DIGITAL LOGIC DESIGN PPT

INSTITUTE OF AERONAUTICAL ENGINEERING

(Autonomous)

Dundigal, Hyderabad -500 043

COMPUTER SCIENCE AND ENGINEERING/INFORMATION TECHNOLOGY

DIGITAL LOGIC DESIGN PPT

AEC020

Course Coordinator

Mr. K.Ravi, Assistant Professor, ECE
Ms. G. Bhavana, Assistant Professor, ECE
Ms. L.Shruthi, Assistant Professor, ECE
Ms. V.Bindusree, Assistant Professor, ECE
Ms. J.Swetha, Assistant Professor, ECE
Ms. Shreya verma, Assistant Professor, ECE

UNIT 1

INTRODUCTION TO DIGITAL LOGIC DESIGN

INTRODUCTION TO DIGITAL LOGIC DESIGN

Digital logic design is a system in electrical and computer engineering that uses simple number values to produce input and output operations.

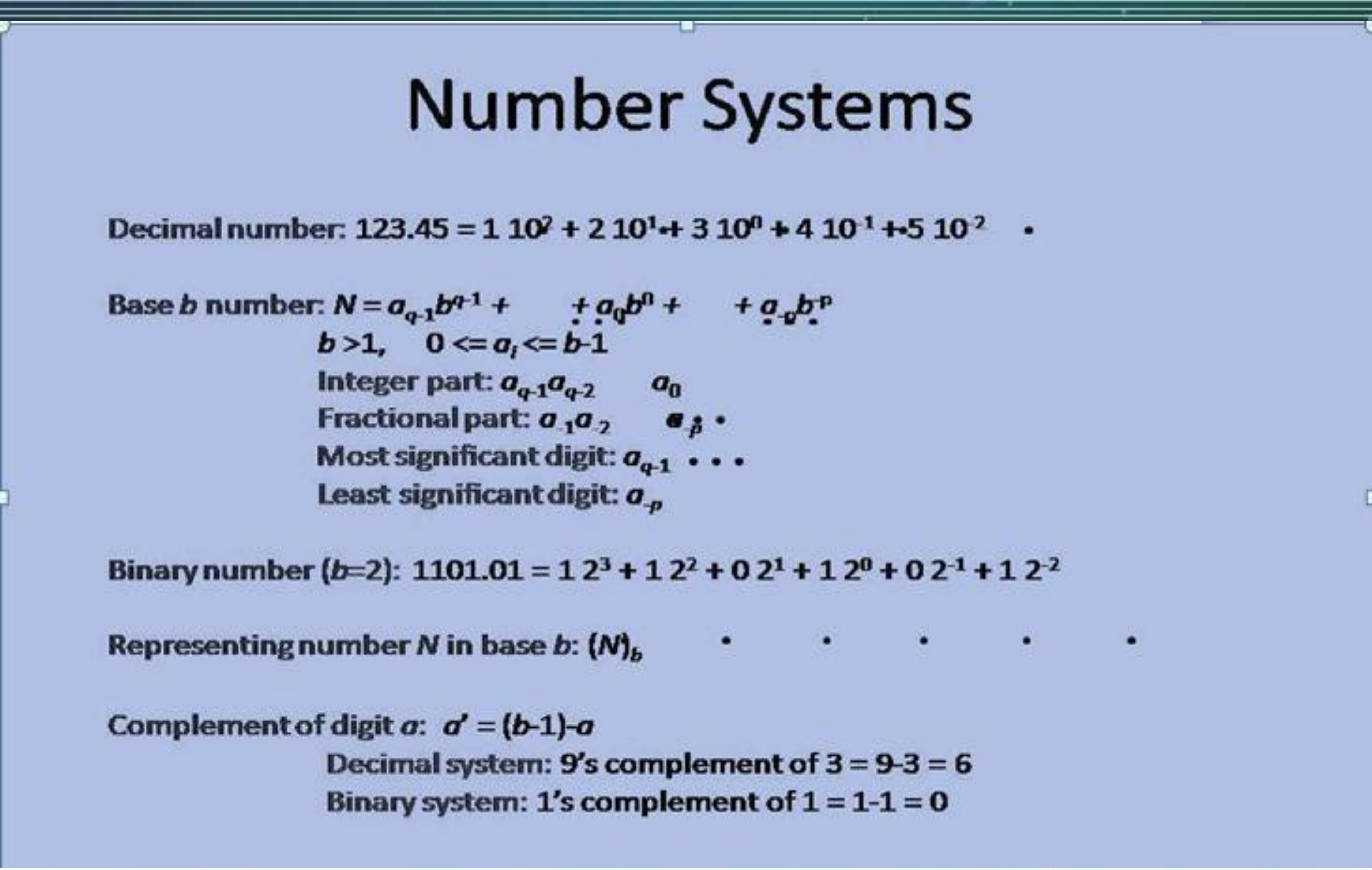
INTRODUCTION TO DIGITAL LOGIC DESIGN

Advantages:

•A digital computer stores data in terms of digits (numbers) and proceeds in discrete steps from one state to the next.

•The states of a digital computer typically involve binary digits which may take the form of the presence or absence of magnetic markers in a storage medium , on-off switches or relays. In digital computers, even letters, words and whole texts are represented digitally.

NUMBER SYSTEMS



Number Systems

Decimal number: $123.45 = 1\ 10^2 + 2\ 10^1 + 3\ 10^0 + 4\ 10^{-1} + 5\ 10^{-2}$ •

Base b number: $N = a_{q-1}b^{q-1} + \cdots + a_0b^0 + \cdots + a_{-p}b^{-p}$

$b > 1, \quad 0 <= a_i <= b-1$

Integer part: $a_{q-1}a_{q-2} \cdots a_0$

Fractional part: $a_{-1}a_{-2} \cdots a_{-p}$ •

Most significant digit: a_{q-1} • • •

Least significant digit: a_{-p}

Binary number (b=2): $1101.01 = 1\ 2^3 + 1\ 2^2 + 0\ 2^1 + 1\ 2^0 + 0\ 2^{-1} + 1\ 2^{-2}$

Representing number N in base b: $(N)_b$ • • • • •

Complement of digit a: $a' = (b-1)-a$

Decimal system: 9's complement of 3 = 9-3 = 6

Binary system: 1's complement of 1 = 1-1 = 0

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.

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 octal digit.

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.

NUMBER BASE CONVERSION

Decimal to Binary conversion:

I.method: is for small no.s The values of various powers of 2 need to be remembered. for conversion of larger no.s have a table of powers of 2 known as the sum of weights method. The set of binary weight values whose sum is equal to the decimal no. is determined.

II.method : It converts decimal integer no. to binary integer no by successive division by 2 & the decimal fraction is converted to binary fraction by **double –dabble method**

NUMBER BASE CONVERSION

Octal to decimal Conversion:

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

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

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 give the hexadecimal fraction known as **the hexadabble method.**

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.

FINDING THE BASE OF THE NUMBER SYSTEM

- Find *r* such that (121)r=(144)8, where r *and* 8 are the bases

$1*8^2 + 4*8+4*8^0 \quad =64+32+4 =100$

$1*r^2+2*r+1*r^0 = r^2+2r+1=(r+1)^2$

$(r+1)^2=100$

$r+1=10$

$r=9$

BINARY ARITHMETIC

Binary Addition:

Rules:

0+0=0

0+1=1

1+0=1

1+1=10

i.e, 0 with a carry of 1.

BINARY ARITHMETIC

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 ARITHMETIC

Binary Division:

Example : 101101_2 by 110

```
110 )    101101    (  111.1
            110
         ________
           1010
         110
         ________
            1001
              110
___________
           110
           110
         _________
         000
```

Ans: 111.1

BINARY ARITHMETIC

9's & 10's Complements:

It is the Subtraction of decimal no.s 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 no. is obtained by subtracting each digit of that decimal no. from 9. The 10's compliment of a decimal no is obtained by adding a 1 to its 9's compliment.

BINARY ARITHMETIC

1’s compliment of n number:

It is obtained by simply complimenting each bit of the no,.& also , 1’s comp of a no, is subtracting each bit of the no. form 1.This complemented value rep the –ve of the original no. One of the difficulties of using 1’s comp is its rep o f zero.Both 00000000 & its 1’s comp 11111111 rep zero.The 00000000 called +ve zero& 11111111 called –ve zero.

BINARY ARITHMETIC

1’s compliment arithmetic:

In 1’s comp subtraction, add the 1’s comp 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 +ve & is in true binary. If the MSB is a 1 (carry or no carry), the result is –ve & is in its is comp form .Take its 1’s comp to get the magnitude inn binary.

BINARY ARITHMETIC

9’s & 10’s Complements:

It is the Subtraction of decimal no.s 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 no. is obtained by subtracting each digit of that decimal no. from 9. The 10’s compliment of a decimal no is obtained by adding a 1 to its 9’s compliment.

BINARY ARITHMETIC

Methods of obtaining 2’s comp of a no:

In 3 ways

By obtaining the 1’s comp of the given no. (by changing all 0’s to 1’s & 1’s to 0’s) & then adding 1.

By subtracting the given n bit no N from 2^n

Starting at the LSB , copying down each bit upto & including the first 1 bit encountered , and complimenting the remaining bits.

BINARY ARITHMETIC

2’s compliment Arithmetic:

The 2’s comp system is used to rep –ve no.s using modulus arithmetic . The word length of a computer is fixed. i.e, if a 4 bit no. is added to another 4 bit no . the result will be only of 4 bits. Carry if any , from the fourth bit will overflow called the Modulus arithmetic.

Ex:1100+1111=1011

9’s & 10’s Complements:

.

It is the Subtraction of decimal no.s 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 no. is obtained by subtracting each digit of that decimal no. from 9. The 10’s compliment of a decimal no is obtained by adding a 1 to its 9’s compliment

BINARY ARITHMETIC

1’s compliment of n number:

It is obtained by simply complimenting each bit of the no,.& also , 1’s comp of a no, is subtracting each bit of the no. form 1.This complemented value rep the –ve of the original no. One of the difficulties of using 1’s comp is its rep o f zero.Both 00000000 & its 1’s comp 11111111 rep zero.The 00000000 called +ve zero& 11111111 called –ve zero.

1’s compliment arithmetic:

In 1’s comp subtraction, add the 1’s comp 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 +ve & is in true binary. If the MSB is a 1 (carry or no carry), the result is –ve & is in its is comp form .Take its 1’s comp to get the magnitude inn binary.

BINARY ARITHMETIC

9’s & 10’s Complements:

It is the Subtraction of decimal no.s 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 no. is obtained by subtracting each digit of that decimal no. from 9. The 10’s compliment of a decimal no is obtained by adding a 1 to its 9’s compliment.

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-weightedcodes:

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 no,s 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}(0100) 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.

BINARY CODED DECIMAL

Excess-3Addition:

Add the xs-3 no.s by adding the 4 bit groups in each column starting from the LSD. If there is no carry starting from the addition of any of the 4-bit groups , subtract 0011 from the sum term of those groups (because when 2 decimal digits are added in xs-3 & there is no carry , result in xs-6). If there is a carry out, add 0011 to the sum term of those groups(because when there is a carry, the invalid states are skipped and the result is normal binary).

BINARY CODED DECIMAL

Excess -3 (XS-3) Subtraction:

Subtract the xs-3 no.s by subtracting each 4 bit group of the subtrahend from the corresponding 4 bit group of the minuend starting form the LSD .if there is no borrow from the next 4-bit group add 0011 to the difference term of such groups (because when decimal digits are subtracted in xs-3 & there is no borrow , result is normal binary). I f there is a borrow , subtract 0011 from the difference term(b coz taking a borrow is equivalent to adding six invalid states , result is in xs-6)

SIGNED BINARY NUMBERS

Representation of signed no.s binary arithmetic in computers:

Two ways of rep signed no.s

Sign Magnitude form

Complemented form

Two complimented forms

1’s compliment form

2’s compliment form

SIGNED BINARY NUMBERS

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 & 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 AND CORRECTING CODES

- Introduction:
  - 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 AND CORRECTING CODES

- Some Common Error Detecting and Correcting Codes
  - Parity Code
  - Repetition Code
  - Cyclic Redundancy Check Code
  - Hamming Code

ERROR DETECTING AND CORRECTING CODES

- **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.
  - This simple parity code suffers from two limitations. Firstly, it cannot detect the error if the number of bits having undergone a change is even.
- **Repetition Code:**
  - The repetition code makes use of repetitive transmission of each data bit in the bit stream. In the case of threefold repetition, ‘1’ and ‘0’ would be transmitted as ‘111’ and ‘000’ respectively.
  - The repetition code is highly inefficient and the information throughput drops rapidly as we increase the number of times each data bit needs to be repeated to build error detection and correction capability.

ERROR DETECTING AND CORRECTING CODES

- **Cyclic Redundancy Check Code:**
  - Cyclic redundancy check (CRC) codes provide a reasonably high level of protection at low redundancy level.

  - The probability of error detection depends upon the number of check bits, n used to construct the cyclic code. It is 100 % for single-bit and two-bit errors. It is also 100 % when an odd number of bits are in error and the error bursts have a length less than . $n + 1$
  - The probability of detection reduces to $1-(1/2)^{n-1}$ for an error burst length equal to $n + 1$, and to $1-(1/2)^{n}$ for an error burst length greater than $n + 1$.

ERROR DETECTING AND CORRECTING 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, which has come to be known as the Hamming distance.
- **Hamming Distance:**
  - The Hamming distance is nothing but the number of bit disagreements between two code words.

ERROR DETECTING AND CORRECTING 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 AND CORRECTING CODES

- The algorithm for writing the generalized Hamming code is as follows:

1. 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.

2. 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.

3. All other bit positions (positions 3, 5, 6, 7, 9, 10, 11 ...) are used to encode data.

4. 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 AND CORRECTING CODES

1. 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.

2. For example, for the generalized form of code given above, various groups of bits formed with different parity bits would be $P_1D_1D_2D_4D_{5....}$, $P_2D_1D_3D_4D_6D_{7....}$, $P_3D_2D_3D_4D_8D_{9....}$, $P_4D_5D_6D_7D_8D_9D_{10}D_{11....}$ and so on. To illustrate the formation of groups further, let us examine the group corresponding to parity bit P_3.

3. 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)*.

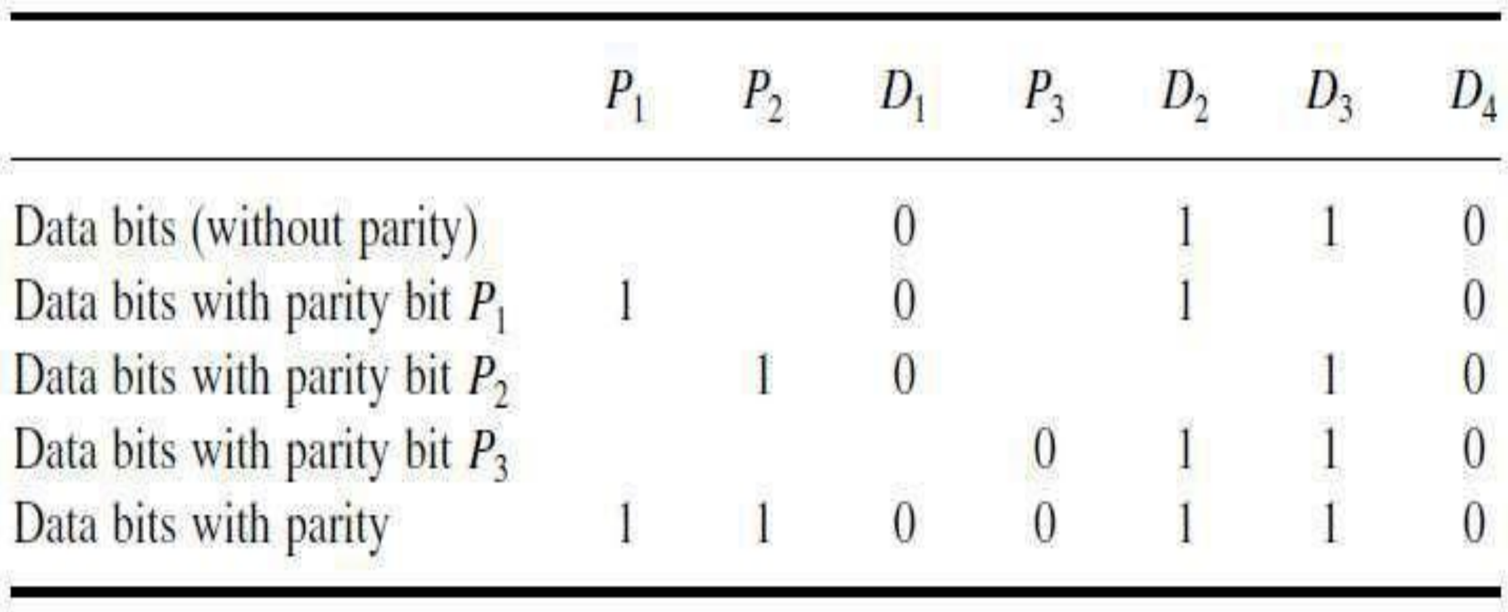
ERROR DETECTING AND CORRECTING CODES

- 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.
- The number of parity bits n required to encode m message bits is the smallest integer that satisfies the condition $(2^n - n) > m$.
- The most commonly used Hamming code is the one that has a code word length of seven bits with four message bits and three parity bits.
- It is also referred to as the Hamming (7, 4) code

ERROR DETECTING AND CORRECTING CODES

- 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

UNIT 2

INTRODUCTION TO BOOLEAN ALGEBRA

BOOLEAN ALGEBRA

- Also known as Switching Algebra
  - › Invented by mathematician George Boole in 1849
  - › Used by Claude Shannon at Bell Labs in 1938
    - To describe digital circuits built from relays
- Digital circuit design is based on
  - › Boolean Algebra
    - Attributes
    - Postulates
    - Theorems
  - › These allow minimization and manipulation of logic gates for optimizing digital circuits

BOOLEAN ALGEBRA ATTRIBUTES

- 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 ATTRIBUTES

- **Variable**: Variables are the different symbols in a Boolean expression
- **Literal:** Each occurrence of a variable or its complement is called a literal
- **Term**: A term is the expression formed by literals and operations at one level

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

  - A, B, C are three variables
  - Eight Literals
  - Expression has five terms including four AND terms and the OR term that combines the first-level AND terms.

BOOLEAN ALGEBRA POSTULATES

- Identity Elements
  - P2a: X+0=X
  - P2b: X•1=X
- Commutativity
  - P3a: X+Y=Y+X
  - P3b: X•Y=Y•X
- Complements
  - P6a: X+X’=1
  - P6b: 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 POSTULATES

- Associativity
  - P4a: (X+Y)+Z=X+(Y+Z)
  - P4b: (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 POSTULATES

- Distributivity
  - P5a: X+(Y•Z) = (X+Y)•(X+Z)
  - P5b: 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 THEOREMS

- Idempotency
  - T1a: X+X=X
  - T1b: X•X=X
- Null elements
  - T2a: X+1=1
  - T2b: X•0=0
- Involution
  - T3: (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 THEOREMS

- Absorption (aka *covering*)
  - T4a: X+(X•Y)=X
  - T4b: X•(X+Y)=X
  - T5a: X+(X’•Y)=X+Y
  - T5b: X•(X’+Y)=X•Y

		OR	AND							
X	Y	X+Y	X•Y	X+ (X•Y)	X• (X+Y)	X’	X’•Y	X+ (X’•Y)	X’+Y	X• (X’+Y)
0	0	0	0	0	0	1	0	0	1	0
0	1	1	0	0	0	1	1	1	1	0
1	0	1	0	1	1	0	0	1	0	0
1	1	1	1	1	1	0	0	1	1	1

BOOLEAN ALGEBRA THEOREMS

- Absorption (aka *combining*)
  - T6a: (X•Y)+(X•Y’)=X
  - T6b: (X+Y)•(X+Y’)=X

		OR	AND					
X	Y	X+Y	X•Y	Y’	X•Y’	(X•Y)+ (X•Y’)	X+Y’	(X+Y)• (X+Y’)
0	0	0	0	1	0	0	1	0
0	1	1	0	0	0	0	0	0
1	0	1	0	1	1	1	1	1
1	1	1	1	0	0	1	1	1

BOOLEAN ALGEBRA THEOREMS

- Absorption (aka *combining*)
  - T7a: (X•Y)+(X•Y’•Z)=(X•Y)+(X•Z)
  - T7b: (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 THEOREMS

- DeMorgan's theorem (very important!)
  - T8a: (X+Y)'= X'•Y'
    - $\overline{X+Y} = \overline{X}\bullet\overline{Y}$ break (or connect) the bar & change the sign
  - T8b: (X•Y)'= X'+Y'
    - X•Y = X+Y break (or connect) the bar & change the sign
  - Generalized DeMorgan's theorem:
    - GT8a: $(X_1+X_2+\ldots+X_{n-1}+X_n)' = X_1'\bullet X_2'\bullet\ldots\bullet X_{n-1}'\bullet X_n'$
    - GT8b: $(X_1\bullet X_2\bullet\ldots\bullet X_{n-1}\bullet X_n)' = X_1'+X_2'+\ldots+X_{n-1}'+X_n'$

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 THEOREMS

- Consensus Theorem
  - T9a: (X•Y)+(X’•Z)+(Y•Z) = (X•Y)+(X’•Z)
  - T9b: (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

MORE THEOREMS?

- Shannon's expansion theorem (also very important!)
  - › T10a: $f(X_1,X_2,\ldots,X_{n-1},X_n)=$
    - $(X_1'\bullet f(0,X_2,\ldots,X_{n-1},X_n))+(X_1\bullet f(1,X_2,\ldots,X_{n-1},X_n))$
      - Can be taken further:
        - $- \; f(X_1,X_2,\ldots,X_{n-1},X_n)=(X_1'\bullet X_2'\bullet f(0,0,\ldots,X_{n-1},X_n))$
          - $+ (X_1\bullet X_2'\bullet f(1,0,\ldots,X_{n-1},X_n)) + (X_1'\bullet X_2\bullet f(0,1,\ldots,X_{n-1},X_n))$
          - $+(X_1\bullet X_2\bullet f(1,1,\ldots,X_{n-1},X_n))$
      - Can be taken even further:
        - $- \; f(X_1,X_2,\ldots,X_{n-1},X_n)=(X_1'\bullet X_2'\bullet\ldots\bullet X_{n-1}'\bullet X_n'\bullet f(0,0,\ldots,0,0))$
          - $+ (X_1\bullet X_2'\bullet\ldots\bullet X_{n-1}'\bullet X_n'\bullet f(1,0,\ldots,0,0)) + \ldots$
          - $+ (X_1\bullet X_2\bullet\ldots\bullet X_{n-1}\bullet X_n\bullet f(1,1,\ldots,1,1))$
  - › T10b: $f(X_1,X_2,\ldots,X_{n-1},X_n)=$
    - $(X_1+f(0,X_2,\ldots,X_{n-1},X_n))\bullet(X_1'+f(1,X_2,\ldots,X_{n-1},X_n))$
      - Can be taken further as in the case of T10a
- We'll see significance of Shannon's expansion theorem later

BOOLEAN ALGEBRA THEOREMS

- Idempotency
  - T1a: X+X=X
  - T1b: X•X=X
- Null elements
  - T2a: X+1=1
  - T2b: X•0=0
- Involution
  - T3: (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 THEOREMS

- Absorption (aka *covering*)
  - T4a: $X+(X\bullet Y)=X$
  - T4b: $X\bullet(X+Y)=X$
  - T5a: $X+(X'\bullet Y)=X+Y$
  - T5b: $X\bullet(X'+Y)=X\bullet Y$

OR AND

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

BOOLEAN ALGEBRA THEOREMS

- Absorption (aka *combining*)
  - T6a: (X•Y)+(X•Y’)=X
  - T6b: (X+Y)•(X+Y’)=X

OR AND

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

BOOLEAN ALGEBRA THEOREMS

- Absorption (aka *combining*)
  - T7a: $(X\bullet Y)+(X\bullet Y'\bullet Z)=(X\bullet Y)+(X\bullet Z)$
  - T7b: $(X+Y)\bullet(X+Y'+Z) = (X+Y)\bullet(X+Z)$

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 THEOREMS

- DeMorgan’s theorem (very important!)
  - T8a: (X+Y)’= X’•Y’
    - $\overline{X+Y} = \overline{X} \bullet \overline{Y}$ break (or connect) the bar & change the sign
  - T8b: (X•Y)’= X’+Y’
    - X•Y = X+Y break (or connect) the bar & change the sign
  - Generalized DeMorgan’s theorem:
    - GT8a: $(X_1+X_2+\ldots+X_{n-1}+X_n)' = X_1' \bullet X_2' \bullet \ldots \bullet X_{n-1}' \bullet X_n'$
    - GT8b: $(X_1 \bullet X_2 \bullet \ldots \bullet X_{n-1} \bullet X_n)' = X_1' + X_2' + \ldots + X_{n-1}' + X_n'$

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 THEOREMS

- Consensus Theorem
  - T9a: (X•Y)+(X’•Z)+(Y•Z) = (X•Y)+(X’•Z)
  - T9b: (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

MORE THEOREMS?

- Shannon's expansion theorem (also very important!)
- › T10a: $f(X_1,X_2,\ldots,X_{n-1},X_n)=$
  - $(X_1'\bullet f(0,X_2,\ldots,X_{n-1},X_n))+(X_1\bullet f(1,X_2,\ldots,X_{n-1},X_n))$
    - Can be taken further:
    - - $f(X_1,X_2,\ldots,X_{n-1},X_n)= (X_1'\bullet X_2'\bullet f(0,0,\ldots,X_{n-1},X_n))$
      - $+ (X_1\bullet X_2'\bullet f(1,0,\ldots,X_{n-1},X_n)) + (X_1'\bullet X_2\bullet f(0,1,\ldots,X_{n-1},X_n))$
      - $+(X_1\bullet X_2\bullet f(1,1,\ldots,X_{n-1},X_n))$
    - Can be taken even further:
    - - $f(X_1,X_2,\ldots,X_{n-1},X_n)= (X_1'\bullet X_2'\bullet\ldots\bullet X_{n-1}'\bullet X_n'\bullet f(0,0,\ldots,0,0))$
      - $+ (X_1\bullet X_2'\bullet\ldots\bullet X_{n-1}'\bullet X_n'\bullet f(1,0,\ldots,0,0)) + \ldots$
      - $+ (X_1\bullet X_2\bullet\ldots\bullet X_{n-1}\bullet X_n\bullet f(1,1,\ldots,1,1))$
- › T10b: $f(X_1,X_2,\ldots,X_{n-1},X_n)=$
  - $(X_1+f(0,X_2,\ldots,X_{n-1},X_n))\bullet(X_1'+f(1,X_2,\ldots,X_{n-1},X_n))$
    - Can be taken further as in the case of T10a
- We'll see significance of Shannon's expansion theorem later

SWITCHING FUNCTIONS

- Objective:

  Understand the logic functions of the digital circuits

- Course Outcomes(CAEC020.05):

  Describe minimization techniques and other optimization techniques for Boolean formulas in general and digital circuits.

SWITCHING FUNCTIONS

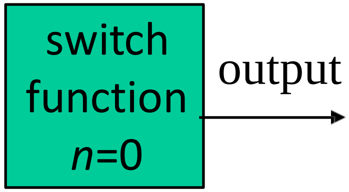
- For *n* variables, there are 2^n possible combinations of values
  - From all 0s to all 1s

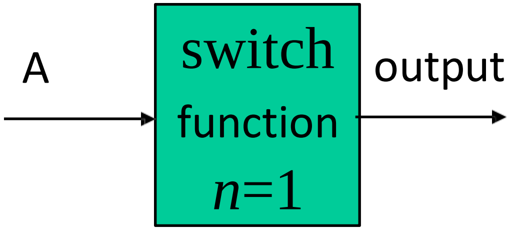
- There are 2 possible values for the output of a function of a given combination of values of *n* variables
  - 0 and 1

- There are 2^{2^n} different switching functions for *n* variables

SWITCHING FUNCTION EXAMPLES

- 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’





A	f_0	f_1	f_2	f_3
0	0	1	0	1
1	0	0	1	1

$f_0 = 0$

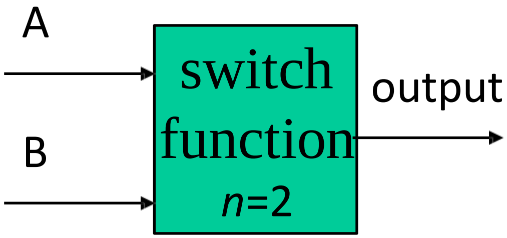
$f_1 = A'$

$f_2 = A$

$f_3 = 1$

SWITCHING FUNCTION EXAMPLES

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



A B	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} 0 0
	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
0 1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1 0	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1 1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

$f_0 = 0$ logic 0

$f_1 = A'B' = (A+B)'$ NOT-OR or NOR

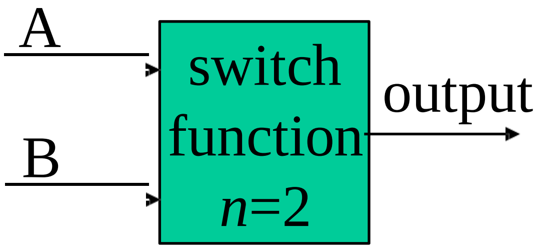
$f_2 = A'B$

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

Most frequently used *Less frequently used* *Least frequently used*

SWITCHING FUNCTION EXAMPLES

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



A B	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}	0 0
	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	
0 1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	
1 0	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	
1 1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	

f_4 = AB’

f_5 = A’B’+AB’ = (A’+A)B’ = B’ invert B

f_6 = A’B+AB’ exclusive-OR

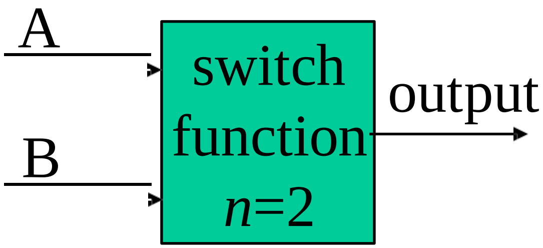
f_7 = A’B’+A’B+AB’=A’(B’+B)+(A’+A)B’

= A’+B’ = (AB)’ NOT-AND or NAND

Most frequently used *Less frequently used* *Least frequently used*

SWITCHING FUNCTION EXAMPLES

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



A B	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}
0 0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
0 1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1 0	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1 1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

$f_8 = AB$ AND

$f_9 = A'B' + AB$ exclusive-NOR

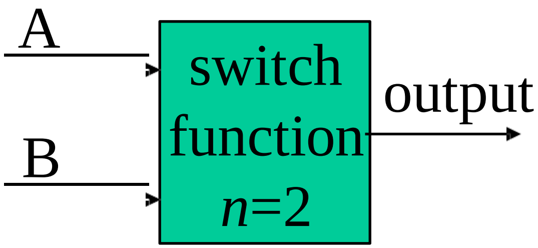
$f_{10} = A'B + AB = (A'+A)B = B$ buffer B

$f_{11} = A'B' + A'B + AB = A'(B'+B) + (A'+A)B = A' + B$

Most frequently used *Less frequently used* *Least frequently used*

SWITCHING FUNCTION EXAMPLES

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



A B	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}	0 0
	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	
0 1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	
1 0	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1	
1 1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1	

f_{12} = AB'+AB = A(B'+B) = A buffer A

f_{13} = A'B'+AB'+AB = A(B'+B)+A'B' = A+A'B' =A+B'

f_{14} = A'B+AB'+AB = A(B'+B)+(A'+A)B =A+B OR

f_{15} = A'B'+A'B+AB'+AB = A'(B'+B)+A(B'+B)
= A'+A = 1 logic 1

Most frequently used *Less frequently used* *Least frequently used*

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 product terms is referred to as a sum of products expression

$$Y = AB + BC + AC$$

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

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

- ***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

$$Y = AB + BC + AC$$

CANONICAL AND STANDERED FORMS

- **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.

- **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.

CANONICAL SUM OF PRODUCTS

- When a Boolean function is expressed as the logical sum of all the minterms from the rows of a truth table, for which the value of the function is 1, it is referred to as the canonical sum of product expression
- For example, if the canonical sum of product form of a three-variable logic function F has the minterms $\bar{A}BC$, $A\bar{B}C$ and $AB\bar{C}$ this can be expressed as the sum of the decimal codes corresponding to these minterms as below..

$$\begin{aligned} F(A,B,C) &= \Sigma(3,5,6) \\ &= m_3 + m_5 + m_6 \\ &= \bar{A}BC + A\bar{B}C + AB\bar{C} \end{aligned}$$

CANONICAL SUM OF PRODUCTS

- 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.
  - Examine for the variables that are missing in each product which is not a minterm. If the missing variable in the minterm is X, multiply that minterm with (X+X’).
  - Multiply all the products and discard the redundant terms.

CANONICAL SUM OF PRODUCTS

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

- Solution: $F\ (A, B, C)\ =\ A\ +\ BC$

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

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

$$= ABC + A\bar{B}C + ABC̄ + A\bar{B}\bar{C} + ABC + \bar{A}BC$$

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

- 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 SUM OF PRODUCTS

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 “Dual of SOP form”.

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

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

CANONICAL SUM OF PRODUCTS

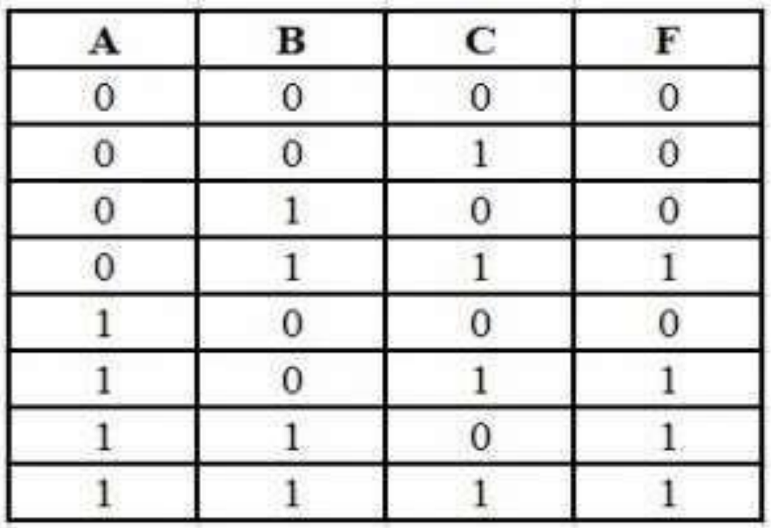
POS form can be obtained by

- Writing an OR term for each input combination, which produces LOW output.
- Writing the input variables if the value is 0, and write the complement of the variable if its value is AND the OR terms to obtain the output function.

CANONICAL SUM OF PRODUCTS

Example:

Boolean expression for majority function F = (A + B + C) (A + B + C ‘) (A + B’ + C) (A’ + B + C)



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

Now write the input variables combination with high output. F = AB + BC + AC.

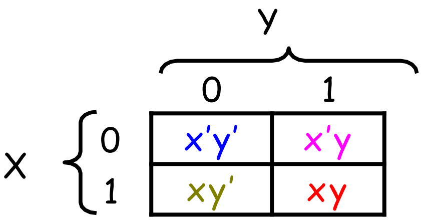
KARANAUGH MAP

- Boolean algebra helps us simplify expressions and circuits
- Karnaugh Map: A graphical technique for simplifying a Boolean expression into either form:
  - minimal sum of products (MSP)
  - minimal product of sums (MPS)
- Goal of the simplification.
  - There are a minimal number of product/sum terms
  - Each term has a minimal number of literals

KARANAUGH MAP

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

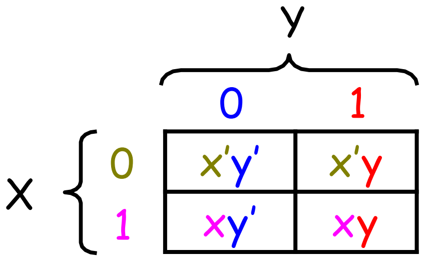
x	y	minterm
0	0	x'y'
0	1	x'y
1	0	xy'
1	1	xy



y

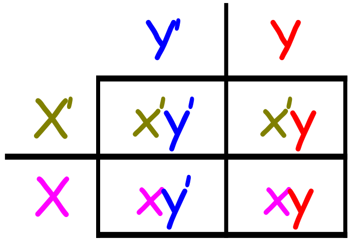
x	0	1
0	x'y'	x'y
1	xy'	xy

- Now we can easily see which minterms contain common literals
  - Minterms on the left and right sides contain y' and y respectively
  - Minterms in the top and bottom rows contain x' and x respectively



y

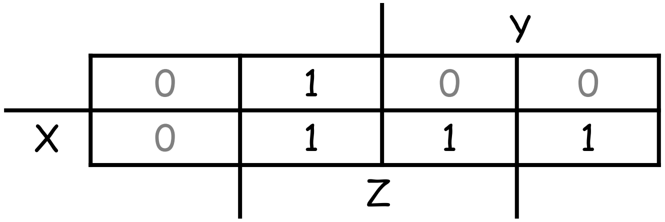
x	0	1
0	x'y'	x'y
1	xy'	xy



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

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 be included in at least one rectangle. Do not include any of the 0s
  - Each group corresponds to one product term



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

KARANAUGH MAP

- Maxterms are grouped to find minimal PoS expression

yz

x	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 minterms form,
  - The easiest way to do this is to make a truth table for the 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 example:

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 minterms is more tricky:

YZ

		00	01	11	10
X	0	x'y'z'	x'y'z	x'yz	x'yz'
	1	xy'z'	xy'z	xyz	xyz'

			y	y
	x'y'z'	x'y'z	x'yz	x'yz'
X	xy'z'	xy'z	xyz	xyz'
		Z	Z	

YZ

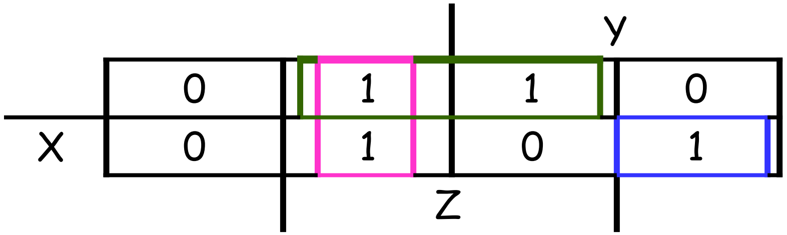
		00	01	11	10
X	0	m_0	m_1	m_3	m_2
	1	m_4	m_5	m_7	m_6

			y	y
	m_0	m_1	m_3	m_2
X	m_4	m_5	m_7	m_6
		Z	Z	

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

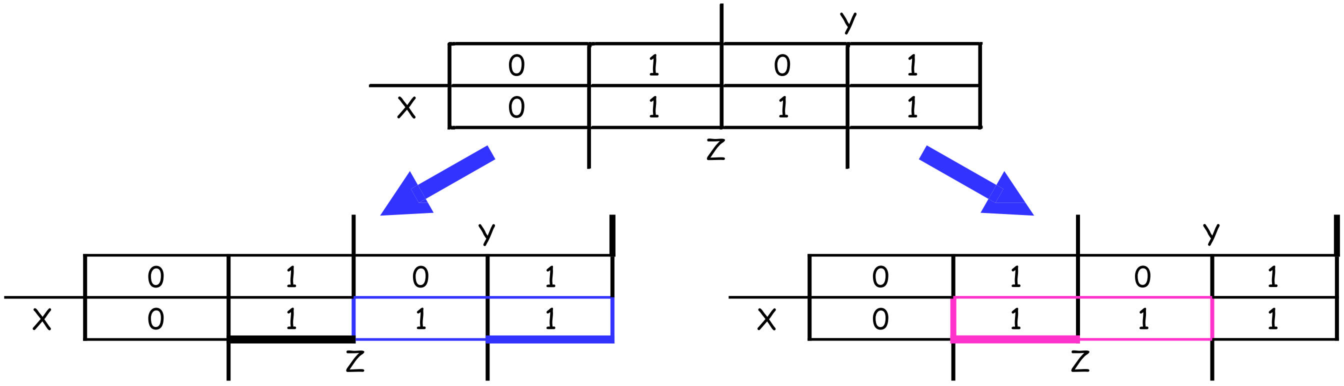


	0	1	1	0
X	0	1	0	1

- 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 ways to include minterm m_7



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

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

$y'z + yz' + xy$

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

$y'z + yz' + xz$

- Remember that overlapping groups is possible, as shown above

3-VARIABLE K-MAP

- Maxterms are grouped to find minimal PoS expression

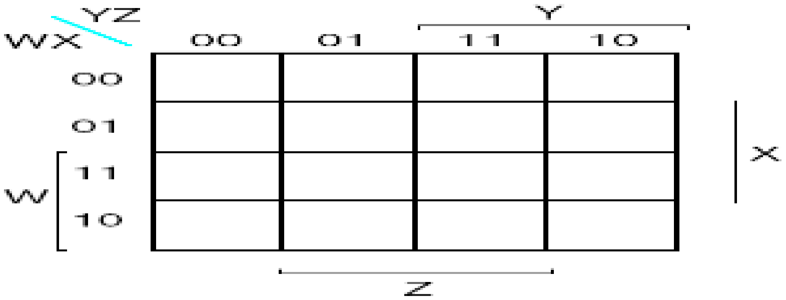
yz

x	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

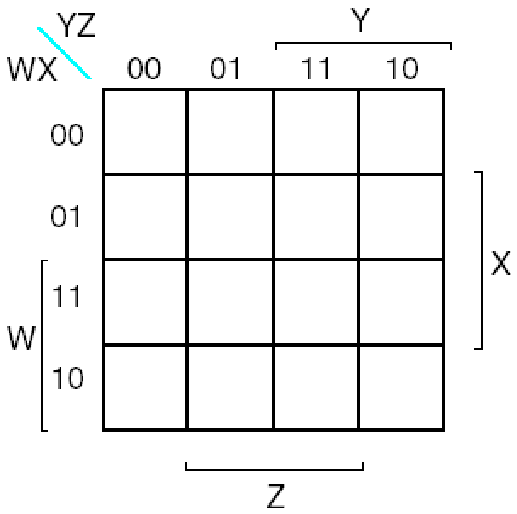
- We can do four-variable expressions too!
  - The minterms in the third and fourth columns, *and* in the third and

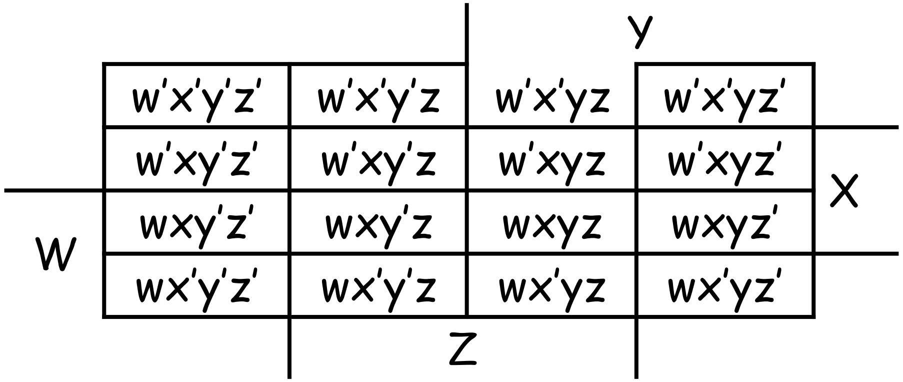
    fourth rows, are switched around.
  - Again, this ensures that adjacent squares have common literals



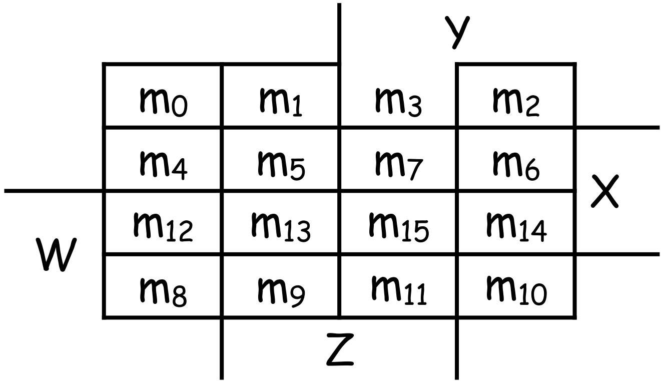
- Grouping minterms is similar to the three-variable case, but:
  - You can have rectangular groups of 1, 2, 4, 8 or 16 minterms
  - You can wrap around all four sides

4-VARIABLE K-MAP





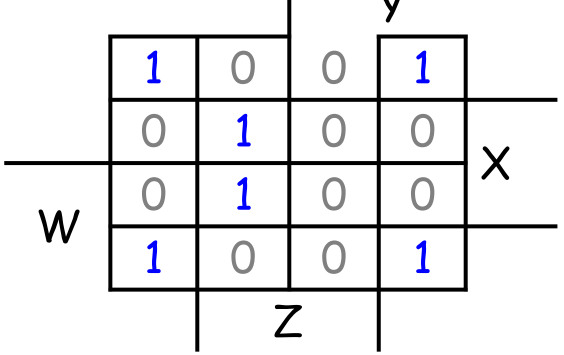
w'x'y'z'	w'x'y'z	w'x'yz	w'x'yz'
w'xy'z'	w'xy'z	w'xyz	w'xyz'
wxy'z'	wxy'z	wxyz	wxyz'
wx'y'z'	wx'y'z	wx'yz	wx'yz'



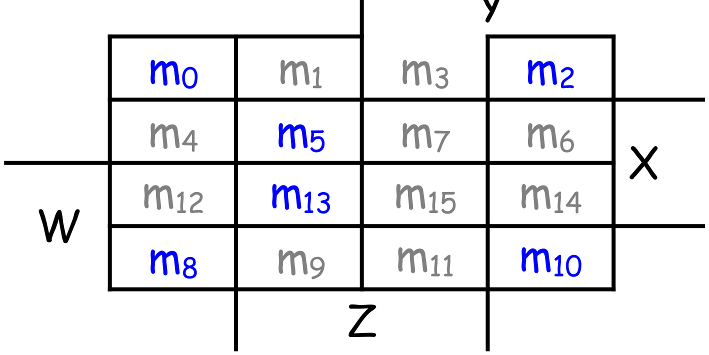
m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6
m_{12}	m_{13}	m_{15}	m_{14}
m_8	m_9	m_{11}	m_{10}

4-VARIABLE K-MAP

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

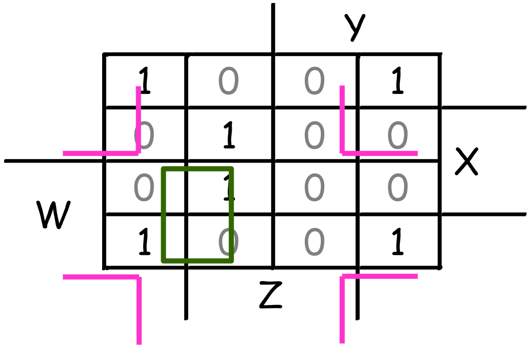


1	0	0	1
0	1	0	0
0	1	0	0
1	0	0	1

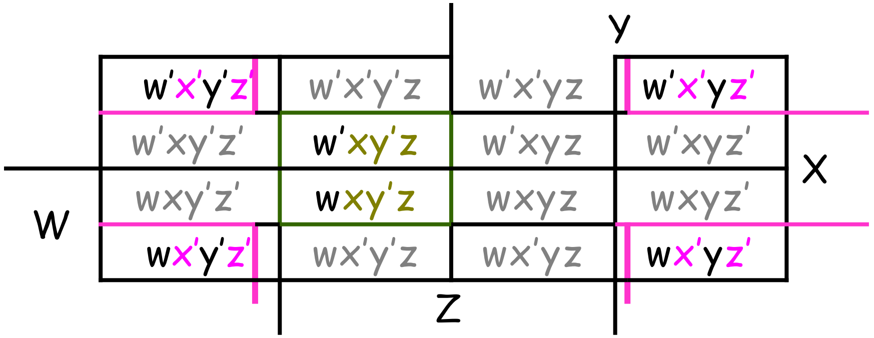


m_0	m_1	m_3	m_2
m_4	m_5	m_7	m_6
m_{12}	m_{13}	m_{15}	m_{14}
m_8	m_9	m_{11}	m_{10}

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



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

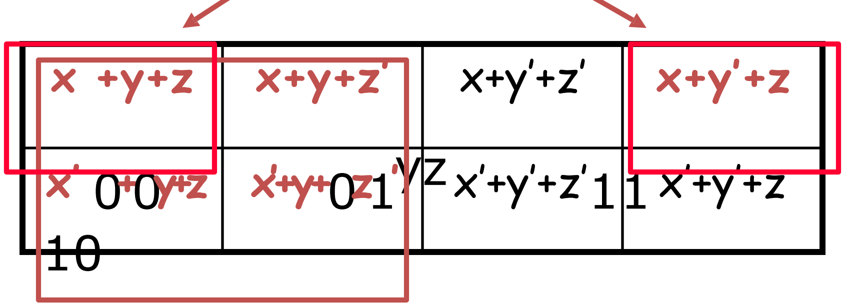


w'x'y'z'	w'x'y'z	w'x'yz	w'x'yz'
w'xy'z'	w'xy'z	w'xyz	w'xyz'
wxy'z'	wxy'z	wxyz	wxyz'
wx'y'z'	wx'y'z	wx'yz	wx'yz'

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

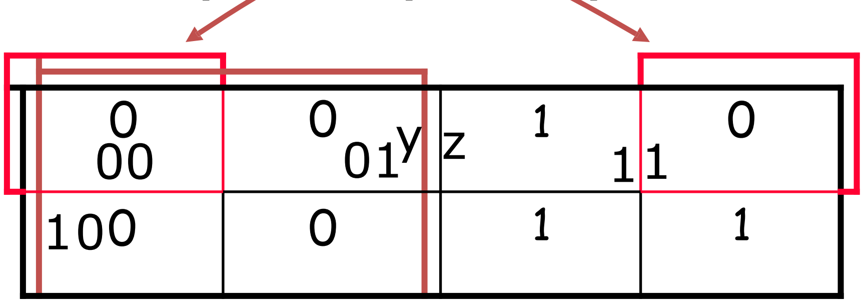
4-VARIABLE K-MAP

- F(W,X,Y,Z)= ∏ M(0,1,2,4,5)



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

F(W,X,Y,Z)= Y . (X + Z)



x \ yz	00	01	11	10
0	0	0	1	0
1	0	0	1	1

5-VARIABLE K-MAP

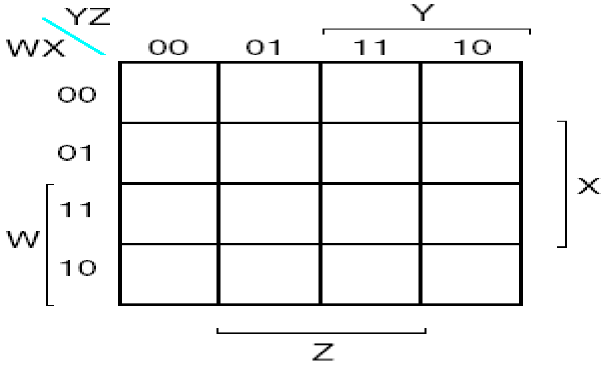
- Objective:

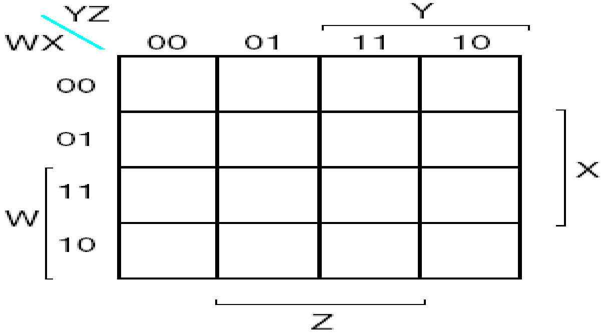
  Understand the 5-Variable K-map

- Course Outcomes(CAEC020.06):

  Evaluate the functions using various types of minimizing algorithms like Karanaugh map method.

