



Dr. Vishwanath Karad

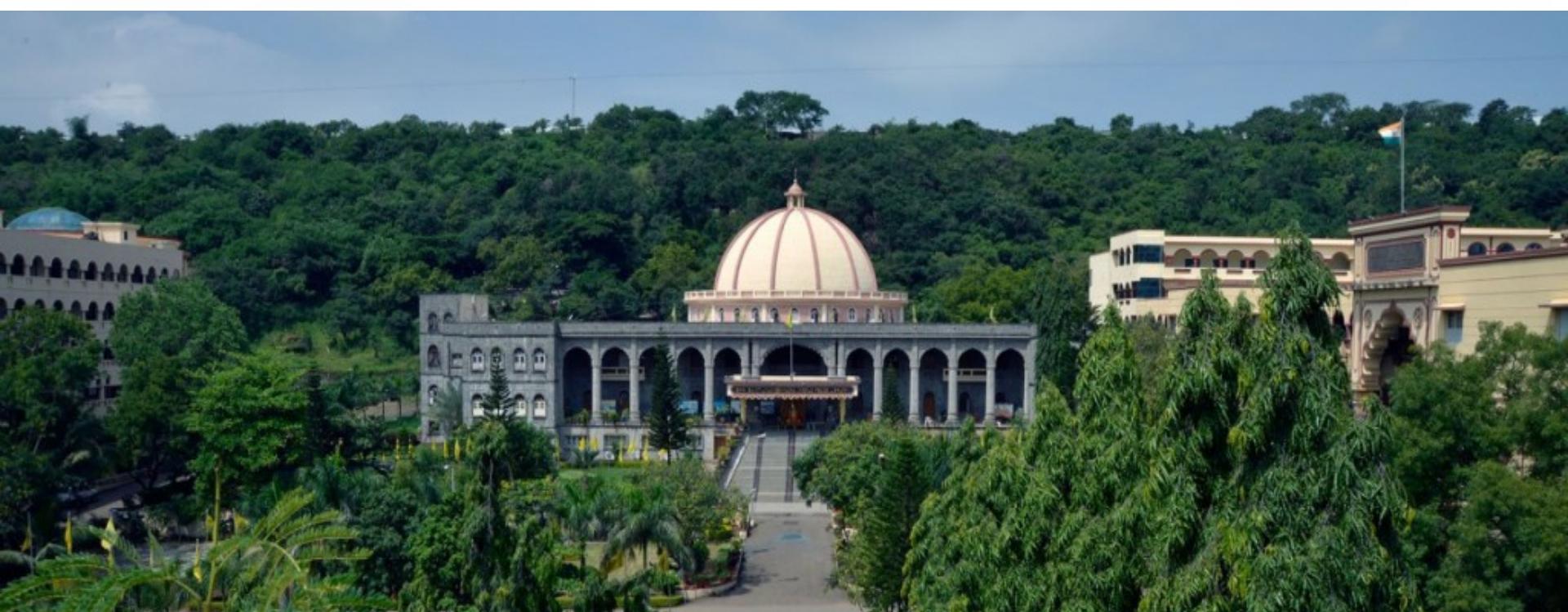
**MIT WORLD PEACE  
UNIVERSITY | PUNE**

TECHNOLOGY, RESEARCH, SOCIAL INNOVATION & PARTNERSHIPS

1

# Basics of Electrical and Electronics Engineering

**ECE101B**





# **DIGITAL INTEGRATED CIRCUITS**

# Fundamentals of Digital Electronics

## Contents:

Digital logic levels, basic gates and universal gates, Boolean algebra, Combinational Logic Circuits, De-Morgan's theorems, SOP, POS, K map, Half Adder, Full adder, flip flops: S R flip flop, JK flip flop, D flip flop, T flip flop, Shift registers, Introduction to Microcontrollers.

## Reference Book:

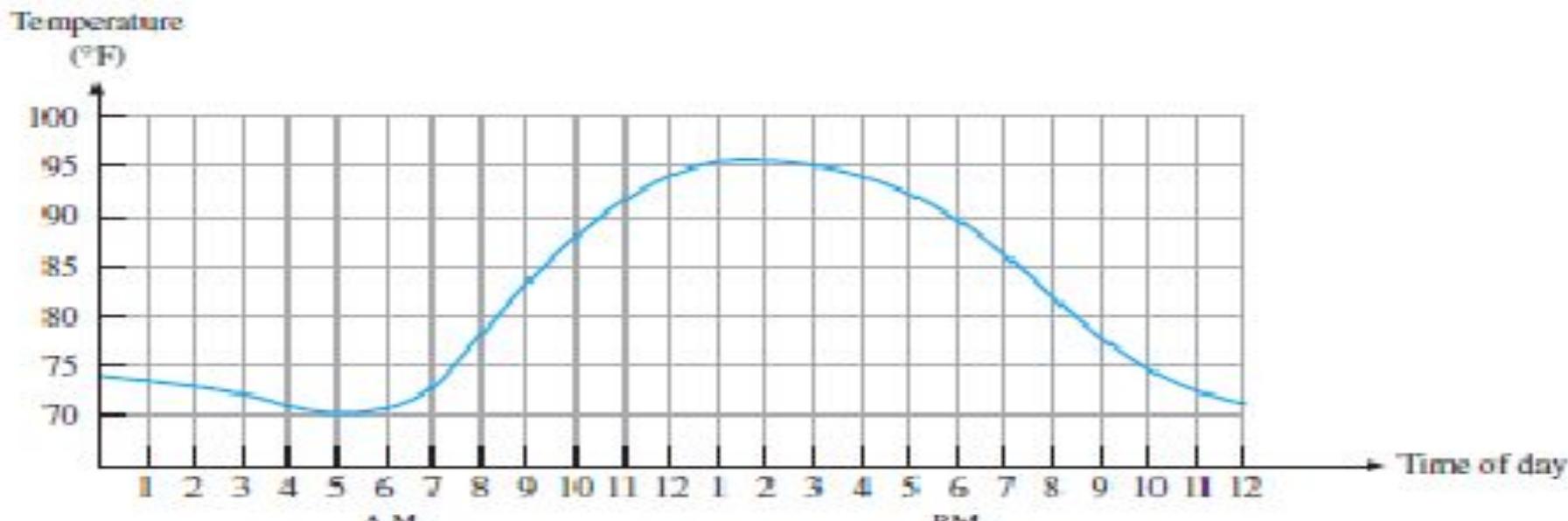
- ❖ R.P. Jain, Modern Digital Electronics. New Delhi: Tata McGraw-Hill, 4th Edition, 2009
- ❖ Digital Fundamentals 11<sup>th</sup> Edition by Thomas L. Floyd

# Digital and Analog Quantities

- For many years, applications of digital electronics were limited to computer systems.
- Digital Technology have wide range of applications
  - ✓ Television
  - ✓ Communication systems
  - ✓ Radar
  - ✓ Navigation
  - ✓ Military Systems
  - ✓ Medical Instrumentations
  - ✓ Industrial Process Control
  - ✓ Consumer electronics etc.
- Electronics circuits can be divided into two broad categories
  - ✓ Digital
  - ✓ Analog

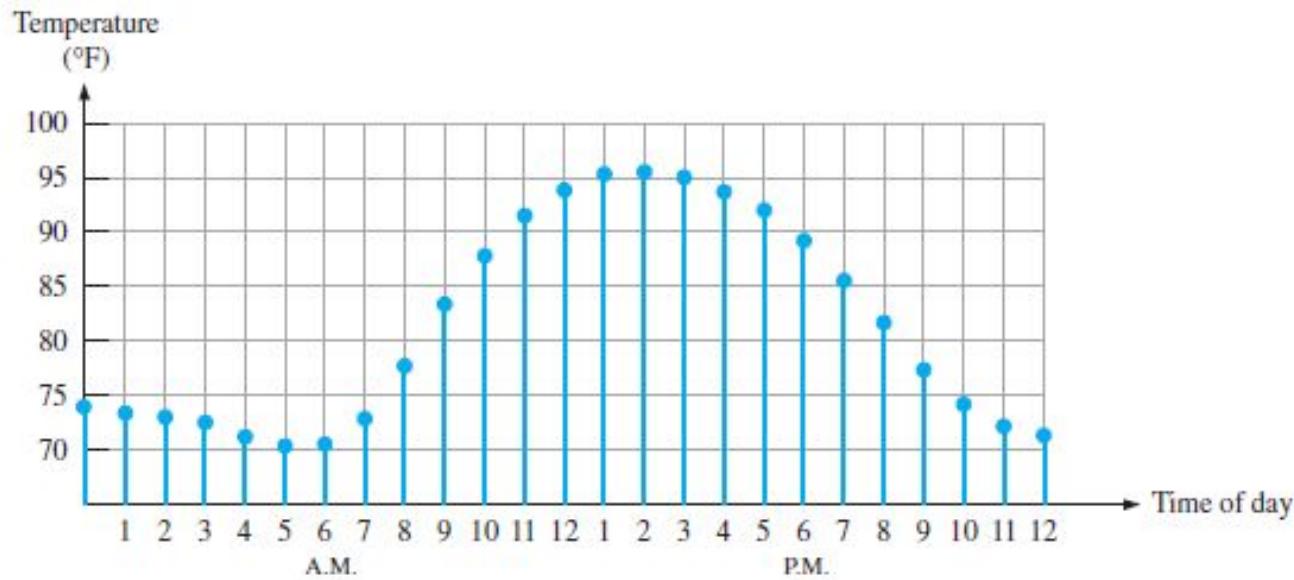
# Digital and Analog Quantities

- Analog: is one having continuous values
- Digital: is one having discrete set of values
- Example: Air temperature changes over continuous range of values
- **Analog Signal**



Graph of an analog quantity (temperature versus time).

# Digital code

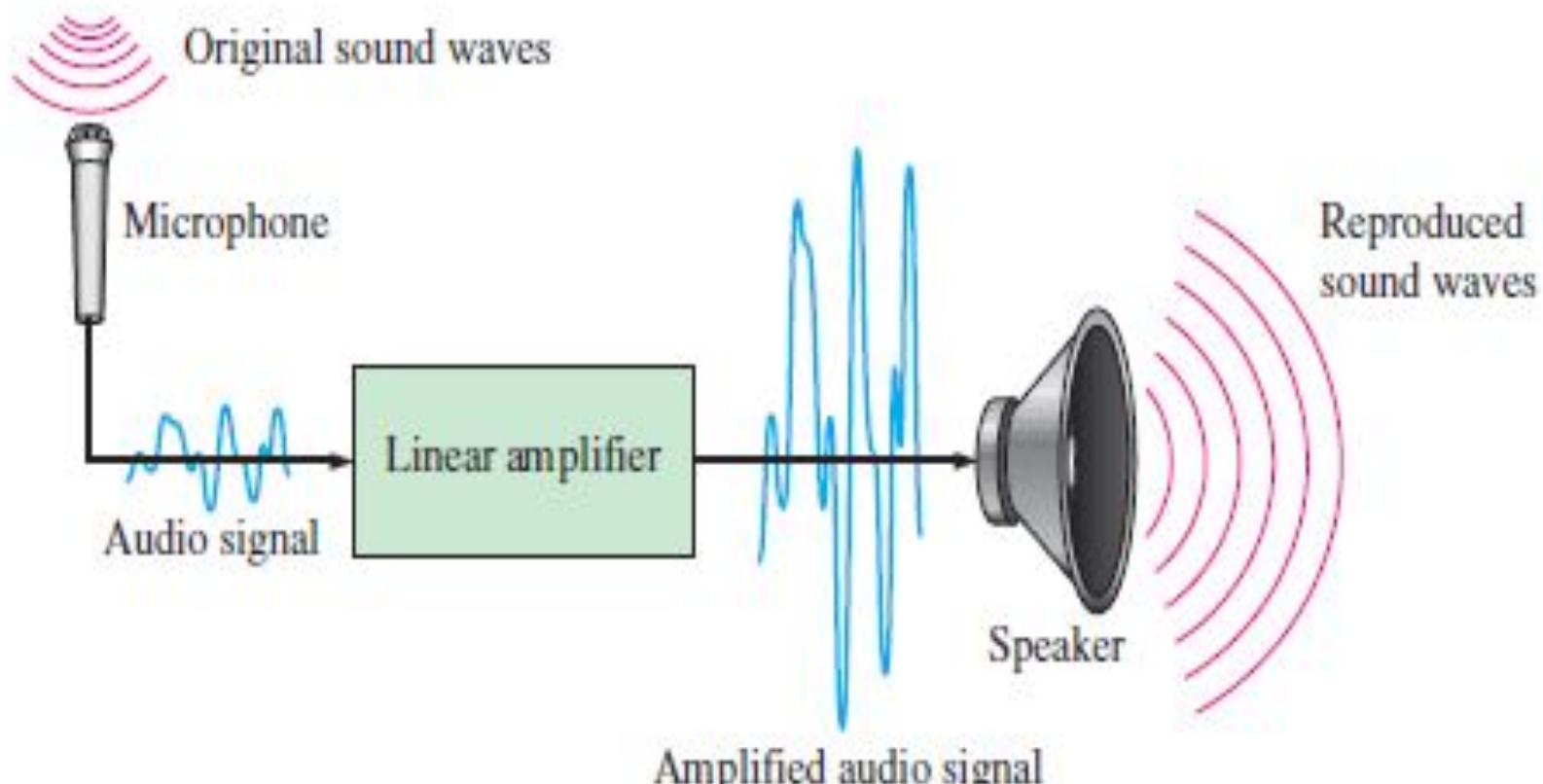


Sampled-value representation (quantization) of the analog quantity in  
Each value represented by a dot can be digitized by representing it as a digital  
code that consists of a series of 1s and 0s.

# Advantages of Digital signals

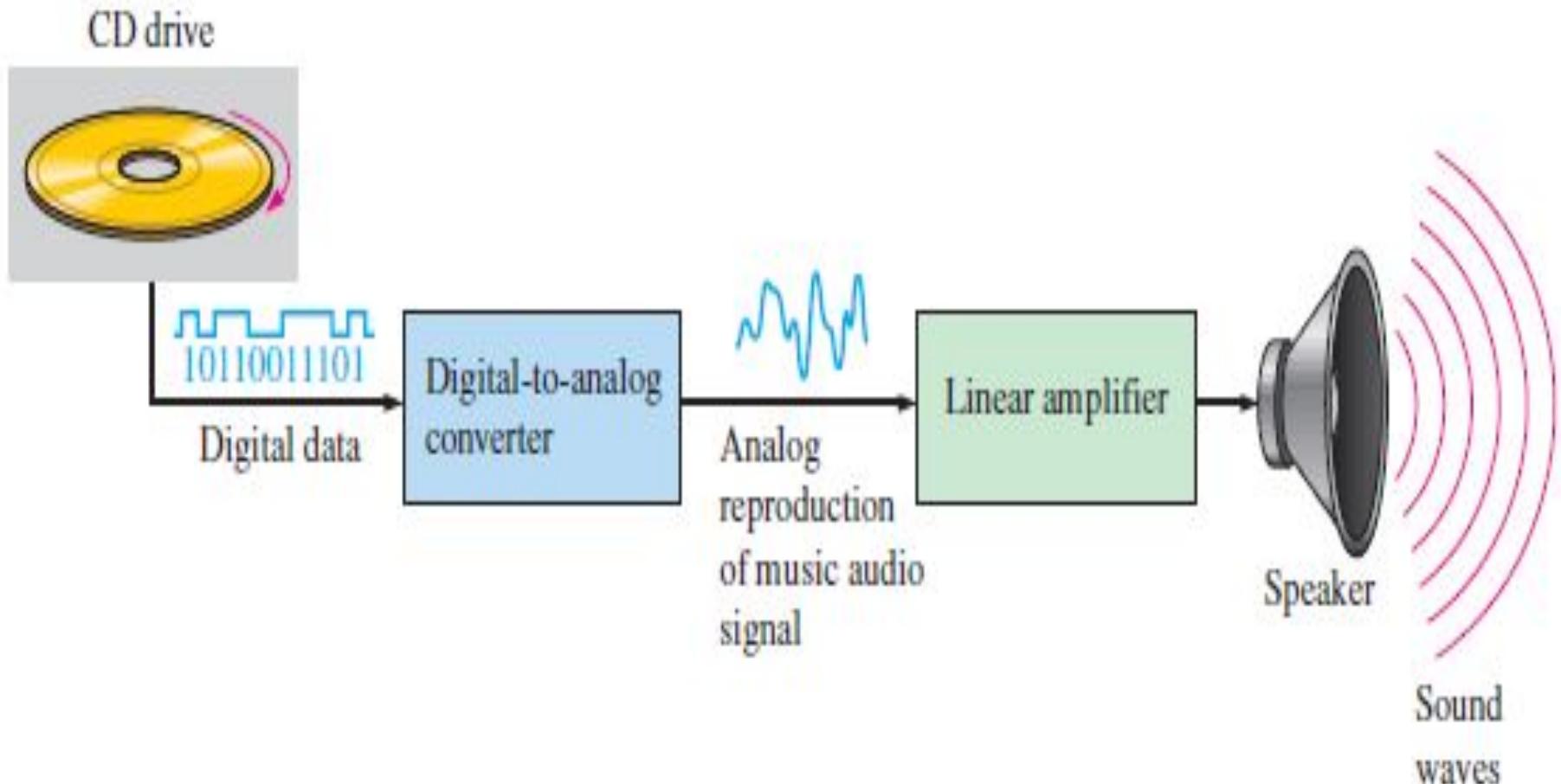
- Data transmission is more effective and reliable
- Digital data has a great advantage when storage is necessary
- Digital data is more accurate and precise
- Its ripple free or noise free

# Analog system



A basic audio public address system.

# A System Using analog and Digital Methods



Basic block diagram of a CD player. Only one channel is shown.

# Mechatronics

- The interdisciplinary field that comprises both mechanical and electronic components is known as mechatronics
- Both digital and analog electronics are used in the control of various mechanical systems.



(b) Robotic arm



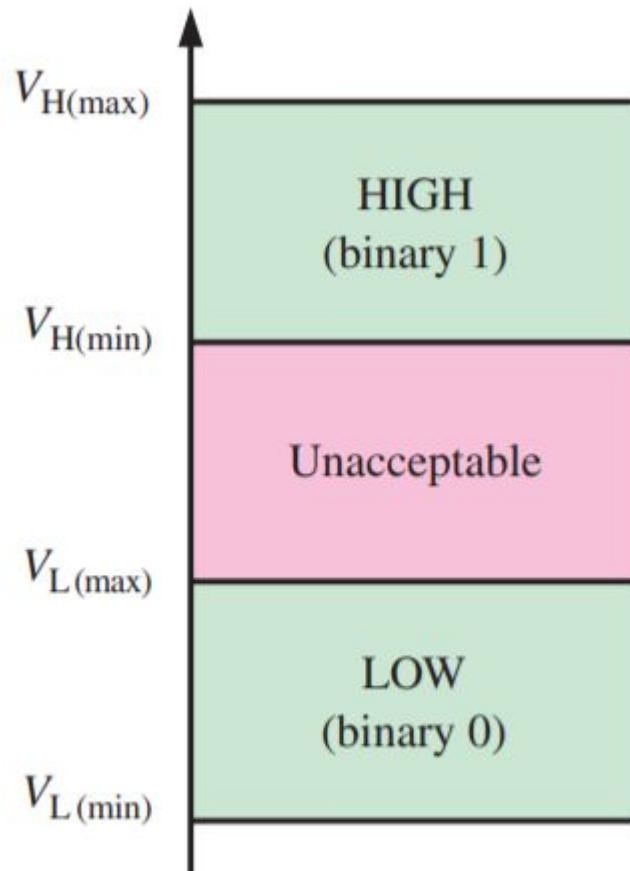
(c) Automotive assembly line

# Digital Electronics

- It uses Binary Digits 0 and 1
- Each of the two digits in the binary system, 1 and 0, is called a bit,
- Two different voltage levels are used to represent the two bits.
- 1 - is represented by the higher voltage, which will be referred as a HIGH,
- 0 - is represented by the lower voltage level, which will be referred as a LOW.
- This is called positive logic and will be used throughout the topic.

**HIGH = 1 and LOW = 0**

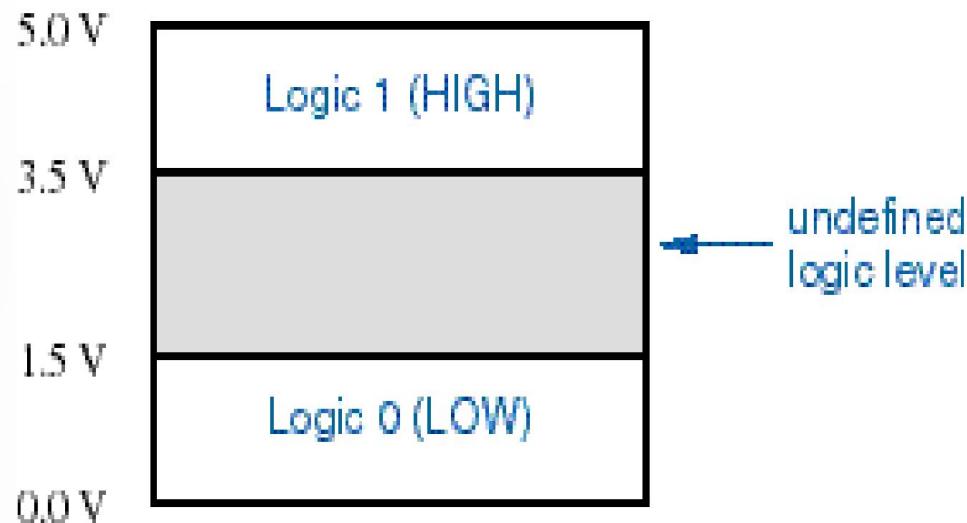
# Logic Level ranges or Voltage for a digital circuits



**FIGURE 1–6** Logic level ranges of voltage for a digital circuit.

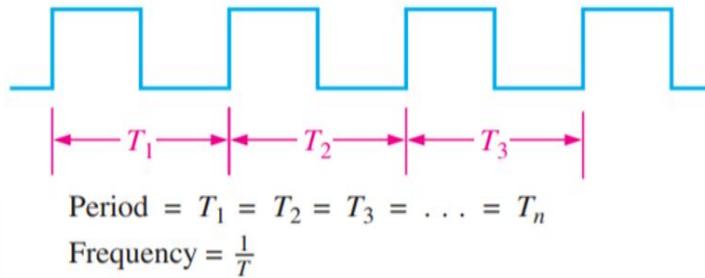
# Logic Levels

- Positive Logic levels



# Examples of digital waveforms

## (a) Periodic (square wave)



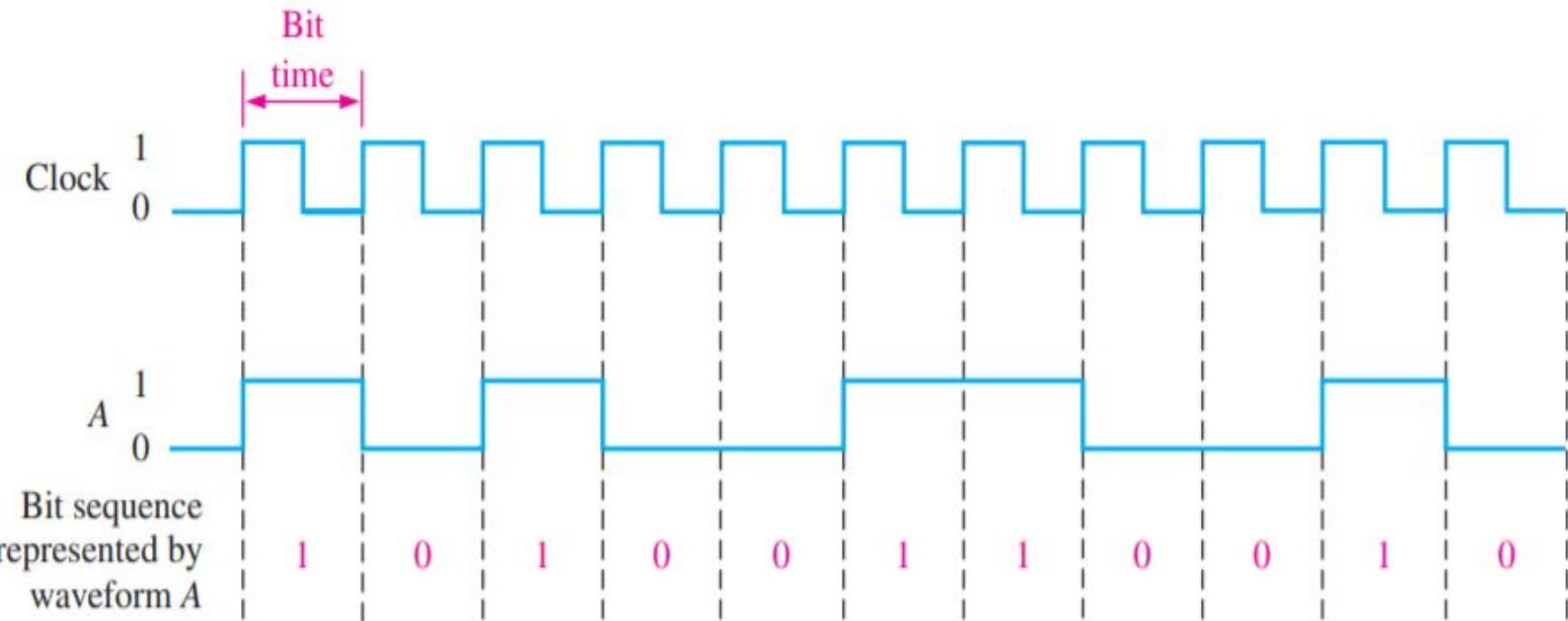
## (b) Nonperiodic



- An important characteristic of a periodic digital waveform is its duty cycle, which is the
- Ratio of the pulse width ( $t_w$ ) to the period (T). It can be expressed as a percentage

