DIGITAL LOGIC DESIGN

(Computer Science Engineering)

PRESENTED BY
VENKATA RAMANA BOLLA
DEPT. OF ECE

INTRODUCTION TO DIGITAL LOGIC DESIGN

- Digital Logic Design is used to develop hardware, such as “**circuit boards** and **Microchip Processors.**
- This hardware processes user input, system protocol and other data in computers, navigational systems, cellphones or other high technology systems.
- Digital Logic Designers build complex electronic components that uses both electrical and computational characteristics.
- Logic Design, basic organization of the circuitry of digital computer, computers perform the calculations using components called logic gates, which are made up of IC’s(Integrated Circuit)
- IC’s receives an input signal process it, and change it into an output signal.

Examples of Digital systems

- A smart phone is a digital system that has software (apps, operating system),input components (touch screen, keyboard, camera, and microphone),output components (screen and speakers), memory components(silicon chip and solid state drives)

Continued……..

ADVANTAGES OF DIGITAL TECHNIQUES

- Digital systems are generally easier to design
- Information storage is easy
- Accuracy and precision are easier to maintain throughout the system
- Operation can be programmed
- Digital circuits are less affected by noise
- More digital circuitry can be fabricated on IC chips

LIMITATIONS OF DIGITAL TECHNIQUES

- The real world is analog
- Processing digitized signals takes time

NUMBER SYSTEM REPRESENTATION

DIGITAL NUMBER SYSTEMS

- Many number systems are used in digital technology
- The most commonly used number systems are **“Decimal, Binary, Octal and Hexadecimal** systems
- A number is represented by $(N)_b$ as ,here ‘N’ is the number and ‘b’ is the base

Number system

Weighted number system

- It is positional weightage system
- Binary, Octal, Decimal and Hexadecimal
- 8421,5421 and so on

Unweighted number systems

It is a non positional weightage system

Gray code, Excess-3 codes are the examples

CHARACTERISTICS OF COMMONLY USED NUMBER SYSTEMS

Number system	Base (or) radix	Symbols used	Example
Binary	2	0,1	1011.011
Octal	8	0,1,2,3,4,5,6,7	3567.456
Decimal	10	0,1,2,3,4,5,6,7,8,9	3974.789
Hexadecimal	16	0,1,2,3,4,5,6,7,8,9,A,B, C,D,E,F	3FA9.AF6

NUMBER SYSTEMS CONVERSION TABLE

Number system conversion table from 1 to 15			
Decimal Number Base-10	Binary Number Base-2	Octal Number Base-8	Hexadecimal Number Base-16
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

DECIMAL NUMBER SYSTEM

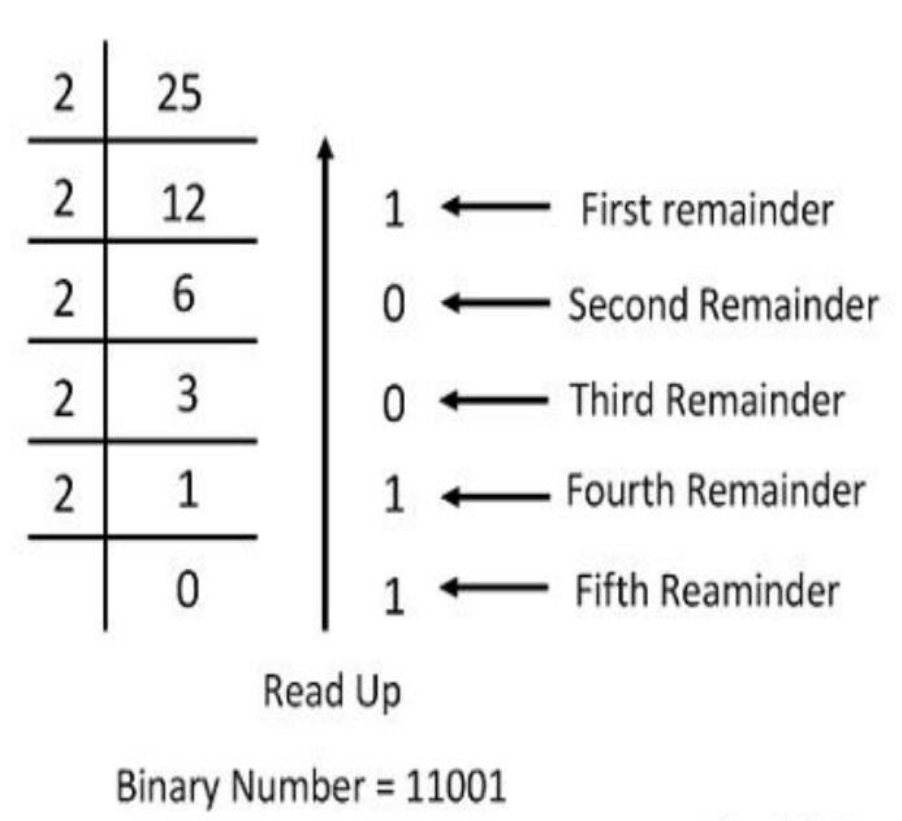
- This system has "**Base 10**"
- It has 10 symbols (0,1,2,3,4,5,6,7,8,9)
- This is a positional value system in which the value of a digit depends on its position
- Let we have $(453)_{10}$ is a decimal number
- The above representation "3" is the Least Significant Bit (LSB), "4" is the Most Significant Bit (MSB)
- Example: $(453)_{10}$
- $4 \times 10^2 + 5 \times 10^1 + 3 \times 10^0 = 453$
- Possible conversions: Decimal to Binary conversion
- Decimal to Octal conversion
- Decimal to Hexadecimal conversion

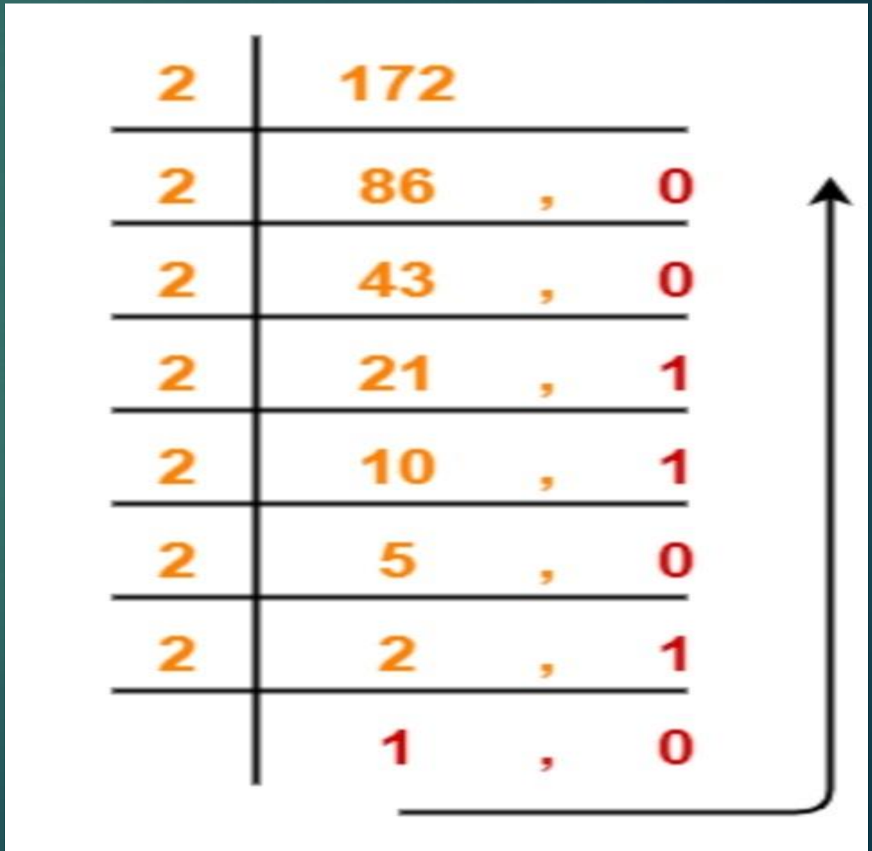
DECIMAL TO BINARY CONVERSION

- **Let us considered an example:**

i). $(25)_{10} = (11001)_2$

ii). $(172)_{10} = (10101100)_2$





FRACTIONAL DECIMAL TO BINARY CONVERSION

- **Let us considered an example:(i) $(0.625)_{10} = (\ ?\ )_2$**

	Real part	Fractional Part
0.625 x 2	1	0.25
0.25 x 2	0	0.50
0.50 x 2	1	0

$$(0.625)_{10} = (\ 0.101\ )_2$$

DECIMAL TO OCTAL CONVERSION

- **Let us considered an example:**

(i) $(425)_{10} = (\ 651\ )_8$ (ii) $(247)_{10} = (367\ )_8$

8	425		
8	53	1	↑
8	6	5	↑
	0	6	↑

$\therefore (425)_{10} = (651)_8$

8	247	
8	30	remainder 7
8	3	remainder 6
	0	remainder 3

Thus $(247)_{10} = (367)_8$

$(6260)_{10} = (\quad)_8$

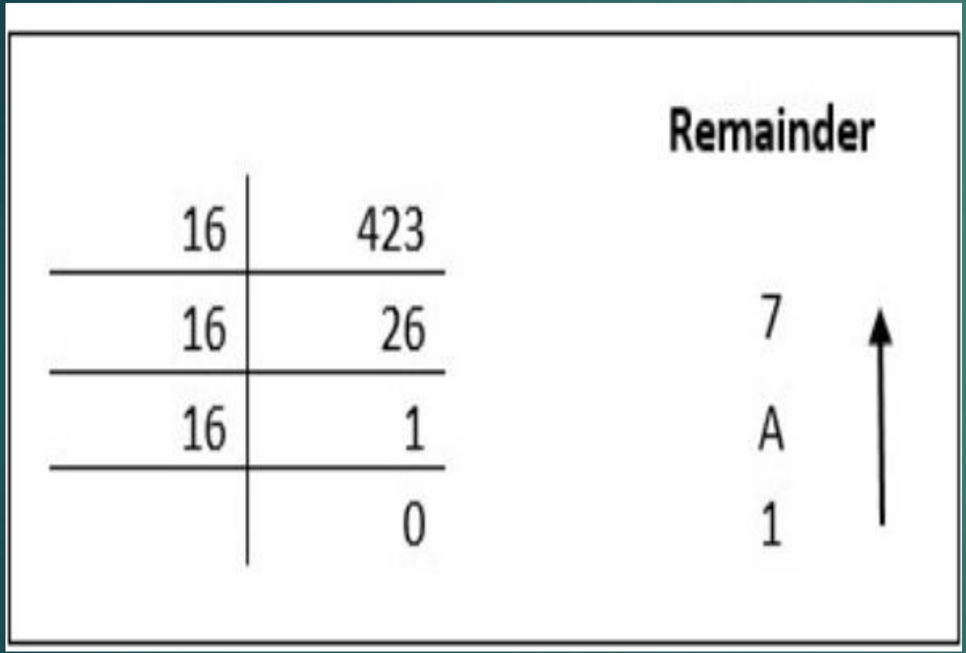
8	6260		
8	782	4	↑
8	97	6	↑
8	12	1	↑
8	1	4	↑
	0	1	↑

$\therefore (6260)_{10} = (14164)_8$

DECIMAL TO HEXADECIMAL CONVERSION

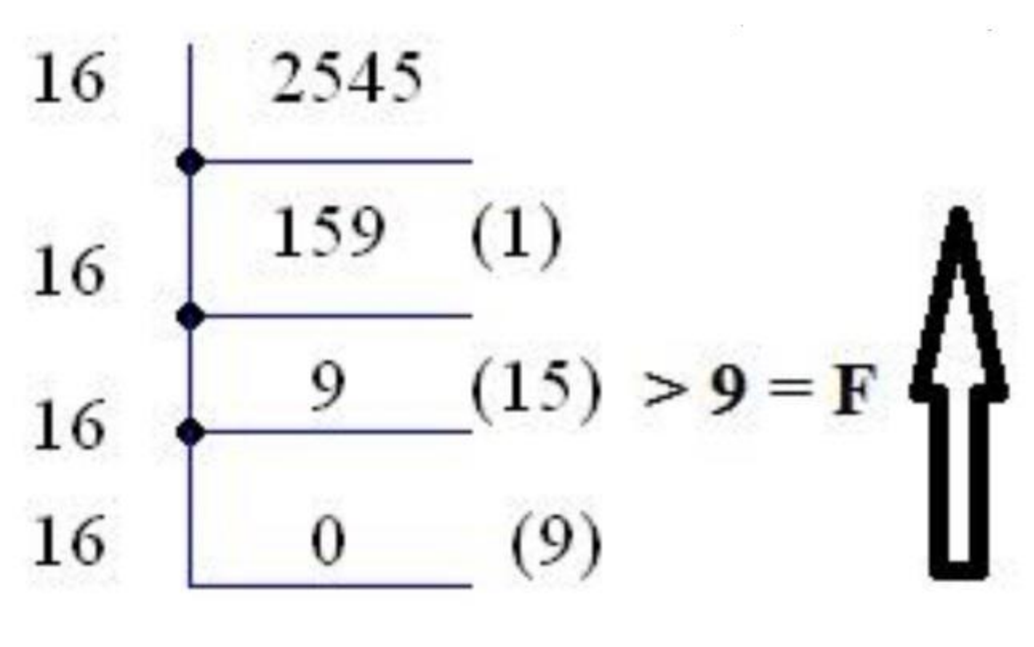
- **Let us considered an example:**

i). $(423)_{10} = (\ ?\ )_{16}$



$(423)_{10} = (\ 1A7\ )_{16}$

ii). $(2545)_{10} = (\ ?\ )_{16}$



$(2545)_{10} = (\ 9F1\ )_{16}$

BINARY NUMBER SYSTEM

- This system has "**Base 2**"
- It has 2 symbols (0,1)
- It has two base numbers: 0 and 1
- In this number system a group of four bits is known as "Nibble"
  group of eight bits is known as "Byte"
  group of sixteen bits is known as "Word"

4 BITS= 1 NIBBLE
8 BITS= 1 BYTE
16 BITS= 1 WORD

- Possible conversions: Binary to Decimal conversion
- Binary to Octal conversion
- Binary to Hexadecimal conversion

