DIGITAL ELECTRONICS
(AECB03)
III SEMESTER
ELECTRICAL AND ELECTRONICS ENGINEERING

PREPARED BY
V.BINDU SREE
J.SRAVANA

Unit-I

FUNDAMENTALS OF DIGITAL SYSTEMS AND LOGIC FAMILIES

NUMBER SYSTEMS

- **Binary number system.**

A method of representing **numbers** that has 2 as its base and uses only the digits 0 and 1.

Ex:10100010

- **Decimal number system**

A number system that uses a notation in which each number is expressed in base 10 by using one of the first nine integers or 0 in each place and letting each place value be a power of 10

Numbers:0,1,2,3,4,5,6,7,8,9

NUMBER SYSTEMS

- **Octal number system**

  The octal numbering system uses the numerals 0-1-2-3-4-5-6-7.

- **Hexa decimal number system**

  The hexadecimal numeral system, often shortened to "hex", is a numeral system made up of 16 symbols (base 16) they are 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E.

NUMBER BASE CONVERSION

Binary to Decimal Conversion:

It is by the positional weights method . In this method, each binary digit of the no. is multiplied by its position weight . The product terms are added to obtain the decimal no

Example:

101011_2 =>

$1 \times 2^0 = 1$

$1 \times 2^1 = 2$

$0 \times 2^2 = 0$

$1 \times 2^3 = 8$

$0 \times 2^4 = 0$

$1 \times 2^5 = 32$

43_{10}

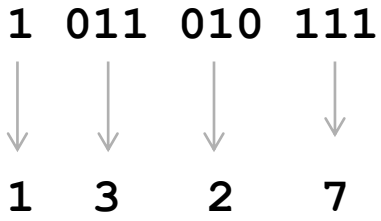
NUMBER BASE CONVERSION

Binary to Octal conversion:

Starting from the binary pt. make groups of 3 bits each, on either side of the binary pt, & replace each 3 bit binary group by the equivalent octaldigit.

$$1011010111_2 = ?_8$$

Example:



$$1011010111_2 = 1327_8$$

NUMBER BASE CONVERSION

Binary to Hexadecimal conversion:

For this make groups of 4 bits each , on either side of the binary pt & replace each 4 bit group by the equivalent hexadecimal digit.

Example:

$1010111011_2 = ?_{16}$

10 1011 1011

↓ ↓ ↓

2 B B

$1010111011_2 = 2BB_{16}$

NUMBER BASE CONVERSION

Decimal to Binary conversion:

Technique

- Divide by two, keep track of the remainder
- First remainder is bit 0 (LSB, least-significant bit)
- Solution $125_{10} = ?_2$

2	125	
2	62	1
2	31	0
2	15	1
2	7	1
2	3	1
2	1	1
	0	1

$$125_{10} = 1111101_2$$

NUMBER BASE CONVERSION

Decimal to Octal Conversion:

- To convert a mixed decimal no to a mixed octal no. convert the integer and fraction parts separately.
- To convert decimal integer no. to octal, successively divide the given no by 8 till the quotient is 0. The last remainder is the MSD The remainder read upwards give the equivalent octal integer no.
- To convert the given decimal fraction to octal, successively multiply the decimal fraction & the subsequent decimal fractions by 8 till the product is 0 or till the required accuracy is the MSD. The integers to the left of the octal pt read downwards give the octal fraction.

NUMBER BASE CONVERSION

Example:

$$1234_{10} = ?_{8}$$

8	1234	
8	154	2
8	19	2
8	2	3
	0	2

$$1234_{10} = 2322_{8}$$

NUMBER BASE CONVERSION

Decimal to Hexadecimal conversion:

- It is successively divide the given decimal no. by 16 till the quotient is zero. The last remainder is the MSB. The remainder read from bottom to top gives the equivalent hexadecimal integer.
- To convert a decimal fraction to hexadecimal successively multiply the given decimal fraction & 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 decimal pt. The first integer is MSB & the integer read from top to bottom .

NUMBER BASE CONVERSION

Example:

$1234_{10} = ?_{16}$

16 | 1234		
16 | 77	2	
16 | 4	13	= D
0	4	

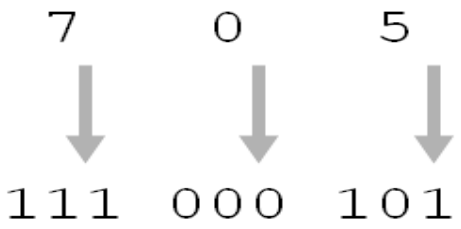
$1234_{10} = 4D2_{16}$

NUMBER BASE CONVERSION

Octal to binary Conversion:

Convert each octal digit to a 3-bit equivalent binary representation

$705_8 = ?_2$



$705_8 = 111000101_2$

NUMBER BASE CONVERSION

Octal to decimal Conversion:

Multiply each digit in the octal no by the weight of its position & add all the product terms Decimal value of the octal no.

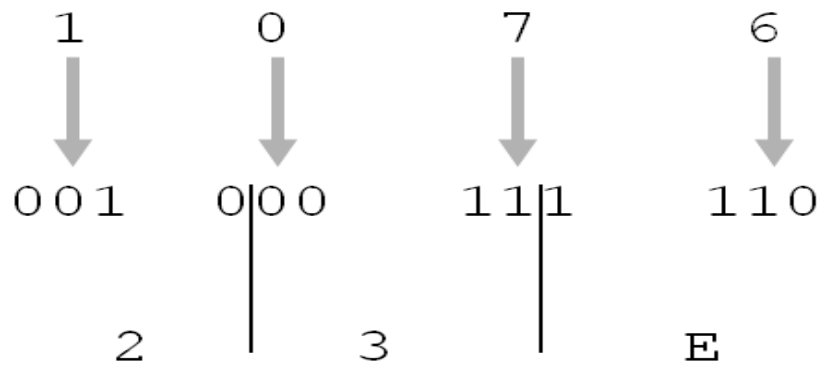
$$724_8 \Rightarrow \quad 4 \times 8^0 = 4$$
$$2 \times 8^1 = 16$$
$$7 \times 8^2 = 448$$
$$\overline{468_{10}}$$

NUMBER BASE CONVERSION

Octal to hexadecimal conversion:

The simplest way is to first convert the given octal no. to binary & then the binary no. to hexadecimal.

$1076_8 = ?_{16}$



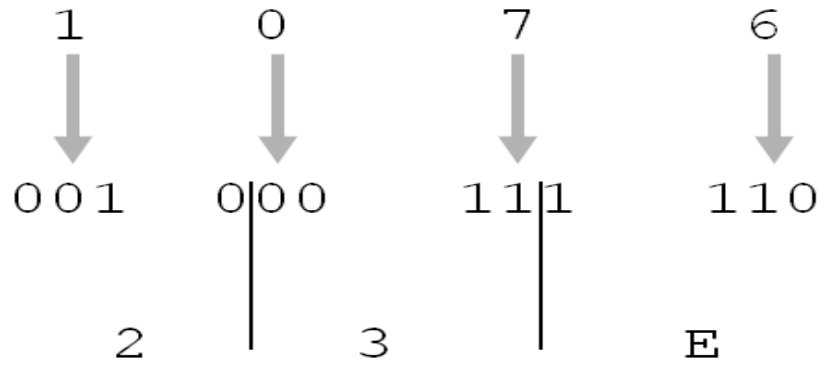
$1076_8 = 23E_{16}$

NUMBER BASE CONVERSION

Octal to hexadecimal conversion:

The simplest way is to first convert the given octal no. to binary & then the binary no. to hexadecimal.

$1076_8 = ?_{16}$



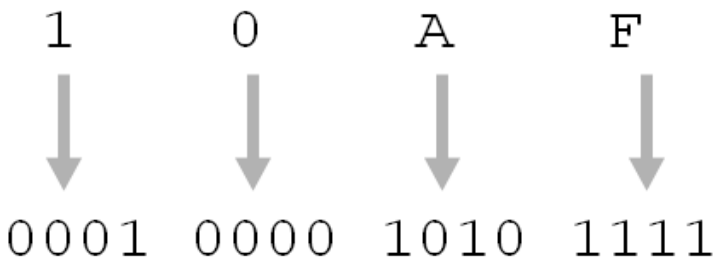
$1076_8 = 23E_{16}$

NUMBER BASE CONVERSION

Hexa decimal to binary Conversion:

Convert each hexadecimal digit to a 4-bit equivalent binary representation

$10AF_{16} = ?_2$



$10AF_{16} = 0001000010101111_2$

NUMBER BASE CONVERSION

Hexa decimal to decimal Conversion:

Convert each hexadecimal digit to a 4-bit equivalent binary representation

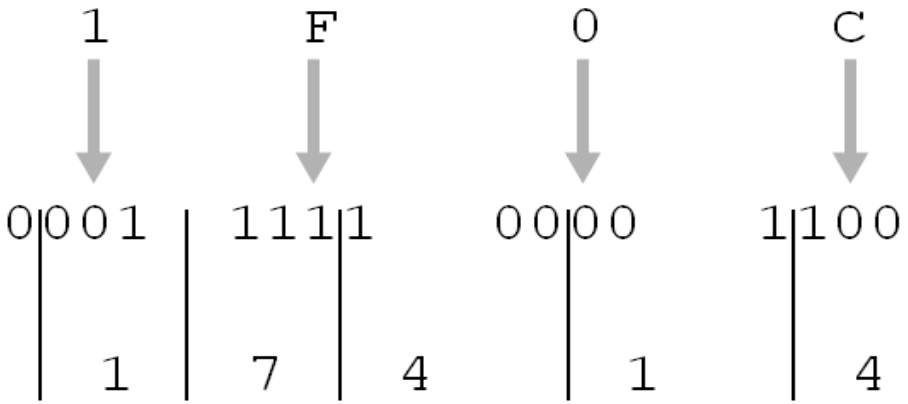
$$
\begin{aligned}
ABC_{16} \Rightarrow \quad & C \times 16^0 = 12 \times 1 = 12 \\
& B \times 16^1 = 11 \times 16 = 176 \\
& A \times 16^2 = 10 \times 256 = 2560 \\
& \qquad\qquad\qquad\qquad\qquad\qquad\quad 2748_{10}
\end{aligned}
$$

NUMBER BASE CONVERSION

Hexa decimal to octal Conversion:

Use binary as an intermediary

$1F0C_{16} = ?_8$



$1F0C_{16} = 17414_8$

BINARYARITHMETIC

Binary Addition:

Rules: 0+0=0

0+1=1

1+0=1

1+1=10

i.e, 0 with a carry of 1.

Binary Subtraction:

Rules: 0-0=0

1-1=0

1-0=1

0-1=1

with a borrow of 1

BINARY ARITHMETIC

Binary multiplication:

Rules:

0x0=0

1x1=0

1x0=0

0x1=0

Binary Division:

Example : 101101_2 by 110

110) 101101 (111.1

110

1010

110

1001

110

110

110

000

COMPLEMENTS

1's compliment of n number:

It is obtained by simply complimenting each bit of the numbers that is all 1's to 0's and all 0's to 1's

Example: 1's complement of 1101001 is 0010110

2's compliment of n number:

It is obtained by simply complimenting each bit of the numbers that is all 1's to 0's and all 0's to 1's and add 1 to the value

Example: 2's complement of $\overline{1101001}$ is 0010110

+1= 0010111

COMPLEMENTS

9's & 10's Complements:

It is the Subtraction of decimal number can be accomplished by the 9's & 10's compliment methods similar to the 1's & 2's compliment methods of binary . The 9's compliment of a decimal number is obtained by subtracting each digit of that decimal number from 9. The 10's compliment of a decimal number is obtained by adding a 1 to its 9's compliment.

COMPLEMENTS

1’s compliment arithmetic:

In1’s complement subtraction, add the 1’s complement of the subtrahend to the minuend. If there is a carryout , bring the carry around & add it to the LSB called the end around carry. Look at the sign bit (MSB) . If this is a 0, the result is positive & is in true binary. If the MSB is a 1 (carry or no carry), the result is negative & is in its is complement form .Take its 1’s complement to get the magnitude in binary.

COMPLEMENTS

2's compliment arithmetic:

The 2's complement system is used to represent negative numbers using modulus arithmetic . The word length of a computer is fixed i.e, if a 4 bit number is added to another 4 bit number the result will be only of 4 bits. Carry if any , from the fourth bit will overflow called the Modulus arithmetic.

SIGNED BINARY NUMBERS

Representation of signed numbers binary arithmetic in computers:

Two ways of representing signed numbers is Sign Magnitude form and Complemented form there are two complimented forms i.e. 1's compliment form and 2's compliment form

BINARY WEIGHTED AND NON- WEIGHTED CODES

Weighted Codes:

The weighted codes are those that obey the position weighting principle, which states that the position of each number represent a specific weight. In these codes each decimal digit is represented by a group of four bits.

In weighted codes, each digit is assigned a specific weight according to its position. For example, in 8421/BCD code, 1001 the weights of 1, 1, 0, 1 (from left to right) are 8, 4, 2 and 1 respectively.

Examples:8421,2421 are all weighted codes.

BINARY WEIGHTED AND NON- WEIGHTED CODES

Non-weighted codes:

The non-weighted codes are not positionally weighted . In other words codes that are not assigned with any weight to each digit position.

Examples:Excess-3(XS-3) and Gray Codes.

BINARY CODED DECIMAL

BCD Addition:

It is individually adding the corresponding digits of the decimal numbers expressed in 4 bit binary groups starting from the LSD .

If there is no carry & the sum term is not an illegal code , no correction is needed.

If there is a carry out of one group to the next group or if the sum term is an illegal code then 6_{10} (0110) is added to the sum term of that group & the resulting carry is added to the next group.

BINARY CODED DECIMAL

BCD Subtraction:

Performed by subtracting the digits of each 4 bit group of the subtrahend the digits from the corresponding 4- bit group of the minuend in binary starting from the LSD . if there is no borrow from the next group , then 6_{10} (0110) is subtracted from the difference term of this group.

Error – Detecting codes:

When binary data is transmitted & processed , it is susceptible to noise that can alter or distort its contents. The 1's may get changed to 0's & 0's may get changed to 1's because digital systems must be accurate to the digit, error can pose a problem. Several schemes have been devised to detect the occurrence of a single bit error in a binary word, so that whenever such an error occurs the concerned binary word can be corrected & retransmitted.

ERROR – DETECTING CODES

- When we talk about digital systems, be it a digital computer or a digital communication set-up, the issue of error detection and correction is of great practical significance.
- Errors creep into the bit stream owing to noise or other impairments during the course of its transmission from the transmitter to the receiver.
- While the addition of redundant bits helps in achieving the goal of making transmission of information from one place to another error free or reliable, it also makes it inefficient.

ERROR – DETECTING CODES

Some Common Error Detecting and CorrectingCodes

- Parity Code
- Hamming Code

Parity Code:

- A parity bit is an extra bit added to a string of data bits in order to detect any error that might have crept into it while it was being stored or processed and moved from one place to another in a digital system.

ERROR – DETECTING CODES

The parity bit can be set to 0 and 1 depending on the type of the parity required.

- For even parity, this bit is set to 1 or 0 such that the no. of "1 bits" in the entire word is even. Shown in fig. (a).
- For odd parity, this bit is set to 1 or 0 such that the no. of "1 bits" in the entire word is odd. Shown in fig. (b).

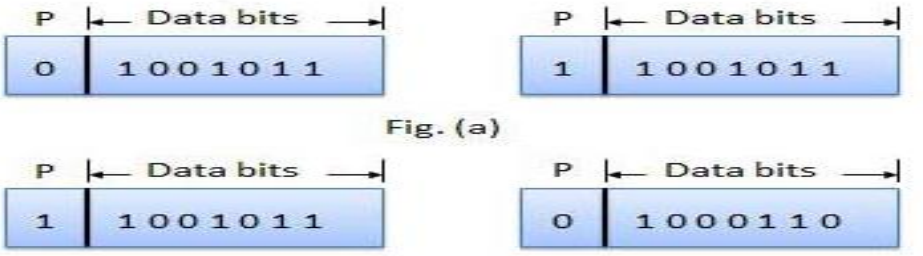


fig. (a) fig. (b)

ERROR – DETECTING CODES

Hamming Code:

- An increase in the number of redundant bits added to message bits can enhance the capability of the code to detect and correct errors.
- If sufficient number of redundant bits arranged such that different error bits produce different error results, then it should be possible not only to detect the error bit but also to identify its location.
- In fact, the addition of redundant bits alters the 'distance' code parameter.

ERROR – DETECTING CODES

- For example, the addition of single-bit parity results in a code with a Hamming distance of at least 2.
- The smallest Hamming distance in the case of a threefold repetition code would be 3.
- Hamming noticed that an increase in distance enhanced the code's ability to detect and correct errors.
- Hamming's code was therefore an attempt at increasing the Hamming distance and at the same time having as high an information throughput rate as possible.

ERROR – DETECTING CODES

- The generalized form of code is $P_1P_2D_1P_3D_2D_3D_4P_4D_5D_6D_7D_8D_9D_{10}D_{11}P_5$…..,where P and D respectively represent parity and data bits.
- We can see from the generalized form of the code that all bit positions that are powers of 2 (positions 1, 2, 4, 8, 16 ...) are used as parity bits.All other bit positions (positions 3, 5, 6, 7, 9, 10, 11 ...) are used to encode data.
- Each parity bit is allotted a group of bits from the data bits in the code word, and the value of the parity bit (0 or 1) is used to give it certain parity.

ERROR – DETECTING CODES

- Groups are formed by first checking bits and then alternately skipping and checking bits following the parity bit. Here, is the position of the parity bit; 1 for P_1, 2 for P_2, 4 for P_3, 8 for P_4 and so on.
- Now, the position of P_3 is at number 4. In order to form the group, we check the first three bits *N-1=3* and then follow it up by alternately skipping and checking four bits *(N=4)*.

ERROR – DETECTING CODES

- The Hamming code is capable of correcting single-bit errors on messages of any length.
- Although the Hamming code can detect two-bit errors, it cannot give the error locations.
- The number of parity bits required to be transmitted along with the message, however, depends upon the message length.

$$(2^n - n) \geq m$$

- The number of parity bits n required to encode m message bits is the smallest integer that satisfies the condition

- The code word sequence for this code is written as $P_1P_2D_1P_3D_2D_3D_4$, with P_1, P_2 and P_3 being the parity bits and D_1, D_2, D_3 and D_4 being the data bits.

- **Generation of Hamming Code:**

	P_1	P_2	D_1	P_3	D_2	D_3	D_4
Data bits (without parity)			0		1	1	0
Data bits with parity bit P_1	1		0		1		0
Data bits with parity bit P_2		1	0			1	0
Data bits with parity bit P_3				0	1	1	0
Data bits with parity	1	1	0	0	1	1	0

BOOLEAN ALGEBRA

- Binary
  - A1a: X=0 if X=1
  - A1b: X=1 if X=0
- Complement
  - aka *invert, NOT*
  - A2a: if X=0, X'=1
  - A2b: if X=1, X'=0
    - The tick mark ' means complement, invert, or NOT
    - Other symbol for complement: X'= $\overline{X}$

X	X'
0	1
1	0

- AND operation
  - A3a: 0•0=0
  - A4a: 1•1=1
  - A5a: 0•1=1•0=0
    - The dot • means AND
    - Other symbol for AND: X•Y=XY (*no symbol*)

X	Y	X•Y
0	0	0
0	1	0
1	0	0
1	1	1

- OR Operation
  - A3b: 1+1=1
  - A4b: 0+0=0
  - A5b: 1+0=0+1=1
    - The plus + means OR

X	Y	X+Y
0	0	0
0	1	1
1	0	1
1	1	1

BOOLEAN ALGEBRA

- **Variable**: Variables are the different symbols in a Boolean expression
- **Literal:** Each occurrence of a variable or its complement is called a literal

$$\bar{A} + A.B + A.\bar{C} + \bar{A}.B.C$$

- **Term**: A term is the expression formed by literals and operations at one level

BOOLEAN ALGEBRA

- Identity Elements
  - a: X+0=X
  - b: X•1=X
- Commutativity
  - a: X+Y=Y+X
  - b: X•Y=Y•X
- Complements
  - a: X+X'=1
  - b: X•X'=0

OR operation

X	Y	X+0	X+Y	Y+X	X'	X+X'
0	0	0	0	0	1	1
0	1	0	1	1	1	1
1	0	1	1	1	0	1
1	1	1	1	1	0	1

AND operation

X	Y	X•1	X•Y	Y•X	X'	X•X'
0	0	0	0	0	1	0
0	1	0	0	0	1	0
1	0	1	0	0	0	0
1	1	1	1	1	0	0

BOOLEAN ALGEBRA

- Associativity
  - a: (X+Y)+Z=X+(Y+Z)
  - b: (X•Y)•Z=X•(Y•Z)

X	Y	Z	X+Y	(X+Y)+Z	Y+Z	X+(Y+Z)	X•Y	(X•Y)•Z	Y•Z	X•(Y•Z)
0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	1	1	1	0	0	0	0
0	1	0	1	1	1	1	0	0	0	0
0	1	1	1	1	1	1	0	0	1	0
1	0	0	1	1	0	1	0	0	0	0
1	0	1	1	1	1	1	0	0	0	0
1	1	0	1	1	1	1	1	0	0	0
1	1	1	1	1	1	1	1	1	1	1

BOOLEAN ALGEBRA

- Distributivity
  - a: X+(Y•Z) = (X+Y)•(X+Z)
  - b: X•(Y+Z) = (X•Y)+(X•Z)

X	Y	Z	X+Y	X+Z	(X+Y)• (X+Z)	Y•Z	X+ (Y•Z)	X•Y	X•Z	X•Y+ X•Z	Y+Z	X• (Y+Z)
0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0	1	0
0	1	0	1	0	0	0	0	0	0	0	1	0
0	1	1	1	1	1	1	1	0	0	0	1	0
1	0	0	1	1	1	0	1	0	0	0	0	0
1	0	1	1	1	1	0	1	0	1	1	1	1
1	1	0	1	1	1	0	1	1	0	1	1	1
1	1	1	1	1	1	1	1	1	1	1	1	1

BOOLEAN ALGEBRA

- Idempotency
  - a: X+X=X
  - b: X•X=X
- Null elements
  - a: X+1=1
  - b: X•0=0
- Involution
  - a: (X')'=X

		OR	AND						
X	Y	X+Y	X•Y	X+X	X•X	X+1	X•0	X'	X''
0	0	0	0	0	0	1	0	1	0
0	1	1	0	0	0	1	0	1	0
1	0	1	0	1	1	1	0	0	1
1	1	1	1	1	1	1	0	0	1

BOOLEAN ALGEBRA

- Absorption
  - ›a: (X•Y)+(X•Y'•Z)=(X•Y)+(X•Z)
  - › b: (X+Y)•(X+Y'+Z) = (X+Y)•(X+Z)

X Y Z	Y'	XY	XY'Z	(XY)+(XY'Z)	XZ	(XY)+(XZ)	X+Y	X+Y'+Z	(X+Y)•(X+Y'+Z)	X+Z	(X+Y)•(X+Z)
0 0 0	1	0	0	0	0	0	0	1	0	0	0
0 0 1	1	0	0	0	0	0	0	1	0	1	0
0 1 0	0	0	0	0	0	0	1	0	0	0	0
0 1 1	0	0	0	0	0	0	1	1	1	1	1
1 0 0	1	0	0	0	0	0	1	1	1	1	1
1 0 1	1	0	1	1	1	1	1	1	1	1	1
1 1 0	0	1	0	1	0	1	1	1	1	1	1
1 1 1	0	1	0	1	1	1	1	1	1	1	1

BOOLEAN ALGEBRA

- DeMorgan's theorem (very important!)
  - a: $(X+Y)' = X' \bullet Y'$

    $X+Y = X \bullet Y$ break (or connect) the bar & change the sign
  - b: $(X.Y)' = X' + Y'$

    $X \bullet Y = X+Y$ break (or connect) the bar & change the sign

Generalized DeMorgan's theorem:

- GT8a: $(X_1+X_2+\ldots+X_{n-1}+X_n)' = X_1' \bullet X_2' \bullet \ldots \bullet X_{n-1}' \bullet X_n'$

OR AND

X	Y	X+Y	X•Y	X'	Y'	(X+Y)'	X'•Y'	(X•Y)'	X'+Y'
0	0	0	0	1	1	1	1	1	1
0	1	1	0	1	0	0	0	1	1
1	0	1	0	0	1	0	0	1	1
1	1	1	1	0	0	0	0	0	0

BOOLEAN ALGEBRA

- Consensus Theorem
  - a: (X•Y)+(X'•Z)+(Y•Z) = (X•Y)+(X'•Z)
  - b: (X+Y)•(X'+Z)•(Y+Z) = (X+Y)•(X'+Z)

X	Y	Z	X'	XY	X'Z	YZ	(XY)+(X'Z)+(YZ)	(XY)+(X'Z)	X+Y	X'+Z	Y+Z	(X+Y)•(X'+Z)•(Y+Z)	(X+Y)•(X'+Z)
0	0	0	1	0	0	0	0	0	0	1	0	0	0
0	0	1	1	0	1	0	1	1	0	1	1	0	0
0	1	0	1	0	0	0	0	0	1	1	1	1	1
0	1	1	1	0	1	1	1	1	1	1	1	1	1
1	0	0	0	0	0	0	0	0	1	0	0	0	0
1	0	1	0	0	0	0	0	0	1	1	1	1	1
1	1	0	0	1	0	0	1	1	1	0	1	0	0
1	1	1	0	1	0	1	1	1	1	1	1	1	1

SWITCHING FUNCTIONS

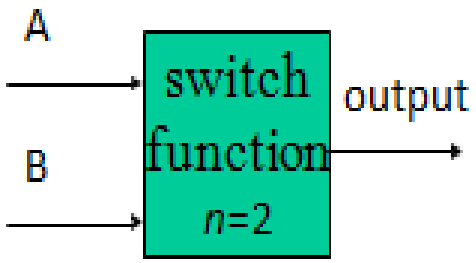
- For n variables, there are 2^n possible combinations of Values from all 0s to all 1s
- There are 2 possible values for the output of a function of a combination of values of n variables i.e. 0 and 1
- There are 2^{2^n} different switching functions for n variables
- n=0 (no inputs) $2^{2^n} = 2^{2^0} = 2^1 = 2$

  Output can be either 0 or 1
- n=1 (1 input, A) $2^{2^n} = 2^{2^1} = 2^2 = 4$

  Output can be 0, 1, A, or A'

SWITCHING FUNCTIONS EXAMPLE

- n=2 (2 inputs, A and B) ⇨ $2^{2^n} = 2^{2^2} = 2^4 = 16$



AB	f_0	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}
00	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
01	0	0	1	1	0	1	1	0	0	1	1	0	0	1	1	
10	0	0	0	0	1	1	1	0	0	0	0	1	1	1	1	
11	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	

$f_1 = A'B' = (A+B)'$

$f_2 = A'B$

$f_3 = A'B'+A'B = A'(B'+B) = A'$

CANONICAL AND STANDERED FORMS

Logical functions are generally expressed in terms of different combinations of logical variables with their true forms as well as the complement forms. Binary logic values obtained by the logical functions and logic variables are in binary form. An arbitrary logic function can be expressed in the following forms.

- Sum of the Products (SOP)
- Product of the Sums (POS)

CANONICAL AND STANDERED FORMS

- **Product Term:** In Boolean algebra, the logical product of several variables on which a function depends is considered to be a product term. In other words, the AND function is referred to as a product term or standard product.
- **Sum Term:** An OR function is referred to as a sum term
- **Sum of Products (SOP):** The logical sum of two or more logical producterms is referred to as a sum of products expression

$$Y = AB + BC + AC$$

- **Product of Sums (POS):** Similarly, the logical product of two or morelogical sum terms is called a product of sums expression

CANONICAL AND STANDERED FORMS

- **Standard form:** The standard form of the Boolean function is when it is expressed in sum of the products or product of the sums fashion $Y = AB + BC + AC$
- **Nonstandard Form:** Boolean functions are also sometimes expressed in nonstandard forms like $F = (AB + CD)(\bar{A}\bar{B} + \bar{C}\bar{D})$, which is neither a sum of products form nor a product of sums form.
- **Minterm**: A product term containing all n variables of the function in either true or complemented form is called the minterm. Each minterm is obtained by an AND operation of the variables in their true form or complemented form.

CANONICAL AND STANDERED FORMS

- **Maxterm:** A sum term containing all n variables of the function in either true or complemented form is called the Maxterm. Each Maxterm is obtained by an OR operation of the variables in their true form or complemented form.
- The canonical sum of products form of a logic function can be obtained by using the following procedure
- Check each term in the given logic function. Retain if it is a minterm, continue to examine the next term in the same manner.
- Multiply all the products and discard the redundant terms.

CANONICAL AND STANDERED FORMS

- **Example:** Obtain the canonical sum of product form of the following function $F(A,B,C) = A + BC$

- Solution:

$$\begin{aligned} F(A,B,C) &= A + BC \\ &= A(B + \bar{B})(C + \bar{C}) + BC(A + \bar{A}) \\ &= (AB + A\bar{B})(C + \bar{C}) + ABC + \bar{A}BC \\ &= ABC + A\bar{B}C + AB\bar{C} + A\bar{B}\bar{C} + ABC + \bar{A}BC \end{aligned}$$

$$= ABC + A\bar{B}C + AB\bar{C} + A\bar{B}\bar{C} + \bar{A}BC \; (as\; ABC + ABC = ABC)$$

- Hence the canonical sum of the product expression of the given function is

$$F(A,B,C) = ABC + A\bar{B}C + AB\bar{C} + A\bar{B}\bar{C} + \bar{A}BC$$

CANONICAL AND STANDERED FORMS

The product of sums form is a method (or form) of simplifying the Boolean expressions of logic gates. In this POS form, all the variables are ORed, i.e. written as sums to form sum terms. All these sum terms are ANDed (multiplied) together to get the product-of-sum form. This form is exactly opposite to the SOP form. So this can also be said as —Dualof SOP form .

