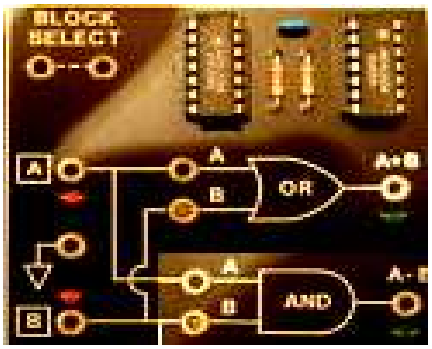
Module – 5 Digital Electronics Fundamentals



5.1 Introduction

5.2 Difference between analog and digital signals

5.3 Number System - Binary, Hexadecimal, Conversion - Decimal to binary, Hexadecimal to decimal and vice-versa

5.4 Boolean algebra

5.5 Basic and Universal Gates

5.6 Half and Full adder

5.7 Multiplexer

5.8 Decoder

5.9 SR and JK flip-flops

5.10 Shift register

5.11 Three bit Ripple Counter

5.12 Basic Communication system

5.13 Principle of operations of Mobile phone

5.1 Introduction

- Digital Electronics is a branch of Electronics which deals with digital circuits and digital systems that exists in only two possible states. These states described with only two discrete values (0, 1) which are used to represent numbers, symbols, alphabetic and other type of information in Digital Electronics.
- *Digital circuits* are the circuits which are basically transistors to create logic gates operate at high speed. It is less susceptible to noise than analog circuits. It is also easier to perform error detection and correction with digital signals.
- *Digital systems* contain digital circuits used to process digital information using a binary system, where data can assume with only two possible values: (0, 1). Ex: calculators, computers, etc.

Representation of Binary values: (0, 1)

- Ideally: "no voltage (= 0 V)" represents a logic **0** and "full voltage (= 5V)" represents a logic **1**.
- Realistically: "low voltage" (e.g., <1v) represents a **0** and "high voltage" (e.g., > 4v) represents a **1**.
- These discrete values can be achieved by using transistor switches.
  Type of logic: (i) *positive logic* and (ii) *negative logic* is shown in Fig. 5.1

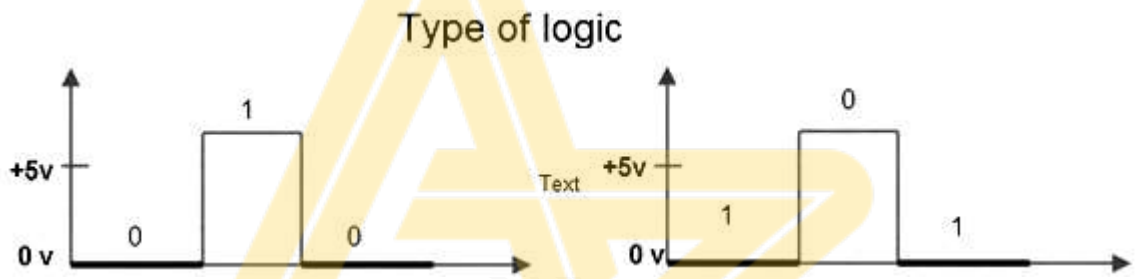


Fig. 5.1 Type of logic: (a) positive logic **(b) negative logic**

- HIGH voltage = 5V → represents logic **1**.
(HIGH = logic 1 = Closed switch = ON state)
- LOW voltage = 0V → represents logic **0**.
(LOW = logic 0 = Open switch = OFF state)

- HIGH voltage = 5V → represents logic **0**.
- LOW voltage = 0V → represents logic **1**.

Digital Waveforms: Digital waveforms as typical voltage ranges for positive logic 1 and logic 0 are shown in fig.5.2.

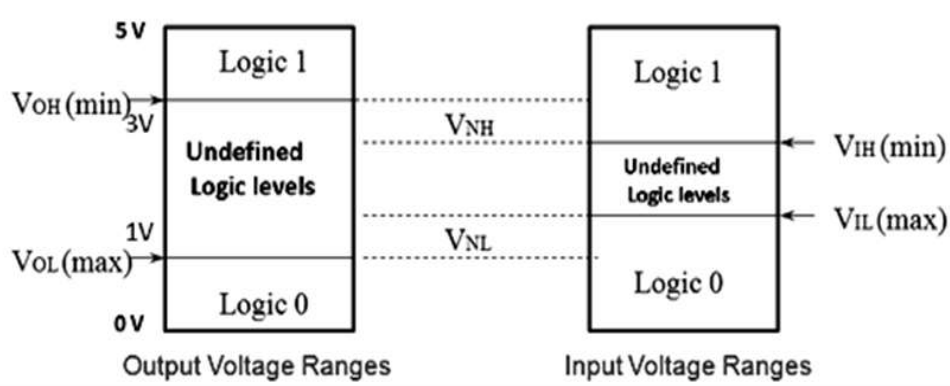
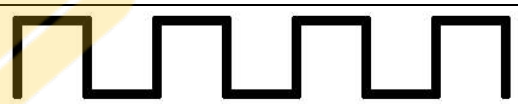


Fig. 5.2 Digital Waveforms as typical acceptable voltage ranges

- In digital electronics the signals are formed with only two voltage values: logic 1 (HIGH) and logic 0 (LOW) and it is called binary digital signal. Therefore, the information contained in the digital signal is represented by the numbers 1and 0. In digital systems the state 1 corresponds to a voltage range from 3V to 5V while the state 0 corresponds to a voltage range from a 0 to 1 volt as shown in fig. 5.2. A logic gate within the 'undefined logic level' will produce an unpredictable output level.
- If noise in the circuit is high enough it can push logic 0 up or drop logic 1 down into the undefined logic level. The magnitude of the voltage required to reach this level is the *noise margin*. A quantitative measure of noise immunity is called *noise margin*.
- Noise immunity of a logic circuit refers to the circuit's ability to tolerate noise voltages on its inputs.
- High Level Noise Margin, $V_{NH} = V_{OH}$ (min) - V_{IH} (min)
- Low Level Noise Margin, $V_{NL} = V_{IL}$ (max) - V_{OL} (max)

5.2 Analog verses Digital signals

Digital systems cover most areas of our life: still pictures, digital video, digital audio, telephone, traffic lights, animation, etc. Analog verses Digital is discussed below.

Parameter	Analog	Digital
Signal	Analog signal is a continuous signal which has infinite number of values in a range.	Digital signals are discrete time signals have limited values (0,1) in a range.
Waves	Denoted by sine waves 	Denoted by square waves 
Representation	Uses continuous range of values to represent information.	Uses discrete or discontinuous values to represent information
Example	Human voice in air, analog electronic devices.	Computers, CDs, DVDs, and other digital electronic devices.
Technology	Analog technology records waveforms as they are.	Samples analog waveforms into a limited set of numbers and records them.
Data transmission	Subjected to deterioration by noise during transmission.	Can be noise-immune without deterioration during transmission.
Response to Noise	More likely to get affected reducing accuracy	Less affected since noise response are analog in nature
Flexibility	Analog hardware is not flexible.	Digital hardware is flexible in implementation.
Uses	Can be used in analog devices only. Best suited for audio and video transmission.	Best suited for Computing and digital electronics.

Benefits of Digital over Analog

- Reproducibility
- Not effected by noise means quality
- Ease of design
- Data protection
- Programmable
- Speed
- Economy

5.3 Number System

Set of values used to represent different quantities is known as Number System.

$$(number)_b = \sum_{n=o}^{N} d_n \; b^n \; = \; d_0 b^0 \; + d_1 b^1 \; + d_2 b^2 \; + \cdots + d_N b^N$$

Where b = number system base or radix
d_n = nth digit
n = number (can start from negative number if the number has a fraction part).
N+1 = the number of digits

The Natural Numbers: The natural numbers are 1, 2, 3, 4, 5, etc. There are infinitely many natural numbers. The whole numbers are the natural numbers together with 0.

The Integers: The integers are the set of real numbers consisting of the natural numbers, their additive, inverses and zero. {..., -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5...}

Some important number systems are as follows.

1. **Decimal number system** - has a base of 10 with each position weighted by a factor of 10.
2. **Octal number system** - has a base of 8 with each position weighted by a factor of 8.
3. **Binary number system** - has a base of 2 with each position weight by a factor of 2.
4. **Hexadecimal number system** - has a base of 16 with each position weighted by a factor of 16.

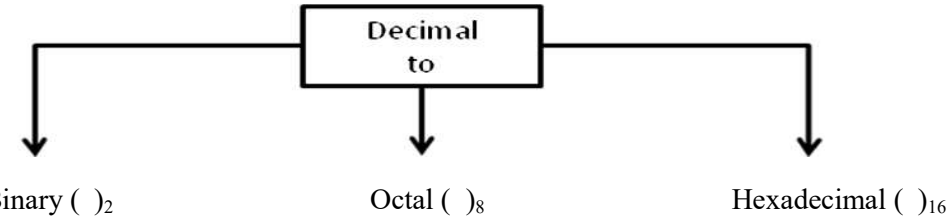
The decimal number system is used in general communication. It can be used to represent both the integer as well as floating point values. The floating point values are generally represented in this system by using a period called decimal point. The decimal point is used to separate the integer part and the fraction part of the given real number. However, the computers use binary number system; they can store and process each type of data in terms of 0s and 1s only. Octal and hexadecimal number systems are used to represent computer data. The details of these number systems are given in the table 1.

