Introduction to Digital System

"Digital systems are designed to store, process, and communicate information in digital form."

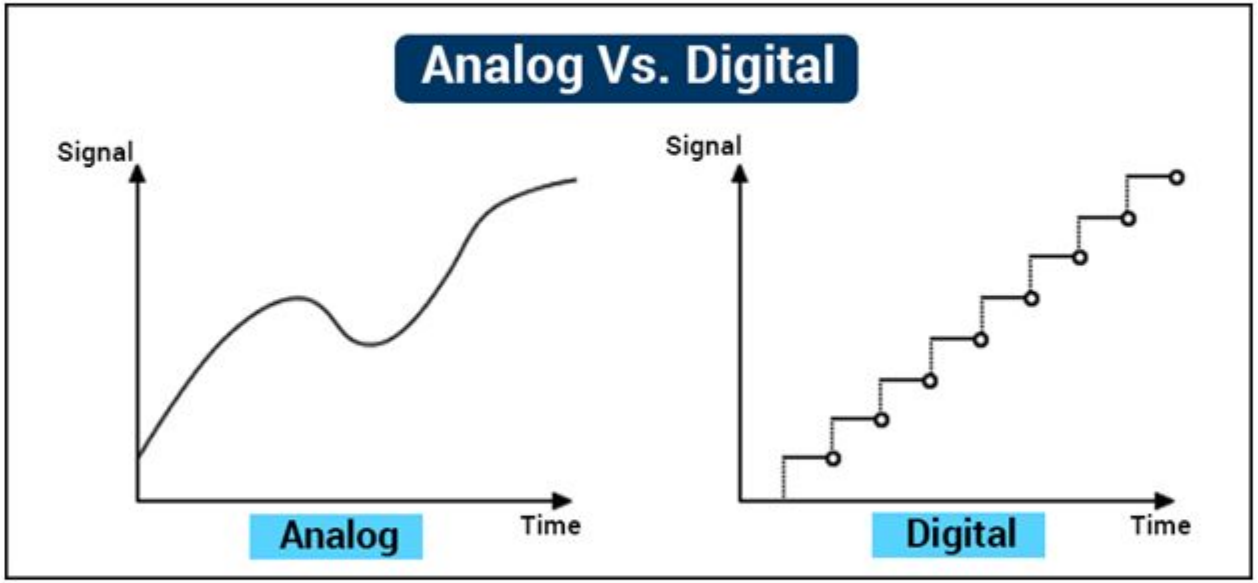
SWADESH KUMAR MAURYA

Introduction

- ❑ Digital system design is process of designing or developing systems which represent information using a binary system.
- ❑ It's easier to store, reproduce, transmit & manipulate digital data & cheaper/easier to design such systems.
- ❑ Microcontrollers, Microprocessors, Memory chips, FPGA are examples of digital IC design. Systems like for example a PC can be built using these components and that forms a digital system or platform.
- ❑ Digitization is implemented in a wide range of applications, including information (computers), telecommunications, control systems, etc. Digital circuits replace many analog systems.

Analog vs Digital Signals

- Analog and digital signals are the types of signals carrying information.
- The major difference between both signals is that the analog signals have continuous electrical signals, while digital signals have non-continuous electrical signals.
- The difference between analog and digital signal can be observed with the various examples of different types of waves*.



* https://byjus.com/p

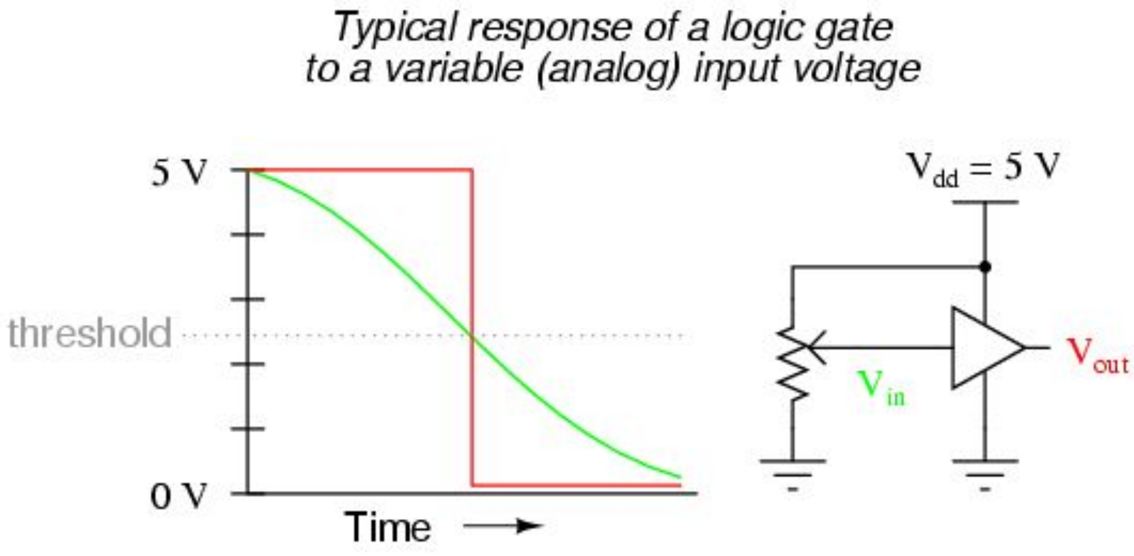
Difference Between Analog And Digital Signal

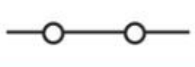
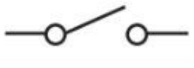
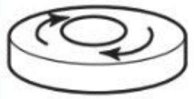
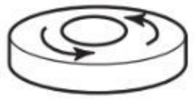
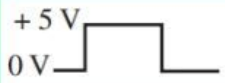
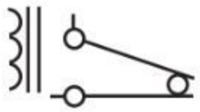
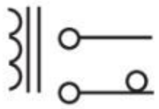
Analog Signals	**Digital Signals**
Continuous signals	Discrete signals
Represented by sine waves	Represented by square waves
Human voice, natural sound, analog electronic devices are a few examples	Computers, optical drives, and other electronic devices
Continuous range of values	Discontinuous values
Records sound waves as they are	Converts into a binary waveform.
Only used in analog devices.	Suited for digital electronics like computers, mobiles and more.

What is Binary?

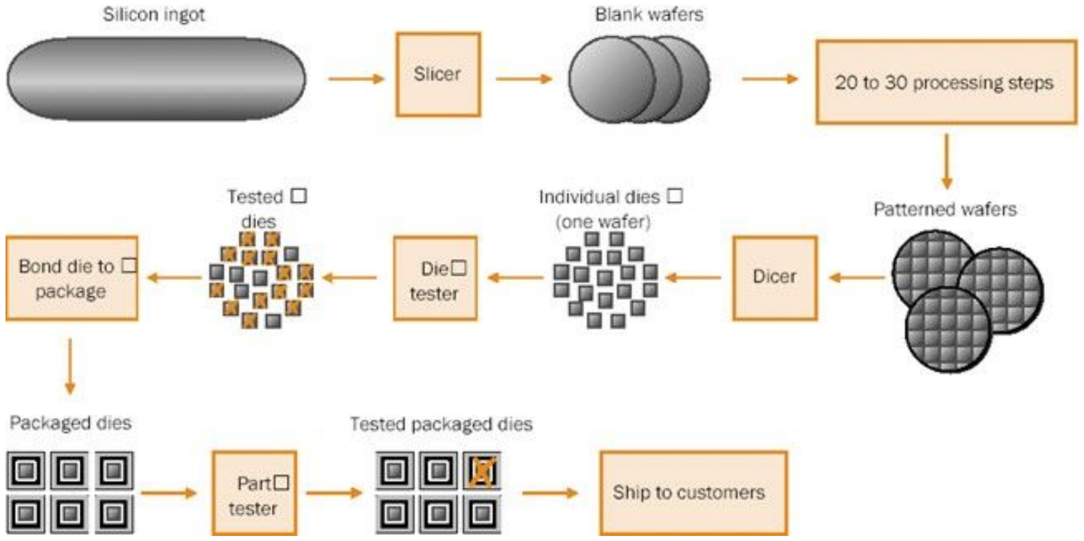
- ❑ Binary is a scheme of numbers that only has two possible values for each digit: 0 and 1.
- ❑ The term also describes any encoding/ decoding system in which there are only two possible states.
- ❑ In digital data storage, memory, communications, or processing the 0 and 1 values are sometimes called "low" and "high" or "On" and "Off".
- ❑ The digital world is represented in binary, but hexadecimal, which is compatible with binary and more easily understood by people, is commonly used. Like previously mentioned, binary uses only the numbers 0 and 1. However, in some cases L/H is used as a counterpart to 0/1 notation.

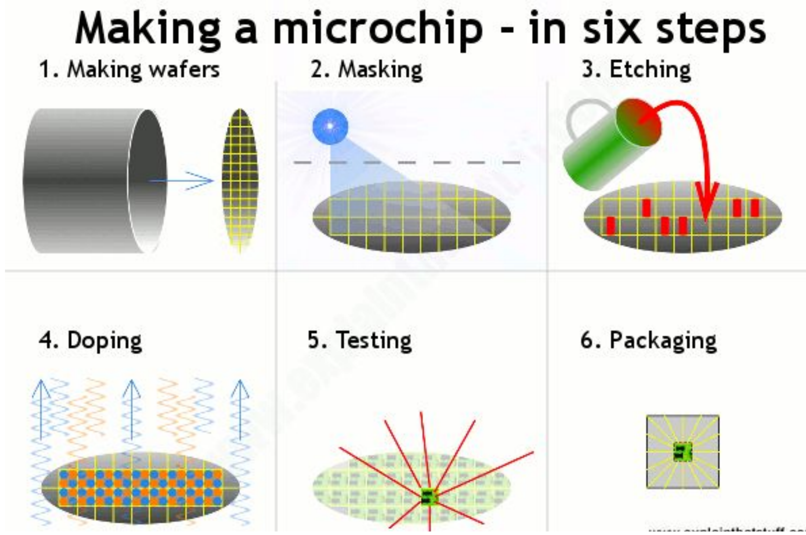
Binary State Representation



Component	On or 1 state	Off or 0 state
SPST switch		
Magnetic core		
Digital pulse		
Relay		

Digital Chip Fabrication



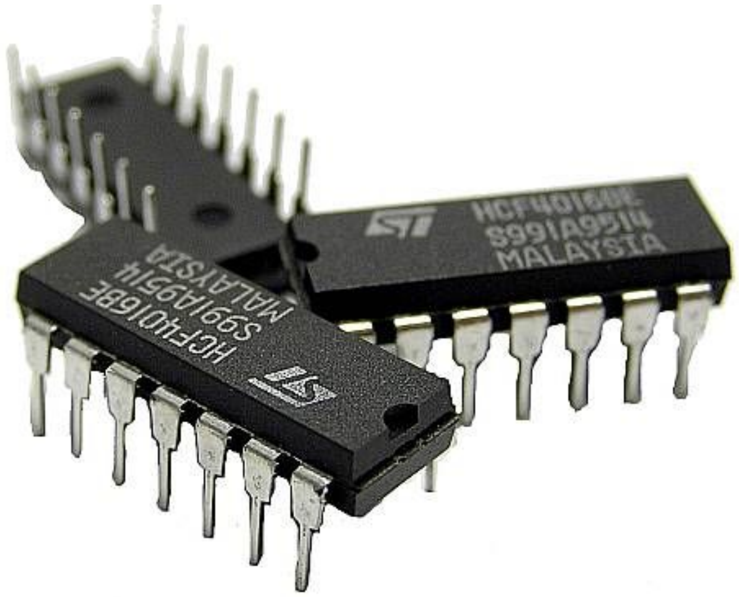


* https://slideplayer.com/slide/5920518/

- The brain of an integrated circuit is skillfully hidden beneath a protective package that you’re used to seeing on a circuit board.
- Package types are all standardized, making it easy to solder to circuit boards or connect to breadboards for prototyping.
- On each package, there’s a set of silver pins, and these allow the IC to connect to other parts of your circuit.

