

# 1. Digital Logic Circuits

Many scientific, industrial and commercial advances have been made possible by the advent of computers. Digital Logic Circuits form the basis of any digital (computer) system. In this topic, we will study the essential features of digital logic circuits, which are at the heart of digital computers. Digital Logic circuits may be subdivided into Combinational Logic Circuits and Sequential Logic Circuits. In EG1108, our focus will be on Combinational Logic Circuits. These circuits can be easily analyzed/designed using Boolean Algebra, which is the mathematics associated with binary systems. We will see how Combinational Logic Circuits can be designed and used for interesting practical problems. Circuit minimization techniques such as Karnaugh maps for simplification of combinatorial logic circuits will also be covered.

The main learning objectives for this topic are listed below.

## Learning Objectives:

- Understand basic terminology, types of logic gates (**AND, OR, NOT, NAND, NOR, XOR**)
- Understand the basic operations used in computers and other digital systems
- Basic rules of Boolean algebra, De Morgan's laws
- Universality of NAND and NOR gates
- Karnaugh maps for circuit minimization

---

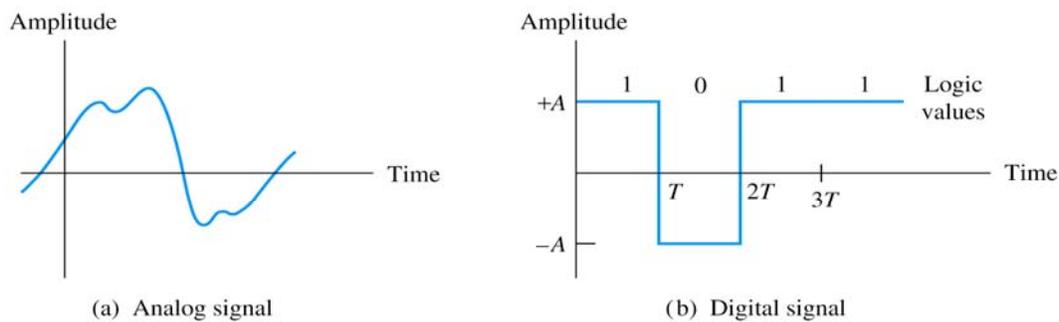
## *Recommended text for this section of the course:*

- (i) Allan R. Hambley, *Electrical Engineering Principles and Applications*, Chapter 7 (7.1, 7.3, 7.4, 7.5)
- (ii) Giorgio Rizzoni, *Principles and Applications of Electrical Engineering*, Chapter 12 (12.1, 12.3, 12.4).

## 1.1. Analog and Digital Signals

The circuits we have considered so far process analog signals, in which the signal  $v_s(t)$  varies continuously with respect to time, as shown below. Notice that the amplitude can take on any value. With the advent of digital computers, another type of signal, '**digital**' signal, is popular. In using digital signals, we sample the analog signals at various instants and convert the continuous analog signal into a set of discrete numbers. This is done by a so called 'Analog to Digital Converter (ADC)' Circuit. The digital signals can only take on values within a finite set of numbers. For example an analog voltage signal may vary continuously within 0 and 50 V. However, a digital meter may display only voltage values in steps of 1V.

Most commonly used digital signals are '**binary**' in nature. That is, they can take on one of two values (0 and 1). The numbers 0 and 1 are represented typically by two voltages in a circuit. For example, 0 can be represented by 0 volt and 1 by 5 volts. An example of a binary signal is shown in the figure on the right.



Analog signals take a continuum of amplitude values. Digital signals take a few discrete amplitudes.

### Advantages of digital systems:

Digital systems are extensively used for most practical applications in our modern world. They have the advantages of greater flexibility in computation, ease of designing using simple, reusable building blocks, greater precision and miniaturization.

The physical world, however, is analog, and hence almost all digital systems need analog-to-digital conversion before being processed by a digital system, the output of which undergoes digital-to-analog conversion to convert it back to analog form (which is what we use and understand).

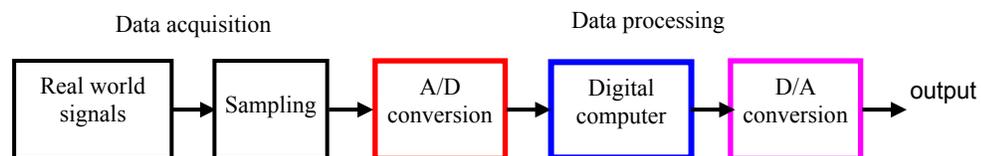


Fig. 1

We will now look at some important building blocks that form the basis of all digital systems.

## 1.2 Boolean Algebra and Logic Gates

Boolean algebra (due to George Boole) is the mathematics of digital logic and is useful in dealing with binary system of numbers. Boolean algebra is used in the analysis and synthesis of logical expressions.

Logical expressions are constructed using logical-variables and -operators. The value of any logical expression boils down to any one of the two logical constants listed below:

- True
- False

In Boolean algebra, a variable  $x$  may take any one of the two possible values 1 and 0.

