Chapter IV

Boolean Algebra

Aim

The aim of this chapter is to:

- introduce Boolean algebra
- explain the fundamental laws of Boolean algebra
- elucidate the concept of truth tables

Objectives

The objectives of this chapter are to:

- enlist the types of logic gates
- explicate the types of combinatorial circuits
- determine the Boolean identities for the various operations

Learning outcome

At the end of this chapter, you will be able to:

- understand the different Boolean laws
- describe the techniques of Logic minimisation
- identify logic gates and truth tables

4.1 Introduction

Boolean algebra (or Boolean logic) is a logical calculus of truth values, developed by George Boole in the 1840s. It resembles the algebra of real numbers, but with the numeric operations of multiplication xy, addition x + y, and negation −x replaced by the respective logical operations of conjunction xoy, disjunction xVy, and negation ¬x.

4.1.1 Fundamental Laws

We imagine a logical variable, A, that takes on the values 0 or 1. If A = 0 then A = 1 and if A = 1 then A = 0. Here are some obvious identities using the AND, OR and NOT operations.

Looking at these identities you can see why the 'plus' symbol was chosen for OR and 'times' was chosen for AND.

OR AND NOT

A + 0 = A A• 0 = 0 A + A =1

A +1 = 1 A• 1 = A A• A = 0

A + A = A A• A = A A = A

A + A =1 A• A = 0

Equality

Two Boolean expressions are equal if and only if their truth tables are identical.

Associative laws

(A + B)+ C = A+ (B + C)

(AB)C = A (BC)

Distributive laws

A (B + C) = AB + AC

Related identities

(A + AB)= A

(A + B)= A+ B

(A+ B)• (A + C) = (A+ BC)

4.1.2 DeMorgan's Theorems

DeMorgan's first Theorem says that complement of sum equals the product of complements.

The LHS of the equation describes a NOR (NOT-OR) gate and the RHS of the equation describes an AND gate with inverted inputs and both the equations have the same truth table.

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

DeMorgan's second Theorem says that complement of product equals the sum of complements.

The LHS of the equation describes a NAND (NOT-AND) gate and the RHS of the equation describes an OR gate with inverted inputs and both the equations have the same truth table.

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

4.1.3 Boolean Identities

Boolean Identities for NOT operation

The NOT operation of a Boolean variable A is denoted by its complement $\bar{A}$. Following are the Boolean identities pertaining to NOT operation.

$\bar{A} = A$

It implies that the double complement of a Boolean variable is the variable itself.

For A = 0, $\bar{A}$ = 1, A = 0 and for A = 1, $\bar{A}$ = 0, A = 1 is true.

$\bar{A} + A = 1$

It means that a variable ORed with its complement always equals 1. If A = 0, $\bar{A}$ = 1 and A + $\bar{A}$ = 1 and when A = 1, $\bar{A}$ = 0 and A + $\bar{A}$ = 1 is correct.

$A \bar{A} = 0$

It means that a variable ANDed with its complement always equals 0. For two possible values of A, 0.1 = 0 when A = 0, 1.0 = 0 when A = 1 is true.

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

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

$= A + B$

Boolean identities for OR operation

The OR operation in the Boolean algebra is denoted by (+). Following are the Boolean identities for the OR operation.

Commutative Law

A + B = B + A

It implies that the input A and B of the OR gate can be interchanged without changing the output Y. It can be justified from the truth table of two-input OR gate. For A = B, it is obvious that it doesn't matter when we interchange A and B. When A = 0 and B = 1, if we interchange A and B, then it will become the case of A = 1 and B = 0 and for both these case the output is 1. Hence it doesn't matter to the output if we interchange A and B inputs.

Associative Law

A + B + C = (A + B) + C = A + (B + C)

It means that the order of combining the input variables has no effect on the output variables. This can be verified from the truth table for three-input OR gate.