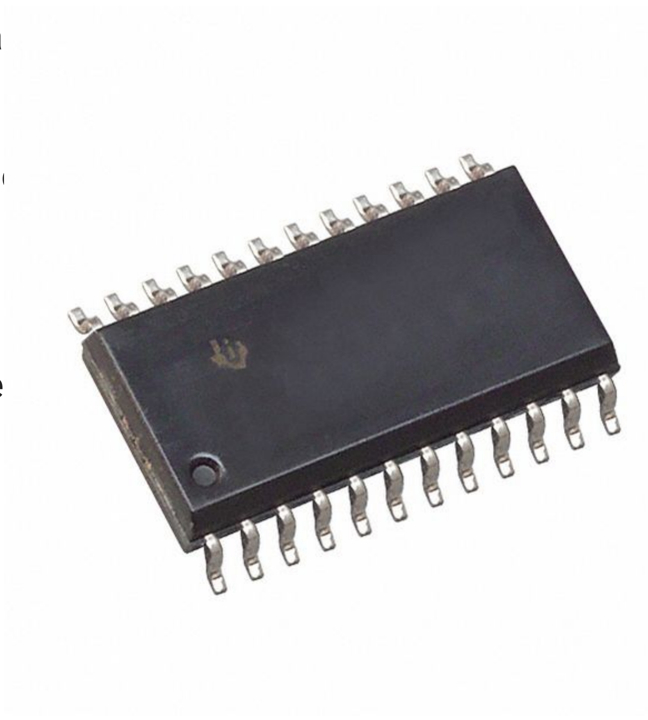
Dual-Inline Packages (DIP)

- ❏ This package type is part of the through-hole family, and you can easily recognize these chips by looking for their long, rectangular shapes with parallel rows of pins.
- ❏ This package type is ideal for use on breadboards and can include anywhere from 4 to 64 pins.
- ❏ *typical Dual-Inline Package type with two rows of pins and a long, rectangular shape.*



Small Outline Packages (SOP)

- SOP ICs are closely related to DIPs, except they're applied a surface-mount components instead of through-hole.
- These chips won't be used in your breadboarding experiments an will require some advanced machinery to apply precisely.
- SOP ICs come in several varieties, including Thin Small-Outlin Packages (TSOP) and Thin-Shrink Small-Outline Package (TSSOP).

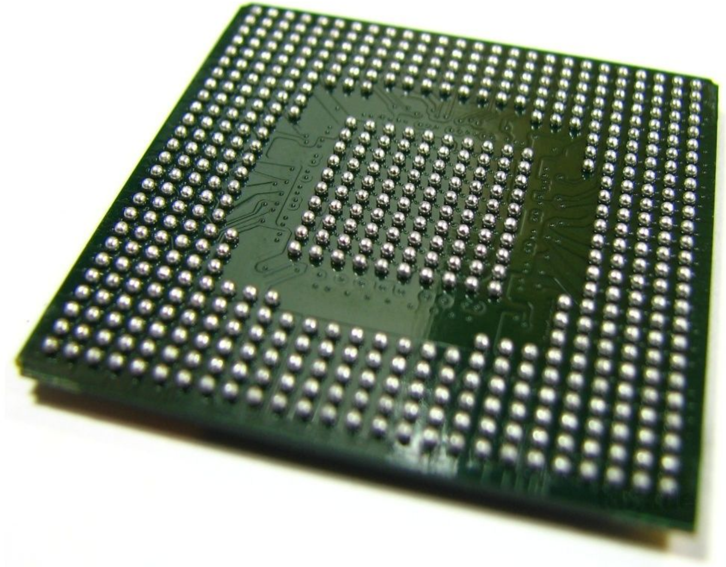


Quad Flat Packages (QFP)

- ❏ This package type is easily identified by pins jutting out from all four directions of the IC.
- ❏ These useful chips can have anywhere from 8 to 70 pins on each side, which can give them a whopping 300 pins to work with during a PCB layout process.
- ❏ You’ll find a ton of microprocessors using the QFP package type, including the popular ATmega328.

Ball Grid Arrays (BGA)

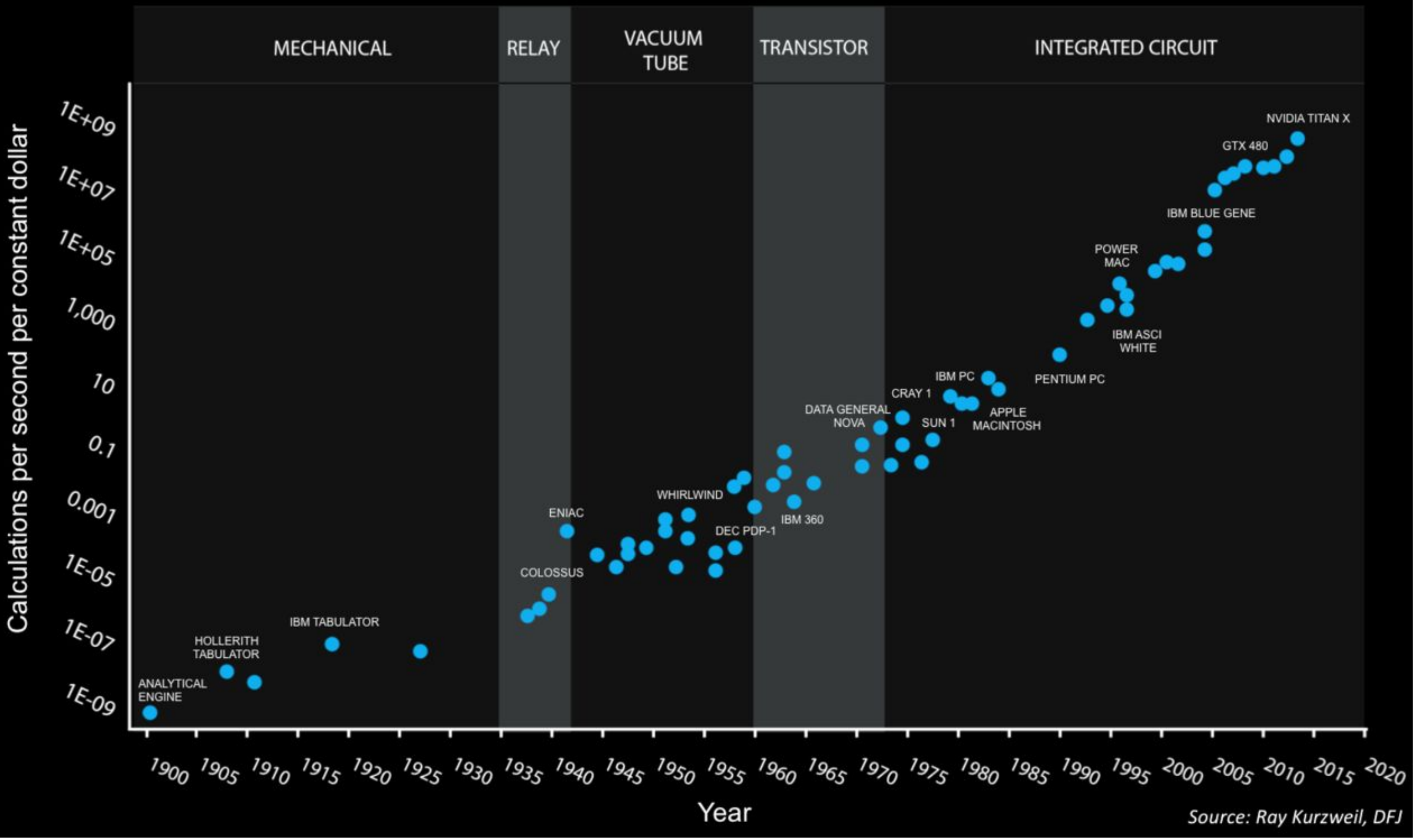
- ❑ The last package type, and also the most advanced is the Ball Grid Array.
- ❑ These complex package types include small balls solder on the bottom arranged in a pattern or grid.
- ❑ Routing all of the pins on a BGA can be quite difficu often taking hours to route the nets out of the tig spacing (called fanout routing).
- ❑ You’ll find the BGA package type used for only the mo advanced microprocessors, like those on the Raspberry Pi.



Moore’s Law and the Future of Integrated Circuits

- ❏ Since integrated circuits came into existence in the 1960s, engineers began putting dozens of components on a single chip in what was called small-scale integration, or SSI.
- ❏ Shortly after, hundreds of components were being placed in the same footprint, then thousands, and millions
- ❏ **Gordon Moore,** co-founder of Intel, sure did. Moore made the observation, and also the prediction that the number of components being placed on a chip was doubling roughly every one to two years, and would continue to do so.
- ❏ This is the famous **Moore’s Law**.

120 Years of Moore’s Law



https://www.autodesk.com/products/eagle/blog/integrated-circuit-moores-law/

Logic Gates

- ❑ Binary logic deals with binary variables and with operations that assume a logical meaning.
- ❑ It is used to describe, in algebraic or tabular form, the manipulation and processing of binary information.
- ❑ The *manipulation of binary information is done by logic circuits called gates* .
- ❑ Gates are blocks of hardware that produce signals of binary 1 or 0 when input logic requirements are satisfied.
- ❑ Each gate has a **distinct graphic symbol** and its operation can be described by means of an algebraic expression.
- ❑ The input-output relationship of the binary variables for each gate can be represented in tabular form by a **truth table**.

Boolean function

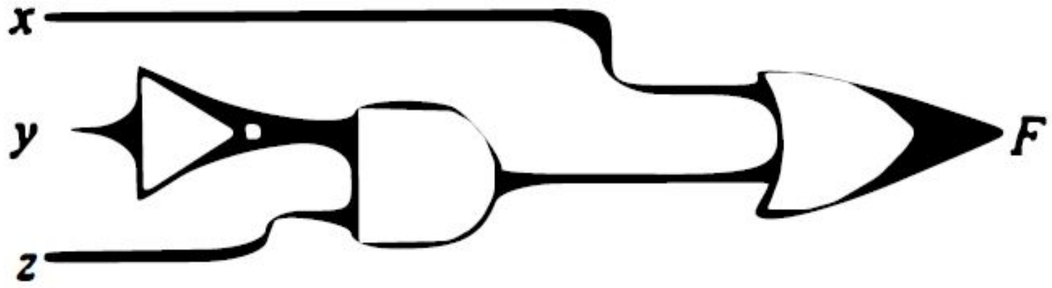
- ❑ Boolean algebra is an algebra that deals with binary variables and logic operations.
- ❑ The variables are designated by letters such as A, B, x, andy.
- ❑ The three basic logic operations are AND, OR, and complement.
- ❑ A Boolean function can be expressed algebraically with binary variables, the logic operation symbols, parentheses, and equal sign.

For a given value of the variables, the Boolean function can be either 1 or 0. Consider, for example, the Boolean function

$F = x + y'z$

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

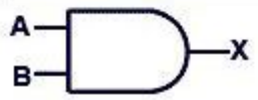
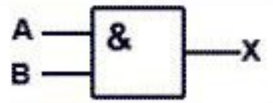
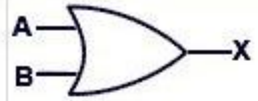
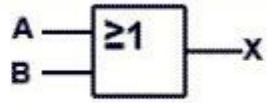
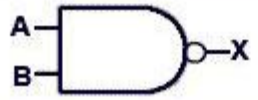
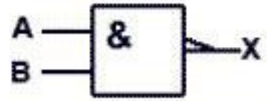
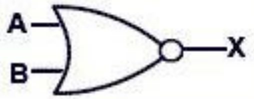
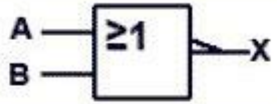
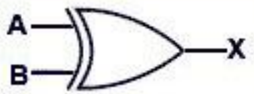
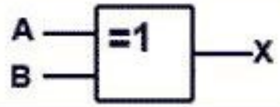
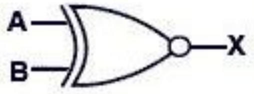
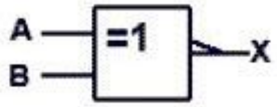
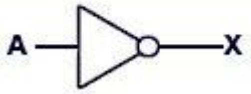
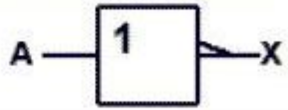
(a) Truth table



(b) Logic diagram

Truth Table and Logic Diagram of function F = x + y'z

GATES

IC No.	Gate	ANSI Symbol	IEC Symbol	Boolean
7408	**AND**			$X = A \cdot B$
7432	**OR**			$X = A + B$
7400	**NAND**			$X = \overline{A \cdot B}$
7402	**NOR**			$X = \overline{A + B}$
7486	**XOR**			$X = A \oplus B$
74266	**XNOR**			$X = \overline{A \oplus B}$
7404	**NOT**			$X = \overline{A}$