BINARY TO DECIMAL CONVERSION

- **Let us considered an example:(i) $(10010)_2 = (\ 18\ )_{10}$**

$(10010)_2$

$= (\ 1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0\ )_{10}$

$= (\ 16 + 0 + 0 + 2 + 0\ )_{10}$

$= (\ 18\ )_{10}$

$(10010.101)_2$

$= (\ 1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3}\ )_{10}$

$= (\ 16 + 0 + 0 + 2 + 0 + 0.5 + 0.125\ )_{10}$

$= (\ 18.625\ )_{10}$

(ii) $(101)_2 = (\ ?\ )_{10}$

$1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 = (5)_{10}$

$(101)_2 = (5)_{10}$

(iii) $(101.11)_2 = (?)_{10}$

$(1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0\ )\ .\ (1 \times 2^{-1} + 1 \times 2^{-2}\ )$

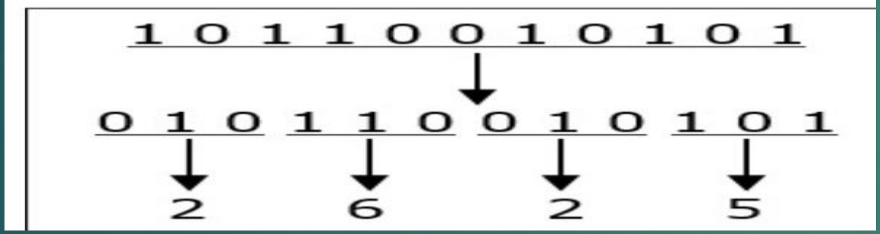
$(101.11)_2 = (5.75)_{10}$

Binary Numbering Scale

Base 2 Number	Base 10 Equivalent	Power	Positional Value
000	0	2^0	1
001	1	2^1	2
010	2	2^2	4
011	3	2^3	8
100	4	2^4	16
101	5	2^5	32
110	6	2^6	64
111	7	2^7	128

BINARY TO OCTAL CONVERSION

- To convert a binary to octal number these steps are followed:
  1. Starting from the least significant bit , make group of three(3) bits
  2. If there are one or two bits less in making the groups,0’s can be added after the MSB
  3. Convert each group into its equivalent octal number



Let us considered an example: (i) $(1010)_2 = \mathbf{(?)_8}$

0 0 1 **0 1 0**

1 **2** $\mathbf{=(12)_8}$

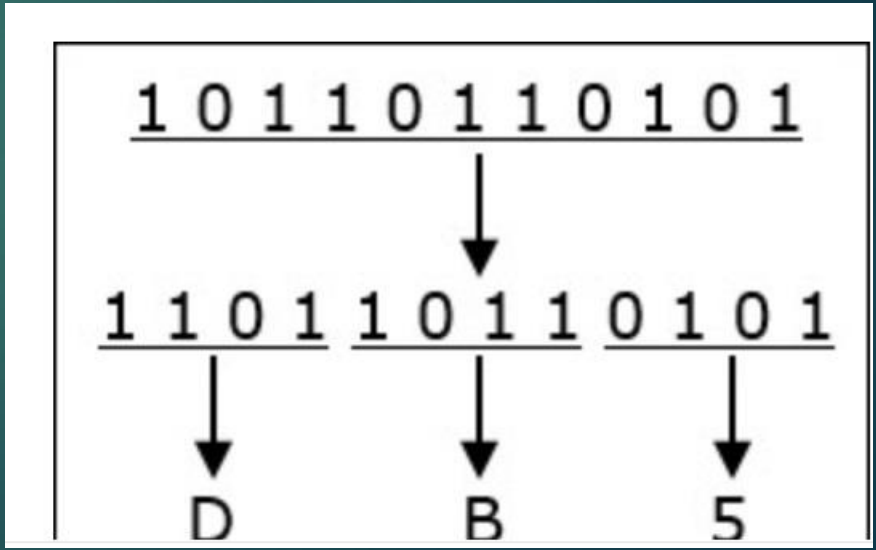
BINARY TO HEXADECIMAL CONVERSION

- For this conversion the binary bit stream is grouped into “Pairs of four bits” starting from LSB
- Hexa number is written for its equivalent binary group

Example: Convert $(10100110101111)_2$ to hexa decimal number system

0010	1001	1010	1111
2	**9**	**A**	**F**

$$(10100110101111)_2 = (29AF)_{16}$$



OCTAL NUMBER SYSTEM

- This system has "**Base 8**"
- It has 8 symbols (0,1,2,3,4,5,6,7)
- It is a method of grouping binary numbers in grouping of "three bits" (3 bits)
- Grouping can be start from Least Significant Bit (LSB) if it is Integer value
- Let we have $(453)_8$ is a octal number
- The above representation "3" is the Least Significant Bit (LSB), "4" is the Most Significant Bit (MSB)
- Example: $(453)_8$
- $4 \times 8^2 + 5 \times 8^1 + 3 \times 8^0$
- Possible conversions: Octal to Binary conversion
- Octal to Decimal conversion
- Octal to Hexadecimal conversion

OCTAL TO BINARY CONVERSION

- Let us considered an example: (i) $(453)_8 = (\ ?\ )_2$
- Solution: 4 5 3
- 100 101 011 $= (100101011)_2$

The above number can be converted to binary format based on "8 4 2 1" code

Total of 8 4 2 1= 8 + 4 + 2 + 1=15

(ii) $\mathbf{(567)_8 = (\ ?\ )_2}$

Solution: 5 6 7

101 110 111 = (101110111)

OCTAL TO DECIMAL CONVERSION

- Let us considered an example: (i) $(453)_8 = (\ ?\ )_{10}$
- Solution: 4 5 3
- 8^2 8^1 $8^0 = 4 \times 8^2 + 5 \times 8^1 + 3 \times 8^0 = 256 + 40 + 3 = (299)_{10}$

(ii) $\mathbf{(6327.4051)_8 = (\ ?\ )_{10}}$

Solution: 6 3 2 7 . 4 0 5 1

8^3 8^2 8^1 8^0 . 8^{-1} 8^{-2} 8^{-3} 8^{-4}

$= 6 \times 8^3 + 3 \times 8^2 + 2 \times 8^1 + 7 \times 8^0 . 4 \times 8^{-1} + 0 \times 8^{-2} + 5 \times 8^{-3} + 1 \times 8^{-4}$

$= 3072 + 192 + 16 + 7. 4/8 + 0 + 5/512 + 1/4096$

$= (3287.5100098)_{10}$

OCTAL TO HEXADECIMAL CONVERSION

- First convert octal to binary after that grouping resultant binary number into four bits gives hexadecimal number
- Let us considered an example: (i) $(453)_8 = (\ 12B\ )_{16}$
- Solution: 4 5 3
- 100 101 011 $= (100101011)_2$

$= 0001 \quad 0010 \quad 1011$

$= \quad 1 \quad 2 \quad B \quad = (\ 12B)_{16}$

(ii) $\mathbf{(567)_8 = (\ 177\ )_{16}}$

Solution: 5 6 7

101 110 111 $= (\ 101110111)_2$

$= \ 0001 \quad 0111 \quad 0111$

$= \quad 1 \quad 7 \quad 7 \quad =(177)_{16}$

HEXADECIMAL NUMBER SYSTEM

- This system has "**Base 16**"
- It has 16 symbols (0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F)
- It is a method of grouping of "four bits" (4 bits)
- Grouping can be start from Least Significant Bit (LSB) if it is Integer value
- Grouping can be start from right side of the radix point if it is fraction value(.1110 1111)
- This number system contains numeric digits (0,1,2--------9) and Alphabets (A,B,C,D,E,F)
- To signify a hexa number , a subscript of "16" or letter "H" is used
- Possible conversions: Hexadecimal to Binary conversion
- Hexadecimal to Octal conversion
- Hexadecimal to Decimal conversion

HEXADECIMAL TO BINARY CONVERSION

- For this conversion replacing each hexa decimal digit by 4 bit binary equivalent
- Let us considered an example: (i) $(2F9A)_{16} = (?)_2$
- Solution: 2 F 9 A
- 0010 1111 1001 1010 $=(0010111110011010)_2$

The above number can be converted to binary format based on "8 4 2 1" code

Total of 8 4 2 1= 8 + 4 + 2 + 1=15(1111)= $(F)_{16}$

(ii) $(5BA7)_{16} = (?)_2$

Solution: 5 B A 7

0101 1011 1010 0111 $= (0101101110100111)_2$

HEXADECIMAL TO OCTAL CONVERSION

- First convert hexadecimal to binary after that resultant binary number can be grouping in 3 bits
- Let us considered an example: (i) $(2F9A)_{16} = (\ ?\ )_8$
- Solution: 2 F 9 A
- 0010 1111 1001 1010 = $(0010111110011010)_2$

$(0010111110011010)_2$ = 000 010 111 110 011 010

= 0 2 7 6 3 2 = $(27632)_8$

(ii) $(5BA7)_{16} = (\ ?\ )_8$

Solution: 5 B A 7

0101 1011 1010 0111 = $(0101101110100111)_2$

$(0101101110100111)_2$ = 000 101 101 110 100 111

= 0 5 5 6 4 7 = $(55647)_8$

HEXADECIMAL TO DECIMAL CONVERSION

- Let us considered an example: (i) $(453)_{16} = (\ ?\ )_{10}$
- Solution : 4 5 3
- $16^2 \quad 16^1 \quad 16^0 = 4 \text{ x } 16^2 + 5 \text{ x } 16^1 + 3 \text{ x } 16^0 = 1024 + 80 + 3 = (1107)_{10}$

(ii) $(3A\ .\ 2F)_{16} = (\ ?\ )_{10}$

Solution: 3 A . 2 F

$16^1 \quad 16^0 \quad . \quad 16^{-1} \quad 16^{-2}$

$= 3 \text{ x } 16^1 + 10 \text{ x } 16^0\ .\ 2 \text{ x } 16^{-1} + 15 \text{ x } 16^{-2}$

$= 48 + 10.\ 2/16 + 15/16^2$

$= (58.51836)_{10}$

BINARY CODED DECIMAL CODE(BCD)

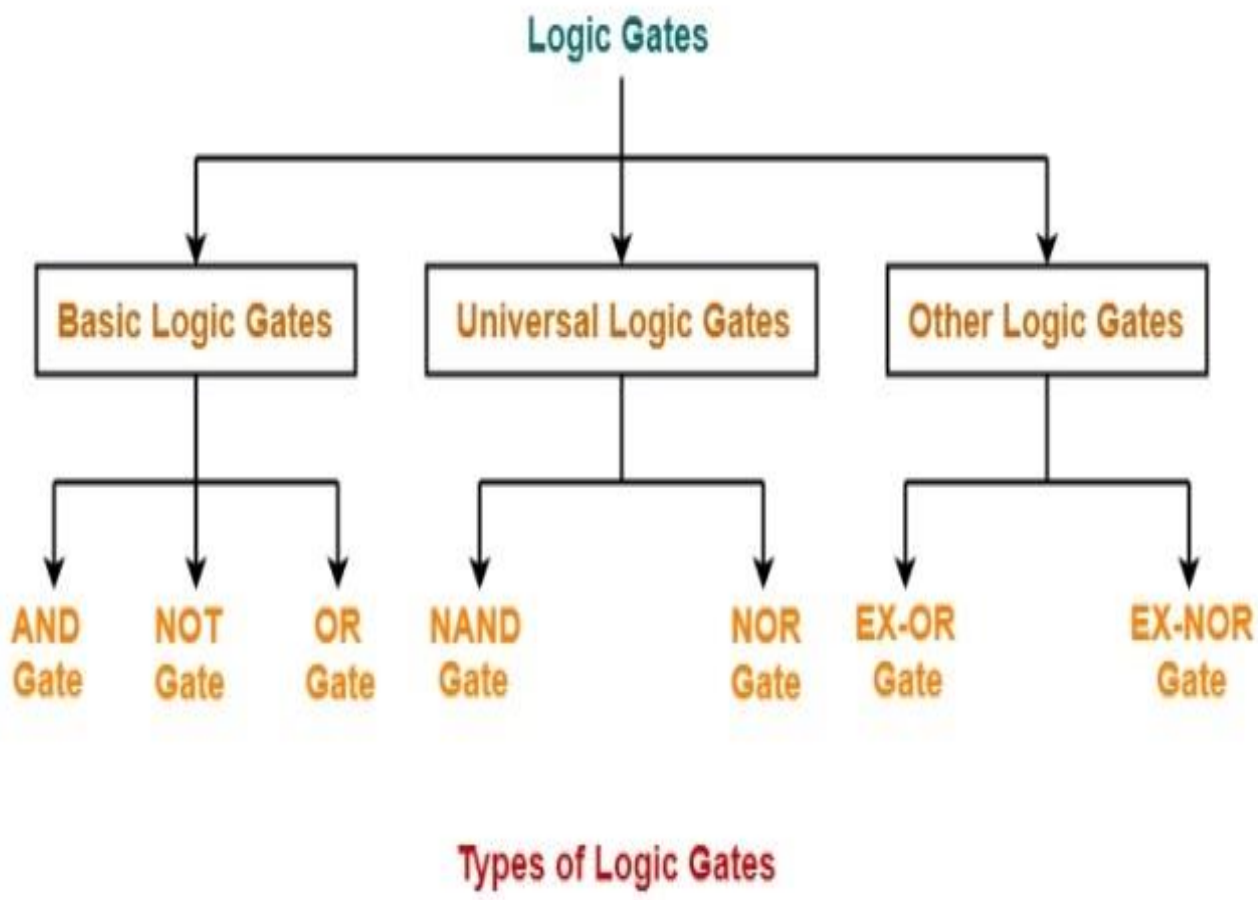
- In this code each digit of a decimal number is represented by binary equivalent
- It is a "4" bit binary code
- It is also known as "8 4 2 1" code or simply "BCD code"
- It is a weighted code system
- Example: $(943)_{decimal}$ =(…….) $_{BCD}$
- Solution: based on 8 4 2 1 code system - - 9 4 3
- 1001 0100 0011
- $(943)_{decimal}$ = $(100101000011)_{BCD}$
- 4 2 1= convert 2 to binary 4 2 1 4 2 1 4 2 1 4 2 1 4 2 1 4 2 1