(A+B) * (A + B + C) * (C +D)

POS form can be obtained by

- Writing an OR term for each input combination, which produces LOW

ALGEBRAIC SIMPLIFICATION

Minimize the following Boolean expression using Boolean identities –

F(A,B,C)=(A+B)(A+C)F(A,B,C)=(A+B)(A+C)Solution

Given, F(A,B,C)=(A+B)(A+C)

F (A,B,C)=A.A+A.C+B.A+B.C [Applying distributive Rule]

F(A,B,C)=A+A.C+B.A+B.C [Applying Idempotent Law]

F(A,B,C)=A(1+C)+B.A+B.C [Applying distributive Law]

F(A,B,C)=A+B.A+B.C [Applying dominance

F(A,B,C)=(A+1).A+B.C [Applying distributive Law]

F(A,B,C)=1.A+B.C [Applying dominance Law]

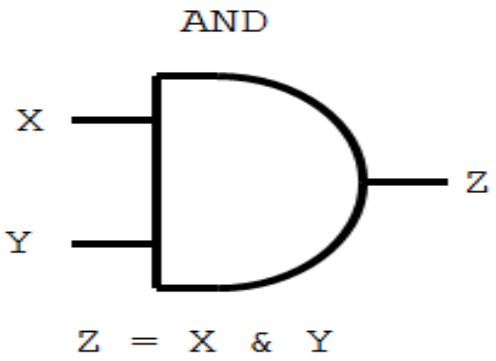
F(A,B,C)=A+B.C [Applying dominance Law]

So, F(A,B,C)=A+BCF(A,B,C)=A+BC is the minimized form.

DIGITAL LOGIC GATES

AND GATE:

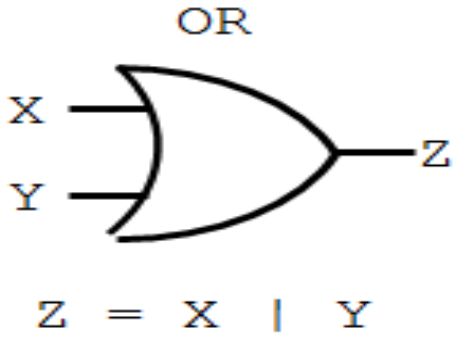
Z=A.B



X	Y	Z
0	0	0
0	1	0
1	0	0
1	1	1

OR GATE:

Z=A+B

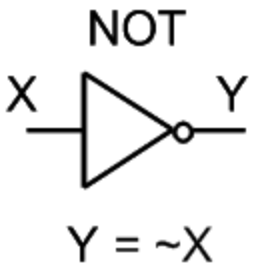


X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	1

DIGITAL LOGIC GATES

NOT GATE:

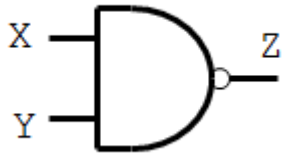
Z=A’



X	Y
0	1
1	0

NAND GATE:

$Z=\overline{A.B}$

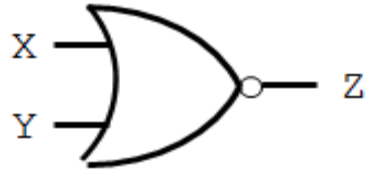


X	Y	Z
0	0	1
0	1	1
1	0	1
1	1	0

DIGITAL LOGIC GATES

NOR GATE:

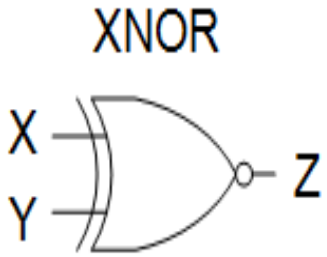
$Z=\overline{A+B}$



X	Y	Z
0	0	1
0	1	0
1	0	0
1	1	0

Ex-OR GATE:

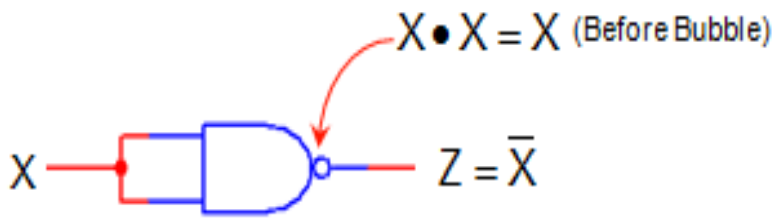
$Z=A \oplus B$



X	Y	Z
0	0	1
0	1	0
1	0	0
1	1	1

MULTILEVEL NAND-NOR REALIZATION

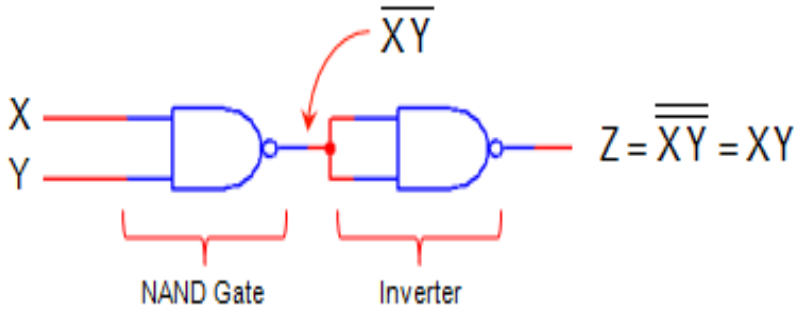
NAND Gate as an Inverter Gate



X	Z
0	1
1	0

Equivalent to Inverter

NAND Gate as an AND Gate

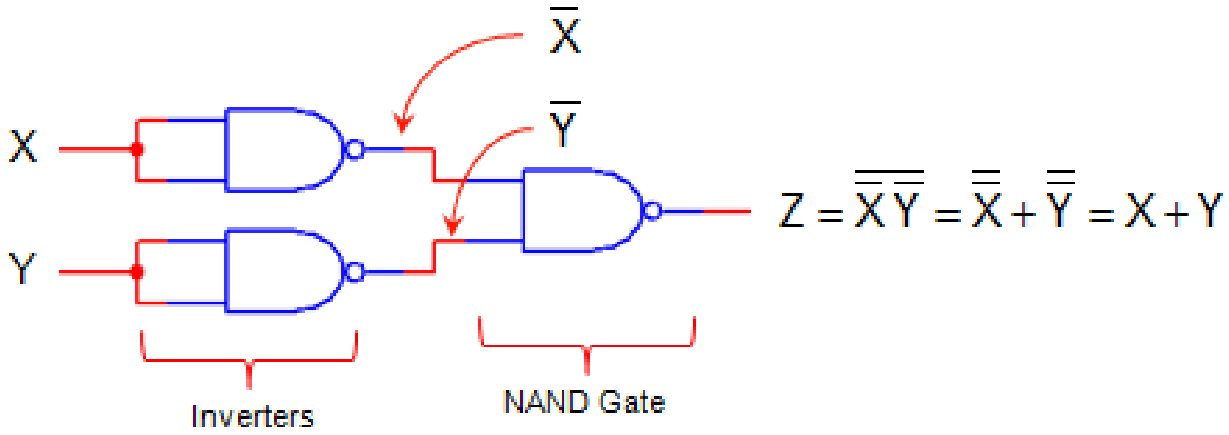


X	Y	Z
0	0	0
0	1	0
1	0	0
1	1	1

Equivalent to AND Gate

MULTILEVEL NAND-NOR REALIZATION

NAND Gate as an OR Gate



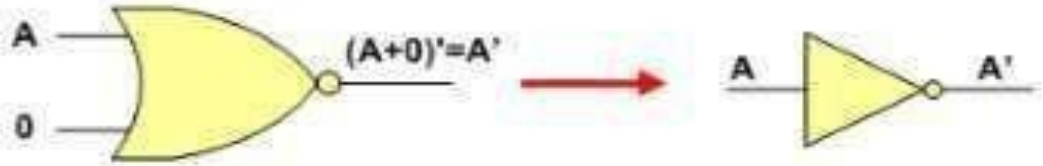
X	Y	Z
0	0	0
0	1	1
1	0	1
1	1	1

Equivalent to OR Gate

MULTILEVEL NAND-NOR REALIZATION

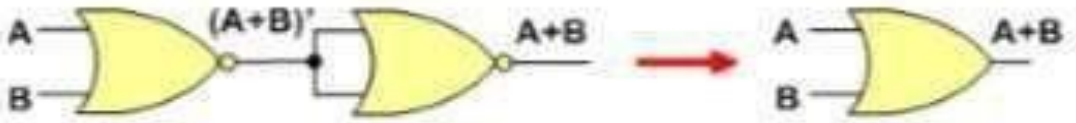
1. All NOR input pins connect to the input signal **A** gives an output **A'**.

2. One NOR input pin is connected to the input signal **A** while all other input pins are connected to logic **0**. The output will be **A'**.



Implementing OR Using only NOR Gates

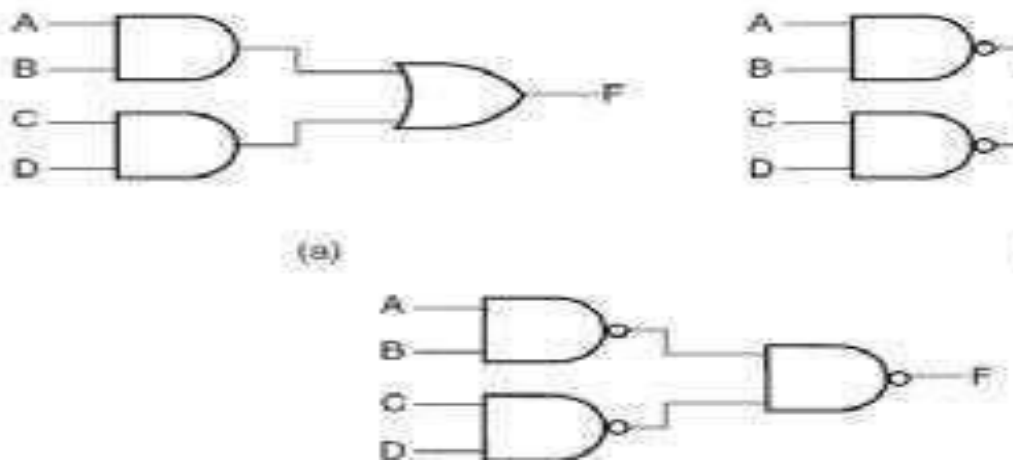
An OR gate can be replaced by NOR gates as shown in the figure (The OR is replaced by a NOR gate with its output complemented by a NOR gate inverter)



MULTILEVEL NAND-NOR REALIZATION

Example1: implement the following function $F = AB + CD$

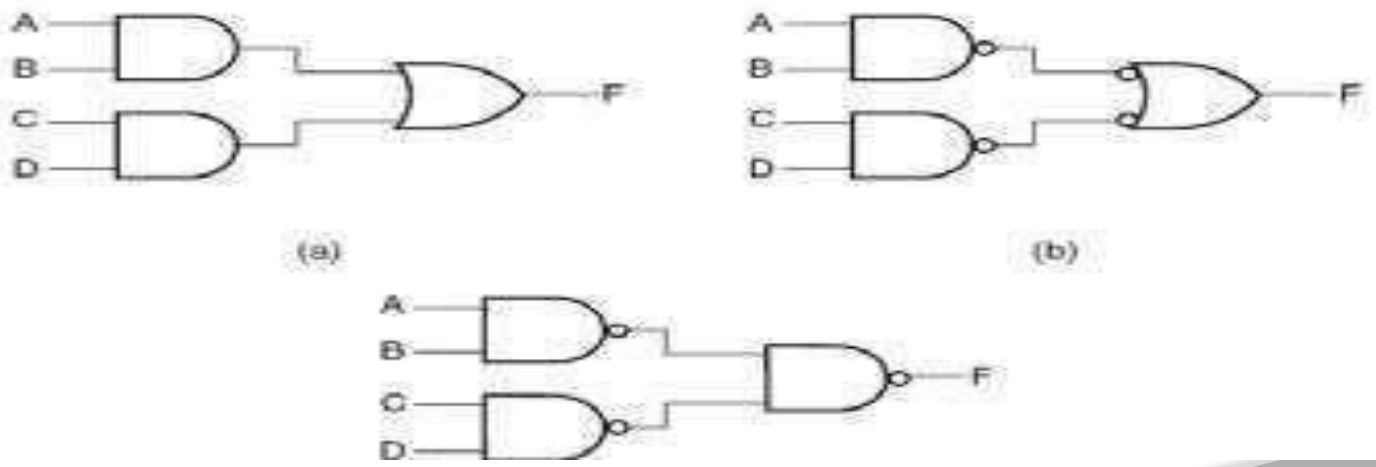
- The implementation of Boolean functions with NAND gates requires that the functions be in sum of products (SOP) form.
- This function can be implemented by three steps.



MULTILEVEL NAND-NOR REALIZATION

Example1: implement the following function $F = AB + CD$

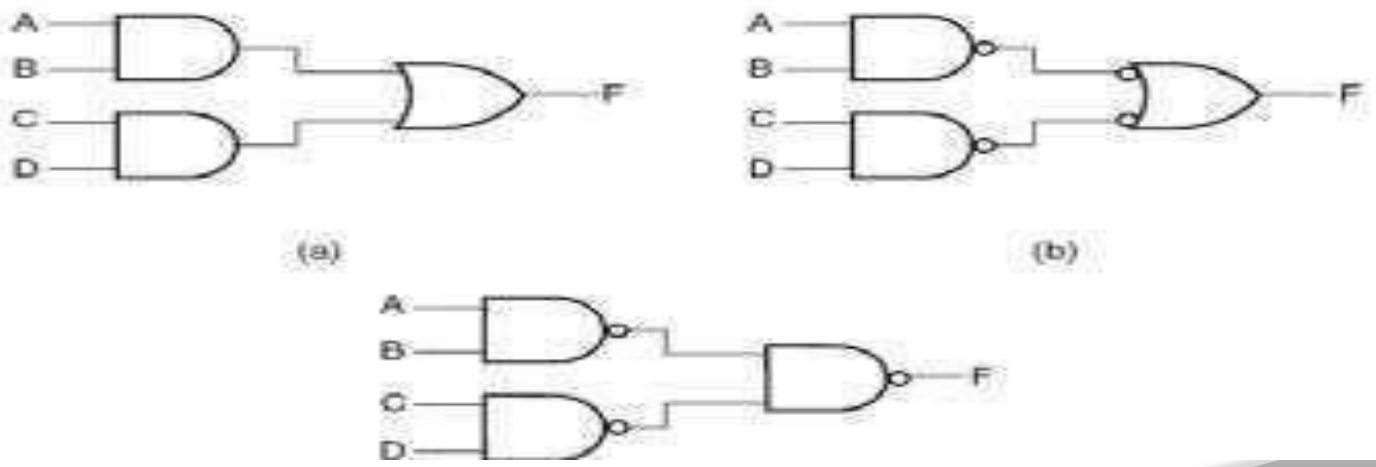
- The implementation of Boolean functions with NAND gates requires that the functions be in sum of products (SOP) form.



MULTILEVEL NAND-NOR REALIZATION

Example1: implement the following function $F = AB + CD$

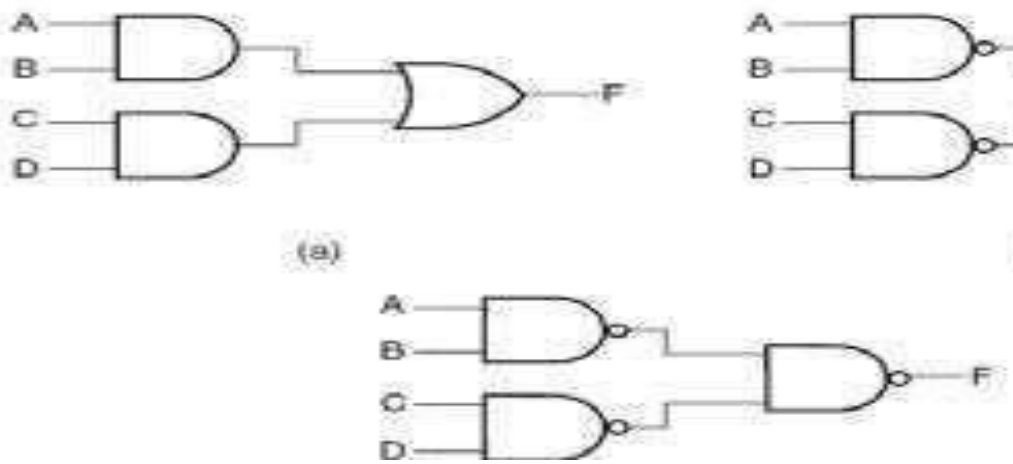
- The implementation of Boolean functions with NAND gates requires that the functions be in sum of products (SOP) form.



MULTILEVEL NAND-NOR REALIZATION

Example1: implement the following function $F = AB + CD$

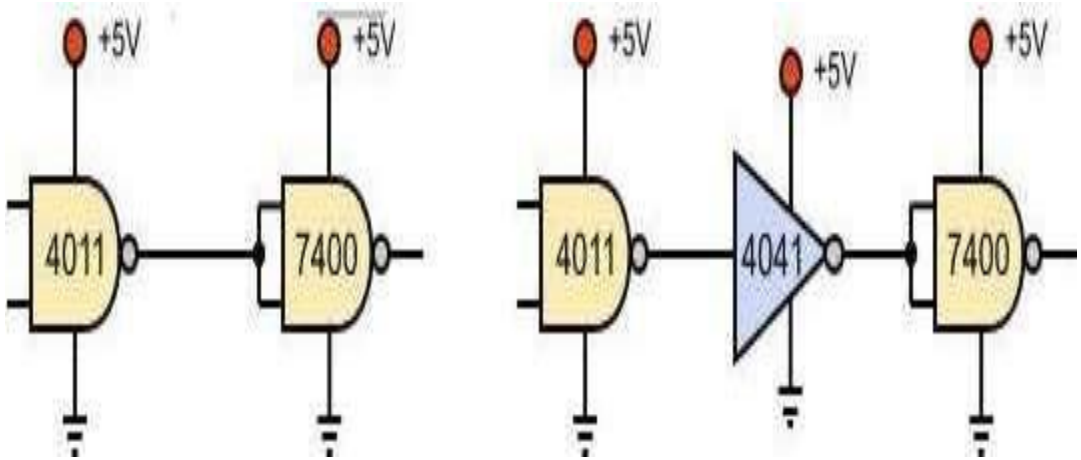
- The implementation of Boolean functions with NAND gates requires that the functions be in sum of products (SOP) form.
- This function can be implemented by three steps.



Logic delay, TTL/CMOS interfacing

CMOS DRIVING TTL AND CMOS DRIVING TTL

Interfacing a CMOS to a TTL under 5Volts power supply



Unit-II

COMBINATIONAL DIGITAL CIRCUITS

KARANAUGHMAP

- Boolean algebra helps us simplify expressions andcircuits
- Karnaugh Map: A graphical technique for simplifying aBooleanexpression into eitherform:
  - minimal sum of products(MSP)
  - minimal product of sums(MPS)
- Goal of thesimplification.
  - There are a minimal number of product/sumterms
  - Each term has a minimal number ofliterals

KARANAUGH MAP

- A two-variable function has four possible minterms. We can re- arrange
  these minterms into a Karnaugh map

x	y	minterm
0	0	x'y'
0	1	x'y
1	0	xy'
1	1	xy

x \ y	0	1
0	x'y'	x'y
1	xy'	xy

- Now we can easily see which minterms contain commonliterals
  - Minterms on the left and right sides contain y' and y respectively

x \ y	0	1
0	x'y'	x'y
1	xy'	xy

x \ y	y'	y
x'	x'y'	x'y
x	xy'	xy

nd x

KARANAUGH MAP

- Make as few rectangles as possible, to minimize the number of products in the final expression.
- Make each rectangle as large as possible, to minimize the number of literals in each term.
- Rectangles can be overlapped, if that makes them larger
- The most difficult step is grouping together all the 1s in the K-map
  - Make rectangles around groups of one, two, four or eight 1s
  - All of the 1s in the map should beincluded inat least one rectangle. Do not include any of the 0s
  - Each group corresponds to one product term

3 VARIABLE K-MAP

- Maxterms are grouped to find minimal PoS

expression

x \ yz	00	01	11	10
0	x +y+z	x+y+z′	x+y′+z′	x+y′+z
1	x′ +y+z	x′+y+z′	x′+y′+z′	x′+y′+z

3 VARIABLE K-MAP

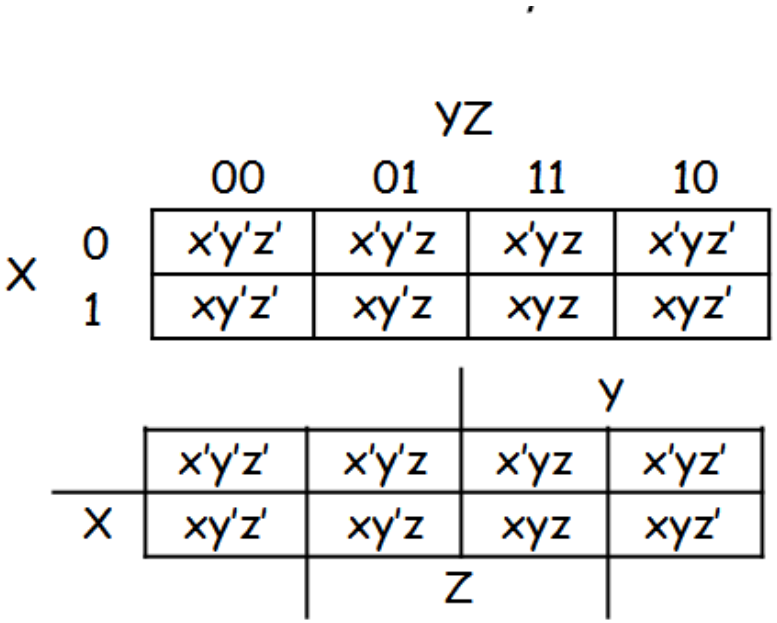
- Let's consider simplifying f(x,y,z) = xy + y'z + xz
- You should convert the expression into a sum of mintermsform,
  - The easiest way to do this is to make a truth table forthe function, and then read off the minterms
  - You can either write out the literals or use the minterm shorthand
  - Here is the truth table and sum of minterms for our

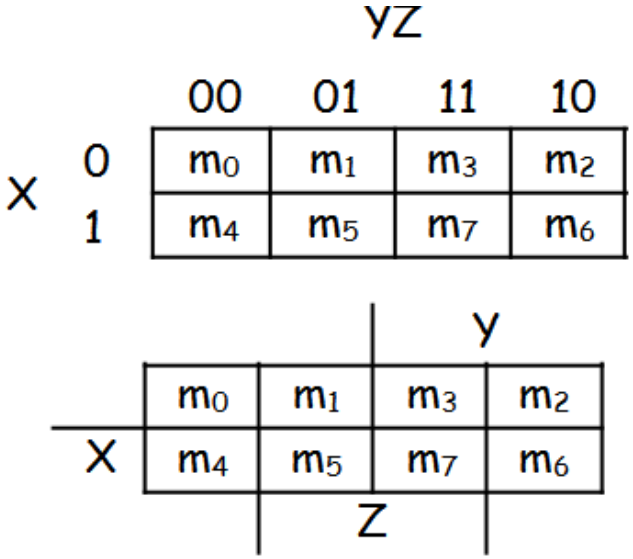
x	y	z	f(x,y,z)
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

$$f(x,y,z) = x'y'z + xy'z + xyz' + xyz$$
$$= m_1 + m_5 + m_6 + m_7$$

3 VARIABLE K-MAP

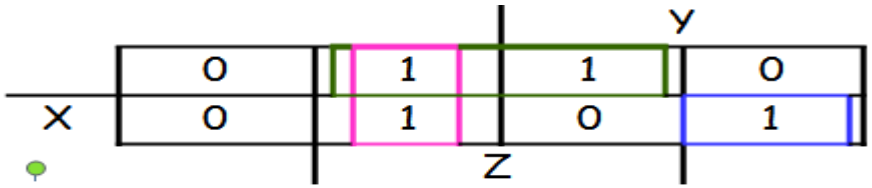
- For a three-variable expression with inputs x, y, z, the arrangement of





3 VARIABLE K-MAP

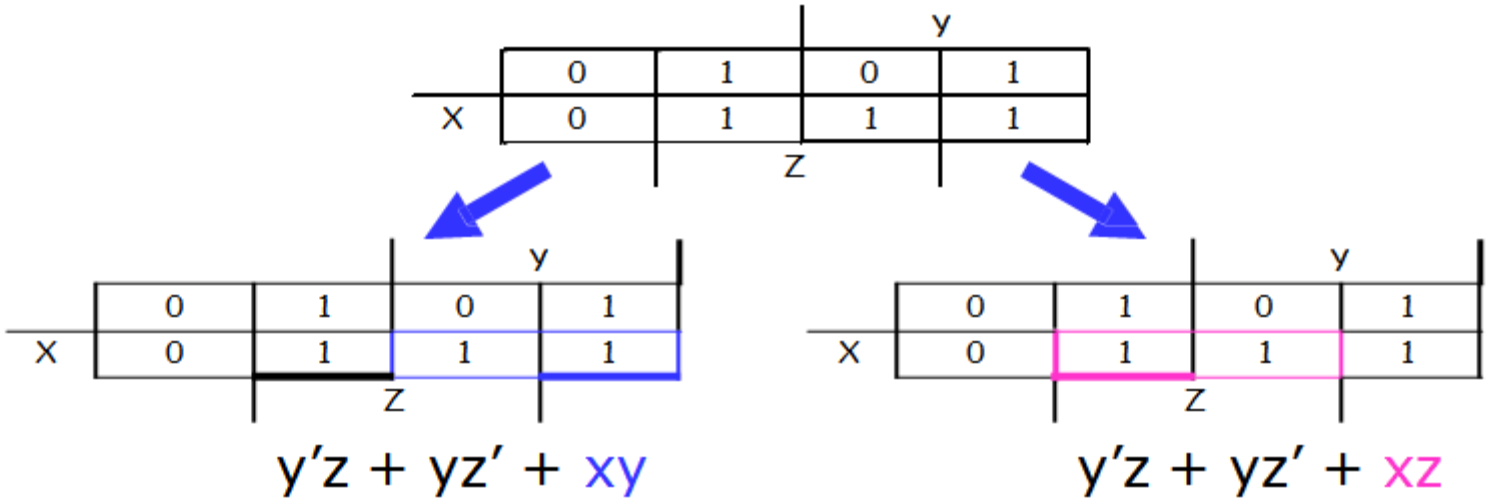
- Here is the filled in K-map, with all groups shown
  - The magenta and green groups overlap, which makes each of them as large as possible
  - Minterm m_6 is in a group all by its lonesome



- The final MSP here is x'z + y'z +xyz'

3 VARIABLE K-MAP

- There may not necessarily be a *unique* MSP. The K-map below yields
  two
  valid and equivalent MSPs, because there are two possible waysto



Remember that overlapping groups is possible, as shown above

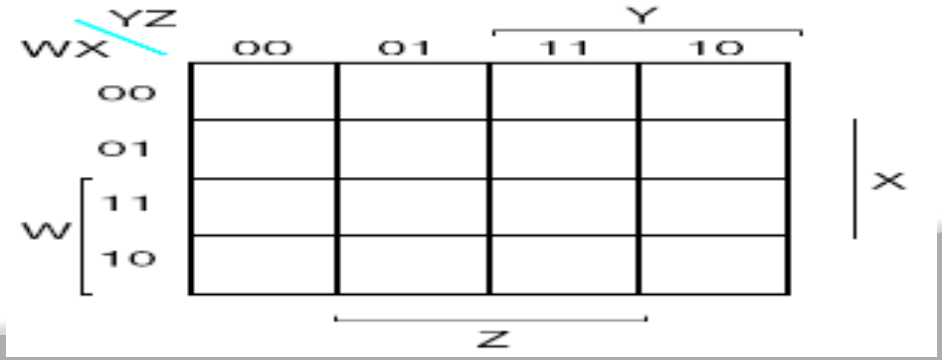
3 VARIABLE K-MAP

- Maxterms are grouped to find minimal PoS expression

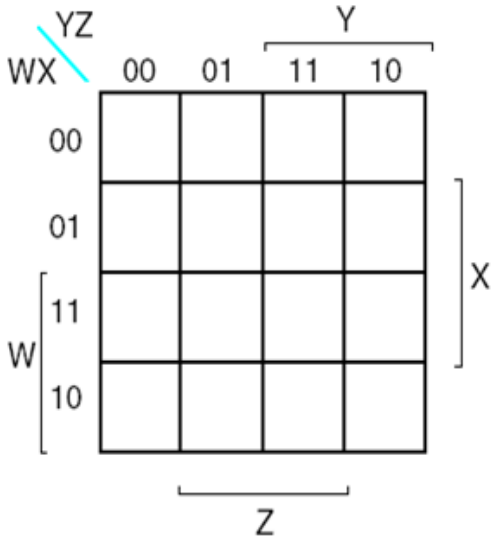
x \ yz	00	01	11	10
0	x+y+z	x+y+z′	x+y′+z′	x+y′+z
1	x′+y+z	x′+y+z′	x′+y′+z′	x′+y′+z

4-VARIABLE K-MAP

- We can do four-variable expressions too!
  - The minterms in the third and fourth columns, *and* in the third and fourth rows, are switched around.
  - Again, this ensures that adjacent squares have common literals
- Grouping minterms is similar to the three-variable case, but:
  - You can have rectangular groups of 1, 2, 4, 8 or 16 minterms
  - You can wrap around all four sides



4-VARIABLE K-MAP



			y	
	w'x'y'z'	w'x'y'z	w'x'yz	w'x'yz'
	w'xy'z'	w'xy'z	w'xyz	w'xyz'
W	wxy'z'	wxy'z	wxyz	wxyz'
	wx'y'z'	wx'y'z	wx'yz	wx'yz'
		Z		