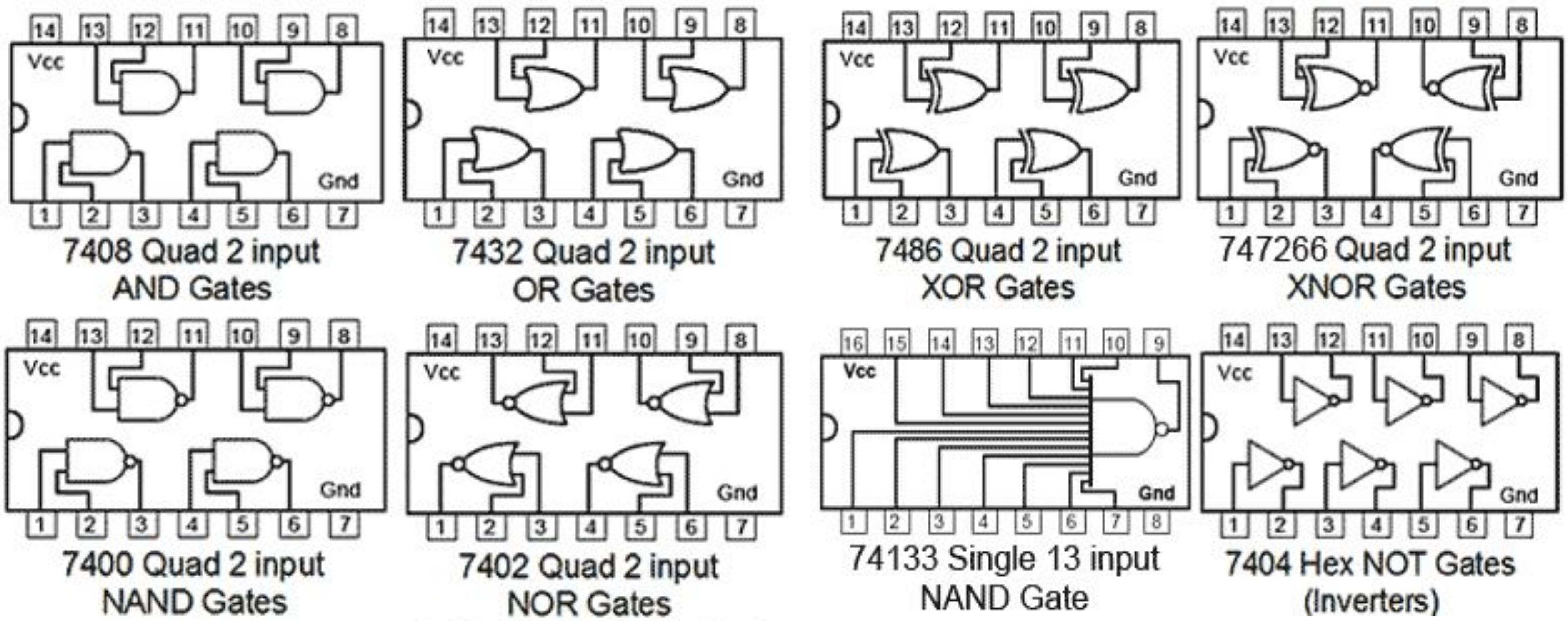
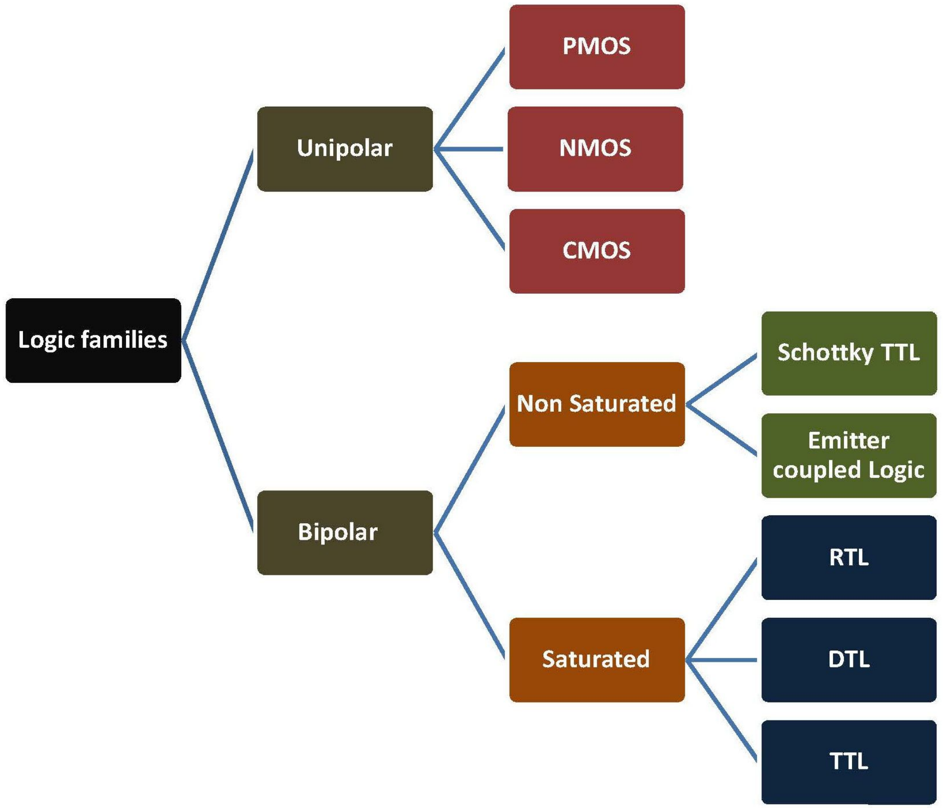

Vcc
Gnd
7408 Quad 2 input AND Gates
7432 Quad 2 input OR Gates
7486 Quad 2 input XOR Gates
747266 Quad 2 input XNOR Gates
7400 Quad 2 input NAND Gates
7402 Quad 2 input NOR Gates
74133 Single 13 input NAND Gate
7404 Hex NOT Gates (Inverters)

Fig. Logic Gates From the 74 series TTL IC Family

Logic Families

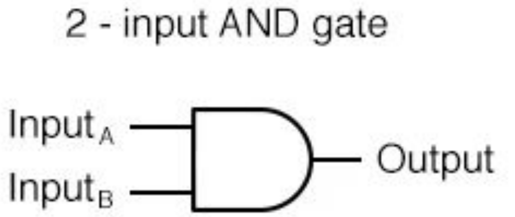
- ❑ All digital systems, computers and microprocessors are assembled from simple circuits called logic circuits.
- ❑ The basic building blocks of logic circuits are logic gates. And logic gates themselves are simple electronic circuits comprising of diodes, transistors and resistors.

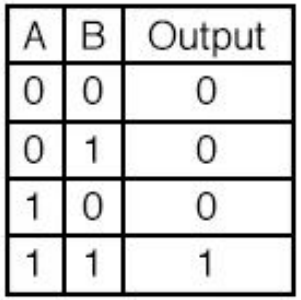


https://easyelectronics.co.in/comparison-of-logic-families/

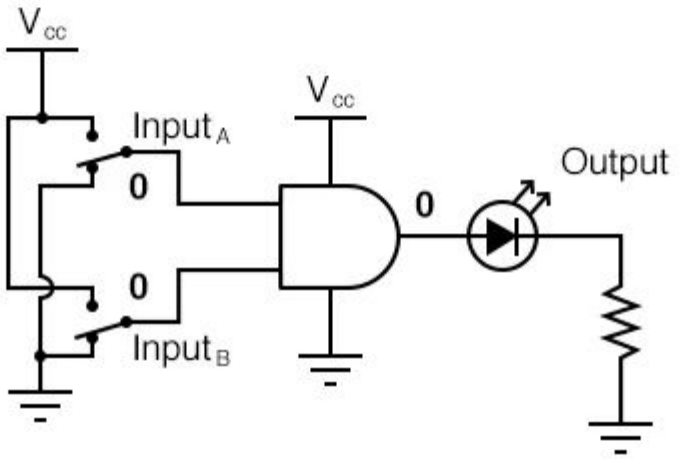
The AND Gate

One of the easiest multiple-input gates to understand is the AND gate, so-called because the output of this gate will be “high” (1) if and only if *all* inputs (first input *and* the second input *and* . . .) are “high” (1). If any input(s) is “low” (0), the output is guaranteed to be in a “low” state as well.





A	B	Output
0	0	0
0	1	0
1	0	0
1	1	1



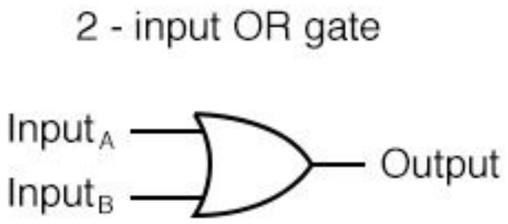
Input_A = **0**

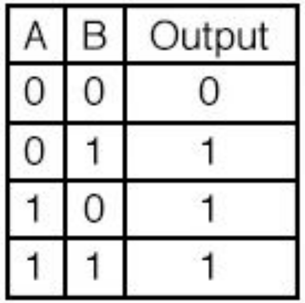
Input_B = **0**

Output = **0** *(no light)*

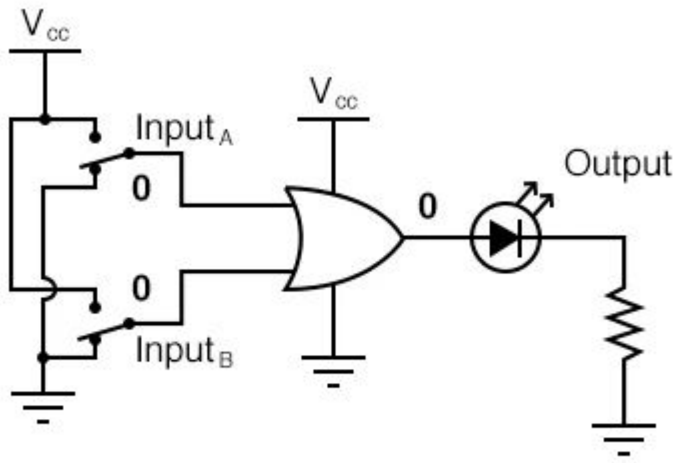
The OR Gate

Our next gate to investigate is the OR gate, so-called because the output of this gate will be “high” (1) if any of the inputs (first input or the second input or . . .) are “high” (1). The output of an OR gate goes “low” (0) if and only if all inputs are “low” (0).





A	B	Output
0	0	0
0	1	1
1	0	1
1	1	1



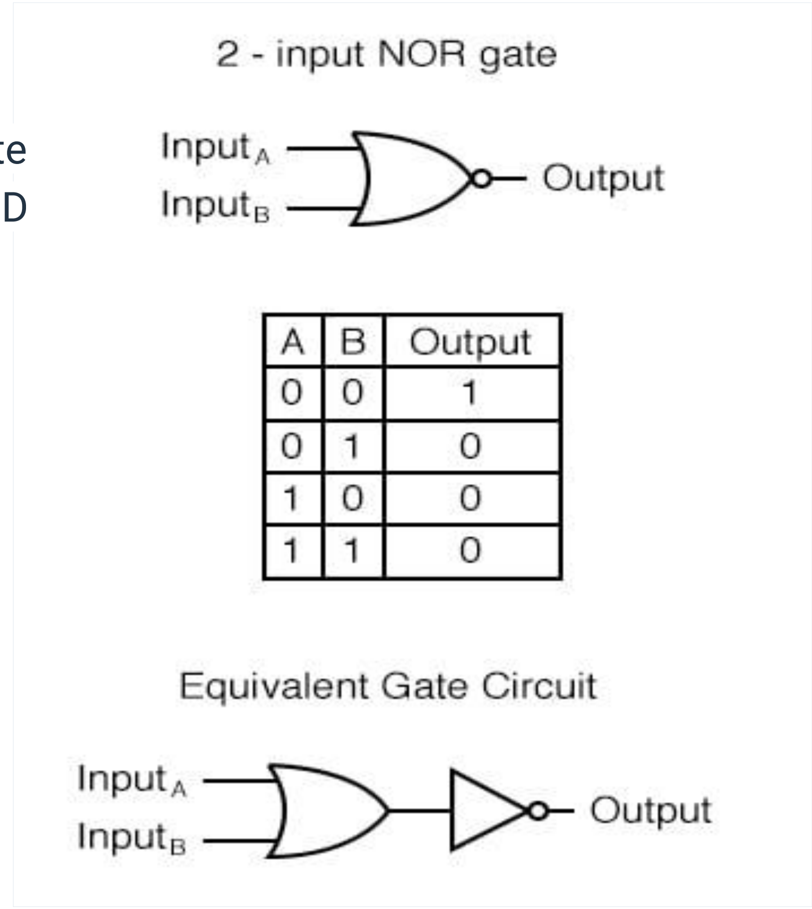
Input$_A$ = **0**

Input$_B$ = **0**

Output = **0** *(no light)*

The NOR Gate

As you might have suspected, the NOR gate is an OR gate with its output inverted, just like a NAND gate is an AND gate with an inverted output.



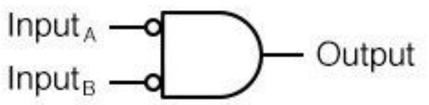
A	B	Output
0	0	1
0	1	0
1	0	0
1	1	0

The Negative-AND Gate *NOR*

A Negative-AND gate functions the same as an AND gate with all its inputs inverted (connected through NOT gates). In keeping with standard gate symbol convention, these inverted inputs are signified by bubbles.

