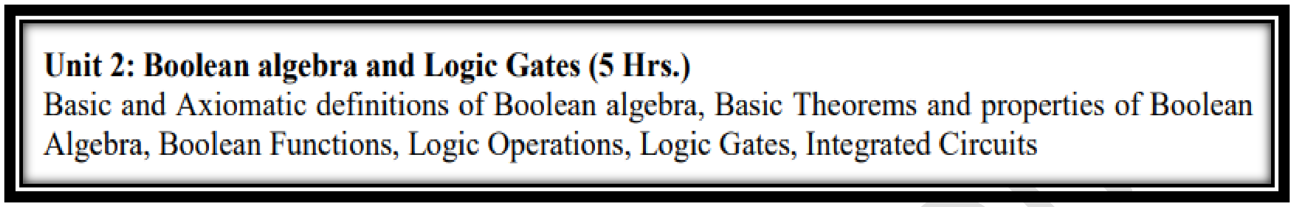
# Unit 2

# Boolean Algebra & Logic Gates



**Unit 2: Boolean algebra and Logic Gates (5 Hrs.)**
Basic and Axiomatic definitions of Boolean algebra, Basic Theorems and properties of Boolean Algebra, Boolean Functions, Logic Operations, Logic Gates, Integrated Circuits

## Basic and Axiomatic Definitions of Boolean Algebra

In 1854 George Boole introduced a systematic treatment of logic and developed for this purpose an algebraic system now called Boolean algebra. In 1938 C. E. Shannon introduced a two-valued Boolean algebra called switching algebra, in which he demonstrated that the properties of bistable electrical switching circuits can be represented by this algebra. Thus, the mathematical system of binary logic is known as Boolean or switching algebra. This algebra is 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. For any given algebra system, there are some initial assumptions, or postulates, that the system follows. We can deduce additional rules, theorems, and other properties of the system from this basic set of postulates. Boolean algebra systems often employ the postulates formulated by E. V. Huntington in 1904.

**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.

That means, the result of any AND, OR and NOT operation results in {0,1}.

P2→ Commutativity: The • and + operators are commutative

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

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

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

$x + (y \bullet z) = (x + y) \bullet (x + z)$, 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.

P6→ There exists at least two elements x, y ∈ B such that x ≠ y. tyey are 0 and 1.

One can formulate many Boolean algebras (viz. set theory, n-bit vectors algebra), depending on the choice of elements of B and the rules of operation. Here, we deal only with a two-valued Boolean algebra, i.e., B = {0, 1}. Two-valued Boolean algebra has applications in set theory and in propositional logic. Our interest here is with the application of Boolean algebra to gate-type circuits.

## Basic Theorems and Properties of Boolean Algebra

### Duality

Postulates of Boolean algebra are found in pairs; one part may be obtained from the other if the binary operators and the identity elements are interchanged. This important property of Boolean algebra is called the duality principle. It states that "Every algebraic expression deducible from the postulates of Boolean algebra remains valid if the operators and identity elements are interchanged". In a two valued Boolean algebra, the identity elements and the elements of the set B are the same: 1 and 0. If the dual of an algebraic expression is desired, we simply interchange OR and AND operators and replace 1's by 0's and 0's by 1's.

### Basic Theorems

The theorems, like the postulates, are listed in pairs; each relation is the dual of the one paired with it. 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:

One variable theorem:

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

Two variable theorem:

Theorem4: Associative (a) x + (y + z) = (x + y) + z (b) x(yz) = (xy)z

Theorem5: Demorgan (a) (x + y)' = x'y' (b) (xy)' = x' + y'

Theorem6: Absorption (a) x + xy = x (b) x(x + y) = x

Proofs:

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

x + x = (x + x) . 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.x=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)

Each step in theorem 1(b) and 1(a) are dual of each other.

THEOREM 2(a): x+1=1

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

THEOREM 2(b): x.0 = 0 by duality of 2(a).

THEOREM 3: (x ')' = x.

From P5, we have x + x' = 1 and x.x' = 0, which defines the complement of x.
The complement of x' is x and is also (x')'.
Therefore, since the complement is unique, we have that (x')' = x.

The theorems involving two or three variables may be proven algebraically from the postulates and the theorems that have already been proven.

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.

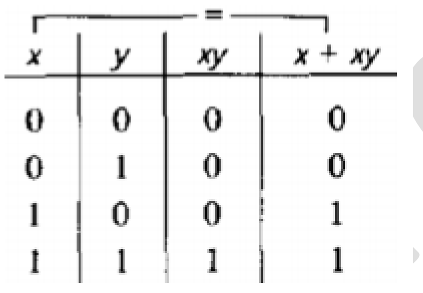
(x + y) + (x'y') = [(x + y) + x'] [(x + y) + y'] [OR distributes over AND (P3)]
= [(y + x) + x'] [(x + y) + y'] [OR is commutative (P2)]
= [y + (x + x')] [x + (y + y')] [OR is associative (Theorem 3(a)), used twice]
= (y + 1)(x + 1) [Complement, x + x' = 1 (P5), twice]

= 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]

Theorems above can also be proved using truth tables (alternative to algebraic simplification). Viz. theorem 6(a) can be proved as:



| $x$ | $y$ | $xy$ | $x + xy$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 1 | 1 | 1 |

## 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’.

## Boolean Functions

Boolean function is an expression formed with:

- Binary variables (each binary variable van take values either 0 or 1).
- Operators (AND, OR, NOT, parenthesis and equality sign).

The resultant value of the Boolean function is either 0 or 1.

For example, consider a Boolean function $F_1 = xyz'$. Here the function $F_1$ is 0 if x=1, y=1 and z=0, otherwise it is 0.

