Chapter 2
Boolean Algebra

Morris Mano, “Digital Logic and Computer Design.” (Pearson India, 2017)

In this topic, we'll learn

- Primitive logic elements used in digital systems.
- Mathematical methods for designing circuits.
- Cost effective design.
- Optimization techniques.

Binary Logic

- Deals with
  - variables that take on two discrete values
  - operations that assume logical meaning.
- Two values the variables take may be called by different names. (e.g., *true* and *false*, *yes* and *no*, etc.)
- Convenient to uses bits and assign the values of 1 and 0.
- Used to describe mathematically the manipulation and processing of binary information.
- Suited for the analysis and design of digital systems.

Definition of Binary Logic

- Consists of binary variables (A, B, C, y, z etc.,) and logical operations.
- Each variable can have two and only two distinct possible values: 0 and 1.
- Basic logical operations: AND, OR and NOT.

AND operation

- Represented by '.' or absence of an operator.

  X.Y=Z or XY=Z.

- Can be considered as two switches connected in series.

AND

X	Y	$Z = X \cdot Y$
0	0	0
0	1	0
1	0	0
1	1	1

OR operation

- Represented by ‘+’.
- Can be considered as two switches connected in parallel.

OR

X	Y	Z = X + Y
0	0	0
0	1	1
1	0	1
1	1	1

NOT Operation

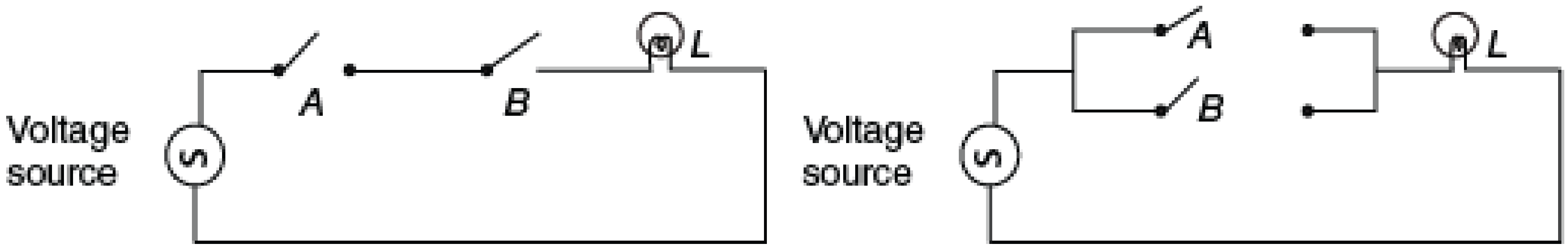
- Represented by a ' or by a bar.

NOT

X	$Z = \bar{X}$
0	1
1	0

Switching Circuits and Binary Signals

- Manual switches *A* and *B* represent two binary variables.
  - equal to 0 when the switch is open and 1 when the switch is closed.
- Lamp *L* represent a third binary variable
  - equal to 1 when the light is on and 0 when off.
- Electronic digital circuits: Switching circuits
  - Active element such as transistor acts as **closed** switch **when conducting** and **open** switch **other wise**.

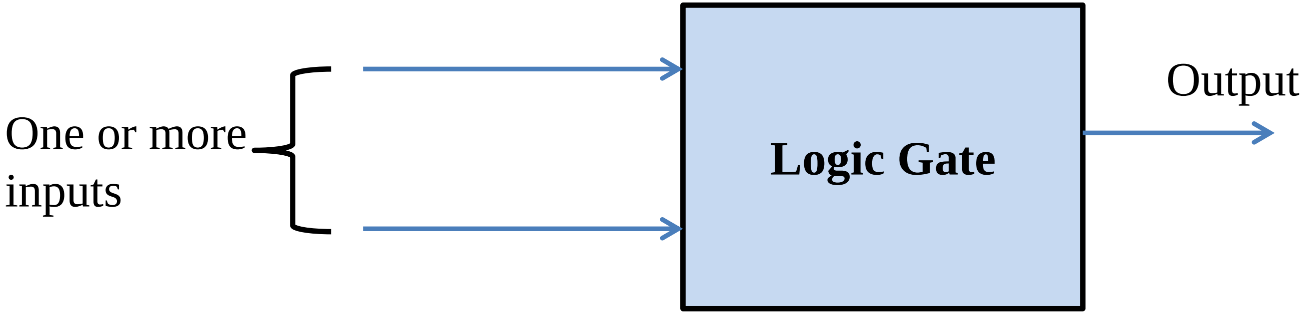


(a) Switches in series – logic AND

(b) Switches in parallel – Logic OR

Logic gates

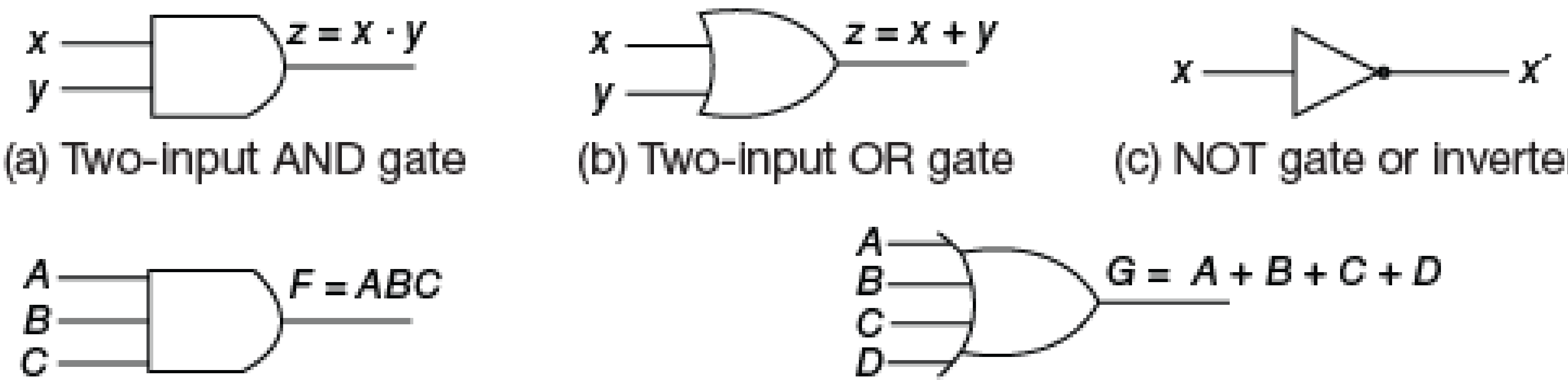
- Electronic digital circuits are also called Logic gates.
  - with the proper input, they establish logical manipulation paths.