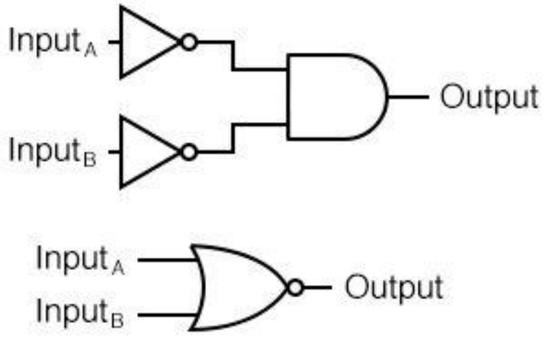
Contrary to most peoples' first instinct, the logical behavior of a Negative-AND gate is not the same as a NAND gate. Its truth table, actually, is identical to a **NOR gate:**

2 - input Negative-AND gate



A	B	Output
0	0	1
0	1	0
1	0	0
1	1	0

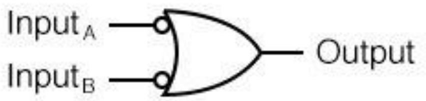
Equivalent Gate Circuits

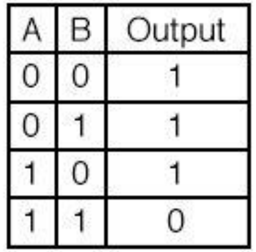


The Negative-OR Gate

same pattern, a Negative-OR gate functions the same as an OR gate with all its inputs inverted. In keeping with standard gate symbol convention, these inverted inputs are signified by bubbles. The behavior and truth table of a Negative-OR gate is the same as for a **NAND gate**:

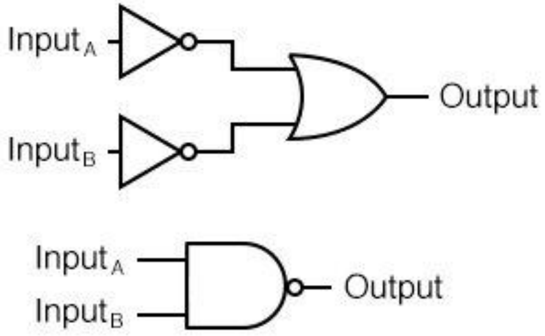
2 - input Negative-OR gate





A	B	Output
0	0	1
0	1	1
1	0	1
1	1	0

Equivalent Gate Circuits



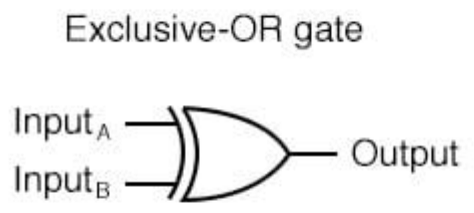
The Exclusive-OR Gate

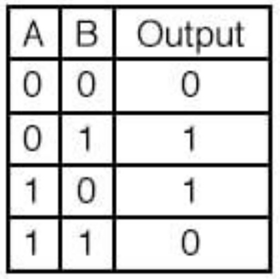
The last six gate types are all fairly direct variations on three basic functions: AND, OR, and NOT. The Exclusive-OR gate, however, is something quite different.

Exclusive-OR gates output a “high” (1) logic level if the inputs are at different logic levels, either 0 and 1 or 1 and 0.

Conversely, they output a “low” (0) logic level if the inputs are at the same logic levels.

The Exclusive-OR (sometimes called XOR) gate has both a symbol and a truth table pattern that is unique:





A	B	Output
0	0	0
0	1	1
1	0	1
1	1	0

- Rule for an AND gate: output is “high” only if first input *and* second input are both “high.”
- Rule for an OR gate: output is “high” if input A *or* input B are “high.”
- Rule for a NAND gate: output is *not* “high” if both the first input and the second input are “high.”
- Rule for a NOR gate: output is *not* “high” if either the first input or the second input are “high.”
- A Negative-AND gate behaves like a NOR gate.
- A Negative-OR gate behaves like a NAND gate.
- Rule for an Exclusive-OR gate: output is “high” if the input logic levels are *different*.
- Rule for an Exclusive-NOR gate: output is “high” if the input logic levels are the *same*.

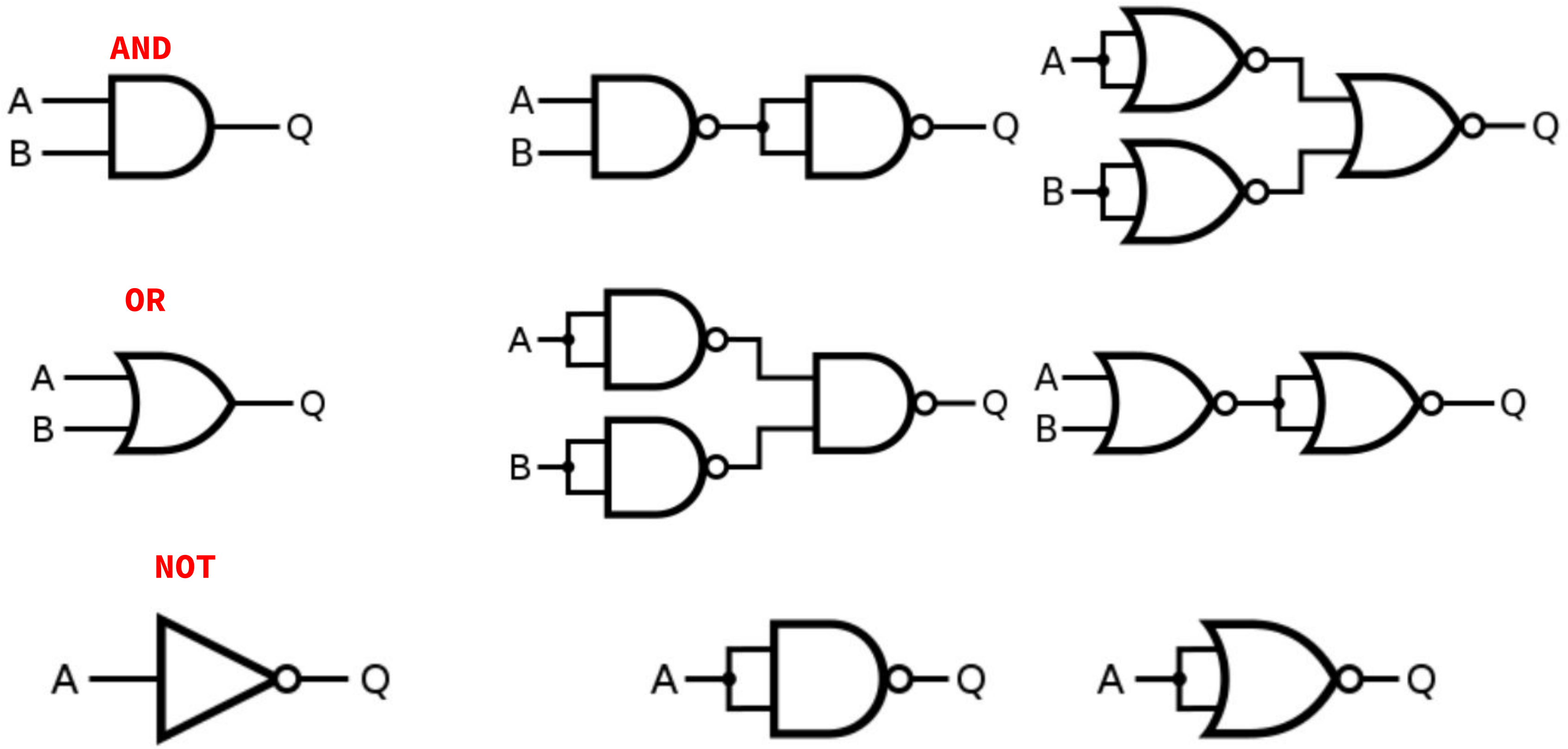
TABLE 1-1 Basic Identities of Boolean Algebra

(1) $x + 0 = x$	(2) $x \cdot 0 = 0$
(3) $x + 1 = 1$	(4) $x \cdot 1 = x$
(5) $x + x = x$	(6) $x \cdot x = x$
(7) $x + x' = 1$	(8) $x \cdot x' = 0$
(9) $x + y = y + x$	(10) $xy = yx$
(11) $x + (y + z) = (x + y) + z$	(12) $x(yz) = (xy)z$
(13) $x(y + z) = xy + xz$	(14) $x + yx = (x + y)(x + z)$
(15) $(x + y)' = x'y'$	(16) $(xy)' = x' + y'$
(17) $(x')' = x$	

Basic Identities of Boolean Algebra

- ❑ The **first eight identities 1-8** show the basic relationship between a single variable and itself, or in conjunction with the binary constants 1 and 0.
- ❑ The next five identities **(9 through 13)** are similar to ordinary algebra. **Identity 14** does not apply in ordinary algebra but is very useful in manipulating Boolean expressions.
- ❑ Identities **15 and 16 are called DeMorgan's theorems**.
- ❑ The last identity **17** states that if a variable is complemented twice, one obtains the original value of the variable.
- ❑ **DeMorgan's** first theorem states that two (or more) variables **NOR´ed** together is the same as the two variables inverted (Complement) and AND´ed, while the second theorem states that two (or more) variables **NAND´ed** together is the same as the two terms inverted (Complement) and OR´ed. That is replace all the OR operators with AND operators, or all the AND operators with an OR operators.

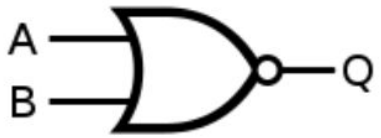
Desired gate | NAND construction | NOR construction

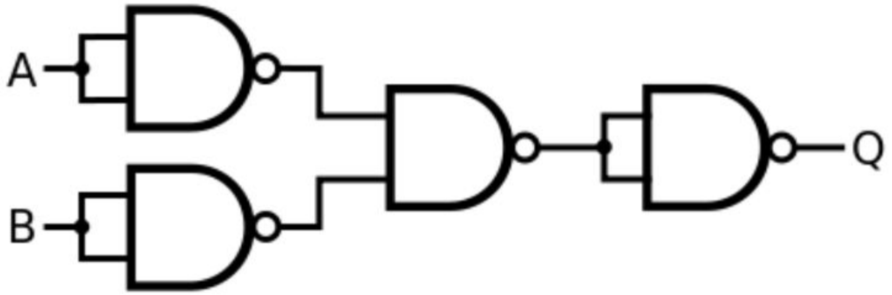


Desired gate

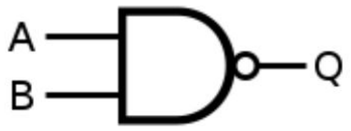
NAND Construction

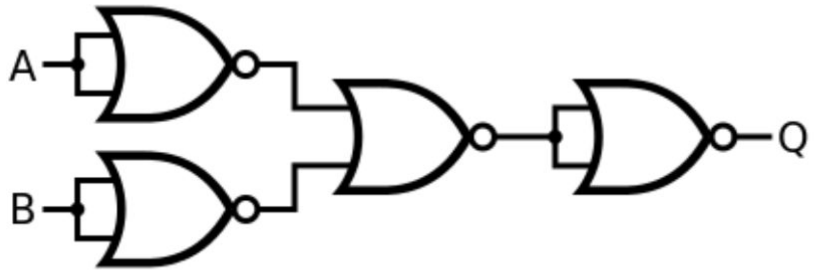
NOR





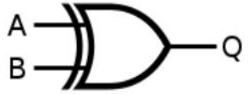
NAND

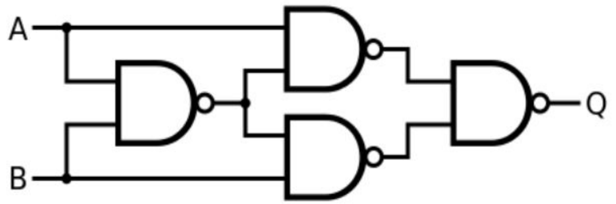


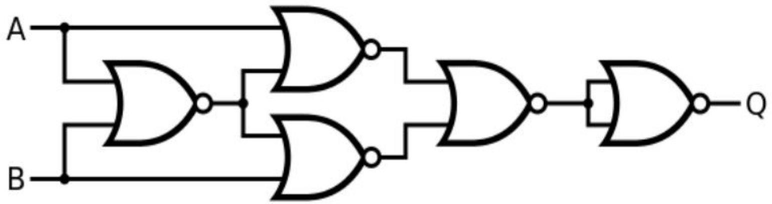


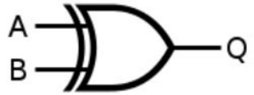
XOR

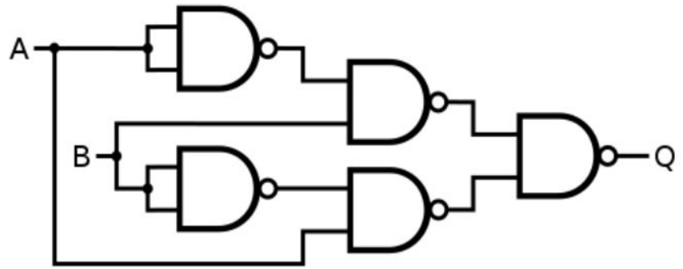
Desired gate | **NAND construction** | **NOR construction**

