DIGITAL LOGIC FUNDAMENTALS

Overview of Number Systems – Binary, Hexa-decimal and BCD numbers. Boolean Algebra - Basic Theorems - Truth Tables and Functionality of Logic Gates – NOT, OR, AND, NOR, NAND, XOR and XNOR. Simple combinational circuits–Half and Full Adders. Introduction to sequential circuits, Clocked S-R and J-K Flip-flops, Simple examples of two-bit Registers and Counters.

INTRODUCTION ABOUT DIGITAL SYSTEM

A Digital system is an interconnection of digital modules and it is a system that manipulates discrete elements of information that is represented internally in the binary form.

Now a day's digital systems are used in wide variety of industrial and consumer products such as automated industrial machinery, pocket calculators, microprocessors, digital computers, digital watches, TV games and signal processing and so on.

Characteristics of Digital systems

- Digital systems manipulate discrete elements of information. Discrete elements are nothing but the digits such as 10 decimal digits or 26 letters of alphabets and so on.
- Digital systems use physical quantities called signals to represent discrete elements.
- In digital systems, the signals have two discrete values (0 or 1) and are therefore said to be binary. A signal in digital system represents one binary digit called a bit. The bit has a value either 0 or 1.

OVERVIEW OF NUMBER SYSTEMS

Modern computers communicate and operate with binary numbers which use only the digits 0 &1. Basic number system used by humans is Decimal number system.

To define any number system, specify

- Base or radix indicates the total number of digits available in the number system such as 2 for Binary number system,8 for octal number system,10 for decimal number system and 16 for Hexa decimal number system.
- The leftmost digit in any number representation, which has the greatest positional weight out of all the digits present in that number, is called the most significant digit **(MSD)** and the rightmost digit, which has the least positional weight out of all the digits present in that number, is called the least significant digit **(LSD).**

1. Binary Number System:

- The binary number has a radix of 2. As r = 2, only two digits are needed, and these are 0 and 1.
- A binary digit is called a bi and binary number consists of a sequence of bits.
- The binary number system is a positional weighted system and weight is expressed as power of 2.
- The **binary point** separates the integer and fraction parts. Each digit (bit) carries a weight based on its position relative to the binary point. The weight of each bit position is one power of 2 greater than the weight of the position to its immediate right.
- The binary number system is used in digital computers because the switching circuits used in these computers use two-state devices such as transistors, diodes, etc. A transistor can be OFF or ON, a switch can be OPEN or CLOSED, a diode can be OFF or ON, etc. These devices have to exist in one of the two possible states. So, these two states can be represented by the symbols 0 and 1, respectively.

a) Binary to Decimal Conversion:

- The decimal value of the binary number is the sum of the products of all its bits multiplied by the weights of their respective positions.
- In general, a binary number with an integer part of (n + 1) bits and a fraction part of k bits can be written as

$$d_n\, d_{n-1}\, d_{n-2} \ldots d_1\, d_0 . d_{-1}\, d_{-2}\, d_{-3} \ldots d_{-k}$$

(Binary Point)

Its decimal equivalent is

$$(d_n * 2^n) + (d_{n-1} * 2^{n-1}) + \cdots (d_1 * 2^1) + (d_0 * 2^0) + (d_{-1} * 2^{-1}) + (d_{-2} * 2^{-2}) \ldots$$

Ex: Convert $101012_{(2)}$ to decimal

Positional Weights	2^4	2^3	2^2	2^1	2^0
Binary Number	1	0	1	0	1

10101 = $(1 * 2^4) + (0 * 2^3) + (1 * 2^2) + (0 * 2^1) + (1 * 2^0) = 16 + 0 + 4 + 0 + 1 = 21_{(10)}$

Ex: Convert $11011.101_{(2)}$ to decimal

Positional Weights	2^4	2^3	2^2	2^1	2^0		2^{-1}	2^{-2}	2^{-3}
Binary Number	1	1	0	1	1	.	1	0	1

11011.101 = $(1 * 2^4) + (1 * 2^3) + (0 * 2^2) + (1 * 2^1) + (1 * 2^0) + (1 * 2^{-1}) + (0 * 2^{-2}) + (1 * 2^{-3})$
= $16 + 8 + 0 + 2 + 1 + 0.5 + 0 + 0.125 = 27.625_{(10)}$

b) Decimal to Binary Conversion:

- In this method, the decimal integer number is converted to the binary integer number by successive division by 2, and the decimal fraction is converted to binary fraction by successive multiplication by 2. This is also known as the **double-dabble method.**
- In the **successive division-by-2 method**, the given decimal integer number is successively divided by 2 till the quotient is zero. The last remainder is the MSB. The remainders read from bottom to top give the equivalent binary integer number.

Ex: Convert $52_{(10)}$ to binary

Successive division		*Remainder*
2	52	
2	26	0
2	13	0
2	6	1
2	3	↑ 0
2	1	| 1
	0	| 1

Reading the remainders from bottom to top, the result is $52_{(10)} = 110100_{(2)}$

In the **successive multiplication-by-2** method, the given decimal fraction and the subsequent decimal fractions are successively multiplied by 2, till the fraction part of the product is 0 or till the desired accuracy is obtained. The first integer obtained is the MSB. Thus, the integers read from top to bottom give the equivalent binary fraction.

Ex: Convert $0.75_{(10)}$ to binary

Given fraction	0.75
Multiply 0.75 by 2	1.50
Multiply 0.50 by 2	1.00

Reading the integers from top to bottom, $0.75_{(10)} = 0.11_{(2)}$.

To convert a **mixed number** to binary, convert the integer and fraction parts separately to binary and then combine them.

Ex: Convert $105.15_{(10)}$ to binary

Conversion of integer 105

Successive division		*Remainder*
2	105	
2	52	1
2	26	0
2	13	0
2	6	1
2	3	0
2	1	1
	0	1

Reading the remainders from bottom to top, $105_{10} = 1101001_2$.

Conversion of fraction 0.15_{10}

Given fraction	0.15
Multiply 0.15 by 2	0.30
Multiply 0.30 by 2	0.60
Multiply 0.60 by 2	1.20
Multiply 0.20 by 2	0.40
Multiply 0.40 by 2	0.80
Multiply 0.80 by 2	1.60

This particular fraction can never be expressed exactly in binary. This process may be terminated after a few steps.
Reading the integers from top to bottom, $0.15_{10} = 0.001001_2$.
Therefore, the final result is, $105.15_{10} = 1101001.001001_2$.

2. Hexadecimal Number System

- Binary numbers are long. These numbers are fine for machines but are too lengthy to be handled by human beings. So, there is a need to represent the binary numbers concisely. One number system developed with this objective is the hexadecimal number system (or Hex) and it has become the most popular means of direct data entry and retrieval in digital systems.
- The hexadecimal number system is a positional-weighted system. The base or radix of this number system is 16, that means, it has 16 independent symbols. The symbols used are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F.
- Since its base is $16 = 2^4$, every 4 binary digit combination can be represented by one hexadecimal digit. So, a hexadecimal number is 1/4th the length of the corresponding binary number, yet it provides the same information as the binary number. A 4-bit group is called a **nibble**. Since computer words come in 8 bits, 16 bits, 32 bits and so on, that is, multiples of 4 bits, they can be easily represented in hexadecimal. The hexadecimal system is particularly useful for human communications with computers.

Hexadecimal	Decimal	Binary
0	**0**	**0000**
1	**1**	**0001**
2	**2**	**0010**
3	**3**	**0011**
4	**4**	**0100**
5	**5**	**0101**
6	**6**	**0110**
7	**7**	**0111**
8	**8**	**1000**
9	**9**	**1001**
A	**10**	**1010**
B	**11**	**1011**
C	**12**	**1100**
D	**13**	**1101**
E	**14**	**1110**
F	**15**	**1111**

a) Binary to Hexadecimal Conversion:

Ex: Convert $1011011011_{(2)}$ to hexadecimal.

Make groups of 4 bits from LSB, and replace each 4-bit group by a hex digit.

			LSB →
Given binary number is		1011011011	
Groups of four bits are	0010	1101	1011
Convert each group to hex	2	D	B

The result is **$2DB_{(16)}$**

Ex: Convert 01011111011.011111$_{(2)}$ to hexadecimal.

Given binary number is			01011111011.011111			
Groups of four bits are	0010	1111	1011	.	0111	1100
Convert each group to hex	2	F	B	.	7	C

The result is **2FB.7C $_{(16)}$**

b) Hexadecimal to Binary Conversion:

Ex: Convert 4BAC$_{16}$ to binary

Given hex number is	4	B	A	C
Convert each hex digit to 4-bit binary	0100	1011	1010	1100

The result is **0100101110101100 $_{(2)}$**

Ex: Convert 3A9E.B0D $_{(16)}$ to binary

Given hex number is	3	A	9	E	.	B	0	D
Convert each hex digit to 4-bit binary	0011	1010	1001	1110	.	1011	0000	1101

The result is **0011101010011110.101100001101$_{(2)}$**

C) Hexadecimal to Decimal Conversion:

To convert a hexadecimal number to decimal, multiply each digit in the hex number by its position weight and add all those product terms.

If the hex number is $\mathbf{d_n\, d_{n-1}\, d_{n-2}\, \ldots d_1\, d_0 . d_{-1}\, d_{-2}\, d_{-3} \ldots d_{-k}}$, its decimal equivalent is

$$(d_n * 16^n) + (d_{n-1} * 16^{n-1}) + \cdots (d_1 * 16^1) + (d_0 * 16^0) + (d_{-1} * 16^{-1}) + (d_{-2} * 16^{-2}) + \cdots$$

Ex: Convert 5C7$_{(16)}$ to decimal.

Multiply each digit of 5C7 by its position weight and add the product terms.

5C716 $= (5 * 16^2) + (12 * 16^1) + (7 * 16^0)$

$= 1280 + 192 + 7$

$= 1479_{(10)}$

Ex: Convert A0F9.0EB $_{(16)}$ to decimal.

A0F9.0EB16 $= (10 * 16^3) + (0 * 16^2) + (15 * 16^1) + (9 * 16^0) + (0 * 16^{-1}) + (14 * 16^{-2}) + (11 * 16^{-3})$

$= 40960 + 0 + 240 + 9 + 0 + 0.0546 + 0.0026$

$= 41209.0572_{(10)}$

d) Decimal to Hexadecimal Conversion:

To convert a decimal integer number to hexadecimal, successively divide the given decimal number by 16 till the quotient is zero. The last remainder is the MSB. The remainders read from bottom to top give the equivalent hexadecimal integer.