0 1 0 1 0 1 1 1 1 1 1 0 0 1 1 1 0 0

2 5 7 6 3 4

LOGIC GATES

TYPES OF LOGIC GATES



- 7408 AND gate
- 7432 OR gate
- 7404 NOT gate or Inverter
- 7400 NAND gate
- 7402 NOR gate
- 7486 XOR gate

LOGIC GATES

- Logic gates are the most fundamental digital circuits that can be construct from diodes, transistors and resistors connected in such way that the circuit output is the result of a basic logic operation(OR,AND,NOT) performed on the inputs.
- It is simply a device that has two or more inputs and one output
- The function of each logic gate will be represented by Boolean expression
- The Boolean ‘0’ and ‘1’ represent the “logic level”

Logic ‘0’	**Logic ‘1’**
False OFF Low Open switch	True ON High Closed switch

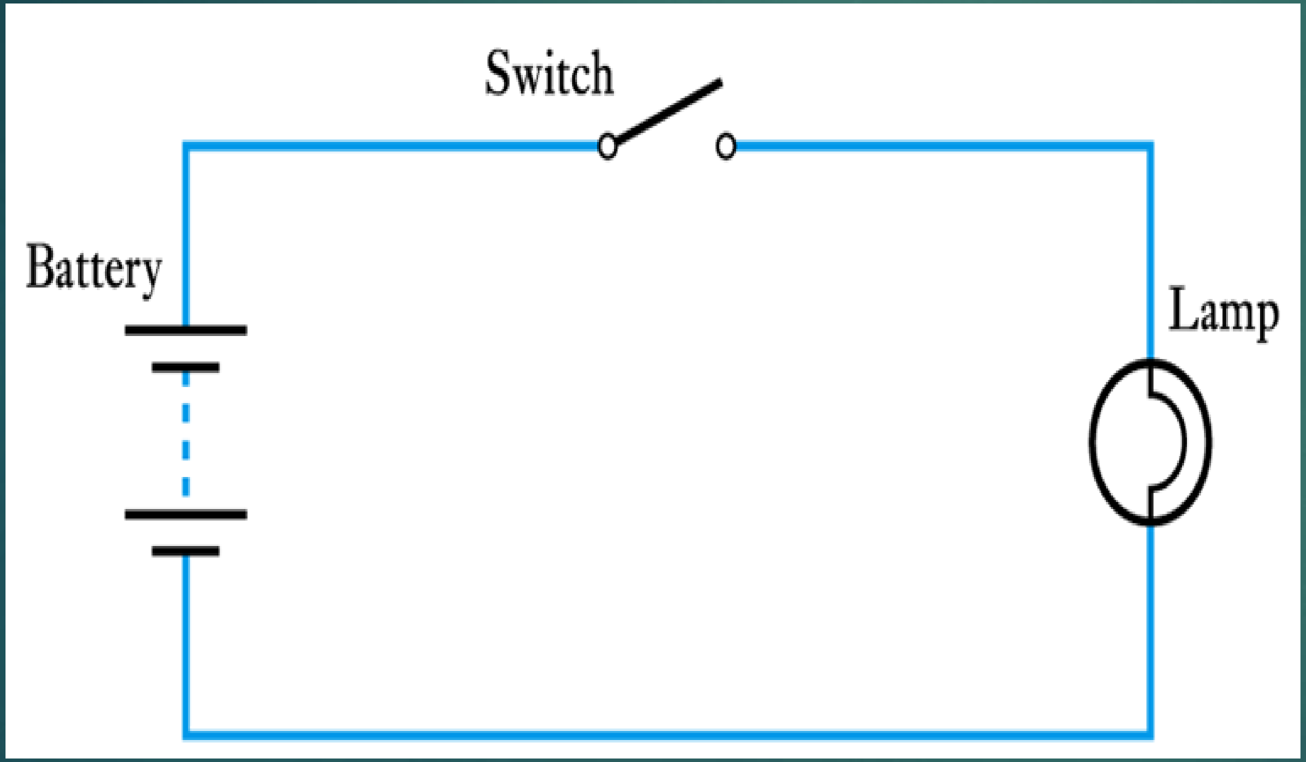
BASIC LOGIC GATES

- Basic logic gates are fundamental logic gates
- There are 3 basic logic gates
- (i) NOT gate
- (ii) AND gate
- (iii) OR gate
- Basic logic gates are Associative & Commutative in nature

NOT GATE :

- This gate can be performed on a single input variable and resulting single output variable
- The "NOT" operation is also referred to as "INVERSION" or "COMPLEMENTATION"
- Output logic is always opposite to the logic level of this input
- Output is always invert to the input. So it is also known as "Inverter"

BASIC LOGIC GATES

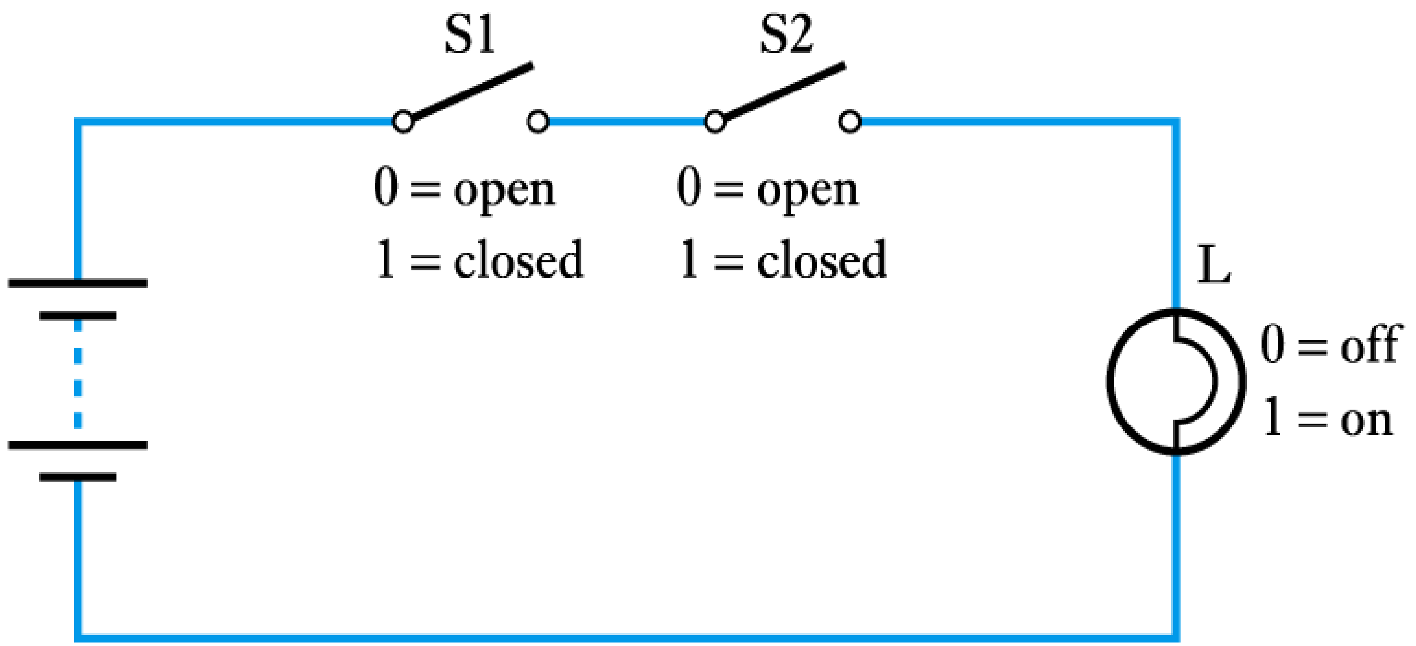


S	L
OPEN	OFF
CLOSED	ON

S	L
0	0
1	1

BASIC LOGIC GATES

- A binary arrangement with two switches in series



(a) Circuit

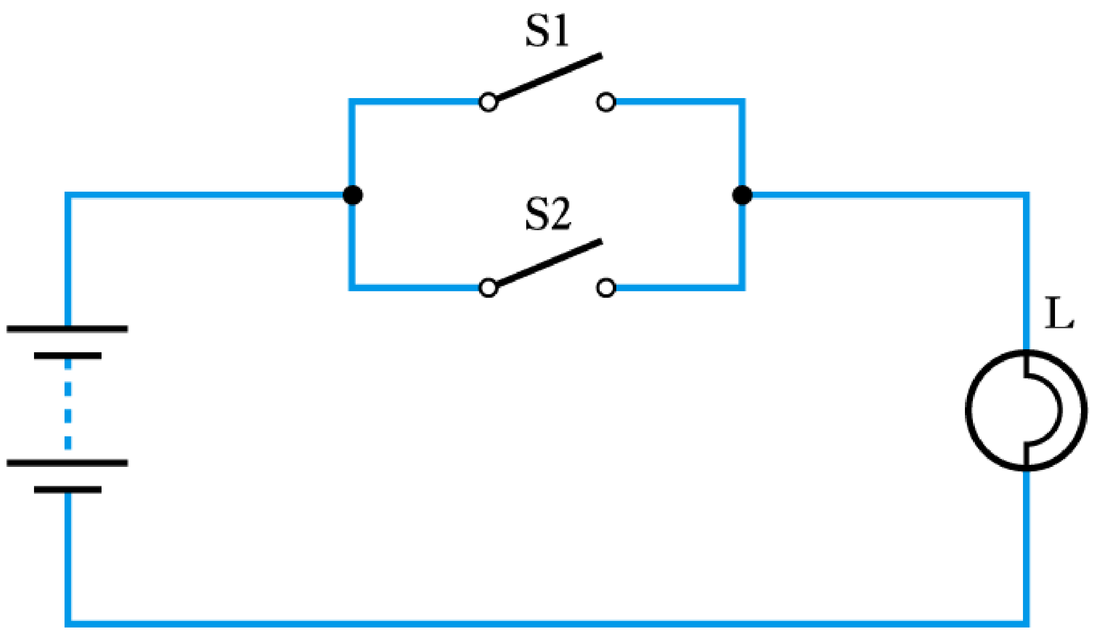
S1	S2	L
0	0	0
0	1	0
1	0	0
1	1	1

(b) Truth table

L = S1 AND S2

BASIC LOGIC GATES

- A binary arrangement with two switches in parallel



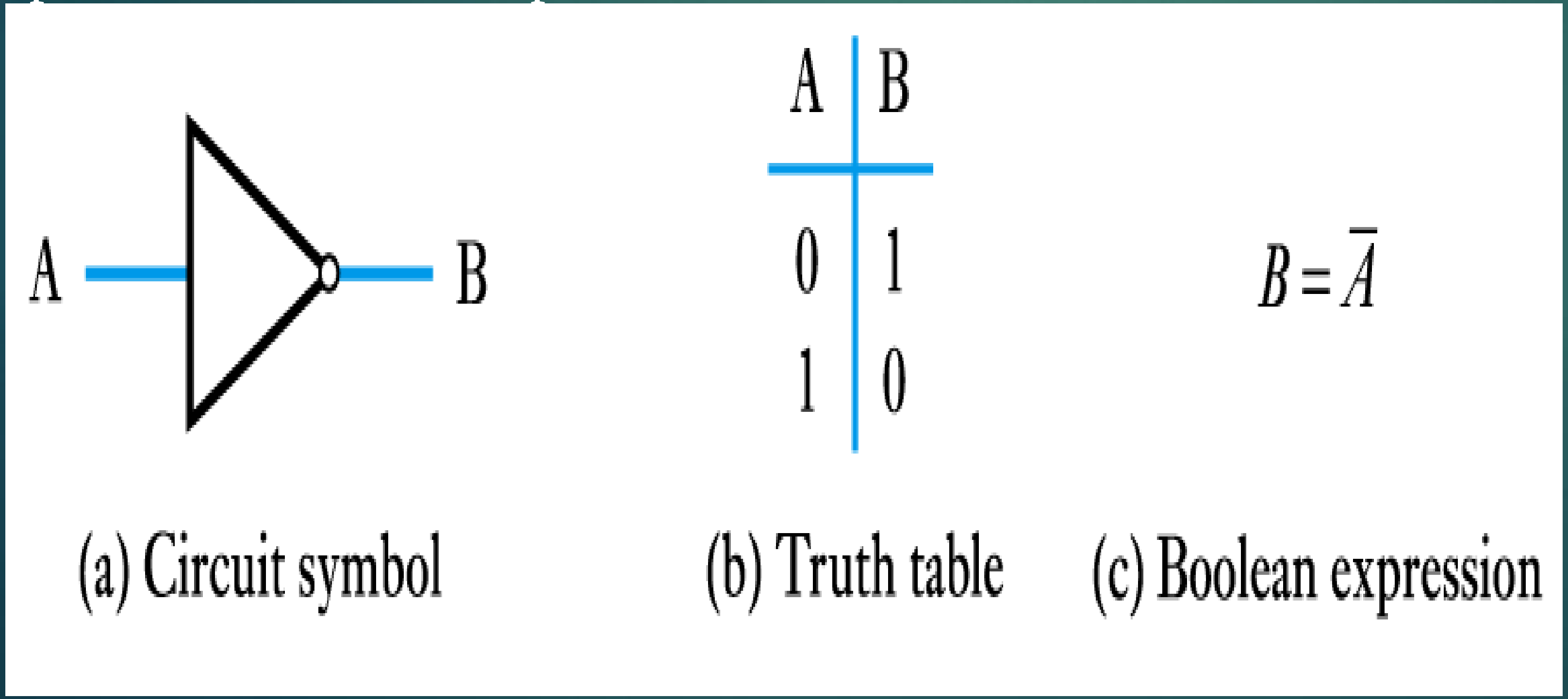
(a) Circuit

S1	S2	L
0	0	0
0	1	1
1	0	1
1	1	1

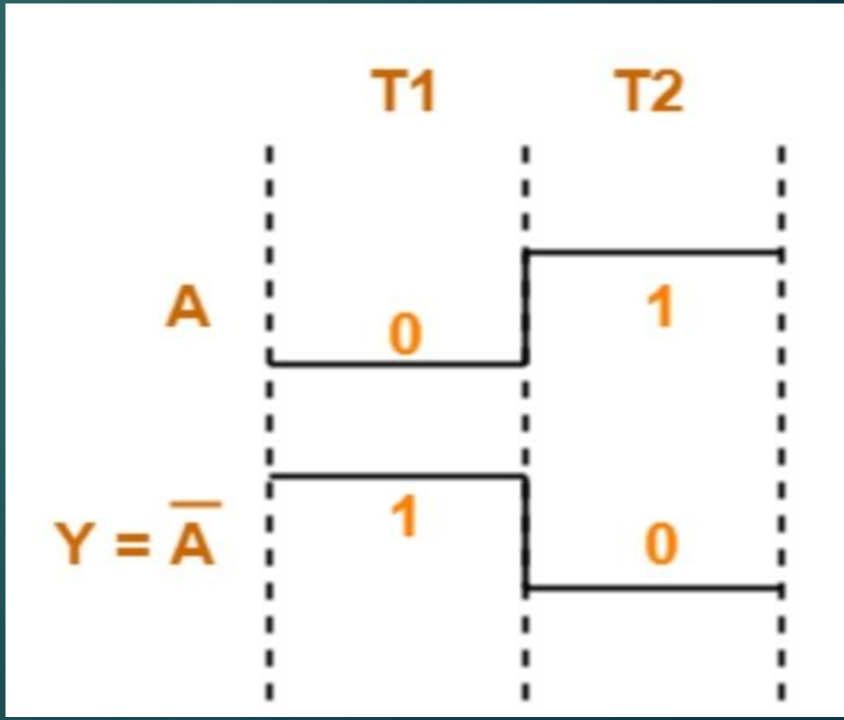
(b) Truth table

L = S1 OR S2

NOT GATE (IC 7404)