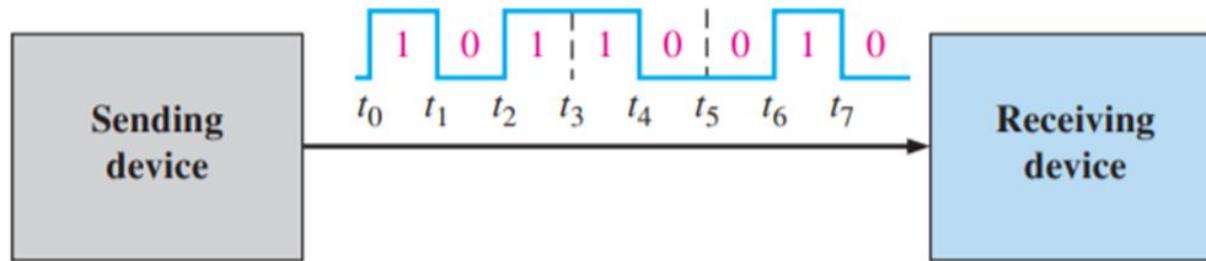
$$\text{Duty cycle} = \left( \frac{t_w}{T} \right) 100\%$$

# Digital waveforms

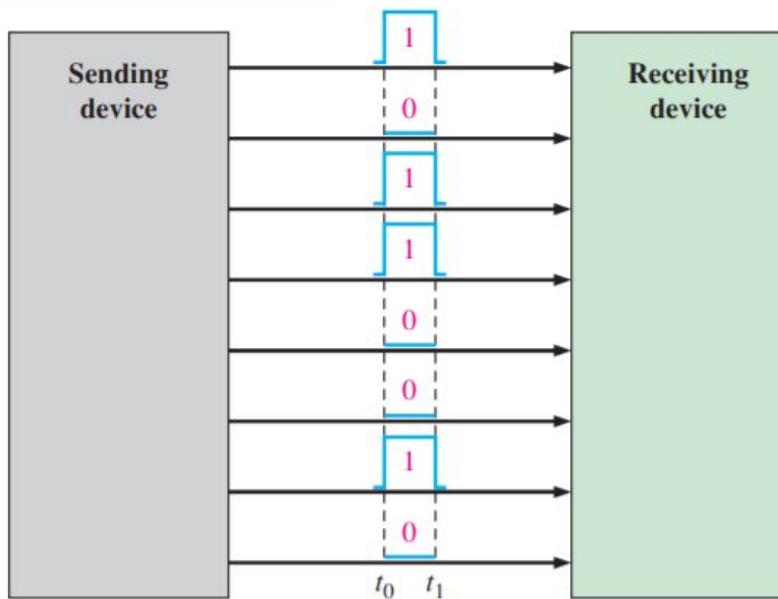


**FIGURE 1-11** Example of a clock waveform synchronized with a waveform representation of a sequence of bits.

# Digital Data Transfer

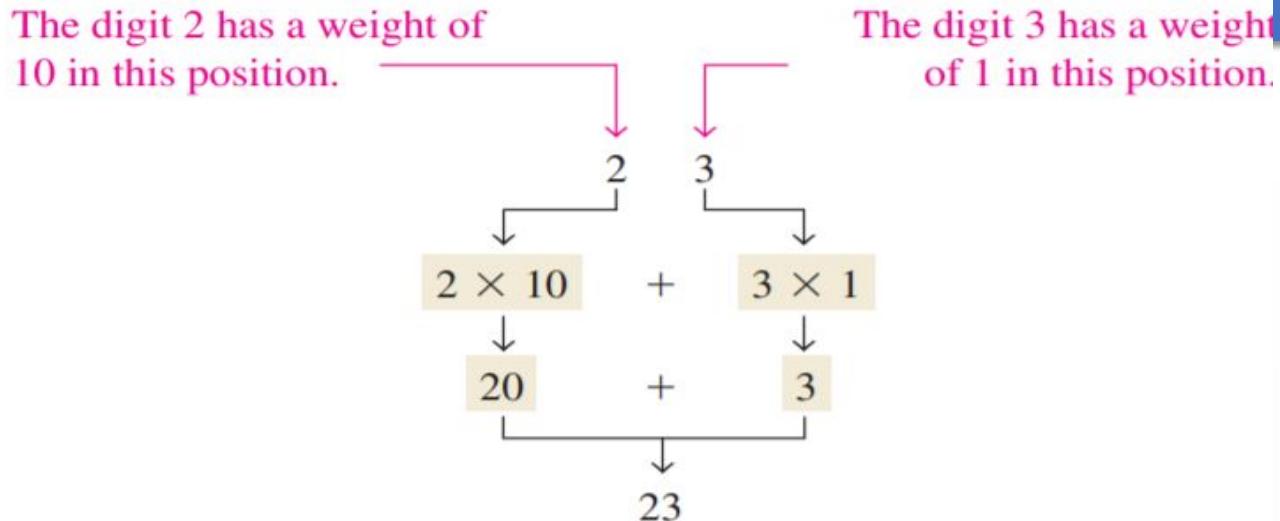


(a) Serial transfer of 8 bits of binary data. Interval  $t_0$  to  $t_1$  is first.



(b) Parallel transfer of 8 bits of binary data. The beginning time is  $t_0$ .

# Decimal Number System



- The decimal number system has a base of 10.
- The position of each digit in a decimal number indicates the magnitude of the quantity represented and can be assigned a weight.
- The weights for whole numbers are positive powers of ten that increase from right to left, beginning with  $10^0 = 1$ .

$$\dots 10^5 10^4 10^3 10^2 10^1 10^0$$

# Binary Numbers

- Binary system has only two digits.
- Binary system with its two digits is a base-two system.
- The two binary digits (bits) are 1 and 0.
- The position of a 1 or 0 in a binary number indicates its weight, or value within the number, just as the position of a decimal digit determines the value of that digit.
- The weights in a binary number are based on powers of two.

Decimal No.	MSB $2^3$	$2^2$	$2^1$	LSB $2^0$
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1
12	1	1	0	0
13	1	1	0	1
14	1	1	1	0
15	1	1	1	1

# Binary Numbers

Binary weights.

**Positive Powers of Two  
(Whole Numbers)**

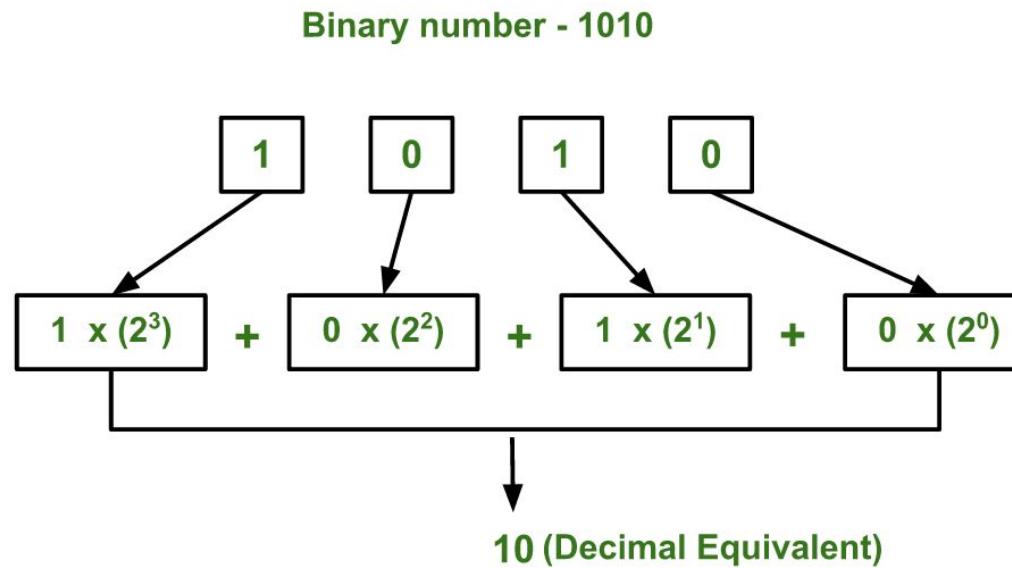
$2^8$	$2^7$	$2^6$	$2^5$	$2^4$	$2^3$	$2^2$	$2^1$	$2^0$
256	128	64	32	16	8	4	2	1

**Negative Powers of Two  
(Fractional Number)**

$2^{-1}$	$2^{-2}$	$2^{-3}$	$2^{-4}$	$2^{-5}$	$2^{-6}$
1/2	1/4	1/8	1/16	1/32	1/64
0.5	0.25	0.125	0.0625	0.03125	0.015625

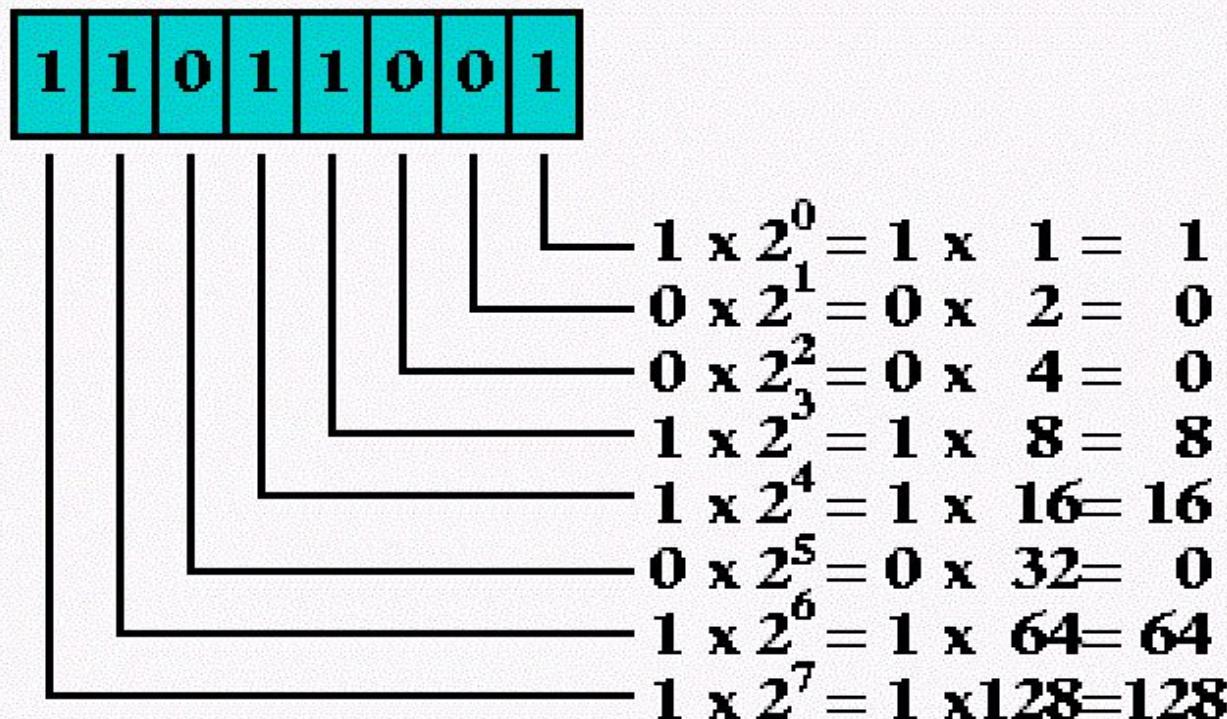
# Binary-to-Decimal Conversion

- The decimal value of any binary number can be found by adding the weights of all bits that are 1 and discarding the weights of all bits that are 0.



- Convert the binary whole number 11011001 to decimal.
- Determine the weight of each bit that is a 1, and then find the sum of the weights to get the decimal number.

# Binary-to-Decimal Conversion



$$1 + 8 + 16 + 64 + 128 = 217$$

# Binary-to-Decimal Conversion

## Steps for Decimal to Binary Conversion

**Step – 1** Divide the decimal number which is to be converted by two

**Step – 2** The remainder which is obtained from step 1 is the least significant bit of the binary number.

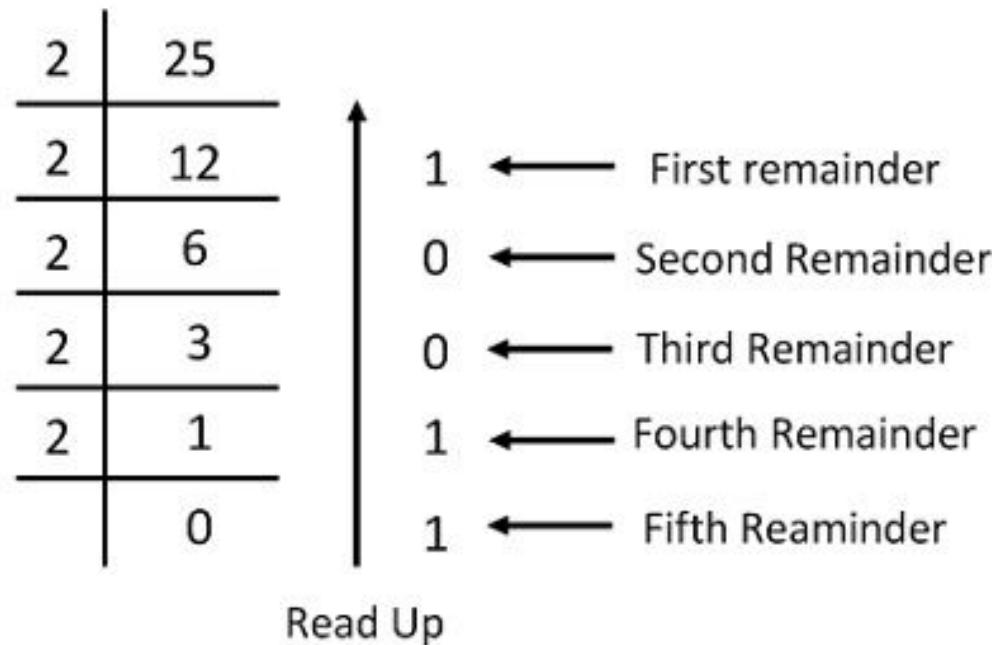
**Step – 3** Divide the quotient which is obtained from the step 2 and the remainder obtained from this is the second least significant bit of the binary number.

**Step – 4** Repeat the process until the quotient remains zero.

**Step – 5** The last remainder obtained from the division is the most significant bit of the binary number. Hence arrange the number from most significant bit to the least significant bit (i.e., from bottom to top).

# Binary-to-Decimal Conversion

Convert decimal 25 to binary



Binary Number = 11001

# Binary-to-Decimal Conversion

$(160)_{10}$

2	160	
2	80	0
2	40	0
2	20	0
2	10	0
2	5	0
2	2	1
	1	0

Binary Number = 10100000

# Binary Addition

- The four basic rules for adding binary digits (bits) are as follows:

$0 + 0 = 0$	Sum of 0 with a carry of 0
$0 + 1 = 1$	Sum of 1 with a carry of 0
$1 + 0 = 1$	Sum of 1 with a carry of 0
$1 + 1 = 10$	Sum of 0 with a carry of 1

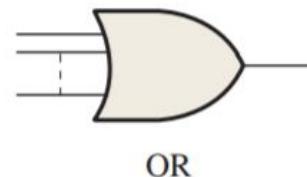
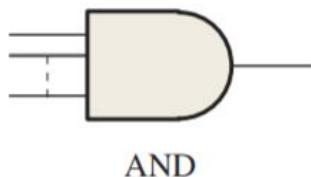
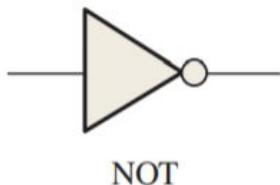
In binary  $1 + 1 = 10$ , not  
2.

Ex. Addition of  $11 + 1$ :

$$\begin{array}{r} & \text{Carry} & \text{Carry} \\ & 1 & 1 \\ \begin{array}{r} 0 \\ + 0 \\ \hline 1 \end{array} & \leftarrow & \leftarrow \\ & 1 & 0 \\ & 0 & 1 \\ & \hline & 0 & 1 \end{array}$$

# Basic Logic Functions

- In the 1850s, the Irish logician and mathematician George Boole developed a mathematical system for formulating logic statements with symbols so that problems can be written and solved in a manner similar to ordinary algebra
- The term logic is applied to digital circuits used to implement logic functions.
- Three basic logic functions - NOT, AND, and OR
- Logic functions are indicated by standard symbols shown below
- The inputs are on the left of each symbol and the output is on the right.
- A circuit that performs a specified logic function (AND, OR) is called a logic gate.
- AND and OR gates can have any number of inputs.



**The basic logic functions and symbols**

# Logic Functions

- In logic functions, the true/false conditions are represented by a HIGH (true) and a LOW (false).

## 1. NOT function/NOT Gate

The NOT function is implemented by a logic circuit known as an inverter

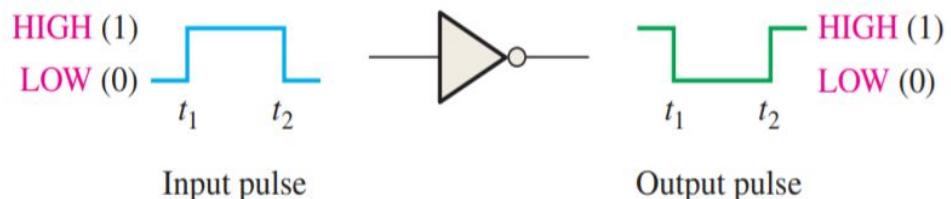


**FIGURE 1-17** The NOT function.

Logic Expression for an Inverter :  $A \rightarrow X = \bar{A}$

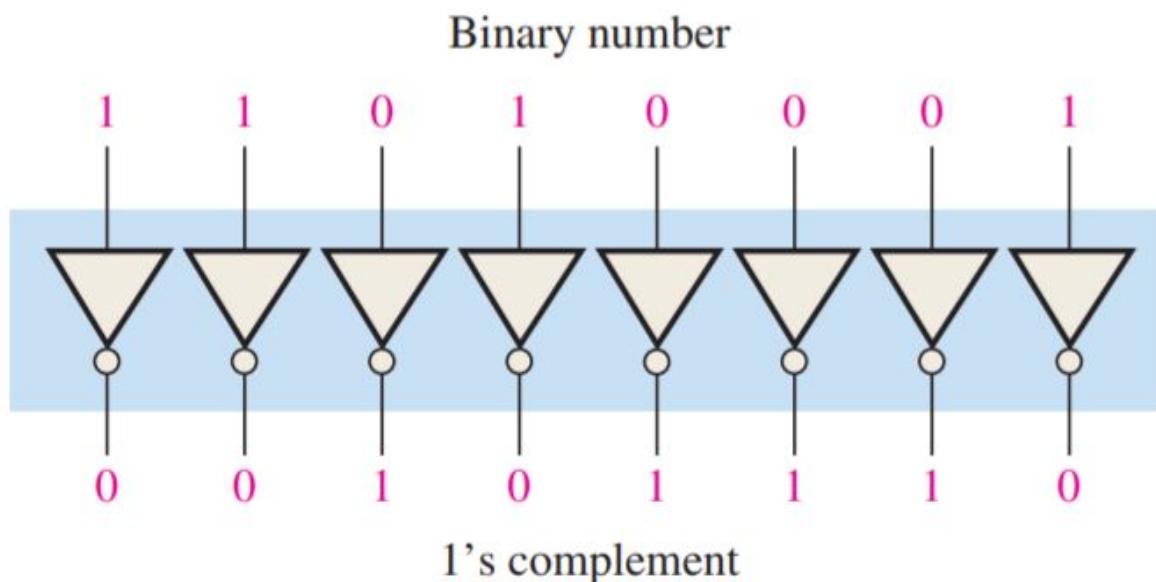
Inverter truth table.

Input	Output
LOW (0)	HIGH (1)
HIGH (1)	LOW (0)



# An Application of Inverter

- Figure shows a circuit for producing the 1's complement of an 8-bit binary number.

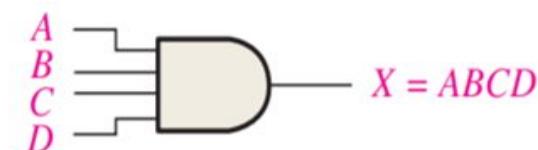
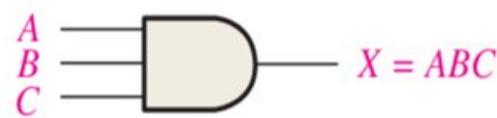
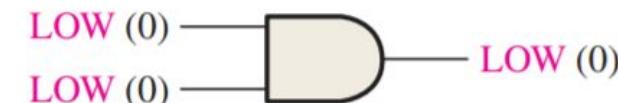
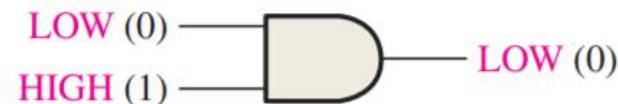


# Logic Functions (AND)

## 2. AND function/ AND Gate

The AND function produces a HIGH output only when all the inputs are HIGH, as indicated in Figure for the case of two inputs.

- When one input is HIGH and the other input is HIGH, the output is HIGH.
- When any or all inputs are LOW, the output is LOW.
- The AND function is implemented by a logic circuit known as an AND gate.



# AND Gate

The total number of possible combinations of binary inputs to a gate is determined by the following formula:

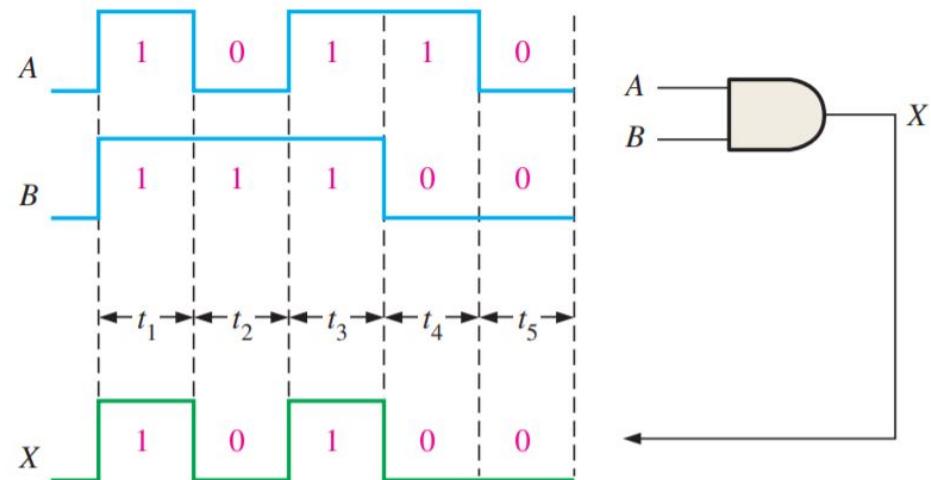
$$N = 2^n$$

Logic Expression for AND gate :  $X = A \bullet B = AB$

Truth table for a 2-input AND gate.

Inputs		Output
A	B	X
0	0	0
0	1	0
1	0	0
1	1	1

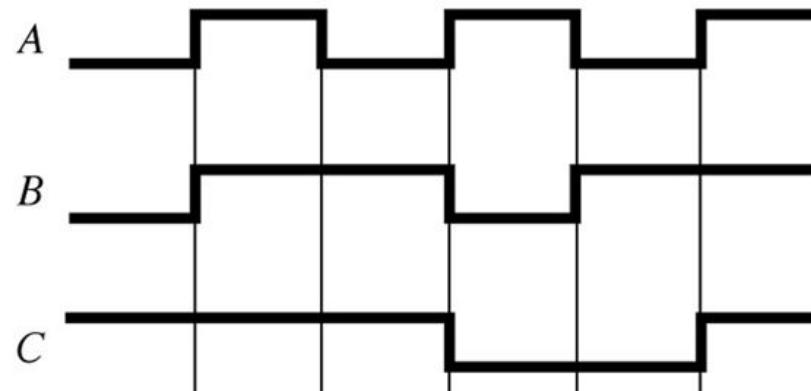
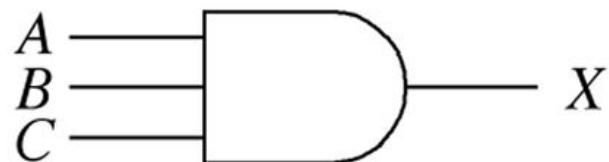
1 = HIGH, 0 = LOW



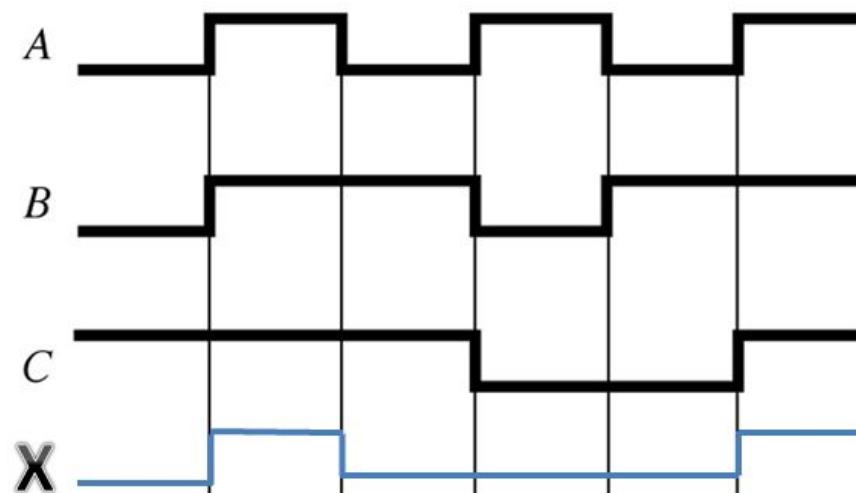
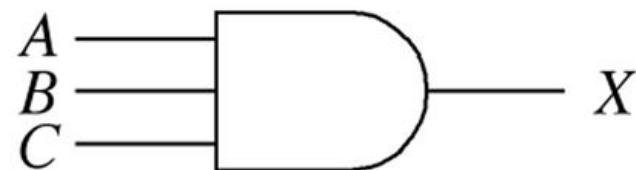
Example of AND gate operation with a timing diagram showing input and output relationships.