$x = 1$  and  $x = 0$

which may represent respectively

- Truth or Falsehood of a statement
- On or Off states of a switch
- High (5V) or low (0V) of a voltage level



Value	Representation in	
	Mathematics	Logic Circuits
TRUE	1	Voltage HIGH (5V)
FALSE	0	Voltage LOW (0V)

Fig. 2

Electrical circuits designed to represent logical expressions are popularly known as logic circuits. Such circuits are extensively used in industrial processes, household appliances, computers, communication devices, traffic signals and microprocessors to make important logical decisions. Logic circuits are usually represented by logic operations involving boolean variables.

There are three basic logic operations as listed below:

- OR operation
- AND operation
- NOT operation

We will illustrate these basic logical operations in the following sections using boolean variables A and B. A logic gate is an electronic circuit/device which makes the logical decisions based on these operations. Logic gates have one or more inputs and only one output. The output is active only for certain input combinations. Logic gates are the building blocks of any digital circuit. Logic gates are also called switches. With the advent of integrated circuits, switches have been replaced by TTL (Transistor Transistor Logic) circuits and CMOS circuits.

### AND Operation:

AND operation is represented by  $C = A \cdot B$

Its associated TRUTH TABLE is shown below. A truth table gives the value of output variable (here C) for all combinations of input variable values (here A and B). Thus in an AND operation, the output will be 1 (True) only if all of the inputs are 1 (True).

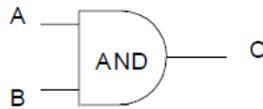


Fig. 3

Truth Table for AND Gate

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

The following relationships can be easily derived from this circuit:

$$\begin{aligned}
 A \cdot A &= A \\
 1 \cdot A &= A \\
 0 \cdot A &= 0 \\
 A \cdot \bar{A} &= 0 \\
 A \cdot B &= B \cdot A \\
 A \cdot (B \cdot C) &= (A \cdot B) \cdot C = A \cdot B \cdot C
 \end{aligned}$$

Note: The  $\cdot$  sign can be omitted when indicating an AND operation. Thus,  $C=A \cdot B$  and  $C=AB$  mean the same operation.

### OR Operation:

OR operation is represented by  $C = A + B$

Here A, B & C are logical (Boolean) variables and the + sign represents the logical addition, called an 'OR' operation. The symbol for the operation (called an OR gate) is shown in Fig. 4. Its associated TRUTH TABLE is shown below. Thus in an OR operation, the output will be 1 (True) if either of the inputs is 1 (True). If both inputs are 0 (False), only then the output will be 0 (False). Notice that though the symbol + is used, the logical addition described above does not follow the rules of normal arithmetic addition.

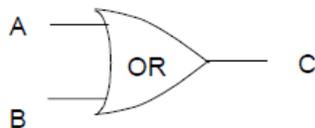


Fig. 4

Truth Table for OR Gate

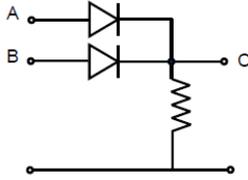
A	B	C
0	0	0
0	1	1
1	0	1
1	1	1

The following relationships can be easily derived from this circuit:

$$\begin{aligned}
 A + \bar{A} &= 1 \\
 A + A &= A \\
 0 + A &= A \\
 1 + A &= 1 \\
 (A + B) + C &= A + (B + C) = A + B + C
 \end{aligned}$$

**Electronic Circuit implementation:**

The following figure shows how an OR gate can be implemented using diodes and a resistor. In general, however, OR and other gates are implemented using electronic circuits based on active transistor (MOSFET or BJT) devices. The focus in our EG1108 module will not be on such circuit implementation of logic gates.

**NOT Operation:**

NOT operation is represented by

$$C = \bar{A}$$

The NOT gate has only one input which is then inverted by the gate. Here,  $\bar{A}$  is the 'complement' of A. The symbol and truth table for the operation are shown below:

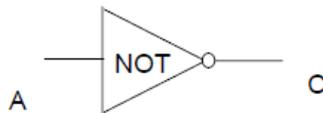


Fig. 5

Truth Table for NOT Gate

A	C
0	1
1	0

**Multivariable Logic Gates:**

Many logic gates can be implemented with more than two inputs, and for reasons of space in circuits, usually multiple input, complex gates are made. You will encounter such gates in real world. The truth tables of 3-input AND and OR gates are shown below.

In general, AND function on  $n$  variables is "1" if and only if ALL its arguments are "1". OR function on  $n$  variables is "1" if and only if at least one of its arguments is "1".