Table 1 Number systems

Number System	Radix / base	Digits / symbols	Example: with power notation	
Decimal	10	0,1,2,3,4,5,6,7,8,9	$(338.6)_{10}$	$3x10^2+3x10^1+8x10^0.\ 6x10^{-1}$
Octal	8	0,1,2,3,4,5,6,7	$(336.2)_8$	$3x8^2+3x8^1+6x8^0.\ 2x8^{-1}$
Binary	2	0,1	$(110.10)_2$	$1x2^2+1x2^1+0x2^0.\ 1x2^{-1}+0x2^{-2}$
Hexadecimal	16	0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F	$(4AB.2)_{16}$	$4x16^2+10x16^1+11x16^0.\ 2x16^{-1}$

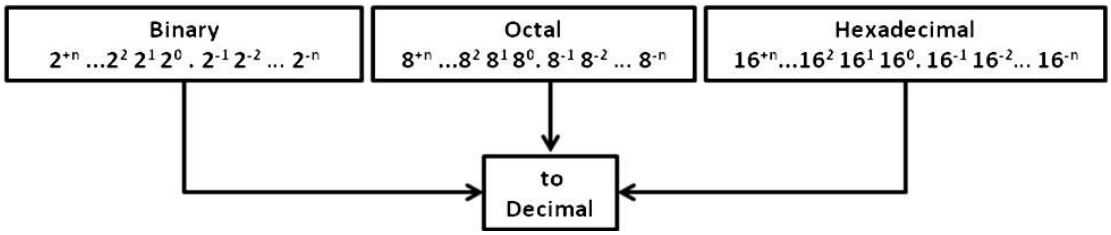
5.3.1 Number conversions: Decimal to other number system (NS)

When all the different number conversions from one to another come at one sequence, certainly learners will get confused. To avoid this and to keep remember easily, we classify all conversions into mainly three groups. With simple rules and examples these are illustrated below.



Rules	
Integer part	**Fraction part**
Successive Division: 1. Divde Decimal integer number by the radix of NS to be converted. 2. Save remainder (read remainders from bottom to top). 3.Repeat steps 1 and 2 until quotient becomes zero	Successive Multiplication: 1. Multiply Decimal fraction number by the Radix of the NS to be converted. 2. Save the whole number (integer) of the result (read remainders from top to bottom). 3. Repeat steps 1 and 2 for the fractional part of step 2 until to the desired position.
Example: 1 Decimal to Binary ⟹ $(25.625)_{10}$ ⟹ $(?)_2$ ⟹ $(11001.101)_2$	
25 ÷ 2 ⟹ 12 (R→ 1) 12 ÷ 2 ⟹ 6 (R→ 0) 6 ÷ 2 ⟹ 3 (R→ 0) 3 ÷ 2 ⟹ 1 (R→ 1) (save remainder (R) backward) 1 ÷ 2 ⟹ 0 (quotient becomes zero) $(25)_{10}$ ⟹ $(11001)_2$	0.625 x 2 ⟹ 1.25 ⟹ 1 0.25 x 2 ⟹ 0.5 ⟹ 0 0.5 x 2 ⟹ 1.0 ⟹ 1 (save whole number forward) $(0.625)_{10} = (0.101)_2$
Example: 2 Decimal to Hexadecimal ⟹ $(675.15)_{10}$ ⟹ $(?)_{16}$ ⟹ $(257.266)_{16}$	
675÷ 16 ⟹ 42 (R→ 3) 42÷ 16 ⟹ 02 (R→ 10) $(675)_{10}$ ⟹ $(2A3)_{16}$ Note: $(10)_{10} = (A)_{16}$ (save remainder (R) backward)	0.15 x 16 ⟹ 2.40 ⟹ 2 0.40 x 16 ⟹ 6.40 ⟹ 6 0.40 x 16 ⟹ 6.40 ⟹ 6 $(0.15)_{10} = (0.266)_{16}$

5.3.2 Number conversion: Other number systems (NS) to Decimal



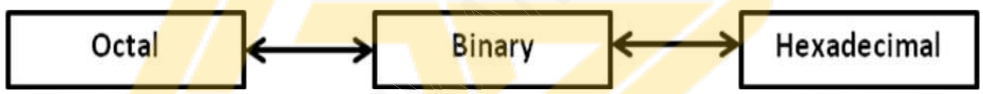
Rules:

1. Obtain the power sequence (positional weights) of the NS to be determined.
2. Place the number in position : *integers* – left of the radix point ⟹ (+ ve powers)
   fraction – right of the radix point ⟹ (- ve powers)
3. Multiply the number with positional weights and add, to get decimal number.

Steps	**Example: Binary to Decimal**	**Example: Hexadecimal to Decimal**
	$(1011.01)_2 \Rightarrow (?)_{10} \Rightarrow (11.25)_{10}$	**$(A2B.1D)_{16} \Rightarrow (?)_{10} \Rightarrow (2603.11252)_{10}$**
power	2^3 2^2 2^1 2^0 . 2^{-1} 2^{-2}	16^3 16^2 16^1 16^0 . 16^{-1} 16^{-2}
weight	8 4 2 1 0.5 0.25	4096 256 16 1 0.0625 0.0039
number	1 0 1 1 . 0 1	0 10 2 11 . 1 13
value	(8 + 0 + 2 + 1) . (0 + 0. 25) = **$(11.25)_{10}$**	(0 + 2560 + 32 + 11) . (0.0625 + 0.052) = **$(2603.11252)_{10}$**

5.3.3 Number conversion: Grouping Binary bits

In this case, Decimal NS will not come into picture. However, either Octal or Hexadecimal NS is converted into Binary NS. Then "grouping" of bits is conducted.



($)_8$ ($)_2$ ($)_{16}$

According to the conversion type, the grouped bits are replaced by Binary Coded Octal (BCO) or Binary Coded Hexadecimal (BCH) as tabulated in the table 2 (see page 7).

Octal ⟹ Binary	**$(\ )_8 \Rightarrow (\ )_2$**	**Hexadecimal ⟹ Binary**	**$(\ )_{16} \Rightarrow (\ )_2$**

- Replace each *octal digit* by its **3-bit** BCO equivalent (see table 2) both for integer and fractional part.
- Replace each *hexadecimal* digit by its 4-bit BCO equivalent (see table 2) both for integer and fractional part.

Example:

$(1073.32)_8 \Rightarrow (?)_2 \Rightarrow (001000111011.011010)_2$

1	**0**	**7**	**3**	**.**	**3**	**2**	[given octal number]
001	000	111	011	.	011	010	[from BCO]

$(A2B.1D)_{16} \Rightarrow (?)_2 \Rightarrow (001000111011.011010)_2$

A	**2**	**B**	**.**	**1**	**D**	[given hexadecimal]
1010	0010	1011	.	0001	1101	[from BCH]

Table 2. Binary Coded Decimal (BCD), Binary Coded Octal (BCO) and Binary Coded Hexadecimal (BCH)

Decimal numbers	Binary Coded Decimal (BCD)	Octal numbers	Binary Coded Octal (BCO)	Hexadecimal Numbers	Binary Coded Hexa-decimal (BCH)
0	0 0 0 0	**0**	0 0 0	**0**	0 0 0 0
1	0 0 0 1	**1**	0 0 1	**1**	0 0 0 1
2	0 0 1 0	**2**	0 1 0	**2**	0 0 1 0
3	0 0 1 1	**3**	0 1 1	**3**	0 0 1 1
4	0 1 0 0	**4**	1 0 0	**4**	0 1 0 0
5	0 1 0 1	**5**	1 0 1	**5**	0 1 0 1
6	0 1 1 0	**6**	1 1 0	**6**	0 1 1 0
7	0 1 1 1	**7**	1 1 1	**7**	0 1 1 1
8	1 0 0 0	**1 0**	0 0 1 0 0 0	**8**	1 0 0 0
9	1 0 0 1	**1 1**	0 0 1 0 0 1	**9**	1 0 0 1
1 0	0 0 0 1 0 0 0 0	**1 2**	0 0 1 0 1 0	**A**	1 0 1 0
1 1	0 0 0 1 0 0 0 1	**1 3**	0 0 1 0 1 1	**B**	1 0 1 1
1 2	0 0 0 1 0 0 1 0	**1 4**	0 0 1 1 0 0	**C**	1 1 0 0
1 3	0 0 0 1 0 0 1 1	**1 5**	0 0 1 1 0 1	**D**	1 1 0 1
1 4	0 0 0 1 0 1 0 0	**1 6**	0 0 1 1 1 0	**E**	1 1 1 0
1 5	0 0 0 1 0 1 0 1	**1 7**	0 0 1 1 1 1	**F**	1 1 1 1
1 6	0 0 0 1 0 1 1 0	**2 0**	0 1 0 0 0 0	**1 0**	0 0 0 1 0 0 0 0

Binary ⟹ Octal $(\ )_2 \Rightarrow (\ )_8$

- Group into set of 3-bits from radix point
  - towards left for integer part
  - towards right for fractional part
- Convert each group to its equivalent octal using BCO. See table 1
- Add zeros to complete last groups as needed.

Binary ⟹ Hexadecimal $(\ )_2 \Rightarrow (\ )_{16}$

- Group into set of 4-bits from radix point
  - towards left for integer part
  - towards right for fractional part
- Convert each group to its equivalent hexadecimal using BCH. See table 1
- Add zeros to complete last groups as needed.

