Unit 2 Boolean Algebra

Structure:

2.1 Introduction

In the last unit, you studied about the different types of number systems and their conversion from one number system to other. You also studied the methods of representing negative numbers. The components in computer accept set of inputs and produce outputs depending upon their functionality. Any basic 2 state (state-1 & state-0) units can be used to construct the digital circuits. These basic units can be represented as a switch based on their state (ON or OFF). In this unit, you will study about the Boolean algebra, basics of logic gates and realization of other gates or logic function using universal gates.

Objectives:

By the end of Unit 2, the learners are able to:

- list rules & laws of Boolean algebra
- explain basic gates
- explain universal gates
- explain exclusive OR gate and exclusive NOR gate
- exercise on realizing circuits with universal gates

2.2 Rules & Laws of Boolean Algebra with Derivations

Boolean Operators

Boolean operators include sum, product and complement. The complement operator is represented by a single quote. The complement operations are the following:

0' = 1

1' = 0

The Boolean sum operator is represented by “+” or “OR”. The sum operations are the following:

0 + 0 = 0 (or) 0 OR 0 = 0

0 + 1 = 1 (or) 0 OR 1 = 1

1 + 0 = 1 (or) 1 OR 0 = 1

1 + 1 = 1 (or) 1 OR 1 = 1

The Boolean product operator is represented by “•” or “AND”. The product operations are the following:

0 • 0 = 0 (or) 0 AND 0 = 0

0 • 1 = 0 (or) 0 AND 1 = 0

1• 0 = 0 (or) 1 AND 0 = 0

1 • 1 = 1 (or) 1 AND 1 = 1

In product operations, the • operator can be dropped while using the algebraic expression, i.e., “x • y” can be represented as “xy” (because x • y = xy). The precedence rules for these Boolean operators are: complement, product, and sum.

Example:

$$\begin{aligned} 1 \cdot 1 + (0 + 1)' &= 1 \cdot 1 + 1' \\ &= 1 \cdot 1 + 0 \\ &= 1 + 0 \\ &= 1 \end{aligned}$$

Boolean Identities

Identity	Name
(a')' = a	Involution Law
a + a' = 1	Complementarity
a • a' = 0	

$a + a = a$ Idempotent Laws
$a \bullet a = a$

$a + 0 = a$ Identity Laws
$a \bullet 1 = a$

$a + 1 = 1$ Dominance Laws
$a \bullet 0 = 0$

$a + b = b + a$ Commutative Laws
$ab = ba$

$a + (b + c) = (a + b) + c$ Associative Laws
$a(bc) = (ab)c$

$a + bc = (a + b)(a + c)$ Distributive Laws
$a(b + c) = ab + ac$

$(ab)' = a' + b'$ DeMorgans Laws
$(a + b)' = a'b'$

$a + (ab) = a$ Absorption Laws
$a(a + b) = a$

$a + a'b = a + b$ Redundancy Laws
$a(a' + b) = ab$

$ab + a'c + bc = ab + a'c$ Consensus Laws
$(a+b)(a'+c)(a+c) = (a+b)(a'+c)$

In Boolean algebra, the duality principle will be applicable to all the Boolean identities. The difference between the given identity and its dual is "•" interchanged with "+", and "0" interchanged with "1". Any Boolean theorem which has been proved will be true for the dual of the theorem.

Truth tables or other identities can be used to prove the Boolean identities. For example, the distributive law is proven true by the last two columns of the table 2.1.

Table 2.1: Distributive law

X	Y	Z	X + Z	XY	YZ	Y (X + Z)	XY + YZ
1	1	1	1	1	1	1	1
1	1	0	1	1	0	1	1
1	0	1	1	0	0	0	0
1	0	0	1	0	0	0	0
0	1	1	1	0	1	1	1
0	1	0	0	0	0	0	0
0	0	1	1	0	0	0	0
0	0	0	0	0	0	0	0

A proof of the absorption law x (x + y) = x using identities:

X (x + y) = (x + 0) (x + y) identity law

= x + 0 • y distributive law

= x + y • 0 commutative law

= x + 0 dominance law

= x identity law

In building digital circuits by using Boolean algebra, the following two basic problems need to be solved.

1) How the Boolean expression can be derived from a given table of values for a Boolean function?
2) Can the Boolean function be represented with an optimum set of operators?

First, question, How the Boolean expression can be derived from a given table 2.2 of values for a Boolean function?

Table 2.2: Boolean expression

x	y	z	F	G
1	1	1	0	0
1	1	0	0	0
1	0	1	0	0
1	0	0	0	0
0	1	1	1	1
0	1	0	0	0
0	0	1	0	1
0	0	0	0	0

From the above table of values, the function F is 1 when y = z = 1 and x = 0. And hence, we get the expression x'yz i.e., the value of the function F will be 1 if and only if x' = y = z = 1.

The function G will have value 1 in two cases: y = z = 1, x =0, and x = y = 0, z = 1. From the above information, the expression for G can be represented in the sum of two product terms: x'yz + x'y'z

The above method explains the way to construct a Boolean expression from the values of a Boolean function. A Boolean product m1, m2, m3…mn is said to be a **minterm** of Boolean variables x1, x2, x3,…xn if mi = xi or mi = xi'. Only if all the variables of minterm are 1, the minterm product will result in 1. For the 1st scenario discussed above, x'yz will only be the minterm that has the value 1. In other words, a Boolean expression for a Boolean function can be quoted as the sum of the minterms. Minterms referred here are those with value 1 for a given combination.

