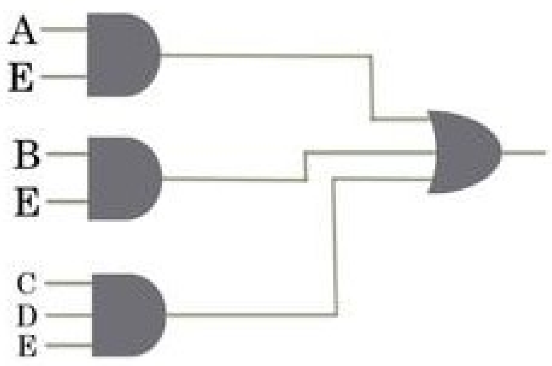
Our Final NAND gate-only circuit:



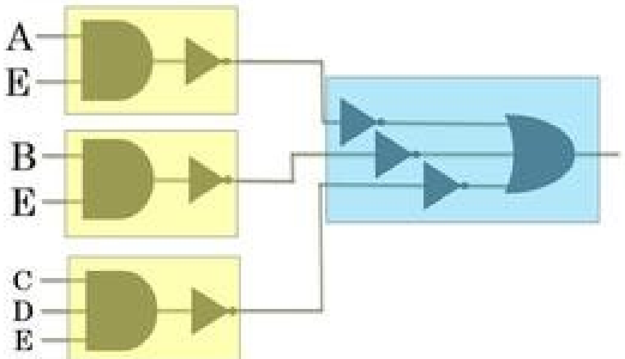
You can verify that this circuit implements: (A+B + CD)E

Method 2 (SOP Method):

The original circuit implements the Boolean function: (A+B+CD)E
We first manipulate this Boolean equation so that it is in the Sum of Products (SOP).
In this case, we simply multiply each term in the parenthesis by E.
We get: AE + BE + CDE.
Since this Boolean equation is now in SOP form, the circuit for this equation will be in a standard two-level implementation, meaning there will be a series of AND gates followed by a single OR gate.
We draw the circuit:

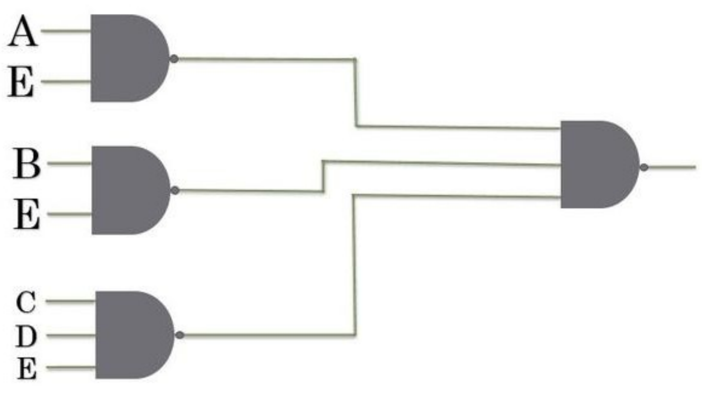


Just like in example 1, we insert two inverter gates between each AND gate and OR gate.



You can probably now figure out what we will do next. We combine the gates in the Yellow areas into one single NAND gate. Also, we know that an OR gate with complemented inputs implements a NAND operation. So we replace the gates in the blue area with a NAND gate.

Our final circuit:



We can conclude that whenever a circuit has a series of AND gates at the first level followed by a single OR gate, we can blindly replace each gate with a NAND gate and the circuit will still implement the same Boolean function.
In this case Method 2, took a lot less effort than Method 1, one might wonder why even bother learning Method 1 when Method 2 seems easier? The answer is that Method 2 requires that the Boolean equation must be represented in Sum of Products form. In this case, it was easy to convert our equation into SOP from, but this is not always the case.
Suppose you have a circuit that implements the Boolean function:

$$(A+B)(C+D)(E+F)(G+H)$$

Converting this to SOP form will be complicated and better with Method 1 in this case.

Implementing Any Circuit Using NOR Gate Only:

We can use similar technique to implement any circuit using NOR gate only.