To convert a decimal fraction to hexadecimal, successively multiply the given decimal fraction and subsequent decimal fractions by 16, till the product is zero or till the required accuracy is obtained, and collect all the integers to the left of the decimal point. The first integer is the MSB and the integers read from top to bottom give the hexadecimal fraction. This is known as the **hex dabble method**.

Ex: Convert $2598.675_{(10)}$ to hex

The given decimal number is a mixed number. Convert the integer and the fraction parts separately to hex.

Conversion of 2598_{10}

Successive division		*Remainder*	
		Decimal	*Hex*
16	2598		
16	162	6	↑ 6
16	10	2	| 2
	0	10	| A

Reading the remainders upwards, $2598_{10} = A26_{16}$.

Conversion of 0.675_{10}

Given fraction is 0.675

0.675×16	| 10.8
0.800×16	| 12.8
0.800×16	| 12.8
0.800×16	↓ 12.8

Reading the integers to the left of hexadecimal point downwards, $0.675_{10} = 0.ACCC_{16}$.
Therefore, $2598.675_{10} = A26.ACCC_{16}$.

Note: Conversion of very large decimal numbers to binary and very large binary numbers to decimal is very much simplified if it is done via the hex route.

Ex: Convert $49056_{(10)}$ to binary

The given decimal number is very large. It is tedious to convert this number to binary directly.
So, convert this to hex first, and then convert the hex to binary.

Successive division		*Remainder*		
		Decimal	*Hex*	*Binary group*
16 | 49056				
16 | 3066		0	↑ 0	0 0 0 0
16 | 191		10	| A	1 0 1 0
16 | 11		15	| F	1 1 1 1
0		11	| B	1 0 1 1

Therefore, $49056_{10} = BFA0_{16} = 1011,1111,1010,0000_2$.

Ex: Convert 1011011101101110 (2) to decimal.

The given binary number is very large. So, perform the binary to decimal conversion via the hex route.

$$1011, 0111, 0110, 1110_2 = B\ 7\ 6\ E_{16}$$
$$B76E_{16} = (11 \times 16^3) + (7 \times 16^2) + (6 \times 16^1) + (14 \times 16^0)$$
$$= 45056 + 1792 + 96 + 14$$
$$= 46958_{10}$$

Binary Coded Decimal (BCD) Numbers:

Numeric codes are codes which represent numeric information, i.e. only numbers as a series of 0s and 1s. Numeric codes used to represent the decimal digits are called Binary Coded Decimal (BCD) codes.

A BCD code is one, in which the digits of a decimal number are encoded - one at a time - into groups of four binary digits. These codes combine the features of decimal and binary numbers. There are a large number of BCD codes. In order to represent decimal digits 0, 1, 2,…,9 it is necessary to use a sequence of at least four binary digits.

Ex: 8421, 2421, 5211

Decimal digit	8 4 2 1	2 4 2 1	5 2 1 1
0	0 0 0 0	0 0 0 0	0 0 0 0
1	0 0 0 1	0 0 0 1	0 0 0 1
2	0 0 1 0	0 0 1 0	0 0 1 1
3	0 0 1 1	0 0 1 1	0 1 0 1
4	0 1 0 0	0 1 0 0	0 1 1 1
5	0 1 0 1	1 0 1 1	1 0 0 0
6	0 1 1 0	1 1 0 0	1 0 1 0
7	0 1 1 1	1 1 0 1	1 1 0 0
8	1 0 0 0	1 1 1 0	1 1 1 0
9	1 0 0 1	1 1 1 1	1 1 1 1
Unused bit combinations	1 0 1 0	0 1 0 1	0 0 1 0
	1 0 1 1	0 1 1 0	0 1 0 0
	1 1 0 0	0 1 1 1	0 1 1 0
	1 1 0 1	1 0 0 0	1 0 0 1
	1 1 1 0	1 0 0 1	1 0 1 1
	1 1 1 1	1 0 1 0	1 1 0 1

LOGIC GATES

- Logic gates are the fundamental building blocks of digital systems, it produces one output level when some combinations of input levels are present, and a different output level when other combinations of input levels are present.
- Inputs and outputs of logic gates can occur only in two levels. These two levels are termed HIGH and LOW, or TRUE and FALSE, or ON and OFF, or simply 1 and 0.
- A table which lists all the possible combinations of input variables and the corresponding outputs is called a **truth table**. It shows how the logic circuit's output responds to various combinations of logic levels at the inputs.
- Logic Gates are mainly classified as
    1. Basic gates - AND, OR and NOT.
    2. Universal gates – NAND, NOR.

BASIC GATES:

1) The AND GATE

An AND gate has two or more inputs but only one output. The output assumes the logic 1 state, only when each one of its inputs is at logic 1 state. The output assumes the logic 0 state even if one of its inputs is at logic 0 state.

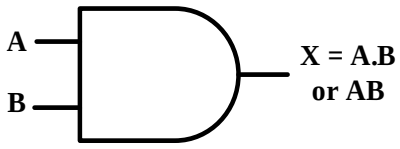
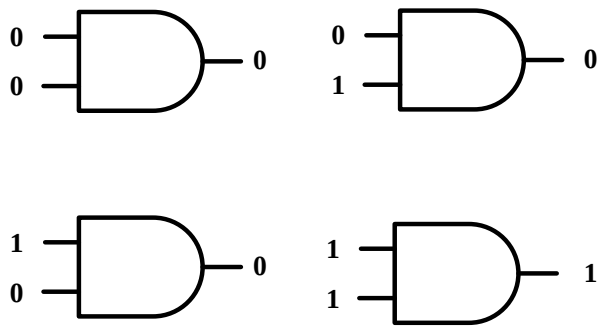


Fig.1 Logic Symbol

The AND gate may, therefore, be defined as a device whose output is 1, if and only if all its inputs are 1. Hence the AND gate is also called an **all or nothing gate**.

With the input variables to the AND gate represented by A, B, the output can be written as X = A.B which is read as 'X is equal to A and B . . .' or 'X is equal to AB . . .', or 'X is equal to A dot B.

The logic symbol **two-input AND** gate is shown in Fig.1 and truth table mentioned in Table1.



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

Table1. Truth Table

Similarly, the logic symbol **three-input AND** gate is shown in Fig.2 and truth table mentioned in Table2.

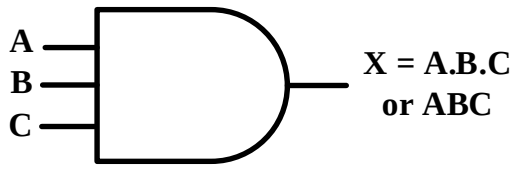


Fig.2 Logic Symbol

Inputs			Output
A	**B**	**C**	**X**
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Table2. Truth Table

2) The OR GATE:

Like an AND gate, an OR gate may have two or more inputs but only one output. The output assumes the logic 1 state, even if one of its inputs is in logic 1 state. Its output assumes the logic 0 state, only when each one of its inputs is in logic 0 state.

An OR gate may, therefore, be defined as a device whose output is 1, even if one of its inputs is 1. Hence an OR gate is also called an **any or all gate**.

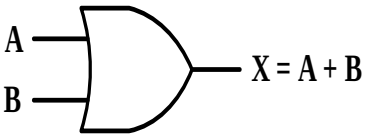
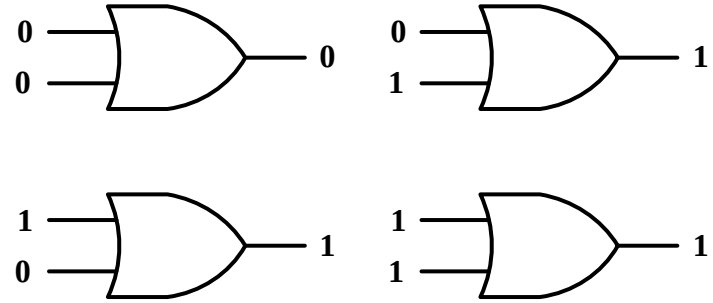


Fig.3 Logic Symbol

It can also be called an **inclusive OR** gate because it includes the condition 'both the inputs can be present'.

The symbol for the OR operation is '+'. With the input variables to the OR gate represented by A, B the output can be written as X = A + B. This is read as 'X is equal to A or B', or 'X is equal to A plus B'.

The logic symbol and the truth table of a two-input OR gate are shown in Fig.3 and truth table mentioned in Table 3.



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

Table 3. Truth Table

Similarly, the logic symbol and the truth table of a **three-input OR** gate are shown in Fig.4 and truth table mentioned in Table 4.

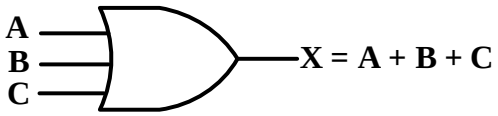


Fig.4 Logic Symbol

Inputs			Output
A	**B**	**C**	**X**
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Table 4. Truth Table

3. The NOT GATE (INVERTER):

A NOT gate, also called an inverter, has only one input and only one output.

It is a device whose output is always the complement of its input. That is, the output of a NOT gate assumes the logic 1 state when its input is in logic 0 state and assumes the logic 0 state when its input is in logic 1 state.

The logic symbol and the truth table of an NOT gate are shown in shown in Fig.5 and truth table mentioned in Table 5.

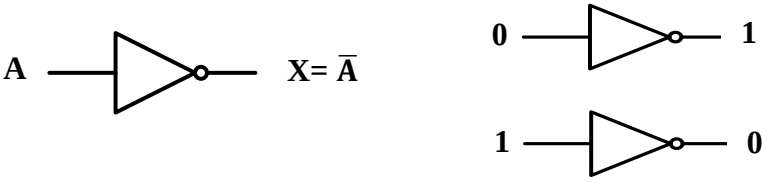


Fig.5 Logic Symbol

Input	Output
A	**X**
0	1
1	0

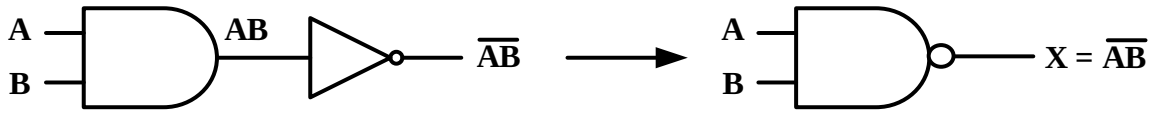
Table 5. Truth Table

The symbol for NOT operation is '–' (bar). When the input variable to the NOT gate is represented by A and the output variable by X, the expression for the output is $\mathbf{X = \overline{A}}$. This is read as 'X is equal to A bar'.

UNIVERSAL GATES

1. The NAND GATE:

NAND means NOT - AND, i.e. the AND output is NOTed. So, a NAND gate is a combination of an AND gate and a NOT gate.