Boolean sum discussed above is also referred to as sum of products (SOP) expansion or disjunctive normal form.

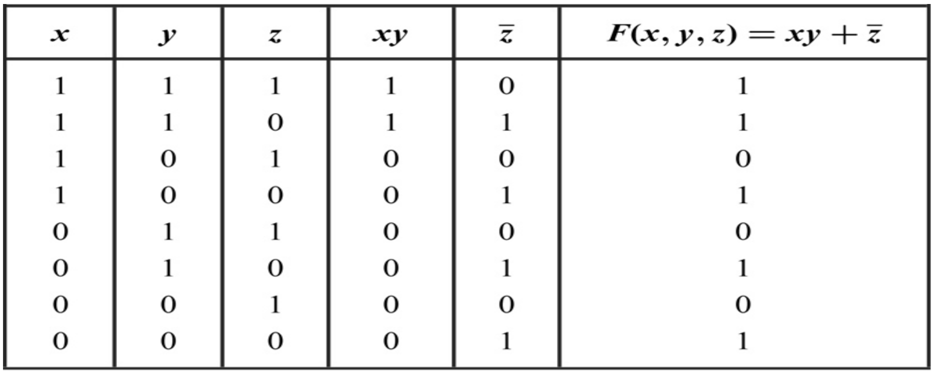
Maxterm is the dual of Minterm. It is also referred to as product of sums (POS) or conjunctive normal form. Duality can be observed from the table 2.3.

Table 2.3: Duality

x	y	z	minterm	maxterm
0	0	0	a'b'c'	a+b+c
0	0	1	a'b'c	a+b+c'
0	1	0	a'bc'	a+b'+c
0	1	1	a'bc	a+b'+c'
1	0	0	ab'c'	a'+b+c
1	0	1	ab'c	a'+b+c'
1	1	0	abc'	a'+b'+c
1	1	1	abc	a'+b'+c'

Now for the Boolean function F(x,y,z) = xy + z^{I} , refer the table 2.4.

Table 2.4: F(x,y,z) = xy + z^{I}



x	y	z	xy	$\bar{z}$	$F(x,y,z) = xy + \bar{z}$
1	1	1	1	0	1
1	1	0	1	1	1
1	0	1	0	0	0
1	0	0	0	1	1
0	1	1	0	0	0
0	1	0	0	1	1
0	0	1	0	0	0
0	0	0	0	1	1

Here the minterm expression will be xyz + xyz’ + x y’z’ + x’yz’ + x’ y’z’ and the max term expression will be (x’+y+z’)(x+y’+z’) (x+y+z’)

2.3 Basic Gates (NOT, AND & OR)

Boolean algebra is a deductive mathematical system closed over the values zero and one (false and true). A *binary operator* “°” defined over this set of values for a given set of Boolean inputs, output results in a single Boolean value. For example, the Boolean AND operator output results in a single Boolean for any combination of two Boolean inputs.

In algebra systems, there are basic set of postulates and initial assumptions which system follows. Using the basic set of postulates of algebra system, the additional set of properties, rules and other theorems can be deduced. Boolean algebra systems often employ the following postulates:

- *Closure*. The Boolean system is said to be *closed* with respect to a given binary operator if, a Boolean result has been obtained for every pair of Boolean input values, For example, logical OR is said to be closed in the Boolean system because it produces a Boolean result for the accepted input Boolean operands..
- *Commutativity*. A binary operator "°" is said to be commutative if X°Y = Y°X for all possible Boolean values X and Y.
- *Associativity*. A binary operator "°" is said to be associative if (X ° Y) ° Z = X ° (Y ° Z) for all Boolean values X, Y, and Z.
- *Distribution*. Two binary operators "°" and "%" are distributive if X ° (Y % Z) = (X ° Y) % (X ° Z) for all Boolean values X, Y, and Z.
- *Identity*. A Boolean value I is said to be the *identity element* with respect to some binary operator " °" if X ° I = Y.
- *Inverse*. A Boolean value I is said to be the *inverse element* with respect to some binary operator "°" if A ° I = B and B-A (i.e., B is the opposite value of A in a Boolean system).

0 and 1 are only possible values in the Boolean system, which are often called as false and true respectively.

The logical AND operation is represented by the symbol "•"; e.g., for Boolean values A and B logical AND result is obtained using A • B expression. The symbol "•" is dropped when the single letter variable names are used; Therefore, logical AND of the variables can be represented as AB. ANDing of variables A and B is also called as the product of variables A and B.

The logical OR operation is represented using the symbol "+"; e.g., for Boolean variables A and B the logical OR operation can be represented as A + B. ORing of Boolean variables A and B is also called as sum of variables A and B.

The unary operator's in digital system are Logical complement, negation, or not. The logical negation is denoted by ('). For example, logical NOT of A is denoted by A'.

If a single Boolean expression has many different operators, the result of that expression depends on the *precedence* of the operators. In Boolean algebra, the following precedence (from lowest to highest) for the Boolean operators are used: logical OR, logical AND, logical NOT, parenthesis. The logical OR and logical AND operators are *left associative.* In a single expression, if there two operators adjacent to each other with the same precedence, the expression must be evaluated from left to right. The logical NOT operation is right associative. As logical NOT is unary operator, the results of both left associativity and right associativity will result in same value.

