Unit 2

Boolean Algebra and Logic Gates

Content

- Basic definition of Boolean Algebra
- Basic Theory of Boolean Algebra
- Boolean Function
- Logic operations
- Logic Gates
- IC Digital Logic Families

Binary logic

- Consists of binary variables and logical operations.
- The variables are designated by letters of the alphabet such as *A*, *B*, C, *x*, *y*, Z, etc., with each variable having two and only two distinct possible values: 1 and 0.

- There are three basic logical operations:

  AND, OR, and NOT.

- **AND:** represented by a dot or by the absence of an operator.

  For example, $x.y = z$ or $xy = z$ is read *"x* **AND** *y* is equal to *z."*

It means that z = 1 if and only if $x = 1$ *and* $y = 1$; otherwise z = 0.

- **OR:** represented by a plus sign.

  For example, $x + y = z$ is read *"x* **OR** *y* is equal to *z,"*

  It means that z = 0 if $x = 0$ *or* if $y = 0$ otherwise z = 1.

- **NOT:** represented by a prime (sometimes by a bar).

  For example, $x' = z$ is read "not *x* is equal to *z,"*

  It means that z is what *x* is not.

  In other words, if $x = 1$, then z = 0; but if $x = 0$, then z = 1.

These above definitions may be listed in a compact form using truth tables. A **truth table** is a table of all possible combinations of the variables showing the relation between the values that the variables may take and the result of the operation.

Truth Table of the Logical Operations

OR

x	y	z = x+y
0	0	0
0	1	1
1	0	1
1	1	1

AND

x	y	z=x.y
0	0	0
0	1	0
1	0	0
1	1	1

NOT

x	z=x'
0	1
1	0

Note : Binary logic should not be confused with binary arithmetic (However we use same symbols here). You should realize that an arithmetic variable designates a number that may consist of many digits. A logic variable is always either 1 or 0. For example, in binary arithmetic, 1 + 1 = 10 (read: "one plus one is equal to 2"), whereas in binary logic, we have 1 + 1 = 1 (read: "one OR one is equal to one").

Boolean Algebra

- mathematical system of binary logic.
- used to describe the operation of complex networks of digital circuits.
- used to transform circuit diagrams to algebraic expressions and vice versa.

Postulates

Boolean algebra is an algebraic structure defined on a set of elements B (Boolean system) together with two binary operators + (OR) and • (AND) and unary operator ' (NOT), provided the following postulates are satisfied:

P1 → Closure: Boolean algebra is closed under the AND, OR, and NOT operations.

P2 → Commutative: The • and + operators are commutative

i.e. $x + y = y + x$ and $x \cdot y = y \cdot x$, for all $x, y \in B$.

P3 → Distribution: • and + are distributive with respect to one another

i.e. $x \cdot (y + z) = (x \cdot y) + (x \cdot z)$.

$x + (y \cdot z) = (x + y) \cdot (x + z)$, *for all* $x, y, z \in B$.

P4 → **Identity**: The identity element with respect to • is 1 and + is 0 i.e. x + 0 = 0 + x = x and x • 1=1• x = x. There is no identity element with respect to logical NOT.

P5 → **Inverse**: For every value x there exists a value x' such that x • x' = 0 and x + x' = 1. This value is the logical complement (or NOT) of x.

- **Theorems of Boolean Algebra**

Duality Theorem

It states that

"***Every algebraic expression deducible from the postulates of Boolean algebra remains valid if the operators and identity elements are interchanged***".

Starting from a boolean relation, we can derive another boolean relation by:

1. Changing each OR(+) sign to AND(•) sign or Vice-versa
2. Complementing any 0 or 1 appearing in the expression.

Eg:

a) $x + 0 = x \rightarrow x \cdot 1 = x$

b) $x(y+z) = xy+xz \rightarrow x+yz = (x+y)(x+z)$

Here: Dual of $x(y+z) = x+yz$ and Dual of $xy+xz = (x+y)(x+z)$

c) $AB + A'C + BC = AB + A'C \rightarrow (A+B)(A'+C)(B+C) = (A+B)(A'+C)$

Here: Dual of $AB + A'C + BC = (A+B)(A'+C)(B+C)$ and

Dual of $AB + A'C = (A+B)(A'+C)$

Basic theorems of Boolean Algebra

The postulates are basic axioms of the algebraic structure and need no proof. The theorems must be proven from the postulates. six theorems of Boolean algebra are given below:

Theorem1:	Idempotence	(a) x + x = x	(b) x.x = x
Theorem2:	Existence: 0 & 1	(a) x + 1 = 1	(b) x.0 = 0
Theorem3:	Involution	(x')' = x	
Theorem4:	Associative	(a) x+(y+z) = (x+y)+z	(b) x(yz) = (xy)z
Theorem5:	De-morgan	(a) (x + y)' = x'y'	(b) (xy)' = x' + y'
Theorem6:	Absorption	(a) x + xy = x	(b) x(x + y) = x

Proofs:

(a) The proofs of the theorems with one variable are presented below:

THEOREM 1(a): $x + x = x$

$=x + x = (x + x) \cdot 1$	(P4: Identity element)
$= (x + x)(x + x')$	(P5: Existence of inverse)
$= x + xx'$	(P3: Distribution)
$=x + 0$	(P5: Existence of inverse)
$=x$	(P4: Identity element)

THEOREM 1(b): $x \cdot x = x$

$=x.x = xx + 0$	(P4: Identity element)
$=xx + xx'$	(P5: Existence of inverse)
$=x(x + x')$	(P3: Distribution)
$= x.1$	(P5: Existence of inverse)
$=x$	(P4: Identity element)

THEOREM 2(a): $x + 1 = 1$

$= x + 1 = 1 \cdot (x + 1)$	(P4: Identity element)
$= (x + x')(x + 1)$	(P5: Existence of inverse)
$= x + x' \cdot 1$	(P3: Distribution)
$= x + x'$	(P4: Identity element)
$= 1$	(P5: Existence of inverse)

THEOREM 2(b): $x.0 = 0$

Prove **De-morgan's theorem**:

THEOREM 5 (a): **(x + y)' = x'y'**

From postulate P5 (Existence of inverse), for every x in a Boolean algebra there is a unique x' such that x + x' = 1 and x • x' = 0
So it is sufficient to show that x'y' is the complement of x + y. We'll do this by showing that (x + y) + (x'y') = 1 and (x + y) • (x'y') = 0.