Truth table for 3-input AND gate

Input			Output
A	B	C	A.B.C
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

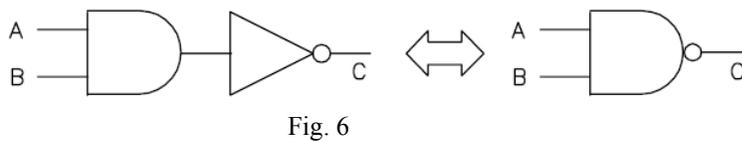
Truth table for 3-input OR gate is shown below:

Input			Output
A	B	C	$A+B+C$
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

### NAND Gate:

We could combine AND and NOT operations together to form a NAND gate. Thus the logical expression for a NAND gate is  $C = \overline{A \bullet B}$

The symbol and truth table are given in the following figure. The NAND gate symbol is given by an AND gate symbol with a circle at the output to indicate the inverting operation.

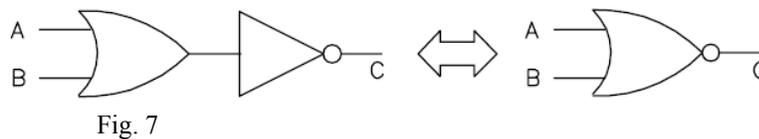


Truth Table for NAND Gate

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

### NOR Gate:

Similarly, OR and NOT gates could be combined to form a NOR gate.



Truth Table for NOR Gate

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

### 1.3 Elements of Boolean Algebra

A symbolic binary logic expression consists of binary variables and the operators AND, OR and NOT (e.g.  $A + B \bullet \bar{C}$ ). The possible values for any Boolean expression can be tabulated in a **truth table**.

Boolean algebra expressions can be implemented by interconnection of AND gates, OR gates, and inverters.

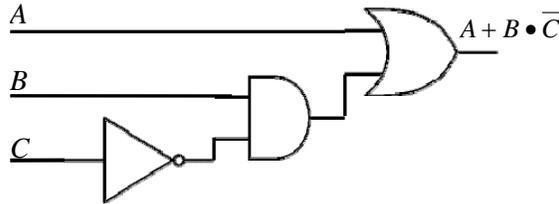


Fig. 8

As can be seen, the number of *simple* gates needed to implement an expression is equal to the number of operations in the Boolean expression. We could use the rules of Boolean Algebra or Karnaugh Maps to simplify a given Boolean expression. This would allow the given expression to be implemented using less number of gates.

### Basic Laws of Boolean Algebra

Some of the basic rules of Boolean algebra that may be used to simplify the Boolean expressions are shown below. These rules may be proved using the truth tables. Essentially, we consider all combinations of inputs and show that in all cases the LHS expression and RHS expression lead to the same result. Such a method of proving logical equations is known as proof by perfect induction.

Rule 1:	$0 + A = A$
Rule 2:	$1 + A = 1$
Rule 3:	$A + A = A$
Rule 4:	$A + \bar{A} = 1$
Rule 5:	$0 \bullet A = 0$
Rule 6:	$1 \bullet A = A$
Rule 7:	$A \bullet A = A$
Rule 8:	$A \bullet \bar{A} = 0$
Rule 9:	$A = \bar{\bar{A}}$
Rule 10:	$A + B = B + A$
Rule 11:	$A \bullet B = B \bullet A$
Rule 12:	$A + (B + C) = (A + B) + C$
Rule 13:	$(A \bullet B) \bullet C = A \bullet (B \bullet C)$
Rule 14:	$A \bullet (B + C) = A \bullet B + A \bullet C$
Rule 15:	$A + A \bullet B = A$
Rule 16:	$A \bullet (A + B) = A$
Rule 17:	$(A + B) \bullet (A + C) = A + B \bullet C$
Rule 18:	$A + \bar{A} \bullet B = A + B$

**Example 1:**

Show by perfect induction that  $A + \overline{A} \bullet B = A + B$ . [See rule 18 above].

**Solution:** Let us create the following table and show the LHS = RHS for all the values of  $A$  and  $B$ .

$A$	$B$	RHS $A + B$	$\overline{A}$	$\overline{A} \bullet B$	LHS $A + \overline{A} \bullet B$
0	0	0	1	0	0
0	1	1	1	1	1
1	0	1	0	0	1
1	1	1	0	0	1

**De Morgan's Laws:**

These laws are very useful in simplifying Boolean expressions. According to De Morgan's theorem:

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

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

Notice that the De Morgan's Laws give the link between the OR operation and the AND operation. Application of De Morgan's theorem makes it easy to design logic circuits using NAND and NOR logic gates which we will soon see.

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

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

Because of the above relationships, any logical function can be implemented by using only 1)AND and NOT gates or 2)OR and NOT gates.

Thus an OR gate can be implemented with AND and NOT gates as shown below.