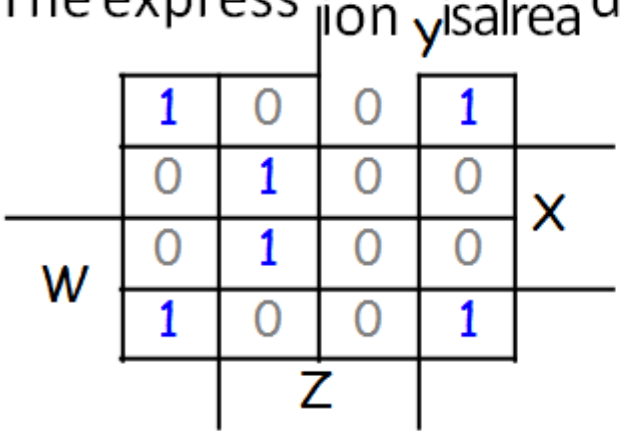
X

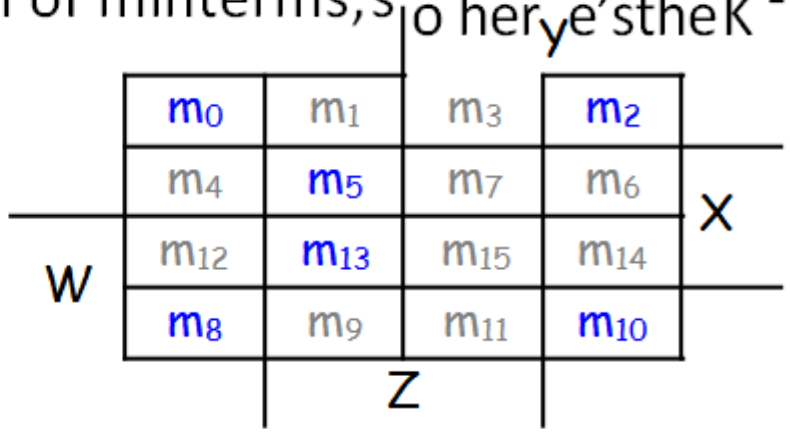
			y	
	m_0	m_1	m_3	m_2
	m_4	m_5	m_7	m_6
W	m_{12}	m_{13}	m_{15}	m_{14}
	m_8	m_9	m_{11}	m_{10}
		Z		

X

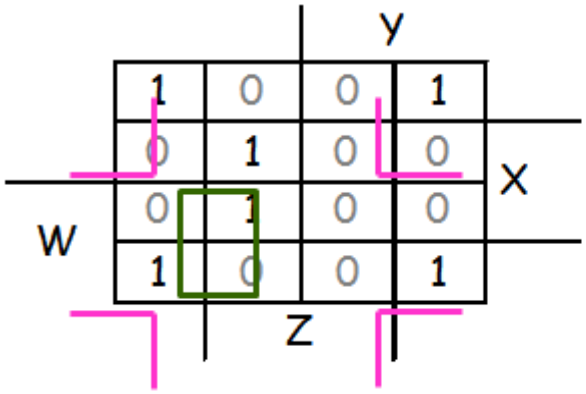
4-VARIABLE K-MAP

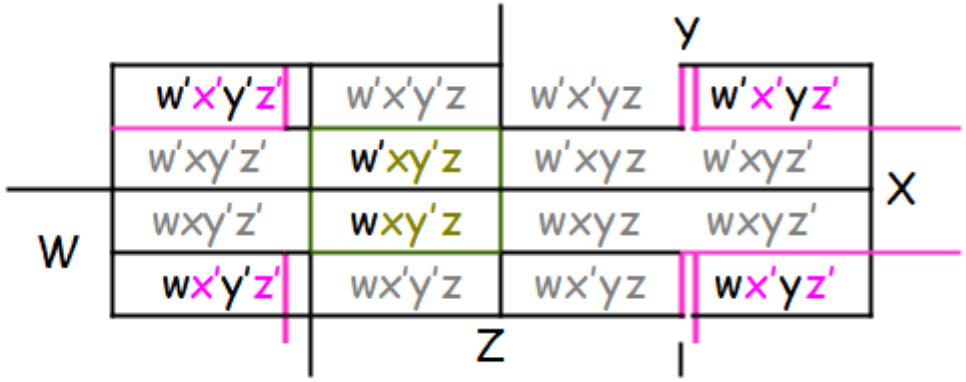
- The expression is already a sum of minterms, so here's the K-map:





- We can make the following groups, resulting in the MSP x'z' + xy'z

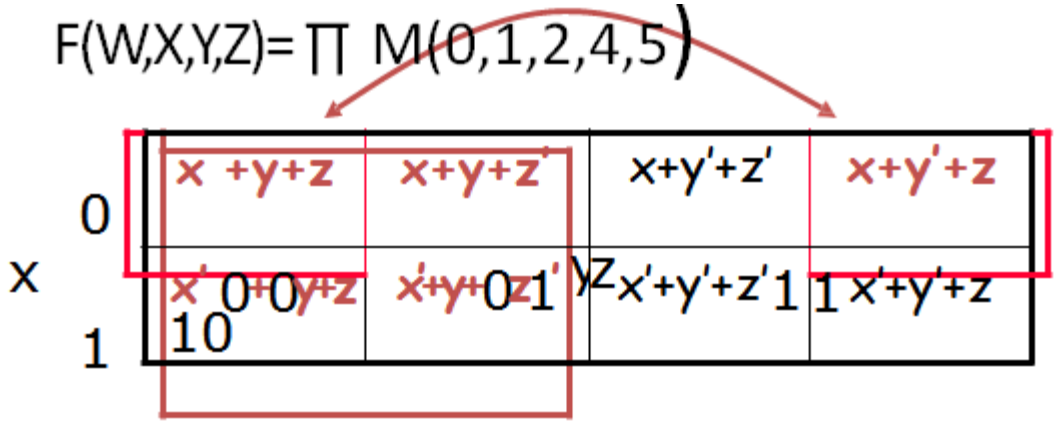




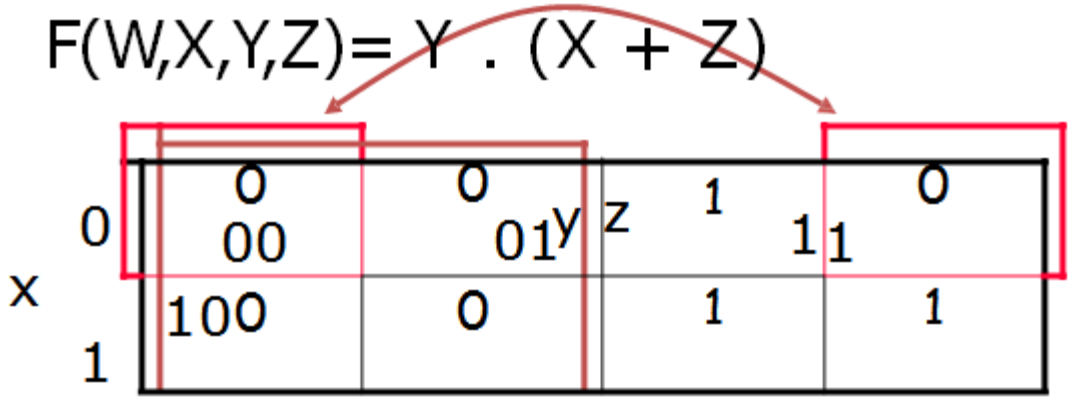
Example: Simplify $m_0+m_2+m_5+m_8+m_{10}+m_{13}$

4-VARIABLE K-MAP

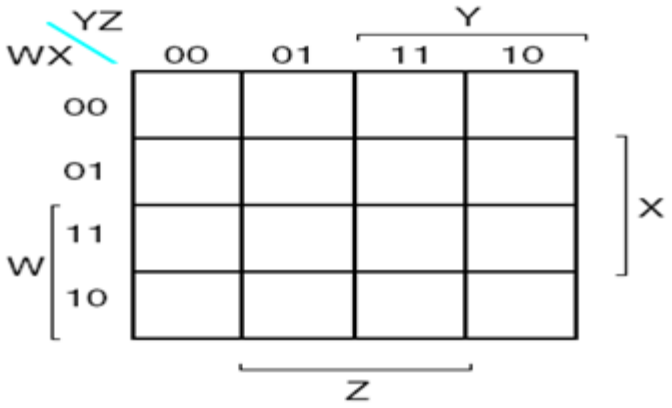
- $F(W,X,Y,Z) = \prod M(0,1,2,4,5)$

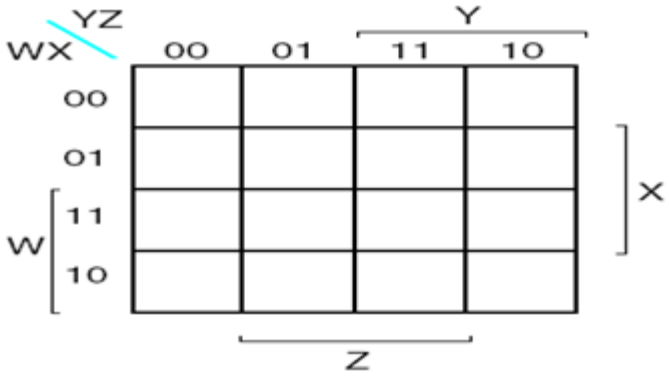


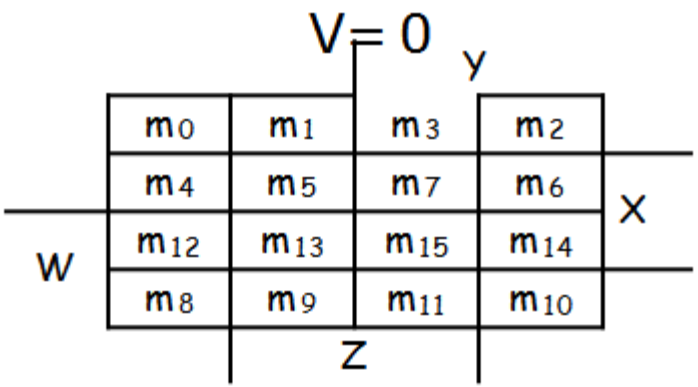
$F(W,X,Y,Z) = Y \cdot (X + Z)$

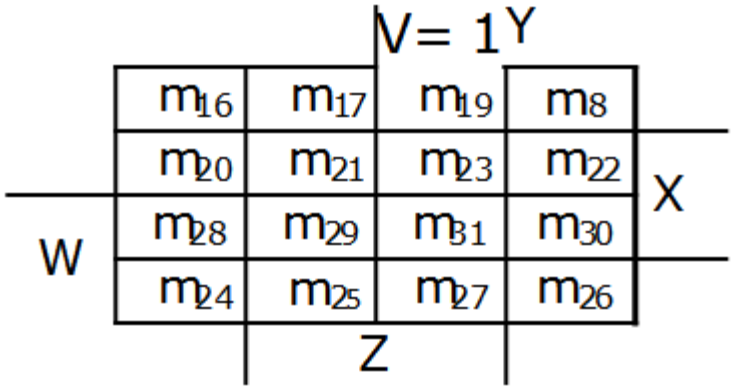


5-VARIABLE K-MAP









5-VARIABLE K-MAP

- In our example, we can write f(x,y,z) in two equivalent ways

$f(x,y,z) = x'y'z + xy'z + xyz' + xyz$

X \ YZ	x'y'z'	x'y'z	x'yz	x'yz'
	xy'z'	xy'z	xyz	xyz'

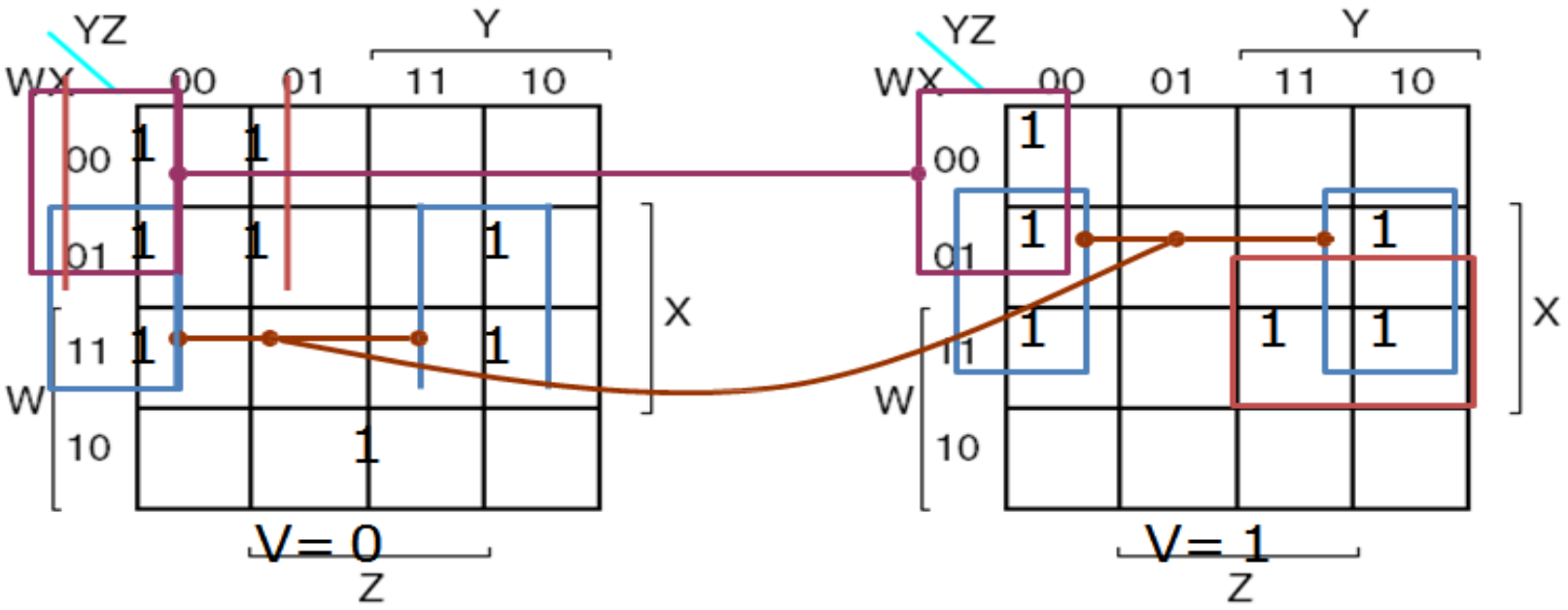
$f(x,y,z) = m_1 + m_5 + m_6 + m_7$

X \ YZ	m_0	m_1	m_3	m_2
	m_4	m_5	m_7	m_6

- In either case, the resulting K-map is shown below

X \ YZ	0	1	0	0
	0	1	1	1

5-VARIABLE K-MAP



f = XZ′

Σm(4,6,12,14,20,22,28,30)

+ V′W′Y′ Σm(0,1,4,5)

+ W′Y′Z′ Σm(0,4,16,20)

+ VWXY Σm(30,31)

+ V′WX′YZ m11

DON'T CARE CONDITION

- You don't always need all 2^n input combinations in ann-variable function
  - If you can guarantee that certain input combinations never occur
  - If some outputs aren't used in the rest of the circuit

x	y	z	f(x,y,z)
0	0	0	0
0	0	1	1
0	1	0	X
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	X
1	1	1	1

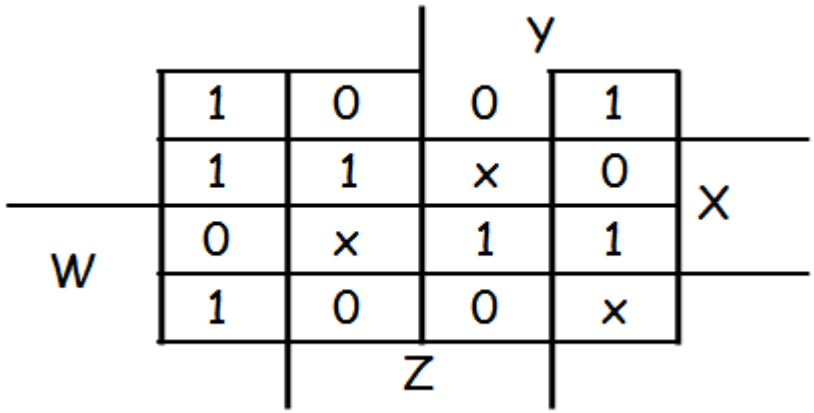
- We mark don't-care outputs in truth tables and K-maps with Xs.

DON'T CARE CONDITION

- Find a MSP for

$$f(w,x,y,z) = \sum m(0,2,4,5,8,14,15),\ d(w,x,y,z) = \sum m(7,10,13)$$

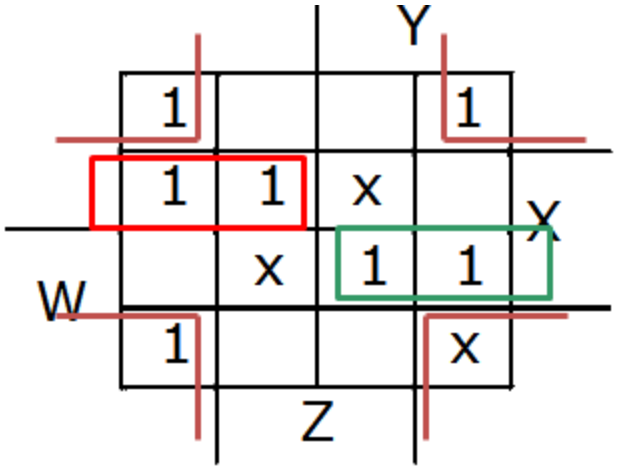
This notation means that input combinations wxyz = 0111, 1010 and 1101(corresponding to minterms m_7, m_{10} and m_{13}) are unused.



DON'T CARE CONDITION

- Find a MSP for:

$$f(w,x,y,z) = \sum m(0,2,4,5,8,14,15),\ d(w,x,y,z) = \sum m(7,10,13)$$



$$f(w,x,y,z) = x'z' + w'xy' + wxy$$

COMBINATIONAL CIRCUITS

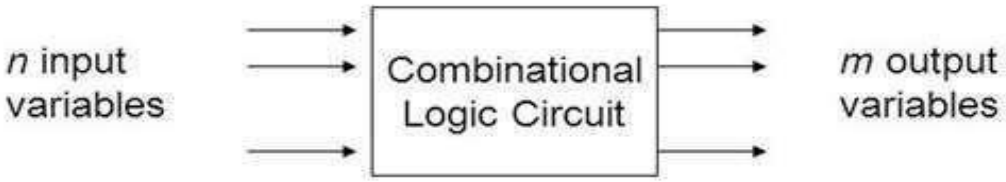
- Combinational circuit is a circuit in which we combine the different gates in the circuit, for example encoder, decoder, multiplexer and demultiplexer.

  Some of the characteristics of combinational circuits are:

- The output of combinational circuit at any instant of time, depends
  only on the levels present at inputterminals.
- The combinational circuit do not use any memory. The previous state
  of input does not have any effect on the present state of the circuit.
- A combinational circuit can have an n number of inputs and m number of outputs.

COMBINATIONAL CIRCUITS

- **Block diagram:**
  possible combinations of input values.



- Specific functions : of combinational circuits
- Adders, subtractors , multiplexers, comparators , encoder, Decoder. MSI Circuits and standard cells

ANALYSIS PROCEDURE

Analysis procedure

To obtain the output Boolean functions from a logic diagram, proceed as follows:

1. Label all gate outputs that are a function of input variables with arbitrary symbols. Determine the Boolean functions for eachgate output.
2. Label the gates that are a function of input variables and previously labeled gates with other arbitrary symbols. Find the Boolean functions for these gates.
3. Repeat the process outlined in step 2 until the outputs of the circuit are obtained.

DESIGN PROCEDURE

Design Procedure

1. The problem is stated
2. The number of available input variables and required output variables is determined.
3. The input and output variables are assigned letter symbols.
4. The truth table that defines the required relationship between inputs and outputs is derived.
5. The simplified Boolean function for each output is obtained.
6. The logic diagram is drawn.

ADDERS

Half Adder

A Half Adder is a combinational circuit with two binary inputs (augends 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.

Fig 1:Block diagram

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

Fig 2:Truth table

$Sum = A'B + AB' = A \oplus B$

$Carry = AB$

BINARY SUBTRACTORS

Full subtractor

The full subtractor perform subtraction of three input bits: the minuend , subtrahend , and borrow in and generates two output bits difference and borrow out.

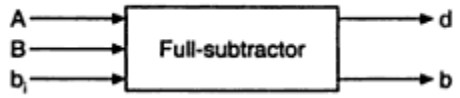


Fig 7:Block diagram

Inputs			Difference	Borrow
A	B	b_i	d	b
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Fig 8: Truth table

$$d = \bar{A}\bar{B}b_i + \bar{A}B\bar{b}_i + A\bar{B}\bar{b}_i + ABb_i = A \oplus B \oplus b_i$$

$$b = \bar{A}\bar{B}b_i + \bar{A}B\bar{b}_i + \bar{A}Bb_i + ABb_i = \bar{A}B + \overline{(A \oplus B)}b_i$$

PARALLEL ADDER AND SUBTRACTOR

A binary parallel adder is a digital circuit that adds two binary numbers in parallel form and produces the arithmetic sum of those numbers in parallel form

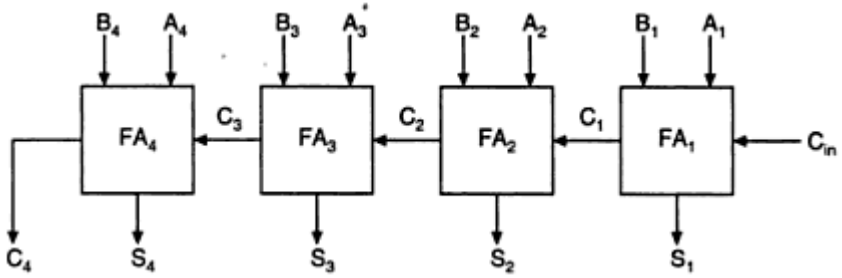


Fig 9:parallel adder

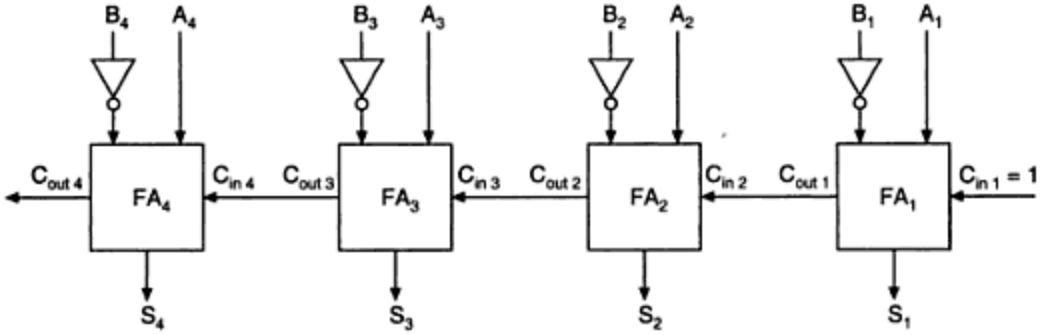


Fig 10:parallel subtractor

CARRY LOOK-A- HEAD ADDER

- In parallel-adder , the speed with which an addition can be performed is governed by the time required for the carries to propagate or ripple through all of the stages of the adder.

- The look-ahead carry adder speeds up the process by eliminating
this ripple carry delay.

$$S_n = P_n \oplus C_n \text{ where } P_n = A_n \oplus B_n$$

$$C_{on} = C_{n+1} = G_n + P_n C_n \text{ where } G_n = A_n \cdot B_n$$

CARRY LOOK-A- HEAD ADDER

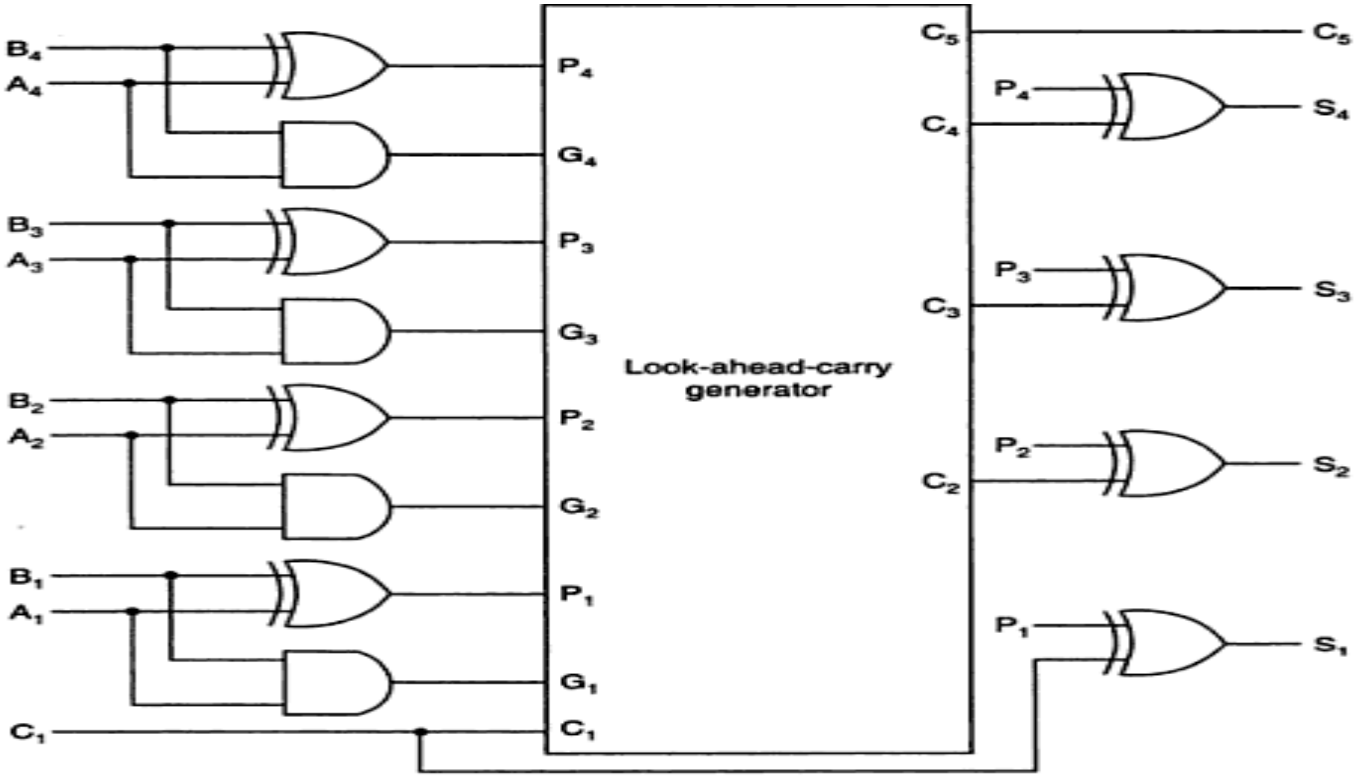


Fig:1 block diagram

BINARY MULTIPLIER

A binary multiplier is an electronic circuit used in digital electronics, such as a computer, to multiply two binary numbers. It is built using binary adders.

Example: (101 x 011)

Partial products are: 101 × 1, 101 × 1, and 101 × 0

$$
\begin{array}{cccccc}
 & & & 1 & 0 & 1 \\
 & & \times & 0 & 1 & 1 \\
\hline
 & & & 1 & 0 & 1 \\
 & & 1 & 0 & 1 & \\
 & 0 & 0 & 0 & & \\
\hline
0 & 0 & 1 & 1 & 1 & 1
\end{array}
$$

BINARY MULTIPLIER

- We can also make an $n \times m$ "block" multiplier and use that to form partial products.
- Example: 2 × 2 – The logic equations for each partial-product binary digit are shown below
- We need to "add" the columns to get the product bits P0, P1, P2, and P3.

		b_1	b_0
		a_1	a_0
		$(a_0 \cdot b_1)$	$(a_0 \cdot b_0)$
+	$(a_1 \cdot b_1)$	$(a_1 \cdot b_0)$	
P_3	P_2	P_1	P_0

BINARY MULTIPLIER

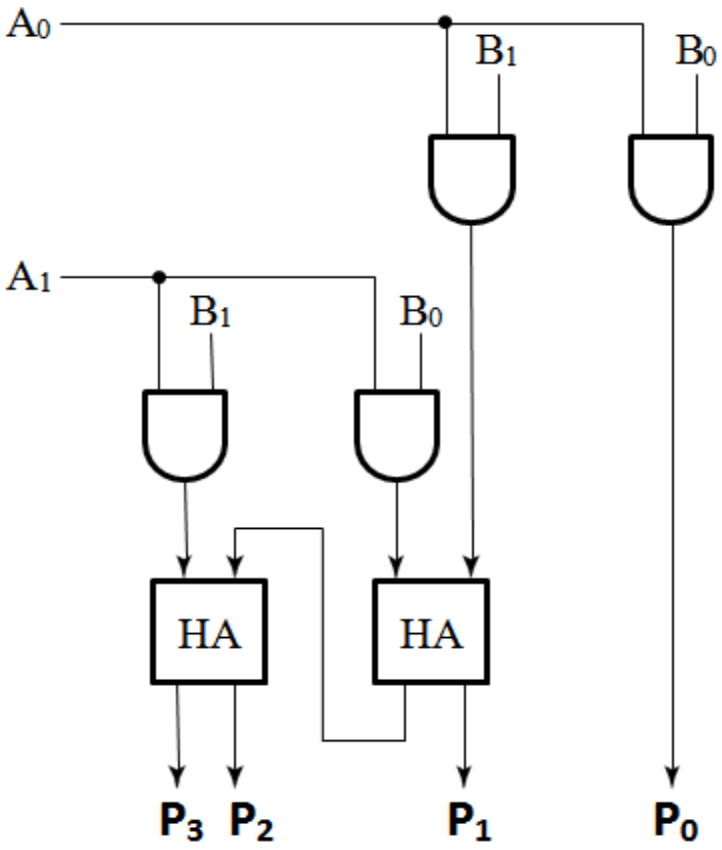


Fig 1: 2 x 2 multiplier array

MAGNITUDE COMPARATOR

Magnitude comparator takes two numbers as input in binary form and determines whether one number is greater than, less than or equal to the other number.