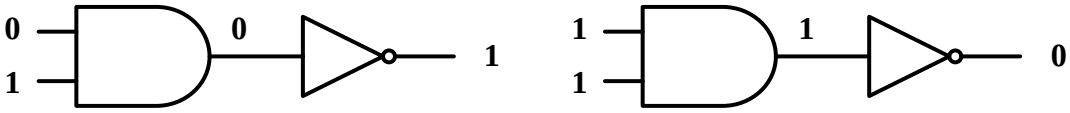
An AND gate followed by a NOT gate **A two – input NAND gate**

Fig.6 Logic Symbol

The expression for the output of the NAND gate can, therefore, be written as $X = \overline{AB}$ and is read as 'X is equal to AB whole bar'. The output is logic 0 level, only when each of the inputs assumes a logic 1 level. For any other combination of inputs, the output is a logic 1 level. The logic symbol for a two-input NAND gate is shown in Fig.6 and its truth table mentioned in Table.6.

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

Table 6. Truth Table



The logic symbol for a three-input NAND gate is shown in Fig.7 and its truth table mentioned in Table.7.

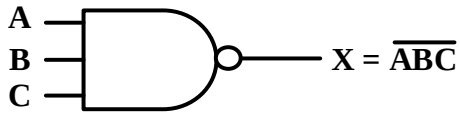
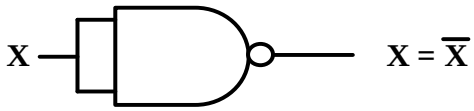


Fig.7 Logic Symbol

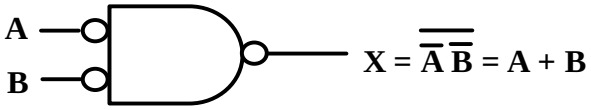
Inputs			Output
A	**B**	**C**	**X**
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Table 7. Truth Table

- A NAND gate can also be used as an **NOT gate** (inverter) by tying all its input terminals together and applying the signal to be inverted to the common terminal.

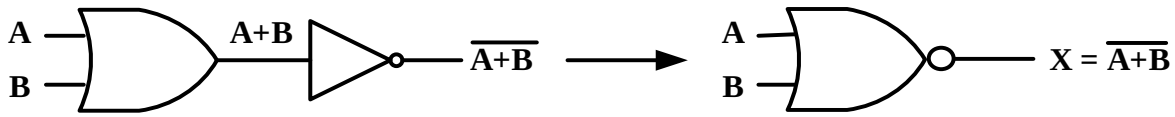


- The bubbled NAND gate is equivalent to **OR gate**



2. The NOR GATE:

NOR means NOT OR, i.e. the OR output is NOTed. So, a NOR gate is a combination of an OR gate and a NOT gate.



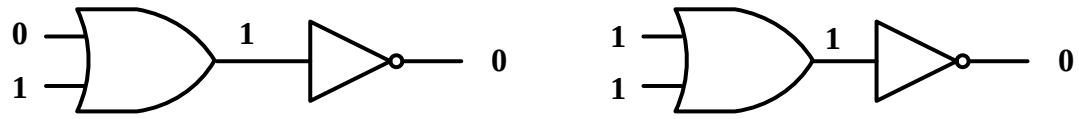
An AND gate followed by a NOT gate **A two – input NAND gate**

Fig.8 Logic Symbol

The expression for the output of the NOR gate is, $\mathbf{X = \overline{A + B}}$ and is read as 'X is equal to A plus B whole bar'. The output is logic 1 level, only when each one of its inputs assumes a logic 0 level. For any other combination of inputs, the output is a logic 0 level. The logic symbol for a two-input NAND gate is shown in Fig.8 and its truth table mentioned in Table.8.

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

Table 8. Truth Table



The logic symbol for a three-input NOR gate is shown in Fig.9 and its truth table mentioned in Table.9

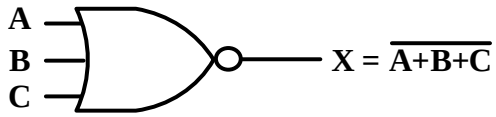
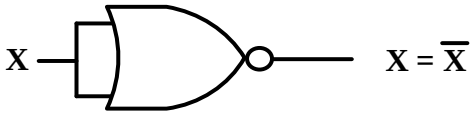


Fig.9 Logic Symbol

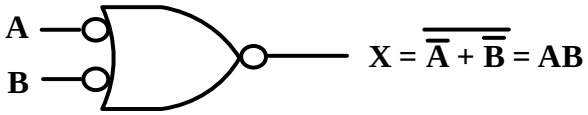
Inputs			Output
A	**B**	**C**	**X**
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

Table 9. Truth Table

- A NOR gate can also be used as an **NOT gate** (inverter), by tying all its input terminals together and applying the signal to be inverted to the common terminal.



- The bubbled NOR gate is equivalent to an **AND gate**



The EXCLUSIVE-OR (X-OR) GATE:

- An X-OR gate is a two input, one output logic circuit, whose output assumes a logic 1 state when one and only one of its two inputs assumes a logic 1 state. Under the conditions when both the inputs assume the logic 0 state, or when both the inputs assume the logic 1 state, the output assumes a logic 0 state.

- Since an X-OR gate produces an output 1 only when the inputs are not equal, it is called an **anti-coincidence gate or inequality detector.**

- The name Exclusive-OR is derived from the fact that its output is a 1, only when exclusively one of its inputs is a 1 (it excludes the condition when both the inputs are 1).

The logic symbol and truth table of a two-input X-OR gate is shown in below Fig. 10 and Table. 10. If the input variables are represented by A and B and the output variable by X, the expression for the output of this gate is written as $\mathbf{X = A \oplus B = \overline{A}B + A\overline{B}}$ and read as 'X is equal to A ex-or B'.

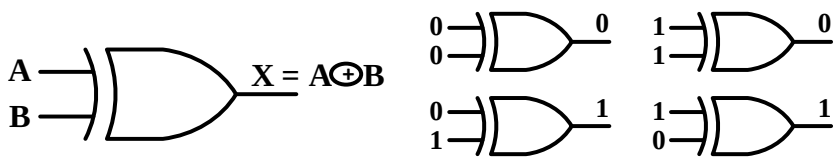


Fig.10 Logic Symbol

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

Table.10 Truth Table

The EXCLUSIVE-NOR (X-NOR) GATE:

❖ An X-NOR gate is a combination of an X-OR gate and a NOT gate. The X-NOR gate is a two input, one output logic circuit, whose output assumes a 1 state only when both the inputs assume a 0 state or when both the inputs assume a 1 state. The output assumes a 0 state, when one of the inputs assumes a 0 state and the other a 1 state.

❖ It is also called a **coincidence gate**, because its output is 1 only when its inputs coincide. It can be used as an **equality detector** because it outputs a 1 only when its inputs are equal.

The logic symbol and truth table of a two-input X-NOR gate is shown in below Fig. 11 and Table. 11. If the input variables are represented by A and B and the output variable by X, the expression for the output of this gate is written as as $\mathbf{X = A \odot B = AB + \overline{A}\overline{B}}$ and read as 'X is equal to A ex-nor B'.

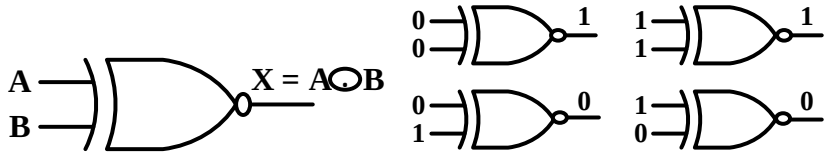


Fig.11 Logic Symbol

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

Table.11 Truth Table

BOOLEAN ALGEBRA

❖ Boolean algebra is an algebraic system consisting of the set of elements (0,1), two **binary operators** called OR and AND and one **unary operator** called NOT. It is the basic mathematical tool in the analysis and synthesis of switching circuits / logic circuits.

❖ Boolean algebra differs from both the ordinary algebra and the binary number system. In Boolean algebra, A + A = A and A . A = A, because the variable A has only a logical value. It doesn't have any numerical significance. In ordinary algebra, A + A = 2A and A. A = A^2, because the variable A has a numerical value here. In Boolean algebra, 1 + 1 = 1, whereas in the binary number system, 1 + 1 = 10, and in ordinary algebra, 1 + 1 = 2.

❖ There is nothing like subtraction or division in Boolean algebra. Also, there are no negative or fractional numbers in Boolean algebra. In Boolean algebra, the multiplication and addition of the variables and functions are also only logical. They actually represent logic operations. **Logical multiplication** is the same as the **AND** operation, and **logical addition** is the same as the **OR** operation.

LAWS OF BOOLEAN ALGEBRA:

1. Complementation Laws

The term complement simply means to invert, i.e. to change 0s to 1s and 1s to 0s.

Law 1:		$\overline{0} = 1$		
Law 2:		$\overline{1} = 0$		
Law 3:	If	$A = 0$,	then	$\overline{A} = 1$
Law 4:	If	$A = 1$,	then	$\overline{A} = 0$
Law 5:		$\overline{\overline{A}} = A$	(double complementation law)	

2. AND Laws

Law 1:	$A \cdot 0 = 0$ (Null law)
Law 2:	$A \cdot 1 = A$ (Identity law)
Law 3:	$A \cdot A = A$
Law 4:	$A \cdot \overline{A} = 0$

3. OR Laws

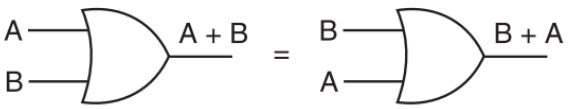
Law 1:	$A + 0 = A$ (Null law)
Law 2:	$A + 1 = 1$ (Identity law)
Law 3:	$A + A = A$
Law 4:	$A + \overline{A} = 1$

4. Commutative Laws

Commutative laws allow change in position of AND or OR variables. There are two commutative laws.

Law 1: A + B = B + A

This law states that, A OR B is the same as B OR A, i.e. no difference which input of an OR gate is connected to A and which to B.



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

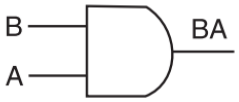
=

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

Law 2: A . B = B. A

This law states that A AND B is the same as B AND A, i.e. no difference which input of an AND gate is connected to A and which to B.

=



A	B	A · B
0	0	0
0	1	0
1	0	0
1	1	1

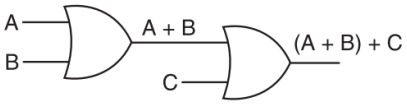
=

B	A	B · A
0	0	0
0	1	0
1	0	0
1	1	1

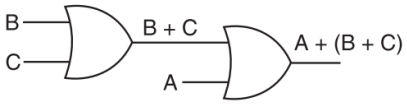
5. Associative Laws

The associative laws allow grouping of variables. There are two associative laws.