Example:

$(1000111011.01101)_2 \Rightarrow (?)_2 \Rightarrow (1073.32)_8$

1000111011.01101 [given binary number]

001, 000, 111, 011 **.** 011, 01**0** [grouping 3-bits]

1 0 7 3 . 3 2 [equivalent octal]

$(1000111011.01101)_2 \Rightarrow (?)_2 \Rightarrow (23B.68)_{16}$

1000111011.01101 [given binary number]

0010, 0011, 1011 **.** 0110, 1**000** [grouping 4-bits]

2 3 B . 6 8 [equivalent hexadecimal]

Octal ⟹ Hexadecimal $(\ )_8 \Rightarrow (\ )_{16}$

- Replace each *octal digit* by its **3-bit** BCO equivalent (as in the table 1) both for integer and fractional part.
- Group into set of 4-bits from radix point
  - towards left for integer part
  - towards right for fractional part
- Convert each group to its equivalent *hexadecimal* using BCH. See table 2

Hexadecimal ⟹ Octal $(\ )_{16} \Rightarrow (\ )_8$

- Replace each *hexadecimal digit* by its **4-bit** BCH equivalent (as in the table 1) both for integer and fractional part.
- Group into set of 3-bits from radix point
  - towards left for integer part
  - towards right for fractional part
- Convert each group to its equivalent *octal* using BCO. See table 2

Example:

$(2\ 3\ 5.6\ 4)_8 \Rightarrow (BCO) \Rightarrow (?)_{16} \Rightarrow (9D.D)_{16}$

2	**3**	**5**	**.**	**6**	**4**	[given octal number]
010	011	101	.	110	100	[3-bit BCO]
0, 1001,	1101		.	1101,	0000	[grouping4-bits]
9	**D**		**.**	**D**	**0**	[equivalent hexadecimal]

$(2\ E\ 5.6\ B)_{16} \Rightarrow (BCH) \Rightarrow (?)_8 \Rightarrow (1345.326)_8$

2	**E**	**5**	**.**	**6**	**B**	[given hexadecimal]
0010	1110	0101	.	0110	1011	[3-bit BCO]
001, 011,	100,	101	.	011, 010,	110	[grouping4-bits]
1	**3**	**4**	**5**	**.**	**3**	**2** **6** [equivalent octal]

5.4 Boolean algebra

Boolean algebra was invented by George Boole in 1854. After 70 years (in 1924) Claude Shannon, who recognized and worked out the relevance of Boole's symbolic logic for the field of engineering. Boolean algebra is a mathematical system for the manipulation of variables that can be used to analyze and simplify the digital (logic) circuits whose outcome would be either 0 or1, since it uses only the binary numbers 0 and 1. With regard to the digital logic, a set of rules are used to describe circuits whose state can either **1** (ON) or **0** be (OFF).

- **Boolean Constants -** these are 0 (false) and 1 (true).
- **Boolean Variables -** variables used to represent input-output of digital circuits that can only take the vales 0 or 1. Example: A, B, X, Y etc.
- **Boolean Functions -** each of the logic functions (such as AND, OR and NOT) are represented by symbols.

Consequently, the "Laws" of Boolean algebra often differ from the "Laws" of real-number algebra. A number of rules can be derived from AND, OR and NOT relations called as basic *Boolean postulates*. Boolean postulates and laws are defined in the table 2 and table 3, respectively.

Table 2: Boolean Postulates

Postulate	Relation	
1	$0 \cdot 0 = 0$	**NOTE**: Every law has two forms: AND form and OR form as shown in the following table 2. This is known as *duality*. These are obtained by changing every AND (·) to OR (+), every OR (+) to AND (·) and all 1's to 0's and vice-versa. It is conventional to write A·B is as AB by dropping AND (·) symbol.
2	$1 + 1 = 1$	
3	$0 + 0 = 0$	
4	$1 \cdot 1 = 1$	
5	$1 \cdot 0 = 0$ or $0 \cdot 1 = 0$	
6	$1 + 0 = 1$ or $0 + 1 = 1$	

Table 3: Boolean Laws

Boolean Laws	AND form	OR form
Commutative	$A\,B = B\,A$	$A + B = B + A$
Associative	$A\,(B\,C) = (A\,B)\,C$	$A + (B + C) = (A + B) + C$
Identity	$A \cdot 1 = A$	$A + 0 = A$
Distributive	$A\,(B + C) = A\,B + A\,C$	$A + B\,C = (A + B)\,(A + C))$
one/zero	$A \cdot 0 = 0$	$A + 1 = 1$
Idempotency	$A\,A = A$	$A + A = A$
Inverse	$A \cdot \overline{A} = 0$	$A + \overline{A} = 1$
De Morgan's law	$\overline{A \cdot B} = \overline{A} + \overline{B}$	$\overline{A + B} = \overline{A} \cdot \overline{B}$
Double negation	$\overline{\overline{A}} = A$	

5.9 De-Morgan's theorem

De-Morgan's theorem is one of the Boolean law which is useful in the implementation of the basic gate operations with alternative gates NAND and NOR, which are readily available in IC form.

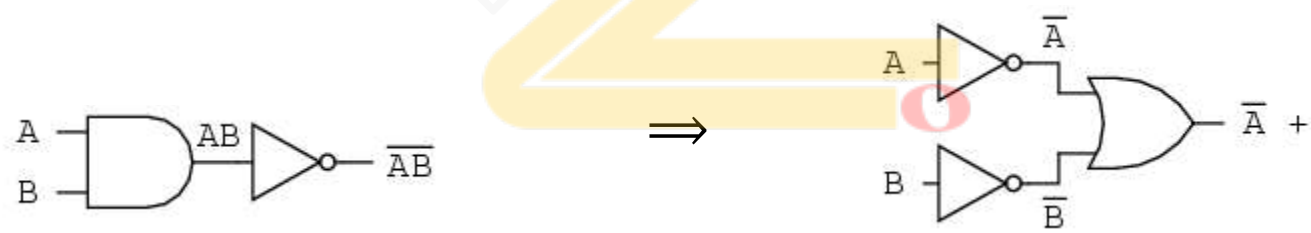
The two De-Morgan's laws can be implemented in Boolean algebra as in the following steps:

(1) If (+) is there then replace it with (·) and if (·) is there then replace it with (+).

(2) Compliment of each of the term is to be found.

De-Morgan's theorem – 1: $\overline{A \cdot B} = \overline{A} + \overline{B}$

Statement: The compliment of the product of variables is equal to the sum of the compliment of each variable.



AND+NOT = NAND is equivalent to Bubbled OR

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

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

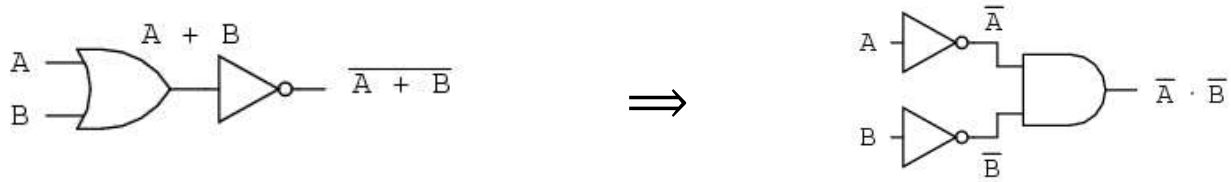
$\leftrightarrow$

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

From the above truth tables LHS is equal to RHS, hence it is proved.

De-Morgan's theorem – 2: $\overline{A+B}=\overline{A}\cdot\overline{B}$

Statement: The compliment of the sum of variables is equal to the product of the compliment of each variable.



OR+NOT = NOR is equivalent to Bubbled AND

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

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

↔

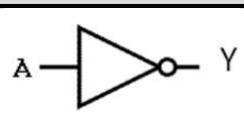
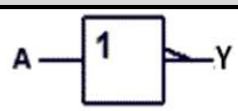
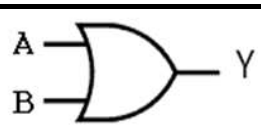
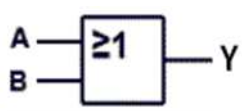
A	B	$\overline{A}$	$\overline{B}$	$\overline{A}+\overline{B}$
0	0	1	1	1
0	1	1	0	1
1	0	0	1	1
1	1	0	0	0

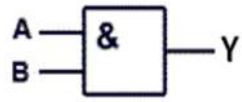
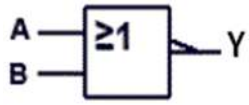
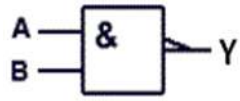
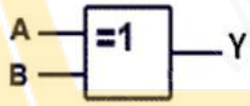
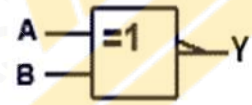
From the above truth tables LHS is equal to RHS, hence it is proved.

5.5 Basic Logic Gates

Logic gate is an electronic circuit having one or more than one input and only one output. Logic gates require a power supply. The inputs of any gate are driven by voltage levels **0** V and **5** V representing logic **0** and logic **1**, respectively. Boolean functions may be practically implemented by using electronic gates. The output of a gate provides two nominal values of voltages either **0** V or **5** V representing logic **0** or logic **1**, respectively. Truth tables are used to show the function of logic gate with input output relationship. The gates are AND, OR, NOT (basic gates), NAND, NOR, EXOR and EXNOR (derived gates). Digital systems (FFs, ALU etc) are said to be constructed by using these logic gates.

Table 4 Traditional & IEEE symbol, Boolean Expression and Truth Table of Logic Gates

Logic Gate	Traditional symbol	IEEE symbol	Boolean Expression	Truth Table Input / Output
1. NOT			$Y=\overline{A}$	A / $Y=\overline{A}$: 0 / 1; 1 / 0
-Only ONE input and one output		- Output is inversion of input		
2. OR			Y= A + B	A B / Y: 0 0 / 0; 0 1 / 1; 1 0 / 1; 1 1 / 1
➢ Two or more inputs (Logic addition) and one output ➢ Output is logic LOW if both inputs are LOW, Otherwise output is logic HIGH				

3. AND			$Y = A \cdot B$	
➢ Two or more inputs (Logic multiplication) and one output ➢ Output is logic HIGH if both inputs are HIGH, Otherwise output is logic LOW				
4. NOR			$Y = \overline{A + B}$	
➢ Two or more inputs (Logic multiplication + inversion) and one output ➢ Output is logic HIGH if both inputs are LOW, Otherwise output is logic HIGH				
5. NAND			$Y = \overline{A \cdot B}$	
➢ Two or more inputs (Logic multiplication + inversion) and one output Output is logic HIGH if both inputs are LOW, Otherwise output is logic HIGH				
6. EX-OR Exclusive OR			$Y = A \oplus B$	
➢ Output is logic HIGH if both inputs are different ➢ Otherwise, output is logic LOW if both inputs are same				
7. EX-NOR Exclusive NOR			$Y = \overline{A \oplus B}$	
➢ Output is logic HIGH if both inputs are same ➢ Otherwise, output is logic LOW if both inputs are different				