1-Bit Magnitude Comparator

A comparator used to compare two bits is called a single bit comparator.

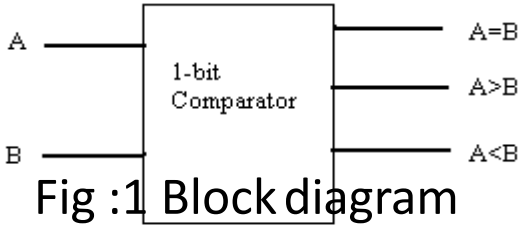


Fig :1 Block diagram

MAGNITUDE COMPARATOR

Inputs		*Outputs*		
A	B	$A > B$	$A = B$	$A < B$
0	0	0	1	0
0	1	0	0	1
1	0	1	0	0
1	1	0	1	0

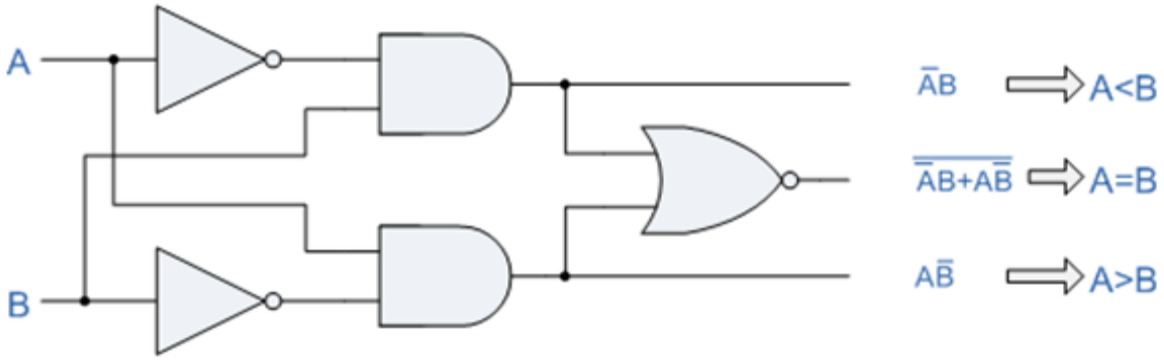


Fig 2:Logic diagram of 1-bit comparator

MAGNITUDE COMPARATOR

- 2 **Bit magnitude comparator**

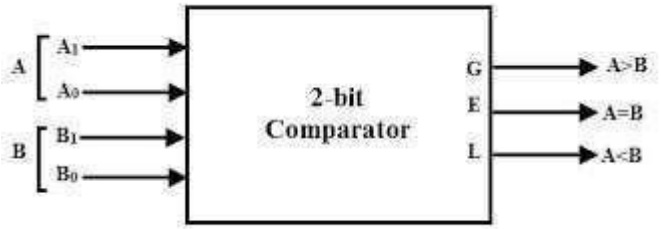


Fig :3 Block diagram

Inputs				Outputs		
A_1	A_0	B_1	B_0	A>B	A=B	A<B
0	0	0	0	0	1	0
0	0	0	1	0	0	1
0	0	1	0	0	0	1
0	0	1	1	0	0	1
0	1	0	0	1	0	0
0	1	0	1	0	1	0
0	1	1	0	0	0	1
0	1	1	1	0	0	1
1	0	0	0	1	0	0
1	0	0	1	1	0	0
1	0	1	0	0	1	0
1	0	1	1	0	0	1
1	1	0	0	1	0	0
1	1	0	1	1	0	0
1	1	1	0	1	0	0
1	1	1	1	0	1	0

Fig :4 Truth table

MAGNITUDE COMPARATOR

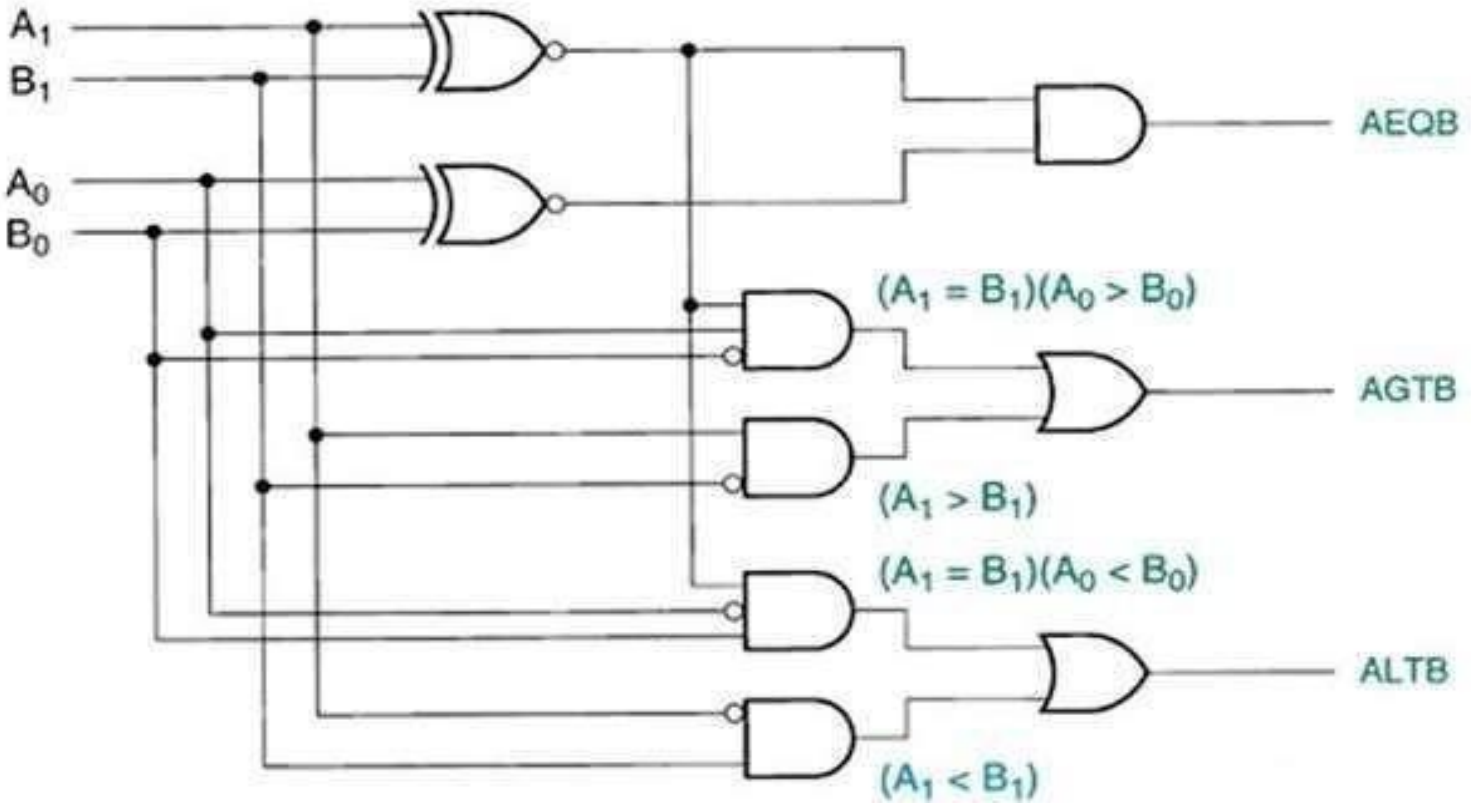
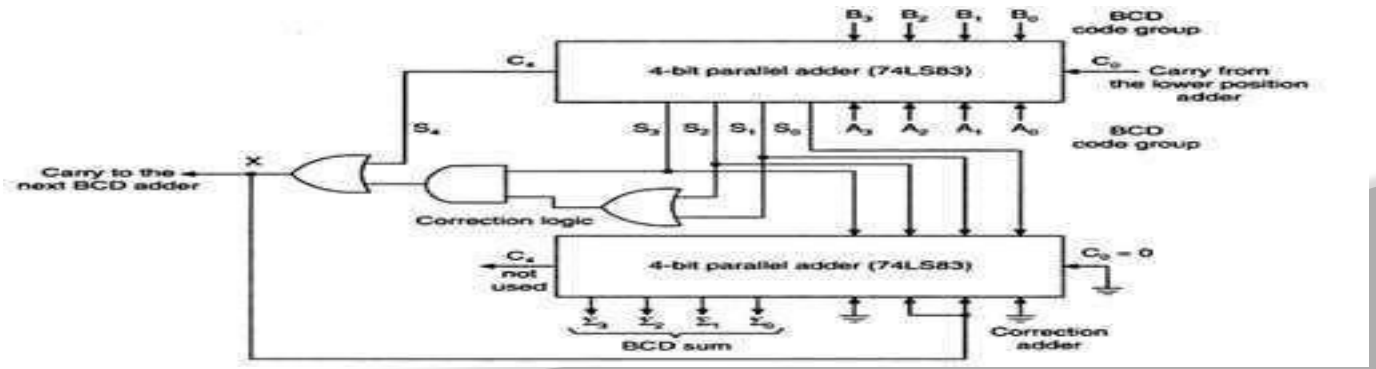


Fig 5:Logic diagram of 2-bit comparator

BCD ADDER

BCD Adder

- Perform the addition of two decimal digits in BCD, together with an input carry from a previous stage.
- When the sum is 9 or less, the sum is in proper BCD form and no correction is needed.
- When the sum of two digits is greater than 9, a correction of 0110 should be added to that sum, to produce the proper BCD result. This will produce a carry to be added to the next decimal position.



DECODER

- A binary decoder is a combinational logic circuit that converts binary information from the **n** coded inputs to a maximum of 2^n unique outputs.
- We have following types of decoders 2x4,3x8,4x16….

2x4 decoder

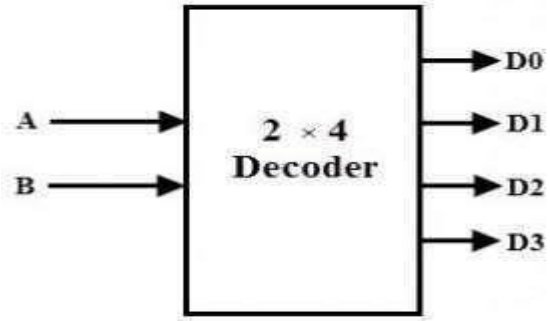


Fig 1: Block diagram

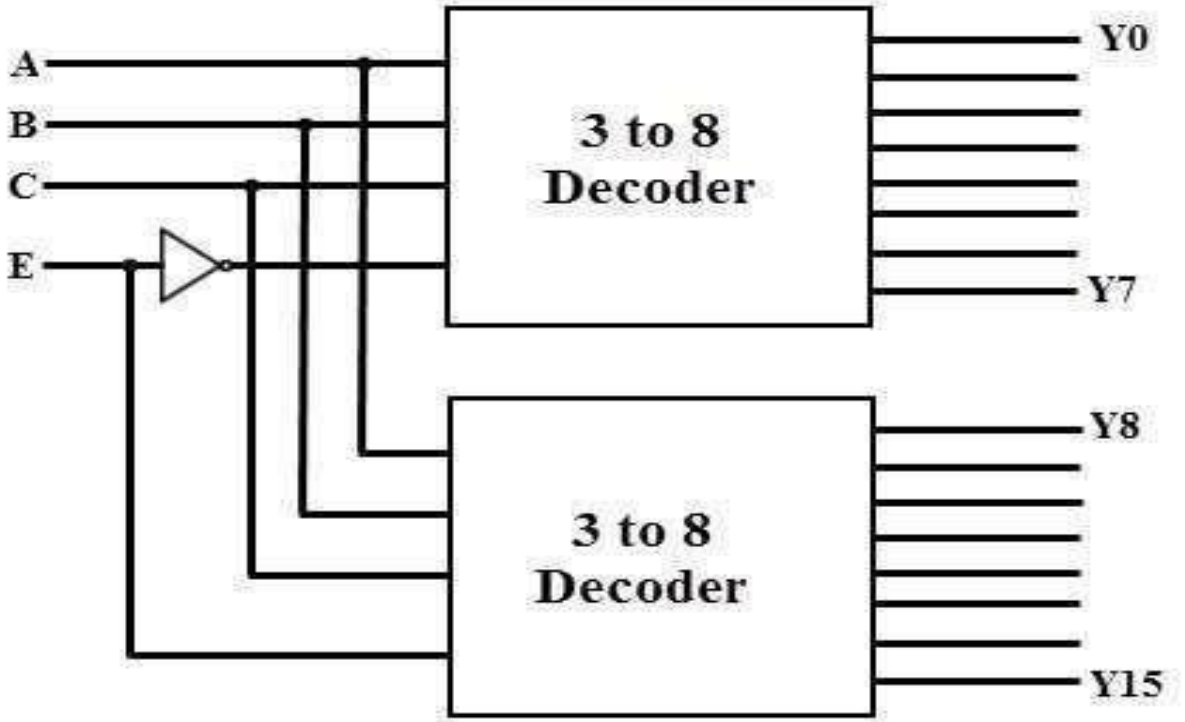
Inputs		Output			
A	B	D_0	D_1	D_2	D_3
0	0	1	0	0	0
0	1	0	1	0	0
0	1	0	0	1	0
1	1	0	0	0	1

Fig 2:Truth table

DECODER

Higher order decoder implementation using lower order.

Ex:4x16 decoder using 3x8 decoders



ENCODERS

- An Encoder is a combinational circuit that performs the reverse operation of Decoder. It has maximum of 2^n input lines and 'n' output lines.
- It will produce a binary code equivalent to the input, which is active High.

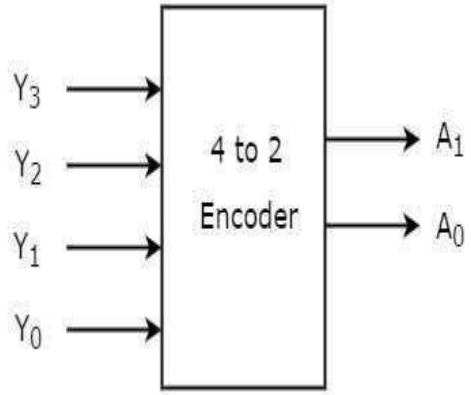


Fig 1:block diagram of 4x2encoder

ENCODERS

Octal to binary encoder

Octal digits		Binary		
		A_2	A_1	A_0
D_0	0	0	0	0
D_1	1	0	0	1
D_2	2	0	1	0
D_3	3	0	1	1
D_4	4	1	0	0
D_5	5	1	0	1
D_6	6	1	1	0
D_7	7	1	1	1

Fig 2:Truth table

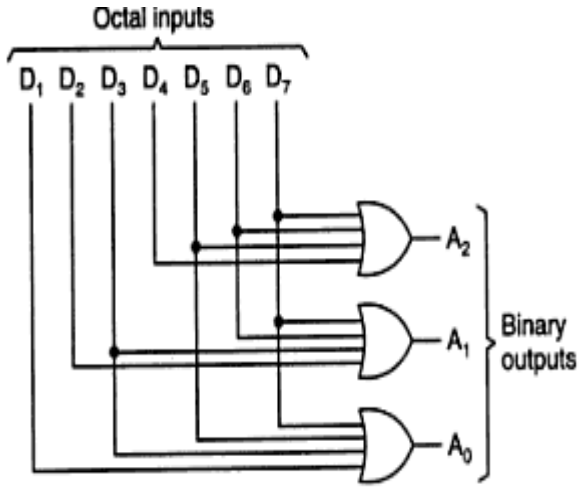


Fig 3: Logic diagram

ENCODERS

Priority encoder

A 4 to 2 priority encoder has four inputs Y_3, Y_2, Y_1 & Y_0 and two outputs A_1 & A_0. Here, the input, Y_3 has the highest priority, whereas the input, Y_0 has the lowest priority.

Inputs				Outputs		
Y_3	Y_2	Y_1	Y_0	A_1	A_0	V
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	x	0	1	1
0	1	x	x	1	0	1
1	x	x	x	1	1	1

Fig 4:Truth table

MULTIPLEXERS

- Multiplexer is a combinational circuit that has maximum of 2^n data inputs, 'n' selection lines and single output line. One of these data inputs will be connected to the output based on the values of selection lines.
- We have different types of multiplexers 2x1,4x1,8x1,16x1,32x1……

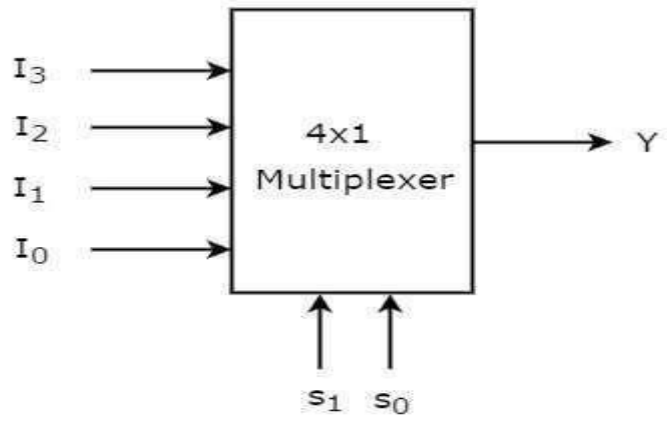


Fig 1: Block diagram

Selection Lines		Output
S_1	S_0	Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

Fig 2: Truth table

MULTIPLEXERS

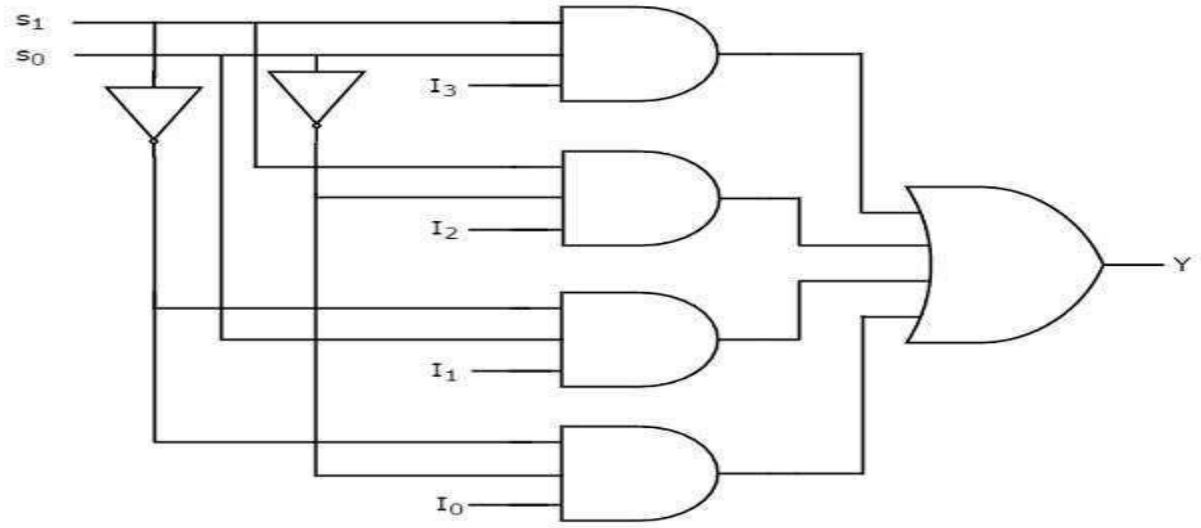


Fig 3: Logic diagram

- Now, let us implement the higher-order Multiplexer using lower-order Multiplexers.

MULTIPLEXERS

- Ex: 8x1 Multiplexer

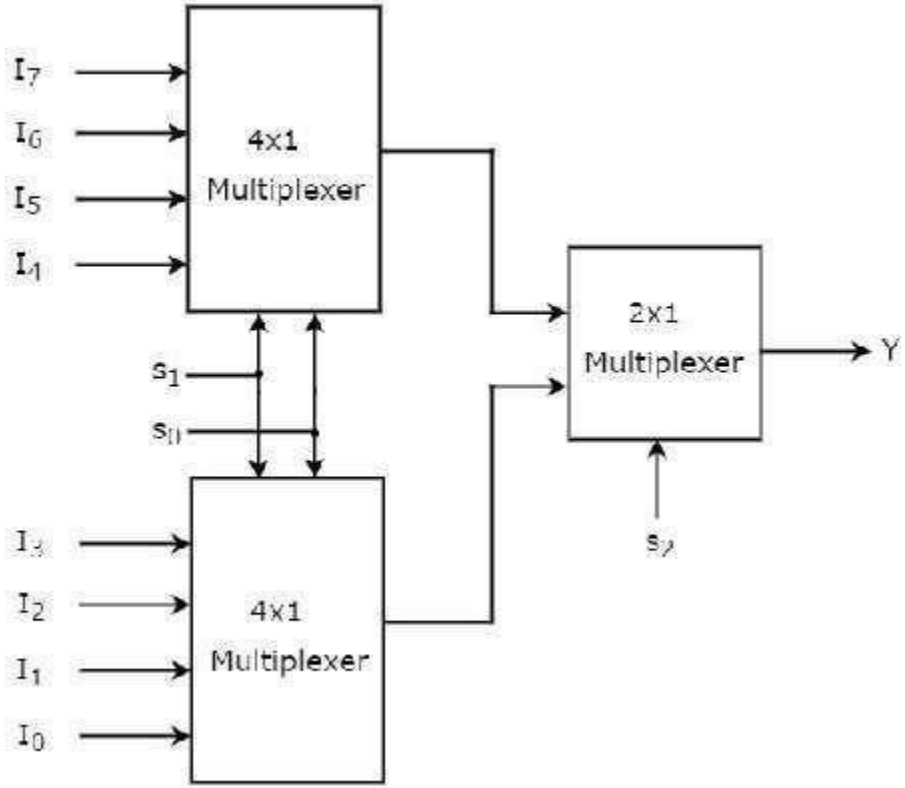
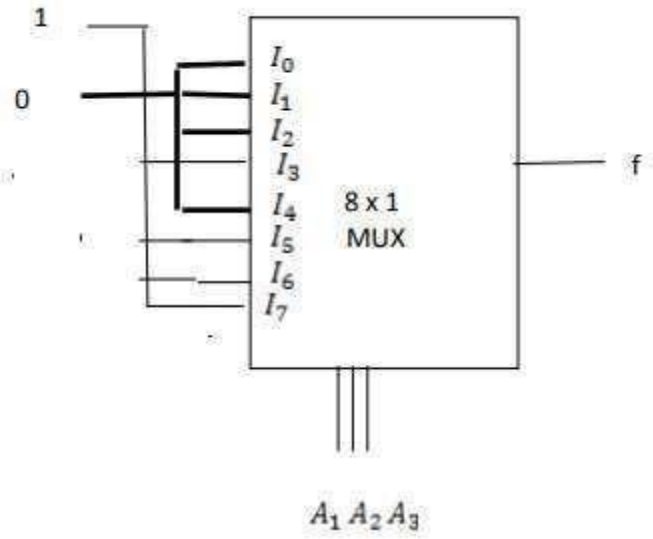


Fig 3: 8x1 Multiplexer diagram

MULTIPLEXERS

- Implementation of Boolean function using multiplexer
- f(A1 , A2 , A3) =Σ(3,5,6,7) implementation using 8x1 mux

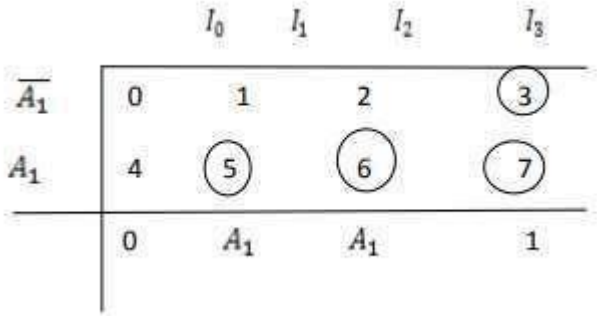


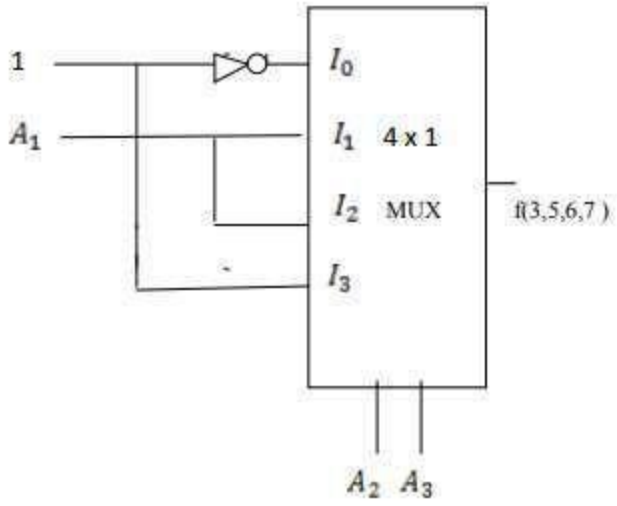
MULTIPLEXERS

f(A1 , A2 , A3) =Σ(3,5,6,7) implementation using 4x1 mux

Method:1

Minterms	A_1	A_2	A_3	f
0	0	0	0	0
1	0	0	1	0
2	0	1	0	0
3	0	1	1	1
4	1	0	0	0
5	1	0	1	1
6	1	1	0	1
7	1	1	1	1

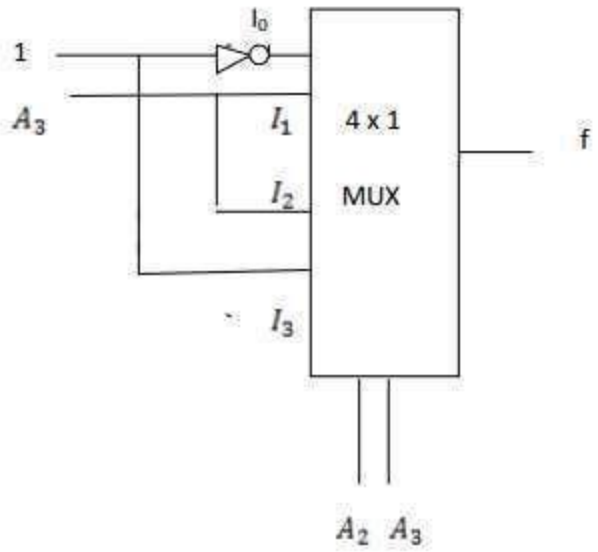




MULTIPLEXERS

Method:2

Minterm	A_1	A_2	A_3	f		
0 1	0 0	0 0	0 1	0 0	f= 0	I_0
2 3	0 0	1 1	0 1	0 1	f= A_3	I_1
4 5	1 1	0 0	0 1	0 1	f= A_3	I_2
6 7	1 1	1 1	0 1	1 1	f= 1	I_3



DEMULTIPLEXER

- A demultiplexer is a device that takes a single input line and routes it to one of several digital output lines.
- A demultiplexer of 2^n outputs has n select lines, which are used to select which output line to send the input.
- We have 1x2,1x4,8x1…. Demultiplexers.

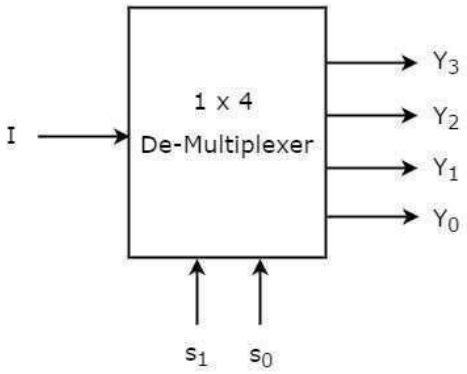


Fig:1 Block diagram

Selection Inputs		Outputs			
S_1	S_0	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	I
0	1	0	0	I	0
1	0	0	I	0	0
1	1	I	0	0	0

Fig :2 Truth table

DEMULTIPLEXER

Boolean functions for each output as

$$Y_3 = s_1 s_0 I$$

$$Y_2 = s_1 s_0' I$$

$$Y_1 = s_1' s_0 I$$

$$Y_0 = s_1' s_0' I$$

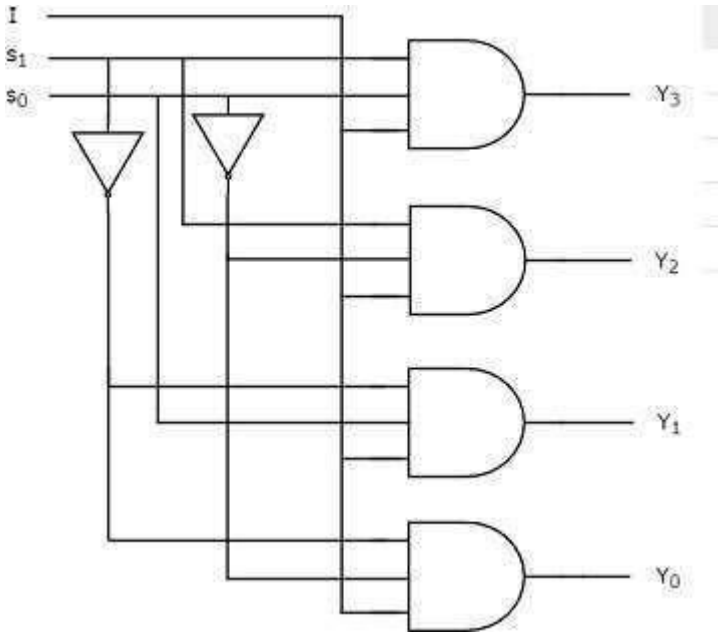


Fig:3 Logic diagram

CODE CONVERTERS

A code converter is a logic circuit whose inputs are bit patterns representing numbers (or character) in one code and whose outputs are the corresponding representation in a different code.