Gates are implemented using transistors.
Signals are voltages or currents.

Symbols for digital logic circuits

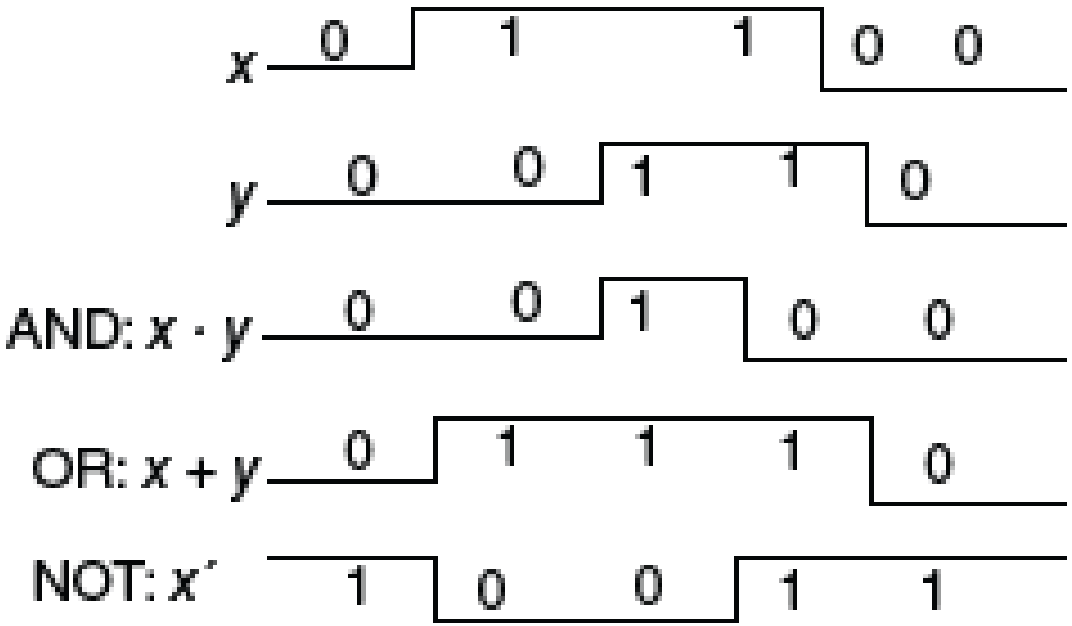
- These circuits are called *gates*.
- Four different names are used:
  - digital circuits, switching circuits, logic circuits, and gates
- We prefer to call them gates.
- Not gate is also called an inverter circuit.



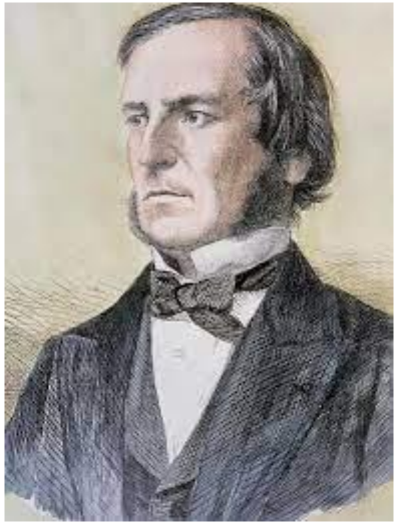
(a) Two-input AND gate (b) Two-input OR gate (c) NOT gate or inverter

(d) Three-input AND gate (e) Four-input OR gate

Input-output signals for gates



Boolean Algebra



- Mathematical system of binary logic is better known as Boolean, or switching, algebra.
  - conveniently used to describe the operation of complex networks of digital circuits.
- Designers of digital systems use Boolean algebra
  - to transform circuit diagrams to algebraic expressions and vice versa.
- Mathematical notion used is Boolean Algebra.
- George Boole, 1854.

History

- In 1854 George Boole introduced a systematic treatment of logic: Boolean Algebra
- In 1938 C. E. Shannon (2) introduced a two-valued Boolean algebra called *switching algebra*,
  - he demonstrated that the properties of bistable electrical switching circuits can be represented by this algebra.
- Postulates formulated by E. V. Huntington in 1904 are used for formal definition of Boolean Algebra.

Definition of Boolean Algebra

- An algebraic structure defined on a set of elements *B* together with two binary operators + and • provided the following (Huntington) postulates are satisfied.

1.

(a) Closure with respect to the operator +.

(b) Closure with respect to the operator •.

2.

(a) An identity element with respect to +, designated by 0:

$$x + 0 = 0 + x = x.$$

(b) An identity element with respect to •, designated by 1:

$$x \bullet 1 = 1 \bullet x = x.$$

Definition of Boolean Algebra (continued)

3.

(a) Commutative with respect to +: $x + y = y + x$.

(b) Commutative with respect to • : $x\,y = y \bullet x$.

4.

(a) • is distributive over + : $x \bullet (y + z) = (x \bullet y) + (x \bullet z)$.

(b) + is distributive over: $x + (y \bullet z) = (x + y) \bullet (x + z)$.

