

**MODULE-1****PRINCIPLES OF COMBINATIONAL LOGIC**

Definition of combinational logic, Canonical forms, Generation of switching equations from truth tables, Karnaugh maps-2,3,4variables, Quine-McCluskey Minimization Technique, Quine-McCluskey using don't care terms.

**1.1 COMBINATIONAL LOGIC**

Introduction Logic circuit may be classified into two categories

1. Combinational logic circuits
2. Sequential logic circuits

A combinational logic circuit contains logic gates only but does not contain storage elements. A sequential logic circuit contains storage elements in addition to logic gates. When logic gates are connected together to give a specified output for certain specified combination of input variables, with no storage involved, the resulting network is known as combinational logic circuit.

In combinational logic circuit the output level is at all times dependent on the combination of input level. The block diagram is shown

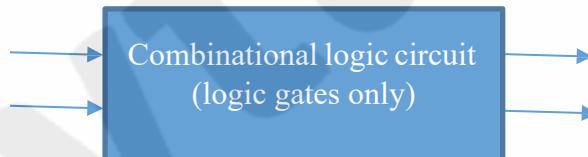


Fig : Block diagram of Combinational circuit

The combinational logic circuit with memory element(s) is called sequential logic circuit. It consists of a combinational circuit to which memory elements are connected to form a feedback path. The memory elements are devices, capable of storing binary information within them. The block diagram is shown.

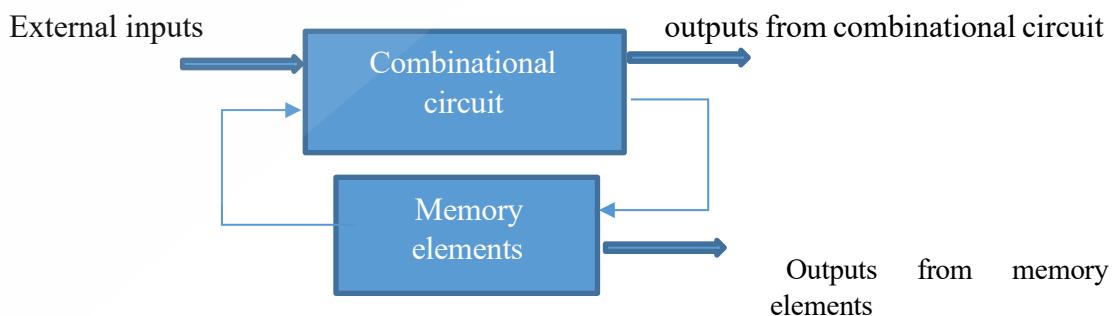


Fig : Block diagram of Sequential circuit

By block diagram, it can be said that the output(s) of sequential logic circuit is (are) dependent not only on external input(s) but also on the present state of the memory element(s). The next state of the memory element(s) is also dependent on external input and the present state. Applications Logic gates find wide applications in Calculators and computers, digital measuring techniques, digital processing of communications, musical instruments, games and domestic appliances etc, for decision making in automatic control of machines and various industrial processes and for building more complex devices such as binary counters etc.

### Laws and Rules of Boolean Algebra

- Laws of Boolean Algebra

The basic laws of Boolean algebra-the commutative laws for addition and multiplication, the associative laws for addition and multiplication, and the distributive law-are the same as in ordinary algebra.

The commutative law  $A+B = B+A$

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

The associative law  $A + (B + C) = (A + B) + C$

$$A(BC) = (AB)C$$

Distributive Law  $A(B + C) = AB + AC$

- Rules of Boolean Algebra

<b>1.</b> $A + 0 = A$	<b>7.</b> $A \cdot A = A$
<b>2.</b> $A + 1 = 1$	<b>8.</b> $A \cdot \bar{A} = 0$
<b>3.</b> $A \cdot 0 = 0$	<b>9.</b> $\bar{\bar{A}} = A$
<b>4.</b> $A \cdot 1 = A$	<b>10.</b> $A + AB = A$
<b>5.</b> $A + A = A$	<b>11.</b> $A + \bar{A}B = A + B$
<b>6.</b> $A + \bar{A} = 1$	<b>12.</b> $(A + B)(A + C) = A + BC$

*A, B, or C* can represent a single variable or a combination of variables.