A Boolean function can be represented either in algebraic form or in the form of truth table. The above Boolean function can be represented in truth table as shown below.

| **x** | **y** | **z** | **$z'$** | **$F_1 = xyz'$** |
|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 0 |

To represent a Boolean function having n Boolean variables, in truth table, we need $2^n$ different input values set. As in above Boolean function with 3 variables, there are $2^3 = 8$ different rows in the truth table. Different combinations of 0's and 1's in the rows is easily obtained by going from binary 0 to $2^n$-1. For each row, value of the Boolean function is either 0 or 1.

In a Boolean function, different operators are represented differently.

- AND operator is represented by dot '.' Or simply by the absence of any operator.
- OR operator is represented by '+'.
- NOT operator is represented by a dash superscripted in the variable name or simply by a bar above the variable name.
- Parenthesis is simply represented by '()'.

A Boolean function can have multiple representations. In other words, there may be a simple representation equivalent to a complex looking Boolean function. Two Boolean functions are said to be same or equivalent if their truth table is identical. For example, consider the two Boolean functions as

$F_2 = x'y'z + x'yz + xy'$ and $F_3 = xy' + x'z$

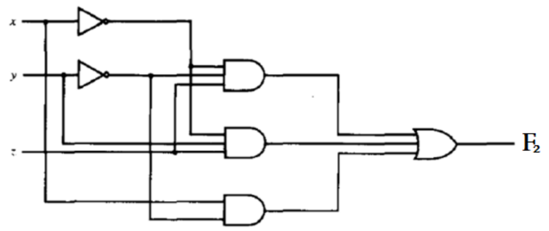
Constructing the truth tables for these Boolean functions;

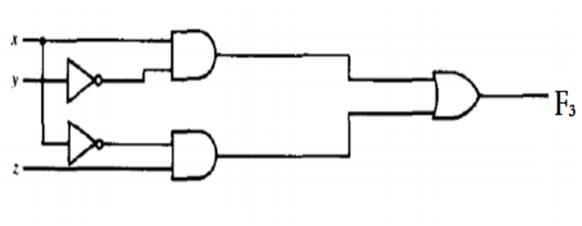
| **x** | **y** | **z** | **$x'$** | **$y'$** | **$z'$** | **$F_2 = x'y'z + x'yz + xy'$** | **$F_3 = xy' + x'z$** |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |

From the above truth table, we can say that the Boolean functions $F_2$ and $F_3$ are same. We can simplify the Boolean Function to get as simpler expression as possible. This will help to transform a Boolean function to a logic circuit (ie. From logic expression to a logic diagram).

We can transform a logic expression (Boolean expression) into logic diagram by using the AND, OR and NOT logic gates.

We can represent the Boolean functions $F_2 = x'y'z + x'yz + xy'$ and $F_3 = xy' + x'z$ in logic diagram as shown below.





Both of the above logic diagrams represent the same logic, however the logic for $F_2$ requires more logic gates and is complicated as well in comparison to that of $F_3$. Thus, it is always important to manipulate the Boolean expression to get a simple form.

## Algebraic Manipulation or simplification of the Boolean Functions

Algebraic manipulation or simplification of Boolean functions is important to simplify the corresponding logic circuit and hence reduce the circuit equipment and costs. We can simplify a Boolean expression by reducing the literals (primes or unprimed Boolean variable) and the terms. Each literal in an expression indicates an input to the logic circuit and each term in an Boolean expression represents a logic gate. So, by reducing the literals, we reduce the number of inputs and by reducing the terms, we reduce the number of equipment needed.

The detail about simplifying the Boolean expression will be discussed more in chapter 3. Here, we discuss only the basics of simplifying Boolean expression using algebraic manipulation. By using algebraic manipulation, we can reduce the number of literals.

Unfortunately, there are no specific rules to give the guarantee of getting simplified final answer. The only method available is to cut-and-try following the basic postulates of Boolean algebra and the basic theorems.

Example: Simplify the following Boolean functions to a minimum number of literals.

**1. $x + x'y$**

**Ans→** $x + x'y = (x+x')(x+y)$

$= 1.(x+y)$

$= x+y$

(number of literals reduced from 3 to 2)

**2. $x(x'+y)$**

**Ans→** $x(x'+y) = xx' + xy$

$= 0 + xy$

$= xy$

(number of literals reduced from 3 to 2 & number of terms reduced from 2 to 1)

**3. $x'y'z + x'yz + xy'$**

**Ans→** $x'y'z + x'yz + xy' = x'z\,(y' + y) + xy'$

$= x'z + xy'$

(number of literals reduced from 8 to 4 & terms reduced from 3 to 2)

**4. $xy + x'z + yz$**

**Ans→** $xy + x'z + yz \quad = xy + x'z + yz(x+x')$

$= xy + x'z + xyz + x'yz$

$= xy(1+z) + x'z\,(1+y)$

$= xy + x'z$

(number of literals reduced from 6 to 4 and terms reduced from 3 to 2)

**5. $(x+y)(x'+z)(y+z)$**

**Ans→** we already proved that, $xy + x'z + yz = xy + x'z$.

From the duality principle, we can write the following expression.

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

## Complement of a Boolean function

The complement of a function F is denoted by F' and is obtained by interchanging 0's for 1's and 1's for 0's in the values of F. Complement of a function may be derived by using the De Morgan's theorem.

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, which can be represented by following equations.

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

**Q. Find the complement of Boolean function $x'yz' + x'y'z$.**

**Ans→** Here $F = x'yz' + x'y'z$

Complement of F is written as;

F’ = (x’yz’ + x’y’z)’

Applying De Morgan’s law successively, we can write;

F’ = (x’yz’)’ (x’y’z)’

= (x+y’+z)(x+y+z’) which is the complement of F.