5. For every element $x \in B$, there exists an element $x' \in B$ (called the complement of x) such

that: (a) $x + x' = 1$ and (b) $x \bullet x' = 0$.

6. There exists at least two elements $x, y \in B$ such that $x \neq y$.

Boolean vs ordinary algebra

- Huntington postulates do not include the associative law. (this is true).
- The distributive law of + over •, i.e., $x + (y \cdot z) = (x + y) \cdot (x + z)$, is valid for Boolean algebra, but not for ordinary algebra.
- Boolean algebra does not have additive or multiplicative inverses; therefore, there are no subtraction or division operations.
- *complement* is not available in ordinary algebra.
- Ordinary algebra deals with the real numbers, which constitute an infinite set of elements. B, for our purpose, is defined as a set with only two elements, 0 and 1 in Boolean algebra.
- **Our interest here is with the application of Boolean algebra to gate-type circuits.**

Two-Valued Boolean Algebra

- Defined on a set of two elements, $B = \{0, 1\}$with
  - rules for the two binary operators + and • as shown below

x	y	$x \cdot y$
0	0	0
0	1	0
1	0	0
1	1	1

x	y	$x + y$
0	0	0
0	1	1
1	0	1
1	1	1

x	x'
0	1
1	0

  - Closure, Identity and commutative properties are easily verified from the table.

Continued…

- *distribute* law: $x \bullet (y + z) = (x \bullet y) + (x \bullet z)$

$x \quad y \quad z$	$y + z$	$x \bullet (y + z)$	$x \bullet y$	$x \bullet z$	$(x \bullet y) + (x \bullet z)$

Continued…

- *distribute* law $x \bullet (y + z) = (x \bullet y) + (x \bullet z)$

x	y	z	$y + z$	$x \bullet (y + z)$	$x \bullet y$	$x \bullet z$	$(x \bullet y) + (x \bullet z)$
0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	0
0	1	0	1	0	0	0	0
0	1	1	1	0	0	0	0
1	0	0	0	0	0	0	0
1	0	1	1	1	0	1	1
1	1	0	1	1	1	0	1
1	1	1	1	1	1	1	1

continued

- Property 5
  - $x + x' = 1$
  - $x \bullet x' = 0$
- Postulate 6 is satisfied because the two-valued Boolean algebra has two distinct elements
  - 1 and 0 with $1 \neq 0$.
- Two binary operators with operation rules equivalent to the AND and OR operations.
- Complement operator equivalent to the NOT operation.
- Equivalent to the binary logic presented earlier.

Postulates and theorems of Boolean algebra

Postulate 2	(a) $x + 0 = x$	(b) $x \cdot 1 = x$
Postulate 5	(a) $x + x' = 1$	(b) $x \cdot x' = 0$
Theorem 1	(a) $x + x = x$	(b) $x \cdot x = x$
Theorem 2	(a) $x + 1 = 1$	(b) $x \cdot 0 = 0$
Theorem 3, involution	$(x')' = x$	
Postulate 3, commutative	(a) $x + y = y + x$	(b) $xy = yx$
Theorem 4, associative	(a) $x + (y + z) = (x + y) + z$	(b) $x(yz) = (xy)z$
Postulate 4, distributive	(a) $x(y + z) = xy + xz$	(b) $x + yz = (x + y)(x + z)$
Theorem 5, DeMorgan	(a) $(x + y)' = x'y'$	(b) $(xy)' = x' + y'$
Theorem 6, absorption	(a) $x + xy = x$	(b) $x(x + y) = x$

- *Duality principle: Arranged as 2 columns*

Operator Precedence

- (1) parentheses, (2) NOT, (3) AND, and (4) OR.
- Eg: $(A + B)' = A' \cdot B'$.

Using basic Boolean theorem prove:

1. $(x + y)(x + z) = x + yz$
2. $xy + xz + yz' = xz + yz'$

Sol. 1: $(x + y)(x + z)$

$xx + xz + xy + yz$

$x + xz + xy + yz$

$x(1 + z) + xy + yz$

$x + xy + yz$

$x(1+y) + yz$

$x + yz$

Sol. 2: $xy + xz + yz'$

$xy(z+z') + xz + yz'$

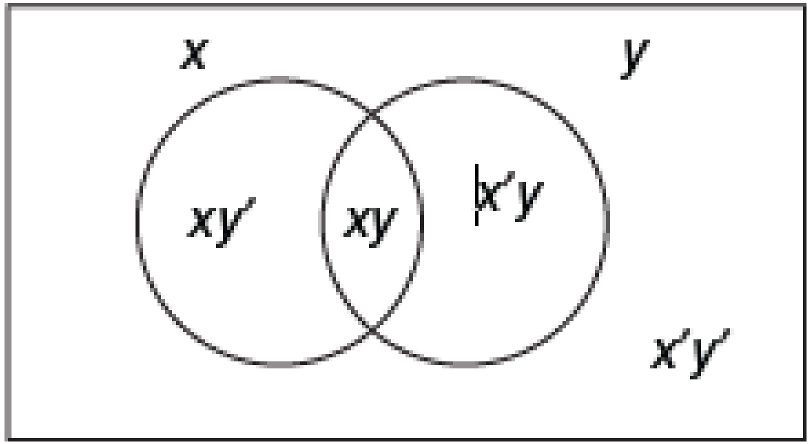
$xyz + xyz' + xz + yz'$

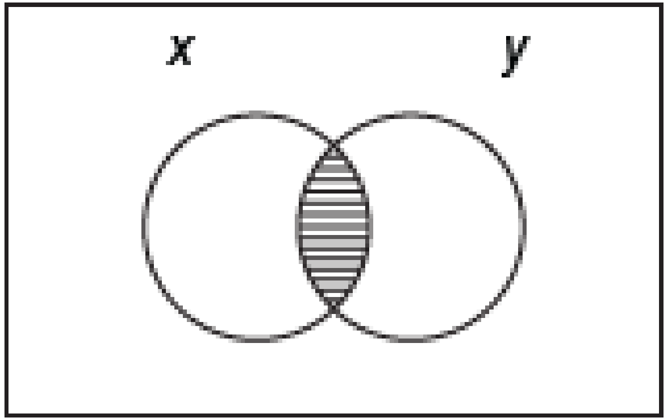
$xz(1 + y) + yz'(1 + x)$

$xz + yz'$

Venn Diagram

- Aids in the visualization of Boolean relations between variables.