# Example 1

Determine output X: Write logical expression and output waveform

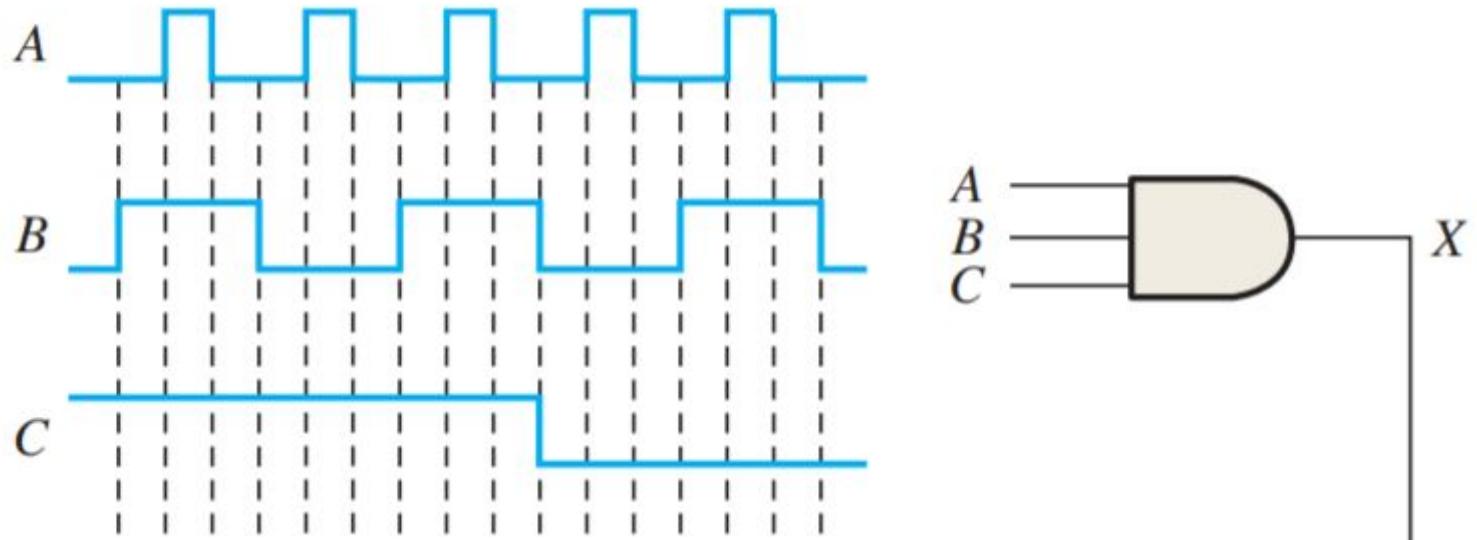


# Example 1 Solution



## Example 2

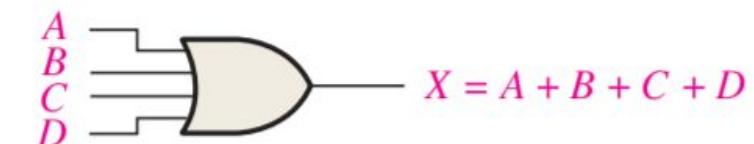
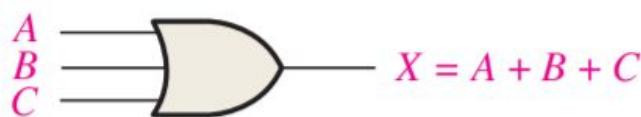
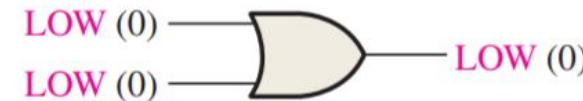
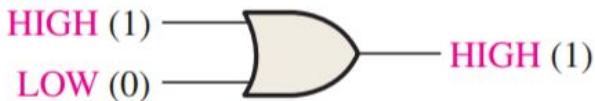
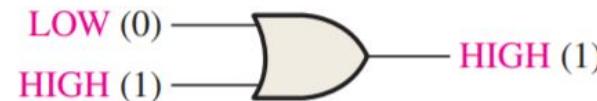
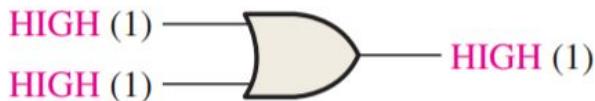
Determine output X: Write logical expression and output waveform



# Logic Functions (OR)

## 3. OR function/ OR Gate

- The OR function produces a HIGH output when one or more inputs are HIGH, as indicated in Figure for the case of two inputs.
- When one input is HIGH or the other input is HIGH or both inputs are HIGH, the output is HIGH.
- When both inputs are LOW, the output is LOW.
- The OR function is implemented by a logic circuit known as an OR gate.



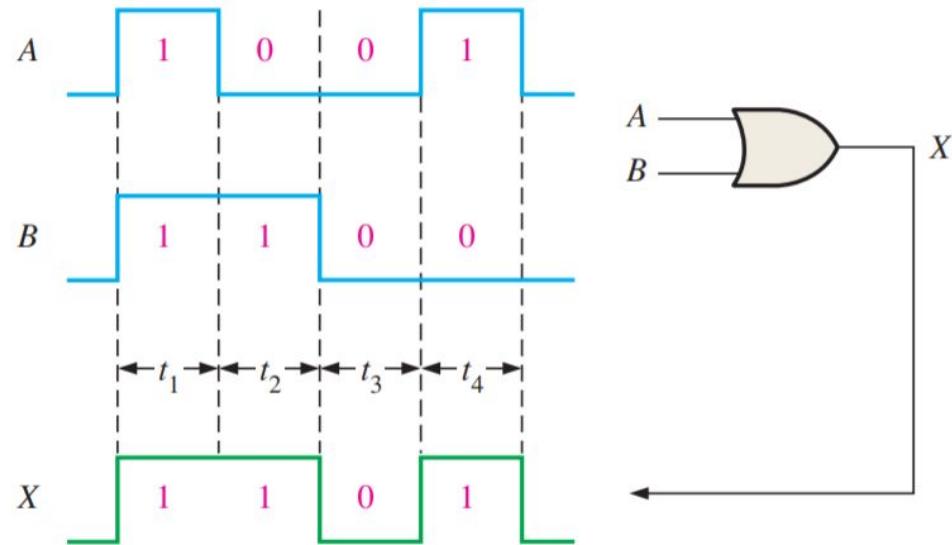
# OR Gate

Logic Expression OR gate  $X = A + B$

Truth table for a 2-input OR gate.

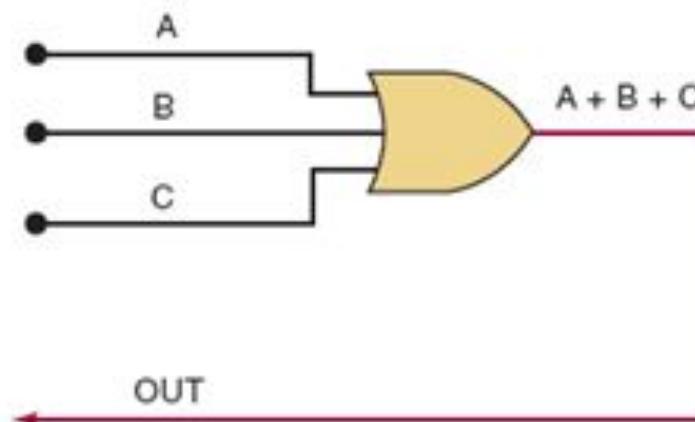
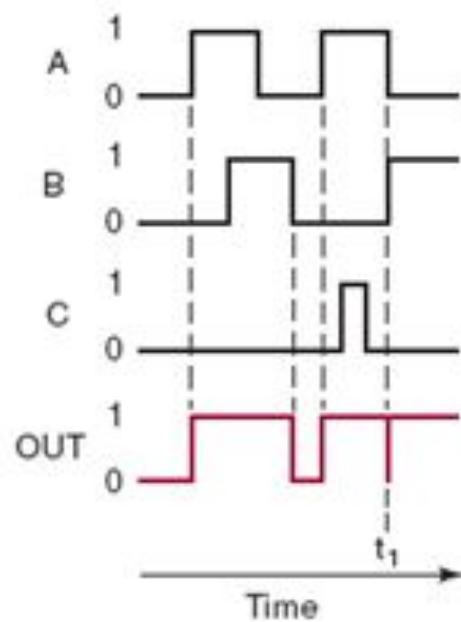
Inputs		Output
A	B	X
0	0	0
0	1	1
1	0	1
1	1	1

1 = HIGH, 0 = LOW



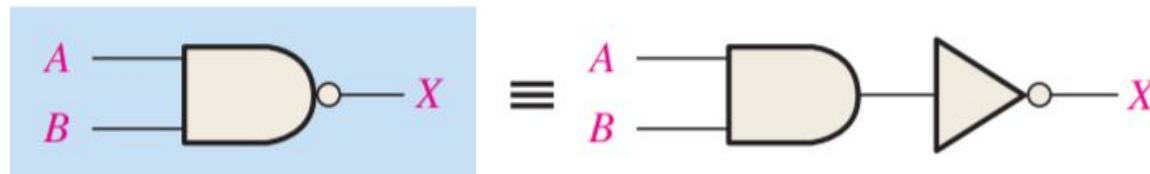
Example of OR gate operation with a timing diagram showing input and output relationships.

# OR Gate



# Logic Function NAND

The NAND gate is the same as the AND gate with the inverted output.



The Boolean expression for the output of a 2-input NAND gate is

$$X = \overline{AB}$$

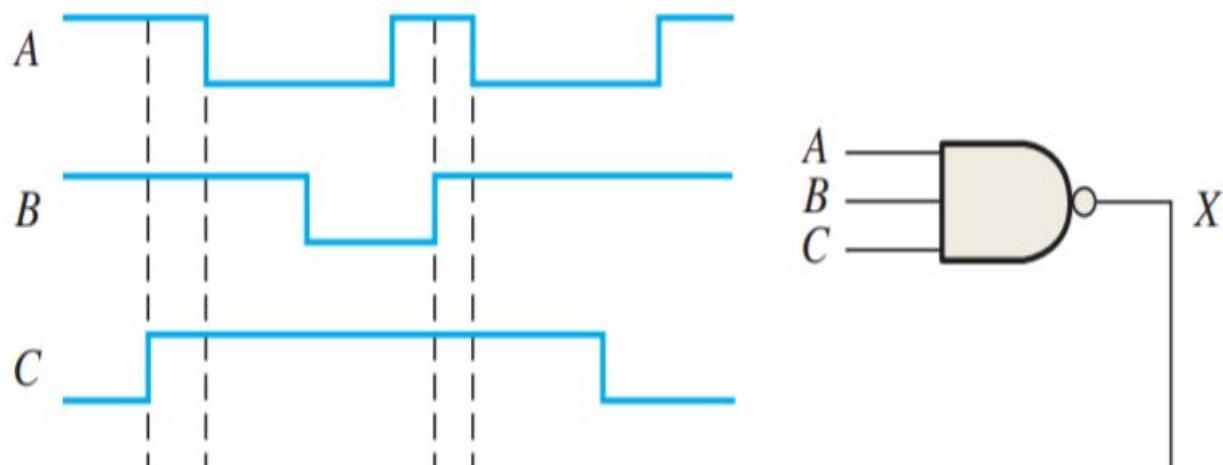
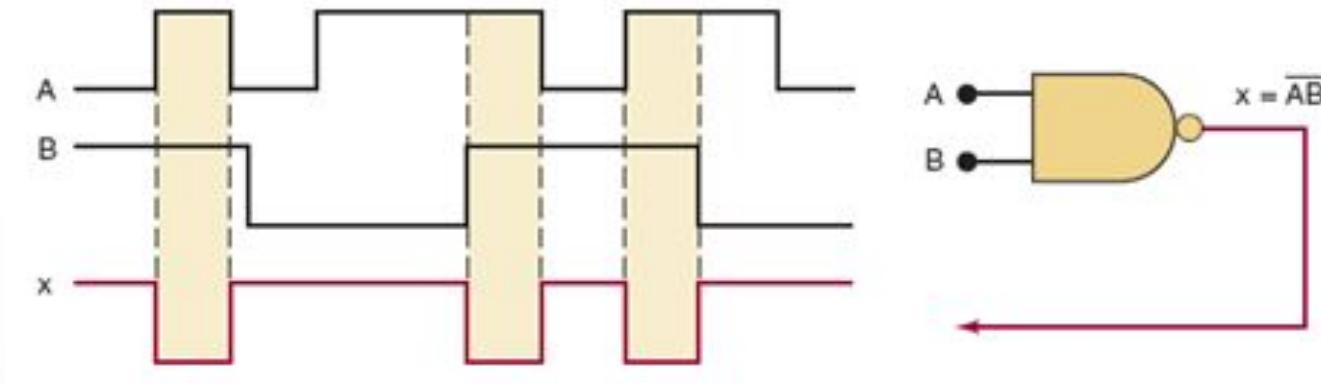
<b>A</b>	<b>B</b>	<b><math>\overline{AB} = X</math></b>
0	0	$\overline{0 \cdot 0} = \bar{0} = 1$
0	1	$\overline{0 \cdot 1} = \bar{0} = 1$
1	0	$\overline{1 \cdot 0} = \bar{0} = 1$
1	1	$\overline{1 \cdot 1} = \bar{1} = 0$

Truth table for a 2-input NAND gate.

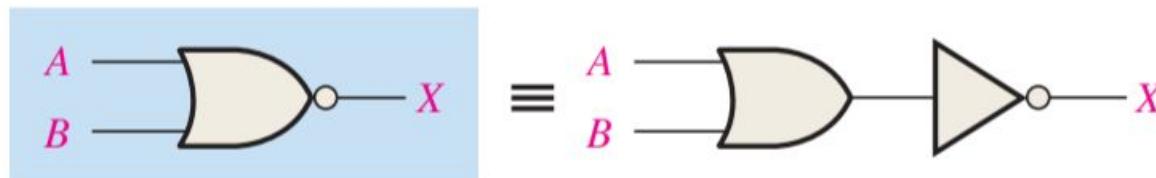
<b>Inputs</b>		<b>Output</b>
<b>A</b>	<b>B</b>	<b>X</b>
0	0	1
0	1	1
1	0	1
1	1	0

1 = HIGH, 0 = LOW.

# Example NAND Gate



# NOR Gate



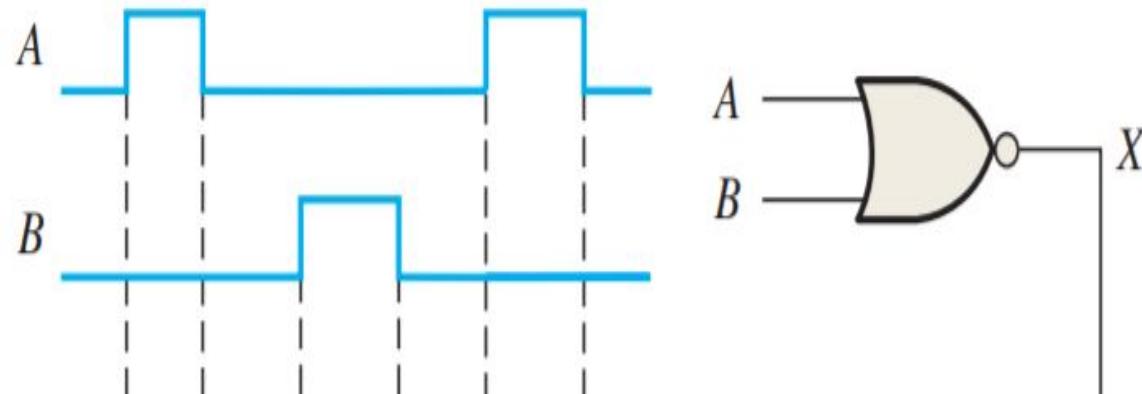
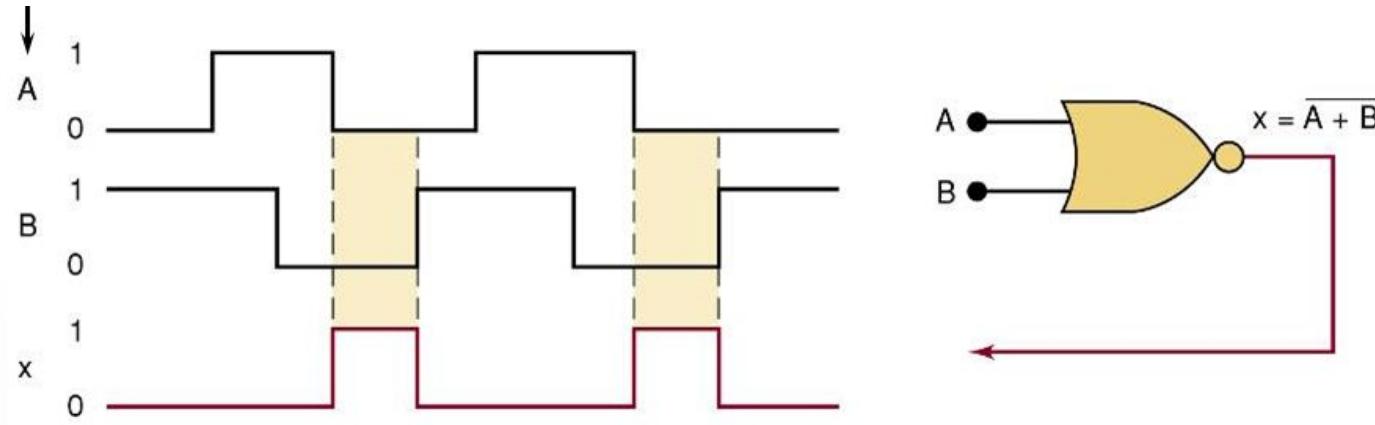
Logic Expressions for a NOR Gate: ->  $X = \overline{A + B}$

Truth table for a 2-input NOR gate.

Inputs		Output	A	B	$\overline{A + B} = X$
A	B	X			
0	0	1	0	0	$\overline{0 + 0} = \overline{0} = 1$
0	1	0	0	1	$\overline{0 + 1} = \overline{1} = 0$
1	0	0	1	0	$\overline{1 + 0} = \overline{1} = 0$
1	1	0	1	1	$\overline{1 + 1} = \overline{1} = 0$

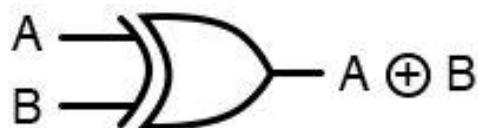
1 = HIGH, 0 = LOW.

# Example NOR gate

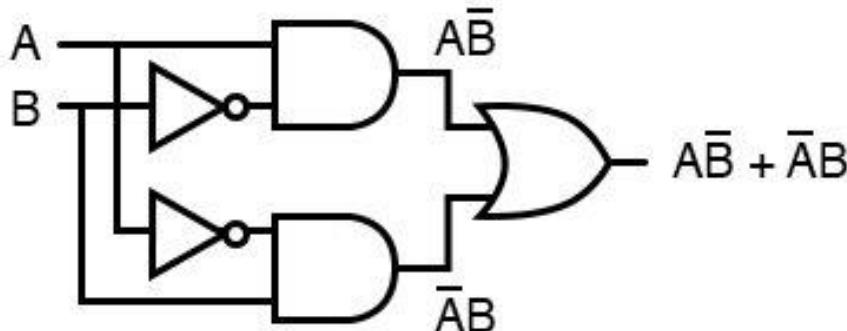


# EX-OR Gate

There are two special gates, i.e., Ex-OR and Ex-NOR. These gates are not basic gates in their own and are constructed by combining with other logic gates.



... is equivalent to . . .



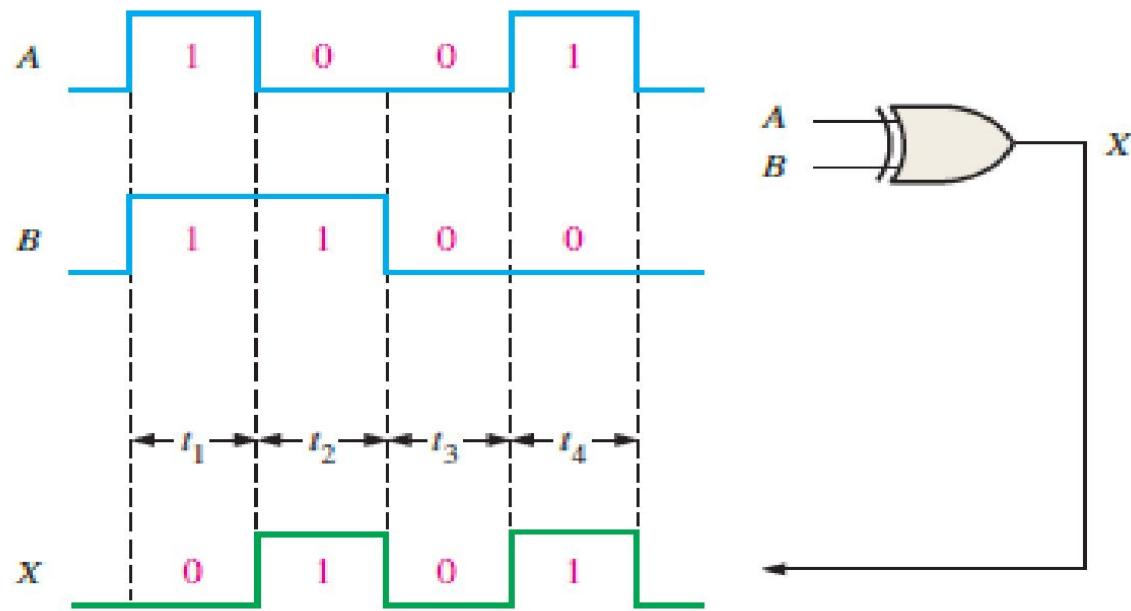
Truth  
Table

INPUTS		OUTPUTS
A	B	$Y = A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

Logic Expressions for a XOR Gate:

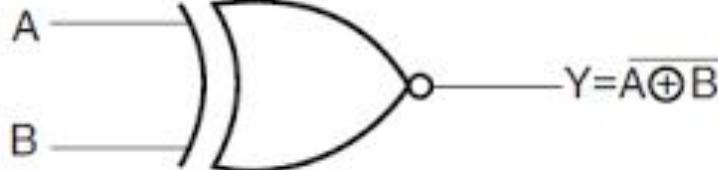
$$A \oplus B = AB + \bar{A}\bar{B}$$

# EX-OR Gate



Example of exclusive-OR gate operation with pulse waveform inputs.