The following sets of postulates are used in digital systems:

Postulate 1: Boolean algebra is closed under the AND, OR, and NOT operations.

Postulate 2: The identity element with respect to • is one and + is zero. There is no identity element with respect to logical NOT.

Postulate 3: The • and + Boolean operators are commutative.

Postulate 4: • and + Boolean operators are distributive with respect to one another. That is,
X • (Y + Z) = (X • Y) + (X • Z) and
X + (Y • Z) = (X + Y) • (X + Z).

Postulate 5: For every value X there exists a value X’ such that

X•X’ = 0 and

X+X’ = 1. X’ value is the logical complement (or NOT) of X.

Postulate 6: • and + Boolean operators are both associative. That is,

(X•Y)•Z = X•(Y•Z) and

(X+Y)+Z = X+(Y+Z).

Using the above postulates any theorem in Boolean algebra can be proved. Following are the some important theorems in Boolean algebra using which a digital system can be constructed.
Theorem-1: X + X = X
Theorem-2: X • X = X
Theorem-3: X + 0 = X
Theorem-4: X • 1 = X
Theorem-5: X • 0 = 0
Theorem-6: X + 1 = 1

Theorem-7: (X + Y)’ = X’ • Y’
Theorem-8: (X • Y)’ = X’ + Y’
Theorem-9: X + X•Y = X
Theorem-10: X •(X + Y) = X
Theorem-11: X + X’Y = X+Y
Theorem-12: X’ • (X + Y’) = X’Y’
Theorem-13: XY + XY’ = X
Theorem-14: (X’+Y’) • (X’ + Y) = Y’
Theorem-15: X + X’ = 1
Theorem-16: X • X’ = 0

A Boolean *expression* is a combination of ones, zeros, and *literals* which are separated by Boolean operators. A literal is an unprimed or primed (negated) variable name. In further discussion, we consider that a single alphabetic character for all variable names. A specific Boolean expression is considered as Boolean function; In general the name of Boolean functions can be any alphabetic character with a possible superscript. For example:

F = A + BC

The Boolean function F first computes the logical AND operation of B and C and then performs logical OR operation with A. If A=1, B=0, and C=1, then value one is returned by the function F (1+0•1 = 1). Table 2.5 can be used as alternate way to represent a Boolean function.

Table 2.5: AND Truth Table

AND	0	1
0	0	0
1	0	1

Truth tables will be more convenient and natural for any binary operators with two input variables. However, when the Boolean function F mentioned above is considered, we can observe that the function F has 3 input variables. For such conditions the above mentioned truth table (for two input variables) cannot be used, instead a truth table for three input variables can be used. Truth table for any number of input variables can be constructed easily. One of the methods to construct the truth table for three or four variables is explained in the following example:

In the table 2.6 for input variables A & B, the possible combinations of zeros and ones are given in four different columns of the table.

Table 2.6

A	**B**
0	0
0	1
1	0
1	1

The table 2.7 shows the truth table for OR function.

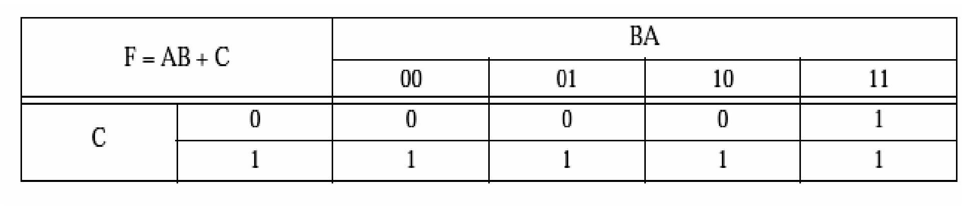
Table 2.7: Truth Table for OR function

OR	0	1
0	0	1
1	1	1

From the truth table 2.9 it can be observed that the possible combinations of zeros and ones for variables C and D are given in the four different rows of the truth table. Alternate method to construct the truth tables is shown in table 2.10. The current method provides a compact representation when compared to the method explained above; also using this method it is easier to fill in the rows and columns of the table. In digital systems one can create many unique Boolean functions. By using the theorems which have been mentioned above, we can show that two or more Boolean functions can be equivalent, i.e., for any set of input combinations, Boolean functions produces same outputs. For example, F=(A+B)’ and F=A’B’ are two different Boolean functions. By theorem seven, it can be easily proved that the two Boolean functions are equivalent. If the number of input variables are fixed, then only finite number of unique Boolean functions possible. For example, for two input variables there will be only 16 possible Boolean functions and for three input variables there will be only 256 unique Boolean functions. For n input variables, the number of unique Boolean functions are $2^\wedge(2^n)$ (i.e. two raised to the two raised to the nth power). For example for three input variable, number of unique Boolean functions are $2^\wedge(2^3) = 256$.