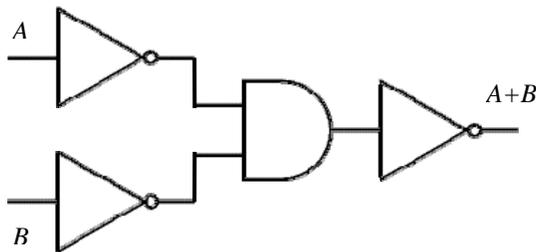


Fig. 9

De Morgan's Laws are valid for more than two variables also. For example,

$$\overline{A + B + C} = \overline{A} \bullet \overline{B} \bullet \overline{C} \quad \text{and} \quad \overline{A \bullet B \bullet C} = \overline{A} + \overline{B} + \overline{C}$$

## 1.4 Universality of NAND and NOR Gates

### Universality of NAND Gates

NAND gates can be used to implement the functions of both a NOT gate and an AND gate.

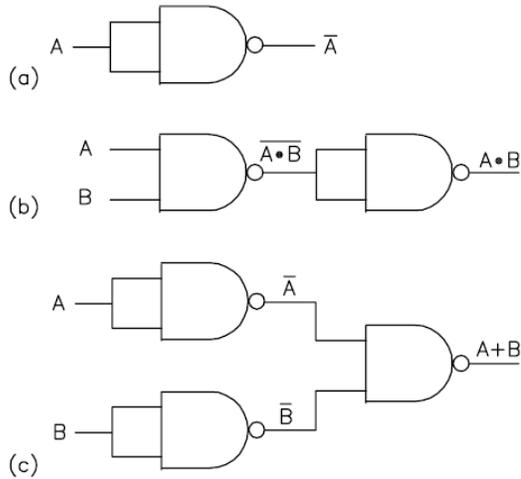


Fig. 10

Thus, any given logic function can be implemented by using NAND gates alone. For this reason, NAND gate is said to be *logically complete*.

**Example 2:** Implement  $Z = A \cdot B + C \cdot \bar{D}$  using only NAND gates.

*Solution:*

$$\begin{aligned} Z &= A \cdot B + C \cdot \bar{D} \\ &= \overline{\overline{A \cdot B + C \cdot \bar{D}}} \\ &= \overline{\overline{A \cdot B} \cdot \overline{C \cdot \bar{D}}} \\ &= \overline{\overline{A \cdot B} \cdot \overline{C \cdot \bar{D}}} \end{aligned}$$

Circuit implementation using NAND gates:

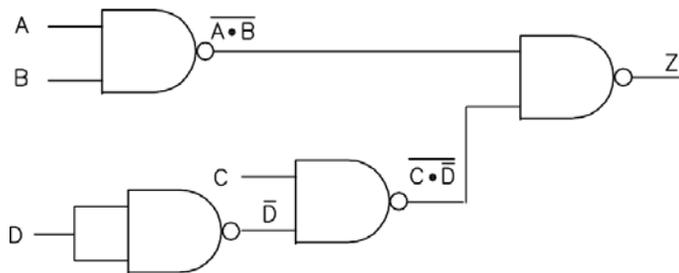


Fig. 11

### Universality of NOR Gates

Likewise, it can be shown that any logic function can be implemented using NOR gates alone. This is so because NOR gates can be used to implement the functions of both a NOT gate and an OR gate.

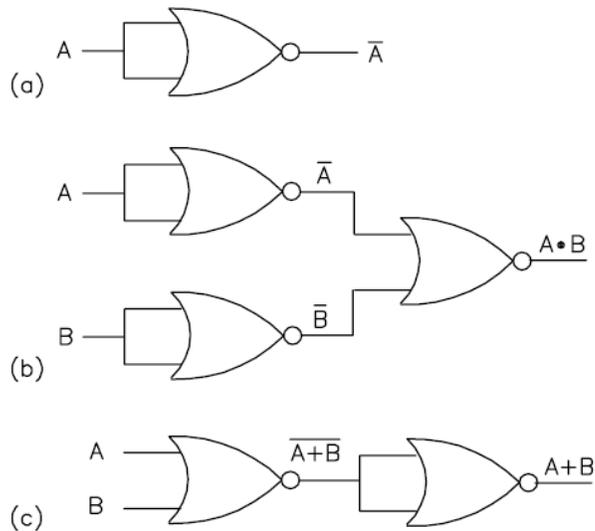


Fig. 12

A NOR gate is functionally complete because AND, OR, and NOT gates can be implemented using NOR gates alone.

**Example 3:** Implement  $Z = A \cdot B + C \cdot \bar{D}$  using only NOR gates.

*Solution:*

Circuit implementation using NOR gates:

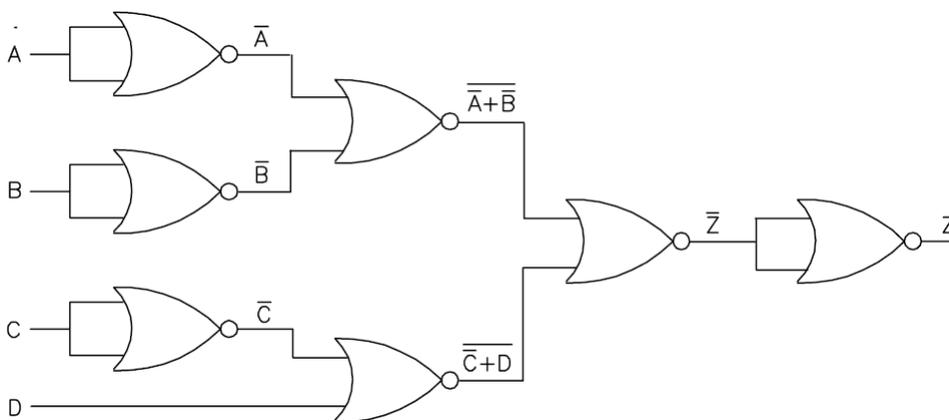


Fig. 13

### 1.5 Exclusive OR Operation:

In logic circuits, exclusive OR operation is represented as shown below.

$$C = A \oplus B$$

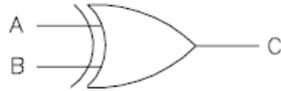


Fig. 14

Its associated TRUTH TABLE:

A	B	$C = A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

**Example 4:** Figure below shows a logical circuit that may be used to achieve exclusive OR operation. Determine the output C.

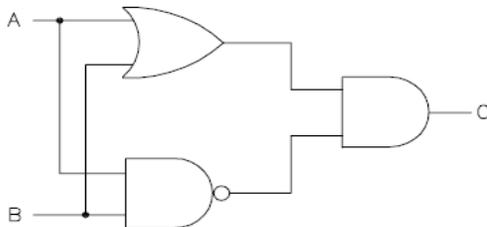


Fig. 15

**Solution:** The output of the OR gate is  $A + B$  while that of NAND gate is  $\overline{A \cdot B}$ .

Hence, the output C is given by  $C = (A + B) \cdot \overline{(A \cdot B)}$ .

**Example 5:** Simplify the expression for C and draw an equivalent logical circuit for exclusive OR operation.

**Solution:**

$$\begin{aligned}
 C &= (A + B) \cdot \overline{A \cdot B} = (A + B) \cdot (\overline{A} + \overline{B}) \\
 &= A \cdot \overline{A} + A \cdot \overline{B} + B \cdot \overline{A} + B \cdot \overline{B} \\
 &= 0 + A \cdot \overline{B} + B \cdot \overline{A} + 0 \\
 &= A \cdot \overline{B} + B \cdot \overline{A}
 \end{aligned}$$

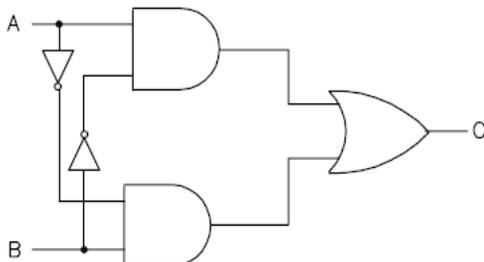


Fig. 16

## 1.6 Designing Logic Circuits

In designing digital circuits, the designer often begins with a truth table describing what the circuit should do. The design task is to determine what type of circuit will perform the function described in the truth table. Although, it may not always be obvious what kind of logic circuit would satisfy the truth table, two simple methods for designing such a circuit are found in standard form of Boolean expression called the *Sum-Of-Products* (or *SOP*) form and *Product-Of-Sums* (or *POS*) forms.

Based on the description of the problem, natural language is first translated into a truth table and Boolean expressions are found methodically using one of these two methods. The Boolean expression is then simplified using rules of Boolean algebra or Karnaugh Maps (which we will study later), so that it can be implemented using minimum number of logic gates for practical implementation.

### Sum-of-Products Implementation

As you might suspect, a Sum-Of-Products Boolean expression is literally a set of Boolean terms added (*summed*) together, each term being a multiplicative (*product*) combination of Boolean variables.

$$\langle \text{sum-of-products-expression} \rangle = \langle \text{term} \rangle + \langle \text{term} \rangle \dots + \langle \text{term} \rangle$$

Product terms that include *all of the input variables* (or their inverses) are called **minterms**. In a sum-of-products expression, we form a product of all the input variables (or their inverses) for each row of the truth table for which the result is logic 1. The output is the logical “sum” of these *minterms*

Sum-Of-Products expressions are easy to generate from truth tables as shown in the example below, by determining which rows of the table have an output of 1, writing one product term for each row, and finally summing all the product terms. This creates a Boolean expression representing the truth table as a whole.

Sum-Of-Products expressions lend themselves well to implementation as a set of AND gates (products) feeding into a single OR gate (sum).

**Example 6:** Obtain W from the truth table in the Sum Of Products (SOP) form. Draw the logical circuit to implement it.

A	B	C	W
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

Using this method, the resulting Boolean expression and its circuit implementation are shown below.

$$W = \bar{A} \cdot \bar{B} \cdot C + \bar{A} \cdot B \cdot \bar{C} + \bar{A} \cdot B \cdot C + A \cdot \bar{B} \cdot C + A \cdot B \cdot \bar{C}$$

Note that each product term in the Boolean expression includes *all of the input variables* (or their inverses).