# EX-NOR Gate

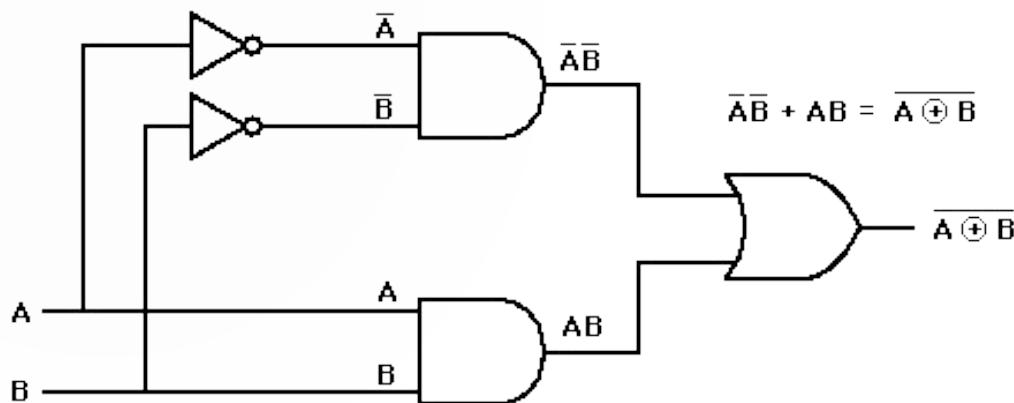


$$Y = (\overline{A \oplus B}) = (A \cdot B + \overline{A} \cdot \overline{B})$$

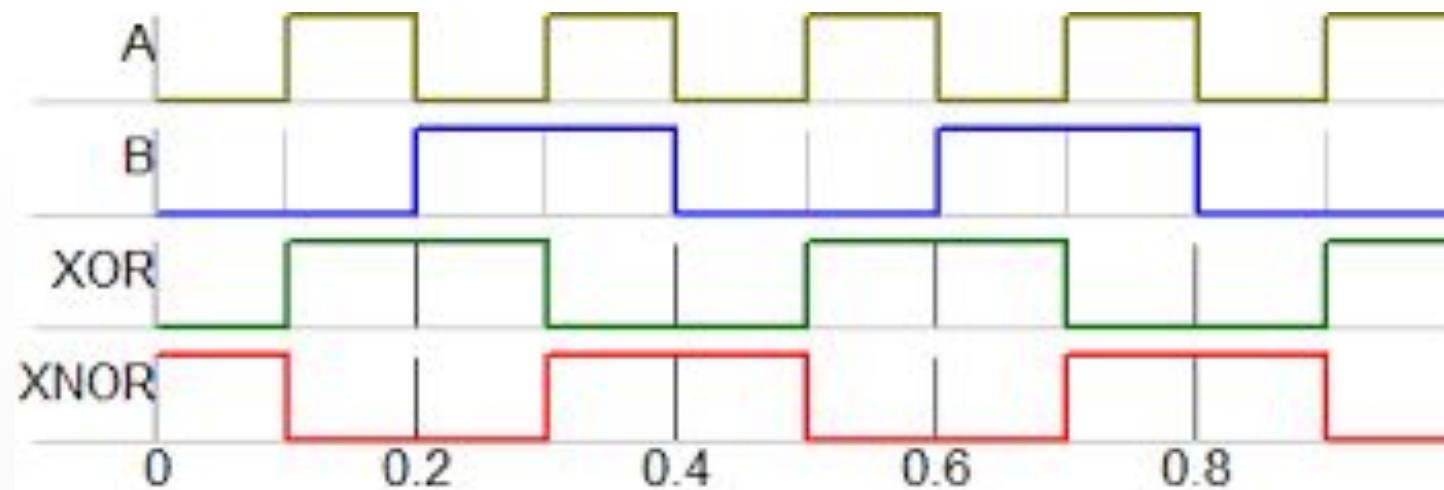
Truth

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1

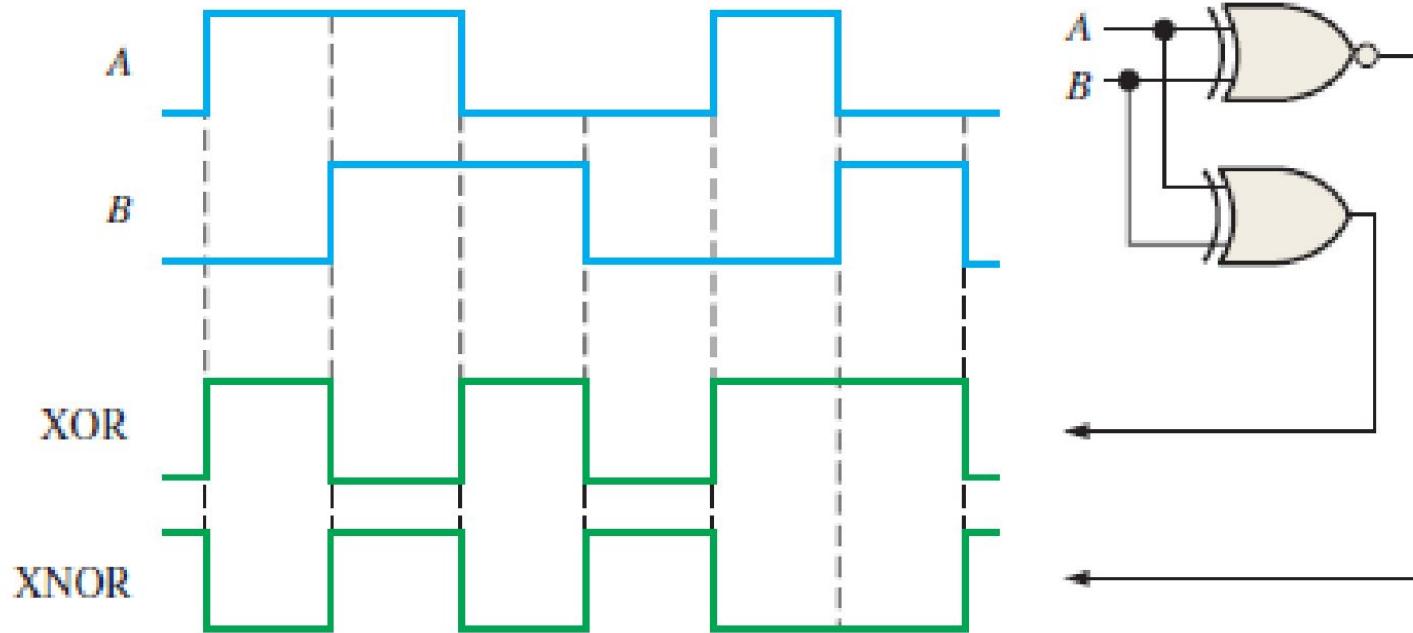
Logic Expressions for a XOR Gate:



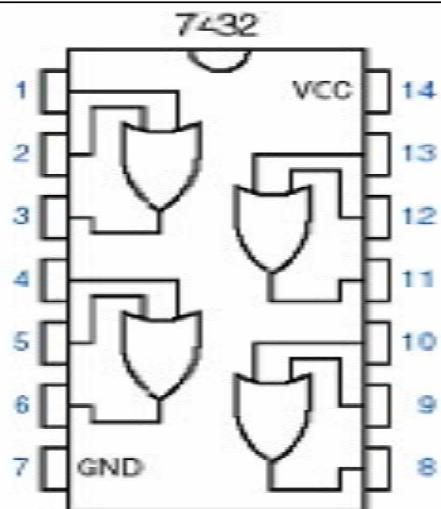
# EX-NOR Gate



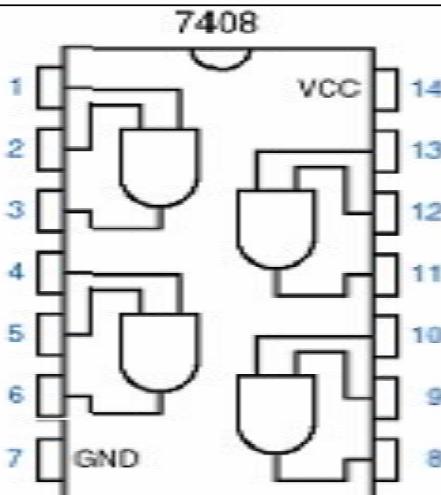
Determine the output waveform for XOR and XNOR gate for given input waveforms A and B



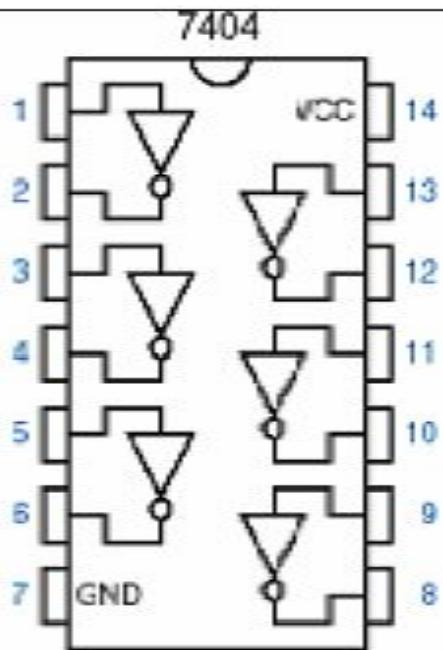
## Pin Diagram



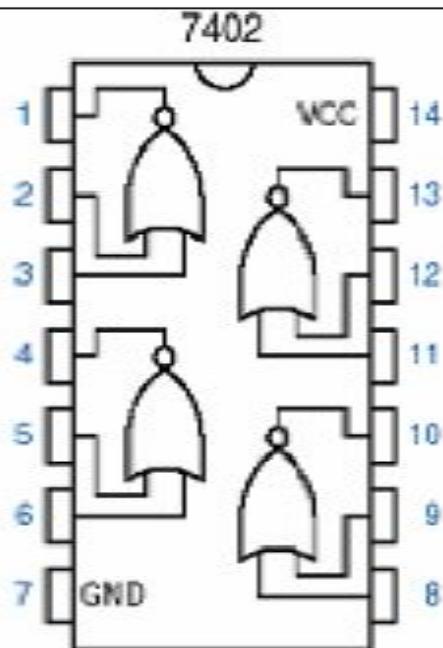
OR Gate



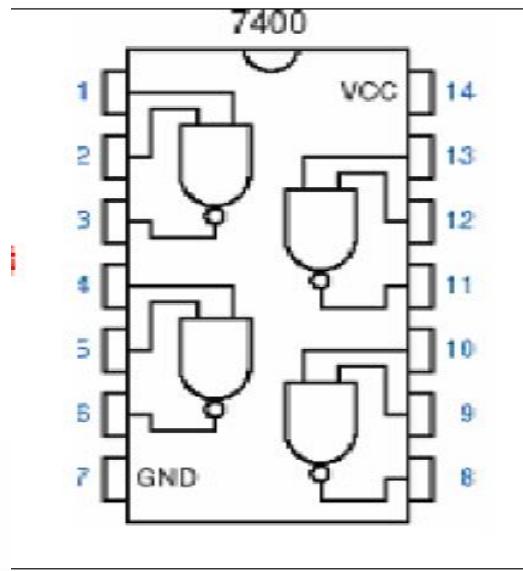
AND Gate



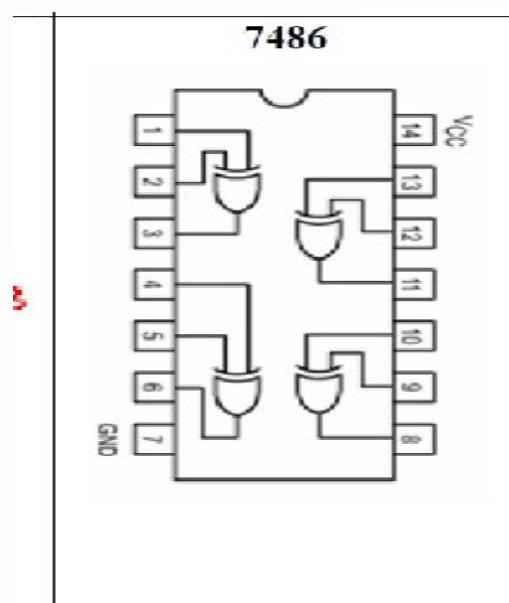
**NOT Gate**



**NOR Gate**



NAND Gate



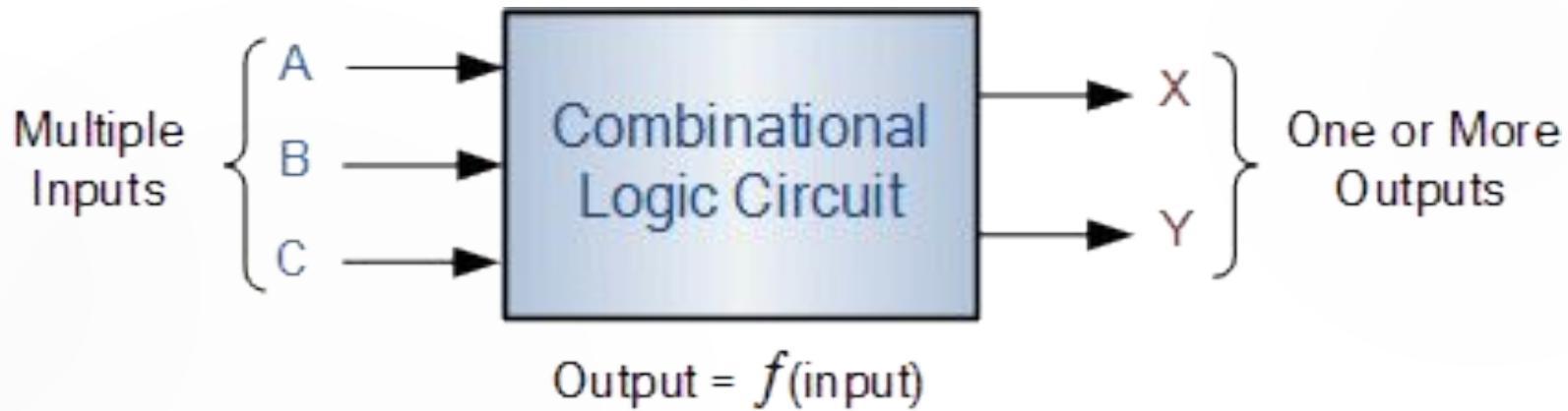
XOR Gate

# Digital Circuits

Basically, Digital Circuits are divided onto two broad categories

- ✓ Combinational circuits
  - **Combinational Circuit** is the type of **circuit** in which output is depend upon the input present at that particular instant.
  
- ✓ Sequential circuits
  - **Sequential circuit** is the type of **circuit** where output at any instant of time depend upon the current input as well as on the previous input/output.
  - It consists of memory element

# Combinational Logic Circuit



# Combinational Logic Circuits

- Combinational logic circuits have no feedback, and any changes to the signals being applied to their inputs will immediately have an effect at the output.
- It has no “memory”, “timing” or “feedback loops”.
- The three main ways of specifying the function of a combinational logic circuit are:
  - Boolean Expression – This forms the algebraic expression showing the operation of the logic circuit
  - Truth Table – Shows all the output states in tabular form for each possible combination of input variable
  - Logic Diagram – This is a graphical representation of a logic circuit

# Combinational Logic Circuit

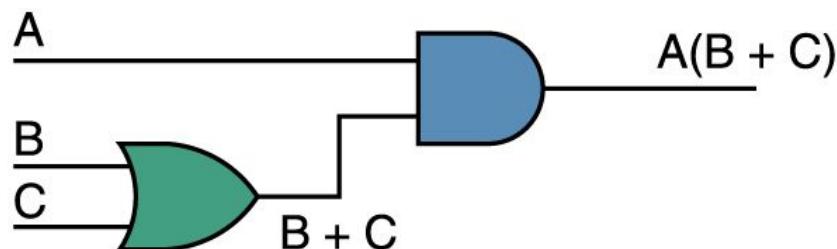
Boolean expression

$$A(B + C)$$

Truth Table

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

Logic Diagram



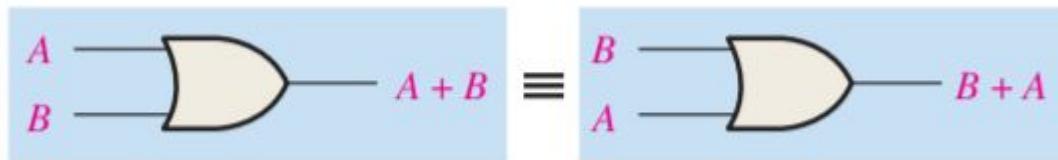
# Boolean Algebra

Boolean algebra is the mathematics of digital logic.

## Commutative Laws :

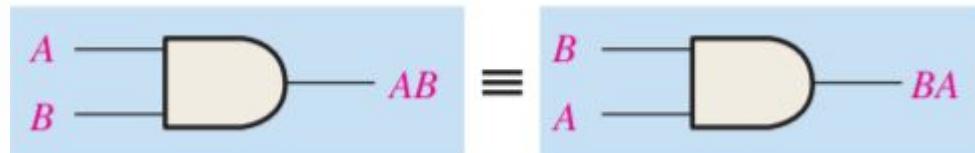
- The commutative law of addition for two variables is written as

$$A + B = B + A$$



- The commutative law of multiplication for two variables is

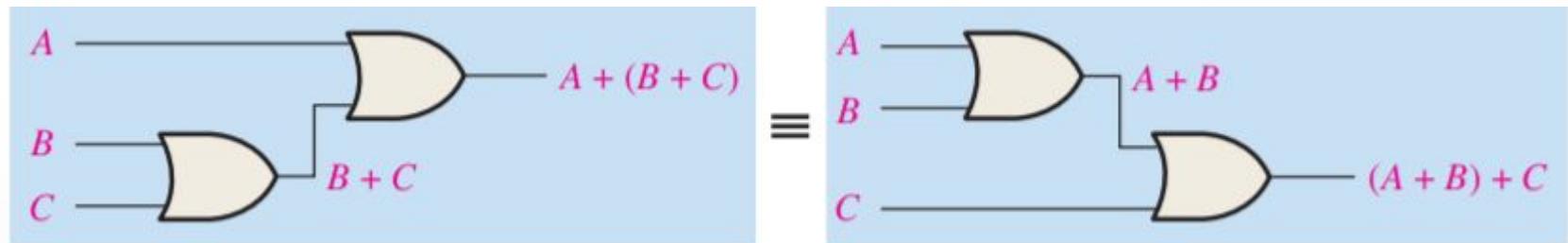
$$AB = BA$$



# Associative Laws

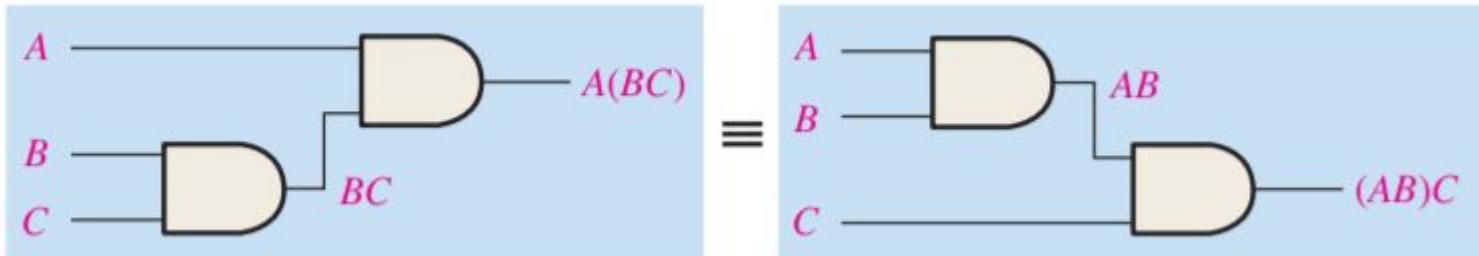
- The associative law of addition is written as follows for three variables:

$$A + (B + C) = (A + B) + C$$



- The associative law of multiplication is written as follows for three variables

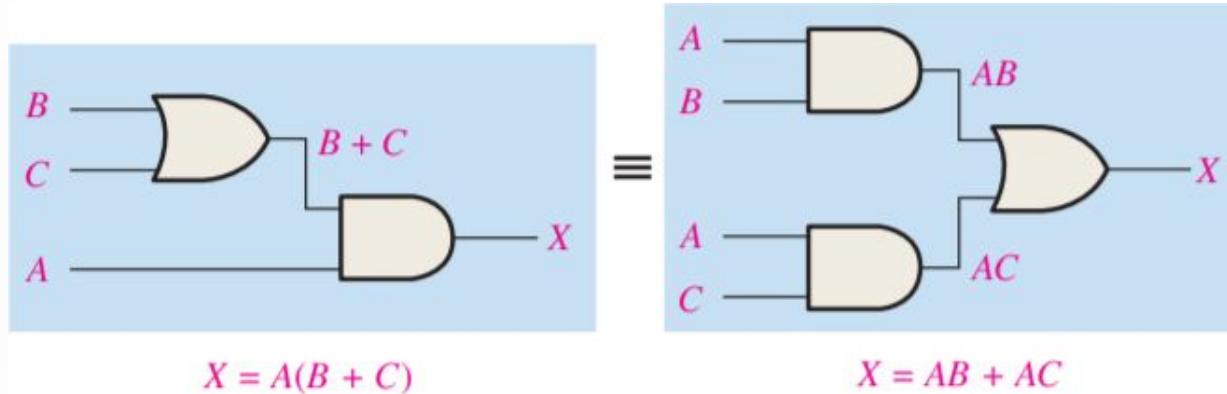
$$A(BC) = (AB)C$$



# Distributive Law

- The distributive law is written for three variables as follows:

$$A(B + C) = AB + AC$$



$$A(B + C) = A \cdot B + A \cdot C$$

(OR Distributive Law)

$$A + (B \cdot C) = (A + B) \cdot (A + C)$$

(AND Distributive Law)

# Basic rules of Boolean algebra.

## Basic rules of Boolean algebra.

$$1. A + 0 = A$$

$$2. A + 1 = 1$$

$$3. A \cdot 0 = 0$$

$$4. A \cdot 1 = A$$

$$5. A + A = A$$

$$6. A + \bar{A} = 1$$

$$7. A \cdot A = A$$

$$8. A \cdot \bar{A} = 0$$

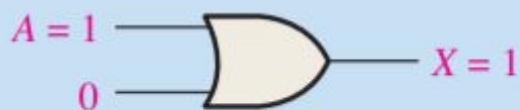
$$9. \bar{\bar{A}} = A$$

$$10. A + AB = A$$

$$11. A + \bar{A}B = A + B$$

$$12. (A + B)(A + C) = A + BC$$

$A, B$ , or  $C$  can represent a single variable or a combination of variables.



Rule  
1

$$X = A + 0 = A$$



Rule  
2

$$X = A + 1 = 1$$



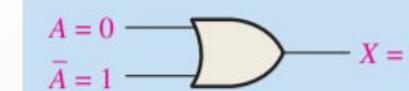
$$X = A \cdot 0 = 0$$



$$X = A \cdot 1 = A$$



$$X = A + A = A$$



$$X = A + \bar{A} = 1$$



$$X = A \cdot A = A$$



$$X = A \cdot \bar{A} = 0$$

Rule  
3

Rule  
4

Rule  
5

Rule  
6

Rule 7

Rule  
8



Rule  
9

**Rule 10:  $A + AB = A$**  This rule can be proved by applying the distributive law, rule 2, and rule 4 as follows:

$$\begin{aligned}
 A + AB &= A \cdot 1 + AB = A(1 + B) && \text{Factoring (distributive law)} \\
 &= A \cdot 1 && \text{Rule 2: } (1 + B) = 1 \\
 &= A && \text{Rule 4: } A \cdot 1 = A
 \end{aligned}$$

Rule 10:  $A + AB = A$ . Open file T04-02 to verify.

<b>A</b>	<b>B</b>	<b>AB</b>	<b><math>A + AB</math></b>
0	0	0	0
0	1	0	0
1	0	0	1
1	1	1	1

↑                                   ↑

equal

A — straight connection

**Rule 11:  $A + \bar{A}B = A + B$**  This rule can be proved as follows:

$$\begin{aligned}A + \bar{A}B &= (A + \bar{A})(A + B) \\&= 1 \cdot (A + B) \\&= A + B\end{aligned}$$

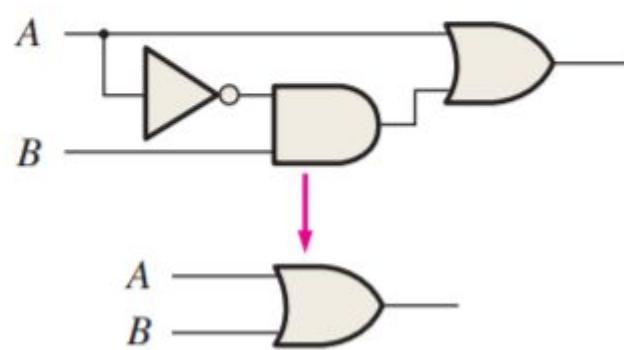
Factoring

Rule 6:  $A + \bar{A} = 1$   
Rule 4: drop the 1

Rule 11:  $A + \bar{A}B = A + B$ . Open file T04-03 to verify.

$A$	$B$	$\bar{A}B$	$A + \bar{A}B$	$A + B$
0	0	0	0	0
0	1	1	1	1
1	0	0	1	1
1	1	0	1	1

↑ equal ↑



**Rule 12:  $(A + B)(A + C) = A + BC$**  This rule can be proved as follows:

$$\begin{aligned}
 (A + B)(A + C) &= AA + AC + AB + BC && \text{Distributive law} \\
 &= A + AC + AB + BC && \text{Rule 7: } AA = A \\
 &= A(1 + C) + AB + BC && \text{Factoring (distributive law)} \\
 &= A \cdot 1 + AB + BC && \text{Rule 2: } 1 + C = 1 \\
 &= A(1 + B) + BC && \text{Factoring (distributive law)} \\
 &= A \cdot 1 + BC && \text{Rule 2: } 1 + B = 1 \\
 &= A + BC && \text{Rule 4: } A \cdot 1 = A
 \end{aligned}$$

Rule 12:  $(A + B)(A + C) = A + BC$ . Open file T04-04 to verify.