Law 1: (A + B) + C = A + (B + C)



=

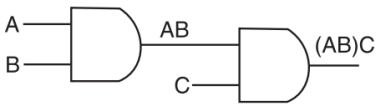


A	B	C	A + B	(A + B) + C
0	0	0	0	0
0	0	1	0	1
0	1	0	1	1
0	1	1	1	1
1	0	0	1	1
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

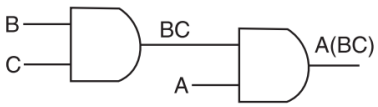
=

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

Law 2: (A.B).C = A.(B.C)



=



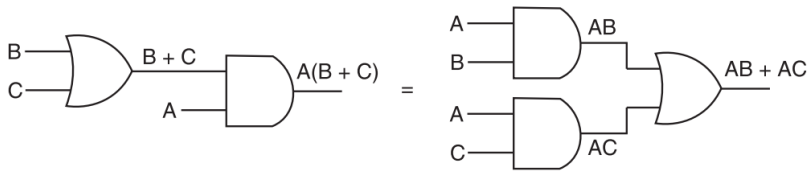
A	B	C	AB	(AB)C
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	0	0
1	0	0	0	0
1	0	1	0	0
1	1	0	1	0
1	1	1	1	1

=

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

6. Distributive Laws

Law 1: A(B + C) = AB + AC

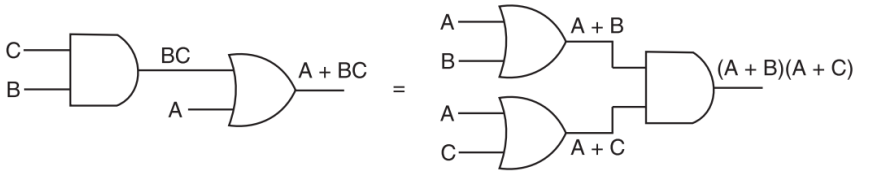


A	B	C	B + C	A(B + C)
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	1	0
1	0	0	0	0
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

=

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

Law 2: A + BC = (A + B)(A + C)



A	B	C	BC	A + BC
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	1	1
1	0	0	0	1
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

=

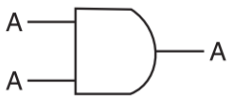
A	B	C	A + B	A + C	(A + B)(A + C)
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	1	0	0
0	1	1	1	1	1
1	0	0	1	1	1
1	0	1	1	1	1
1	1	0	1	1	1
1	1	1	1	1	1

7. Idempotence Laws

Law 1: A . A = A

If A = 0, then $A \cdot A = 0 \cdot 0 = 0 = A$

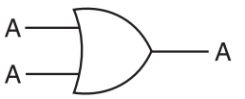
If A = 1, then $A \cdot A = 1 \cdot 1 = 1 = A$



Law 2: A + A = A

If A = 0, then A + A = 0 + 0 = 0 = A

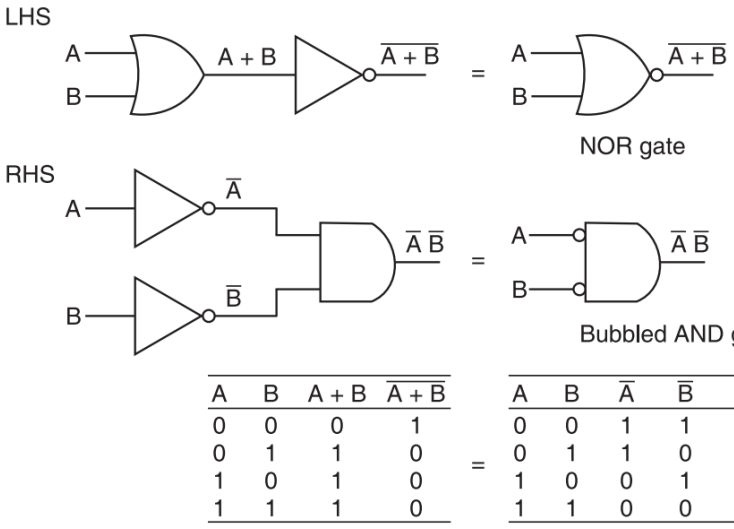
If A = 1, then A + A = 1 + 1 = 1 = A



De Morgan's Theorem:

Law 1 : $\overline{A+B} = \overline{A}\,\overline{B}$

This law states that the complement of a sum of variables is equal to the product of their individual complements.



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

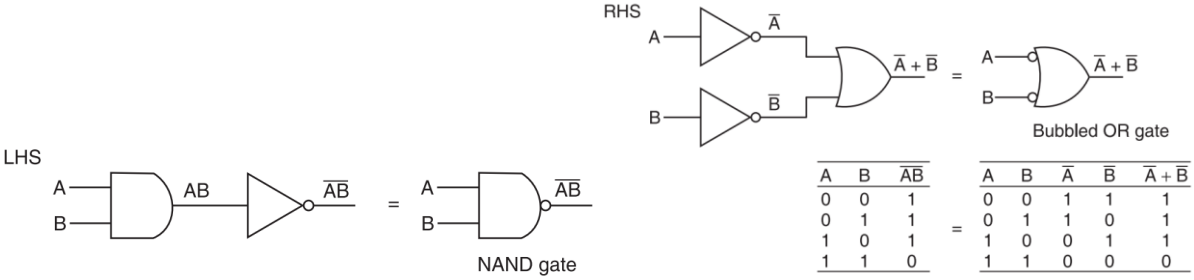
=

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

It shows that the NOR gate is equivalent to a bubbled AND gate.

Law 2 : $\overline{AB} = \overline{A} + \overline{B}$

This law states that the complement of the product of variables is equal to the sum of their individual complements.



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

It shows that the NAND gate is equivalent to a bubbled OR gate.

Operator Precedence:

The operator precedence for evaluating Boolean expression is (i) parenthesis (ii) NOT (iii) AND and (iv) OR. In other words, the expression inside the parenthesis must be evaluated before all other operations. The next operation that holds precedence is the complement, then follows the AND and finally the OR.

Ex: Reduce the expression $f = A\,[B + \bar{C}\,(\overline{AB + A\bar{C}}\,)]$

The given expression is

$f = A\,[B + \bar{C}\,(\overline{AB + A\bar{C}}\,)$

$f = A\,[B + \bar{C}\,(\overline{AB}\,.\,\overline{A\bar{C}}\,)$ - (DE Morgan's Law - 1)

$f = A\,[B + \bar{C}\,(\bar{A} + \bar{B})\,(\bar{A} + C)]$ - (DE Morgan's Law - 2)

$f = A\,[B + \bar{C}\,(\bar{A}\bar{A} + \bar{A}C + \bar{B}\bar{A} + \bar{B}C\,)]$ - (Distributive Law - 1)

$f = A\,[B + \bar{C}\,\bar{A} + \bar{C}\,\bar{A}\,C + \bar{C}\,\bar{B}\,\bar{A} + \bar{C}\,\bar{B}C\,]$ - (Idempotence Laws)

$f = A\,[B + \bar{C}\,\bar{A} + 0 + \bar{C}\,\bar{B}\,\bar{A} + 0\,]$ - (AND Laws)

$f = AB + A\bar{C}\,\bar{A} + A\bar{C}\,\bar{B}\,\bar{A}$ - (Distributive Law - 1)

$f = AB + 0 + 0$ - (AND Laws)

$f = AB$

Ex: Reduce the expression $f = A + B\,[AC + (B + \bar{C}\,)D]$

The given expression is

$f = A + B\,[AC + (B + \bar{C}\,)D]$

$f = A + B\,[AC + BD + \bar{C}\,D\,]$ - (Distributive Law - 1)

$f = A + BAC + BBD + B\bar{C}\,D$ - (Distributive Law - 1)

$f = A + BAC + BD + B\bar{C}\,D$ - (Idempotence Laws)

$f = A(1 + BC) + BD(1 + \bar{C}\,)$ - (Factor)

$f = A(1) + BD(1)$ - (OR Laws)

$f = A + BD$

Ex: Reduce the expression $f = (\overline{A + \bar{B}\bar{C}})(A\bar{B} + ABC)$

Demorganize $(\overline{A + \bar{B}\bar{C}})$	$= (\bar{A}\,\overline{\bar{B}\bar{C}})(A\,\bar{B} + ABC)$
Simplify	$= (\bar{A}BC)(A\,\bar{B} + ABC)$
Multiply	$= \bar{A}BC\,A\,\bar{B} + \bar{A}BC\,ABC$
Rearrange	$= A\bar{A}B\,\bar{B}C + A\bar{A}BBCC$
	$= 0 + 0 = 0$

Ex: Reduce the expression $f = B + (BC)(B + \bar{B}C)(B + D)$

Multiply the first two terms	$= (BB + BCB + B\,\bar{B}C + BC\,\bar{B}C)(B + D)$
Reduce	$= (B + BC + 0 + 0)(B + D)$
Factor	$= B(1 + C)(B + D)$
Reduce	$= B(B + D)$
Expand	$= BB + BD$
Simplify	$= B(1 + D) = B$

Ex: Show that $\mathbf{AB + A\bar{B}C + B\bar{C} = AC + B\bar{C}}$

$$\begin{aligned} AB + A\bar{B}C + B\bar{C} &= A(B + \bar{B}C) + B\bar{C} \\ &= A(B + \bar{B})(B + C) + B\bar{C} \\ &= AB + AC + B\bar{C} \\ &= AB(C + \bar{C}) + AC + B\bar{C} \\ &= ABC + AB\bar{C} + AC + B\bar{C} \\ &= AC(1 + B) + B\bar{C}(1 + A) \\ &= AC + B\bar{C} \end{aligned}$$

Ex: Show that $\mathbf{A\bar{B}C + B + B\bar{D} + AB\bar{D} + \bar{A}C = B + C}$

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

COMBINATIONAL CIRCUITS

- A combinational circuit consists of input variables, logic gates, and output variables as shown in Fig.12.
- The **output** of a combinational circuit **depends on its present inputs only**.
- For 'n' input variables, there are 2^n possible combinations of binary input values. For each possible input combination, there is one and only one possible output combination.

Fig.12 The block diagram of a combinational circuit

Half-Adder:

A combinational circuit that performs the addition of two bits is called a half-adder, with two binary inputs (augend and addend bits) and two binary outputs (sum and carry bits). It adds the two inputs (A and B) and produces the sum (S) and the carry (C) bits. The truth table and block diagram of a half-adder are shown in Fig.13.

