Digital Logic Fundamentals

Semester - 1

Unit-1: Introduction

Digital Systems – Binary Numbers – Conversions – Types – Codes – Storage and Registers – Binary Logic – Boolean Algebra Theorems and Properties – Functions – Canonical andStandard Forms – Other Logic Operations – Digital Logic Gates

Digital System

A Digital system is an interconnection of digital modules and it is a system that manipulatesdiscrete elements of information that is represented internally in the binary form. Now a day's digital systems are used in wide variety of industrial and consumer products such as automated industrial machinery, pocket calculators, microprocessors, digital computers, digital watches, TV games and signal processing and so on.

Number System-Binary, Octal, Decimal, Hexadecimal - Conversion from one system toanother number system.

Number :

The way of quantifying anything , represented through various combination of symbolsis called number.

Digit :

The various symbols representing a single number in any number system is called digit.

E.g. Decimal number system (Arabic numerals): Digits: 0,1,2,3,4,5,6,7,8,9.

Radix / Base (r) :

The maximum number of different digits of any number system. E.g Decimal NS, r =10

Number system:

The properly structured number formation is called Number system. In number systemthere are different symbols and each symbol has an absolute value and also has place value.

In general a number in a system having base or radix ' r ' can be written asNumber various

combination digits according to position

Nr = [Integer part . Fractional part]

↑ Radix point

$= d_n\, d_{n-1} \ldots d_1 d_0 \,.\, d_{-1} d_{-2} \ldots d_{-m}$ The value,

$N_{10} = \mathbf{d_n x\, r^n} + \mathbf{d_{n-1} x\, r^{n-1}} + \ldots + \mathbf{d_1 x\, r^1} + \mathbf{d_0 x\, r^0} + \mathbf{d_{-1} x\, r^{-1}} + \mathbf{d_{-2} x\, r^{-2}} + \ldots + \mathbf{d_{-m} x\, r^{-m}}$

* The right most digit of any number is called Least Significant Digit

* The lef most digit of any number is called Most Significant Digit

TYPES OF NUMBER SYSTEM:-

There are four types of number systems. They are

1. Decimal number system
2. Binary number system
3. Octal number system
4. Hexadecimal number system

DECIMAL NUMBER SYSTEM:-

The decimal number system contain ten unique symbols 0,1,2,3,4,5,6,7,8 and 9.

- In decimal system 10 symbols are involved, so the base or radix is 10.
- It is a positional weighted system.

$$(d_n \times 10^n) + (d_{n-1} \times 10^{n-1}) + (d_{n-2} \times 10^{n-2}) + \ldots + (d_0 \times 10^0) + (d_{-1} \times 10^{-1}) + (d_{-2} \times 10^{-2}) + \ldots + (d_{-m} \times 10^{-m})$$

For example:-

$$9256.26 = 9 \times 1000 + 2 \times 100 + 5 \times 10 + 6 \times 1 + 2 \times (1/10) + 6 \times (1/100)$$

$$= 9 \times 10^3 + 2 \times 10^2 + 5 \times 10^1 + 6 \times 10^0 + 2 \times 10^{-1} + 6 \times 10^{-2}$$

BINARY NUMBER SYSTEM:-

- The binary number system is a positional weighted system.
- The base or radix of this number system is 2.
- It has two independent symbols, The symbols used are 0 and 1.
- A binary digit is called a bit

$$(d_n \times 2^n) + (d_{n-1} \times 2^{n-1}) + (d_{n-2} \times 2^{n-2}) + \ldots + (d_0 \times 2^0) + (d_{-1} \times 2^{-1}) + (d_{-2} \times 2^{-2}) + \ldots + (d_{-k} \times 2^{-k})$$

OCTAL NUMBER SYSTEM:-

- It is also a positional weighted system.
- Its base or radix is 8.
- It has 8 independent symbols 0,1,2,3,4,5,6 and 7.
- Its base 8 = 2^3 , every 3- bit group of binary can be represented by an octal digit.

HEXADECIMAL NUMBER SYSTEM:-

- The hexadecimal number system is a positional weighted system.
- The base or radix of this number system is 16.
- The symbols used are 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E and F
- The base 16 = 24 , every 4 – bit group of binary can be represented by an hexadecimaldigit.

CONVERSION FROM ONE NUMBER SYSTEM TO ANOTHER :-

1. BINARY NUMBER SYSTEM:-

(a) Binary to decimal conversion:- In this method, each binary digit of the number is multiplied by its positional weight and the product terms are added to obtain decimalnumber.

$$(111.101)_2 = (1 \times 2^2) + (1 \times 2^1) + (1 \times 2^0) + (1 \times 2^{-1}) + (0 \times 2^{-2}) + (1 \times 2^{-3})$$

$$= 4 + 2 + 1 + 0.5 + 0 + 0.125$$

$$= (7.625)_{10}$$

(b) Binary to Octal conversion:- For conversion binary to octal the binary numbers are divided into groups of 3 bits each, starting at the binary point and proceeding towards leftand right.

(i) Convert $(101111010110.110110011)_2$ into octal.

Solution :

Group of 3 bits are	101	111	010	110	.	110	110	011
Convert each group into octal =	5	7	2	6	.	6	6	3

The result is $(5726.663)_8$

(c) Binary to Hexadecimal conversion:- For conversion binary to hexadecimal number the binary numbers starting from the binary point, groups are made of 4 bits each, on either side of the binary point.

(ii) Convert $(01011111011.011111)_2$ into hexadecimal.

Solution:

Given Binary number		010	1111	1011	.	0111	11
Group of 3 bits are	=	0010	1111	1011	.	0111	1100
Convert each group into octal =		2	F	B	.	7	C

The result is $(2FB.7C)_{16}$

2. DECIMAL NUMBER SYSTEM:-

(a) Decimal to binary conversion:- In the conversion the integer number are converted to thedesired base using successive division by the base or radix.

For example: (i) Convert $(52)_{10}$ into binary.

(ii) Convert $(105.15)_{10}$ into binary.

Solution:

Integer part		Fraction part
2 | 105		0.15 x 2 = 0.30
2 | 52	— 1	0.30 x 2 = 0.60
2 | 26	— 0	0.60 x 2 = 1.20
2 | 13	— 0	0.20 x 2 = 0.40
2 | 6	— 1	0.40 x 2 = 0.80
2 | 3	— 0	0.80 x 2 = 1.60
2 | 1	— 1	
0	— 1	

Result of $(105.15)_{10}$ is $(1101001.001001)_2$

Decimal to octal conversion:- To convert the given decimal integer number to octal,successively divide the given number by 8 till the quotient is 0.

(i) Convert $(378.93)_{10}$ into octal.

Solution:

8 | 378		0.93 x 8 = 7.44
8 | 47	— 2	0.44 x 8 = 3.52
8 | 5	— 7	0.52 x 8 = 4.16
0	— 5	0.16 x 8 = 1.28

Result of $(378.93)_{10}$ is $(572.7341)_8$

(b)Decimal to hexadecimal conversion:-

(i) Convert $(2598.675)_{10}$ into hexadecimal.