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

↑ equal ↑

# DeMorgan's theorem

1. The complement of a product of variables is equal to the sum of the complements of the variables.

OR

The complement of two or more ANDed variables is equivalent to the OR of the complements of the individual variables.

$$\overline{AB} = \overline{A} + \overline{B}$$

2. The complement of a sum of variables is equal to the product of the complements of the variables.

OR

The complement of two or more ORed variables is equivalent to the AND of the complements of the individual variables.

$$\overline{A+B} = \overline{A} \cdot \overline{B}$$

# De Morgan's Theorem 1

$$\overline{A \cdot B} = \overline{A} + \overline{B}$$

B

A	B	$A \cdot B$	$\overline{A \cdot B}$		$\overline{A}$	$\overline{B}$	$\overline{\overline{A} + \overline{B}}$
0	0	0	1		1	1	1
0	1	0	1		1	0	1
1	0	0	1		0	1	1
1	1	1	0		0	0	0

EQUAL

**NAND = Bubbled OR**

# De Morgan's Theorem 2

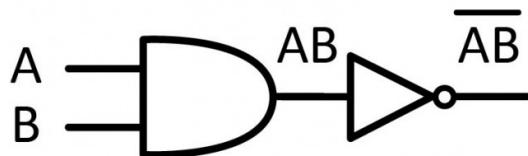
$$\overline{A + B} = \overline{A} \cdot \overline{B}$$

A	B	A + B	$\overline{A + B}$		$\overline{A}$	$\overline{B}$	$\overline{A} \cdot \overline{B}$
0	0	0	1		1	1	1
0	1	1	0		1	0	0
1	0	1	0		0	1	0
1	1	1	0		0	0	0

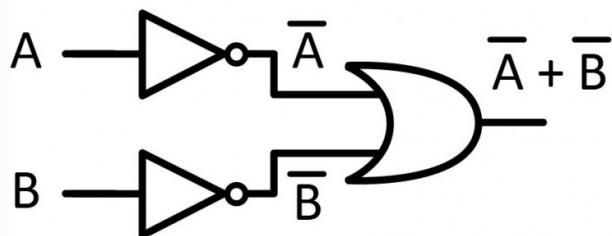
EQUAL

**NOR = Bubbled AND**

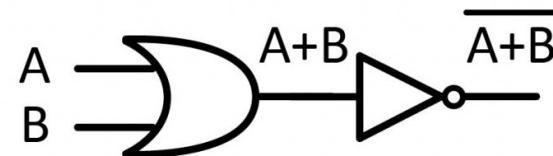
# De Morgan's Law



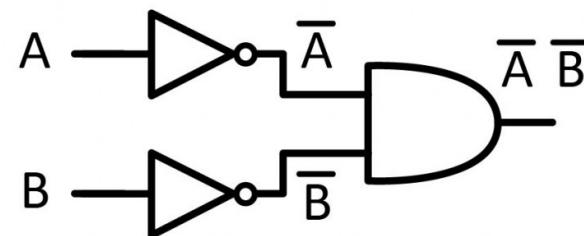
is equivalent to



$$\overline{AB} = \overline{A} + \overline{B}$$



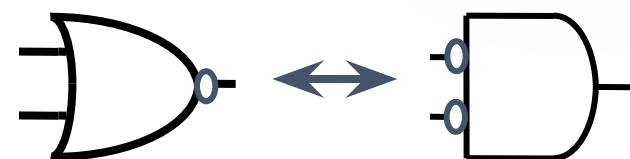
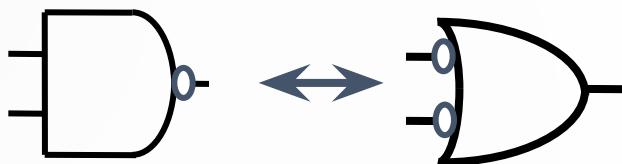
is equivalent to



$$\overline{A+B} = \overline{A} \overline{B}$$

# Summary

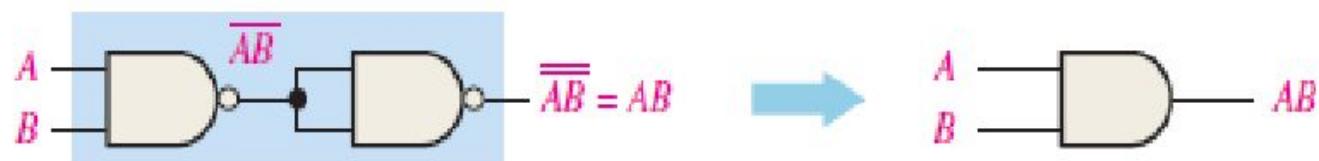
- NAND is the same as OR with complemented inputs
- NOR is the same as AND with complemented inputs



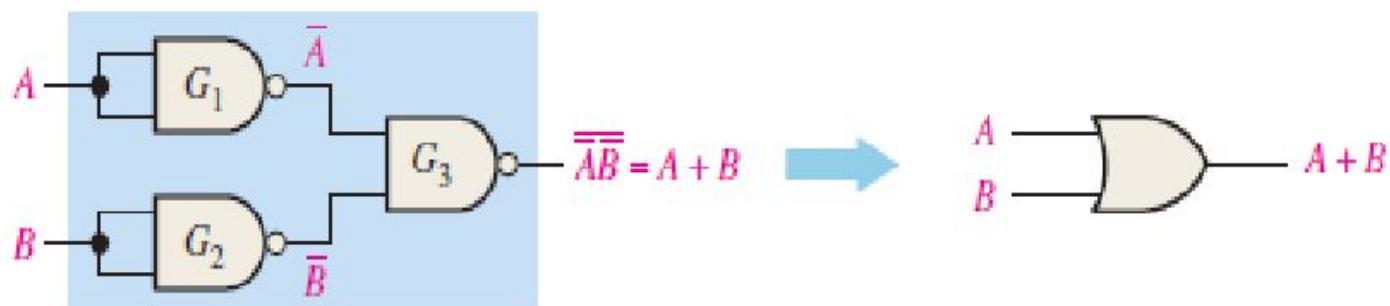
# NAND gate as universal logic gate



(a) One NAND gate used as an inverter



(b) Two NAND gates used as an AND gate

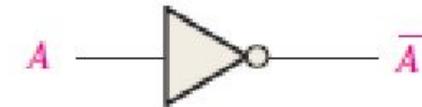
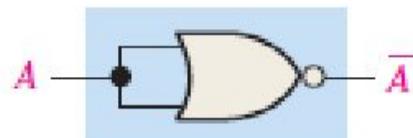


(c) Three NAND gates used as an OR gate

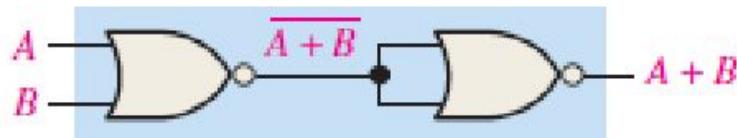
# OR gate using NAND gate

$$A + B = \overline{\overline{A} + \overline{B}} = \overline{\overline{A} \cdot \overline{B}}$$

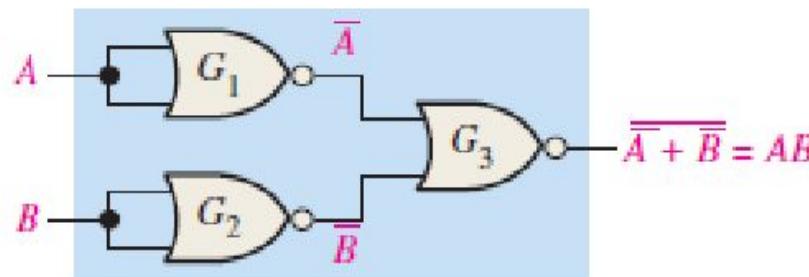
# NOR gate as universal logic gate



(a) One NOR gate used as an inverter



(b) Two NOR gates used as an OR gate



(c) Three NOR gates used as an AND gate

# Exercise

Apply DeMorgan's theorem to the expression  $\overline{\overline{X}} + \overline{\overline{Y}} + \overline{\overline{Z}}$ .

$$= X \cdot Y \cdot Z$$

Apply DeMorgan's theorem to the expression  $\overline{\overline{W}}\overline{\overline{X}}\overline{\overline{Y}}\overline{\overline{Z}}$ .

$$= W + X + Y + Z$$

Apply DeMorgan's theorems to each of the following expressions:

(a)  $\overline{(A + B + C)D} = \overline{(A + B + C)} + \overline{D} = \overline{A} \overline{B} \overline{C} + \overline{D}$

(b)  $\overline{ABC + DEF} = \overline{A} \overline{B} \overline{C} \cdot \overline{D} \overline{E} \overline{F} = (\overline{A} + \overline{B} + \overline{C}) \cdot (\overline{D} + \overline{E} + \overline{F})$

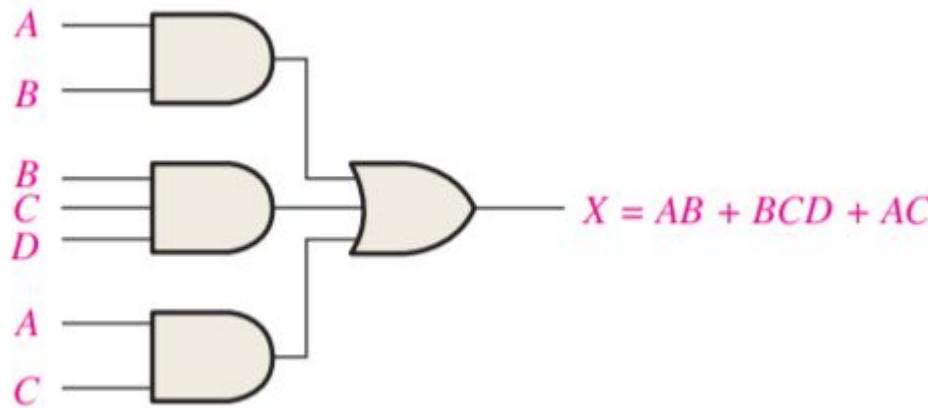
(c)  $\overline{\overline{AB} + \overline{CD} + EF}$

# Standard Forms of Boolean Expressions

1. Sum-of-products form (SOP)
2. Product-of-sums form (POS)

## The Sum-of-Products (SOP) Form:

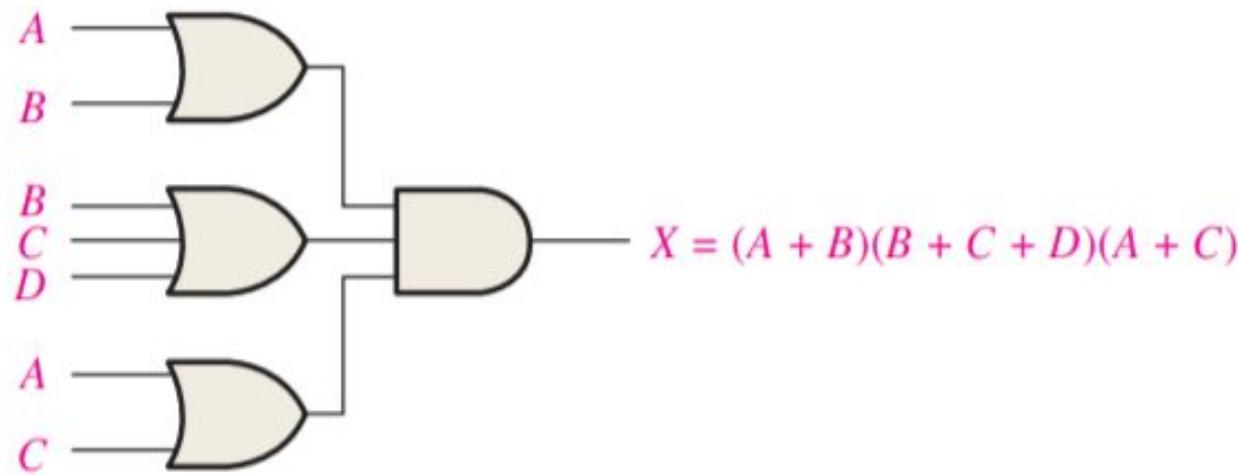
- An SOP expression can be implemented with one OR gate and two or more AND gates.



Implementation of the SOP expression  $AB + BCD + AC$ .

# The Product-of-Sums (POS) Form

- When two or more sum terms are multiplied, the resulting expression is a product-of-sums (POS).
- Implementing the **POS** expression simply requires **AND** ing the outputs of two or more **OR** gates

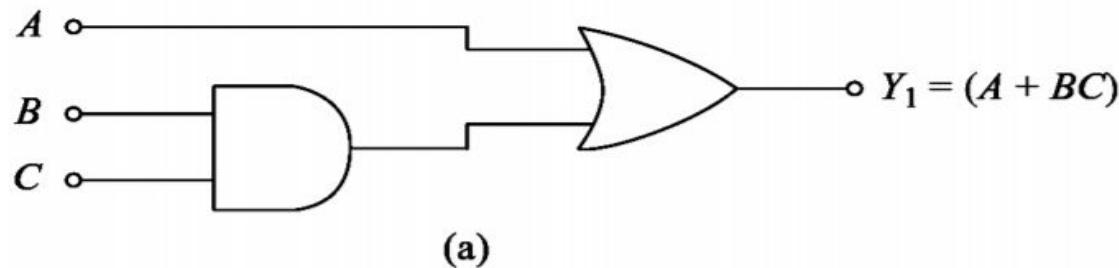


Implementation of the POS expression  $(A + B)(B + C + D)(A + C)$ .

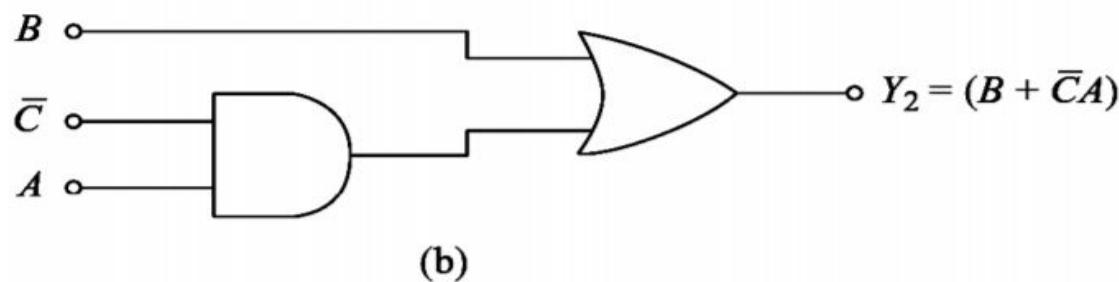
# Combinational Circuit Realization

$$Y = (A + BC)(B + \bar{C}A)$$

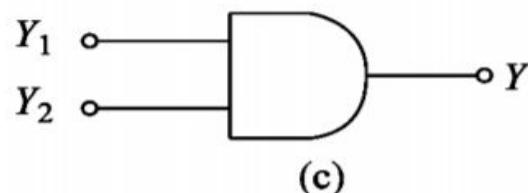
Realization of logic expression (3 level ):



(a)



(b)



(c)

# Combinational Circuit realization (SOP)

$$Y = (A + BC)(B + \bar{C}A)$$

SOP form:

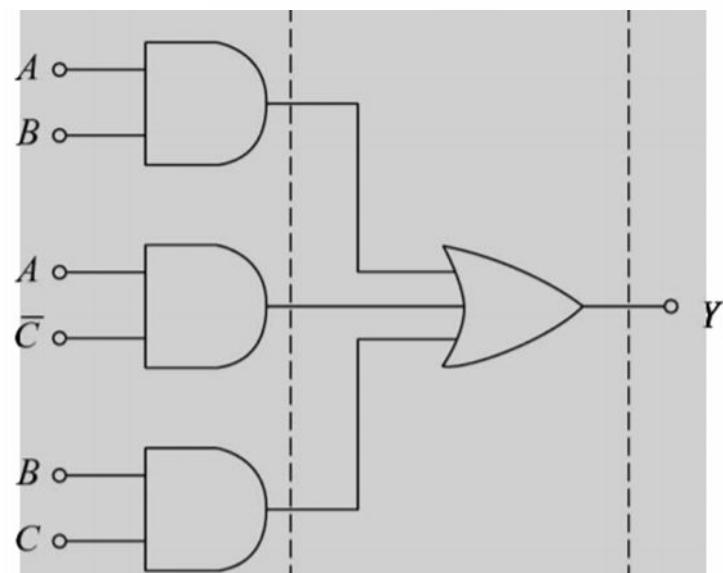
$$\begin{aligned} Y &= A(B + \bar{C}A) + (BC)(B + \bar{C}A) \\ &= AB + A\bar{C}A + BCB + BC\bar{C}A \\ &= AB + A\bar{C} + BC \end{aligned}$$

$$A\bar{C}A = (A \cdot A) \cdot \bar{C} = A\bar{C} \quad (\text{Theorem 1.6})$$

$$BCB = (B \cdot B) \cdot C = BC \quad (\text{Theorem 1.6})$$

$$BC\bar{C}A = B \cdot (C \cdot \bar{C}) \cdot A = B \cdot 0 \cdot A \quad (\text{Theorem 1.8})$$

$$= 0 \quad (\text{Theorem 1.4})$$



**2 level realization**

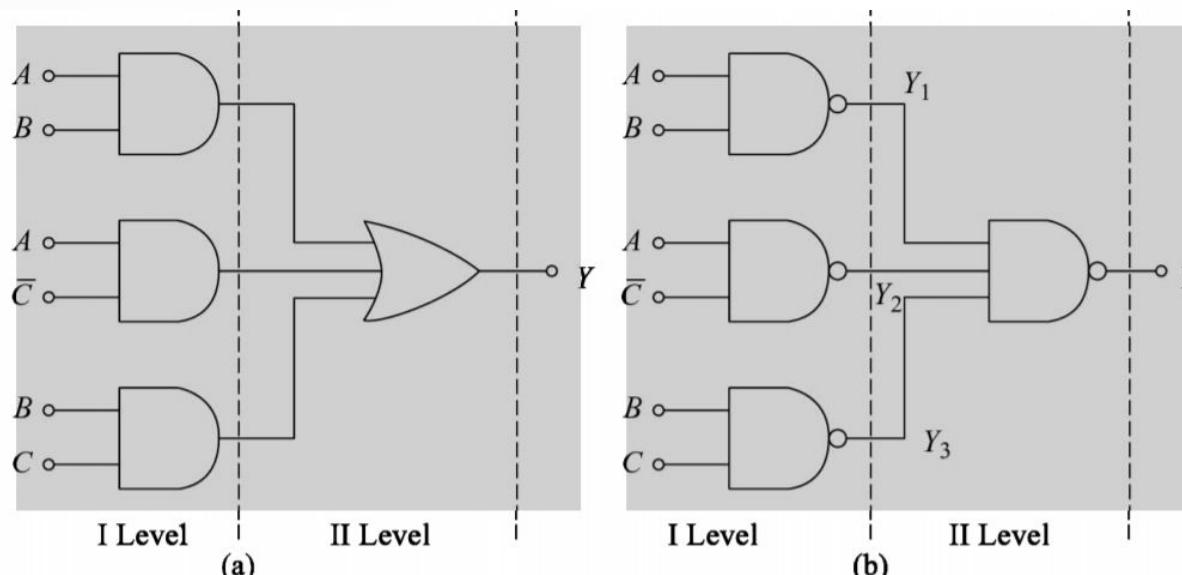
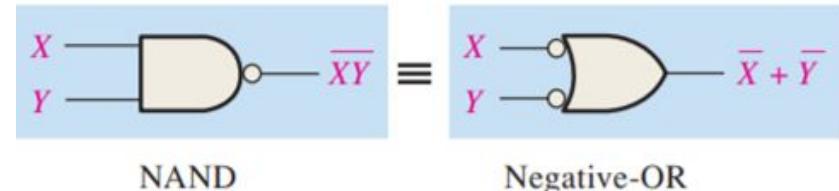
# SOP realization using only one type of gate(NAND)

$$\begin{aligned}\overline{\overline{Y}} &= \overline{\overline{AB + A\bar{C} + BC}} \\ &= \overline{\overline{AB} \cdot \overline{A\bar{C}} \cdot \overline{BC}} \\ Y &= \overline{Y_1 \cdot Y_2 \cdot Y_3}\end{aligned}$$

$$Y_1 = \overline{AB}$$

$$Y_2 = \overline{A\bar{C}}$$

$$Y_3 = \overline{BC}$$

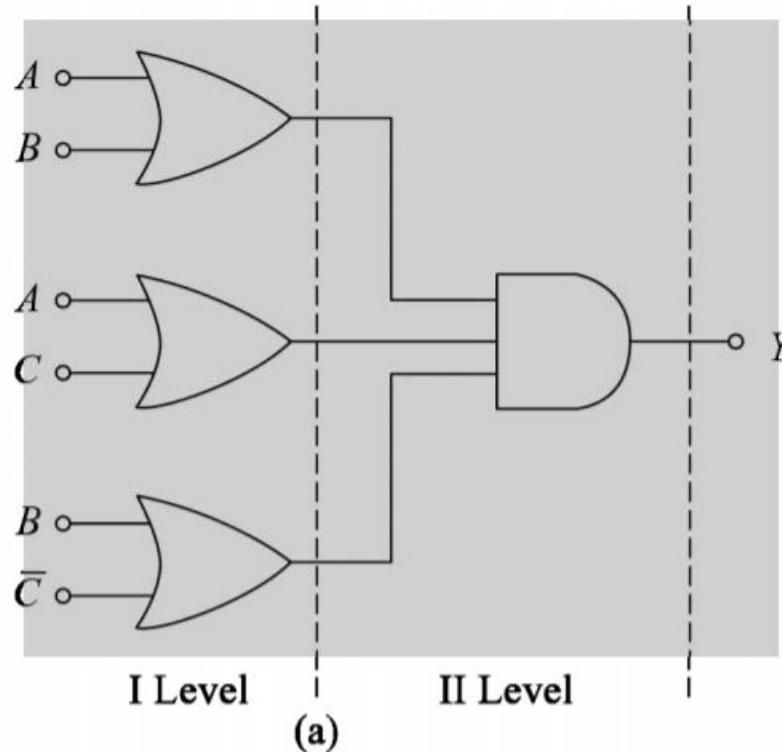


SOP realized using NAND gates only

# Combinational Circuit Realization(POS)

$$Y = (A + BC)(B + \bar{C}A)$$

$$\begin{aligned} Y &= (A + B)(A + C)(B + \bar{C})(B + A) && \text{(Theorem 1.10)} \\ &= (A + B)(A + C)(B + \bar{C}) && \text{(Theorem 1.6)} \end{aligned}$$



# POS realization using only one type of gate(NOR)

$$\begin{aligned}\bar{\bar{Y}} &= \overline{\overline{(A+B)(A+C)(B+\bar{C})}} \\ &= \overline{\overline{(A+B)} + \overline{(A+C)} + \overline{(B+\bar{C})}}\end{aligned}$$

$$Y = \overline{Y_A + Y_B + Y_C}$$

$$Y_A = \overline{A+B}$$

$$Y_B = \overline{A+C}$$

$$Y_C = \overline{B+\bar{C}}$$

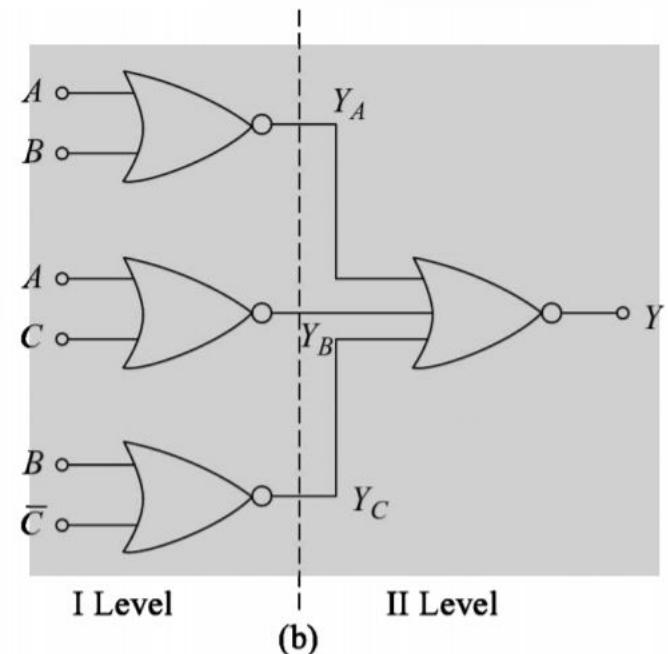
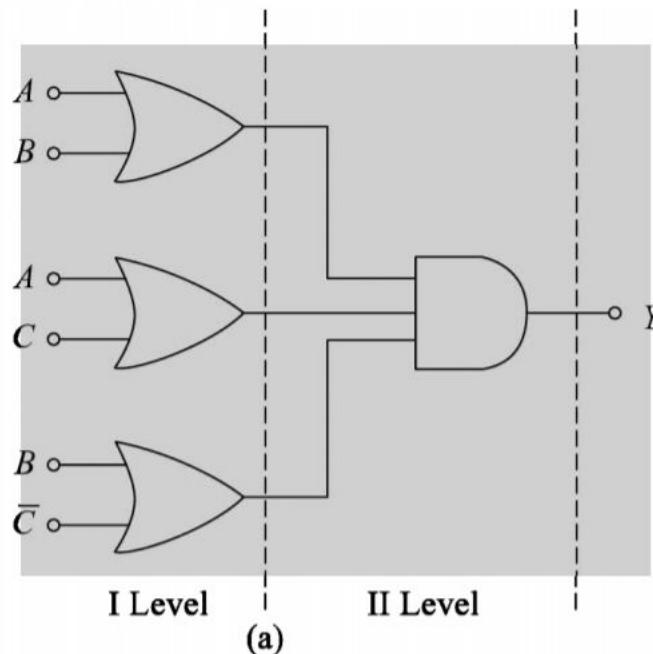
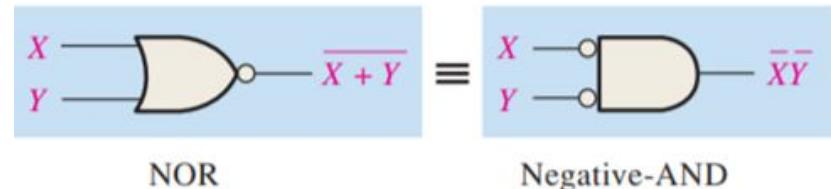