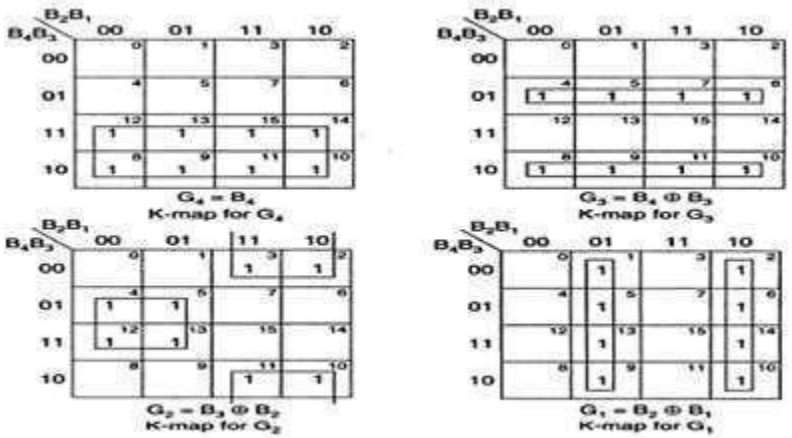
Design of a 4-bit binary to gray code converter

4-bit binary				4-bit Gray			
B_4	B_3	B_2	B_1	G_4	G_3	G_2	G_1
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	0	0	1
1	1	1	1	1	0	0	0

Fig :1 Truth table

CODE CONVERTERS

K-map simplification



$$G_4 = \Sigma m(8, 9, 10, 11, 12, 13, 14, 15) \qquad G_4 = B_4$$

$$G_3 = \Sigma m(4, 5, 6, 7, 8, 9, 10, 11) \qquad G_3 = \bar{B}_4 B_3 + B_4 \bar{B}_3 = B_4 \oplus B_3$$

$$G_2 = \Sigma m(2, 3, 4, 5, 10, 11, 12, 13) \qquad G_2 = \bar{B}_3 B_2 + B_3 \bar{B}_2 = B_3 \oplus B_2$$

$$G_1 = \Sigma m(1, 2, 5, 6, 9, 10, 13, 14) \qquad G_1 = \bar{B}_2 B_1 + B_2 \bar{B}_1 = B_2 \oplus B_1$$

CODE CONVERTERS

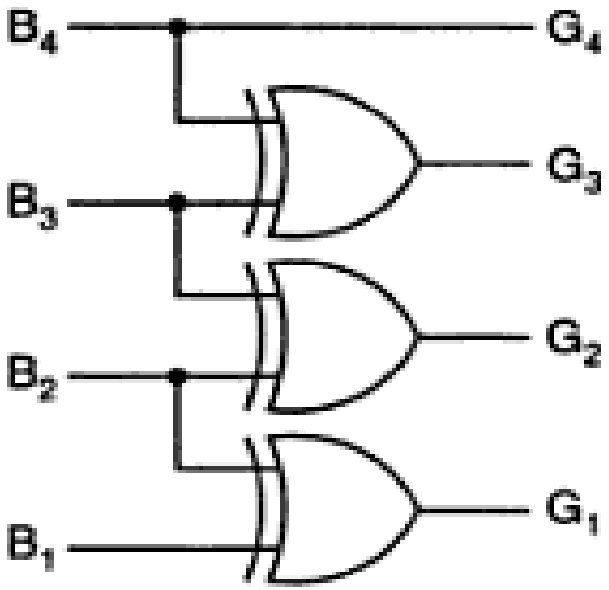


Fig: 2 Logic diagram

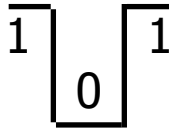
HAZARDS AND GLITCHES

- **glitch**: unwanted output
- A circuit with the potential for a glitch has a **hazard**.
- Glitches occur when different pathways have different delays
  - Causes circuit noise
  - Dangerous if logic makes a decision while output is unstable

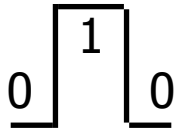
  Solutions
  - Design hazard-free circuits
  - Difficult when logic is multilevel
  - Wait until signals are stable

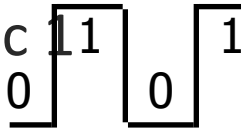
TYPES OF HAZARDS

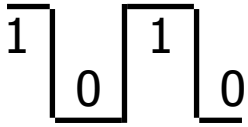
- Static 1-hazard
  - Output should stay logic 1
  - Gate delays cause brief glitch to logic 0



- Static 0-hazard
  - Output should stay logic 0
  - Gate delays cause brief glitch to logic 1



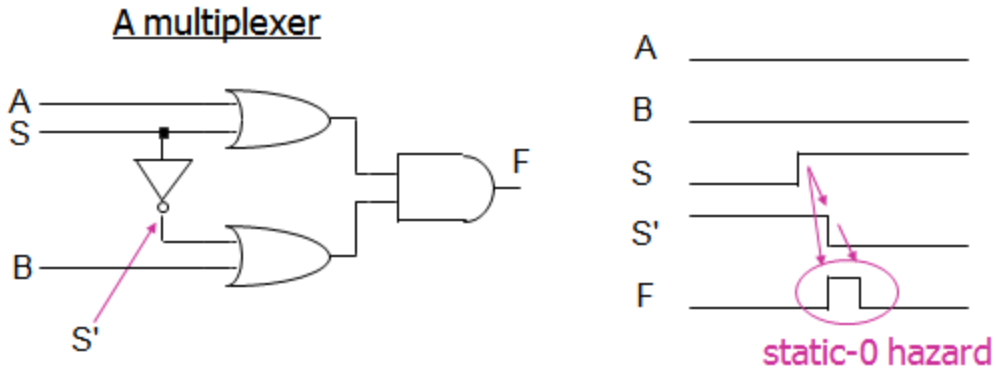




- Dynamic hazards
  - Output should toggle cleanly
  - Gate delays cause multiple transitions

STATIC HAZARDS

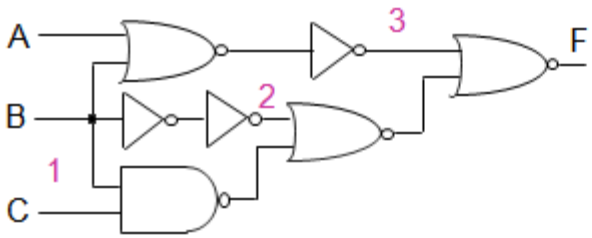
- Often occurs when a literal and its complement momentarily assume the same value
  - Through different paths with different delays
  - Causes an (ideally) static output to *glitch*

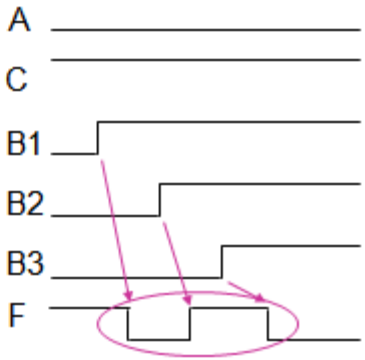


DYNAMIC HAZARDS

- Often occurs when a literal assumes multiple values
  - Through different paths with different delays
  - Causes an output to toggle multiple times

Dynamic hazards

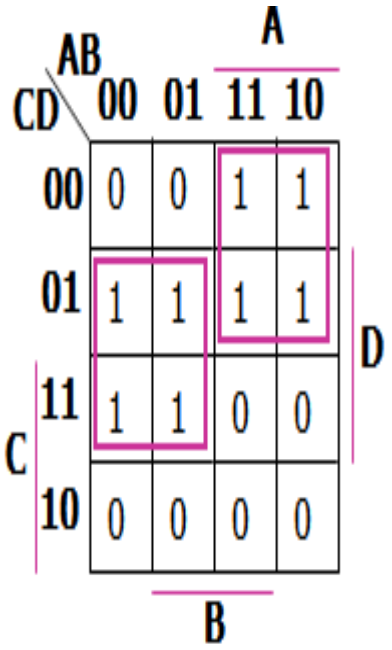




Dynamic hazard

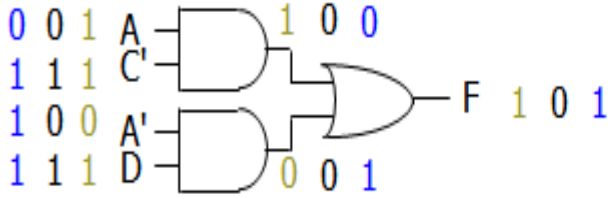
ELIMINATING STATIC HAZARDS

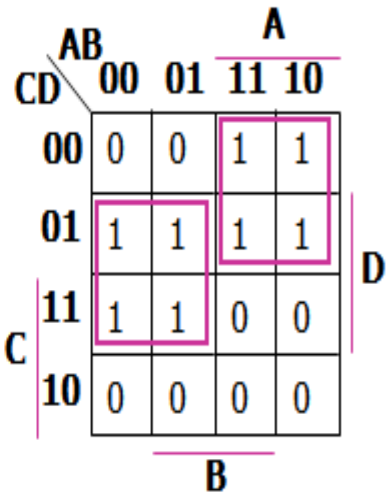
- Key idea: Glitches happen when a changing input spans separate K-map encirclements
  - Example: 1101 to 0101 change can cause a static-1 glitch



- ABCD: 1101 → 0101

F = AC' + A'D

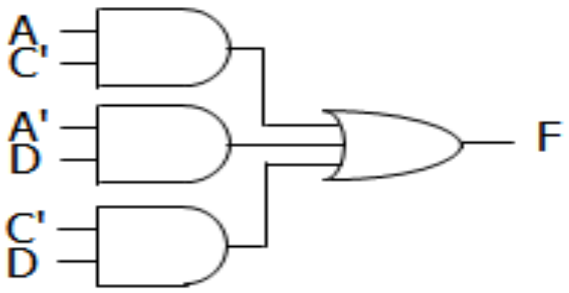


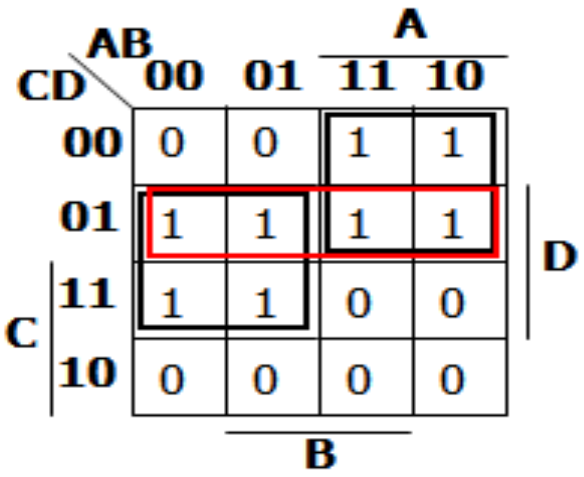


ELIMINATING STATIC HAZARDS

- Solution: Add redundant K-map encirclements
  - Ensure that all single-bit changes are covered by same block
  - First eliminate static-1 hazards: Use SOP form
  - If need to eliminate static-0 hazards, use POS form
- Technique only works for 2-level logic

$$F = AC' + A'D + C'D$$





Unit-III

SEQUENTIAL CIRCUITS AND SYSTEMS

SEQUENTIAL CIRCUITS

- Gated latch is a basic latch that includes input gating and a control signal.

The latch retains its existing state when the control input is equal to 0.

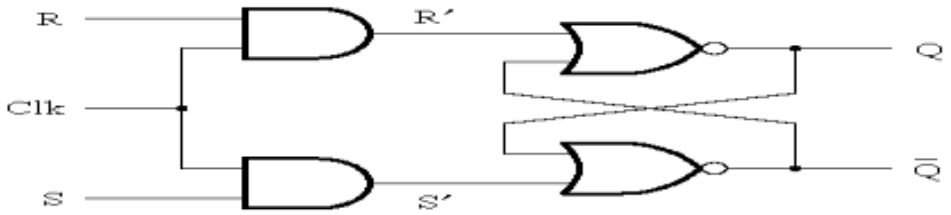
- Its state may be changed when the control signal is equal to 1.In our discussion we referred to the control input as the clock.

- We consider two types of gated latches:
  - Gated SR latch uses the *S* and *R* inputs to set the latch to 1
  - Gated D latch uses the *D* input to force the latch into a state that has the same logic value as the *D*input.

SEQUENTIAL CIRCUITS

- Basic latch is a feedback connection oftwo NOR gates or two NAND gates.
- It can store one bit of information.
- It can be set to 1 using the S input andreset to 0 using the *R* input.
- A feedback loop with even numberof inverters
- If A = 0, B = 1 or when A = 1, B =0
- This circuit is not useful due to the lack ofa mechanism for changing its state

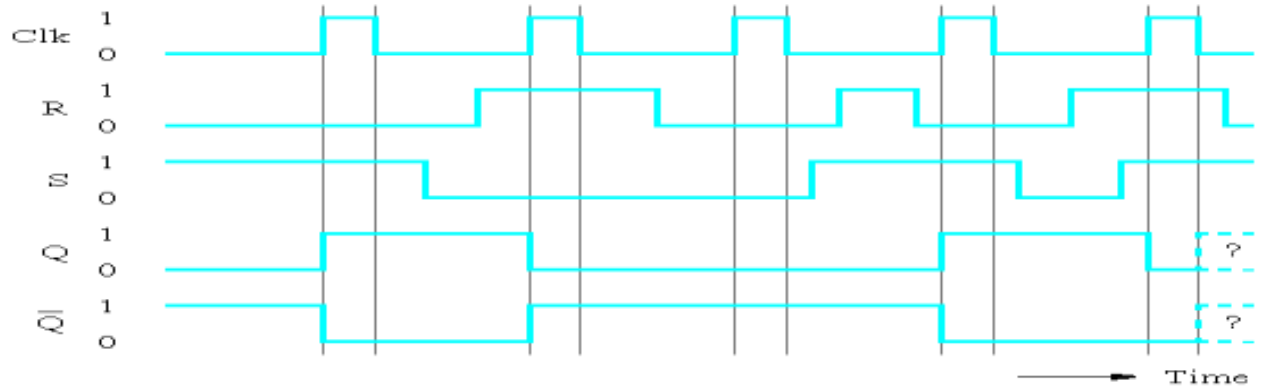
RS LATCH:



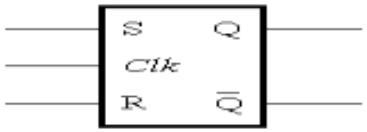
(a) Circuit

Clk	S	R	Q(t + 1)
0	x	x	Q(t) (no change)
1	0	0	Q(t) (no change)
1	0	1	0
1	1	0	1
1	1	1	x

(b) Characteristic table

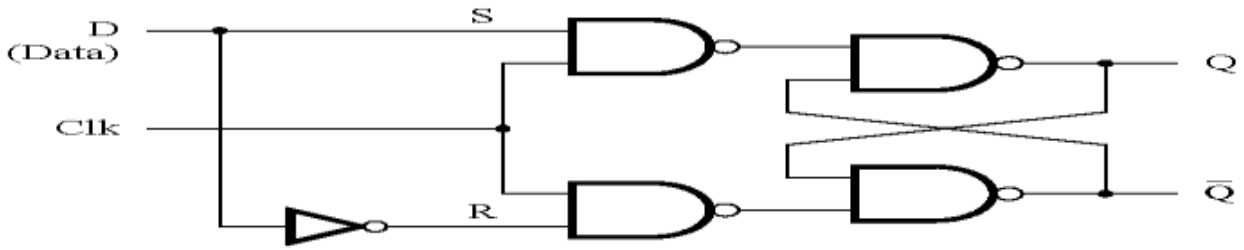


(c) Timing diagram



(d) Graphical symbol

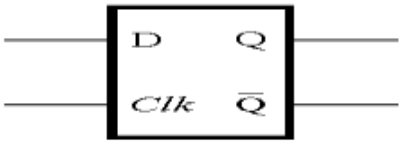
D LATCH:



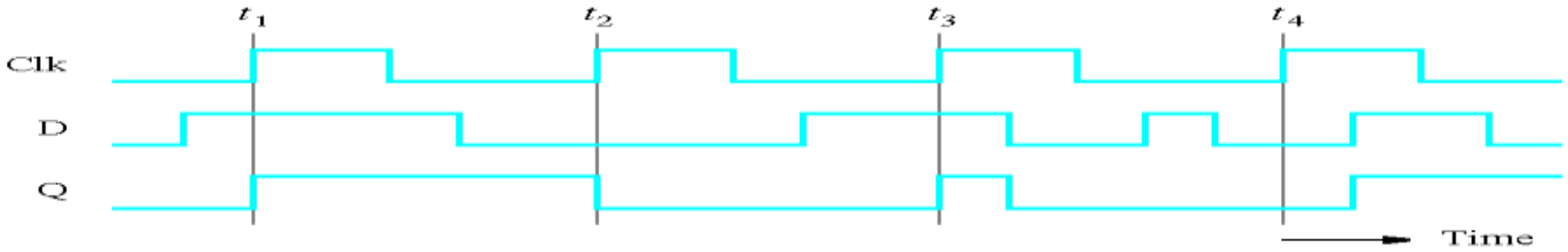
(a) Circuit

Clk	D	Q(t + 1)
0	x	Q(t)
1	0	0
1	1	1

(b) Characteristic table



(c) Graphical symbol



(d) Timing diagram

Flip flops:

- A flip-flop is a storage element based onthe gated latch principle.
- It can have its output state changed onlyon the edge of the controlling clocksignal.

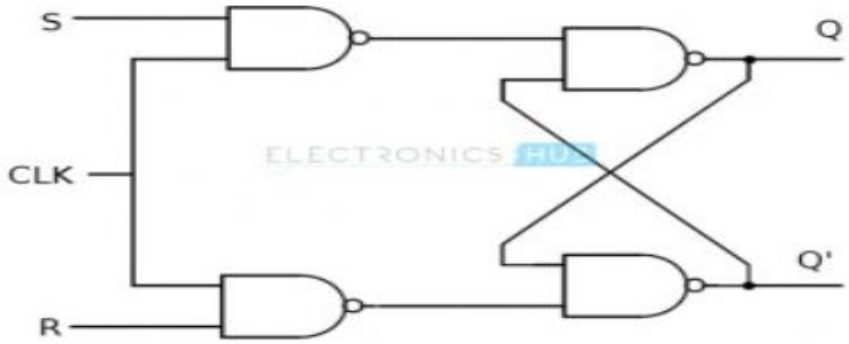
Types Of Flip-flops:

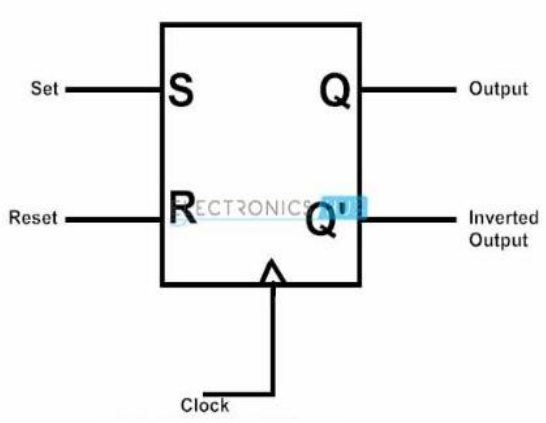
- SR flip-flop (Set, Reset)
- T flip-flop (Toggle)
- D flip-flop (Delay)
- JK flip-flop

Flip flops:

- Edge-triggered flip-flop is affected only by the input values present when the active edge of the clock occurs
- Master-slave flip-flop is built with two gated
- latches
  - The master stage is active during half of the clock cycle, and the slave stage is active during the other half.
  - The output value of the flip-flop changes on the edge of the clock that activates the transfer into the slave stage

SR Flip flop



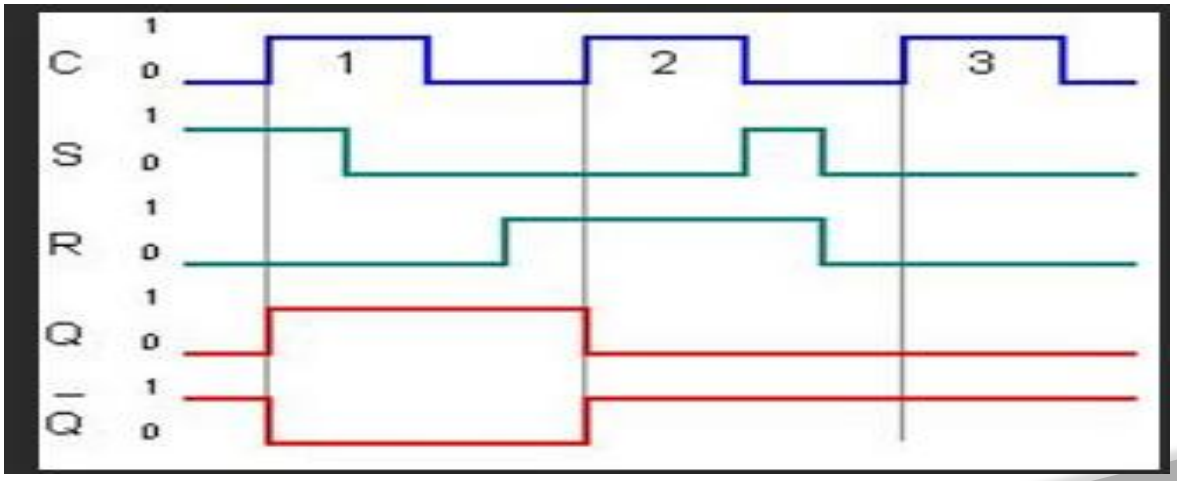


INPUTS			OUTPUT	STATE
CLK	S	R	Q	
X	0	0	No Change	Previous
↑	0	1	0	Reset
↑	1	0	1	Set
↑	1	1	-	Forbidden

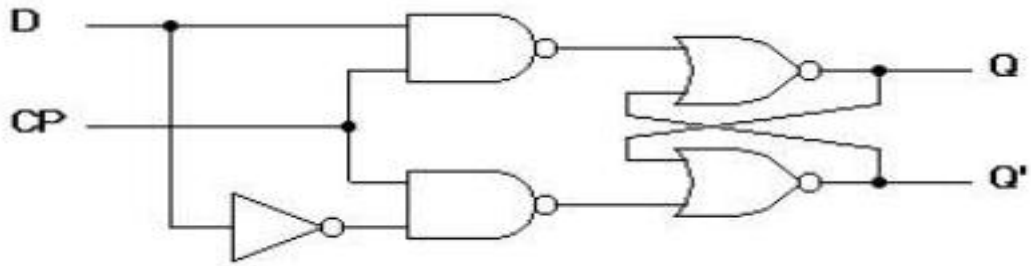
SR FLIPFLOP Excitation Table &Timing Diagram

SR FLIP-FLOP			
Q(t)	Q(t+1)	S	R
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

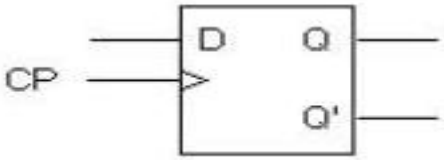
EXCITATION TABLE OF SR FLIP-FLOP



D FLIPFLOP:



(a) Logic diagram with NAND gates



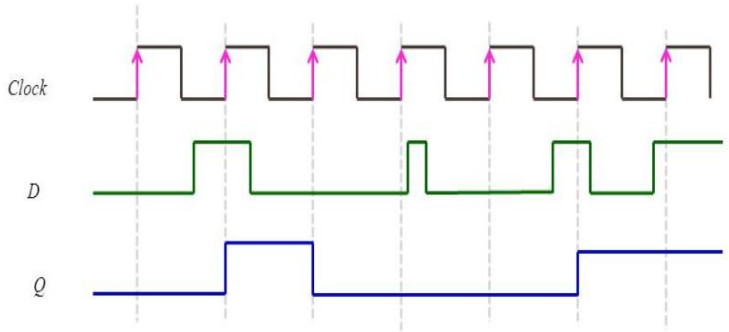
(b) Graphical symbol

Q	D	Q(t+1)
0	0	0
0	1	1
1	0	0
1	1	1

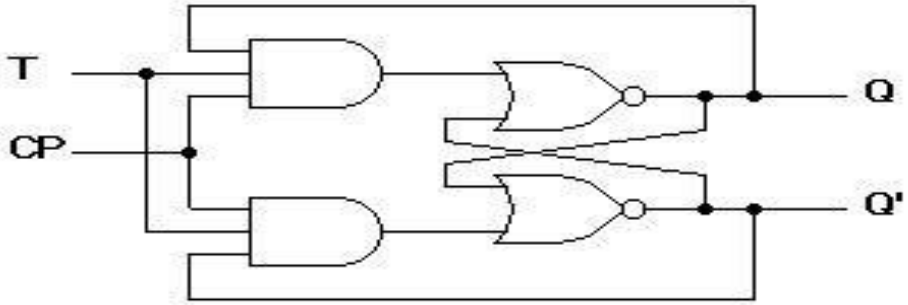
(c) Transition table

D FLIPFLOP Excitation Table &Timing Diagram:

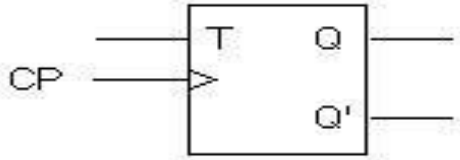
Present state (Q_n)	Next state (Q_{n+1})	D
0	0	0
0	1	1
1	0	0
1	1	1



T FLIPFLOP :



(a) Logic diagram

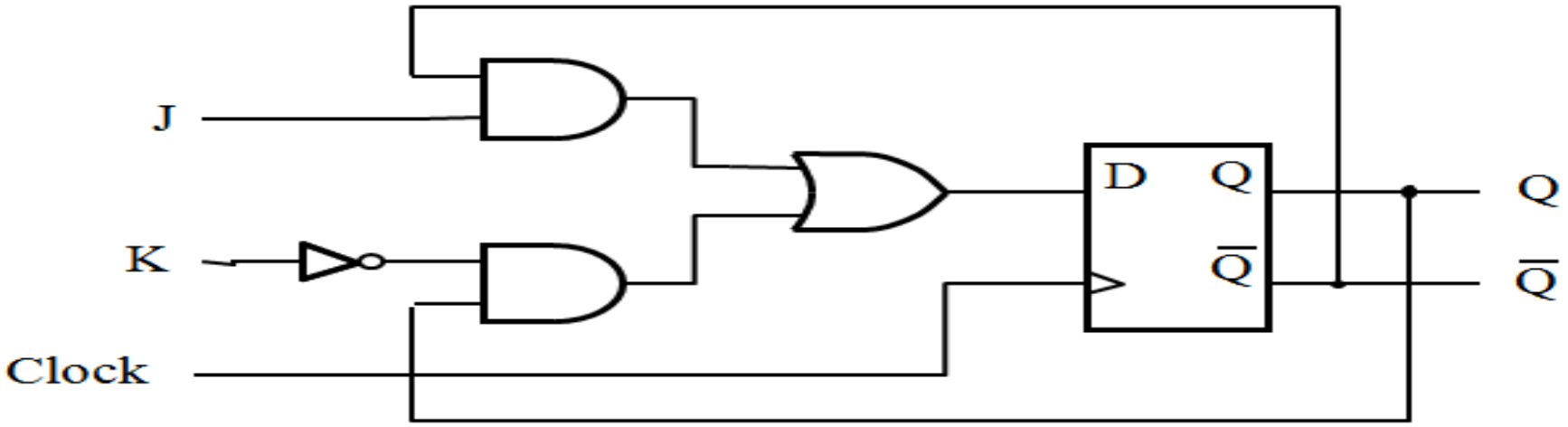


(b) Graphical symbol

Q	T	Q(t+1)
0	0	0
0	1	1
1	0	1
1	1	0

(c) Transition table

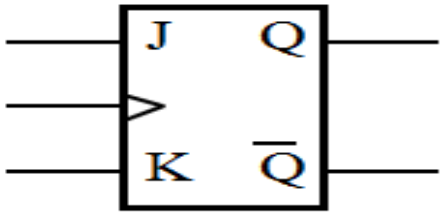
JK FLIPFLOP



(a) Circuit

J	K	Q (t+1)
0	0	Q (t)
0	1	0
1	0	1
1	1	$\overline{Q}$ (t)

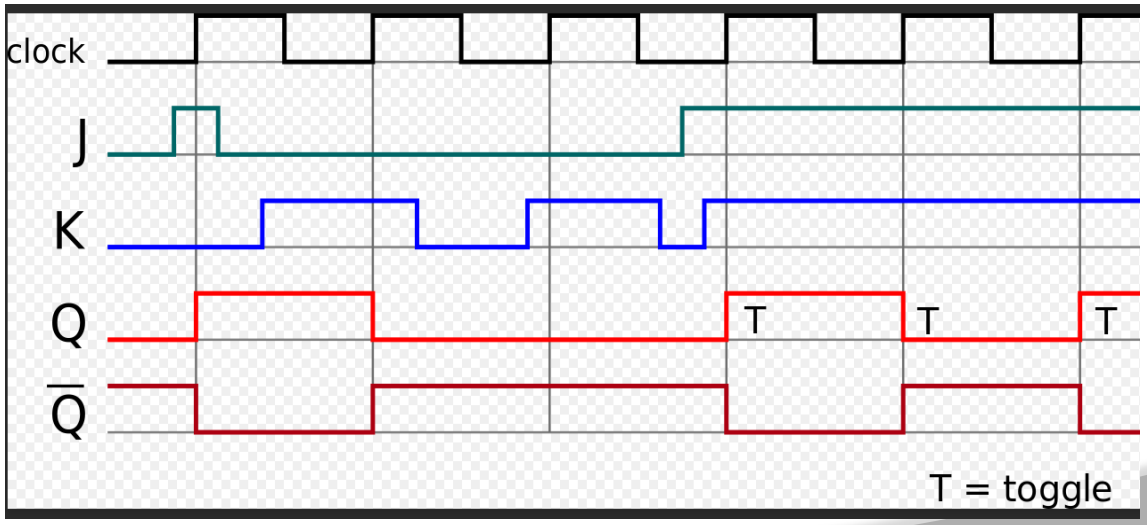
(b) Characteristic table



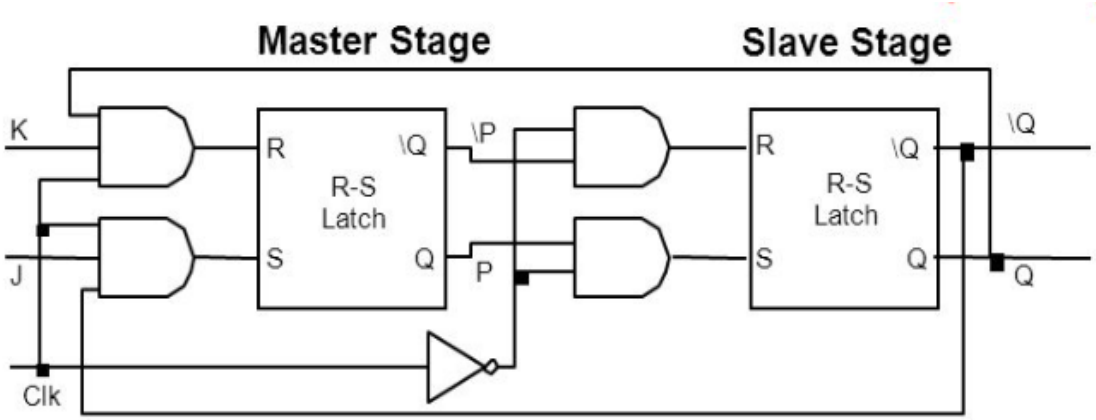
(c) Graphical symbol

JK FLIPFLOP

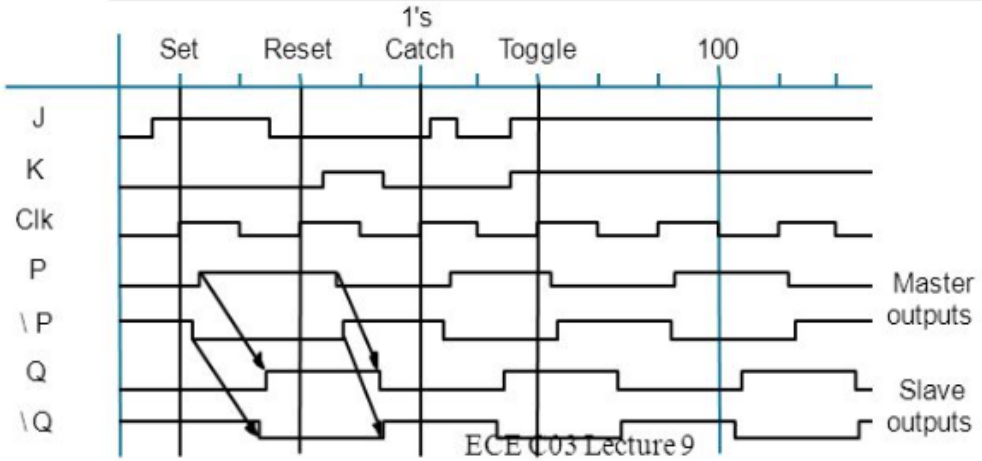
Q Output		Inputs	
Present State	Next State	J_n	K_n
0	0	0	x
0	1	1	x
1	0	x	1
1	1	x	0



Master Slave JK FLIPFLOP



Uses time to break feedback path from outputs to inputs!