X= X+ XY

Boolean Functions

- A Boolean function is an expression formed with
  - binary variables
  - the two binary operators OR and AND,
  - the unary operator NOT,
  - parentheses, and equal sign
- For a given value of the variables, the function can be either 0 or 1.

e.g.: $F_1 = xyz'$

Boolean function represented as an **algebraic expression.**

Truth Table for Boolean function $F_1 = xyz'$

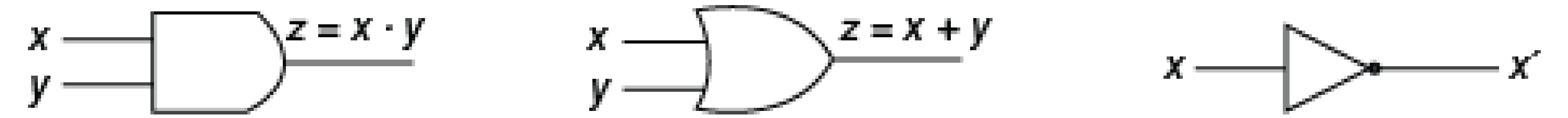
x	y	z	F_1
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

Truth Table for Boolean function $F_1 = xyz'$, $F_2 = x + y' z$, $F_3 = x' y' z + x' yz + xy'$, and $F_4 = \mathrm{xy}' + x' z$

-

x	y	z	F_1
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	1
1	1	1	0

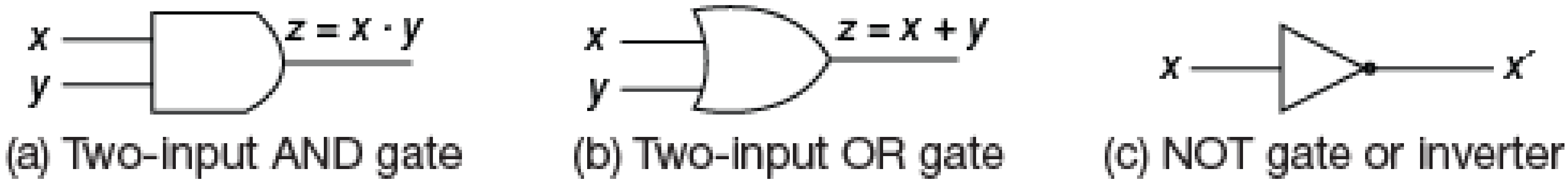
Implementation of Boolean function with gates



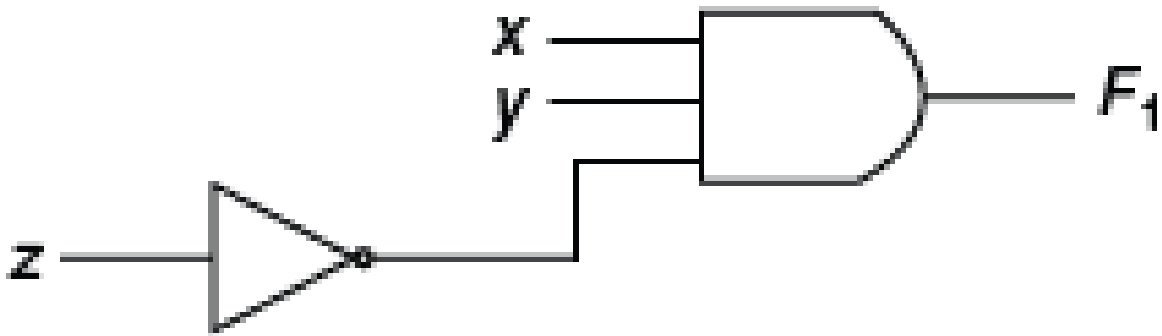
(a) Two-input AND gate (b) Two-input OR gate (c) NOT gate or inverter

- $F_1 = xyz'$

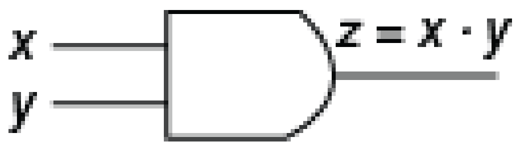
Implementation of Boolean function with gates



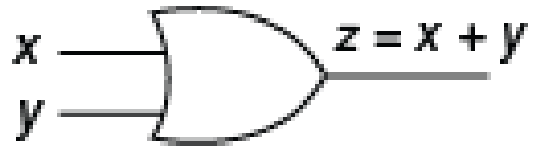
- $F_1 = xyz'$



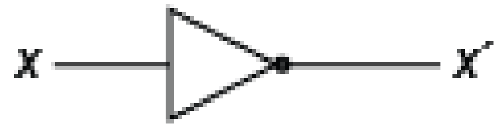
Implementation of Boolean function with gates