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

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

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

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

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

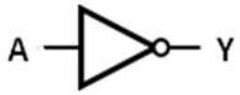
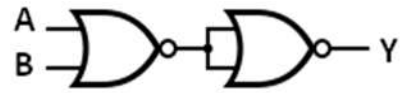
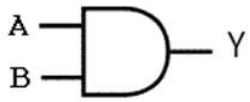
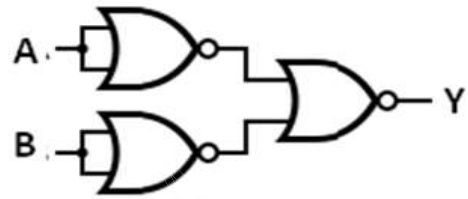
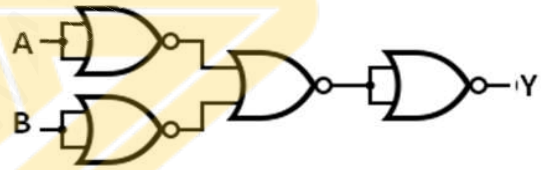
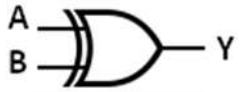
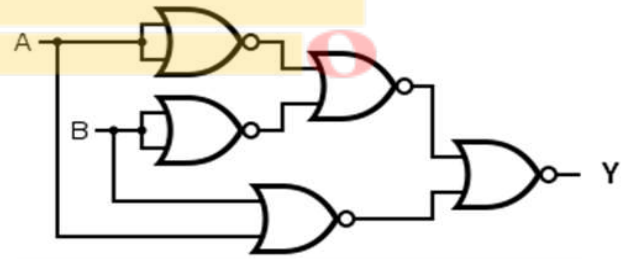
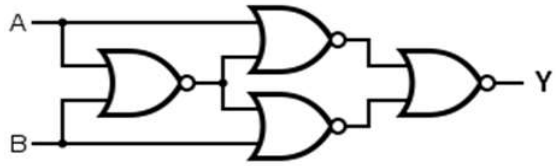
5.5.1 Universal Logic Gates

The NAND and NOR gates are universal gates. Operations of all logic gates can be performed using only these gates; hence they called as universal gates. In practice, NAND and NOR gates are economical and easier to fabricate and they are the basic gates used in all IC digital logic families.

Implementation of logic gates: using only NOR Gates

It can be proved that any Boolean function can be implemented and realized (the operations of all logic gates can be verified) using only NOR gates. It is illustrated in the table 4.

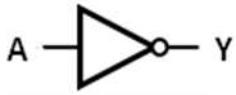
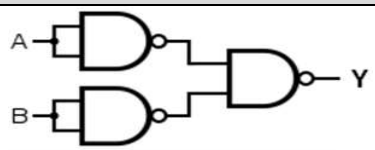
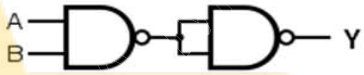
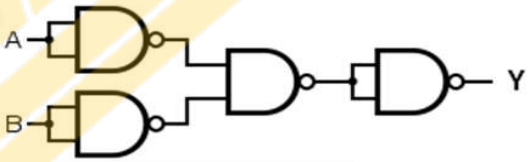
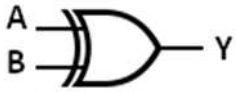
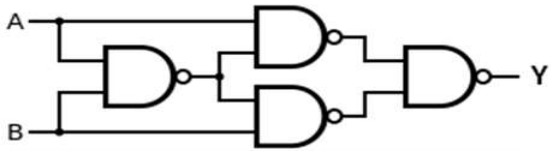
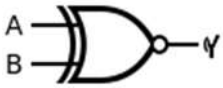
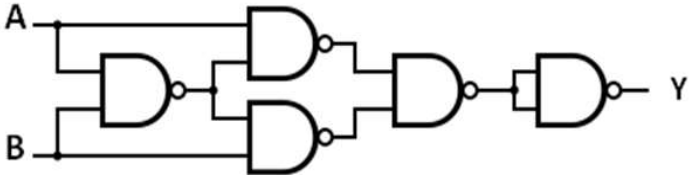
Table 5: Implementation of logic gates using only NOR Gates

Desired gate		Boolean expression	Implementation using only NOR gates
1. NOT		$Y = \overline{A}$	
NOT gate can be implemented by tying the two inputs of the NOR gate together.			
2. OR		$Y = A + B$	
OR gate can be implemented by simply one NOR gate followed by a second whose inputs are joined.			
3. AND		$Y = A \cdot B$	
AND gate is implemented by inverting the inputs to a 3rd NOR gate.			
4. NAND		$Y = \overline{A \cdot B}$	
NAND gate is implemented by using an AND gate (previous discussion) in series with a NOR gate.			
5. XOR		$Y = A \oplus B$	
XOR gate is implemented by connecting the output of 3 NOR gates (connected as an AND gate) and the output of a NOR gate to the respective inputs of a NOR gate. That is Y= (A AND B) NOR (A NOR B).			
6. XNOR		$Y = \overline{A \oplus B}$	
XNOR gate can be constructed from four NOR gates implementing the expression Y = (A NOR (A NOR B)) NOR (B NOR (A NOR B)).			

Implementation of logic gates: Using only NAND Gates

It can be proved that any Boolean function can be implemented and realized (the operations of all logic gates can be verified) using only NAND gates. It is illustrated in the table 6.

Table 6: Implementation of logic gates using only NAND Gates

Desired gate		Boolean expression	Implementation using only NAND gates
1. NOT		$Y = \overline{A}$	
NOT gate can be implemented by tying the two inputs of the NAND gate together.			
2. OR		$Y = A + B$	
OR gate is implemented by inverting the inputs to a 3rd NAND gate.			
3. AND		$Y = A \cdot B$	
AND gate can be implemented by simply one NAND gate followed by a second whose inputs are joined together.			
4.NOR		$Y = \overline{A + B}$	
NOR gate is implemented by using an OR gate (previous discussion) in series with a NAND gate.			
5. XOR		$Y = A \oplus B$	
XOR gate is implemented similarly to OR gate, except with an additional NAND gate inserted such that if both inputs are high, the inputs to the final NAND gate will also be high, and the output will be low.			
6. XNOR		$Y = \overline{A \oplus B}$	
An XNOR gate is simply an XOR gate with an inverted output.			

5.5.2 Simplification of Boolean Expressions

Boolean Expressions represent Complex combinational logic circuits. The simpler the Boolean expression, the smaller the circuit that will result. Reduction of a logic circuit means the same logic function with fewer gates and/or inputs. Simpler circuits are cheaper to build, consume less power, and run faster than complex circuits. With this in mind, it is desired always to reduce Boolean functions to their simplest form.

Steps to reduce a logic circuit

- Write the Boolean Equation/expression for the logic function. Apply as appropriate rules and laws as possible in order to decrease the number of terms and variables in the expression.

1. **Solve Bracketed quantities -** Inside any parentheses look for more parentheses
2. **NOTs, ANDs, ORs**
3. If an expression has a bar over it, perform the operations inside the expression first and then invert the result

- Draw the logic diagram for the reduced Boolean Expression using basic logic gates.

Example 1: $A\,B\,C + A\,\overline{B}\,C + A\,B\,\overline{C} + \overline{A}B\,C$

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

$= A\,C\,[B + \overline{B}] + A\,B\,\overline{C} + \overline{A}\,B\,C$ $\qquad \because B + \overline{B} = 1$

$= A\,C\,[1] + A\,B\,\overline{C} + \overline{A}\,B\,C$

$= C\,[A + \overline{A}\,B] + A\,B\,\overline{C}$ $\qquad \because A + \overline{A}B = A + B$

$= C\,[A + B] + A\,B\,\overline{C}$

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

$= A\,[C + \overline{C}\,B] + B\,C$ $\qquad \because C + \overline{C}\,B = C + B$

$= A\,[C + B] + B\,C$

$Y = A\,C + A\,B + B\,C$

Example 2: $\overline{(A + B)}\ (\overline{A} + \overline{C})\ (\overline{B} + C)$

$Y = \overline{(A + B)}\ (\overline{A} + \overline{C})\ (\overline{B} + C)$ $\qquad \because \overline{(A + B)} = \overline{A} \cdot \overline{B}$

$= \overline{A} \cdot \overline{B}\,[\,\overline{A}\,\overline{B} + \overline{A}\,C + \overline{B}\,\overline{C}\,]$

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

$= \overline{A}\,\overline{B}\,[\,1 + \overline{C}\,] + \overline{A}\,\overline{B}\,\overline{C}$ $\qquad \because 1 + \overline{C} = 1$

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

$= \overline{A}\,\overline{B}\,[1 + \overline{C}\,]$

$Y = \overline{A}\,\overline{B}$

Example 3: $A + (\overline{B}\,C)\,[\,\overline{A} + B + \overline{C}\,]\,(A + \overline{B})$

$Y = A + \overline{B}\,C\,[\,A\,\overline{A} + A\,B + A\,\overline{C} + \overline{A}\,\overline{B} + B\,\overline{B} + \overline{B}\,\overline{C}\,]$ $\quad \because A\,\overline{A} = 0$ and $B\,\overline{B} = 0$

$= A + \overline{B}\,C\,[\,A\,B + A\,\overline{C} + \overline{A}\,\overline{B} + \overline{B}\,\overline{C}\,]$

$= A + A\,B \cdot \overline{B}\,C + \overline{A}\,\overline{B} \cdot \overline{B}\,C + A\,\overline{B} \cdot C\,\overline{C} + \overline{B}\,\overline{B} \cdot C\,\overline{C}$ $\quad \because B\,\overline{B} = 0$ and $C\,\overline{C} = 0$

$= A + \overline{A}\,\overline{B}\,C$ $\quad \because A + \overline{A}B = A + B$

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

5.6 Arithmetic Circuits: Half and Full adder

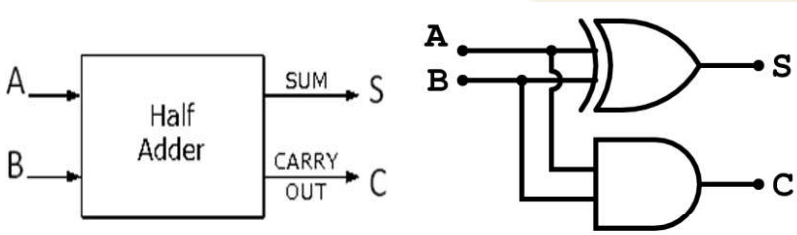
An arithmetic circuit is a digital logic circuit that performs addition of numbers. In computers and other types of processors, adders are used to calculate addresses, addition and multiplication operations and table indices in the ALU.