(Referring to the table above)

Proof Rule 10:  $A + AB = A$

This rule can be proved by applying the distributive law, rule 2, and rule 4 as follows:

$$\begin{aligned} A + AB &= A(1 + B) && \text{Factoring (distributive law)} \\ &= A \cdot 1 && \text{Rule 2: } (1 + B) = 1 \\ &= A && \text{Rule 4: } A \cdot 1 = A \end{aligned}$$

Rule 11.

$$A + AB = A + B$$

This rule can be proved as follows:

$$A + AB = (A + AB) + AB \quad \text{Rule 10: } A + AB$$

$$\begin{aligned}
 &= (AA + AB) + AB && \text{Rule 7: } A = AA \\
 &= AA + AB + AA + AB && \text{Rule 8: adding } AA = 0 \\
 &= (A + A)(A + B) && \text{Factoring} \\
 &= 1 \cdot (A + B) && \text{Rule 6: } A + A = 1 \\
 &= A + B && \text{Rule 4: drop the 1}
 \end{aligned}$$

Rule 12.  $(A + B)(A + C) = A + BC$

This rule can be proved as follows:

$$(A + B)(A + C) = AA + AC + AB + BC \quad \text{Distributive law}$$

$$\begin{aligned}
 &= A + AC + AB + BC && \text{Rule 7: } AA = A \\
 &= A(1 + C) + AB + BC && \text{Rule 2: } 1 + C = 1 \\
 &= A \cdot 1 + AB + BC && \text{Factoring (distributive law)} \\
 &= A(1 + B) + BC && \text{Rule 2: } 1 + B = 1 \\
 &= A \cdot 1 + BC && \text{Rule 4: } A \cdot 1 = A \\
 &= A + BC
 \end{aligned}$$

## DEMORGAN'S THEOREMS

The complement of a product of variables is equal to the sum of the individual complements of the variables.

$$\bar{\bar{X}} \cdot \bar{\bar{Y}} = X + Y$$

The complement of a sum of variables is equal to the product of the individual complements of the variables.

$$\bar{\bar{X}} + \bar{\bar{Y}} = X \cdot Y$$

## 1.2. CANONICAL FORMS AND NORMAL FORMS

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.

<b>x</b>	<b>y</b>	<b>Min terms</b>	<b>Max terms</b>
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.

### 1.3 GENERATION OF SWITCHING EQUATION FROM TRUTH TABLE

#### Canonical SoP and PoS forms

A truth table consists of a set of inputs and output(s). 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 (Minterm canonical 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) 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.

Example:

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 (f) 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 (f).

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.

$$f = m_3 + m_5 + m_6 + m_7$$

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

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 (Maxterm canonical 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) 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 (f) 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 (f).

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

$$f=M_0.M_1.M_2.M_4$$

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

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,  $f=(p+q+r).(p+q+r').(p+q'+r).(p'+q+r)$  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 and PoS forms

We discussed two canonical forms of representing the Boolean output(s). Similarly, there are two standard forms of representing the Boolean output(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 1<sup>st</sup> and 4<sup>th</sup> terms, 2<sup>nd</sup> and 5<sup>th</sup> terms, 3<sup>rd</sup> and 6<sup>th</sup> terms.

$$\Rightarrow f = qr(p' + p) + pr(q' + q) + pq(r' + r)$$

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

$$\Rightarrow f = qr(1) + pr(1) + pq(1)$$

**Step 4** – Use **Boolean postulate**,  $x \cdot 1 = x$  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.

$$f = (p+q+r).(p+q+r').(p+q'+r).(p'+q+r)$$

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**,  $x \cdot x = x$ . 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.

$$\Rightarrow f = (p+q+r).(p+q+r).(p+q+r).(p+q+r').(p+q'+r).(p'+q+r)$$

**Step 2** – Use **Distributive law**,  $x + (y \cdot z) = (x+y) \cdot (x+z)$  for 1<sup>st</sup> and 4<sup>th</sup> parenthesis, 2<sup>nd</sup> and 5<sup>th</sup> parenthesis, 3<sup>rd</sup> and 6<sup>th</sup> parenthesis.

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

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

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

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

$$\Rightarrow f = (p+q).(p+r).(q+r)$$

$$\Rightarrow f = (p+q).(q+r).(p+r)$$

This is the simplified Boolean function. Therefore, the **standard PoS form** corresponding to given canonical PoS form is  $f = (p+q).(q+r).(p+r)$ . 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.

### 1.4. K-MAPS FOR 2 TO 5 VARIABLES

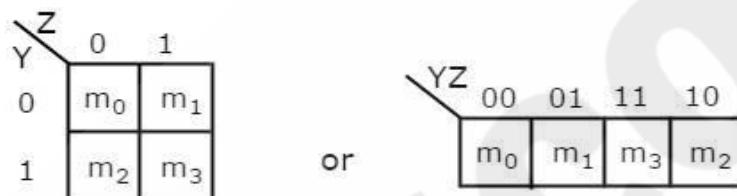
We have simplified the Boolean functions using Boolean postulates and theorems. It is a time-consuming process and we have to re-write the simplified expressions after each step.