Solution:

	Remainder Decimal	Hex		Hex
16 | 2598			0.675 x 16 = 10.8	A
16 | 162	— 6	6	0.800 x 16 = 12.8	C
16 | 10	— 2	2	0.800 x 16 = 12.8	C
0	— 10	A	0.800 x 16 = 12.8	C

Result of $(2598.675)_{10}$ is **$(A26.ACCC)_{16}$**

3. OCTAL NUMBER SYSTEM:-

(a) Octal to binary conversion:- To convert a given a octal number to binary, replace eachoctal digit by its 3-bit binary equivalent.

For example:

Convert $(367.52)_8$ into binary.

Solution:

Given Octal number is	3	6	7	.	5	2	
Convert each group octal to binary	= 011	110	111	.	101	010	

Result of $(367.52)_8$ is **$(011110111.101010)_2$**

(b) Octal to decimal conversion:- For conversion octal to decimal number, multiply each digitin the octal number by the weight of its position and add all the product terms

For example: -

Convert $(4057.06)_8$ to decimal

Solution:

$$(4057.06)_8 = 4 \times 8^3 + 0 \times 8^2 + 5 \times 8^1 + 7 \times 8^0 + 0 \times 8^{-1} + 6 \times 8^{-2}$$
$$= 2048 + 0 + 40 + 7 + 0 + 0.0937$$
$$= (2095.0937)_{10}$$

Result is **$(2095.0937)_{10}$**

(c) Octal to hexadecimal conversion:- For conversion of octal to Hexadecimal, first convertthe given octal number to binary and then binary number to hexadecimal

(4) HEXADECIMAL NUMBER SYSTEM :- (a)Hexadecimal to binary conversion:- For conversionof hexadecimal to binary, replace hexadecimal digit by its 4 bit binary group

Convert $(3A9E.B0D)_{16}$ into binary.

Solution:

Given Hexadecimal number is	3	A	9	E	.	B	0	D	
Convert each hexadecimal digit to 4 bit binary	= 0011	1010	1001	1110	.	1011	0000	1101	

Result of $(3A9E.B0D)_8$ is **$(0011101010011110.101100001101)_2$**

(b)Hexadecimal to decimal conversion:- For conversion of hexadecimal to decimal, multiplyeach digit in the hexadecimal number by its position weight and add all those product terms.

Convert $(A0F9.0EB)_{16}$ to decimal

Solution:

$(A0F9.0EB)_{16} = (10 \times 16^3)+(0 \times 16^2)+(15 \times 16^1)+(9 \times 16^0)+(0 \times 16^{-1})+(14 \times 16^{-2})+(11 \times 16^{-3})$

$= 40960 + 0 + 240 + 9 + 0 + 0.0546 + 0.0026$

$= (41209.0572)_{10}$

Result is $(41209.0572)_{10}$

((c) Hexadecimal to Octal conversion:- For conversion of hexadecimal to octal, first convertthe given hexadecimal number to binary and then binary number to octal

Arithmetic Operation-Addition, Subtraction, Multiplication, Division, 1's & 2'scomplement of Binary numbers& Subtraction using complements method

1. BINARY ADDITION:-

The binary addition rules are as follows carry carry

0 + 0 = 0 ; 0 + 1 = 1 ; 1 + 0 = 1 ; 1 + 1 = 1 0 , SUM 1+1+1 = 1 1 SUM

Add $(100101)_2$ and $(1101111)_2$.
Solution :-

```
  100101
+ 1101111
 10010100
```

Result is $(10010100)_2$

2. BINARY SUBTRACTION:-

The binary subtraction rules are as follows

0 - 0 = 0 ; 1 - 1 = 0 ; 1 - 0 = 1 ; 0 - 1 = 1 1 , with a borrow of 1

Substract $(111.111)_2$ from $(1010.01)_2$.
Solution :-

```
  1010.010
-  111.111
  0010.011
```

Result is $(0010.011)_2$

3. BINARY MULTIPLICATION:-

The binary multiplication rules are as follows 0 x 0 = 0 ; 1 x

1 = 1 ; 1 x 0 = 0 ; 0 x 1 = 0

Multiply $(1101)_2$ by $(110)_2$.
Solution :-

```
    1 1 0 1
x     1 1 0
    0 0 0 0
  1 1 0 1
+ 1 1 0 1
1 0 0 1 1 1 0
```

Result is **$(1001110)_2$**

4. BINARY DIVISION:-

The binary division is very simple and similar to decimal number system.So we have only 2

rules $0 \div 1 = 0$ $1 \div 1 = 1$

```
110 ) 101101 ( 111.1
  - 110
    1010
     110
     1001
      110
      110
      110
      000
```

Result is **$(111.1)_2$**

1's COMPLEMENT REPRESENTATION :-

The 1's complement of a binary number is obtained by changing each 0 to 1 and each 1 to0.

Find $(1100)_2$ 1's complement.

Solution :-

Given	1	1	0	0
1's complement is	0	0	1	1

Result is **$(0011)_2$**

2's COMPLEMENT REPRESENTATION :-

The 2's complement of a binary number is a binary number which is obtained by adding 1 tothe 1's complement of a number.

2's complement = 1's complement + 1

Find $(1010)_2$ 2's complement.

Solution :-

Given		1	0	1	0
1's complement is		0	1	0	1
	+				1
2's complement		0	1	1	0

Result is **$(0110)_2$**

SIGNED NUMBER :-

In sign – magnitude form, additional bit called the sign bit is placed in front of the number.If the sign bit is 0, the number is positive. If it is a 1, the number is negative.

0 1 0 1 0 0 1 = +41
↑
Sign bit

1 1 0 1 0 0 1 = -41
↑
Sign bit

SUBSTRACTION USING COMPLEMENT METHOD :

1's COMPLEMENT:-

In 1's complement subtraction, add the 1's complement of subtrahend to the minuend. If there is a carry out, then the carry is added to the LSB. This is called end around carry. If theMSB is 0, the result is positive. If the MSB is 1, the result is negative and is in its 1's complement form. Then take its 1's complement to get the magnitude in binary.

Subtract $(10000)_2$ from $(11010)_2$ using 1's complement.

Solution:-

```
  11010                    11010                      =  26
- 10000      =>        + 01111  (1's complement)      = - 16
          Carry  →     1 01001                          + 10
                     +        1
                       01010   = +10
```

Result is **+10**

2's COMPLEMENT:-

In 2's complement subtraction, add the 2's complement of subtrahend to the minuend. If there is a carry out, ignore it. If the MSB is 0, the result is positive. If the MSB is 1, the resultis negative and is in its 2's complement form. Then take its 2's complement to get the magnitude in binary.

Subtract $(1010100)_2$ from $(1010100)_2$ using 2's complement.

Solution:-

```
  1010100                  1010100                    =   84
- 1010100      =>        + 0101100  (2's complement)  = - 84
                         1 0000000 ( Ignore the carry)     0
                     =           0 (result = 0)
```

Hence MSB is 0. The answer is positive. So it is +0000000 = **0**

Digital Code & its Types

DIGITAL CODES:-

In practice the digital electronics requires to handle data which may be numeric, alphabets and special characters. This requires the conversion of the incoming data into binary formatbefore it can be processed. There is various possible ways of doing this and this process is called encoding. To achieve the reverse of it, we use decoders.