A + A = A

It means that any variable ORed with itself equals the variable. We can justify this Boolean identity by substituting the two possible values of A. For A = 0, 0 + 0 = 0 and for A = 1, 1 + 1 = 1 is true (refer to truth table for two-input OR gate).

A + 1 = 1

If one input of the OR gate is high the output is high no matter what is the other input. For A = 0, 0 + 1 = 1 and for A = 1, 1 + 1 = 1 is true.

A + 0 = A

It means a Boolean variable ORed with 0 equals the variable. For A = 0, 0 + 0 = 0 and A = 1, 1 + 0 = 1 is true.

Boolean Identities for AND Operation

The AND operation is denoted by (.) in Boolean algebra. Following are the Boolean identities pertaining to AND operation.

Commutative Law

A. B = B. A

It implies that the inputs of the AND gate can be interchanged without changing the output Y which can be justified from the truth table of two-input AND gate. Note that (.) is suppressed many times and we can write the Commutative law as A B = B A.

Associative Law

A B C = (A B) C = A (B C)

It means that the order of combining the input variables has no effect on the output variables. This can be verified from the truth table for three-input AND gate.

A A = A

It means that any variable ANDed with itself equals the variable. For A = 0, 0. 0 = 0 and for A = 1, 1. 1 = 1 is true (refer to truth table for two-input AND gate).

A. 1 = A

If one input of the AND gate is high the output is equal to the input. For A = 0, 0.1 = 0 and for A = 1, 1.1 = 1 is true.

A. 0 = 0

If one input of the AND gate is low the output is low irrespective of the other input. For A = 0, 0.0 = 0 and for A = 1, 1.0 = 0 is true.

Distributive law

A (B + C) = A B + A C

We can write down the truth tables for LHS and RHS of the Boolean equation and verify they are same.

4.2 Logic Minimisation

- Given a truth table, it is always possible to write down a correct logic expression simply by forming an OR of the ANDs of all input variables for which the output is true.
- However, for an arbitrary truth table such a procedure could produce a very lengthy and cumbersome expression which might be needlessly inefficient to implement with gates.
- There are several methods for simplification of Boolean logic expressions. The process is usually called "logic minimisation" and the goal is to form a result which is efficient. Two methods we will discuss are algebraic minimisation and Karnaugh maps. For more complex expressions the Quine-McKluskey method may be appropriate. Karnaugh maps are also limited to problems with up to 4 binary inputs.

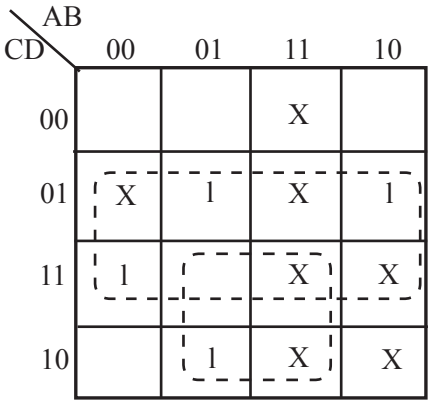


Fig. 4.1 K-map

(Source: http://www.ee.surrey.ac.uk/Projects/Labview/minimisation/isf.html)

4.2.1 Karnaugh Maps (K-Maps)

The Karnaugh map uses a rectangle divided into rows and columns in such a way that any product term in the expression to be simplified can be represented as the intersection of a row and a column. The rows and columns are labeled with each term in the expression and its complement. The labels must be arranged so that each horizontal or vertical move changes the state of one and only one variable.

A Karnaugh map is visual display of minterms required for a sum-of-product solution.

Sum-of-Products Equations and Logic Circuits (SOP)

There are four possible ways to AND two input signals that are in the complemented and un-complemented form (B, A, AB) also called as fundamental products. Table 4.1 lists each of the fundamental product producing high outputs.