Fig. 5.3

*Realisation of Eq. (5.5) Using (a) OR-AND (b) NOR-NOR*

# Construct SOP from a Truth Table

A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

SOP Implementation  
from a  
Truth Table

$F = A\bar{B}C + \bar{A}BC + \bar{A}\bar{B}C$

# Construct POS from a Truth Table

A	B	C	F	POS implementation from a Truth Table
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	1	1
1	0	0	0	0
1	0	1	1	1
1	1	0	0	0
1	1	1	0	0

$$F = (\bar{A} + \bar{B} + \bar{C}) (\bar{A} + \bar{B} + C) (\bar{A} + B + C) (A + \bar{B} + C) (A + B + C)$$

# Canonical SOP and POS

- Each individual term in canonical SOP form is called as **minterm** and in canonical POS form as **maxterm**

Minterms			Maxterms			
X	Y	Z	Term	Designation	Term	Designation
0	0	0	$\bar{X}\bar{Y}\bar{Z}$	m0	$X+Y+Z$	Mo
0	0	1	$\bar{X}\bar{Y}Z$	m1	$X+Y+\bar{Z}$	M1
0	1	0	$\bar{X}YZ$	m2	$X+\bar{Y}+Z$	M2
0	1	1	$\bar{X}Y\bar{Z}$	m3	$X+\bar{Y}+\bar{Z}$	M3
1	0	0	$X\bar{Y}\bar{Z}$	m4	$\bar{X}+Y+Z$	M4
1	0	1	$X\bar{Y}Z$	m5	$\bar{X}+Y+\bar{Z}$	M5
1	1	0	$XY\bar{Z}$	m6	$\bar{X}+\bar{Y}+Z$	M6
1	1	1	$XYZ$	m7	$\bar{X}+\bar{Y}+\bar{Z}$	M7

# Shorthand form of canonical SOP

A	B	C	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0

SOP Implementation  
from a  
Truth Table

$$F = A\bar{B}C + \bar{A}BC + \bar{A}\bar{B}C$$

$$F = m1 + m3 + m5$$

$$F = \Sigma m (1, 3, 5)$$

# Shorthand form of canonical POS

A	B	C	F	POS implementation from a Truth Table
0	0	0	0	
0	0	1	1	
0	1	0	0	
0	1	1	1	
1	0	0	0	
1	0	1	1	
1	1	0	0	
1	1	1	0	

$$F = (\bar{A} + \bar{B} + \bar{C}) (\bar{A} + \bar{B} + C) (\bar{A} + B + C) (A + \bar{B} + C) (A + B + C)$$

$$F = M_0 \cdot M_2 \cdot M_4 \cdot M_6 \cdot M_7$$

$$F = \prod M (0, 2, 4, 6, 7)$$

# The Karnaugh Map( K map)

- A Karnaugh map provides a systematic method for simplifying Boolean expressions
- K-map produce the simplest SOP or POS expression possible, known as the minimum expression.
- A Karnaugh map is **similar to a truth table** because it presents all of the possible values of input variables and the resulting output for each value.
- Karnaugh map is an **array of cells** in which each cell represents a binary value of the input variables.
- Karnaugh maps can be used for **expressions with two, three, four, and five variables,**

# K-maps (contd...)

	A	0	1
B	0	2	
0	1	3	

(a)

	AB	00	01	11	10
C	0	0	2	6	4
1	1	3	7	5	

(b)

	CD	AB	00	01	11	10
00	0	4		12	8	
01	1	5		13	9	
11	3	7		15	11	
10	2	6		14	10	

(c)

Fig. 5.5

*Karnaugh-maps (a) Two-variable  
(b) Three-variable (c) Four-variable*

- K-maps adjacencies wrap around edges
  - Wrap from first to last column
  - Wrap top row to bottom row
- Numbering scheme is based on Gray-code (only a single bit changes in code for adjacent map cells)
- K-maps are hard to draw and visualize for more than 4 dimensions, and virtually impossible for more than 6 dimensions

A	B	C	D
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1
1	0	0	0
1	0	0	1
1	0	1	0
1	0	1	1
1	1	0	0
1	1	0	1
1	1	1	0
1	1	1	1

# K-maps (contd...)

	A	0	1
B	0	$\bar{A}\bar{B}$	$A\bar{B}$
1	$\bar{A}B$	$AB$	

(a)

	A	0	1
B	0	$A + B$	$\bar{A} + B$
1	$A + \bar{B}$	$\bar{A} + \bar{B}$	

(b)

	AB	00	01	11	10
C	0	$\bar{A}\bar{B}\bar{C}$	$\bar{A}B\bar{C}$	$A\bar{B}\bar{C}$	$A\bar{B}\bar{C}$
1	$\bar{A}\bar{B}C$	$\bar{A}BC$	$ABC$	$A\bar{B}C$	

(c)

	AB	00	01	11	10
C	0	$A + B + C$	$A + \bar{B} + C$	$\bar{A} + \bar{B} + C$	$\bar{A} + B + C$
1	$A + B + \bar{C}$	$A + \bar{B} + \bar{C}$	$\bar{A} + \bar{B} + \bar{C}$	$\bar{A} + B + \bar{C}$	

	AB	00	01	11	10
CD	00	$\bar{A}\bar{B}\bar{C}\bar{D}$	$\bar{A}\bar{B}\bar{C}\bar{D}$	$A\bar{B}\bar{C}\bar{D}$	$A\bar{B}\bar{C}\bar{D}$
01	$\bar{A}\bar{B}\bar{C}D$	$\bar{A}B\bar{C}D$	$AB\bar{C}D$	$A\bar{B}CD$	
11	$\bar{A}\bar{B}CD$	$\bar{A}BCD$	$ABC\bar{D}$	$A\bar{B}CD$	
10	$\bar{A}\bar{B}C\bar{D}$	$\bar{A}BC\bar{D}$	$ABC\bar{D}$	$A\bar{B}C\bar{D}$	

**Two variable K-map**

**Three variable K-map**

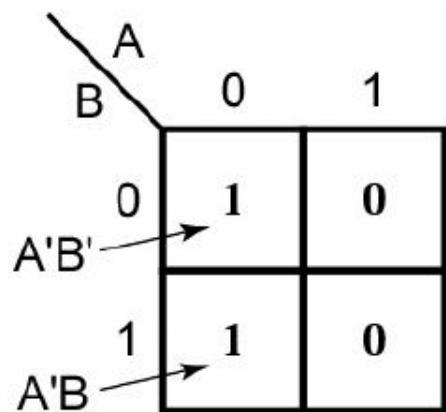
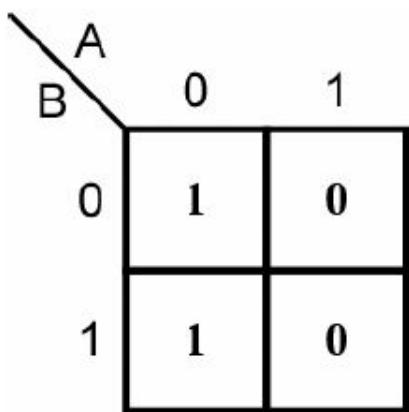
**Four variable K-map**

# Procedure for Obtaining Simplified Boolean equation from K-map

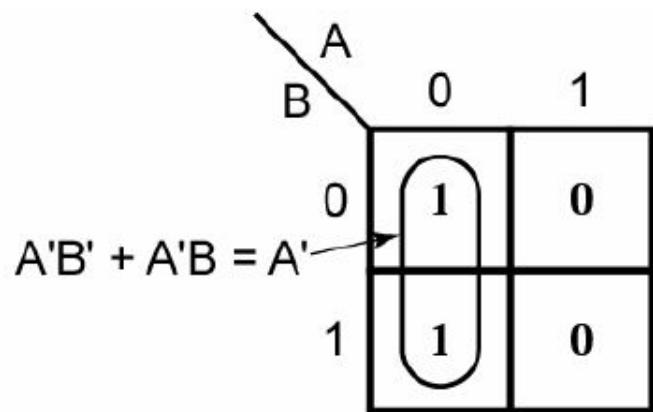
1. Identify adjacent ones,
2. See the values of the variables associated with these cells
3. Only one variable will be different and it gets eliminated
4. Other variables will appear in ANDed form in the term, it will be in the uncomplemented form if it is 1 and in complemented form if it is 0.
5. Determine the term corresponding to each group of adjacent ones.
6. These terms are ORed to get the simplified equation in SOP form.

# K-map examples (2 variable)

A	B	F
0	0	1
0	1	1
1	0	0
1	1	0



$$F = A'B' + A'B$$

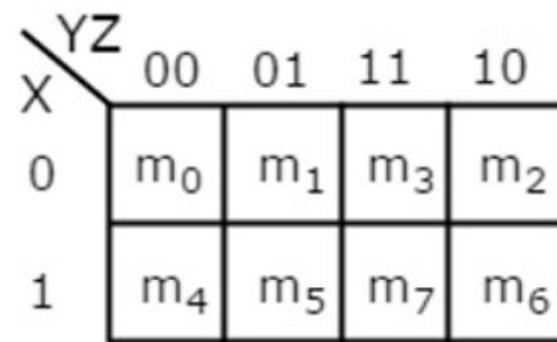


$$F = A'$$

# K-map examples (3 variable)

x	y	z	F
0	0	0	m <sub>0</sub>
0	0	1	m <sub>1</sub>
0	1	0	m <sub>2</sub>
0	1	1	m <sub>3</sub>
1	0	0	m <sub>4</sub>
1	0	1	m <sub>5</sub>
1	1	0	m <sub>6</sub>
1	1	1	m <sub>7</sub>

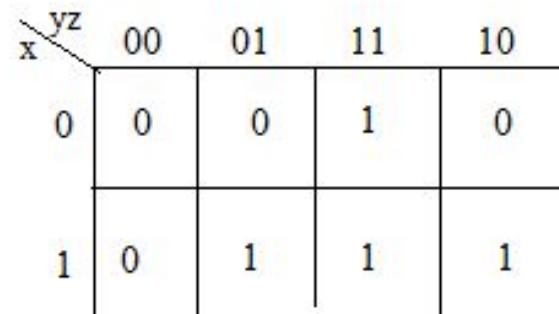
3-variable Truth table of F



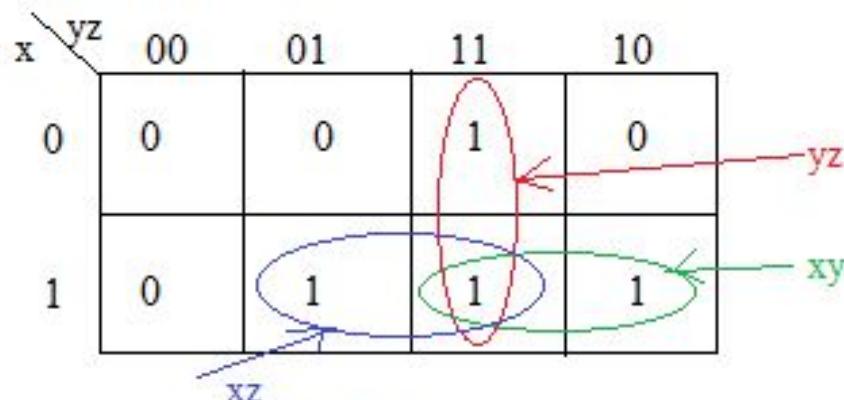
# K-map examples (3 variable)

x	y	z	F	minterm
0	0	0	0	
0	0	1	0	
0	1	0	0	
0	1	1	1	$m_3$
1	0	0	0	
1	0	1	1	$m_5$
1	1	0	1	$m_6$
1	1	1	1	$m_7$

Truth Table of 3-variable Majority Function



K-Map of Majority Function



Pair in K-Map

Solution is :  $Y = \text{yz} + \text{xz} + \text{xy}$

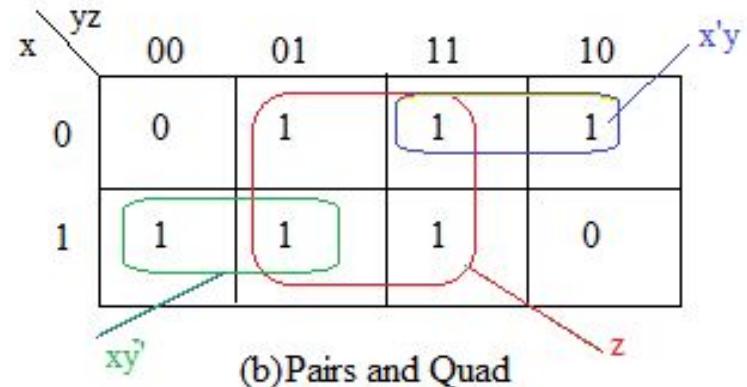
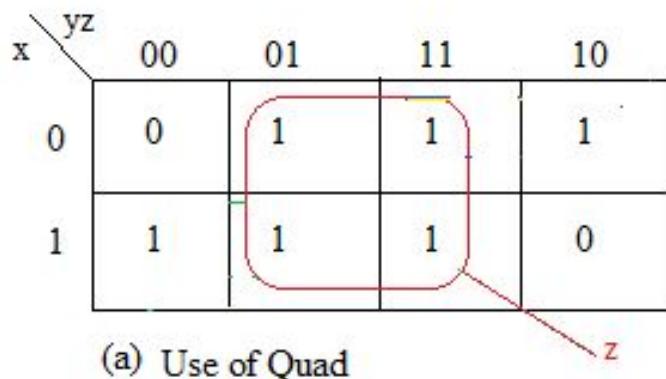
# K-map examples (3 variable)

Example:

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

Solution:

The equation has six minterms. So, enter 1's at appropriate positions in the K-Map



Solution is:  $Y = \textcolor{red}{z} + \textcolor{blue}{x'y} + \textcolor{green}{xy'}$

# K-map examples (3 variable)

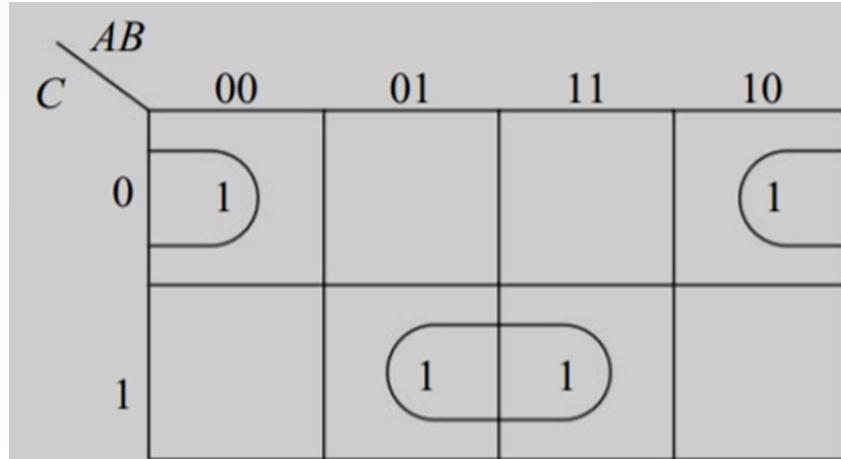
Simplify the K-map of Fig. 5.9.

The canonical SOP form of equation can be written by inspection as

$$Y = \bar{A}\bar{B}\bar{C} + \bar{A}BC + ABC + A\bar{B}\bar{C}$$

$$Y = (\bar{A} + A)\bar{B}\bar{C} + (\bar{A} + A)BC$$

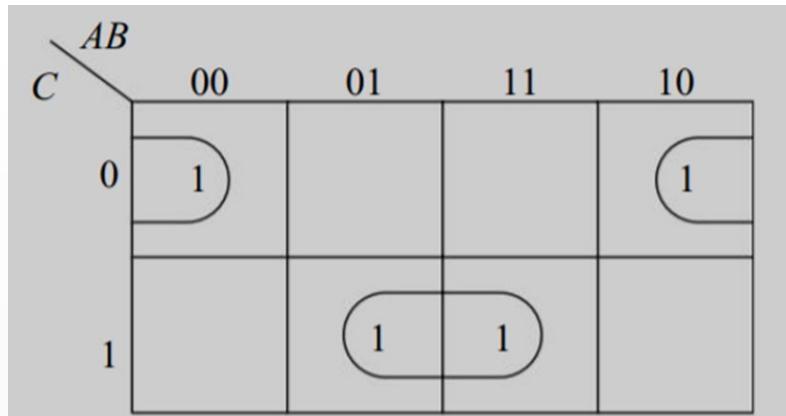
$$= \bar{B}\bar{C} + BC \text{ (Theorems 1.7 and 1.2)}$$



A	B	C	Y		Y	
0	0	0	1	A	B	C
0	0	1	0			
0	1	0	0			
0	1	1	1	A	B	C
1	0	0	1	A	B	C
1	0	1	0			
1	1	0	0			
1	1	1	1	A	B	C

Fig. 5.9

# K-map examples (3 variable)

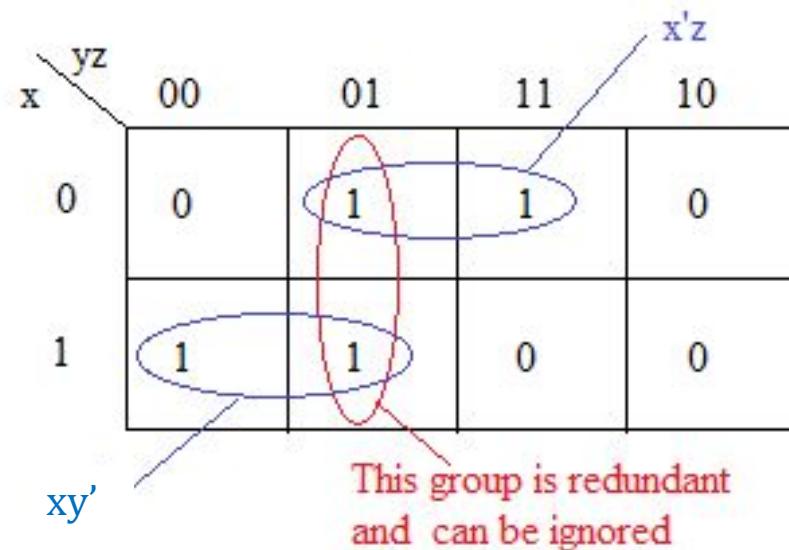


A	B	C	Y	Y
				$\bar{A}$ $\bar{B}$ $\bar{C}$
0	0	0	1	$\bar{A} \bar{B} \bar{C}$
0	0	1	0	$\bar{A} \bar{B} C$
0	1	0	0	$\bar{A} B \bar{C}$
0	1	1	1	$\bar{A} B C$
1	0	0	1	$A \bar{B} \bar{C}$
1	0	1	0	$A \bar{B} C$
1	1	0	0	$A B \bar{C}$
1	1	1	1	$A B C$

$$= \bar{B}\bar{C} + BC$$

# Redundant Groups

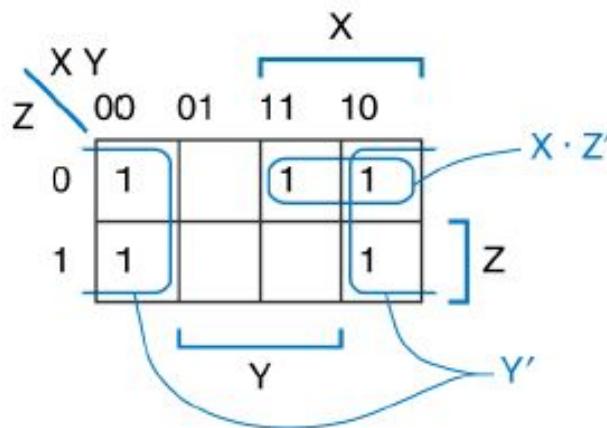
- A redundant group is one whose all the 1's have been consumed by other groups. So, there is no need to form such group. Whenever you see that all 1's of a group have been exhausted, simply ignore that group.



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

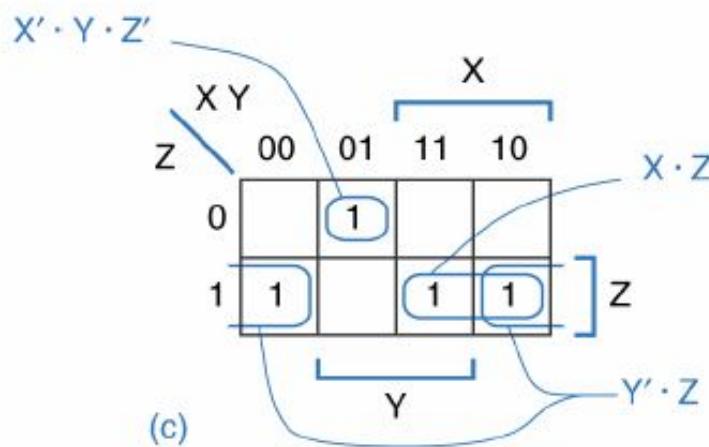
# K-map examples (3 variable)

1.  $F = \Sigma m(0, 1, 4, 5, 6)$



$$F = Y' + X Z'$$

2.  $F = \Sigma m(1, 2, 5, 7)$



$$F = X' Y Z' + X Z + Y' Z$$

# K-map examples (4 variable)

w	x	y	z	F
0	0	0	0	m0
0	0	0	1	m1
0	0	1	0	m2
0	0	1	1	m3
0	1	0	0	m4
0	1	0	1	m5
0	1	1	0	m6
0	1	1	1	m7
1	0	0	0	m8
1	0	0	1	m9
1	0	1	0	m10
1	0	1	1	m11
1	1	0	0	m12
1	1	0	1	m13
1	1	1	0	m14
1	1	1	1	m15

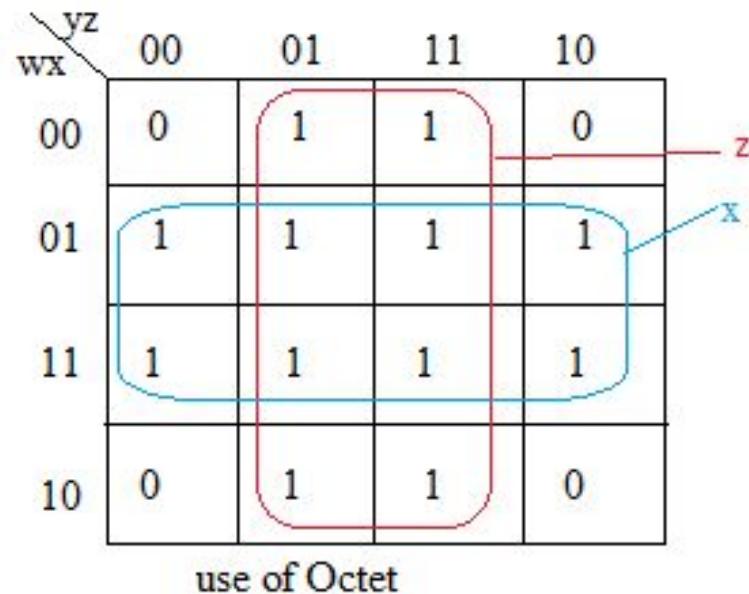
Truth Table 4-variable function

WX \ YZ

00	01	11	10
$m_0$	$m_1$	$m_3$	$m_2$
$m_4$	$m_5$	$m_7$	$m_6$
$m_{12}$	$m_{13}$	$m_{15}$	$m_{14}$
$m_8$	$m_9$	$m_{11}$	$m_{10}$

# K-map examples (4 variable)

$$F(W,X,Y,Z) = \Sigma m(1,3,4,5,6,7,9,11,12,13,14,15)$$

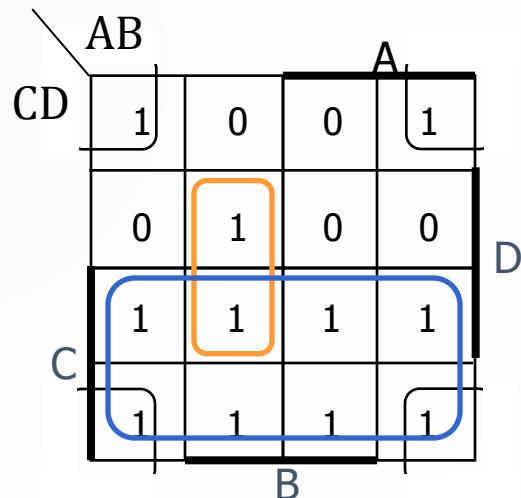


$$Y = \textcolor{blue}{X} + \textcolor{red}{Z}$$

# K-map examples (4 variable)

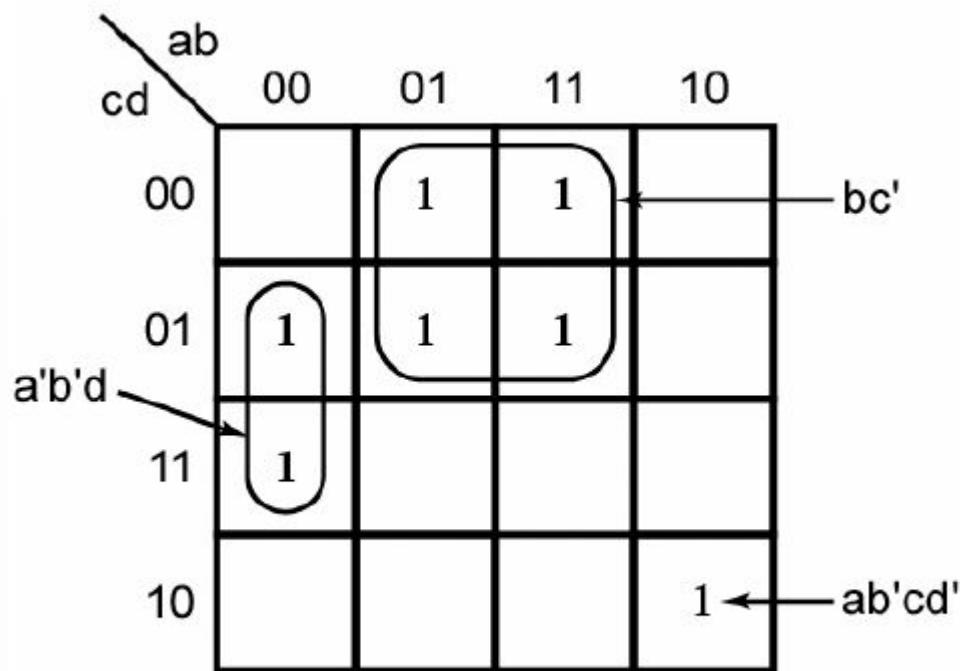
$$F(A,B,C,D) = \Sigma m(0,2,3,5,6,7,8,10,11,14,15)$$

$$= C + A'BD + B'D'$$



# K-map examples (4 variable)

$$f_1 = \Sigma m(1, 3, 4, 5, 10, 12, 13)$$



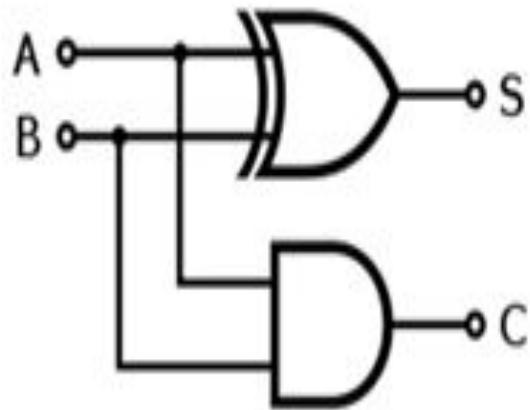
$$= bc' + a'b'd + ab'cd'$$

# Half Adder

- A *half-adder* is an arithmetic circuit block that can be used to add two 1 bit numbers. Such a circuit thus has **two inputs** that represent the two bits to be added and **two outputs**, with one producing the **SUM** output and the other producing the **CARRY**.
- Possible input combinations and the corresponding outputs are as given in the truth table.
- The Boolean expressions for the SUM and CARRY outputs are given by the following equations.