To overcome this difficulty, **Karnaugh** introduced a method for simplification of Boolean functions in an easy way. This method is known as Karnaugh map method or K-map method. It is a graphical method, which consists of  $2^n$  cells for ‘n’ variables. The adjacent cells are differed only in single bit position.

K-Map method is most suitable for minimizing Boolean functions of 2 variables to 5 variables. Now, let us discuss about the K-Maps for 2 to 5 variables one by one.

## 2 Variable K-Map

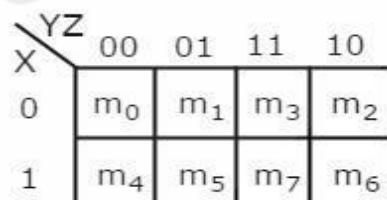
The number of cells in 2 variable K-map is four, since the number of variables is two. The following figure shows **2 variable K-Map**.



- There is only one possibility of grouping 4 adjacent min terms.
- The possible combinations of grouping 2 adjacent min terms are  $\{(m_0, m_1), (m_2, m_3), (m_0, m_2) \text{ and } (m_1, m_3)\}$ .

## 3 Variable K-Map

The number of cells in 3 variable K-map is eight, since the number of variables is three. The following figure shows **3 variable K-Map**.



- There is only one possibility of grouping 8 adjacent min terms.
- The possible combinations of grouping 4 adjacent min terms are  $\{(m_0, m_1, m_3, m_2), (m_4, m_5, m_7, m_6), (m_0, m_1, m_4, m_5), (m_1, m_3, m_5, m_7), (m_3, m_2, m_7, m_6) \text{ and } (m_2, m_0, m_6, m_4)\}$ .
- The possible combinations of grouping 2 adjacent min terms are  $\{(m_0, m_1), (m_1, m_3), (m_3, m_2), (m_2, m_0), (m_4, m_5), (m_5, m_7), (m_7, m_6), (m_6, m_4), (m_0, m_4), (m_1, m_5), (m_3, m_7) \text{ and } (m_2, m_6)\}$ .
- If  $x=0$ , then 3 variable K-map becomes 2 variable K-map.

#### 4 Variable K-Map

The number of cells in 4 variable K-map is sixteen, since the number of variables is four. The following figure shows **4 variable K-Map**.

	Y\Z	00	01	11	10
W\X	00	$m_0$	$m_1$	$m_3$	$m_2$
	01	$m_4$	$m_5$	$m_7$	$m_6$
	11	$m_{12}$	$m_{13}$	$m_{15}$	$m_{14}$
	10	$m_8$	$m_9$	$m_{11}$	$m_{10}$

- There is only one possibility of grouping 16 adjacent min terms.
- Let  $R_1, R_2, R_3$  and  $R_4$  represents the min terms of first row, second row, third row and fourth row respectively. Similarly,  $C_1, C_2, C_3$  and  $C_4$  represents the min terms of first column, second column, third column and fourth column respectively. The possible combinations of grouping 8 adjacent min terms are  $\{(R_1, R_2), (R_2, R_3), (R_3, R_4), (R_4, R_1), (C_1, C_2), (C_2, C_3), (C_3, C_4), (C_4, C_1)\}$ .
- If  $w=0$ , then 4 variable K-map becomes 3 variable K-map.

#### Minimization of Boolean Functions using K-Maps

If we consider the combination of inputs for which the Boolean function is ‘1’, then we will get the Boolean function, which is in **standard sum of products** form after simplifying the K-map.

Similarly, if we consider the combination of inputs for which the Boolean function is ‘0’, then we will get the Boolean function, which is in **standard product of sums** form after simplifying the K-map.

Follow these **rules for simplifying K-maps** in order to get standard sum of products form.

- Select the respective K-map based on the number of variables present in the Boolean function.
- If the Boolean function is given as sum of min terms form, then place the ones at respective min term cells in the K-map. If the Boolean function is given as sum of products form, then place the ones in all possible cells of K-map for which the given product terms are valid.