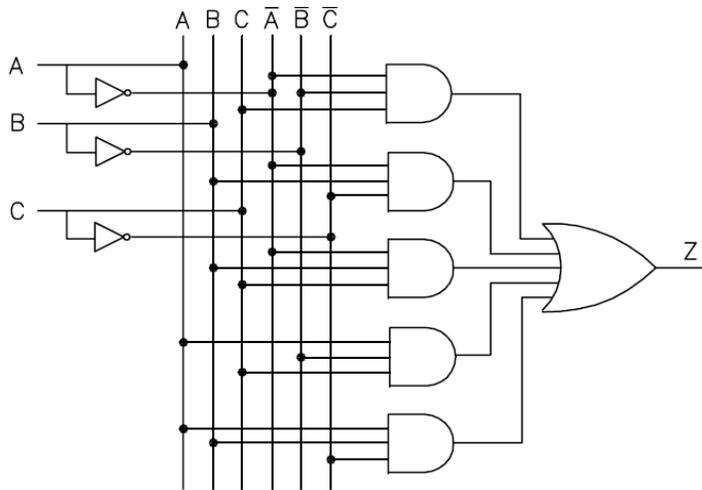


Fig. 17

**Example 7:** Simplify the logical expression  $W$  obtained in example 5 and show its implementation.

*Solution:* The Boolean expression can be simplified as follows.

$$\begin{aligned}
 W &= \bar{A} \cdot \bar{B} \cdot C + \bar{A} \cdot B \cdot \bar{C} + \bar{A} \cdot B \cdot C + A \cdot \bar{B} \cdot C + A \cdot B \cdot \bar{C} \\
 &= \bar{A} \cdot B + \bar{B} \cdot C + A \cdot B \cdot \bar{C} \\
 &= \bar{B} \cdot C + B(\bar{A} + A \cdot \bar{C}) = \bar{B} \cdot C + B(\bar{A} + \bar{C}) \\
 &= \bar{B} \cdot C + \bar{A} \cdot B + B \cdot \bar{C}
 \end{aligned}$$

Implementation of  $W$  after simplification:

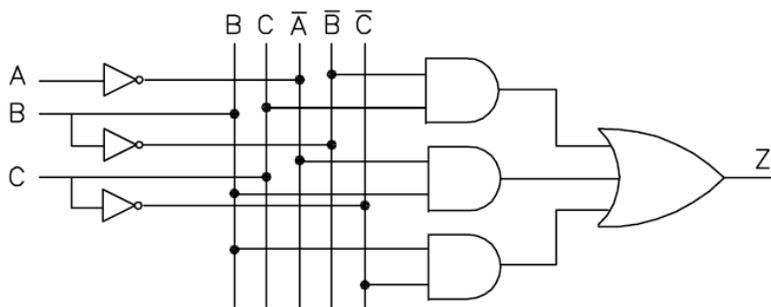


Fig. 18

## Product-of-Sums Implementation

An alternative to generating a Sum-Of-Products expression to account for all the "high" (1) output conditions in the truth table is to generate a *Product-Of-Sums*, or *POS*, expression, to account for all the "low" (0) output conditions instead. *POS* Boolean expressions can be generated from truth tables quite easily, by determining which rows of the table have an output of 0, writing one sum term for each row, and finally multiplying all the sum terms. This creates a Boolean expression representing the truth table as a whole.

These "sum" terms that include all of the input variables (or their inverses) are called *maxterms*. For POS implementation, the output variable is the **logical product of maxterms**.

Product-Of-Sums expressions lend themselves well to implementation as a set of OR gates (sums) feeding into a single AND gate (product).

**Example 8:** Find Z in terms of A, B and C in product-of-sum (POS) form from the following truth table.

A	B	C	Z
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

To begin, we identify which rows in the last truth table column have "low" (0) outputs, and write a Boolean sum term that would equal 0 for that row's input conditions. For instance, in the fifth row of the truth table, where A=1, B=0, and C=0, the sum term would be  $(\bar{A} + B + C)$ , since that term would have a value of 0 if and only if  $\bar{A}=0$ , B=0, and C=0. Similarly, maxterms for the 6<sup>th</sup> and 7<sup>th</sup> row can be found. The completed Product-Of-Sums expression, of course, is the multiplicative combination of these three sum terms.

$$Z = (\bar{A} + B + C) \bullet (\bar{A} + B + \bar{C}) \bullet (\bar{A} + \bar{B} + C)$$

Whereas a Sum-Of-Products expression could be implemented in the form of a set of AND gates with their outputs connecting to a single OR gate, a Product-Of-Sums expression can be implemented as a set of OR gates feeding into a single AND gate.