5-variable K-map





V= 0

Y

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

X

W

Z

V= 1

Y

m_{16}	m_{17}	m_{19}	m_8
m_{20}	m_{21}	m_{23}	m_{22}
m_{28}	m_{29}	m_{31}	m_{30}
m_{24}	m_{25}	m_{27}	m_{26}

X

W

Z

5-VARIABLE K-MAP

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

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

y

	x'y'z'	x'y'z	x'yz	x'yz'
x	xy'z'	xy'z	xyz	xyz'

z

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

y

	m_0	m_1	m_3	m_2
x	m_4	m_5	m_7	m_6

z

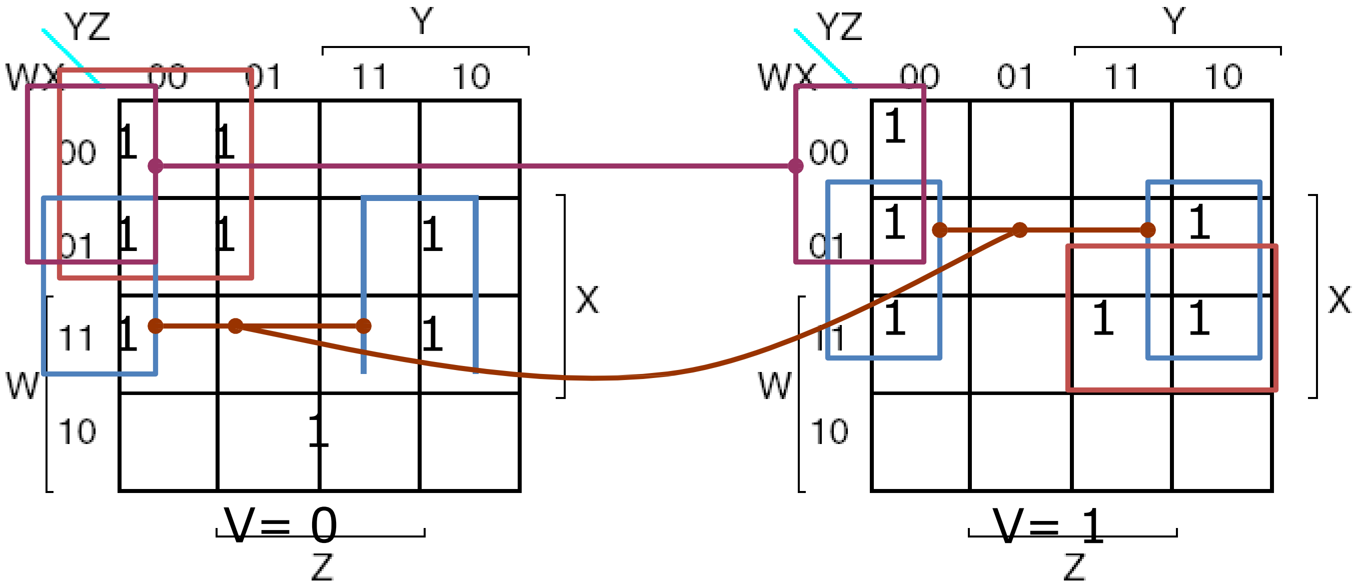
- In either case, the resulting K-map is shown below

y

	0	1	0	0
x	0	1	1	1

z

5-VARIABLE K-MAP



f = XZ’

Σm(4,6,12,14,20,22,28,30)

+ V’W’Y’ Σm(0,1,4,5)

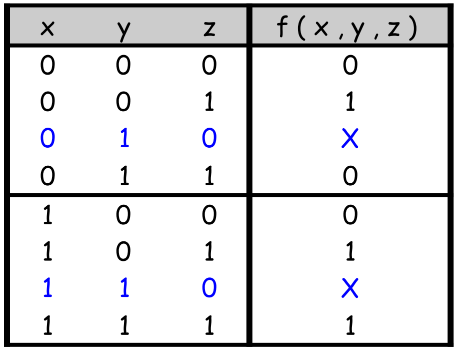
+ W’Y’Z’ Σm(0,4,16,20)

+ VWXY Σm(30,31)

+ V’WX’YZ m11

DON'T CARE CONDITION

- You don't always need all 2^n input combinations in an n-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

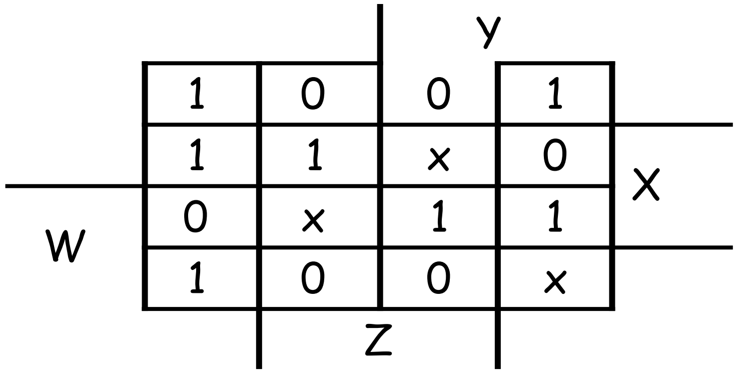
- We mark don't-care outputs in truth tables and K-maps with Xs.

DON'T CARE CONDITION

- Find a MSP for

f(w,x,y,z) = Σm(0,2,4,5,8,14,15), d(w,x,y,z) = Σm(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.

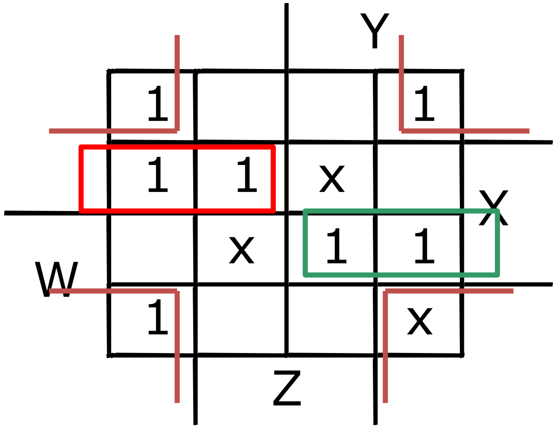


1	0	0	1
1	1	x	0
0	x	1	1
1	0	0	x

DON’T CARE CONDITIONS

- Find a MSP for:

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



1			1
1	1	x	
	x	1	1
1			x

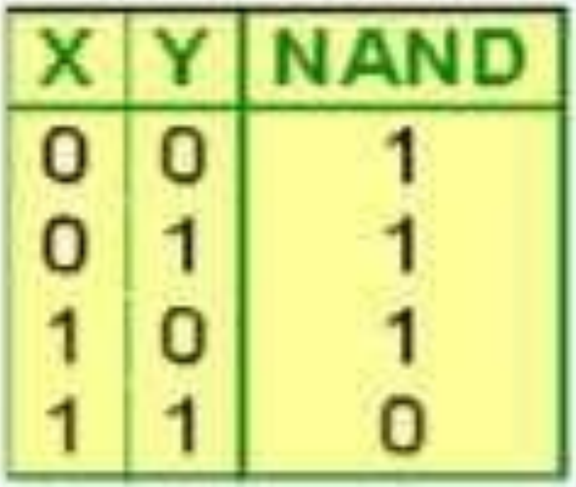
f(w,x,y,z)= x’z’ + w’xy’ + wxy

NAND NOR IMPLEMENTATION

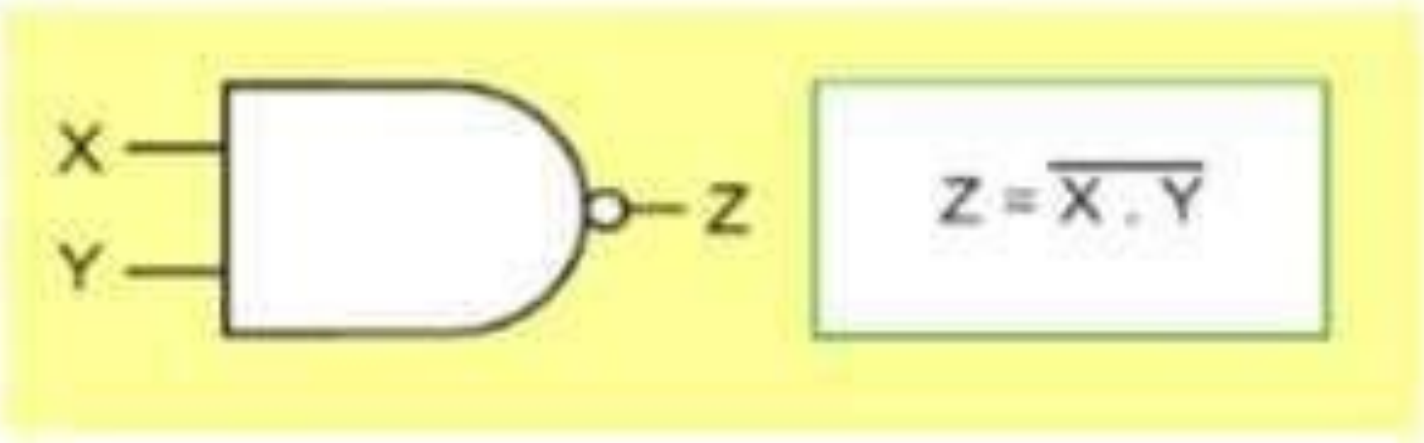
- The objectives of this lesson are to learn about:

1. Universal gates - NAND and NOR.
2. How to implement NOT, AND, and OR gate using NAND gates only.
3. How to implement NOT, AND, and OR gate using NOR gates only.
4. Equivalent gates.

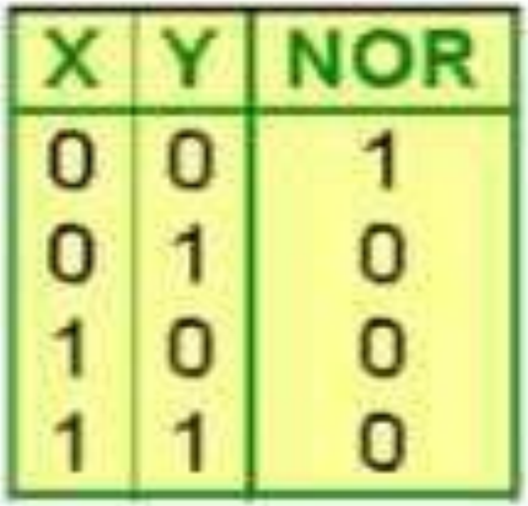
NAND NOR IMPLEMENTATION



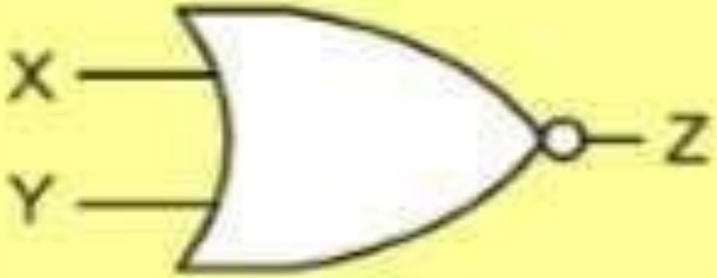
X	Y	NAND
0	0	1
0	1	1
1	0	1
1	1	0



$Z = \overline{X . Y}$



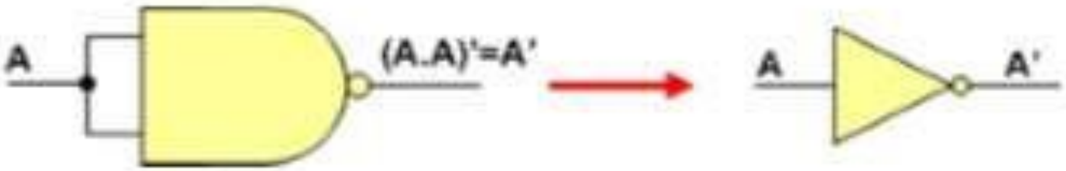
X	Y	NOR
0	0	1
0	1	0
1	0	0
1	1	0



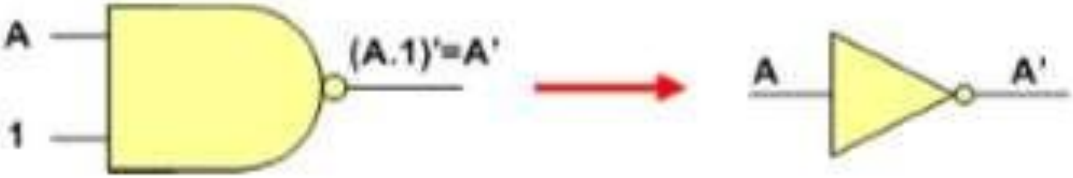
$$Z = \overline{X + Y}$$

NAND NOR IMPLEMENTATION

1. All NAND input pins connect to the input signal **A** gives an output **A'**.

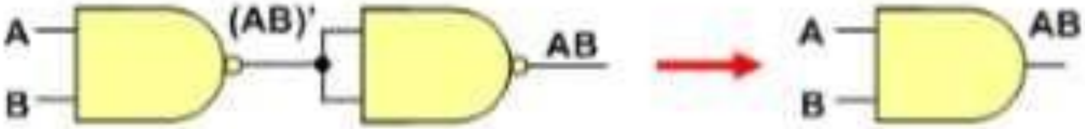


2. One NAND input pin is connected to the input signal **A** while all other input pins are connected to logic 1. The output will be **A'**.



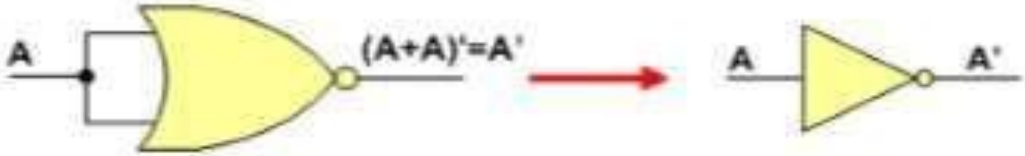
Implementing AND Using only NAND Gates

An AND gate can be replaced by NAND gates as shown in the figure (The AND is replaced by a NAND gate with its output complemented by a NAND gate inverter).

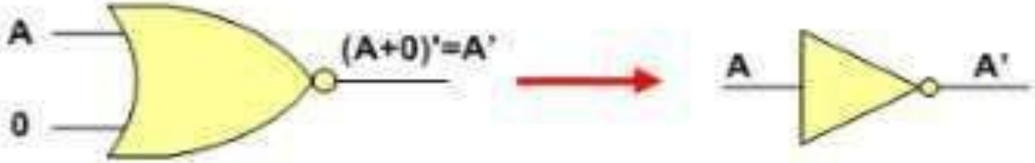


NAND NOR IMPLEMENTATION

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

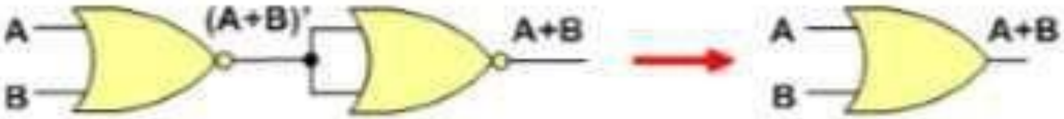


2. One NOR input pin is connected to the input signal **A** while all other input pins are connected to logic **0**. The output will be **A'**.



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)



TWO LEVEL Implementation

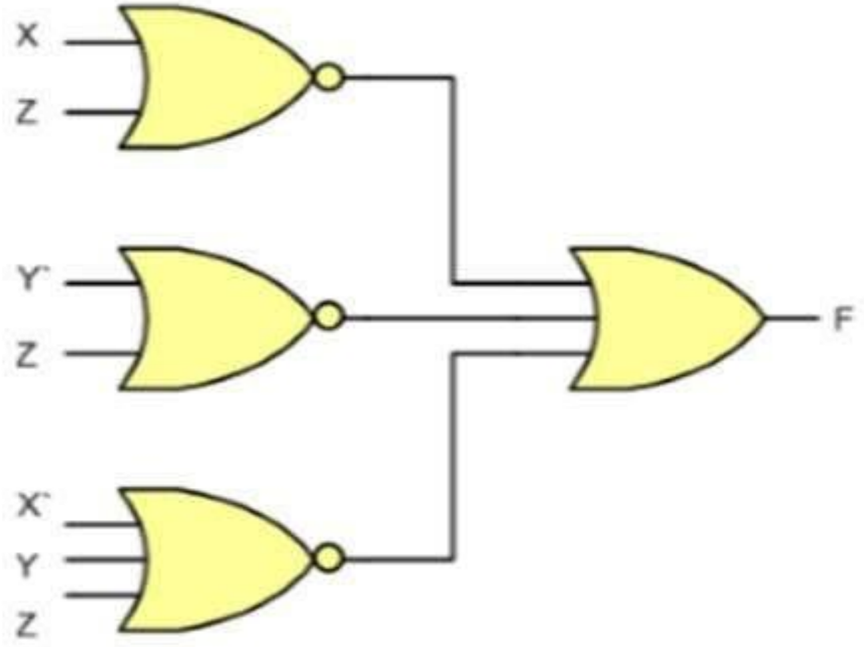
OR NAND Function:

$$F = \overline{(X+Z).(\overline{Y}+Z).(\overline{X}+Y+Z)} \text{ or}$$

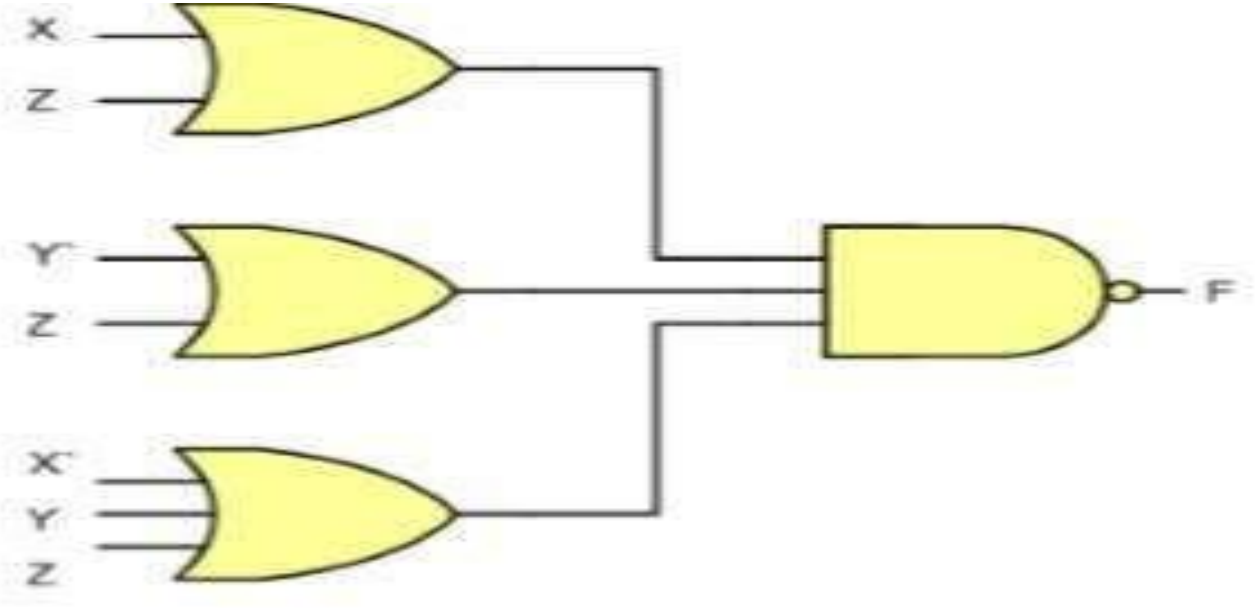
$$\overline{F} = (X+Z)(\overline{Y}+Z)(\overline{X}+Y+Z)$$

Since ‘F’ is in POS form Z can be implemented by using NOR NOR circuit. Similarly complementing the output we can get F,or by using NOR –OR Circuit as shown in figure

TWO LEVEL Implementation



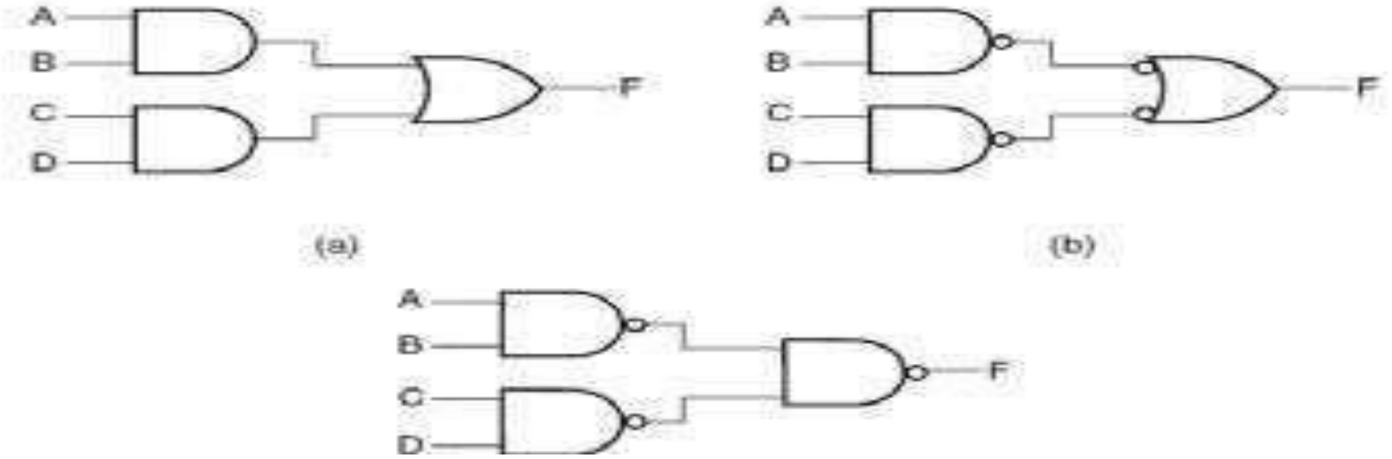
TWO LEVEL Implementation



It can also be implemented using OR-NAND circuit as it is equivalent to NOR-OR circuit

Two Level Implementation

- **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.
- The Rule
- This function can This function can be implemented by three different ways as shown in the circuit diagram a, b, c

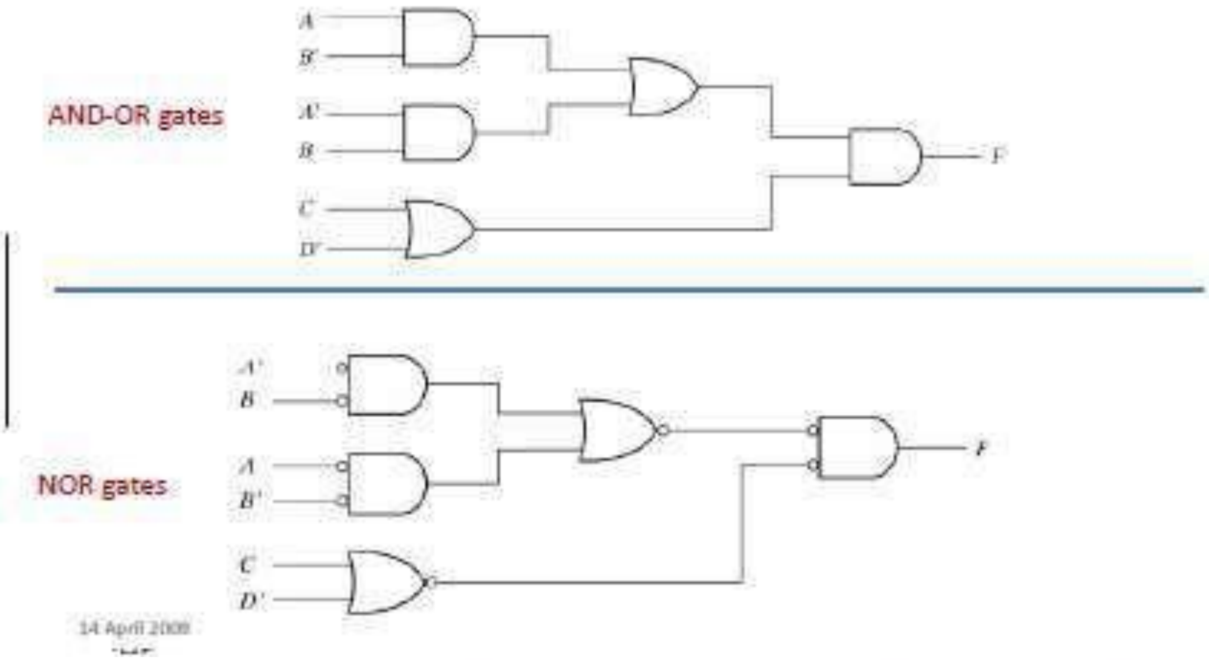


(a)

(b)

Two Level Implementation

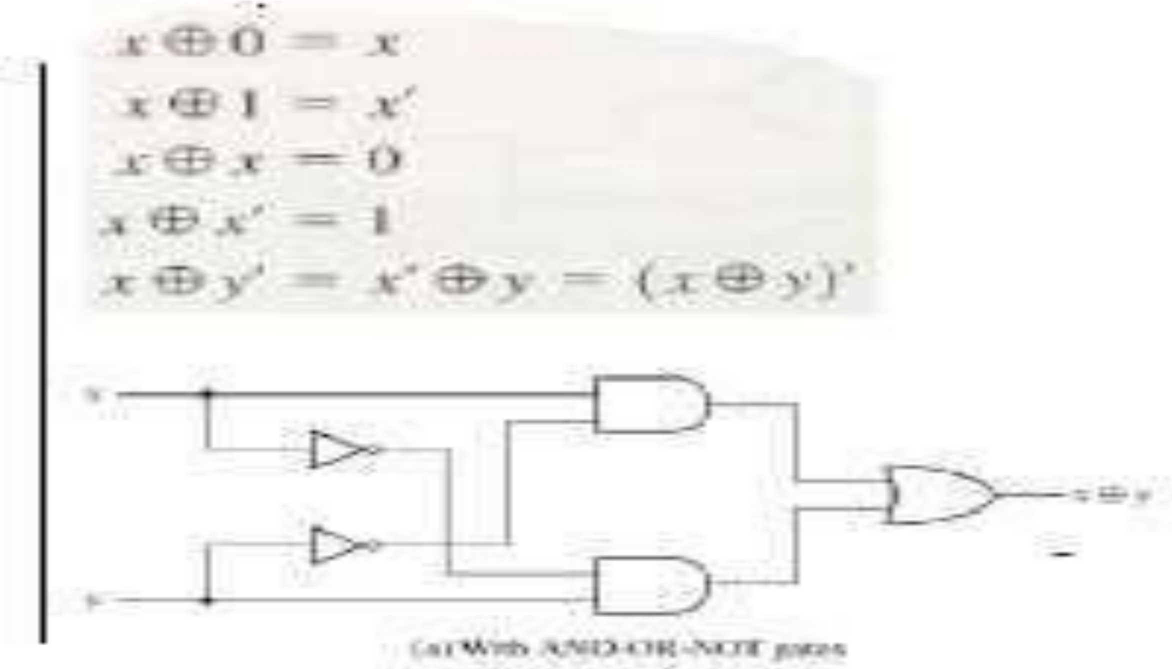
Example 2: Consider the following Boolean function, implement the circuit diagram by using multilevel NOR gate. $F = (AB' + A'B)i(C + D')$



Two Level Implementation

Exclusive-OR (XOR) Function:

XOR: x   y = xy’ + x’y



$x \oplus 0 = x$

$x \oplus 1 = x'$

$x \oplus x = 0$

$x \oplus x' = 1$

$x \oplus y' = x' \oplus y = (x \oplus y)'$

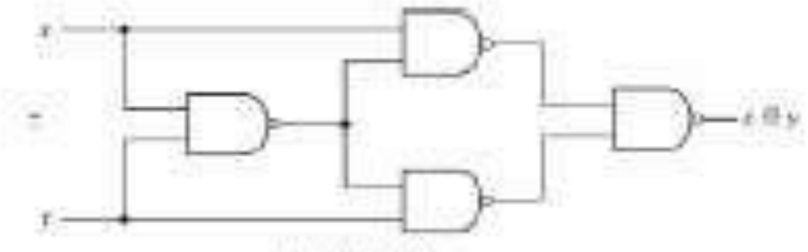
(a) With AND-OR-NOT gates

Two Level Implementation

Exclusive-NOR = equivalence

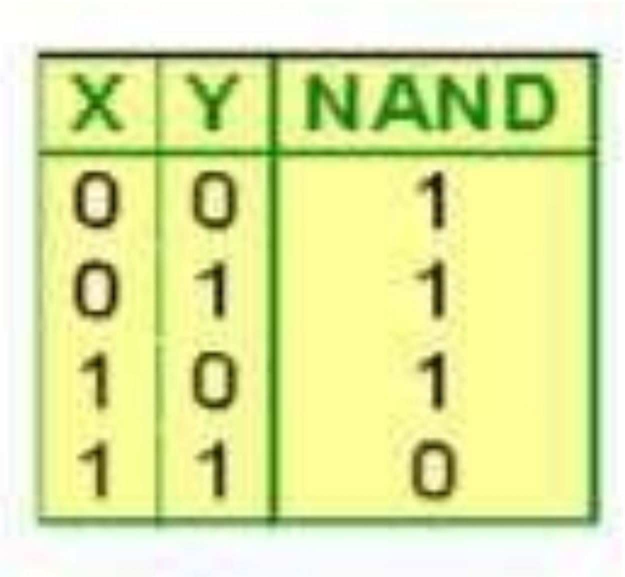
(x y)’ = (xy’ + x’y)’

= (x’ + y)(x + y’) = x’y’ + xy

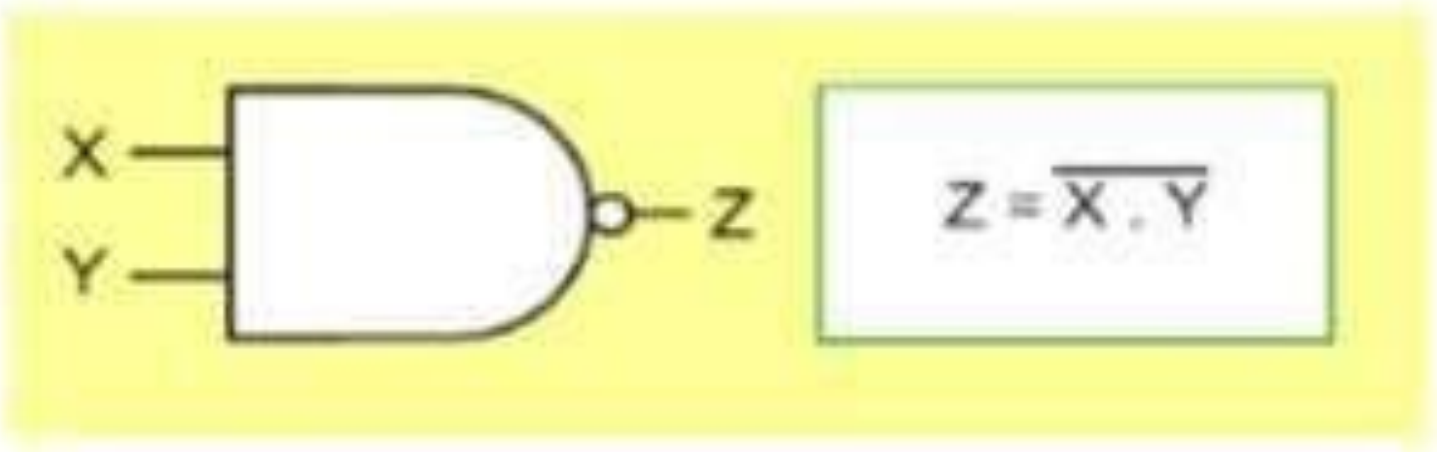


(b) With NAND gates

TWO LEVEL Implementation



X	Y	NAND
0	0	1
0	1	1
1	0	1
1	1	0



$Z = \overline{X \,.\, Y}$

TWO LEVEL Implementation

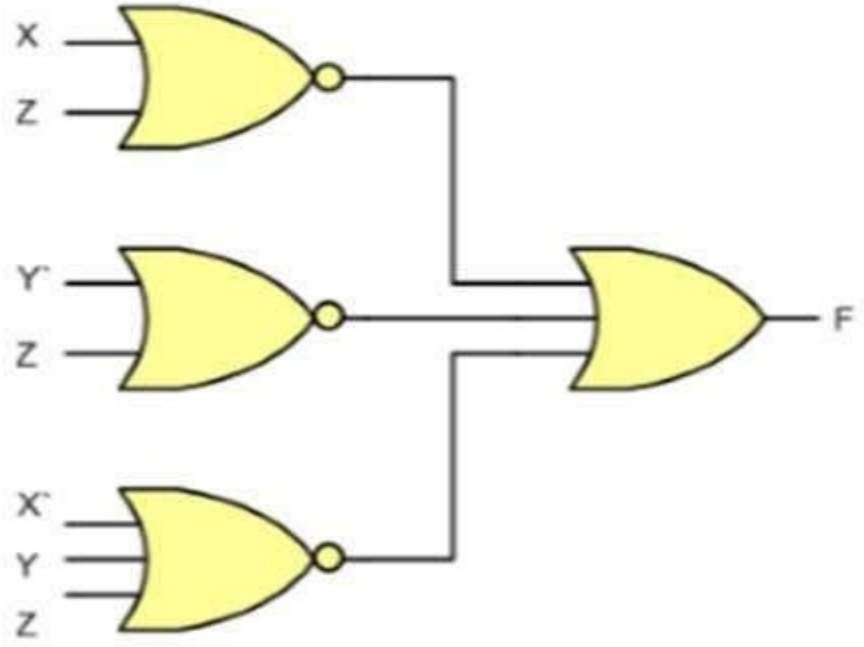
OR NAND Function:

$$F = \overline{(X+Z).(\overline{Y}+Z).(\overline{X}+Y+Z)} \text{ or}$$

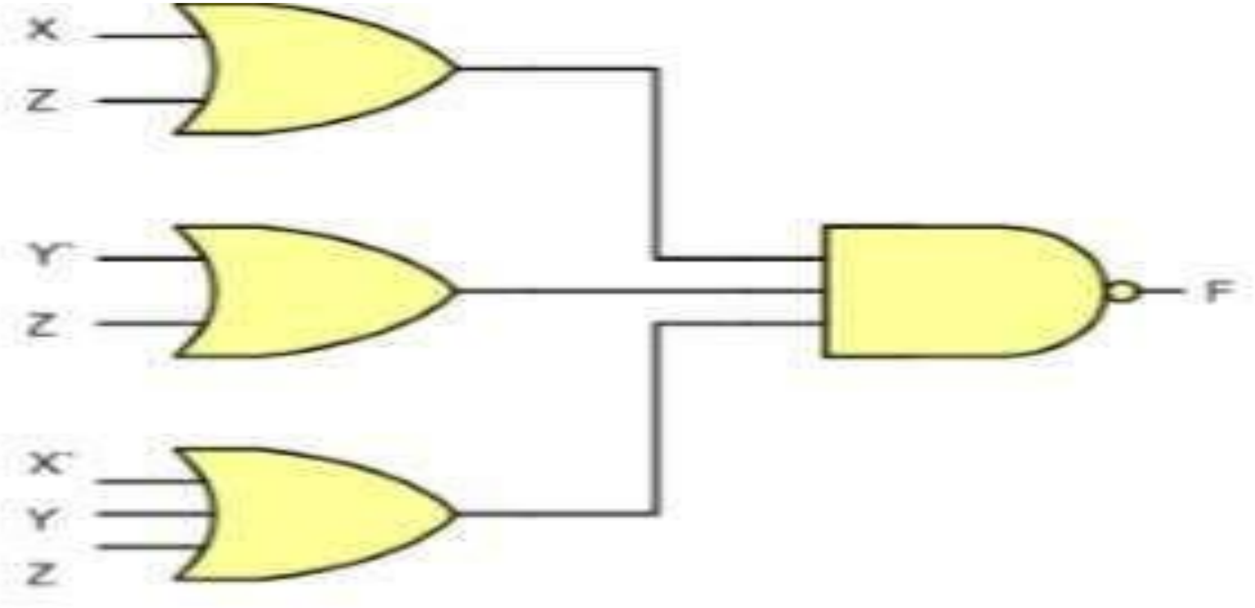
$$\overline{F} = (X+Z)(\overline{Y}+Z)(\overline{X}+Y+Z)$$

Since ‘F’ is in POS form Z can be implemented by using NOR NOR circuit. Similarly complementing the output we can get F,or by using NOR –OR Circuit as shown in figure

TWO LEVEL Implementation



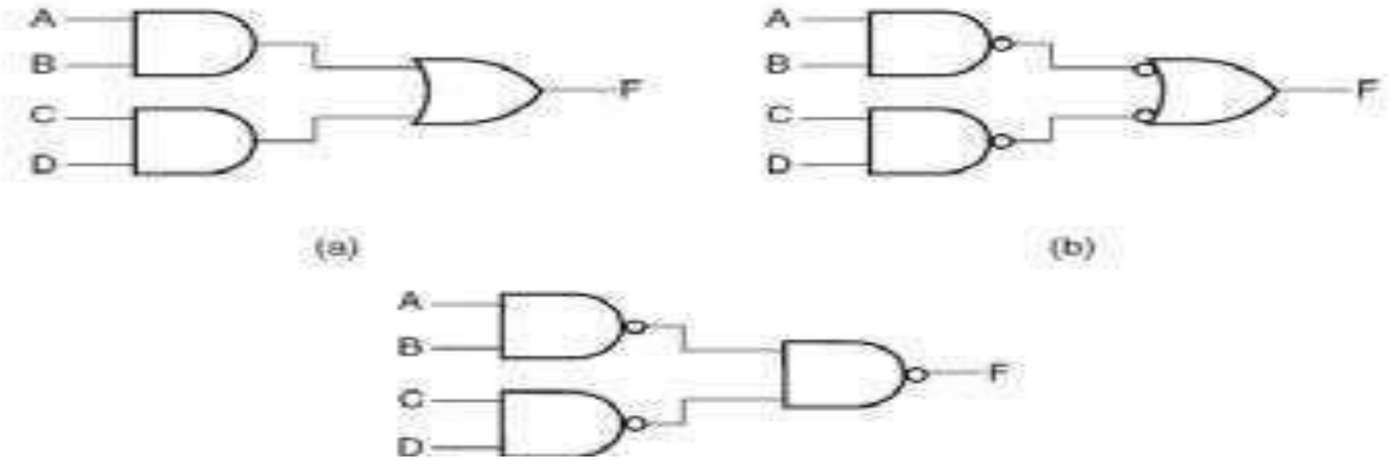
TWO LEVEL Implementation



It can also be implemented using OR-NAND circuit as it is equivalent to NOR-OR circuit

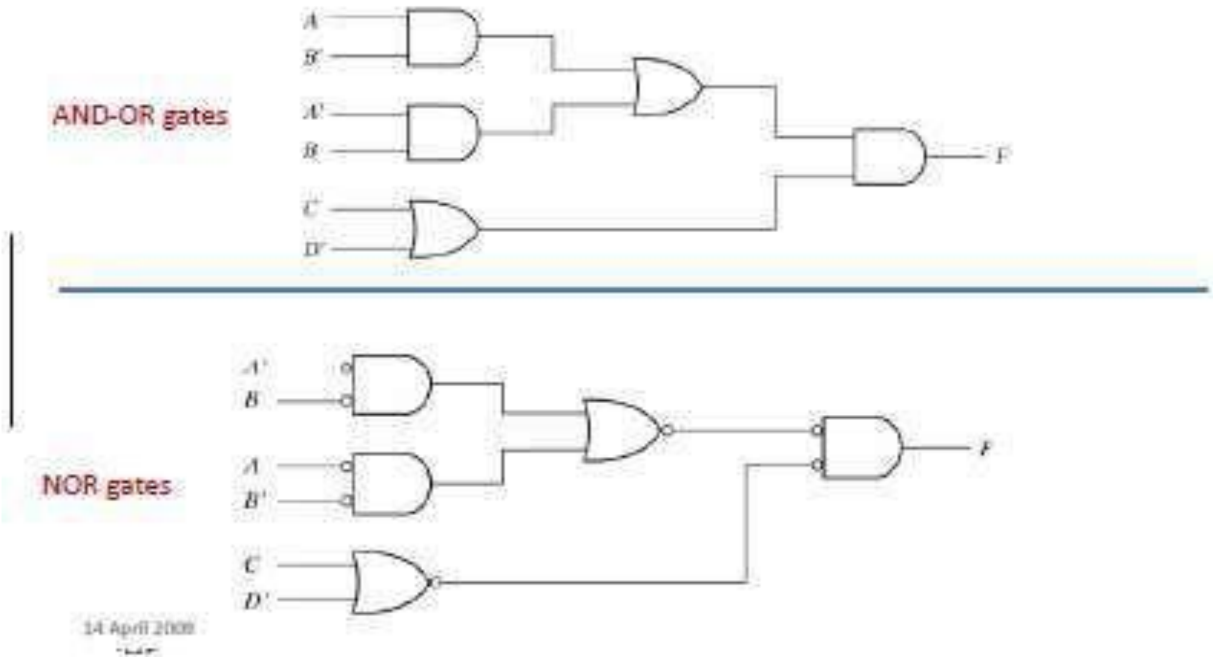
Two Level Implementation

- **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.
- The Rule
- This function can This function can be implemented by three different ways as shown in the circuit diagram a, b, c



Two Level Implementation

Example 2: Consider the following Boolean function, implement the circuit diagram by using multilevel NOR gate. *F* = (*AB* ′ + *A*′*B*)i(*C* + *D*′)



Two Level Implementation

Exclusive-OR (XOR) Function:

XOR: xy’ + x’y

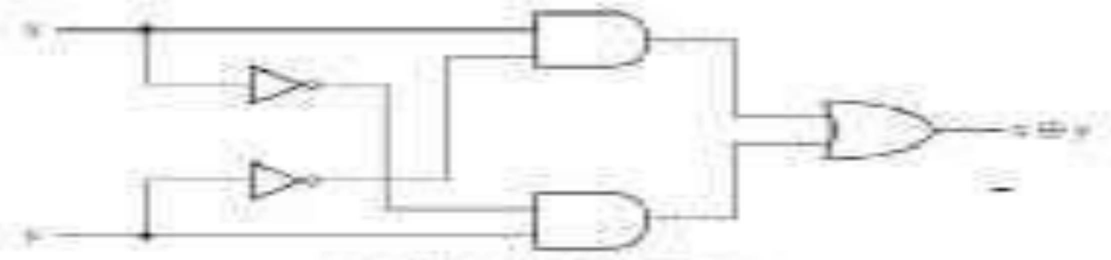
$$x \oplus 0 = x$$

$$x \oplus 1 = x'$$

$$x \oplus x = 0$$

$$x \oplus x' = 1$$

$$x \oplus y' = x' \oplus y = (x \oplus y)'$$

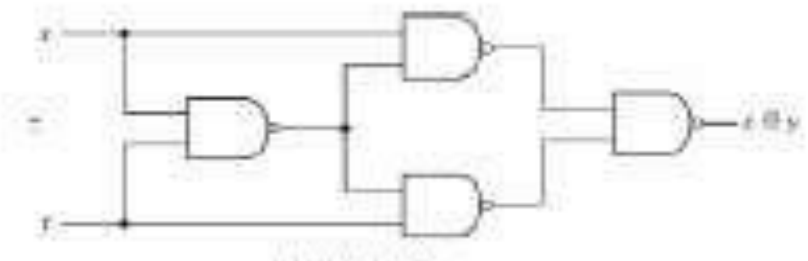


(a) With AND-OR-NOT gates

Two Level Implementation

Exclusive-NOR = equivalence

= (x’ + y)(x + y’) = x’y’ + xy



(b) With NAND gates

UNIT 3

COMBINATIONAL CIRCUITS

Combinational Circuits

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

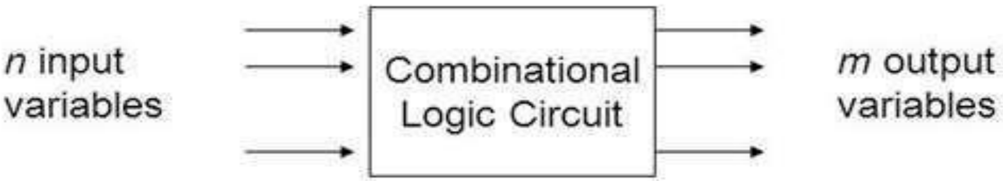
Some of the characteristics of combinational circuits are following:

- The output of combinational circuit at any instant of time, depends only on the levels present at input terminals.
- 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:**

2^n possible combinations of input values.



