ELECTRONIC PRINCIPLES AND DEVICES

Department of Electronics and Communication Engineering

UNIT - III: DIGITAL ELECTRONICS

Contents:

INTRODUCTION

Digital Electronics vs. Analog Electronics:

- Analog electronics use continuous signals to represent and process information where as digital electronics, use discrete signals to represent and process information.
- Analog electronics can be used to amplify signals, filter noise, and perform a wide variety of other functions.
- Some common components used in analog electronics include resistors, capacitors, inductors, and transistors.
- Digital systems are often preferred for their ability to store and transmit data with a high degree of accuracy.
- Digital systems are made up of components such as gates, and flip-flops, which are used to manipulate binary data.

INTRODUCTION

Digital Systems:

- Digital systems have such a prominent role in everyday life that we refer to the present technological period as the digital age.
- Digital systems are used in communication, business transactions, traffic control, spacecraft guidance, medical treatment, weather monitoring, the Internet, and many other commercial, industrial, and scientific enterprises.
- We have digital telephones, digital televisions, digital versatile discs, digital cameras, handheld devices, and, of course, digital computers.

UNIT - III: DIGITAL ELECTRONICS

❖ Digital Electronic circuits process data that contains binary values 1’s and 0’s.

❖ A Logic “1” is also referred as HIGH voltage or TRUE or ON state

❖ A Logic “0” is also referred as LOW voltage or FALSE or OFF state

❖ A binary digit “0” or “1” is called a bit

❖ Digital information is stored using a series of binary values 1’s and

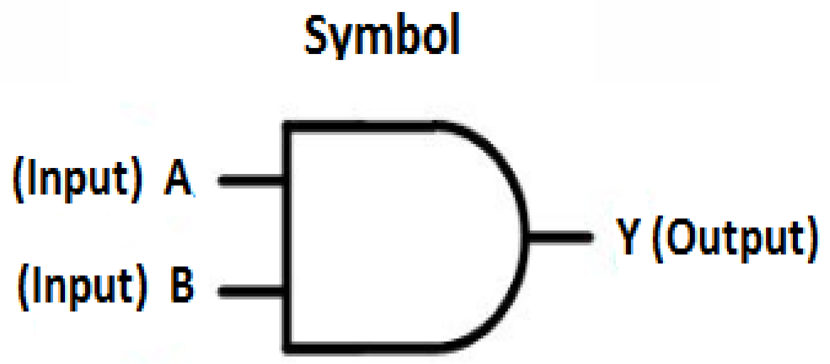
❖Digital systems such as digital telephones, digital cameras, computers, handheld portable devices and other high technology systems stores and process these binary values (1’s and 0’s)

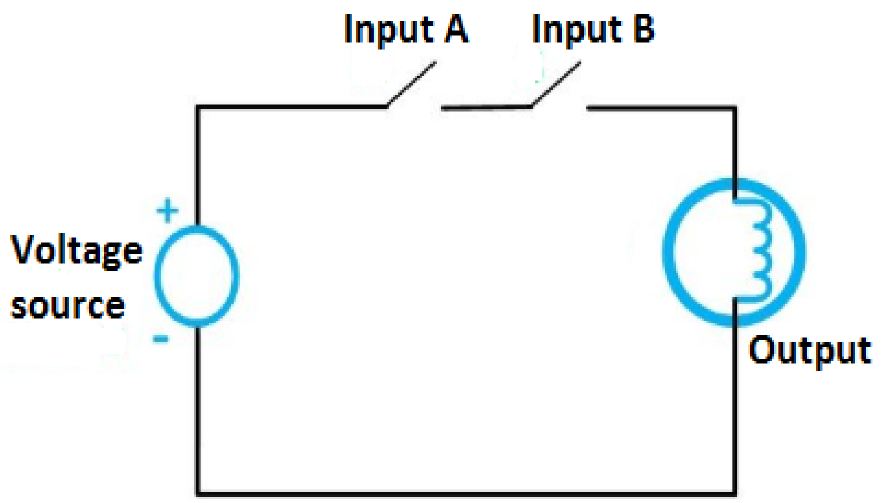
Boolean Algebra and Logic gates

- ❖ Boolean Algebra is the mathematics used to analyse and simplify logic or digital circuits.
- ❖ Digital circuits are constructed by using logic gates.
- ❖ Logic gates are the basic building blocks of digital systems, performs logical functions based on Boolean algebra.
- ❖ Logic gates have one or more inputs and only one output.
- ❖ Number of possible input states is 2^n. Where n is Number of inputs
- ❖ Logic gates are implemented using diodes or transistors which acts as electronic switches
- ❖ Logic gates :(i) Basic Gates: AND, OR, NOT,
  (ii) Derived Gates: NAND, NOR, EXOR and EXNOR gates

Boolean Algebra and Logic gates

- ❖ **AND Gate** is an electronic circuit that gives output as logic “1” (HIGH) only if all inputs are “1”. Otherwise output is Logic “0” (LOW)
- ❖ Boolean algebra representation for AND gate is **Y = A.B**
- ❖ AND gate is represented as **series connection** of two switches





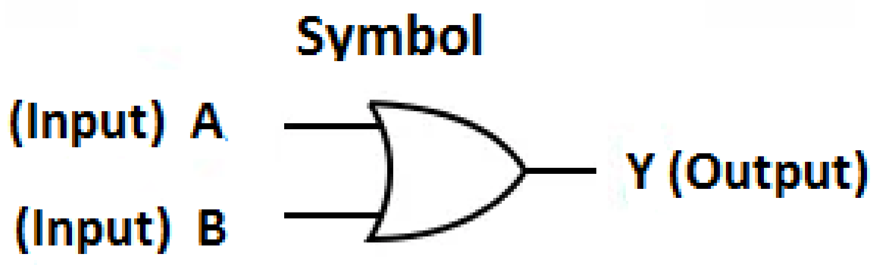
Truth Table

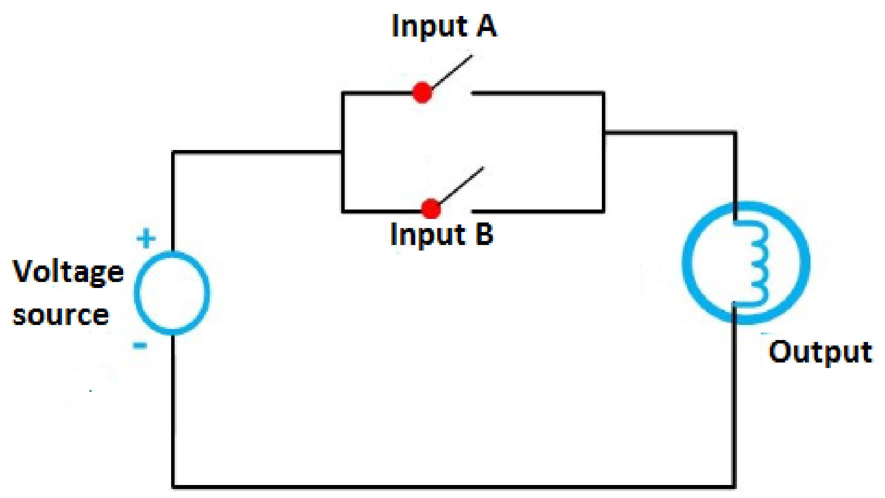
Inputs		Output
A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

ELECTRONIC PRINCIPLES AND DEVICES

Boolean Algebra and Logic gates

- ❖ **OR Gate** is an electronic circuit that gives as Logic “1” (HIGH) if any of the inputs are HIGH .Output of OR gate goes Logic “0” (LOW) only if all inputs are Logic “0” (LOW)
- ❖ Boolean algebra representation of OR gate is Y = A + B
- ❖ OR gate is represented as **parallel connection** of two switches



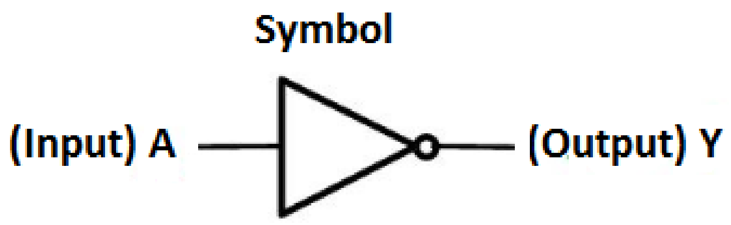


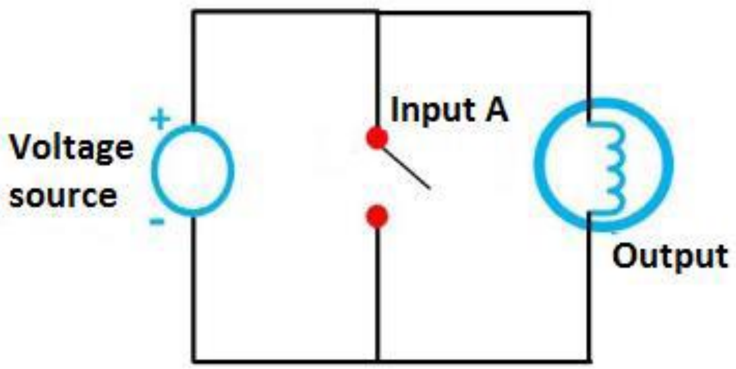
Truth Table

Inputs		Output
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

Boolean Algebra and Logic gates

- ❖ **NOT Gate** is a single input and single output gate which performs the inversion of the applied binary input signal.
- ❖ Hence it is also called as Inverter Gate.
- ❖ Boolean expression for NOT gate is $Y = \overline{A}$
- ❖ NOT Gate can be represented by connecting a switch parallel to bulb





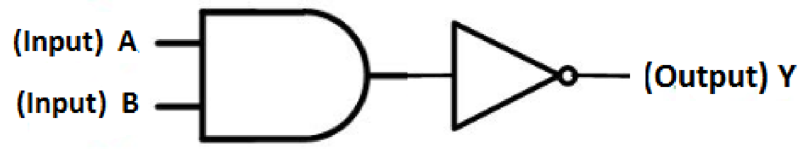
Truth Table

Input	Output
A	Y
0	1
1	0

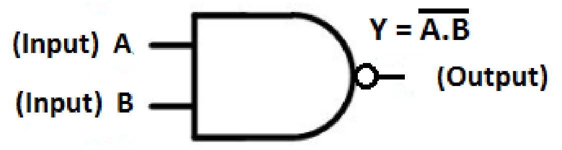
ELECTRONIC PRINCIPLES AND DEVICES

Boolean Algebra and Logic gates

NAND Gate



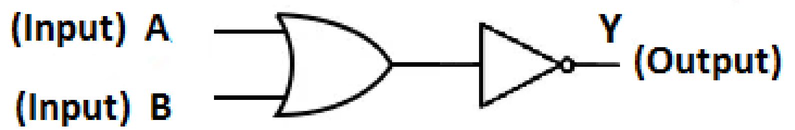
Symbol



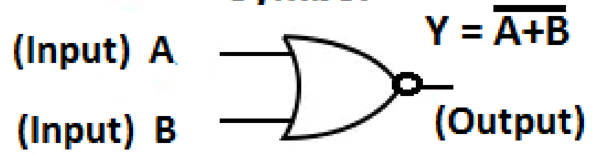
Truth Table

NAND		
Input		Output
A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

NOR Gate



Symbol



Truth Table

NOR		
Input		Output
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

Boolean Algebra and Logic gates

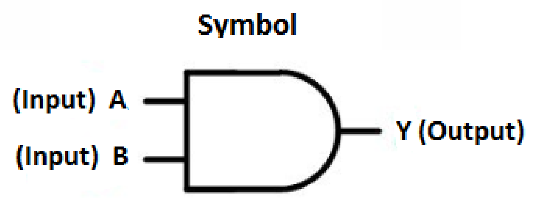
❖ Logical Boolean Laws:

❖ NOT / Inversion Law:

(i) If A = 1 then A’ = 0 and if A = 0 then A’ = 1
i.e. complement of 1 is 0 and vice versa.

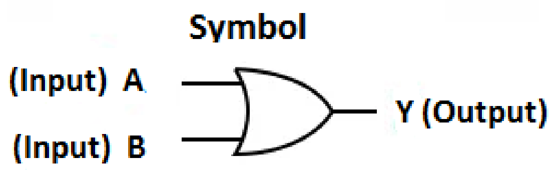
(ii) (A’)’ = A itself

❖AND Law:



A.B = Y	A.B = Y
0.0 = 0	A.0 = 0
0.1 = 0	A.1 = A
1.0 = 0	A.A = A
1.1 = 1	A.A’ = 0

❖OR Law:



A+B = Y	A+B = Y
0+0 = 0	A+0 = A
0+1 = 1	A+1 = 1
1+0 = 1	A+A = A
1+1 = 1	A+A’ = 1

Boolean Algebra and Logic gates

❖ Principle of duality in Boolean Algebra

❖ In Boolean Algebra , One type of expression can be converted into another type of expression by replacing “0 with 1” , “ 1 with 0” , “(+) sign with (.) sign” and vice versa.

Example:

AND (.)	OR (+)
0.0 = 0	1+1 = 1
0.1 = 0	1+0 = 1
1.0 = 0	0+1 = 1
1.1 = 1	0+0 = 0

Expression	Dual Expression
$\overline{0} = 1$	$\overline{1} = 0$
$0.1 = 0$	$1 + 0 = 1$
$A.0 = 0$	$A + 1 = 1$
$A.B = B.A$	$A + B = B + A$
$A.\overline{A} = 0$	$A + \overline{A} = 1$

THANK YOU

Department of Electronics and Communication

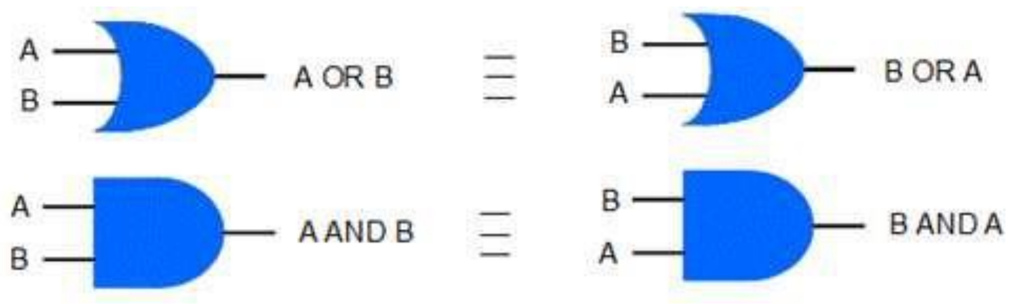
Basic Theorem and Properties of Boolean Algebra

❖ Boolean Laws and Theorems are used to simplify the Boolean expressions. Hence reduce the number of logic gates.

❖ Commutative Laws:

(i) A+ B = B+ A
(ii) A.B = B.A

Proof: Logic circuits



Proof: Truth Table

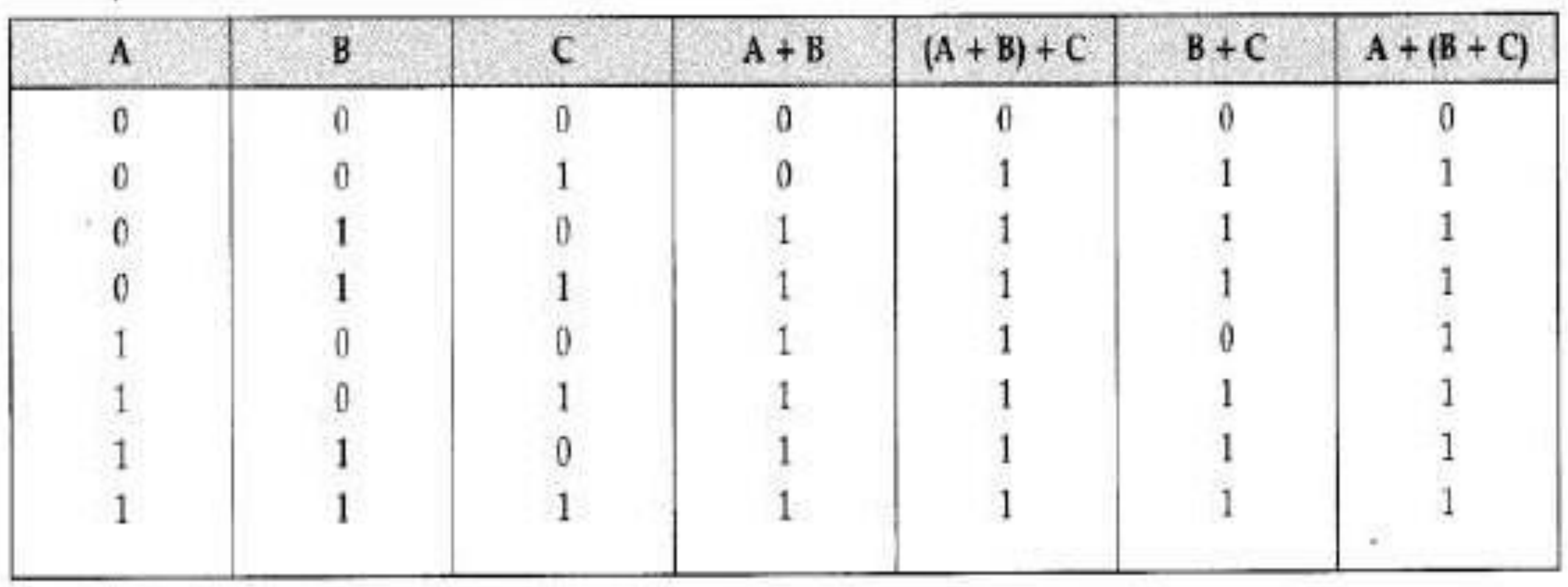
A	B	(A+B)	(B+A)	(A.B)	(B.A)
0	0	0	0	0	0
0	1	1	1	0	0
1	0	1	1	0	0
1	1	1	1	1	1

Basic Theorem and Properties of Boolean Algebra

❖ **Associative Law:**

(A+B)+C = A+(B+C)   Boolean addition

❖ Proof:



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

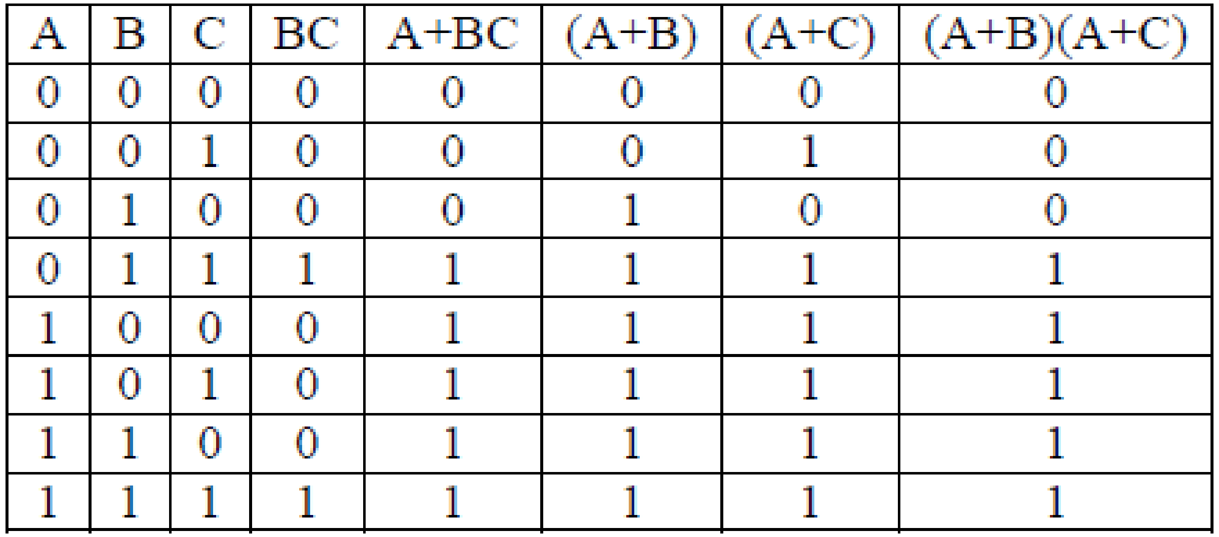
❖ From principle of duality: (A.B).C = A.(B.C)   Boolean Multiplication

Basic Theorem and Properties of Boolean Algebra

❖ **Distributive Law:**

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

❖ Proof: Truth Table

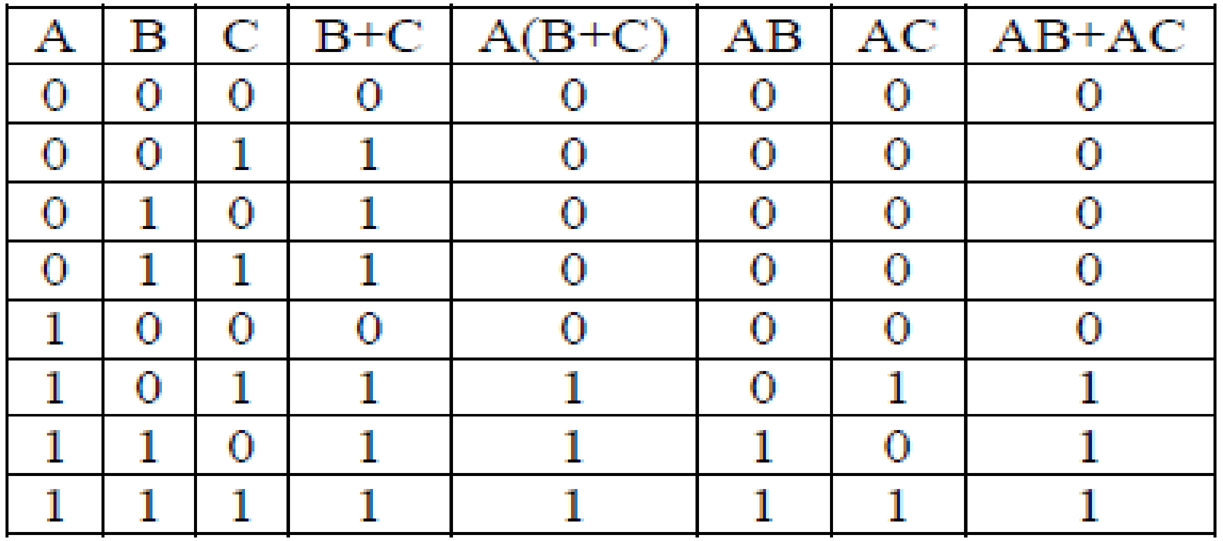


A	B	C	BC	A+BC	(A+B)	(A+C)	(A+B)(A+C)
0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0
0	1	0	0	0	1	0	0
0	1	1	1	1	1	1	1
1	0	0	0	1	1	1	1
1	0	1	0	1	1	1	1
1	1	0	0	1	1	1	1
1	1	1	1	1	1	1	1

Dual of distributive law:

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

❖ Proof: Truth Table



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

Basic Theorem and Properties of Boolean Algebra

De Morgan's Theorem

(i) The complement of the sum of 2 variables is equal to the product of the complements of individual variables:

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

(ii) The complement of the product of 2 variables is equal to the sum of the complements of individual variables:

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

A	B	$\overline{A}$	$\overline{B}$	$A+B$	$A.B$	$\overline{A+B}$	$\overline{A}\,.\,\overline{B}$	$\overline{A\,.\,B}$	$\overline{A}+\overline{B}$
0	0	1	1	0	0	1	1	1	1
0	1	1	0	1	0	0	0	1	1
1	0	0	1	1	0	0	0	1	1
1	1	0	0	1	1	0	0	0	0

Basic Theorem and Properties of Boolean Algebra

Absorption Theorem:

(i) A+AB = A

LHS: $= A + AB$

$= A.1 + AB \rightarrow$ since $A.1 = A$

$= A(1+B) \rightarrow$ since $1 + B = 1$

$= A.1$

$= A =$ RHS

(ii) A(A+B) = A

LHS $= A (A + B)$

$= A.A + A.B$

$= A+AB \rightarrow$ since $A . A = A$

$= A (1 + B)$

$= A.1$

$= A =$ RHS

(iii) $A+\bar{A}B = A+B$

LHS $= A + \bar{A}B$

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

$= (1) . (A + B) \rightarrow$ since $A + \bar{A} = 1$

$= A + B =$ RHS

(iv) $A.(\bar{A}+B) = AB$

LHS $= A.(\bar{A} + B)$

$= A. \bar{A} + A.B \rightarrow (A\bar{A} = 0)$

$= AB =$ RHS

Basic Theorem and Properties of Boolean Algebra

Consensus Theorem:

$AB + \bar{A}C + BC = AB + \bar{A}C$

$LHS = AB + \bar{A}C + BC$

$= AB + \bar{A}C + BC.1$

$= AB + \bar{A}C + BC(A + \bar{A}) \rightarrow$ since $A + \bar{A} = 1$

$= AB + \bar{A}C + ABC + \bar{A}BC$

$= AB(1 + C) + \bar{A}C(1 + B)$

$1 + B = 1 + C = 1$

$= AB + \bar{A}C = RHS$

Note: Here BC is redundant term

❖ **Dual of consensus theorem**:

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

PES
UNIVERSITY
CELEBRATING 50 YEARS

THANK YOU

Department of Electronics and Communication

Boolean Functions, Canonical and Standard Form

- ❖ Boolean function described by an algebraic expression consists of **binary variables, the constants 0 and 1, and the logic operation symbols**.