=(x + y) + (x'y')	
= [(x + y) + x'] [(x + y) + y']	[OR distributes over AND (P3)]
= [(y + x) + x'] [(x + y) + y']	[OR is commutative (P2)]
= [y + (x + x')] [x + (y + y')]	[OR is associative (Theorem 3(a)), used twice]
= (y + 1)(x + 1)	[Complement, x + x' = 1 (P5), twice]
= 1 • 1	[x + 1 = 1, (Theorem 2), twice]
= 1	[Idempotent, x • x = x (Theorem 1)]

Also,

=(x + y)(x'y')	
= (x'y') (x + y)	[AND is commutative (P2)]
= [(x'y') x] + [(x'y') y]	[AND distributes over OR (P3)]
= [(y'x')x] + [(x'y')y]	[AND is commutative (P2)]
= [y'(x'x)] + [x'(y'y)]	[AND is associative (Theorem 3(b)), twice]
= [y'(xx')] + [x'(yy')]	[AND is commutative, twice]
= [y' • 0] + [x' • 0]	[Complement, x • x' = 0, twice]
= 0 + 0	[x • 0 = 0, twice]
= 0	[Idempotent, x + x = x]

THEOREM 5(a): **(xy)' = x' + y'** can be proved similarly as in Theorem 5(a).

Operator Precedence

The operator precedence for evaluating Boolean expressions is

1. Parentheses ()
2. NOT '
3. AND .
4. OR +

In other words, the expression inside the parentheses must be evaluated before all other operations. The next operation that holds precedence is the complement, then follows the AND, and finally the OR.

Example:

(a+b.c).d' ➔Here, we first evaluate ‘b.c’ and OR it with ‘a’ followed by ANDing with complement of ‘d’.

Summary: Boolean Laws

Boolean identities

Name	AND version	OR version
Identity	$x \cdot 1 = x$	$x + 0 = x$
Complement	$x \cdot \overline{x} = 0$	$x + \overline{x} = 1$
Commutative	$x \cdot y = y \cdot x$	$x + y = y + x$
Distribution	$x \cdot (y+z) = xy+xz$	$x + (y \cdot z) = (x+y)\,(x+z)$
Idempotent	$x \cdot x = x$	$x + x = x$
Null	$x \cdot 0 = 0$	$x + 1 = 1$

Name	AND version	OR version
Involution	$\overline{\overline{x}} = x$	---
Absorption	$x \cdot (x+y) = x$	$x + (x \cdot y) = x$
Associative	$x \cdot (y \cdot z) = (x \cdot y) \cdot z$	$x + (y + z) = (x + y) + z$
de Morgan	$\overline{x \cdot y} = \overline{x} + \overline{y}$	$\overline{x + y} = \overline{x} \cdot \overline{y}$

Boolean Functions

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

- **Boolean function represented as an algebraic expression:**

  Consider Boolean function F1 = xyz'.

  Function F1 is equal to 1 if x=1, y=1 and z=0; otherwise F1 =0.

Other examples are: F2 = x + y'z,

F3 = x'y'z + x'yz + xy',

F4 = xy' + x'z etc.

- **Boolean function represented in a truth table:**

The number of rows in the truth table is 2^n, where n is the number of binary variables in the function, The 1's and 0's combinations for each row is easily obtained from the binary numbers by counting from 0 to $2^n - 1$.

x	y	z	F_1	F_2	F_3	F_4
0	0	0	0	0	0	0
0	0	1	0	1	1	1
0	1	0	0	0	0	0
0	1	1	0	0	1	1
1	0	0	0	1	1	1
1	0	1	0	1	1	1
1	1	0	1	1	0	0
1	1	1	0	1	0	0

Note:

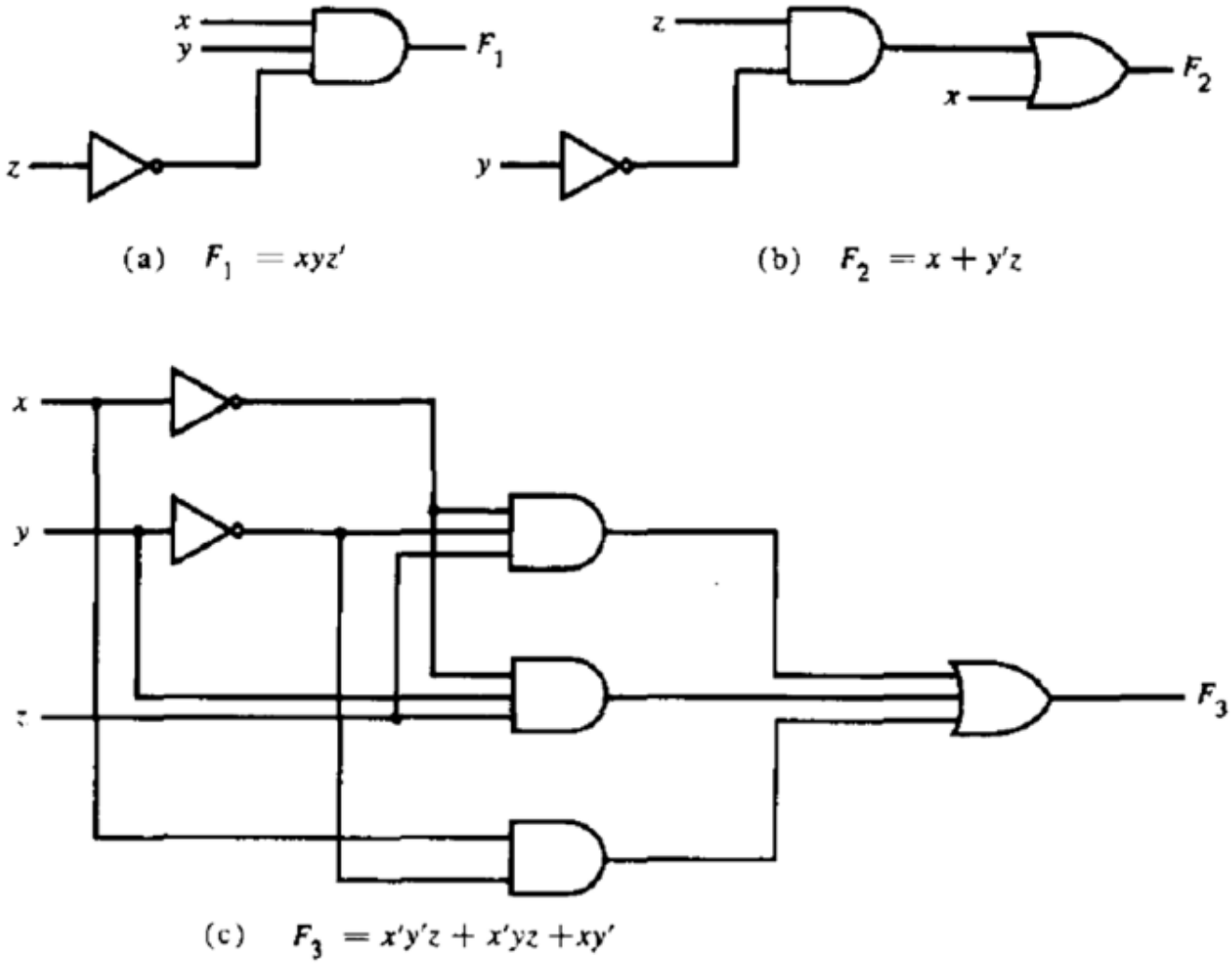
Is it possible to find two algebraic expressions that specify the same function?

Answer: **Yes**.

Being straightforward, the manipulation of Boolean algebra is applied mostly to the problem of finding simpler expressions for the same function.

Example: Did you or did you not notice that Functions F3 and F4 are same although they have different combinations of binary variables with in them.

- A Boolean function may be transformed from an algebraic expression into a logic diagram composed of AND, OR, and NOT gates.



(a) $F_1 = xyz'$

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

(c) $F_3 = x'y'z + x'yz + xy'$

Algebraic manipulation and simplification of Boolean function

Simplify the following Boolean functions to a minimum number of literals(A ***literal*** **is a primed or unprimed (i.e. complemented or un-complemented) variable.**).

1. $x + x'y = (x + x')(x + y) = 1.(x + y) = x + y$

2. $x(x' + y) = xx' + xy = 0 + xy = xy$

3. From before example of F_3 and F_4

$F_3 = x'y'z + x'yz + xy' = x'z(y' + y) + xy' = x'z + xy' = F_4$

Complement of a function

The complement of a function F is F' and is obtained from an interchange of 0's for 1's and 1's for 0's in the value of F. The complement of a function may be derived algebraically through DeMorgan's theorem. De-Morgan's theorems can be extended to three or more variables. The three-variable form of the first De-Morgan's theorem is derived below.

(A + B + C)'	= (A + X)'	let B + C = X
	= A'X'	(DeMorgan)
	= A' (B + C)'	substitute B + C = X
	= A'(B'C')	(DeMorgan)
	= A'B'C'	(associative)

DeMorgan's theorems for any number of variables resemble in form the two variable case and can be derived by successive substitutions similar to the method used in the above derivation. These theorems can be generalized as follows:

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

The generalized form of De Morgan's theorem states that the complement of a function is obtained by interchanging AND and OR operators and complementing each literal.

Two ways of getting complement of a Boolean function:

1. ***Applying DeMorgan's theorem:***

Question: Find the complement of the functions

F1 = x'yz' + x'y'z and F2 = x(y'z' + yz).

By applying DeMorgan's theorem as many times as necessary, the complements are obtained as follows:

F1' = (x'yz' + x'y'z)' = (x'yz')'(x'y'z)' = (x + y' + z)(x + y + z')

F2' = [x(y'z' + yz)]' = x'+(y'z' + yz)' = x'+ (y'z')'·(yz)' = x'+(y +z)(y'+z')

2. First finding dual of the algebraic expression and complementing each literal

Question: Find the complement of the functions F1 and F2 of example above by taking their duals and complementing each literal.

- F1 = x'yz' + x'y'z.

  The dual of F1 is (x' + y + z')(x' + y' + z).

  Complement each literal: (x + y' + z)(x + y + z ') = F1' .

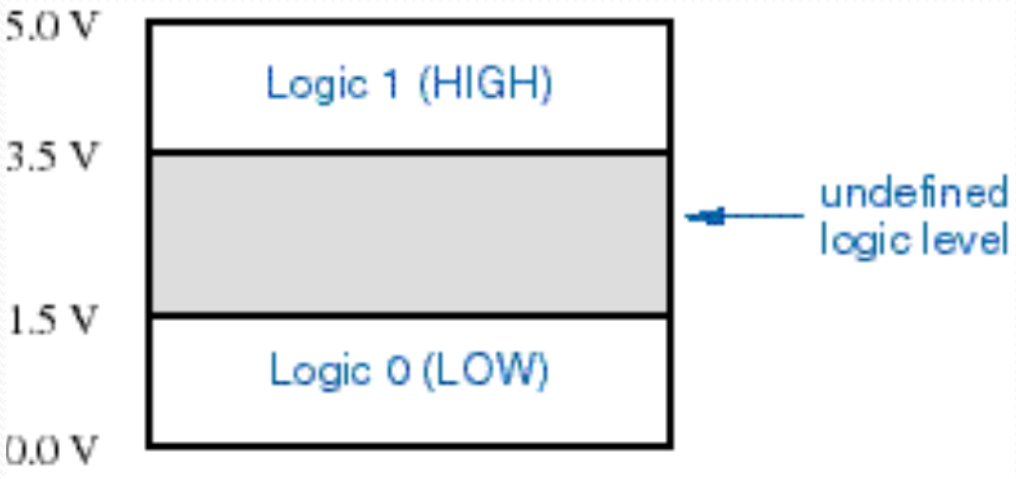
- F2 = x(y'z' + yz).