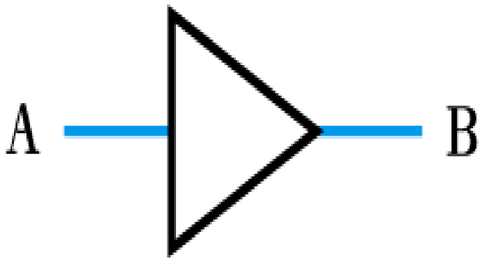
- The output of logic gate is high('1') if its input is low('0').
- The output of logic gate is low('0') if its input is high('1').
- Truth Table:
- A truth table is a means for describing how a logic circuit output depends on the logic levels present at circuits input
- 

(a) Circuit symbol (b) Truth table (c) Boolean expression



BUFFER GATE

- Its output always equals to the its input.



(a) Circuit symbol

A	B
0	0
1	1

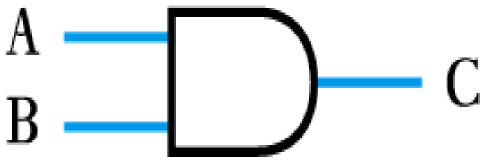
(b) Truth table

$$B = A$$

(c) Boolean expression

AND GATE(IC 7408)

- The "AND" operation is performed exactly like a multiplication of 1's and 0's
- The output of AND gate is high('1') when all the inputs are high('1').
- The output of AND gate is low(0) if any of its input is low('0').



(a) Circuit symbol

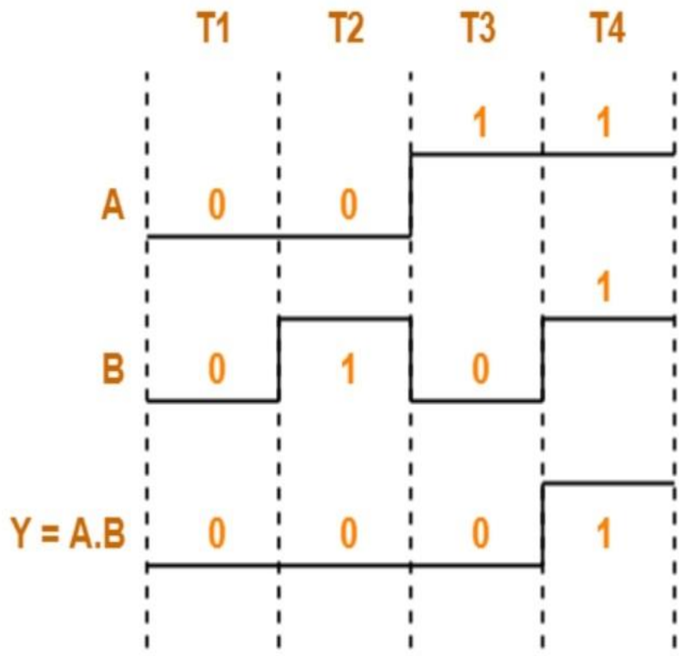
A	B	C
0	0	0
0	1	0
1	0	0
1	1	1

(b) Truth table

$$C = A \cdot B$$

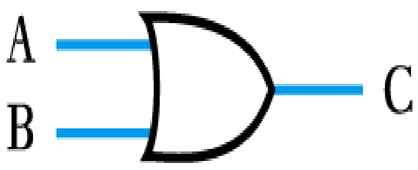
(c) Boolean expression

TIMING DIAGRAM:



OR GATE(IC 7432)

- In"OR" operation any one of the input is high, ouput is high."+" sign stands for OR operation
- The output of OR gate is high('1') when any one of the inputs are high('1').
- The output of OR gate is low(0) when both input are low('0').



(a) Circuit symbol

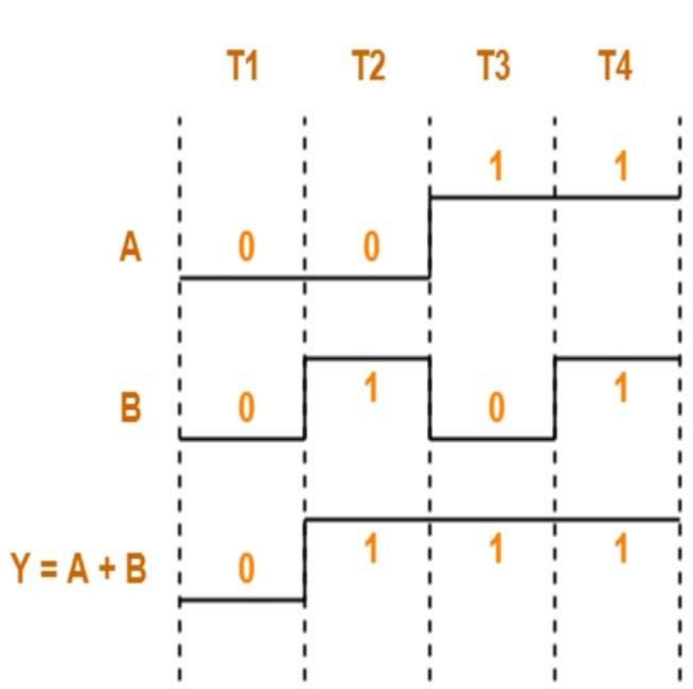
A	B	C
0	0	0
0	1	1
1	0	1
1	1	1

(b) Truth table

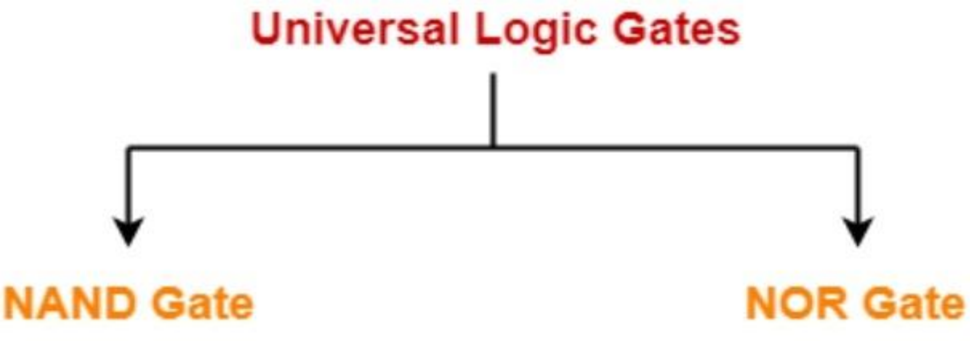
$$C = A + B$$

(c) Boolean expression

TIMING DIAGRAM:



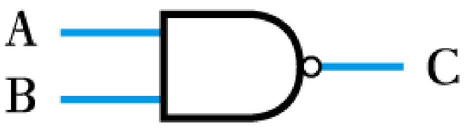
UNIVERSAL LOGIC GATES



They have the following properties-

- Universal gates are not associative in nature.
- Universal gates are commutative in nature.

NAND GATE



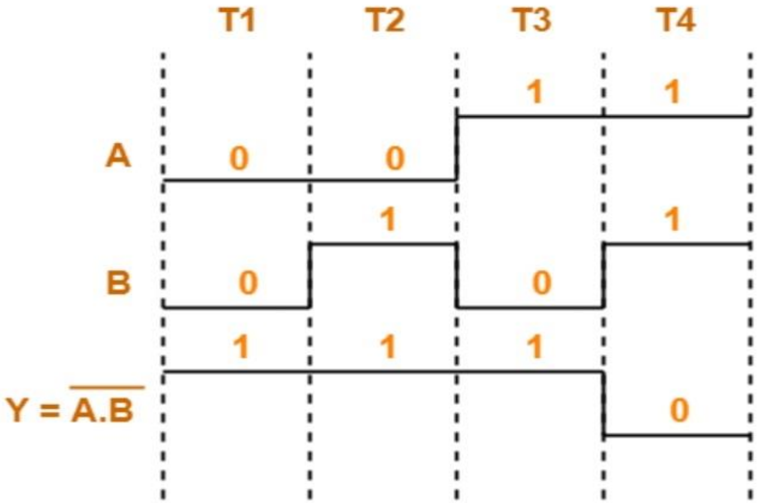
(a) Circuit symbol

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

(b) Truth table

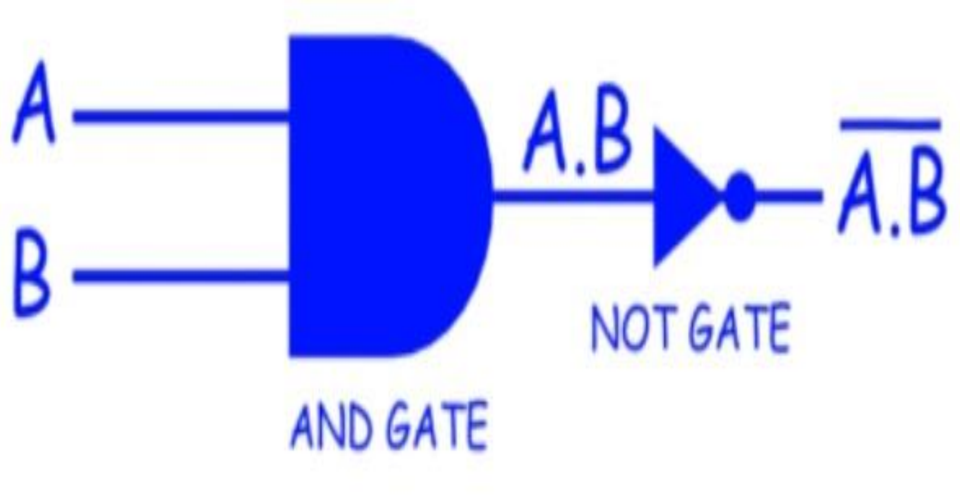
$$C = \overline{A \cdot B}$$

(c) Boolean expression



Nand gate(IC 7400)

- Operation:



A NAND Gate is constructed by connecting a NOT Gate at the output terminal of the AND Gate.

The output of NAND gate is high ('1') if at least one of its inputs is low ('0').

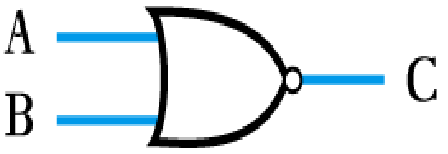
The output of NAND gate is low ('0') if all of its inputs are high ('1').

"NAND" gate operation like an "AND" gate followed by an "INVERTER". So we can say it is "NOT-AND"operation.

- In NAND operation: "ENABLE" input is Logic 1"
  "DISABLE "input is Logic "0"

NOR GATE(IC 7402)

- The “NOR” operation is performed exactly opposite of “OR” operation
- The output NOR gate is high (‘1’) when all the inputs are low(‘0’) .
- The output NOR gate is low(‘0’) if any one of the input is high(‘1’)



(a) Circuit symbol

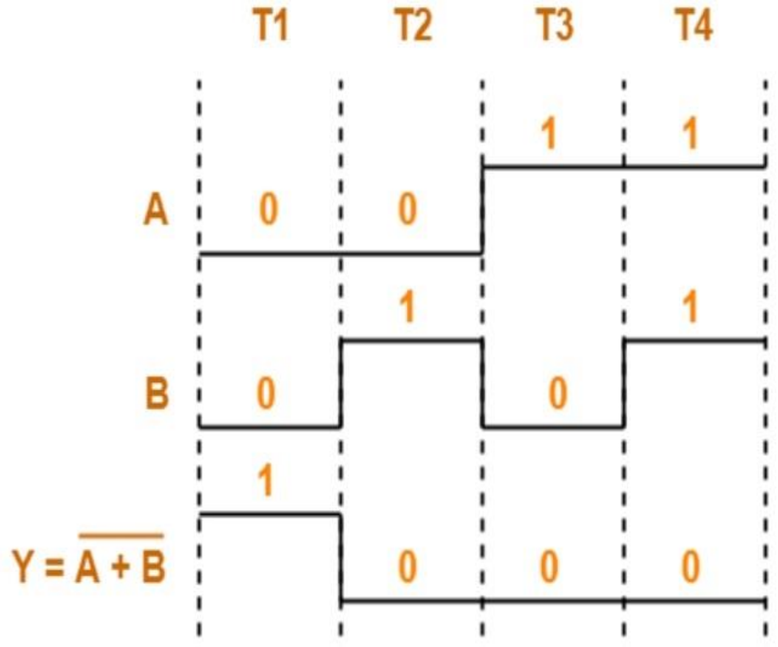
A	B	C
0	0	1
0	1	0
1	0	0
1	1	0

(b) Truth table

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

(c) Boolean expression

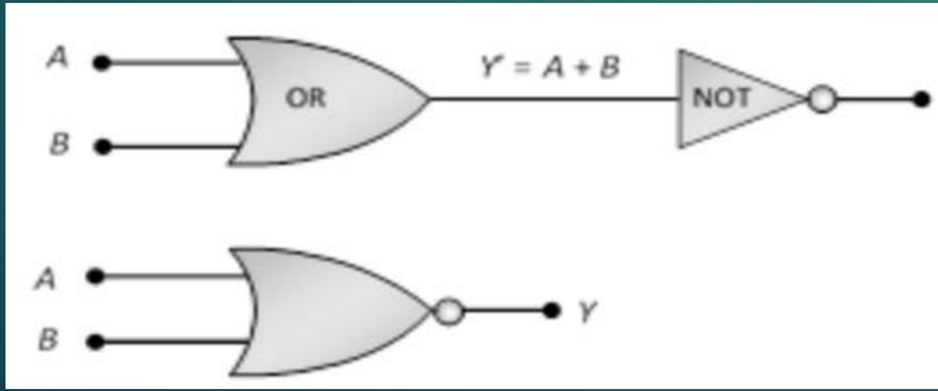
TIMING DIAGRAM:



Continued………

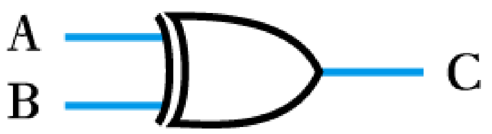
- "NOR" gate operation like an "OR" gate followed by an "INVERTER". So we can say it is "NOT- OR"operation.
- NOR gate output is exact inverse of the OR gate output for all possible input conditions.
- "NOR" gate output goes "LOW" when any of the input is "HIGH".
- In NOR operation: "ENABLE" input is Logic "0"

  "DISABLE "input is Logic "1"



EX-OR GATE(IC 7486)

- “EXOR” operation is not a basic operation and can be performed using the basic gates or universal gates
- It acts like a “odd no of 1’s detector” in the input
- It is also called “stair case switch”
- It is mostly used in “parity generator and detector”



(a) Circuit symbol

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

(b) Truth table

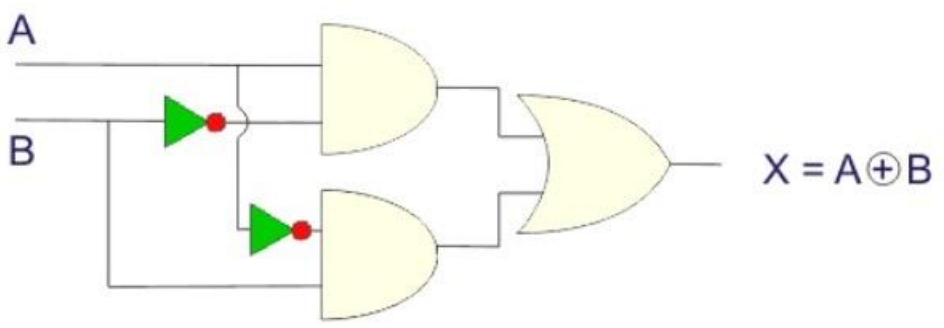
$$C = A \oplus B$$

(c) Boolean expression

Continued……….

- “EXOR” gate performs the “modulo sum” operation without including carry is known as “EXOR” gate.
- An EXOR gate is normally two input logic gate
- When only one of the input is ‘1’ output is logic ‘1’, otherwise output is logic ‘0’
- XOR gate is also known as “Anti coincidence gate or Inequality detector”

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

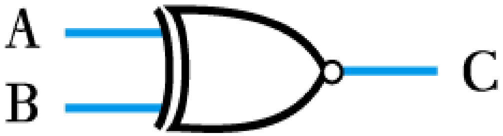


From the left side expression:If A=B $\Rightarrow$ X=0

If A ≠ B $\Rightarrow$ X=1

EX-NOR GATE(IC 74266)

- “EX-NOR” gate also called “Equivalence gate” or “ Coincidence logic circuit”
- It acts like a “even no of 0’s detector”
- It is used in arithmetic circuit
- EX-NOR operation like an EXOR gate followed by an “Inverter”



(a) Circuit symbol

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

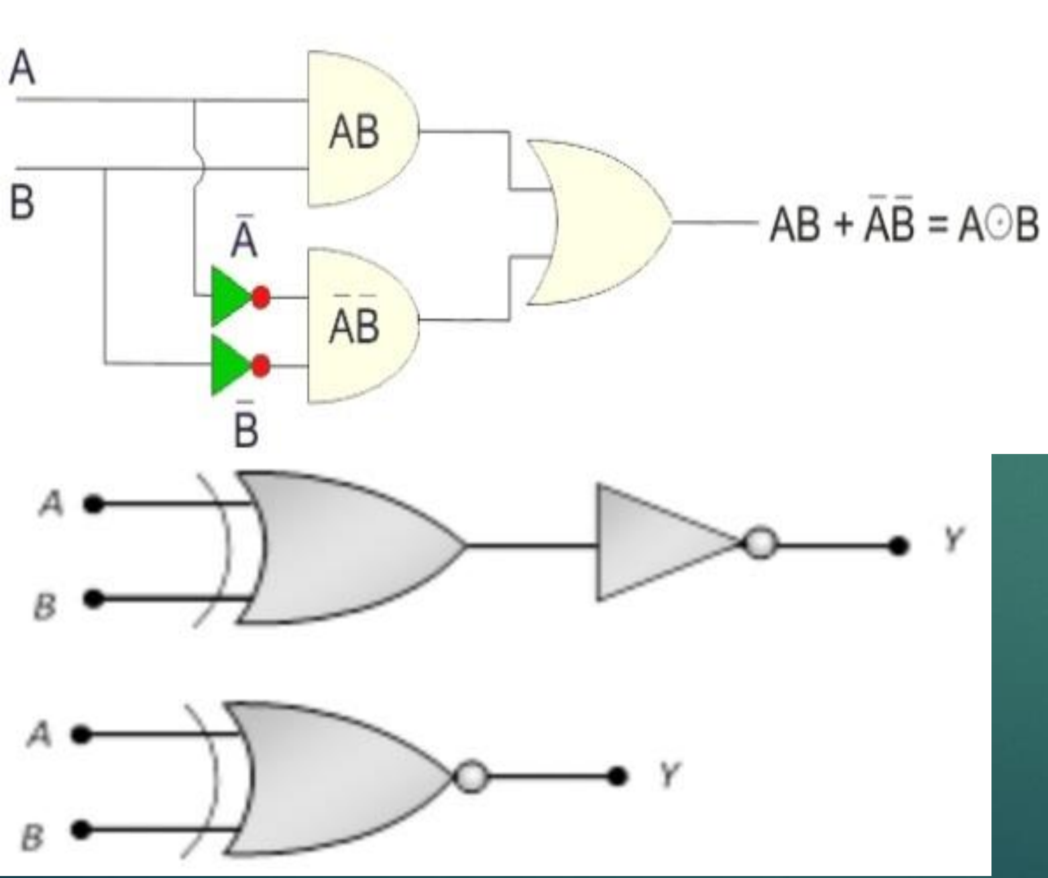
(b) Truth table

$$C = \overline{A \oplus B}$$

(c) Boolean expression

Continued………

- The logical "EX-NOR" operation is represented by dot is surrounded by a circle
- It acts like a "even no of 0's detector"



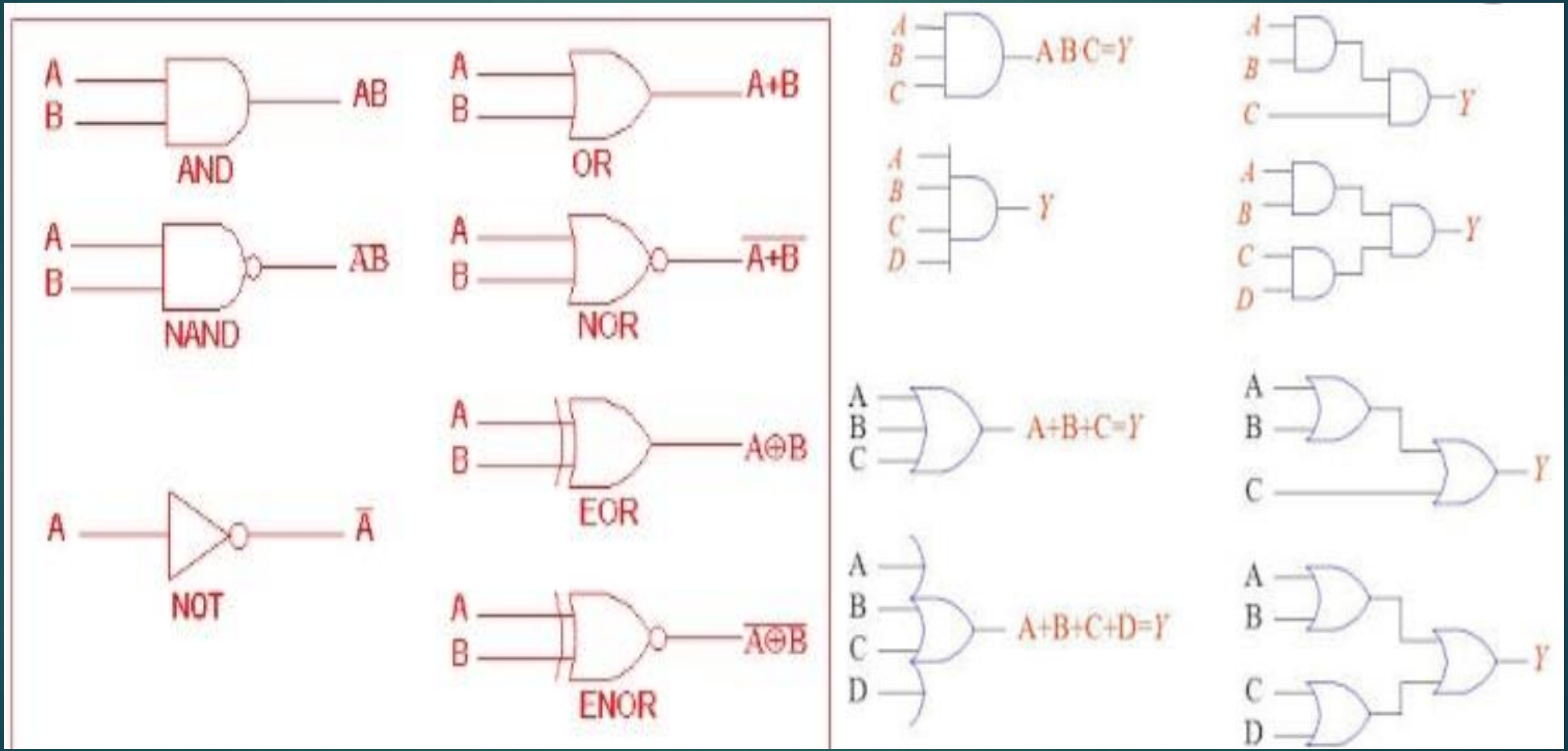
$$When,\ A = 0,\ and\ B = 0,\ AB + \bar{A}\,\bar{B} = 0.0 + \bar{0}.\,\bar{0} = 0.0 + 1.1 = 1.$$

$$When,\ A = 0,\ and\ B = 1,\ AB + \bar{A}\,\bar{B} = 0.1 + \bar{0}\,.\bar{1} = 0.1 + 1.0 = 0.$$

$$When,\ A = 1,\ and\ B = 0,\ AB + \bar{A}\,\bar{B} = 1.0 + \bar{1}\,.\bar{0} = 1.0 + 0.1 = 0.$$

$$When,\ A = 1,\ and\ B = 1,\ AB + \bar{A}\,\bar{B} = 1.1 + \bar{1}\,.\bar{1} = 1.1 + 0.0 = 1.$$

SUMMERY OF ALL LOGIC GATES



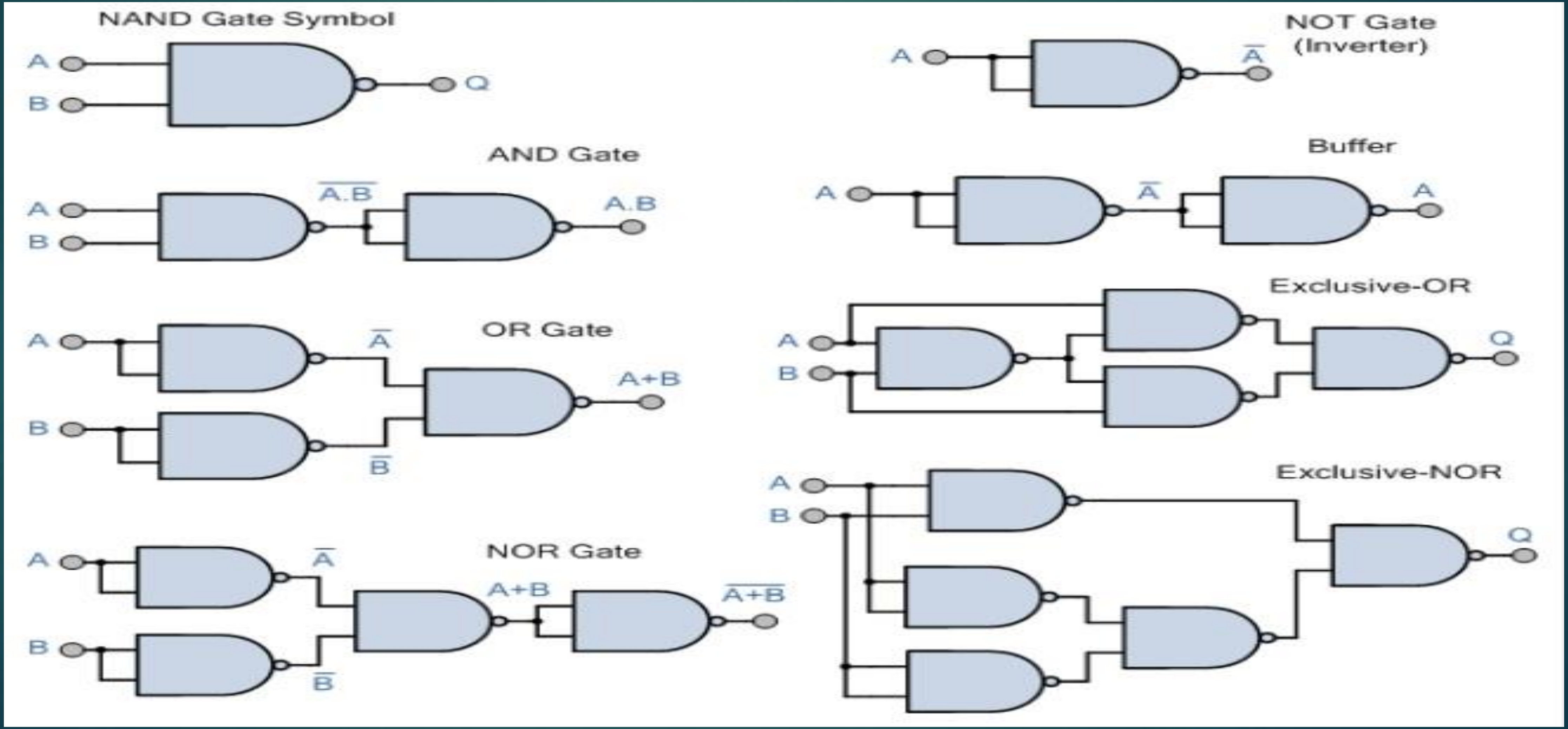
SUMMERY OF ALL LOGIC GATES

Inputs		Truth Table Outputs For Each Gate					
A	B	AND	NAND	OR	NOR	EX-OR	EX-NOR
0	0	0	1	0	1	0	1
0	1	0	1	1	0	1	0
1	0	0	1	1	0	1	0
1	1	1	0	1	0	0	1