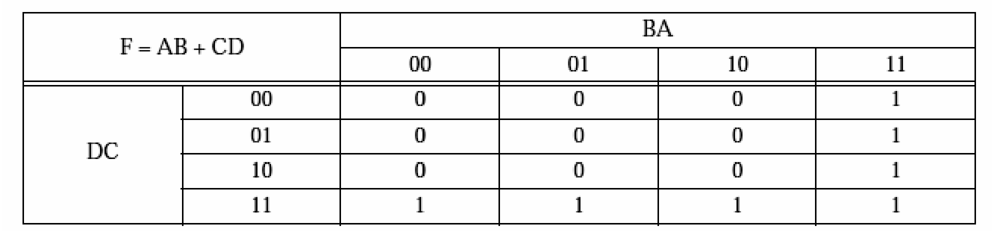
Tables 2.8 and 2.9 show truth tables for three variables and four variables respectively.

Table 2.8: Truth Table for a Function with Three Variables



F = AB + C		BA			
		00	01	10	11
C	0	0	0	0	1
	1	1	1	1	1

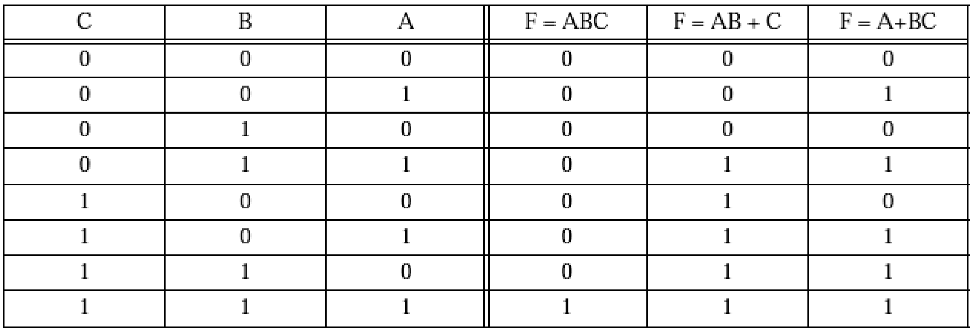
Table 2.9: Truth Table for a Function with Four Variables



F = AB + CD		BA			
		00	01	10	11
DC	00	0	0	0	1
	01	0	0	0	1
	10	0	0	0	1
	11	1	1	1	1

The table 2.10 shows the another format for truth tables

Table 2.10: Another Format for Truth Table



C	B	A	F = ABC	F = AB + C	F = A+BC
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	0	0
0	1	1	0	1	1
1	0	0	0	1	0
1	0	1	0	1	1
1	1	0	0	1	1
1	1	1	1	1	1

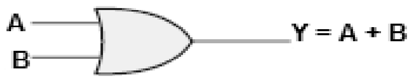
From the above discussion, it is clear that we have only three basic logic operations: OR, AND and NOT. So we have three basic gates to produce basic logic operations. These are OR gate, AND gate and NOT gate. Before we study these basic gates let us see what a logic gate is.

A logic gate is an electronic circuit which has one or more inputs but only one output. Logic gate produces logical operation on binary numbers.

Now let us study basic logic gates.

OR gate: OR gate has two or more inputs and only one output. The operation of this gate is such that it produces a high output (i.e. logic 1) when one or more of inputs are high and it produces a low output (i.e. logic 0) when all the inputs are low.

The logic symbol, truth table and Boolean expression for 2-input OR gate is shown in figure 2.1.



A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

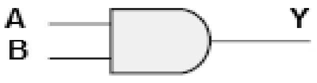
(a) (b) (c)

Figure 2.1: OR gate (a) Logic symbol (b) Boolean expression (c) Truth table

Note that A and B are the inputs and Y is the output. The Boolean expression Y = A + B is read as Y equals A OR B.

AND gate: AND gate has two or more inputs and only one output. This gate produces a high output (i.e. logic 1) when all of inputs are high and it produces a low output (i.e. logic 0) when one or more of its inputs are low.

The logic symbol, truth table and Boolean expression for 2-input OR gate is shown in figure 2.2.



Y= A. B

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

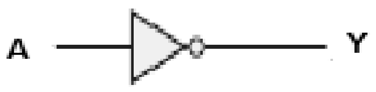
(a) (b) (c)

Figure 2.2: AND gate (a) Logic symbol (b) Boolean expression (c) Truth table

Note that A and B are the inputs and Y is the output. The Boolean expression Y = A .B is read as Y equals A AND B.

NOT gate: A NOT gate has only one input and only one output. This gate produces the output which is the inversion (i.e. complement) of the input. Suppose if the input is 1(HIGH), then its output is 0 i.e. logic 0 (LOW). NOT gate is also known as an *inverter.* If the input variable is A, the inverted output is known as NOT A. This can also be shown as A', or A with a bar over the top i.e. $\overline{A}$.

The logic symbol, truth table and Boolean expression for NOT gate is shown in figure 2.3.



(a)

$Y = \overline{A}$

(b)

A	Y
0	1
1	0

(c)

Figure 2.3: NOT gate (a) Logic symbol (b) Boolean expression (c) Truth table

Note that A is the input and Y is the output. The Boolean expression Y = $\overline{A}$ is read as Y equals NOT A. This can also be read as Y equals A bar.

2.4 Universal Gates (NAND & NOR)

The NAND gate is a digital logic gate that behaves in a manner that corresponds to the truth table to the left. When both the inputs to NAND gate are high, then output of the gate will be LOW. The output of gate will be HIGH when one or both the inputs of NAND gate are LOW. As any type of gate or Boolean function can be implemented using NAND gates, they are also known as Universal gates.