**Q. Find the complement of Boolean function x(y’z’ +yz).**

**Ans→** Here F = x (y’z’ +yz)

Complement of F is written as;

F’ = [x (y’z’ +yz)]’

Applying De Morgan’s law successively, we can write;

F’ = x’+ (y’z’ +yz)’

= x’ + (y’z’)’.(yz)’

= x’ + (y+z)(y’+z’) which is the complement of F.

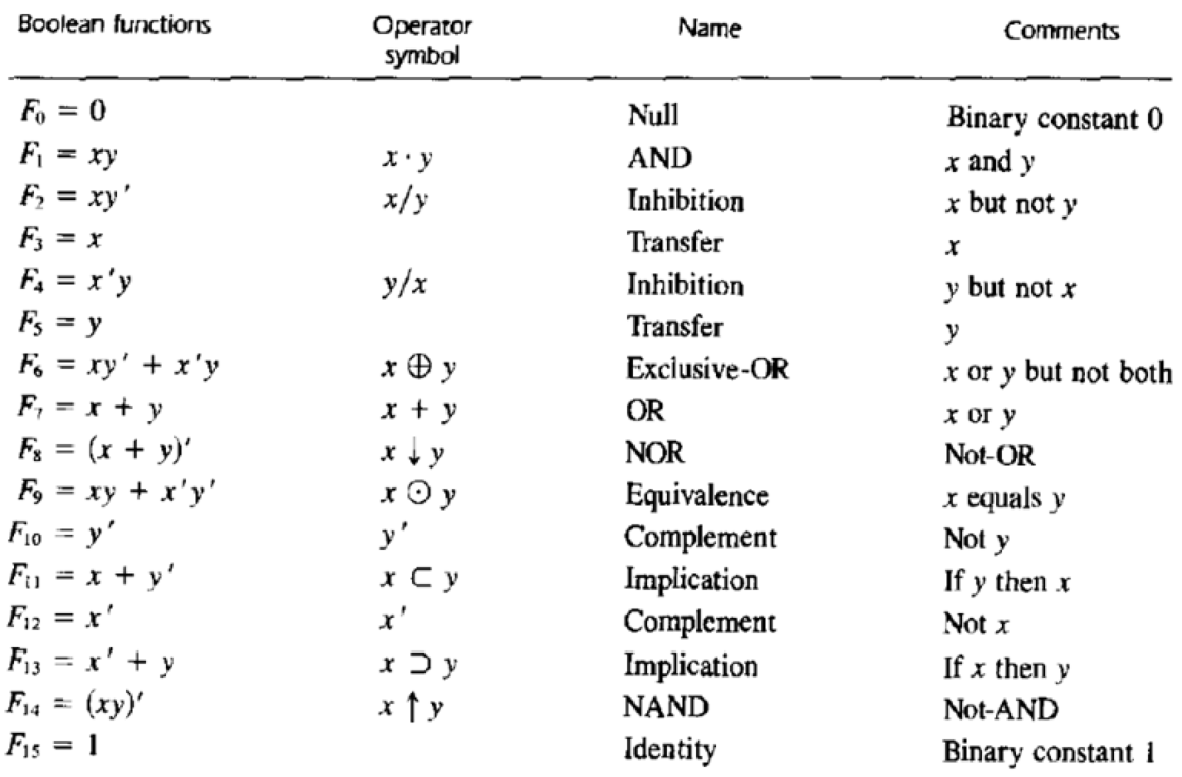
**Note:** Easiest method to find the complement of a function is;

- Firstly, find the dual of the function.
- And then complement each literal.

For example, for the function F = x’yz’ + x’y’z
The dual of F is (x’+y+z’)(x’+y’+z)
Now, complementing each literal we get the complement of F as follows;
F’ = (x+y’+z)(x+y+z’)

## Logic Operations:

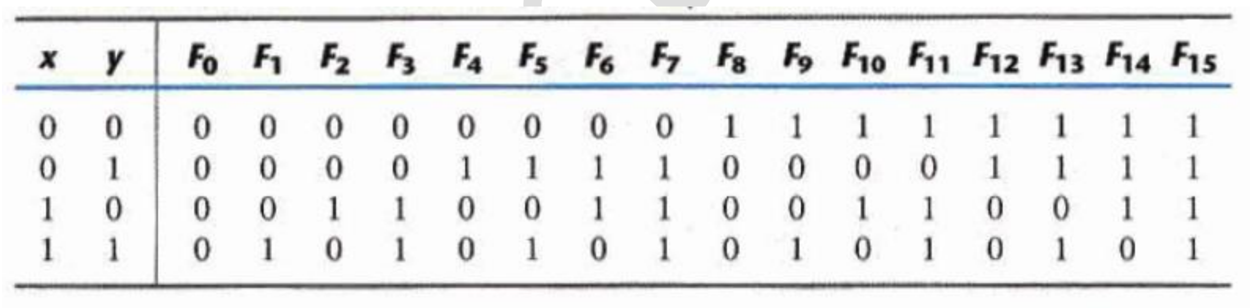
AND, OR and NOT are the fundamental logic operations. With the combinations of these basic logic operations, we can construct other compound logic operations. It is found that with n binary variables, we can construct $2^{2n}$ different logic operations or Boolean functions. For 2 binary variables, we can construct $2^{2x2}$ =16 different logic operations as shown below.



| Boolean functions | Operator symbol | Name | Comments |
|---|---|---|---|
| $F_0 = 0$ | | 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 |

Table: **Boolean Expressions for the 16 Functions of Two Variables**

The truth table for above 16 functions with two binary variables is as shown below.