- ❖ Boolean function is **evaluated to logic-1 or logic-0** for a given value of the binary variables.

- ❖ Boolean function can be represented in a **truth table**.

- ❖ A Boolean function can be implemented as **digital circuit**, Which is constructed by using logic gates.

- ❖ Example: F = A + B'C, in this Boolean Equation, F = 1

  if A = 1 or if B = 0 and C = 1

  F = 0 Otherwise

Boolean Functions, Canonical and Standard Form

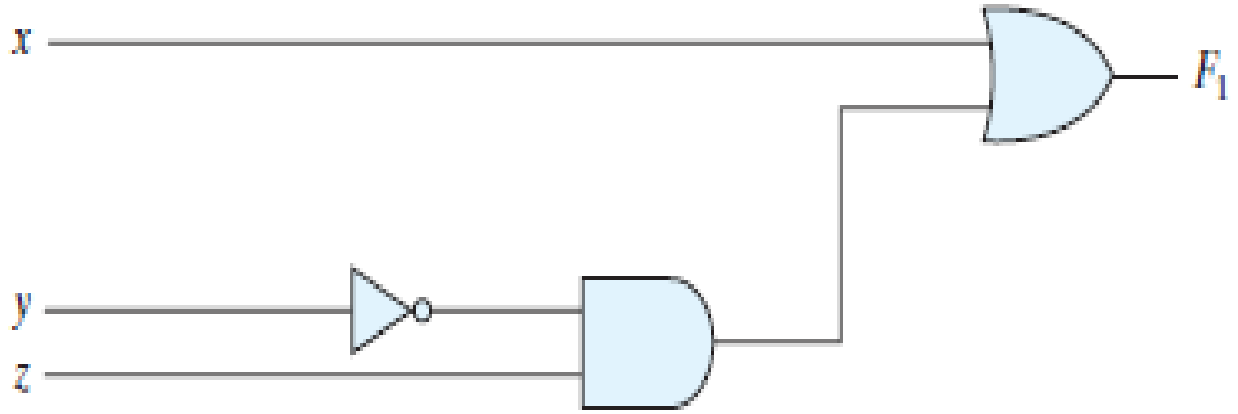
❖ Example: $F_1 = x + y'z$

❖ F_1 contains either 0 or 1 for each of these combinations. The table shows that the function is equal to 1 when x = 1 or when yz = 01 and is equal to 0 otherwise.

Truth Table:

x	y	z	F_1
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

Gate Level Implementation:



Boolean Functions, Canonical and Standard Form

❖ Example: Simplify and realize the given Boolean function using basic gates

$F_2 = x'y'z + x'yz + xy'$................(1)

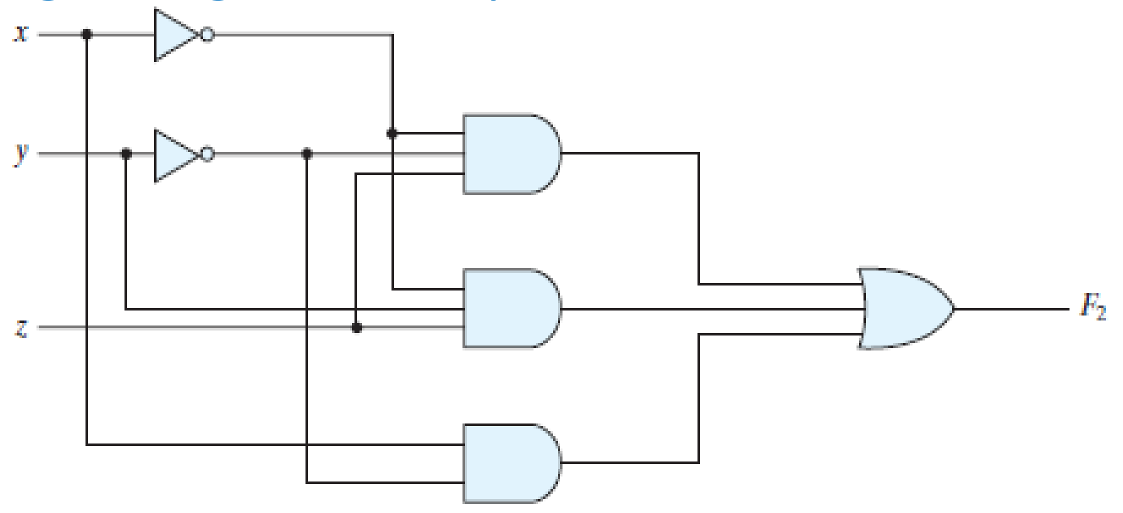
$F_2 = x'z(y' + y) + xy'$

$F_2 = x'z + xy'$..............(2)

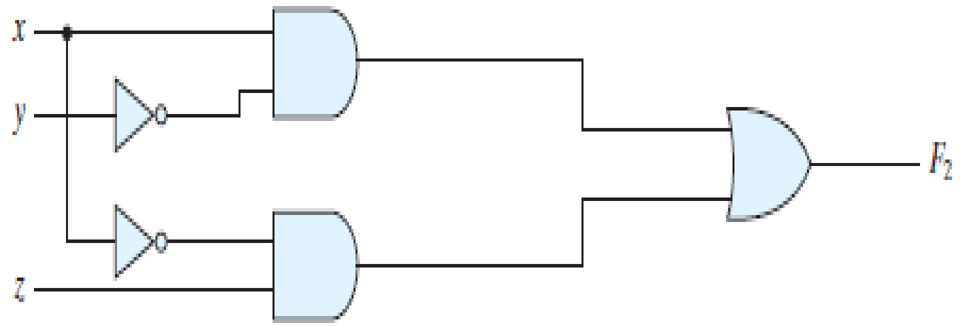
Truth Table

x	y	z	F_2
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

Logic Diagram for eq.1



Logic Diagram for eq.2

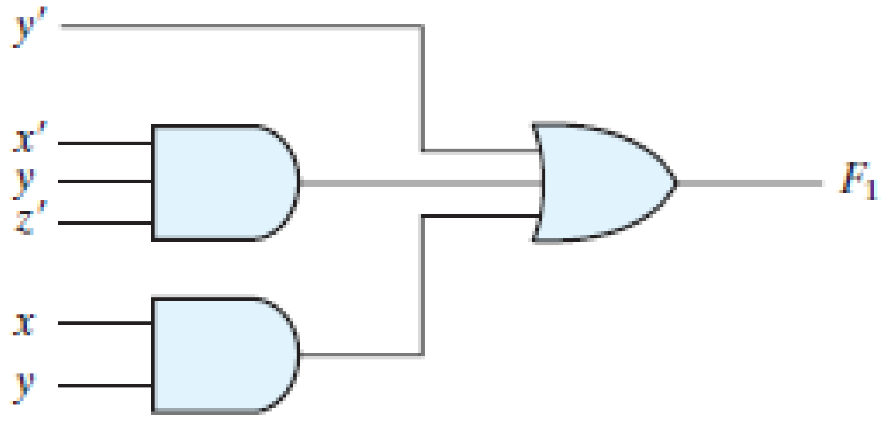


Boolean Functions, Canonical and Standard Form

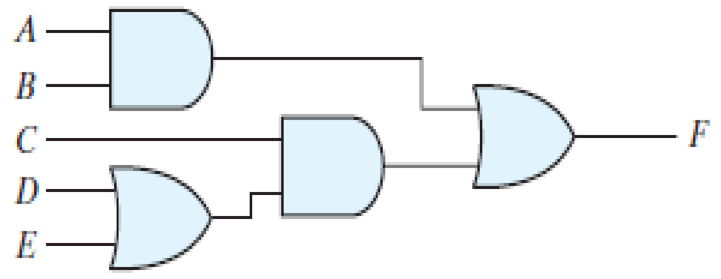
❖ **Sum of Products: (SOP)**

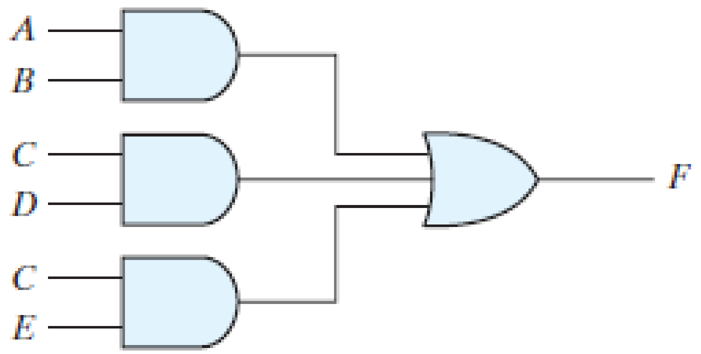
- The sum of products is a Boolean expression containing **AND** terms, called **product terms,** with one or more literals each. The **sum** denotes the **OR**ing of these terms.
- Two- Level Logic implementation of SOP

❖ Example 1 : $F_1 = y' + xy + x'yz'$



Example 2: F = AB + C (D+E)





F = AB + CD + CE

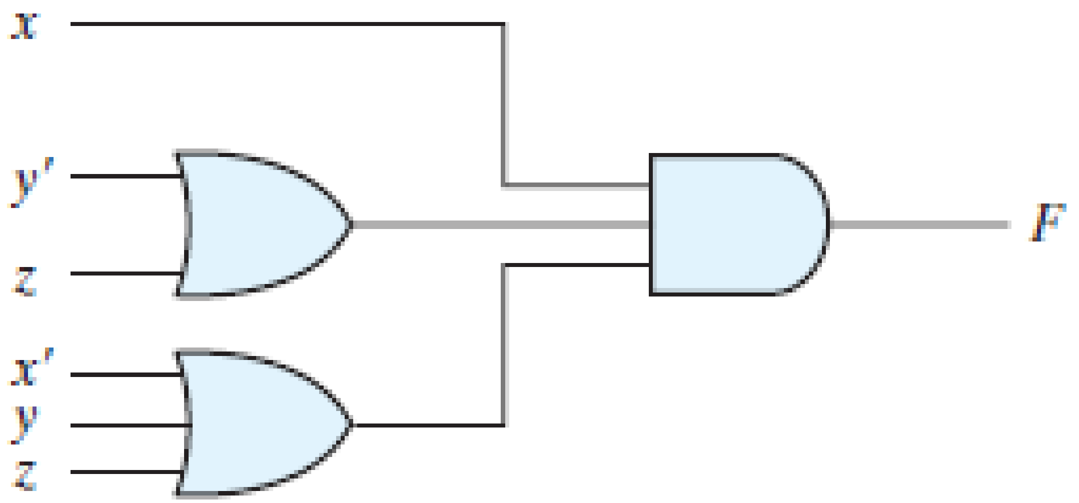
Boolean Functions, Canonical and Standard Form

❖ Product Of Sum: POS

➢ A product of sum (POS) is a Boolean expression containing OR terms, called sum terms. Each term may have any number of literals. The product denotes the ANDing of these terms

➢ Example 1 :

F = x.(y' + z).(x' + y + z)



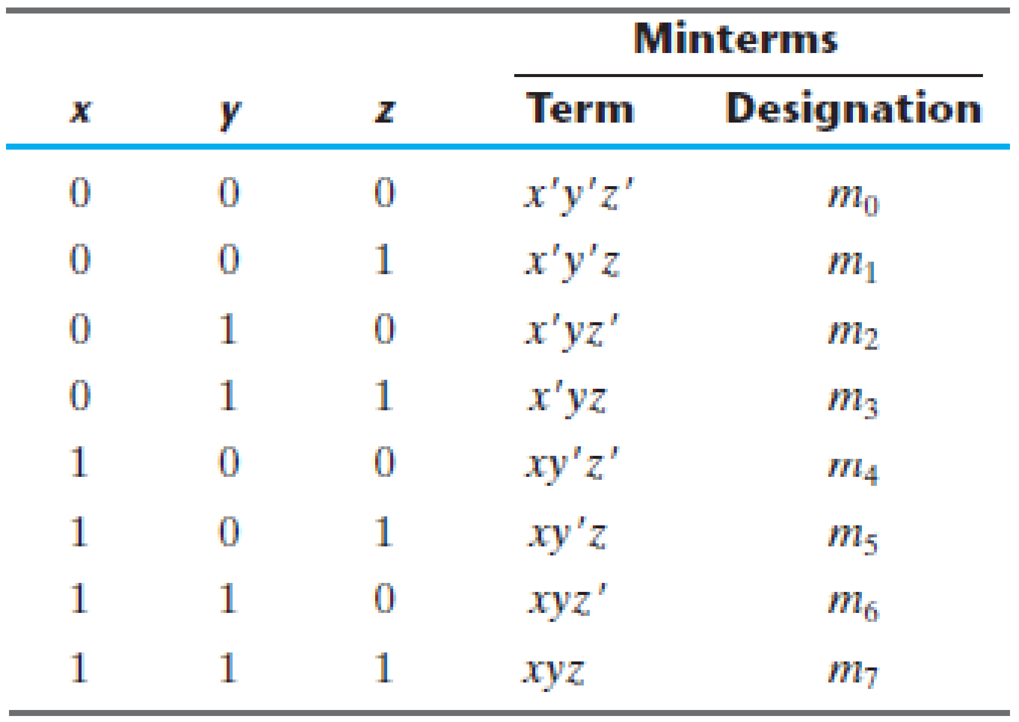
➢ Example 2: Y = (A+B). (C+D)
Implement the above POS using basic gates.

Boolean Functions, Canonical and Standard Form

❖ **Canonical SOP Form**:

- ➢ Each product term contains all the literals of that function either in true or complement form
- ➢ **Example:** F(A,B,C) = ABC + A'B'C + A'BC' +A'B'C'
- ➢ Each product term is called as **minterm**
- ➢ In a truth table the output value for these minterms is 1.

Truth Table for three variable function:



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

ELECTRONIC PRINCIPLES AND DEVICES

Boolean Functions, Canonical and Standard Form

❖ **Example:** Consider the functions f1 and f2 in the following truth table and write canonical SOP for the same.

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

❖ Canonical SOP Form for f1

f1 = x'y'z + xy'z' + xyz = m1 + m4 + m7

= Σ (1, 4, 7)

❖ Canonical SOP Form for f2

f2 = x'yz + xy'z + xyz' + xyz = m3 + m5 + m6 + m7

= Σ (3, 5, 6, 7)

Boolean Functions, Canonical and Standard Form

❖ **Canonical POS Form**:

- Each sum term contains all the literals of that function either in true or complement form
- **Example:** F(A,B,C) = (A +B+C) (A'+B'+C) (A'+B+C')
- Each sum term is called as **Maxterm**
- For three variable function: Truth Table

			Maxterms	
x	y	z	**Term**	**Designation**
0	0	0	$x + y + z$	M_0
0	0	1	$x + y + z'$	M_1
0	1	0	$x + y' + z$	M_2
0	1	1	$x + y' + z'$	M_3
1	0	0	$x' + y + z$	M_4
1	0	1	$x' + y + z'$	M_5
1	1	0	$x' + y' + z$	M_6
1	1	1	$x' + y' + z'$	M_7

Boolean Functions, Canonical and Standard Form

❖ **Example:** Consider the functions f1 and f2 in the following truth table and write canonical POS for the same.

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

❖ Canonical POS Form for f1

$$f1 = (x + y + z)(x + y' + z)(x + y' + z')(x' + y + z')(x' + y' + z)$$
$$= M0. M2 .M3 . M5 . M6$$
$$= \pi (0, 2, 3, 5, 6)$$

❖ Canonical POS Form for f2

$$f2 = (x + y + z)(x + y + z')(x + y' + z)(x' + y + z)$$
$$= M0. M1. M2. M4$$
$$= \pi (0, 1, 2, 4)$$

Boolean Functions, Canonical and Standard Form

❖ **Example:** Find the Minterms for the given expression: (Convert into canonical SOP Form): $F = A + BC$

Solution:

$F = A.1.1 + BC.1$

$F = A.(B+B').(C+C'). + BC.(A+A')$

$F = A.(BC + BC' + B'C + B'C') \quad + ABC + A'BC$

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

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

$F = m7 + m6 + m5 + m4 + m3$

$F = \Sigma (3,4,5,6,7)$

Examples on Boolean Functions

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

1. $x(x' + y) = xx' + xy = 0 + xy = xy.$
2. $x + x'y = (x + x')(x + y) = 1(x + y) = x + y.$
3. $(x + y)(x + y') = x + xy + xy' + yy' = x(1 + y + y') = x.$
4. $$\begin{aligned} xy + x'z + yz &= xy + x'z + yz(x + x') \\ &= xy + x'z + xyz + x'yz \\ &= xy(1 + z) + x'z(1 + y) \\ &= xy + x'z. \end{aligned}$$
5. $(x + y)(x' + z)(y + z) = (x + y)(x' + z)$, by duality from function 4.

Examples on Boolean Functions

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

1. $F\,(X, Y, Z) = XYZ + \bar{X}\,Y + XY\bar{Z}$

$$= XY(Z + \bar{Z}) + \bar{X}\,Y$$

$$= XY + \bar{X}\,Y$$

$$= Y(X + \bar{X}\,)$$

$$= Y$$

Examples on Boolean Functions

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

2. $F(a, b, c) = (a + b + c')(a'b' + c)$

$= aa'b' + ac + a'b'b + bc + a'b'c' + cc'$

$= ac + bc + a'b'c'$

Example: Reduce the following Boolean expressions to three literals

3. $F(a, b, c) = a'c' + abc + ac'$

$= c' (a + a') + abc$

$= c' + abc = (c' + ab)(c' + c) = ab + c'$ (using distributive law)

PES UNIVERSITY
CELEBRATING 50 YEARS

ELECTRONIC PRINCIPLES AND DEVICES

Department of Electronics and Communication Engineering

Problems on Boolean algebra

❖ **Simplify the given Boolean expression and implement using basic gates**

F = A.B.C + A’ + A. B’.C

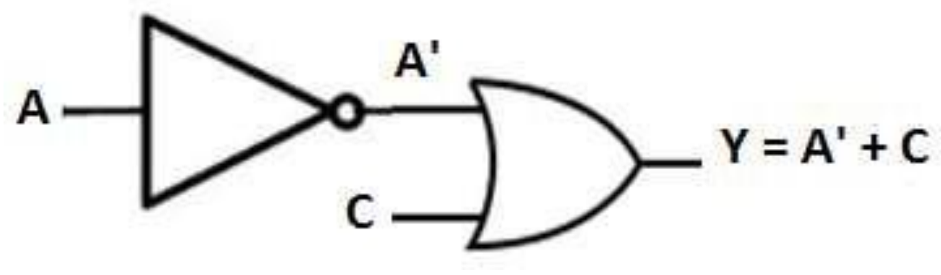
F = A.C (B + B’)+ A’ where B + B’ = 1

F = A.C + A’

Distributive Law
A + B.C = (A+B) . (A+C)

F = (A’+A) . (A’+ C)

F = A’ + C



Logic Diagram

Problems on Boolean algebra

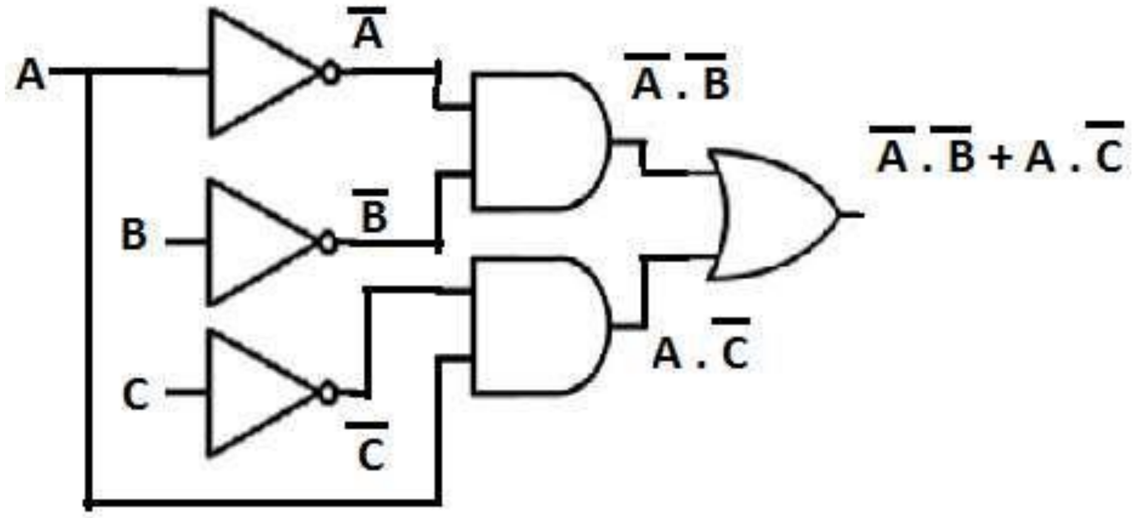
❖ **Simplify and realize the given function using basic gates:**

Y = A’B’C’ + AB’C’ + A’B’ + AC’

Y = B’C’ (A’ + A) + A’B’ + AC’

Y = B’C’ . (1) + A’B’ + AC’

By Consensus Theorem: **Y = A’B’ + AC’**


A
B
C
$\overline{A}$
$\overline{B}$
$\overline{C}$
$\overline{A}\,.\,\overline{B}$
$A\,.\,\overline{C}$
$\overline{A}\,.\,\overline{B} + A\,.\,\overline{C}$

Problems on Boolean algebra

❖ **Simplify the given Boolean Expression:**

$Y = A + A'.B + A'.B'.C + A'.B'.C'.D + A'.B'.C'.D'.E$

$Y = A + A' (B + B'.C + B'.C'.D + B'.C'.D'.E)$

Absorption Law: $A + A'.B = A + B$

$Y = A + B + B'.C + B'.C'.D + B'.C'.D'.E$

$Y = A + B + B' (C + C'.D + C'.D'.E)$

$Y = A + B + C + C'.D + C'.D'.E$

$Y = A + B + C + C' (D + D'.E)$

$Y = A + B + C + D + D'.E$

$\mathbf{Y = A + B + C + D + E}$

Problems on Boolean algebra

- Evaluate the Boolean function $Y = C + C'B + BA'$

  when $A = 1$, $B = 0$ and $C = 1$

$Y = 1 + 1'.0 + 0.1' \quad = 1 + 0 + 0$

$Y = 1$

Problems on Boolean algebra

❖ **Simplify the given Boolean expression and implement the circuit**

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

$F = AB + AB + AC + BB + BC$ Distributive Law

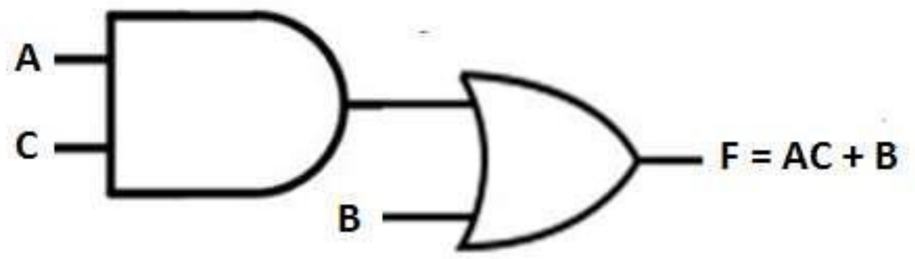
$F = AB + AC + B + BC$

$F = AB + AC + B(1 + C)$

$F = AB + AC + B$

$F = AC + B(A + 1)$

$F = AC + B$



Problems on Boolean algebra

❖ **Simplify and realize the given function using:** **(i) Basic Gates:**
(ii) NAND Gates