# Half Adder Truth Table

Input		Output	
A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1



Sum = S

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

$$\begin{aligned} \text{Sum} &= A\bar{B} + \bar{A}B \\ &= A \oplus B \end{aligned}$$

Carry = C

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

$$\text{Carry} = AB$$

# Full Adder

- A *full adder circuit* is an arithmetic circuit block that can be used to add *three bits* to produce a **SUM** and a **CARRY** output.
- Such a building block is needed in order to add binary numbers with a **large number of bits**.
- The full adder circuit overcomes the limitation of the half-adder, which can be used to add two bits only

# Full Adder

Input			Output	
A	B	Cin	Sum	Carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$$\text{Sum} = S$$

A	BC <sub>IN</sub>	00	01	11	10
0		0	1	0	1
1		1	0	1	0

$$\begin{aligned}
 S &= A'B'C + A'BC' + AB'C' + ABC \\
 &= A'(B'C + BC') + A(B'C' + BC) \\
 &= \overline{A}(\overline{B} \oplus C) + \overline{A}(\overline{B} \oplus \overline{C}) \\
 &= A \oplus B \oplus C
 \end{aligned}$$

$$\text{Carry} = C$$

A	BC <sub>IN</sub>	00	01	11	10
0		0	0	1	0
1		0	1	1	1

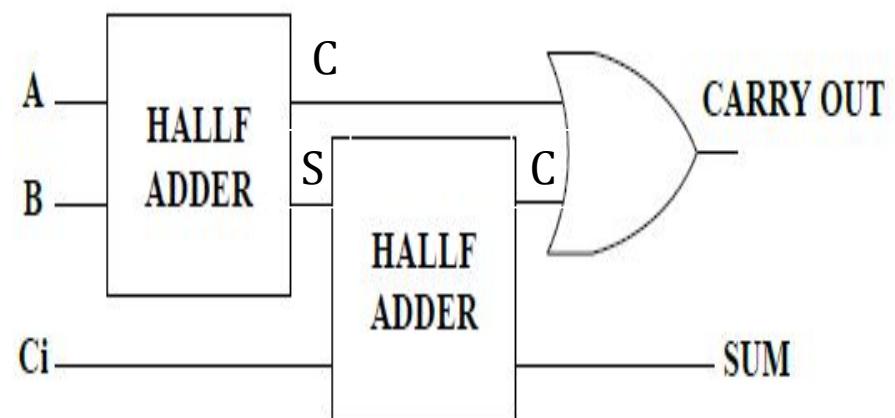
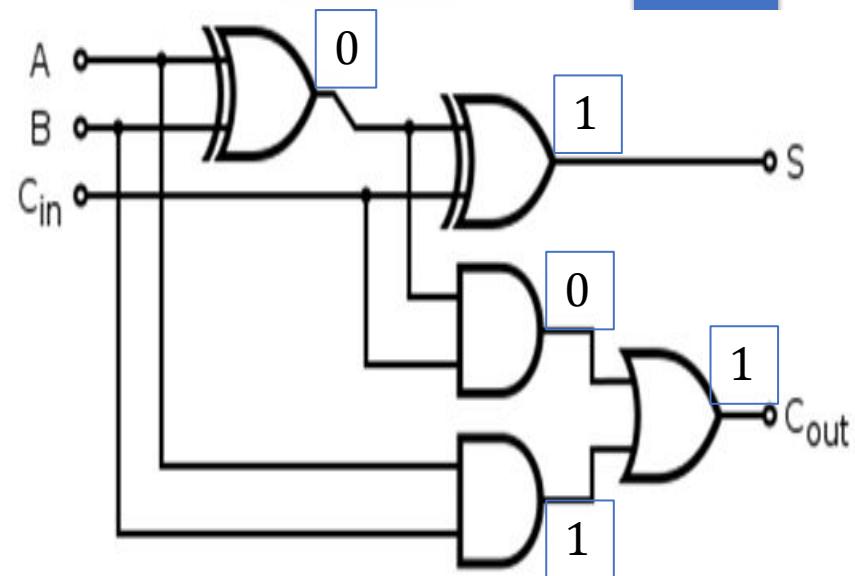
$$C = AB + BC + AC$$

$$S = \text{Sum}(1,2,4,7)$$

$$C = \text{Carry}(3,5,6,7)$$

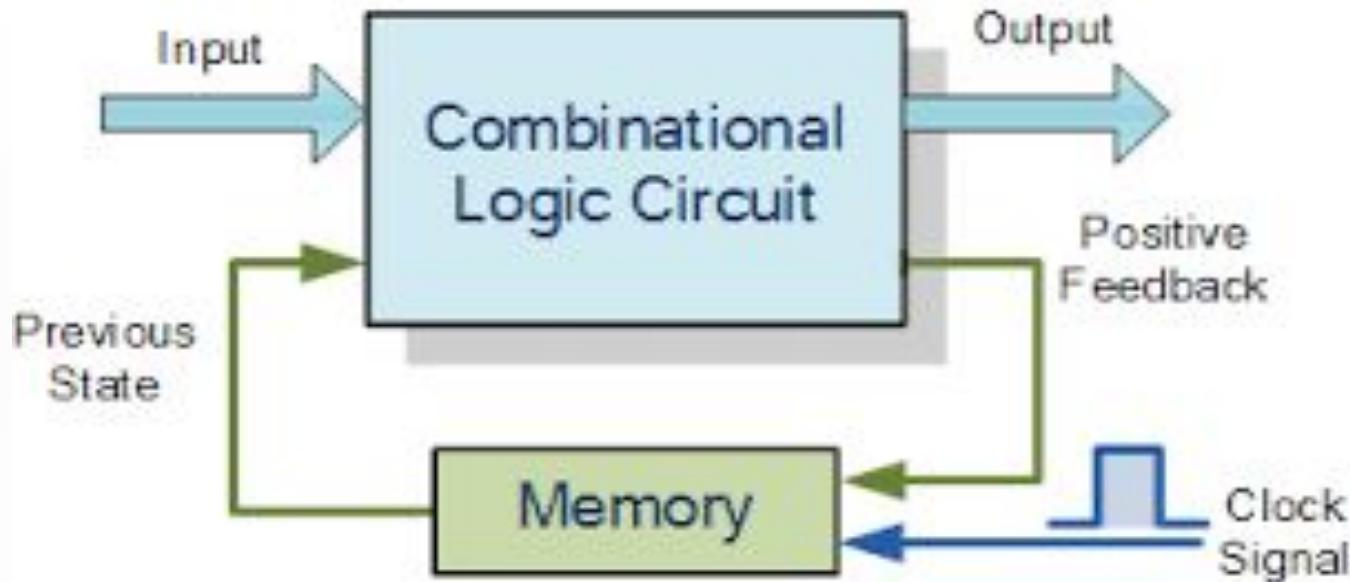
# Full Adder

Input			Output	
A	B	Cin	Sum	Carry
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



# Sequential Logic Circuit

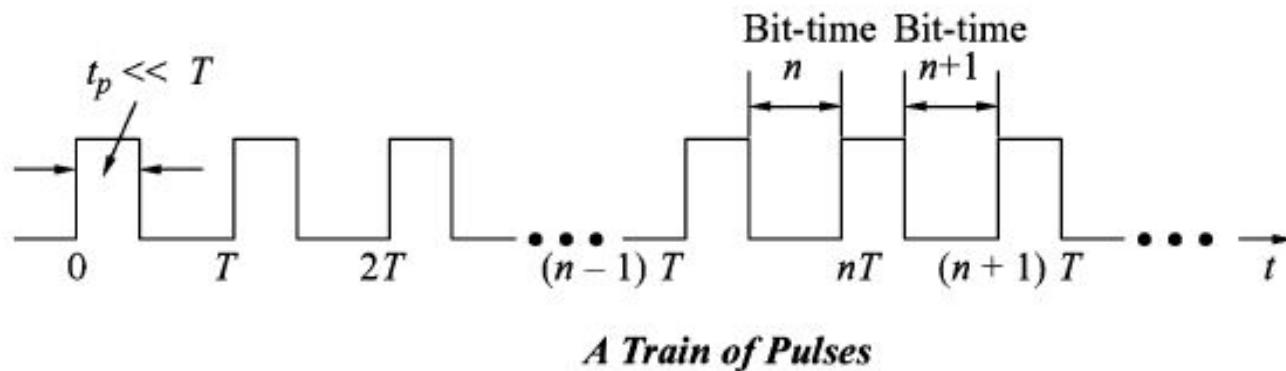
- **Sequential logic** is a type of logic circuit whose output depends not only on the present value of its input signals but on the sequence of past inputs, the input history.
- Sequential logic is combinational logic with memory.



- Sequential logic circuits are classified in 2 categories
  - ✓ Synchronous
  - ✓ Asynchronous

# Clock Pulse

- A Synchronization is achieved by the timing device known as system clock which generates a periodic train of clock pulses shown in figure.
- The outputs are affected with the application of clock pulse.



# Flip-Flop (1-Bit Memory Cell )

1. The outputs  $Q$  and  $\bar{Q}$  are always complementary.
2. The circuit has two stable states; in one of the stable state  $Q = 1$  which is referred to as the 1 state (*or set state*) whereas in the other stable state  $Q = 0$  which is referred to as the 0 state (*or reset state*).
3. If the circuit is in 1 state, it continues to remain in this state and similarly if it is in 0 state, it continues to remain in this state. This property of the circuit is referred to as *memory*, i.e. it can store 1-bit of digital information.

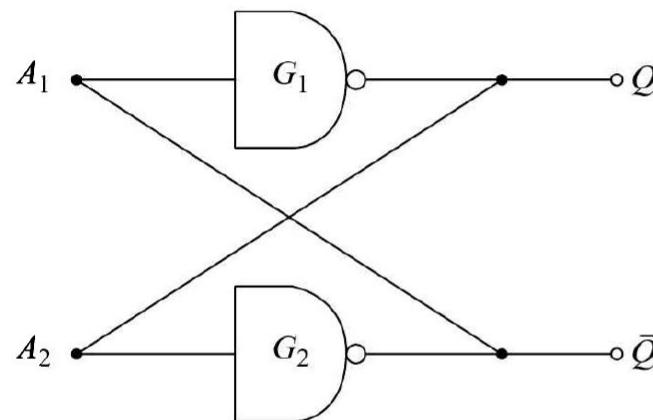


Fig. 7.3      *Cross-coupled Inverters as a  
Memory Element*

# Basic SR Flip-flop

- The simplest way to make any basic single bit set-reset SR flip-flop is to connect together a pair of cross-coupled 2-input NAND gates as shown
- There is feedback from each output to one of the other NAND gate inputs. This device consists of two inputs, one called the Set, S and the other called the Reset, R with two corresponding outputs Q and its inverse or complement  $\bar{Q}$  as shown below.

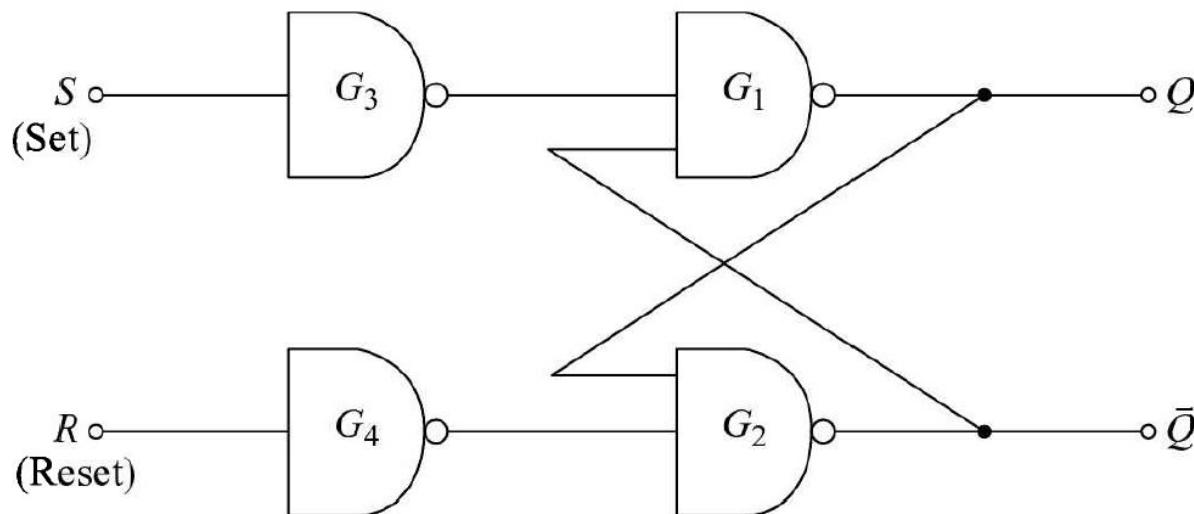


Fig. 7.4     ***The Memory Cell with Provision for Entering Data***

# Clocked SR Flip-Flop

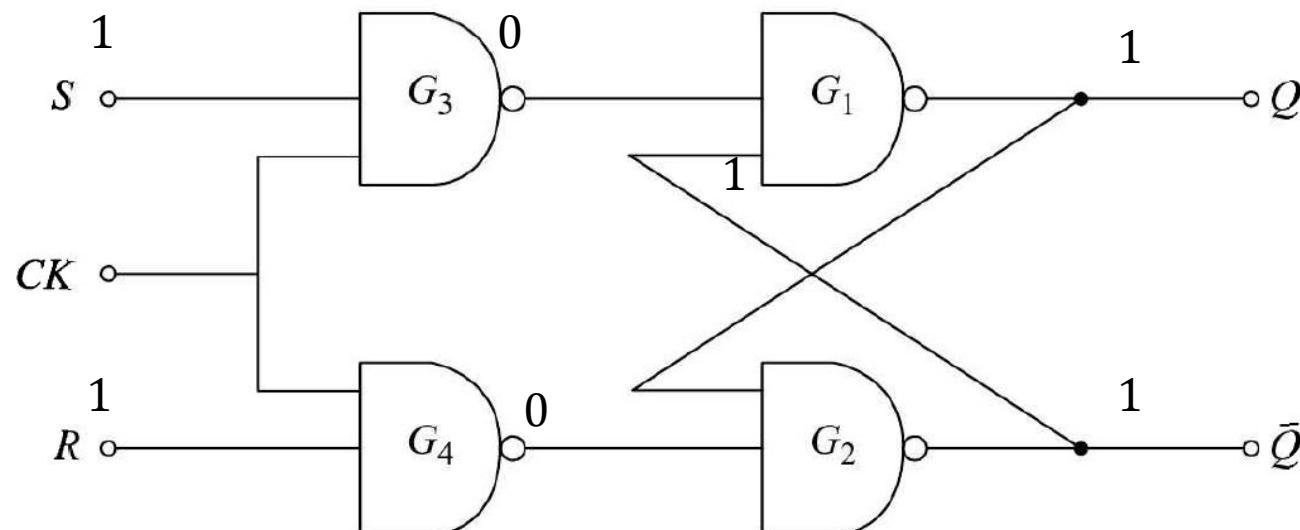


Fig. 7.5    **A Clocked S-R FLIP-FLOP**  
-----

# SR Flip-Flop

## Block diagram

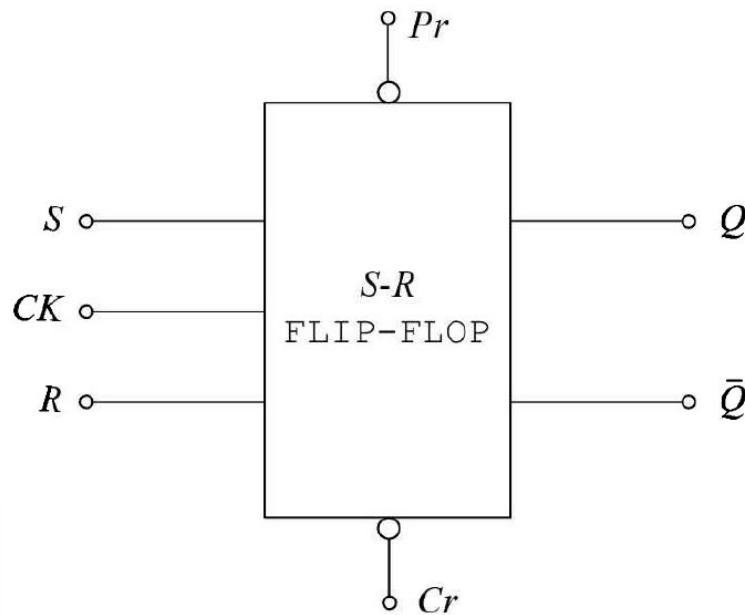


Table 7.1      ***Truth Table of S-R FLIP-FLOP***

Inputs		Output
$S_n$	$R_n$	$Q_{n+1}$
0	0	$Q_n$
1	0	1
0	1	0
1	1	?

# SR Operation

S.N.	Condition	Operation
1	<b><math>S = R = 0</math> : No change</b>	If $S = R = 0$ then output of NAND gates 3 and 4 are forced to become 1. Hence $R'$ and $S'$ both will be equal to 1. Since $S'$ and $R'$ are the input of the basic S-R latch using NAND gates, there will be no change in the state of outputs.
2	<b><math>S = 0, R = 1, E = 1</math></b>	Since $S = 0$ , output of NAND-3 i.e. $R' = 1$ and $E = 1$ the output of NAND-4 i.e. $S' = 0$ . Hence $Q_{n+1} = 0$ and $Q_{n+1} \text{ bar} = 1$ . This is reset condition.
3	<b><math>S = 1, R = 0, E = 1</math></b>	Output of NAND-3 i.e. $R' = 0$ and output of NAND-4 i.e. $S' = 1$ . Hence output of S-R NAND latch is $Q_{n+1} = 1$ and $Q_{n+1} \text{ bar} = 0$ . This is the set condition.
4	<b><math>S = 1, R = 1, E = 1</math></b>	As $S = 1, R = 1$ and $E = 1$ , the output of NAND gates 3 and 4 both are 0 i.e. $S' = R' = 0$ . Hence the <b>Race</b> condition will occur in the basic NAND latch.

# J-K Flip-Flop

Uncertainty in the state of SR flip-flop when S=R=1 can be eliminated by converting it into JK flip-flop

$$S = J \cdot \bar{Q}$$

$$R = K \cdot Q$$

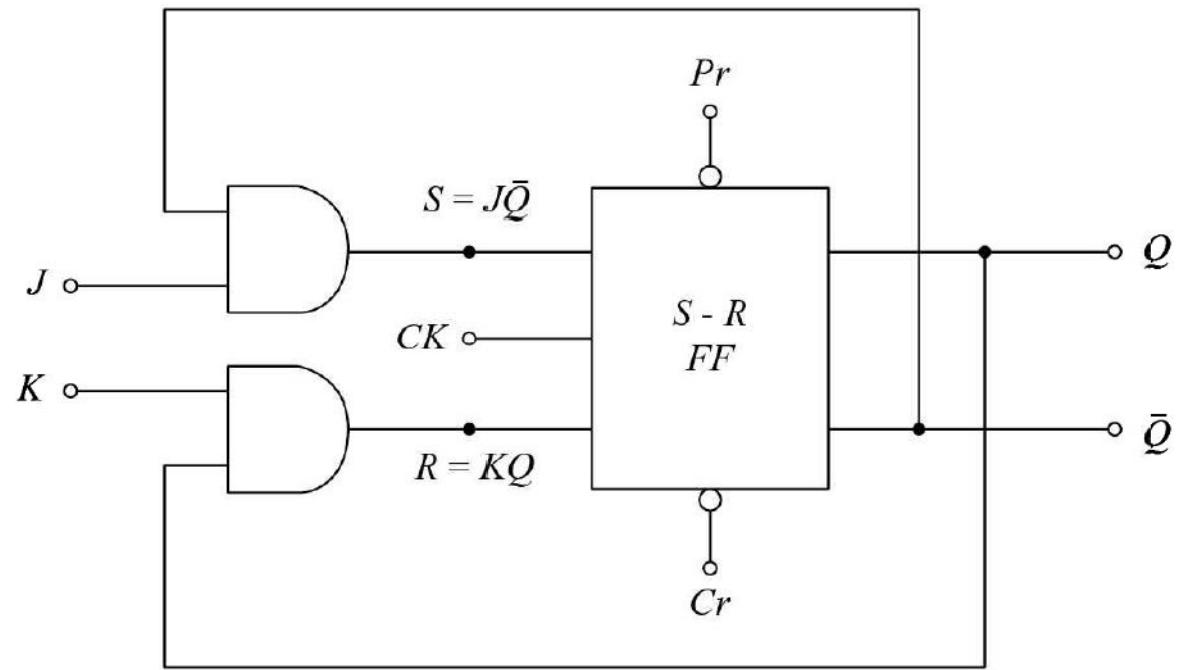


Fig. 7.8 *An S-R FLIP-FLOP Converted into J-K FLIP-FLOP*

# Truth Table from Figure

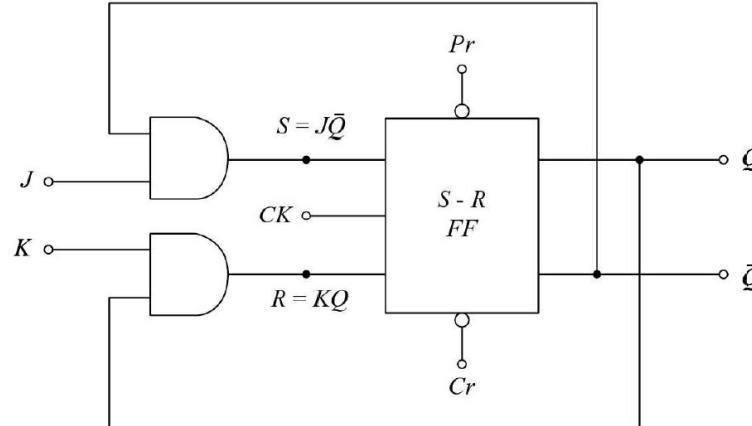


Fig. 7.8 An S-R FLIP-FLOP Converted into J-K FLIP-FLOP

Data inputs		Outputs		Inputs to S-R FF		Output
$J_n$	$K_n$	$Q_n$	$\bar{Q}_n$	$S_n$	$R_n$	$Q_{n+1}$
0	0	0	1	0	0	0
0	0	1	0	0	0	1] = $Q_n$
1	0	0	1	1	0	1] = 1
1	0	1	0	0	0	1]
0	1	0	1	0	0	0] = 0
0	1	1	0	0	1	0]
1	1	0	1	1	0	1] = $\bar{Q}_n$
1	1	1	0	0	1	0]

# Truth Table of J-K Flip-Flop

Fig: Block Diagram

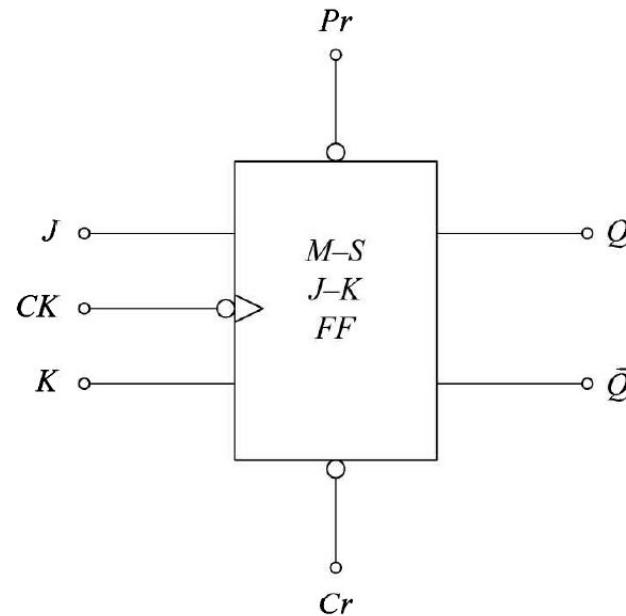


Table 7.3b *Truth Table of J-K FLIP-FLOP*

Inputs		Output
$J_n$	$K_n$	$Q_{n+1}$
0	0	$Q_n$
1	0	1
0	1	0
1	1	$\bar{Q}_n$

# D Flip-Flop

- If we use only middle two rows of SR or JK flip-flop, We obtain D Flip-flop.