- Check for the possibilities of grouping maximum number of adjacent ones. It should be powers of two. Start from highest power of two and upto least power of two. Highest power is equal to the number of variables considered in K-map and least power is zero.
- Each grouping will give either a literal or one product term. It is known as **prime implicant**. The prime implicant is said to be **essential prime implicant**, if atleast single ‘1’ is not covered with any other groupings but only that grouping covers.
- Note down all the prime implicants and essential prime implicants. The simplified Boolean function contains all essential prime implicants and only the required prime implicants.

**Note 1** – If outputs are not defined for some combination of inputs, then those output values will be represented with **don't care symbol** ‘x’. That means, we can consider them as either ‘0’ or ‘1’. **Note 2** – If don't care terms also present, then place don't cares ‘x’ in the respective cells of K-map. Consider only the don't cares ‘x’ that are helpful for grouping maximum number of adjacent ones. In those cases, treat the don't care value as ‘1’.

### 1.5. THE TABULATION METHOD (QUINE-MC CLUSKEY ALGORITHM)

For function of five or more variables, it is difficult to be sure that the best selection is made. In such case, the tabulation method can be used to overcome such difficulty. The tabulation method was first formulated by Quine and later improved by McCluskey. It is also known as Quine-McCluskey method.

The Quine–McCluskey algorithm (or the method of prime implicants) is a method used for minimization of boolean functions. It is functionally identical to Karnaugh mapping, but the tabular form makes it more efficient for use in computer algorithms, and it also gives a deterministic way to check that the minimal form of a Boolean function has been reached.

The method involves two steps:

- Finding all prime implicants of the function.
- Use those prime implicants in a prime implicant chart to find the essential prime implicants of the function, as well as other prime implicants that are necessary to cover the function.

**Finding prime implicants** : Minimizing an arbitrary function:

A B C D f

m0	0	0	0	0	0
m1	0	0	0	1	0
m2	0	0	1	0	0
m3	0	0	1	1	0
m4	0	1	0	0	1
m5	0	1	0	1	0
m6	0	1	1	0	0
m7	0	1	1	1	0
m8	1	0	0	0	1
m9	1	0	0	1	x
m10	1	0	1	0	1
m11	1	0	1	1	1
m12	1	1	0	0	1
m13	1	1	0	1	0
m14	1	1	1	0	x
m15	1	1	1	1	1

One can easily form the canonical sum of products expression from this table, simply by summing the minterms (leaving out don't-care terms) where the function evaluates to one:

$$F(A,B,C,D) = A'BC'D' + AB'C'D' + AB'CD' + AB'CD + ABC'D' + ABCD$$

Of course, that's certainly not minimal. So to optimize, all minterms that evaluate to one are first placed in a minterm table. Don't-care terms are also added into this table, so they can be combined with minterms:

Number of 1s Minterm Binary Representation

1	m4	0100	
	m8	1000	
2	m9	1001	
	m10	1010	
	m12	1100	
3	m11	1011	
	m14	1110	
4	m15	1111	

At this point, one can start combining minterms with other minterms. If two terms vary by only a single digit changing, that digit can be replaced with a dash indicating that the digit doesn't matter. Terms that can't be combined any more are marked with a "\*". When going from Size 2 to Size 4, treat '-' as a third bit value. Ex: -110 and -100 or -11- can be combined, but not -110 and 011-. (Trick: Match up the '-' first.)

Number of 1s	Minterm	0-Cube	Size 2 Implicants	Size 4 Implicants
1	m4	0100	m(4,12) -100*	m(8,9,10,11) 10--*
	m8	1000	m(8,9) 100-	m(8,10,12,14) 1--0*
			m(8,10) 10-0	
2	m9	1001	m(8,12) 1-00	m(10,11,14,15) 1-1-*
	m10	1010		
	m12	1100	m(9,11) 10-1	
			m(10,11) 101-	
3	m11	1011	m(10,14) 1-10	
	m14	1110	m(12,14) 11-0	
4	m15	1111	m(11,15) 1-11	
			m(14,15) 111-	

At this point, the terms marked with \* can be seen as a solution. That is the solution is

$$F = AB' + AD' + AC + BC'D'$$

If the karnaugh map was used, we should have obtain an expression simplier than this.

### Prime implicant chart

None of the terms can be combined any further than this, so at this point we construct an essential prime implicant table. Along the side goes the prime implicants that have just been generated, and along the top go the minterms specified earlier. The don't care terms are not placed on top - they are omitted from this section because they are not necessary inputs.

	4	8	10	11	12	15	
m(4,12)	X				X		-100 (BC'D')
m(8,9,10,11)		X	X	X			10--(AB')
m(8,10,12,14)		X	X		X		1--0 (AD')
m(10,11,14,15)			X	X		X	1-1- (AC)