| $x$ | $y$ | $F_0$ | $F_1$ | $F_2$ | $F_3$ | $F_4$ | $F_5$ | $F_6$ | $F_7$ | $F_8$ | $F_9$ | $F_{10}$ | $F_{11}$ | $F_{12}$ | $F_{13}$ | $F_{14}$ | $F_{15}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 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 |

he 16 functions listed can be subdivided into three categories:

1. Two functions that produce a constant 0 or 1 ($F_0$ and $F_{15}$).
2. Four functions with unary operations: complement and transfer ($F_3$, $F_5$, $F_{10}$, $F_{12}$).
3. Ten functions with binary operators that define eight different operations: AND, OR, NAND, NOR, exclusive OR, equivalence ($F_9$), inhibition ($F_2$ and $F_4$), and implication.

| Operation | Description |
|---|---|
| Constant Boolean Functions | It is a unary operation. That generates 0 or 1 irrespective of any input (See $F_3$ and $F_5$). |
| Complement Functions | It is a unary operation that generates the output opposite of the input (See $F_{10}$ and $F_{12}$). |
| Transfer Functions | It is a unary operation that simply generates the output same as input (See $F_3$ and $F_5$). |

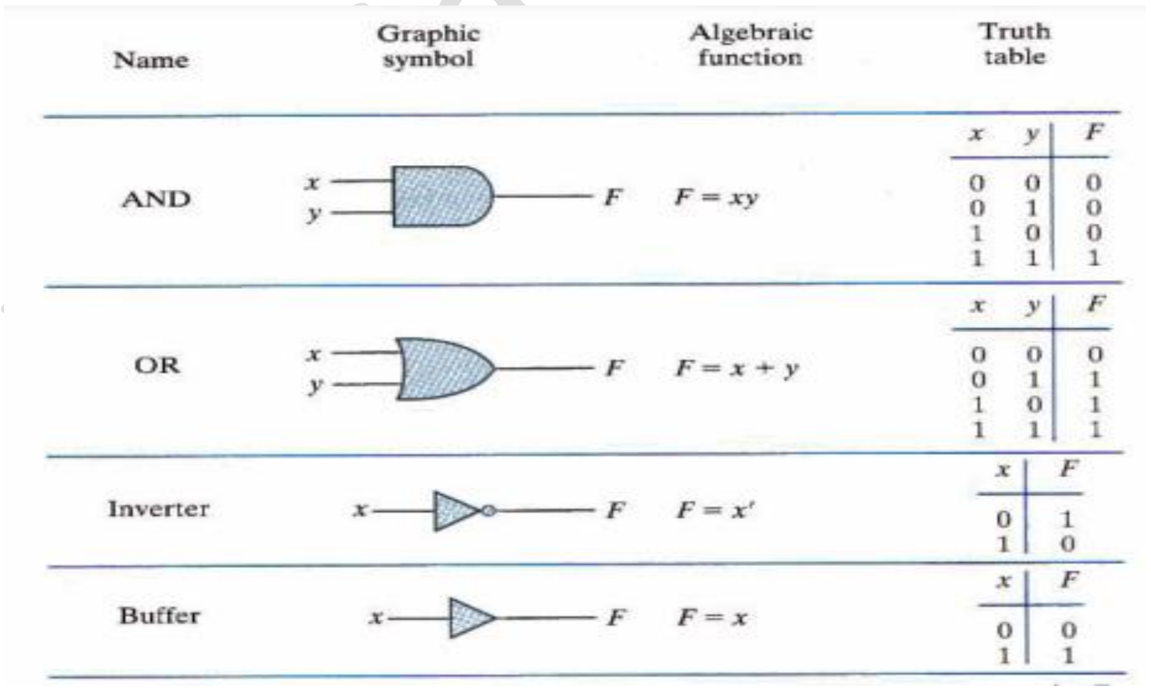
| | |
|---|---|
| Equivalence | It is a binary operation that generates true if both the inputs are identical, otherwise false. In fact, if there are even number of 0's, the result is 1. |
| Inhibition and implication | They are the compound Boolean operations used by logician as per requirement. These are not commutative and associative, so they are not practical to use as standard Boolean functions. |
| AND, OR, NAND, NOR, XOR | They are the binary operations most commonly used in logic design. |

## Logic Gates

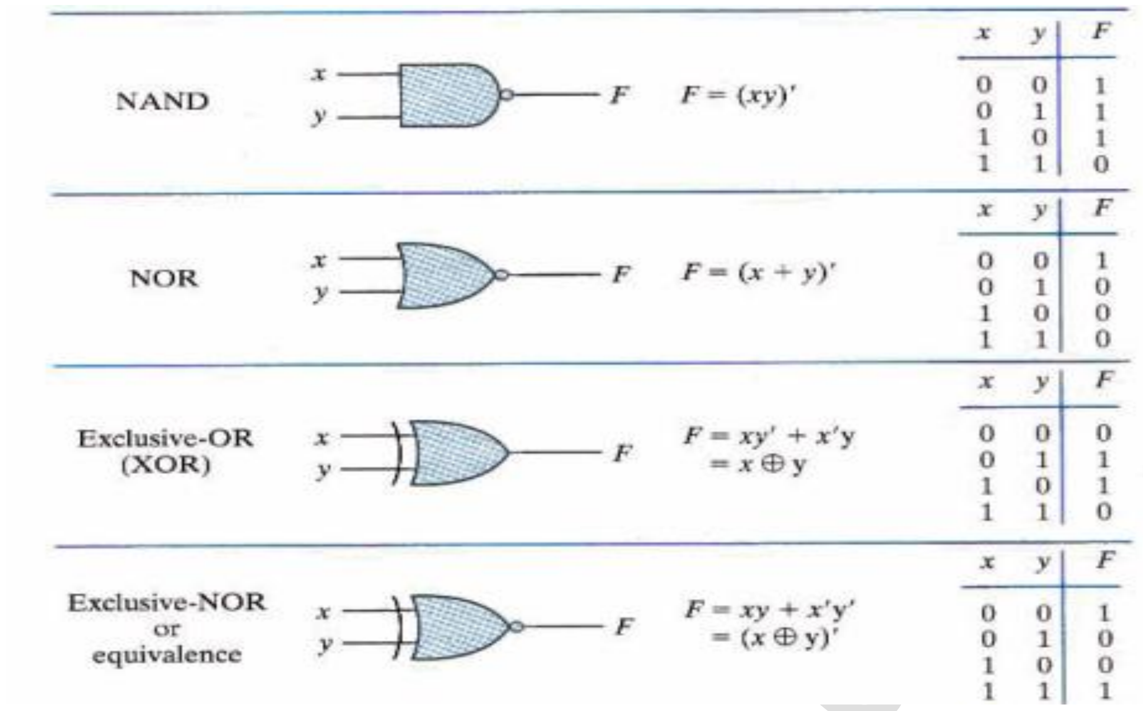
Boolean functions are expressed in terms of AND, OR, and NOT logic operations, and hence are easier to implement with these types of gates. The possibility of constructing gates for the other logic operations is of practical interest. Factors to be weighed when considering the construction of other types of logic gates are:

- The feasibility and economy of producing the gate with physical components.
- The possibility of extending the gate to more than two inputs.
- The basic properties of the binary operator such as commutativity and associativity.
- The ability of the gate to implement Boolean functions alone or in conjunction with other gates.

Of the 16 functions defined in Table above, two are equal to a constant and four others are repeated twice. There are only ten functions left to be considered as candidates for logic gates. Two, inhibition and implication, are not commutative or associative and thus are impractical to use as standard logic gates. The other eight: complement, transfer, AND, OR, NAND, NOR, exclusive-OR, and equivalence, are used as standard gates in digital design. The graphic symbols and truth tables of the eight gates are shown below:



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



| | | | x | y | F |
|---|---|---|---|---|---|
| NAND | | $F = (xy)'$ | 0 | 0 | 1 |
| | | | 0 | 1 | 1 |
| | | | 1 | 0 | 1 |
| | | | 1 | 1 | 0 |
| NOR | | $F = (x + y)'$ | 0 | 0 | 1 |
| | | | 0 | 1 | 0 |
| | | | 1 | 0 | 0 |
| | | | 1 | 1 | 0 |
| Exclusive-OR (XOR) | | $F = xy' + x'y = x \oplus y$ | 0 | 0 | 0 |
| | | | 0 | 1 | 1 |
| | | | 1 | 0 | 1 |
| | | | 1 | 1 | 0 |
| Exclusive-NOR or equivalence | | $F = xy + x'y' = (x \oplus y)'$ | 0 | 0 | 1 |
| | | | 0 | 1 | 0 |
| | | | 1 | 0 | 0 |
| | | | 1 | 1 | 1 |

x
y
F
x
y
F
x
y
F
x
y
F

Note:
- In XOR gate, odd no of 1's in the input give 1 as output.
- In Equivalence Boolean operation, even no of 0's in input give 1 as output. It is not standard operation and has no separate logic gate for this operation.

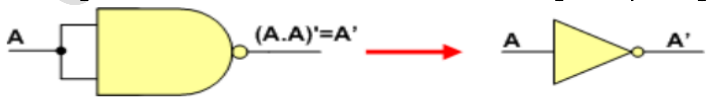
You must be able to draw the logic gate circuit diagram for any given Boolean functions using any of the above logic gates.

**NAND and NOR Gates as Universal Logic gates:**

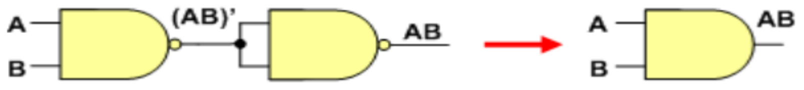
A universal gate is a gate which can implement any Boolean function without need to use any other gate type. The NAND and NOR gates are universal gates. In practice, this is advantageous since NAND and NOR gates are economical and easier to fabricate and are the basic gates used in all IC digital logic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

We can use NAND gate as universal Gate. Since Boolean expression uses AND, OR and NOT logic, to prove that NAND gate can be used as universal gate, we have to show that we can construct AND, OR and NOT logic by using only NAND gate.

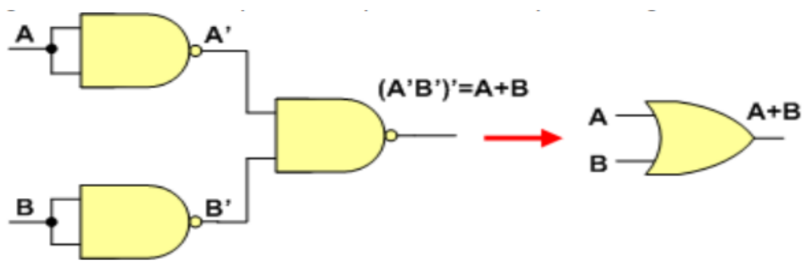
- NAND gate as inverter: We can construct NOT gate by using NAND gate as follows.



- NAND gate as AND gate: We can construct AND gate logic by using two NAND gates as follows.

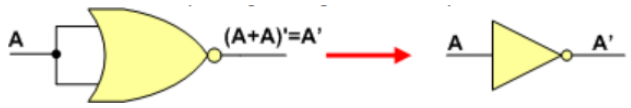


- NAND gate as OR gate: We can construct OR gate logic by using three NAND gates as follows.

