COMP 001 – INTRODUCTION TO COMPUTING

UNIT 3

Hardware

Part 2: Digital Logic System

Learning Outcomes:

- Define what Boolean algebra is
- Identify the different logic gates
- Illustrate the representation of the different logic gates
- Convert Boolean algebra expression into a logic circuit
- Create truth tables for the corresponding logic circuits and Boolean expression
- Explain basic theorems and postulates on digital logic system

Boolean algebra

Boolean algebra, named after the English mathematician George Boole, is a branch of mathematics that deals with operations on logical values with binary variables. The category of algebra in which the variable's values are the truth values, **true** and **false**, ordinarily denoted *1* and *0* respectively. Generally, Boolean algebra is a **branch of mathematics** that deals with the **manipulation of variables** that can assume only two truth values, **true or false**, denoted by **1** and **0**, respectively.

It is used to analyze and simplify digital circuits or digital gates. Instead of using arithmetic operators like addition, subtraction, and multiplication, utilize three basic logical operators: **AND**, **OR**, and **NOT**. It has been fundamental in the development of digital electronics and is provided for in all modern programming languages. It is also used in set theory and statistics.

The **AND** (*Conjunction*) function takes two or more Boolean input values and generates a single output. The function **output is 1 if and only if *every* input value is 1**. **If one or more inputs are 0**, the function **output is 0.** Symbolically, the AND function is represented using the dot (·) or Λ symbol, or by simply concatenating the inputs. For example, applying the AND function in the Boolean values **a** and **b** could be expressed as a . b, a ∧ b, or simply ab. The AND operation is known as the "Boolean product".

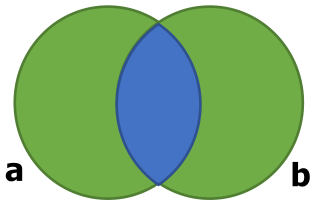
- If A = 1, B = 1, then A . B = 1
- If A = 1, B = 0, Or A = 0, B = 1, then A . B = 0
- If A = 0, B = 0, then A . B = 0

The **OR** (*Disjunction*) function also has two or more Boolean inputs, but its **output is 1 if *any* of its inputs is 1**. It is **only 0 when all its inputs are 0**. It is denoted using the + or V symbol; the OR of **a** and **b** is expressed as a + b or a V b. The OR operation is known as the "Boolean sum".

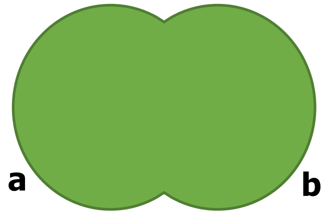
- If A = 1, B = 1, then A + B = 1
- If A = 1, B = 0, Or A = 0, B = 1, then A + B = 1
- If A = 0, B = 0, then A + B = 0

The **NOT** (*Negation*) function, unlike the AND and OR functions, takes in **only one inp**ut. **Its output is the opposite of the input value**. That is, when the input is 0, the output is 1, and when the input is 1 the output is 0. There are several ways to represent the NOT of as input, say *a*, including *a*', ~*a*, !*a*, /*a*, ¬*a*, and $\overline{a}$. There are several other functions that can be created.

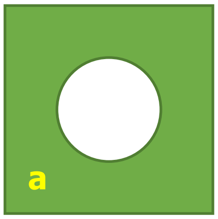
- If A = 1, then using NOT operation we have (A)’ = 0
- If A = 0, then using the NOT operation we have (A)’ = 1



a ∧ b



a ∨ b



¬*a*

Boolean Expression

A logical statement that results in a Boolean value, either True or False. Sometimes, synonyms are used to express the statement such as ‘Yes’ for ‘True’ and ‘No’ for ‘False’. Also, 1 and 0 are used for digital circuits for True and False, respectively.

Boolean expressions are statements that use logical operators, i.e., AND, OR, XOR, and NOT. Thus, if we write **1 AND 1 = 1**, then it is a Boolean expression.

Truth Tables

A truth table is a convenient mechanism to specify the **output** of any **Boolean function**, from the most fundamental to the most complex, **for all possible combinations of input values**. A table that gives all the possible values of logical variables and the combination of the variables. It is possible to convert the Boolean equation into a truth table.

a	b	a ∧ b	a V b	¬a
1	1	1	1	0
1	0	0	1	0
0	1	0	1	1
0	1	0	0	1

Truth tables are a nice way to show how these fundamental functions work and most useful when used to specify the behavior of more complex functions.