(a) Two-input AND gate



(b) Two-input OR gate



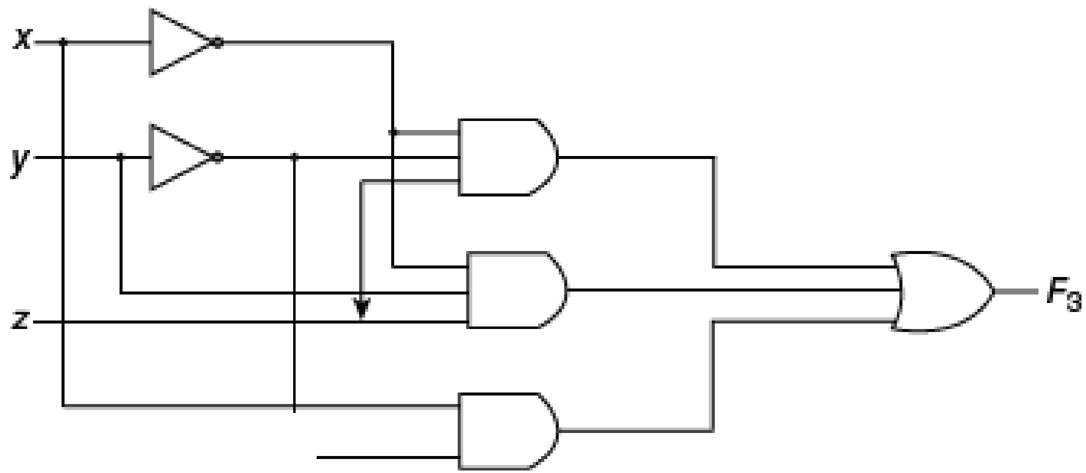
(c) NOT gate or inverter

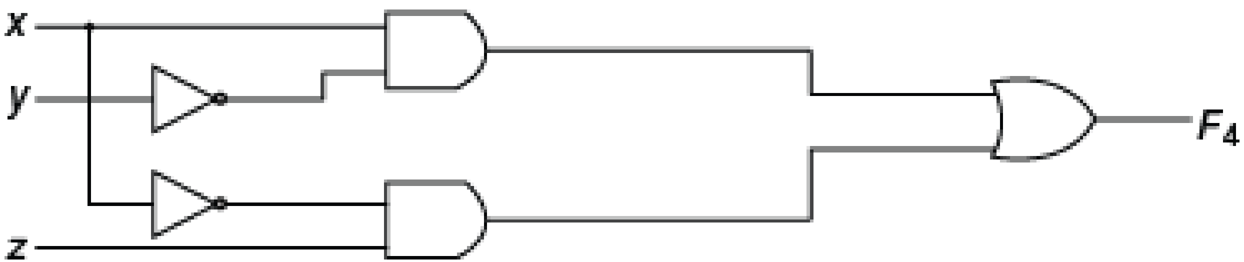
- $F_2 = x + y' z$
- $F_3 = x' y' z + x' yz + xy'$
- $F_4 = \mathrm{xy}' + x' z$

Implementation of Boolean function with gates

(a) Two-input AND gate (b) Two-input OR gate (c) NOT gate or inverter

- $F_2 = x + y' z$
- $F_3 = x' y' z + x' yz + xy'$
- $F_4 = \mathrm{xy}' + x' z$





How to manipulate Boolean functions to obtain equal and simpler expressions?

Best form of Boolean Function

- Depends on the particular application.
- Here we consider criterion of equipment minimization.

Algebraic Manipulation

- *Literal* is a primed or unprimed variable.
- While implementing
  - each literal in the function designates an input to a gate.
  - each term is implemented with a gate.
- Minimization of the number of literals and the number of terms results in a circuit with less equipment.
- Here we will emphasize on **literal minimization**.
- Algebraic manipulation
  - Unfortunately, there are **no specific rules** to follow that will guarantee the final answer.
  - employ the postulates, the basic theorems, and other manipulation methods

Simplify the following equations

- $x + x' y$
- $x (x' + y)$
- $x' y' z + x' yz + xy'$
- $xy + x' z + yz$
- $(x + y) (x' + z) (y + z)$

Simplify the following equations

- $x + x' y = x + y$
- $x (x' + y) = x y$
- $x' y' z + x' yz + xy' = x' z + x y'$
- $xy + x' z + yz = xy + x' z$
- $(x + y) (x' + z) (y + z) = (x + y) (x' + z)$

Complement of a Function

- Complement of a function may be derived algebraically through De Morgan’s theorem.
- De Morgan’s theorems can be extended to three or more variables.

Complement of a Function

- Complement of a function may be derived algebraically through De Morgan's theorem.
- De Morgan's theorems can be extended to three or more variables.

$$(A + B + C + D + \ldots + F)' = A'B'C'D' \ldots F'$$

$$(ABCD \ldots F)' = A' + B' + C' + D' + \ldots + F'$$

- Complement of a function is obtained by interchanging AND and OR operators and complementing each literal.

Find the complement of functions using De Morgan's theorem:

- $F_1 = x'yz' + x'y'z$
- $F_2 = x(y'z' + yz)$

Find the complement of functions using duals:

1. ***Find the dual of the function.***
2. ***Complement the literal***

- $F_1 = x' y z' + x' y' z$

1. $(x' + y + z').(x' + y' + z)$
2. $\boldsymbol{(x + y' + z).(x + y + z')}$

- $F_2 = x(y' z' + y z)$

1.
2.

Canonical Forms

- A binary variable may appear in
  - Normal form (x)
  - Complimentary form (x')
- Hence for two variables x and y there are four possibilities for AND operation
  - $x'y'$, $x'y$, xy', and xy
  - Each of these 4 terms is called a **min-term** or **standard product**.
  - n variables : 2^n minterms (binary numbers from 0 to 2^n - 1).
    - Each minterm: AND term of n variables which is primed if the corresponding bit is 0. (Symbol m_j, where j is the decimal equivalent of binary number)

Minterms

Table 2-3 Minterms and maxterms for three binary variables

			Minterms		Maxterms	
x	y	z	Term	Designation	Term	Designation
0	0	0	$x'y'z'$	m_0	$x + y + z$	M_0
0	0	1	$x'y'z$	m_1	$x + y + z'$	M_1
0	1	0	$x'yz'$	m_2	$x + y' + z$	M_2
0	1	1	$x'yz$	m_3	$x + y' + z'$	M_3
1	0	0	$xy'z'$	m_4	$x' + y + z$	M_4
1	0	1	$xy'z$	m_5	$x' + y + z'$	M_5
1	1	0	xyz'	m_6	$x' + y' + z$	M_6
1	1	1	xyz	m_7	$x' + y' + z'$	M_7

Canonical Forms (continued..)