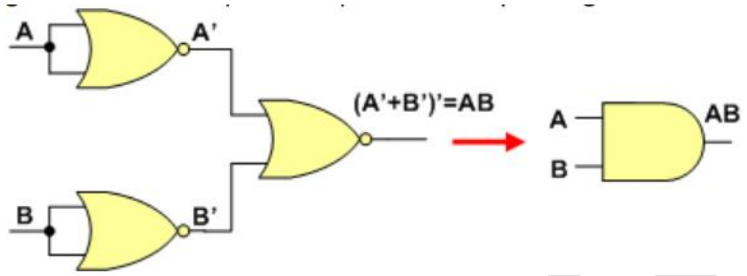


Similarly, NOR gat can also be used as universal gate. We can construct the fundamental AND, OR and NOT gates by using only NOR gate as follows.

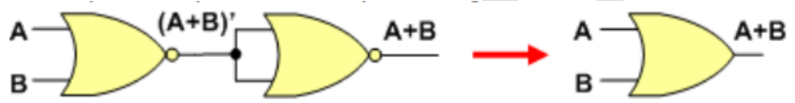
- NOR gate as NOT gate: We can construct NOT gate from a single NOR gate as follows;



- NOR gate as AND gate: We can construct AND gate from three NOR gates as follows;



- NOR gate as OR gate: We can construct AND gate from two NOR gates as follows;



Hence, we can use NAND gate and NOR gates to implement any logical expression into circuits.

**Multiple Input Logic Gates:**

Single input logic gates like NOT gate and buffers always have single input. The other logic gates can be extended to have more than one inputs. A gate can be extended to have multiple inputs if the binary operation it represents is commutative and associative.

For example, consider an OR gate. 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. Similarly, the AND function also possesses commutative and associative property. Hence, we can have multiple input AND gate and OR gate.

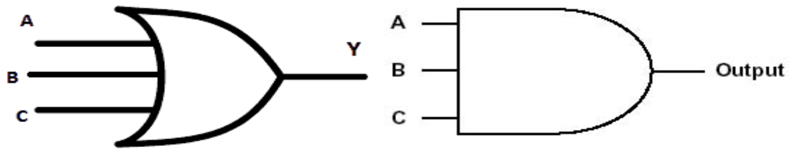


Fig: Three input OR gate and AND gate

The NOR function is commutative ie. X↓Y↓Z = Z↓Y↓X but not associative X↓(Y↓Z) ≠ ( X↓Y)↓Z.

X↓(Y↓Z) = [X+(Y+Z)']' = X'(Y+Z) = X'Y + X'Z

$$(X\downarrow Y)\downarrow Z = [(X+Y)'+Z]' = (X+Y)Z' = XZ' + YZ'$$

But we can define NOR gate as the complement of OR gate. Then it becomes commutative.

$$X\downarrow Y\downarrow Z = (X+Y+Z)'$$

Similar case (as that of NOR gate) exists for NAND gate also.

## Integrated Circuits

An Integrated circuit is an association (or connection) of various electronic devices such as resistors, capacitors and transistors etched (or fabricated) to a semiconductor material such as silicon or germanium. It is also called as a chip or microchip. An IC can function as an amplifier, rectifier, oscillator, counter, timer and memory. Sometime ICs are connected to various other systems to perform complex functions.

Following are the benefits of using IC's.

- In consumer electronics, ICs have made possible the development of many new products, including personal calculators and computers, digital watches, and video games.
- They have also been used to improve or lower the cost of many existing products, such as appliances, televisions, radios, and high-fidelity equipment.
- The logic and arithmetic functions of a small computer can now be performed on a single VLSI chip called a microprocessor.
- Complete logic, arithmetic, and memory functions of a small computer can be packaged on a single printed circuit board, or even on a single chip.

### Levels of Integration

During 1959 two different scientists invented IC's. Jack Kilby from Texas Instruments made his first germanium IC during 1959 and Robert Noyce made his first silicon IC during the same year. But ICs were not the same since the day of their invention; they have evolved a long way. Integrated circuits are often classified by the number of transistors and other electronic components they contain.

On the basis of the level of integration, the IC's are of following types;

- SSI (small-scale integration): Up to 100 electronic components per chip.
- MSI (medium-scale integration): From 100 to 3,000 electronic components per chip.
- LSI (large-scale integration): From 3,000 to 100,000 electronic components per chip.
- VLSI (very large-scale integration): From 100,000 to 1,000,000 electronic components per chip.
- ULSI (ultra-large-scale integration): More than 1 million electronic components per chip.

### SIP and DIP IC's

On the basis of fabrication model, IC's may be SIP or DIP.

1. **SIP**: A single in-line package (SIP) is an electronic device package which has one row of connecting pins. It is not as popular as the dual in-line package (DIP) which contains two rows of pins, but has been used for packaging RAM chips and multiple resistors with a common pin. SIPs group RAM chips together on a small board. The board itself has a single row of pin-leads that resembles a comb extending from its bottom edge, which plug into a special socket on a system or system-expansion board. SIPs are commonly found in memory modules.

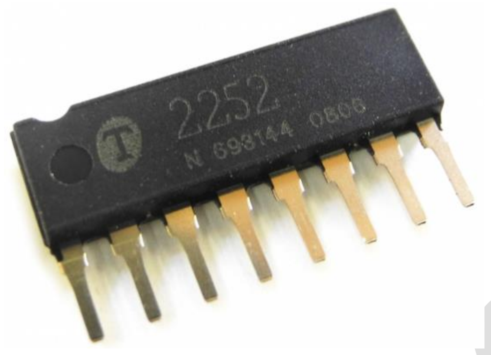


Fig: SIP8 IC

2. **DIP**: Dual in-line package (DIP) is a type of semiconductor component packaging. DIPs can be installed either in sockets or permanently soldered into holes extending into the surface of the printed circuit board. DIP is relatively broadly defined as any rectangular package with two uniformly spaced parallel rows of pins pointing downward, whether it contains an IC chip or some other device(s), and whether the pins emerge from the sides of the package and bend downwards. A DIP is usually referred to as a DIPn, where n is the total number of pins.