Correct Toggle Operation

Conversion Of FLIPFLOP:

1. Draw the block diagram of the target flip flop from the given problem.
2. Write truth table for the target flip-flop.
3. Write excitation table for the available flip-flop.
4. Draw k-map for target flip-flop.
5. Draw the block diagram.

Jk to SR FLIPFLOP:

- **Characteristic Table**

S	R	Q(t+1)	Operation
0	0	$Q(t)$	No change
0	1	0	Reset
1	0	1	Set
1	1	?	Undefined

J	K	Q(t+1)	Operation
0	0	$Q(t)$	No change
0	1	0	Reset
1	0	1	Set
1	1	$\overline{Q}(t)$	Complement (Toggle)

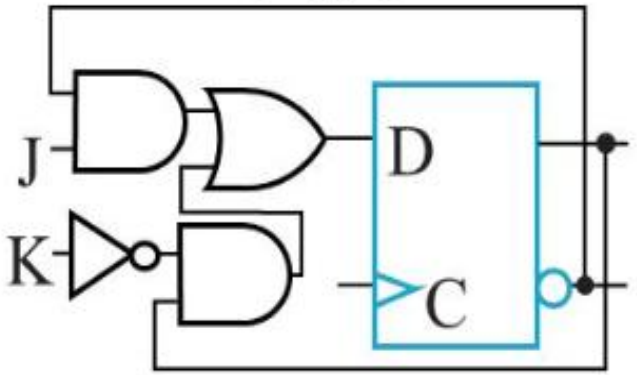
- **Characteristic Equation**

$$Q(t+1) = J\,\overline{Q} + \overline{K}\,Q$$

- **Excitation Table**

Q(t)	Q(t+1)	J	K	Operation
0	0	0	X	No change
0	1	1	X	Set
1	0	X	1	Reset
1	1	X	0	No Change

Change



Characteristic Equation:

Flip-flop	Characteristic Equation
D	$Q(t+1) = D$
T	$Q(t+1) = T \oplus Q(t)$
SR	$Q(t+1) = S + R' Q(t)$
JK	$Q(t+1) = J Q(t)' + K' Q(t)$

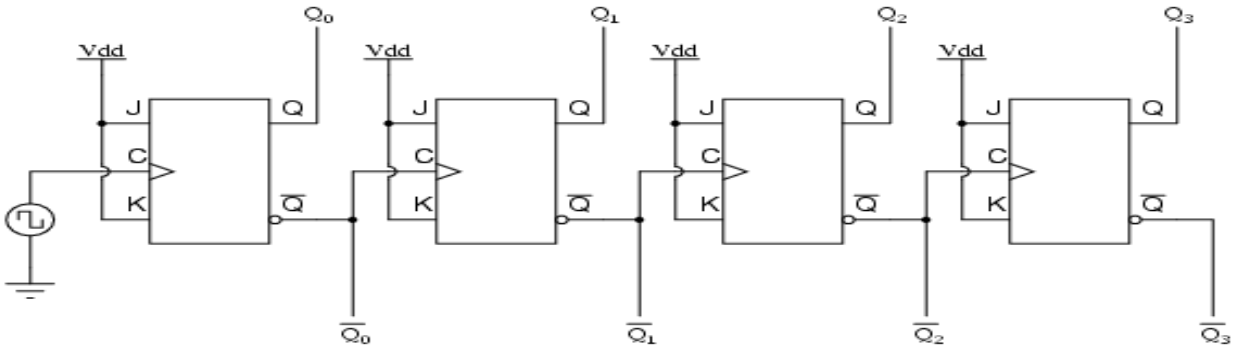
Counters:

- Counters are a specific type of sequential circuit.
- Like registers, the state, or the flip-flop values themselves, serves as the “output.”
- The output value increases by one on each clock cycle.
- After the largest value, the output “wraps around” back to 0.
- Counters can act as simple clocks to keep track of “time.”
- You may need to record how many times something has happened.
  - How many bits have been sent or received?
  - How many steps have been performed in some computation?

Asynchronous Counters:

- Asynchronous counter created from two JK flip-flops An asynchronous (ripple) counter is a single d-type flip-flop, with its J (data) input fed from its own inverted output. This circuit can store one bit, and hence can count from zero to one before it overflows (starts over from 0).

Asynchronous Up/Down Counters:



"Up" count sequence

Q_0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
Q_1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
Q_2	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
Q_3	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

"Down" count sequence

$\overline{Q_0}$	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
$\overline{Q_1}$	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0
$\overline{Q_2}$	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0
$\overline{Q_3}$	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0

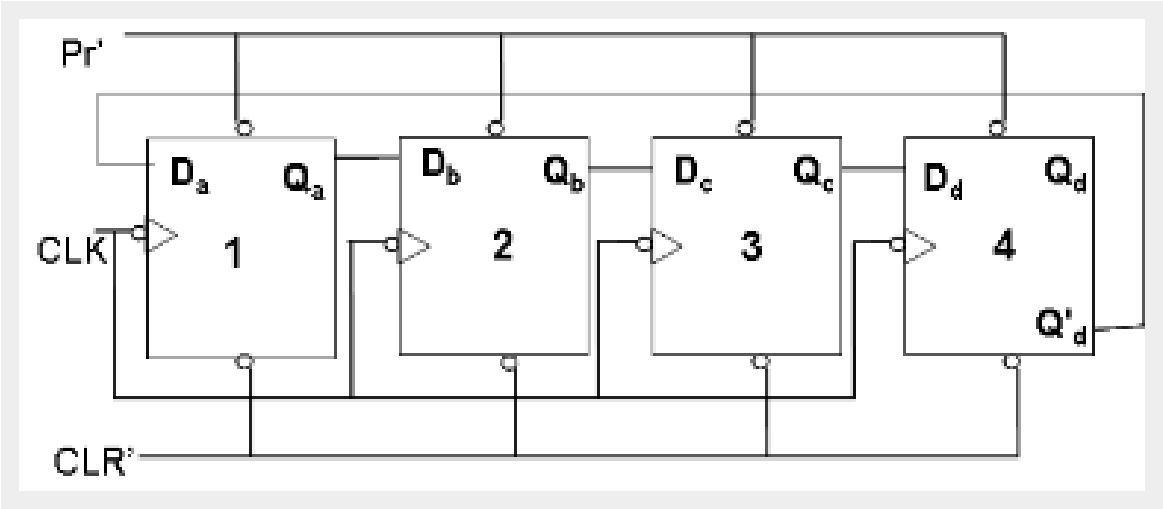
Synchronous Counters:

- The counters which use clock signal to change their transition are called “Synchronous counters”. This means the synchronous counters depends on their clock input to change state values. In synchronous counters, all flip flops are connected to the same clock signal and all flip flops will trigger at the same time.

Types of Counters:

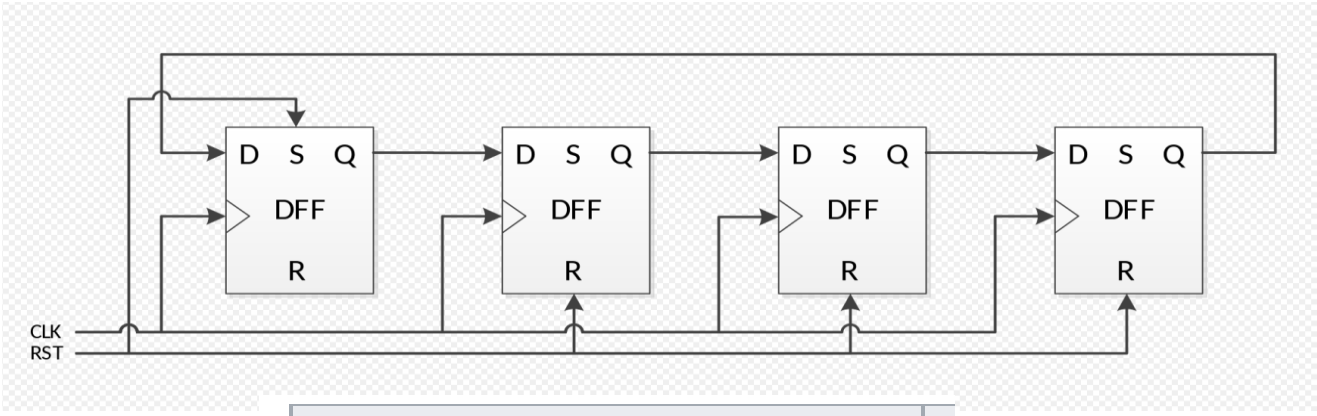
- Binary counters
- 4 bit synchronous UP counter
- 4 bit synchronous DOWN counter
- 4 bit synchronous UP / DOWN counter
- Loadable counters
- BCD counters
- Ring counters
- Johnson counters etc.

Johnson Counters:



Johnson counter				
State	Q0	Q1	Q2	Q3
0	0	0	0	0
1	1	0	0	0
2	1	1	0	0
3	1	1	1	0
4	1	1	1	1
5	0	1	1	1
6	0	0	1	1
7	0	0	0	1
0	0	0	0	0

Ring Counters:



Straight ring counter				
State	Q0	Q1	Q2	Q3
0	1	0	0	0
1	0	1	0	0
2	0	0	1	0
3	0	0	0	1
0	1	0	0	0
1	0	1	0	0
2	0	0	1	0
3	0	0	0	1
0	1	0	0	0

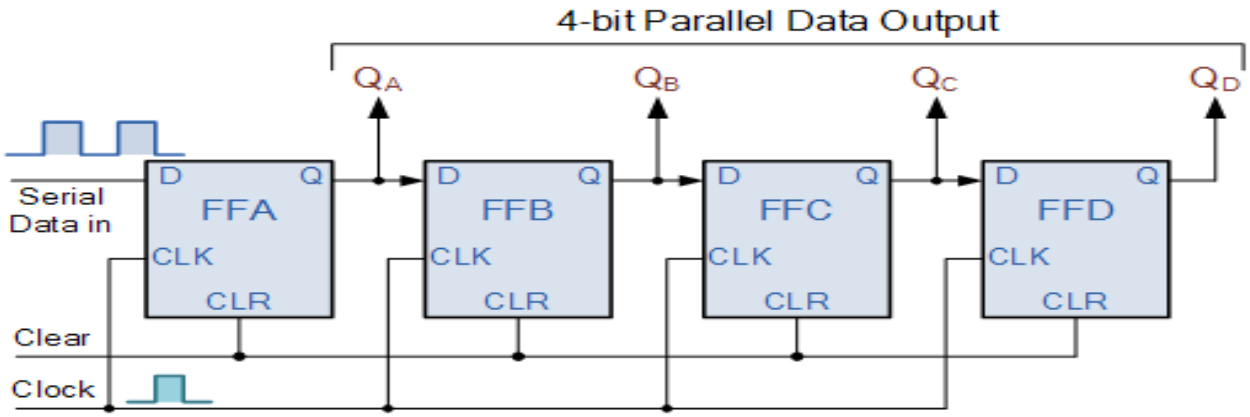
Shift Register:

- Shift registers, like counters, are a form of sequential logic. Sequential logic, unlike Combinational Logic is not only affected by the present inputs, but also, by the prior history. In other words, sequential logic remembers past events.

Types of Shift Registers:

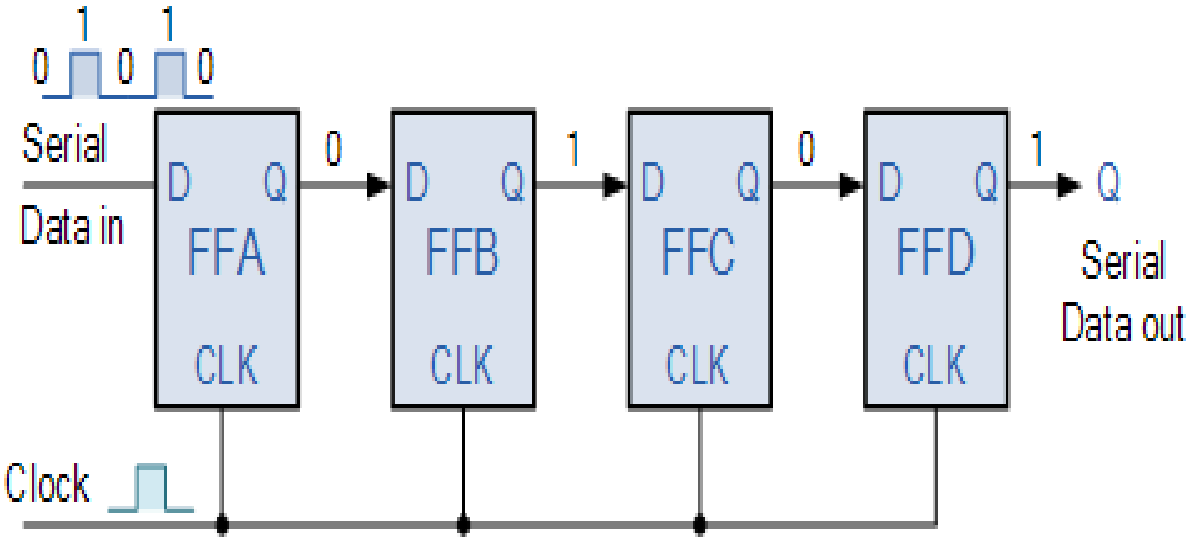
- Serial-in/serial-out
- Parallel-in/serial-out
- Serial-in/parallel-out
- Universal parallel-in/parallel-out

Serial in to Parallel Output:

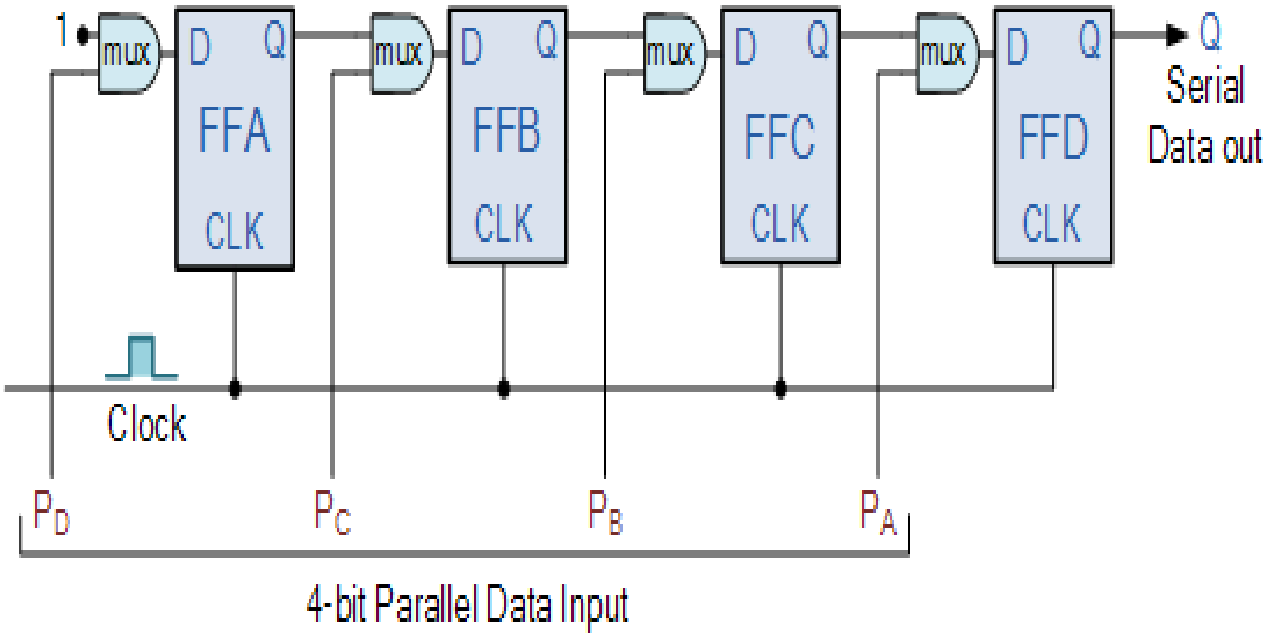


Clock Pulse No	QA	QB	QC	QD
0	0	0	0	0
1	1	0	0	0
2	0	1	0	0
3	0	0	1	0
4	0	0	0	1
5	0	0	0	0

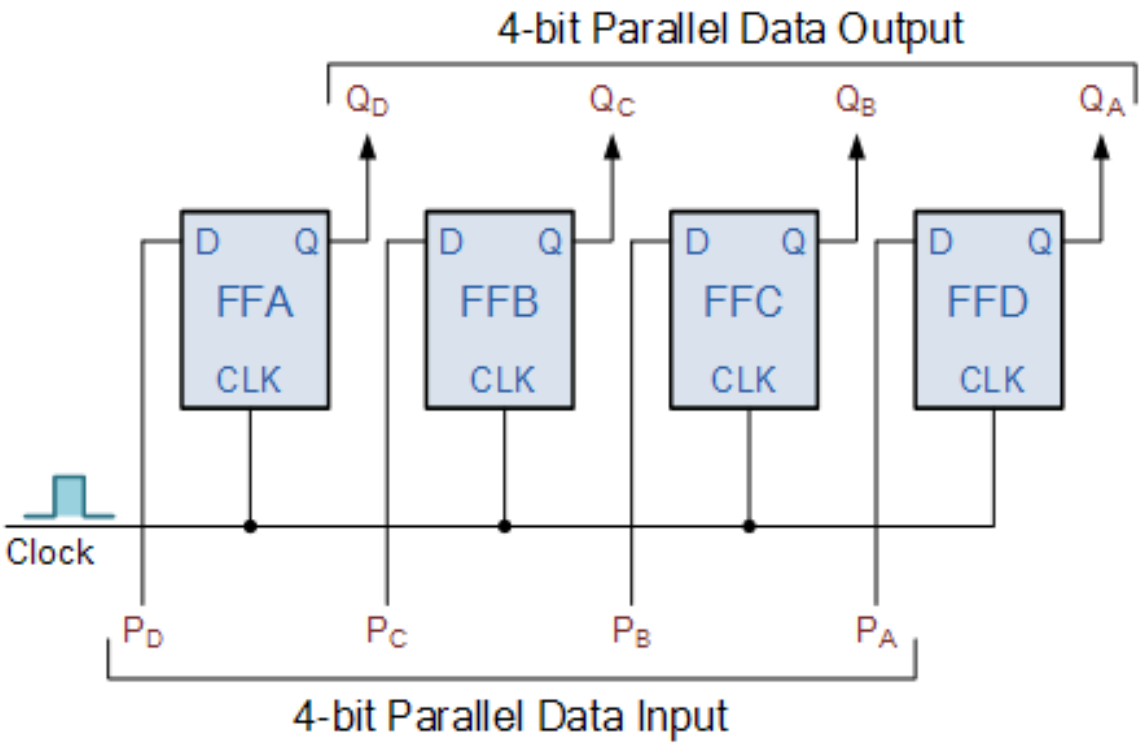
Serial in to Serial Output:



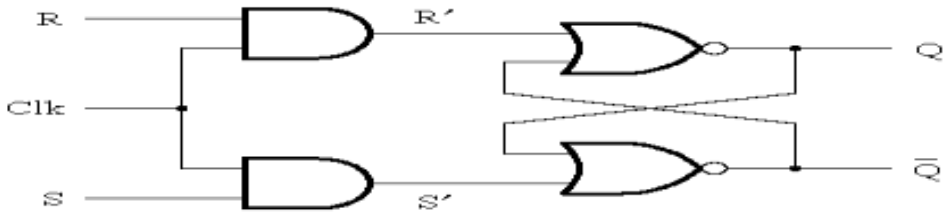
Parallel in to Serial Output:



Parallel in to Parallel Output:



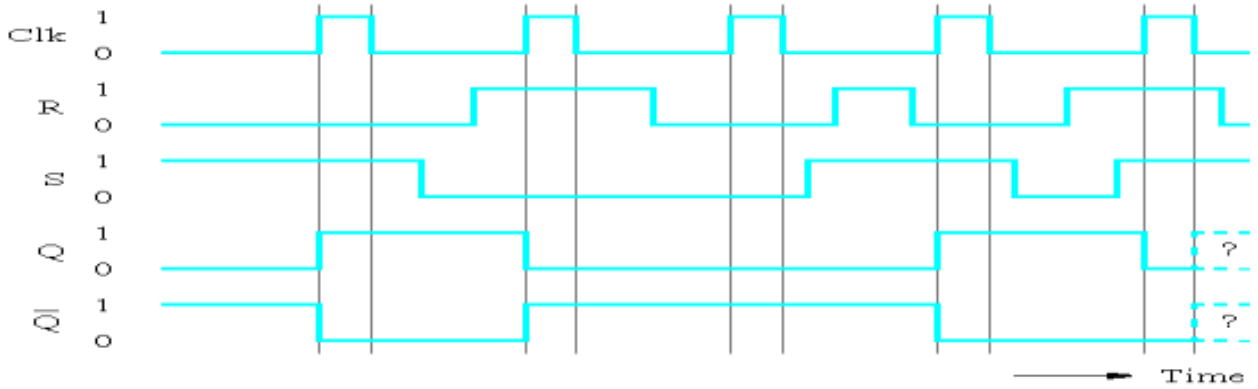
RS LATCH:



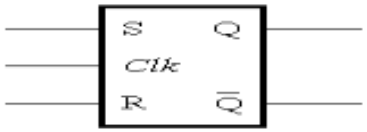
(a) Circuit

Clk	S	R	Q(t + 1)
0	x	x	Q(t) (no change)
1	0	0	Q(t) (no change)
1	0	1	0
1	1	0	1
1	1	1	x

(b) Characteristic table

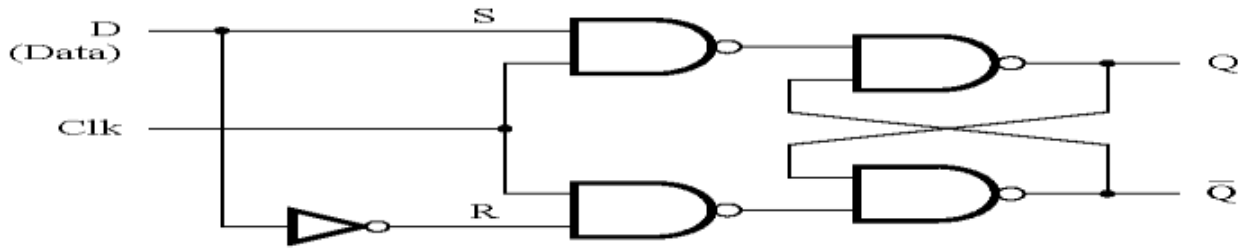


(c) Timing diagram



(d) Graphical symbol

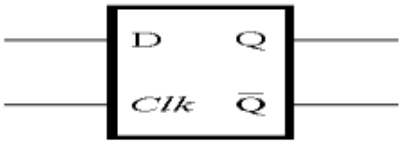
D LATCH:



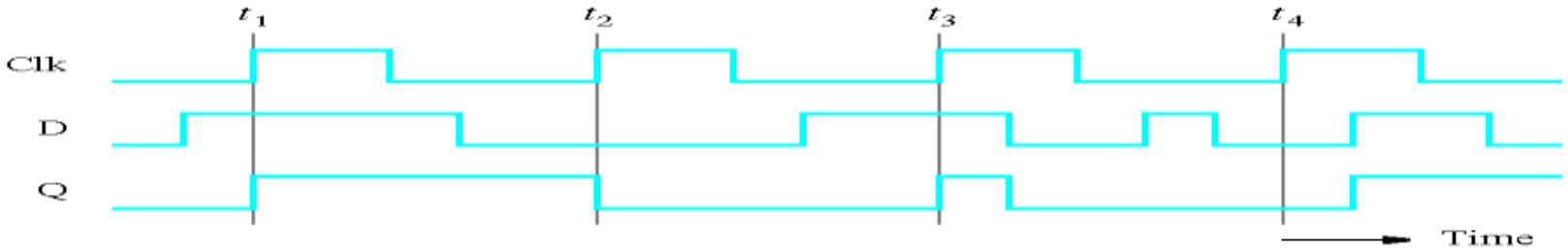
(a) Circuit

Clk	D	Q(t + 1)
0	x	Q(t)
1	0	0
1	1	1

(b) Characteristic table



(c) Graphical symbol



(d) Timing diagram

Flip flops:

- A flip-flop is a storage element based onthe gated latch principle.
- It can have its output state changed onlyon the edge of the controlling clocksignal.