Look at a truth table for the following Boolean function: **F (x,y,z)** = $x\bar{z}+y$

x	y	z	$\bar{z}$	$x\bar{z}$	$x\bar{z}+y$
0	0	0	1	0	0
0	0	1	0	0	0
0	1	0	0	0	1
0	1	1	0	0	1
1	0	0	1	1	1
1	0	1	0	0	0
1	1	0	1	1	1
1	1	1	0	0	1

As seen in the given table above, there is a rule of precedence. Like Arithmetic, Boolean operations follow also the rules of precedence or priority.

Operator	Symbol	Precedence
NOT	$\overline{\ }$ *(or)* ¬	Highest
AND	**.** *(or)* ∧	Middle
OR	**+** *(or)* ∨	Lowest

Boolean Expression with Truth Table

Example #1

Boolean Expression: $xz+x\overline{y}+\overline{x}z$

x	y	z	xz	$\overline{y}$	$x\overline{y}$	$\overline{x}$	$\overline{x}z$	$xz+x\overline{y}+\overline{x}z$
0	0	0	**0**	**1**	**0**	**1**	**0**	**0**
0	0	1	**0**	**1**	**0**	**1**	**1**	**1**
0	1	0	**0**	**0**	**0**	**1**	**0**	**0**
0	1	1	**0**	**0**	**0**	**1**	**1**	**1**
1	0	0	**0**	**1**	**1**	**0**	**0**	**1**
1	0	1	**1**	**1**	**1**	**0**	**0**	**1**
1	1	0	**0**	**0**	**0**	**0**	**0**	**0**
1	1	1	**1**	**0**	**0**	**0**	**0**	**1**

Example #2

Boolean Expression: $(x \wedge y) \vee (y \wedge \neg z)$

x	y	z	$x \wedge y$	$\neg z$	$y \wedge \neg z$	$(x \wedge y) \vee (y \wedge \neg z)$
0	0	0	**0**	**1**	**0**	**0**
0	0	1	**0**	**0**	**0**	**0**
0	1	0	**0**	**1**	**1**	**1**
0	1	1	**0**	**0**	**0**	**0**
1	0	0	**0**	**1**	**0**	**0**
1	0	1	**0**	**0**	**0**	**0**
1	1	0	**1**	**1**	**1**	**1**
1	1	1	**1**	**0**	**0**	**1**

Example #3

Boolean Expression: $\overline{x}yz + x\overline{y}z + xy\overline{z} + xyz$

x	y	z	$\overline{x}$	$\overline{y}$	$\overline{z}$	$\overline{x}yz$	$x\overline{y}z$	$xy\overline{z}$	xyz	$\overline{x}yz + x\overline{y}z + xy\overline{z} + xyz$
0	0	0	**1**	**1**	**1**	**0**	**0**	**0**	**0**	**0**
0	0	1	**1**	**1**	**0**	**0**	**0**	**0**	**0**	**0**
0	1	0	**1**	**0**	**1**	**0**	**0**	**0**	**0**	**0**
0	1	1	**1**	**0**	**0**	**1**	**0**	**0**	**0**	**1**
1	0	0	**0**	**1**	**1**	**0**	**0**	**0**	**0**	**0**
1	0	1	**0**	**1**	**0**	**0**	**1**	**0**	**0**	**1**
1	1	0	**0**	**0**	**1**	**0**	**0**	**1**	**0**	**1**
1	1	1	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**

Practice Exercise 3.2.1

Complete the Truth table below corresponding to the given Boolean Expression.

Boolean Function: **$AB+A\overline{B}C+\overline{C}D$**

A	B	C	D	$\overline{B}$	$\overline{C}$	AB	$A\overline{B}C$	$\overline{C}D$	$AB+A\overline{B}C+\overline{C}D$
0	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	0	0	0						
1	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						

Answer:

A	B	C	D	$\overline{B}$	$\overline{C}$	AB	$A\overline{B}C$	$\overline{C}D$	$AB+A\overline{B}C+\overline{C}D$
0	0	0	0	1	1	0	0	0	0
0	0	0	1	1	1	0	0	1	1
0	0	1	0	1	0	0	0	0	0
0	0	1	1	1	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0
0	1	0	1	0	1	0	0	1	1
0	1	1	0	0	0	0	0	0	0
0	1	1	1	0	0	0	0	0	0
1	0	0	0	1	1	0	0	0	0
1	0	0	1	1	1	0	0	1	1
1	0	1	0	1	0	0	1	0	1
1	0	1	1	1	0	0	1	0	1
1	1	0	0	0	1	1	0	0	1
1	1	0	1	0	1	1	0	1	1
1	1	1	0	0	0	1	0	0	1
1	1	1	1	0	0	1	0	0	1

Mathematically speaking, an algebra is a theory encompassing a set of elements and a set of operations performed on those elements. For our Boolean algebra, we have seen that the set of elements is {0, 1} and the set of operations is {AND, OR, NOT}. An algebra must also have a set of **axioms** (also called

postulates) that are essentially the most basic rules or properties of the algebra. These are the Boolean algebra laws and theorems which are the set of rules that are required to reduce or simplify any given complex Boolean expression. The following is a list of Boolean algebra laws that are most commonly used.

Boolean Law	Description	Expression
Identity Law	‘identity’ means that when 0 is added i.e., ‘OR’, and when 0 is ‘OR’ with a variable it gives the variable back	A + 0 = A A . 1 = A
Null Law	‘null’ means that when 1 is added i.e., ‘OR’, and when 0 is ‘AND’ with a variable it gives 1 and 0 respectively	A + 1 = 1 A . 0 = 0
Idempotent Law	when a function takes one value for both its inputs, the value of the function is equal to the value of the input	A + A = A A . A = A
Inverse Law	states that for any variable, the sum of the variable and its complement (negation) always equals the universal set	¬(¬A) = A A . ¬A = 0 A + ¬A = 1
Absorption Law	reduction in a complicated expression to a simpler one by absorbing like terms	A + (A . B) = A = A + (A . B) = (A . 1) + (A . B) = A (1 + B) = A A . (A + B) = A = A (A + B) = (A + 0) . (A + B) = A + (0 . B) = A
Consensus Law	It is the conjunction of all the unique literals of the terms, excluding the literal that appears unnegated in one term and negated in the other.	A . (A’ + B) = AB = A . A’ + A . B = 0 + AB = AB A + (A’B) = A + B = A(1 + B) + A’B = A + AB + A’B = A + B(A + A’) = A + B
Commutative Law	the order of operands does not affect the result when performing these operations	A + B = B + A A . B = B . A
Associative Law	when only one type of operation occurs in a function, the order in which it is performed does not change the value of the function	(A + B) + C = A + (B + C) (A . B) . C = A . (B . C)
Distributive Law	uses both 2-operand functions, with each being distributed over the	A + (BC) = (A + B)(A + C) A . (B + C) = (A . B) + (A . C)

	other	
De Morgan’s Law	First Law: the complement of the product (AND) of two Boolean variables (or expressions) is equal to the sum(OR) of the complement of each Boolean variable (or expression	Such A = 1, B = 0, therefore (A . B)’ = A’ + B’ (1 . 0)’ = (1)’ + (0)’ (0)’ = 0 + 1 1 = 1
	Second Law: the complement of the sum (OR) of two Boolean variables (or expressions) is equal to the product (AND) of the complement of each Boolean variable (or expression)	Such A = 1, B = 0, therefore (A + B)’ = A’ . B’ (1 + 0)’ = (1)’ . (0)’ (1)’ = 0 . 1 0 = 0

Example #1

- Convert A+B+AC to its simplest form.

Boolean Law	**Proof**
Commutative Law	A + B + (A . C) = A + (A . C) + B
Absorption Law *(A + (A . B) = A*	**A + (A . C)** + B = A + B
	A + B

- Thus, A+B+AC = A + B

Prove

A	**B**	**C**	**AC**	**A+B+AC**	**A+B**
0	0	0	0	**0**	**0**
0	0	1	0	**0**	**0**
0	1	0	0	**1**	**1**
0	1	1	0	**1**	**1**
1	0	0	0	**1**	**1**
1	0	1	1	**1**	**1**
1	1	0	0	**1**	**1**
1	1	1	1	**1**	**1**

Example #2

- Convert (A+B)(A+C) to its simplest form.