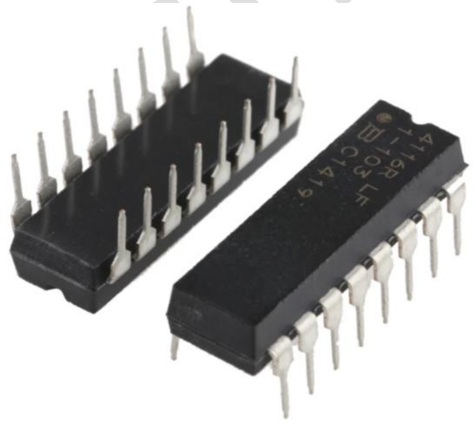


Fig: DIP6 IC

Based on the packaging materials, DIP ICs are of various types:

- Ceramic Dual In-line Package (CERDIP or CDIP)
- Plastic Dual In-line Package (PDIP)
- Shrink Plastic Dual In-line Package (SPDIP) -A denser version of the PDIP

**SIMM (Single In-line Memory Module) and DIMM (Dual In-line Memory Module)**

SIMM stands for Single In-line Memory Module, SIMM is a circuit board that holds six to nine memory chips per board, the ninth chip usually an error checking chip (parity) and were commonly used with Intel Pentium or Pentium compatible motherboards. SIMMs are rarely used today and have been widely replaced by DIMMs. SIMMs are available in two flavors: 30 pin and 72 pin. 30-pin SIMMs are the older standard, and were popular on third and fourth generation motherboards. 72-pin SIMMs are used on fourth, fifth and sixth generation PCs.

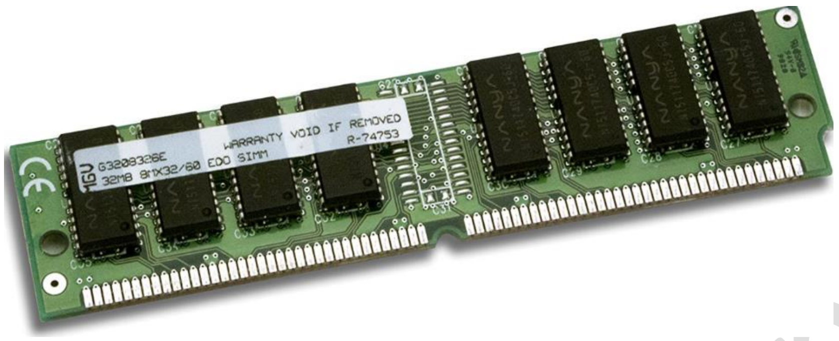


Fig: 32 MB 72 Pin SIMM

DIMM stands for Dual In-line Memory Module, DIMM is a circuit board that holds memory chips. DIMMs have a 64-bit path because of the Pentium Processor requirements. Because of the new bit path, DIMMs can be installed one at a time, unlike SIMMs on a Pentium that would require two to be added. Below is an example image of a 512MB DIMM memory stick.

Fig: 512MB 168 pin DIMM

Following are the major differences between SIMM and DIMM:

| SIMM | DIMM |
|---|---|
| SIMM supports 32 bit channel for data transferring. | DIMM supports 64 bit channel for data transferring. |
| SIMM consumes 5 volts of power. | DIMM consumes 3.3 volts of power. |
| SIMM provides less storage capacity. | DIMM provides high storage capacity. |
| The classic or most common pin configuration of the SIMM module is 72 pins. | The foremost common pin configuration of the DIMM module is 168 pins. |
| SIMMs are the older technology. | DIMMs are the replacement of the DIMMs. |
| SIMMs are installed in pairs at a time. | DIMMs are installed one at a time. |

| | |
|---|---|
| There is single notch in SIMM. | There are two notches in DIMM. |

## IC 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. Different logic families have been introduced commercially. Some of most popular are:

- **TTL (transistor-transistor logic):** The TTL family evolved from a previous technology that used diodes and transistors for the basic NAND gate. This technology was called DTL for diode-transistor logic. Later the diodes were replaced by transistors to improve the circuit operation and the name of the logic family was changed to TTL. It is ***more common*** in use.

- **ECL (emitter-coupled logic):** Emitter-coupled logic (ECL) circuits provide the ***highest speed*** among the integrated digital logic families. ECL is used in systems such as supercomputers and signal processors, where high speed is essential. The transistors in ECL gates operate in a non-saturated state, a condition that allows the achievement of propagation delays of 1 to 2 nanoseconds.

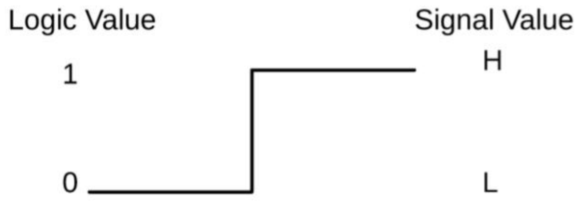
- **MOS (metal-oxide semiconductor):** The metal-oxide semiconductor (MOS) is a unipolar transistor that depends upon the flow of only one type of carrier, which may be electrons (n-channel) or holes (p-channel), this is in contrast to the bipolar transistor used in TTL and ECL gates, where both carriers exist during normal operation. A p-channel MOS is referred to as PMOS and an n-channel as NMOS. NMOS is the one that is commonly used in circuits with only one type of MOS transistor. It is used when ***high component*** density is required.