$Y = (A'.B + A' + A.B)'$

De Morgan's Theorem

$Y = (A'.B)' \;.\; (A')' \;.\; (A.B)'$

$Y = ((A')' + B')\;.\;(A)\;.\;(A' + B')$

$Y = (A + B')\;.\;((A.A') + A.B')$

$Y = (A + B')\;.\;(A.B')$

$Y = A.A.B' + A.B'.B'$

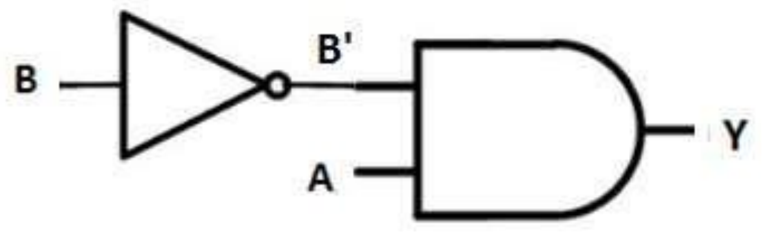
$Y = A.B' + A.B'$

$Y = A.B'$

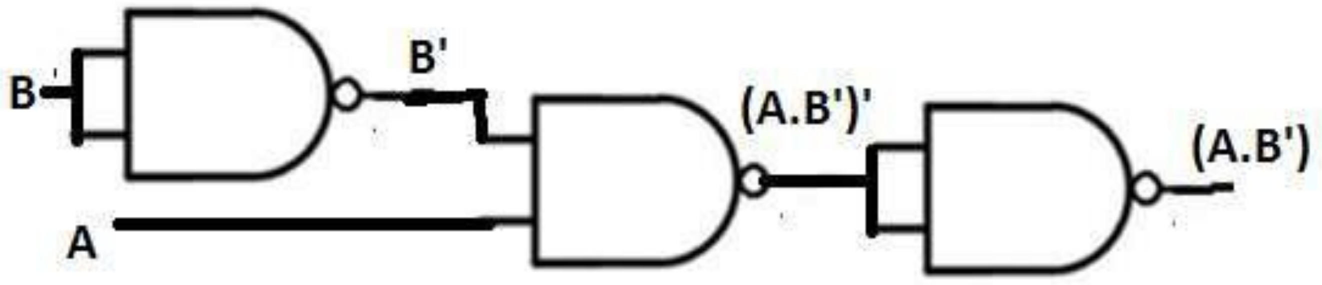
Problems on Boolean algebra

❖ Y = A.B’

❖ Using Basic Gates:



❖ Using NAND Gates:



PES UNIVERSITY
CELEBRATING 50 YEARS

THANK YOU

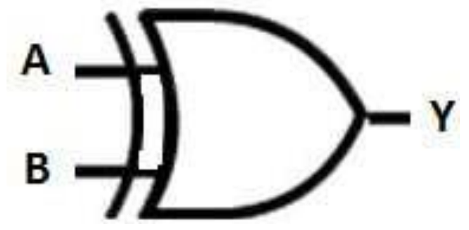
Department of Electronics and Communication

Other Logical Operation, Digital Logic gates (XOR and XNOR)

- ❖ **XOR Gate** is a digital logic Gate which has two or more inputs and only one output that performs Exclusive OR operation. Hence it is also called as Ex-OR or XOR
- ❖ For two input XOR gate output is **logic-1** only when one of its input is logic-1 (unequal input i.e..A = 0 and B = 1 or A = 1 and B = 0).
- ❖ Output of XOR gate is **logic-0** if both inputs are same (i.e.., A = 0 and B = 0 or A = 1 and B = 1)
- ❖ Truth Table of XOR Gate:

Input A	Input B	Output Y
0	0	0
0	1	1
1	0	1
1	1	0

- ❖ Logic Symbol



- ❖ Logical Expression

$Y = A.B' + A'.B$

$Y = A \text{ XOR } B$

$Y = A \oplus B$

Other Logical Operation, Digital Logic gates (XOR and XNOR)

❖ XOR gate performs modulo sum operation without including carry.
i.e., 0+0 = 0 , 0+1 = 1, 1+0 = 1, 1+1 = 0 (carry 1)

$A.B' + A'.B = A \oplus B$

LHS= $A.B' + A'.B$

Case1: If A = 0 and B = 0
$0.0' + 0'.0 = 0.1 + 1.0 = 0$

Case2: If A = 0 and B = 1
$0.1' + 0'.1 = 0.0 + 1.1 = 1$

Case3: If A = 1 and B = 0
$1.0' + 1'.0 = 1.1 + 0.0 = 1$

Case4: If A = 1 and B = 1
$1.1' + 1'.1 = 1.0 + 0.1 = 0$

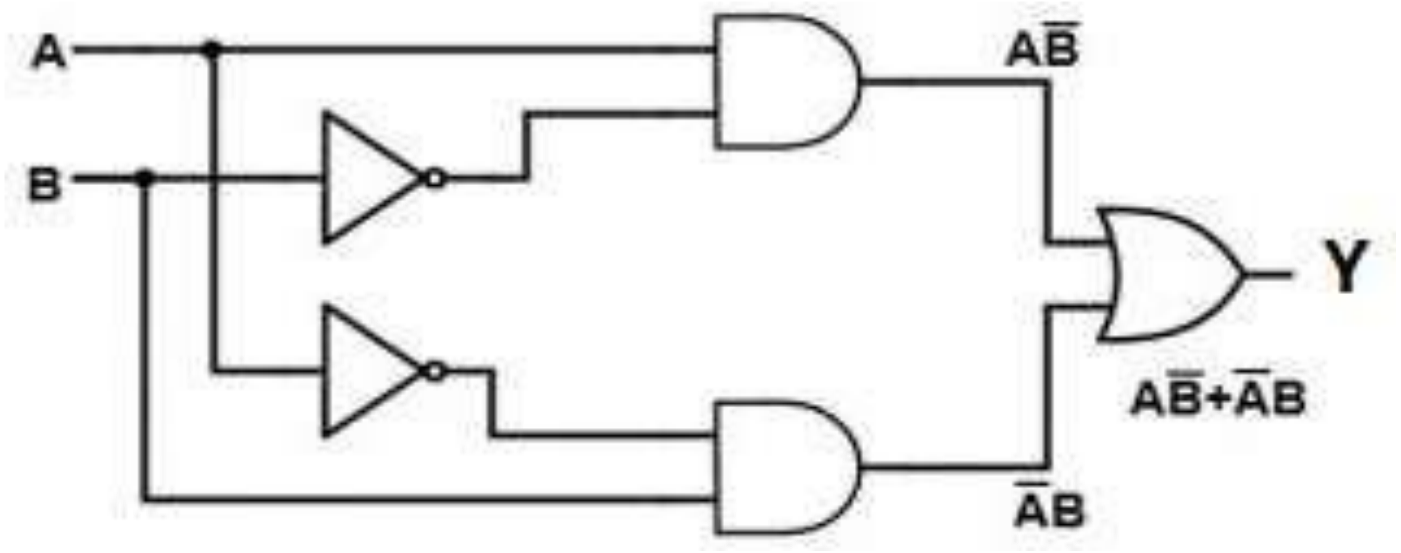
Hence,
$Y = A \oplus B = A.B' + A'.B$

Other Logical Operation, Digital Logic gates (XOR and XNOR)

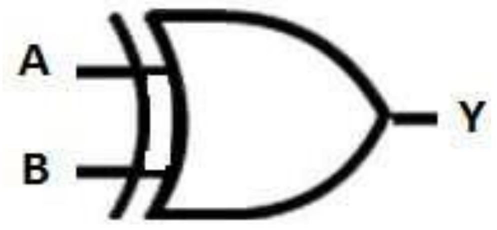
❖ Implementation of XOR Gate using basic gates:

$$Y = A \oplus B = A.B' + A'.B$$

❖ Logic Diagram



❖ Symbol



Other Logical Operation, Digital Logic gates (XOR and XNOR)

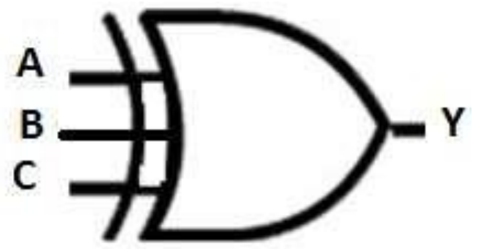
❖ **3 - Input XOR Gate:**

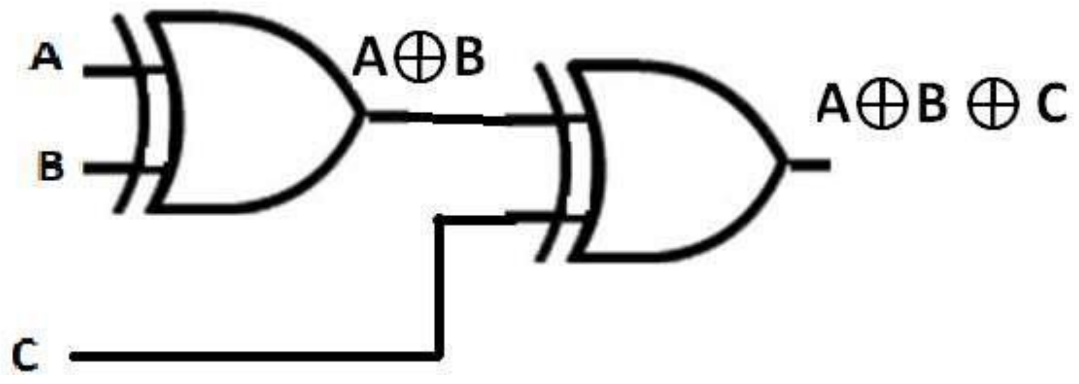
$$Y = A \oplus B \oplus C$$

Truth Table

A	B	C	$A \oplus B$	$A \oplus B \oplus C$
0	0	0	0	0
0	0	1	0	1
0	1	0	1	1
0	1	1	1	0
1	0	0	1	1
1	0	1	1	0
1	1	0	0	0
1	1	1	0	1

Logic Symbol





Associative Law:

$(A \oplus B) \oplus C = A \oplus (B \oplus C)$

❖ Even or odd Parity Bits

Other Logical Operation, Digital Logic gates (XOR and XNOR)

- **XNOR Gate**: Logical Complement of XOR Gate

- For two input XNOR Gate if both inputs are same i.e.., A = 0 and B = 0 or A = 1 and B = 1. Then output of logic gate is logic-1 (High). Output of XNOR is logic-0 if inputs are unequal.

- Equality detector.

Truth Table

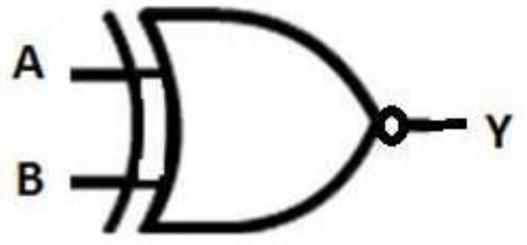
Input A	Input B	Output Y
0	0	1
0	1	0
1	0	0
1	1	1

- **Boolean Expression**

$$Y = A.B + A'.B'$$

$$Y = A \odot B$$

- **Symbol**



Other Logical Operation, Digital Logic gates (XOR and XNOR)

❖ XNOR Gate:

$Y = (A \oplus B)'$

$Y = (A.B' + A'.B)'$

Using De Morgan's Theorem

$Y = (A.B')' . (A'.B)'$

$Y = (A'+B) . (A + B')$

$Y = A'.A + A'.B' + A.B + B.B'$

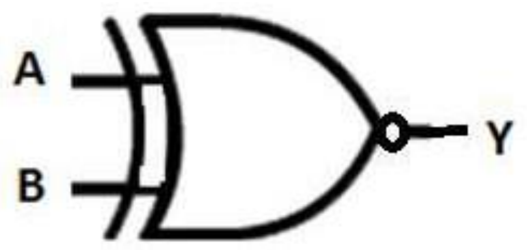
$Y = 0 + A'.B' + A.B + 0$

$Y = A'.B' + A.B$

$Y = (A \oplus B)' = A \odot B$

Y = (A XOR B)' = (A XNOR B)

❖ Symbol:

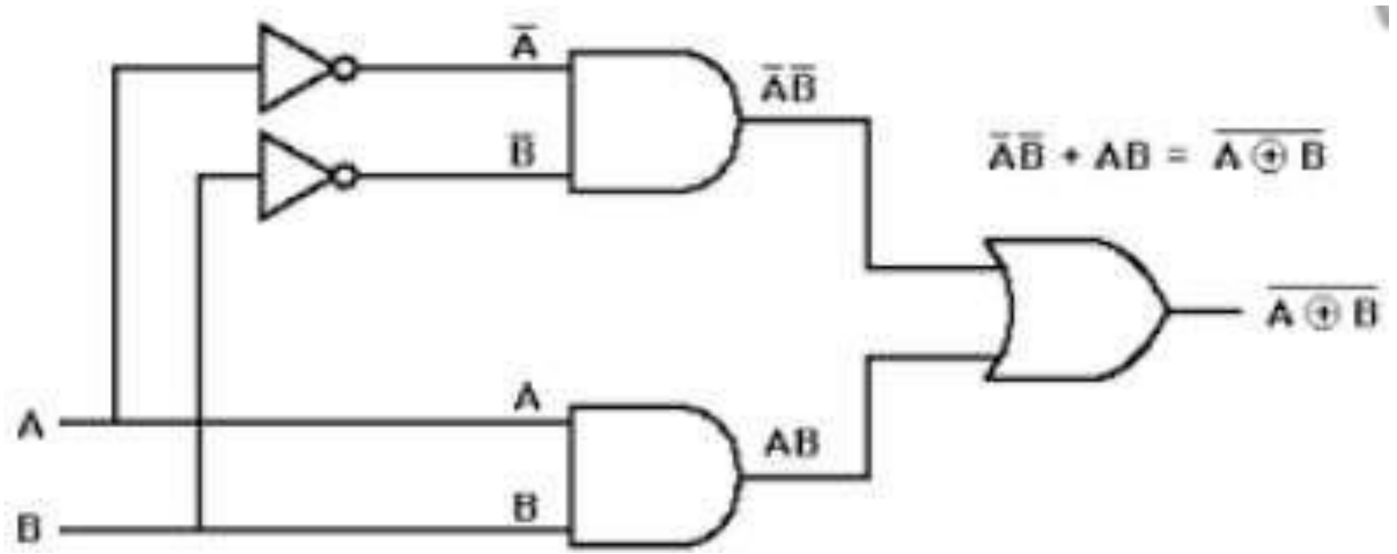


Other Logical Operation, Digital Logic gates (XOR and XNOR)

❖ **Implement the XNOR Gate using basic gates:**

Y = (A.B’ + A’. B)’

❖ **Logic Diagram:**

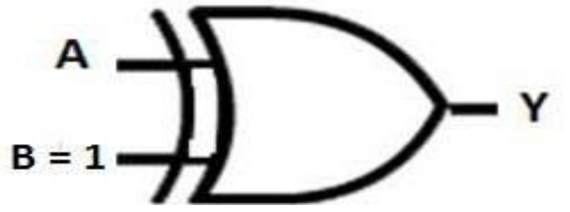


Y = (A⨁B)’ = A⨀B

Other Logical Operation, Digital Logic gates (XOR and XNOR)

❖ Properties of XOR Gate:

- Identity element: $A \oplus 0 = A$
- $A \oplus 1 = A'$
- $A \oplus A = 0$
- Commutative Law: $A \oplus B = B \oplus A$
- Associative Law : $A \oplus (B \oplus C) = (A \oplus B) \oplus C$



Input A	Input B	Output Y
0	0	0
0	1	1
1	0	1
1	1	0

Other Logical Operation, Digital Logic gates (XOR and XNOR)

❖ Applications of XOR and XNOR

- XOR gate is used in processor’s Arithmetic Logic Unit (ALU) for binary addition.
- XOR logic gate is used to generate pseudorandom numbers in hardware.
- To generate parity bits and error detection
- Equality detector

PES UNIVERSITY

CELEBRATING 50 YEARS

THANK YOU

Department of Electronics and Communication

Realization of Boolean expression using Universal Gates

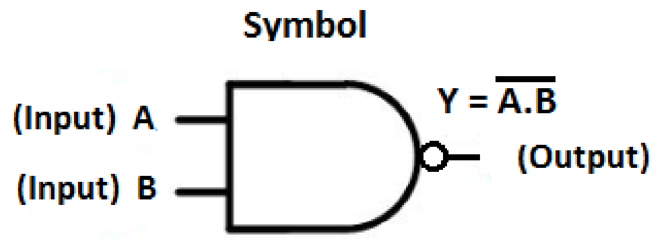
- **Universal Gates** : (i) NAND Gate
  (ii) NOR Gate

- Any digital logic circuit can be implemented by using NAND or NOR logic gates.

- NAND and NOR gates are easier to fabricate with electronic components and are used in all Integrated Circuit (IC’s) digital logic families.

Realization of Boolean expression using Universal Gates

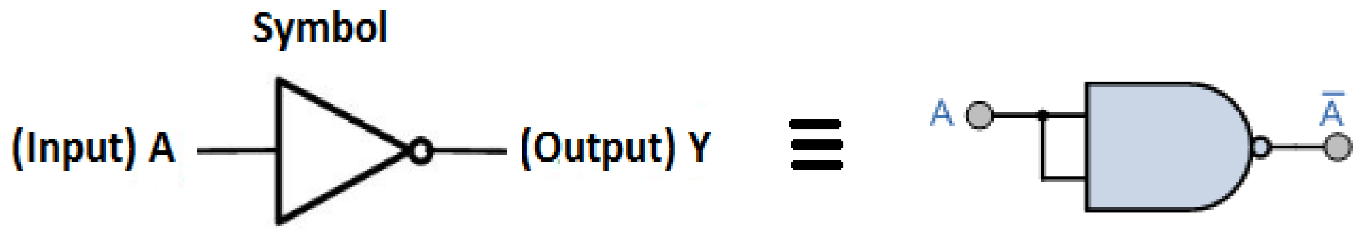
❖ Realization of logic gates using NAND Gates:



Truth Table

NAND		
Input		Output
A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

❖ NOT Gate:

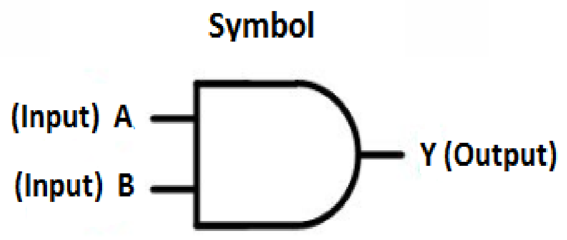


Y = (A.A)’ = (A)’ = A’

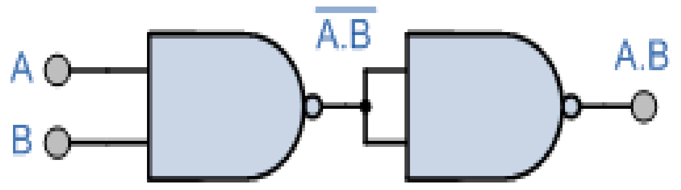
Reference: https://www.electronics-tutorials.ws/logic/universal-gates.html

Realization of Boolean expression using Universal Gates

❖ AND Gate:

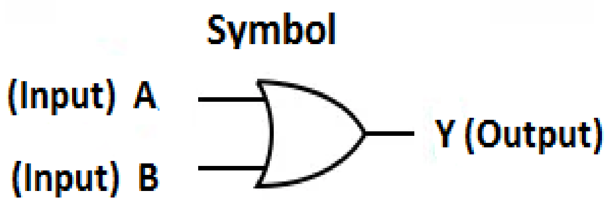


≡

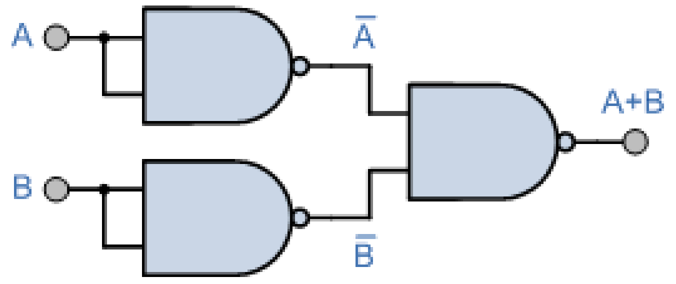


$Y = ((A.B)')' = A.B$

❖ OR Gate:



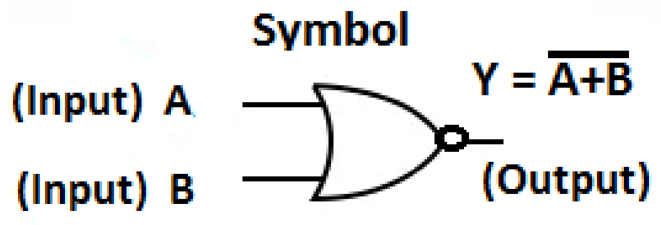
≡



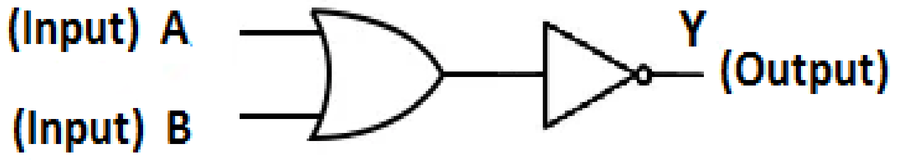
$Y = ((A+B)')' = (A' . B')' = A+B$

Realization of Boolean expression using Universal Gates

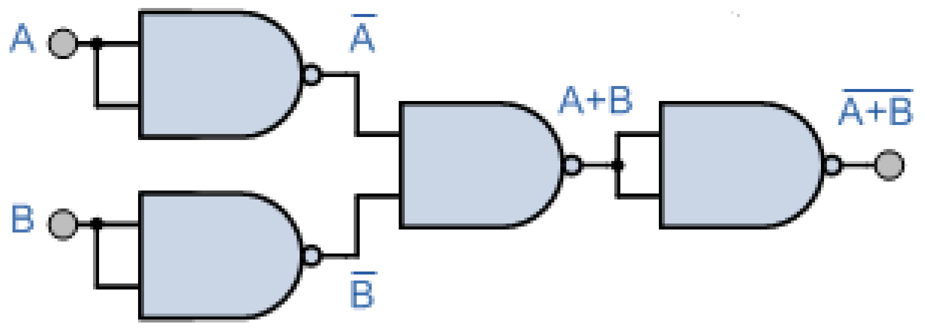
❖ NOR Gate:



≡

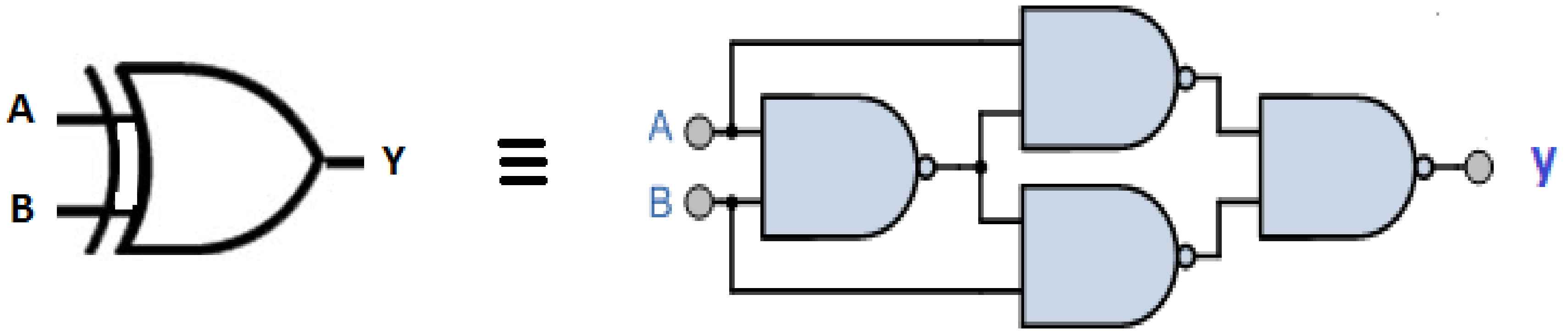


≡



Realization of Boolean expression using Universal Gates

❖ XOR Gate:



Y = A ⊕ B

$Y = A.B' + A'B = A\,(A.B)' + B.\,(A.B)'$

$Y = ((A\,(A.B)' + B.\,(A.B)')')'$

$Y = (\ (A.\,(A.B)')' \,.\, (B.(A.B)')'\ )'$

$Y = A\,(A.B)' + B.\,(A.B)'$

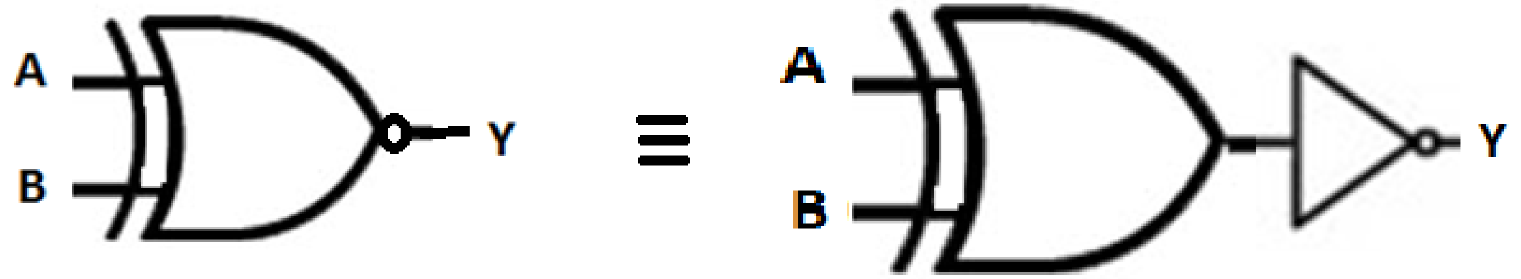
$Y = A\,(A' + B') + B.\,(A' + B')$

$Y = A.A' + A.B' + A'.B + B.B'$

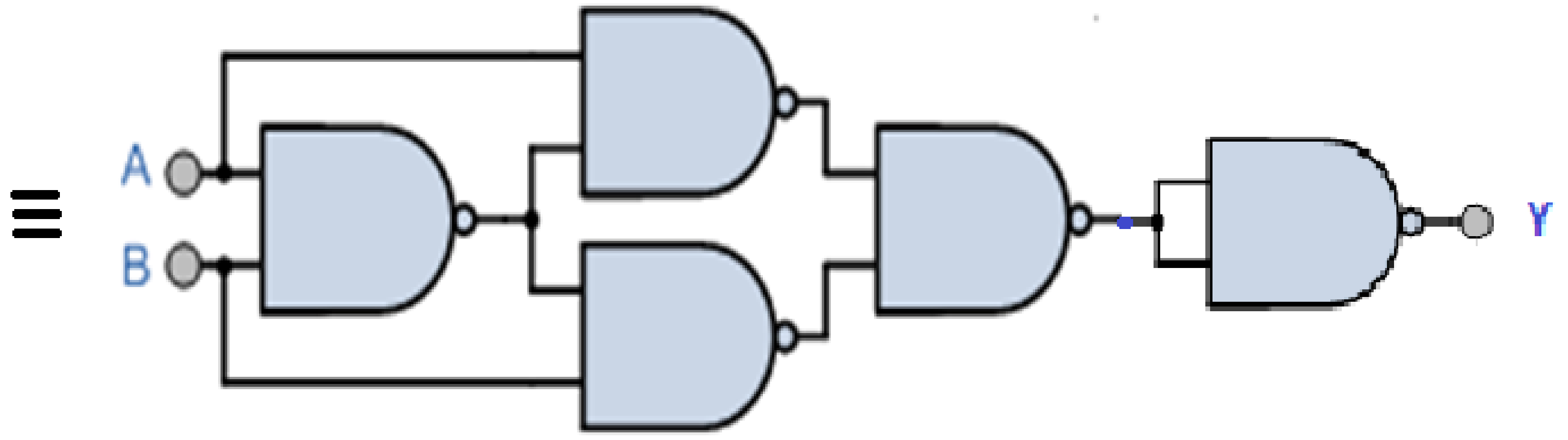
$Y = A.B' + A'.B$

Realization of Boolean expression using Universal Gates

❖ **XNOR Gate**:

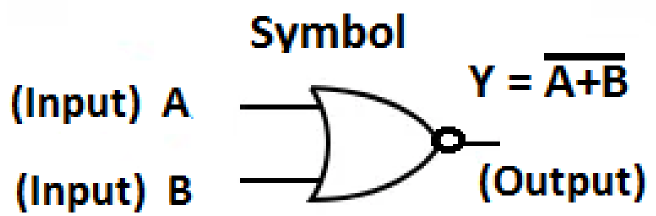


Y = (A⊕B)' = A⊙B



Realization of Boolean expression using Universal Gates

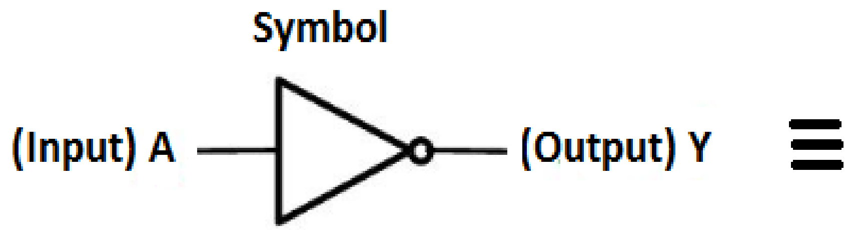
❖ Realization of logic gates using NOR Gates:

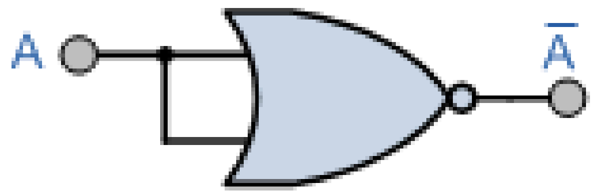


Truth Table

NOR		
Input		Output
A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

❖ NOT Gate:

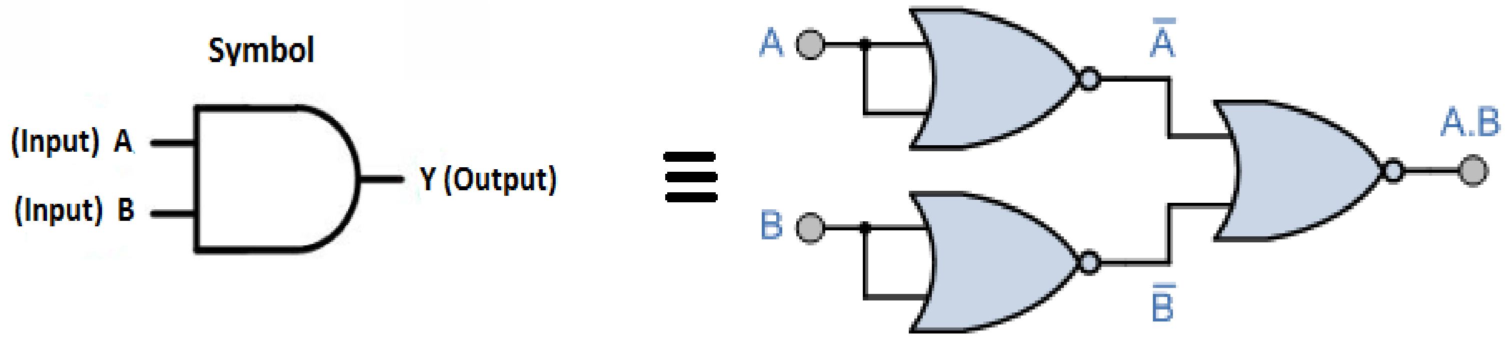




Y = (A + A)’ = (A)’ = A’

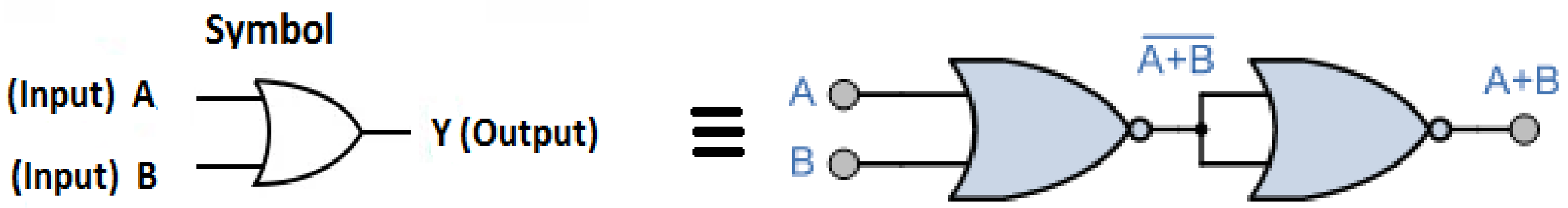
Realization of Boolean expression using Universal Gates

❖ AND Gate:



Y = ((A.B)’)’ = (A’ + B’)’ = A.B

❖ OR Gate:

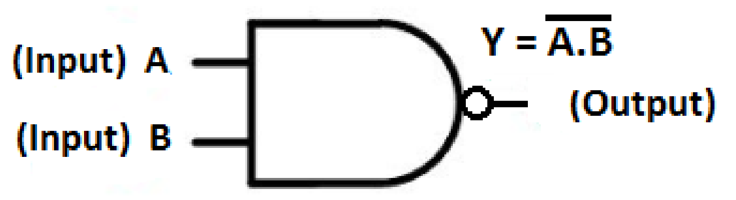


Y = ((A+B)’)’ = A + B

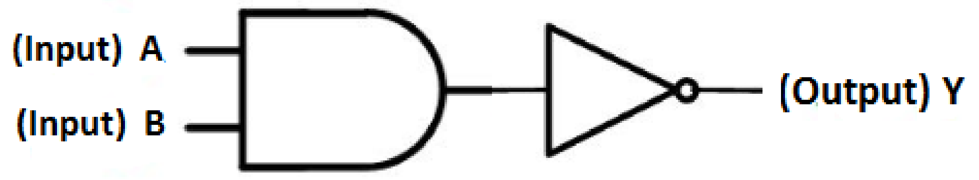
Realization of Boolean expression using Universal Gates

❖ NAND Gate:

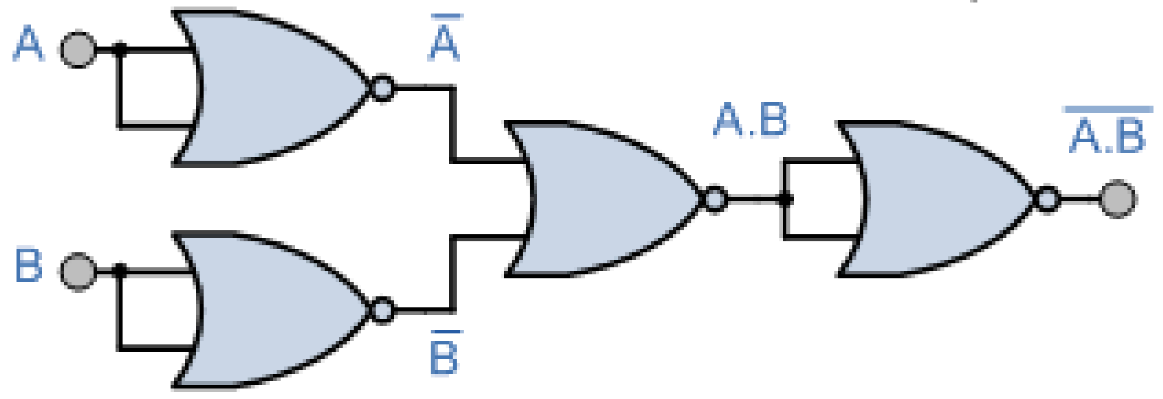
Symbol



≡

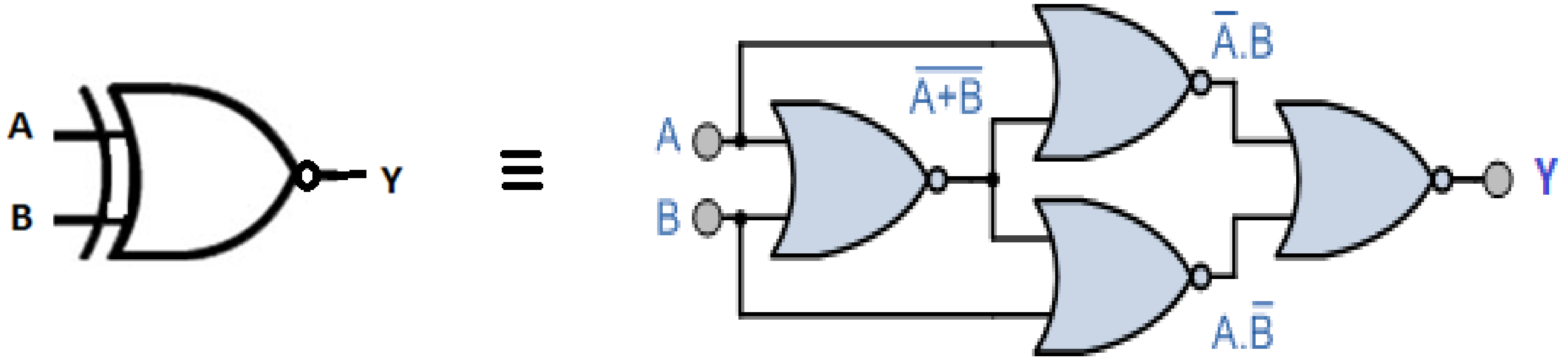


≡



Realization of Boolean expression using Universal Gates

❖ **XNOR Gate**:



$Y1 = ((A+B)' + A)' = (A+B).\ A' = A'.B$

$Y2 = ((A+B)' + B)' = (A+B).\ B' = A.B'$

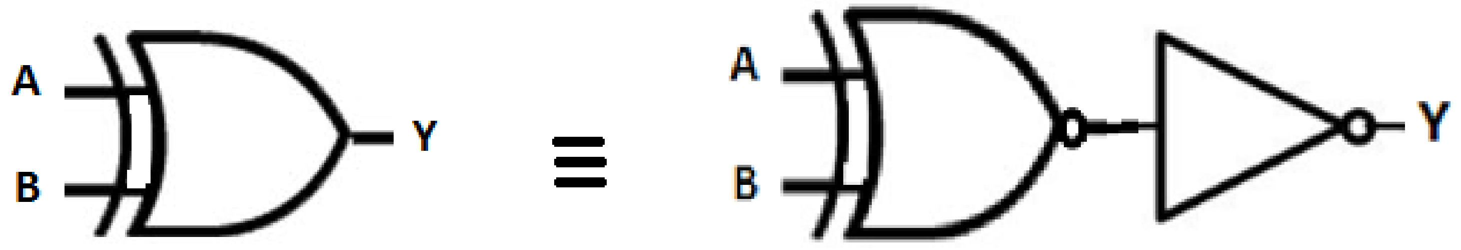
$Y = (A'.B + A.B')' = (A'.B)' \ . \ (A.B')' = (A + B')\ .\ (A' + B)$

$Y = A.B + A'.B'$

$Y = A \odot B$

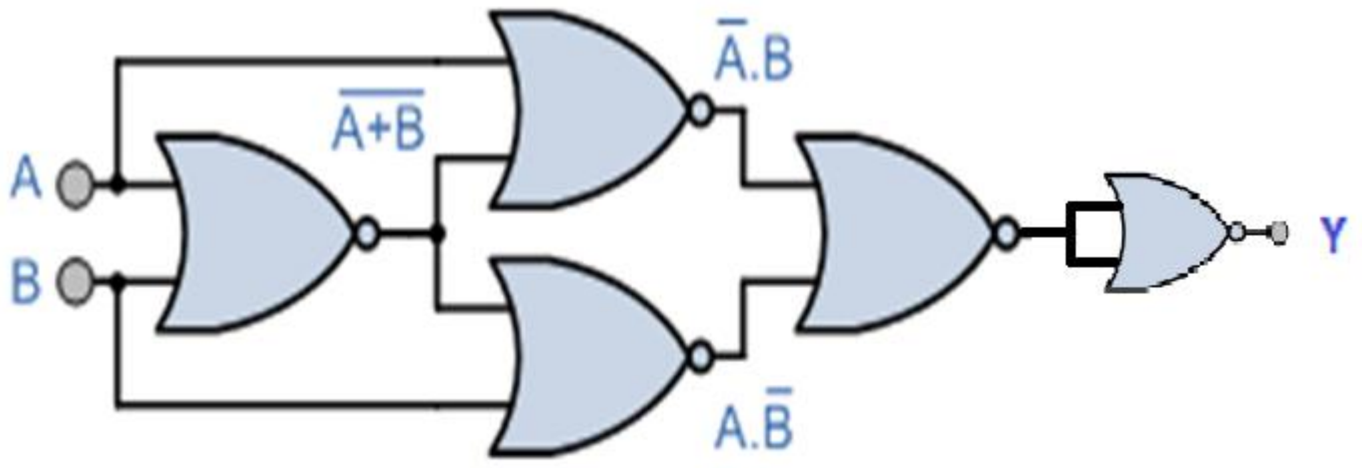
Realization of Boolean expression using Universal Gates

❖ **XOR Gate:**



Y = (A⊙B)’ = A⊕B

≡



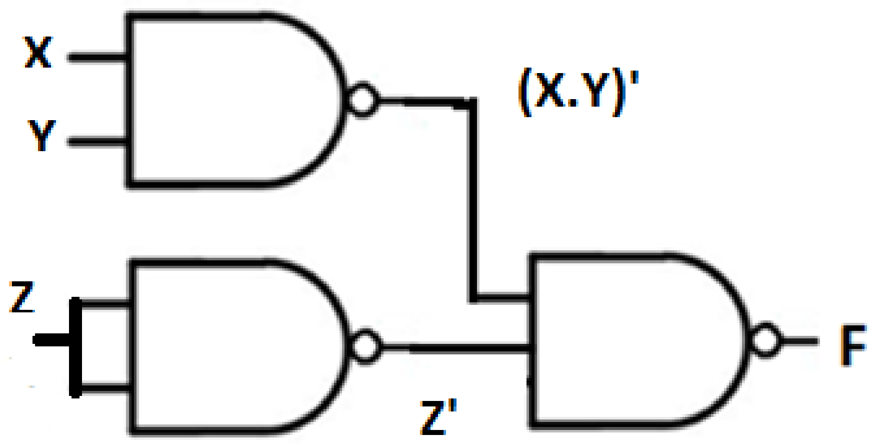
Realization of Boolean expression using Universal Gates

❖ **Implement the given function using NAND Gates only.**

F = X.Y + Z

F = ((X.Y + Z)’)’

F = ((X.Y)’ . Z’))’



THANK YOU

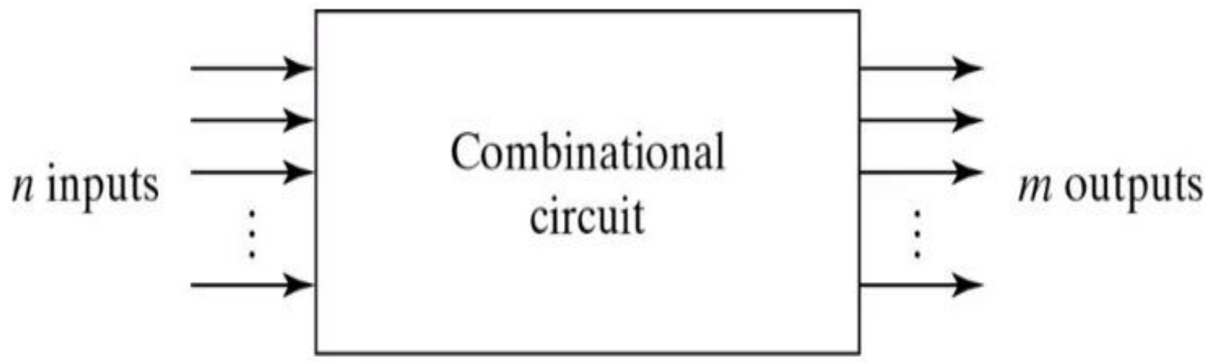
Prof.

Department of Electronics and Communication Engineering

@pes.edu

Combinational Logic Circuits: Half Adder and Full adder

❖ **Combinational circuits** are constructed by interconnection of logic gates.
whose outputs at any time are determined from only the present combination of inputs



❖ A combinational circuit performs an operation that can be specified logically by a set of **Boolean functions**

❖ Examples: Binary Adders, Multiplexers, etc.

Combinational Logic Circuits: Half Adder and Full adder

❖ The basic arithmetic operation is the addition of two binary digits.

❖ Addition of binary bits : 0 + 0 = 0,
0 + 1 = 1,
1 + 0 = 1,
1 + 1 = 10.

Examples:

$$\begin{array}{rl} 0 & \Rightarrow \text{Carry} \\ 11 & \\ +\ 10 & \\ \hline 101 & \Rightarrow \text{Sum} \end{array} \qquad \begin{array}{r} 1 \\ 64 \\ +\ 28 \\ \hline 92 \end{array}$$

❖ Binary Adders:

- **Half Adder** : A combinational circuit that performs the addition of two bits.
- **Full Adder:** A combinational circuit that performs the addition of three bits

❖ Half Adder and Full Adder are the basic components of Adders such as: Ripple carry adder, Carry look ahead adder, other fast adders.

Reference: "Digital Design with an Introduction to Verilog HDL" M Morris Mano, Michale D Ciletti

Combinational Logic Circuits: Half Adder and Full adder

❖ Half Adder

- ➢ x and y are the two binary inputs
- ➢ Sum (s) and Carry (c) are the two binary outputs

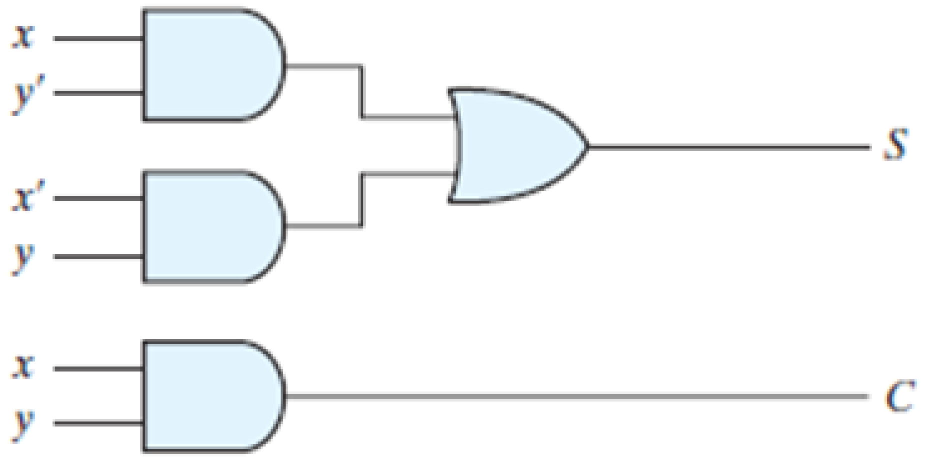
Boolean Expression:

s = x'.y + x.y'

c = x.y

Truth Table

x	y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0



Logic Diagram

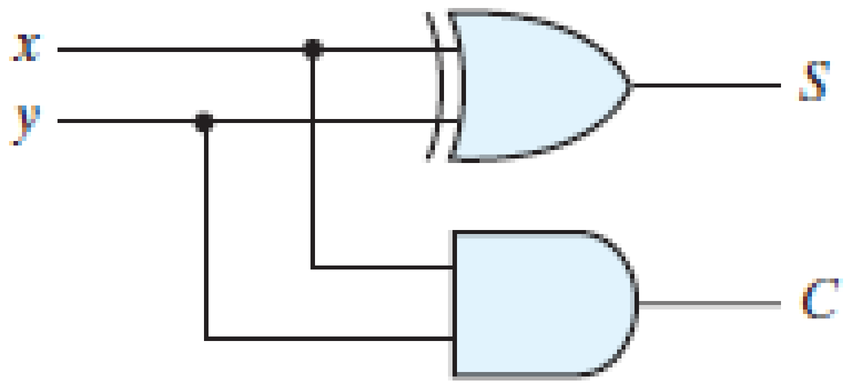
Combinational Logic Circuits: Half Adder and Full adder

❖ **Half Adder:** $s = x \oplus y$

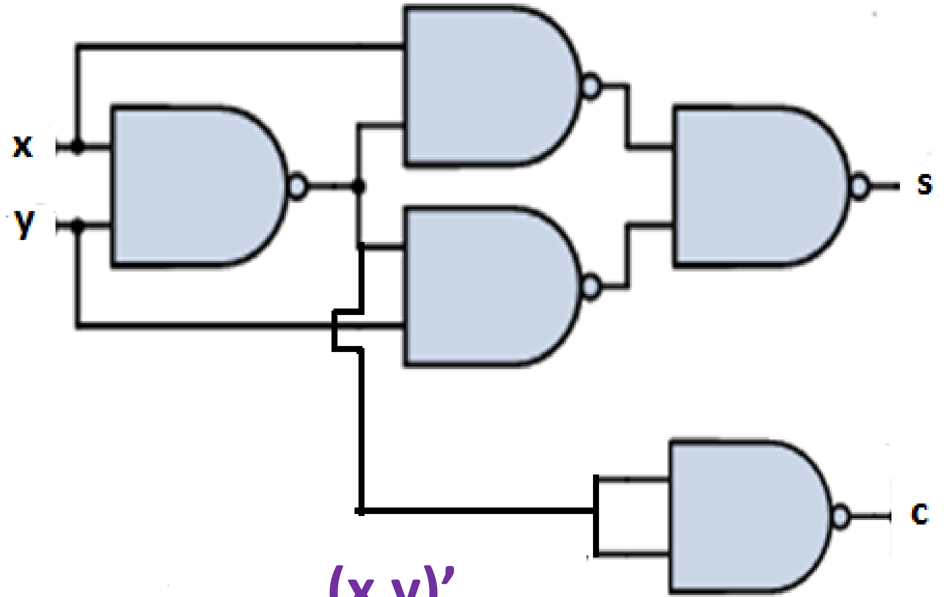
$c = x.y$

Sum expression: $s = x'.y + x.y'$

❖ **Half Adder using Logic Gates**



❖ **Half Adder using NAND Gates only**



(x.y)’