- Specific functions :of combinational circuits

  Adders,subtractors,multiplexers,comprators,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 each gate 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.

BINARY ADDERS

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.

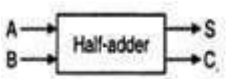
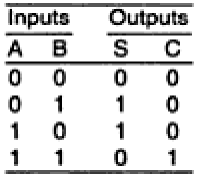


Fig 1:Block diagram



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

$\text{Sum}=A'B+AB'=A \oplus B$

$\text{Carry}=AB$

BINARY ADDERS

Full Adder

The full-adder adds the bits A and B and the carry from the previous column called the carry-in C_{in} and outputs the sum bit S and the carry bit called the carry-out C_{out} .

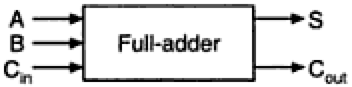
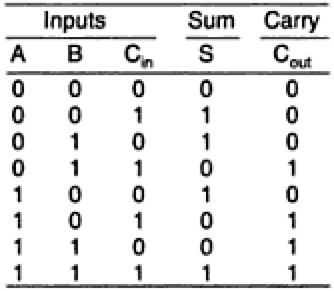


Fig 3: block diagram



Inputs			Sum	Carry
A	B	C_{in}	S	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Fig 4:Truth table

$$S = \overline{A}\,\overline{B}C_{in} + \overline{A}B\overline{C}_{in} + A\overline{B}\,\overline{C}_{in} + ABC_{in}$$

$$C_{out} = \overline{A}BC_{in} + A\overline{B}C_{in} + AB\overline{C}_{in} + ABC_{in}$$

$$S = A \oplus B \oplus C_{in}$$

$$C_{out} = AC_{in} + BC_{in} + AB$$

BINARY SUBTRACTORS

Half Subtractor

A Half-subtractor is a combinational circuit with two inputs A and B and two outputs difference(d) and barrow(b).

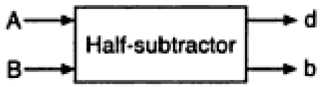
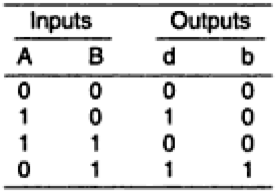


Fig 5:Block diagram



Inputs		Outputs	
A	B	d	b
0	0	0	0
1	0	1	0
1	1	0	0
0	1	1	1

Fig 6: Truth table

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

$b=A'B$

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 .

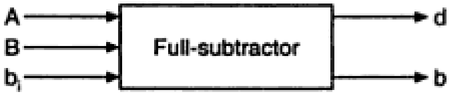
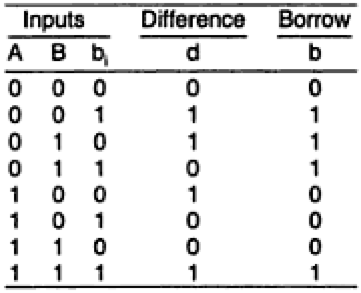


Fig 7:Block diagram



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

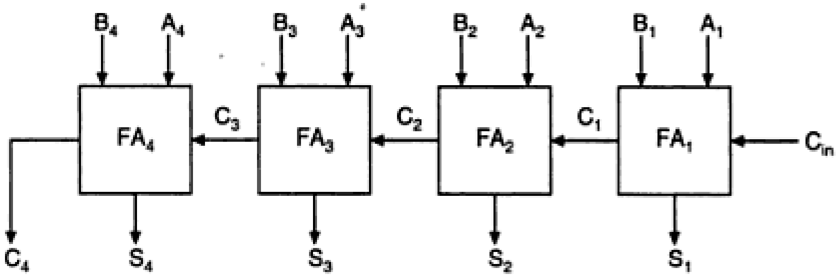


Fig 9:parallel adder

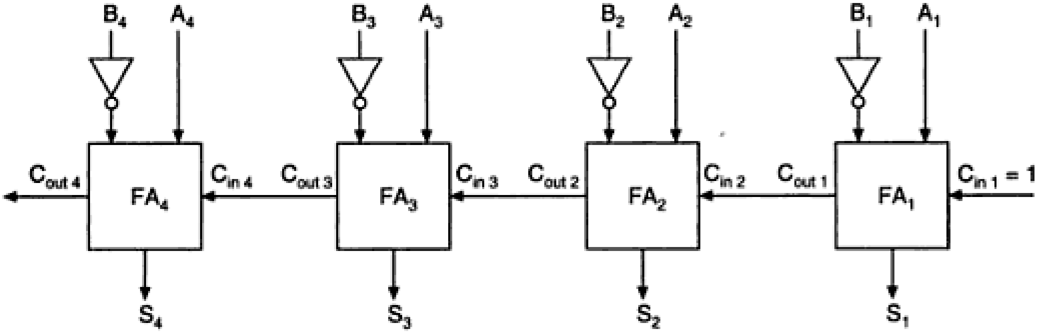


Fig 10:parallel subtractor

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.

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

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

CARRY LOOK-A- HEAD ADDER

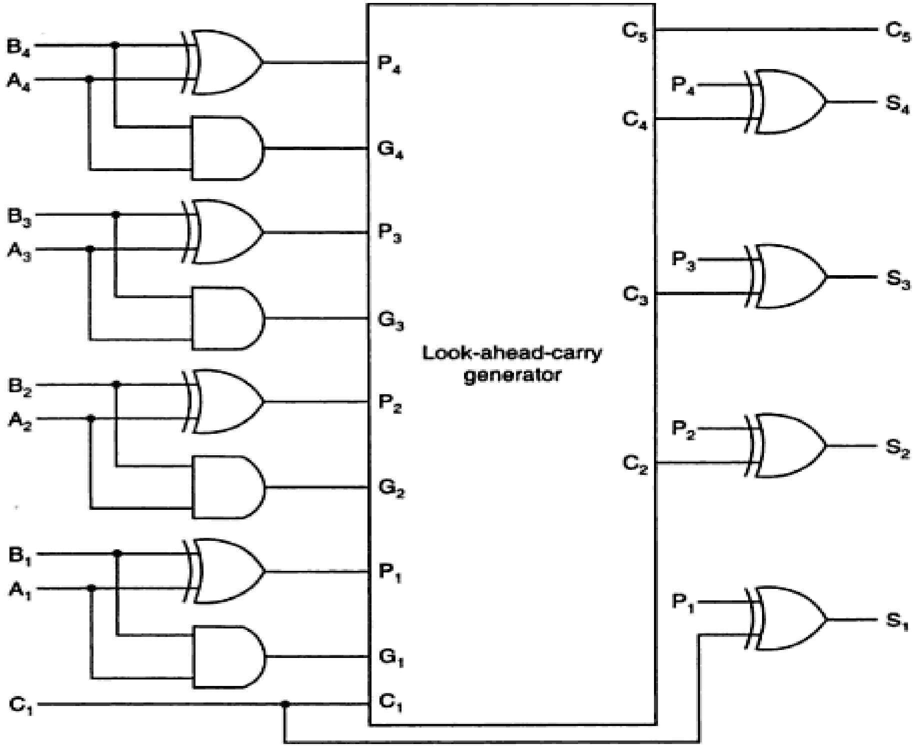


Fig:1 block diagram

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* × *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

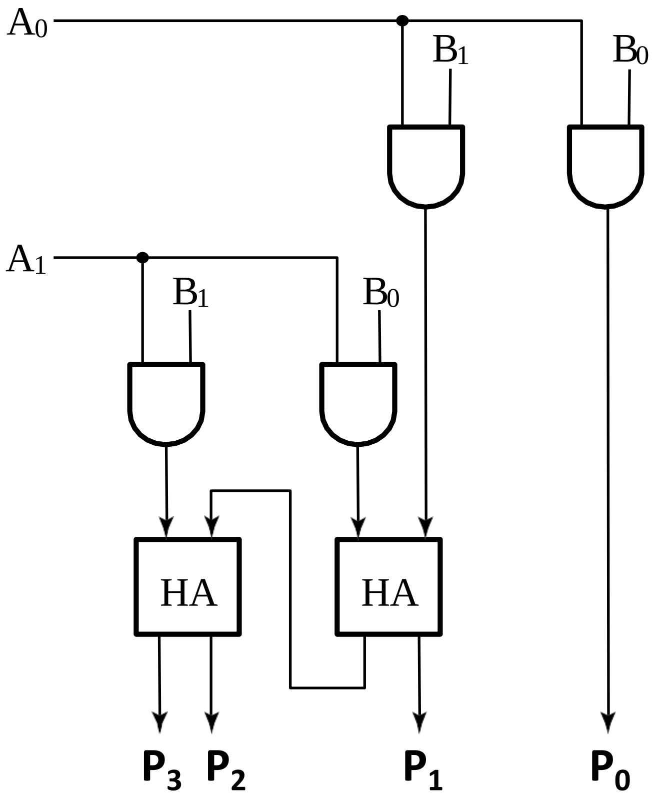


Fig 1: 2 x 2 multiplier array

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.

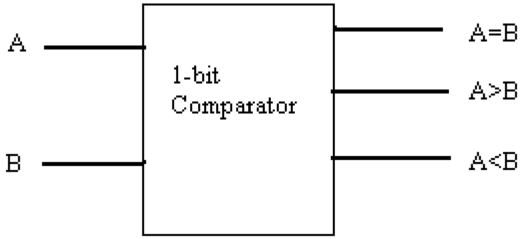


Fig :1 Block diagram

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

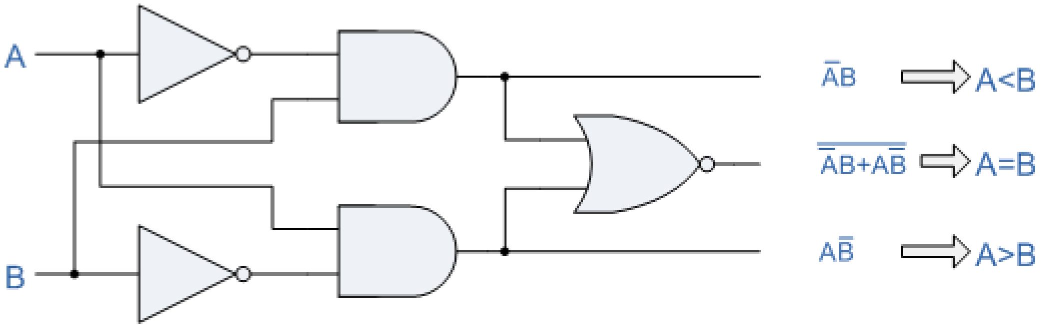


Fig 2:Logic diagram of 1-bit comparator

MAGNITUDE COMPARATOR

- 2 **Bit magnitude comparator**

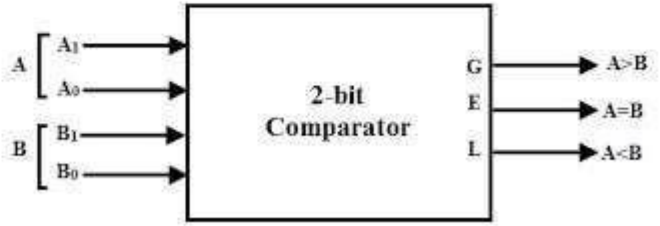
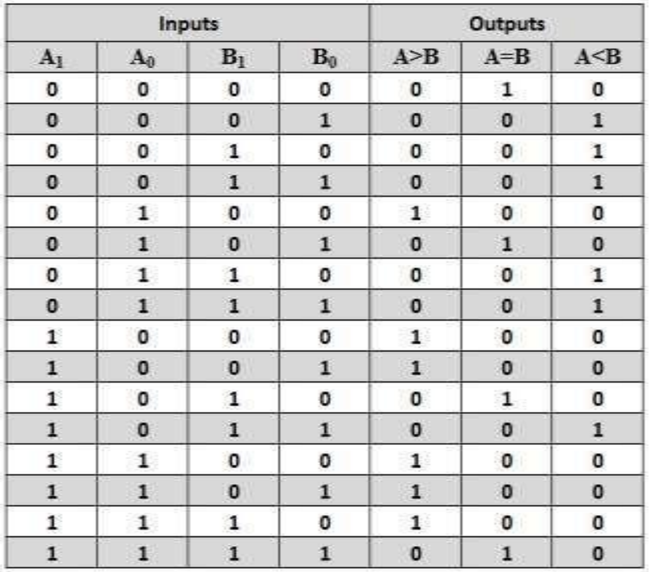


Fig :3 Block diagram



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

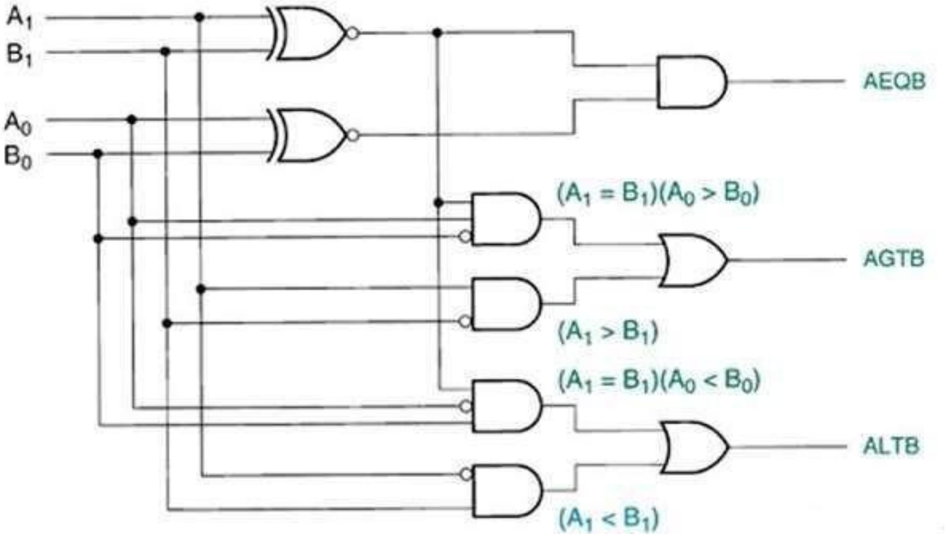
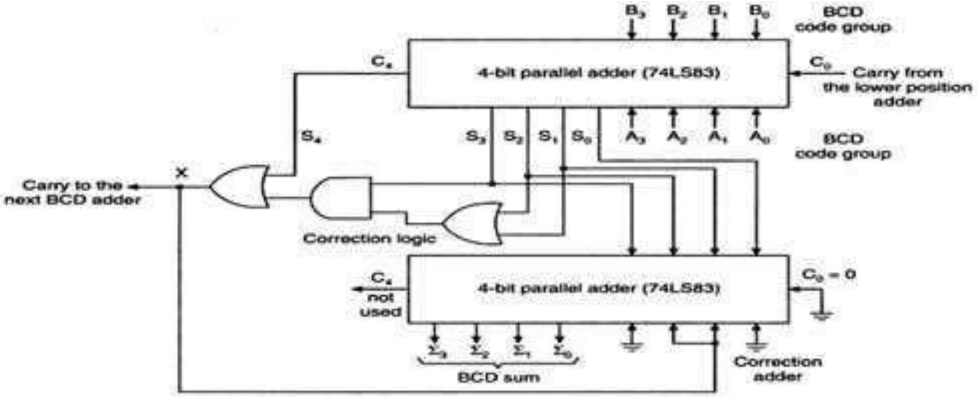


Fig 5:Logic diagram of 2-bit comparator

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^nunique outputs.

- We have following types of decoders 2x4,3x8,4x16….

2x4 decoder

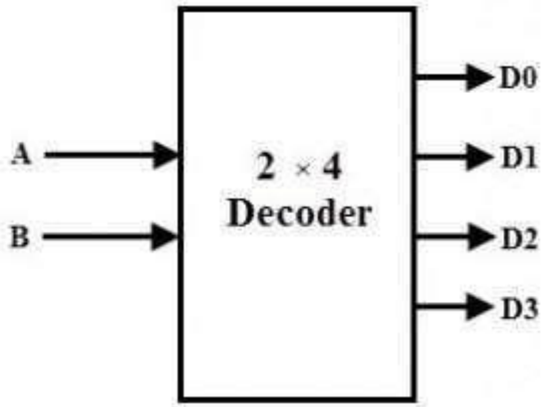
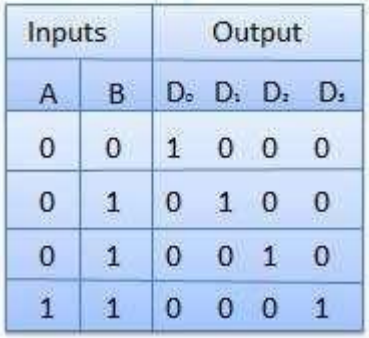


Fig 1: Block diagram



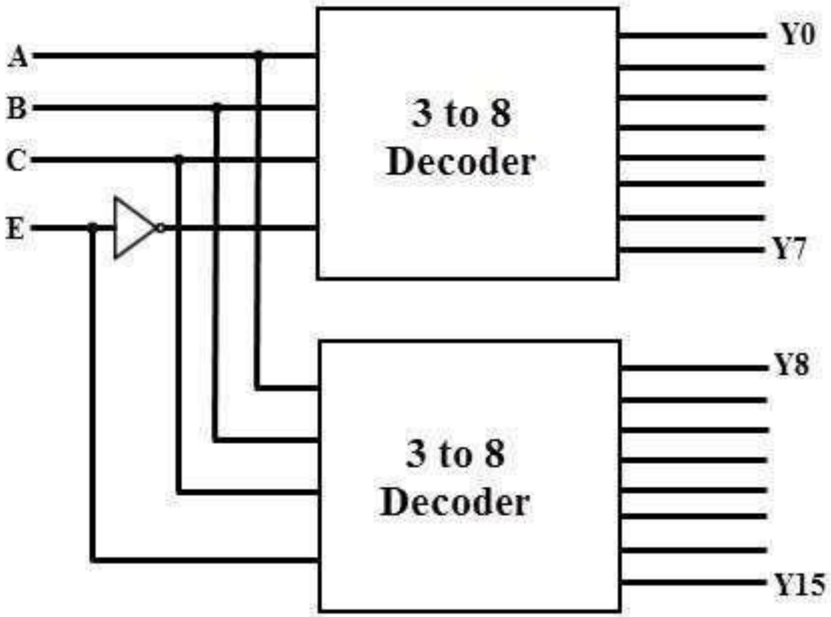
Inputs		Output			
A	B	D_0	D_1	D_2	D_3
0	0	1	0	0	0
0	1	0	1	0	0
0	1	0	0	1	0
1	1	0	0	0	1

Fig 2:Truth table

DECODERS

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.

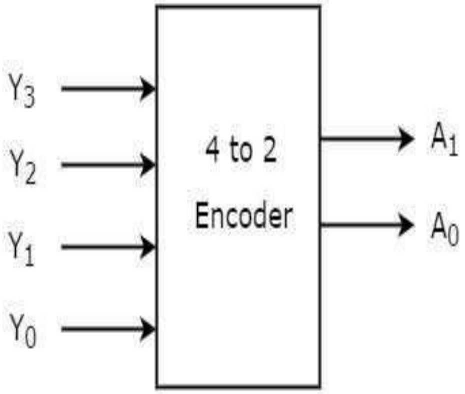
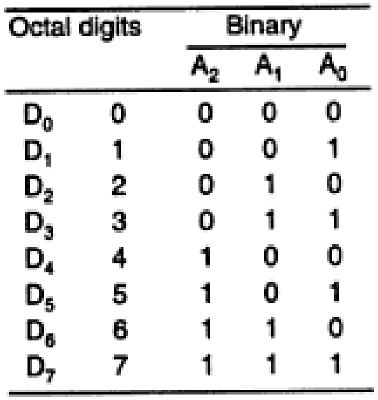


Fig 1:block diagram of 4x2 encoder

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

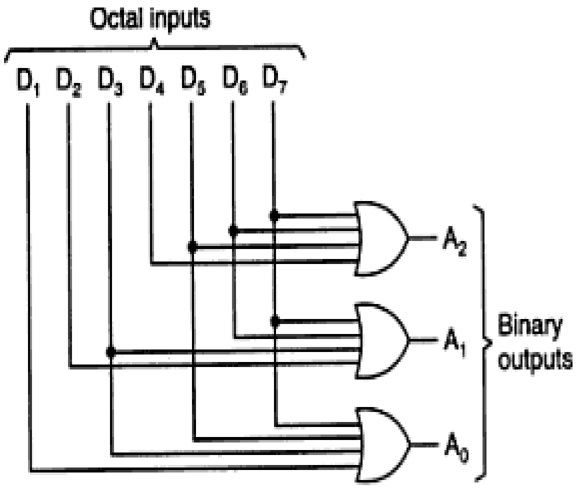
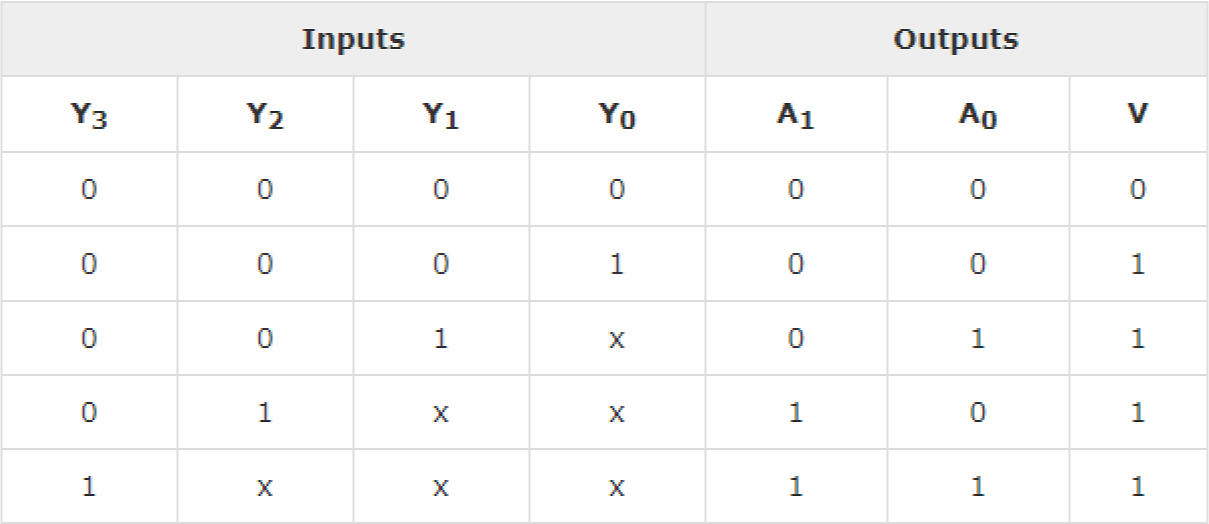


Fig 3: Logic diagram

ENCODER

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……

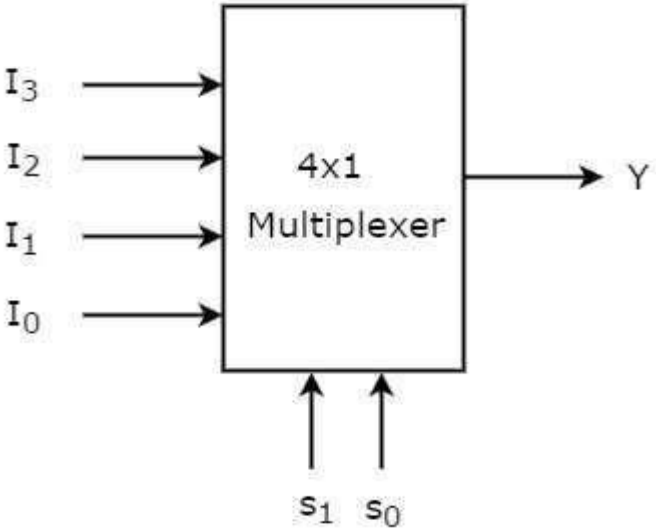
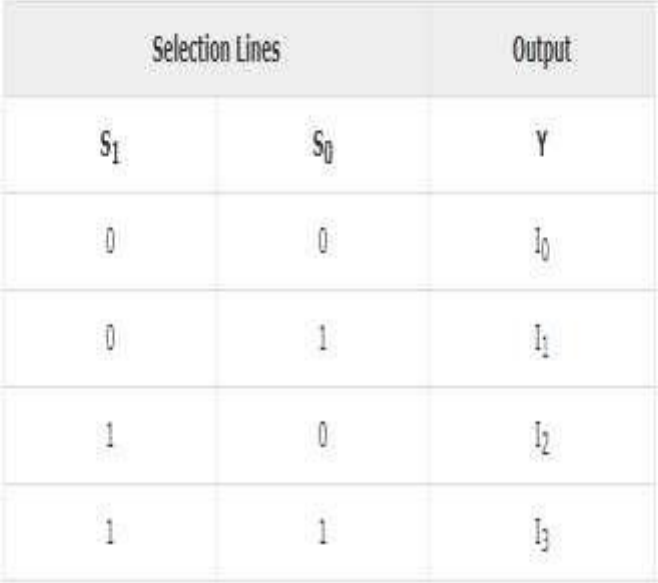


Fig 1: Block diagram



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

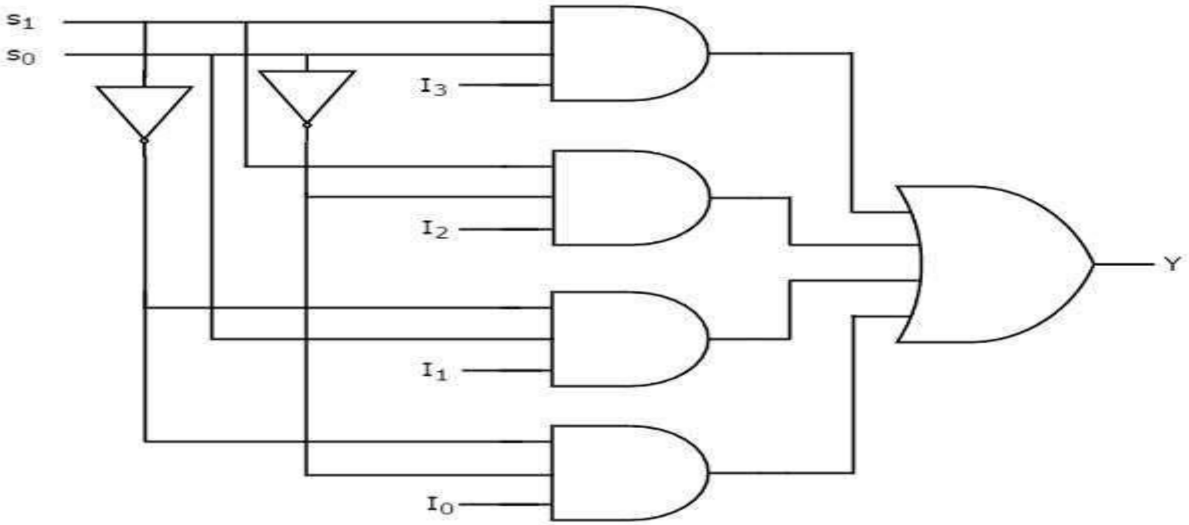


Fig 3: Logic diagram

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

MULTIPLEXERS

- Ex: 8x1 Multiplexer

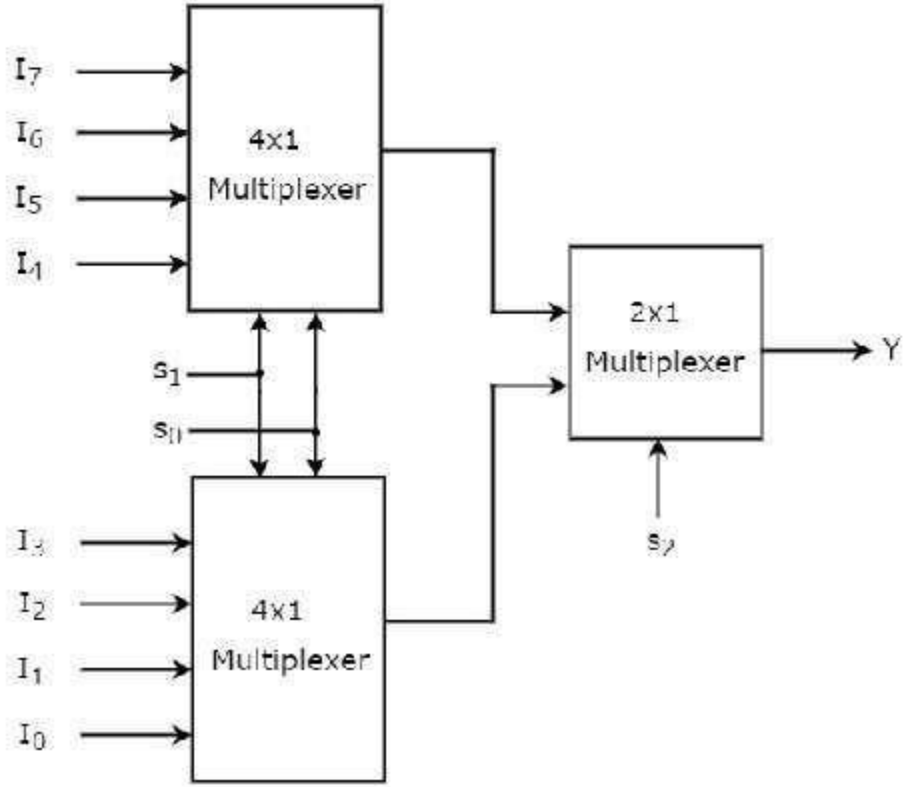
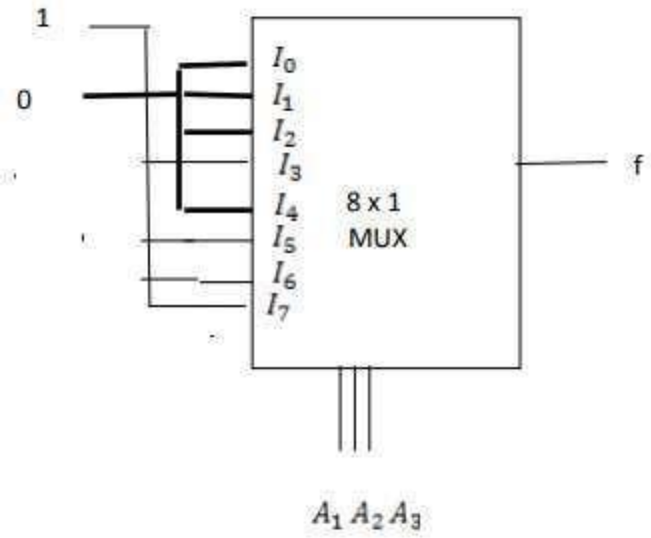


Fig 3: 8x1 Multiplexer diagram

MULTIPLEXERS

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



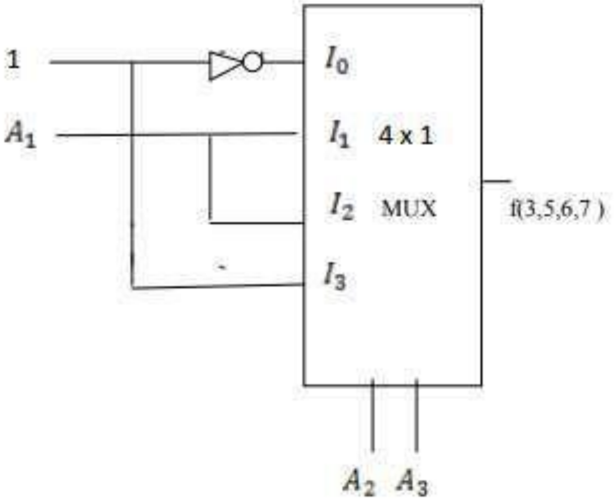
MULTIPLEXERS

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

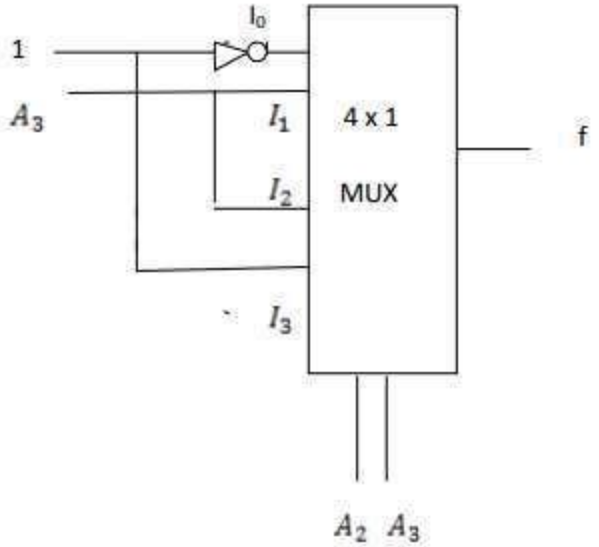
	I_0	I_1	I_2	I_3
$\overline{A_1}$	0	1	2	3
A_1	4	5	6	7
	0	A_1	A_1	1



MULTIPLEXERS

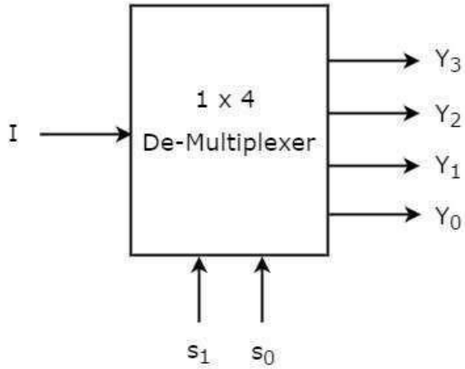
Method:2

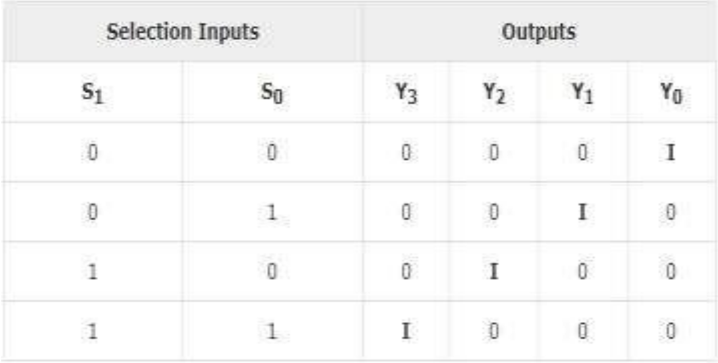
Minterm	A_1	A_2	A_3	f		
0	0	0	0	0		
1	0	0	1	0	f= 0	I_0
2	0	1	0	0		
3	0	1	1	1	f= A_3	I_1
4	1	0	0	0		
5	1	0	1	1	f= A_3	I_2
6	1	1	0	1		
7	1	1	1	1	f= 1	I_3



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.





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:1 Block diagram

Fig :2 Truth table

DEMULTIPLEXER

Boolean functions for each output as

$$Y_3 = s_1 s_0 I$$

$$Y_2 = s_1 s_0' I$$

$$Y_1 = s_1' s_0 I$$

$$Y_0 = s_1' s_0' I$$

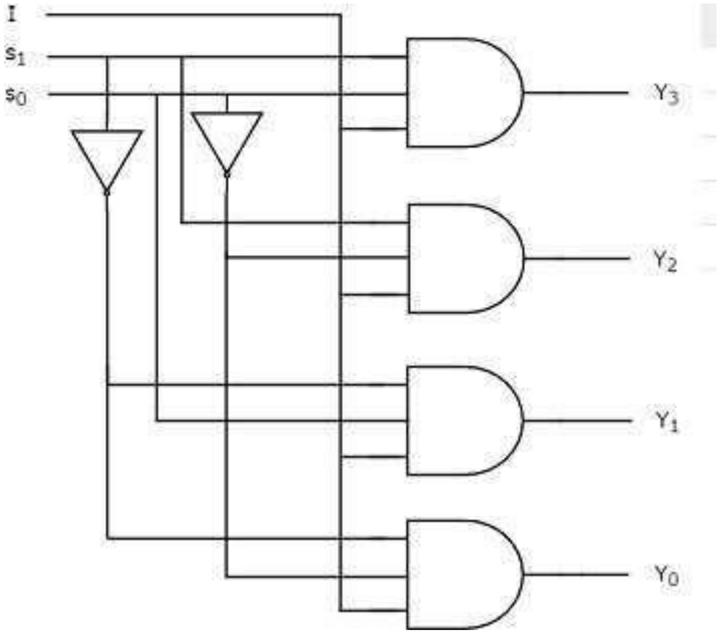
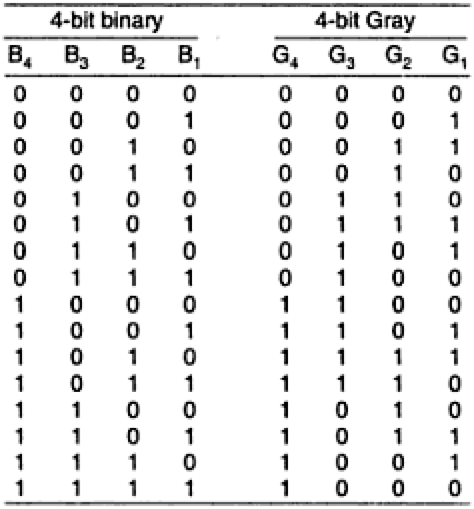


Fig:3 Logic diagram

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

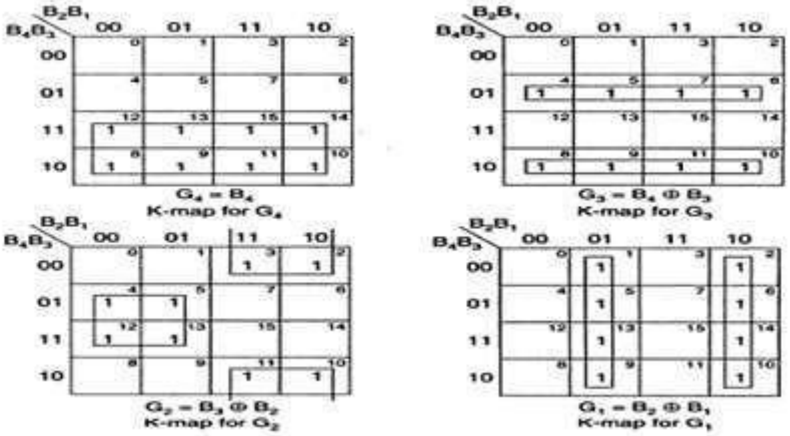


4-bit binary				4-bit Gray			
B_4	B_3	B_2	B_1	G_4	G_3	G_2	G_1
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1
1	0	1	0	1	1	1	1
1	0	1	1	1	1	1	0
1	1	0	0	1	0	1	0
1	1	0	1	1	0	1	1
1	1	1	0	1	0	0	1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

Fig :1 Truth table

CODE CONVERTERS

K-map simplification



$B_4B_3 \backslash B_2B_1$	00	01	11	10
00				
01				
11	1	1	1	1
10	1	1	1	1

$G_4 = B_4$

K-map for G_4

$B_4B_3 \backslash B_2B_1$	00	01	11	10
00				
01	1	1	1	1
11				
10	1	1	1	1

$G_3 = B_4 \oplus B_3$

K-map for G_3

$B_4B_3 \backslash B_2B_1$	00	01	11	10
00			1	1
01	1	1		
11	1	1		
10			1	1

$G_2 = B_3 \oplus B_2$

K-map for G_2

$B_4B_3 \backslash B_2B_1$	00	01	11	10
00		1		1
01		1		1
11		1		1
10		1		1

$G_1 = B_2 \oplus B_1$

K-map for G_1

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

$$G_3 = \Sigma m(4, 5, 6, 7, 8, 9, 10, 11) \qquad G_3 = \bar{B}_4B_3 + B_4\bar{B}_3 = B_4 \oplus B_3$$

$$G_2 = \Sigma m(2, 3, 4, 5, 10, 11, 12, 13) \qquad G_2 = \bar{B}_3B_2 + B_3\bar{B}_2 = B_3 \oplus B_2$$

$$G_1 = \Sigma m(1, 2, 5, 6, 9, 10, 13, 14) \qquad G_1 = \bar{B}_2B_1 + B_2\bar{B}_1 = B_2 \oplus B_1$$

CODE CONVERTERS

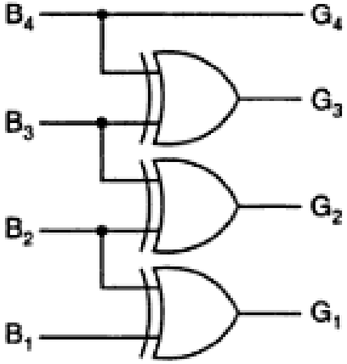


Fig: 2 Logic diagram

UNIT 4

INTRODUCTION TO SEQUENTIAL LOGIC CIRCUITS

SEQUENTIAL LOGIC CIRCUITS

Sequential logic circuit consists of a combinational circuit with storage elements connected as a feedback to combinational circuit

- output depends on the sequence of inputs (past and present)
- stores information (state) from past inputs

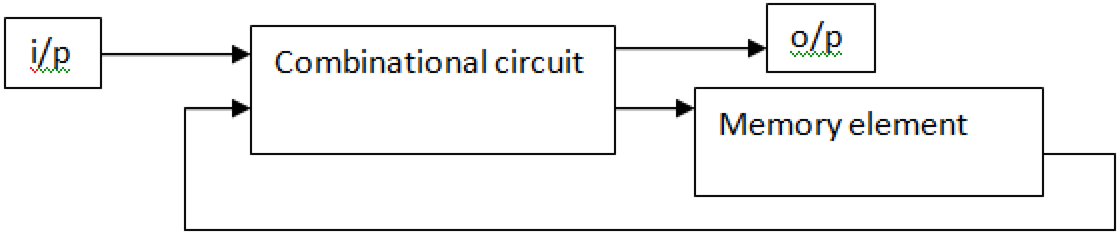
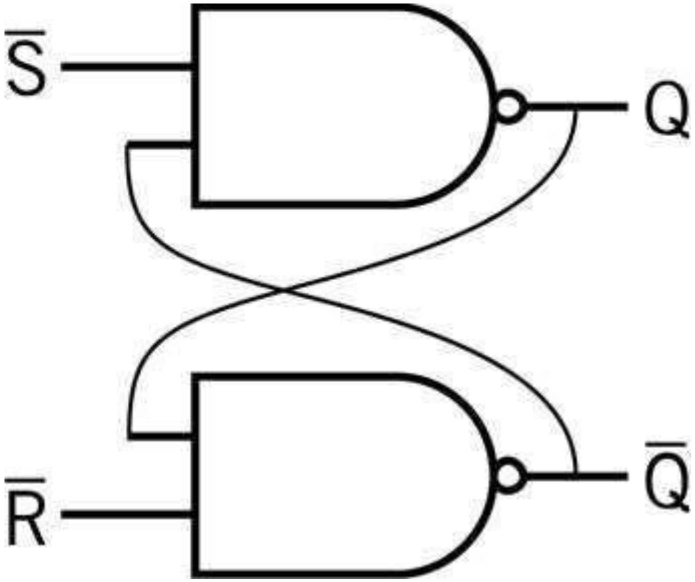


Figure 1: Sequential logic circuits

SEQUENTIAL LOGIC CIRCUITS

- Output depends on
  - Input
  - Previous state of the circuit
- Flip-flop: basic memory element
- State table: output for all combinations of input and previous states(Truth Table)



SEQUENTIAL LOGIC CIRCUITS

1. Sequential circuit receives the binary information from external inputs and with the present state of the storage elements together determine the binary value of the outputs.
2. The output in a sequential circuit are a function of not only the inputs, but also the present state of the storage elements.
3. The next state of the storage elements is also a function of external inputs and the present state.
4. There are two main types of sequential circuits

   1. synchronous sequential circuits

   2. asynchronous sequential circuits

SEQUENTIAL LOGIC CIRCUITS

Synchronous sequential circuits

It is a system whose behaviour can be defined from the knowledge of its signals at discrete instants of time

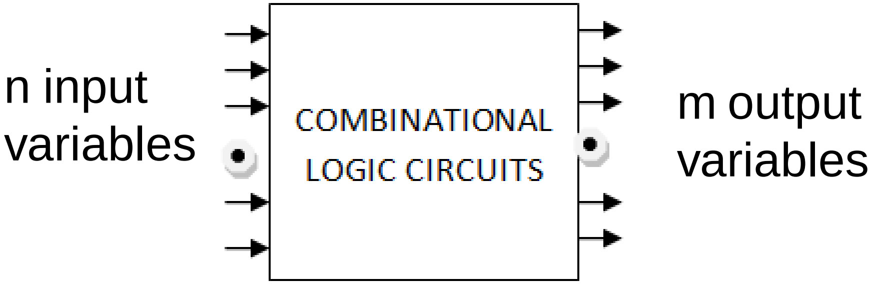
Asynchronous sequential circuits

It depends upon the input signals at any instant of time and the order in which the input changes

Combinational vs. Sequential

COMBINATIONAL LOGIC CIRCUIT

Combinational logic circuit consists of input variables, logic gates and output variables. The logic gate accepts signals from the inputs and generates signals to the outputs.