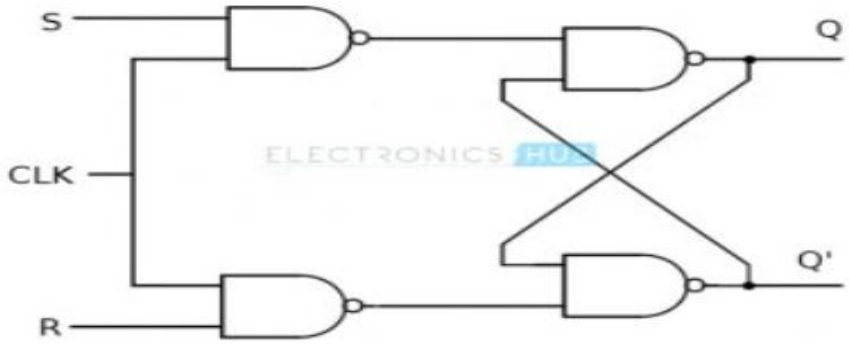
Types Of Flip-flops:

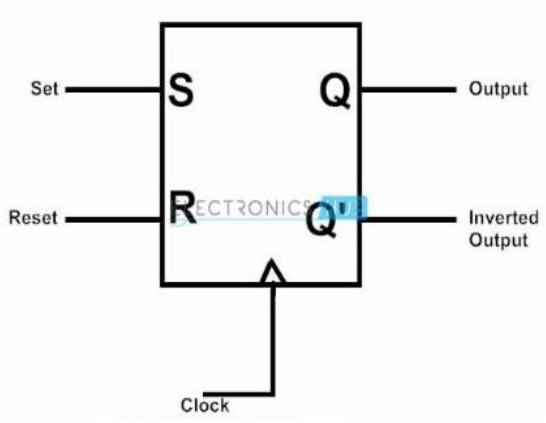
- SR flip-flop (Set, Reset)
- T flip-flop (Toggle)
- D flip-flop (Delay)
- JK flip-flop

Flip flops:

- Edge-triggered flip-flop is affected only by the input values present when the active edge of the clock occurs
- Master-slave flip-flop is built with two gated
- latches
  - The master stage is active during half of the clock cycle, and the slave stage is active during the other half.
  - The output value of the flip-flop changes on the edge of the clock that activates the transfer into the slave stage

SR Flip flop



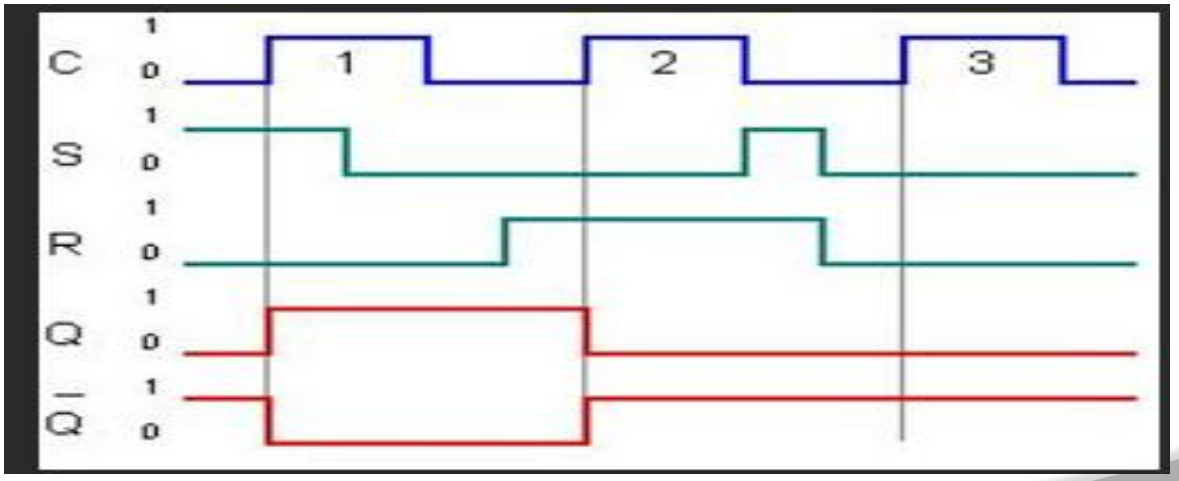


INPUTS			OUTPUT	STATE
CLK	S	R	Q	
X	0	0	No Change	Previous
↑	0	1	0	Reset
↑	1	0	1	Set
↑	1	1	-	Forbidden

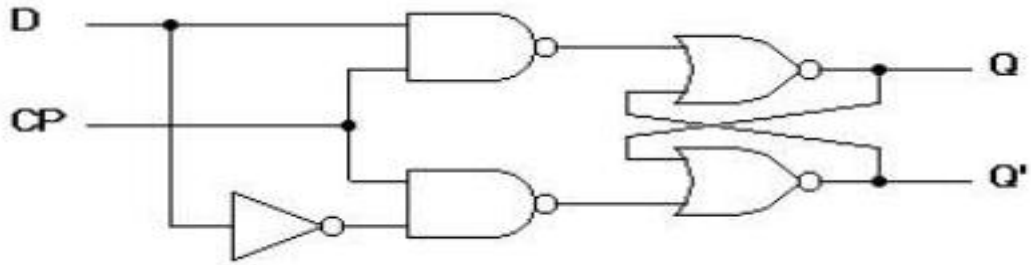
SR FLIPFLOP Excitation Table &Timing Diagram

SR FLIP-FLOP			
Q(t)	Q(t+1)	S	R
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

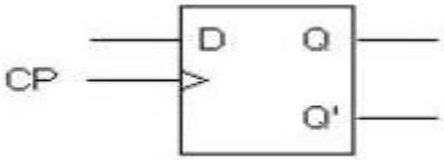
EXCITATION TABLE OF SR FLIP-FLOP



D FLIPFLOP:



(a) Logic diagram with NAND gates



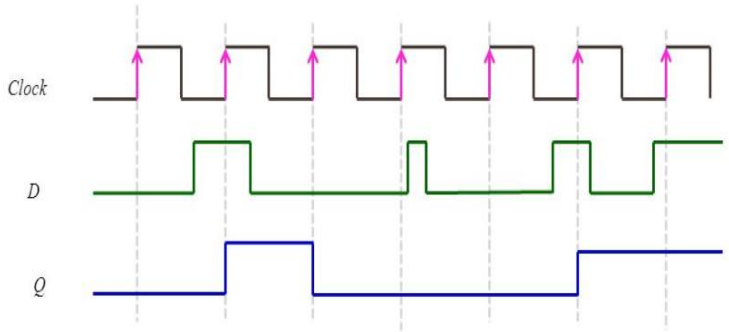
(b) Graphical symbol

Q	D	Q(t+1)
0	0	0
0	1	1
1	0	0
1	1	1

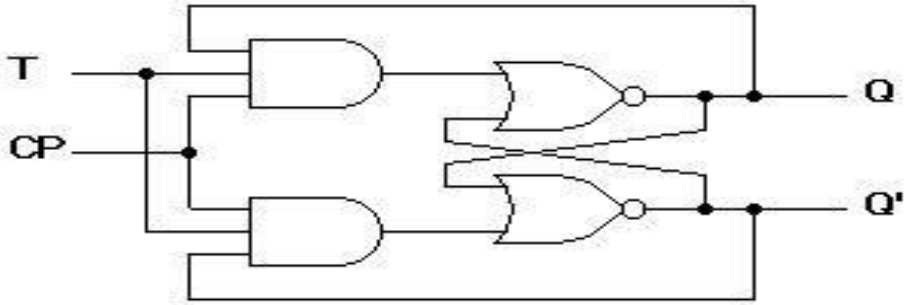
(c) Transition table

D FLIPFLOP Excitation Table &Timing Diagram:

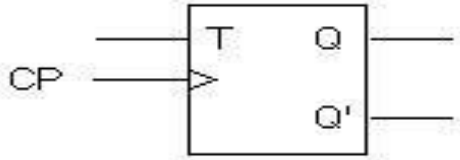
Present state (Q_n)	Next state (Q_{n+1})	D
0	0	0
0	1	1
1	0	0
1	1	1



T FLIPFLOP :



(a) Logic diagram

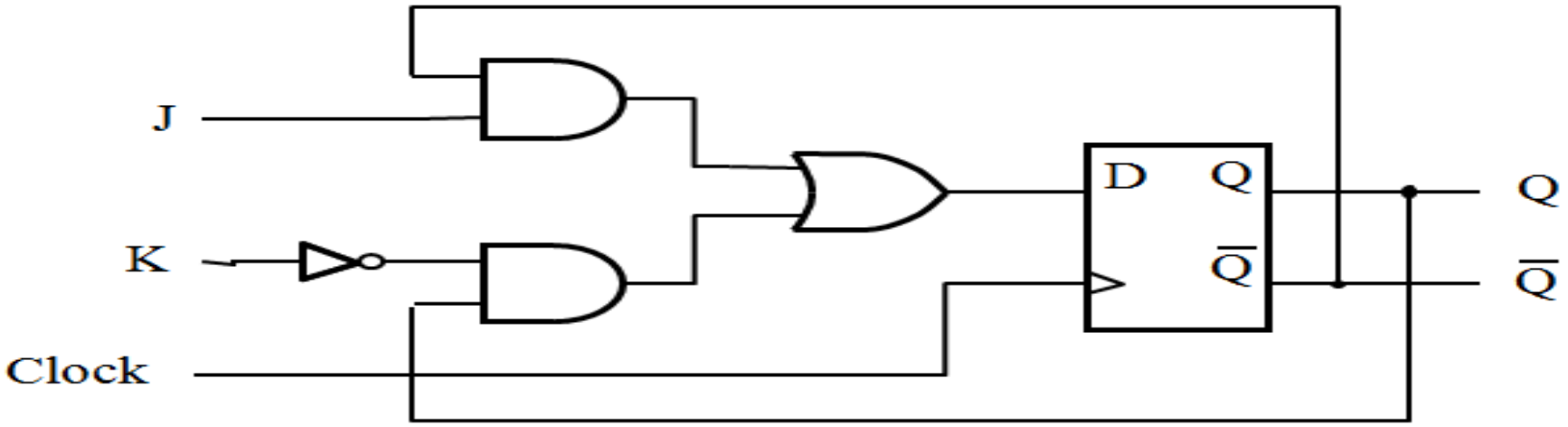


(b) Graphical symbol

Q	T	Q(t+1)
0	0	0
0	1	1
1	0	1
1	1	0

(c) Transition table

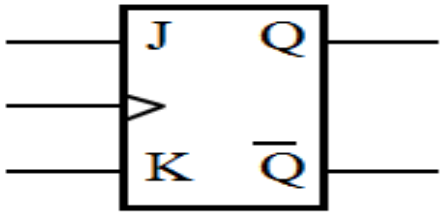
JK FLIPFLOP



(a) Circuit

J	K	Q (t+1)
0	0	Q (t)
0	1	0
1	0	1
1	1	$\overline{Q}$ (t)

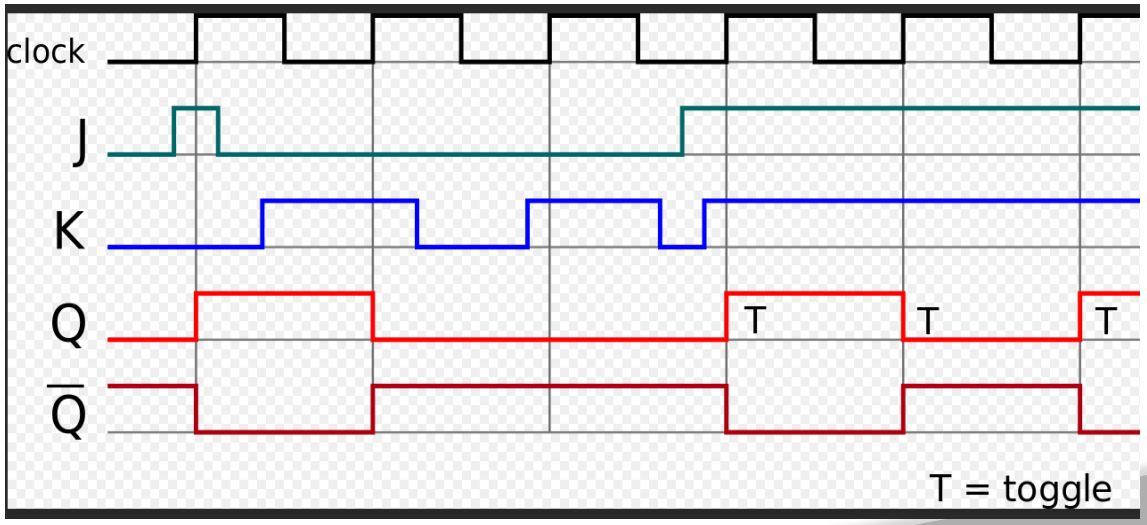
(b) Characteristic table



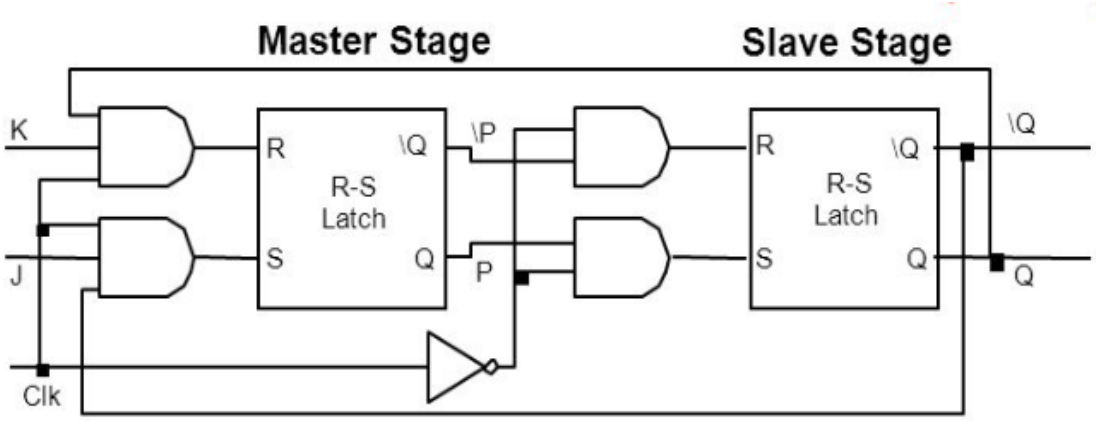
(c) Graphical symbol

JK FLIPFLOP

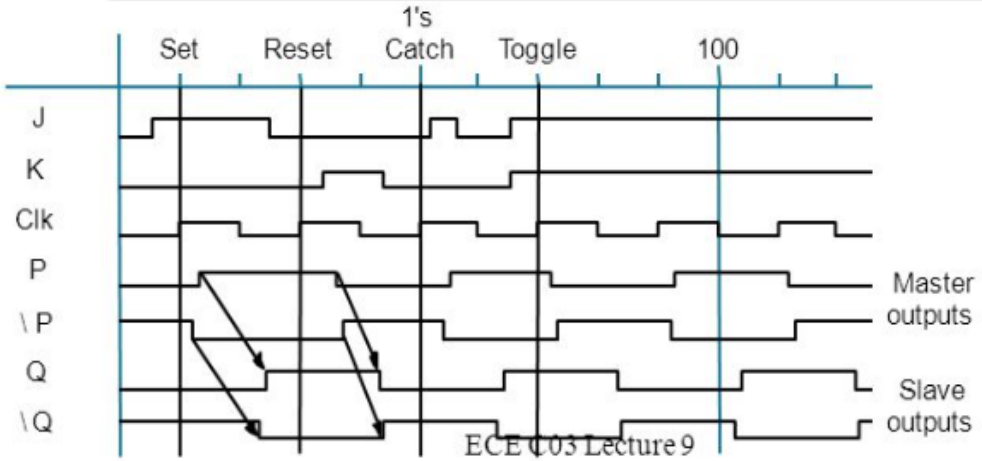
Q Output		Inputs	
Present State	Next State	J_n	K_n
0	0	0	x
0	1	1	x
1	0	x	1
1	1	x	0



Master Slave JK FLIPFLOP



Uses time to break feedback path from outputs to inputs!



Correct Toggle Operation

Jk to SR FLIPFLOP:

- **Characteristic Table**

S	R	Q(t+1)	Operation
0	0	$Q(t)$	No change
0	1	0	Reset
1	0	1	Set
1	1	?	Undefined

J	K	Q(t+1)	Operation
0	0	$Q(t)$	No change
0	1	0	Reset
1	0	1	Set
1	1	$\overline{Q}(t)$	Complement (Toggle)

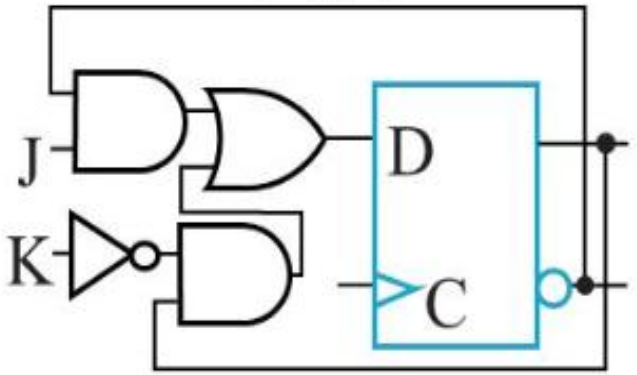
- **Characteristic Equation**

$$Q(t+1) = J\,\overline{Q} + \overline{K}\,Q$$

- **Excitation Table**

Q(t)	Q(t+1)	J	K	Operation
0	0	0	X	No change
0	1	1	X	Set
1	0	X	1	Reset
1	1	X	0	No Change

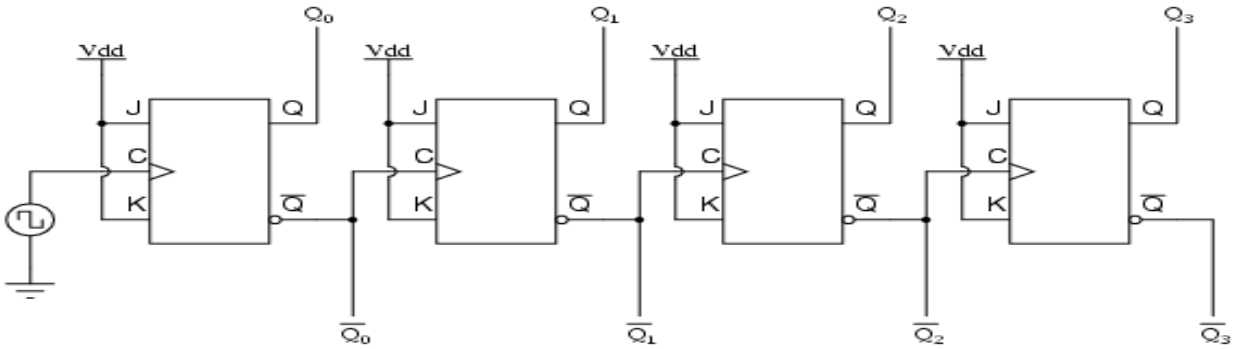
Change



Characteristic Equation:

Flip-flop	Characteristic Equation
D	$Q(t+1) = D$
T	$Q(t+1) = T \oplus Q(t)$
SR	$Q(t+1) = S + R' Q(t)$
JK	$Q(t+1) = J Q(t)' + K' Q(t)$

Asynchronous Up/Down Counters:



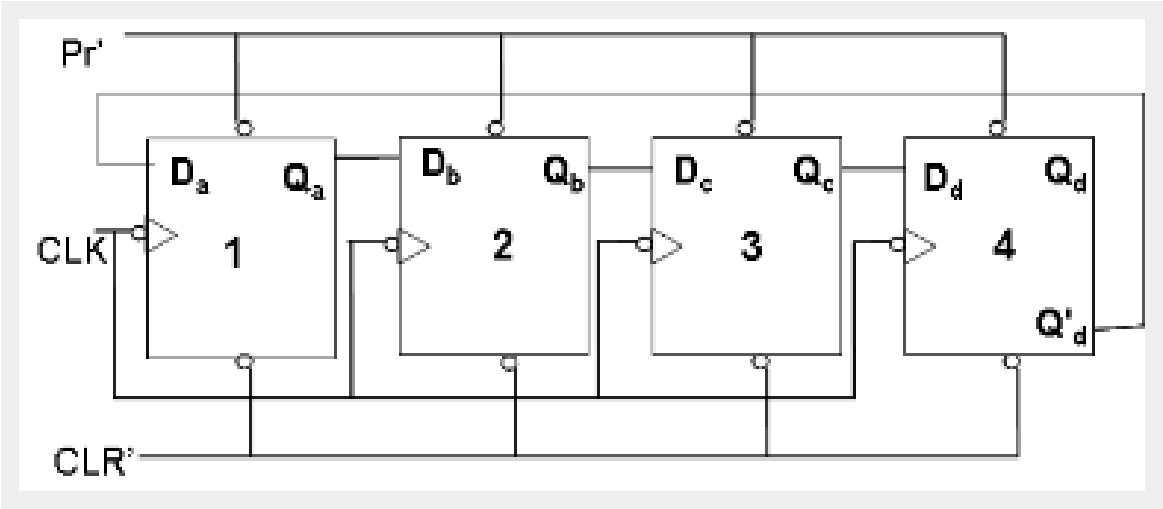
"Up" count sequence

Q_0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
Q_1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
Q_2	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
Q_3	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

"Down" count sequence

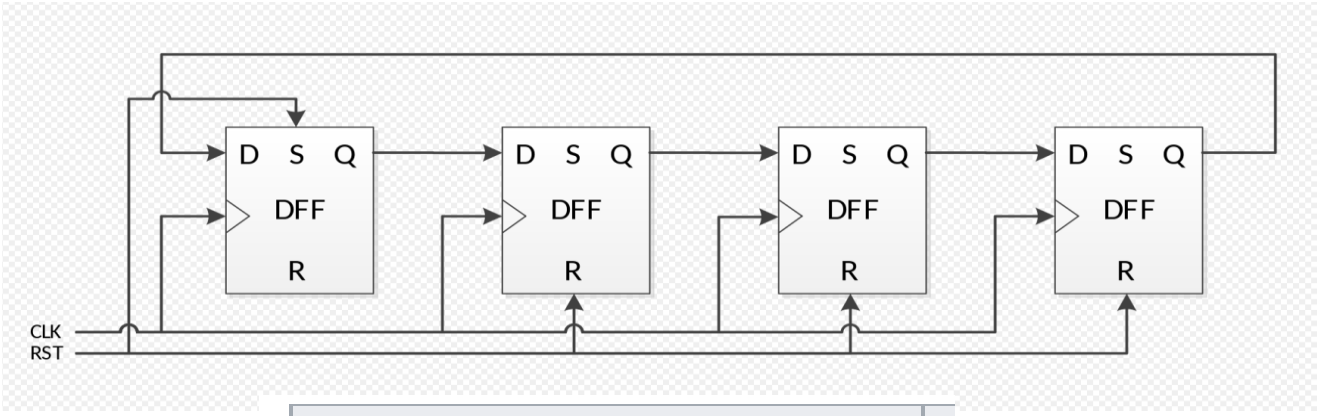
$\overline{Q_0}$	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0
$\overline{Q_1}$	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0
$\overline{Q_2}$	1	1	1	1	0	0	0	0	1	1	1	1	0	0	0	0
$\overline{Q_3}$	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0

Johnson Counters:



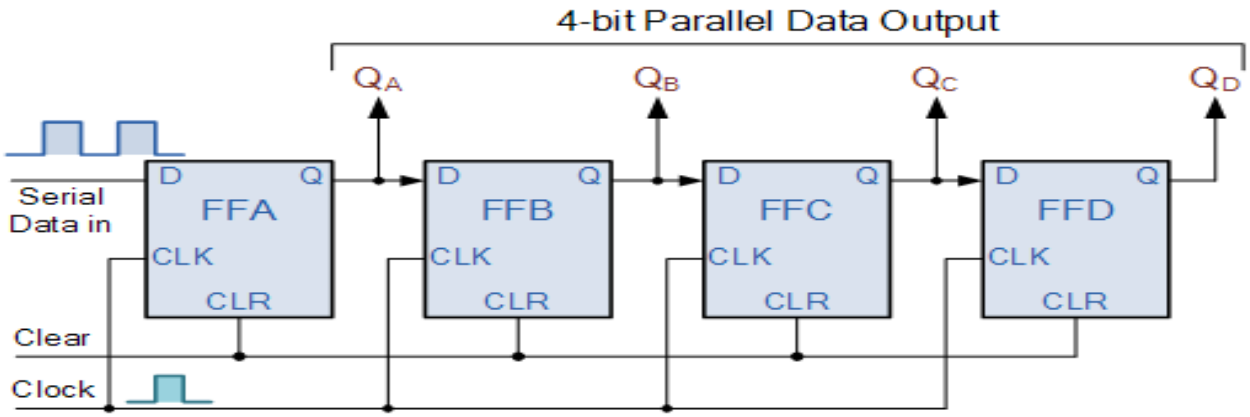
Johnson counter				
State	Q0	Q1	Q2	Q3
0	0	0	0	0
1	1	0	0	0
2	1	1	0	0
3	1	1	1	0
4	1	1	1	1
5	0	1	1	1
6	0	0	1	1
7	0	0	0	1
0	0	0	0	0

Ring Counters:



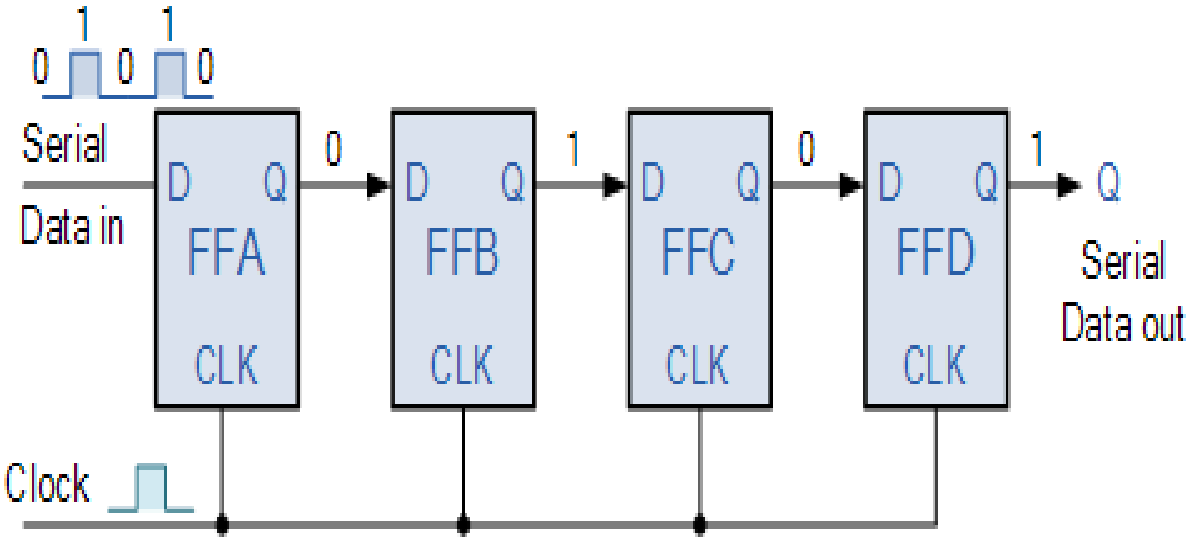
Straight ring counter				
State	Q0	Q1	Q2	Q3
0	1	0	0	0
1	0	1	0	0
2	0	0	1	0
3	0	0	0	1
0	1	0	0	0
1	0	1	0	0
2	0	0	1	0
3	0	0	0	1
0	1	0	0	0

Serial in to Parallel Output:

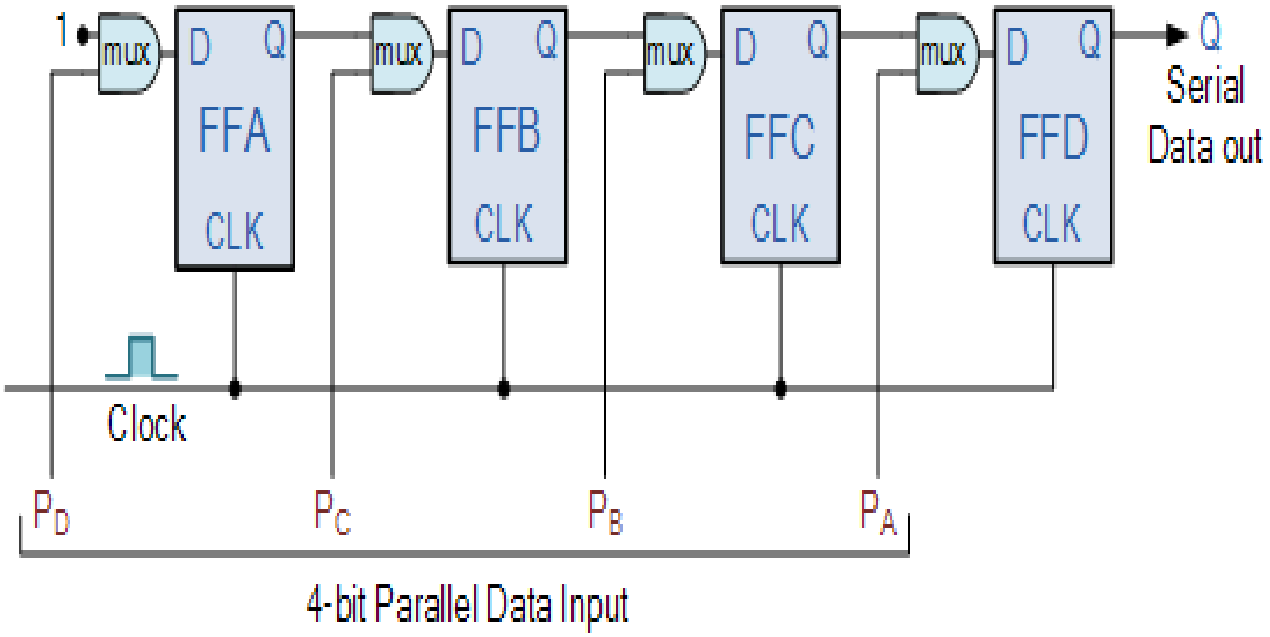


Clock Pulse No	QA	QB	QC	QD
0	0	0	0	0
1	1	0	0	0
2	0	1	0	0
3	0	0	1	0
4	0	0	0	1
5	0	0	0	0

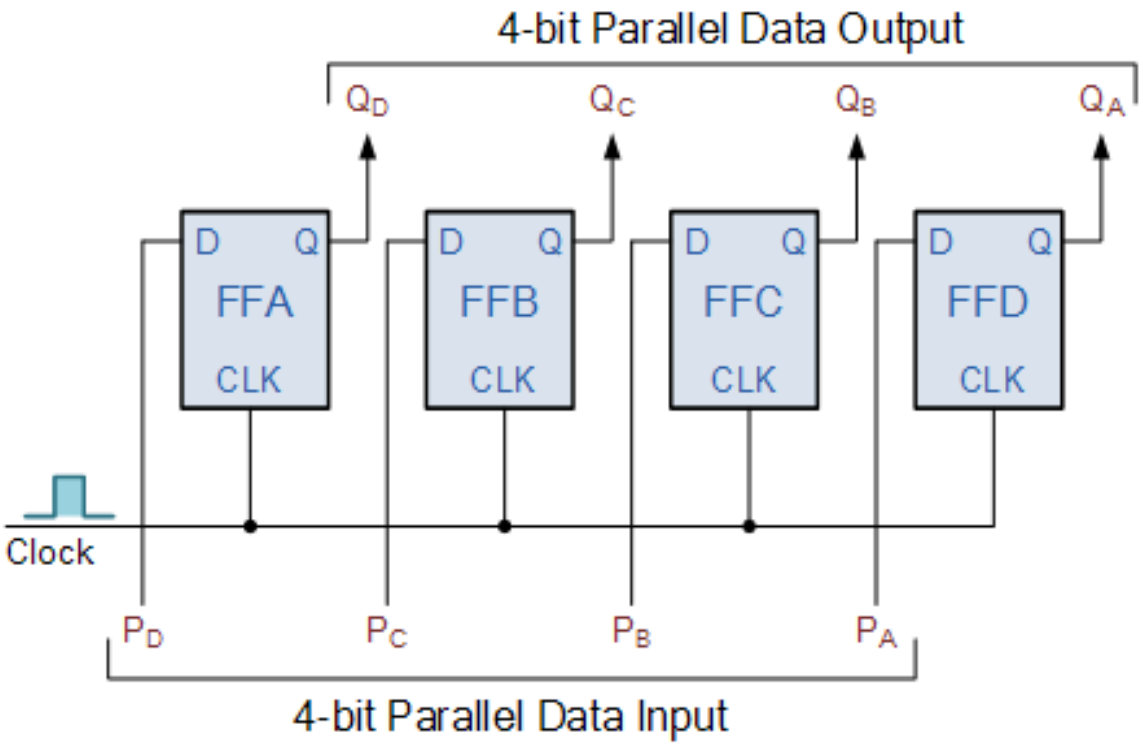
Serial in to Serial Output:



Parallel in to Serial Output:



Parallel in to Parallel Output:



Unit-IV

A/D AND D/A CONVERTERS

Data Converters:

Introduction:

- In electronics a digital to analog converter is a system that converts a digital signal into analog signal.
- An analog to digital converter is a system that converts a analog signal into digital signal.