A	B	Fundamental Product	Minterms
0	0	$\bar{A}\,\bar{B}$	m_0
0	1	B	m_1
1	0	$A\,\bar{B}$	m_2
1	1	AB	m_3

Table. 4.1 Fundamental products for two inputs

(Source: http://www.egyankosh.ac.in/bitstream/123456789/13358/1/Unit-8.pdf)

For instance, is high when A are B are low. The fundamental products B, and AB are also represented by minterms m_0, m_1, m_2 and m_3, where the suffix i of mi comes from the decimal equivalent of the binary number (Each individual term in standard SOP form is called as minterm). For 3 inputs A, B and C, there are 23 minterms, m_0, m_1, m_2, m_3, m_4, m_5, m_6 and m_7 as listed in table 4.2.

A	B	C	Fundamental Product	Minterms
0	0	0	$\bar{A}\,\bar{B}\,\bar{C}$	m_0
0	0	1	$\bar{A}\,\bar{B}\,\bar{C}$	m_1
0	1	0	$\bar{A}\,\bar{B}\,\bar{C}$	m_2
0	1	1	$\bar{A}\,\bar{B}\,\bar{C}$	m_3
1	0	0	$\bar{A}\,\bar{B}\,\bar{C}$	m_4
1	0	1	$\bar{A}\,\bar{B}\,\bar{C}$	m_5
1	1	0	$\bar{A}\,\bar{B}\,\bar{C}$	m_6
1	1	1	$\bar{A}\,\bar{B}\,\bar{C}$	m_7

Table 4.2 Fundamental products for three inputs

(Source: http://www.egyankosh.ac.in/bitstream/123456789/13358/1/Unit-8.pdf)

For example, when A = 1, B = 1, C = 0, the fundamental product results a high output for the case Y = A B = 11 = 1. SOP equation is a function of Boolean variables, which states the fundamental product terms or minterms, which will give a high output for the given inputs. For instance, the output Y is a function of three Boolean variables A, B and C whose minterms are listed which will give a high outputs.

Y = F (A, B, C) = Σ m (1, 2, 3, 4) = + + +

Let us try to find the truth table from the given SOP equation. List all the possible values of A, B and C in the order of increasing minterm index i.e., In the truth table, place 1s whose minterms are listed in the SOP equation and at the other places draw 0s as in Table 4.3.

A	B	C	Minterms	Y
0	0	0	m_0	0
0	0	1	m_1	1 ($\bar{A}\,\bar{B}\,\bar{C}$)
0	1	0	m_2	1 ($\bar{A}\,\bar{B}\,\bar{C}$)
0	1	1	m_3	1 ($\bar{A}\,\bar{B}\,\bar{C}$)
1	0	0	m_4	1 ($\bar{A}\,\bar{B}\,\bar{C}$)
1	0	1	m_5	0
1	1	0	m_6	0
1	1	1	m_7	0

Table 4.3 Truth table for SOP equation
(Source: http://www.egyankosh.ac.in/bitstream/123456789/13358/1/Unit-8.pdf)

Product of Sums (POS)
The Product of Sums form represents an expression as a product of maxterms. Each individual term in standard POS form is called as maxterm.
F(X, Y ...) = Product (bk + Mk), where bk is 0 or 1 and Mk is a maxterm**.**

Drawing Karnaugh Maps
Consider the following example:
Y = F (A, B) = Σ m (1, 2, 3) = B+ +
The truth table for this SOP equation is shown in Table 4.4

A	B	Y	Minterms
0	0	0	m_0
0	1	1	m_1
1	0	1	m_2
1	1	1	m_3

Table 4.4 Truth table for SOP equation
(Source: http://www.egyankosh.ac.in/bitstream/123456789/13358/1/Unit-8.pdf)

Following are the steps required to produce the K-map from the truth table:
Begin by drawing Fig 4.2 (a) Write down the value of suffix of minterms or the equivalent decimal numbers as shown in Fig. 4.2 (b)

From the truth table of Table 4.4 write down the values of Y for the corresponding minterms. The values of Y for the minterms which are there is the SOP equation is 1, for other minterms it is 0.