Boolean Law	**Proof**
Distributed Law *(A . (B + C) = (A . B) + (A . C))*	(A + B) . (A + C) = (A **.** A) + (A . C) + (B . A) + (B **.** C)
Idempotent Law *(A . A = A)*	(A.A) + (A.C) + (B.A) + (B.C) = **A** + (A.C) + (B.A) + (B.C)
Absorption Law *(A + (A . B) = A*	A + (A.C) + (B.A) + (B.C) = **A(1+C)** + (B.A) + (B.C)
Commutative and Null Law *(A + 1 = 1)*	A(**C+1**) + (B.A) + (B.C) = **A(1)** + (B.A) + (B.C)
Identity Law *(A.1 = A)*	**A.(1)** + (B.A) + (B.C) = A + (B.A) + (B.C)
Commutative and Absorption Law *(A + (A . B) = A(1+B))*	**A + (A.B)** + (B.C) = A(1+B) + (B.C)
Commutative and Null Law *(A + 1 = 1)*	A(**B+1**) + (B.C) = A(1) + (B.C)
Identity Law *(A.1 = A)*	**A(1)** + (B.C) = A + (B.C)
	A + BC

- Thus, (A+B)(A+C) = A + BC

Prove

A	B	C	A+B	A+C	BC	(A+B)(A+C)	A+BC
0	0	0	0	0	0	**0**	**0**
0	0	1	0	1	0	**0**	**0**
0	1	0	1	0	0	**0**	**0**
0	1	1	1	1	1	**1**	**1**
1	0	0	1	1	0	**1**	**1**
1	0	1	1	1	0	**1**	**1**
1	1	0	1	1	0	**1**	**1**
1	1	1	1	1	1	**1**	**1**

Example #3

- Simplify the following Boolean expression: AB($\overline{B}$C+AC).

Boolean Law	**Proof**
Distributed Law *(A . (B + C) = (A . B) + (A . C))*	AB($\overline{B}$C+AC) = A.B.$\overline{B}$.C + A.B.A.C
Associative Law	A.B.$\overline{B}$.C + A.B.A.C = A.B.$\overline{B}$.C + A.A.B.C
Idempotent Law *(A . A = A)*	A.B.$\overline{B}$.C + **A.A**.B.C = A.B.$\overline{B}$.C + A.B.C
Inverse Law *(A . ¬A = 0)*	A.**B.$\overline{B}$**.C + A.B.C = A.0.C + A.B.C
Null Law *(A . 0 = 0)*	**A.0.C** + A.B.C = 0 + A.B.C
Identity Law (A + 0 = A)	**0** + A.B.C = A.B.C
	ABC

- Thus, AB($\overline{B}$C+AC) = ABC

A	B	C	$\overline{B}$	AB	$\overline{B}$C	AC	$\overline{B}$C+AC	AB($\overline{B}$C+AC)	ABC
0	0	0	1	0	0	0	0	**0**	**0**
0	0	1	1	0	1	0	1	**0**	**0**
0	1	0	0	0	0	0	0	**0**	**0**
0	1	1	0	0	0	0	0	**0**	**0**
1	0	0	1	0	0	0	0	**0**	**0**
1	0	1	1	0	1	1	1	**0**	**0**
1	1	0	0	1	0	0	0	**0**	**0**
1	1	1	0	1	0	1	1	**1**	**1**

Practice Exercise 3.2.2

Simplify the following Boolean Expression.

1. AB + A$\overline{B}$
2. $\overline{A}$BC + AC
3. (A+B$\overline{C}$)(A$\overline{B}$+C)
4. $\overline{A}$B+AB$\overline{C}$+B$\overline{C}$+A$\overline{B}$ $\overline{C}$

Answer:

1. A
2. C(A+B)
3. $A(\overline{B}+C)$
4. $B(\overline{A} + \overline{C})$

Logic Gates

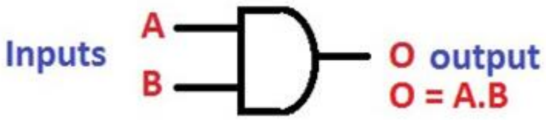
Chips with integrated circuits that can perform these tasks were created by engineers. In order to implement the AND function internally, two or more pins on the chip must be connected to the function's inputs. To transfer data to these inputs, circuit designers connect these pins to other parts of their circuitry or to external inputs. An additional pin on the device is linked to the function's output. When necessary, circuit designers can send this value to or from other areas of the circuit. A logic gate is the component of the chip that carries out the operation. Logic gates serve as a foundational element for digital circuits that can carry out the essential logical operations of digital circuits.