WEIGHTED AND NON-WEIGHTED CODES

There are two types of binary codes

1) **Weighted binary codes** : In weighted codes, for each position (or bit) ,there is specific weight attached. For example, in binary number, each bit is assigned particular weight 2^n where 'n' is the bit number for n = 0,1,2,3,4 the weights are 1,2,4,8,16 respectively. Example
:- BCD

2) Non- weighted binary codes:

Non-weighted codes are codes which are not assigned with any weight to each digit position, i.e., each digit position within the number is not assigned fixed value. Example:-Excess – 3 (XS -3) code and Gray codes

BINARY CODED DECIMAL (BCD):- BCD is a weighted code. In weighted codes, each successive digit from right to left represents weights equal to some specified value and to get the equivalent decimal number add the products of the weights by the corresponding binary digit. 8421 is the most common because 8421 BCD is the most natural amongst theother possible codes.

BCD ADDITION:-

Addition of BCD (8421) is performed by adding two digits of binary, starting from least significant digit. In case if the result is an illegal code (greater than 9) or if there is a carry outof one then add 0110(6) and add the resulting carry to the next most significant.

Add 679.6 from 536.8 using BCD addition.

Solution:-

6 7 9.6	0110 0111 1001 . 0110	(679.6 in BCD)
+ 5 3 6.8	=>+ 0101 0011 0110 . 1000	(536.8 in BCD)
1 2 1 6.4	1011 1010 1111 . 1110	(All are illegal codes)
	+ 0110 +0110 +0110 .+0110	(Add 0110 to each)
	0001 0010 0001 0110 . 0100	
	1 2 1 6 . 4	(corrected sum = 1216.4)

Result is **1216.4**

BCD SUBTRACTION:-

The BCD subtraction is performed by subtracting the digits of each 4 – bit group of the subtrahend from corresponding 4 – bit group of the minuend in the binary starting from theLSD. If there is no borrow from the next higher group[then no correction is required. If there is a borrow from the next group, then 6 (0110) is subtracted from the difference term of this group.

Subtract 147.8 from 206.7 using 8421 BCD code.

Solution:-

```
  2 0 6.7        0010  0000  0110 . 0111   ( 206.7 in BCD)
- 1 4 7.8   =>-  0001  0100  0111 . 1000   (147.8 in BCD)
  5 8.9         0000  1011  1110 . 1111   ( Borrows are present)
                - 0110 -0110 .- 0110
                0101  1000  .  1001
                  5     8   .    9       ( corrected difference = 58.9)
```

Result is $(58.9)_{10}$

EXCESS THREE(XS-3) CODE:-

The Excess-3 code, also called XS-3, is a non- weighted BCD code. This derives it name from the fact that each binary code word is the corresponding 8421 code word plus 0011(3). It is a sequential code. It is a self complementing code.

Excess-3 code is non-weighted and self complementary code. A self complementary binary codes are always compliment themselves. The complement of a binary number can be obtained from that number by replacing 0's with 1's and 1's with 0's. The sum of binary number and its complement is always equal to decimal 9. In other words, the 1's complement of an excess-3 code is the excess-3 code for the 9's complement of the corresponding decimal number. For example, the excess-3 code for decimal number 5 is 1000 and 1's complement of 1000 is 0111, which is excess-3 code for decimal number 4, and it is 9's complement of number 5.

ASCII CODE:-

The American Standard Code for Information Interchange (ASCII) pronounced as 'ASKEE' is widely used alphanumeric code. This is basically a 7 bit code. The number of different bit patterns that can be created with 7 bits is 27 = 128 , the ASCII can be used to encode both the uppercase and lowercase characters of the alphabet (52 symbols) and some special symbols in addition to the 10 decimal digits. It is used extensively for printers and terminals that interface with small computer systems. The table shown below shows the ASCII groups.

GRAY CODE:-

The gray code is a non-weighted code. It is not a BCD code. It is cyclic code because successive words in this differ in one bit position only i.e it is a unit distance code. Gray codeis used in instrumentation and data acquisition systems where linear or angular displacement is measured.

BINARY- TO – GRAY CONVERSION:-

If an n-bit binary number is represented by $B_n\ B_{n-1}$ ----- B_1 and its gray code equivalent by $G_n\ G_{n-1}$ ----- G_1, where B_n and G_n are the MSBs , then gray code bits are obtained from the binary code as follows

$G_n = B_n$

$G_{n-1} = B_n \oplus B_{n-1}$

.

.

.

.

$G_1 = B_2 \oplus B_1$

Where the symbol $\oplus$ stands for Exclusive OR (X-OR)

GRAY- TO - BINARY CONVERSION:-

If an n-bit gray number is represented by $G_n\ G_{n-1}$ ------- G_1 and its binary equivalent by $B_n\ B_{n-1}$ ----- B_1, then binary bits are obtained from Gray bits as follows :

$B_n = G_n$

$B_{n-1} = B_n \oplus G_{n-1}$

.

.

.

.

$B_1 = B_2 \oplus G_1$

Boolean algebra, Boolean expressions, Demorgan's Theorems.

BOOLEAN ALGEBRA INTRODUCTION:-

- Switching circuits are also called logic circuits, gates circuits and digital circuits.
- Boolean algebra is a system of mathematical logic. It is an algebraic system consisting of the set of elements (0,1), two binary operators called OR and AND and unary operatorcalled NOT.
- It is the basic mathematical tool in the analysis and synthesis of switching circuits.
- It is a way to express logic functions algebraically.

AXIOMS AND LAWS OF BOOLEAN ALGEBRA:-

Axioms or postulates of Boolean algebra are set of logical expressions that are acceptedwithout proof and upon which we can build a set of useful theorems.

Axiom 1:	0 . 0 = 0	Axiom 5:	0 + 0 = 0	Axiom 9:	$\bar{1} = 0$
Axiom 2:	0 . 1 = 0	Axiom 6:	0 + 1 = 1	Axiom 10:	$\bar{0} = 1$
Axiom 3:	1 . 0 = 0	Axiom 7:	1 + 0 = 1		
Axiom 4:	1 . 1 = 1	Axiom 8:	1 + 1 = 1		

1. **Complementation Laws**:-

   The term complement simply means to invert, i.e. to changes 0s to 1s and 1s to 0s.The five laws of complementation are as follows:

   Law 1: $\bar{0} = 1$

   Law 2: $\bar{1} = 0$

   Law 3: if A = 0, then $\bar{A} = 1$Law 4: if A = 1,then $\bar{A} = 0$

   Law 5: $\bar{\bar{A}} = 0$ (double complementation law)

2. **OR Laws**:-

   The four OR laws are as followsLaw 1: A + 0 = A (Null law)

   Law 2: A + 1 = 1(Identity law)Law 3: A + A = A

   Law 4: A +$\bar{A} = 1$

3. **AND Laws**:-

   The four AND laws are as followsLaw 1: A . 0 = 0 (Null law)

   Law 2: A . 1 = A (Identity law)Law 3: A . A = A

   Law 4: A .$\bar{A} = 0$

4. **Commutative Laws:-**

   Commutative laws allow change in position of AND or OR variables. There are twocommutative laws.