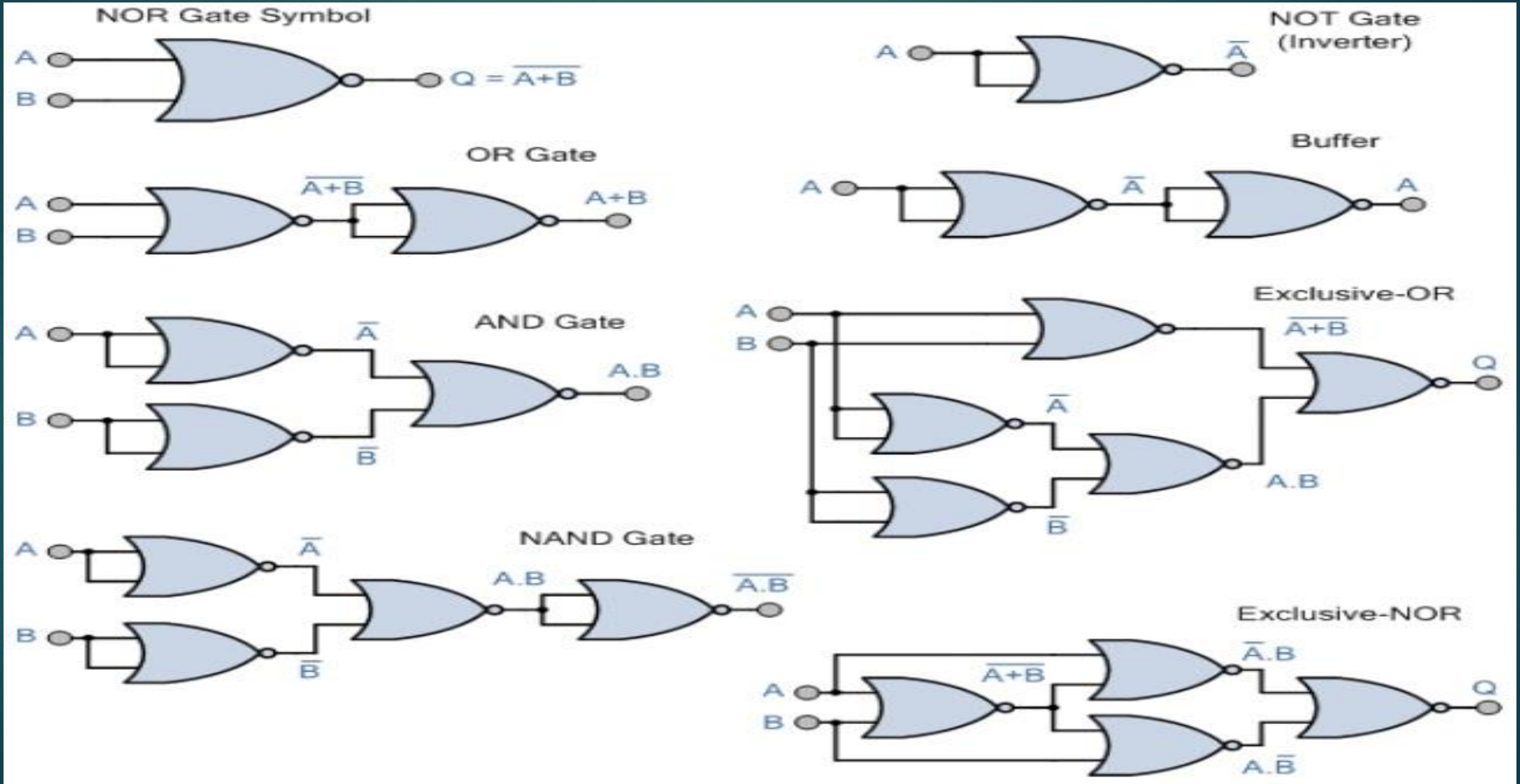
Logic Function	Boolean Notation
AND	$A.B$
OR	$A+B$
NOT	$\overline{A}$
NAND	$\overline{A.B}$
NOR	$\overline{A+B}$
EX-OR	$(A.\overline{B}) + (\overline{A}.B)$ or $A \oplus B$
EX-NOR	$(A.B) + (\overline{A}.\overline{B})$ or $\overline{A \oplus B}$

UNIVERSALITY OF LOGIC GATES

NAND GATE AS UNIVERSAL GATE



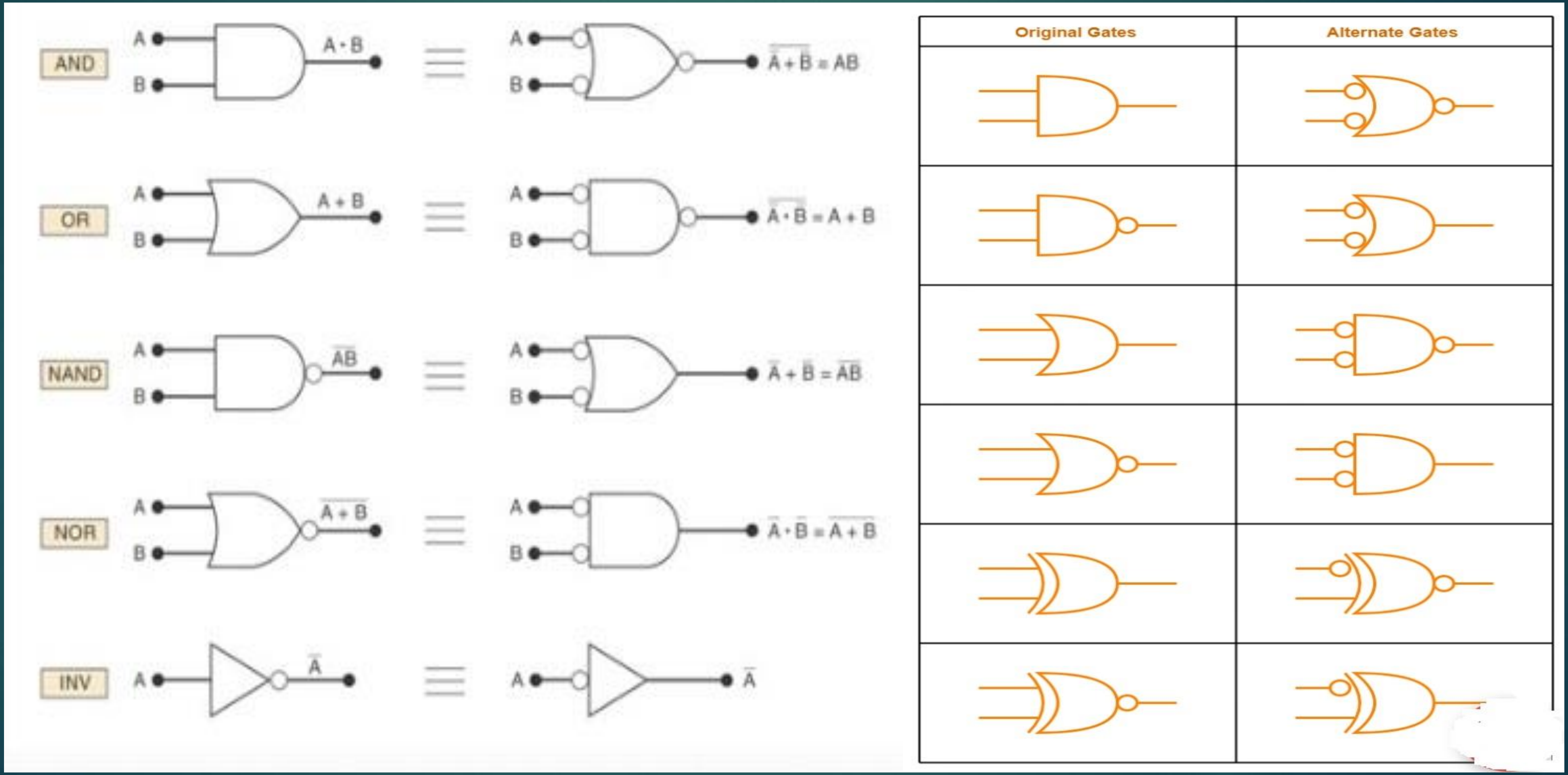
NOR GATE AS UNIVERSAL GATE



REQUIRED UNIVERSAL LOGIC GATES TABLE

Logic Gates	No.of NAND gates required	No.of NOR gates required
NOT	1	1
AND	2	3
OR	3	2
EX-OR	4	5
EX-NOR	5	4
NAND	1	4
NOR	4	1

ALTERNATE LOGIC GATES symbols



ALTERNATE LOGIC GATES TABLE

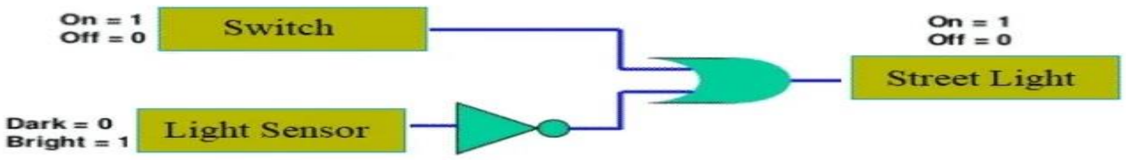
Normal Gate	Alternate gate
AND	Bubbled - NOR
OR	Bubbled – NAND
NAND	Bubbled – OR
NOR	Bubbled – AND
EX-OR	Single input bubbled- EX-NOR
EX-NOR	Single input bubbled- EX-OR
AND-OR Logic	NAND-NAND Logic
OR-AND Logic	NOR-NOR Logic

APPLICATIONS OF LOGIC GATES

- NAND are used in Burglar alarms and buzzers
- They are used in push button switches
- They are used in street light systems
- AND gates are used to transfer of the data
- They are also used in “Door bells”

- Switch: On = 1, Off = 0
- Light Sensor: Dark = 0, Bright = 1
- Street Light: On = 1, Off = 0

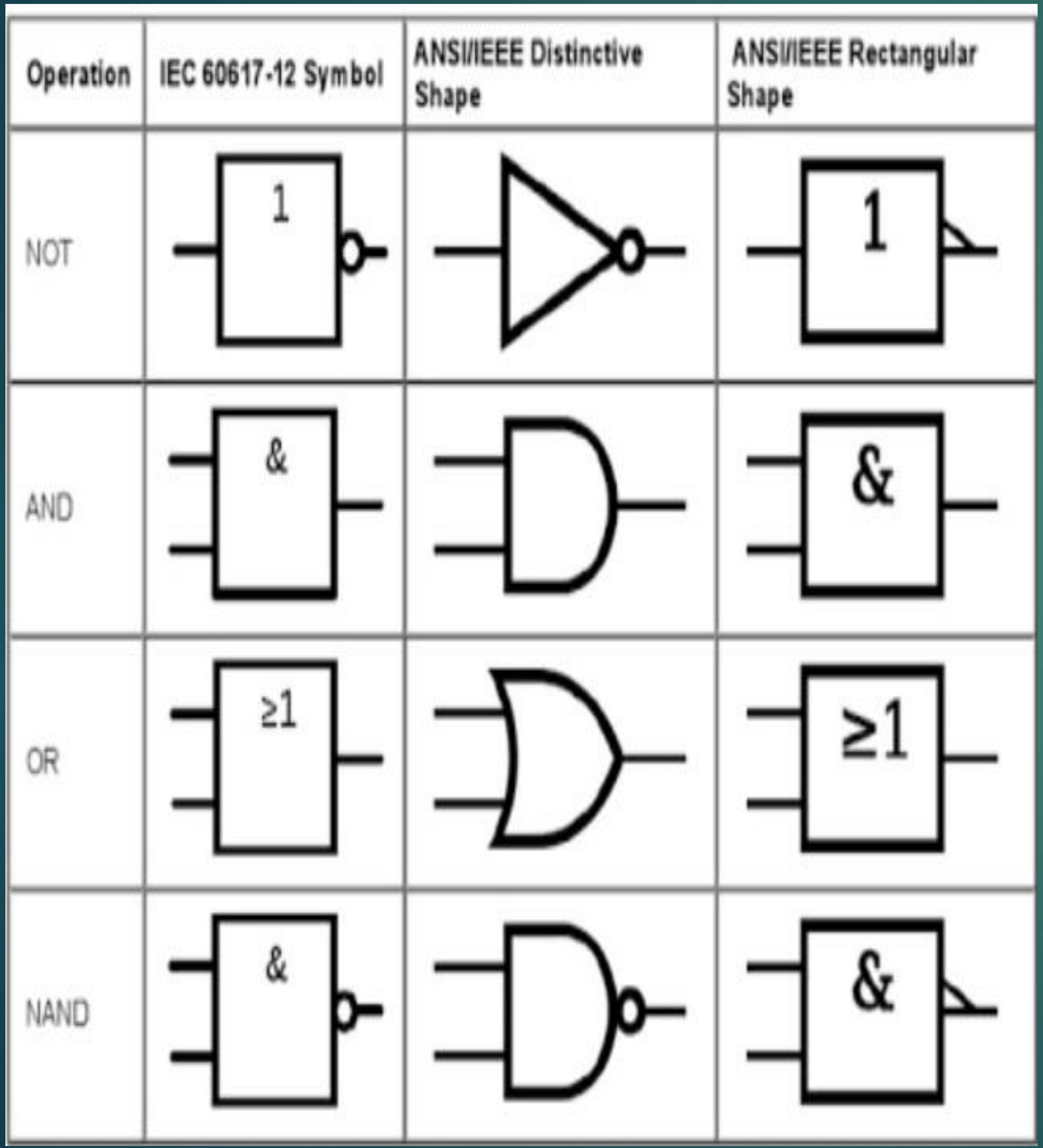
Based on this, we can prepare a Truth Table as shown in Fig. 10