- Similarly for an OR
  - n variables from 2^n terms called **maxterms** or **standard sums**.
  - each variable being unprimed if the corresponding bit is a 0 and primed if a 1. (Symbol: M_j)

Minterms and maxterms for three binary variables

			Minterms		Maxterms	
x	y	z	Term	Designation	Term	Designation
0	0	0	$x'y'z'$	m_0	$x+y+z$	M_0
0	0	1	$x'y'z$	m_1	$x+y+z'$	M_1
0	1	0	$x'yz'$	m_2	$x+y'+z$	M_2
0	1	1	$x'yz$	m_3	$x+y'+z'$	M_3
1	0	0	$xy'z'$	m_4	$x'+y+z$	M_4
1	0	1	$xy'z$	m_5	$x'+y+z'$	M_5
1	1	0	xyz'	m_6	$x'+y'+z$	M_6
1	1	1	xyz	m_7	$x'+y'+z'$	M_7

Minterms and Maxterms

Table 2-3 Minterms and maxterms for three binary variables

			Minterms		Maxterms	
x	y	z	Term	Designation	Term	Designation
0	0	0	$x'y'z'$	m_0	$x + y + z$	M_0
0	0	1	$x'y'z$	m_1	$x + y + z'$	M_1
0	1	0	$x'yz'$	m_2	$x + y' + z$	M_2
0	1	1	$x'yz$	m_3	$x + y' + z'$	M_3
1	0	0	$xy'z'$	m_4	$x' + y + z$	M_4
1	0	1	$xy'z$	m_5	$x' + y + z'$	M_5
1	1	0	xyz'	m_6	$x' + y' + z$	M_6
1	1	1	xyz	m_7	$x' + y' + z'$	M_7

Boolean function as sum of minterms

x	y	z	Function f_1	Function f_2
0	0	0	0	0
0	0	1	1	0
0	1	0	0	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	I	0	0	1
1	1	1	1	1

			Minterms	
x	y	z	Term	Designation
0	0	0	$x'y'z'$	m_0
0	0	1	$x'y'z$	m_1
0	1	0	$x'yz'$	m_2
0	1	1	$x'yz$	m_3
1	0	0	$xy'z'$	m_4
1	0	1	$xy'z$	m_5
1	1	0	xyz'	m_6
1	1	1	xyz	m_7

Maxterms directly from the truth table

x	y	z	Function f_1	Function f_2
0	0	0	0	0
0	0	1	1	0
0	1	0	0	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	I	0	0	1
1	1	1	1	1

- Any Boolean function can be expressed as a product of maxterms or sum of minterms.
- Boolean functions expressed as a sum of minterms or product of maxterms are said to be in *canonical form.*

Sum of Minterms

- Minterms whose sum defines the Boolean function are those that give the l’s of the function in a truth table.
- Conversion to sum of minterms:
  - If a term misses one or more variables, it is ANDed with an expression such as $x + x'$, where x is one of the missing variables.
- E.g.: Express the Boolean function $F = A + B'C$ in a sum of minterms.

Sum of Minterms

- Minterms whose sum defines the Boolean function are those that give the l's of the function in a truth table.
- Conversion to sum of minterms:
  - If a term misses one or more variables, it is ANDed with an expression such as $x + x'$, where x is one of the missing variables.
- E.g.: Express the Boolean function $F = A + B'C$ in a sum of minterms.
- $F=m_1+m_4+m_5+m_6+m_7 = \Sigma\,(1,4, 5,6,7)$

Product of Maxterms

1. Use distributive law to bring to a form of OR terms.
2. Missing variable x in each OR term is ORed with xx'.

E.g.: $F = xy + x'z$

Product of Maxterms

1. Use distributive law to bring to a form of OR terms.
2. Missing variable x in each OR term is ORed with xx'.

E.g.: $F = xy + x'z = M_0 M_2 M_4 M_5 = \Pi(0,2,4,5)$

Table 2-3 Minterms and maxterms for three binary variables

			Minterms		Maxterms	
x	y	z	Term	Designation	Term	Designation
0	0	0	$x'y'z'$	m_0	$x + y + z$	M_0
0	0	1	$x'y'z$	m_1	$x + y + z'$	M_1
0	1	0	$x'yz'$	m_2	$x + y' + z$	M_2
0	1	1	$x'yz$	m_3	$x + y' + z'$	M_3
1	0	0	$xy'z'$	m_4	$x' + y + z$	M_4
1	0	1	$xy'z$	m_5	$x' + y + z'$	M_5
1	1	0	xyz'	m_6	$x' + y' + z$	M_6
1	1	1	xyz	m_7	$x' + y' + z'$	M_7

Conversion between canonical forms

- The complement of a function expressed as the sum of minterms equals the sum of minterms missing from the original function.
- $F(A,B,\mathrm{C}) = \Sigma(1,4,5,6,7)$
- $F'(A,B,\mathrm{C}) = \Sigma()$

Conversion between canonical forms

- The complement of a function expressed as the sum of minterms equals the sum of minterms missing from the original function.
- $F(A,B,\text{C}) = \Sigma(1,4,5,6,7)$
- $F'(A,B,\text{C}) = \Sigma(0,2,3)$
- Complement of F ′ by De Morgan's theorem gives F in another form