Inputs		Outputs	
A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

(a) Truth table

(b) Block diagram

	1	1	0	0
	+1	+0	+1	+0
(carry)1	0	1	1	0

Fig.13 Half-adder

This simple addition consists of 4 possible operations, **Carry**

0 + 0 = 0, 0 + 1 = 1, 1 + 0 = 1, and 1 + 1 = 10

The first three operations produce a sum whose length is one digit, but when both augend and addend bits are equal to 1, the binary sum consists of two digits. The higher significant bit of this result is called a **carry.** When the augend and addend numbers contain more significant digits, the carry obtained from the addition of two bits is added to the next higher-order pair of significant bits.

In the **sum column,** the output is HIGH (1) when either input is HIGH (1) but not when both inputs are the same and in the **carry column,** the output is HIGH (1) when both inputs are the HIGH (1). Therefore,

The sum (S) is the X-OR of A and B, $\mathbf{S = A \oplus B = \bar{A}B + A\bar{B}}$ and

The carry (C) is the AND of A and B, $\mathbf{C = AB}$

A half-adder can, therefore, be realized by using one X-OR gate and one AND gate as shown in Fig.14.

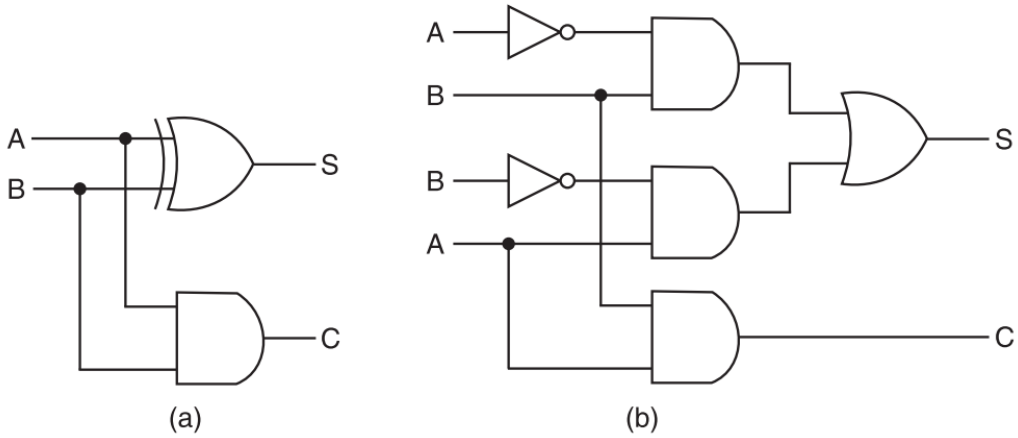


Fig.14 Logic diagrams of half-adder

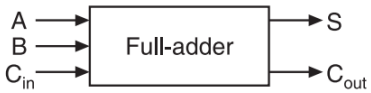
Full-Adder:

A combinational circuit that performs the addition of three bits (two significant bits and previous carry) is called a full-adder and outputs a sum bit and a carry bit.

The full-adder adds the bits **A** and **B** and the carry from the previous column called the carry-in **C_{in}** and outputs the sum bit **S** and the carry bit called the carry-out **Cout**. The variable S gives the value of the least significant bit (LSB) of the sum. The variable C_{out} gives the output carry. The truth table and the block diagram of a full-adder are shown in Fig.15.

Inputs			Sum	Carry
A	B	C_{in}	S	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

(a) Truth table



(b) Block diagram

Fig.15 Full-adder

The eight rows under the input variables (A, B, & C_{in}) designate all possible combinations of 1s and 0s. The 1s and 0s for the output variables (S & C_{out}) are determined from the arithmetic sum of the input bits.

1. When all the bits are 0s, the output is 0.
2. The S output is equal to 1 when only 1 input is equal to 1 or when all the inputs are equal to 1.
3. The C_{out} has a carry of 1 if two or three inputs are equal to 1.

From the truth table, a circuit that will produce the sum and carry bits in response to every possible combination of A, B, and C_{in} is described by

$$\mathbf{S = \bar{A}\bar{B}C_{in} + \bar{A}B\bar{C}_{in} + A\bar{B}\bar{C}_{in} + ABC_{in} = (\bar{A}B + A\bar{B})\bar{C}_{in} + (\bar{A}\bar{B} + AB)C_{in} = A \oplus B \oplus C_{in}} \quad \&$$

$$\mathbf{C_{out} = \bar{A}BC_{in} + A\bar{B}C_{in} + AB\bar{C}_{in} + ABC_{in} = AB + (A \oplus B)\, C_{in} = AB + AC_{in} + BC_{in}}$$

The logic diagram of the full-adder using two X-OR gates and two AND gates (i.e. two half-adders) and one OR gate is shown in Fig.16.

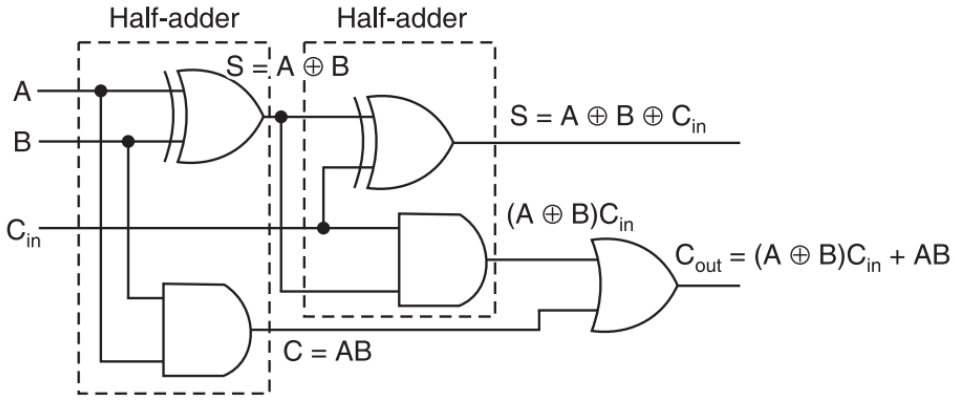


Fig.16 Logic diagrams of full-adder

INTRODUCTION TO SEQUENTIAL CIRCUITS

- Basically, switching circuits may be combinational switching circuits or sequential switching circuits.
- In Combinational switching circuits any prior input level conditions have no effect on the present outputs, because combinational logic circuits have no memory.
- In, sequential switching circuits output levels at any instant of time are dependent not only on the present inputs level, but also on the state of the circuit, i.e. on the prior input level conditions (i.e. on its past inputs). The past history is provided by feedback from the output back to the input. It means that sequential switching circuits have memory. Sequential circuits are thus made of combinational circuits and memory elements. The past history is provided by feedback from the output back to the input.

  Ex: Counters, shift registers, serial adders, sequence generators, logic function generators, etc.

  Fig.17 shows a block diagram of a sequential circuit. The memory elements are connected to the combinational circuit as a feedback path.

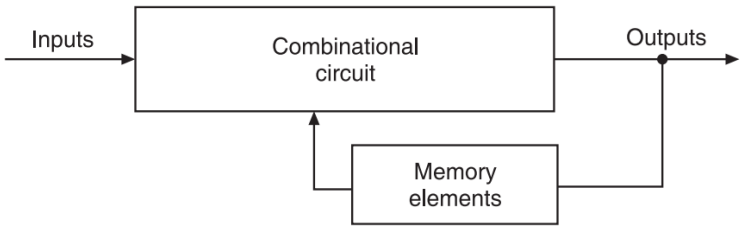


Fig.17 Block diagram of a sequential circuit

The information stored in the memory element at any given time defines the present state of the sequential circuit. The present state and the external inputs determine the outputs and the next state of the sequential circuit.

- The sequential circuits may be classified as **synchronous sequential circuits** and **asynchronous sequential circuits** depending on the timing of their signals.
  - The sequential circuits which are controlled by a clock are called synchronous sequential circuits **(Flip-Flops).** These circuits will be active only when clock signal is present.
  - The sequential circuits which are not controlled by a clock are called asynchronous sequential circuits (**Latches**), i.e. the sequential circuits in which events can take place any time the inputs are applied are called asynchronous sequential circuits.

FLIP-FLOPS:

Flip-flops are the basic building blocks of most sequential circuits. A flip-flop (FF), known more formally as a bistable multivibrator, has two stable states (0 or 1). It can remain in either of the states indefinitely. Its state can be changed by applying the proper triggering signal. It is also called a binary or one-bit memory.

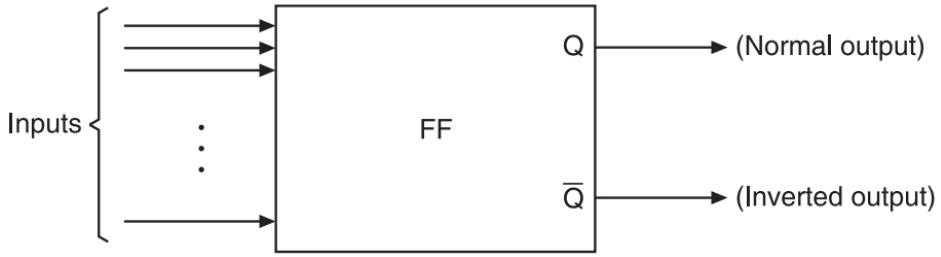
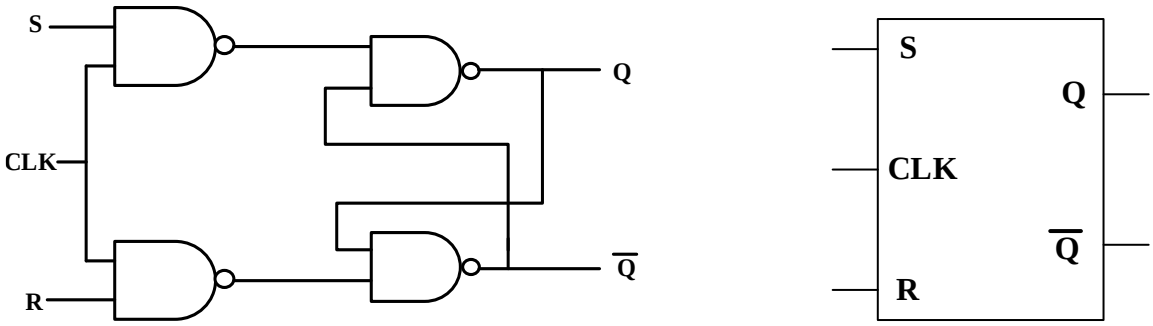


Fig.18 Flip-Flop symbol

Fig.18 shows the general type of symbol used for a flip-flop. The flip-flop has two outputs, labelled Q and $\overline{Q}$. The Q output is the normal output of the flip-flop and $\overline{Q}$ is the inverted output. The output as well as its complement are available for each flip-flop. The **state of the flip-flop always refers to the state of the normal output Q.** The inverted output $\overline{Q}$ is in the opposite state. A flip-flop is said to be in HIGH state or logic 1 state or SET state when Q = 1, and in LOW state or logic 0 state or RESET state or CLEAR state when Q = 0.