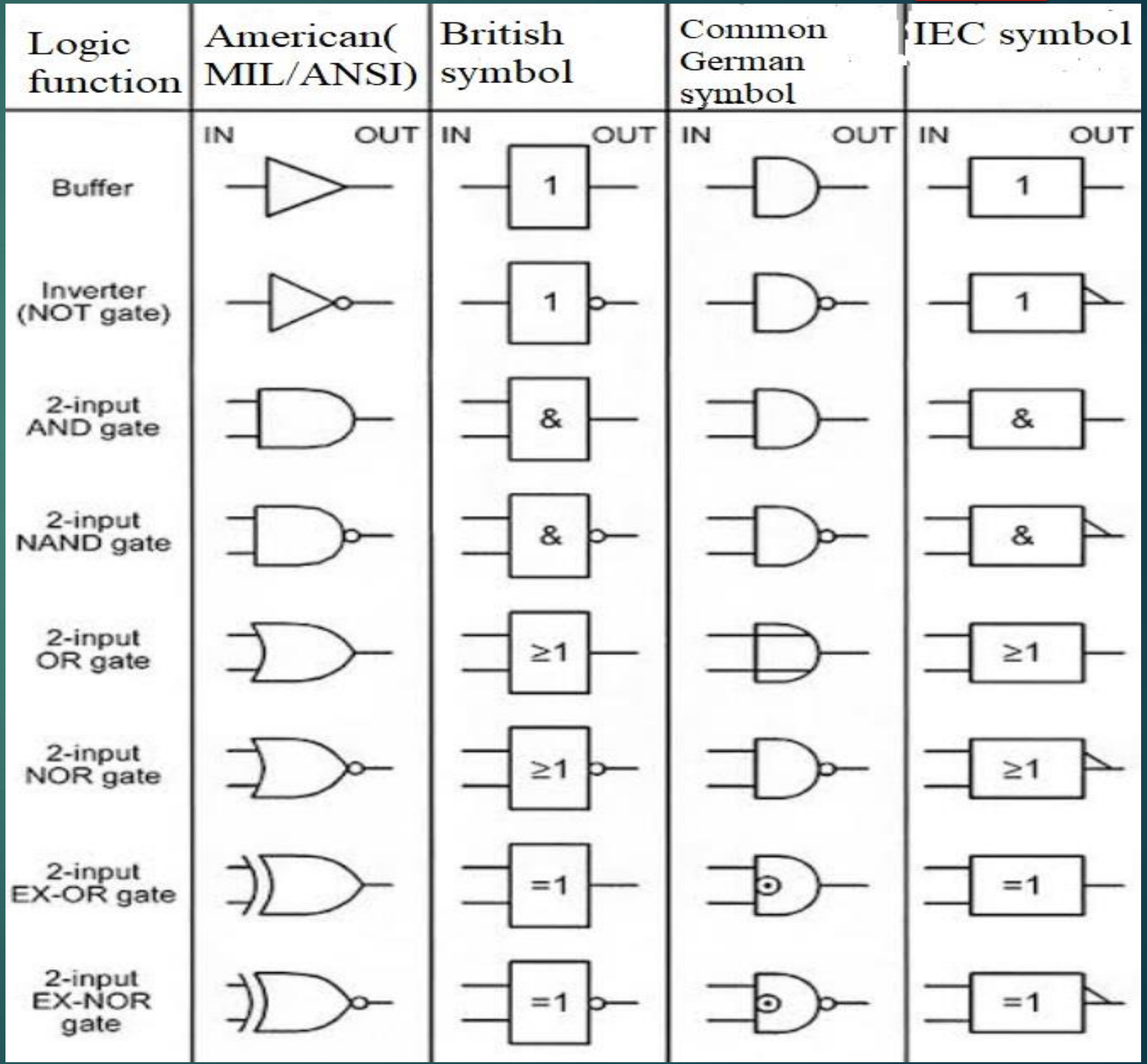


TRUTH TABLE

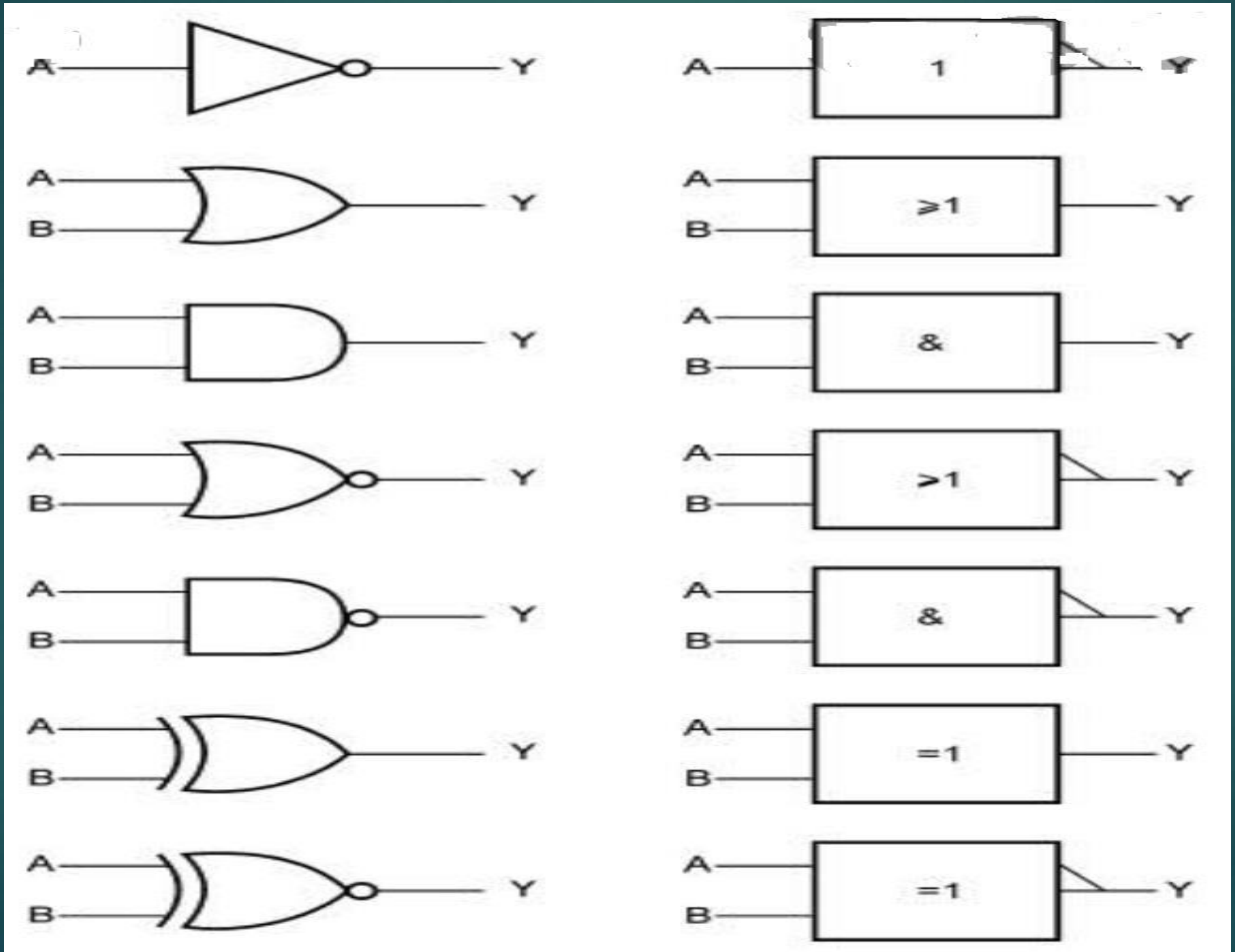
Switch	Light Sensor	Street Light
On	Dark	On
On	Bright	On
Off	Dark	On
Off	Bright	Off

ANSI/IEEE STANDARD LOGIC SYMBOLS





ANSI/IEEE STANDARD LOGIC SYMBOLS



BOOLEAN ALGEBRA

BOOLEAN ALGEBRA

- "Boolean Algebra" is a tool for the analysis and design of digital systems
- In this Algebra there are no fractions, no negative numbers, no square roots, no cube roots, no decimal, no logarithms etc.
- Boolean Constants:
- There are two Boolean constants. (i) '0' (False)

  (ii) '1'(True)
- Boolean Variables:
- Variables that can only take the values of '0' or '1'
- Boolean Functions:
- Each of the logic functions(such as AND,OR and NOT)are represented by symbols as described above
- The mathematical system of binary logic is called Boolean algebra or switching algebra

BOOLEAN ALGEBRAIC LAWS

- Commutative Law: $A + B = B + A$ and $A . B = B . A$
- Associative Law: $A + (B + C) = (A + B) + C = A + B + C$

  $A .(B . C) = (A . B) . C = A .B . C$
- Distributive Law: $A (B + C) = A B + A C$

  $(A + B) (C + D) = A C + A D + B C + B D$
- Involution Law: $\bar{\bar{A}} = A$
- Idempotent Law: $A + A = A$ & $A . A = A$
- Dominant Law: $A + 1 = 1$ & $A . 0 = 0$
- Complement Law: $A + \bar{A} = 1$ & $A . \bar{A} = 0$
- Identity Law: $A + 0 = A$ & $A . 0 = 0$

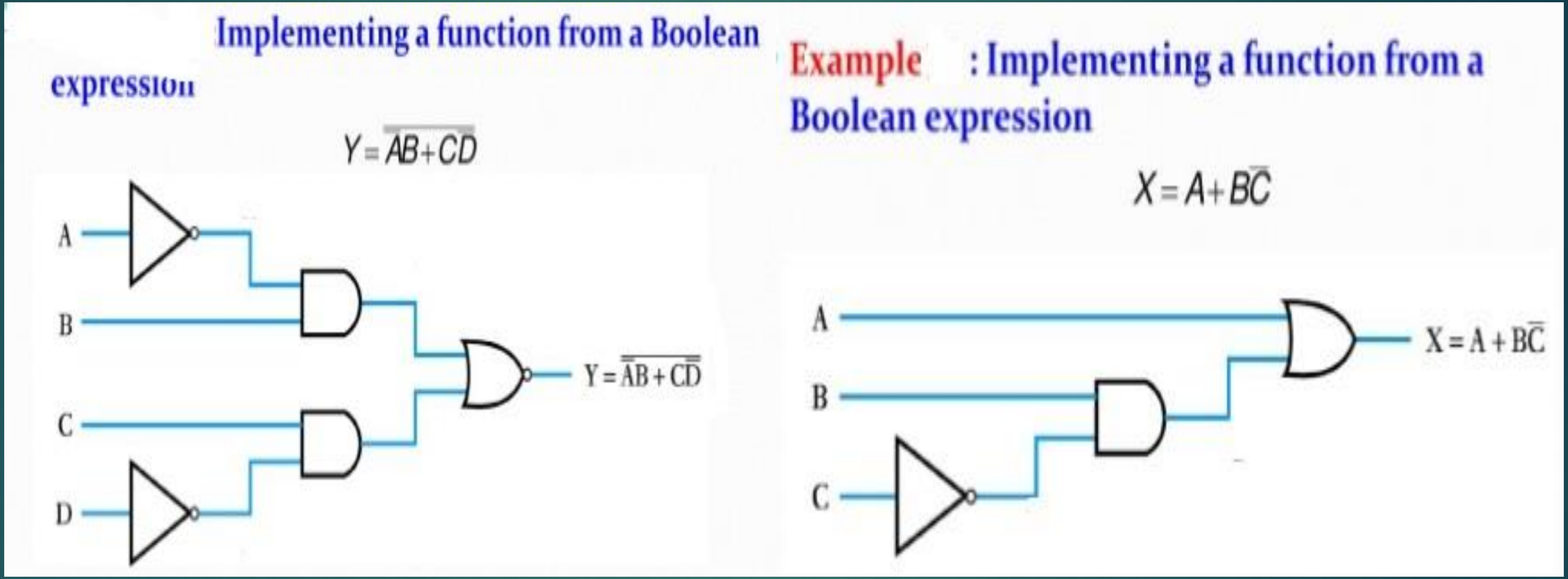
BOOLEAN ALGEBRAIC THEOREMS

- AND operation Theorem: $A \,.\, A = A$
  $A \,.\, 0 = 0$
  $A \,.\, 1 = A$
  $A \,.\, \bar{A} = 0$
- Distribution Theorem: $A + B\,C = (A + B)\,(A + C)$
- Involution Theorem: $\bar{\bar{A}} = A$
- Transposition Theorem : $(A + B)\,(A + C) = A + B\,C$
- De Morgan's Theorem: $\overline{A.B} = \bar{A} + \bar{B}$
  $\overline{A + B} = \bar{A}\,.\,\bar{B}$
- OR operation Theorem : $A + \bar{A} = 1$ & $A + A = A$
  $A + 0 = A$
  $A + 1 = 1$