A logic gate is a fundamental building block of digital circuits that performs a logical operation on one or more binary inputs and produces a single binary output. Logic gates can be implemented using electrical circuits, optical devices, or transistors and resistors. They are used to execute numerous logical operations that are required by any digital circuit. The simplest logic gate of all is the inverter, which performs the Boolean NOT function.

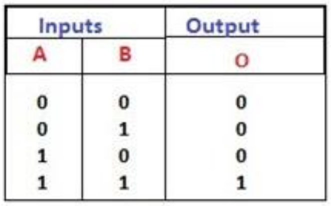
Logic gates make decisions based on a combination of digital signals coming from their inputs. Most logic gates have two inputs and one output wherein logic gates are based on Boolean algebra. At any given moment, every terminal is in one of the two binary conditions, false or true. False represents 0, and true represents 1. Depending on the type of logic gate being used and the combination of inputs, the binary output will differ.

There are several types of logic gates, each with its own specific function based on Boolean logic, which deals with binary variables and logical operations. Some of the common logic gates include:

1. **AND Gate**: It produces an output of 1 (or TRUE) only if all its inputs are 1; otherwise, it produces an output of 0 (or FALSE).



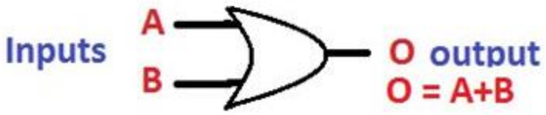
Symbol



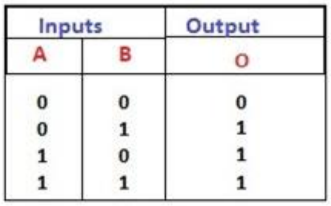
Inputs		Output
A	B	O
0	0	0
0	1	0
1	0	0
1	1	1

Truth table

2. **OR Gate**: This gate produces an output of 1 if at least one of its inputs is 1. It only produces an output of 0 if all inputs are 0.



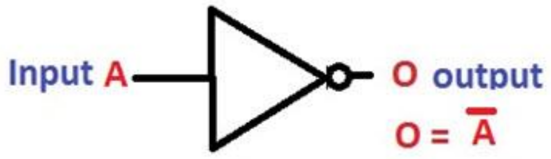
Symbol

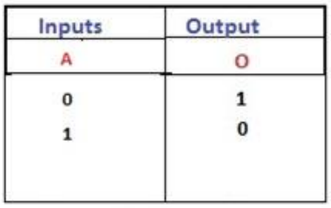


Inputs		Output
A	B	O
0	0	0
0	1	1
1	0	1
1	1	1

Truth table

3. **NOT Gate** (*Inverter*): It has only one input and produces the opposite (complementary) value at the output. If the input is 1, the output is 0, and vice versa.

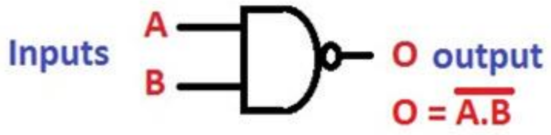


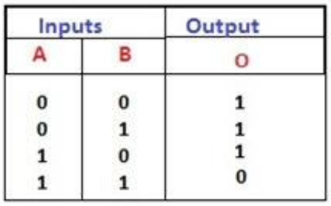


Inputs	Output
A	O
0	1
1	0

Symbol Truth table

4. **NAND Gate**: This gate is a combination of an AND gate followed by a NOT gate. It produces the opposite result of an AND gate; its output is 0 only when all inputs are 1.

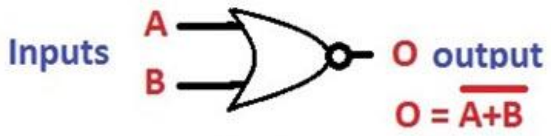


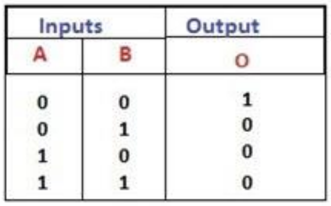


Inputs		Output
A	B	O
0	0	1
0	1	1
1	0	1
1	1	0

Symbol Truth table

5. **NOR Gate**: Similar to a combination of an OR gate followed by a NOT gate, this gate produces an output of 1 only if all inputs are 0.