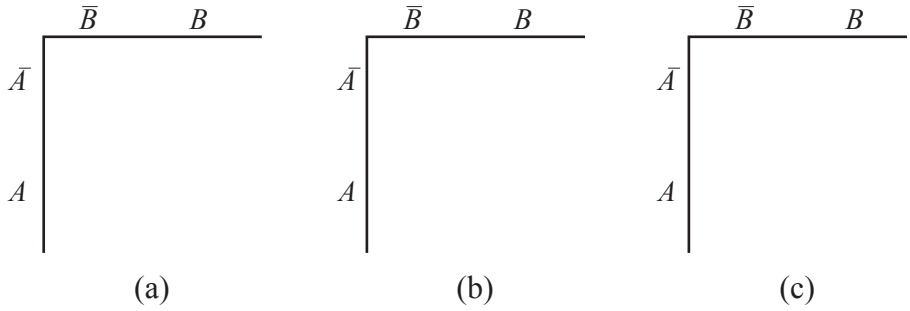


Fig. 4.2 Steps for construction of 2-variable K-map (a); (b); (c)
(Source: http://www.egyankosh.ac.in/bitstream/123456789/13358/1/Unit-8.pdf)

Don't Care Conditions

In some digital systems, some inputs are never supplied, thereby no output is visible. We call such cases "don't care conditions" and they are represented by X in the truth table as depicted in Fig. 1.3. Whenever we see an X in the truth table while encircling the 1s in the K-map to form the largest group, we can assume Xs as 1s. After it has been included in all the groups, disregard the Xs in the truth tables by assuming them as 0s. As for Fig. 1.3, the SOP equation after Karnaugh simplifications are Y = B.

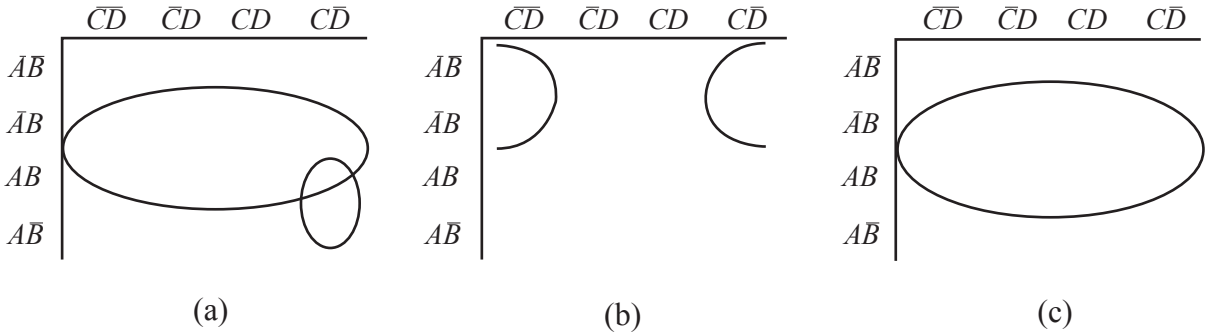


Fig. 4.3 (a) Overlapped groups; (b) Rolling; (c) Don't care conditions of K-maps
(Source: http://www.egyankosh.ac.in/bitstream/123456789/13358/1/Unit-8.pdf)

4.2.2 Quine – McCluskey Method

The Quine – McCluskey algorithm (or the method of prime implicants) is a method used for minimisation of Boolean functions which was developed by W.V. Quine and Edward J. McCluskey. 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. It is sometimes referred to as the tabulation method.

The method involves two steps:

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

Thus, logic expressions can be represented in one of the standard forms: SOP or POS and then simplified using K-maps or Quine- McCluskey method.