Table 2-3 Minterms and maxterms for three binary variables

			Minterms		Maxterms	
x	y	z	Term	Designation	Term	Designation
0	0	0	$x'y'z'$	m_0	$x + y + z$	M_0
0	0	1	$x'y'z$	m_1	$x + y + z'$	M_1
0	1	0	$x'yz'$	m_2	$x + y' + z$	M_2
0	1	1	$x'yz$	m_3	$x + y' + z'$	M_3
1	0	0	$xy'z'$	m_4	$x' + y + z$	M_4
1	0	1	$xy'z$	m_5	$x' + y + z'$	M_5
1	1	0	xyz'	m_6	$x' + y' + z$	M_6
1	1	1	xyz	m_7	$x' + y' + z'$	M_7

Conversion between canonical forms

- The complement of a function expressed as the sum of minterms equals the sum of minterms missing from the original function.
- $F(A,B, \mathrm{C}) = \Sigma(1,4,5,6,7)$
- $F\,'(A,B, \mathrm{C}) = \Sigma(0,2,3)$
- Complement of F ′ by De Morgan's theorem gives F in another form
- $F=(m_0 + m_2 + m_3)' = m'_0 .m'_2 .m'_3 = M_0 M_2 M_3 = \Pi\ (0, 2, 3)$
- $m_j' = M_j$

General conversion procedure

- To convert from one canonical form to another
  - interchange the symbols Σ and Π
  - list those numbers missing from the original form.
- *Convert* $F(x, y, z) = \Pi\ (0, 2, 4, 5)$ to Sum of minterms form.
- $\sum (1,3,6,7)$

Standard Forms

- Two canonical forms of Boolean algebra are basic forms that one obtains from reading a function from the truth table.
  - seldom the ones with the least number of literals (should include by definition all variables)
- Standard form
  - function may contain one, two or any number of literal.
  - 2 types: the sum of products and product of sums.
  - $F_1 = y' + xy + x'yz'$: SOP
  - $F_2 = x(y' + z)(x' + y + z' + w)$: POS

Conversion from non-standard form to standard form

- $F3 = (AB + CD)(A'B' + C'D')$

  $= \mathrm{ABA'B'} + \mathrm{ABC'D'} + \mathrm{A'B'CD} + \mathrm{CDC'D'}$

  $= \mathrm{ABC'D'} + \mathrm{A'B'CD}$

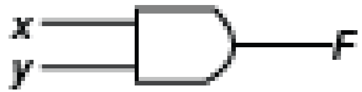
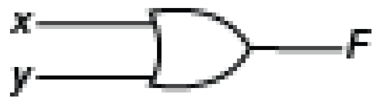
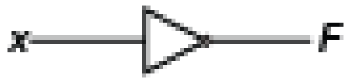
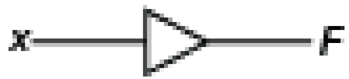
Other Logic Operations

x	y	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
Operator Symbol				/		/		$\oplus$	+	$\downarrow$	$\odot$		$\subset$		$\supset$	$\uparrow$	

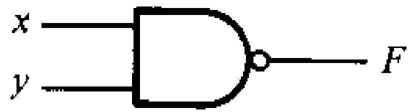
Boolean expressions for the 16 functions of two variables

$F_0 = 0$		Null	Binary constant 0
$F_1 = xy$	$x \cdot y$	AND	x and y
$F_2 = xy'$	x/y	Inhibition	x but not y
$F_3 = x$		Transfer	x
$F_4 = x'y$	y/x	Inhibition	y but not x
$F_5 = y$		Transfer	y
$F_6 = xy' + x'y$	$x \oplus y$	Exclusive-OR	x or y but not both
$F_7 = x + y$	$x + y$	OR	x or y
$F_8 = (x + y)'$	$x \downarrow y$	NOR	Not-OR
$F_9 = xy + x'y'$	$x \odot y$	Equivalence*	x equals y
$F_{10} = y'$	y'	Complement	Not y
$F_{11} = x + y'$	$x \subset y$	Implication	If y then x
$F_{12} = x'$	x′	Complement	Not x
$F_{13} = x' + y$	$x \supset y$	Implication	If x then y
$F_{14} = (xy)'$	$x \uparrow y$	NAND	Not-AND
$F_{15} = 1$		Identity	Binary constant 1

Digital logic gates

Name	Graphic symbol	Algebraic function	Truth table
AND		$F = xy$	x y | F 0 0 | 0 0 1 | 0 1 0 | 0 1 1 | 1
OR		$F = x + y$	x y | F 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 1
Inverter		$F = x'$	x | F 0 | 1 1 | 0
Buffer		$F = x$	x | F 0 | 0 1 | 1

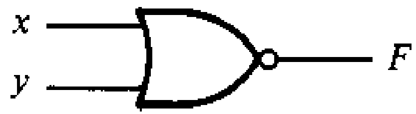
NAND



$$F = (xy)'$$

x	y	F
0	0	1
0	1	1
1	0	1
1	1	0

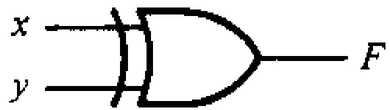
NOR



$$F = (x + y)'$$

x	y	F
0	0	1
0	1	0
1	0	0
1	1	0

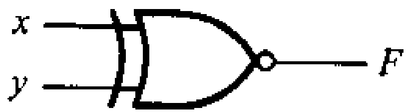
Exclusive-OR (XOR)



$$F = xy' + x'y = x \oplus y$$

x	y	F
0	0	0
0	1	1
1	0	1
1	1	0

Exclusive-NOR or equivalence



$$F = xy + x'y' = x \odot y$$

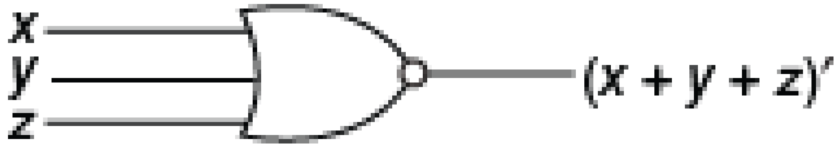
x	y	F
0	0	1
0	1	0
1	0	0
1	1	1

Extension to Multiple Inputs

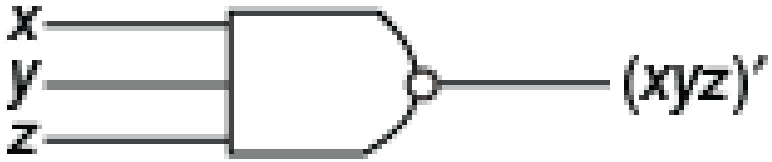
- A gate can be extended to have multiple inputs if binary operation it represents is
  - commutative
  - and associative
  - AND and OR are commutative and associative.
  - gate inputs can be interchanged.
- What about NAND and NOR?