The functional completeness of the NAND gate has to be taken as advantage while constructing logic circuits in digital systems. Constructing the complicated logic circuits using NAND gates yields more compact results and cost saving when compared to the logic circuits constructed using AND, OR and NOT gates.

An n-input NAND gates can also be constructed. When all the inputs to NAND gate are high, then output of the gate will be LOW. The output of gate will be HIGH when one or all the inputs of NAND gate are LOW. N-input NAND gates instead of operating as simple binary operator, they operate as n-ary operators. Algebraically, these can be expressed as the function NAND (a, b,..., n), which is logically equivalent to NOT (a AND b AND ... AND n).

There are two symbols for NAND gates: the 'distinctive' symbol (refer figure 2.4) and the 'rectangular' symbol (refer figure 2.5).

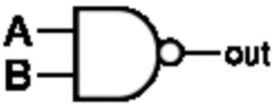


Figure 2.4: Distinctive NAND Symbol

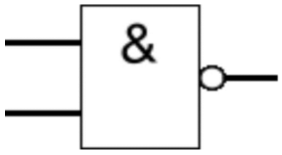


Figure 2.5: Rectangular NAND symbol

NAND operator "|" as follows:

x	y	x | y
0	0	1
0	1	1
1	0	1
1	1	0

The **NOR** gate is a digital logic gate and it behaves such that when both the inputs of NOR gate are LOW, the output of the gate will be HIGH. A LOW output results when one or both inputs of the NOR gate are HIGH. Negation of OR gate output results in NOR. NOR is a functionally complete operation – As any type of gate or Boolean function can be implemented using NOR gates, they are also known as Universal gates. As the OR operator can change LOW to HIGH but not vice versa, it is said to be monotonic

In most of the circuit implementations including CMOS (Complementary Metal Oxide semiconductor) and TTL (Transistor-Transistor Logic), the negation is not freely available in such logic families, NOR gate followed by an inverter can be used to implement OR gate. A significant exception is some forms of the domino logic family. The figure 2.6 shows the NOR symbol.

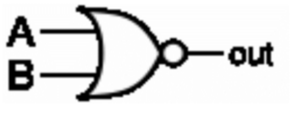


Figure 2.6: American NOR symbol

The figure 2.7 shows the IEC NOR symbol.

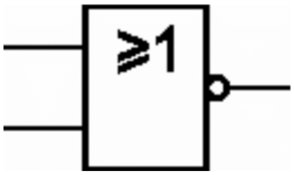


Figure 2.7: IEC NOR symbol

NOR operator: ?

x	y	x ? y
0	0	1
0	1	0
1	0	0
1	1	0

Now to prove that we can construct any Boolean function using only NAND gates, we need only show how to build an inverter (NOT), AND gate, and OR gate from a NAND (since we can create any Boolean function using only AND, NOT, and OR). Building an inverter is easy; just connect the two inputs together. Once we build an inverter, building an AND gate is easy – just invert the output of a NAND gate. After all, NOT (NOT (A AND B)) is equivalent to A. AND B. Of course, it takes *two* NAND gates to construct a single AND. We have already discussed about AND, OR and NOT Gates. Let us once again look at their logic symbols shown in figure 2.8 as these are taken as references while constructing the these gates using universal gates i.e using NOR and NAND gates.

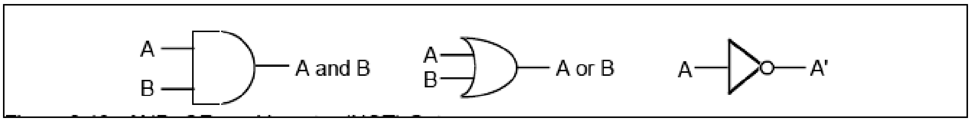


Figure 2.8: AND, OR and Inverted NOT Gates

Using DeMorgan's theorems, an OR gate can be easily constructed from NAND gates.

DeMorgan's Theorem 1 : (X OR Y)^{I} = X^{I} AND Y^{I}. i.e. $(X + Y)^{I} = X^{I} . Y^{I}$

Demorgan's Theorem 2: (X AND Y)^{I} = X^{I} OR Y^{I}. i.e. $(X . Y)^{I} = X^{I} + Y^{I}$

Now you might be wondering why we would even bother with this. After all, why not just use logical AND, OR, and inverter gates directly? Following are the reasons for this.

NAND gate and NOR gates are called universal gates because any logic function or any logic gate can be realized using either only NAND gates or only NOR gates. For example, we can construct or realize the operation of OR, AND, and NOT gates using only NAND gates or only NOR gates.

First, when compared to other gates NAND gates are less expensive to construct. Second, constructing complex digital systems using same building blocks will be easier than constructing the digital systems using different basic blocks. The correspondence between NAND and NOR logic is orthogonal to the correspondence between the two canonical forms appearing in this unit (sum of minterms vs. product of maxterms). While NOR logic is useful for many circuits, most electronic designs use NAND logic.

16 possible Boolean functions of two variables are shown in table 2.11.