- **CMOS (complementary metal-oxide semiconductor):** Complementary MOS (CMOS) technology uses one PMOS and one NMOS transistor connected in a complementary fashion in all circuits. The most important advantages of MOS over bipolar transistors are the high packing density of circuits, a simpler processing technique during fabrication, and a more economical operation because of the ***low power*** consumption. Currently, silicon-based Complementary Metal Oxide Semiconductor (CMOS) technology dominates due to its high circuit density, high performance, and low power consumption. Alternative technologies based on Gallium Arsenide (GaAs) and Silicon Germanium (SiGe) are used selectively for very high speed circuits.

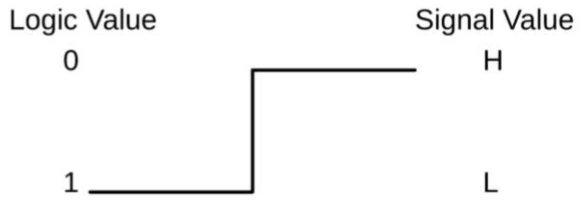
- **IIL (Integrated Injection Logic):** Integrated injection logic (IIL, I 2 L, or I2L) is a class of digital circuit technology built with multiple collector bipolar junction transistors (BJT). When introduced it had ***speed*** comparable to TTL yet was almost as ***low power*** as CMOS, making it ideal for use in VLSI (and larger) integrated circuits. Although the logic voltage levels are very close (High: 0.7V, Low: 0.2V), I2 L has high noise immunity because it operates by current instead of voltage. Sometimes also known as Merged Transistor Logic.

## 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.



(a) Positive Logic

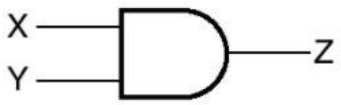


(b) Negative Logic

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. The terms positive and negative are somewhat misleading since both signals may be positive or both may be negative. It is not the actual signal values that determine the type of logic, but rather the assignment of logic values to the relative amplitudes of the two signal levels.

| X | Y | Z |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

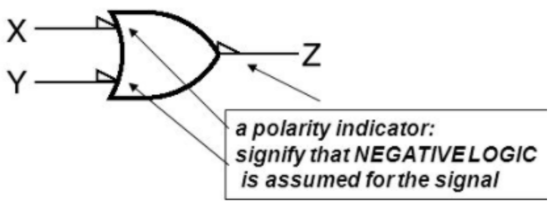
(c) Truth table for positive logic



(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


X
Y
Z
a polarity indicator: signify that NEGATIVE LOGIC is assumed for the signal

(f) Negative-logic OR gate

## Special Characteristics of IC logic families

For each specific implementation technology, there are details that differ in their electronic circuit design and circuit parameters. The most important parameters used to characterize an implementation technology are:

- **Fan Out:** The total number of logic gates that can be powered by the output of a single logic gate is called fan out. It is desired to have an IC with higher Fan Out value.

- **Fan In:** Maximum number of inputs in a logic gate inside an IC is called Fan In.

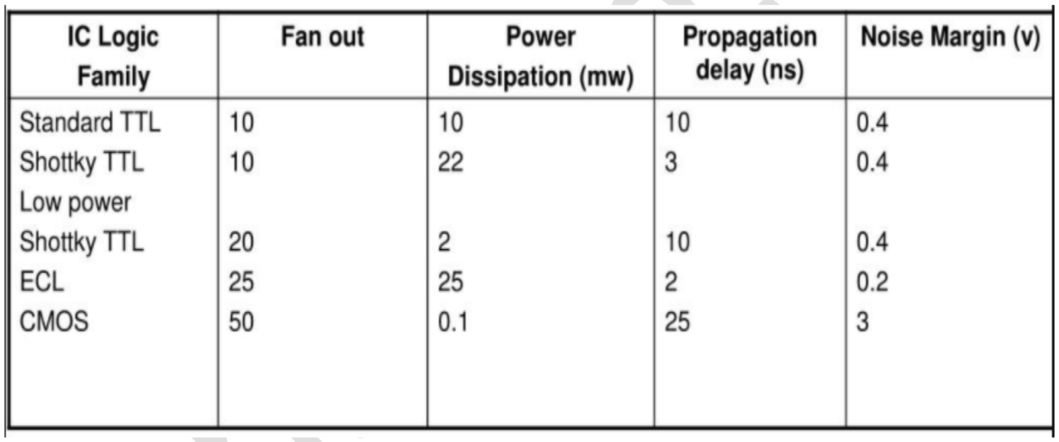
- **Power Dissipation:** Power consumed by a single logic level. Or, we can also understand it as the power consumed per logic gate. It is desired to have an IC having less power dissipation.

- **Propagation Delay:** It is the time interval between the value of an input is changed and the corresponding result is obtained. It is desired to have less propagation delay for faster ICs.

- **Noise Margin:** It is the amount of voltage that can be increased without affecting the output logic. An IC having high noise margin can tolerate higher voltage fluctuation.

- **Breadth:** The total number of functionalities an IC possesses is called as its breadth. An IC with higher breadth means a versatile IC that can be used for multiple operations.

Following table shows the typical characteristics or parameters of the ICs of different logic families.



| IC Logic Family | Fan out | Power Dissipation (mw) | Propagation delay (ns) | Noise Margin (v) |
|---|---|---|---|---|
| Standard TTL | 10 | 10 | 10 | 0.4 |
| Shottky TTL | 10 | 22 | 3 | 0.4 |
| Low power Shottky TTL | 20 | 2 | 10 | 0.4 |
| ECL | 25 | 25 | 2 | 0.2 |
| CMOS | 50 | 0.1 | 25 | 3 |

We can use this table to compare different IC logic families and select an appropriate IC according to the requirement. For example, if we want faster ICs, we may choose one having low propagation delay (ie ECL). If we need ICs having low power consumption, we may choose COMS logic family.