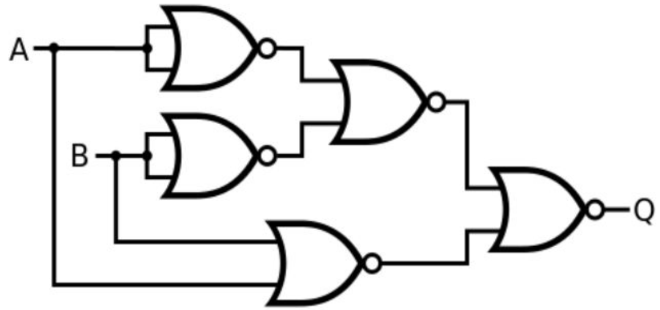










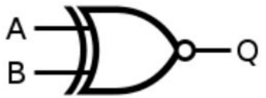


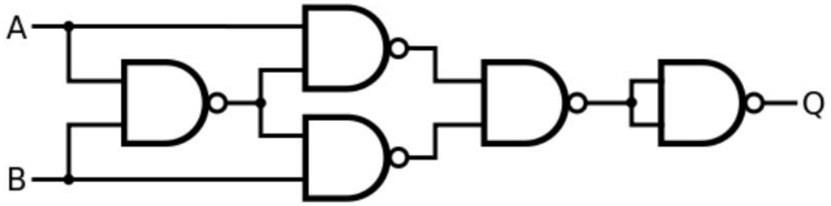
$$(A+\overline{B})\cdot(\overline{A}+B)\equiv(A\cdot B)+(\overline{A}\cdot\overline{B})$$

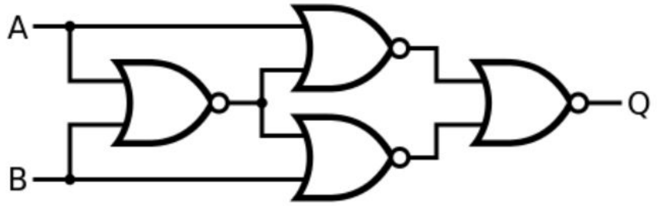
Desired gate

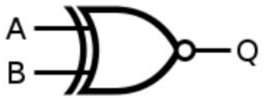
NAND construction

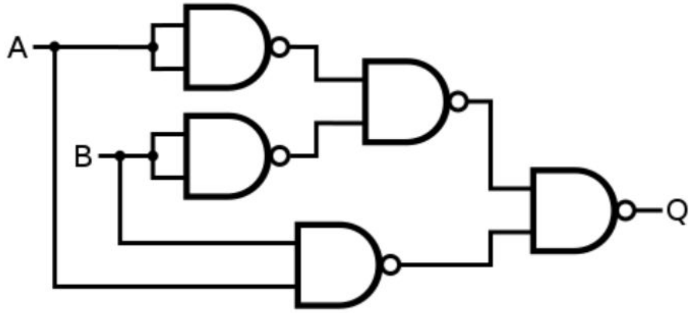
NOR construction

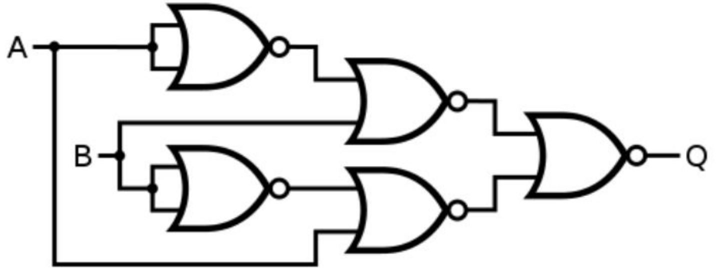






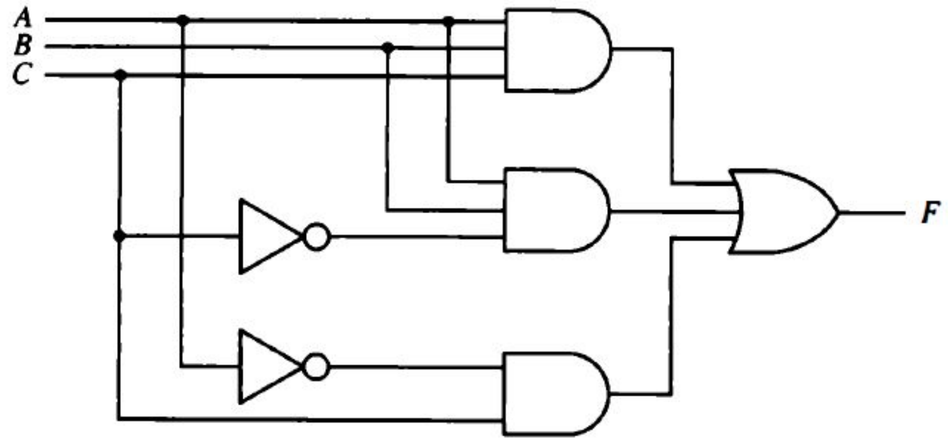




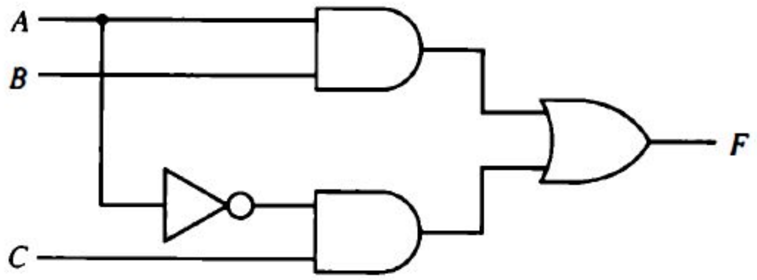


X-NOR

Boolean Function Minimization



(a) $F = ABC + ABC' + A'C$



(B) $F = AB + A'C$

Figure 1-6 Two logic diagrams for the same Boolean function.

Map Simplification

- ❑ The complexity of the **logic diagram** that implements a Boolean function is related directly to the complexity of the **algebraic expression** from which the function is implemented.
- ❑ The truth table representation of a function is unique, but the function can appear in many different forms when expressed algebraically.
- ❑ Simplification procedure with boolean algebra is sometimes difficult because it lacks specific rules for predicting each succeeding step in the manipulative process. **The map method provides a simple, straightforward procedure for simplifying Boolean expressions.**
- ❑ This method is pictorial arrangement of the truth table which allows an easy interpretation for choosing the **minimum number of terms needed to express the function algebraically**. The map method is also known as the **Karnaugh map or K-map.**

Canonical and Standard Form

Canonical Form

- In Boolean algebra,Boolean function can be expressed as
  Canonical Disjunctive Normal Form (**A.B.C)** known as **minterm**
  Canonical Conjunctive Normal Form (**A+B+C)** known as **maxterm** .
  In Minterm, we look for the functions where the output results in “1” while
  In Maxterm we look for function where the output results in “0”.

- **Sum of minterm** also known as Sum of products (SOP) .
- **Product of Maxterm** also known as Product of sum (POS).

Boolean functions expressed as a ***sum of minterms or product of maxterms*** are said to be in **Canonical form.**

Standard Form

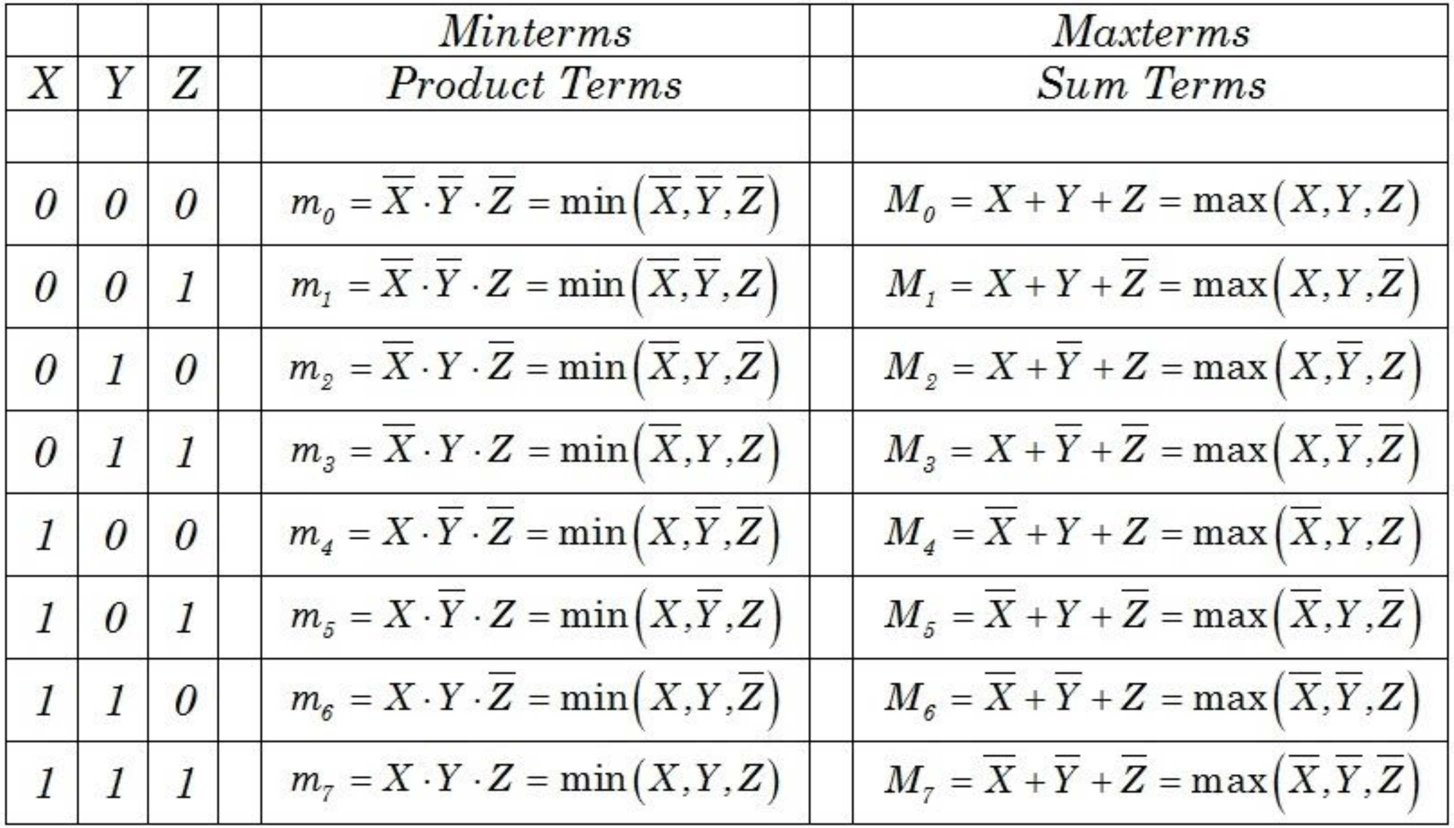
- ❑ A Boolean variable can be expressed in either true form or complemented form.
- ❑ In standard form **Boolean function will contain all the variables in either true form or complemented** form while in canonical number of variables depends on the output of SOP or POS.

- ❑ A Boolean function can be expressed algebraically from a given truth table by forming a :

  Minterm for each combination of the variables that produces a 1 in the function and then taking the OR of all those terms.

  Maxterm for each combination of the variables that produces a 0 in the function and then taking the AND of all those terms.

Truth table representing Minterm and Maxterm



				Minterms		Maxterms
X	Y	Z		Product Terms		Sum Terms
0	0	0		$m_0 = \overline{X} \cdot \overline{Y} \cdot \overline{Z} = \min(\overline{X}, \overline{Y}, \overline{Z})$		$M_0 = X + Y + Z = \max(X, Y, Z)$
0	0	1		$m_1 = \overline{X} \cdot \overline{Y} \cdot Z = \min(\overline{X}, \overline{Y}, Z)$		$M_1 = X + Y + \overline{Z} = \max(X, Y, \overline{Z})$
0	1	0		$m_2 = \overline{X} \cdot Y \cdot \overline{Z} = \min(\overline{X}, Y, \overline{Z})$		$M_2 = X + \overline{Y} + Z = \max(X, \overline{Y}, Z)$
0	1	1		$m_3 = \overline{X} \cdot Y \cdot Z = \min(\overline{X}, Y, Z)$		$M_3 = X + \overline{Y} + \overline{Z} = \max(X, \overline{Y}, \overline{Z})$
1	0	0		$m_4 = X \cdot \overline{Y} \cdot \overline{Z} = \min(X, \overline{Y}, \overline{Z})$		$M_4 = \overline{X} + Y + Z = \max(\overline{X}, Y, Z)$
1	0	1		$m_5 = X \cdot \overline{Y} \cdot Z = \min(X, \overline{Y}, Z)$		$M_5 = \overline{X} + Y + \overline{Z} = \max(\overline{X}, Y, \overline{Z})$
1	1	0		$m_6 = X \cdot Y \cdot \overline{Z} = \min(X, Y, \overline{Z})$		$M_6 = \overline{X} + \overline{Y} + Z = \max(\overline{X}, \overline{Y}, Z)$
1	1	1		$m_7 = X \cdot Y \cdot Z = \min(X, Y, Z)$		$M_7 = \overline{X} + \overline{Y} + \overline{Z} = \max(\overline{X}, \overline{Y}, \overline{Z})$

Minterm and Maxterm

- ❏ Each combination of the variables in a truth table is called a **minterm**.
- ❏ For example, the truth table of Fig. 1-3 contains eight minterms.
- ❏ When expressed in a truth table a function of n variables will have 2" minterms, equivalent to the 2" binary numbers obtained from n bits.
- ❏ A Boolean function is equal to 1 for some minterms and to 0 for others.
- ❏ The information contained in a truth table may be expressed in compact form by listing the decimal equivalent of those minterms that produce a 1 for the function.
- ❏ For example, the truth table of Fig. 1-3 can be expressed as follows

$$F(x, y, z) = \sum(1, 4, 5, 6, 7)$$

The letters in parentheses list the binary variables in the order that they appear in the truth table. The symbol ∑ stands for the sum of the min terms that follow in parentheses.