Clocked S-R Flip-Flop

The S and R inputs of the S-R flip-flop are called the synchronous control inputs because data on these inputs affect the flip-flop's output only on the triggering (positive going or negative going) edge of the clock pulse (CLK). Without a clock pulse, the S and R inputs cannot affect the output. So, it is also called a **Synchronous S-R Flipflop.** The logic diagram & the logic symbol are shown in Fig.19. and truth table & waveforms are shown in Fig.21 respectively.



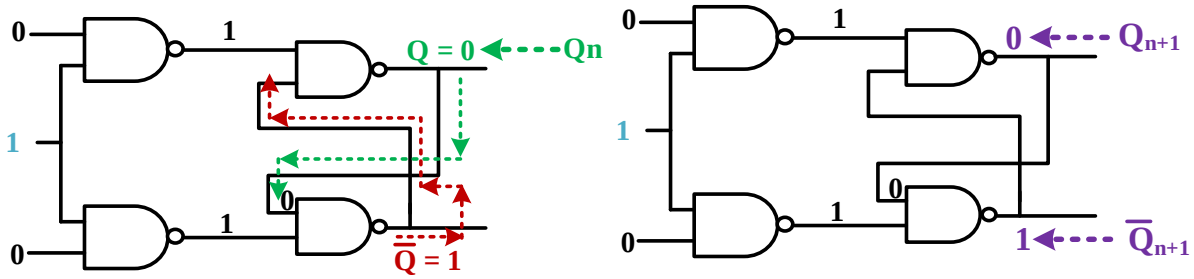
(a) Logic Diagram **(b) Logic Symbol**

Fig.19 Clocked S-R Flip Flop

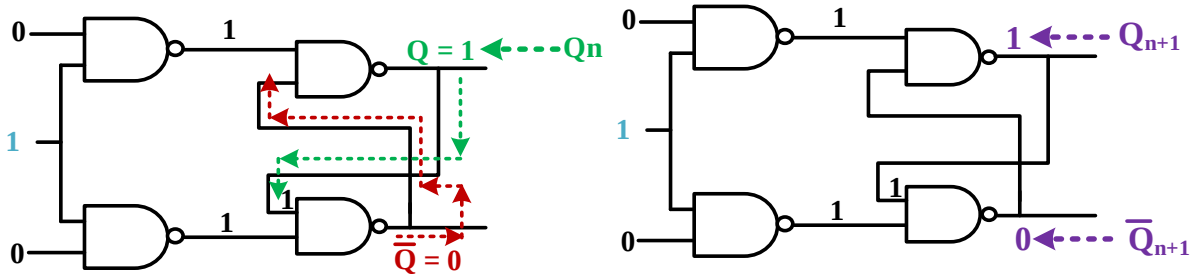
$\mathbf{Q_n}$ represents the state of the flip-flop before applying the inputs (i.e. the **present state** PS of the flip-flop). $\mathbf{Q_{n+1}}$ represents the state of the flip-flop after the application of the inputs (i.e. the **next state** NS of the flip-flop).

A. S= 0, R= 0: NO CHANGE (NC)

This is the normal resting state and it has no effect on the output state. Q and $\overline{Q}$ will remain in whatever state they were prior to the occurrence of this input condition.



(i) $Q_n = 0$

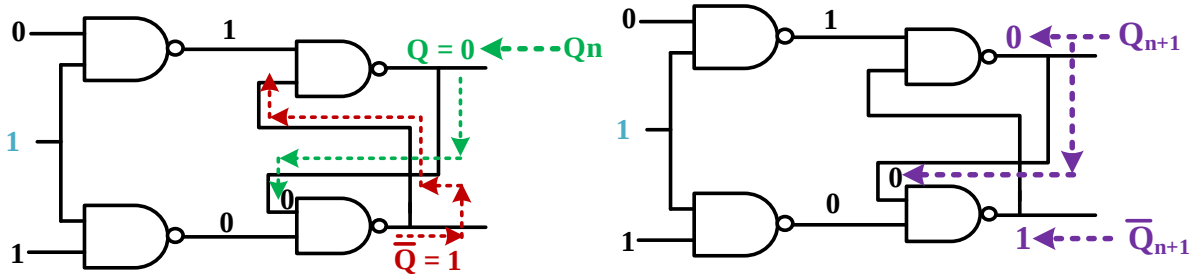


(ii) $Q_n = 1$

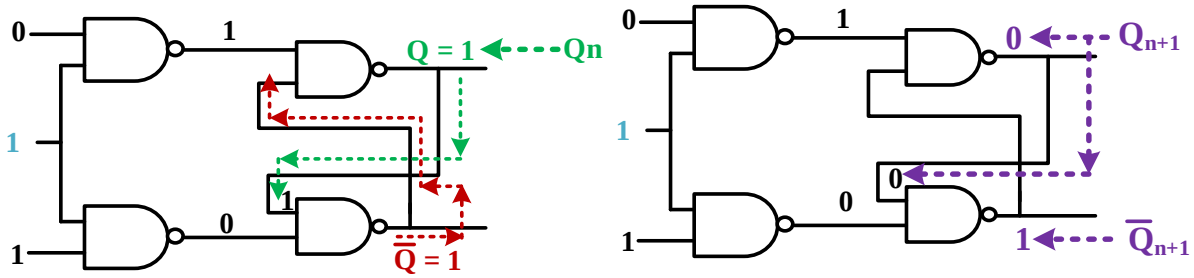
<u>CASE A: S= 0, R= 0 & CLK = 1</u>

B. S=0, R = 1: RESET

This will always reset Q = 0, where it will remain even after RESET returns to 0.



(i) $Q_n = 0$

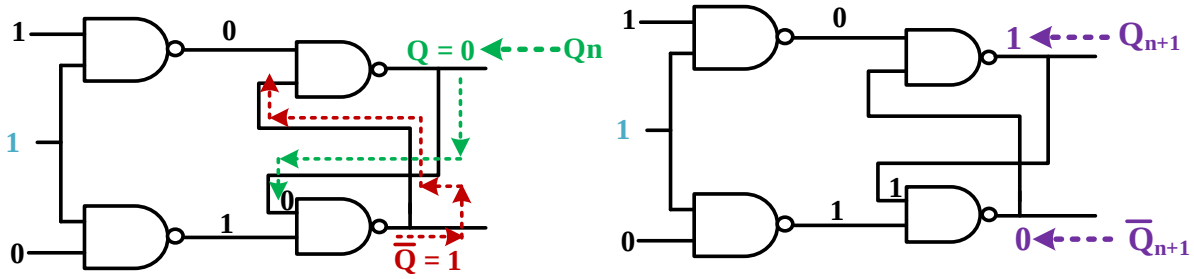


(ii) $Q_n = 1$

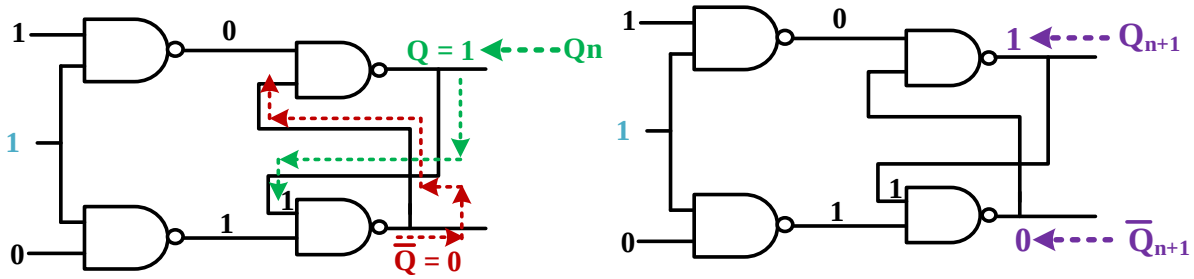
<u>CASE B: S= 0, R= 1 & CLK = 1</u>

C. S=1, R = 0: SET

This will always set Q = 1, where it will remain even after SET returns to 0.



(i) $Q_n = 0$

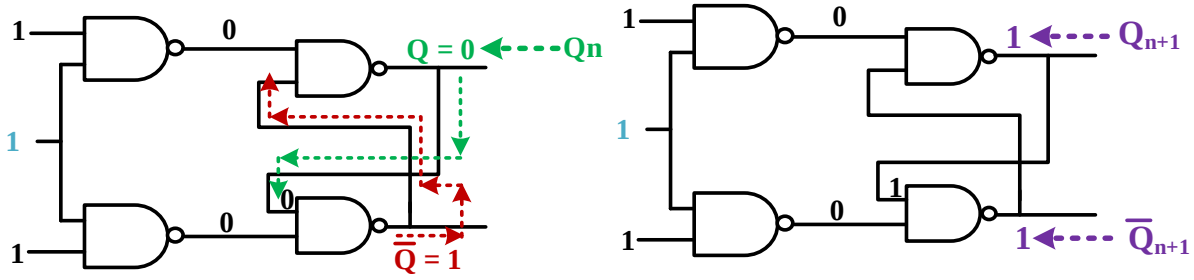


(ii) $Q_n = 1$

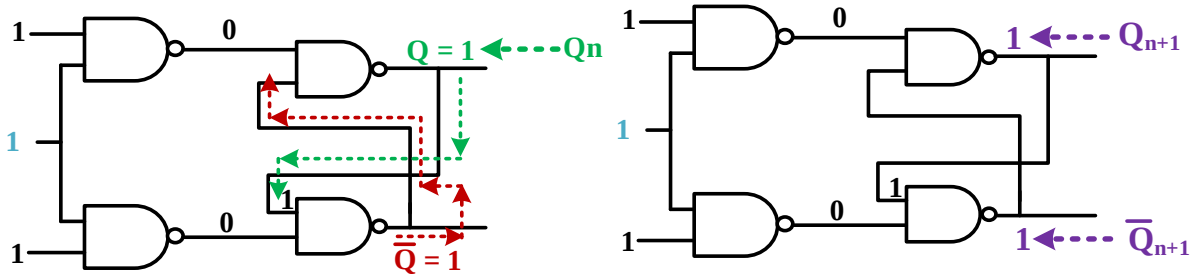
CASE C: S= 1, R= 0 & CLK = 1

D. S=1, R = 1: INDETERMINATE

This condition tries to SET and RESET at the same time, and it produces $Q = \overline{Q} = 1$. If the inputs are returned to zero simultaneously, the resulting output state is erratic and unpredictable. This input condition should not be used. It is forbidden.