Combinational Logic Circuits: Half Adder and Full adder

❖ Full Adder:

- x, y and z are three binary inputs.
- S is sum and C is carry outputs

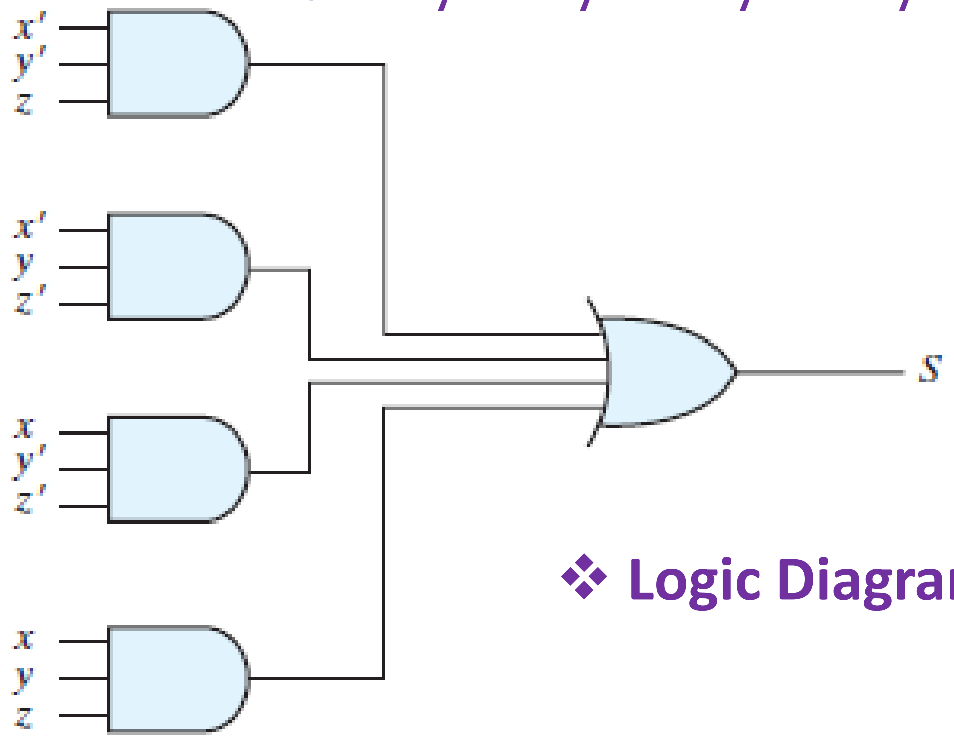
❖ Truth Table:

x	y	z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

❖ Boolean Expression:

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

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



❖ **Logic Diagram**

Reference: “Digital Design with an Introduction to Verilog HDL” M Morris Mano, Michale D Ciletti

Combinational Logic Circuits: Half Adder and Full adder

❖ Full Adder:

➢ **Carry Expression**:

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

C = x'yz + xy'z + xy (z'+z)

C = x'yz + xy'z + xy

C = x'yz + x(y'z + y)   Absorption Law

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

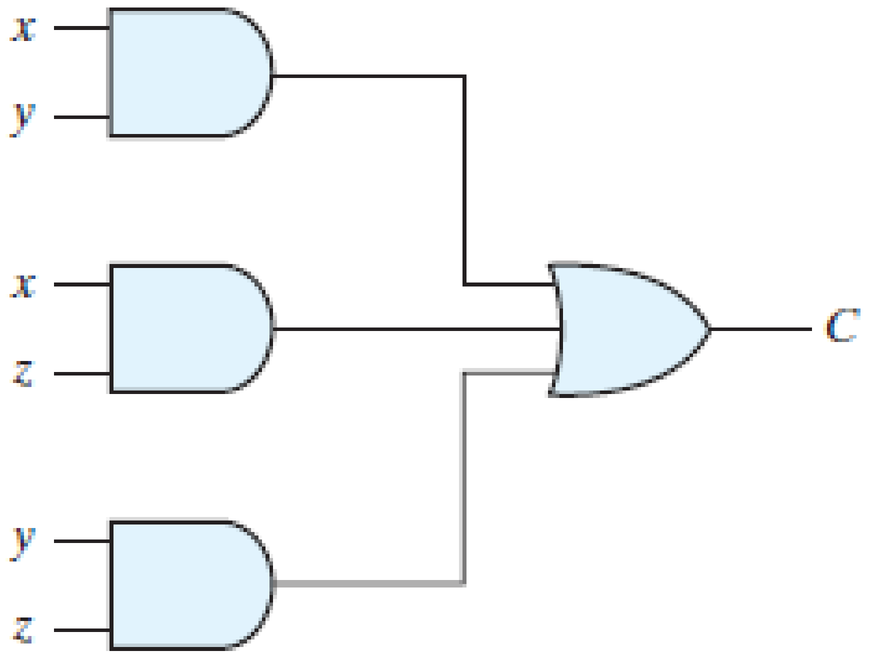
C = x'yz + xz + xy

C = z(x'y + x) + xy

C = z(y + x) + xy

C = yz + xz + xy

❖ **Logic Diagram**



Reference: "Digital Design with an Introduction to Verilog HDL" M Morris Mano, Michale D Ciletti

Combinational Logic Circuits: Half Adder and Full adder

❖ **Boolean Expression for Sum:**

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

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

$S = x'(y \oplus z) + x((y \oplus z)')$

$\mathbf{S = x \oplus (y \oplus z)}$

❖ **Boolean Expression for Carry:**

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

$C = z.(x'y + xy') + xy(z' + z)$

$C = z.(x \oplus y) + xy.(1)$

$\mathbf{C = (x \oplus y).z + xy}$

Reference: "Digital Design with an Introduction to Verilog HDL" M Morris Mano, Michale D Ciletti

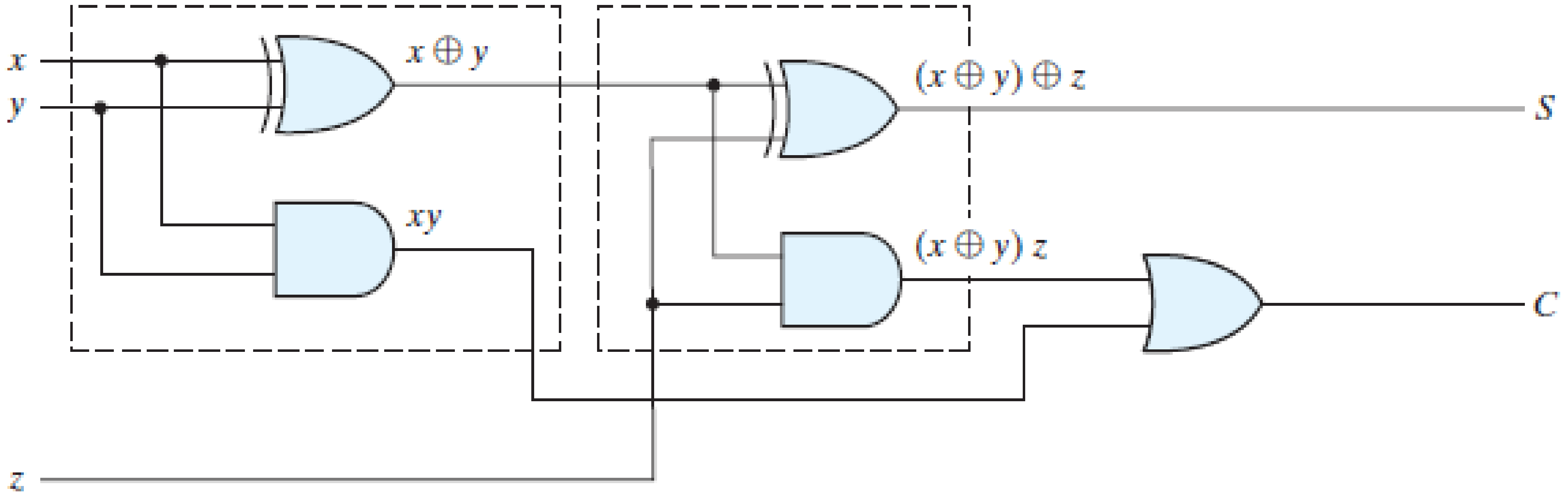
Combinational Logic Circuits: Half Adder and Full adder

❖ Full Adder Boolean Expression:

$S = (x \oplus y) \oplus z$

$C = (x \oplus y).z + x.y$

❖ Implementation of full adder with two half adders and an OR gate



Reference: "Digital Design with an Introduction to Verilog HDL" M Morris Mano, Michale D Ciletti

Combinational Logic Circuits: Half Adder and Full adder

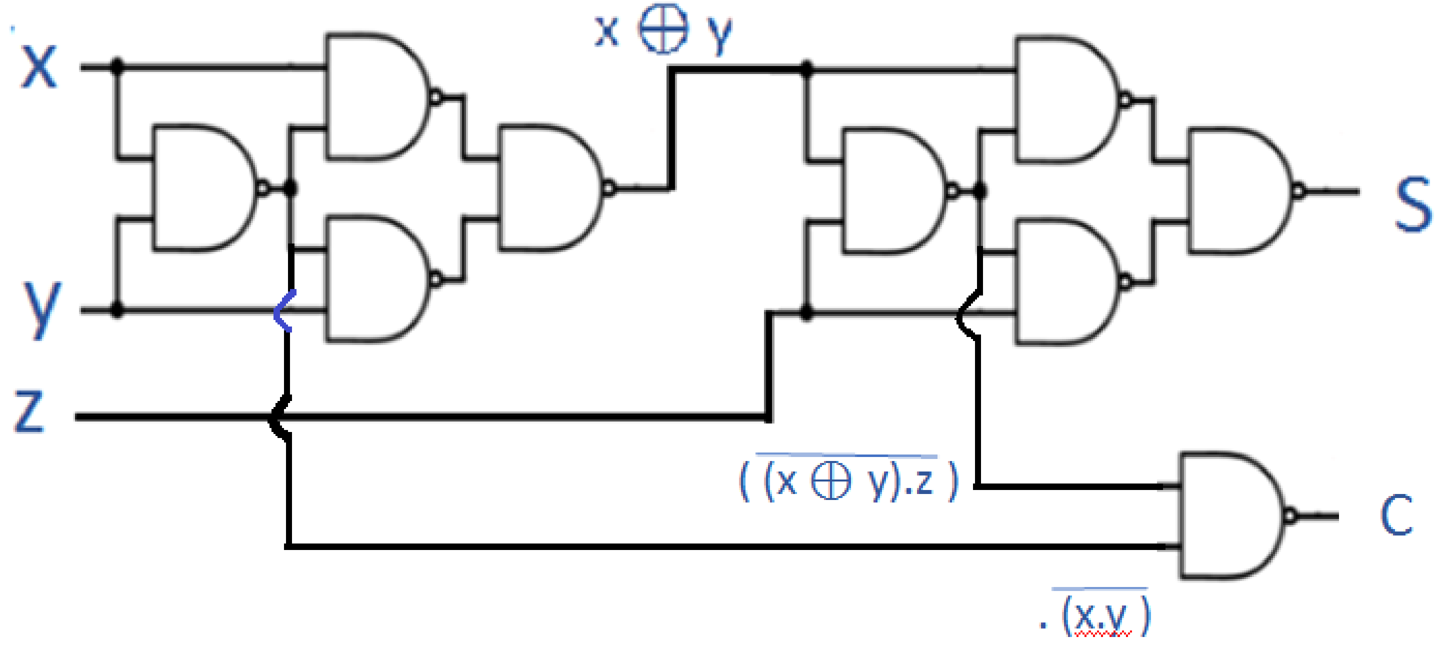
- Boolean Expression: $S = (x \oplus y) \oplus z$

  $C = (x \oplus y).z + x.y$

- Full adder circuit using NAND Gates

$$C = \overline{\overline{((x \oplus y).z + x.y)}}$$

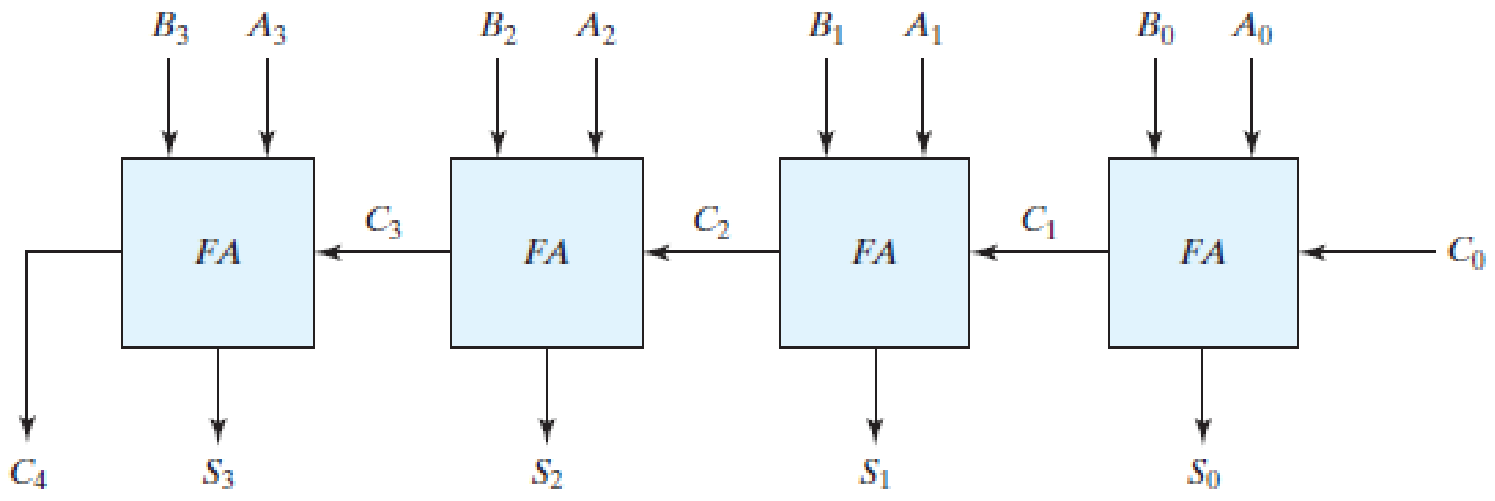
$$C = \overline{(\overline{(x \oplus y).z}) . \overline{(x.y)}}$$



Combinational Logic Circuits: Half Adder and Full adder

❖ Four-bit adder

Using four Full adder Ripple adder circuit is constructed.



❖ Example

```
       1110
  A =  1101
+ B =  0111
      -----
      10100
```

Reference: “Digital Design with an Introduction to Verilog HDL” M Morris Mano, Michale D Ciletti

Combinational Logic Circuits: Half Adder and Full adder

Summary:

- Half Adder Circuits: (i) Using Basic Gates
  (ii) Using NAND Gates

- Full Adder Circuits: (i) Using Basic Gates
  (ii) Using NAND Gates

THANK YOU

Prof.

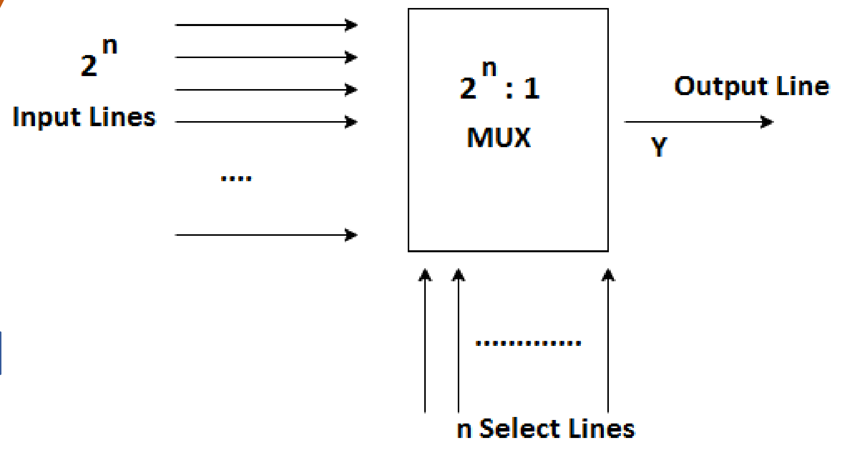
Department of Electronics and Communication Engineering

@pes.edu

Multiplexer and Demultiplexer

❖ Multiplexer is a combinational circuit that has **multiple inputs and a single output.** The select line determines which input line is connected to the output.

❖ MUX has **2^n input lines** and **n selection lines** whose bit combinations determine which input is selected to the **output line Y**

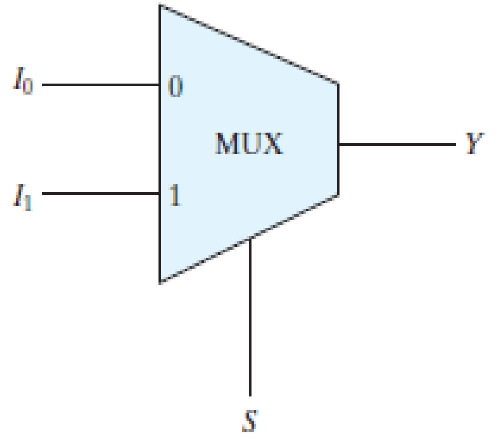


❖ Main application of MUX is data compression, and · shares a single transmission channel

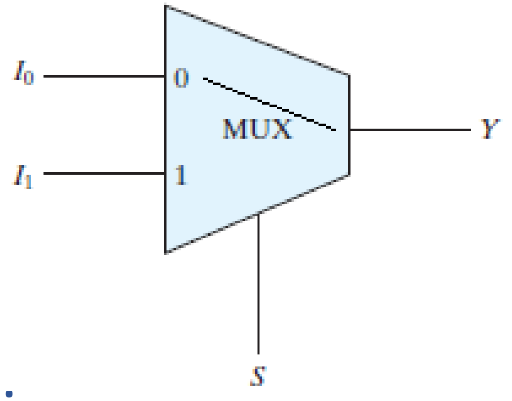
❖ The function of a de-multiplexer is to inverse the function of the multiplexer.

❖ Two-to-one-line multiplexer: (2:1 MUX)

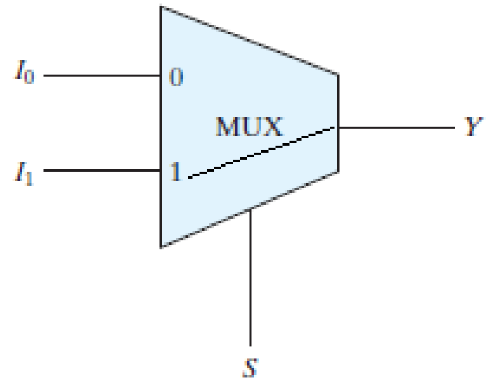
Block Diagram



When S = 0 then Y = Io



When S = 1, then Y = I₁



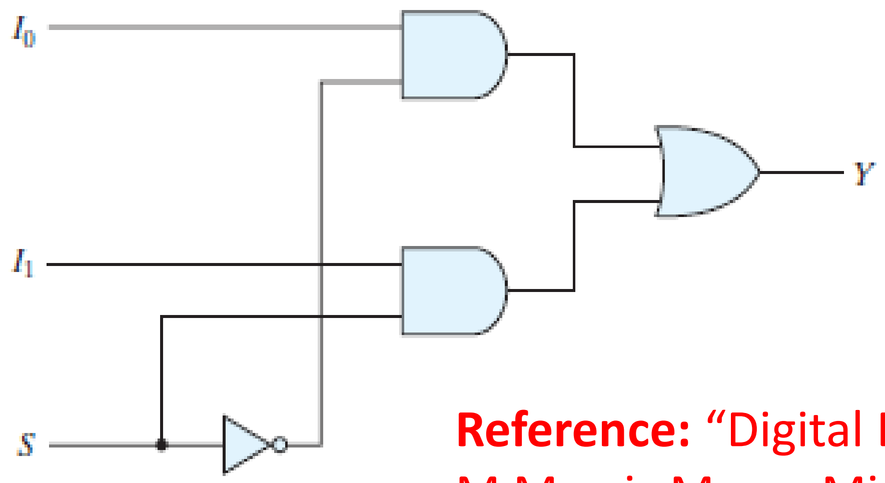
Functional Table

S	Y
0	I_0
1	I_1

Boolean Expression:

$Y = S'. I_0 + S. I_1$

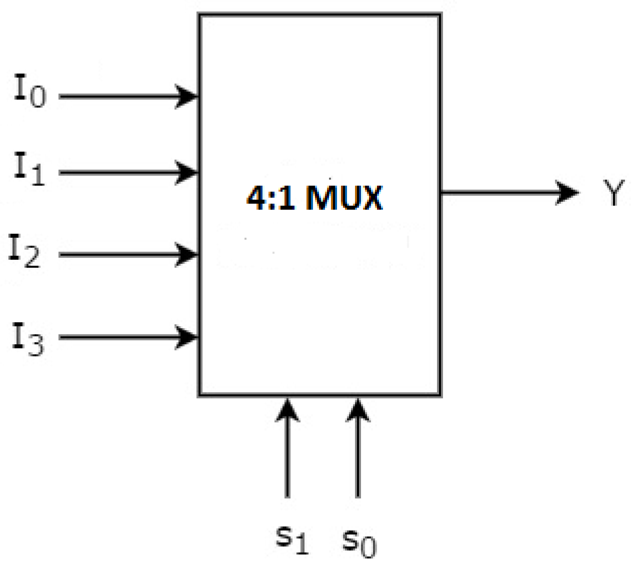
Logic Diagram:



Reference: "Digital Design with an Introduction to Verilog HDL" M Morris Mano, Michale D Ciletti

❖ Four-to-one-line multiplexer: (4:1 MUX)

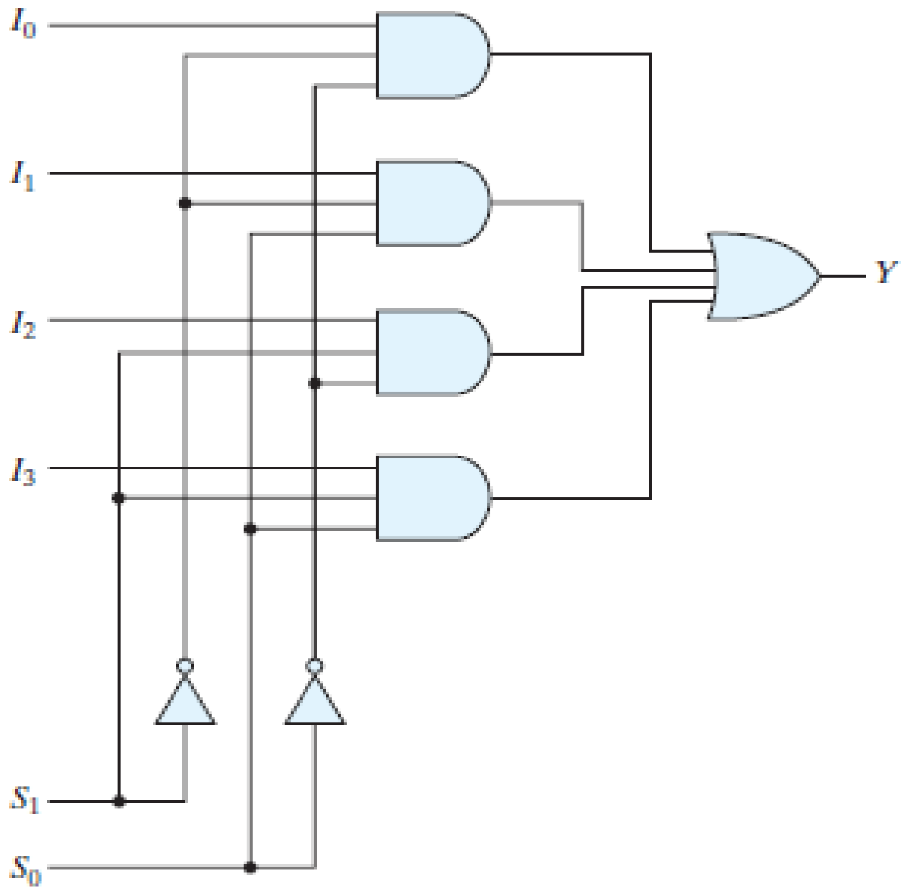
Block Diagram



Function Table

S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

Logic Diagram

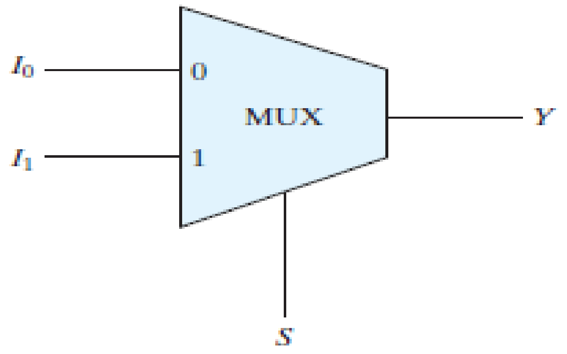


Boolean Function:

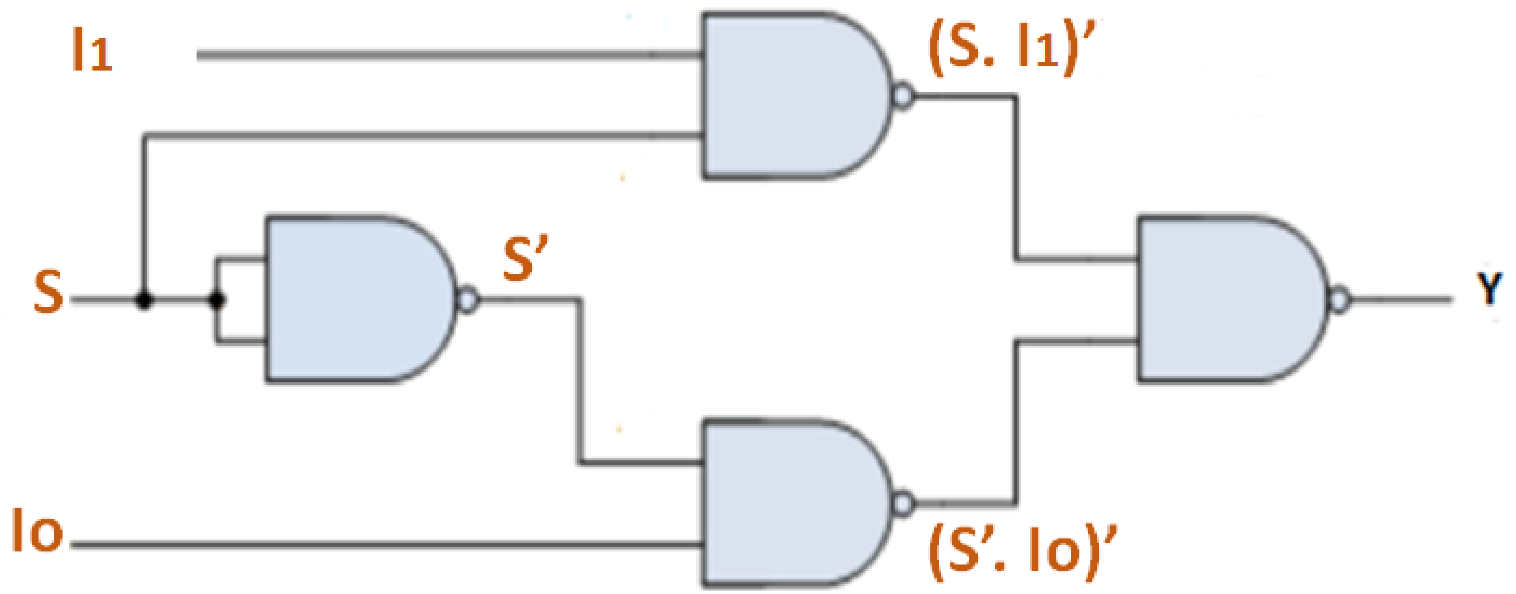
$$Y = S_1{}'S_0{}'I_0 + S_1{}'S_0I_1 + S_1S_0{}'I_2 + S_1S_0I_3$$

2:1 Multiplexer using NAND Gates only

Block Diagram



Logic Diagram



Boolean Expression

$Y = S'. I_0 + S. I_1$

$Y = ((S'. I_0 + S. I_1)')'$

From De Morgans Law

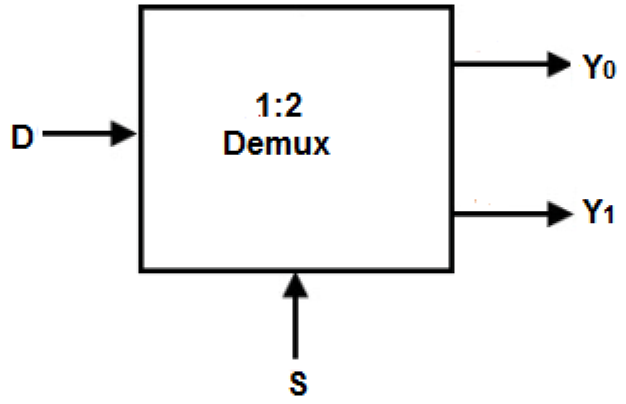
$Y = ((S'. I_0)' . (S. I_1)')'$

1: 2 Demultiplexer

❖ Demultiplexer is a combinational circuit that takes data from one input line and distributes over multiple output lines.

❖ The select input decides which output line is connected to the input data line.

Block Diagram



Function Table

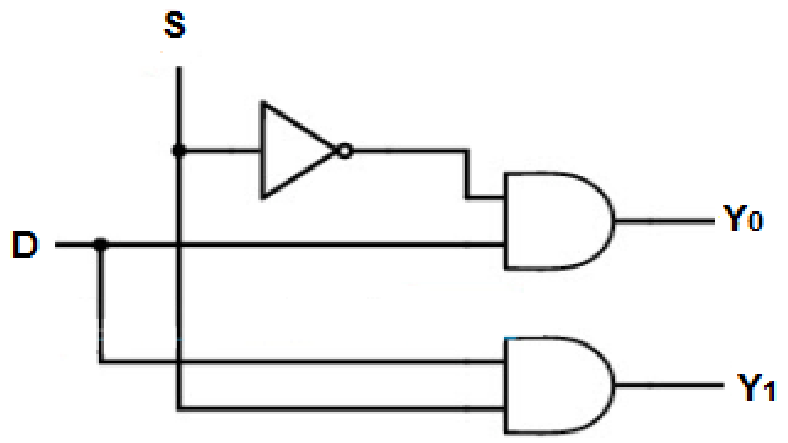
Select Input	Outputs	
S	Y_0	Y_1
0	D	-
1	-	D

Boolean Functions

$Y_0 = S'.D$

$Y_1 = S.D$

Logic Diagram



1:2 Demultiplexer using NAND Gates

Boolean Functions

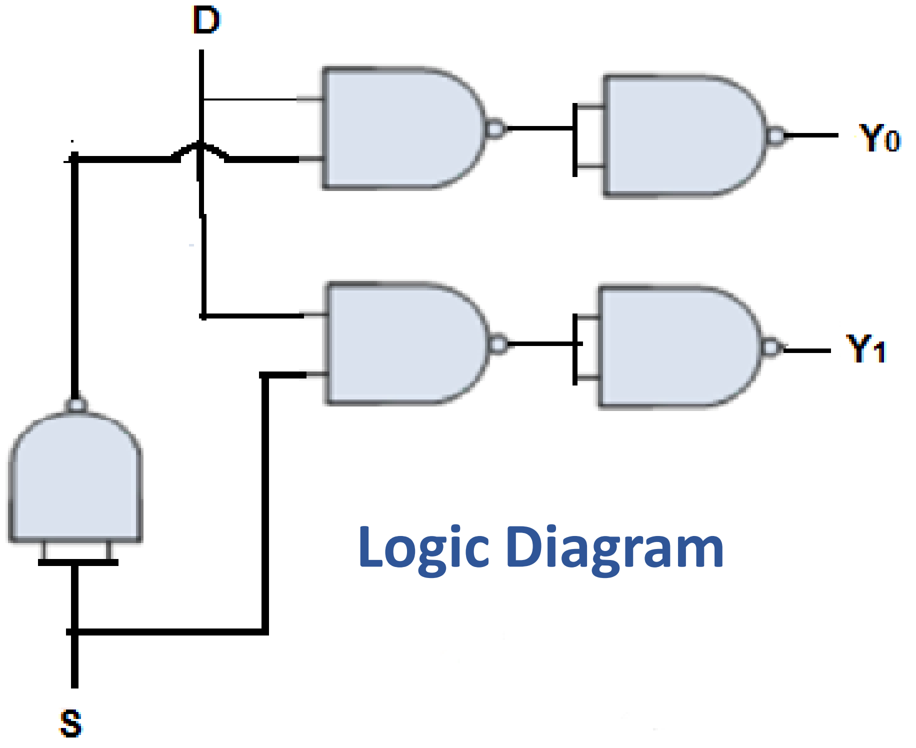
$Y_0 = S'.D$

$Y_1 = S.D$

Boolean Functions

$Y_0 = ((S'.D)')'$

$Y_1 = ((S.D)')'$



Logic Diagram

❖ Applications: communication system for data transmission, serial to parallel converter

Reference: "Digital Design with an Introduction to Verilog HDL" M Morris Mano, Michale D Ciletti

THANK YOU

Prof.

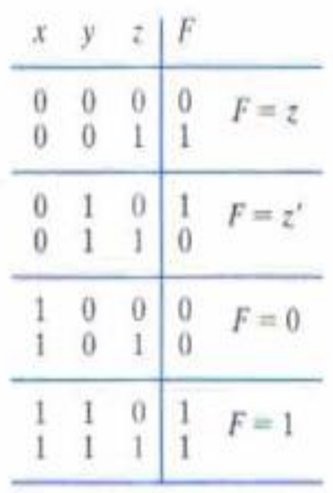
Department of Electronics and Communication Engineering

@pes.edu

Multiplexer and Demultiplexer

❖ Implementation of a Boolean function using MUX : ***F(x, y, z)*** = **Z (1,2,6,7)**

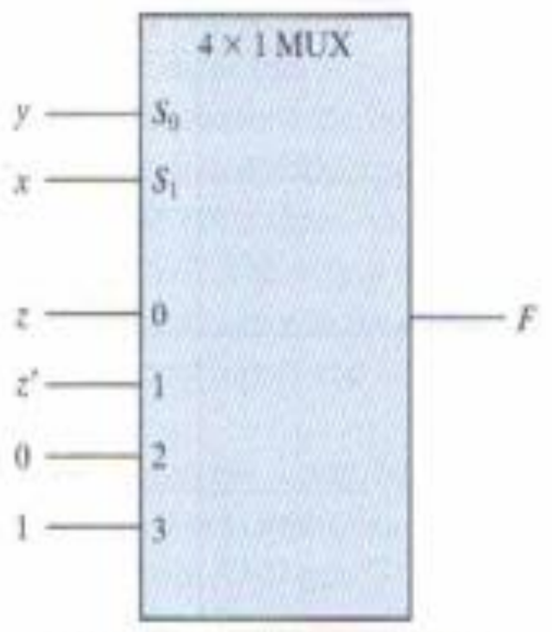
Truth Table



x	y	z	F	
0	0	0	0	F = z
0	0	1	1	
0	1	0	1	F = z'
0	1	1	0	
1	0	0	0	F = 0
1	0	1	0	
1	1	0	1	F = 1
1	1	1	1	

(a) Truth table

Multiplexer Implementation



(b) Multiplexer implementation

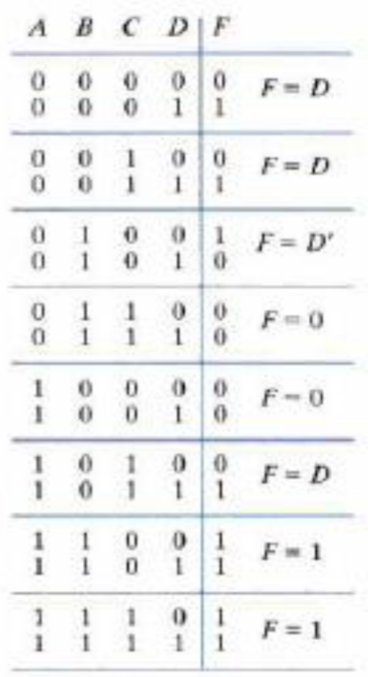
FIGURE 4.27
Implementing a Boolean function with a multiplexer

The function of three variables can he implemented with a four-to-one-line multiplexer as shown in Fig 4.27.
The two variables x and y are applied to the selection lines in the order; x is connected to the S1 input and y to the So input. The values for the data input lines are determined from the truth table of the function.
When xy = 00, output F is equal to z because F = 0 when z = 0 and F = 1 when z = I.
This requires that variable z be applied to data input 0. The operation of the multiplexer is such that when xy = 00, data input 0 has a path to the output, and that maker F equal to z.
In a similar fashion, we can determine the required input to data lines 1, 2, and 3 from the value of F when xy = 01, 10, and 11, respectively. This particular example shows all four possibilities that can be obtained for the data inputs

Multiplexer and Demultiplexer

❖ Implementation of a Boolean function using MUX : $F(A, B, C, D) = \Sigma(1, 3, 4, 11, 12, 13, 14, 15)$

Truth Table



A	B	C	D	F	
0	0	0	0	0	$F = D$
0	0	0	1	1	
0	0	1	0	0	$F = D$
0	0	1	1	1	
0	1	0	0	1	$F = D'$
0	1	0	1	0	
0	1	1	0	0	$F = 0$
0	1	1	1	0	
1	0	0	0	0	$F = 0$
1	0	0	1	0	
1	0	1	0	0	$F = D$
1	0	1	1	1	
1	1	0	0	1	$F = 1$
1	1	0	1	1	
1	1	1	0	1	$F = 1$
1	1	1	1	1	

Multiplexer Implementation

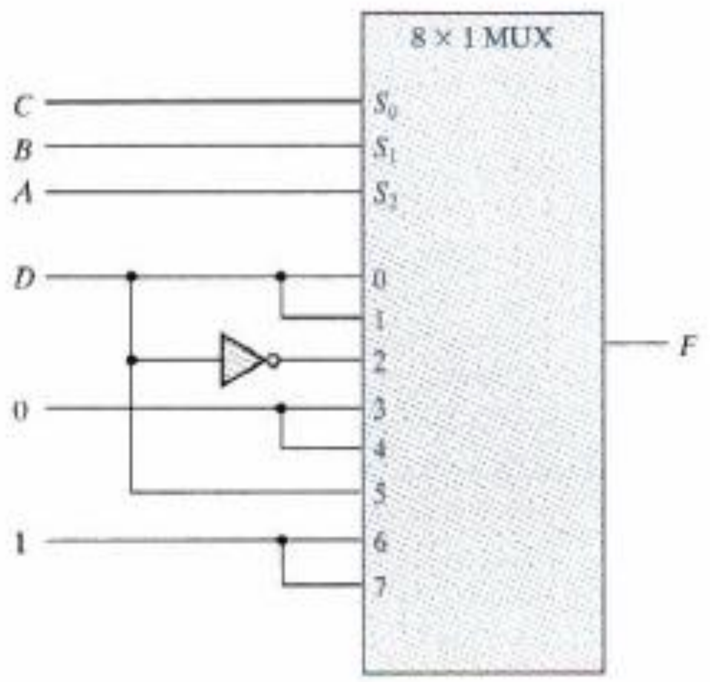


FIGURE 4.28
Implementing a four-input function with a multiplexer

This function is implemented with a multiplexer with three selection inputs as shown in Fig 4.28
Note that the first variable A must be connected to selection input S2 so that A,B, and C correspond to selection inputs S2, S1, and So, respectively.
The values for the data inputs are determined from the truth table listed in the figure.
The corresponding data line number is determined from the binary combination of ABC.
For example the table shows that when ABC = 101, F = D, so the input variable D is applied to data input 5.
The binary constants 0 and 1 correspond to two fixed signal values.
When integrated circuits are used, logic 0 corresponds to signal ground and logic 1 is equivalent to the power signal, depending on the technology (e.g., 5 volts).

Multiplexer and Demultiplexer

❖ Three states Gates :

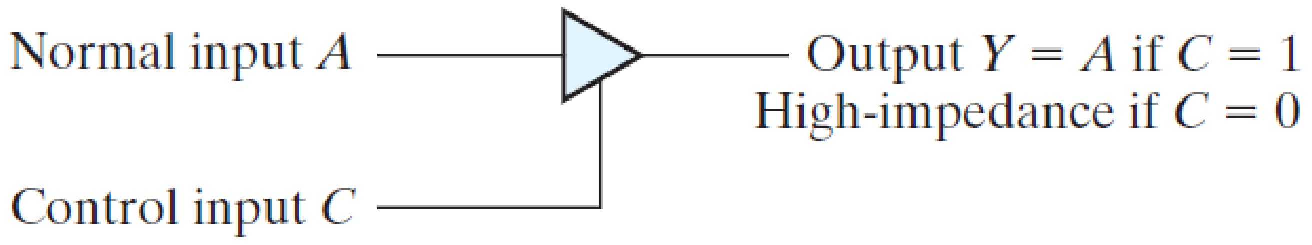
A multiplexer can be constructed with three state gates that digital circuits exhibit three states. Two of the states are signals equivalent to logic 1 and logic 0 as in a conventional gate.
The third state is a high-impedance state in which

(1) the logic behaves like an open circuit, which means that the output appears to be disconnected,
(2) the circuit has no logic significance , and
(3) the circuit connected to the output of the three-state gate is not affected by the inputs to the gate.

Three-state gate may perform any convention logic, such as; AND or NAND. However the one most commonly used is the buffer gate.

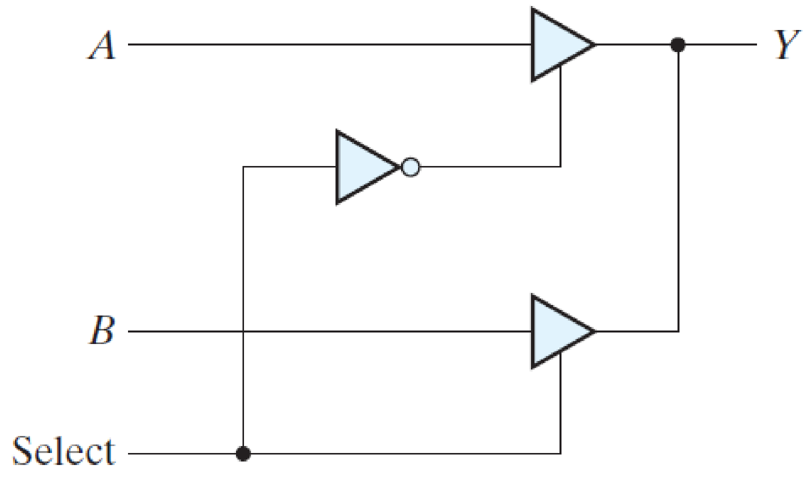
❖ Three states Gates :

- The graphic symbol for a three gate buffer gate is shown in the following Figure.
- It is distinguished from a normal buffer by an Input control Line entering the bottom of the symbol.
- The buffer has a normal input. An output and a control input that determine the state of the output.
- When the control Input is equal to I. be output is enabled and the gate behaves like a conventional Buffer, with the output equal to the normal input.
- When the control input is 0, the output is disabled and the gate goes to a high-impedance state, regardless of the value in the normal input.
- The high--impedance state of a three-state gate provides, a special feature not available in other gates. Because of this features, large number of three-state gate outputs can be connected with wires to form a common line without endangering loading effects.



❖ Three states Gates :

The following Figure shows the construction of a two-to-one-line multiplexer with 2 three-state buffers and an inverter.



(a) 2-to-1-line mux

THANK YOU

Prof.

Department of Electronics and Communication Engineering

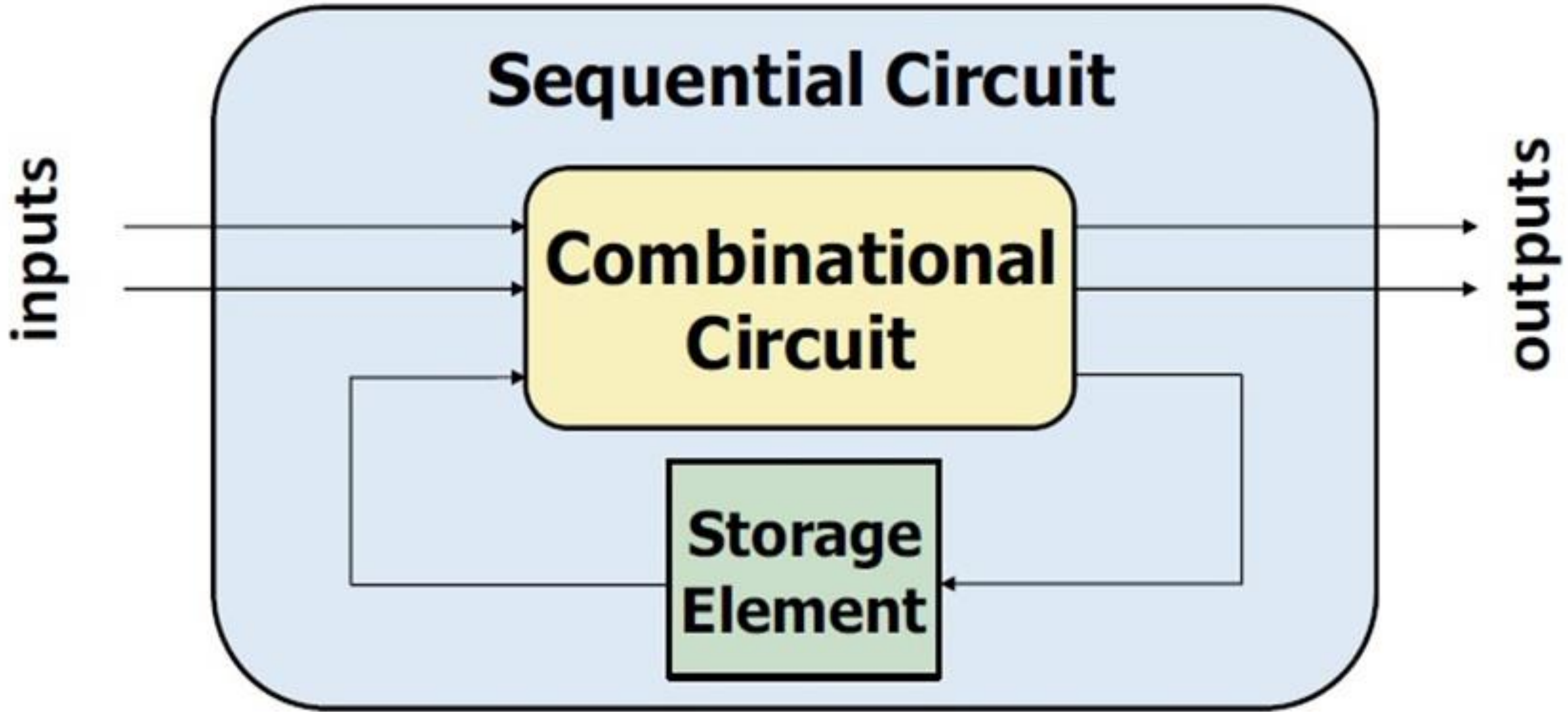
@pes.edu

Points addressed in this lecture

- Introduction to sequential circuits
- Types of sequential circuits
- R-S latch
- Problems in R-S latch
- R- S Flip Flop: Characteristic Table
- Conclusion Remarks

Introduction: Sequential Circuits

- Combinational Circuit output depend **only** on **present** input.
- We want circuits that produce the output depending on the **current** and **past** input values - Circuits with **memory**.
- How do we design such a circuit that **stores information**?



Types of sequential circuits

- Sequential Circuits are of **two** types

1. **Synchronous Circuits:**

- ❑ In synchronous sequential circuits, the state of the device changes only at discrete times in response to a **clock pulse**.