  The dual of F1 is x + (y' + z')(y + z).

  Complement each literal: x' + (y + z)(y' + z ') = F2' .

Logic Gates

- An electronic circuits that operate on one or more input signals to produce an output signal.
- Electrical signals such as voltages or currents exist throughout a digital system in either one of two recognizable values (bi-state 0 or 1). Voltage-operated circuits respond to two separate voltage ranges that represent a binary variable equal to logic 1 or logic 0.
- Digital logic gates are the physical building blocks of integrated circuits used for the execution of logical operations or tasks by utilizing the Boolean logic.



The graphic symbols and truth tables of the eight gates are shown below:

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

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

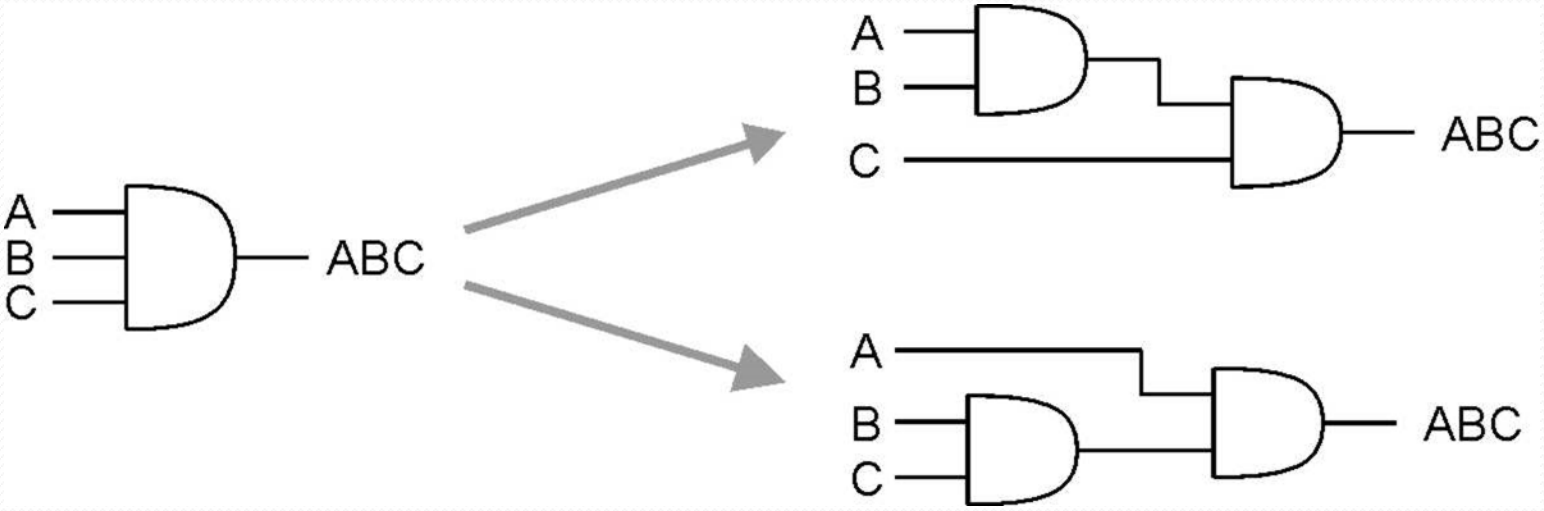
x	y	F
0	0	1
0	1	0
1	0	0
1	1	1

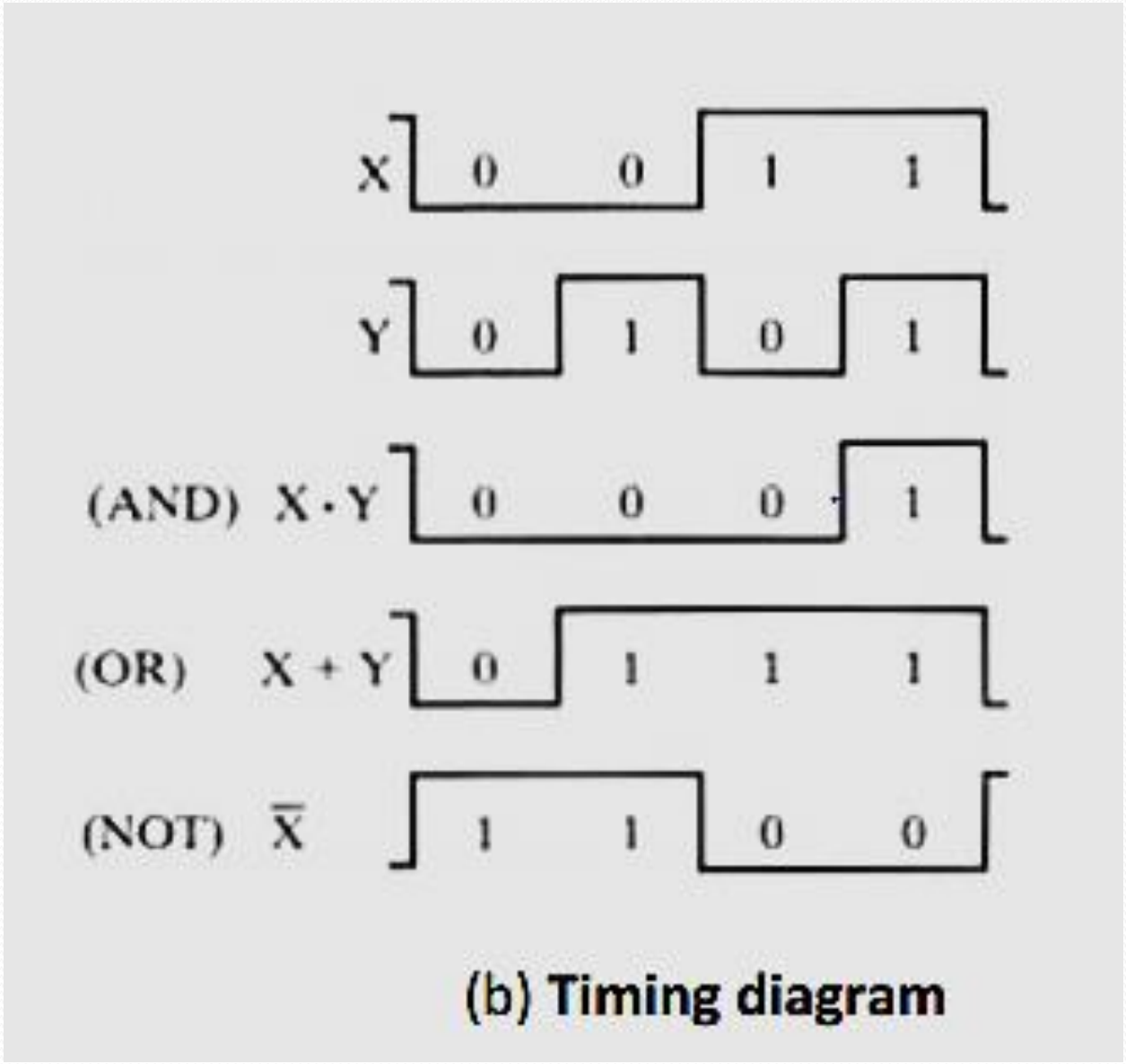
Note:

- These circuits, called *gates*, are blocks of hardware that produce a logic-1 or logic-0 output signal if input logic requirements are satisfied.
- Note that four different names have been used for the same type of circuits: digital circuits, switching circuits, logic circuits, and gates.
- AND and OR gates may have more than two inputs.
- NOT gate is single input circuit, it simply inverts the input.
- **AND, OR and NOT are the Basic Gates.**
- NAND and NOR gates require only 2 transistors.(Why?)
- AND and OR need 3 transistors.(Why?)

- AND/OR can take any number of inputs.
  - AND = 1 if all inputs are 1.
  - OR = 1 if any input is 1.
  - Similar for NAND/NOR.

Can implement with multiple two-input gates,





(b) Timing diagram

Explanation of the above timing diagram:

The two input signals X and Y to the AND and OR gates take on one of four possible combinations: 00, 01, 10, or 11. These input signals are shown as timing diagrams, together with the timing diagrams for the corresponding output signal for each type of gate. The horizontal axis of a timing diagram represents time, and the vertical axis shows a signal as it changes between the two possible voltage levels. The low level represents logic 0 and the high level represents logic 1. AND gate responds with a logic-1 output signal when both input signals are logic-1. The OR gate responds with a logic-1 output signal if either input signal is logic-1.

Universal gates

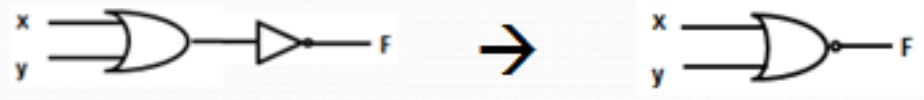
A universal gate is a gate which can implement any Boolean function without need to use any other gate type. **The NAND and NOR gates are universal gates.** In practice, this is advantageous since NAND and NOR gates are economical and easier to fabricate and are the basic gates used in all IC digital logic families.

NAND = AND + NOT or NOT-AND

X	y	F
0	0	1
0	1	1
1	0	1
1	1	0

NOR = OR + NOT or NOT-OR

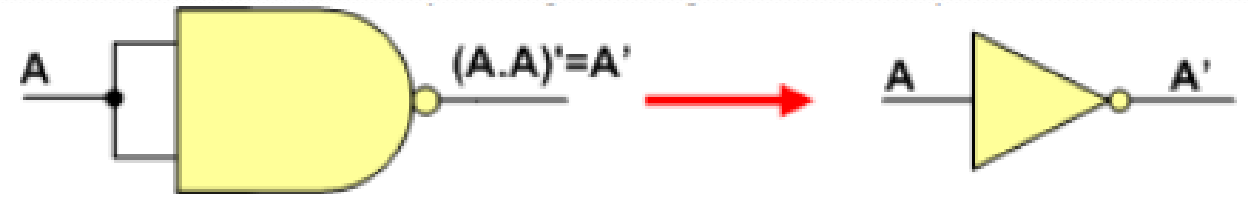
X	y	F
0	0	1
0	1	0
1	0	0
1	1	0



1. NAND Gate is a Universal Gate

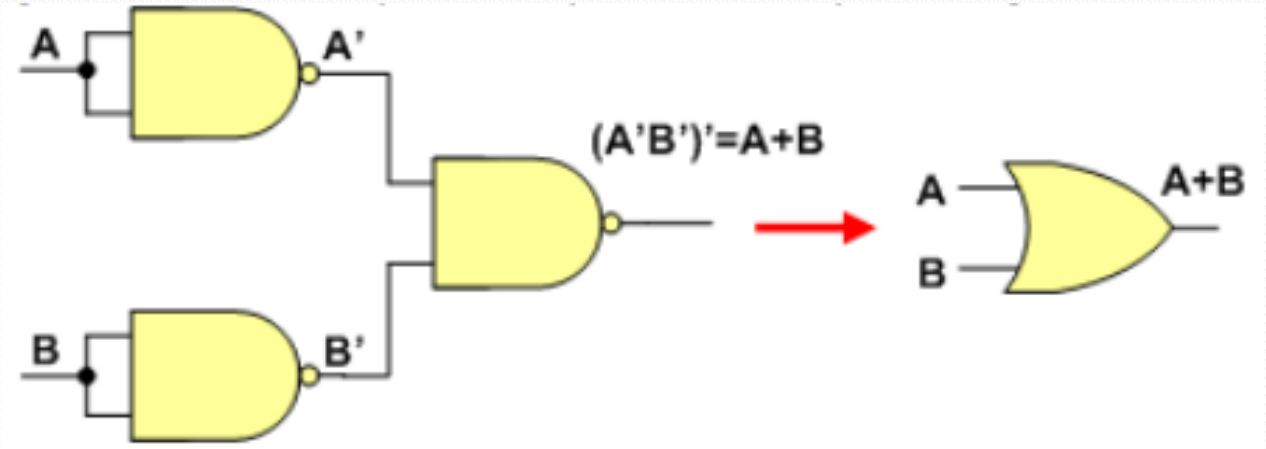
To prove that any Boolean function can be implemented using only NAND gates, we will show that the AND, OR, and NOT operations can be performed using only these gates.