(i) $Q_n = 0$



(ii) $Q_n = 1$

CASE D: S= 1, R= 1 & CLK = 1

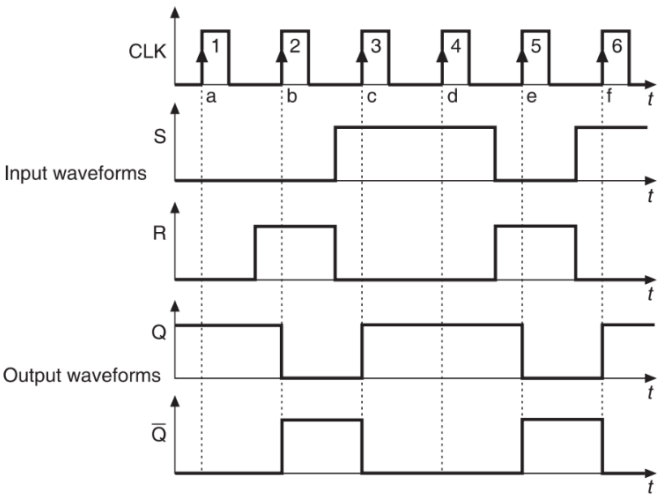
If both the inputs are made HIGH, the output is unpredictable, i.e. both Q and $\overline{Q}$ may be HIGH, or both may be LOW or any one of them may be HIGH and the other LOW. This condition is described as not allowed, unpredictable, invalid or indeterminate.

Fig. 20 Edge Triggering

The process of applying the control signal to change the state of a flip-flop is called **triggering**. There are two types of triggering the flip-flops: level triggering and edge triggering.

In **edge triggering**, the input signals affect the flip-flop only if they are present at the positive going or negative going edge of the clock pulse as shown in Fig.20.

CLK	S	R	Q_n	Q_{n+1}	State
↑	0	0	0	0	No Change (NC)
	0	0	1	1	
↑	0	1	0	0	Reset
	0	1	1	0	
↑	1	0	0	1	Set
	1	0	1	1	
↑	1	1	0	X (0 or 1)	Indeterminate
	1	1	1	X – don't care	
0	X	X	0	0	No Change (NC)
0	X	X	1	1	



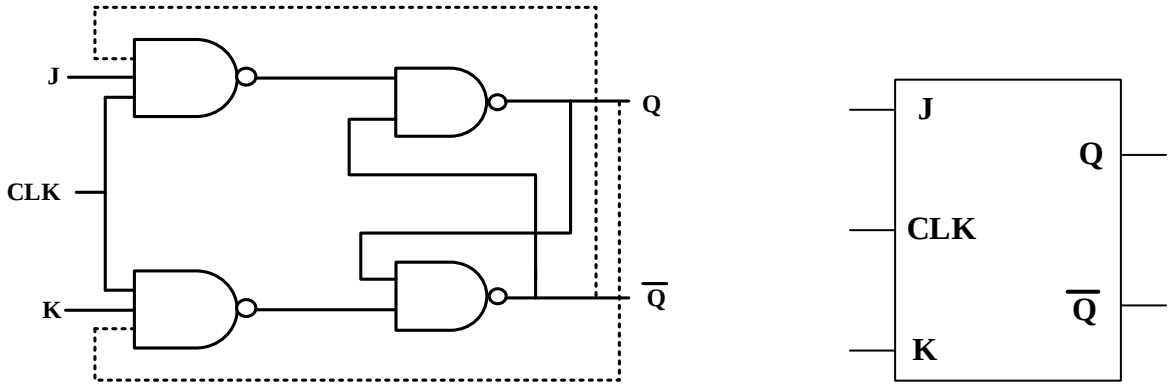
(a) Truth Table **(b) Waveforms: positive edge-triggered**

Fig.21

1. Initially, S = 0 and R = 0 and **Q is assumed to be HIGH**.
2. At the positive-going transition of the first clock pulse (i.e. at a), both S and R are LOW. So, no change of state takes place. Q remains HIGH and $\overline{Q}$ remains LOW.
3. At the leading edge of the second clock pulse (i.e. at b), S = 0 and R = 1. So, the flip-flop resets. Hence, Q goes LOW and $\overline{Q}$ goes HIGH.
4. At the positive-going edge of the third clock pulse (i.e. at c), S = 1 and R = 0. So, the flip-flop sets. Hence, Q goes HIGH and $\overline{Q}$goes LOW.
5. At the rising edge of the fourth clock pulse, S = 1 and R = 0. Since the flip-flop is already in a SET state, it remains SET. That is, Q remains HIGH and $\overline{Q}$ remains LOW.
6. The fifth pulse resets the flip-flop at its positive-going edge because S = 0 and R = 1 is the input condition and Q = 1 at that time.
7. The sixth pulse sets the flip-flop at its rising edge because S = 1 and R = 0 is the input condition and Q = 0 at that time.

Clocked J-K Flip-Flop

The J-K flip-flop is very versatile and also the most widely used. The J and K designations for the synchronous control inputs have no known significance. The functioning of the J-K flip-flop is identical to that of the S-R flip-flop, except that it has **no invalid state like that of the S-R flip-flop**. The logic diagram & the logic symbol are shown in Fig.22 and truth table & waveforms are shown in Fig.23 respectively.

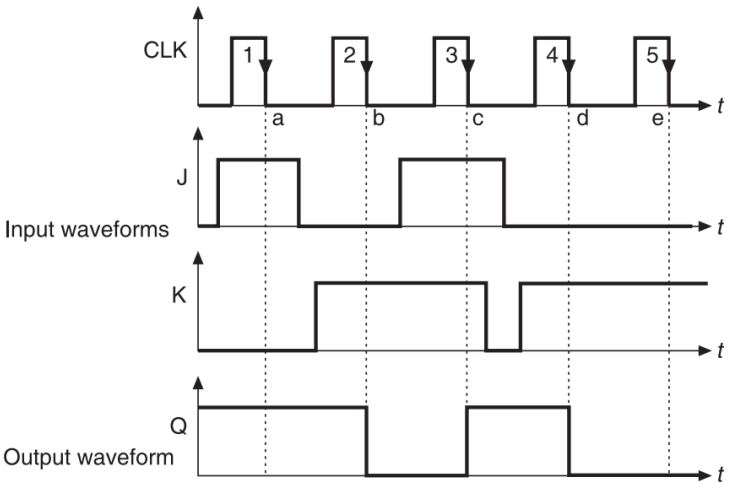


(a) Logic Diagram **(b) Logic Symbol**

Fig.22 Clocked J-K Flip Flop

- When J = 0 and K = 0, **no change** of state takes place even if a clock pulse is applied.
- When J = 0 and K = 1, the flip-flop **resets** at the edge of the clock pulse.
- When J = 1 and K = 0, the flip-flop **sets** at the edge of the clock pulse.
- When J = 1 and K = 1, the flip-flop **toggles,** i.e. goes to the opposite state at edge of the clock pulse. In this mode, the flip-flop toggles or changes state for each occurrence of the of the clock pulse.

CLK	J	K	Q_n	Q_{n+1}	State
↓	0	0	0	0	No Change (NC)
	0	0	1	1	
↓	0	1	0	0	Reset
	0	1	1	0	
↓	1	0	0	1	Set
	1	0	1	1	
↓	1	1	0	1	Toggle
	1	1	1	0	
0	X	X	0	0	No Change (NC)
0	X	X	1	1	



(a) Truth Table **(b) Waveforms: negative edge-triggered**

Fig.23

1. Initially J = 0, K = 0 and CLK = 0. **Assume that the initial state of the flip-flop is a 1, i.e. Q = 1 initially.**
2. At the negative-going edge of the first clock pulse (i.e. at a), J = 1 and K = 0. So, Q remains as a 1 and, therefore, $\overline{Q}$ as a 0.
3. At the trailing edge of the second clock pulse (i.e. at b), J = 0 and K = 1. So, the flip-flop resets. That is, Q goes to a 0 and $\overline{Q}$ to a 1.
4. At the falling edge of the third clock pulse (i.e. at c), both J and K are a 1. So, the flip-flop toggles. That is, Q changes from a 0 to a 1 and $\overline{Q}$ from a 1 to a 0.
5. At the negative-going transition of the fourth clock pulse (i.e. at d), J = 0 and K = 1. So, the flip-flop RESETS, i.e. Q goes to a 0 and $\overline{Q}$ to a 1.
6. At the negative going edge of the fifth clock pulse (i.e. at d), J = 0 and K = 1. So the flipflop remains reset, i.e. Q remains as 0 and $\overline{Q}$ remains as 1.

SHIFT REGISTERS

- Shift registers are a type of logic circuits, are used basically for the storage and transfer of digital data.
- A flip-flop (FF) can store only one bit of data, a 0 or a 1, it is referred to as a single-bit register. When more bits of data are to be stored, a number of FFs are used. A register is a set of FFs used to store binary data.
- A register might be used to accept input data from an alphanumeric keyboard and then present the data at the input of a microprocessor chip.
- Data may be available (I/P) in parallel form or in serial form.
  - Multi-bit data is said to be in **parallel form** when all the bits are available (accessible) simultaneously. The data is said to be in **serial form** when the data bits appear sequentially (one after the other, in time) at a single terminal.
- Data may also be transferred (O/P) in parallel form or in serial form.
  - **Parallel data transfer** is the simultaneous transmission of all bits of data from one device to another. **Serial data transfer** is the transmission of one bit of data at a time from one device to another. Serial data must be transmitted under the synchronization of a clock, since the clock provides the means to specify the time at which each new bit is sampled.