Adders are classified into two types:
(1) Half Adder and
(2) Full Adder

5.6.1. Half Adder

The half adder circuit is a digital adder circuit capable of adding only two binary bits A and B. It has two inputs: A and B and adds these two input bits at a time and produce a carry(C) and sum(S) This process follows the binary addition rules. It can be constructed using one AND and one XOR gate.

- Its block diagram, logic circuit and truth table is shown in the fig. 5.3.



Inputs		Outputs	
A	B	Sum(S)	Carry(C)
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Fig. 5.3 Block diagram, logic circuit and truth table of an Half Adder

Truth table gives input-output relationship, from which we observe that,

(i) The output Sum (S) follows XOR operation between A and B inputs

$S = \overline{A} \bullet B + A \bullet \overline{B} =$ $\qquad$ **C = A· B**

(ii) The output Carry (C) follows AND operation between A and B inputs

Hence, half adder circuit is a combination of XOR and AND operations. Implementation of Half Adder circuit using basic gates is shown in the fig.5.4.

Limitation: Half Adder circuit cannot receive an input carry bit.

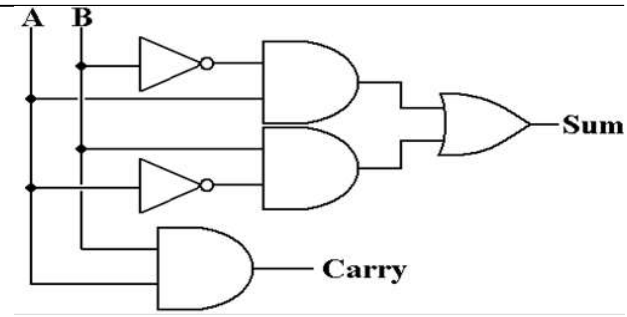


Fig.5.4 Implementation of half adder circuit using basic gates

5.6.2. Full Adder

The full-adder circuit is a digital adder circuit capable of adding three bit binary numbers (2 bits: A and B and one bit carry in C_{in}). This circuit consists of three inputs (A, B and C_{in}) and two outputs (S and C_{out}). It has three inputs (A, B and C_{in}) and adds these input bits at a time and produce a carry(C) and sum(S) This process follows the binary addition rules. It can be constructed using two half adders where it consists 2 ANDs, 2 XORs, and 1 OR. Block diagram, logic circuit and truth table is shown in the fig. 5.5.

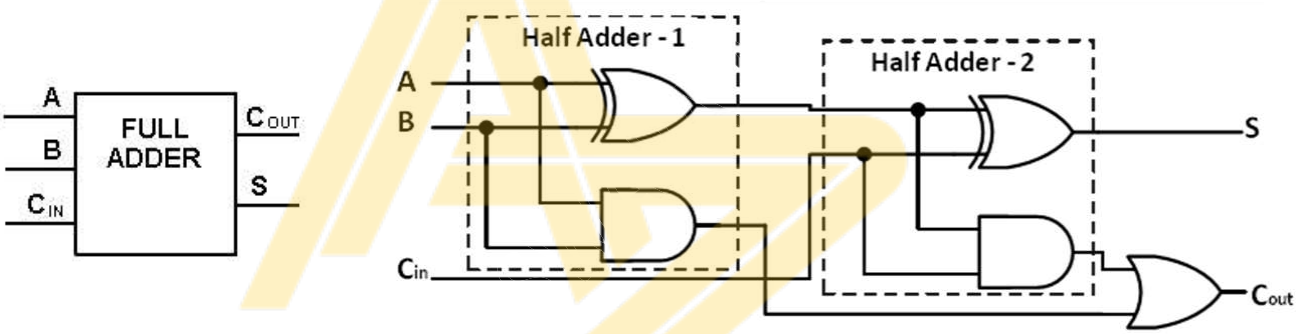


Fig. 5.5 (a) Block diagram (b) Circuit diagram of Full Adder

Inputs			Outputs	
A	**B**	**C_{in}**	**Sum (S)**	**Carry (C_{out})**
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

From the truth table it is observes that,

(i) Sum (S) output is equal to 1, when only one input is equal to1 or when all three inputs are equal to1. For this Boolean expression can be written as

$$S = Cin\overline{A}\overline{B} + \overline{Cin}\overline{A}B + CinAB + \overline{Cin}A\overline{B}$$
$$= Cin(\overline{A}\overline{B} + AB) + \overline{Cin}(\overline{A}B + A\overline{B})$$
$$= Cin(\overline{A \oplus B}) + \overline{Cin}(A \oplus B)$$
$$\mathbf{S} = Cin \oplus A \oplus B$$

(ii) Output has a carry1, if two or three inputs are equal to1. For this Boolean expression can be written as

$$Cout = AB + Cin(\overline{A}B + A\overline{B}) = AB + Cin(A \oplus B)$$

5.7 Multiplexer

Multiplexer is a device that has multiple inputs and a single line output. The data select lines determine which input is connected to the output. It is also called a data selector shown in the fig.5.6 and fig. 5.7.

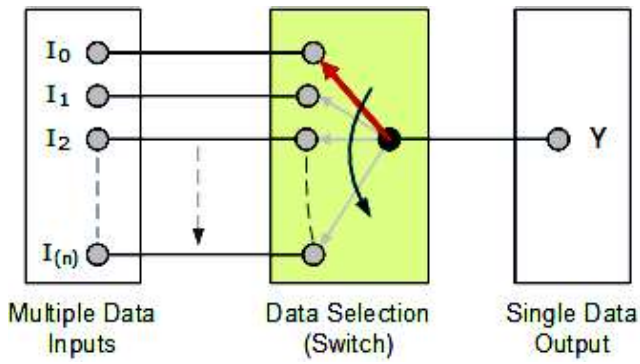
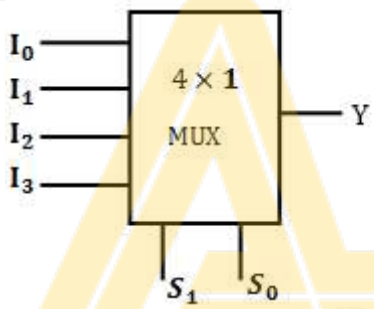


Fig. 5.6 Block diagram of Multiplexer



Truth table

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

Fig. 5.7 Logic diagram of 4:1 Multiplexer

Multiplexers are capable of handling both analog and digital applications. In analog applications, multiplexers are made up of relays and transistor switches, whereas in digital applications, the multiplexers are built from standard logic gates.

Multiplexer Types

Multiplexers are classified into four types:

- 2-1 multiplexer (1select line)
- 4-1 multiplexer (2 select lines)
- 8-1 multiplexer(3 select lines)
- 16-1 multiplexer (4 select lines)

Applications of Multiplexers

1. **Communication System -** increases the transmission of data (audio, video) from different channels through single lines or cables.
2. **Computer Memory -** to maintain a huge amount of memory in the computers, and also to reduce the number of copper lines required to connect the memory to other parts of the computer.
3. **Telephone Network -** multiple audio signals are integrated on a single line of transmission with the help of a multiplexer.
4. **Transmission from the Computer System of a Satellite**- to transmit the data signals from the computer system of a spacecraft or a satellite to the ground system by using a GSM satellite.

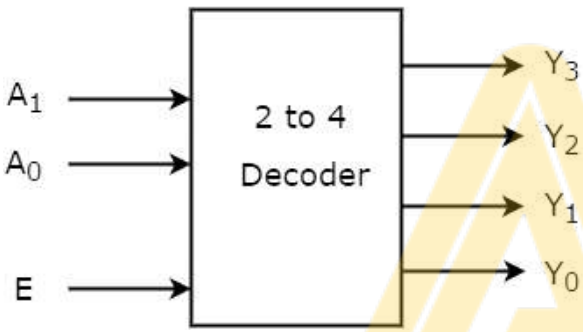
5.8 Decoder

A decoder is a combinational logic circuit that takes multiple inputs and gives multiple outputs. A decoder circuit takes binary data of 'n' inputs into '2^n' unique output. In addition to input pins, it has an enable pin. This enable pin makes the circuit active / inactive. It is shown in the fig.5.8.

Example: 2-to-4 line binary decoder:

It consists of an array of four AND gates. Thumb rule with decoders is that, if the number of inputs is considered as n (here n = 2) then the number of output will always be equal to $\mathbf{2^n}$ ($2^2 = 4$). The Decoder has 2 input lines and 4 output lines; hence this type of Decoder is called as 2:4 Decoders.

2:4 decoder has two inputs A_1 and A_0 and four outputs Y_3, Y_2, Y_1 & Y_0. Its block diagram and truth table is shown in the fig. One of these four outputs will be logic '1' for each combination of inputs when enable, E is logic '1'.



Truth Table

Enable	Inputs		Outputs			
E	A_1	A_0	Y_3	Y_2	Y_1	Y_0
0	x	x	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

Fig.5.8 Decoder block diagram and truth table

From truth table, we can write the **Boolean functions** for each output as

$Y_3 = E(A_1A_0)$ $\quad$ $Y_2 = E(A_1\overline{A_0})$ $\quad$ $Y_1 = E(\overline{A_1}A_0)$ $\quad$ $Y_0 = E(\bar{A}_1\bar{A}_0)$

Each output is having one product term. We can implement these four product terms by using four AND gates having three inputs (A_1, A_0 and E) each and two NOTs. The circuit diagram of 2 to 4 decoder is shown in the fig.5.9.