- **Implementing an Inverter Using only NAND Gate:** All NAND input pins connected to the input signal A gives an output A'.



- **Implementing AND Using only NAND Gates:** The AND is replaced by a NAND gate with its output complemented by a NAND gate inverter.

- **Implementing OR Using only NAND Gates:** The OR gate is replaced by a NAND gate with all its inputs complemented by NAND gate inverters.

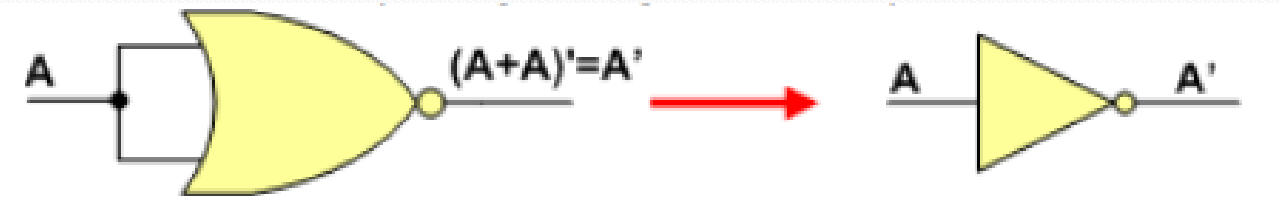


Thus, the **NAND gate** is a universal gate since it can implement the AND, OR and NOT functions.

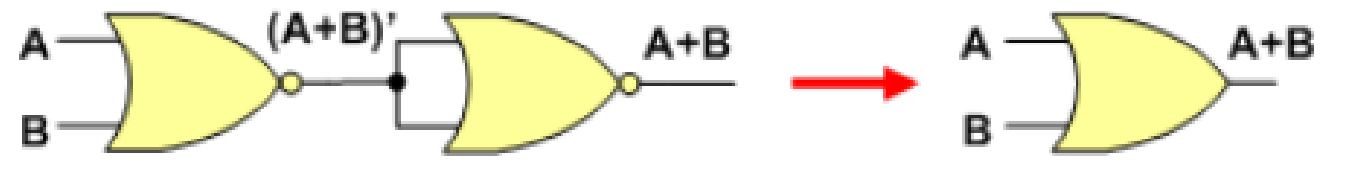
2. NOR Gate is a Universal Gate:

To prove that any Boolean function can not be implemented using only NOR gates, we will show that the AND, OR, and NOT operations can be performed using only these gates.

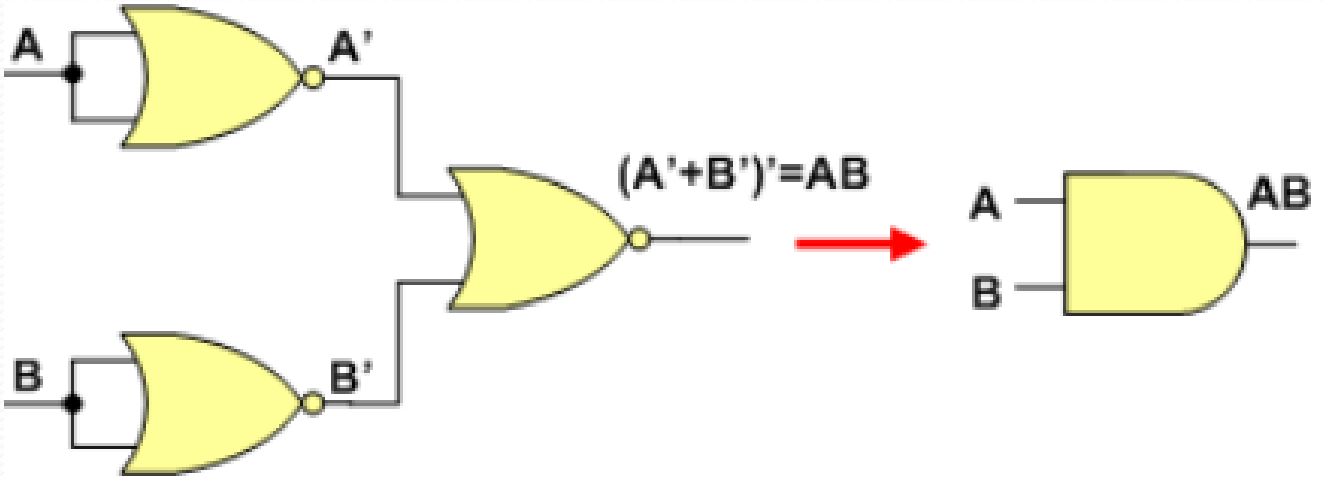
- **Implementing an Inverter Using only NOR Gate:** All NOR input pins connect to the input signal **A** gives an output **A'**.



- **Implementing OR Using only NOR Gates:** The OR is replaced by a NOR gate with its output complemented by a NOR gate inverter.



- **Implementing AND Using only NOR Gates:** The AND gate is replaced by a NOR gate with all its inputs complemented by NOR gate inverters.