BOOLEAN ALGEBRAIC THEOREMS

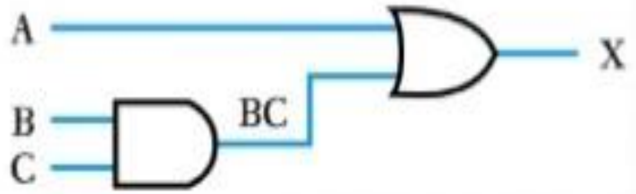
- Duality Theorem: It is one of the elegant theorem proved in advance mathematics
- Dual expression is equivalent to write a negative logic of the given Boolean relation
- Changes each "OR" sign by an "AND" sign and vice-versa
- Complement any "0" or "1" appearing in expression
- Keep literals as it is.
- Example: write self dual expression of Boolean relation
- $\bar{A}\,B\,C + A\,B\,\bar{C} + A\,\bar{B}\,\bar{C} = (\bar{A} + B + C)\,(A + B + \bar{C})\,(A + \bar{B} + \bar{C})$
- Complementary theorem:
- Change each "OR" sign by "AND" and vice-versa
- Complement any "0" or "1" appearing in expression
- Complement the individual literals
- Example: write complement function of Boolean relation
- $\bar{A}\,B\,C + A\,B\,\bar{C} + A\,\bar{B}\,\bar{C} = (A + \bar{B} + \bar{C})\,(\bar{A} + \bar{B} + C)\,(\bar{A} + B + C)$

IMPLEMENTING logic CIRCUITS FROM BOOLEAN EXPRESSIONS



IMPLEMENTING BOOLEAN EXPRESSIONS from logic CIRCUITS

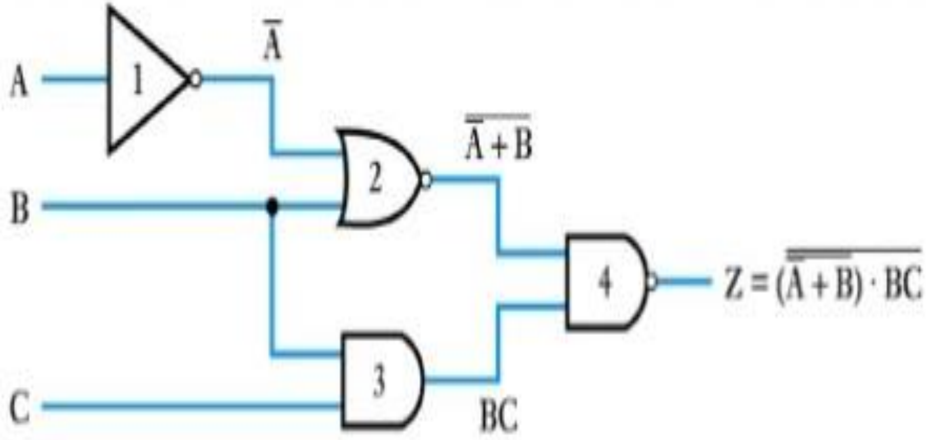
1. Generating a Boolean expression from a logic diagram



2 Implementing a function from a Boolean expression

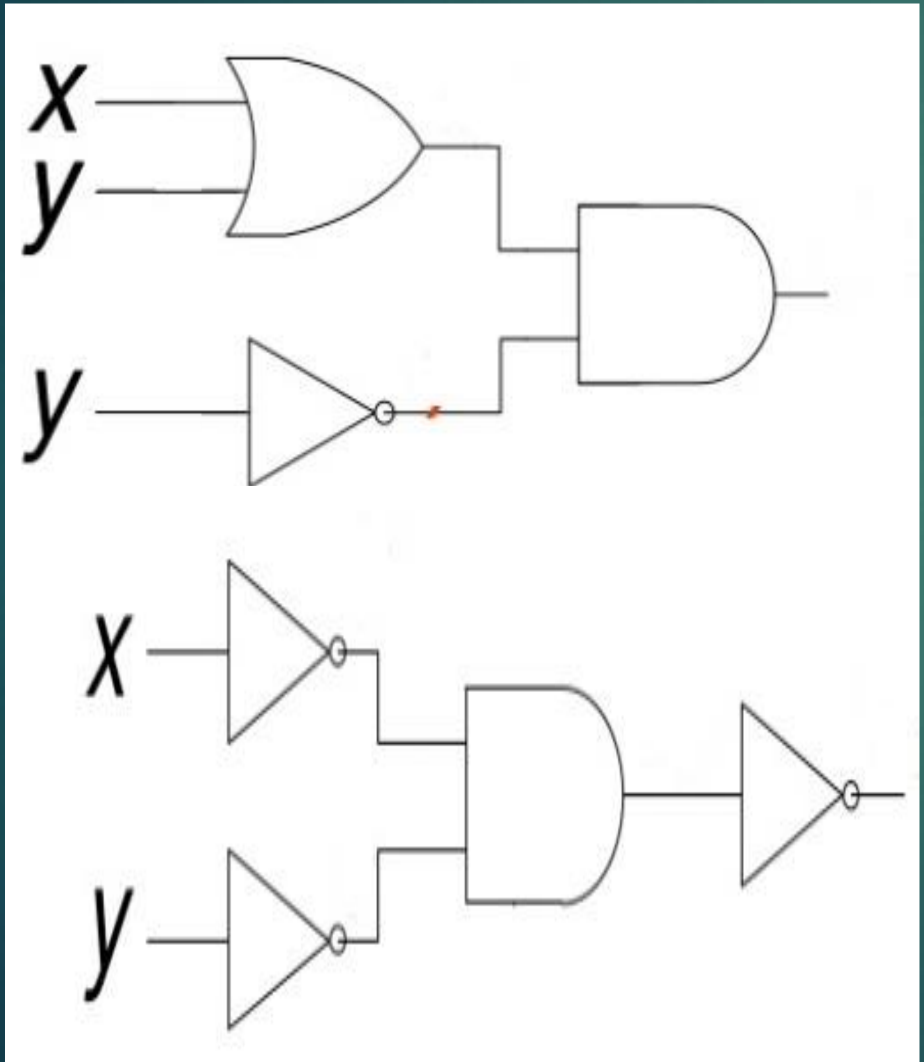
$$X = (A+B) \bullet \overline{(AB)}$$

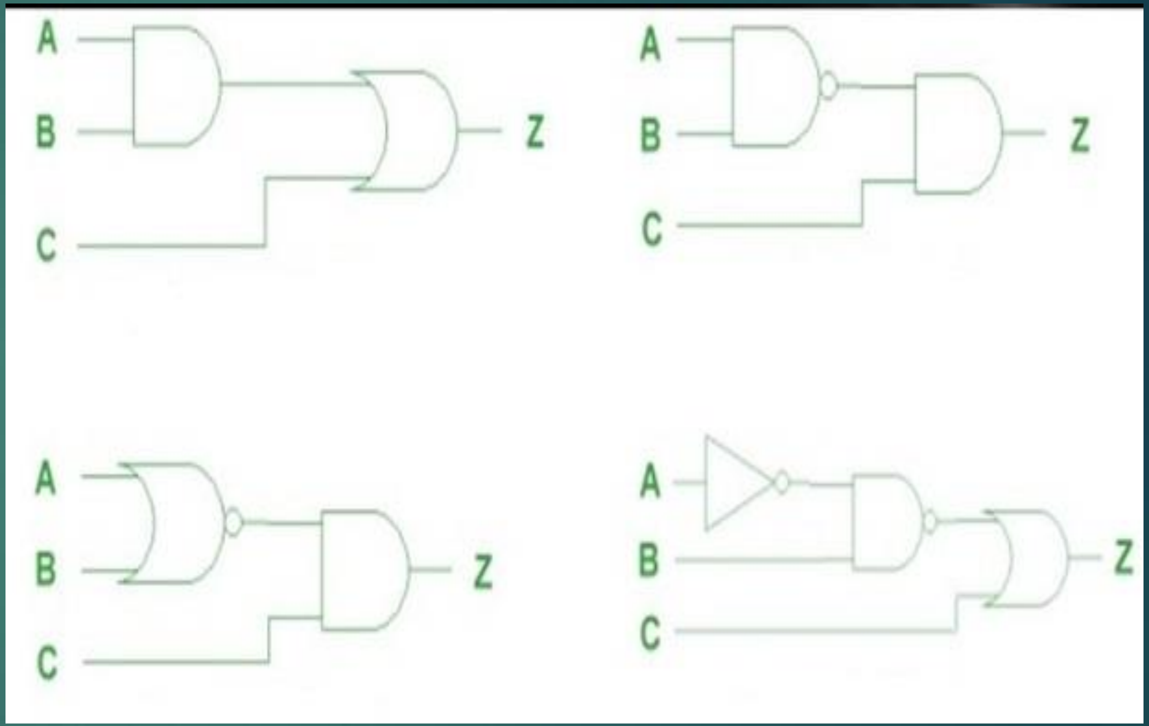
Example :Generating a Boolean expression from a logic diagram



IMPLEMENTING BOOLEAN EXPRESSIONS from logic CIRCUITS

- Findout the output for the following logic circuits





ERROR DETECTION AND CORRECTION

INTRODUCTION

- In analog representation the bits 0 and 1 corresponds to two different ranges
- During the transmission of binary data from one system to other system noise may be added
- Due to this noise there may be a errors in the received data at other system
- That means a bit 0 may change to 1 or a bit 1 may change to 0
- We cannot avoid the interference of noise
- But we can get back the original data first by detecting whether any errors present and then correcting those errors
- So there are two types of codes are available :(i) Error detection codes
- (ii) Error correction codes
- Error detection codes: These codes are used to detect the errors present in the received data bit stream
- These codes are contain some bits ,which are included appended to the original bit stream
- These codes are detect the error, if it is occurred during the transmission of original data
- Example: Parity codes, Hamming codes

Continued………

- Error correction codes: Error correction codes are used to correct the errors present in the received data bit stream.
- So that we will get original data
- Error correction codes also use the similar strategy of error detection codes
- Example: Parity Hamming codes
- Therefore to detect and correct the errors additional bits are appended to the data bits at the time of transmission
- Parity code: It is to include append one parity bit either to the of MSB or to the right of LSB of original bit stream
- There are two types of parity codes: (i) Even parity code
- (ii) Odd parity code
- Even parity code: The value of even parity bit should be zero ,if even no.of ones present in the binary code. Other wise it should be 1
- So that even no.of ones present in even parity code.
- Even parity code contains the data bits and even parity bits

EVEN PARITY

- Even parity code:

Binary Code	Even Parity bit	Even Parity Code
000	0	0000
001	1	0011
010	1	0101
011	0	0110
100	1	1001
101	0	1010
110	0	1100
111	1	1111

ODD PARITY

- Odd parity code:The value of odd parity bit should be “zero”,if odd no.of ones Present in the binary code.
- Otherwise it should be “1”.
- So that odd number of ones present in odd parity code.
- Odd parity code contains the data bits and odd parity bits.

Binary Code	Odd Parity bit	Odd Parity Code
000	1	0001
001	0	0010
010	0	0100
011	1	0111
100	0	1000
101	1	1011
110	1	1101
111	0	1110

HAMMING CODE

- Hamming code:In data communication a Hamming code is an error correcting code
- Hamming codes can “detect single and upto 3 bits error”, and “correct single bit error” as well
- Parity:Parity (from latin paritas,means equal) is a techniques that checks whether data has been lost or written over,when it is moved from one place in storage to another or when it is transmitted between computers

ERROR CORRECTION & DETECTION

- Example: Given data (or) message bits 1001
- Hamming code formula: $2^K \geq n + k + 1$
- From the above formula “k” represents the “Parity bits”
- “ n ” represents the no.of data bits
- From the above example n = 4 bits & “ k” is 3
- $2^K \geq 4 + k + 1 = 2^K \geq 5 + k$
- From the above equation “k” satisfied only by “3”
- $2^3 \geq 4 + 3 + 1 = 8$

ERROR CORRECTION AND DETECTION

Bit Designation	D_7	D_6	D_5	P_4	D_3	P_2	P_1
Bit location	7	6	5	4	3	2	1
Bit location number	111	110	101	100	011	010	001
Data bits (or) message bits	1	0	0	?	1	?	?

Q.Construct a 7,4 hamming code with odd parity for the message 1001

111	110	101	100	011	010	001
D_7	D_6	D_5	P_4	D_3	P_2	P_1
1	0	0	?	1	?	?

- For finding $P_1 \rightarrow$ collect the data bits which are having first bit as 1(i.e.,1,3,5,7)
- For finding $P_2 \rightarrow$ collect the data bits which are having second bit as 1(i.e.,2,3,6,7)
- For finding $P_4 \rightarrow$ collect the data bits which are having third bit as 1(i.e.,4,5,6,7)

Contd…

- $P_1 \rightarrow 1,3,5,7 \rightarrow P_1\,101$ should have odd Parity, hence $P_1 = 1$
- $P_2 \rightarrow 2,3,6,7 \rightarrow P_2\,101$ should have odd Parity, hence $P_2 = 1$
- $P_4 \rightarrow 4,5,6,7 \rightarrow P_4\,001$ should have odd Parity, hence $P_4 = 0$

Transmitted data $\rightarrow D_7\ \ D_6\ \ D_5\ \ P_4\ \ D_3\ \ P_2\ \ P_1$

1 0 0 0 1 1 1

Q) A 7,4 hamming code is received as 1010111 using odd parity then Determine the transmitted message.

- Received data→D_7 D_6 D_5 P_4 D_3 P_2 P_1

  1 0 1 0 1 1 1

- Sol: first we have to find the error

  location with the order P_4 P_2 P_1

- P_4→4,5,6,7→0101→Even parity → error occurred hence $P_4 = 1$
- P_2→2,3,6,7→1101→odd parity → no error hence $P_2 = 0$
- P_1→1,3,5,7→1111→Even parity →error occurred hence $P_1 = 1$

Contd….

- Error location is given by

  $P_4\ P_2\ P_1 = 1\ 0\ 1 =$ 5th position

  $4\ 2\ 1$

- Received data $\rightarrow$

D_7	D_6	D_5	P_4	D_3	P_2	P_1
1	0	1	0	1	1	1
		$\downarrow$				
		0				

- Corrected data $\rightarrow$

D_7	D_6	D_5	P_4	D_3	P_2	P_1
1	0	0	0	1	1	1

- Transmitted message $\rightarrow$ 1001