   Law 1: A + B = B + ALaw 2: A . B = B . A

5. **Associative Laws:-**

   The associative laws allow grouping of variables. There are 2 associative laws.Law 1: (A + B) + C = A + (B + C)

   Law 2: (A .B) C = A (B .C)

6. **Distributive Laws:-**

   The distributive laws allow factoring or multiplying out of expressions. There aretwo distributive laws.

   Law 1: A (B + C) = AB + AC Law 2: A + BC = (A+B) (A+C)

   Proof:

   RHS = (A+B) (A+C) = AA + AC + BA + BC

   = A + AC + AB + BC

   = A (1+ C + B) + BC

   = A. 1 + BC (1 +C + B = 1 + B = 1, FROM OR Law 2)

   = A + BC = LHS

7. Redundant Literal Rule (RLR):-

Law 1: $A + \bar{A}B = A + B$ Proof

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

$= 1.(A + B)$

$= A + B$

Law 2: $A(\bar{A} + B) = AB$ Proof

$A(\bar{A} + B) = A\bar{A} + AB = 0 + AB = AB$

8. Idempotence Laws:- Idempotence means same value. Law 1: $A.A = A$

Law 2: $A + A = A$

9. Absorption Laws:-

There are two laws:

Law 1: $A + A \cdot B = A$

Proof: $A + A \cdot B$

$= A(1 + B)$

$= A \cdot 1 = A$

Law 2: $A(A + B) = A$

Proof : $A(A + B)$

$= A \cdot A + A \cdot B$

$= A + AB$

$= A(1 + B)$

$= A \cdot 1 = A$

12. De Morgan's Theorem:-

De Morgan's theorem represents two laws in Boolean algebra.

This law states that the complement of a sum of variables is equal to the product oftheir individual complements.

Law 1: $\overline{A + B} = \bar{A} \cdot \bar{B}$

Proof

A	B	A + B	$\overline{A + B}$
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0

=

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

Law 2: $\overline{A.B} = \bar{A} + \bar{B}$

This law states that the complement of a product of variables is equal to the sum oftheir individual complements.

Proof

A	B	A . B	$\overline{A.B}$
0	0	0	1
0	1	0	1
1	0	0	1
1	1	1	0

=

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

Canonical and Standard Form

Canonical Form – In Boolean algebra, the Boolean function can be expressed as Canonical Disjunctive Normal Form known as minterm and some are expressed as Canonical Conjunctive Normal Form known as maxterm.

In Minterm, we look for the functions where the output results in “1” while in Maxterm we look for functions where the output results in “0”.

We perform the Sum of minterm also known as the Sum of products (SOP).
We perform Product of Maxterm also known as Product of sum (POS).

Advantages of Canonical Form:
Uniqueness: The canonical form of a boolean function is unique, which means that there is only one possible canonical form for a given function.
Clarity: The canonical form of a boolean function provides a clear and unambiguous representation of the function.
Completeness: The canonical form of a boolean function can represent any possible boolean function, regardless of its complexity.

Disadvantages of Canonical Form:
Complexity: The canonical form of a boolean function can be complex, especially for functions with many variables.
Computation: Computing the canonical form of a boolean function can be computationally expensive, especially for large functions.
Redundancy: The canonical form of a boolean function can be redundant, which means that it can contain unnecessary terms or variables that do not affect the function.

Advantages of Standard Form:
Simplicity: The standard form of a boolean function is simpler than the canonical form, making it easier to understand and work with.
Efficiency: The standard form of a boolean function can be implemented using fewer logic gates than the canonical form, which makes it more efficient in terms of hardware and computation.
Flexibility: The standard form of a boolean function can be easily modified and combined with other functions to create new functions that meet specific design requirements.

Disadvantages of Standard Form:
Non-uniqueness: The standard form of a boolean function is not unique, which means that there can be multiple possible standard forms for a given function.
Incompleteness: The standard form of a boolean function may not be able to represent some complex boolean functions.
Ambiguity: The standard form of a boolean function can be ambiguous, especially if it contains multiple equivalent expressions.

Represent Logic Expression: SOP & POS formsSUM - OF -

PRODUCTS FORM:-

- This is also called disjunctive Canonical Form (DCF) or Expanded Sum of Products Formor Canonical Sum of Products Form.
- In this form, the function is the sum of a number of products terms where eachproduct term contains all variables of the function either in complemented or uncomplemented form.

The or product term which contains all the variables of the functions either incomplemented uncomplemented form is called a **minterm**.

- The minterm is denoted as mo, m1, m2 … . An 'n' variable function can have 2^n minterms.

$$f(A, B, C) = \sum m(1, 2, 3, 5)$$

PRODUCT- OF - SUMS FORM:-

- This form is also called as Conjunctive Canonical Form (CCF) or Expanded Product - of – Sums • This is by considering the combinations for which f = 0
- Each term is a sum of all the variables.

The sum term which contains each of the 'n' variables in either complemented oruncomplemented form is called a **maxterm.**

- Maxterm is represented as M0, M1, M2, …….f(A, B, C) = ΠM (0, 4, 6, 7)

Logic gates: AND, OR, NOT, NAND, NOR, Exclusive-OR, Exclusive-NOR--Symbol, Function, expression, truth table & timing diagram

LOGIC GATES:-

- Logic gates are the fundamental building blocks of digital systems.
- There are 3 basic types of gates AND, OR and NOT.
- Logic gates are electronic circuits because they are made up of a number of electronicdevices and components.
- Inputs and outputs of logic gates can occur only in 2 levels(logic 1, logic 0). These twolevels are termed HIGH and LOW, or TRUE and FALSE, or ON and OFF
- *The table which lists all the possible combinations of input variables and thecorresponding output of any logic circuit/device, called a* ***truth table***.

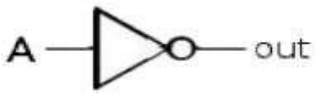
DIFFERENT TYPES OF LOGIC GATES

NOT GATE (INVERTER):-

- A NOT gate, also called and inverter, has only one input and one output.
- It is a device whose output is always the complement of its input.
- The output of a NOT gate is the logic 1 state when its input is in logic 0 state and thelogic 0 state when its inputs is in logic 1 state.

IC No. :- 7404

Logic Symbol



Truth table

INPUT A	OUTPUT $\overline{A}$
0	1
1	0

AND GATE:-

- An AND gate has two or more inputs but only one output.
- The output is logic 1 state only when each one of its inputs is at logic 1 state.
- The output is logic 0 state even if one of its inputs is at logic 0 state.

IC No.:- 7408

Logic Symbol

Truth Table

		OUTPUT
A	B	Q=A . B
0	0	0
0	1	0
1	0	0
1	1	1

OR GATE:-

- An OR gate may have two or more inputs but only one output.
- The output is logic 1 state, even if one of its input is 1
- The output is logic 0 state, only when each one of its inputs is in logic state.

IC No.:- 7432

Logic Symbol

Truth Table

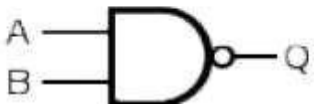
INPUT		OUTPUT
A	B	Q=A + B
0	0	0
0	1	1
1	0	1
1	1	1

NAND GATE:-

- NAND gate is a combination of an AND gate and a NOT gate.
- The output is logic 0 when each of the input is logic 1 and for any other combinationof inputs, the output is logic 1.