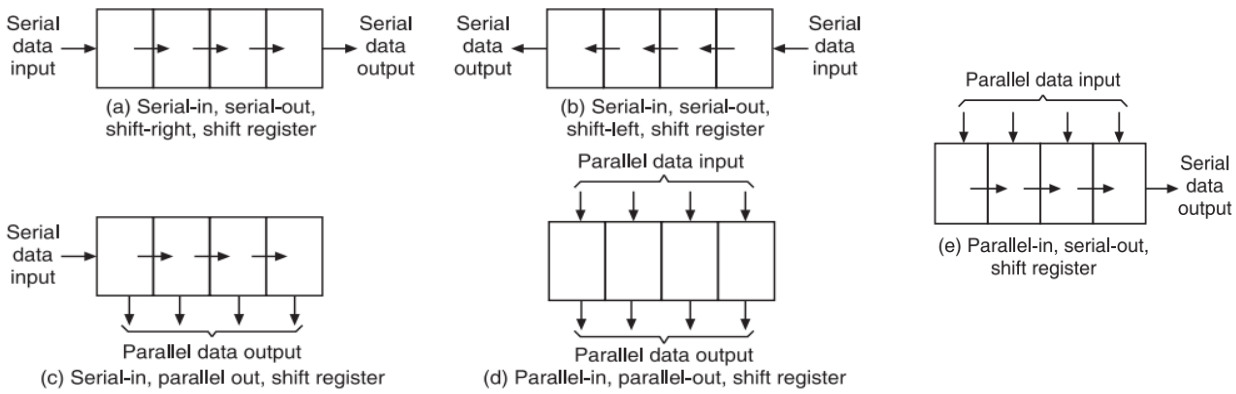
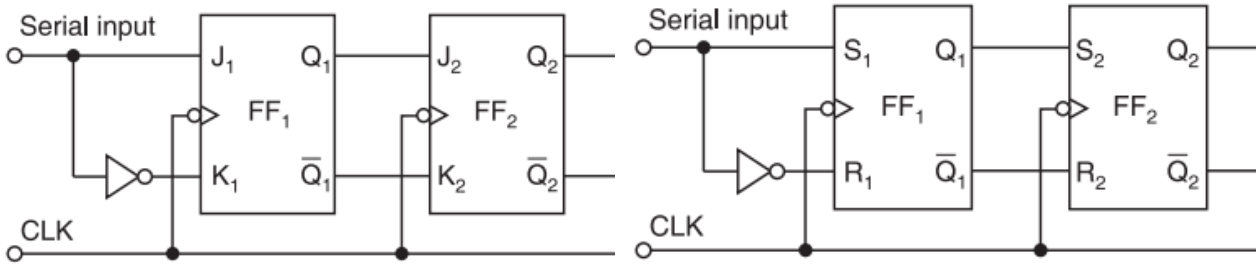


Fig. 24 Data transfer in registers



(a) Using J-K FFs **(b) Using S-R FFs**

Fig. 25 2-bit serial-in, serial-out, (SISO) shift register

A shift register can be constructed using J-K FFs or S-R FFs as shown in Fig. 25a and 25b, respectively. The data is applied at the J(S) input of the first FF. The complement of this is fed to the K(R) terminal of the first FF. The Q output of the first FF is connected to J(S) input of the second FF. The input (data) is applied at the leftmost of the FF (MSB) while the output is taken from the right most FF (LSB).

MSB **LSB**

Input: **1** **0**

1. Initially, say all the bits of the register are 0. So, whenever there is no clock pulse (CLK), then irrespective of the input, all the FF will store its current data (No change state). i.e., $Q_1 = Q_2 = 0$.

2. Starting from LSB, to shift it into the shift register, apply the value (here it is 0) at the input terminal. Before CLK, all values are zero and once clock edge (here it is negative edge triggering) arrives, then all the FFs will respond to their present inputs. For CLK -1, $Q_1 = 0$ ($J_1 = 0$: RESET state); $Q_2 = 0$ (due to previous Q_1, i.e., value at CLK-0, $Q_1 = J_2 = 0$) and for CLK -2, $Q_1 = 1$ ($J_1 = 1$: SET state); $Q_2 = 0$ (value at CLK-1, $Q_1 = J_2 = 0$).

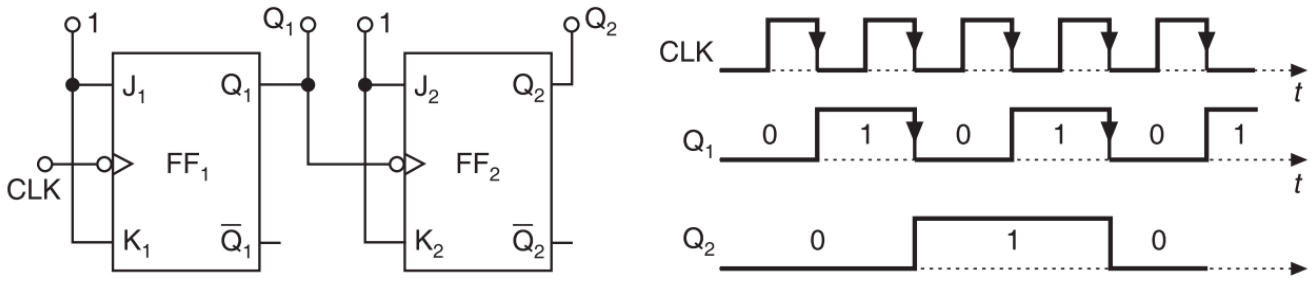
CLK		Input	Q_1	Q_2	
NO	0	X	0	0	Previous stored data in the last FF (FF-2) is discarded after zero clock
↓	1	0	0	0	Discarded after first clock
↓	2	1	1	0	Output at Q_2 of FF-2
↓	3	-	-	1	

COUNTERS

- A digital counter is a set of flip-flops (FFs) whose states change in response to pulses applied at the input to the counter.
- The FFs are interconnected such that their combined state at any time is the binary equivalent of the total number of pulses that have occurred up to that time. Thus, as its name implies, a counter is used to count pulses.
- A counter used to perform the timing function as in digital watches, to create time delays, to produce non-sequential binary counts, to generate pulse trains, and to act as frequency counters, etc.

Two-bit Ripple Up-counter:

The 2-bit up-counter counts in the order 0, 1, 2, 3, 0, 1, …, i.e. 00, 01, 10, 11, 00, 01,…, etc. Fig.26 shows a 2-bit ripple up-counter, using **negative edge-triggered** J-K FFs, and its timing diagram.



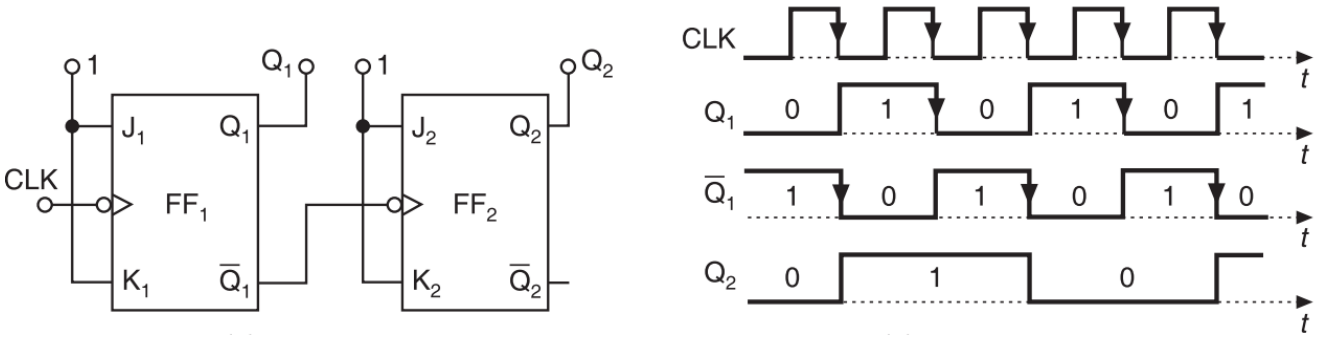
(a) Logic Diagram **(b) Timing Diagram**

Fig.26

- The **counter is initially reset to 00**. When the first clock pulse is applied, FF_1 toggles at the negative-going edge of this pulse, therefore, Q_1 goes from LOW to HIGH. This becomes a positive-going signal at the clock input of FF_2. So, FF_2 is not affected, and hence, the state of the counter after one clock pulse is $Q_1 = 1$ and $Q_2 = 0$, i.e. 01.
- At the negative-going edge of the second clock pulse, FF_1 toggles. So, Q_1 changes from HIGH to LOW and this negative-going signal applied to CLK of FF_2 activates FF_2, and hence, Q_2 goes from LOW to HIGH. Therefore, $Q_1 = 0$ and $Q_2 = 1$, i.e. 10 is the state of the counter after the second clock pulse.
- At the negative-going edge of the third clock pulse, FF_1 toggles. So Q_1 changes from a 0 to a 1. This becomes a positive-going signal to FF_2, hence, FF_2 is not affected. Therefore, $Q_2 = 1$ and $Q_1 = 1$, i.e. 11 is the state of the counter after the third clock pulse.
- At the negative-going edge of the fourth clock pulse, FF_1 toggles. So, Q_1 goes from a 1 to a 0. This negative-going signal at Q_1 toggles FF_2, hence, Q_2 also changes from a 1 to a 0. Therefore, $Q_2 = 0$ and $Q_1 = 0$, i.e. 00 is the state of the counter after the fourth clock pulse.
- For subsequent clock pulses, the counter goes through the same sequence of states with Q_1 as the LSB and Q_2 as the MSB. The counting sequence is thus 00, 01, 10, 11, 00, 01,…, etc.

Two-bit Ripple Down-counter:

A 2-bit down-counter counts in the order 0, 3, 2, 1, 0, 3,…, i.e. 00, 11, 10, 01, 00, 11, …, etc. Fig.27 shows a 2-bit ripple down-counter, using **negative-edge triggered** J-K FFs, and its timing diagram.



(a) Logic Diagram **(b) Timing Diagram**

Fig.27

- For down counting, $\overline{Q_1}$ of FF1 is connected to the clock of FF_2. Let initially all the FFs be reset, i.e. let the count be 00. At the negative-going edge of the first clock pulse, FF_1 toggles, so, Q_1 goes from a 0 to a 1 and $\overline{Q_1}$ goes from a 1 to a 0. This negative-going signal at $\overline{Q_1}$ applied to the clock input of FF_2, toggles FF_2 and, therefore, Q_2 goes from a 0 to a 1. So, after one clock pulse $Q_2 = 1$ and $Q1 = _1$, i.e. the state of the counter is 11.
- At the negative-going edge of the second clock pulse, Q_1 changes from a 1 to a 0 and $\overline{Q_1}$ from a 0 to a 1. This positive-going signal at $\overline{Q_1}$ does not affect FF_2 and, therefore, Q_2 remains at a 1. Hence, the state of the counter after the second clock pulse is 10.
- At the negative-going edge of the third clock pulse, FF_1 toggles. So, Q_1 goes from a 0 to a 1 and $\overline{Q_1}$ from a 1 to a 0. This negative-going signal at $\overline{Q_1}$ toggles FF_2 and, so, Q_2 changes from a 1 to a 0. Hence, the state of the counter after the third clock pulse is 01.
- At the negative-going edge of the fourth clock pulse, FF_1 toggles. So, Q_1 goes from a 1 to a 0 and $\overline{Q_1}$ from a 0 to a 1. This positive-going signal at $\overline{Q_1}$ does not affect FF_2. So, Q_2 remains at a 0. Hence, the state of the counter after the fourth clock pulse is 00. For subsequent clock pulses the counter goes through the same sequence of states, i.e. the counter counts in the order 00, 11, 10, 01, 00, and 11 …