Thus, the **NOR gate** is a universal gate since it can implement the AND, OR and NOT functions.

Extending gates to multiple inputs

The gates except for the inverter and buffer, can be extended to have more than two inputs. A gate can be extended to have multiple inputs if the binary operation it represents is commutative and associative.

- The AND and OR operations, defined in Boolean algebra, possess these two properties. For the OR function, we have $x + y = y + x$ *commutative* and $(x + y) + z = x + (y + z) = x + y + z$ *associative*, which indicates that the gate inputs can be interchanged and that the OR function can be extended to three or more variables.
- The NAND and NOR functions are commutative and but not associative [$x \downarrow (y \downarrow z) \neq (x \downarrow y) \downarrow z$].

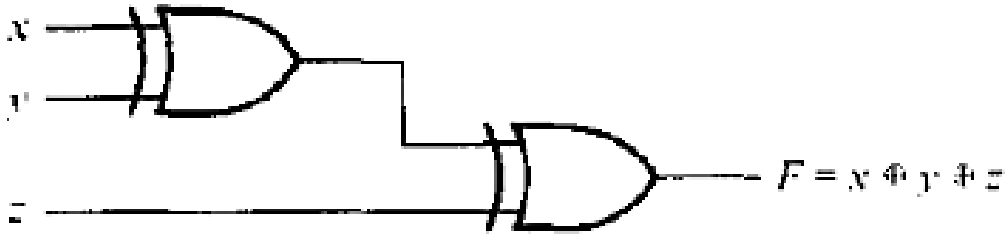
Their gates can be extended to have more than two inputs, provided the definition of the operation is slightly modified. We define the multiple NOR (or NAND) gate as a complemented OR (or AND) gate

i.e. $x \downarrow y \downarrow z = (x + y + z)'$ and $x \uparrow y \uparrow z = (xyz)'$.

(a) Three-input NOR gate

(b) Three-input NAND gate

- The exclusive-OR and equivalence gates are both commutative and associative and can be extended to more than two inputs.



(a) Using 2-input gates

(b) 3-input gate

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

(c) Truth table

Fig: 3-input XOR gate

IC digital logic Families:

Continuing the introduction of integrated Circuits (Chips) in unit 1, let's introduce different logic families along with their characteristics

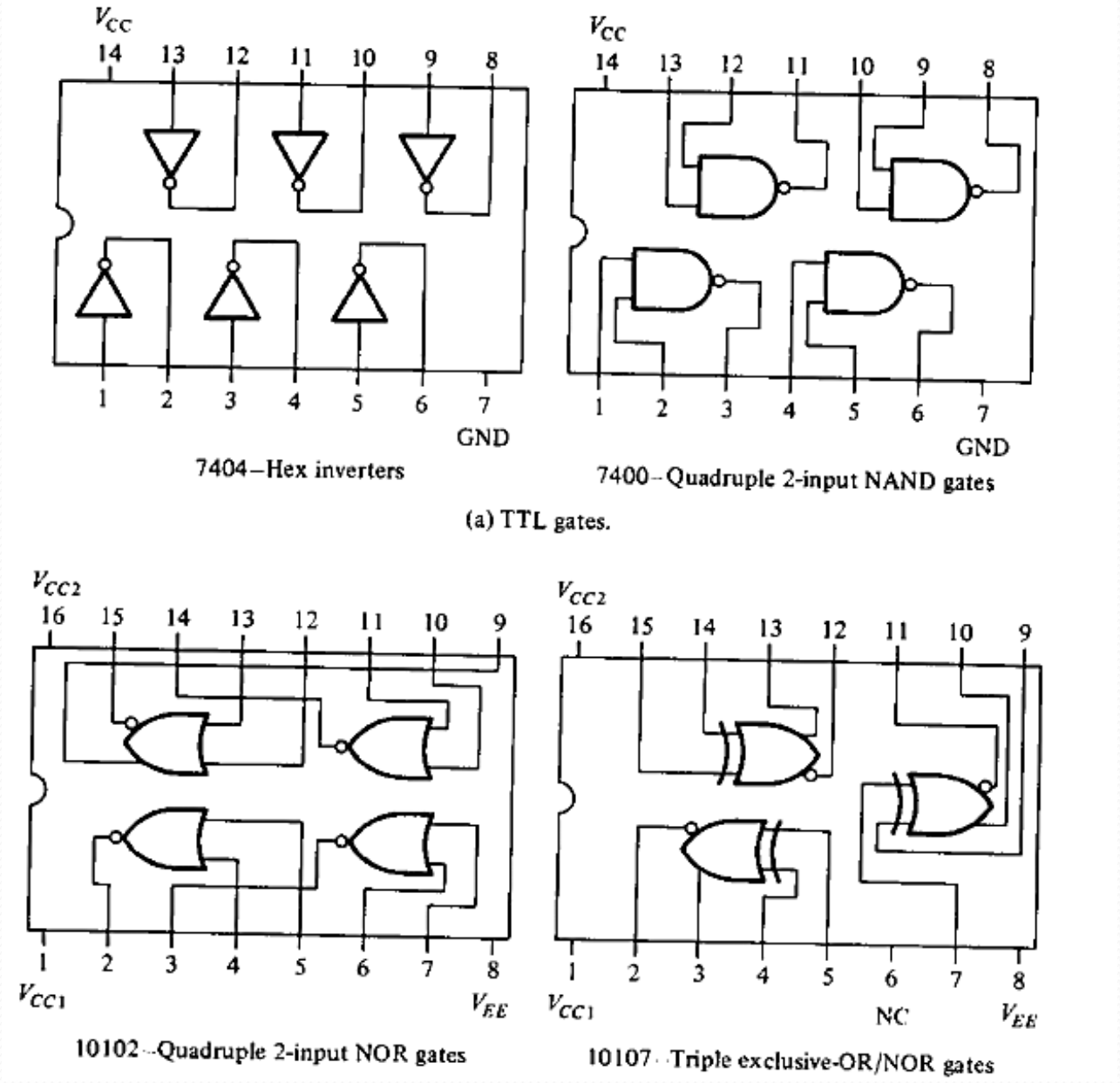
Digital logic families

Digital logic family refers to the specific circuit technology to which digital integrated circuits belong. Family has its own basic electronic circuit upon which more complex digital circuits and components are developed. The basic circuit in each technology is a NAND, NOR, or an inverter gate. The **electronic components used in the construction of the basic circuit are usually used as the name of the technology**

Digital logic gates are the physical building blocks of integrated circuits used for the execution of logical operations or tasks by utilizing the Boolean logic.