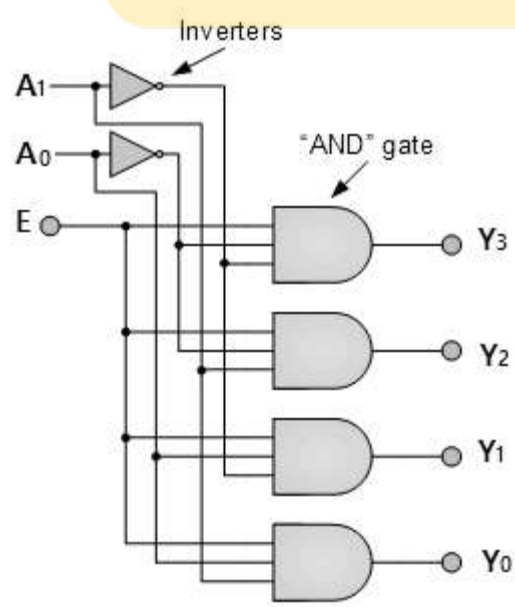


Fig.5.9 circuit diagram of 2 : 4 decoder

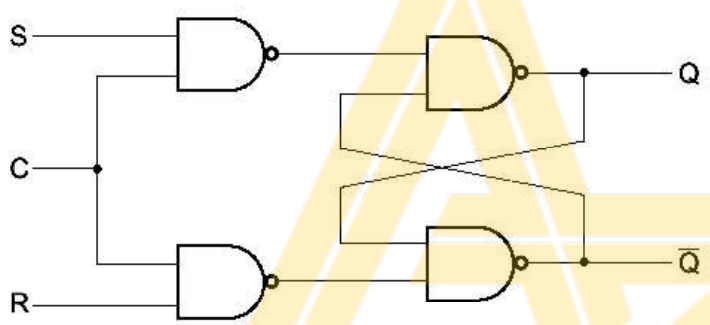
Applications of Decoder

- In analog to digital conversion in analog decoders.
- Data multiplexing,
- Memory address decoding,
- 7 segment display.

5.9 SR Flip-Flop and JK Flip Flop

5.9.1 SR flip-flop (with clock)

SR flip-flop is a one-bit memory bi-stable device that has two inputs, one is labelled **S**, will SET the device (meaning the output Q = 1), and other is labelled **R,** will RESET the device (meaning the output Q = 0). Then the SR description stands for "Set-Reset". FFs are made from latches and FFs respond only on specific times. Logic diagram, truth table and timing diagram is shown in the fig.5.10.



(a) Logic diagram

C	S	R	Next state of Q
0	X	X	No change
1	0	0	No change
1	0	1	Q = 0; Reset state
1	1	0	Q = 1; Set state
1	1	1	Undefined

(b) Function table

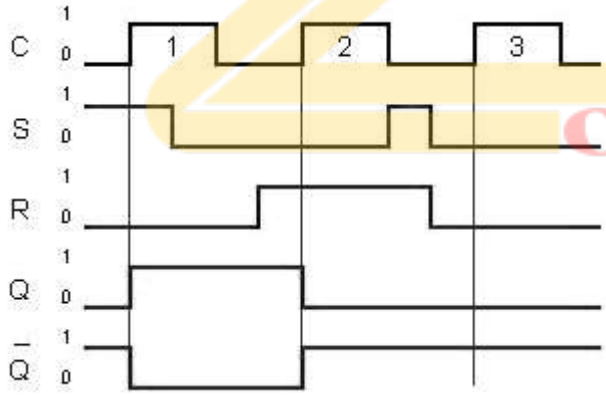


Fig.5.10 (c) Timing diagram

Working: The function of SR flip flop is described in the truth table and analyzed as in the timing diagram. The table shows *four* useful modes of operation.

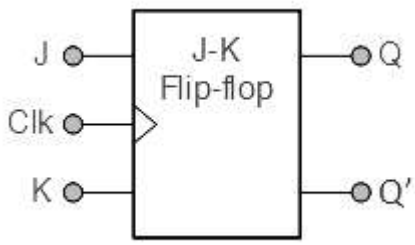
Whenever the clock C is LOW, the inputs S and R are never affect the output**.** The clock has to be HIGH for the inputs to get active.

1. **Hold state:** When S = R = 0 and clock = 1; output of SR FF, Q = 0; so output Q remains as previous output , therefore, FF is in the hold mode. In the hold mode, the data inputs have no effect on the outputs. The outputs "hold" the last data present.

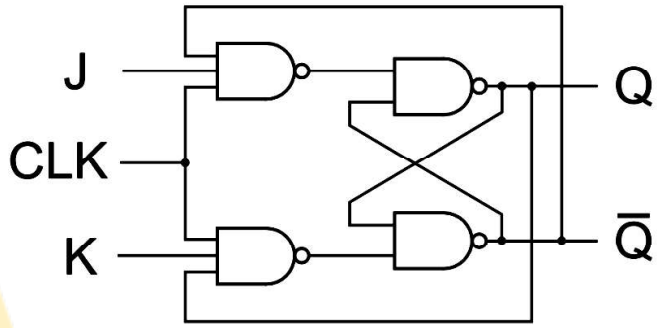
2. **Reset state:** When S = 0, R =1 and clock = 1; output of SR FF, Q = 0; which RESETs the flip flop.
3. **Set State:** When S =1, R = 0 and clock = 1; output of SR FF, Q = 1; which SETs the flip flop.
4. **Invalid state:** When S =1, R = 1 and clock = 1; output of SR FF, Q = $\overline{Q}$ = 1, so it is invalid.

5.9.2. JK flip-flop

To overcome the *invalid state* of SR flip flop an extra feedback from the output to input is given, then such FF is called JK flip flop. The logic symbol, circuit, truth table and timing diagram of the JK flip-flop is illustrated in Fig. 5.11.



(a) **Logic Symbol**



(b) circuit using NAND gates

J	K	CLK	OUTPUT Q
0	0	↑	No change
0	1	↑	Reset (Q = 0)
1	0	↑	Set (Q= 1)
1	1	↑	Toggle

(c) Truth Table

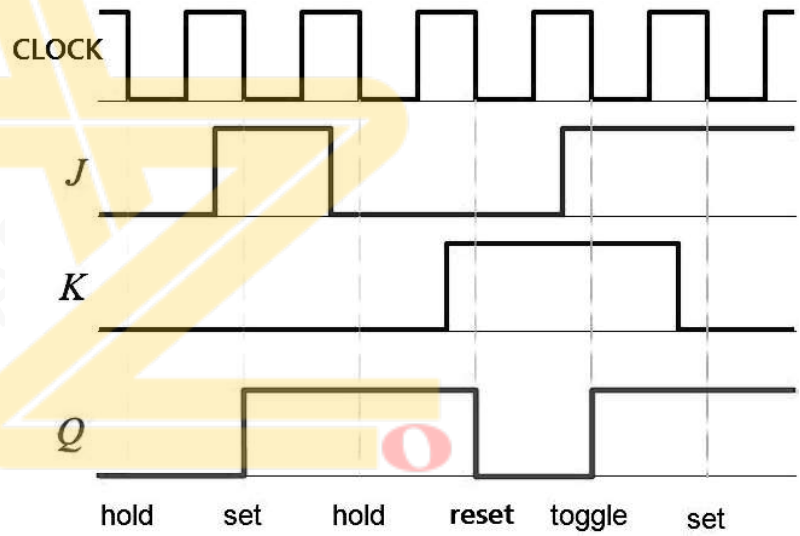


Fig.5.11 : JK flip-flop. **(d)Timing diagram**

The inputs J and K are the data inputs. JK Flip Flop is same as RS flip-flop with the same SET and RESET input. The difference is that the JK Flip Flop does not have the *invalid states* of the RS FF (when S = R = 1).

This table shows four useful modes of operation.

1. **Hold state:** When J = K = 0 and clk = 1; output remains in previous state, so the flip-flop is in the hold mode. In the hold mode, the data inputs have no effect on the outputs. The outputs "hold" the last data present.
2. **Reset state:** When J = 0 K =1 and clk = 1; output of JK FF, Q = 0; which RESETs the flip flop.
3. **Set state:** When J = 1 K = 0 and clk = 1; output of JK FF, Q = 1; which SETs the flip flop.
4. **Toggle state:** When J = 1 K = 1 and clk = 1; the output Q, turns *off-on*. This *off-on* action is called toggling. Each clock pulse toggles the outputs to switch to their opposite states.

Race around condition in JK flip-flop

Toggle: switching of state either from 0 to 1 or 1 to 0, which makes the output of the flip-flop unstable or uncertain.

Within a single clock period if J = K = 1 and clock = 1, output changes its state (toggle) more than one time. This is called ***race around condition.*** This problem can be avoided by introduced the concept of Master Slave JK flip flop.

5.10 Shift register

Register: A set of N flip-flops is called register. Each flip-flop stores one bit (1 or 0).

Two basic functions: data storage and data movement.

Shift Register: is a register can be used for the storage or the transfer of binary data. This sequential circuit receives the data from its inputs and then "shifts" it to its output for every clock cycle, hence the name *shift register***.** Shift Register is made of the number of individual Flip Flops. For example, an 4-bit wide shift register is constructed from four individual FFs, as shown in the fig.5.12. They used inside calculators or computers to store data. Generally, shift registers operate in one of *four different modes* with the basic movement of data through a shift register being:

1. **Serial-in to Parallel-out** (SIPO) - the register is loaded with serial data, one bit at a time, with the stored data being available at the output in parallel form.
2. **Serial-in to Serial-out** (SISO) - the data is shifted serially "IN" and "OUT" of the register, one bit at a time in either a left or right direction under clock control.
3. **Parallel-in to Serial-out** (PISO) - the parallel data is loaded into the register simultaneously and is shifted out of the register serially one bit at a time under clock control.
4. **Parallel-in to Parallel-out** (PIPO) - the parallel data is loaded simultaneously into the register, and transferred together to their respective outputs by the same clock pulse.