IC No.:- 7400 two input NAND gate

Logic Symbol



Truth Table

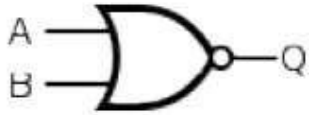
INPUT		OUTPUT
A	B	Q= $\overline{A \cdot B}$
0	0	1
0	1	1
1	0	1
1	1	0

NOR GATE:-

- NOR gate is a combination of an OR gate and a NOT gate.
- The output is logic 1, only when each one of its input is logic 0 and for any othercombination of inputs the output is a logic 0 level.

IC No.:- 7402 two input NOR gate

Logic Symbol



Truth Table

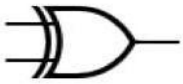
INPUT		OUTPUT
A	B	Q= $\overline{A + B}$
0	0	1
0	1	0
1	0	0
1	1	0

EXCLUSIVE – OR (X-OR) GATE

- An X-OR gate is a two input, one output logic circuit.
- The output is logic 1 when one and only one of its two inputs is logic 1. When both theinputs is logic 0 or when both the inputs is logic 1, the output is logic 0.

IC No.:- 7486

Logic Symbol



INPUTS are **A** and **B**

OUTPUT is **Q** = $A \oplus B$

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

Truth Table

INPUT		OUTPUT
A	B	Q = A⊕ B
0	0	0
0	1	1
1	0	1
1	1	0

EXCLUSIVE – NOR (X-NOR) GATE

- An X-NOR gate is the combination of an X-OR gate and NOT gate

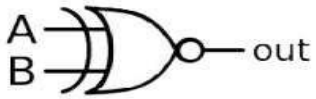
- An X-NOR gate is a two input, one output logic circuit.

The output is logic 1 only when both the inputs are logic 0 or when both the inputs is 1.

- The output is logic 0 when one of the inputs is logic 0 and other is 1

IC No.:- 74266

Logic Symbol



INPUT		OUTPUT
A	B	OUT =A XNOR B
0	0	1
0	1	0
1	0	0
1	1	1

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

= A XNOR B

Universal Gates & its Realisation

UNIVERSAL GATES:-

There are 3 basic gates AND, OR and NOT, there are two universal gates NAND and NOR.Both NAND and NOR gates can perform all logic functions i.e. AND, OR, NOT, EXOR and EXNOR.

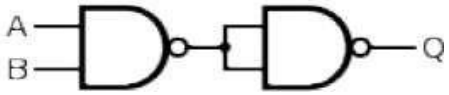
NAND GATE:-

a) **Inverter from NAND gate**

Input = A
Output Q = $\overline{A}$

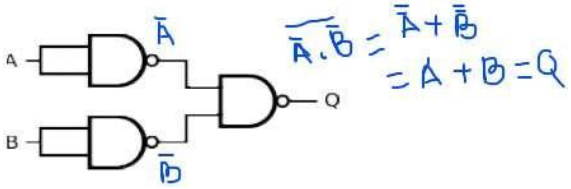
b) **AND gate from NAND gate**

Input s are **A** and **B**
Output **Q = A.B**



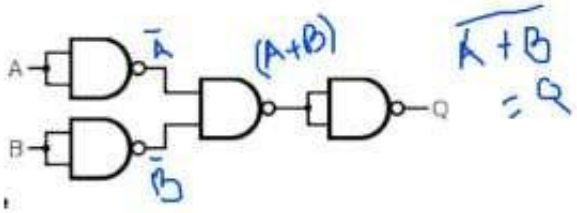
c) **OR gate from NAND gate**

Inputs are **A** and **B**
Output **Q = A+B**



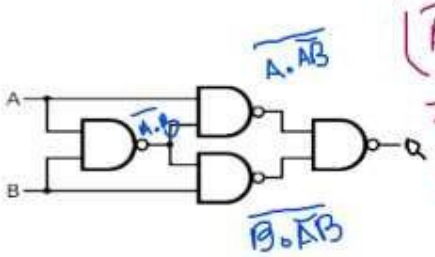
d) **NOR gate from NAND gate**

Inputs are **A** and **B**
Output **Q = $\overline{A+B}$**



e) **EX-OR gate from NAND gate**

Inputs are **A** and **B**
Output **Q = A $\bar{B}$ + $\bar{A}$B**



$\overline{A.\bar{A}B}$

$\overline{B.\bar{A}B}$

$\overline{(A.\bar{A}B)}\,\overline{(B.\bar{A}B)}$

$= \overline{A.\bar{A}B} + \overline{B.\bar{A}B}$

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

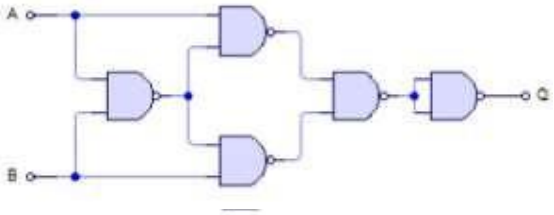
$= A\bar{B} + \bar{A}B = Q$

e) **EX-OR g**

Inputs ar
Output **Q**

f) **EX-NOR gate From NAND gate**

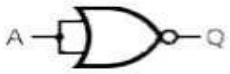
Inputs are **A** and **B**
Output **Q = A B + $\bar{A}$ $\bar{B}$**