Data Converters:

- **Classification of ADCs**
  - Direct type ADC.
  - Integrating type ADC
- **Direct type ADCs**
    - Flash (comparator) type converter
    - Counter type converter
    - Tracking or servo converter.
    - Successive approximation type converter

Data Converters:

Integrating type converters:

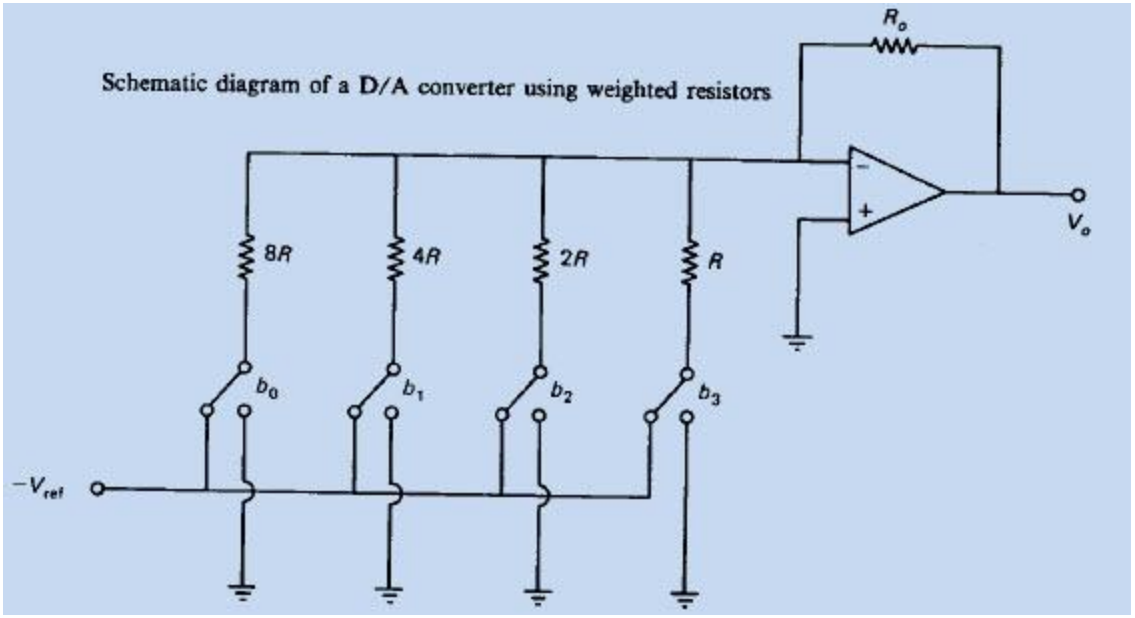
An ADC converter that perform conversion in an indirect manner by first changing the analog I/P signal to a linear function of time or frequency and then to a digital code is known as integrating type A/D converter.

DAC TECHNIQUES:

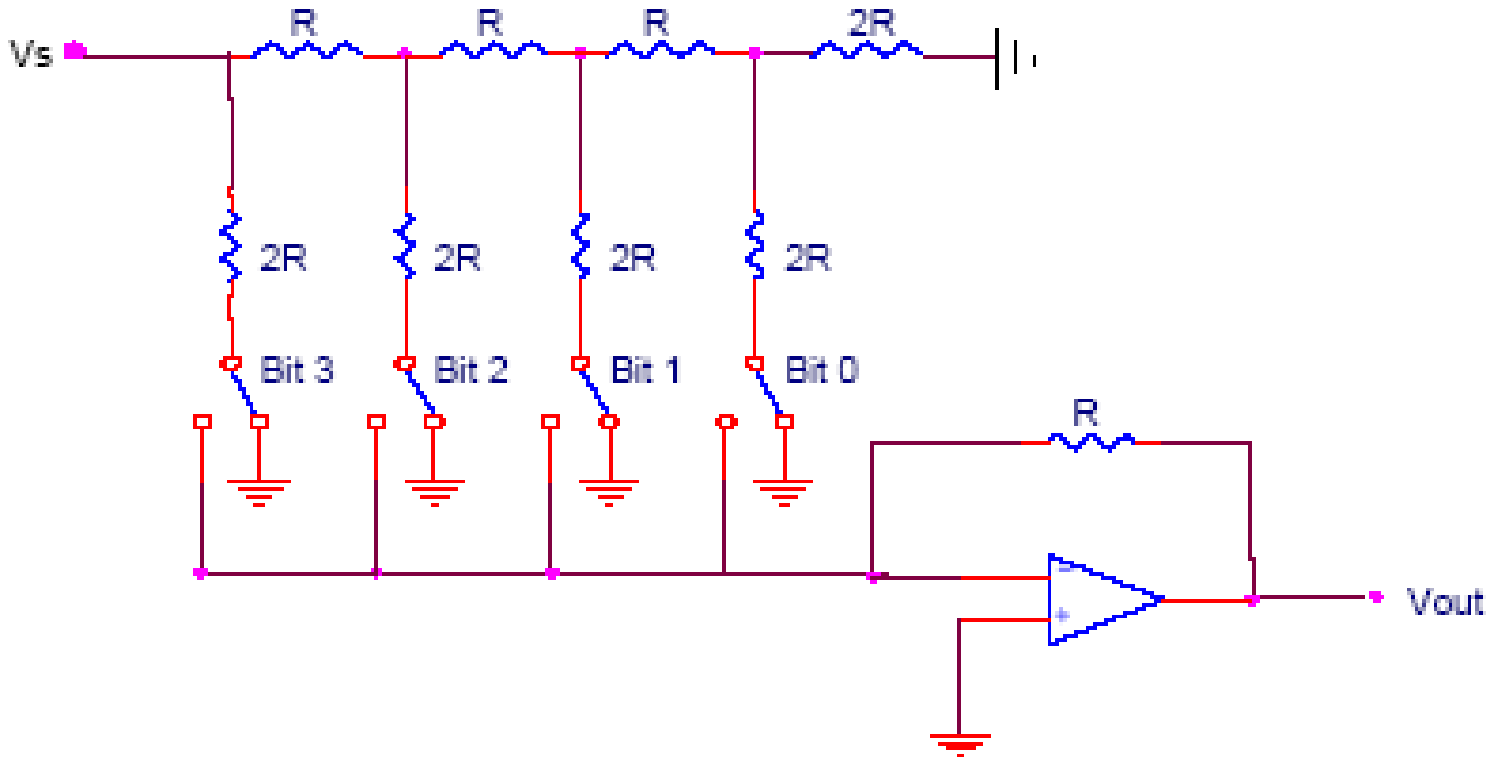
- Weighted resistor DAC
- R-2R ladder DAC
- Inverted R-2R ladder DAC
- IC 1408 DAC

DAC TECHNIQUES:

Weighted Resistor DAC

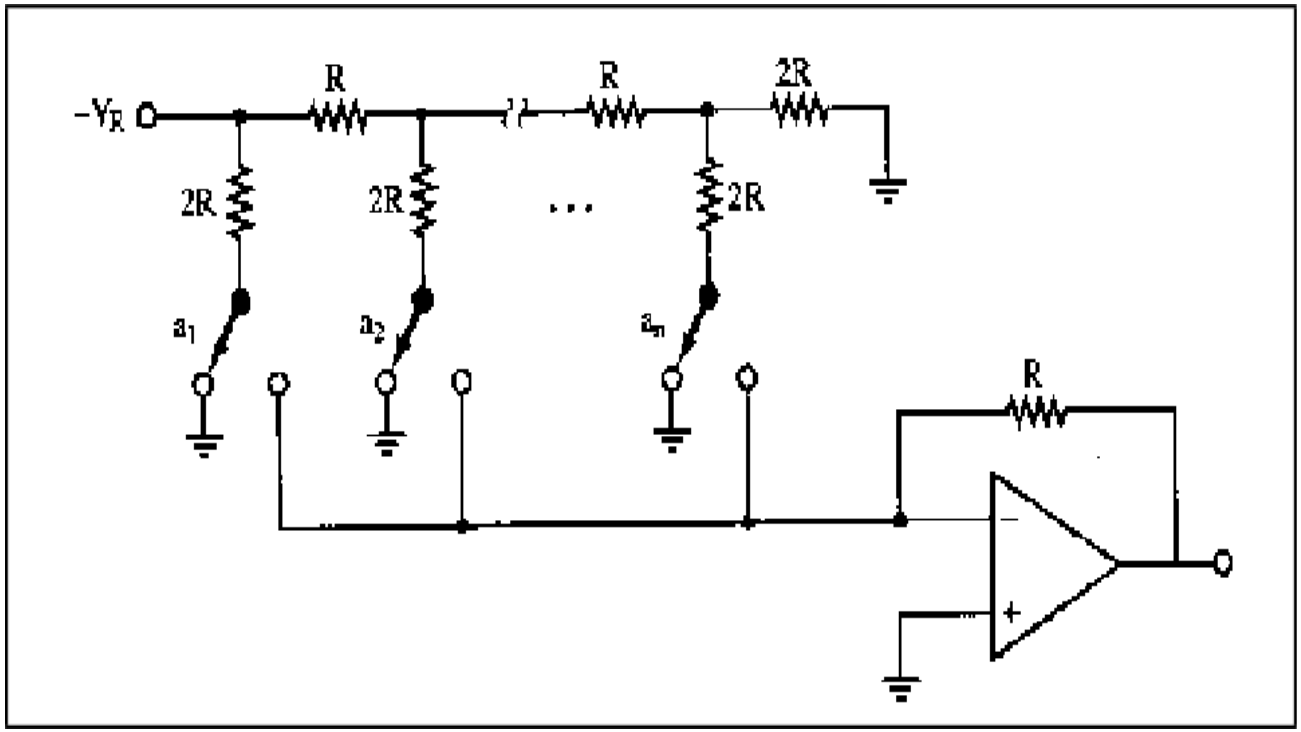


DAC TECHNIQUES:



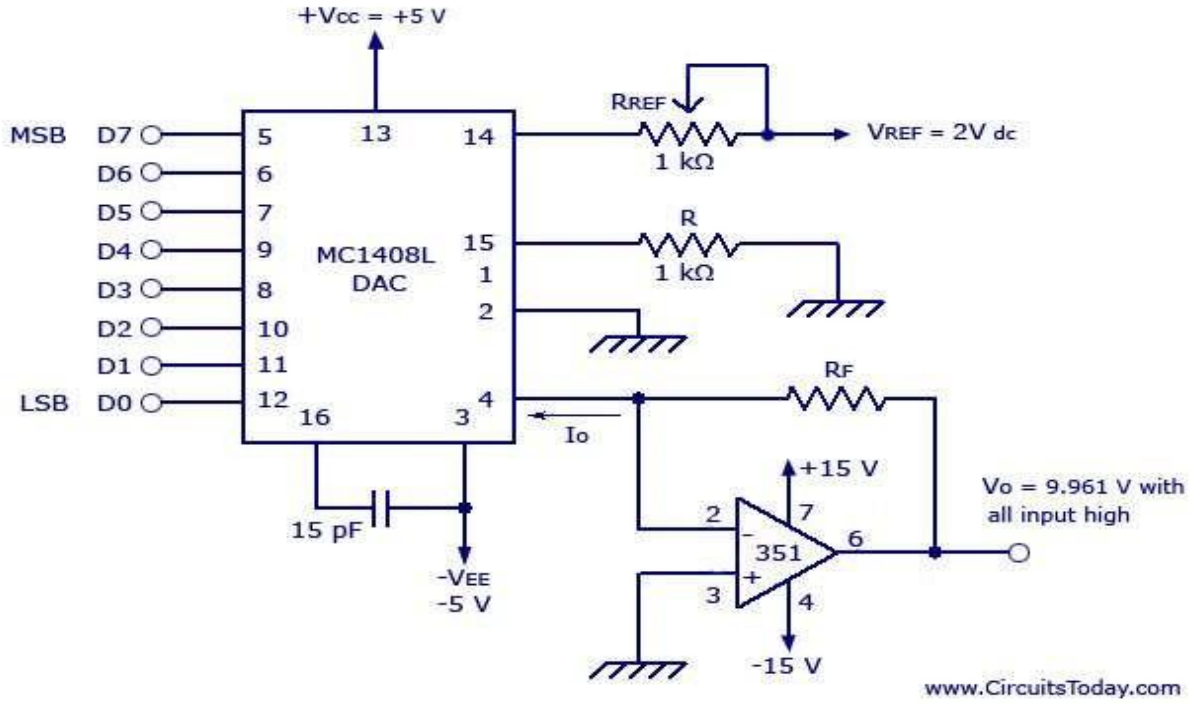
INVERTED R-2R DAC:

Inverted R-2R DAC



INVERTED R-2R DAC:

IC 1408 DAC



MC1408 D/A Converter With Current Output

INVERTED R-2R DAC:

IC 1408 DAC Specifications:

- Resolution
- Non-linearity or Linearity Error
- Gain error and Offset Error
- Settling Time

INVERTED R-2R DAC:

IC 1408 DAC Applications:

- Microcomputer interfacing
- CRT Graphics Generation
- Programmable Power Supplies
- Digitally controlled gain circuits
- Digital Filters

DAC CLASSIFICATIONS AND SPECIFICATIONS:

DAC characteristics:

- Resolution
- Reference Voltage
- Speed
- Settling Time
- Linearity

DAC CLASSIFICATIONS AND SPECIFICATIONS:

Resolution:

- The change in output voltage for a change of the LSB.
- Related to the size of the binary representation of the voltage. (8-bit)
- Higher resolution results in smaller steps between voltage values

DAC CLASSIFICATIONS AND SPECIFICATIONS:

Reference Voltage:

- Multiplier DAC

  -Reference voltage is a constant set by the manufacturer

- Non-Multiplier DAC

  -Reference voltage is variable

- Full scale Voltage

  -Slightly less than the reference voltage (V_{ref}-V_{LSB})

DAC CLASSIFICATIONS AND SPECIFICATIONS:

Speed:

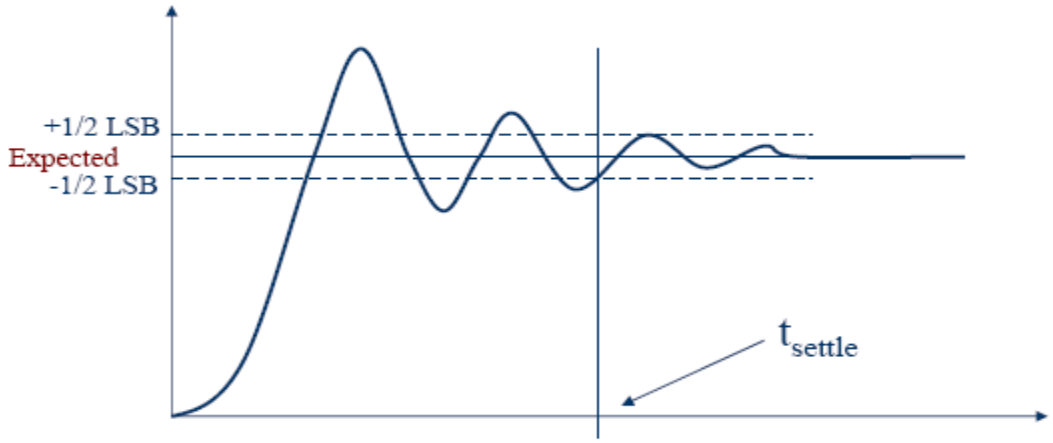
- Also called the conversion rate or sampling rate –rate at which the register value is updated.
- For sampling rates of over 1 MHz a DAC is designated as high speed.

- Speed is limited by the clock speed of the microcontroller and the settling time of the DAC.

DAC CLASSIFICATIONS AND SPECIFICATIONS:

Settling Time:

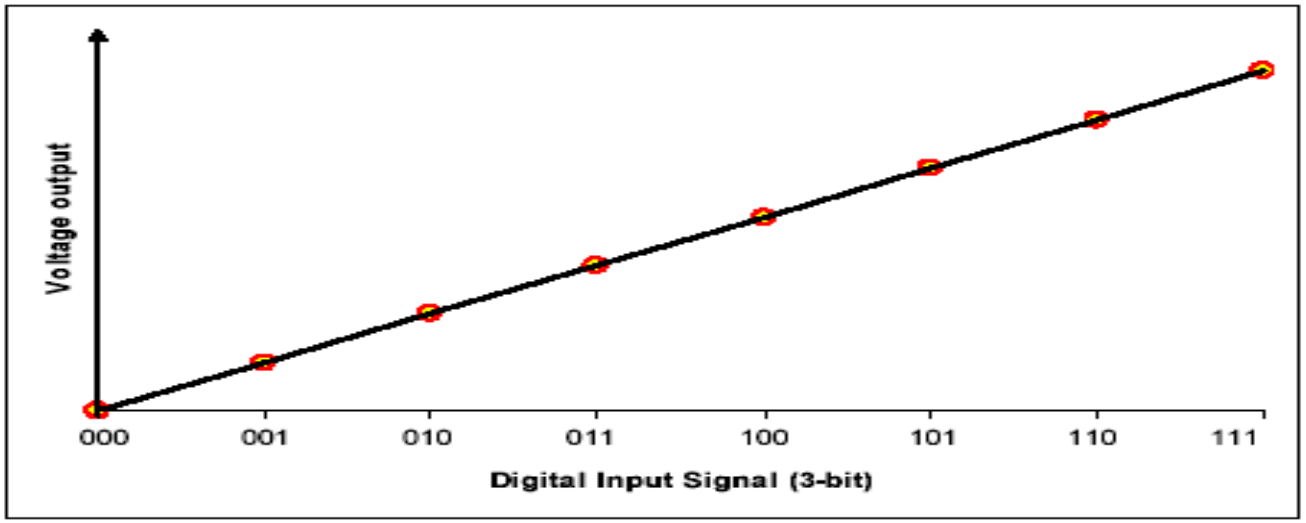
- Time in which the DAC output settles at the desired value ± ½ V_{LSB}.
- Faster DACs decrease the settling time.



DAC CLASSIFICATIONS AND SPECIFICATIONS:

Linearity:

- Represents the relationship between digital values and analog outputs.
- Should be related by a single proportionality constant. (constant slope).



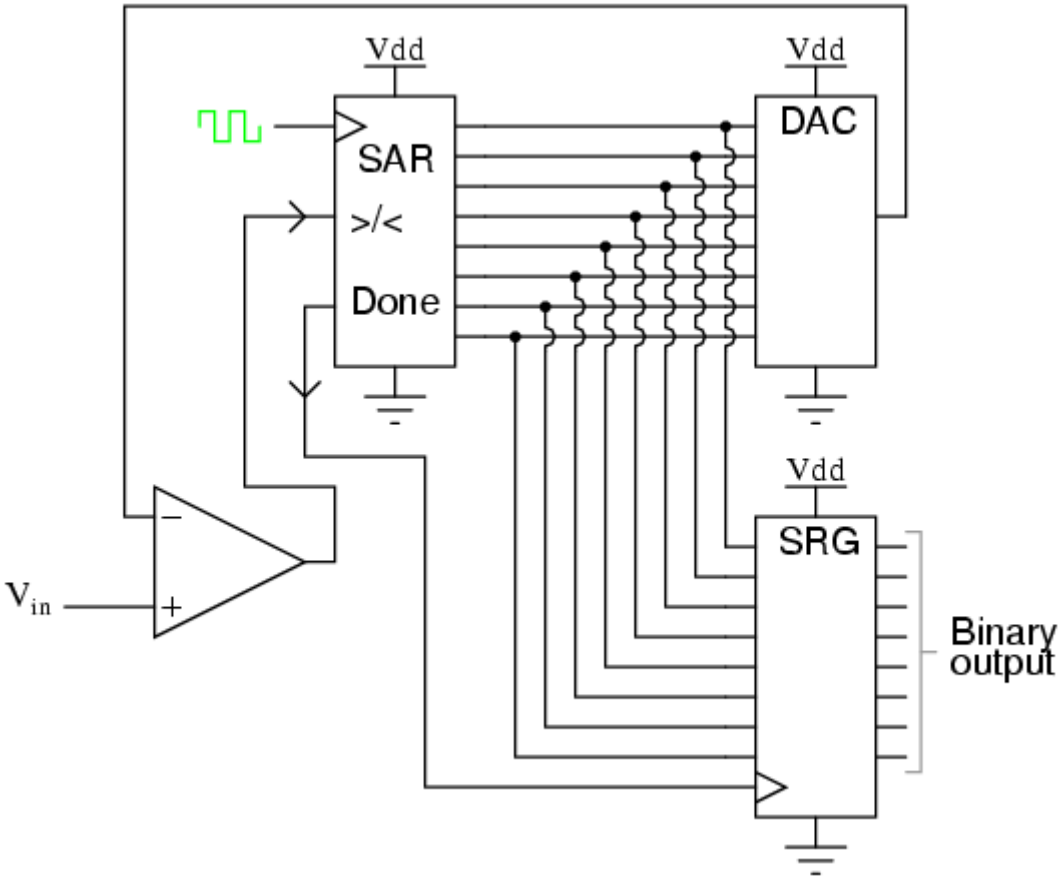
ADC:

ADC Techniques:

- Flash ADC
- Sigma-delta ADC
- Dual slope converter
- Successive approximation converter

SUCCESSIVE APPROXIMATION :

- A Successive Approximation Register (SAR) is added to the circuit
- Instead of counting up in binary sequence, this register counts by trying all values of bits starting with the MSB and finishing at the LSB.
- The register monitors the comparators output to see if the binary count is greater or less than the analog signal input and adjusts the bits accordingly


Vdd
Vdd
SAR
>/<
Done
DAC
Vdd
SRG
Binary
output
V_{in}
−
+

SUCCESSIVE APPROXIMATION :

Advantages

- Capable of high speed and reliable
- Medium accuracy compared to other ADC types
- Good tradeoff between speed and cost
- Capable of outputting the binary number in serial (one bit at a time) format.

SUCCESSIVE APPROXIMATION :

Disadvantages

- Higher resolution successive approximation ADC's will be slower
- Speed limited to ~5Msamples/s

FLASH CONVERSION:

FLASH CONVERTERS

- Consists of a series of comparators, each one comparing the input signal to a unique reference voltage.
- The comparator outputs connect to the inputs of a priority encoder circuit, which produces a binary output

FLASH CONVERSION:

FLASH CONVERTERS

- Consists of a series of comparators, each one comparing the input signal to a unique reference voltage.
- The comparator outputs connect to the inputs of a priority encoder circuit, which produces a binary output

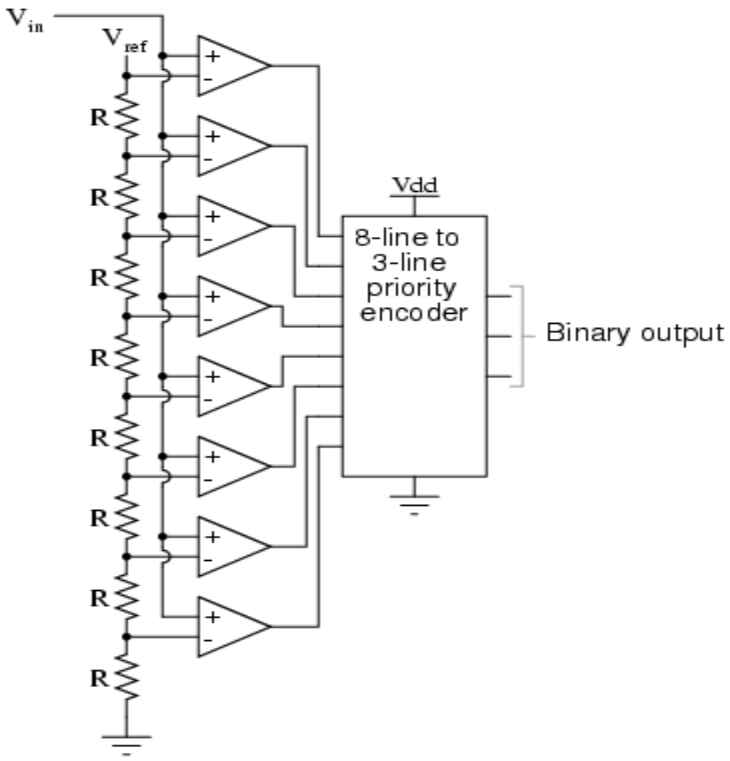
FLASH CONVERSION:

FLASH CONVERTERS

- Consists of a series of comparators, each one comparing the input signal to a unique reference voltage.
- The comparator outputs connect to the inputs of a priority encoder circuit, which produces a binary output

FLASH CONVERSION:

FLASH CONVERTERS



FLASH CONVERSION:

FLASH CONVERTERS

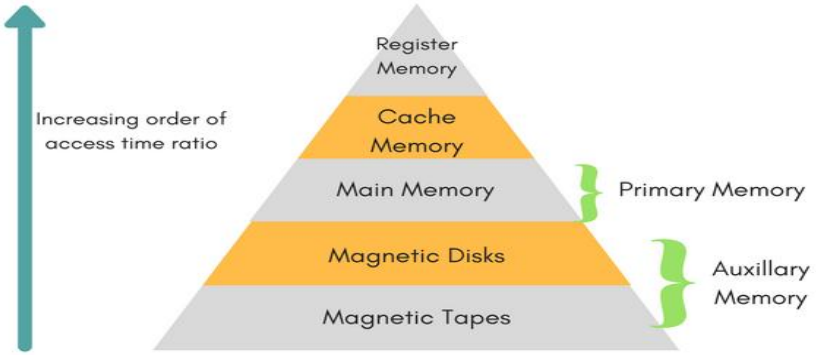
- As the analog input voltage exceeds the reference voltage at each comparator, the comparator outputs will sequentially saturate to a high state.
- The priority encoder generates a binary number based on the highest-order active input, ignoring all other active inputs.

MODULE-V

SEMICONDUCTOR MEMORIES AND PROGRAMMABLE LOGIC DEVICES

Memory Organization:

- A memory unit is the collection of storage units or devices together. The memory unit stores the binary information in the form of bits. Memory/storage is classified into 2 categories:
- Volatile Memory: This loses its data, when power is switched off.
- Non-Volatile Memory: This is a permanent storage and does not lose any data when power is switched off.



Memory :

- Auxiliary memory access time is generally 1000 times that of the main memory, hence it is at the bottom of the hierarchy.
- The main memory occupies the central position because it is equipped to communicate directly with the CPU and with auxiliary memory devices through Input/output processor (I/O).
- he cache memory is used to store program data which is currently being executed in the CPU.

Memory Access Method :

- Each memory type, is a collection of numerous memory locations. To access data from any memory, first it must be located and then the data is read from the memory location. Following are the methods to access information from memory locations:
- **Random Access**: Main memories are random access memories, in which each memory location has a unique address. Using this unique address any memory location can be reached in the same amount of time in any order.
- **Sequential Access**: This methods allows memory access in a sequence or in order.
- **Direct Access**: In this mode, information is stored in tracks, with each track having a separate read/write head.

Main Memory :

- The memory unit that communicates directly within the CPU, Auxillary memory and Cache memory, is called main memory. It is the central storage unit of the computer system. It is a large and fast memory used to store data during computer operations. Main memory is made up of RAM and ROM, with RAM integrated circuit chips holing the major share.
- RAM: Random Access Memory
  - DRAM: Dynamic RAM, is made of capacitors and transistors, and must be refreshed every 10~100 ms. It is slower and cheaper than SRAM.
  - SRAM: Static RAM, has a six transistor circuit in each cell and retains data, until powered off.

Main Memory :

- **Auxiliary Memory**
- Devices that provide backup storage are called auxiliary memory. **For example:** Magnetic disks and tapes are commonly used auxiliary devices. Other devices used as auxiliary memory are magnetic drums, magnetic bubble memory and optical disks.
- It is not directly accessible to the CPU, and is accessed using the Input/Output channels.

Main Memory :

- Memory size

  The size of memory sticks these days range from 64 MB up to 32 GB giving you plenty of options when it comes to choosing the size of storage space on your memory stick.

- Content-addressable memory (CAM) :Content-addressable memory (CAM) is a special type of computer memory used in certain very-high-speed searching applications. It is also known as associative memory, associative storage, or associative array, although the last term is more often used for a programming data structure.

Main Memory :

- A charge-coupled device (CCD) is a device for the movement of electrical charge, usually from within the device to an area where the charge can be manipulated, for example conversion into a digital value. This is achieved by "shifting" the signals between stages within the device one at a time.
- In recent years CCD has become a major technology for digital imaging (MOS) capacitors. These capacitors are biased above the threshold for inversion when image acquisition begins, allowing the conversion of incoming photons into electron charges at the semiconductor-oxide interface;
- the CCD is then used to read out these charges. exacting quality demands, such as consumer and professional digital cameras, active pixel sensors, also known as complementary metal-oxide-semiconductors (CMOS) are generally used; the large quality advantage CCDs enjoyed early on has narrowed over time.

Programmable logic device

- A programmable logic device (PLD) is an electronic component used to build reconfigurable digital circuits. Unlike integrated circuits (IC) which consist of logic gates and have a fixed function, a PLD has an undefined function at the time of manufacture. Before the PLD can be used in a circuit it must be programmed

FPGA:

- FPGAs use a grid of logic gates, and once stored, the data doesn't change, similar to that of an ordinary gate array. The term "field-programmable" means the device is programmed by the customer, not the manufacturer.
- FPGAs are usually programmed after being soldered down to the circuit board, in a manner similar to that of larger CPLDs. In most larger FPGAs, the configuration is volatile and must be re-loaded into the device whenever power is applied or different functionality is required. Configuration is typically stored in a configuration PROM or EEPROM. EEPROM versions may be in-system programmable (typically via JTAG).