For n input variables there are 2n possible combinations of binary input variables

Combinational vs. Sequential

SEQUENTIAL LOGIC CIRCUITS

Sequential logic circuit consists of a combinational circuit with storage elements connected as a feedback to combinational circuit

- output depends on the sequence of inputs (past and present)
- stores information (state) from past inputs

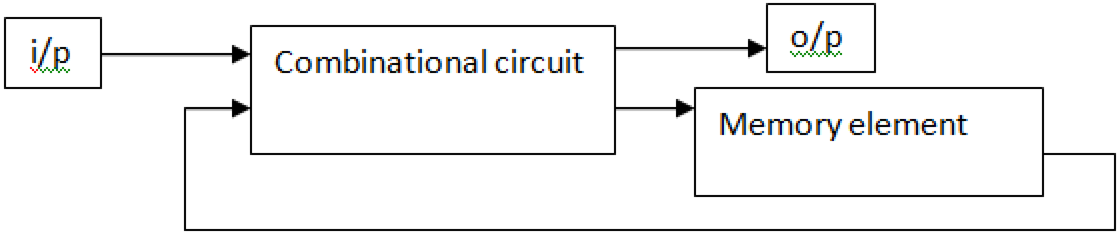


Figure 1: Sequential logic circuits

Combinational vs. Sequential

Combinational Circuit

always gives the same output for a given set of inputs

ex: adder always generates sum and carry, regardless of previous inputs

Sequential Circuit

stores information

output depends on stored information (state) plus input

so a given input might produce different outputs, depending on the stored information

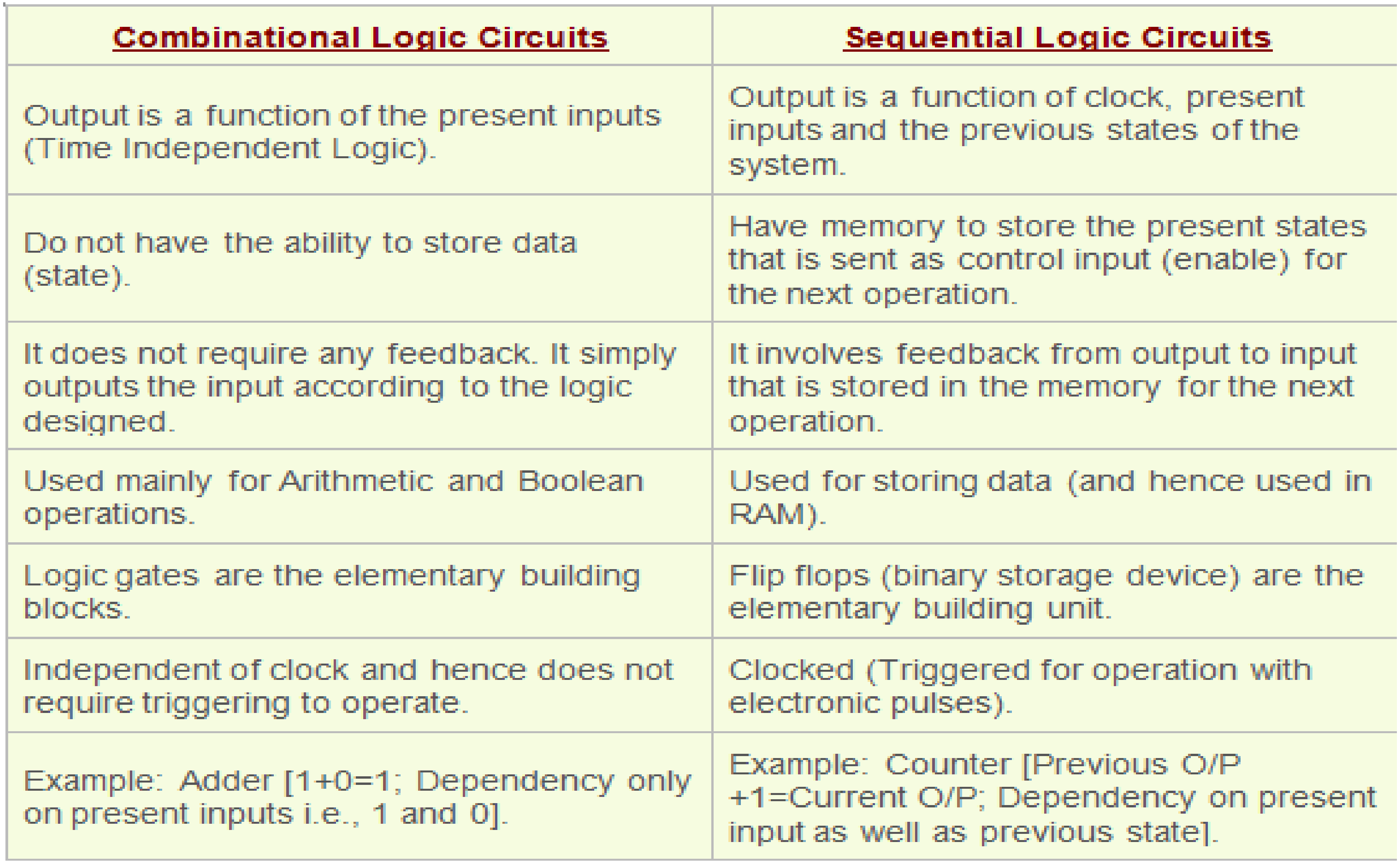
example: ticket counter

advances when you push the button

output depends on previous state

useful for building "memory" elements and "state machines"

Combinational vs. Sequential



Combinational Logic Circuits	Sequential Logic Circuits
Output is a function of the present inputs (Time Independent Logic).	Output is a function of clock, present inputs and the previous states of the system.
Do not have the ability to store data (state).	Have memory to store the present states that is sent as control input (enable) for the next operation.
It does not require any feedback. It simply outputs the input according to the logic designed.	It involves feedback from output to input that is stored in the memory for the next operation.
Used mainly for Arithmetic and Boolean operations.	Used for storing data (and hence used in RAM).
Logic gates are the elementary building blocks.	Flip flops (binary storage device) are the elementary building unit.
Independent of clock and hence does not require triggering to operate.	Clocked (Triggered for operation with electronic pulses).
Example: Adder [1+0=1; Dependency only on present inputs i.e., 1 and 0].	Example: Counter [Previous O/P +1=Current O/P; Dependency on present input as well as previous state].

LATCHES

STORAGE ELEMENTS

Storage elements in a digital circuit can maintain a binary state indefinitely, until directed by an input signal to switch states. The major difference among various storage elements are the number of input they posses and the manner in which the inputs affect the binary state.There are two types of storage elements

1. Latches
2. Flipflops

Storage elements that operate with signal level are referred as latch and those controlled by a clock transition are referred as flipflops.

LATCHES

1. LATCHES:

A latch has a feedback path, so information can be retained by the device. Therefore latches can be memory devices, and can store one bit of data for as long as the device is powered. As the name suggests, latches are used to "latch onto" information and hold in place. Latches are very similar to flip-flops, but are not synchronous devices, and do not operate on clock edges as flip-flops do. Latch is a level sensitive device. Latch is a monostable multivibrator

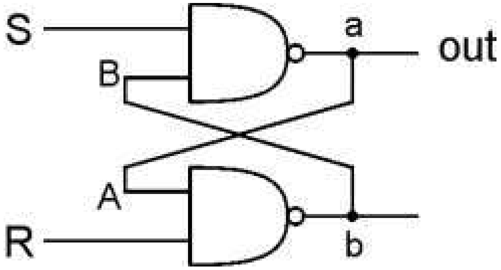
2. FLIPFLOPS:

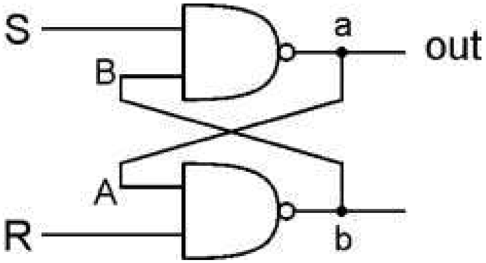
A **flip-flop** is a circuit that has two stable states and can be used to store state information. A flip-flop is a bistable multivibrator. The circuit can be made to change state by signals applied to one or more control inputs and will have one or two outputs. It is the basic storage element in sequential logic. Flip-flops and latches are fundamental building blocks of digital electronics systems used in computers, communications, and many other types of systems. Flipflop is a edge sensitive device.

LATCHES

SR LATCH

An **SR latch** (Set/Reset) is an asynchronous device: it works independently of control signals and relies only on the state of the S and R inputs. In the image we can see that an SR flip-flop can be created with two NOR gates that have a cross-feedback loop. SR latches can also be made from NAND gates, but the inputs are swapped and negated. In this case, it is sometimes called an **SR latch**.





R is used to “reset” or “clear” the element – set it to zero. S is used to “set” the element – set it to one.

If both R and S are one, out could be <u>either</u> zero or one. “quiescent” state -- holds its previous value. note: if a is 1, b is 0, and vice versa

LATCHES

GATED D-LATCH

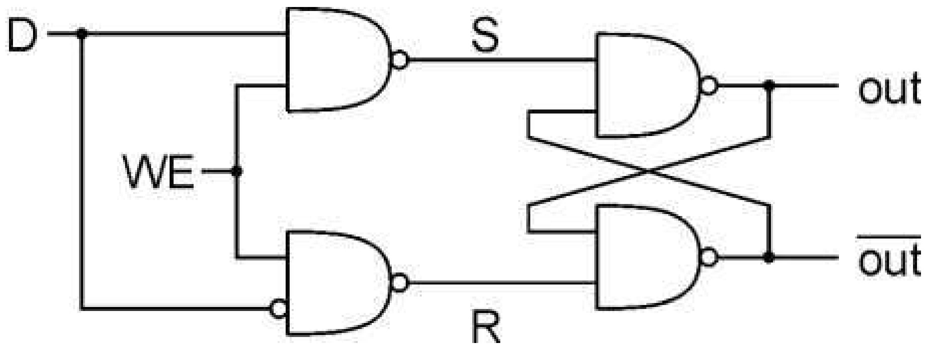
The **D latch** (D for "data") or **transparent latch** is a simple extension of the gated SR latch that removes the possibility of invalid input states. Two inputs: D (data) and WE (write enable)

when WE = 1, latch is set to value of D

S = NOT(D), R = D

when WE = 0, latch holds previous value

S = R = 1



FLIPFLOPS:RS FLIPFLOP

Flip flops

A flip flop is an electronic circuit with two stable states that can be used to store binary data. The stored data can be changed by applying varying inputs. Flip-flops and latches are fundamental building blocks of digital electronics systems used in computers, communications, and many other types of systems. Flip-flops and latches are used as data storage elements. There are 4 types of flipflops

1. RS flip flop
2. Jk flip flop
3. D flip flop
4. T flip flop

Applications of Flip-Flops

These are the various types of flip-flops being used in digital electronic circuits and the applications like Counters, Frequency Dividers, Shift Registers, Storage Registers

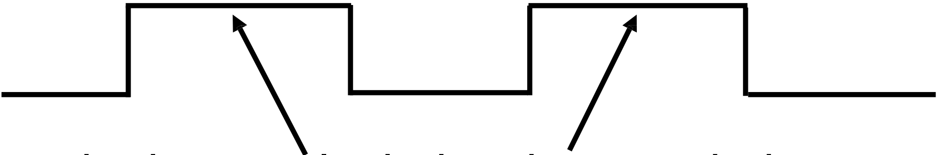
FLIPFLOPS:RS FLIPFLOP

EDGE-TRIGGERED FLIP FLOPS

Characteristics

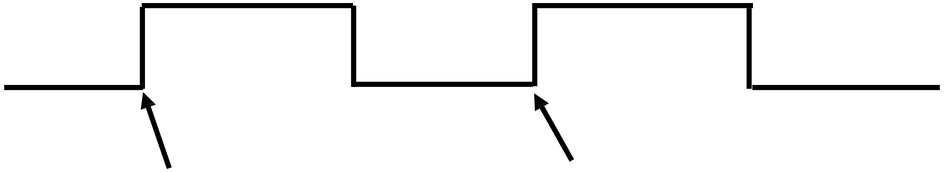
- **State transition occurs at the rising edge or falling edge of the clock pulse**

Latches



respond to the input only during these periods

Edge-triggered Flip Flops (positive)



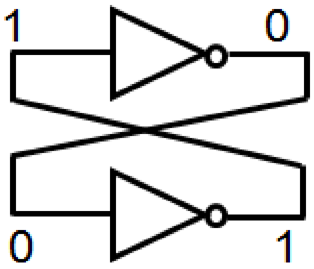
respond to the input only at this time

FLIPFLOPS:RS FLIPFLOP

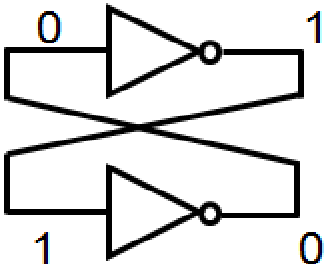
FLIP FLOPS

Characteristics

- **2 stable states**
- **Memory capability**
- **Operation is specified by a Characteristic Table**

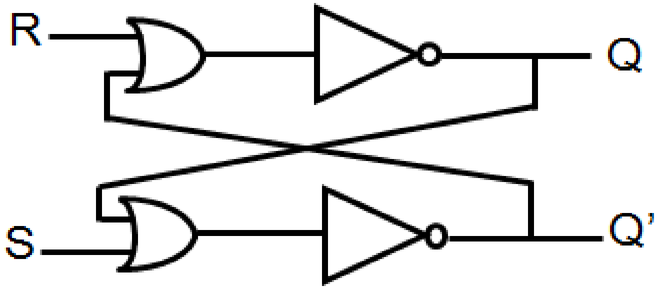


0-state



1-state

In order to be used in the computer circuits, state of the flip flop should have input terminals and output terminals so that it can be set to a certain state, and its state can be read externally.



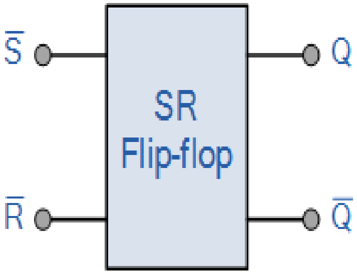
S	R	Q(t+1)
0	0	Q(t)
0	1	0
1	0	1
1	1	indeterminate (forbidden)

FLIPFLOPS:RS FLIPFLOP

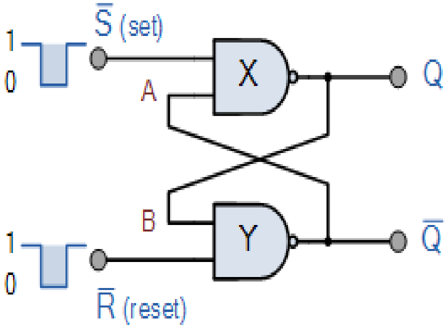
SR Flip-Flop

The **SR flip-flop**, also known as a *SR Latch*, can be considered as one of the most basic sequential logic circuit possible. This simple flip-flop is basically a one-bit memory bistable device that has two inputs, one which will “SET” the device (meaning the output = “1”), and is labelled **S** and one which will “RESET” the device (meaning the output = “0”), labelled **R**.The reset input resets the flip-flop back to its original state with an output Q that will be either at a logic level “1” or logic “0” depending upon this set/reset condition.

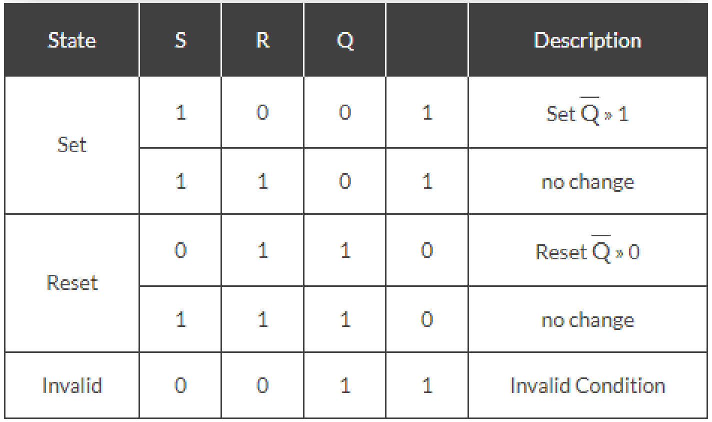
A basic NAND gate SR flip-flop circuit provides feedback from both of its outputs back to its opposing inputs and is commonly used in memory circuits to store a single data bit. Then the SR flip-flop actually has three inputs, Set, Reset and its current output Q relating to it’s current state or history.



Symbol



Circuit



State	S	R	Q		Description
Set	1	0	0	1	Set $\overline{Q}$ » 1
	1	1	0	1	no change
Reset	0	1	1	0	Reset $\overline{Q}$ » 0
	1	1	1	0	no change
Invalid	0	0	1	1	Invalid Condition

Truth Table for this Set-Reset Function

FLIPFLOPS:JK FLIPFLOP

JK Flip flop

The JK Flip-flop is similar to the SR Flip-flop but there is no change in state when the J and K inputs are both LOW. The basic S-R NAND flip-flop circuit has many advantages and uses in sequential logic circuits but it suffers from two basic switching problems.

1. the Set = 0 and Reset = 0 condition (S = R = 0) must always be avoided
2. if Set or Reset change state while the enable (EN) input is high the correct latching action may not occur

Then to overcome these two fundamental design problems with the SR flip-flop design, the **JK flip Flop** was developed by the scientist name Jack Kirby.

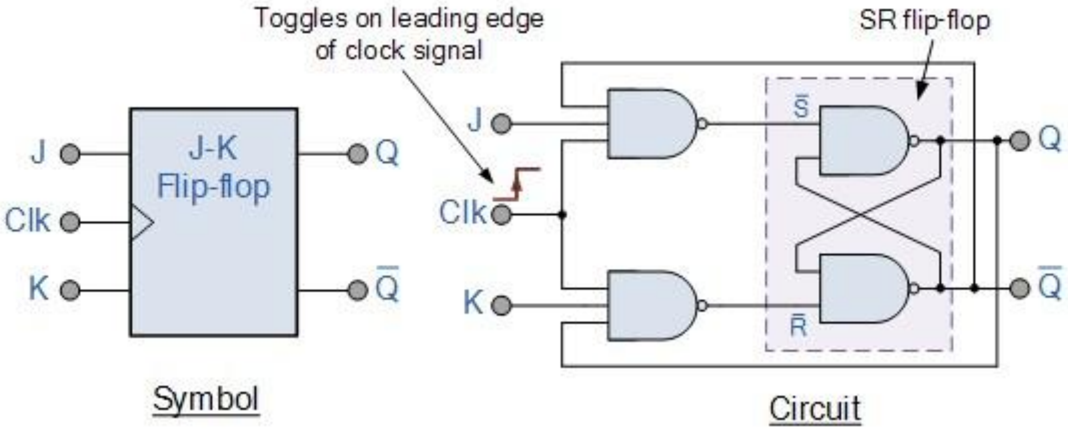
The **JK flip flop** is basically a gated SR flip-flop with the addition of a clock input circuitry that prevents the illegal or invalid output condition that can occur when both inputs S and R are equal to logic level “1”. Due to this additional clocked input, a JK flip-flop has four possible input combinations, “logic 1”, “logic 0”, “no change” and “toggle”. The symbol for a JK flip flop is similar to that of an *SR Bistable Latch* as seen in the previous tutorial except for the addition of a clock input.

FLIPFLOPS:JK FLIPFLOP

❖ Both the S and the R inputs of the previous SR bistable have now been replaced by two inputs called the J and K inputs, respectively after its inventor Jack Kilby. Then this equates to: J = S and K = R.

❖ The two 2-input AND gates of the gated SR bistable have now been replaced by two 3-input NAND gates with the third input of each gate connected to the outputs at Q and Q. This cross coupling of the SR flip-flop allows the previously invalid condition of S = "1" and R = "1" state to be used to produce a "toggle action" as the two inputs are now interlocked.

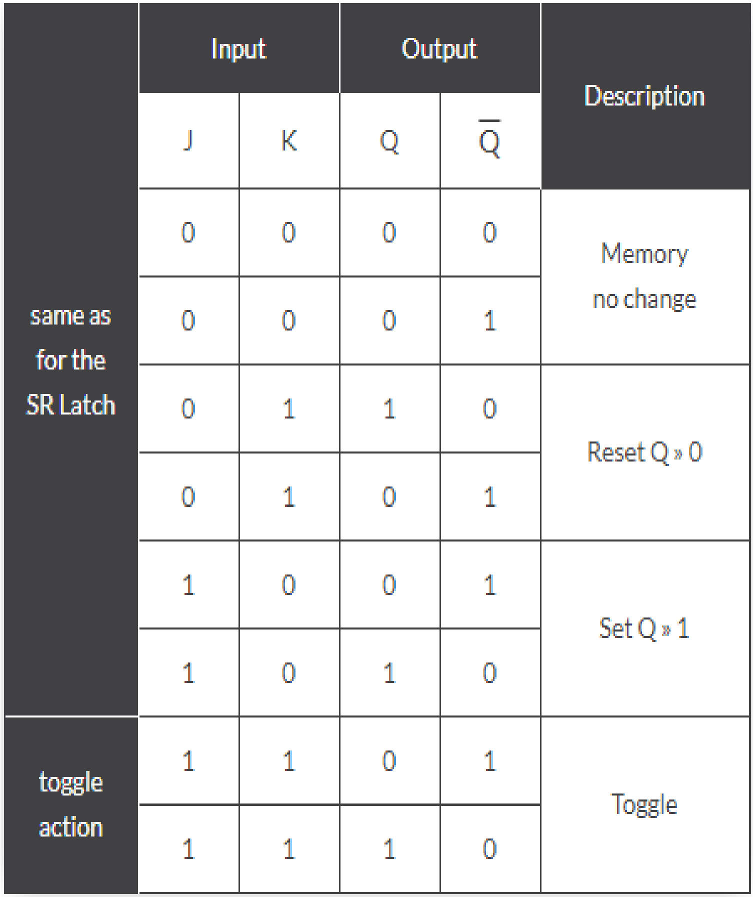
❖ If the circuit is now "SET" the J input is inhibited by the "0" status of Q through the lower NAND gate. If the circuit is "RESET" the K input is inhibited by the "0" status of Q through the upper NAND gate. As Q and Q are always different we can use them to control the input. When both inputs J and K are equal to logic "1", the JK flip flop toggles as shown in the following truth table.



FLIPFLOPS:JK FLIPFLOP

The Truth Table for the JK Function

Then the JK flip-flop is basically an SR flip flop with feedback which enables only one of its two input terminals, either SET or RESET to be active at any one time thereby eliminating the invalid condition seen previously in the SR flip flop circuit. Also when both the J and the K inputs are at logic level “1” at the same time, and the clock input is pulsed “HIGH”, the circuit will “toggle” from its SET state to a RESET state, or visa-versa. This results in the JK flip flop acting more like a T-type toggle flip-flop when both terminals are“HIGH”.

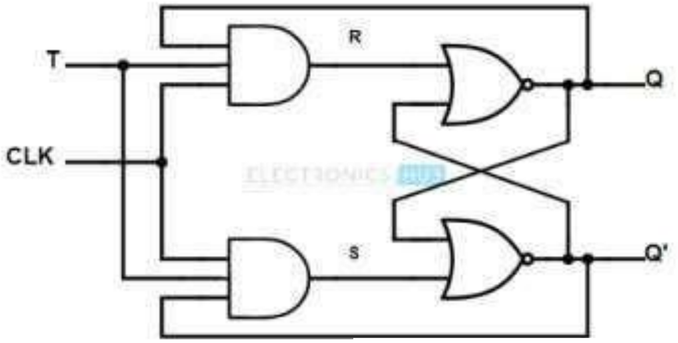


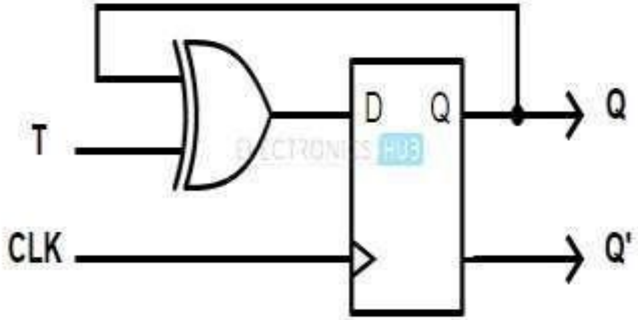
	Input		Output		Description
	J	K	Q	$\overline{Q}$	
same as for the SR Latch	0	0	0	0	Memory no change
	0	0	0	1	
	0	1	1	0	Reset Q » 0
	0	1	0	1	
	1	0	0	1	Set Q » 1
	1	0	1	0	
toggle action	1	1	0	1	Toggle
	1	1	1	0	

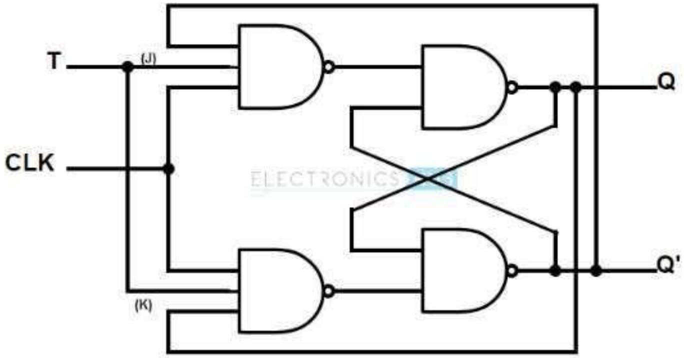
FLIPFLOPS:T FLIPFLOP

T FLIP FLOP

We can construct a T flip flop by any of the following methods. Connecting the output feedback to the input, in SR flip flop. Connecting the XOR of T input and Q PREVIOUS output to the Data input, in D flip flop. Hard – wiring the J and K inputs together and connecting it to T input, in JK flip – flop.





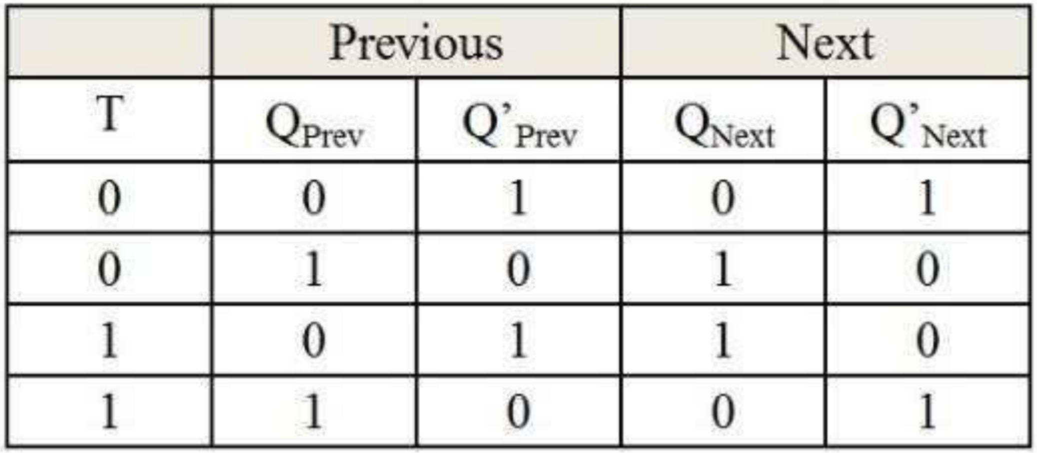


FLIPFLOPS:T FLIPFLOP

Working

T flip – flop is an edge triggered device i.e. the low to high or high to low transitions on a clock signal of narrow triggers that is provided as input will cause the change in output state of flip – flop. T flip – flop is an edge triggered device.

Truth Table of T flip – flop



	Previous		Next	
T	Q_{Prev}	Q'_{Prev}	Q_{Next}	Q'_{Next}
0	0	1	0	1
0	1	0	1	0
1	0	1	1	0
1	1	0	0	1

FLIPFLOPS:T FLIPFLOP

- If the output Q = 0, then the upper NAND is in enable state and lower NAND gate is in disable condition. This allows the trigger to pass the S inputs to make the flip – flop in SET state i.e. Q = 1.
- If the output Q = 1, then the upper NAND is in disable state and lower NAND gate is in enable condition. This allows the trigger to pass the R inputs to make the flip – flop in RESET state i.e. Q =0.

In simple terms, the operation of the T flip – flop is

When the T input is low, then the next sate of the T flip flop is same as the present state.

T = 0 and present state = 0 then the next state = 0

T = 1 and present state = 1 then the next state = 1

When the T input is high and during the positive transition of the clock signal, the next state of the T flip – flop is the inverse of present state.

T = 1 and present state = 0 then the next state = 1

T = 1 and present state = 1 then the next state = 0

Applications

Frequency Division Circuits.

2 – Bit Parallel Load Registers.

D FLIP FLOP

The D-type flip-flop is a modified Set-Reset flip-flop with the addition of an inverter to prevent the S and R inputs from being at the same logic level

One of the main disadvantages of the basic **SR NAND Gate Bistable** circuit is that the indeterminate input condition of SET = “0” and RESET = “0” is forbidden.

This state will force both outputs to be at logic “1”, over-riding the feedback latching action and whichever input goes to logic level “1” first will lose control, while the other input still at logic “0” controls the resulting state of the latch.

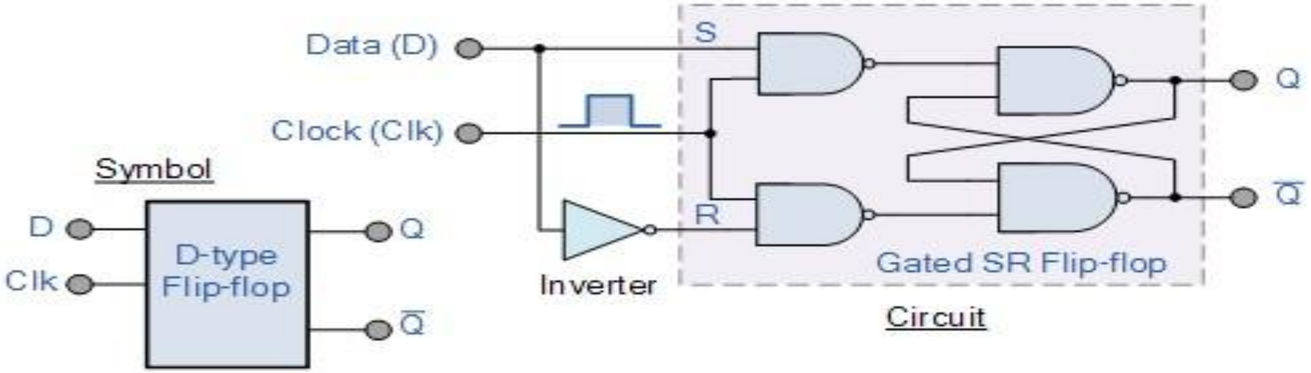
But in order to prevent this from happening an inverter can be connected between the “SET” and the “RESET” inputs to produce another type of flip flop circuit known as a **Data Latch**, **Delay flip flop**, **D-type Bistable**, **D-type Flip Flop** or just simply a **D Flip Flop** as it is more generally called.

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

Then this single data input, labelled “D” and is used in place of the “Set” signal, and the inverter is used to generate the complementary “Reset” input thereby making a level-sensitive D-type flip-flop from a level-sensitive SR-latch as now S = D and R = not D as shown.

FLIPFLOPS:D FLIPFLOP

D-type Flip-Flop Circuit



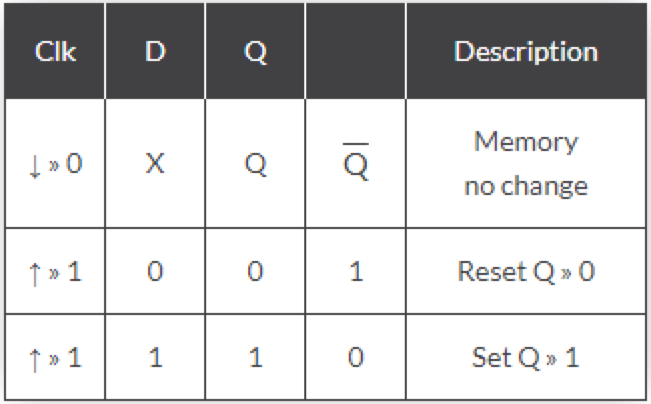
We remember that a simple SR flip-flop requires two inputs, one to “SET” the output and one to “RESET” the output. By connecting an inverter (NOT gate) to the SR flip-flop we can “SET” and “RESET” the flip-flop using just one input as now the two input signals are complements of each other. This complement avoids the ambiguity inherent in the SR latch when both inputs are LOW, since that state is no longer possible. Thus this single input is called the “DATA” input. If this data input is held HIGH the flip flop would be “SET” and when it is LOW the flip flop would change and become “RESET”. However, this would be rather pointless since the output of the flip flop would always change on every pulse applied to this data input.

FLIPFLOPS:D FLIPFLOP

To avoid this an additional input called the “CLOCK” or “ENABLE” input is used to isolate the data input from the flip flop’s latching circuitry after the desired data has been stored. The effect is that D input condition is only copied to the output Q when the clock input is active. This then forms the basis of another sequential device called a **D Flip Flop**.

The “D flip flop” will store and output whatever logic level is applied to its data terminal so long as the clock input is HIGH. Once the clock input goes LOW the “set” and “reset” inputs of the flip-flop are both held at logic level “1” so it will not change state and store whatever data was present on its output before the clock transition occurred. In other words the output is “latched”at either logic “0” or logic “1”.

Truth Table for the D-type Flip Flop



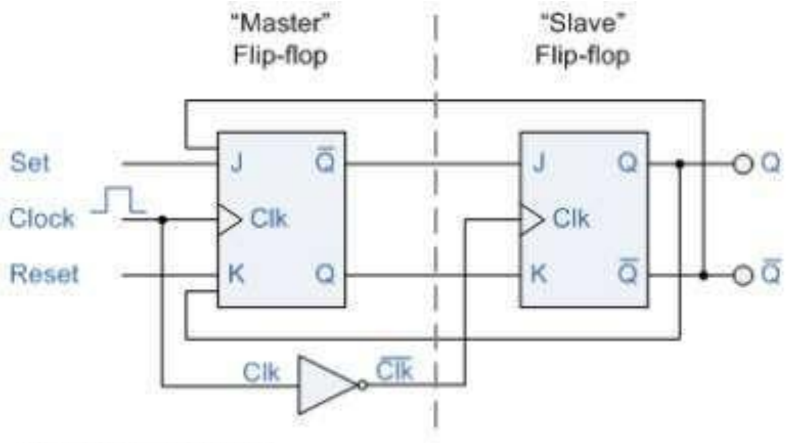
Clk	D	Q		Description
↓ » 0	X	Q	$\overline{Q}$	Memory no change
↑ » 1	0	0	1	Reset Q » 0
↑ » 1	1	1	0	Set Q » 1

Note that: ↓ and ↑ indicates direction of clock pulse as it is assumed D-type flip flops are edge triggered

FLIPFLOPS:MASTER SLAVE FLIPFLOP

MASTER SLAVE FLIPFLOP

Master-slave flip flop is designed using two separate flip flops. Out of these, one acts as the master and the other as a slave. The figure of a master-slave J-K flip flop is shown below.



Master Slave Flip Flop

From the above figure you can see that both the J-K flip flops are presented in a series connection. The output of the master J-K flip flop is fed to the input of the slave J-K flip flop. The output of the slave J-K flip flop is given as a feedback to the input of the master J-K flip flop. The clock pulse [Clk] is given to the master J-K flip flop and it is sent through a NOT Gate and thus inverted before passing it to the slave J-K flip flop.

FLIPFLOPS:MASTER SLAVE FLIPFLOP

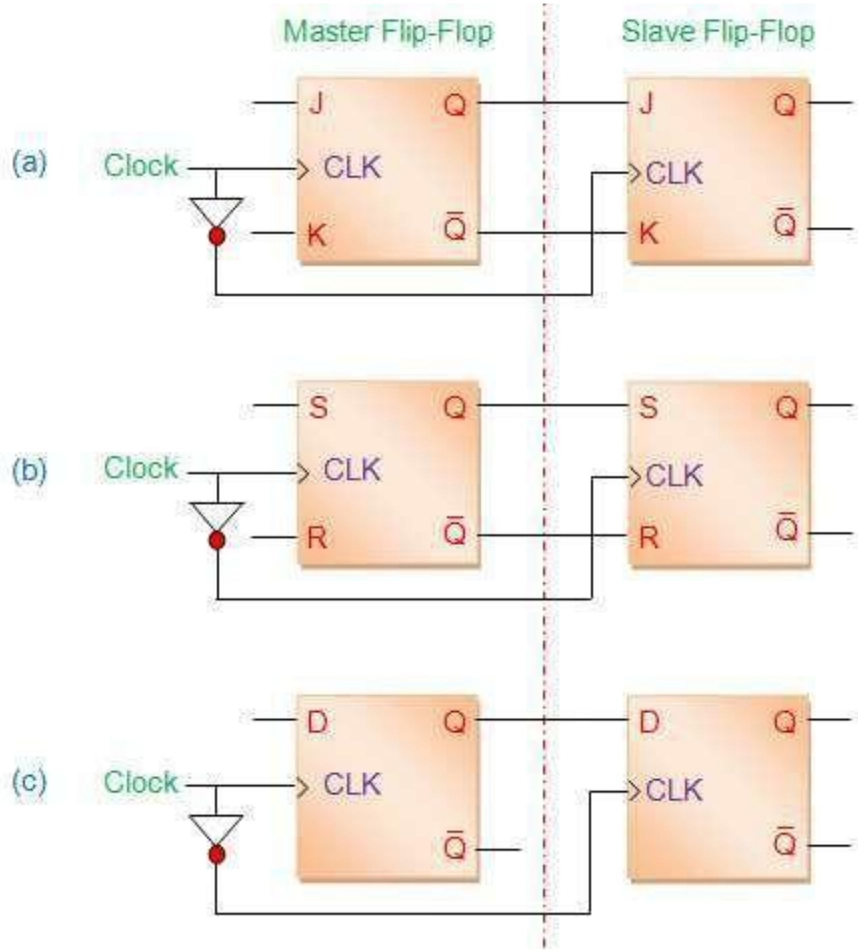


Figure 1 (a) Master-Slave JK flip-flop (b) Master-Slave SR flip-flop (c) Master-Slave D flip-flop

FLIPFLOPS:MASTER SLAVE FLIPFLOP

The truth table corresponding to the working of the flip-flop shown in Figure is given by Table I. Here it is seen that the outputs at the master-part of the flip-flop (data enclosed in red boxes) appear during the positive-edge of the clock (red arrow). However at this instant the slave-outputs remain latched or unchanged. The same data is transferred to the output pins of the master-slave flip-flop (data enclosed in blue boxes) by the slave during the negative edge of the clock pulse (blue arrow). The same principle is further emphasized in the timing diagram of **master-slave flip-flop** shown by Figure 3. Here the green arrows are used to indicate that the slave-output is nothing but the master-output delayed by half-a-clock cycle. Moreover it is to be noted that the working of any other type of master-slave flip-flop is analogous to that of the master slave JK flip-flop explained here.

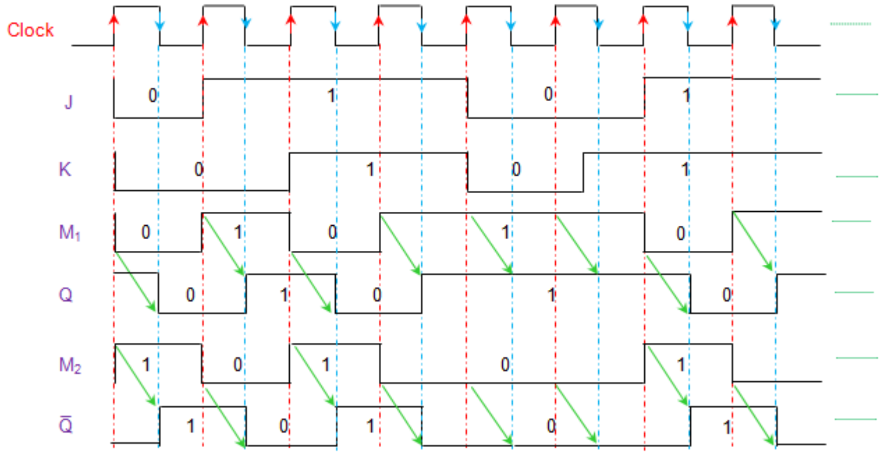
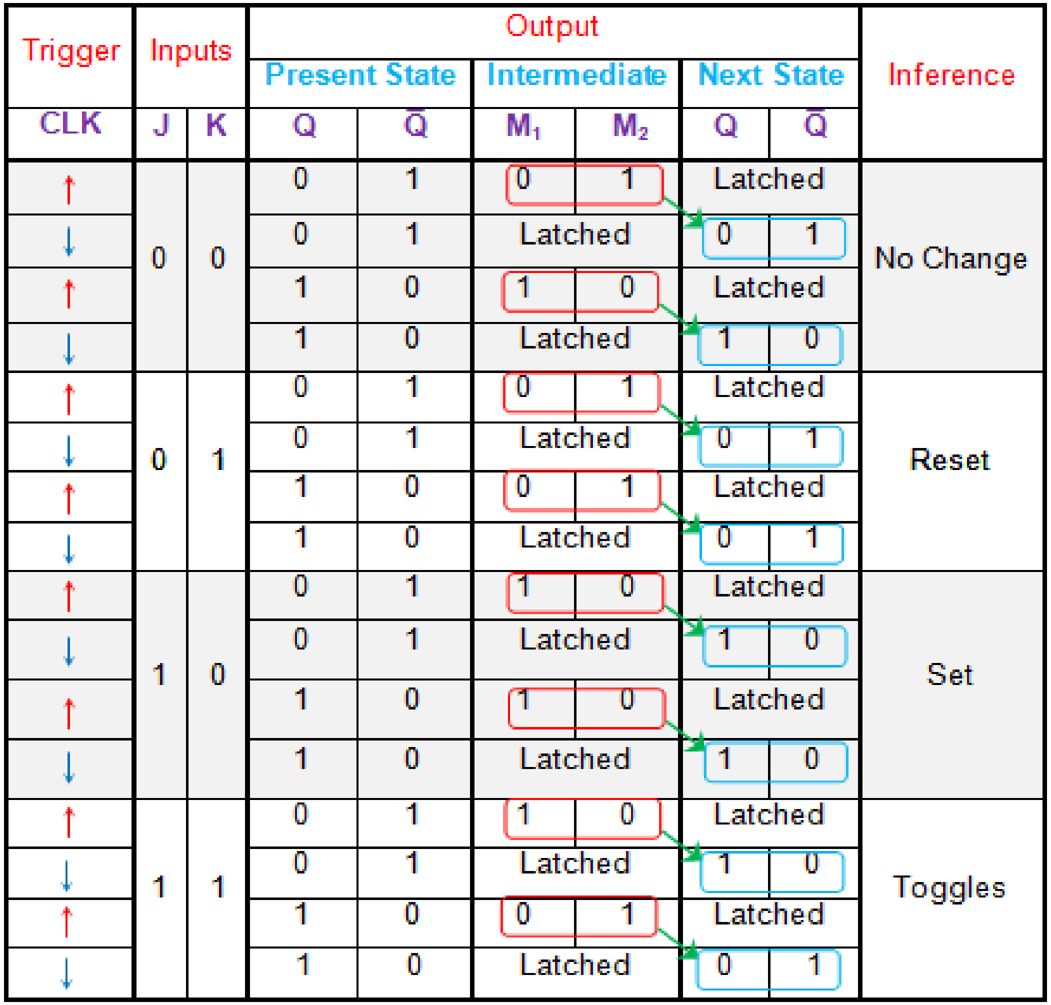


Figure 3 Timing diagram for master-slave JK flip-flop

FLIPFLOPS:MASTER SLAVE FLIPFLOP

Truth Table



Trigger	Inputs		Output						Inference
			Present State		Intermediate		Next State		
CLK	J	K	Q	$\bar{Q}$	M_1	M_2	Q	$\bar{Q}$	
↑	0	0	0	1	0	1	Latched		No Change
↓			0	1	Latched		0	1	
↑			1	0	1	0	Latched		
↓			1	0	Latched		1	0	
↑	0	1	0	1	0	1	Latched		Reset
↓			0	1	Latched		0	1	
↑			1	0	0	1	Latched		
↓			1	0	Latched		0	1	
↑	1	0	0	1	1	0	Latched		Set
↓			0	1	Latched		1	0	
↑			1	0	1	0	Latched		
↓			1	0	Latched		1	0	
↑	1	1	0	1	1	0	Latched		Toggles
↓			0	1	Latched		1	0	
↑			1	0	0	1	Latched		
↓			1	0	Latched		0	1	

Table I Truth table for master-slave JK flip-flop

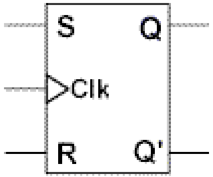
FLIPFLOPS:EXCITATION FUNCTIONS

In electronics design, an **excitation table** shows the minimum inputs that are necessary to generate a particular next state (in other words, to **"excite"** it to the next state) when the current state is known. They are similar to truth tables and state tables, but rearrange the data so that the current state and next state are next to each other on the left-hand side of the table, and the inputs needed to make that state change happen.
All flip-flops can be divided into four basic types: **SR**, **JK**, **D** and **T**. They differ in the number of inputs and in the response invoked by different value of input signals.