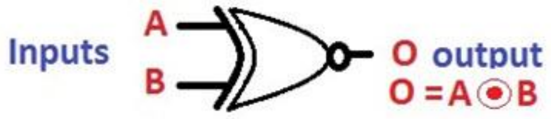


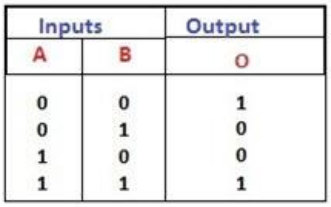


Inputs		Output
A	B	O
0	0	1
0	1	0
1	0	0
1	1	0

Symbol Truth table

6. **XOR Gate** (*Exclusive OR*): It produces an output of 1 if the number of 1 inputs is odd; otherwise, it produces an output of 0.





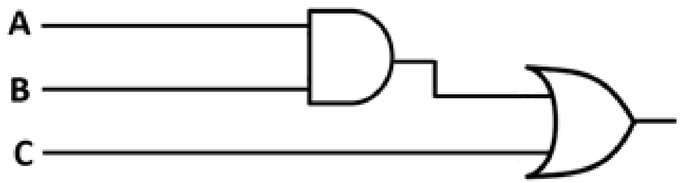
Inputs		Output
A	B	O
0	0	1
0	1	0
1	0	0
1	1	1

Symbol Truth table

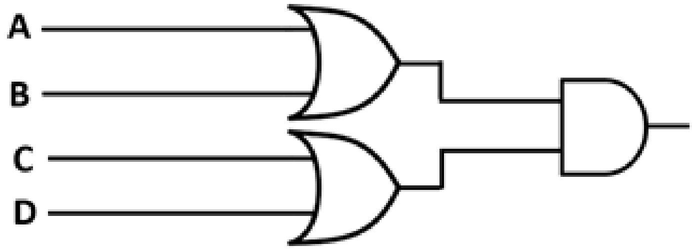
Example #1.

Translate the given Boolean expression **AB+C** into a logic circuit.

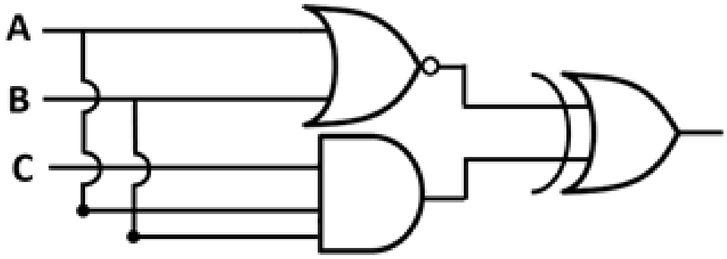
Solution:



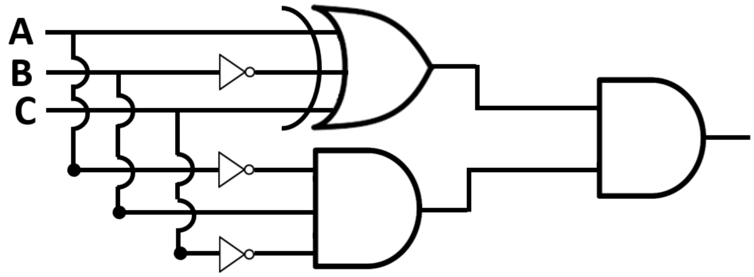
Example #2.
Want is the equivalent digital logic circuit of the Boolean expression (A+B)(C+D).
Solution:



Example #3.
Convert the Boolean expression into a digital logic circuit.
Boolean Expression: $\overline{(A+B)}$⊕(ABC)
Solution:



Example #4.
Convert the Boolean expression into a digital logic circuit.
Boolean Expression: (A⊕$\overline{B}$⊕C)($\overline{A}$B$\overline{C}$)



Practice Exercise 3.2.3

Convert the given Boolean Expression into a digital logic circuit.

1. C(AB⊕A$\overline{B}$)
2. ABC + AB$\overline{C}$ + $\overline{A}$B$\overline{C}$
3. ($\overline{A}$+B).(A+B+$\overline{C}$).($\overline{C}$)
4. (AB)⊕(AC)⊕(ABC)