Table 2.11: 16 possible Boolean functions of two variables

Function #	Description
0	Zero or Clear. Always returns zero regardless of A and B input values.
1	Logical NOR (NOT (A OR B)) = (A+B)'
2	Inhibition = BA' (B, not A). Also equivalent to B>A or A < B.
3	NOT A. Ignores B and returns A'.
4	Inhibition = AB' (A, not B). Also equivalent to A>B or B<A.
5	NOT B. Returns B' and ignores A
6	Exclusive-or (XOR) = A ⊕ B. Also equivalent to A≠B.
7	Logical NAND (NOT (A AND B)) = (A•B)'
8	Logical AND = A•B. Returns A AND B.
9	Equivalence = (A = B). Also known as exclusive-NOR (not exclusive-or).
10	Copy B. Returns the value of B and ignores A's value.
11	Implication, B implies A = A + B'. (if B then A). Also equivalent to B >= A.
12	Copy A. Returns the value of A and ignores B's value.
13	Implication, A implies B = B + A' (if A then B). Also equivalent to A >= B.
14	Logical OR = A+B. Returns A OR B.
15	One or Set. Always returns one regardless of A and B input values.

2.5 Exclusive-OR and Exclusive-NOR Gates

The XOR gate (sometimes EOR gate) is a digital logic gate that implements exclusive disjunction – it behaves according to the truth table to the right. If either one of the input is HIGH, then the output will be HIGH. The output of the gate results a LOW value when either the inputs of the gate are LOW or both are HIGH.

This function is called addition modulo 2. In digital systems, the binary adder are implemented using XOR gate. A half adder consists of an XOR gate and an AND gate. The table 2.12 shows the truth table of XOR gate.

Table 2.12: Truth table of XOR gate

INPUT		**OUTPUT**
A	**B**	**A XOR B**
0	0	0
0	1	1
1	0	1
1	1	0

The figure 2.9 shows the logic symbol of two input XOR gate and figure 2.10 shows its rectangular symbol.

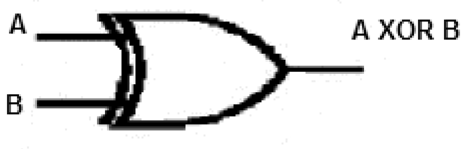


Figure 2.9: Logic Symbol of XOR gate

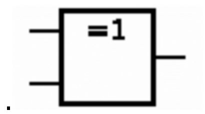


Figure 2.10: Rectangular XOR symbol

X-NOR Gate

The XNOR gate is a digital logic gate whose function is the inverse of the exclusive OR (XOR) gate. The two-input version implements logical equality, behaving according to the truth table to the right. If the inputs to XOR gate are same, then the output will be HIGH. A LOW output will result if both the inputs to the XOR gate are not same. The table 2.13 shows the truth table of XNOR gate.

Table 2.13: Truth table of XNOR gate

INPUT		OUTPUT
A	**B**	**A XNOR B**
0	0	1
0	1	0
1	0	0
1	1	1

The figure 2.11 shows the logic symbol of two input XOR gate and figure 2.12 shows its rectangular symbol.

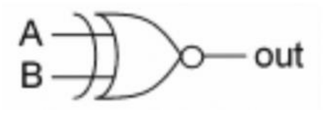


Figure 2.11: X-NOR symbol

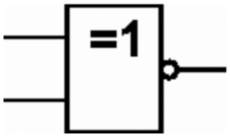


Figure 2.12: Rectangular X-NOR symbol

The XOR or Mod-2 addition operation is defined by the equation

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

An alternative way of expressing this relationship is

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

The laws of Association, Commutation and Distribution are also valid for the XOR operation. They are

$A \oplus (B \oplus C) = (A \oplus B) \oplus C$	Association
$A \oplus B = B \oplus A$	Commutation
$A(B \oplus C) = AB \oplus AC$	Distribution

If Boolean algebraic equations are written in terms of the XOR function, the following identities may prove useful:

$$A \oplus A = 0 \quad A \oplus \bar{A} = 1$$
$$A \oplus 1 = \bar{A} \quad A \oplus 0 = A$$
$$A + B = A \oplus B \oplus AB$$
$$A + B = A \oplus B \text{ if } AB = 0$$

If $A \oplus B = C$ then $A = B \oplus C$, $B = A \oplus C$ and $A \oplus B \oplus C = 0$
$A_1 \oplus A_2 \oplus \ldots \oplus A_n = 0$ for an even number of variables of value 1, and 1 for an odd number of variables of value 1.

XNOR. Exclusive NOR = NOT of (A XOR B) or $\overline{A \oplus B}$,

Which is sometimes referred to as the coincidence function and is written as $\mathbf{A \odot B}$.

Now we can construct any circuit for any given Boolean expression using logic gates. Let see the example 1.

Example 1: Draw a logic circuit for (A + B) C.

Solution: The logic circuit for (A + B) C. is shown in figure 2.12

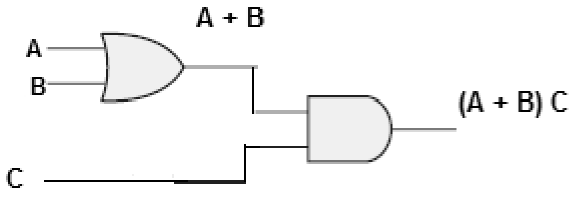


Figure 2.12: logic circuit for the expression (A + B)C.