The ***characteristic table*** in the third column of Table 1 defines the state of each flip-flop as a function of its inputs and previous state. **Q** refers to the present state and **Q(next)** refers to the next state after the occurrence of the clock pulse. The characteristic table for the RS flip-flop shows that the next state is equal to the present state when both inputs S and R are equal to 0. When R=1, the next clock pulse clears the flip-flop. When S=1, the flip-flop output Q is set to 1. The equation mark (?) for the next state when S and R are both equal to 1 designates an indeterminate next state.
The characteristic table for the JK flip-flop is the same as that of the RS when J and K are replaced by S and R respectively, except for the indeterminate case. When both J and K are equal to 1, the next state is equal to the complement of the present state, that is, Q(next) = Q'.
The next state of the D flip-flop is completely dependent on the input D and independent of the present state.
The next state for the T flip-flop is the same as the present state Q if T=0 and complemented if T=1.

FLIPFLOPS:EXCITATION FUNCTIONS

SR Flip flop



FLIP-FLOP SYMBOL

S	R	$Q_{(next)}$
0	0	Q
0	1	0
1	0	1
1	1	?

CHARACTERISTIC TABLE

$$Q_{(next)} = S + R'Q$$

$$SR = 0$$

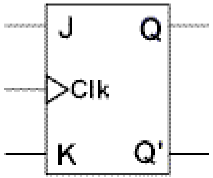
CHARACTERISTIC EQUATION

Q	$Q_{(next)}$	S	R
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

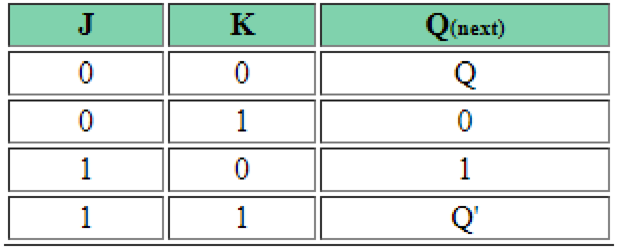
EXCITATION TABLE

FLIPFLOPS:EXCITATION FUNCTIONS

JK Flip flop



FLIP-FLOP SYMBOL

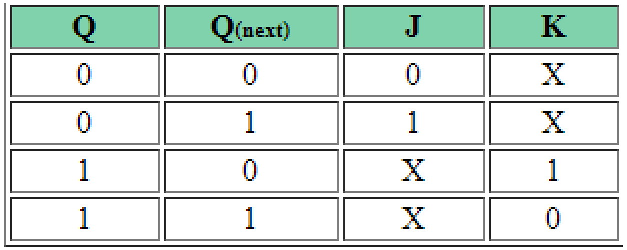


J	K	$Q_{(next)}$
0	0	Q
0	1	0
1	0	1
1	1	Q'

CHARACTERISTIC TABLE

$$Q_{(next)} = JQ' + K'Q$$

CHARACTERISTIC EQUATION

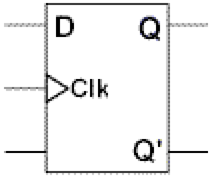


Q	$Q_{(next)}$	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

EXCITATION TABLE

FLIPFLOPS:EXCITATION FUNCTIONS

D Flip flop



FLIP-FLOP SYMBOL

D	Q(next)
0	0
1	1

CHARACTERISTIC TABLE

$$Q_{(next)} = D$$

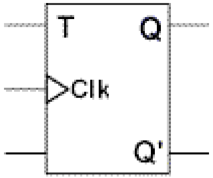
CHARACTERISTIC EQUATION

Q	Q(next)	D
0	0	0
0	1	1
1	0	0
1	1	1

EXCITATION TABLE

FLIPFLOPS:EXCITATION FUNCTIONS

T Flip flop



FLIP-FLOP SYMBOL

T	Q(next)
0	Q
1	Q'

CHARACTERISTIC TABLE

$$Q_{(next)} = TQ' + T'Q$$

CHARACTERISTIC EQUATION

Q	Q(next)	T
0	0	0
0	1	1
1	0	1
1	1	0

EXCITATION TABLE

CONVERTION OF ONE FLIP FLOP TO ANOTHER FLIP FLOP

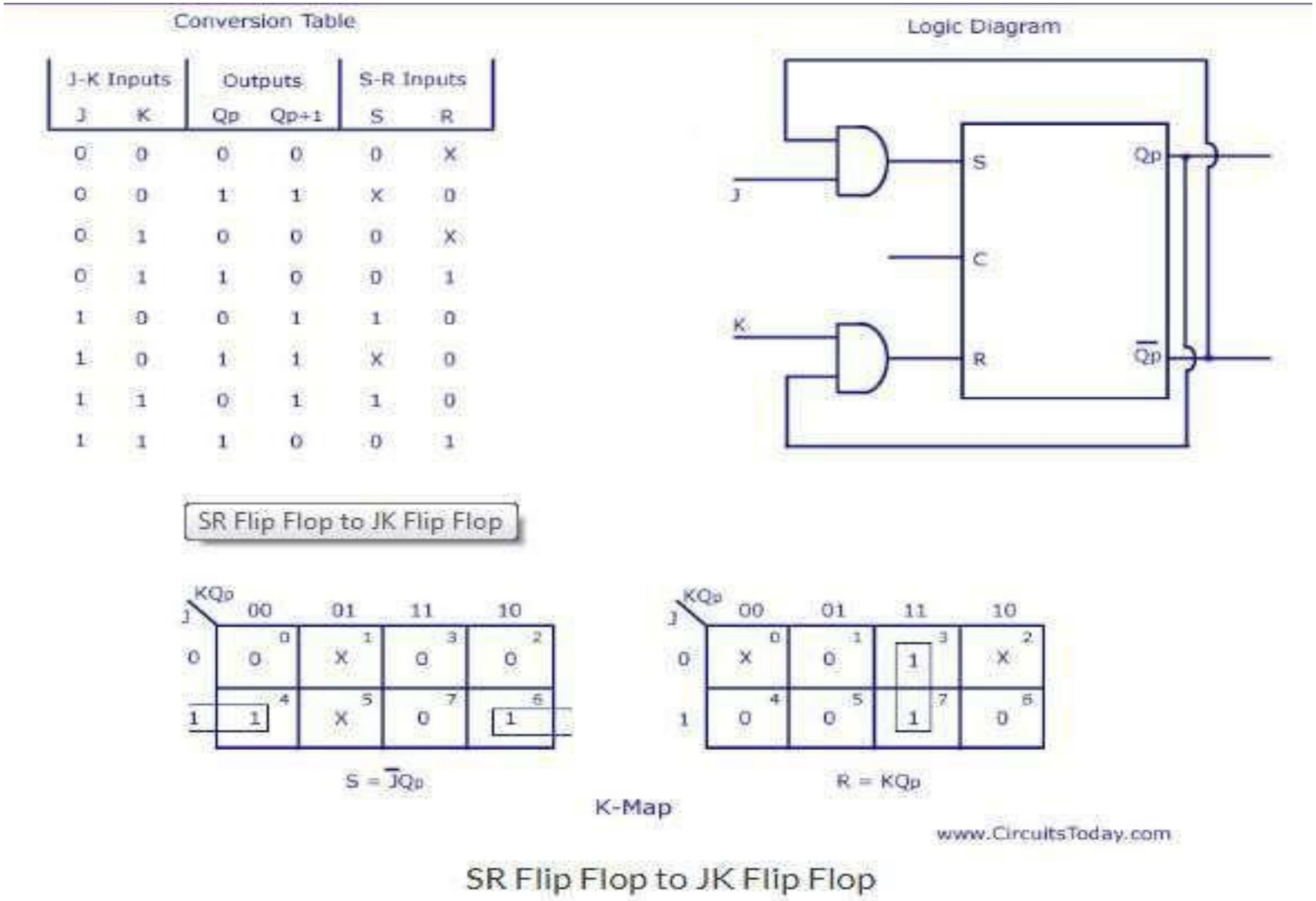
CONVERTION OF SR FLIP FLOP TO JK FLIPFLOP

J and K will be given as external inputs to S and R. As shown in the logic diagram below, S and R will be the outputs of the combinational circuit.

The truth tables for the flip flop conversion are given below. The present state is represented by Qp and Qp+1 is the next state to be obtained when the J and K inputs are applied.

For two inputs J and K, there will be eight possible combinations. For each combination of J, K and Qp, the corresponding Qp+1 states are found. Qp+1 simply suggests the future values to be obtained by the JK flip flop after the value of Qp. The table is then completed by writing the values of S and R required to get each Qp+1 from the corresponding Qp. That is, the values of S and R that are required to change the state of the flip flop from Qp to Qp+1 are written.

CONVERTION OF ONE FLIP FLOP TO ANOTHER FLIP FLOP



Conversion Table

J-K Inputs		Outputs		S-R Inputs	
J	K	Qp	Qp+1	S	R
0	0	0	0	0	X
0	0	1	1	X	0
0	1	0	0	0	X
0	1	1	0	0	1
1	0	0	1	1	0
1	0	1	1	X	0
1	1	0	1	1	0
1	1	1	0	0	1

Logic Diagram

SR Flip Flop to JK Flip Flop

J \ KQp	00	01	11	10
0	0^{0}	X^{1}	0^{3}	0^{2}
1	1^{4}	X^{5}	0^{7}	1^{6}

$S = J\overline{Q}p$

J \ KQp	00	01	11	10
0	X^{0}	0^{1}	1^{3}	X^{2}
1	0^{4}	0^{5}	1^{7}	0^{6}

$R = KQp$

K-Map

www.CircuitsToday.com

SR Flip Flop to JK Flip Flop

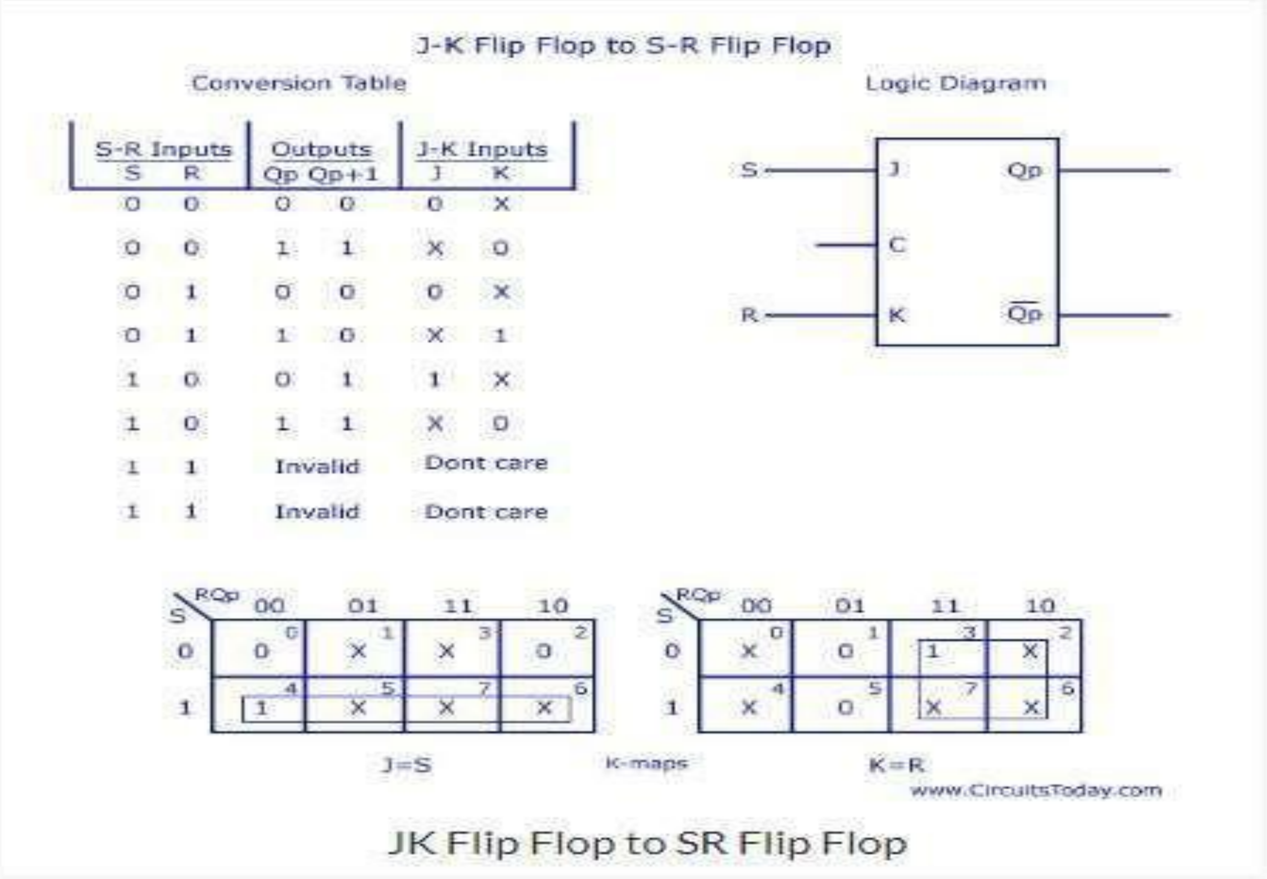
CONVERTION OF ONE FLIP FLOP TO ANOTHER FLIP FLOP

CONVERTION OF JK FLIP FLOP TO SR FLIP FLOP

This will be the reverse process of the above explained conversion. S and R will be the external inputs to J and K. As shown in the logic diagram below, J and K will be the outputs of the combinational circuit. Thus, the values of J and K have to be obtained in terms of S, R and Qp. The logic diagram is shown below.

A conversion table is to be written using S, R, Qp, Qp+1, J and K. For two inputs, S and R, eight combinations are made. For each combination, the corresponding Qp+1 outputs are found ut. The outputs for the combinations of S=1 and R=1 are not permitted for an SR flip flop. Thus the outputs are considered invalid and the J and K values are taken as “don’t cares”.

CONVERTION OF ONE FLIP FLOP TO ANOTHER FLIP FLOP



J-K Flip Flop to S-R Flip Flop

Conversion Table

S-R Inputs		Outputs		J-K Inputs	
S	R	Qp	Qp+1	J	K
0	0	0	0	0	X
0	0	1	1	X	0
0	1	0	0	0	X
0	1	1	0	X	1
1	0	0	1	1	X
1	0	1	1	X	0
1	1	Invalid		Dont care	
1	1	Invalid		Dont care	

Logic Diagram

S \ RQp	00	01	11	10
0	^{0} 0	^{1} X	^{3} X	^{2} 0
1	^{4} 1	^{5} X	^{7} X	^{6} X

J=S

K-maps

S \ RQp	00	01	11	10
0	^{0} X	^{1} 0	^{3} 1	^{2} X
1	^{4} X	^{5} 0	^{7} X	^{6} X

K=R

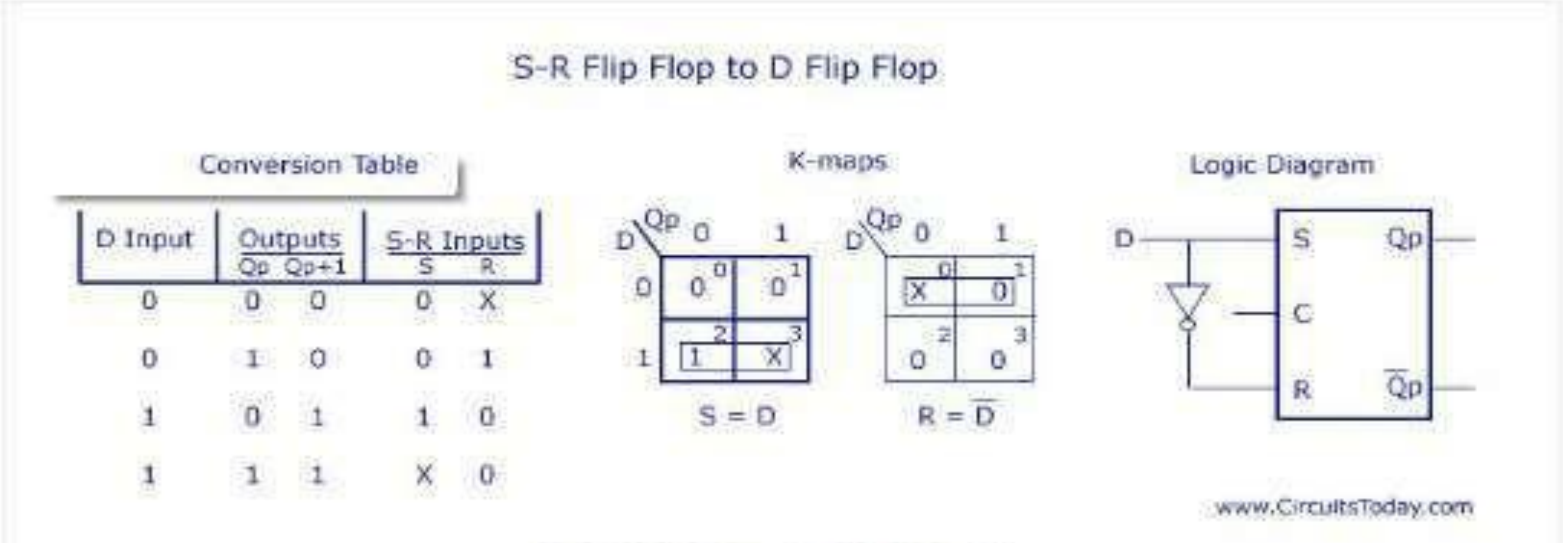
www.CircuitsToday.com

JK Flip Flop to SR Flip Flop

CONVERTION OF ONE FLIP FLOP TO ANOTHER FLIP FLOP

CONVERTION OF SR FLIP FLOP TO D FLIPFLOP

As shown in the figure, S and R are the actual inputs of the flip flop and D is the external input of the flip flop. The four combinations, the logic diagram, conversion table, and the K-map for S and R in terms of D and Qp are shown below.



S-R Flip Flop to D Flip Flop

Conversion Table

D Input	Outputs		S-R Inputs	
	Qp	Qp+1	S	R
0	0	0	0	X
0	1	0	0	1
1	0	1	1	0
1	1	1	X	0

K-maps

D \ Qp	0	1
0	0	0
1	1	X

S = D

D \ Qp	0	1
0	X	0
1	0	0

R = $\overline{D}$

Logic Diagram

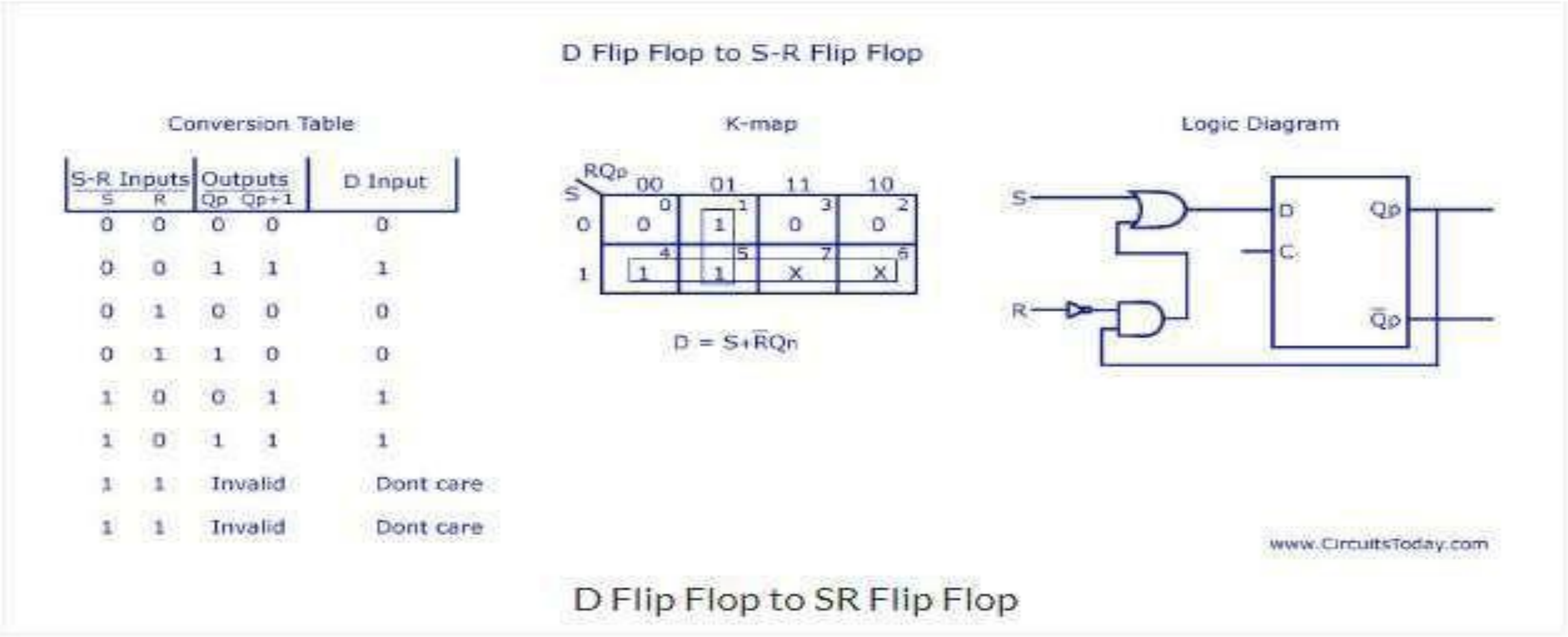
www.CircuitsToday.com

SR Flip Flop to D Flip Flop

CONVERTION OF ONE FLIP FLOP TO ANOTHER FLIP FLOP

CONVERTION OF D FLIP FLOP TO SR FLIPFLOP

D is the actual input of the flip flop and S and R are the external inputs. Eight possible combinations are achieved from the external inputs S, R and Qp. But, since the combination of S=1 and R=1 are invalid, the values of Qp+1 and D are considered as “don’t cares”. The logic diagram showing the conversion from D to SR, and the K-map for D in terms of S, R and Qp are shown below.



D Flip Flop to S-R Flip Flop

Conversion Table

S-R Inputs		Outputs		D Input
S	R	Qp	Qp+1	
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	0	0
1	0	0	1	1
1	0	1	1	1
1	1	Invalid		Dont care
1	1	Invalid		Dont care

K-map

S \ RQp	00	01	11	10
0	0	1	0	0
1	1	1	X	X

$D = S+\overline{R}Qp$

Logic Diagram

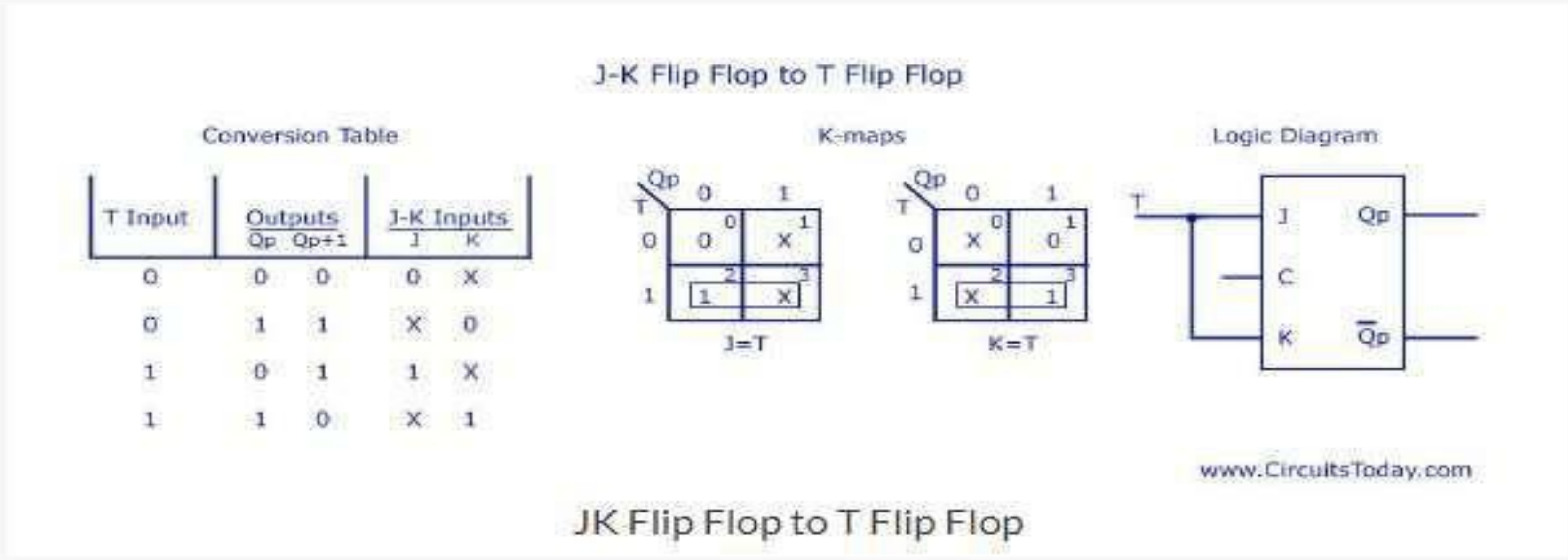
www.CircuitsToday.com

D Flip Flop to SR Flip Flop

CONVERTION OF ONE FLIP FLOP TO ANOTHER FLIP FLOP

CONVERTION OF JK FLIP FLOP TO T FLIP FLOP

J and K are the actual inputs of the flip flop and T is taken as the external input for conversion. Four combinations are produced with T and Qp. J and K are expressed in terms of T and Qp. The conversion table, K-maps, and the logic diagram are given below.



J-K Flip Flop to T Flip Flop

Conversion Table

T Input	Outputs		J-K Inputs	
	Qp	Qp+1	J	K
0	0	0	0	X
0	1	1	X	0
1	0	1	1	X
1	1	0	X	1

K-maps

T \ Qp	0	1
0	0	X
1	1	X

J=T

T \ Qp	0	1
0	X	0
1	X	1

K=T

Logic Diagram

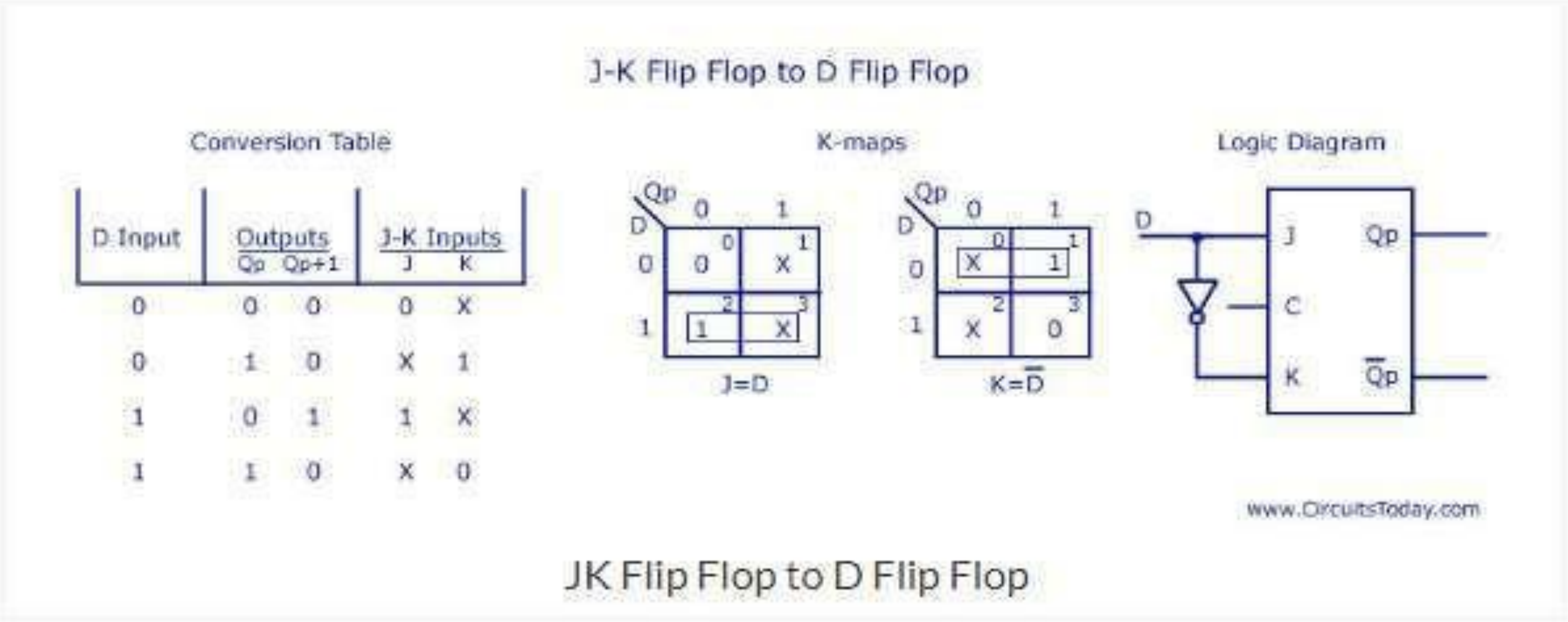
www.CircuitsToday.com

JK Flip Flop to T Flip Flop

CONVERTION OF ONE FLIP FLOP TO ANOTHER FLIP FLOP

CONVERTION OF JK FLIP FLOP TO D FLIPFLOP

D is the external input and J and K are the actual inputs of the flip flop. D and Qp make four combinations. J and K are expressed in terms of D and Qp. The four combination conversion table, the K-maps for J and K in terms of D and Qp, and the logic diagram showing the conversion from JK to D are given below.



J-K Flip Flop to D Flip Flop

Conversion Table

D Input	Outputs		J-K Inputs	
	Qp	Qp+1	J	K
0	0	0	0	X
0	1	0	X	1
1	0	1	1	X
1	1	0	X	0

K-maps

D \ Qp	0	1
0	0	X
1	1	X

J=D

D \ Qp	0	1
0	X	1
1	X	0

K=$\overline{D}$

Logic Diagram

www.CircuitsToday.com

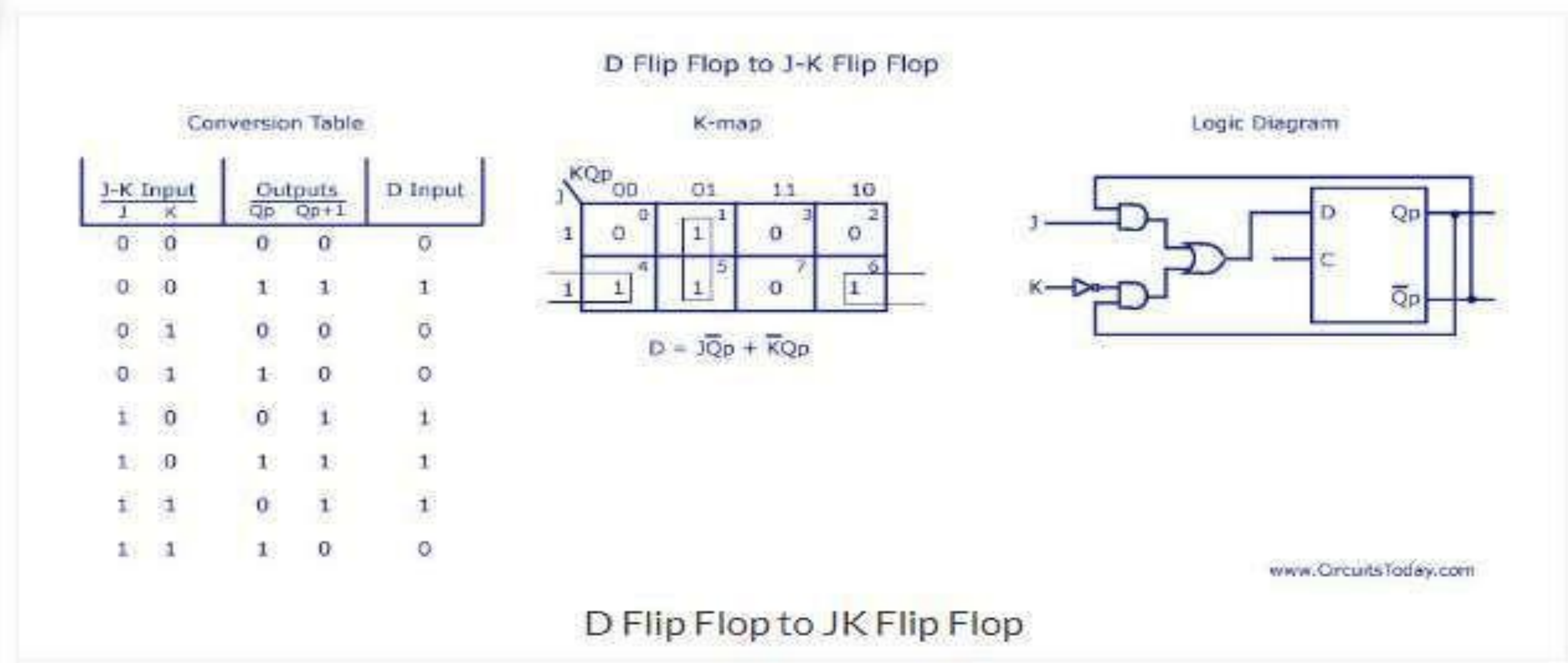
JK Flip Flop to D Flip Flop

CONVERTION OF ONE FLIP FLOP TO ANOTHER FLIP FLOP

CONVERTION OF D FLIP FLOP TO JK FLIPFLOP

In this conversion, D is the actual input to the flip flop and J and K are the external inputs. J, K and Qp make eight possible combinations, as shown in the conversion table below. D is expressed in terms of J, K and Qp.

The conversion table, the K-map for D in terms of J, K and Qp and the logic diagram showing the conversion from D to JK are given in the figure below.



D Flip Flop to J-K Flip Flop

Conversion Table

J-K Input		Outputs		D Input
J	K	Qp	Qp+1	
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	0	0
1	0	0	1	1
1	0	1	1	1
1	1	0	1	1
1	1	1	0	0

K-map

J \ KQp	00	01	11	10
1	0^{0}	1^{1}	0^{3}	0^{2}
1	1^{4}	1^{5}	0^{7}	1^{6}

$$D = J\overline{Q}p + \overline{K}Qp$$

Logic Diagram

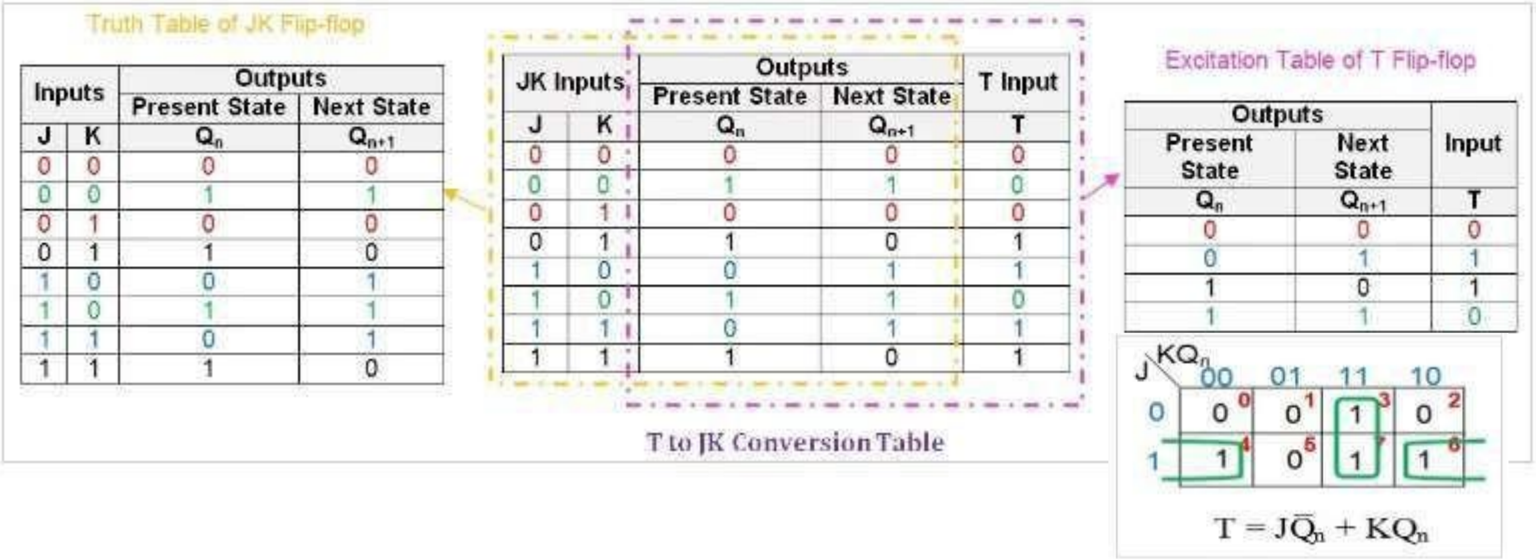
www.CircuitsToday.com

D Flip Flop to JK Flip Flop

CONVERTION OF ONE FLIP FLOP TO ANOTHER FLIP FLOP

CONVERTION OF T FLIP FLOP TO JK FLIP FLOP

We begin with the T-to-JK conversion table (see Figure 5), which combines the information in the JK flip-flop's truth table and the T flip-flop's excitation table.



Truth Table of JK Flip-flop

Inputs		Outputs	
		Present State	Next State
J	K	Q_n	Q_{n+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

JK Inputs		Outputs		T Input
		Present State	Next State	
J	K	Q_n	Q_{n+1}	T
0	0	0	0	0
0	0	1	1	0
0	1	0	0	0
0	1	1	0	1
1	0	0	1	1
1	0	1	1	0
1	1	0	1	1
1	1	1	0	1

T to JK Conversion Table

Excitation Table of T Flip-flop

Outputs		Input
Present State	Next State	
Q_n	Q_{n+1}	T
0	0	0
0	1	1
1	0	1
1	1	0

J \ KQ_n	00	01	11	10
0	0^{0}	0^{1}	1^{3}	0^{2}
1	1^{4}	0^{5}	1^{7}	1^{6}

$$T = J\overline{Q}_n + KQ_n$$

Next, we need to obtain the simplified Boolean expression for the T input in terms of J, K, and Q_n. The expression for the T input as $J\overline{Q}_n + KQ_n$. This means that to convert the T flip-flop into a JK flip-flop, the T input is driven by the output of a two-input OR gate which has as inputs

J ANDed with the negation of the present-state Q_n, i.e., $\overline{Q}_n$

K ANDed with the present-state, Q_n

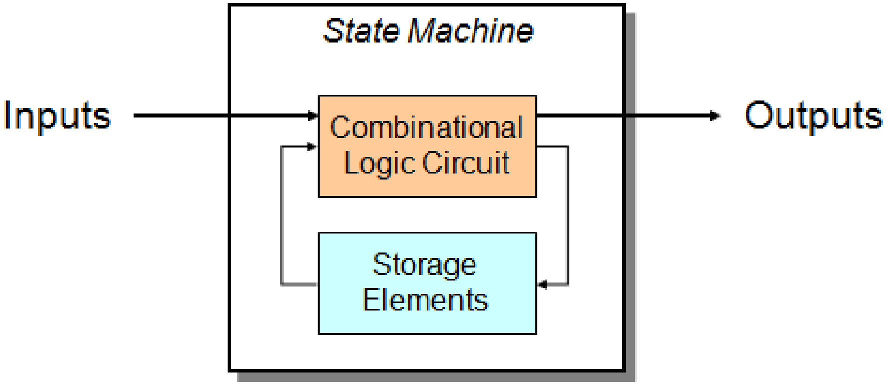
STATE MACHINES

State Machine

Another type of sequential circuit

Combines combinational logic with storage

"Remembers" state, and changes output (and state) based on inputs and current state



State

The state of a system is a snapshot of all the relevant elements of the system at the moment the snapshot is taken.

Examples:

The state of a basketball game can be represented by the scoreboard.

Number of points, time remaining, possession, etc.

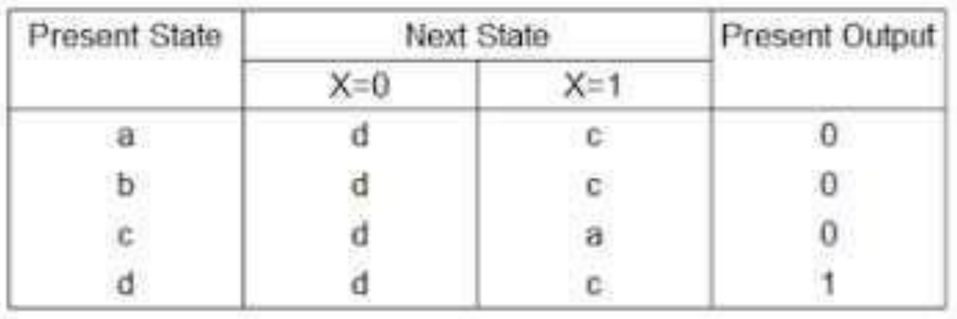
The state of a tic-tac-toe game can be represented by the placement of X's and O's on the board.

STATE TABLES AND STATE DIAGRAMS

In this model the effect of all previous inputs on the outputs is represented by a state of the circuit. Thus, the output of the circuit at any time depends upon its current state and the input. These also determine the next state of the circuit. The relationship that exists among the inputs, outputs, present states and next states can be specified by either the **state table** or the **state diagram**.

State Table

The state table representation of a sequential circuit consists of three sections labeled *present state*, *next state* and *output*. The present state designates the state of flip-flops before the occurrence of a clock pulse. The next state shows the states of flip-flops after the clock pulse, and the output section lists the value of the output variables during the present state.



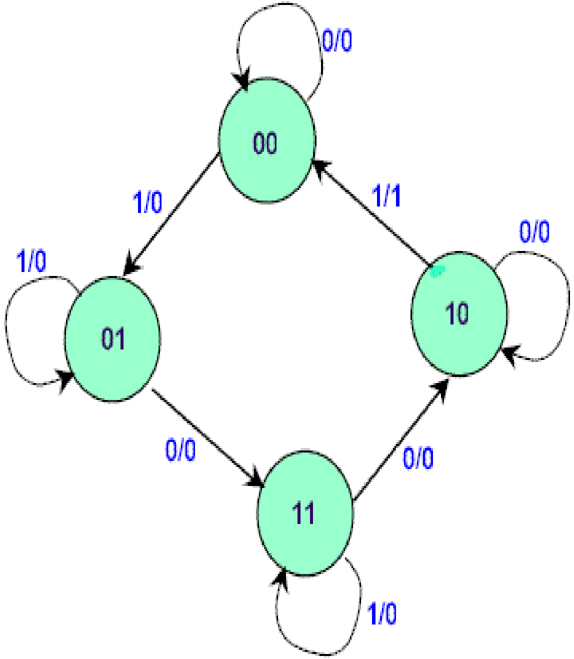
Present State	Next State		Present Output
	X=0	X=1	
a	d	c	0
b	d	c	0
c	d	a	0
d	d	c	1

STATE MACHINES

State Diagram

In addition to graphical symbols, tables or equations, flip-flops can also be represented graphically by a state diagram. In this diagram, a state is represented by a circle, and the transition between states is indicated by directed lines (or arcs) connecting the circles.

The binary number inside each circle identifies the state the circle represents. The directed lines are labelled with two binary numbers separated by a slash (/). The input value that causes the state transition is labelled first. The number after the slash symbol / gives the value of the output. For example, the directed line from state 00 to 01 is labelled 1/0, meaning that, if the sequential circuit is in a present state and the input is 1, then the next state is 01 and the output is 0. If it is in a present state 00 and the input is 0, it will remain in that state. A directed line connecting a circle with itself indicates that no change of state occurs. The state diagram provides exactly the same information as the state table and is obtained directly from the state table.

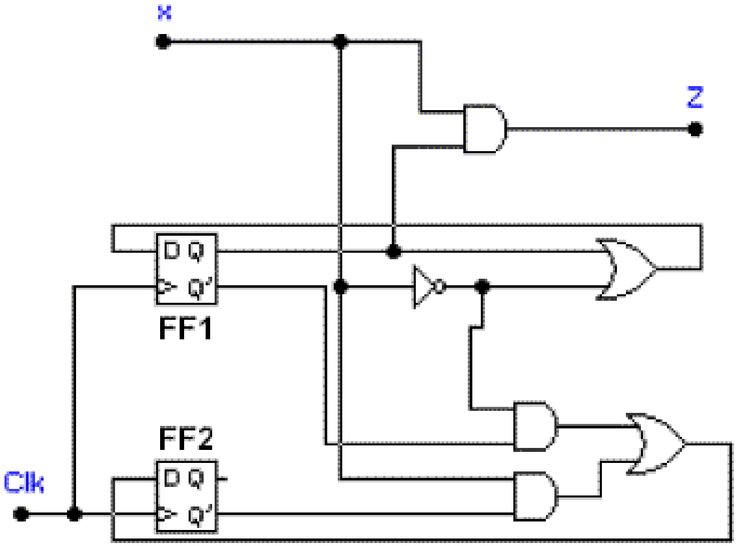


State Diagram

STATE MACHINES

Example:

Consider a sequential circuit



The behavior of the circuit is determined by the following Boolean expressions:

Z = x*Q1

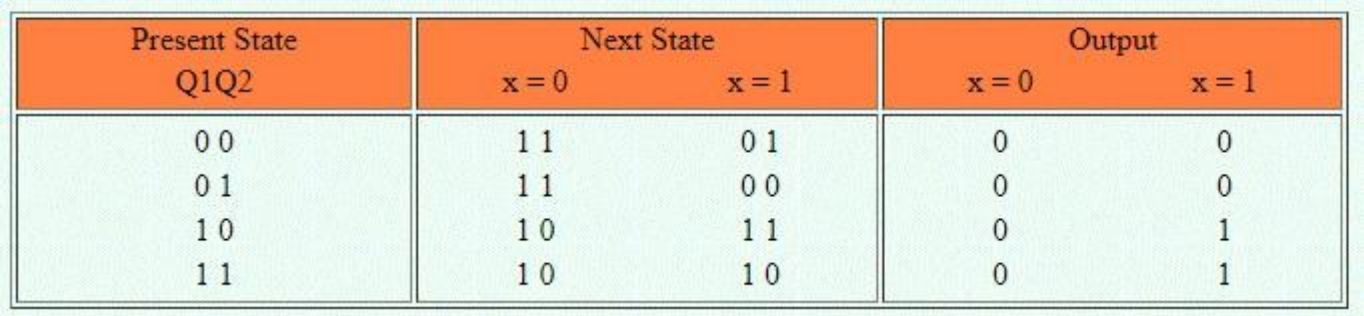
D1 = x' + Q1

D2 = x*Q2' + x'*Q1'

These equations can be used to form the state table. Suppose the present state (i.e. Q1Q2) = 00 and input x = 0. Under these conditions, we get Z = 0, D1 = 1, and D2 = 1. Thus the next state of the circuit D1D2 = 11, and this will be the present state after the clock pulse has been applied. The output of the circuit corresponding to the present state Q1Q2 = 00 and x = 1 is Z = 0. This data is entered into the state table as shown in Table 2.