Minterm Representation

Express the **Boolean function** F = A + B’C as standard sum of minterms.

Solution – In the sub exp A B, C variable are missing

A = A(B + B’) = AB + AB’

This function is still missing one variable, so

A = AB(C + C’) + AB'(C + C’) = ABC + ABC’+ AB’C + AB’C’

The second term B’C is missing one variable; hence,

B’C = B’C(A + A’) = AB’C + A’B’C

Combining all terms, we have

F = A + B’C = ABC + ABC’ + *AB’C* + AB’C’ + *AB’C* + A’B’C

we finally obtain

F = A’B’C + AB’C’ + AB’C + ABC’ + ABC

= m1 + m4 + m5 + m6 + m7

SOP is represented as Sigma(1, 4, 5, 6, 7)

Minterm Representation

Example – Express the Boolean function F = xy + x’z as a product of maxterms

- **Solution –**

  F = xy + x’z = (xy + x’)(xy + z) = (x + x’)(y + x’)(x + z)(y + z)

  (x’ + y)(x + z)(y + z)

  x’ + y = x’ + y + ***zz’*** *= (x’+ y + z)(x’ + y + z’)*

  x + z = x + z + ***yy’*** *= (x + y + z)(x + y’ + z)*

  y + z = y + z + ***xx’*** *= (x + y + z)(x’ + y + z)*

F = (x + y + z)(x + y’ + z)(x’ + y + z)(x’ + y + z’)

*= M0*M2*M4*M5*

POS is represented as **Π(0, 2, 4, 5)**

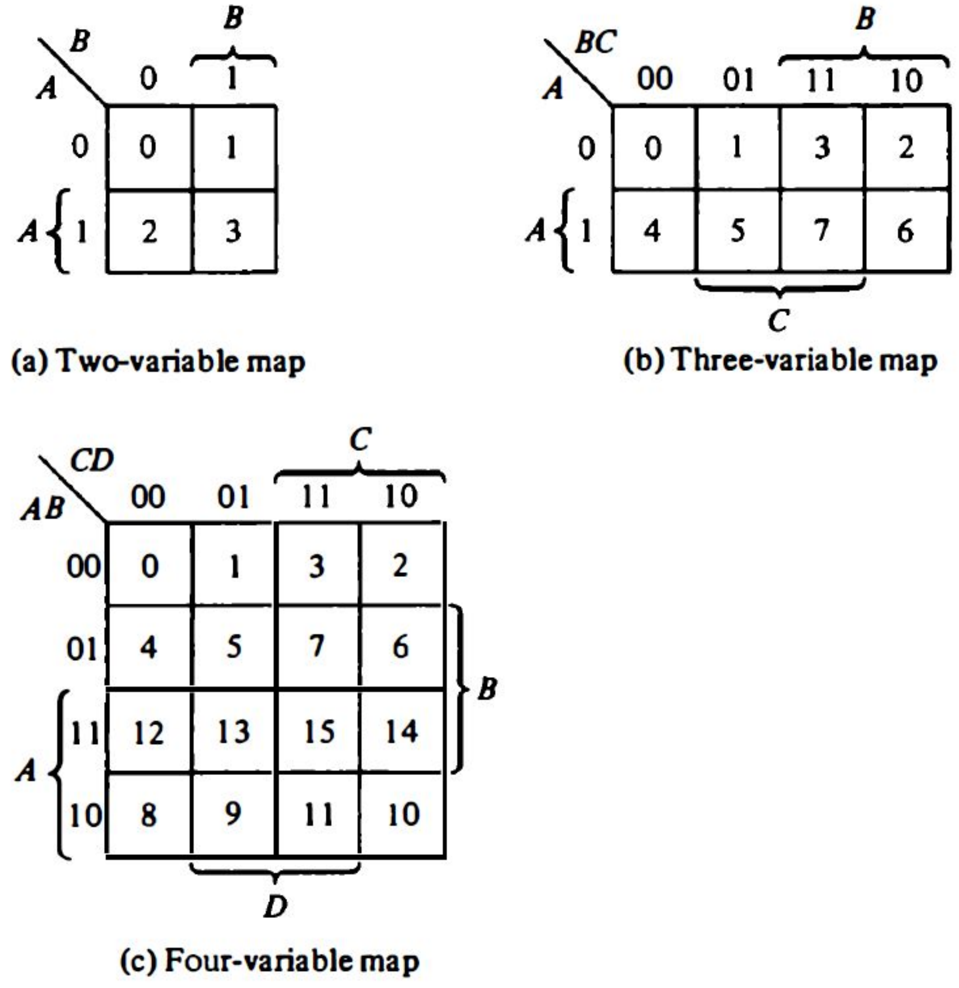
Convert Boolean expression in **standard form**

$F=y'+xz'+xyz$

Solution

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

$F = xy'z+ xy'z'+x'y'z+x'y'z'+ xyz'+xy'z'+xyz$



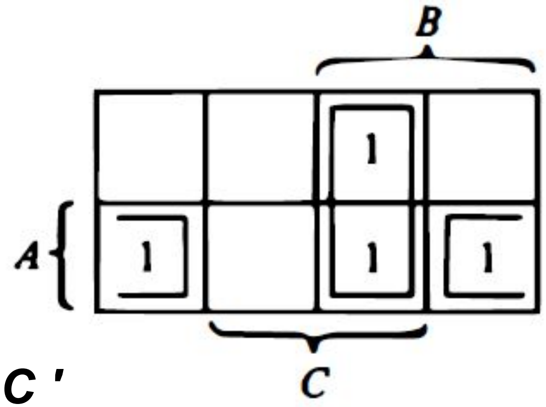
(a) Two-variable map

(b) Three-variable map

(c) Four-variable map

Figure 1-7 Maps for two-, three-, and four-variable functions.

Figure 1-8 Map for F(A, B, C) = Σ(3,4,6,7).



F = BC + AC '

F = C+AB '

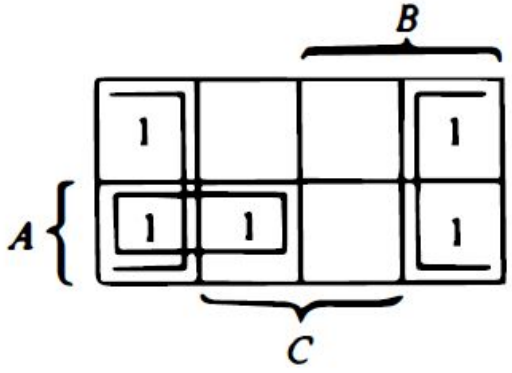
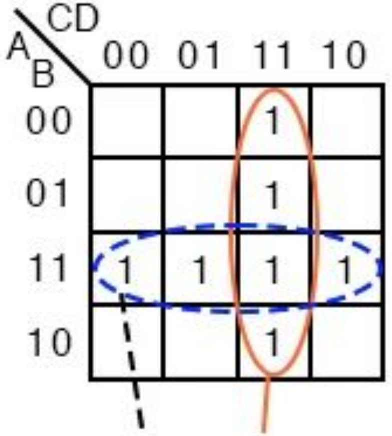


Figure 1-9 Map for F(A, B, C) = Σ(0,2,4,5,6).

Q. $F(A, B, C, D) = \sum(0, 1, 2, 6, 8, 9, 10)$

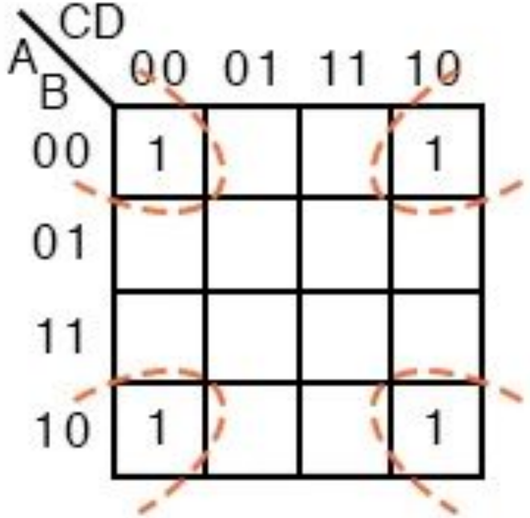
$F = B'D' + B'C' + A'CD'$

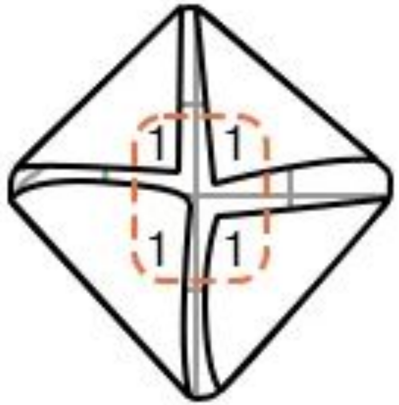
$$\text{Out} = \overline{A}\,\overline{B}CD + \overline{A}BCD + ABCD + A\overline{B}CD + AB\overline{C}\,\overline{D} + AB\overline{C}D + ABC\overline{D}$$



$$\text{Out} = AB + CD$$

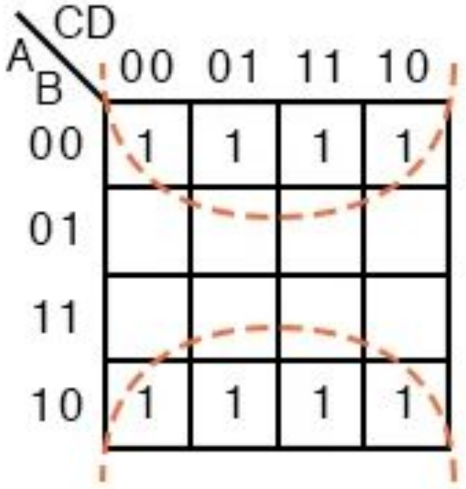
$$\text{Out} = \overline{A}\,\overline{B}\,\overline{C}\,\overline{D} + \overline{A}\,\overline{B}C\overline{D} + A\overline{B}\,\overline{C}\,\overline{D} + A\overline{B}C\overline{D}$$





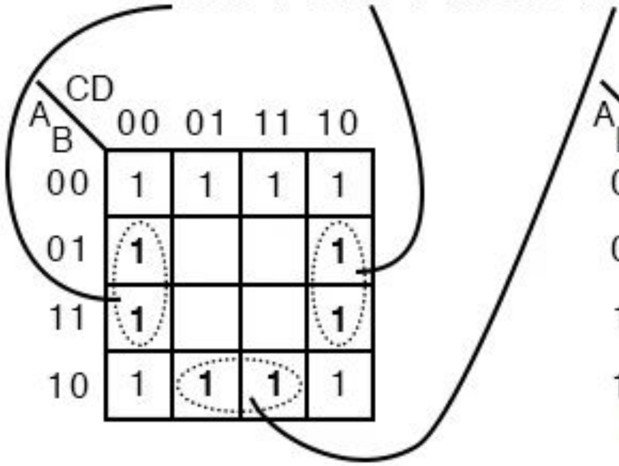
$$\text{Out} = \overline{B}\,\overline{D}$$

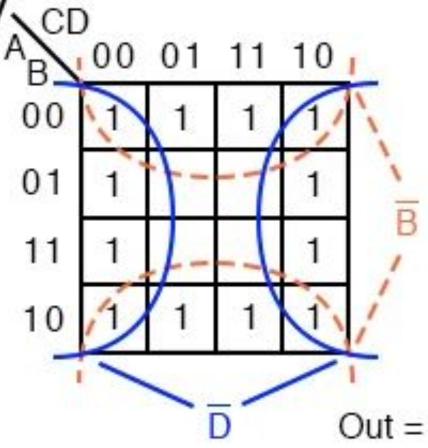
$$\text{Out} = \overline{A}\,\overline{B}\,\overline{C}\,\overline{D} + \overline{A}\,\overline{B}\,\overline{C}D + \overline{A}\,\overline{B}CD + \overline{A}\,\overline{B}C\overline{D}$$
$$+ A\overline{B}\,\overline{C}\,\overline{D} + A\overline{B}\,\overline{C}D + A\overline{B}CD + A\overline{B}C\overline{D}$$