2.6 Exercise on Realizing Circuits with Universal Gates

Now let see the realization of other circuits or gates using only NOR gate/s.

Realizing Circuits with NOR Gates:

The figure 2.13 shows the construction of NOT gate using NOR Gate.

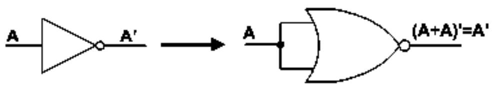


Figure 2.13: NOT Gate Construction using NOR Gate

The figure 2.14 shows the construction of OR gate using NOR Gates only.

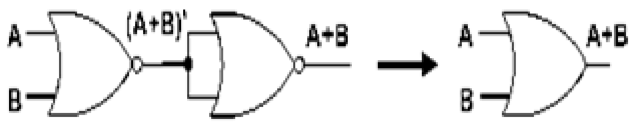


Figure 2.14: OR Gate Construction using NOR Gates

The figure 2.15 shows the construction of AND Gate using NOR Gates only.

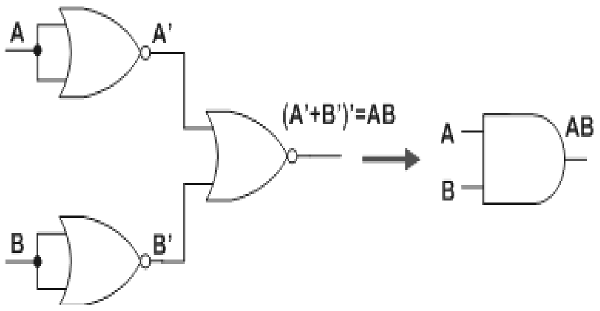


Figure 2.15: AND Gate Construction using NOR Gates

The figure 2.16 shows the construction of NAND Gate using NOR Gates only.

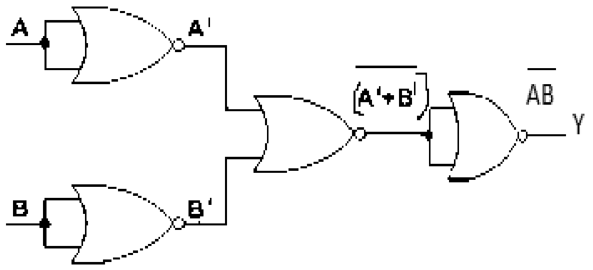


Figure 2.16: NAND Gate Construction using NOR Gates

The figures 2.17 and 2.18 show the construction of XNOR and XOR gates using NOR Gates only.

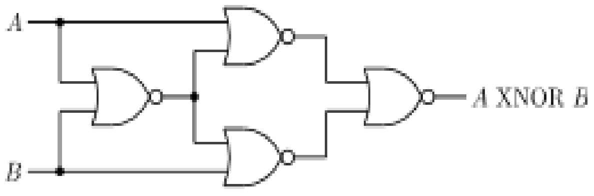


Figure 2.17: XNOR Gate Construction using NOR Gates

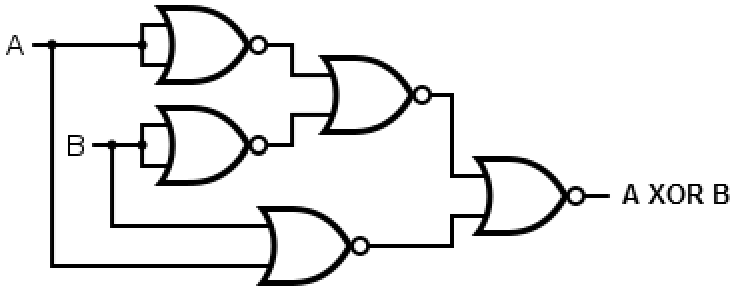


Figure 2.18: XOR Gate Construction using NOR Gates

Realizing Circuits with NAND Gates: The figures 2.19 through 2.24 shows the various gates realized using NAND gates only.