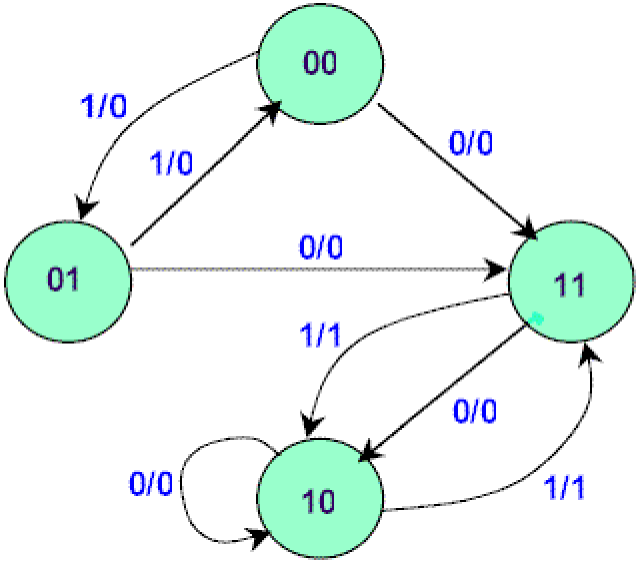
STATE MACHINES

State table for the sequential circuit



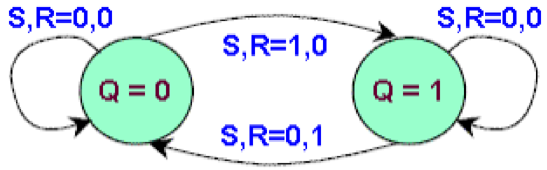
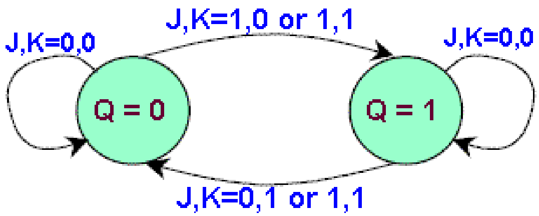
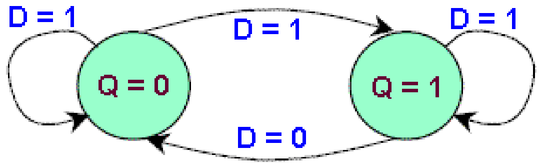
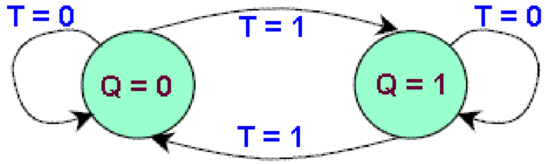
Present State	Next State		Output	
Q1Q2	x = 0	x = 1	x = 0	x = 1
0 0	1 1	0 1	0	0
0 1	1 1	0 0	0	0
1 0	1 0	1 1	0	1
1 1	1 0	1 0	0	1

The state diagram for the sequential circuit



STATE MACHINES

state diagrams of the four types of flip-flops

NAME	STATE DIAGRAM
SR	
JK	 J,K=1,0 or 1,1 J,K=0,0 J,K=0,0 Q = 0 Q = 1 J,K=0,1 or 1,1
D	
T	

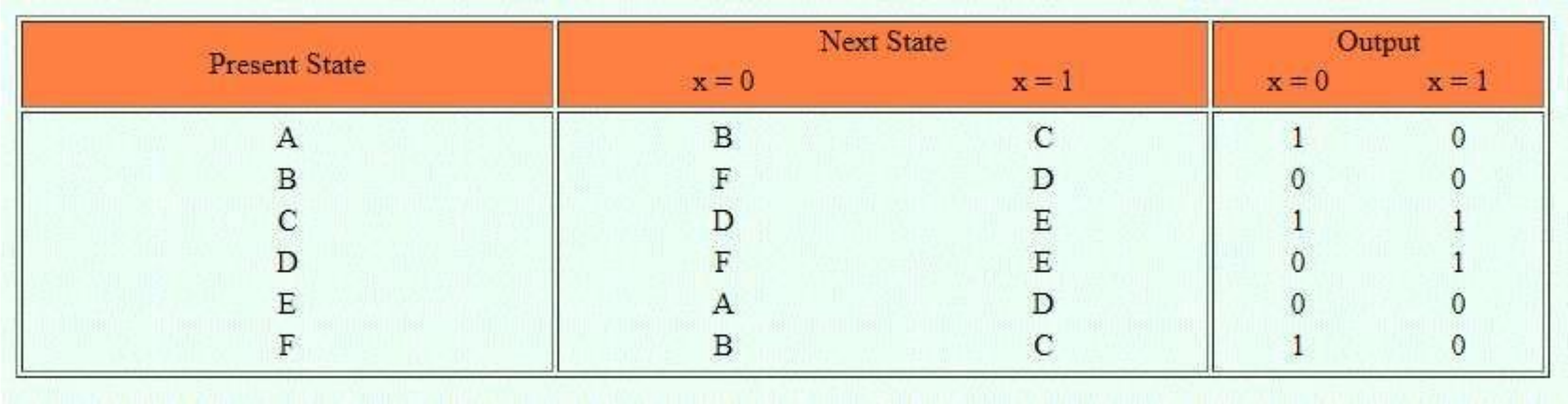
STATE REDUCTION

State Reduction

Any design process must consider the problem of minimising the cost of the final circuit. The two most obvious cost reductions are reductions in the number of flip-flops and the number of gates.

The number of states in a sequential circuit is closely related to the complexity of the resulting circuit. It is therefore desirable to know when two or more states are equivalent in all aspects. The process of eliminating the equivalent or redundant states from a state table/diagram is known as **state reduction**.

Example: Let us consider the state table of a sequentialcircuit

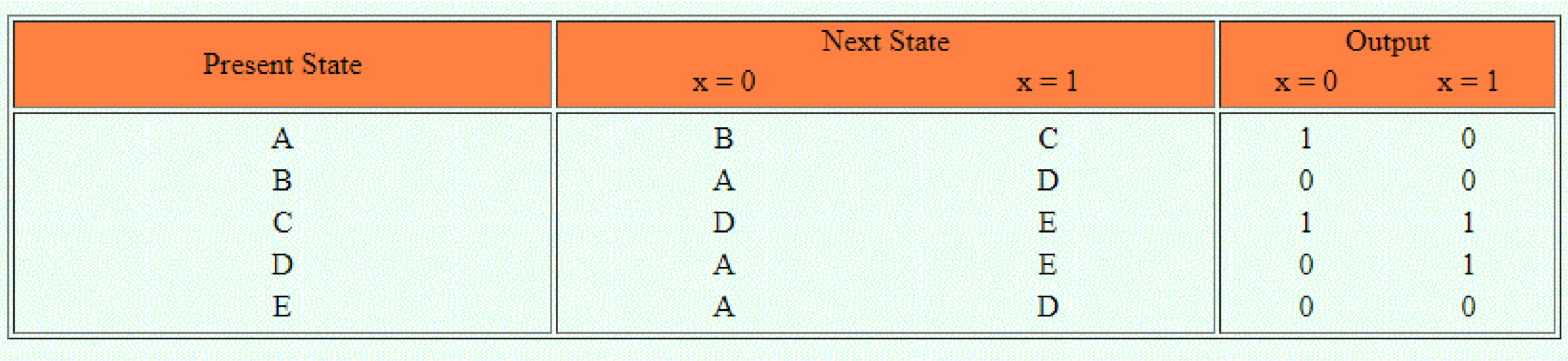


Present State	Next State		Output	
	x = 0	x = 1	x = 0	x = 1
A	B	C	1	0
B	F	D	0	0
C	D	E	1	1
D	F	E	0	1
E	A	D	0	0
F	B	C	1	0

State table

STATE REDUCTION

It can be seen from the table that the present state A and F both have the same next states, B (when x=0) and C (when x=1). They also produce the same output 1 (when x=0) and 0 (when x=1). Therefore states A and F are equivalent. Thus one of the states, A or F can be removed from the state table. For example, if we remove row F from the table and replace all F's by A's in the columns, the state table is modified

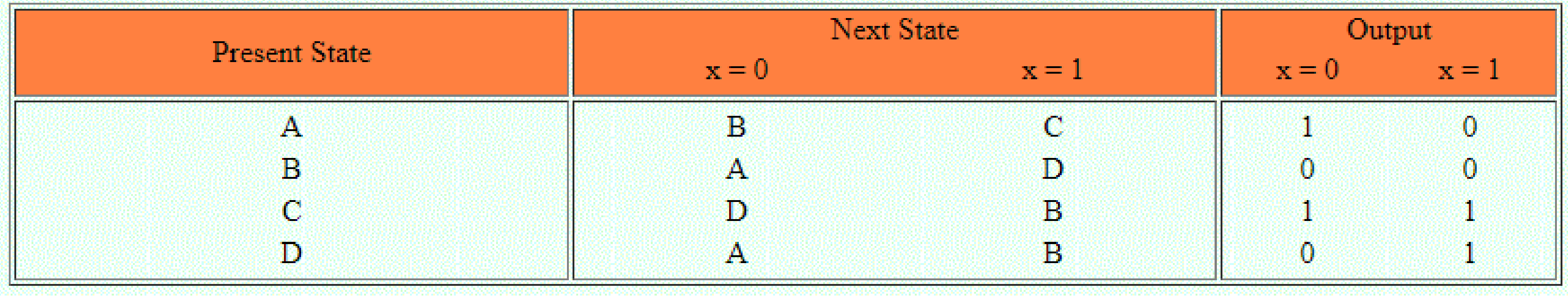


Present State	Next State		Output	
	x = 0	x = 1	x = 0	x = 1
A	B	C	1	0
B	A	D	0	0
C	D	E	1	1
D	A	E	0	1
E	A	D	0	0

State F removed

It is apparent that states B and E are equivalent. Removing E and replacing E's by B's results in the reduce table

STATE REDUCTION



Present State	Next State		Output	
	x = 0	x = 1	x = 0	x = 1
A	B	C	1	0
B	A	D	0	0
C	D	B	1	1
D	A	B	0	1

Reduced state table

The removal of equivalent states has reduced the number of states in the circuit from six to four. Two states are considered to be **equivalent** if and only if for every input sequence the circuit produces the same output sequence irrespective of which one of the two states is the starting state.

STATE ASSIGNMENT

STATE ASSIGNMENT

Each circuit state given in a state table has to be assigned a unique value, which represents combinations of flip – flop output states.

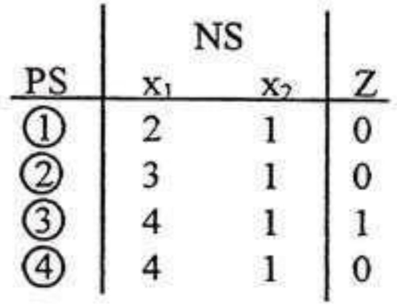
- A circuit having 2 internal states requires one flip – flop in its implementation
- A circuit having 3 or 4 internal states requires two flip – flops in its implementation
- A circuit having 5→ 8 internal states requires three flip – flops in its implementation etc.

It should be noted that although assignments are arbitrary, one assignment might be more economical than another.

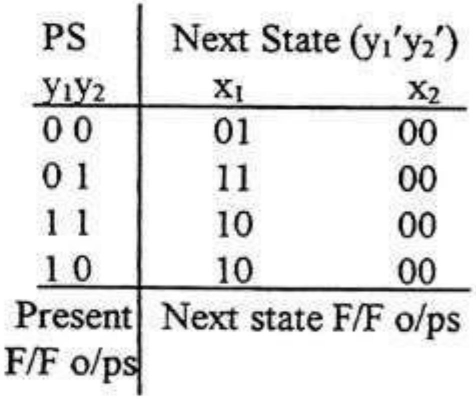
Consider the state table shown below for a circuit having two input pulses $\mathbf{x}_1$, $\mathbf{x}_2$ and a level output **Z**.

Since the circuit has four internal states then two flip-flops are required. Let the two flip-flop outputs be represented by variables $\mathbf{y}_1$ and $\mathbf{y}_2$, which can have combinations of values $\mathbf{y}_1\mathbf{y}_2$ = 00, 01, 11, 10. The state table can then be translated into a state table with secondary assignments as shown. Note that this is just one of many possible assignments (in fact there are 24)

STATE ASSIGNMENT



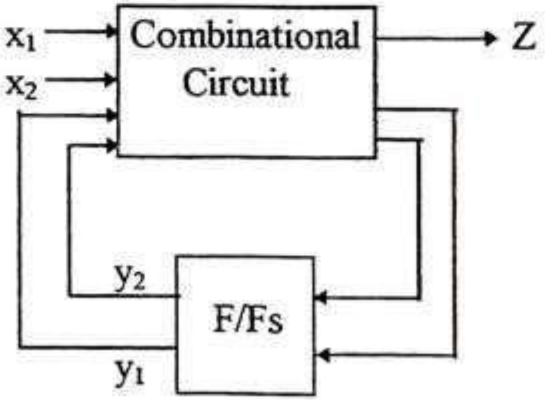
PS	NS x₁	NS x₂	Z
①	2	1	0
②	3	1	0
③	4	1	1
④	4	1	0



PS y_1y_2	Next State ($y_1'y_2'$) x_1	x_2
0 0	01	00
0 1	11	00
1 1	10	00
1 0	10	00
Present F/F o/ps	Next state F/F o/ps	

Example of state assignment

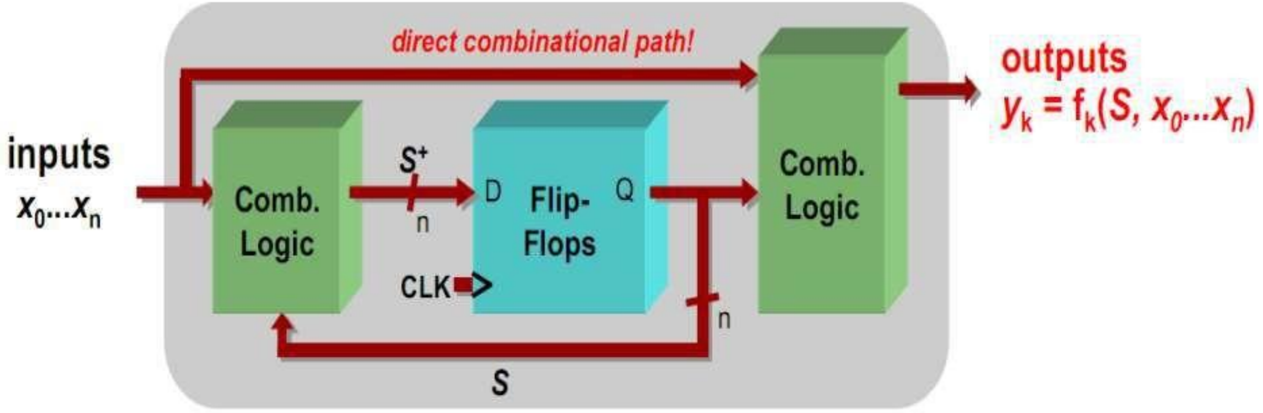
With **y_1y_2** =0 (i.e. in state **1**), if **x_1** is applied then **y_1y_2**must change to 01 (i.e. state **2). That is,** the flip/flop generating **y_1** must not disturbed, but the **y_2**generating flip-flop requires an input such that the circuit settles in state **2**, (for example a SET input if using SR flip-flops).

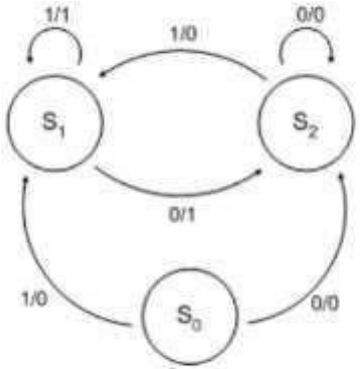


MEALY AND MOORE STATE MACHINES

Mealy state machine

In the theory of computation, a Mealy machine is a finite state transducer that generates an output based on its current state and input. This means that the state diagram will include both an input and output signal for each transition edge. In contrast, the output of a Moore finite state machine depends only on the machine's current state; transitions are not directly dependent upon input. The use of a Mealy FSM leads often to a reduction of the number of states. However, for each Mealy machine there is an equivalent Moore machine.



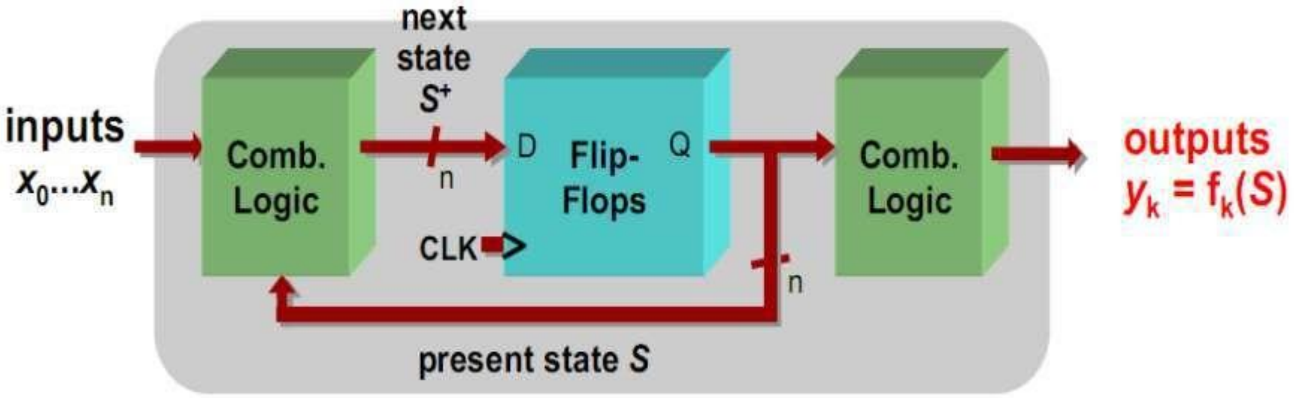


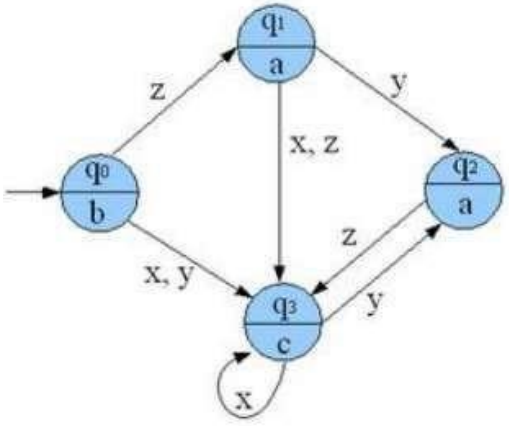
$$\text{Next state} = F(\text{current state, input})$$
$$\text{Output} = G(\text{current state, input})$$

MEALY AND MOORE STATE MACHINES

Moore state machine

In the theory of computation, a Moore machine is a finite state transducer where the outputs are determined by the current state alone (and do not depend directly on the input). The state diagram for a Moore machine will include an output signal for each state. Compare with a Mealy machine, which maps transitions in the machine to outputs. The advantage of the Moore model is a simplification of the behavior.



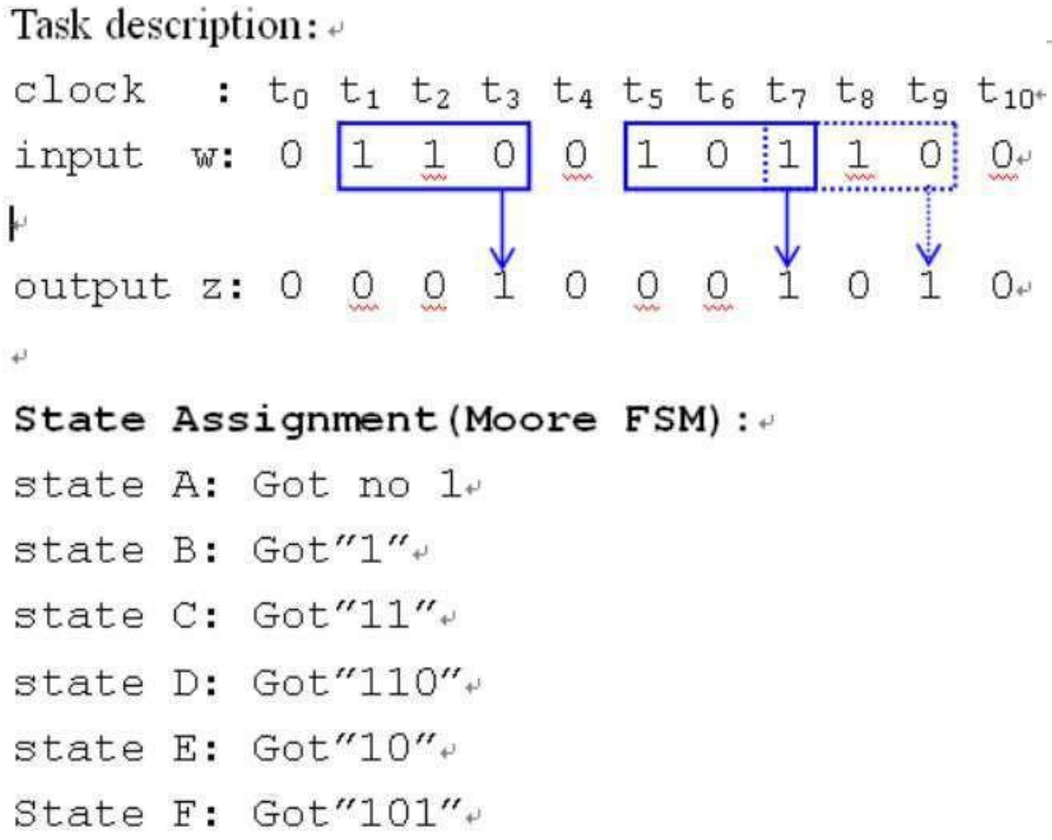


Output = G(current state)

MEALY AND MOORE STATE MACHINES

Examples for Mealy and Moore machines

Derive a minimal state table for a single-input and single-output Moore-type FSM that produces an output of 1 if in the input sequence it detects either 110 or 101 patterns. Overlapping sequences should be detected. (Show the detailed steps of your solution.)



Task description:

clock	:	t_0	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
input	w:	0	1	1	0	0	1	0	1	1	0	0
output	z:	0	0	0	1	0	0	0	1	0	1	0

State Assignment (Moore FSM):

state A: Got no 1

state B: Got"1"

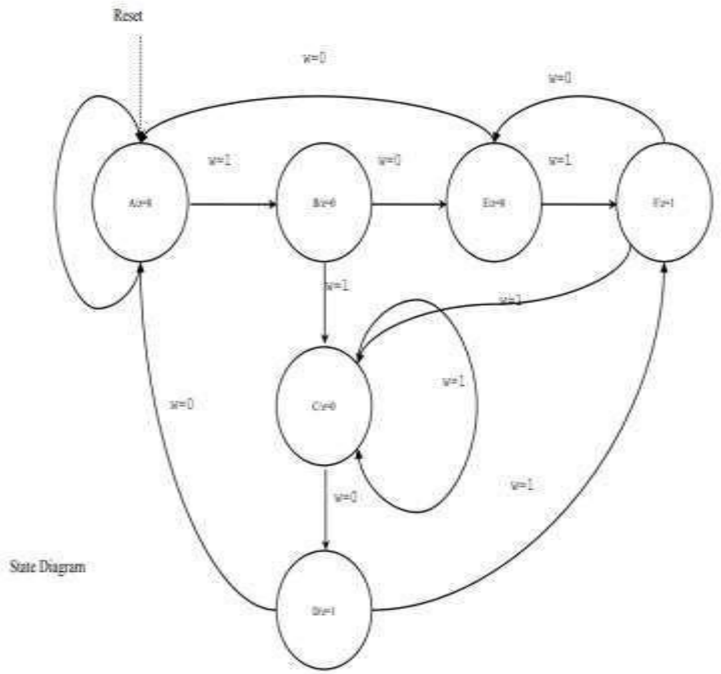
state C: Got"11"

state D: Got"110"

state E: Got"10"

State F: Got"101"

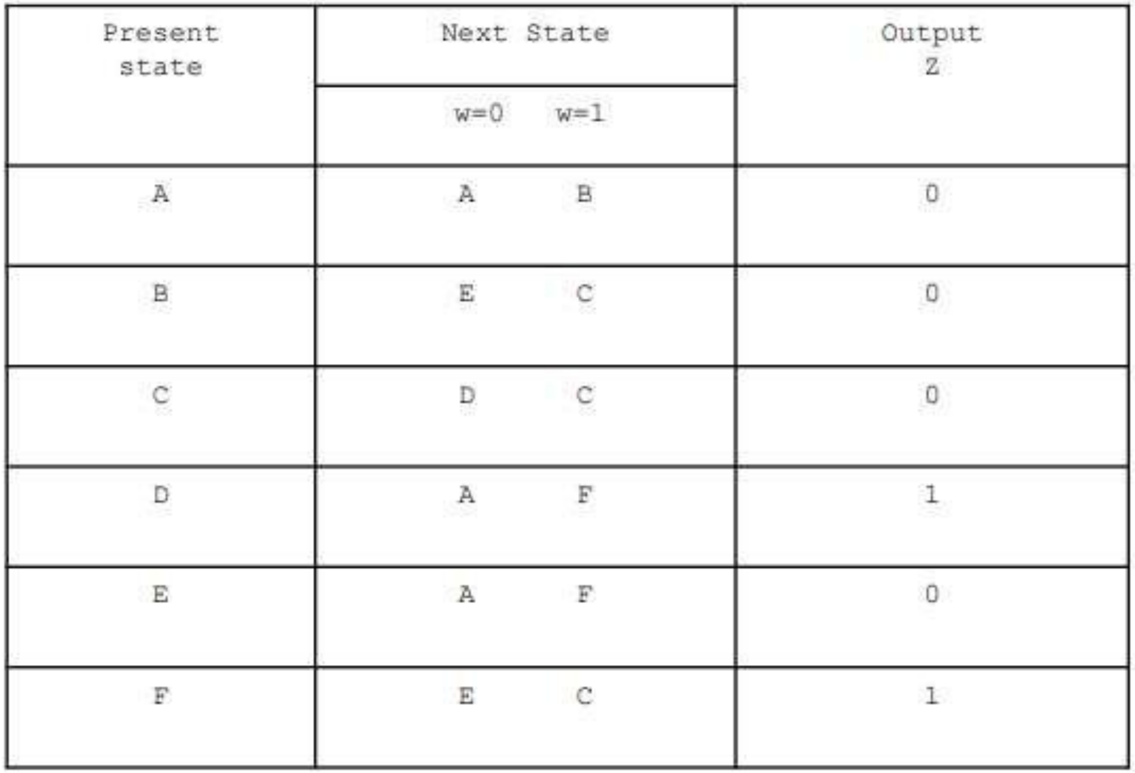
State Diagram



State Diagram

MEALY AND MOORE STATE MACHINES

Sate Table (Moore FSM)



Present state	Next State		Output Z
	w=0	w=1	
A	A	B	0
B	E	C	0
C	D	C	0
D	A	F	1
E	A	F	0
F	E	C	1

6 states need 3 flip-flops

MEALY AND MOORE STATE MACHINES

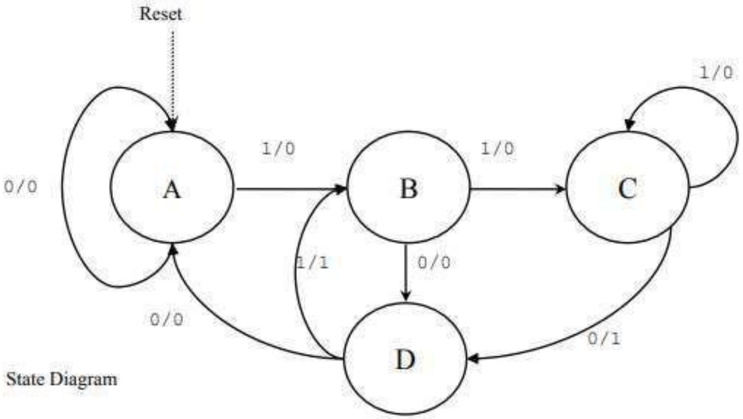
State Assignment (Mealy FSM):

state A: Got no 1

state B: Got”1”

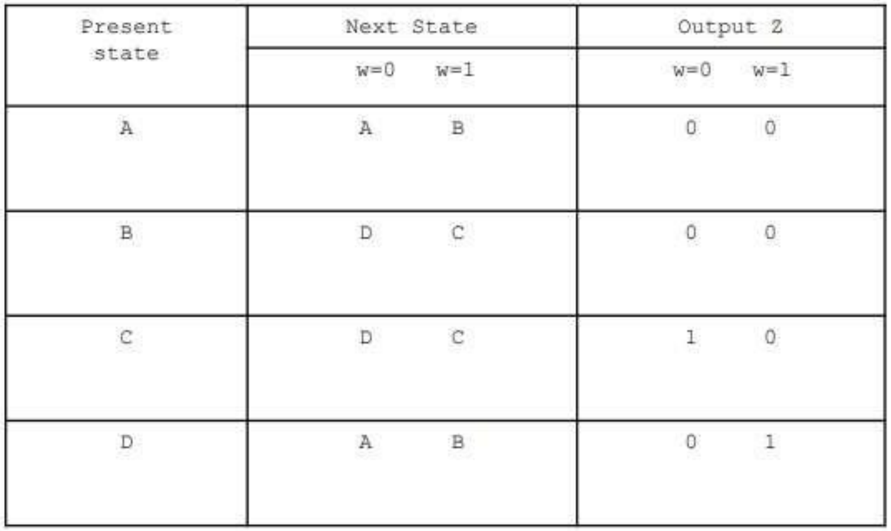
state C: Got”11”

state D: Got”10”



State Diagram

Sate Table (Mealy FSM)



Present state	Next State		Output 2	
	w=0	w=1	w=0	w=1
A	A	B	0	0
B	D	C	0	0
C	D	C	1	0
D	A	B	0	1

4 states need 2 flip-flops

MEALY AND MOORE MACHINES

Sequential Logic Implementation

- Models for representing sequential circuits
  - Abstraction of sequential elements
  - Finite state machines and their state diagrams
  - Inputs/outputs
  - Mealy, Moore, and synchronous Mealy machines
- Finite state machine design procedure
  - Verilog specification
  - Deriving state diagram
  - Deriving state transition table
  - Determining next state and output functions
  - Implementing combinational logic

MEALY AND MOORE MACHINES

Mealy vs. Moore Machines

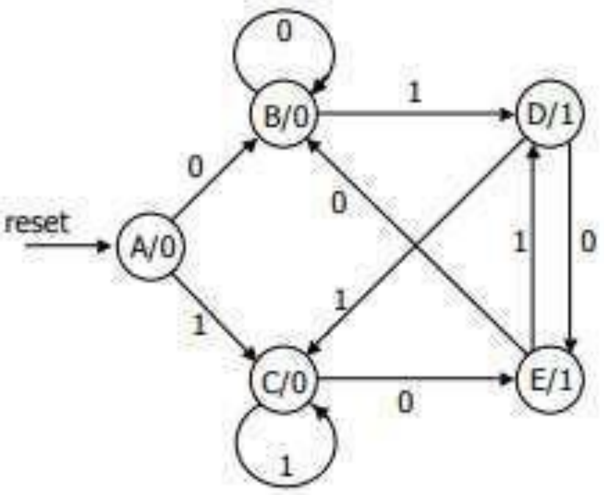
Moore: outputs depend on current state only

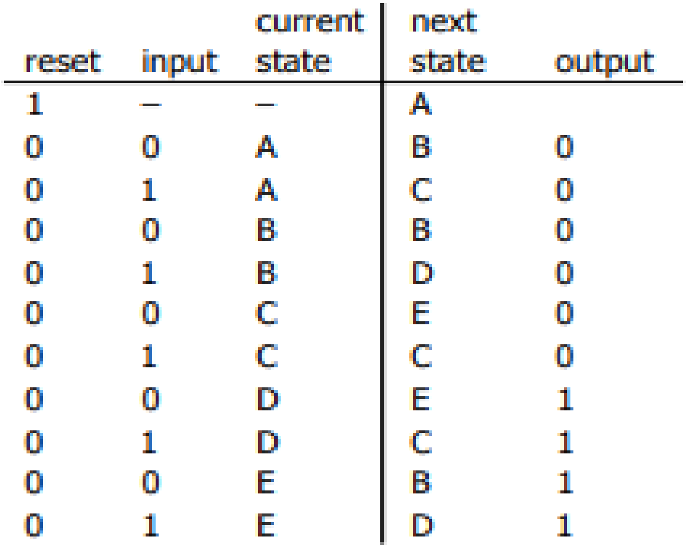
Mealy: outputs depend on current state and inputs

- ➢Ant brain is a Moore Machine (Output does not react immediately to input change)
- ➢We could have specified a Mealy FSM (Outputs have immediate reaction to inputs . As inputs change, so does next state, doesn’t commit until clocking event)

Specifying Outputs for a Moore Machine

Output is only function of state. Specify in state bubble in state diagram. Example: sequence detector for 01 or 10



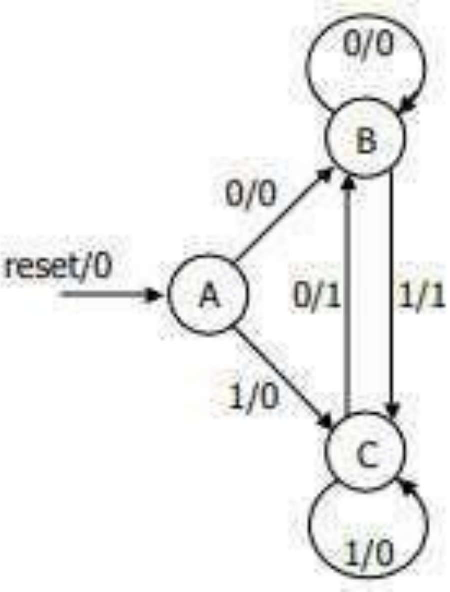


reset	input	current state	next state	output
1	–	–	A	
0	0	A	B	0
0	1	A	C	0
0	0	B	B	0
0	1	B	D	0
0	0	C	E	0
0	1	C	C	0
0	0	D	E	1
0	1	D	C	1
0	0	E	B	1
0	1	E	D	1

MEALY AND MOORE MACHINES

Specifying Outputs for a Mealy Machine

Output is function of state and inputs .Specify output on transition arc between states. Example: sequence detector for 01 or 10



		current	next	
reset	input	state	state	output
1	–	–	A	0
0	0	A	B	0
0	1	A	C	0
0	0	B	B	0
0	1	B	C	1
0	0	C	B	1
0	1	C	C	0

MEALY AND MOORE MACHINES

Comparison of Mealy and Moore Machines

- Mealy Machines tend to have less states

  Different outputs on arcs (n^2) rather than states (n)

- Moore Machines are safer to use

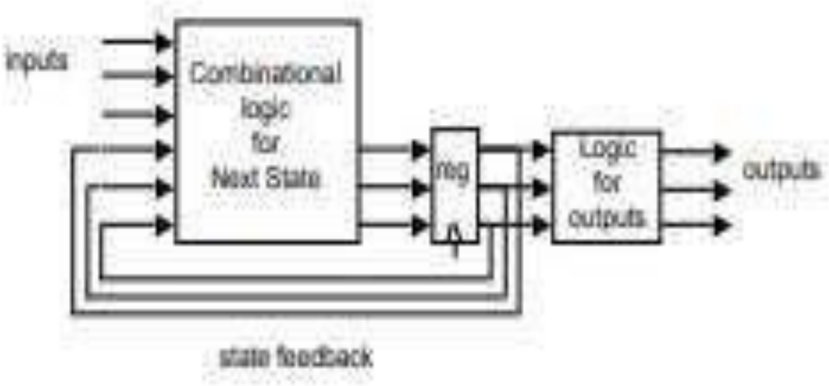
  Outputs change at clock edge (always one cycle later)

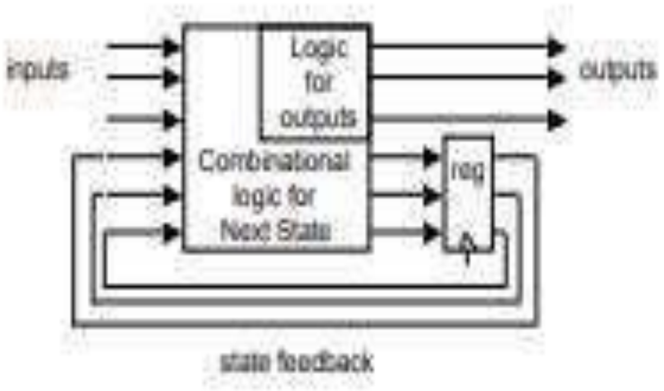
  In Mealy machines, input change can cause output change as soon as logic is done – a big problem when two machines are interconnected – asynchronous feedback

- Mealy Machines react faster to inputs

  React in same cycle – don't need to wait for clock

  In Moore machines, more logic may be necessary to decode state into outputs – more gate delays after





synchronous sequential circuits a clock signal consisting of pulses, controls the state variables which are represented by flip-flops. they are said to operate in pulse mode.

asynchronous circuits state changes are not triggered by clock pulses. they depend on the values of the input and feedback variables.

two conditions for proper operation:

1.-inputs to the circuit must change one at a time and must remain constant until the circuit reaches stable state.

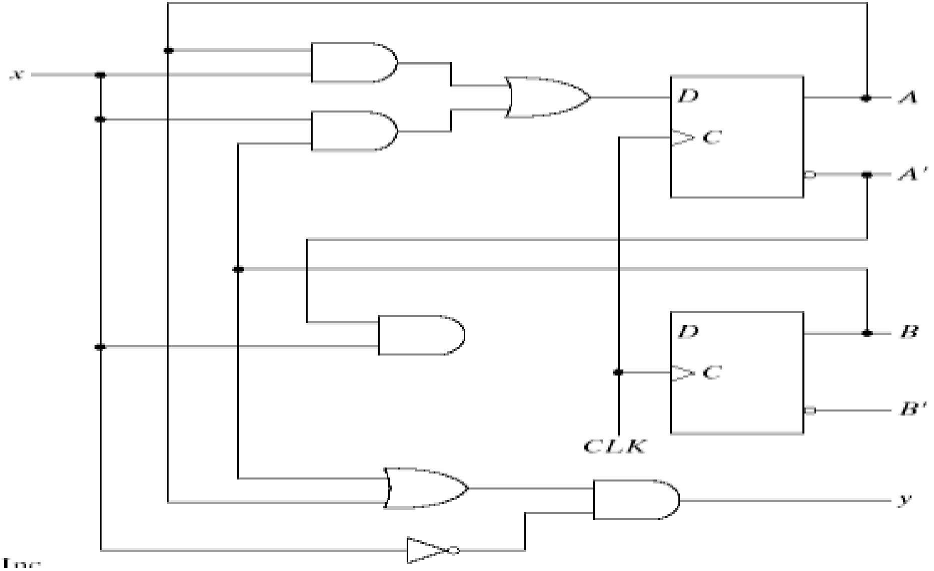
2.-feedback variables should change also one at a time. when all internal signals stop changing, then the circuit is said to have reached stable state when the inputs satisfy condition 1 above, then the circuit is said to operate in fundamental mode.

Analysis of Clocked Sequential Circuits

The analysis of a sequential circuit consists of obtaining a table or a diagram for the time sequence of inputs, outputs, and internal states. It is also possible to write Boolean expressions that describe the behavior of the sequential circuit. These expressions must include the necessary time sequence, either directly or indirectly.

State Equations

The behavior of a clocked sequential circuit can be described algebraically by means of state equations. A state equation specifies the next state as a function of the present state and inputs. Consider the sequential circuit shown in Fig. 5-15. It consists of two D flip-flops A and B, an input x and an output y.



State equation

$A(t+1) = A(t)\ x(t) + B(t)\ x(t)$

$B(t+1) = A`(t)\ x(t)$

A state equation is an algebraic expression that specifies the condition for a flip-flop state transition. The left side of the equation with (t+1) denotes the next state of the flip-flop one clock edge later. The right side of the equation is Boolean expression that specifies the present state and input conditions that make the next state equal to 1.

$$Y(t) = (A(t) + B(t))\ x(t)`$$

State Table

The time sequence of inputs, outputs, and flip-flop states can be enumerated in a state table (sometimes called transition table).

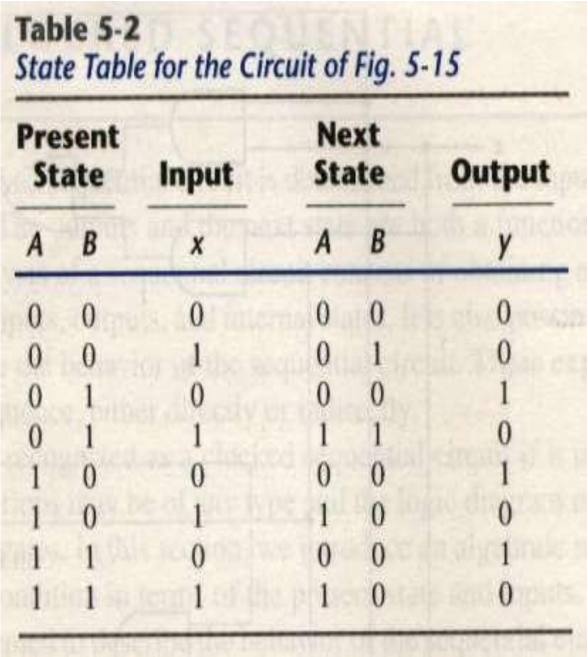


Table 5-2
State Table for the Circuit of Fig. 5-15

Present State		Input	Next State		Output
A	B	x	A	B	y
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	1
0	1	1	1	1	0
1	0	0	0	0	1
1	0	1	1	0	0
1	1	0	0	0	1
1	1	1	1	0	0

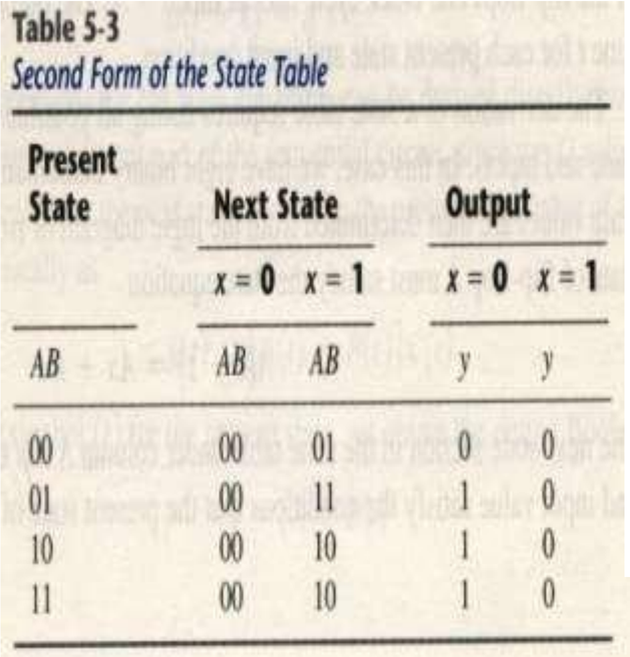


Table 5-3
Second Form of the State Table

Present State	Next State		Output	
	$x = 0$	$x = 1$	$x = 0$	$x = 1$
AB	AB	AB	y	y
00	00	01	0	0
01	00	11	1	0
10	00	10	1	0
11	00	10	1	0

State Diagram

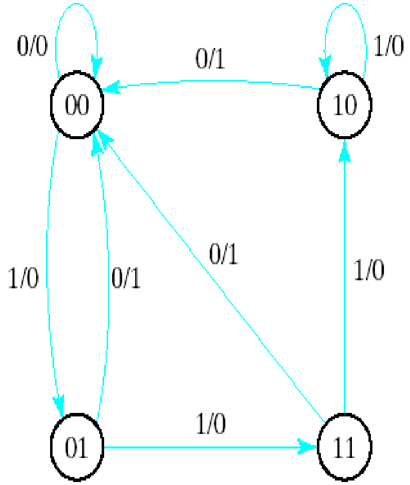


Fig. 5-16 State Diagram of the Circuit of Fig. 5-15

1/0 : means input =1
output=0

The information available in a state table can be represented graphically in the form of a state diagram. In this type of diagram, a state is represented by a circle, and the transitions between states are indicated by directed lines connecting the circles.

Flip-Flop Input Equations

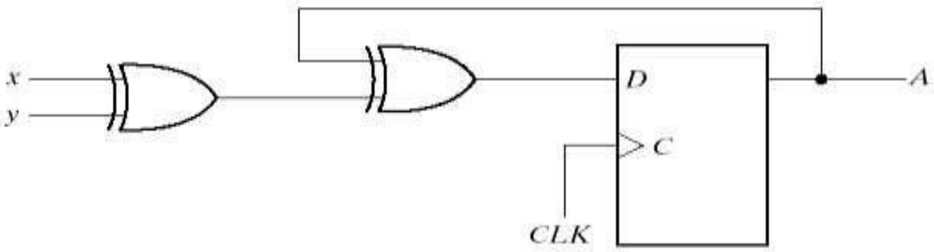
The part of the combinational circuit that generates external outputs is described algebraically by a set of Boolean functions called output equations. The part of the circuit that generates the inputs to flip-flops is described algebraically by a set of Boolean functions called flip-flop input equations. The sequential circuit of Fig. 5-15 consists of two D flip-flops A and B, an input x, and an output y. The logic diagram of the circuit can be expressed algebraically with two flip-flop input equations and an output equation:

$D_A = Ax + Bx$,$D_B = A`x$ and $y = (A + B)x`$

Analysis with D Flip-Flop

The circuit we want to analyze is described by the input equation $D_A = A \oplus x \oplus y$

The D_A symbol implies a D flip-flop with output A. The x and y variables are the inputs to the circuit. No output equations are given, so the output is implied to come from the output of the flip-flop.