NOR GATE:-

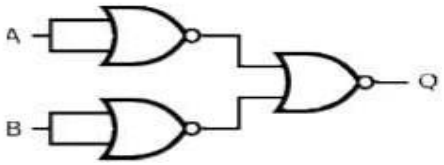
a) **Inverter from NOR gate**

Input = **A**
Output **Q = $\bar{A}$**



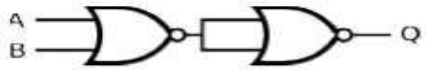
b) **AND gate from NOR gate**

Input s are **A** and **B**
Output **Q = A.B**



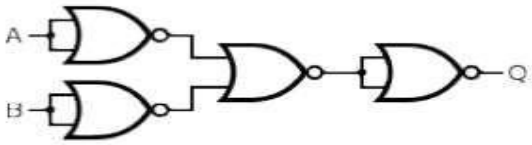
c) OR gate from NOR gate

Inputs are **A** and **B**
Output **Q = A+B**



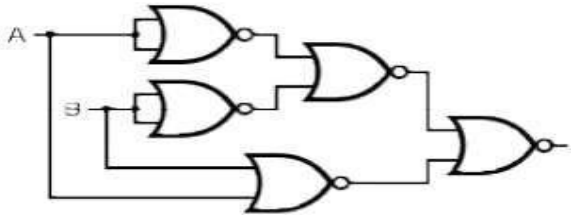
d) NAND gate from NOR gate

Inputs are **A** and **B**
Output **Q = $\overline{A.B}$**



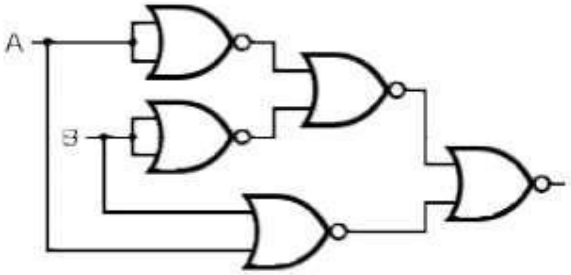
e) EX-OR gate from NOR gate

Inputs are **A** and **B**
Output **Q = A $\overline{B}$ + $\overline{A}$B**



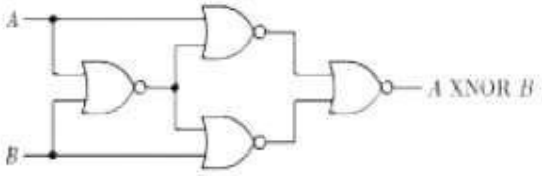
e) EX-OR gate from NOR gate

Inputs are **A** and **B**
Output **Q = A $\overline{B}$ + $\overline{A}$B**



f) EX-NOR gate From NOR gate

Inputs are **A** and **B**
Output **Q = A B + $\overline{A}$ $\overline{B}$**



Unit-2: Gate-Level Minimization

Map Method – Four-Variable K-Map – Product-of-Sums Simplification – Don't-Care Conditions – NAND and NOR Implementation – Other Two-Level Implementations – Exclusive-OR Function – Hardware Description Language

KARNAUGH MAP OR K- MAP:-

The simplification of Boolean expressions using Boolean algebraic rules is not unique and most of the cases, the resultant expression is not in minimal form. In order to get the uniqueness and final minimal form, K-map technique will be used.

- The K- map is a chart or a graph, composed of an arrangement of adjacent cells, each representing a particular combination of variables in sum or product form

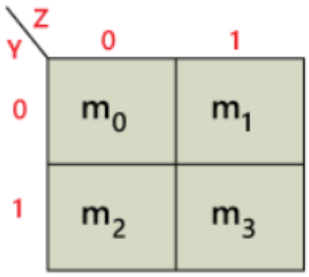
. • The K- map is systematic method of simplifying the Boolean expression.

Mapping of SOP Expression:-

- The n variable K-map has 2^n squares. These squares are called cells.
- A '1' is placed in any square indicates that corresponding minterm is included in the output expression, and a 0 or no entry in any square indicates that the corresponding minterm does not appear in the expression for output.

2 Variable K-map

There is a total of 4 variables in a 2-variable K-map. There are two variables in the 2-variable K-map. The following figure shows the structure of the 2-variable K-map:

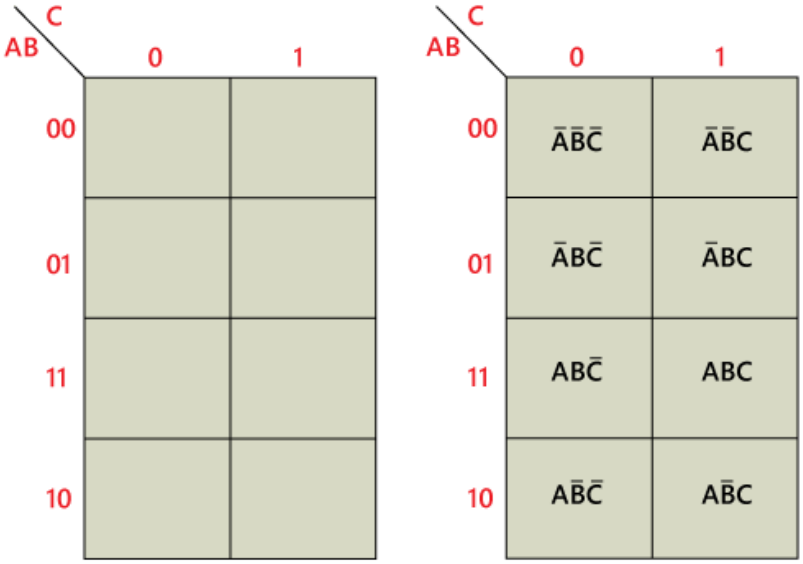


- In the above figure, there is only one possibility of grouping four adjacent minterms.
- The possible combinations of grouping 2 adjacent minterms are {(m_0, m_1), (m_2, m_3), (m_0, m_2) and (m_1, m_3)}.

Karnaugh map (3 & 4 Variables) & Minimization of logicalexpressions, don't care conditions

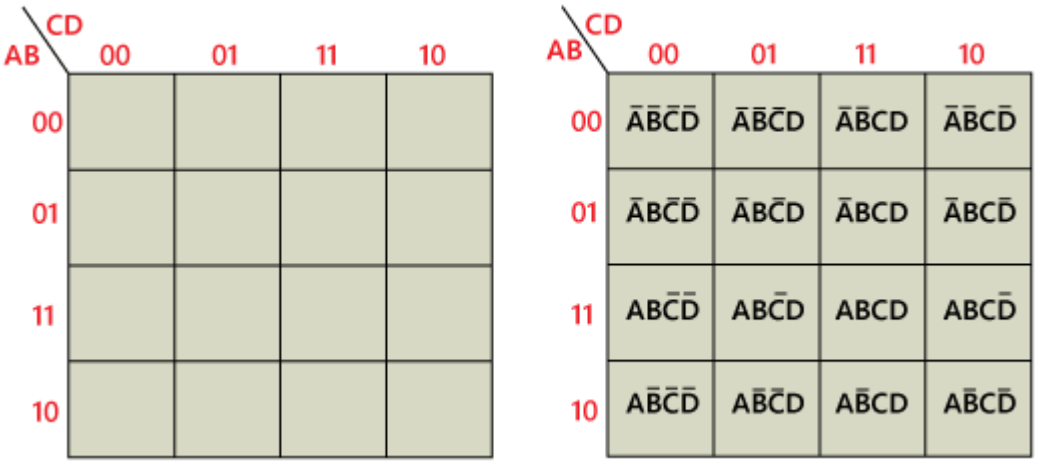
3-variable K-map

The 3-variable K-map is represented as an array of eight cells. In this case, we used A, B, and C for the variable. We can use any letter for the names of the variables. The binary values of variables A and B are along the left side, and the values of C are across the top. The value of the given cell is the binary values of A and B at left side in the same row combined with the value of C at the top in the same column. For example, the cell in the upper left corner has a binary value of 000, and the cell in the lower right corner has a binary value of 101.



The 4-Variable Karnaugh Map

The 4-variable K-map is represented as an array of 16 cells. Binary values of A and B are along the left side, and the values of C and D are across the top. The value of the given cell is the binary values of A and B at left side in the same row combined with the binary values of C and D at the top in the same column. For example, the cell in the upper right corner has a binary value of 0010, and the cell in the lower right corner has a binary value of 1010



Simplification of boolean expressions using Karnaugh Map

As we know that K-map takes both SOP and POS forms. So, there are two possible solutions for K-map, i.e., minterm and maxterm solution. Let's start and learn about how we can find the minterm and maxterm solution of K-map.