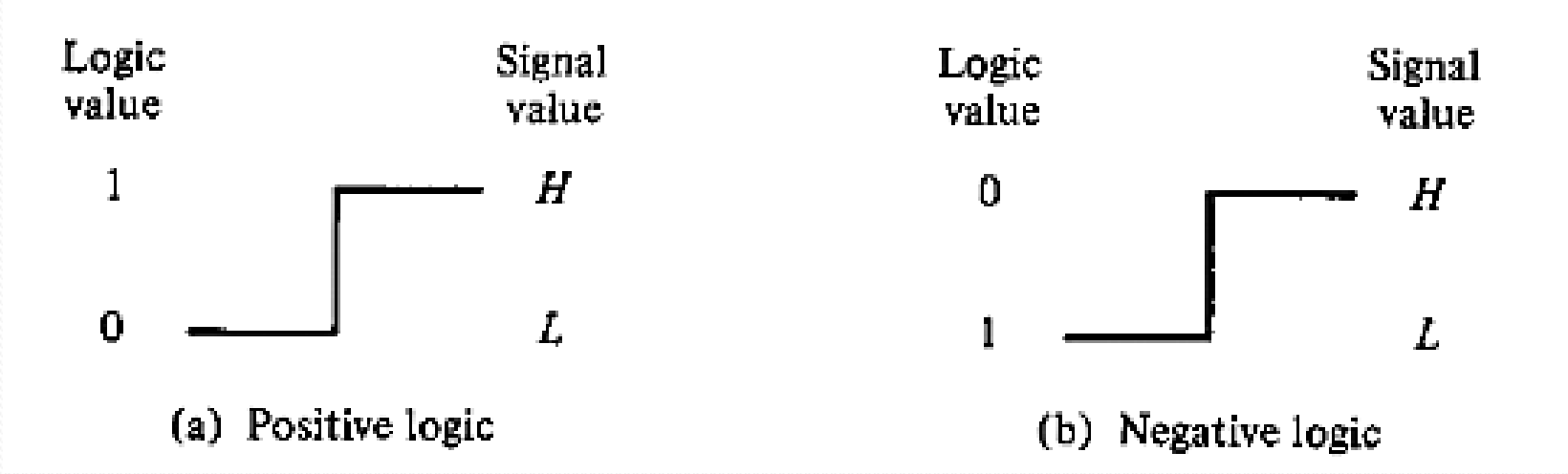
These logic gates are primarily implemented using electronic semiconductor switches such as diodes or transistors. But, later pneumatic logic, molecules, optics, fluidic logic, relay logic (electromagnetic relays) and mechanical elements are used to implement logic gates. Though, in practically logic gates are built using CMOS technology, FETs (field effect transistors), MOSFETs (metal oxide semiconductor field effect transistors).

- TTL (Transistor-Transistor Logic)
- MOS (metal-oxide semiconductor)
- CMOS (Complementary metal-oxide semiconductor)
- IIL (Integrated injection logic)


V_{CC}
14 13 12 11 10 9 8
1 2 3 4 5 6 7
GND
7404—Hex inverters
V_{CC}
14 13 12 11 10 9 8
1 2 3 4 5 6 7
GND
7400—Quadruple 2-input NAND gates
(a) TTL gates.
V_{CC2}
16 15 14 13 12 11 10 9
1 2 3 4 5 6 7 8
V_{CC1}
V_{EE}
10102—Quadruple 2-input NOR gates
V_{CC2}
16 15 14 13 12 11 10 9
1 2 3 4 5 6 7 8
V_{CC1}
NC
V_{EE}
10107—Triple exclusive-OR/NOR gates

Positive and Negative Logic

The binary signal at the inputs and outputs of any gate has one of two values, except during transition. One signal value represents logic-1 and the other logic-0. So there is a possibility of two different assignments of signal level to logic value, as shown in Fig.



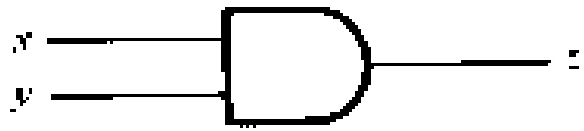
- Choosing the high-level *H* to represent logic-1 defines a **positive logic system**.
- Choosing the low-level *L* to represent logic-1 defines a **negative logic system**.

x	y	z
L	L	L
L	H	L
H	L	L
H	H	H

(a) Truth table with *H* and *L*

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

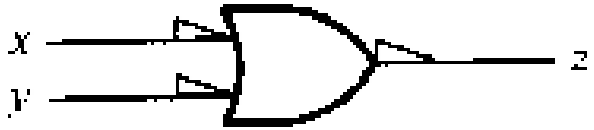
(c) Truth table for positive logic



(d) Positive logic AND gate

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

(e) Truth table for negative logic



(f) Negative logic OR gate

Fig: Demonstration of Positive and negative logic

Characteristics of digital logic families (Technology Parameters)

- **Fan-in :** the number of inputs to a gate.
- **Fan-out :** the number of standard loads driven by a gate output.
- **Power Dissipation :** the amount of power needed by the gate.
- **Propagation delay :** The signals through a gate take a certain amount of time to propagate from the inputs to the output. This interval of time is defined as the **propagation delay of the gate**.
- **Noise Margin :** *Noise margin* is the maximum noise voltage added to an input signal of a digital circuit that does not cause an undesirable change in the circuit output.


Thank You!