EX: FLIP-FLOPs

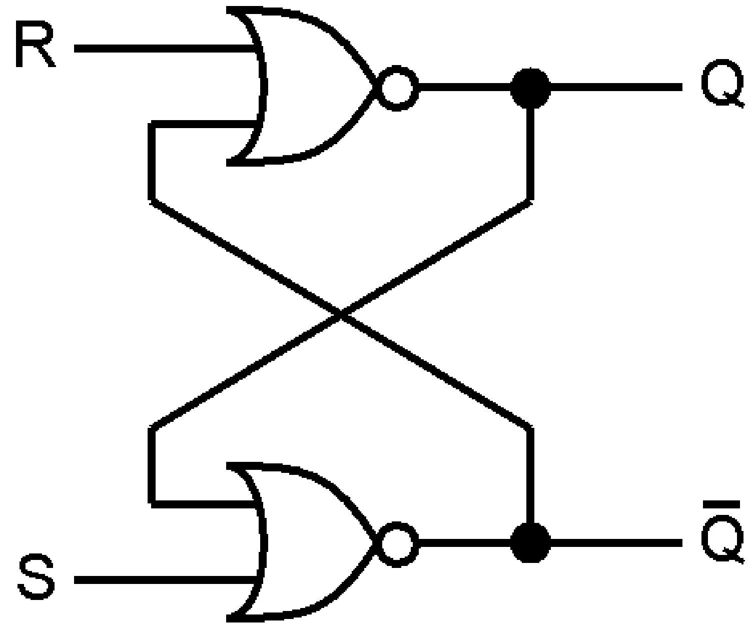
2. **Asynchronous Circuits:**

- ❑ Asynchronous circuit is **not synchronized** by a clock signal; the outputs of the circuit change directly in response to the change in the inputs.

EX: Latches

Types of sequential circuits – RS latch by Using NOR Gates

Circuit Diagram

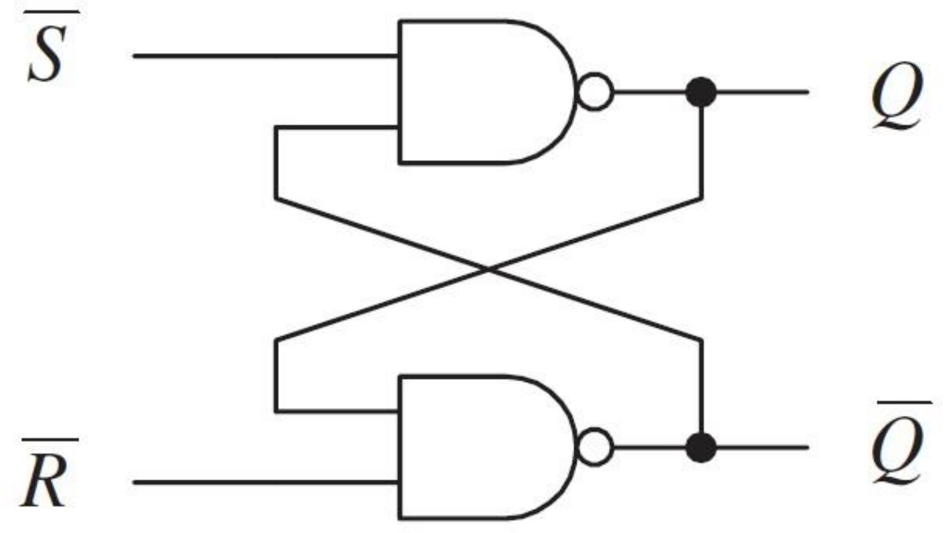


Function Table

INPUTS		OUTPUTS		Status of RS Latch
S	R	Q	Q'	
0	0	Q_{-1}	Q'_{-1}	PREVIOUS STATE OR NO CHANGE
0	1	1	0	RESET
1	0	0	1	SET
1	1	0	0	FORBIDDEN

Types of sequential circuits – S’R’ latch by Using NAND Gates

Circuit Diagram

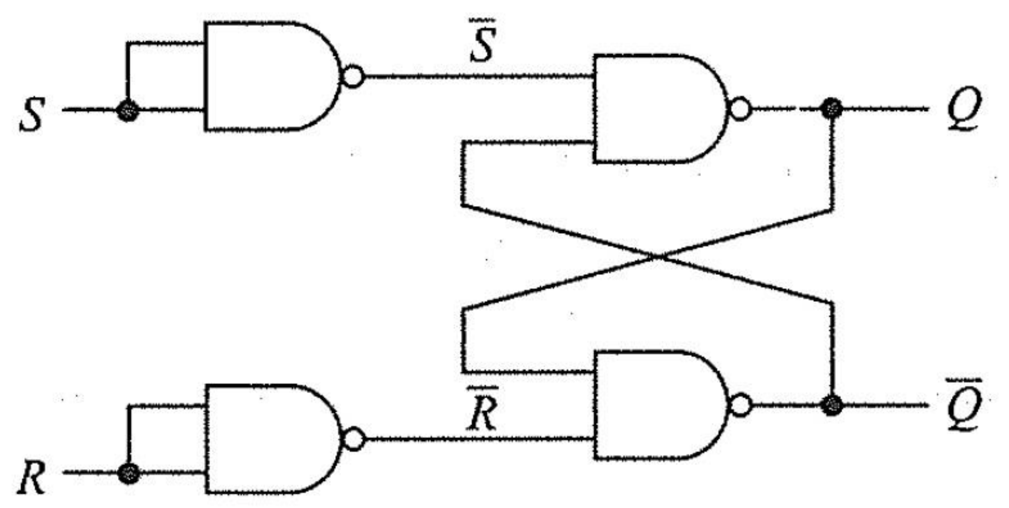


Function Table

INPUTS		OUTPUTS		Status of S’R’ Latch
S’	R’	Q	Q’	
0	0	1	1	FORBIDDEN
0	1	1	0	SET
1	0	0	1	RESET
1	1	Q_{-1}	Q'_{-1}	PREVIOUS STATE

Types of sequential circuits – SR latch by Using NAND Gates

Circuit Diagram

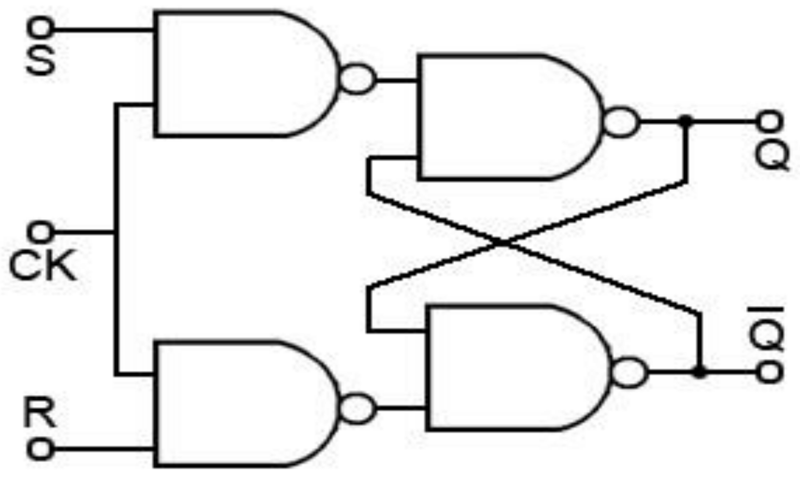


Function Table

INPUTS		OUTPUTS		Status of SR Latch
S	R	Q	Q’	
0	0	Q_{-1}	Q'_{-1}	PREVIOUS STATE
0	1	0	1	RESET
1	0	1	0	SET
1	1	1	1	FORBIDDEN

The S-R (Set - Reset) Flip – Flop / Gated R-S latch

Logic Diagram



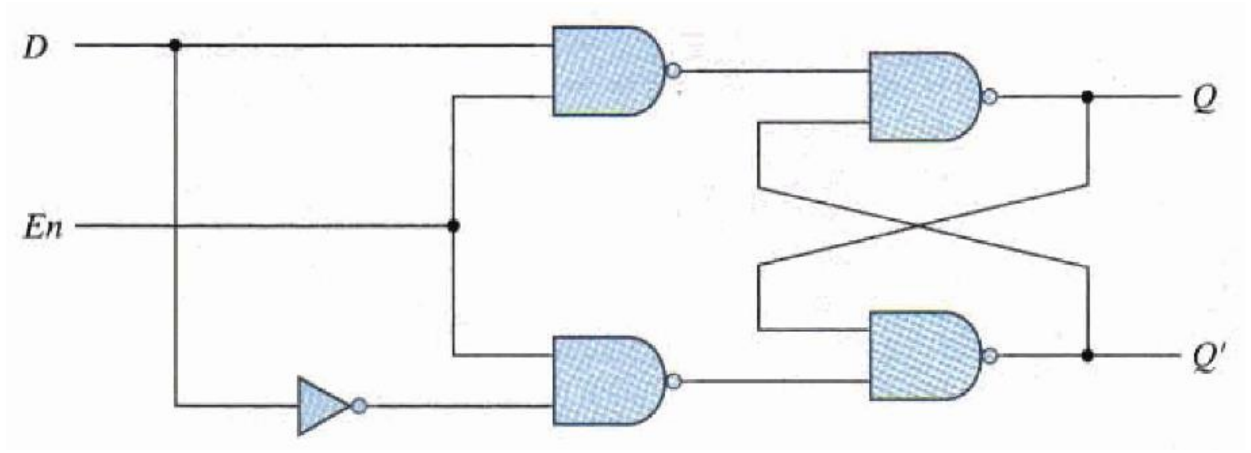
Function Table

INPUTS			OUTPUTS		Status of SR Flip-Flop
CK	S	R	Q	Q’	
0	X	X	Q_{-1}	Q'_{-1}	PREVIOUS STATE
1	0	0	Q_{-1}	Q'_{-1}	PREVIOUS STATE
1	0	1	0	1	RESET
1	1	0	1	0	SET
1	1	1	1	1	FORBIDDEN

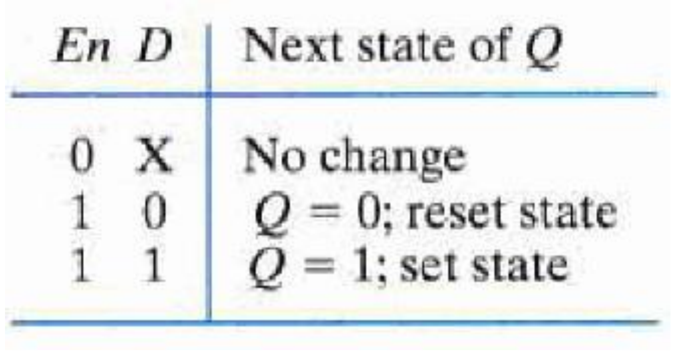
Electronic Principles and Devices

D Flip-Flop

Logic Diagram



Function Table



En	*D*	Next state of *Q*
0	X	No change
1	0	$Q = 0$; reset state
1	1	$Q = 1$; set state

Conclusion Remarks:

1. Flip – flops are the **building blocks** of sequential circuits.
   But these are built from **latches**.

2. Flip- flops **continuously checks** its inputs and changes its outputs correspondingly only at time instant determined by the **clock signal**.

3. Flip – flops is **sensitive** to signal change. They transfer data only at single time instant and data can't be changed till next signal change.
   - Therefore they are used as **registers**.

4. It is an edge triggered circuit, means that the output and the next state input changes, when there's a change in the clock pulse whether it can may be **POSITIVE (+ve)** or **NEGATIVE (-ve)** clock pulse.

THANK YOU

Prof.

Department of Electronics and Communication Engineering

@pes.edu

Points addressed in this lecture

Recap

- D Flip-Flop: Working, Function Table
- JK Flip-Flop: Working, Function Table
- T Flip-Flop: Working, Function Table
- Conclusion Remarks

Recap:

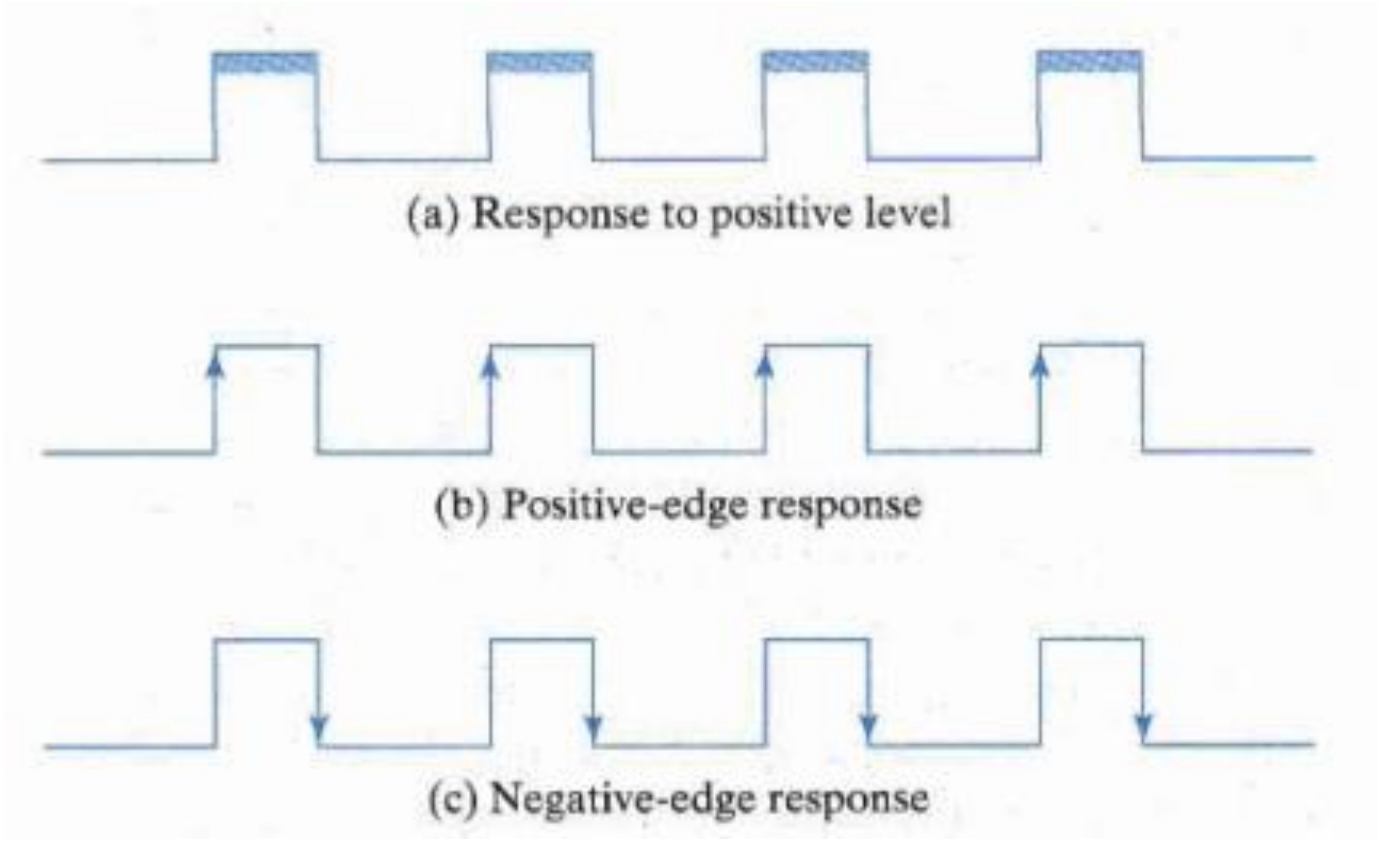
1. Flip – flops are the **building blocks** of sequential circuits.
   But these are built from **latches**.

2. Flip- flops **continuously checks** its inputs and changes its outputs correspondingly only at time instant determined by the **clock signal**.

3. Flip – flops is **sensitive** to signal change. They transfer data only at single time instance and data can't be changed till next signal change.
   - Therefore they are used as **registers**.

4. It is an edge triggered circuit, means that the output and the next state input changes, when there's a change in the clock pulse whether it can may be **POSITIVE (+ve)** or **NEGATIVE (-ve) clock pulse.**

Clock Response in Flip – Flop:

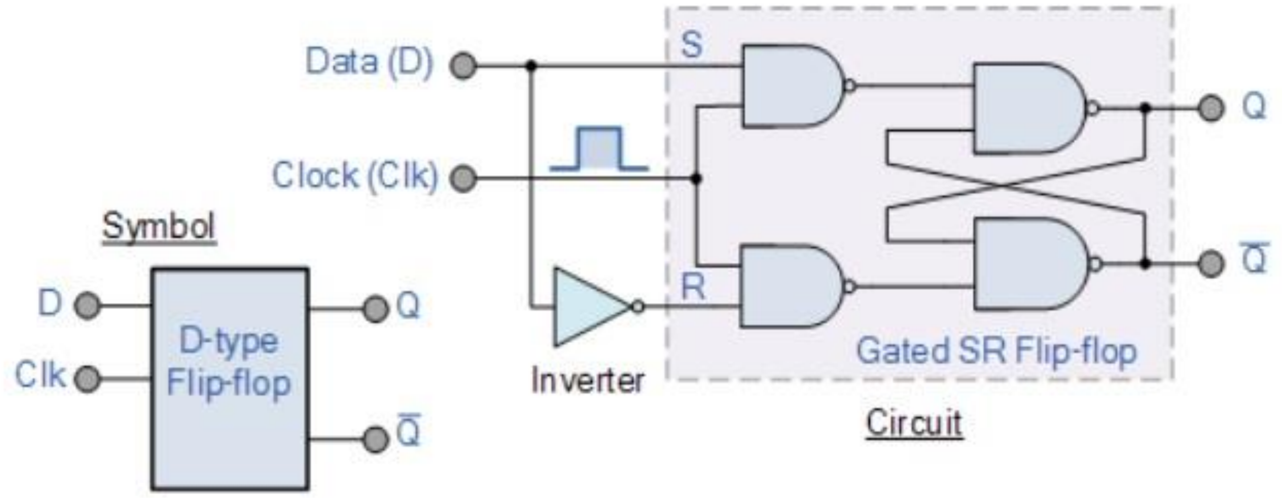

(a) Response to positive level
(b) Positive-edge response
(c) Negative-edge response

Flip-Flops will change the Output when the clock remains as logic 1 as shown in figure (a)

Flip-Flops will change the Output when the clock changes from logic 0 to Logic 1 as shown in figure (b)

Flip-Flops will change the Output when the clock changes from logic 1 to Logic 0 as shown in figure (c)

The Data (D) Flip – Flop:



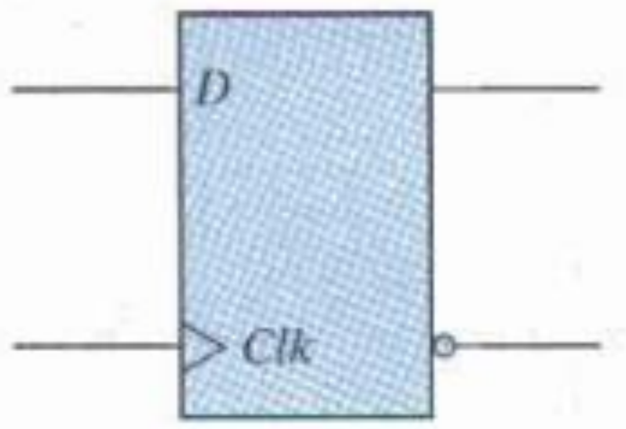
- ❑ The **D Flip Flop** is by far the most important of the clocked flip-flops as it ensures that ensures that inputs **S** and **R** are **never equal** to one at the same time.
- ❑ The **D Flip Flop** are constructed from a SR flip-flop with an **inverter** added between the **S** and the **R** inputs to allow for a single D (Data) input.

D Flip – Flop Function Table

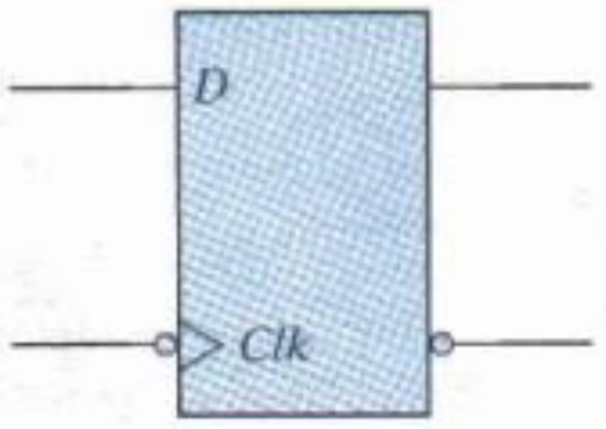
INPUTS		OUTPUTS		Status of D Flip-Flop
CLK	D	Q	Q’	
0	X	Q_{-1}	Q'_{-1}	PREVIOUS STATE
1	0	0	1	RESET
1	1	1	0	SET

- ❑ The D Flip-flop will store and output whatever logic level is applied to its **DATA** input.
- ❑ Once the clock input goes LOW the “set” and “reset” inputs of the flip-flop are both held at logic level “1”, so it will not change its state.
- ❑ ↓ and ↑ indicates direction of clock pulse as it is assumed D-type flip flops are edge triggered.

Positive & Negative Edge Triggered 'D' Flip – Flop:



(a) Positive-edge



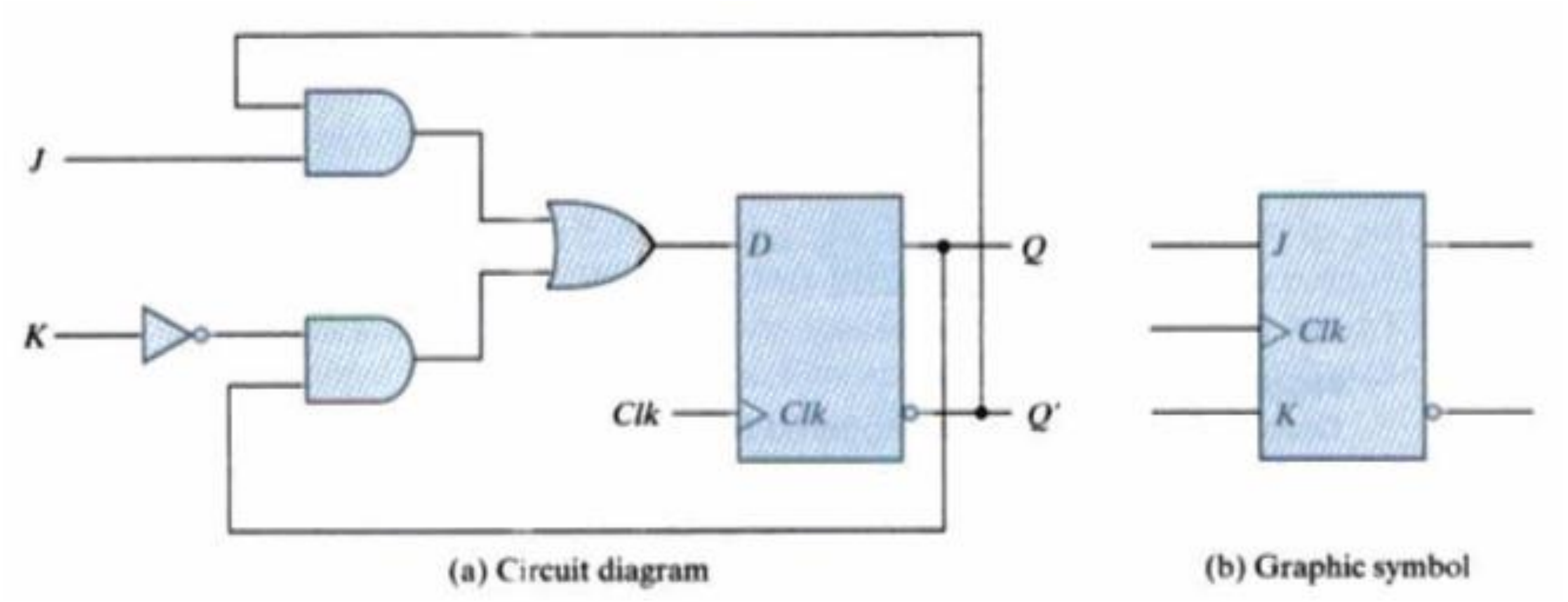
(a) Negative-edge

In D Flip-Flops, Absence of the bubble in front of the Clk indicates Positive edge triggered as shown in figure (a)

In D Flip-Flops, Presence of the bubble in front of the Clk indicates Positive edge triggered as shown in figure (b)

Electronic Principles and Devices

JK Flip – Flop:



(a) Circuit diagram (b) Graphic symbol

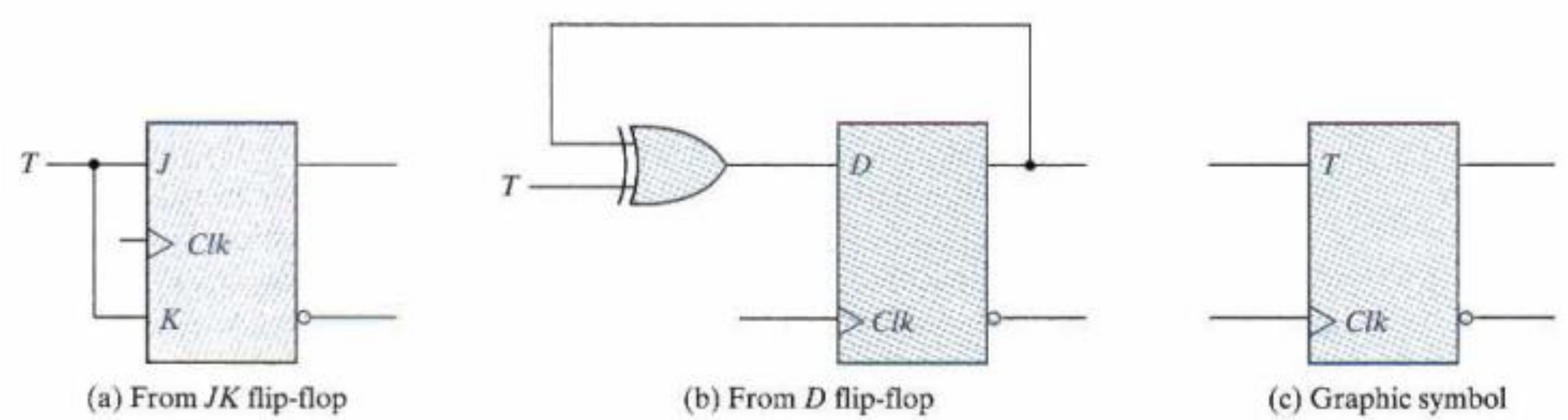
- The Inputs are **J-K** along with the Clk
  - J-K inputs comes after its inventors **Jack Kilby**.