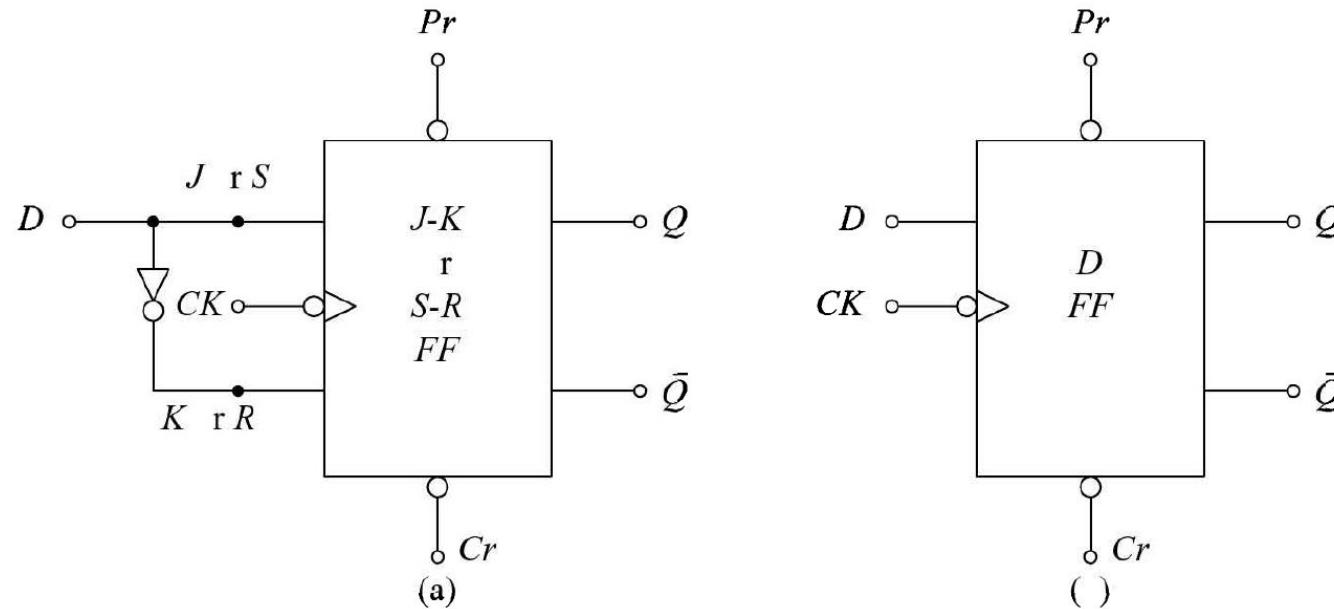


Fig. 7.14 (a) A J-K or S-R FLIP-FLOP Converted into a D-type FLIP-FLOP (b) its Logic Symbol

Table 7.4 Truth Table of a D-type FLIP-FLOP

Input	Output
$D_n$	$Q_{n+1}$
0	0
1	1

# Operation of D Flip-Flop

S.N	Condition	Operation
1	<b>E = 0</b>	Latch is disabled. Hence no change in output.
2	<b>E = 1 and D = 0</b>	If E = 1 and D = 0 then S = 0 and R = 1. Hence irrespective of the present state, the next state is $Q_{n+1} = 0$ and $Q_{n+1} \text{ bar} = 1$ . This is the reset condition.
3	<b>E = 1 and D = 1</b>	If E = 1 and D = 1, then S = 1 and R = 0. This will set the latch and $Q_{n+1} = 1$ and $Q_{n+1} \text{ bar} = 0$ irrespective of the present state.

# T Flip-Flop

- In JK flip-flop, if  $J=K$ , the resulting Flip-flop is called as T Flip-flop.

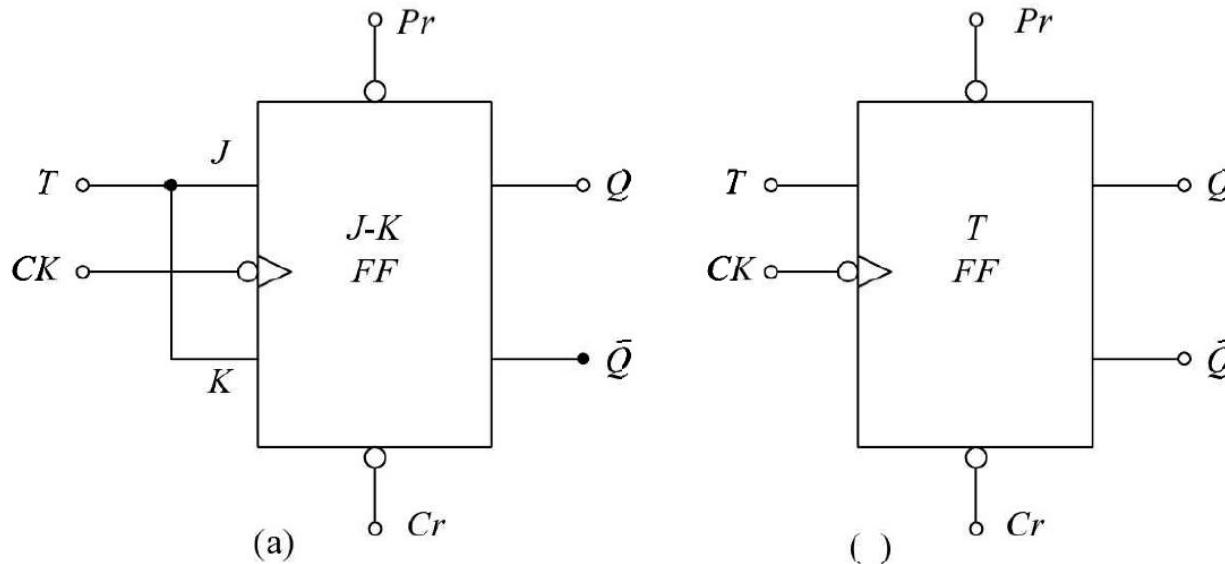


Fig. 7.15 (a) A J-K FLIP-FLOP **Converted into a T-type FLIP-FLOP** (b) its Logic Symbol

Table 7.5      **Truth Table of T-type FLIP-FLOP**

Input	Output
$T_n$	$Q_{n+1}$
0	$Q_n$
1	$\bar{Q}_n$

# Operation of T Flip-Flop

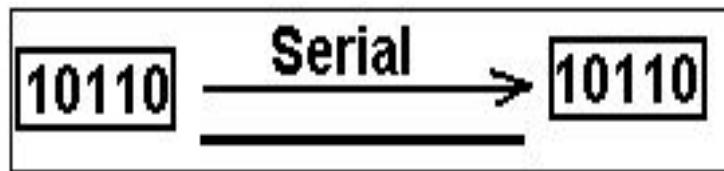
S.N	Condition	Operation
1	$T = 0, J = K = 0$	The output Q and Q bar won't change
2	$T = 1, J = K = 1$	Output will toggle corresponding to every leading edge of clock signal.

# Shift Register

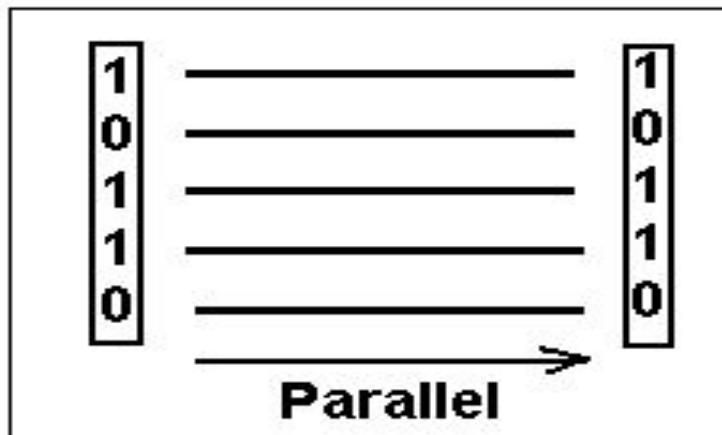
- Register is a digital circuit with two basic functions:
  1. Data storage
  2. Data Movement
- A register can consist of one or more flip flops used to store and shift data
- Shift Registers are an important Flip-Flop configuration with a wide range of applications, including:
  - ❖ Computer and Data Communications
  - ❖ Serial and Parallel Communications
  - ❖ Multi-bit number storage
  - ❖ Sequencing
  - ❖ Basic arithmetic such as scaling (a serial shift to the left or right will change the value of a binary number a power of 2)
  - ❖ Logical operations

# Parallel versus Serial

- Serial communications: provides a binary number as a sequence of binary digits, one after another, through one data line.

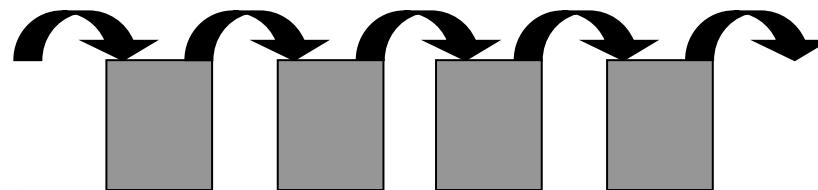


- Parallel communications: provides a binary number as binary digits through multiple data lines at the same time.

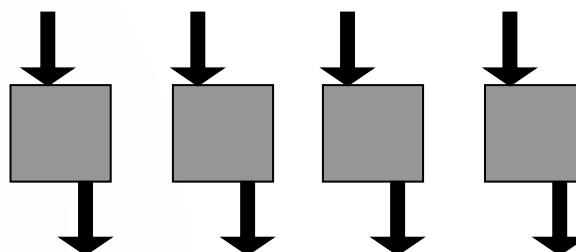


# Shift Registers

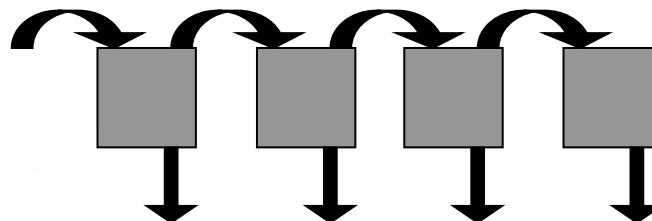
- Shift Registers are devices that store and move data bits in serial fashion (to the left or the right),



- In parallel fashion

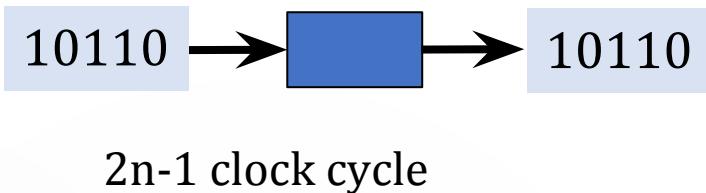


- A combination of serial and parallel fashion

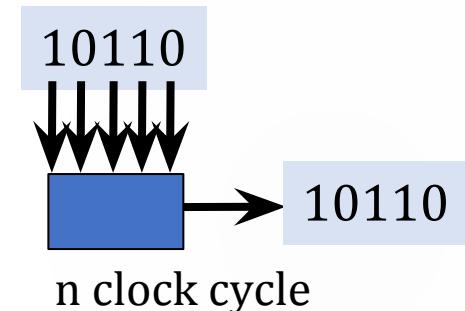


# Data Transfer Methods/ Modes

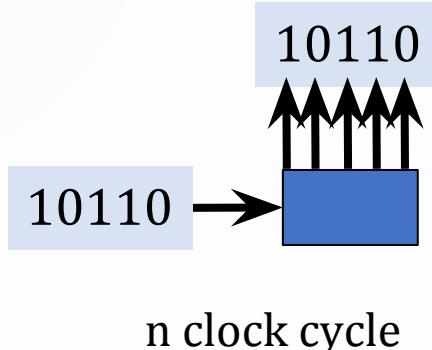
## 1. SISO: Serial In, Serial Out



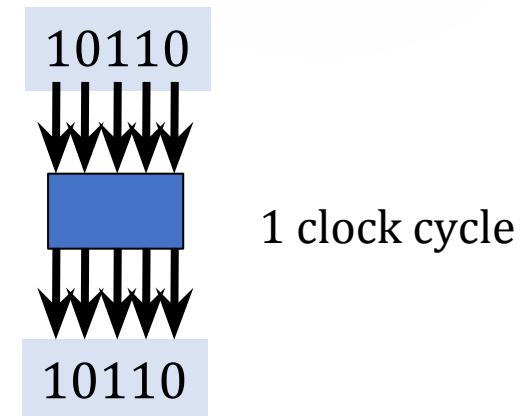
## 3. PISO: Parallel In, Serial Out



## 2. SIPO: Serial In, Parallel Out



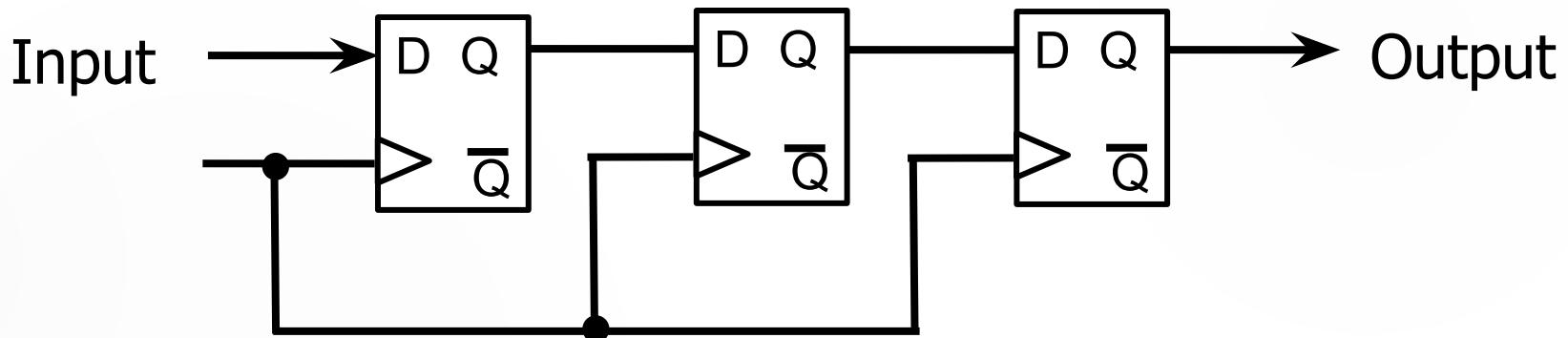
## 4. PIPO: Parallel In, Parallel Out



How many clock edges are required for each operation?

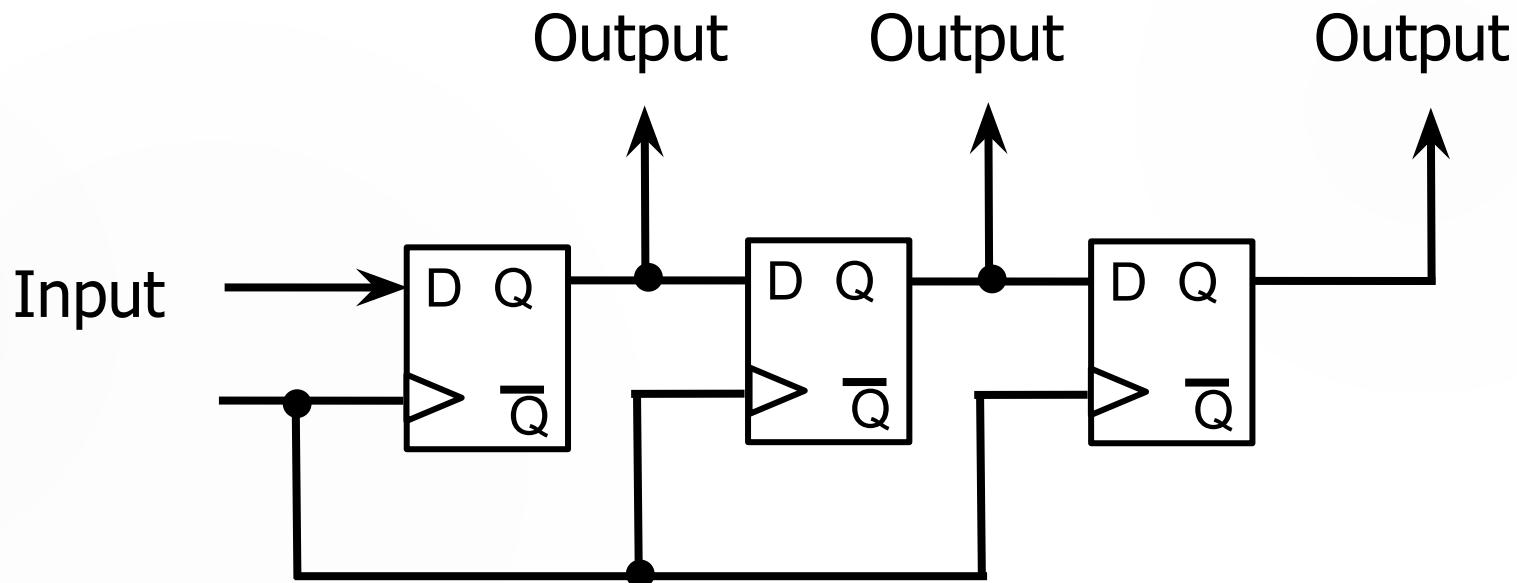
# SISO Flip-Flop Shift Register

- A **Serial In Serial Out** shift register has a single input and a single output



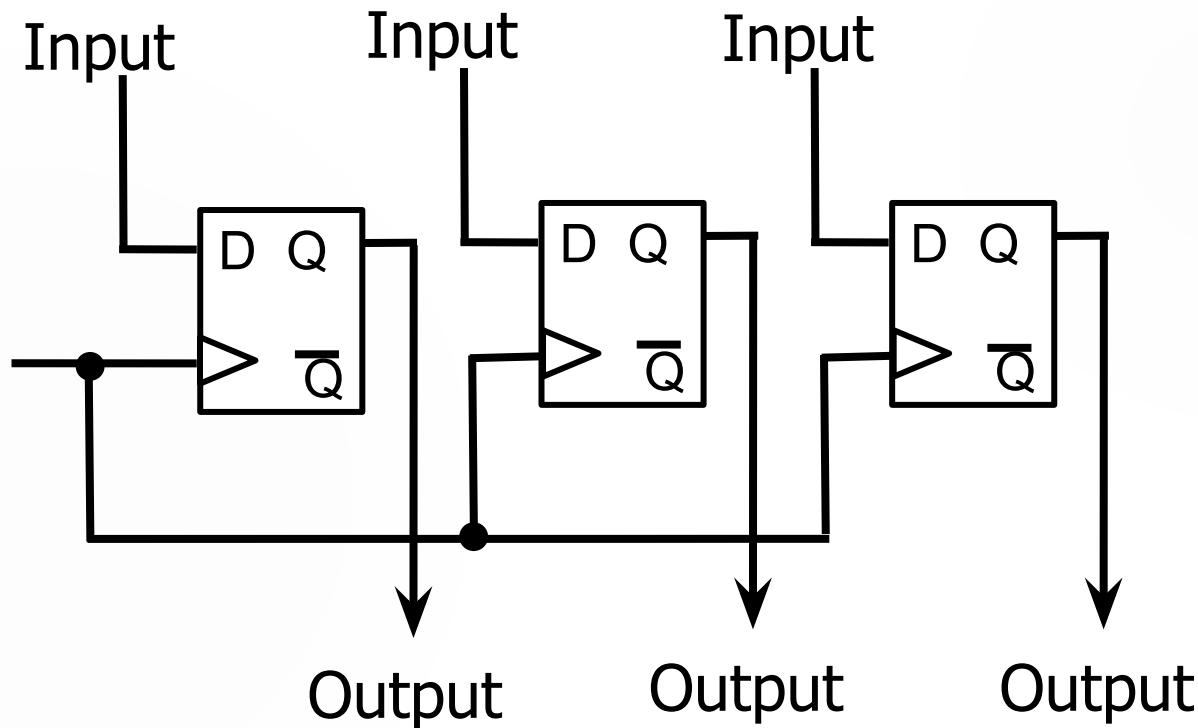
# SIFO Flip-Flop Shift Register

- **Serial In Parallel Out** shift register has a single input and access to all outputs



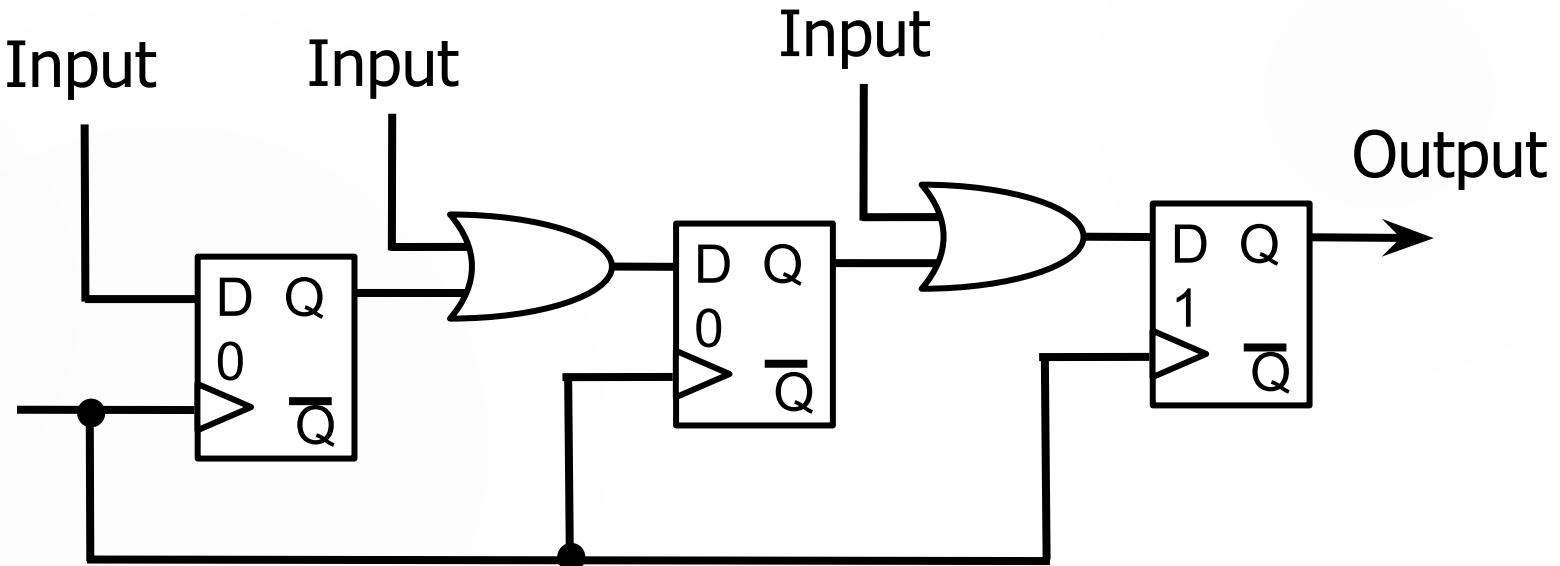
# PIPO Flip-Flop Shift Register

- **Parallel In Parallel Out** register has the simplest configuration. It represents a memory device.



# PISO Flip-Flop Shift Register

- **Parallel In Serial Out** shift register requires additional gates, and the parallel input must revert to logic low.



# Universal Shift Registers

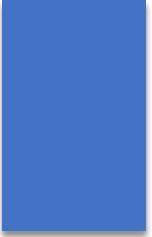
- Universal Shift Registers can be configured to operate in a variety of modes. For instance, they can be configured to have either Serial or Parallel Input/Output.
- Mode of Operations
  - SISO
  - SIPO
  - PISO
  - PIPO

# Contributions of IC technology

- Timers
- Memories
- Microprocessors and micro-controllers
- Embedded Systems etc.

# Additional links for more information:

- <http://www.ee.surrey.ac.uk/Projects/CAL/digital-logic/gatesfunc/>
- <https://www.electronicshub.org/half-adder-and-full-adder-circuits/>



# Thank You