Out = $\overline{B}$

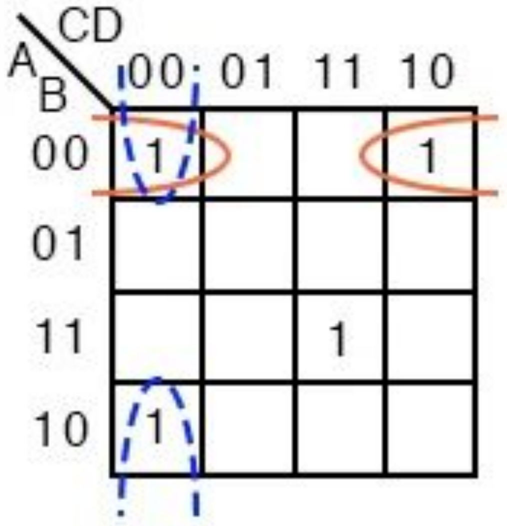
$$\text{Out} = \overline{A}\,\overline{B}\,\overline{C}\,\overline{D} + \overline{A}\,\overline{B}\,\overline{C}D + \overline{A}\,\overline{B}CD + \overline{A}\,\overline{B}C\overline{D}$$
$$+ B\overline{C}\,\overline{D} + BC\overline{D} + A\overline{B}\,\overline{C}\,\overline{D} + A\overline{B}D + A\overline{B}C\overline{D}$$





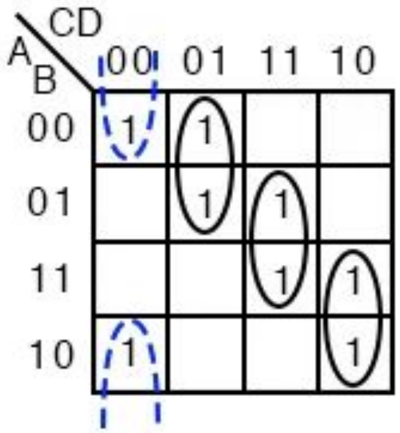
Out = $\overline{B} + \overline{D}$

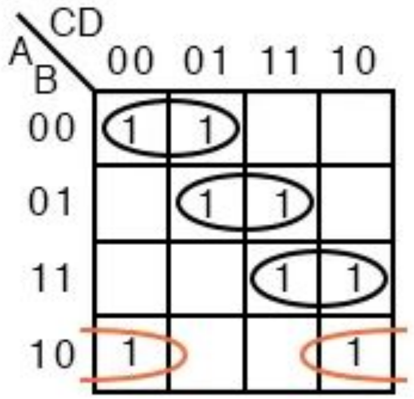
$$Out = \overline{A}\,\overline{B}\,\overline{C}\,\overline{D} + \overline{A}\,\overline{B}\,C\,\overline{D} + A\,\overline{B}\,\overline{C}\,\overline{D} + A\,B\,C\,D$$



$$Out = \overline{B}\,\overline{C}\,\overline{D} + \overline{A}\,\overline{B}\,\overline{D} + A\,B\,C\,D$$

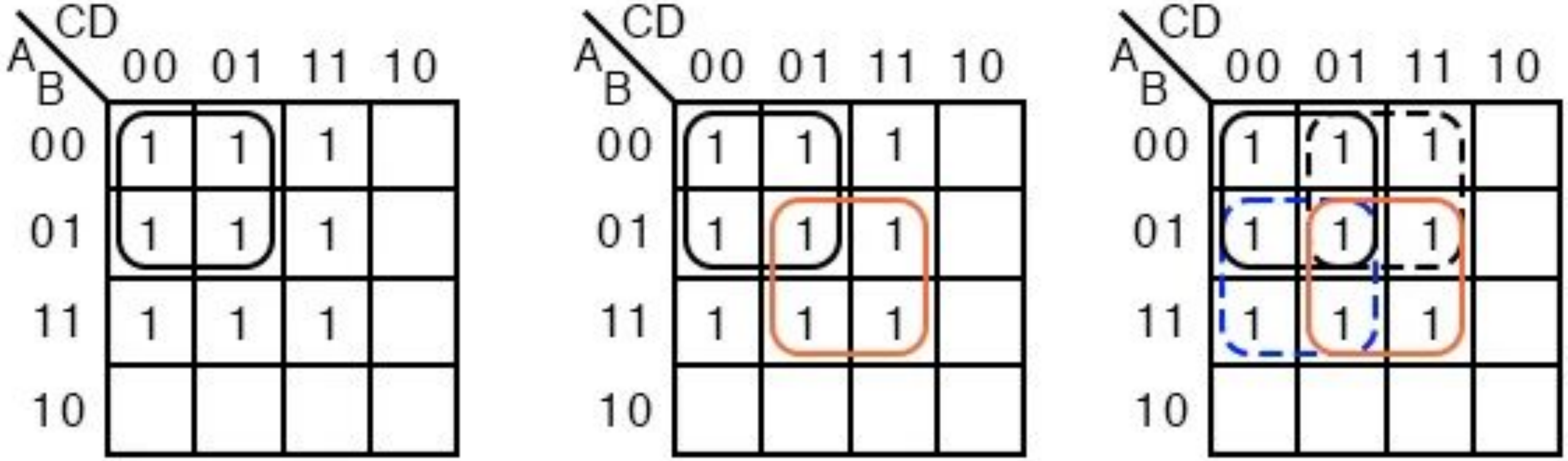
$$Out = \overline{A}\,\overline{B}\,\overline{C}\,\overline{D} + \overline{A}\,\overline{B}\,\overline{C}\,D + \overline{A}\,B\,\overline{C}\,D + \overline{A}\,B\,C\,D$$
$$+ A\,B\,C\,D + A\,B\,C\,\overline{D} + A\,\overline{B}\,\overline{C}\,\overline{D} + A\,\overline{B}\,C\,\overline{D}$$





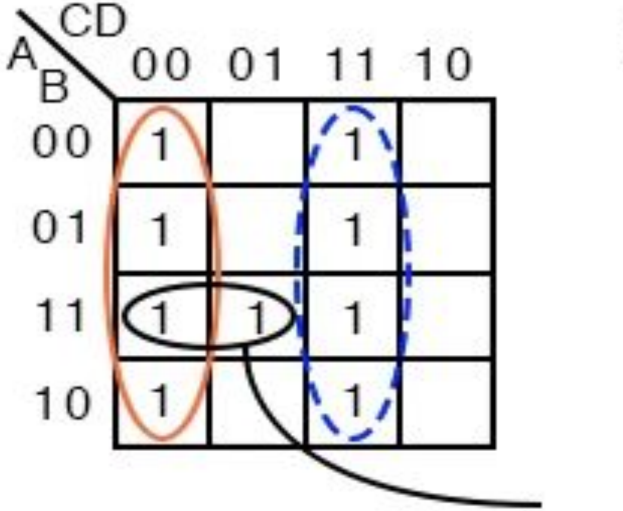
$$Out = \overline{B}\,\overline{C}\,\overline{D} + \overline{A}\,\overline{C}\,D + B\,C\,D + A\,C\,\overline{D}$$
$$Out = \overline{A}\,\overline{B}\,\overline{C} + \overline{A}\,B\,D + A\,B\,C + A\,\overline{B}\,\overline{D}$$

$$\begin{aligned}\text{Out} = {} & \overline{A}\,\overline{B}\,\overline{C}\,\overline{D} + \overline{A}\,\overline{B}\,\overline{C}D + \overline{A}\,\overline{B}CD \\ & + \overline{A}B\overline{C}\,\overline{D} + \overline{A}B\overline{C}D + \overline{A}BCD \\ & + AB\overline{C}\,\overline{D} + AB\overline{C}D + ABCD\end{aligned}$$

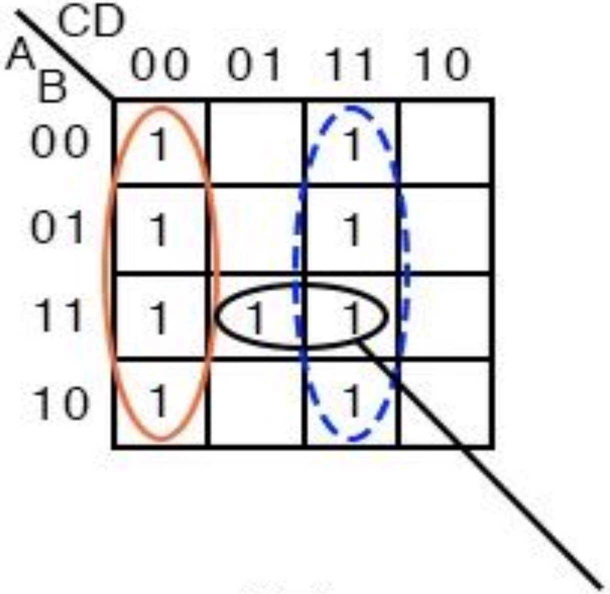


$$\text{Out} = \overline{A}\,\overline{C} + \overline{A}D + B\overline{C} + BD$$

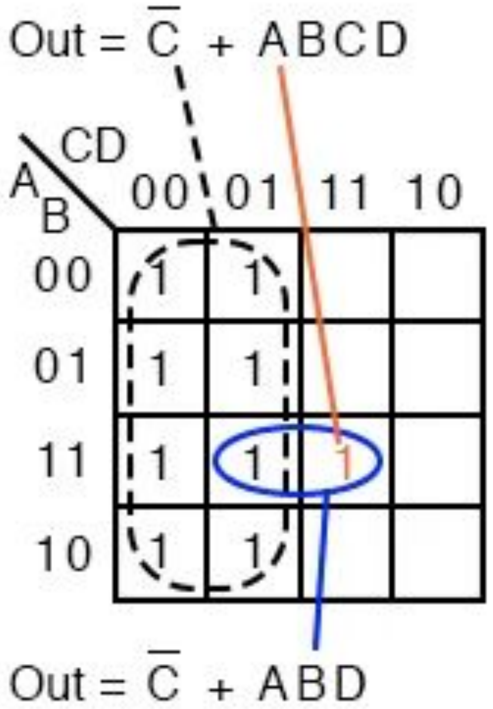
$$Out = \overline{A}\overline{B}\overline{C}\overline{D} + \overline{A}\overline{B}CD + \overline{A}B\overline{C}\overline{D} + \overline{A}BCD + AB\overline{C}\overline{D} + AB\overline{C}D + ABCD + A\overline{B}\overline{C}\overline{D} + A\overline{B}CD$$



$$Out = \overline{C}\overline{D} + CD + AB\overline{C}$$



$$Out = \overline{C}\overline{D} + CD + ABD$$



Simplification by Boolean Algebra

$Out = \overline{C} + ABCD$

Applying rule $A + \overline{A}B = A + B$ to the $\overline{C} + ABCD$ **term**

$Out = \overline{C} + ABD$

Q. Simplify the following Boolean functions using three-variable maps.

a. f(x, y, z) = ∑ (0, 1,5, 7)

b. f(x,y.z) = ∑(1,2,3,6,7)

c. F(x, y.z) = ∑(3, 5, 6, 7)

cl. f(A,,B, C) = ∑ (0, 2, 3, 4,6)

Q. Simplify the following Boolean functions using four-variable maps.

a. F(A, B, C, D) = ∑(4, 6, 7, 15)

b. F(A, B. C, D) = ∑ (3, 7. 11, 13. 14. 15)

c. F(A, B, C, D) = ∑ (0,1, 2, 4,5, 7, 11, 15)

d. F(A. B. c. D) = ∑ (0, 2, 4, 5, 6, 7, 8, 10, 13, 15)

Don't-Care Conditions

- ❏ The 1' s and 0' s in the map represent the min terms that make the function equal to 1 or 0.
- ❏ There are occasions when it does not matter if the function produces 0 or 1 for a given minterm.
- ❏ Since the function may be either 0 or 1, we say that we don't care what the function output is to be for this minterm.
- ❏ Min terms that don't-care conditions may produce either 0 or 1 for the function are said to be don't-care conditions and are marked with an x in the map.
- ❏ These don't-care conditions can be used to provide further simplification of the algebraic expression.