4.3 Combinatorial Circuits

The circuits and switching arrangements used in electronics are very complex but, although this chapter only deals with simple circuits, the functioning of all microchip circuits is based on the ideas in this chapter. The flow of electrical pulses which represent the binary digits 0 and 1 (known as bits) is controlled by combinations of electronic devices. These logic gates act as switches for the electrical pulses. Special symbols are used to represent each type of logic gate.

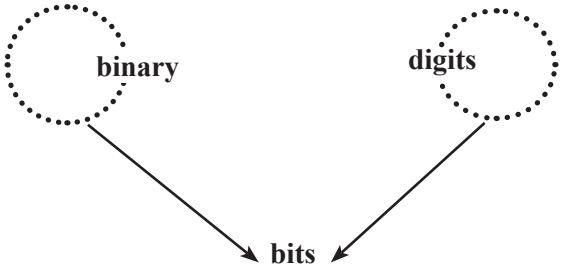


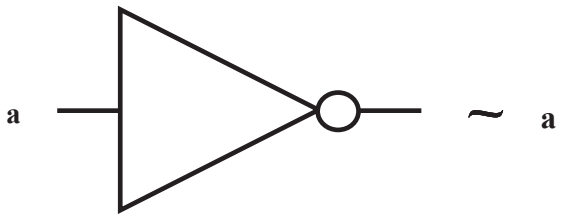
Fig. 4.4 Combinatorial circuits
(Source: http://www.cimt.plymouth.ac.uk/projects/mepres/alevel/discrete_ch11.pdf)

4.3.1 NOT Gate

The NOT Gate is capable of reversing the input pulse. The truth table for a NOT gate is as follows:

INPUT	OUTPUT
a	~a
0	1
1	0

Table 4.5 Truth table



This is a* NOT *gate

Fig. 4.5 NOT gate
(Source: http://www.cimt.plymouth.ac.uk/projects/mepres/alevel/discrete_ch11.pdf)

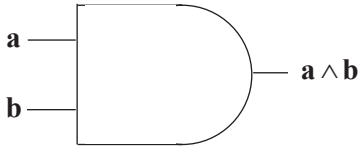
The NOT gate receives an input, either a pulse (1) or no pulse (0) and produces an output as follows:
If input a is 1, output is 0; and if input a is 0, output is 1.

4.3.2 AND Gate

The AND gate receives two inputs a and b, and produces an output denoted by $a \wedge b$. The truth table for an AND gate is as follows:

INPUT	INPUT	OUTPUT
a	b	$a \wedge b$
0	0	0
0	1	0
1	0	0
1	1	1

Table 4.6 Truth table



This is an* AND *gate

Fig. 4.6 AND gate
(Source: http://www.cimt.plymouth.ac.uk/projects/mepres/alevel/discrete_ch11.pdf)

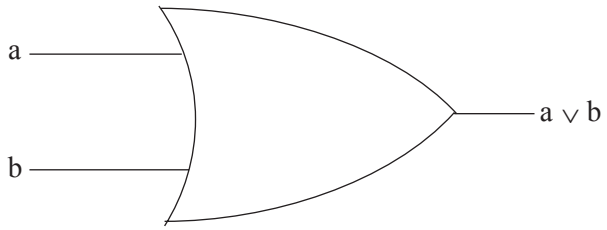
The only way that the output can be 1 is when a AND b are both 1.
In other words there needs to be an electrical pulse at a AND before the AND gate will output an electrical pulse.

4.3.3 OR Gate

The OR gate receives two inputs a and b, and produces an output denoted by $a \vee b$. The truth table for an OR gate is as follows:

INPUT	INPUT	OUTPUT
a	b	$a \vee b$
0	0	0
0	1	1
1	0	1
1	1	1

Table 4.7 Truth table



This is an* OR *gate

Fig. 4.7 OR gate
(Source: http://www.cimt.plymouth.ac.uk/projects/mepres/alevel/discrete_ch11.pdf)