Extension to Multiple Inputs

- A gate can be extended to have multiple inputs if binary operation it represents is
  - commutative
  - and associative
  - AND and OR are commutative and associative.
  - gate inputs can be interchanged.



- What about NAND and NOR?
  - Commutative but not associative.

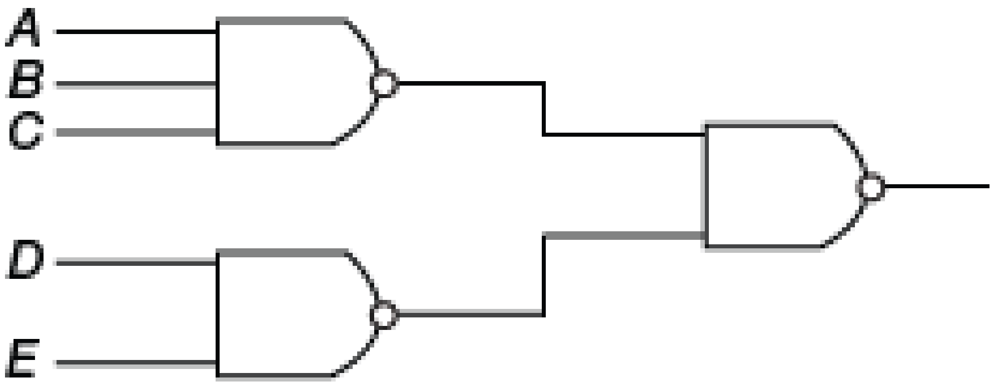


- To overcome this we define NAND and NOR as

$$x \downarrow y \downarrow z = (x + y + z)'$$

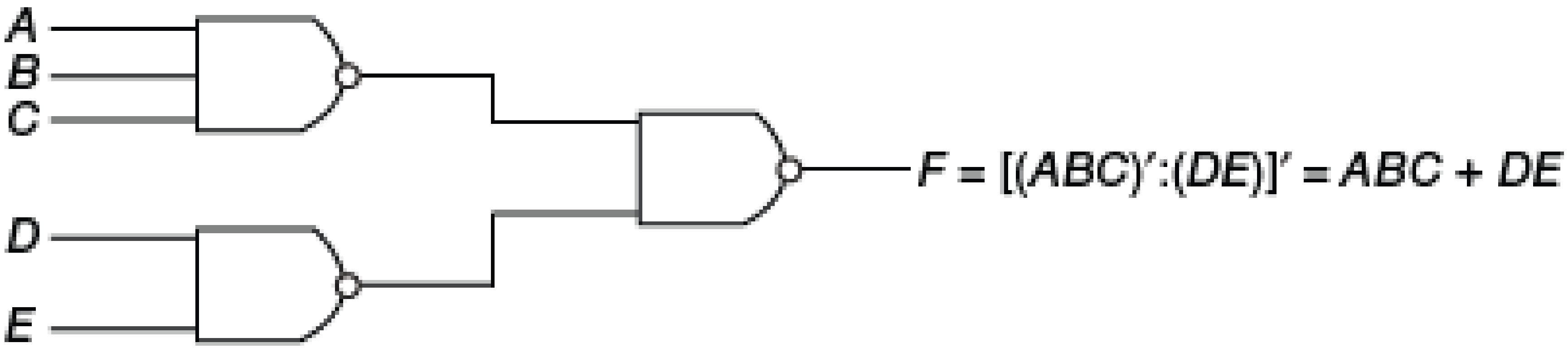
$$x \uparrow y \uparrow z = (xyz)'$$

Cascaded NAND



- Write the output

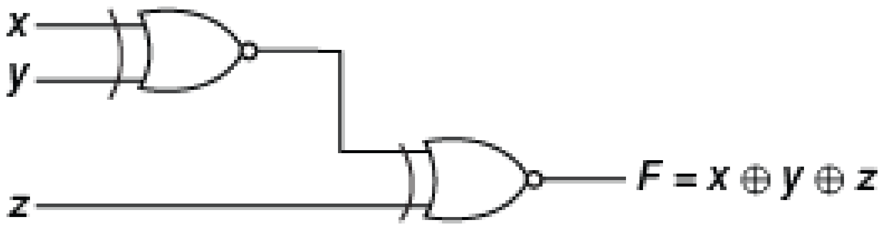
Cascaded NAND

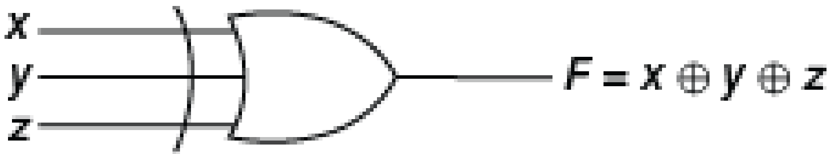


- Correct parentheses should be used to signify the proper sequence.
- Sum of products can be implemented with NAND gates

Cascaded XOR and Equivalence

- Both are commutative and associative.
  - can be extended to more than two inputs.





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

Construct AND/OR Gate using NOR/NAND Gate