- Consider the following Boolean function together with the **don't-care minterms**:

  $F(A, B, C) = \sum(0, 2, 6)$

  $d(A, B, C) = \sum(1, 3, 5)$

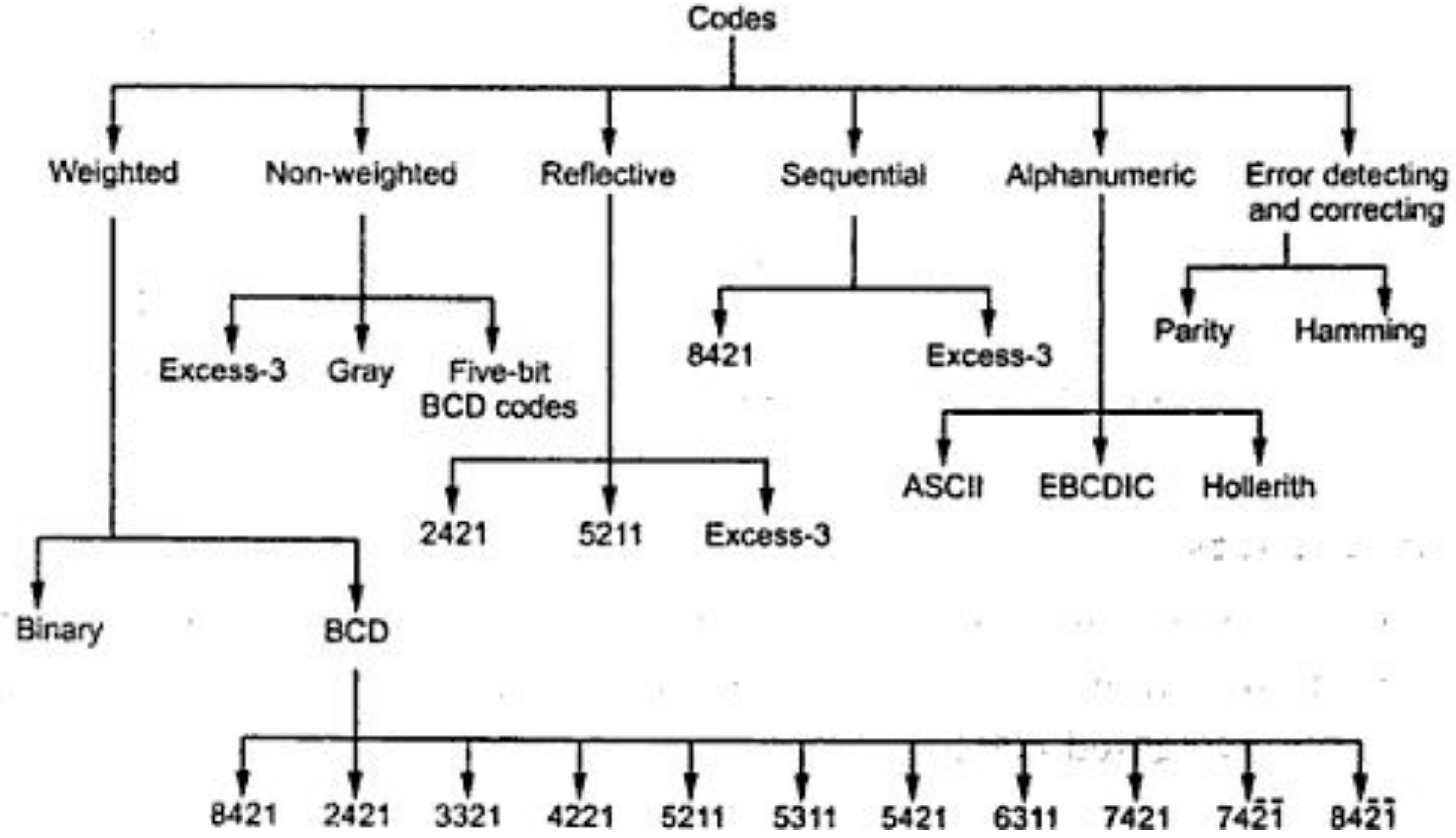
- The minterms of F are marked with 1's, those of d are marked with x 's, and the remaining squares are marked with O's .
- The 1's and x ' s are combined in any convenient manner so as to enclose the maximum number of adjacent squares.
- It is not necessary to include all or any of the x 's, but all the l's must be included.
- By including the don't-care minterms 1 and 3 with the l's in the first row we obtain the term A' . The remaining 1 for minterm 6 is combined with min term 2 to obtain the term BC '.

The simplified expression is

$$F = A' + BC'$$

- ❑ Note that **don't-care** minterm 5 was not included because it does not contribute to the simplification of the expression.
- ❑ Note also that if don't-care minterms 1 and 3 were not included with the 1's, the simplified expression for F would have been

$$F = A'C' + BC'$$



Binary codes block diagram

Number System

A digital system can understand positional number system only where there are a few symbols called digits and these symbols represent different values depending on the position they occupy in the number.

A value of each digit in a number can be determined using

- The digit
- The position of the digit in the number
- The base of the number system (where base is defined as the total number of digits available in the number system).

Decimal Number System

The number system that we use in our day-to-day life is the decimal number system. Decimal number system has base 10 as it uses 10 digits from 0 to 9. In decimal number system, the successive positions to the left of the decimal point represents units, tens, hundreds, thousands and so on.

Each position represents a specific power of the base (10). For example, the decimal number 1234 consists of the digit 4 in the units position, 3 in the tens position, 2 in the hundreds position, and 1 in the thousands position, and its value can be written as

$(1\times1000) + (2\times100) + (3\times10) + (4\times1)$

$(1\times10_3) + (2\times10_2) + (3\times10_1) + (4\times10_0)$

$1000 + 200 + 30 + 1$

1234

Binary Number System

Characteristics

- Uses two digits, 0 and 1.
- Also called base 2 number system
- Each position in a binary number represents a 0 power of the base (2). Example: 2_0
- Last position in a binary number represents an x power of the base (2). Example: 2_x where x represents the last position - 1.

Example

Binary Number: 10101_2

Calculating Decimal Equivalent −

Note: 10101_2 is normally written as 10101

Step	Binary Number	Decimal Number
Step 1	10101_2	$((1 \times 2_4) + (0 \times 2_3) + (1 \times 2_2) + (0 \times 2_1) + (1 \times 2_0))_{10}$
Step 2	10101_2	$(16 + 0 + 4 + 0 + 1)_{10}$
Step 3	10101_2	21_{10}

Octal Number System

- Uses eight digits, 0,1,2,3,4,5,6,7.
- Also called base 8 number system
- Each position in an octal number represents a 0 power of the base (8). Example: 80
- Last position in an octal number represents an x power of the base (8). Example: 8x where x represents the last position - 1.

Step	Octal Number	Decimal Number
Step 1	12570_8	$((1 \times 8^4) + (2 \times 8^3) + (5 \times 8^2) + (7 \times 8^1) + (0 \times 8^0))_{10}$
Step 2	12570_8	$(4096 + 1024 + 320 + 56 + 0)_{10}$
Step 3	12570_8	5496_{10}

Hexadecimal Number System

- Uses 10 digits and 6 letters, 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F.
- Letters represents numbers starting from 10. A = 10, B = 11, C = 12, D = 13, E = 14, F = 15.
- Also called base 16 number system.
- First position in a hexadecimal number represents a 0 power of the base (16). Example 16^0.
- Last position in a hexadecimal number represents an x power of the base (16). Example 16^x where x represents the last position - 1.

Step	Hexadecimal Number	Decimal Number
Step 1	$19FDE_{16}$	$((1 \times 16^4) + (9 \times 16^3) + (F \times 16^2) + (D \times 16^1) + (E \times 16^0))_{10}$
Step 2	$19FDE_{16}$	$((1 \times 16^4) + (9 \times 16^3) + (15 \times 16^2) + (13 \times 16^1) + (14 \times 16^0))_{10}$
Step 3	$19FDE_{16}$	$(65536 + 36864 + 3840 + 208 + 14)_{10}$
Step 4	$19FDE_{16}$	106462_{10}

Binary Number System:

- ❑ According to digital electronics and mathematics, a binary number is defined as a number that is expressed in the binary system or base 2 numeral system.
- ❑ It describes numeric values by two separate symbols; 1 (one) and 0 (zero).
- ❑ The base-2 system is the positional notation with 2 as a radix.
- ❑ The binary system is applied internally by almost all latest computers and computer-based devices because of its direct implementation in electronic circuits using logic gates.
- ❑ Every digit is referred to as a **bit**.
- ❑ A single binary digit is called a “**Bit”.** A binary number consists of several bits. Examples are:

10101 is a five-bit binary number
101 is a three-bit binary number
100001 is a six-bit binary number

Number Conversion

- Binary to Decimal
- Decimal to Binary
- Hexadecimal to Decimal
- Hexadecimal to Decimal-Binary
- Octal to Decimal- Binary

Practice Convert:

11010011_2 to decimal.

10111011_2 to decimal.

$34F2_{16}$ to decimal/ To Hexadecimal

$FFFF_{16}$ to decimal/ To Hexadecimal

Binary Numbers Representation

The Binary numbers divided into the following two groups − Unsigned numbers and Signed numbers.

Unsigned Numbers

- ❑ Unsigned numbers contain only magnitude of the number. They don’t have any sign.
- ❑ That means all unsigned binary numbers are positive. As in decimal number system, the placing of positive sign in front of the number is optional for representing positive numbers.
- ❑ Therefore, all positive numbers including **zero** can be treated as unsigned numbers if positive sign is not assigned in front of the number.

Signed Numbers

- ❑ Signed numbers contain both sign and magnitude of the number. Generally, the sign is placed in front of number.
- ❑ So, we have to consider the positive sign for positive numbers and negative sign for negative numbers.
- ❑ Therefore, all numbers can be treated as signed numbers if the corresponding sign is assigned in front of the number.
- ❑ If sign bit is zero, which indicates the binary number is positive. Similarly, if sign bit is one, which indicates the binary number is negative.

Representation of Un-Signed Binary Number

The bits present in the un-signed binary number holds the magnitude of a number. That means, if the un-signed binary number contains ‘N’ bits, then all N bits represent the magnitude of the number, since it doesn’t have any sign bit.

Example:

Consider the decimal number 108. The binary equivalent of this number is 1101100. This is the representation of unsigned binary number.

$\mathbf{108_{10} = 110\ 1100}$

It is having 7 bits. These 7 bits represent the magnitude of the number 108.

Representation of Signed Binary Numbers

The Most Significant Bit MSB of signed binary numbers is used to indicate the sign of the numbers. Hence, it is also called as sign bit.

- The positive sign is represented by placing ‘0’ in the sign bit. Similarly, the negative sign is represented by placing ‘1’ in the sign bit.
- If the signed binary number contains ‘N’ bits, then N−1 bits only represent the magnitude of the number since one bit MSB is reserved for representing sign of the number.

There are three types of representations for signed binary numbers

- ***Sign-Magnitude form***
- ***1’s complement form***
- ***2’s complement form***

Sign-Magnitude form

In sign-magnitude form, the **MSB** is used for representing sign of the number and the **remaining bits** represent the magnitude of the number. So, just include sign bit at the left most side of unsigned binary number. This representation is similar to the signed decimal numbers representation.

Example

- Consider the negative decimal number -108.
- The magnitude of this number is 108. We know the unsigned binary representation of 108 is 1101100.
- It is having 7 bits. All these bits represent the magnitude.
- Since the given number is negative, consider the sign bit as one, which is placed on left most side of magnitude.

$$-108_{10} = \mathbf{1}\ 110\ \ 1100_{2}$$

1’s complement form

- The 1’s complement of a number is obtained by complementing all the bits of signed binary number. So, 1’s complement of positive number gives a negative number.
- Similarly, 1’s complement of negative number gives a positive number.

That means, if you perform two times 1’s complement of a binary number including sign bit, then you will get the original signed binary number.

Example

Consider the negative decimal number -108. The magnitude of this number is 108.

We know the signed binary representation of 108 is 0110 1100.

It is having 8 bits. The MSB of this number is **zero**, which indicates **positive** number. Complement of zero is one and vice-versa. So, replace zeros by ones and ones by zeros in order to get the negative number.

$$-108_{10} = 10010011_2$$

2’s complement form

The 2’s complement of a binary number is obtained by **adding one to the 1’s complement of signed binary number**. So, 2’s complement of positive number gives a negative number. Similarly, 2’s complement of negative number gives a positive number.

That means, if you perform two times 2’s complement of a binary number including sign bit, then you will get the original signed binary number.

Example:

Consider the negative decimal number -108. We know the 1’s complement of $(108)_{10}$ is $(10010011)_2$

2’s compliment of 108_{10} = *1’s complement of* 108_{10} + *1.*

$= 10010011 + 1$

$= 10010100_2$

Signed Binary Arithmetic

The basic arithmetic operations are addition and subtraction.

1. Consider the two signed binary numbers A & B, which are **represented in 2’s complement** form.
2. We can perform the addition of these two numbers, which is similar to the addition of two unsigned binary numbers.
3. But, if the resultant sum contains carry out from sign bit, then discard ***ignore*** it in order to get the correct value.
4. If resultant sum is positive, you can find the magnitude of it directly. But, if the resultant sum is negative, then take 2’s complement of it in order to get the magnitude.

Let us perform the addition of two decimal numbers -7 and -4 using 2’s complement method.

The 2’s complement representation of -7 and -4 with 5 bits each are shown below.

The addition of these two numbers is

$$-7_{10} + -4_{10} = 11001_2 + \ 11100_2$$
$$-7_{10} + -4_{10} = 110101_2$$