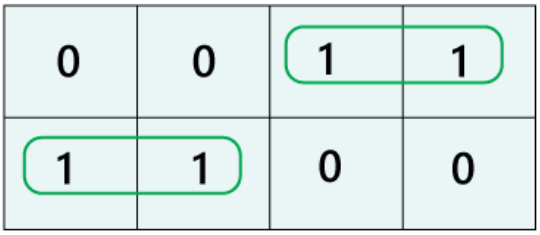
Minterm Solution of K Map

There are the following steps to find the minterm solution or K-map:

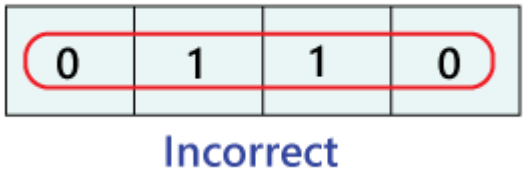
Step 1: Firstly, we define the given expression in its canonical form.

Step 2: Next, we create the K-map by entering 1 to each product-term into the K-map cell and fill the remaining cells with zeros.

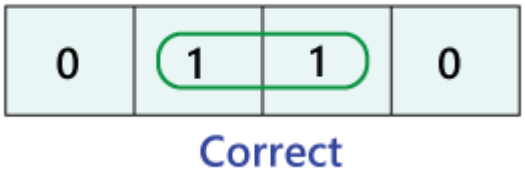
Step 3: Next, we form the groups by considering each one in the K-map.



Notice that each group should have the largest number of 'ones'. A group cannot contain an empty cell or cell that contains 0.



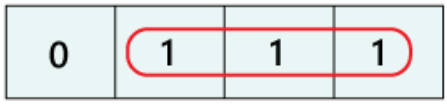
Incorrect



Correct

In a group, there is a total of 2n number of ones. Here, n=0, 1, 2, …n.

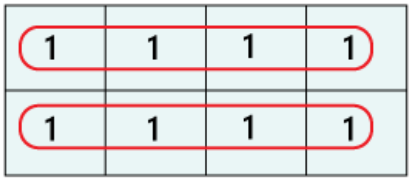
Example: 20=1, 21=2, 22=4, 23=8, or 24=16.



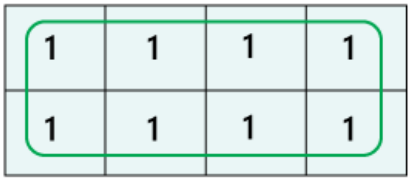
Incorrect

Correct

We group the number of ones in the decreasing order. First, we have to try to make the group of eight, then for four, after that two and lastly for 1.

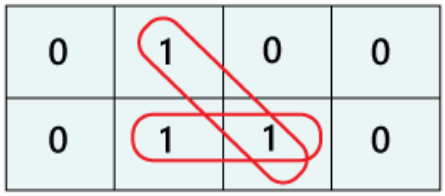


Incorrect

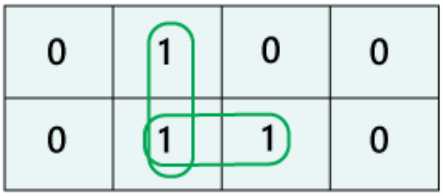


Correct

In horizontally or vertically manner, the groups of ones are formed in shape of rectangle and square. We cannot perform the diagonal grouping in K-map.

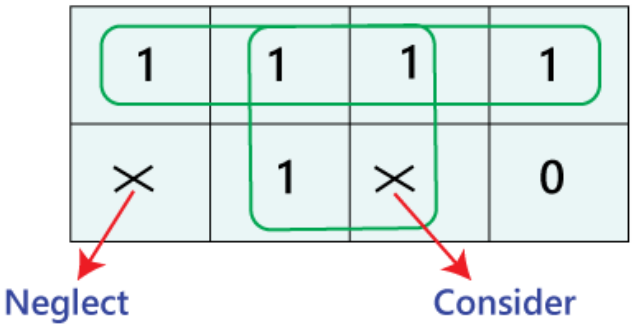


Incorrect



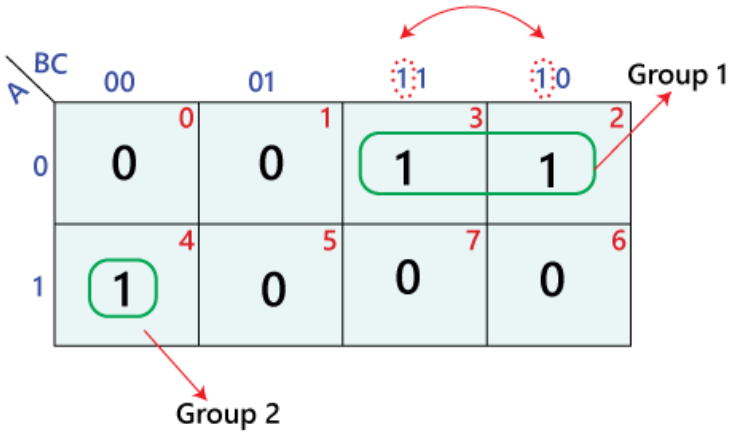
Correct

We can consider the 'don't care condition' only when they aid in increasing the group-size. Otherwise, 'don't care' elements are discarded.



Step 4:
In the next step, we find the boolean expression for each group. By looking at the common variables in cell-labeling, we define the groups in terms of input variables. In the below example, there is a total of two groups, i.e., group 1 and group 2, with two and one number of 'ones'.

In the first group, the ones are present in the row for which the value of A is 0. Thus, they contain the complement of variable A. Remaining two 'ones' are present in adjacent columns. In these columns, only B term in common is the product term corresponding to the group as A'B. Just like group 1, in group 2, the one's are present in a row for which the value of A is 1. So, the corresponding variables of this column are B'C'. The overall product term of this group is AB'C'.

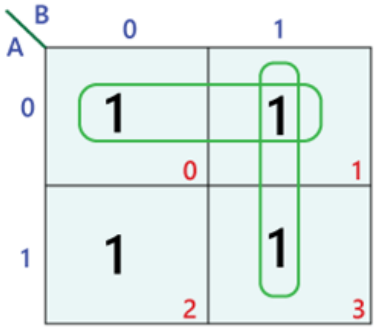


Step 5:

Lastly, we find the boolean expression for the Output. To find the simplified boolean expression in the SOP form, we combine the product-terms of all individual groups. So the simplified expression of the above k-map is as follows:

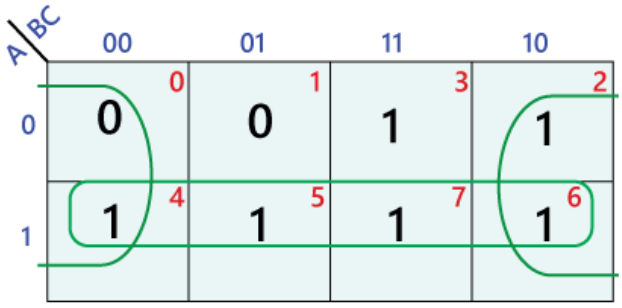
A'+AB'C'
Let's take some examples of 2-variable, 3-variable, 4-variable, and 5-variable K-map examples.