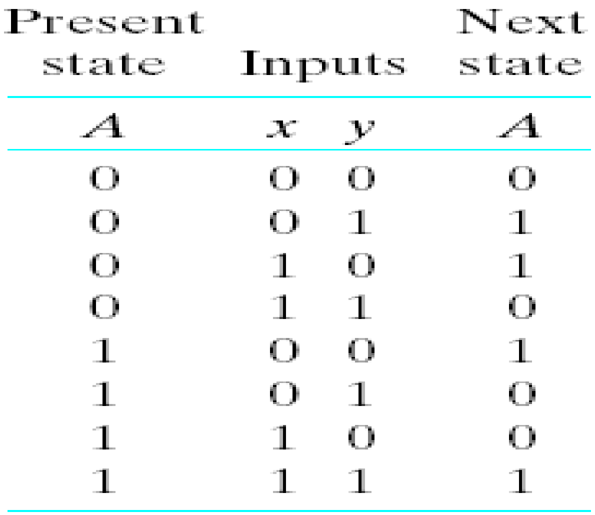
(a) Circuit diagram

SYNCHRONOUS AND ASYNCHRONOUS SEQUENTIAL CIRCUITS

The binary numbers under A ⊕ y are listed from 000 through 111 as shown in Fig. 5-17(b). The next state values are obtained from the state equation

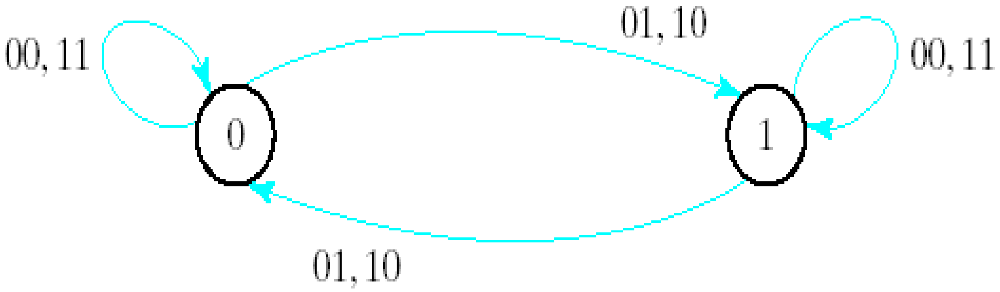
$$A(t+1) = A \oplus x \oplus y$$

The state diagram consists of two circles-one for each state as shown in Fig. 5-17(c)



Present state	Inputs		Next state
A	x	y	A
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

(b) State table



(c) State diagram

ASYNCHRONOUS SEQUENTIAL CIRCUIT

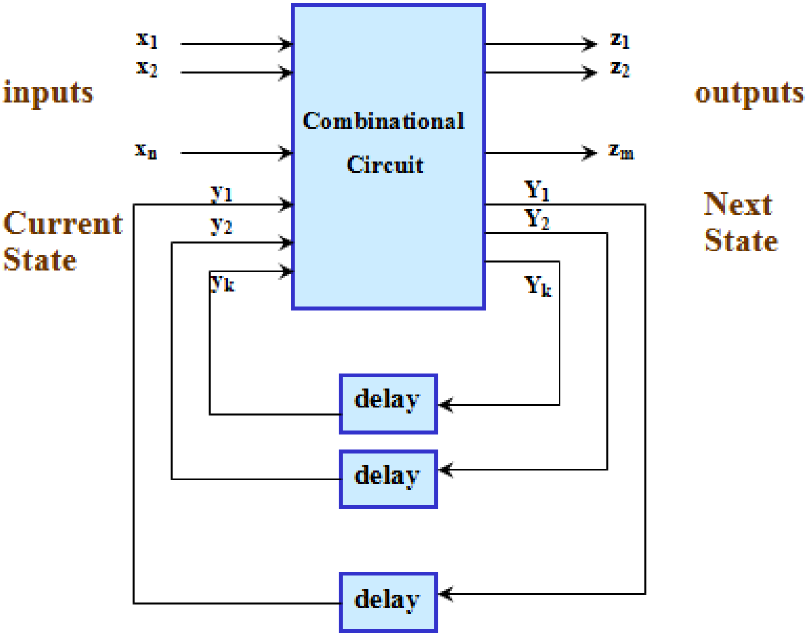
Analysis of asynchronous circuits

Procedure:

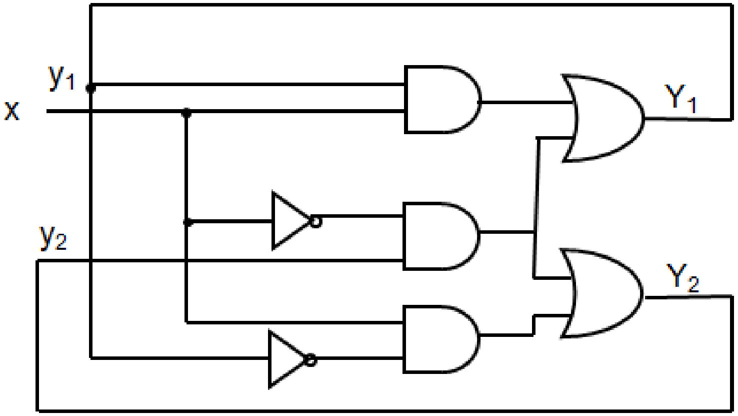
- Cut all feedback paths and insert a delay element at each point where cut was made
- Input to the delay element is the next state variable y_i while the output is the present value y_i .
- Derive the next-sate and output expressions from the circuit
- Derive the excitation table
- Derive the flow table
- Derive a state-diagram from the flow table
- Asynchronous circuits don’t use clock pulses
  - State transitions by changes in inputs
- Storage Elements:
  - Clock less storage elements or Delay elements
- In many cases, as combinational feedback
  - Normally much harder to design

SYNCHRONOUS AND ASYNCHRONOUS SEQUENTIAL CIRCUITS

$y_i = Y_i$ in steady state (but may be different during transition) Simultaneous change in two (or more) inputs is prohibited. The time between two changes must be less than the time of stability.



Analysis



1. Find feedback loops and name feedback variables appropriately.
2. Find boolean expressions of Y_i's in terms of y_i's and inputs.

$$Y_1 = x.y_1 + x'.y_2$$
$$Y_2 = x.y_1' + x'.y_2$$

3.Draw a map by using rows: y_i’s, columns: inputs, entries: Y_i’s

$y_1 y_2$	x = 0	x = 1
00	0	0
01	1	0
11	1	1
10	0	1

$Y_1 = x.y_1 + x'.y_2$

$y_1 y_2$	x = 0	x = 1
00	0	1
01	1	1
11	1	0
10	0	0

$Y_2 = x.y_1' + x'.y_2$

$y_1 y_2$	x = 0	x = 1
00	00	01
01	11	01
11	11	10
10	00	10

(Transition Table) $Y_1 Y_2$

4.To have a stable state, Y must be = y (circled)

$y_1 y_2$	x = 0	x = 1
00	00	01
01	11	01
11	11	10
10	00	10

(Transition Table) Y_1 Y_2

At y_1y_2x = 000, if x: 0 → 1

then Y_1Y_2: 00 → 01

then y_1y_2 = 01 (2nd row): stable

SYNCHRONOUS AND ASYNCHRONOUS SEQUENTIAL CIRCUITS

In general, if an input takes the circuit to an unstable state, y_i's change untila stable state is found.

General state of circuit:

y_1y_2x:

There are 4 stable states:

000, 011, 110, 101

and 4 unstable states.

$y_1 y_2$	X = 0	X = 1
00	00	01
01	11	01
11	11	10
10	00	10

Flow Table

As Transition Table (but with symbolic states):

	X = 0	X = 1
a	a	b
b	c	b
c	c	d
d	a	d

State Table As synchronous

present state	next state X=0	next state X=1
00	00	01
01	11	01
10	00	10
11	11	10

SYNTHESIS OF ASYNCHROUNOUS CIRCUITS

This topic is not covered in this course. it belongs to a more advanced logic design course.This it is very important in todays digital systems design because clocks are so fast that they present propagation delays making subsystems to operate out of synchronization.

Techniques for synthesis of asynchronous circuits include

The hoffman or classic synthesis approach

Handshaking signaling for two subsystems to communicate asynchronously

SHIFT REGISTERS

Introduction :

Shift registers are a type of sequential logic circuit, mainly for storage of digital data. They are a group of flip-flops connected in a chain so that the output from one flip-flop becomes the input of the next flip-flop. Most of the registers possess no characteristic internal sequence of states. All the flip-flops are driven by a common clock, and all are set or reset simultaneously. Shift registers are divided into two types.

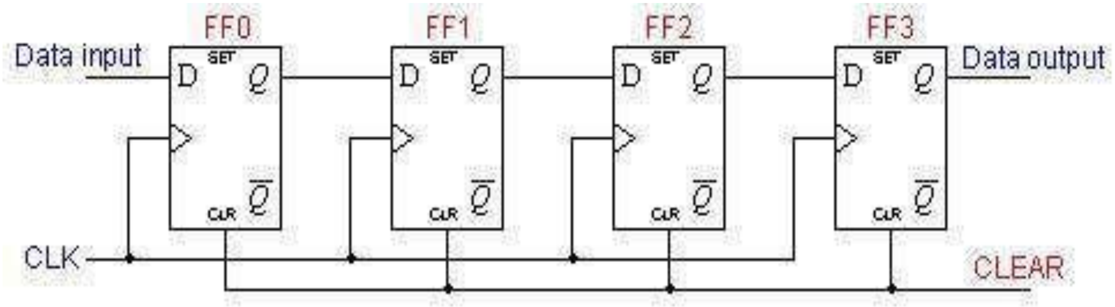
1. Uni directional shift registers
   1. Serial in – serial out shift register
   2. Serial in – parallel out shift register
   3. Parallel in – serial out shift register
   4. Parallel in – parallel out shift register

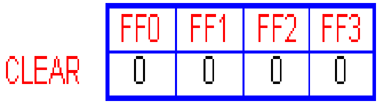
2. Bidirectional shift registers
   1. Left shift register
   2. Right shift register

SHIFT REGISTERS

1.Serial in – serial out shift register

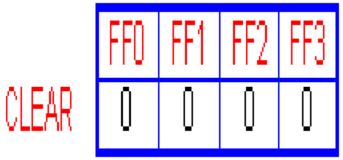
A basic four-bit shift register can be constructed using four D flip-flops, as shown below. The operation of the circuit is as follows. The register is first cleared, forcing all four outputs to zero. The input data is then applied sequentially to the D input of the first flip-flop on the left (FF0). During each clock pulse, one bit is transmitted from left to right. Assume a data word to be 1001. The least significant bit of the data has to be shifted through the register from FF0 to FF3.




CLEAR

FF0	FF1	FF2	FF3
0	0	0	0

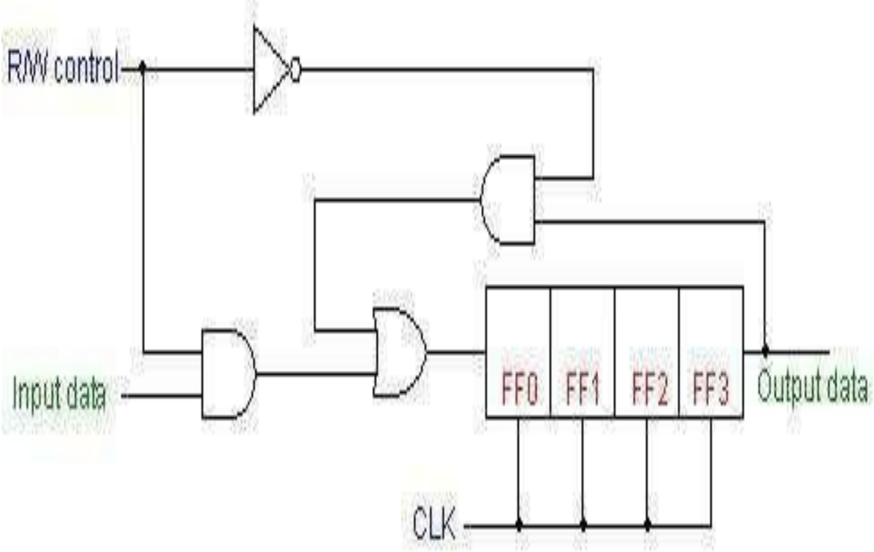
In order to get the data out of the register, they must be shifted out serially. This can be done destructively or non-destructively. For destructive readout, the original data is lost and at the end of the read cycle, all flip-flops are reset to zero.


CLEAR

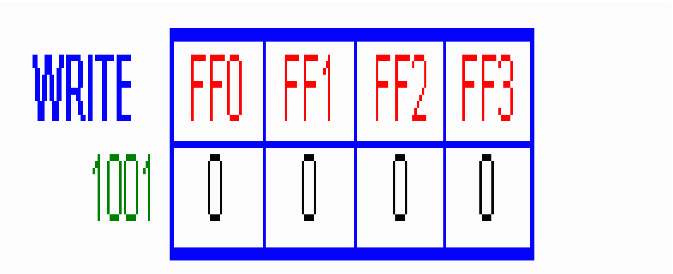
FF0	FF1	FF2	FF3
0	0	0	0

SHIFT REGISTERS

To avoid the loss of data, an arrangement for a non-destructive reading can be done by adding two AND gates, an OR gate and an inverter to the system. The construction of this circuit is shown below



The data is loaded to the register when the control line is HIGH (ie WRITE). The data can be shifted out of the register when the control line is LOW (ie READ). This is shown in the animation below.



WRITE

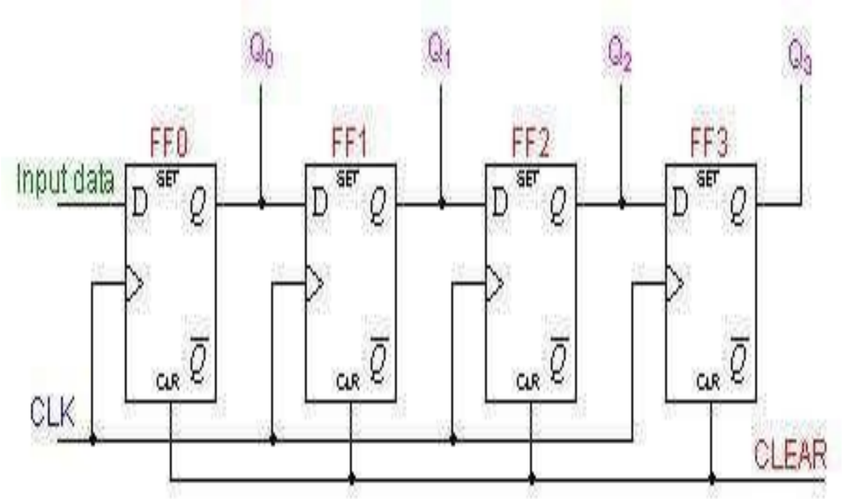
1001

FF0	FF1	FF2	FF3
0	0	0	0

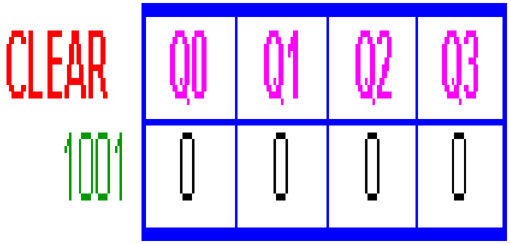
SHIFT REGISTERS

2.Serial in – parallel out shift register

The difference is the way in which the data bits are taken out of the register. Once the data are stored, each bit appears on its respective output line, and all bits are available simultaneously.



In the animation below, we can see how the four-bit binary number 1001 is shifted to the Q outputs of the register.

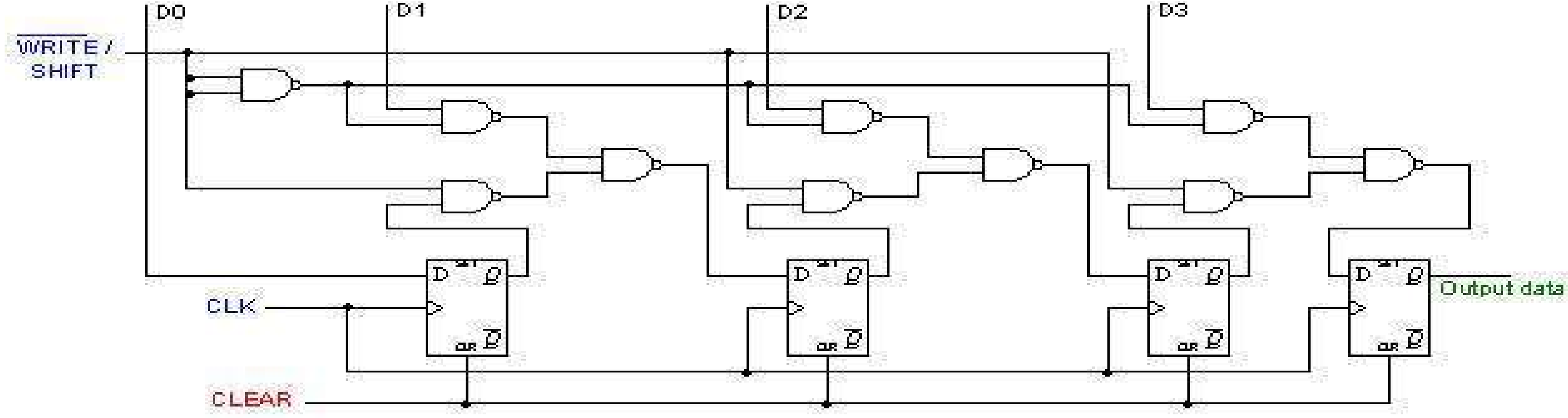


CLEAR	Q0	Q1	Q2	Q3
1001	0	0	0	0

SHIFT REGISTERS

3.Parallel in – serial out shift register

A four-bit parallel in - serial out shift register is shown below. The circuit uses D flip-flops and NAND gates for entering data (ie writing) to the register.



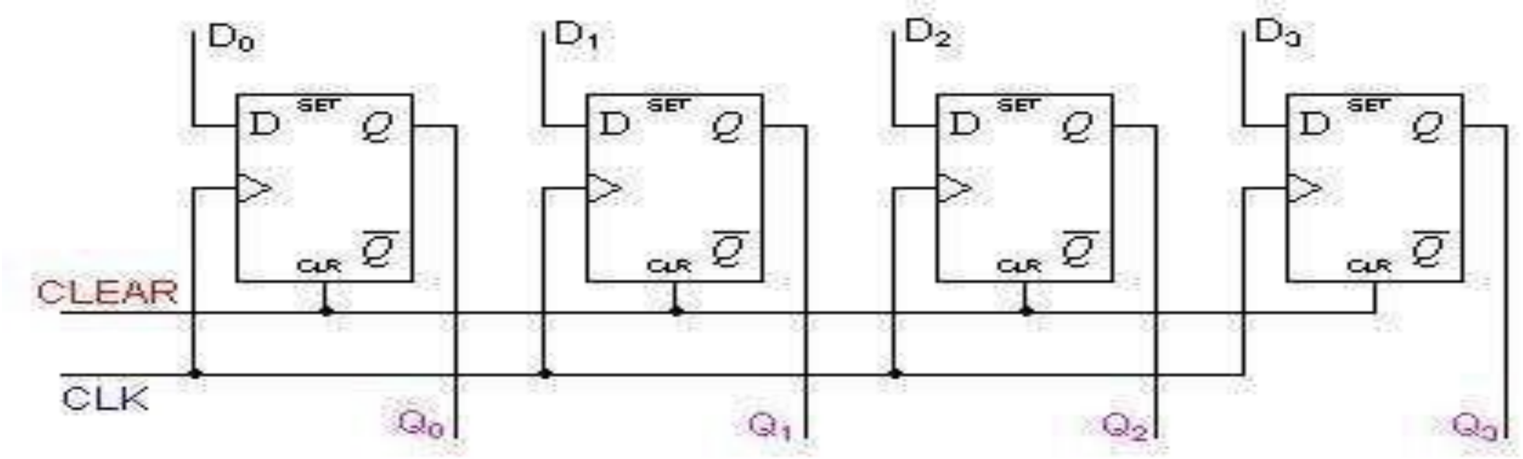
D0, D1, D2 and D3 are the parallel inputs, where D0 is the most significant bit and D3 is the least significant bit. To write data in, the mode control line is taken to LOW and the data is clocked in. The data can be shifted when the mode control line is HIGH as SHIFT is active high. The register performs right shift operation on the application of a clock pulse, as shown in the animation below.

	Q0	Q1	Q2	Q3
CLEAR	0	0	0	0

SHIFT REGISTERS

4.Parallel in –parallel out shift register

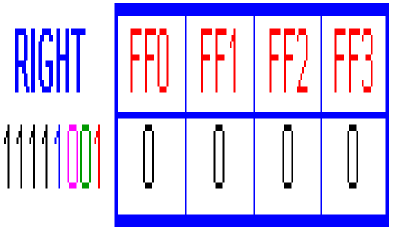
For parallel in - parallel out shift registers, all data bits appear on the parallel outputs immediately following the simultaneous entry of the data bits. The following circuit is a four-bit parallel in - parallel out shift register constructed by D flip-flops.



The D's are the parallel inputs and the Q's are the parallel outputs. Once the register is clocked, all the data at the D inputs appear at the corresponding Q outputs simultaneously.

Bidirectional Shift Registers

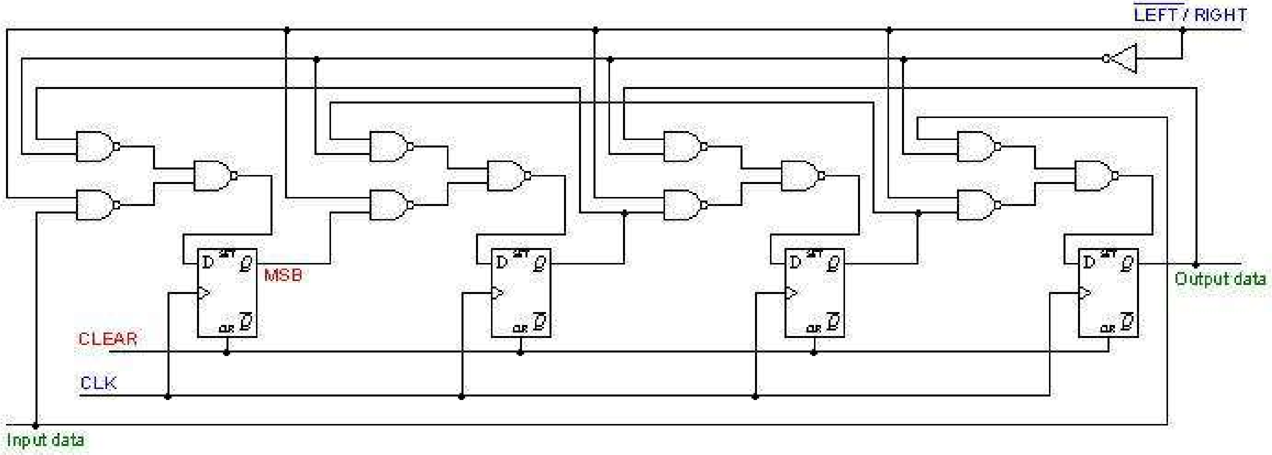
The registers discussed so far involved only right shift operations. Each right shift operation has the effect of successively dividing the binary number by two. If the operation is reversed (left shift), this has the effect of multiplying the number by two. With suitable gating arrangement a serial shift register can perform both operations. A *bidirectional*, or *reversible*, shift register is one in which the data can be shift either left or right. A four-bit bidirectional shift register using D flip-flops is shown below



RIGHT

11111001

FF0	FF1	FF2	FF3
0	0	0	0



Here a set of NAND gates are configured as OR gates to select data inputs from the right or left adjacent bitable, as selected by the LEFT/RIGHT control line.

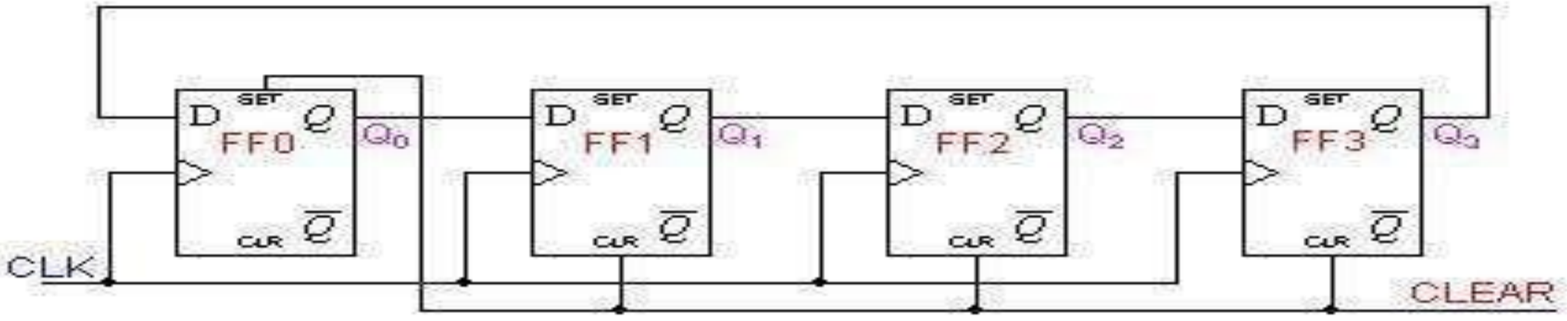
The animation below performs right shift four times, then left shift four times. Notice the order of the four output bits are not the same as the order of the original four input bits.

COUNTERS

Two of the most common types of shift register counters are introduced here: the Ring counter and the Johnson counter. They are basically shift registers with the serial outputs connected back to the serial inputs in order to produce particular sequences. These registers are classified as counters because they exhibit a specified sequence of states.

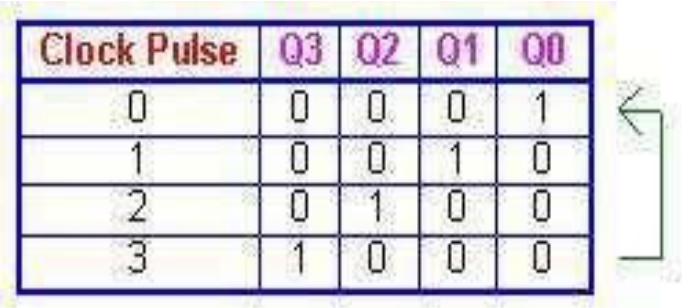
Ring Counters

A ring counter is basically a circulating shift register in which the output of the most significant stage is fed back to the input of the least significant stage. The following is a 4-bit ring counter constructed from D flip-flops. The output of each stage is shifted into the next stage on the positive edge of a clock pulse. If the CLEAR signal is high, all the flip-flops except the first one FF0 are reset to 0. FF0 is preset to 1 instead.

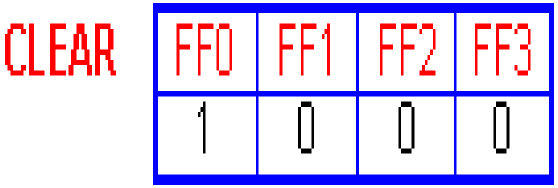


SHIFT REGISTERS AND COUNTERS

Since the count sequence has 4 distinct states, the counter can be considered as a mod-4 counter. Only 4 of the maximum 16 states are used, making ring counters very inefficient in terms of state usage. But the major advantage of a ring counter over a binary counter is that it is self-decoding. No extra decoding circuit is needed to determine what state the counter is in.



Clock Pulse	Q3	Q2	Q1	Q0
0	0	0	0	1
1	0	0	1	0
2	0	1	0	0
3	1	0	0	0



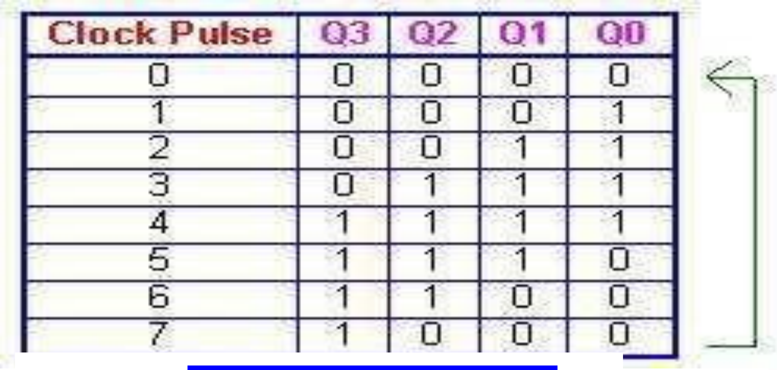
CLEAR

FF0	FF1	FF2	FF3
1	0	0	0

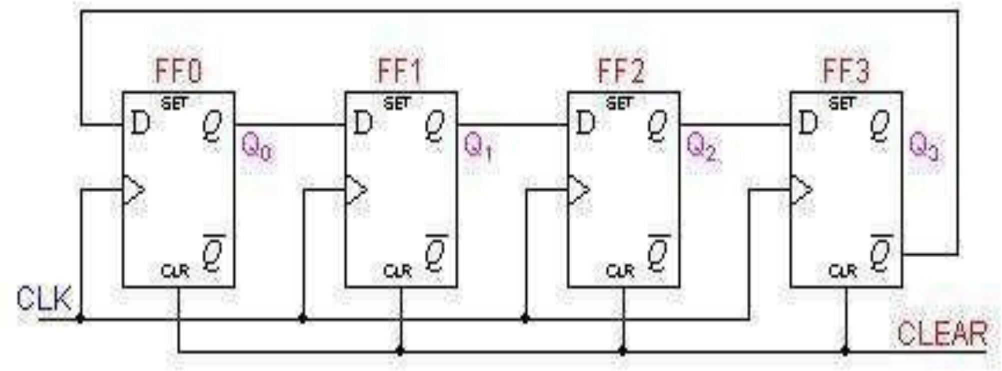
SHIFT REGISTERS AND COUNTERS

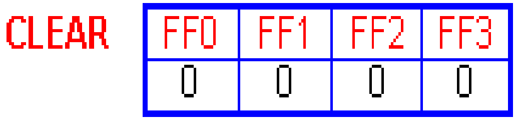
Johnson Counters

Johnson counters are a variation of standard ring counters, with the inverted output of the last stage fed back to the input of the first stage. They are also known as twisted ring counters. An *n*-stage Johnson counter yields a count sequence of length 2*n*, so it may be considered to be a mod-2*n* counter. The circuit above shows a 4-bit Johnson counter. The state sequence for the counter is given in the table as well as the animation on the left.



Clock Pulse	Q3	Q2	Q1	Q0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	1
3	0	1	1	1
4	1	1	1	1
5	1	1	1	0
6	1	1	0	0
7	1	0	0	0





CLEAR

FF0	FF1	FF2	FF3
0	0	0	0

Again, the apparent disadvantage of this counter is that the maximum available states are not fully utilized. Only eight of the sixteen states are being used.

Beware that for both the Ring and the Johnson counter must initially be forced into a valid state in the count sequence because they operate on a subset of the available number of states. Otherwise, the ideal sequence will not be followed.

RANDOM ACCESS MEMORY

UNIT 5
RANDOM ACCESS MEMORY

RANDOM ACCESS MEMORY

- A memory unit is a collection of storage cells together with associated circuits needed to transfer information in and out of the device.
- Memory cells can be accessed for information transfer to or from any desired random location and hence the name random access memory, abbreviated RAM.
- A memory unit stores binary information in groups of bits calledwords.

  1 byte = 8 bits

  1 word = 2 bytes.
- The communication between a memory and its environment is achieved through data input and output lines, address selection lines, and control lines that specify the direction of transfer.

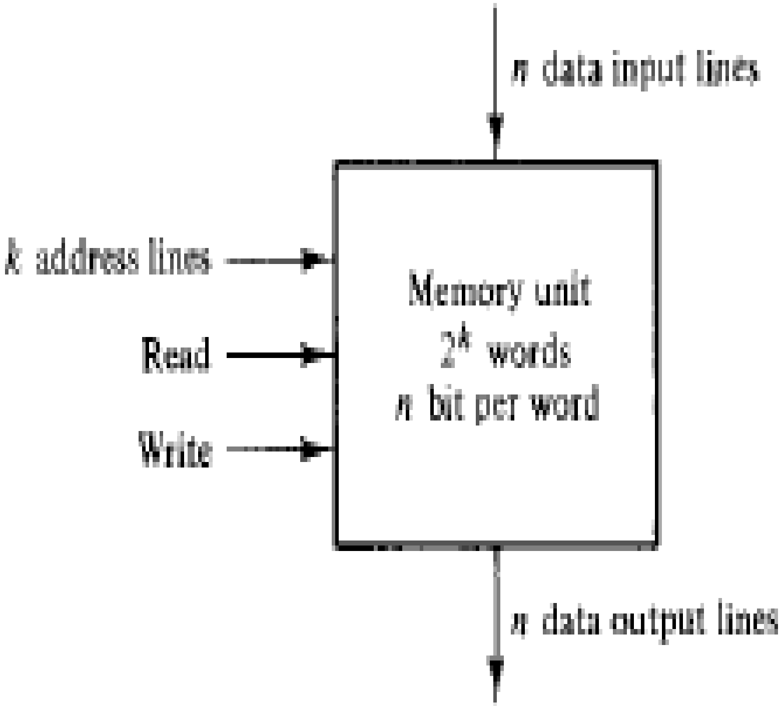


Fig 1: Block diagram of memory unit

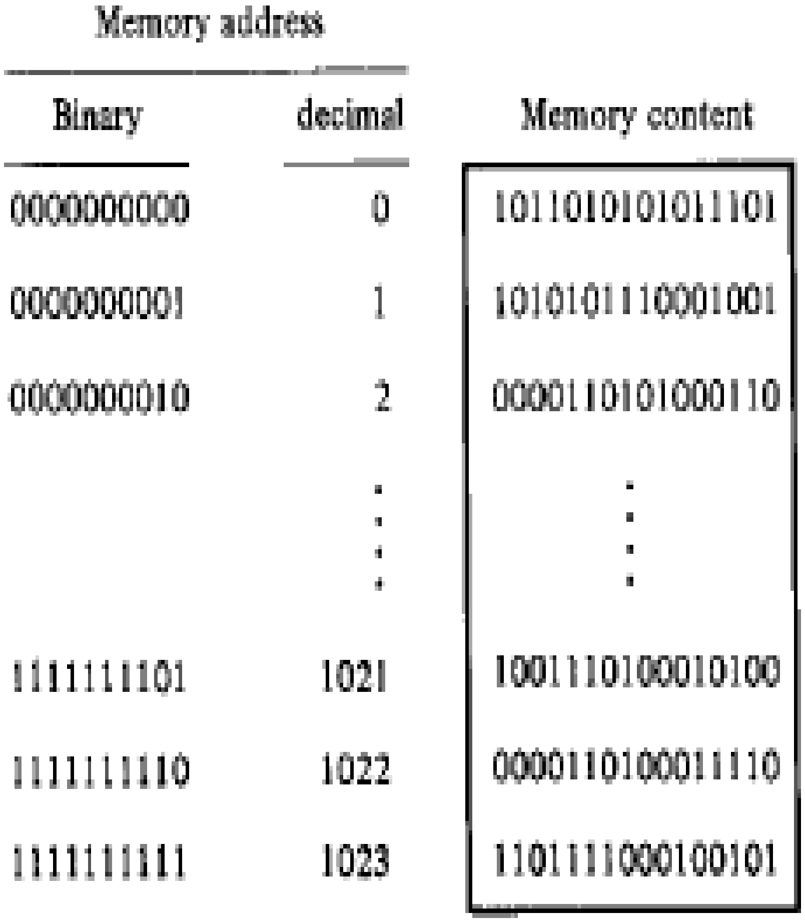
RANDOM ACCESS MEMORY

Content of a memory

- Each word in memory is assigned an identification number, called an address, starting from 0 up to 2^k-1, where k is the number of address lines.
- The number of words in a memory with one of the letters $K=2^{10}$, $M=2^{20}$, or $G=2^{30}$.

$64K = 2^{16}$ $2M = 2^{21}$

$4G = 2^{32}$



Memory address		
Binary	decimal	Memory content
0000000000	0	1011010101011101
0000000001	1	1010101110001001
0000000010	2	0000110101000110
	⋮	⋮
1111111101	1021	1001110100010100
1111111110	1022	0000110100011110
1111111111	1023	1101111000100101

Fig 2: Content of a 1024 x 16 memory

RANDOM ACCESS MEMORY

Write and Read operations

- Transferring a new word to be stored into memory:

1. Apply the binary address of the desired word to the address lines.
2. Apply the data bits that must be stored in memory to the data input lines.
3. Activate the write input.

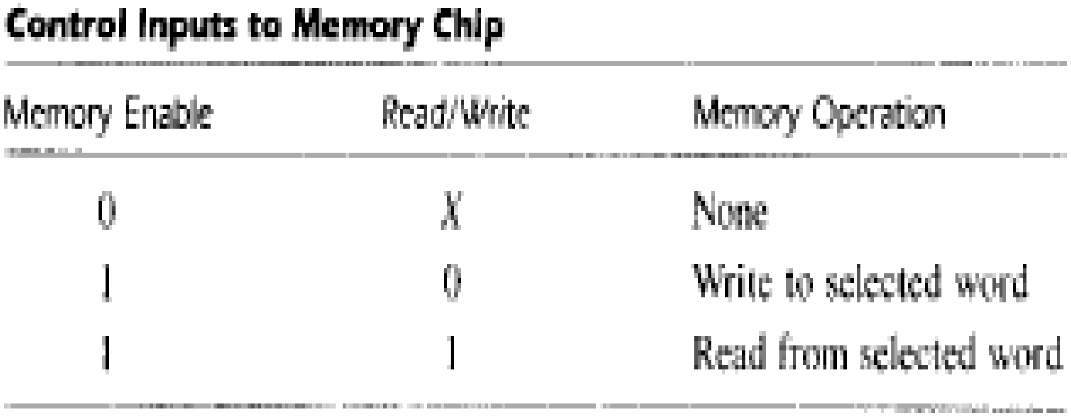
Write and Read operations

- Transferring a stored word out of memory:

1. Apply the binary address of the desired word to the address lines.
2. Activate the read input.

- Commercial memory sometimes provide the two control inputs for reading and writing in a somewhat different configuration in table 1.

Table 1



Control Inputs to Memory Chip

Memory Enable	Read/Write	Memory Operation
0	X	None
1	0	Write to selected word
1	1	Read from selected word

TYPES OF ROM

READ ONLY MEMORY (ROM)

- Computers almost always contain a small amount of read-only memory that holds instructions for starting up the computer. Unlike RAM, ROM cannot be written to.
- Because data stored in ROM cannot be modified (at least not very quickly or easily), it is mainly used to distribute firmware (software that is very closely tied to specific hardware, and unlikely to require frequent updates).
- It is non-volatile which means once you turn off the computerthe information is still there.

TYPES OF ROM

READ ONLY MEMORY (ROM)

- A type of memories that can only be read from any selected address in sequence.
- Stored data cannot be changed at all or cannot be changed without specialized equipment.
- Writing a data is not permitted.
- Reading data from any address does not destruct the content of read address.
- Usually to store data that is used repeatedly in system application.

TYPES OF ROM

Programmable ROM (PROM)

- Is a memory chip on which data can be written only once.
- Once a program has been written onto a PROM, it remains there forever.
- Nonvolatile memory - unlike RAM, PROM's retain their contents when the computer is turned off.
- The difference between a PROM and a ROM (read-only memory) is that a PROM is manufactured as blank memory, whereas a ROM is programmed during the manufacturing process.
- To write data onto a PROM chip, you need a special device called a PROM programmer or PROM burner.
- PROM uses some type of fusing process to store bits. Fusible link is programmed open or left intact to represent 0 or 1. The link cannot be changed once it is programmed.

TYPES OF ROM

EPROM

- Once it is erased, it can be reprogrammed.
- Two basic types
- **Ultraviolet (UV) EPROM**

  UV EPROM can be recognized by transparent quartz lid on the package.

  Entire UV EPROM data can be erased by exposing the transparent quartz lid to the high intensity UV light.
- **Electrically EPROM (EEPROM)**

  Individual bytes in EEPROM can be erased and programmed by electrical pulses (voltage).

TYPES OF ROM

Mask ROMs

- Usually referred to simply as ROM (the oldest type of solid state ROM)
- The data are permanently stored in the memory during the manufacturing process and it cannot be changed.
- Most IC ROMs utilize the presence or absence of a transistor connection at a row/column junction to represent a 1 or 0.
- Mask ROM and PROM can be of either MOS or bipolar technology.
- Despite the simplicity of mask ROM, economies of scale and **field-programmability** often make reprogrammable technologies more flexible and inexpensive.

MEMORY DECODING

MEMORY DECODING

- Decoder
  - select the memory word specified by the input address
- 2-dimensional coincident decoding is a more efficient decoding scheme for large memories

MEMORY DECODING

- The equivalent logic of a binary cell that stores one bit of information is shown below.

Read/Write = 0, select = 1, input data to S-R latch

Read/Write = 1, select = 1, output data from S-R latch

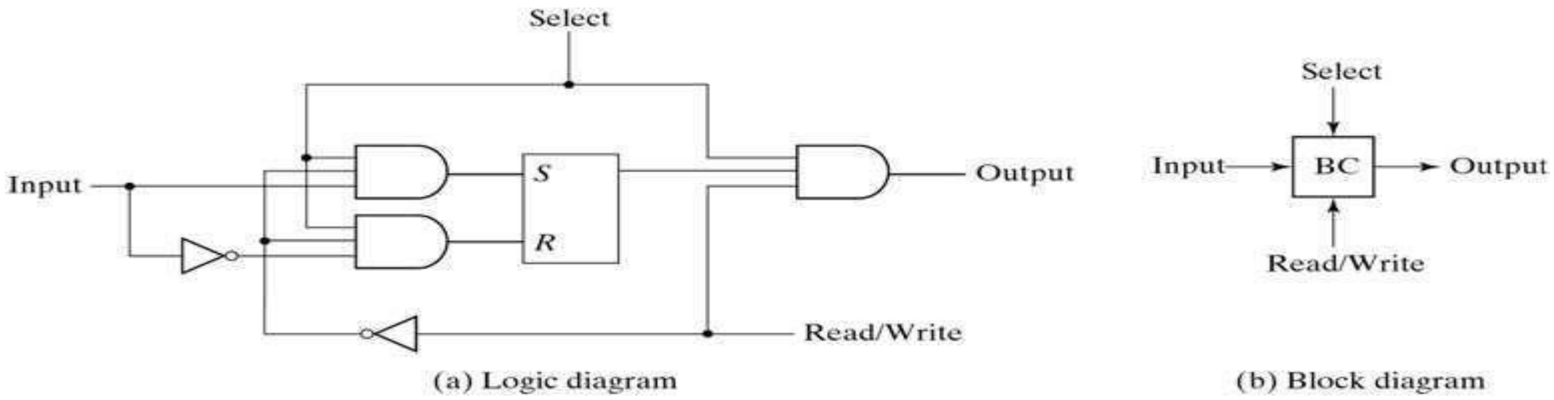
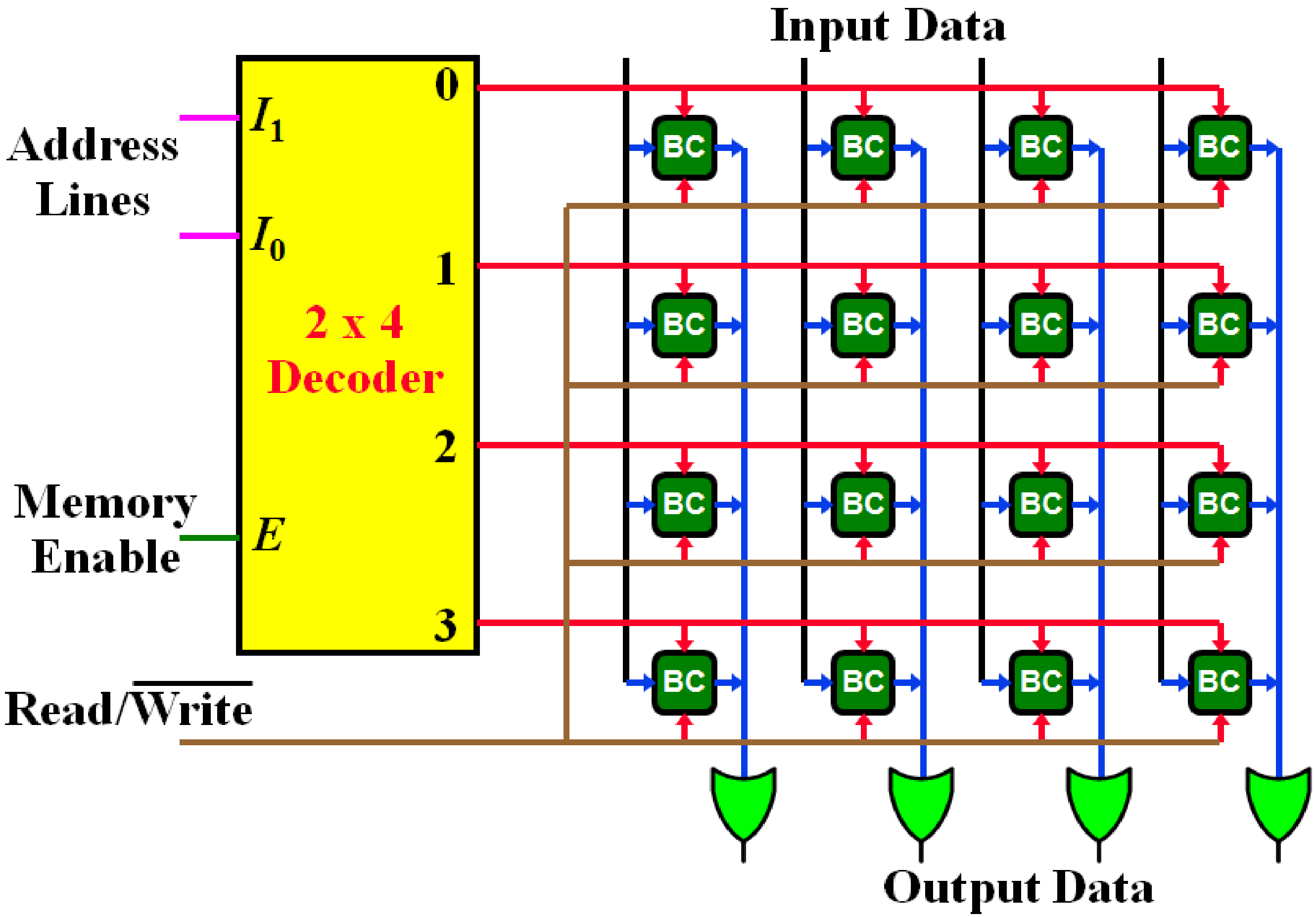


Figure 1: Memory cell

MEMORY DECODING

MEMORY ARRAY



MEMORY DECODING

4 x 4 RAM

- There is a need for decoding circuits to select the memory word specified by the input address.
- During the read operation, the four bits of the selected word go through OR gates to the output terminals.
- During the write operation, the data available in the input lines are transferred into the four binary cells of the selected word.
- A memory with 2^k words of n bits per word requires k address lines that go into $kx2^k$ decoder.