## Karnaugh Maps

From the previous examples we can see that rules of Boolean algebra can be applied in order to simplify expressions. Apart from being laborious (and requiring us to remember all the laws) this method can lead to solutions which, though they appear minimal, are not. The Karnaugh map provides a simple and straight-forward method of minimising boolean expressions. With the Karnaugh map Boolean expressions having up to four and even six variables can be simplified easily. The simplified logical expression is then used so that minimum hardware is employed in the implementation of logical circuits.

A Karnaugh map provides a pictorial method of grouping together expressions with common factors and therefore eliminating unwanted variables. The values inside the squares are copied from the output column of the truth table, therefore there is one square in the map for every row in the truth table. Around the edge of the Karnaugh map are the values of the two input variable. A is along the top and B is down the left hand side. The diagram below explains this.

### Boolean Expressions in Two Variables:

Consider the following truth table.

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

The logical expression X is given by  $X = \bar{A} \cdot \bar{B} + A \cdot B$

The Karnaugh map of the above truth table is shown in the following figure. The values inside the squares are copied from the output column of the truth table, therefore there is one square in the map for every row in the truth table. Around the edge of the Karnaugh map are the values of the two input variable A and B and their inverses.

In other words, we may say that Karnaugh map is a graphical representation of the truth table.

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

Consider the logical expression  $Y = A \cdot B + \bar{A} \cdot B$

Its Karnaugh map is shown below. The two adjacent squares may be combined together as shown by the loop.

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

Referring to the map above, the two adjacent 1's are grouped together. Through inspection it can be seen that variable A has its true and false form within the group. This eliminates variable A leaving only variable B which only has its true form. The minimised answer therefore is  $Y = B$ .

It simply means that we are combining the two terms of the above expression Y as shown below:

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

Therefore, as the variable A changes from its normal form to its inverse form ( $\bar{A}$ ) when we move from one square to the adjacent one, the simplified expression of Y will be independent of A.

Taking another example, the expression  $Z = \bar{A} \cdot B + A \cdot B + A \cdot \bar{B}$  is simplified as follows.

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

First, combining the two adjacent squares in row 1, we get B. Next, combining the two adjacent squares in column 1, we get A. Hence, we get Z as shown below:

$$Z = B + A$$

*Therefore we can easily conclude that, combining two adjacent squares in Karnaugh map eliminates one variable from the resulting Boolean expression of the corresponding squares.*

### Boolean Expressions in Three Variables:

Consider the following truth table.

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

The corresponding Boolean expression using SOP is:

$$X = \bar{A} \cdot \bar{B} \cdot \bar{C} + \bar{A} \cdot \bar{B} \cdot C + \bar{A} \cdot B \cdot \bar{C} + A \cdot B \cdot \bar{C}$$

Figure below shows the Karnaugh map of the above truth table. The expression X may be simplified by combining two adjacent squares as shown.

	AB	$A\bar{B}$	$\bar{A}\bar{B}$	$\bar{A}B$
C	0	0	1	0
$\bar{C}$	1	0	1	1

The simplified expression of X is:  $X = \bar{A} \cdot \bar{B} + B \cdot \bar{C}$

Now, consider another expression  $Y$  given below:

$$Y = A \cdot B \cdot \bar{C} + A \cdot \bar{B} \cdot \bar{C} + \bar{A} \cdot \bar{B} \cdot \bar{C} + \bar{A} \cdot B \cdot \bar{C}$$

The Karnaugh map of  $Y$  is shown below. In this case, we are able to combine four adjacent squares. Note that

	AB	A $\bar{B}$	$\bar{A}\bar{B}$	$\bar{A}B$
C	0	0	0	0
$\bar{C}$	1	1	1	1

Y can also be obtained as:

$$Y = A \cdot \bar{C}(B + \bar{B}) + \bar{A} \cdot \bar{C}(\bar{B} + B)$$

$$= A \cdot \bar{C} + \bar{A} \cdot \bar{C} = \bar{C}$$

Consider another example:

$$Z = A \cdot \bar{B} \cdot C + \bar{A} \cdot \bar{B} \cdot C + A \cdot \bar{B} \cdot \bar{C} + \bar{A} \cdot \bar{B} \cdot \bar{C}$$

The corresponding Karnaugh map is shown below:

	AB	A $\bar{B}$	$\bar{A}\bar{B}$	$\bar{A}B$
C	0	1	1	0
$\bar{C}$	0	1	1	0

In this case also, we are able to combine 4 adjacent squares. Note that

- Combining the two adjacent squares in columns 2 and 3 of row 1, the variable  $A$  gets eliminated, and we are left with  $Z_1 = \bar{B} \cdot C$
- Combining the two adjacent squares in columns 2 and 3 of row 2, the variable  $A$  gets eliminated and we are left with  $Z_2 = \bar{B} \cdot \bar{C}$

Combining these two expressions,  $Z = \bar{B} \cdot C + \bar{B} \cdot \bar{C} = \bar{B}$

Finally, let us consider another expression  $W$  below:

$$W = A \cdot B \cdot C + \bar{A} \cdot B \cdot C + A \cdot B \cdot \bar{C} + \bar{A} \cdot B \cdot \bar{C}$$