JK Flip – Flop:

Function Table

INPUTS			OUTPUTS		Status of JK Flip-Flop
CLK	J	K	Q	Q’	
0	X	X	Q_{-1}	Q'_{-1}	PREVIOUS STATE
1	0	0	Q_{-1}	Q'_{-1}	PREVIOUS STATE
1	0	1	0	1	RESET
1	1	0	1	0	SET
1	1	1	Q'_{-1}	Q_{-1}	TOGGLE

T Flip – Flop:



T Flip-Flop using JK Flip-Flop & D Flip-Flop

- T Flip-Flop is a **synchronous** device, where high to low or low to high transitions is passed through **clock signal** which **changes** the output state of Flip-Flop.

T Flip – Flop: - Function Table

INPUTS		OUTPUTS		Status of T Flip-Flop
CLK	T	Q	Q’	
0	X	Q_{-1}	Q'_{-1}	PREVIOUS STATE
1	0	Q_{-1}	Q'_{-1}	PREVIOUS STATE
1	1	Q'_{-1}	Q_{-1}	TOGGLE

Advantages of T flip-flop:

- These Flip-Flops has a **toggle input** and a **clock**.
  When a clock is triggered it **inverts** the value of Flip-Flops.

- They are used for designing the **counters**.

Conclusion Remarks :

- **JK flip-flop** is a sequential circuit, whose state transitions are **synchronized** with the **clock pulse** (+ve / -ve) edge triggered.

- In **JK flip-flop** all the rows are a valid state in the characteristic table, also when J=K=1, the output toggles, **Toggle state**.

- **D flip-flop** is a sequential circuit, whose output is same as the input and hence also called as **transparent flip-flop**.

- **T flip-flop** is a synchronous sequential circuit, whose state transitions are **synchronized** with the **clock pulse**.

- **T flip-flop** finds its application in counters, **Toggle state** is used.

THANK YOU

Prof.

Department of Electronics and Communication Engineering

@pes.edu

Registers:

- A *register* is a group of flip-flops, each one of which shares a common clock and is capable of storing one bit of information.
- An *n* -bit register consists of a group of *n* flip-flops capable of storing *n* bits of binary information.
- In addition to the flip-flops, a register may have combinational gates that perform certain data-processing tasks.
- In its broadest definition, a register consists of a group of flip-flops together with gates that affect their operation.
- The flip-flops hold the binary information, and the gates determine how the information is transferred into the register.

4-bit Register:

- Various types of registers are available commercially.
- The simplest register is one that consists of only flip-flops, without any gates.
- Following Figure (next slide) shows such a register constructed with four *D* -type flip-flops to form a four-bit data storage register.
- The common clock input triggers all flip-flops on the positive edge of each pulse, and the binary data available at the four inputs are transferred into the register.
- The four outputs can be sampled at any time to obtain the binary information stored in the register.
- The input Clear goes to the active-low R (reset) input of all four flip-flops. When this input goes to 0, all flip-flops are reset asynchronously.
- The R inputs must be maintained at logic 1 (i.e., de-asserted) during normal clocked operation.

4-bit Register:

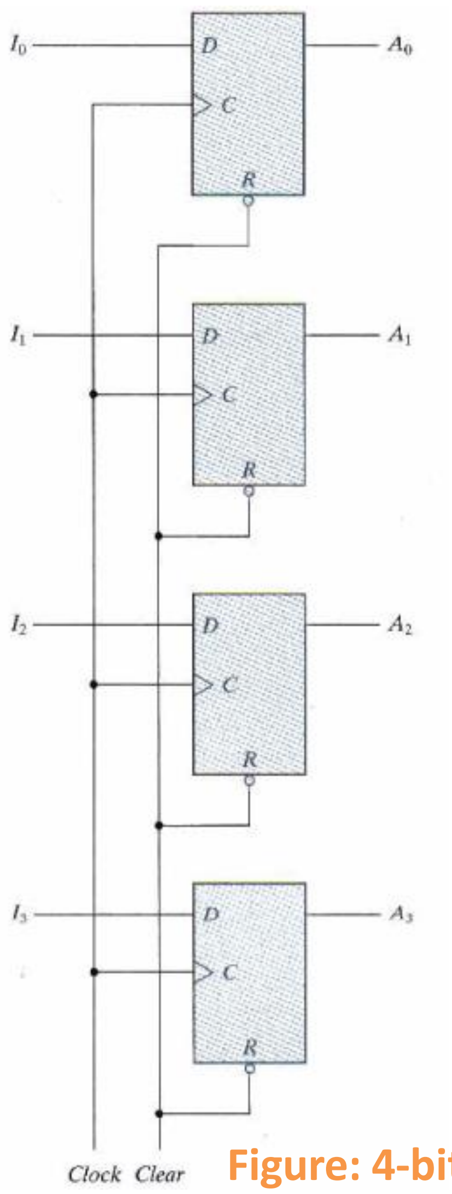


Figure: 4-bit Register

- The transfer of new information into a register is referred to as *loading or updating* the register.
- If all the bits of the register are loaded simultaneously with a common clock pulse, we say that the loading is done *in parallel* .
- A clock edge applied to the *C* inputs of the register of Figure, will load all four inputs in parallel.
- In this configuration, if the contents of the register must be left unchanged, the inputs must be held constant or the clock must be inhibited from the circuit.

SHIFT REGISTERS (SISO)

- A register capable of shifting the binary information held in each cell to its neighboring cell, in a selected direction, is called a *shift register.*
- The logical configuration of a shift register consists of a chain of flip-flops in cascade, with the output of one flip-flop connected to the input of the next flip-flop.
- All flip-flops receive common clock pulses, which activate the shift of data from one stage to the next.
- There are four different types of shift registers:
  - Serial In Serial Out (SISO)
  - Serial In Parallel Out (SIPO)
  - Parallel In Serial Out (PISO)
  - Parallel In Parallel Out (PIPO)

REGISTERS (SISO)

- The simplest possible shift register (SISO) is one that uses only flip-flops, as shown in the following Figure .
- The output of a given flip-flop is connected to the *D* input of the flip-flop at its right.
- This shift register is unidirectional (left-to-right).
- Each clock pulse shifts the contents of the register one bit position to the right.

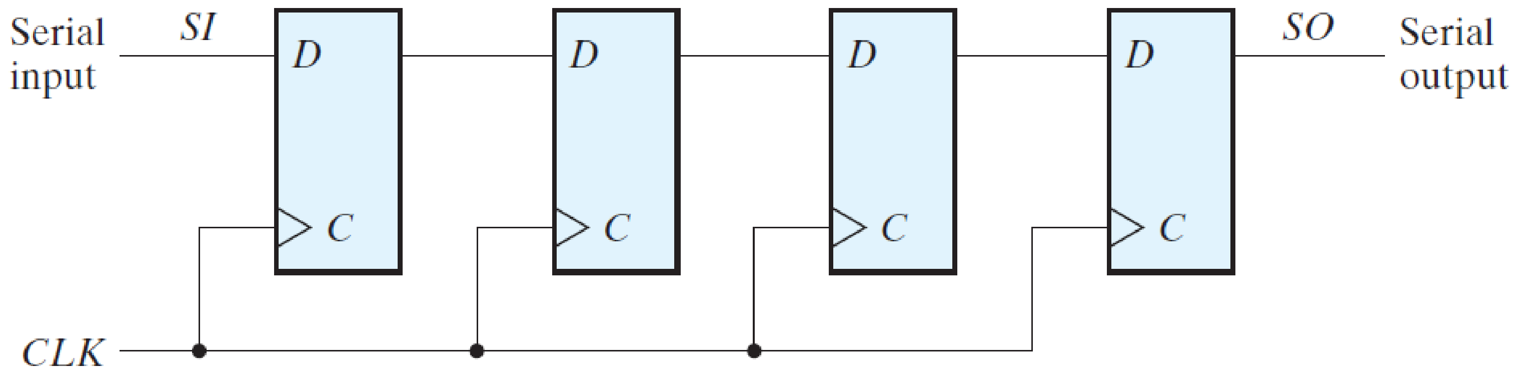
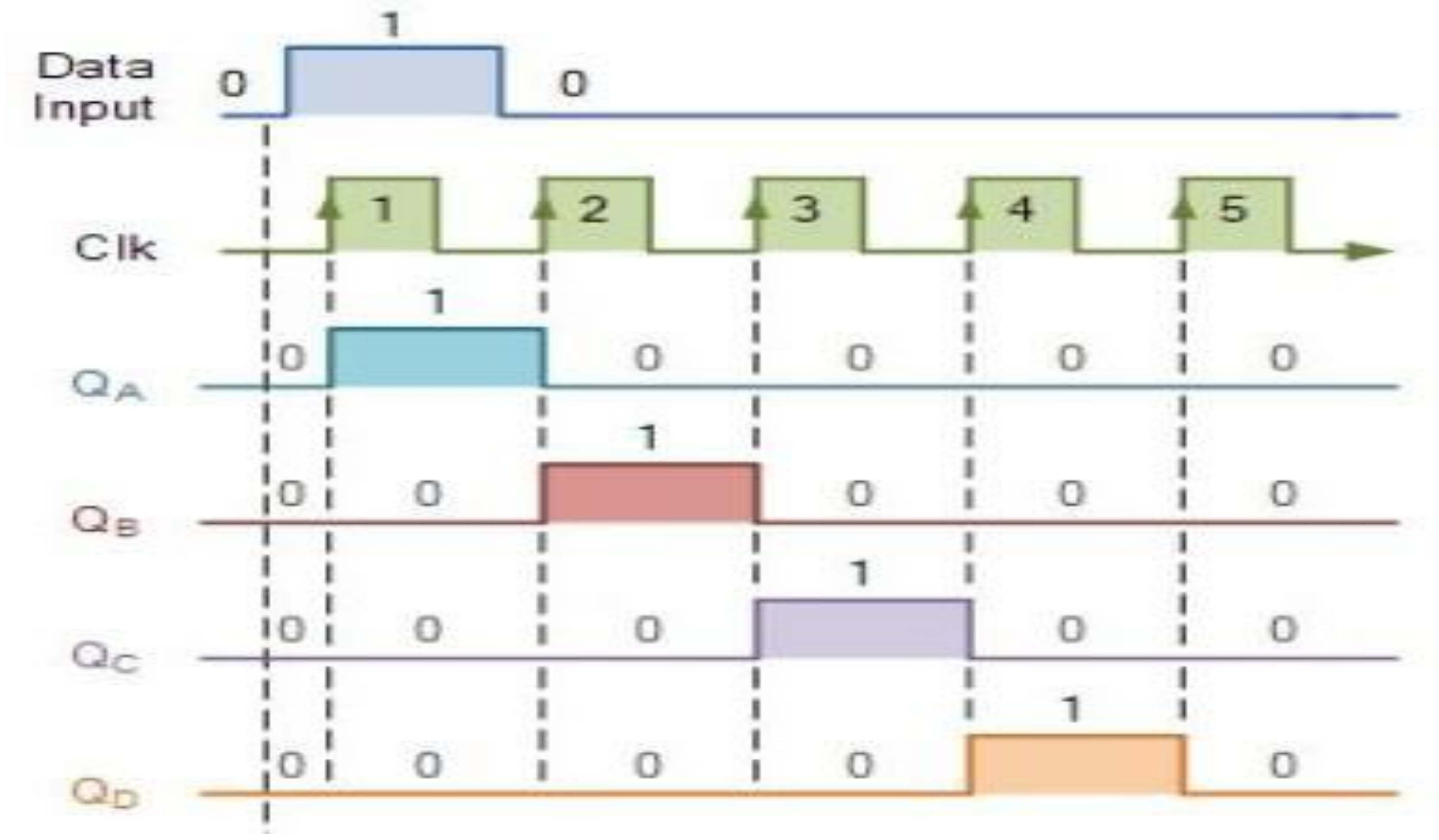


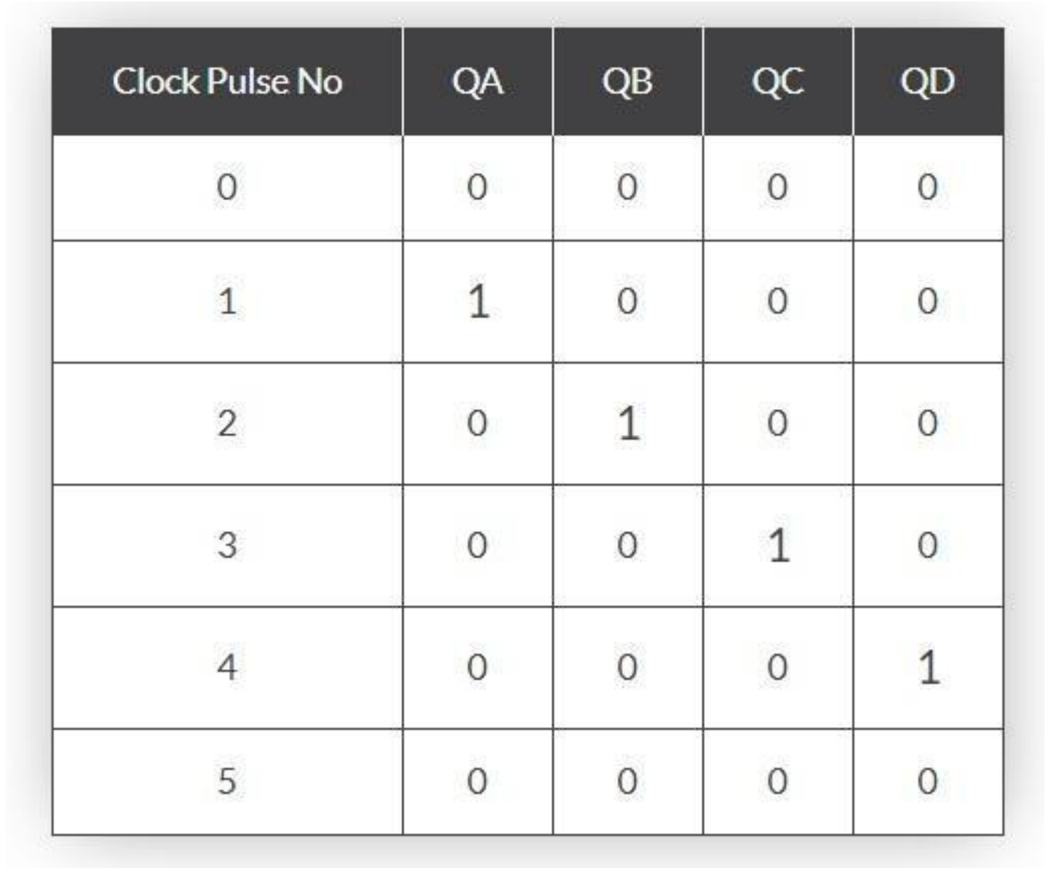
FIGURE: Four-bit shift register

REGISTERS (SISO)



Timing Diagram for SISO register

4-bit Serial Input Serial Output (SISO) register:



Clock Pulse No	QA	QB	QC	QD
0	0	0	0	0
1	1	0	0	0
2	0	1	0	0
3	0	0	1	0
4	0	0	0	1
5	0	0	0	0

Data movement through a shift register

4-bit Serial Input Serial Output (SISO) register:

Example:

The contents of a four-bit register is initially 0110. The register is shifted six times to the right with the serial input being 1011100. What is the content of the register after each shift?

Answer:

0110; 0011; 0001; 1000; 1100; 1110; 0111; 1011

Synchronous and Asynchronous counters (Introduction):

- A register (group of flip-flops) that goes through a prescribed sequence of states upon the application of input pulses is called a *counter* .
- The input pulses may be clock pulses, or they may originate from some external source and may occur at a fixed interval of time or at random.
- The sequence of states may follow the binary number sequence or any other sequence of states.
- A counter that follows the binary number sequence is called a *binary counter* .
- An n -bit binary counter consists of n flip-flops and can count in binary from 0 through $2^n - 1$.

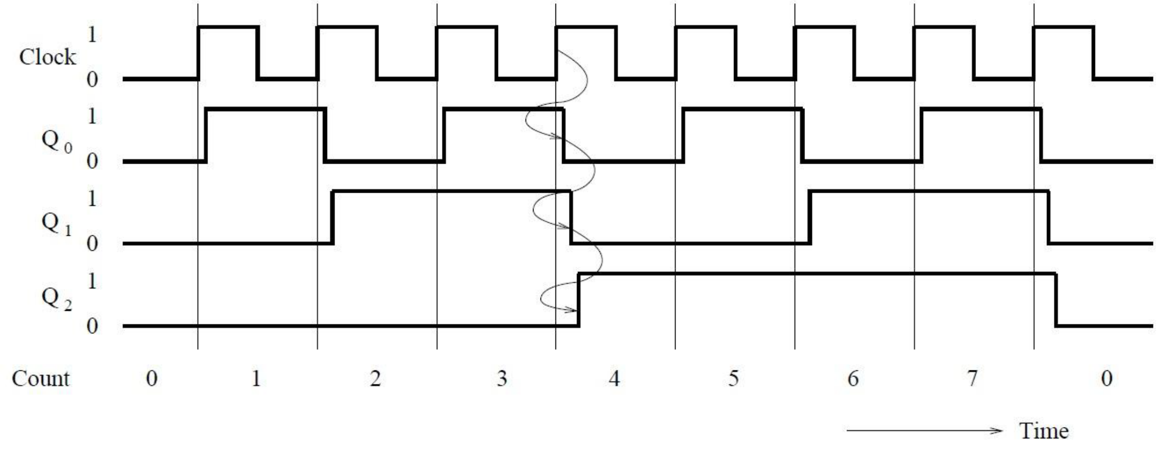
Synchronous and Asynchronous counters (Introduction):

- Counters are available in two categories: ripple (or asynchronous) counters and synchronous counters.
- In a ripple counter, a flip-flop output transition serves as a source for triggering other flip-flops.
- In other words, the *C* input of some or all flip-flops are triggered, not by the common clock pulses, but rather by the transition that occurs in other flip-flop outputs.
- In a synchronous counter, the *C* inputs of all flip-flops receive the common clock.

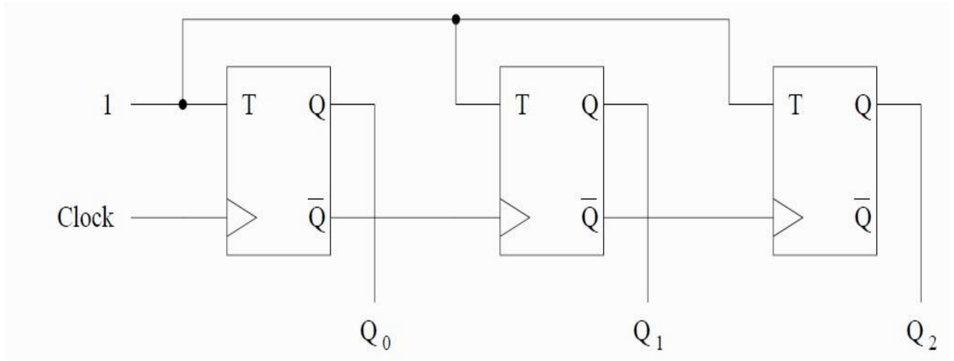
3 bit Asynchronous up- counter:

Truth Table

Number of clock pulses	Q_2 (MSB)	Q_1	Q_0 (LSB)
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1



Timing Diagram

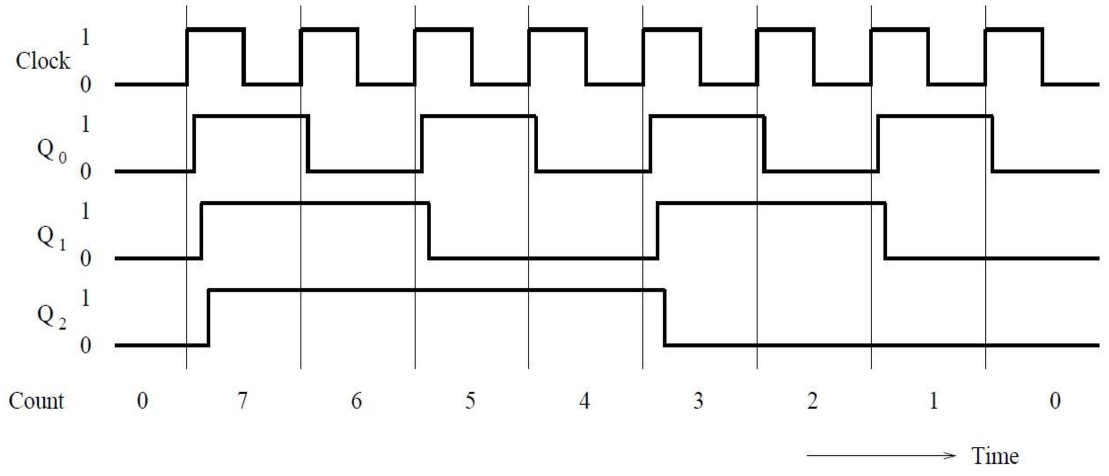


Circuit diagram

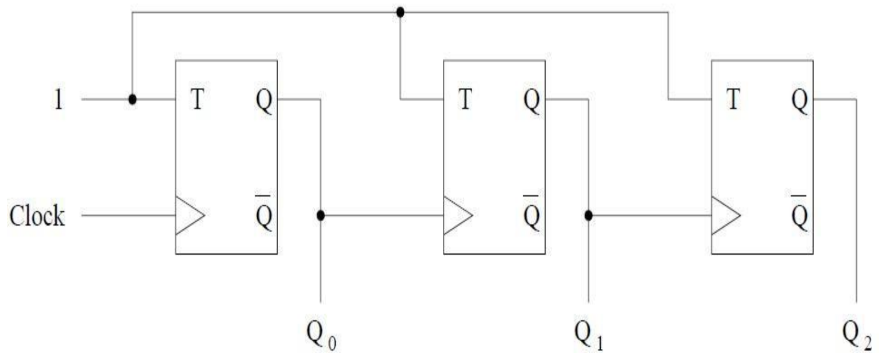
3 bit Asynchronous down - counter:

Truth Table

Number of clock pulses	Q_2 (MSB)	Q_1	Q_0 (LSB)
0	0	0	0
1	1	1	1
2	1	1	0
3	1	0	1
4	1	0	0
5	0	1	1
6	0	1	0
7	0	0	1



Timing Diagram



Circuit diagram

Electronic Principles and Devices

Concluding Remarks :

- **Registers, 4-bit registers**

- **4-bit SISO register** using D – flip flop, a synchronous sequential circuit which shifts the data value by one bit to the right.

- Counters

- **3 bit Asynchronous up- counter**, using T flip-flop.

- **3 bit Asynchronous down- counter**, using T flip-flop.

PES UNIVERSITY

CELEBRATING 50 YEARS

THANK YOU

Department of Electronics and Communication