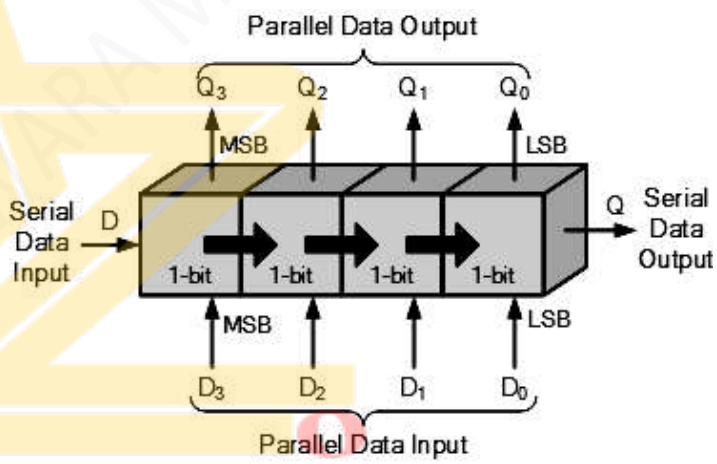


Fig.5.12 Four-bit wide shift register

5.10.1 4-bit Serial-in to Serial-out (SISO)

A basic four-bit SISO shift register can be constructed using four D flip-flops, as shown in fig.5.13. The operation of circuit is as follows. Assuming for the 4-bit shift register, we need to shift the data 1101.

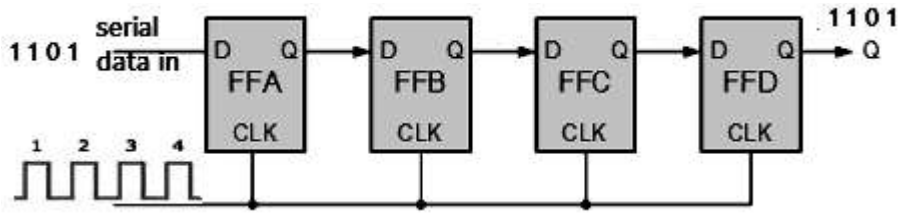


Fig.5.13 **Four-bit Serial-in to Serial-out shift register**

Operational Steps:

1. The register is first cleared, forcing all four outputs to zero.
2. The input data (1101) is then applied sequentially to the D input of the first flip-flop on the left (FFA).

Clock cycle	FF0	FF1	FF2	FF3
First	1	0	0	0
Second	0	1	0	0
Third	1	0	1	0
Fourth	1	1	0	1

3. During each clock pulse, one bit is transmitted from left to right.
4. During the first clock cycle as we apply the data (1101) serially, similarly for four clock cycles the outputs of the each flip flop is displayed in the above functional table.

5.11 Three bit Ripple Counter

A ripple counter is an asynchronous counter where only the first FF is clocked by an external clock. All the subsequent FFs are clocked by the output of the preceding FF. Asynchronous counters are also called ripple-counters, because of the way the clock pulse ripples it way through the flip-flops. Each stage acts as a divide-by-2 counter on the previous stage's signal. The Q out of each stage acts as both an output bit and as the clock signal for the next stage. It can chain as many ripple counters together as need.

A three bit ripple counter shown in the fig.5.14, will count $2^3 = 8$ numbers [count (0-7) → minimum 0 (000) and maximum 7 (111)] and an n-bit ripple counter will count 2^n numbers.

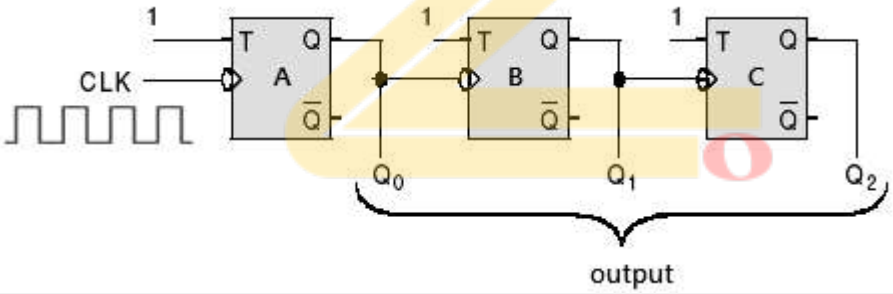


Fig.5.14 Three bit Ripple Counter using T- FF

Count	Q_2	Q_1	Q_0
0	0	0	0
1	0	0	1
2	0	1	0
3	1	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

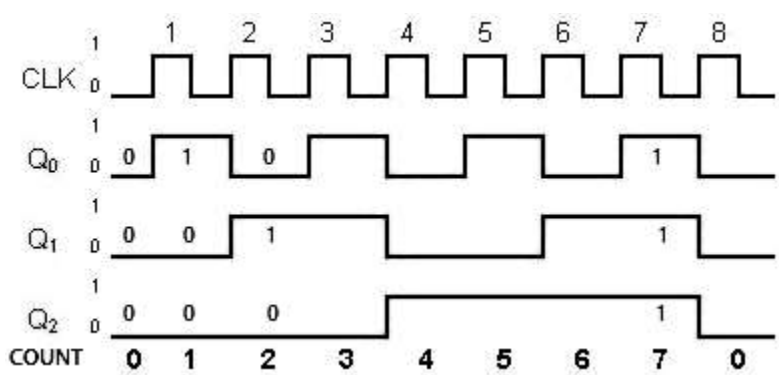


Fig.5.15 Function table and timing diagram for 3- bit Ripple Counter

The clock inputs of the three FFs are connected in cascade. The T input of each FF is connected to logic 1, which means that the state of the FF will toggle at each edge of its clock. FFA is connected to the clock line and other two FFB and FFC have driven by the Q output of the preceding FF. Therefore, they toggle their state whenever the preceding FF changes its state from Q = 1 to Q = 0. So as to take the output from Q_0, Q_1 and Q_2. Then we get the count sequence with different counter states as mention in the fig.5.15.

5.12 Basic Communication system

Communication is the process of establishing connection or link between two points for information exchange. The block diagram of basic communication system is shown in the fig.5.16.

Information source: The various massages are in the form of words, picture, code, symbol, sound signal, video etc. However, out of these one message is selected & conveyed or communicated. It is used to produce required message which has to be transmitted.

Transducer: is a device which converts one form of signal to another form. The message produced by source is not electrical; hence an input transducer is used to convert to an electrical signal. Ex: μ-phone.

Transmitter: The main function of transmitter is to process the electric signal from different aspects. Modulation, amplification and filtering of massage are done to transmit signal over long distance.

Channel: This medium is either wired (twisted pair, co-axial, Optical Fiber Cable (OFC), etc) or wireless (free space, radio communication, etc), through which the message travel from the transmitter to receiver.

Noise: It is an unwanted signal which disturbs the message. They can interface with signal at any point in a communication system.

Receiver: The main function of receiver is to reproduce the massage signal in electrical form. It performs the functions like de-modulation, filtering, amplification, etc. Ex: loud- speaker

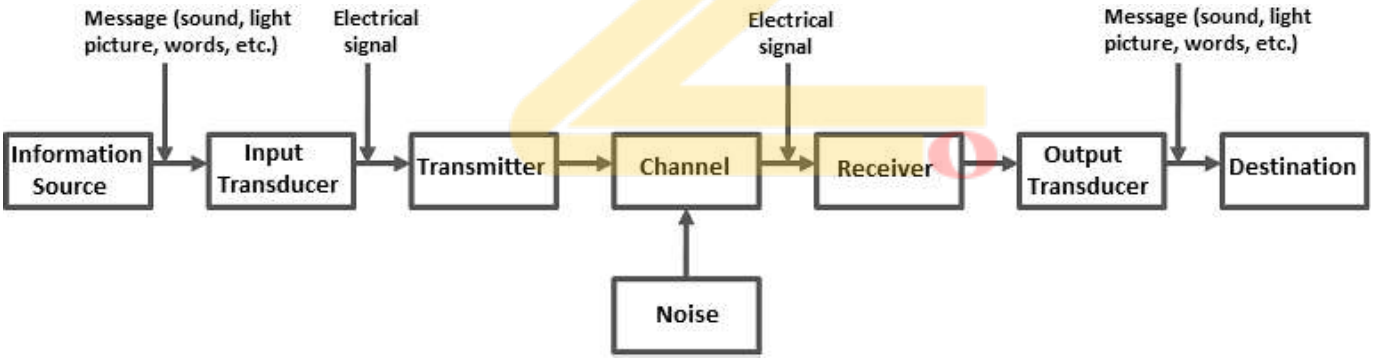


Fig.5.16 Basic Communication system

5.12 Principle of operations of Mobile phone

Radio Waves

Cell phones use radio waves to communicate. Radio waves transport digitized voice or data in the form of oscillating electric and magnetic fields, called the electromagnetic field (EMF). The rate of oscillation is called frequency. Radio waves carry the information and travel in air at the speed of light.

Cell phones transmit radio waves in all directions. The waves can be absorbed and reflected by surrounding objects before they reach the nearest cell tower. For example, when the phone is placed next to your head during a call, a significant portion (over half in many cases) of the emitted energy is absorbed into your head and body. In this event, much of the cell phone's EMF energy is wasted and no longer available for communication.

Antenna

Cell phones contain at least one radio antenna in order to transmit or receive radio signals. An antenna converts an electric signal to the radio wave (transmitter) and vice versa (receiver). Some cell phones use one antenna as the transmitter and receiver while others, such as the iPhone 5, have multiple transmitting or receiving antennas.

An antenna is a metallic element (such as copper) engineered to be a specific size and shape for transmitting and receiving specific frequencies of radio waves. While older generation cell phones have external or extractable antennas, modern cell phones contain more compact antennas inside the device thanks to advanced antenna technologies. It's important to understand that any metallic components in the device (such as the circuit board and the metal frame for the iPhone) can interact with the transmission antenna(s) and contribute to the pattern of the transmitted signal.