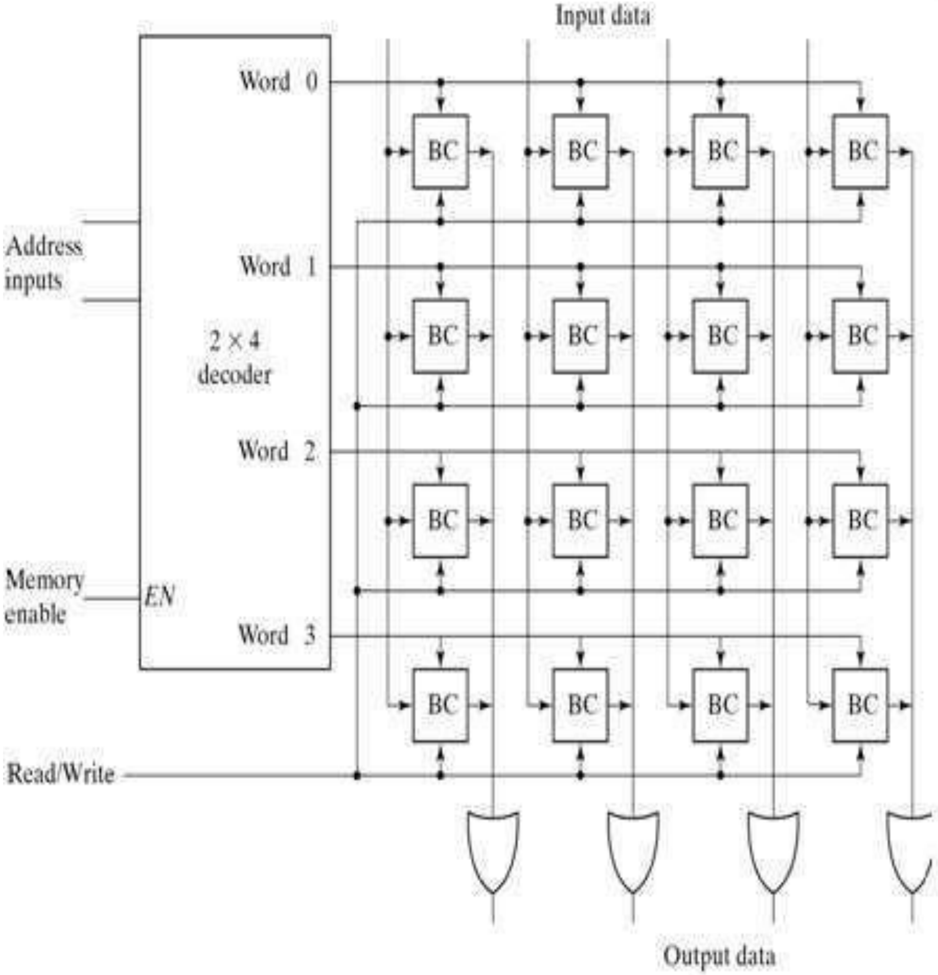


Fig 2: Diagram of 4 x 4 RAM

MEMORY DECODING

Coincident decoding

- A decoder with k inputs and 2^k outputs requires 2^k AND gates with k inputs per gate.
- Two decoding in a two-dimensional selection scheme can reduce the number of inputs per gate.
- 1K-word memory, instead of using a single 10X1024 decoder, we use two 5X32 decoders.

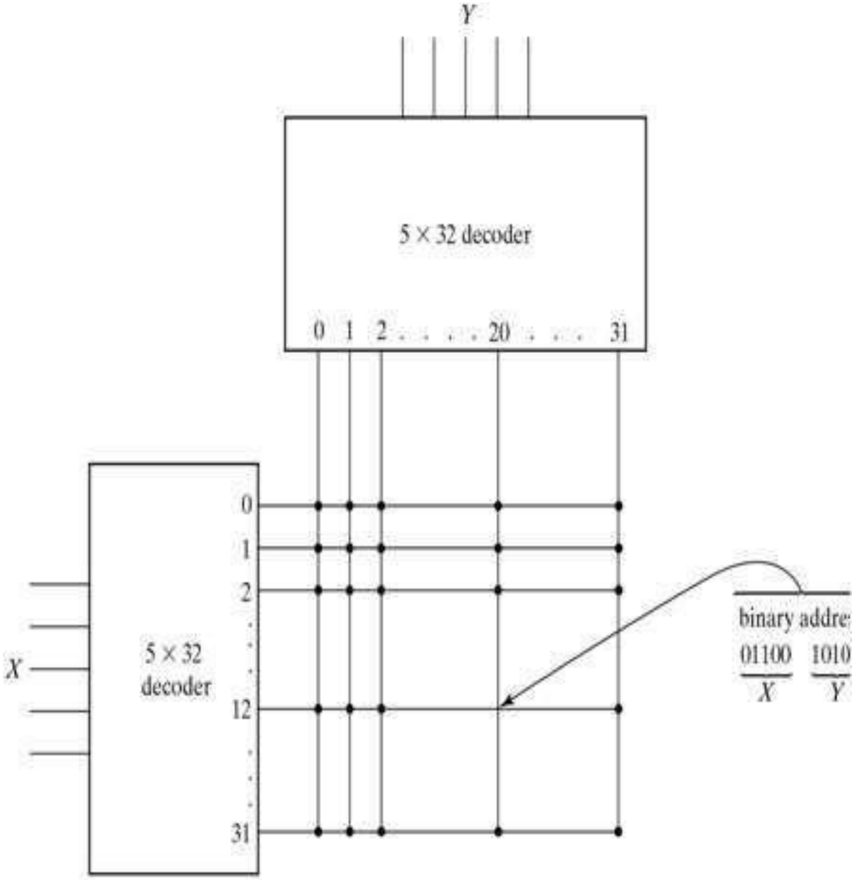


Figure 1: Two-dimensional Decoding structure for a 1K word memory

MEMORY DECODING

Address multiplexing

- DRAMs typically have four times the density of SRAM.

- The cost per bit of DRAM storage is three to four times less than SRAM. Another factor is lower power requirement.

MEMORY DECODING

Address multiplexing

- Address multiplexing will reduce the number of pins in the IC package.

- In a two-dimensional array, the address is applied in two parts at different times, with the row address first and the column address second. Since the same set of pins is used for both parts of the address, so can decrease the size of package significantly.

MEMORY DECODING

Address Multiplexing of a 64K DRAM

- After a time equivalent to the settling time of the row selection, RAS goes back to the 1 level.
- Registers are used to store the addresses of the row and column.
- CAS must go back to the 1 level before initialing another memory operation.

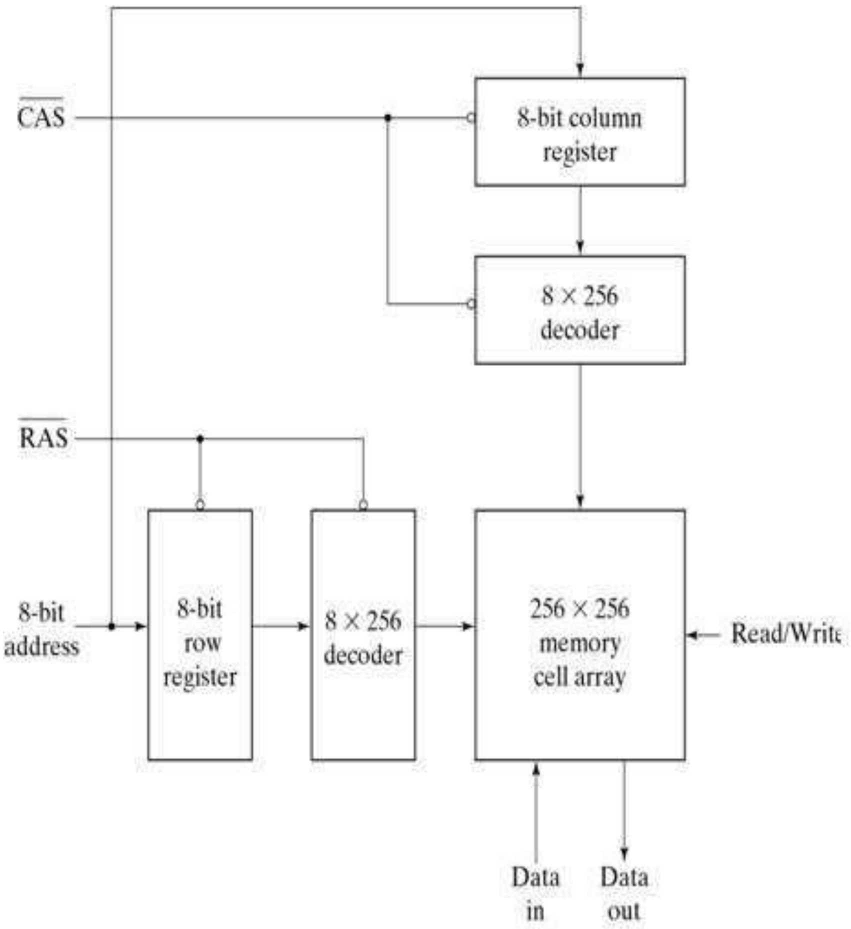


Fig 2: Address Multiplexing for a 64K DRAM

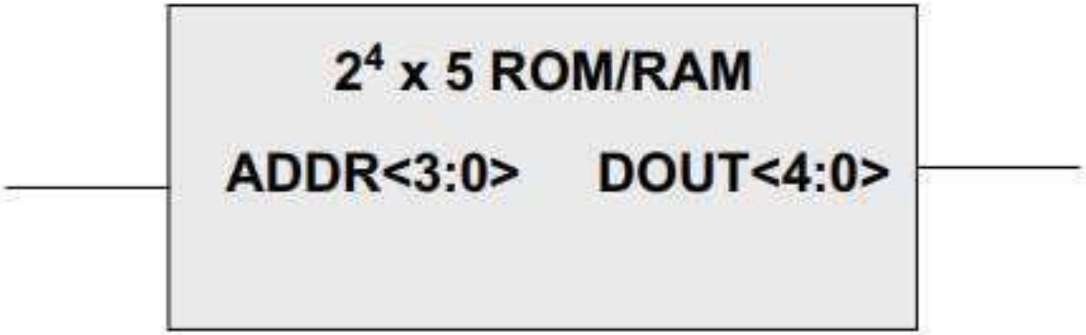
ADDRESS AND DATA BUS

ADDRESS BUS AND DATA BUS

Memory structures are crucial in digital design. – ROM, PROM, EPROM, RAM, SRAM, (S)DRAM, RDRAM,..

All memory structures have an address bus and a data bus – Possibly other control signals to control output etc.

E.g. 4 Bit Address bus with 5 Bit Data Bus



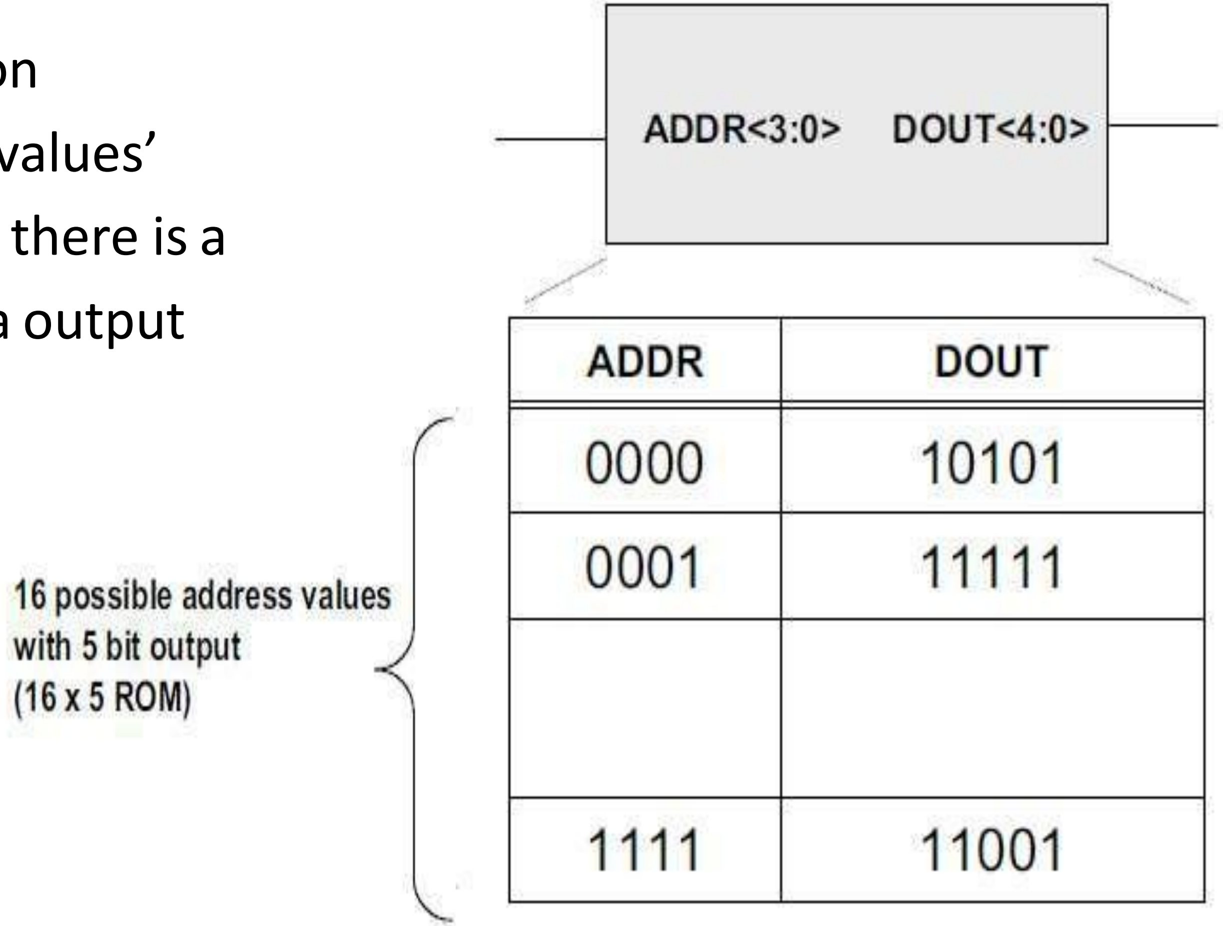
ADDRESS AND DATA BUS

ADDRESS BUS AND DATA BUS

Internal organization

– ‘Lookup Table of values’

– For each address there is a corresponding data output



ADDR	DOUT
0000	10101
0001	11111
1111	11001

16 possible address values
with 5 bit output
(16 x 5 ROM)

ADDRESS AND DATA BUS

ADDRESS BUS

- Address signals are required to specify the location in the memory from which information is accessed(read or written).
- A set of parallel address lines known as the address bus carry the address information.
- The number of bits (lines) comprising the address bus depends upon the size of the memory.
- For example, a memory having four locations to store data has four unique (00, 01, 10, 11) specified by a 2-bit address bus.
- The size of the address bus depends upon the total addressable locations specified by the formula 2^n, where n is the number of bits. Thus 2^4 = 16 (n=4) specifies 4 bits to uniquely identify 16 different locations.

ADDRESS AND DATA BUS

DATA BUS

- Data lines are required to retrieve the information from the memory array during a read operation and to provide the data that is to be stored in the memory during a write operation.
- As the memory reads or writes one data unit at a time therefore the data lines should be equal to the number of data bits stored at each addressable location in the memory.
- A memory organized as a byte memory reads or writes byte data values, therefore the number of data lines or the size of the data bus should be 8-bits or 1 byte.
- A memory organized to store nibble data values requires a 4-bit wide data bus. Generally, the wider the data bus more data can be accessed at each read or write operation

SEQUENTIAL MEMORY

SEQUENTIAL MEMORY

- Output depends on stored information (current state) and may be on current inputs
- Example:
- state = Score board of basket state = Score board of basketball game (number of points time game (number of points, time remaining, possession)
- input = which team scored the point
- output = point increase for the team that just scored.
- Sequential Circuits are built out of combinational logic and one or more memory/storage elements
- E.g. Registers, Memories, Counters, Control Unit .

SEQUENTIAL MEMORY

1-Bit memory element Characteristics

- Need a unit that can
- Retain/Remember a single bit with possible values ( ) state i.e. 0 or 1.
- This will allow us to read a previous value stored „
- Able to change the value (state). For a bit we can:
- Set the bit to 1
- Reset, or clear, the bit to 0
- To remember/retain their state values, rely on concept of feedback „
- Feedback digital circuits occurs when an output is looped back to the input „
- A simple example of this concept is shown below
- If Q is 0 it will always be 0, if it is 1 it will always be 1

SEQUENTIAL MEMORY

Flip Flop

Flip flop is a sequential circuit which generally samples its inputs and changes its outputs only at particular instants of time and not continuously. Flip flop is said to be edge sensitive or edge triggered rather than being level triggered like latches.

- **S-R Flip Flop**

It is basically S-R latch using NAND gates with an additional enable input. It is also called as level triggered SR-FF. For this, circuit in output will take place if and only if the enable input (E) is made active. In short this circuit will operate as an S-R latch if E = 1 but there is no change in the output if E = 0.

Flip Flop

- **Master Slave JK Flip Flop**

  Master slave JK FF is a cascade of two S-R FF with feedback from the output of second to input of first. Master is a positive level triggered. But due to the presence of the inverter in the clock line, the slave will respond to the negative level. Hence when the clock = 1 (positive level) the master is active and the slave is inactive. Whereas when clock = 0 (low level) the slave is active and master is inactive.

Flip Flop

- **Delay Flip Flop / D Flip Flop**

  Delay Flip Flop or D Flip Flop is the simple gated S-R latch with a NAND inverter connected between S and R inputs. It has only one input. The input data is appearing at the output after some time. Due to this data delay between i/p and o/p, it is called delay flip flop. S and R will be the complements of each other due to NAND inverter. Hence S = R = 0 or S = R = 1, these input condition will never appear. This problem is avoid by SR = 00 and SR = 1 conditions.

Flip Flop

- **Toggle Flip Flop / T Flip Flop**

  Toggle flip flop is basically a JK flip flop with J and K terminals permanently connected together. It has only input denoted by **T** .

CACHE MEMORY

CACHE

- A small amount of fast memory that sits between normal main memory and CPU
- May be located on CPU chip or module
- Intended to allow access speed approaching register speed
- When processor attempts to read a word from memory, cache is checked first

CACHE MEMORY

Cache Memory Principles

- If data sought is not present in cache, a block of memory of fixed size is read into the cache
- Locality of reference makes it likely that other words in the same block will be accessed soon

CACHE MEMORY

CACHE VIEW OF MEMORY

- N address lines => 2n words of memory
- Cache stores fixed length blocks of K words
- Cache views memory as an array of M blocks where M = 2^n/K
- A block of memory in cache is referred to as a line. K is the line size
- Cache size of C blocks where C < M (considerably)
- Each line includes a tag that identifies the block being stored
- Tag is usually upper portion of memory address

CACHE MEMORY

Cache operation – overview

- CPU requests contents of memory location
- Check cache for this data
- If present, get from cache (fast)
- If not present, read required block from main memory to cache
- Then deliver from cache to CPU
- Cache includes tags to identify which block of main memory is in each cache slot

PROGRAMMABLE LOGIC ARRAY

Programmable Logic Array

- A programmable logic array (PLA) is a type of logic device that can be programmed to implement various kinds of combinational logic circuits.
- The device has a number of AND and OR gates which are linked together to give output or further combined with more gates or logic circuits.

PROGRAMMABLE LOGIC ARRAY

Programmable Logic Array

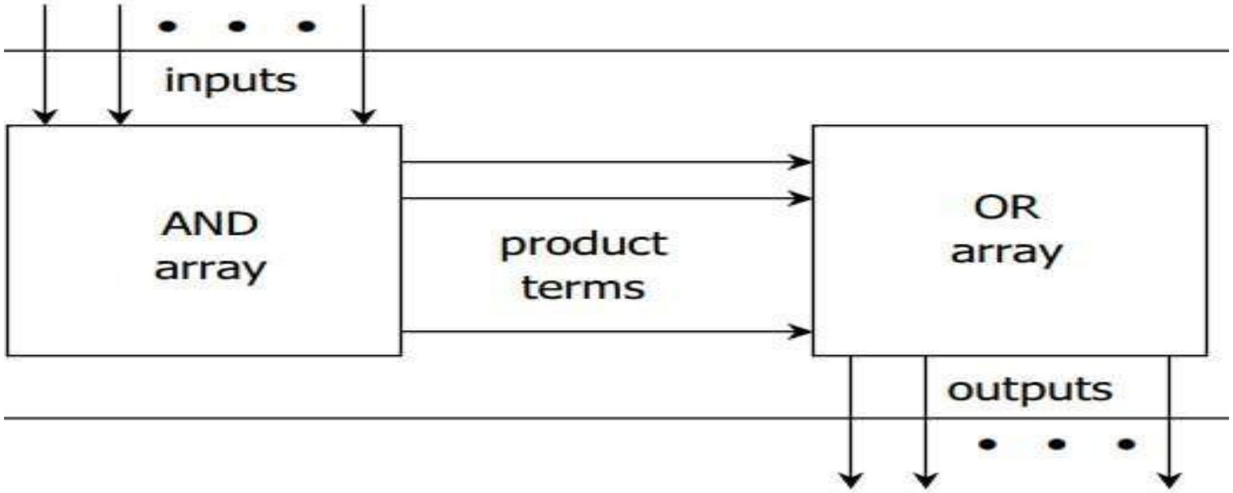


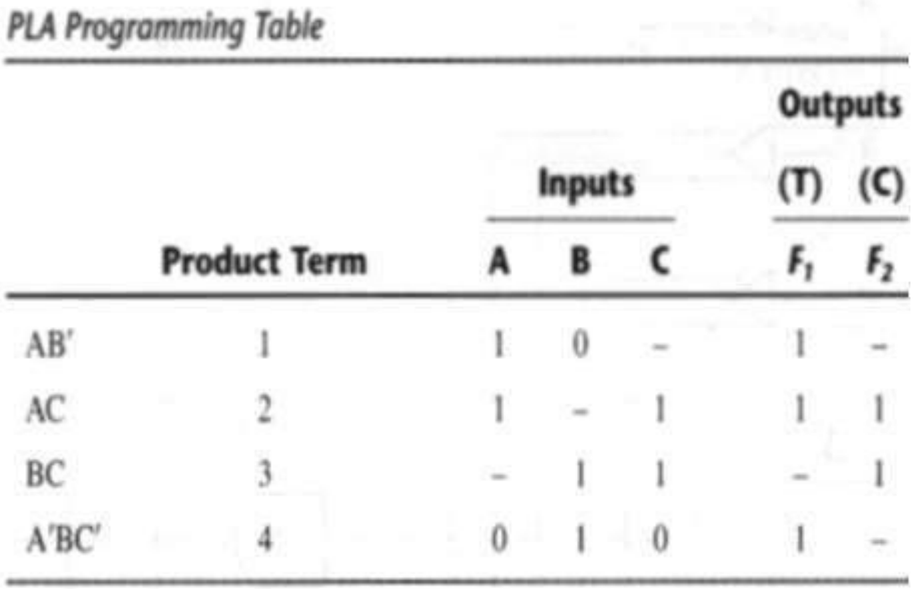
Fig 1: Block diagram of PLA

PROGRAMMABLE LOGIC ARRAY

PLA

F1 =AB’+AC+A’BC’

F2 = (AC+BC)’



PLA Programming Table

		Inputs			Outputs	
					(T)	(C)
Product Term		A	B	C	F_1	F_2
AB′	1	1	0	–	1	–
AC	2	1	–	1	1	1
BC	3	–	1	1	–	1
A′BC′	4	0	1	0	1	–

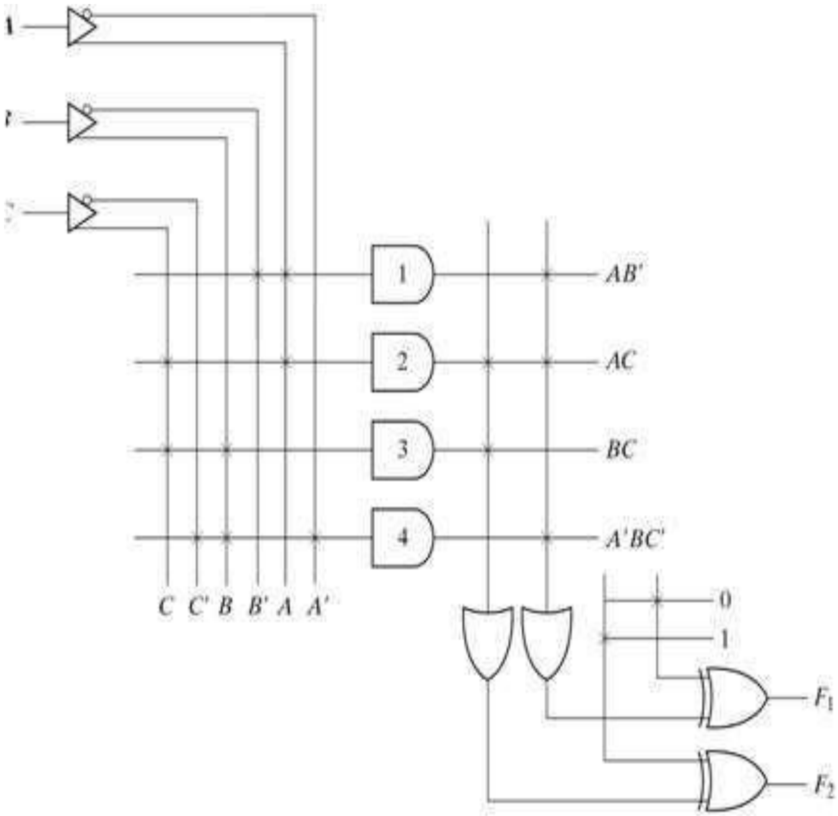


Fig 2: PLA with 3-inputs 4 product terms and 2 outputs

PROGRAMMABLE LOGIC ARRAY

Programmable Logic Array

- **Advantages**
- PLA architecture is more efficient than a PROM.
- **Disadvantage**
- PLA architecture has two sets of programmable fuses due to which PLA devices are difficult to manufacture, program and test.
- **Applications:**
- PLA is used to provide control over data path.
- PLA is used as a counter.
- PLA is used as decoders.
- PLA is used as a BUS interface in programmed I/O

PROGRAMMABLE LOGIC ARRAY

Programming Table

1. First: lists the product terms numerically
2. Second: specifies the required paths between inputs and AND gates
3. Third: specifies the paths between the AND and OR gates
4. For each output variable, we may have a T(ture) or C(complement) for programming the XOR gate

PROGRAMMABLE LOGIC ARRAY

Simplification of PLA

- Careful investigation must be undertaken in order to reduce the number of distinct product terms, PLA has a finite number of AND gates.

- Both the true and complement of each function should be simplified to see which one can be expressed with fewer product terms and which one provides product terms that are common to other functions.

PROGRAMMABLE LOGIC ARRAY

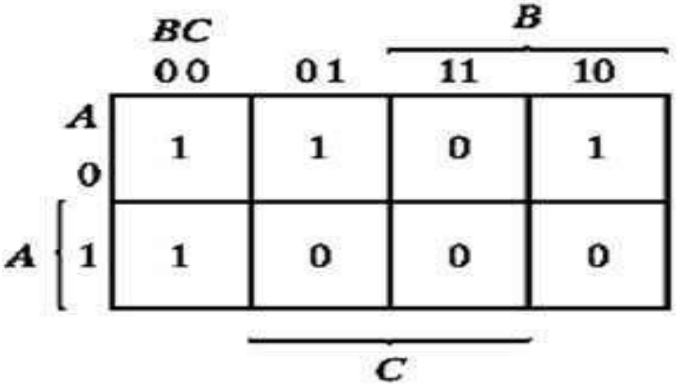
Example

Implement the following two Boolean functions with a PLA:

$F_1(A, B,C)$ = ? (0, 1, 2, 4)

$F_2(A, B, C)$ = ? (0, 5, 6, 7)

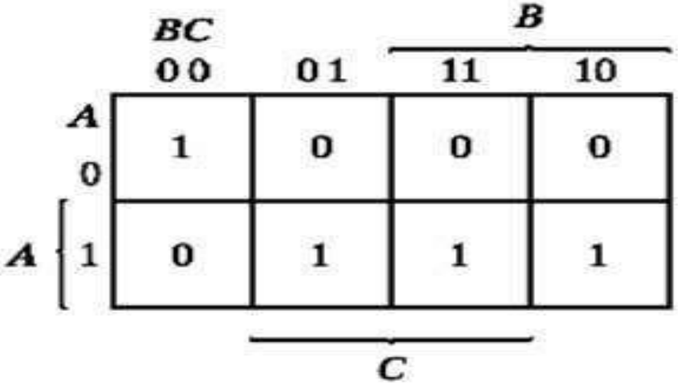
The two functions are simplified in the maps of given figure



A \ BC	00	01	11	10
0	1	1	0	1
1	1	0	0	0

$F_1 = A'B' + A'C' + B'C'$

$F_1 = (AB + AC + BC)'$



A \ BC	00	01	11	10
0	1	0	0	0
1	0	1	1	1

$F_2 = AB + AC + A'B'C'$

$F_2 = (A'C + A'B + AB'C')'$

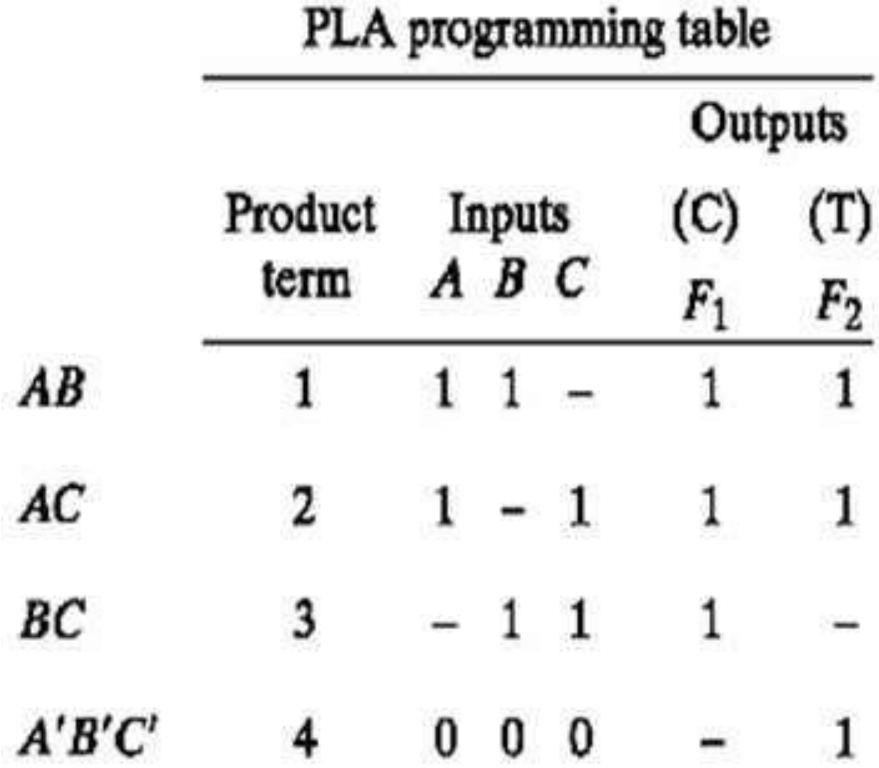
PROGRAMMABLE LOGIC ARRAY

PLA table by simplifying the function

- Both the true and complement of the functions are simplified in sum of products.
- We can find the same terms from the group terms of the functions of F_1, F_1',F_2 and F_2' which will make the minimum terms.

F1 = (AB +AC + BC)’

F2 = AB + AC + A’B’C’



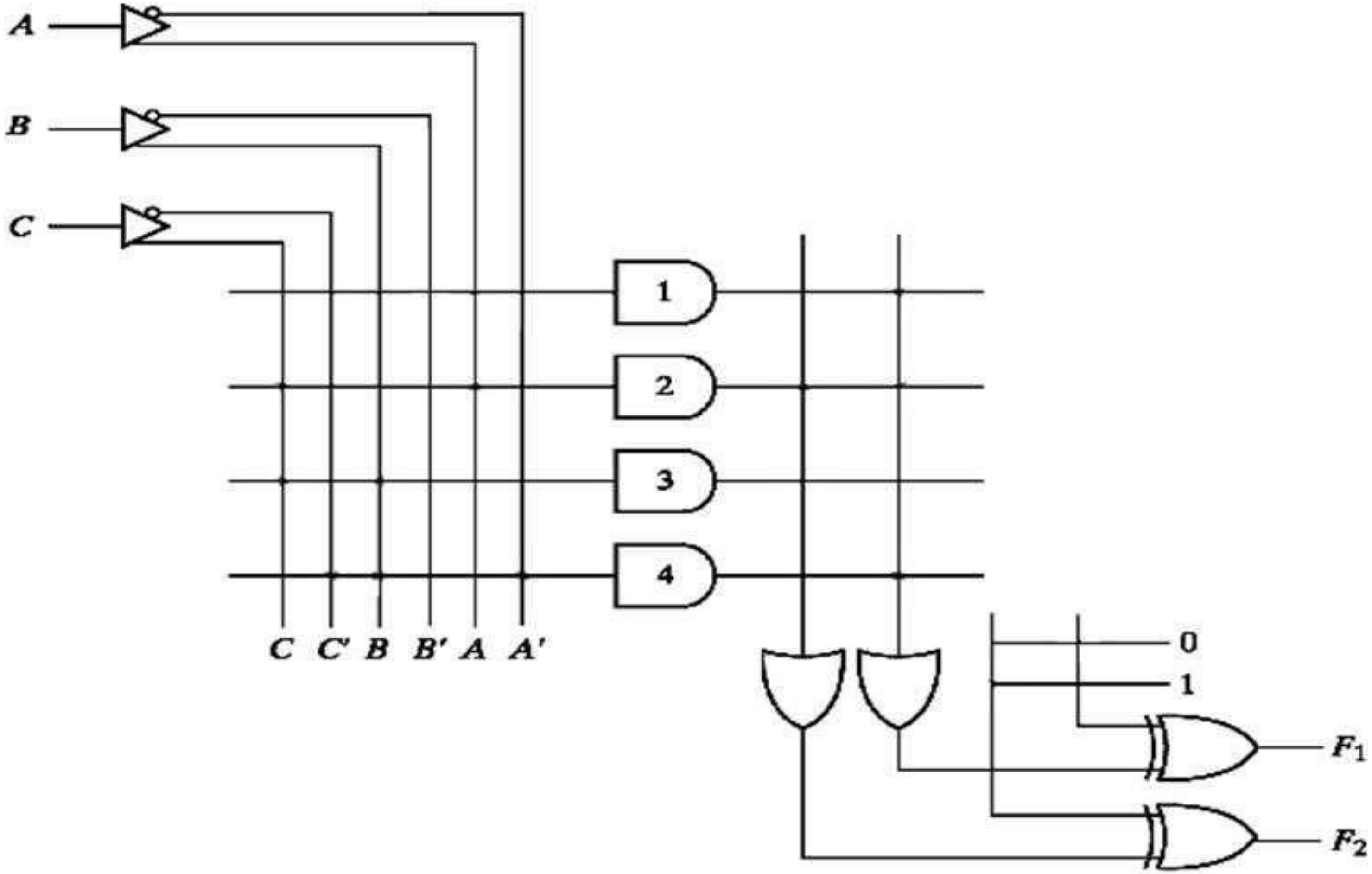
PLA programming table

	Product term	Inputs			Outputs	
		A	B	C	(C) F_1	(T) F_2
AB	1	1	1	–	1	1
AC	2	1	–	1	1	1
BC	3	–	1	1	1	–
$A'B'C'$	4	0	0	0	–	1

Fig 1: Solution to example

PROGRAMMABLE LOGIC ARRAY

PLA implementation



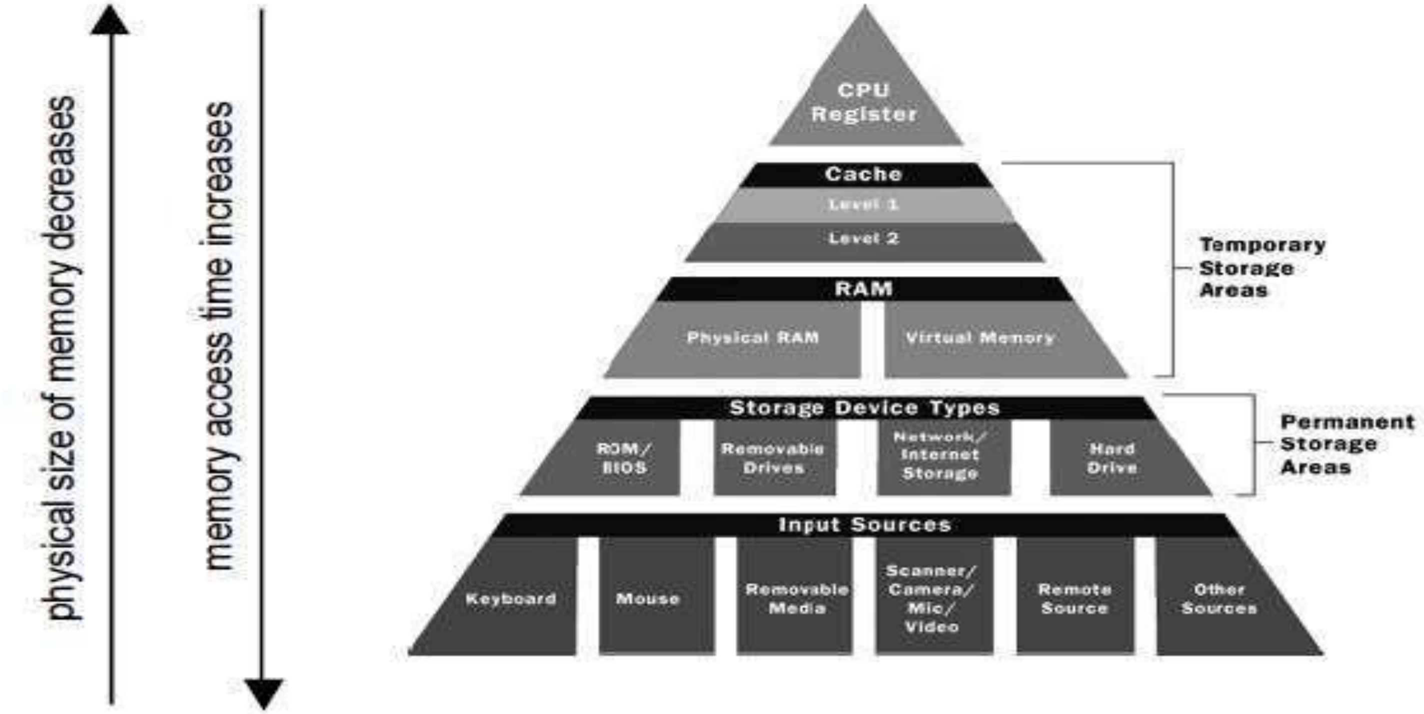
MEMORY HIAERARCHY

Memory Hierarchy

- For any memory:
  — How fast?
  — How much?
  — How expensive?
- Faster memory => greater cost per bit
- Greater capacity => smaller cost / bit
- Greater capacity => slower access
- Going down the hierarchy:
  — Decreasing cost / bit
  — Increasing capacity
  — Increasing access time
  — Decreasing frequency of access by processor

MEMORY HIAERARCHY

Memory Hierarchy - Diagram



MEMORY HIAERARCHY

MEMORY HIAERARCHY

- Registers
  - In CPU
- Internal or Main memory
  - May include one or more levels of cache
  - “RAM”
- External memory
  - Backing store

MEMORY HIAERARCHY

HIAERARCHY LIST

- Registers
- L1 Cache
- L2 Cache
- Main memory
- Disk cache
- Magnetic Disk
- Optical
- Tape

MEMORY HIAERARCHY

Locality of Reference

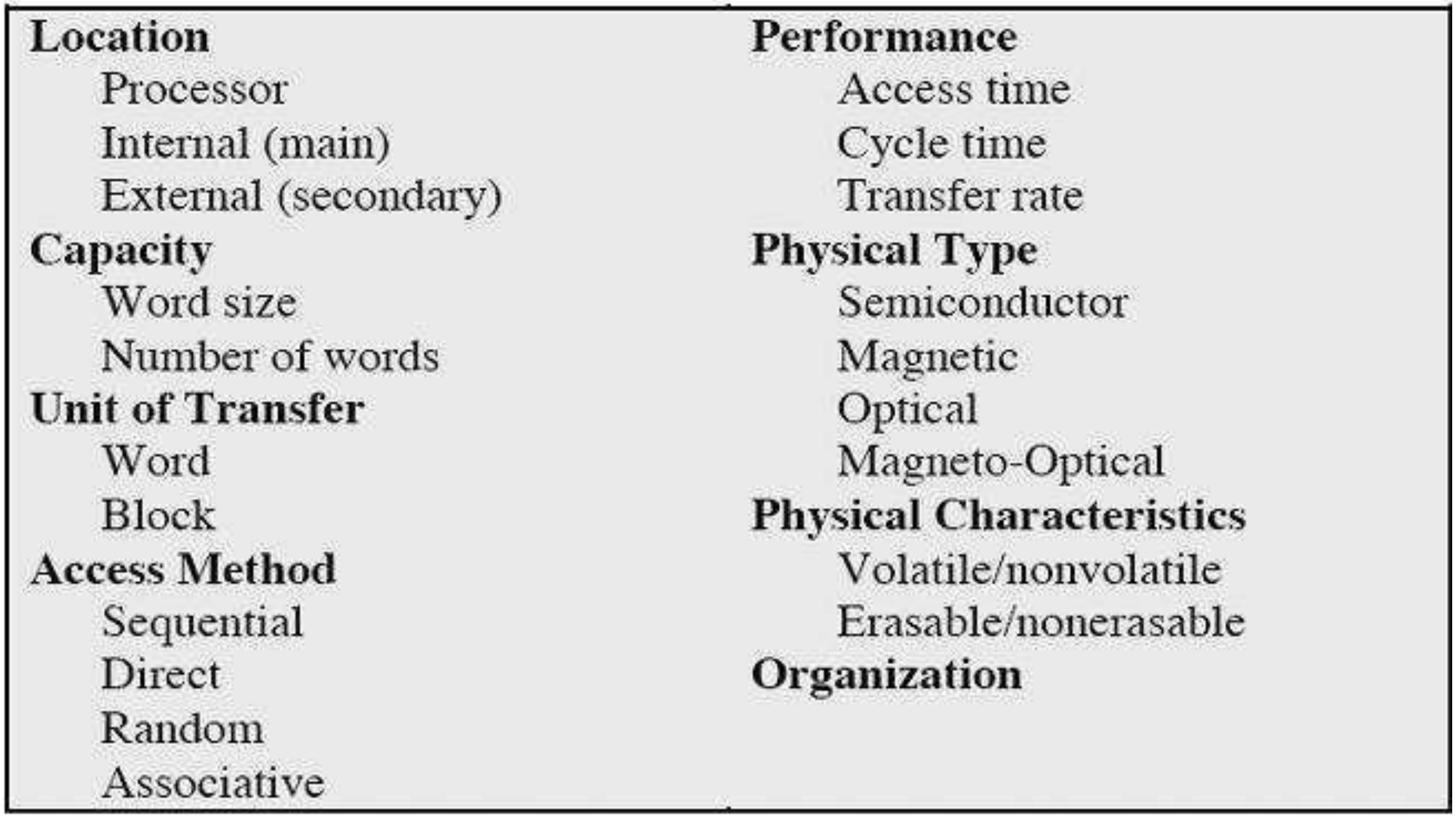
- Two or more levels of memory can be used to produce average access time approaching the highest level
- The reason that this works well is called “locality of reference”
- In practice memory references (both instructions and data) tend to cluster

— Instructions: iterative loops and repetitive subroutine calls

— Data: tables, arrays, etc. Memory references cluster in short run

MEMORY HIAERARCHY

Characteristics of Memory Systems



Location
- Processor
- Internal (main)
- External (secondary)

Capacity
- Word size
- Number of words

Unit of Transfer
- Word
- Block

Access Method
- Sequential
- Direct
- Random
- Associative

Performance
- Access time
- Cycle time
- Transfer rate

Physical Type
- Semiconductor
- Magnetic
- Optical
- Magneto-Optical

Physical Characteristics
- Volatile/nonvolatile
- Erasable/nonerasable

Organization

MEMORY HIAERARCHY

Capacity

- Word size

—The natural unit of organisation

—Typically number of bits used to represent an integer in the processor

- Number of words

—Most memory sizes are now expressed in bytes

—Most modern processors have byte-addressable memory but some have word addressable memory

— Memory capacity for A address lines is 2A addressable units

MEMORY HIAERARCHY

Access Methods

- Sequential

—Start at the beginning and read through in order

—Access time depends on location of data and previous location

— e.g. tape

- Direct

— Individual blocks have unique address

— Access is by jumping to vicinity plus sequential search

— Access time depends on location and previous location

— e.g. disk

MEMORY HIAERARCHY

Access Methods

- Random

  — Individual addresses identify locations exactly

  — Access time is independent of location or previous access

  — e.g. RAM

- Associative

  —Data is located by a comparison with contents of a portion of the store

  — Access time is independent of location or previous access

  — All memory is checked simultaneously; access time is constant

  — e.g. cache

MEMORY HIAERARCHY

Performance

- Cycle Time

  —Primarily applied to RAM; access time + additional time before a second access can start

  —Function of memory components and system bus, not the processor

- Transfer Rate – the rate at which data can be transferred into or out of a memory unit

  — For RAM TR = 1 / (cycle time)