Example 1: Y=A'B' + A'B+AB



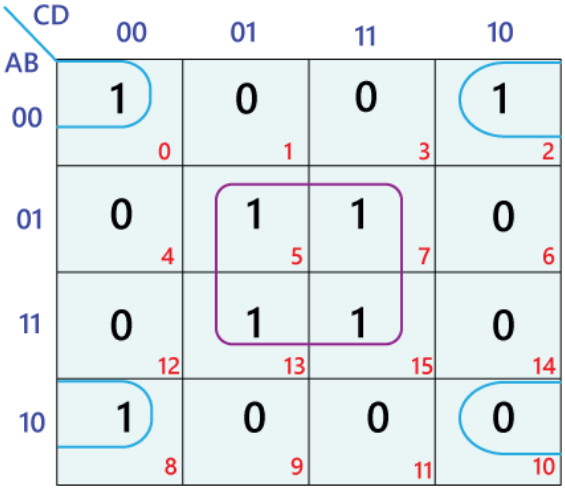
Simplified expression: Y=A'+B

Example 2: Y=A'B'C'+A' BC'+AB' C'+AB' C+ABC'+ABC



Simplified expression: Y=A+C'

Example 3: Y=A'B'C' D'+A' B' CD'+A' BCD'+A' BCD+AB' C' D'+ABCD'+ABCD



Simplified expression: Y=BD+B'D'

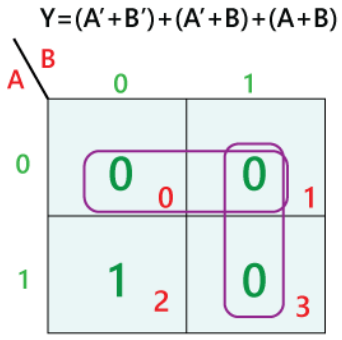
Maxterm Solution of K-Map

To find the simplified maxterm solution using K-map is the same as to find for the minterm solution. There are some minor changes in the maxterm solution, which are as follows:

1. We will populate the K-map by entering the value of 0 to each sum-term into the K-map cell and fill the remaining cells with one's.
2. We will make the groups of 'zeros' not for 'ones'.
3. Now, we will define the boolean expressions for each group as sum-terms.
4. At last, to find the simplified boolean expression in the POS form, we will combine the sum-terms of all individual groups.

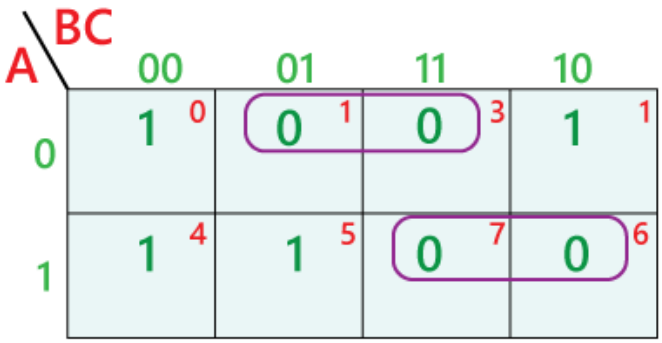
Let's take some example of 2-variable, 3-variable, 4-variable and 5-variable K-map examples

Example 1: Y=(A'+B')+(A'+B)+(A+B)



Simplified expression: A'B

Example 2: Y=(A + B + C') + (A + B' + C') + (A' + B' + C) + (A' + B' + C')



Simplified expression: Y=(A + C') .(A' + B')

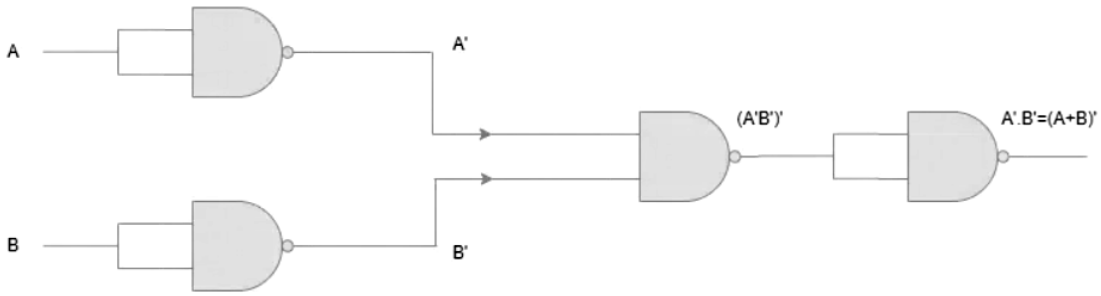
Implementation of NOR Gate from NAND Gate

Following is the implementation of NOR gate from NAND gate. First, we produce complements of inputs and further implement NOR logic.

Logic Diagram for Implementation of NOR Gate from NAND Gate

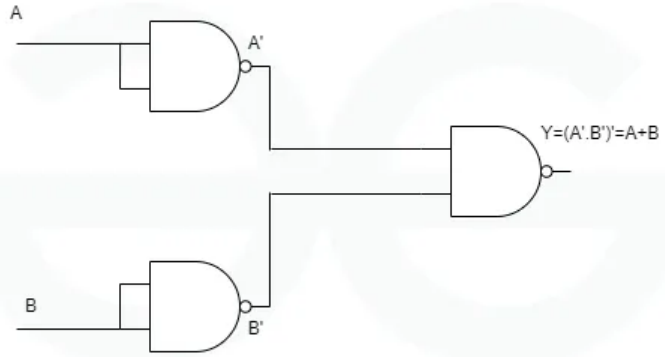
Below is the logic diagram for the implementation of NOR gate from NAND gate.

- To Implement NOR Gate as a NAND Gate,We will Take two NAND gate and connect its two terminal together which will act as Inverter.
- Input A and B are Fed to the NAND Gate having both Input terminal Together.So, the Output of Input A NAND gate Will give A' ,Similarly B will give B'.
- Then Both this Output is Fed to the NAND gate which will give Output as (A'B')'.
- Finally,a NAND gate is connected to this Output Having Both the Input Terminals Together, which will give output as A'.B'=(A+B)' which is the Output of NOR gate.



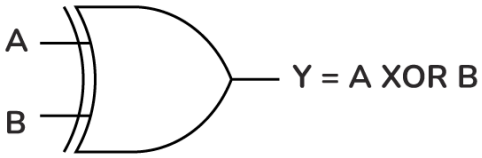
Implementation of OR Gate from NAND Gate

The Implementation of OR Gate from NAND Gate is executed by using De Morgan's theorem, which asserts that the complement of the AND operation is equivalent to the NAND operation, so that it can be used to construct an OR gate by using NAND gates. So that We can effectively execute the functionality of an OR gate by combining numerous NAND gates in a particular arrangement. So that we can implement an OR Gate by using NAND Gates.



What is XOR Gate?

XOR gate is a logic gate that results in an output high with an odd number of high inputs. In other words, if the number of 1's is odd in the input then, the output of XOR is 1. XOR gate gives output 1 when all the inputs are different. The XOR is represented as ⊕.



Truth Table

B (Input)	B (Input 2)	Y = A ⊕ B
0	0	0
0	1	1
1	0	1
1	1	0

XOR Gate

In the above table if the inputs are different then the output is 1 otherwise 0.

Expression for 2-Input XOR Gate

We get the expression of XOR gate as sum of products of complement of first input with second input and first input with complement of second input. From the above truth table, the expression of XOR gate is:

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