Many modern smart phones also contain more than one type of antenna. In addition to the cellular antenna, they may also have Wi-Fi, Bluetooth and/or GPS antennas.

Connectivity

As mentioned earlier, a cell phone is a two-way wireless communication device and needs both the inbound signal (reception) and the outbound signal (transmission) to work. The magnitude of the received signal from the cell tower is called the "signal strength", which is commonly indicated by the "bars" on your phone. The connectivity between a cell phone and its cellular network depends on both signals and is affected by many factors, such as the distance between the phone and the nearest cell tower, the number of impediments between them and the wireless technology (e.g. GSM vs. CDMA). A poor reception (fewer bars) normally indicates a long distance and/or much signal interruption between the cell phone and the cell tower.

In order to conserve battery life, a cell phone will vary the strength of its transmitted signal and use only the minimum necessary to communicate with the nearest cell tower. When your cell phone has poor connectivity, it transmits a stronger signal in order to connect to the tower, and as a result your battery drains faster. That's why good connectivity not only reduces dropped calls, but also saves battery life.

Exercise

1. What is a logic circuit? What is the need of Boolean Algebra in digital electronics?
2. Differentiate between analog and digital signals.
3. State and prove Demorgan's theorem for three variables.
4. Realize two input EXOR and EXNOR gates using

   (i) only NOR gates (ii) only NAND gates.
5. Design full adder circuit and implement it using two half adders.
6. Design a logic circuit using basic logic gates with three inputs ABC and output Y that goes low only when A = 1, and B and C are different.
7. Realize basic logic gates from NOR and NAND gates
8. Design a logic circuit, symbol and truth table of EXOR and EXNOR gates
9. Simplify the following Boolean Expressions:

   (i) $Y = XYZ + X\overline{Y}Z + XY\overline{Z}$

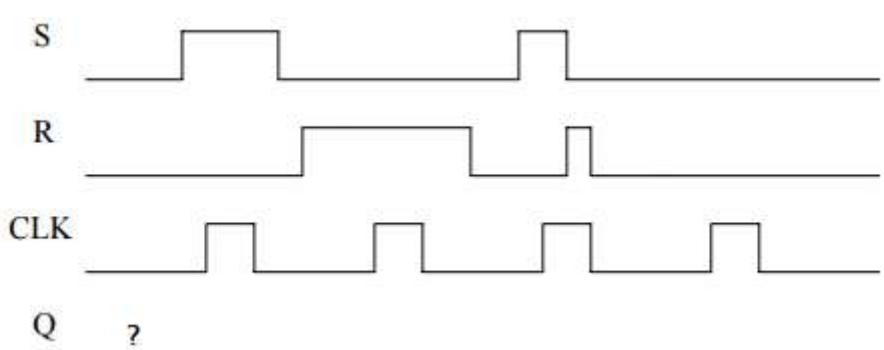
   (ii) $Y = \overline{X + Y(\overline{Z} + \overline{Y})}$

   (iii) $Y = (A\overline{B} + \overline{A}C)(BC + B\overline{C})ABC$

   (iv) $Y = (A + \overline{B} + \overline{C})(A + \overline{B} + C)$

   (v) $Y = ABC + A\overline{B}C$

   (vi) $Y = AB + \overline{AC} + A\overline{B}C(\overline{B} + C)$
10. With relevant diagrams explain the working principle of Multiplexer and Decoder.
11. Draw a diagram of a clocked (synchronous) R-S latch constructed using four NAND gates. Consider the following inputs to this latch and draw a graph of how the output Q varies as R, S and CLK vary. Assume that the latch is initially in the RESET state (Q = 0) and there is no delay in switching the latch.



12. Analyze the toggling method in J K FF.

Multiple Choice Questions

1. Convert hexadecimal value 16 to decimal.

   A. 22_{10} **C.** 10_{10}
   B. 16_{10} **D.** 20_{10}

2. Convert the following decimal number to 8-bit binary. $(187)_{10}$

   A. 10111011_2 **B.** 11011101_2
   C. 10111101_2 **D.** 10111100_2

3. Convert binary 111111110010 to hexadecimal.

   A. $EE2_{16}$ **B.** $FF2_{16}$
   C. $2FE_{16}$ **D.** $FD2_{16}$

4. Convert the following binary number to decimal. 01011_2

   A. 11 **B.** 35
   C. 15 **D.** 10

5. Convert the binary number 1001.0010_2 to decimal.

   A. 90.125 **B.** 9.125
   C. 125 **D.** 12.5

6. Adding in binary, a decimal 26 + 27 will produce a sum of:

   A. 111010 **B.** 110110
   C. 110101 **D.** 101011

7. Add the following hexadecimal numbers.

	3C	14	3B
	+25	+28	+DC
A.	60	3C	116
B.	62	3C	118
C.	61	3C	117
D.	61	3D	117

8. The output of an AND gate with three inputs, A, B, and C, is HIGH when _____.

   A. A = 1, B = 1, C = 0 **B.** A = 0, B = 0, C = 0
   C. A = 1, B = 1, C = 1 **D.** A = 1, B = 0, C = 1

9. Which of the following logical operations is represented by the + sign in Boolean algebra?

   A. inversion **B.** AND
   C. OR **D.** complementation

10. Output will be a LOW for any case when one or more inputs are zero for a(n):

   A. OR gate **B.** NOT gate
   C. AND gate **D.** NOR gate

11. A small circle on the output of a logic gate is used to represent the:

A. Comparator operation. **B.** OR operation.
C. NOT operation. **D.** AND operation.

12. A NOR gate with one HIGH input and one LOW input:

A. will output a HIGH **B.** functions as an AND
C. will not function **D.** will output a LOW

13. The output of a NOR gate is HIGH if _____.

A. all inputs are HIGH **B.** any input is HIGH
C. any input is LOW **D.** all inputs are LOW

14. Which of the examples below expresses the associative law of addition:

A. $A + (B + C) = (A + B) + C$ **B.** $A + (B + C) = A + (BC)$
C. $A(BC) = (AB) + C$ **D.** $ABC = A + B + C$

15. Convert the following SOP expression to an equivalent POS expression.

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

A. $(\overline{A} + \overline{B} + \overline{C})(A + B + \overline{C})(\overline{A} + B + C)$ **B.** $(A + B + C)(A + \overline{B} + C)(A + \overline{B} + \overline{C})$
C. $(\overline{A} + \overline{B} + \overline{C})(A + \overline{B} + C)(A + \overline{B} + C)$ **D.** $(A + B + C)(\overline{A} + B + \overline{C})(A + \overline{B} + C)$

16. Determine the values of A, B, C, and D that make the sum term $\overline{A} + B + \overline{C} + D$ equal to zero.

A. A = 1, B = 0, C = 0, D = 0 **B.** A = 1, B = 0, C = 1, D = 0
C. A = 0, B = 1, C = 0, D = 0 **D.** A = 1, B = 0, C = 1, D = 1

17 Which of the following expressions is in the sum-of-products (SOP) form?

A. $(A + B)(C + D)$ **B.** $(A)B(CD)$
C. $AB(CD)$ **D.** $AB + CD$

18. How many gates would be required to implement the following Boolean expression before simplification? XY + X(X + Z) + Y(X + Z)

A. 1 **B.** 2
C. 4 **D.** 5

19. The output of an exclusive-NOR gate is 1. Which input combination is correct?

A. A = 1, B = 0 **B.** A = 0, B = 1
C. A = 0, B = 0 **D.** none of the above

20. If A and B are the inputs of a half adder, the sum is given by

A. A AND B B. A OR B
C. A XOR B D. A EXOR B

21. If A and B are the inputs of a half adder, the carry is given by

A. A AND B B. A OR B
C. A XOR B D. A EXOR B

22. Half-adders have a major limitation in that they cannot
A. Accept a carry bit from a present stage B. Accept a carry bit from a next stage
C. Accept a carry bit from a previous stage D. None of the Mentioned

23. In parts of the processor, adders are used to calculate
A. Addresses B. Table indices
C. Increment and decrement operators D. All of the Mentioned

24. How many inputs must a full-adder have?

A. 2 **B.** 3
C. 4 **D.** 5

25. What is the hold condition of a flip-flop?
A. both *S* and *R* inputs activated **B.** no active *S* or *R* input
C. only *S* is active **D.** only *R* is active

26. How can parallel data be taken out of a shift register simultaneously?
A. Use the *Q* output of the first FF. **B.** Use the *Q* output of the last FF.
C. Tie all of the *Q* outputs together. **D.** Use the *Q* output of each FF.

27. On a positive edge-triggered S-R flip-flop, the outputs reflect the input condition when
A. clock pulse is LOW **B.** clock pulse is HIGH
C. clock pulse transitions from LOW to HIGH **D.** clock pulse transitions from HIGH to LOW

28. One example of the use of an *S-R* flip-flop is as a(n):
A. racer **B.** astable oscillator
C. binary storage register **D.** transition pulse generator

29. How is a *J-K* flip-flop made to toggle?
A. $J = 0, K = 0$ **B.** $J = 1, K = 0$
C. $J = 0, K = 1$ **D.** $J = 1, K = 1$

30. A J-K flip-flop with J = 1 and K = 1 has a 20 kHz clock input. The Q output is
A. constantly LOW **B.** constantly HIGH
C. a 20 kHz square wave **D.** a 10 kHz square wave

31. How many different states does a 3-bit asynchronous counter have?

A. 2 **B.** 4
C. 8 **D.** 16

32. One of the major drawbacks to the use of asynchronous counters is:

A. low-frequency applications are limited because of internal propagation delays
B. high-frequency applications are limited because of internal propagation delays
C. asynchronous counters do not have major drawbacks and are suitable for use in high- and low-frequency counting applications
D. asynchronous counters do not have propagation delays and this limits their use in high-frequency applications **Answer: B**