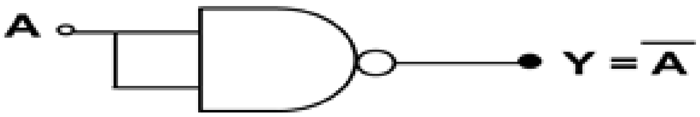


Figure 2.19: NOT Gate construction using NAND Gate

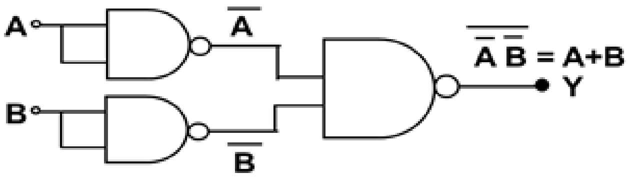


Figure 2.20: OR Gate construction using NAND Gate

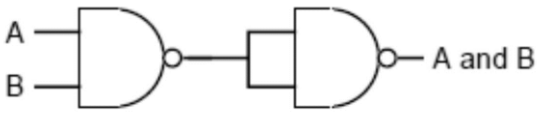


Figure 2.21: AND construction using NAND Gate

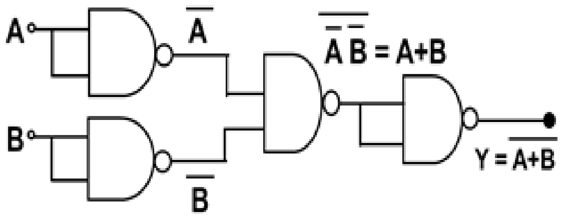


Figure 2.22: NOR construction using only NAND Gates

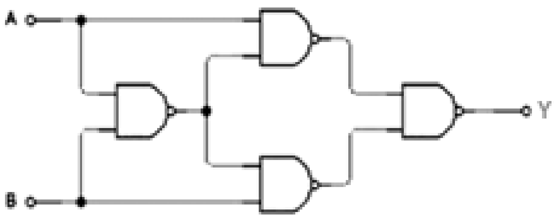


Figure 2.23: XOR Gate construction using NAND Gate

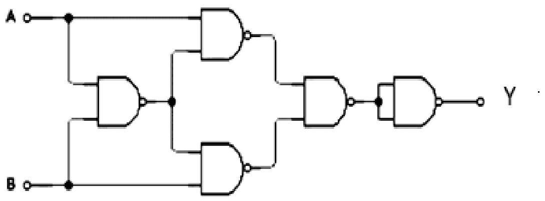


Figure 2.24: XNOR Gate construction using NAND gate

Self Assessment Questions

1. If x,y,z contains values 0,1,1 then (x+z) y is _______.
2. x+y.0 is ____________ Law.
3. x+1 =1 is __________ Law.
4. Boolean product for 1 and 0 in AND has ____ value.
5. Boolean sum for 1 and 0 in OR has _____ value.
6. A Boolean *expression* is a combination of ones, zeros, twos and *literals* which are separated connected by Boolean operators. (True or False ?)
7. The NAND Gate output for input values 0 and 1 is _______.
8. The NOR Gate output for input values 1 and 0 is _______.
9. The XNOR gate is a digital logic gate whose function is the inverse of the exclusive OR (XOR) gate. (State True or False?).
10. Draw a logic circuit for AB + $\overline{AC}$
11. Draw a logic circuit for $\overline{(A+B)}$ (C + D) $\overline{C}$.
12. The XOR or Mod-2 addition operation is defined by the equation is _________.
13. Exclusive NOR is defined as __________.

2.7 Summary

Let us recapitulate the important concepts discussed in this unit:

A logic gate is an electronic circuit which has one or more inputs but only one output. Logic gate produces logical operation on binary numbers.

- A logic gate is an elementary building block of a digital circuit.
- There are three fundamental logic gates namely, AND, OR and NOT.
- We have other logic gates like NAND, NOR, XOR and XNOR.
- NAND and NOR gates are called the universal gates.

2.8 Terminal Questions

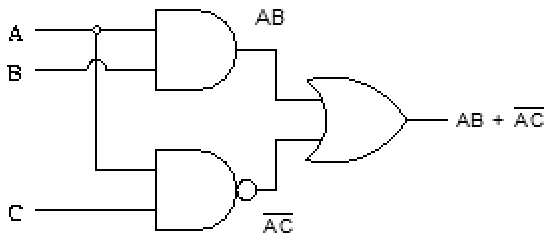
1. What is a logic gate?
2. List the fundamental logical gates.
3. Why NAND and NOR gates are called as universal gates?
4. How AND gate can be realized using NOR gate?
5. How OR gate can be realized using NAND gate?
6. Give the truth table of XOR gates for two inputs.
7. Draw a logic circuit for (A + B) C.
8. Draw a logic circuit for $A + BC + \overline{D}$.
9. Draw a logic circuit for $\overline{AB} + AC$.
10. Draw a logic circuit for $(A + B)(C + D)\overline{C}$.

2.9 Answers

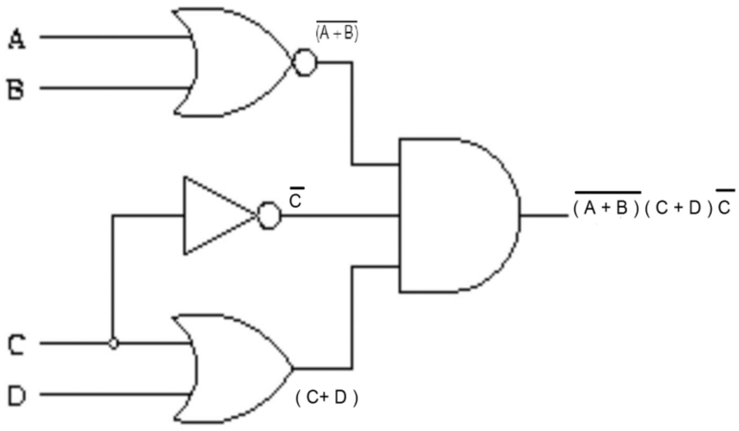
Self Assessment Questions

1. 1
2. Commutative
3. Dominance
4. 0
5. 1
6. False
7. 1
8. 0
9. True

10.



11.



12. $A \oplus B = \overline{A}B + A\overline{B}$

13. NOT of (A XOR B)

Terminal Questions

1. Refer to section 2.3
2. Refer to section 2.3
3. Refer to section 2.4
4. Refer to section 2.6
5. Refer to section 2.6
6. Refer to section 2.6
7. Refer to section 2.5 for method
8. Refer to section 2.5 for method
9. Refer to section 2.5 for method
10. Refer to section 2.5 for method