The Karnaugh map of  $W$  is shown below. Note that the resulting expression should be independent of  $A$  and  $C$ . So,  $W$  is simplified as:  $W = B$

	AB	A $\bar{B}$	$\bar{A}\bar{B}$	$\bar{A}B$
C	1	0	0	1
$\bar{C}$	1	0	0	1

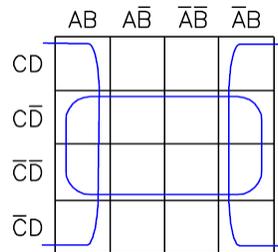
**Therefore, we can conclude that combining four adjacent squares in Karnaugh map eliminates two variables from the resulting Boolean expression of the corresponding squares.**

## Boolean Expressions in Four Variables

Knowing how to generate Gray code should allow us to build larger maps. Actually, all we need to do is look at the left to right sequence across the top of the 3-variable map, follow a similar sequence for the other two variables and write it down on the left side of the 4-variable map.

Karnaugh map of four variables  $A$ ,  $B$ ,  $C$  and  $D$  is shown in the following figure. As we have shown in the previous examples, we may easily prove that:

*Combining eight adjacent squares in Karnaugh map eliminates three variables from the resulting Boolean expression of the corresponding squares.*



**Example 9:** Simplify the Boolean expression:

$$X = \bar{A} \cdot \bar{B} \cdot \bar{C} \cdot D + \bar{A} \cdot \bar{B} \cdot C \cdot D + \bar{A} \cdot B \cdot \bar{C} \cdot D + \bar{A} \cdot B \cdot C \cdot D$$

Karnaugh map of  $X$  is shown in Figure 12.24. As we could combine 4 adjacent squares as shown below, the simplified expression should be independent of two variables. Adjacent squares in a row suggest that the resultant expression should be independent of  $B$ . Similarly, adjacent squares in a column suggest that it should also be independent of  $C$ . Hence, the simplified expression of  $X$  is given by:  $X = \bar{A} \cdot D$

	AB	A $\bar{B}$	A $\bar{B}$	A $\bar{B}$
CD	0	0	1	1
C $\bar{D}$	0	0	0	0
C $\bar{D}$	0	0	0	0
C $\bar{D}$	0	0	1	1

**Example 10:** Simplify the following Boolean expression using Karnaugh map.

$$Y = A \cdot B \cdot \bar{C} \cdot D + A \cdot \bar{B} \cdot \bar{C} \cdot D + \bar{A} \cdot \bar{B} \cdot \bar{C} \cdot D + \bar{A} \cdot B \cdot \bar{C} \cdot D + \bar{A} \cdot \bar{B} \cdot \bar{C} \cdot \bar{D} + \bar{A} \cdot B \cdot \bar{C} \cdot \bar{D} + \bar{A} \cdot B \cdot C \cdot \bar{D} + \bar{A} \cdot B \cdot C \cdot D + \bar{A} \cdot \bar{B} \cdot C \cdot D$$

*Solution:* Karnaugh map of  $Y$  is shown below. There are four loops enclosing 4-adjacent squares. First, consider the loop 1. The resulting expression for these squares should be independent of  $C$  and  $D$ . Next, consider loop 2. The resulting expression of these squares should be independent of  $B$  and  $D$ . Thirdly, consider loop 3. The resulting expression of these squares should be independent of  $B$  and  $C$ . Finally, consider loop 4.

The resulting expression of these squares should be independent of  $A$  and  $B$ . Hence, we get

$$Y = \bar{A} \cdot B + \bar{A} \cdot \bar{C} + \bar{A} \cdot D + \bar{C} \cdot D$$

	AB	A $\bar{B}$	$\bar{A}\bar{B}$	$\bar{A}B$	
CD	0	0	1	1	Loop 3
C $\bar{D}$	0	0	0	1	Loop 1
$\bar{C}\bar{D}$	0	0	1	1	Loop 2
$\bar{C}D$	1	1	1	1	Loop 4

Reductions could be done with Boolean algebra. However, the Karnaugh map is faster and easier, especially if there are many logic reductions to do.

### Karnaugh maps: Complete Simplification Process

1. Draw out the pattern of output 1's and 0's in a matrix of input values
2. Construct the K map and place 1s and 0s in the squares according to the truth table.
3. Group the isolated 1s which are *not* adjacent to any other 1s. (single loops)
4. Group any pair which contains a 1 adjacent to only one other 1. (double loops)
5. Group any quad that contains one or more 1s that have not already been grouped, *making sure to use the minimum number of groups.*
6. Group any pairs necessary to include any 1s that have not yet been grouped, *making sure to use the minimum number of groups.*
7. Form the OR sum of all the terms generated by each group.

Compared to the algebraic method, the K-map process is a more orderly process requiring fewer steps and always producing a minimum expression. It must be noted that the minimum expression is generally **NOT** unique.