SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

SCHOOL OF BIO AND CHEMICAL ENGINEERING

DEPARTMENT OF BIOMEDICAL ENGINEERING

UNIT – I – Fundamentals of Digital Systems – SBMA1401

I. Number System and Boolean Algebra

Introduction to number systems- Types and Conversions, Binary Arithmetic, Signed Binary Numbers, Binary Codes - BCD, ASCII, Excess-3 codes, Gray codes, Boolean Algebra - De-Morgans Theorem, Reduction of Switching Equations Using Boolean Algebra

1.1 Introduction to number systems

Many number systems are in use in digital technology. The most common are the decimal, binary, octal, and hexadecimal systems. The decimal system is clearly the most familiar to us because it is tools that we use every day.

Types of Number Systems are

- Decimal Number system
- Binary Number system
- Octal Number system
- Hexadecimal Number system

Table: Types of Number Systems

DECIMAL	BINARY	OCTAL	HEXADECIMAL
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

Table: Number system and their Base value

Numbering Systems		
System	Base	Digits
Binary	2	0 1
Octal	8	0 1 2 3 4 5 6 7
Decimal	10	0 1 2 3 4 5 6 7 8 9
Hexadecimal	16	0 1 2 3 4 5 6 7 8 9 A B C D E F

Decimal system: Decimal system is composed of 10 numerals or symbols. These 10 symbols are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9. Using these symbols as digits of a number, we can express any quantity. The decimal system is also called the base-10 system because it has 10 digits. Even though the decimal system has only 10 symbols, any number of any magnitude can be expressed by using our system of positional weighting.

10^3	10^2	10^1	10^0		10^{-1}	10^{-2}	10^{-3}
=1000	=100	=10	=1	.	=0.1	=0.01	=0.001
Most Significant Digit				Decimal point			Least Significant Digit

Example: 3.14_{10} , 52_{10} ,1024_{10}

Binary System: In the binary system, there are only two symbols or possible digit values, 0 and 1. This base-2 system can be used to represent any quantity that can be represented in decimal or other base system.

2^3	2^2	2^1	2^0		2^{-1}	2^{-2}	2^{-3}
=8	=4	=2	=1	.	=0.5	=0.25	=0.125
Most Significant Digit				Binary point			Least Significant Digit

In digital systems the information that is being processed is usually presented in binary form. Binary quantities can be represented by any device that has only two operating states or possible conditions. E.g.. A switch is only open or closed. We arbitrarily (as we define them) let an open switch represent binary 0 and a closed switch represent binary 1. Thus we can represent any binary number by using series of switches.
Binary 1: Any voltage between 2V to 5V Binary 0: Any voltage between 0V to 0.8V
Not used: Voltage between 0.8V to 2V in 5 Volt CMOS and TTL Logic, this may cause error in a digital circuit. Today's digital circuits works at 1.8 volts, so this statement may not hold true for all logic circuits.

Octal System: The octal number system has a base of eight, meaning that it has eight possible digits: 0,1,2,3,4,5,6,7.

8^3	8^2	8^1	8^0		8^{-1}	8^{-2}	8^{-3}
=512	=64	=8	=1	.	=1/8	=1/64	=1/512
Most Significant Digit				Octal point			Least Significant Digit

Hexadecimal System: The hexadecimal system uses base 16. Thus, it has 16 possible digit symbols. It uses the digits 0 through 9 plus the letters A, B, C, D, E, and F as the 16 digit symbols.

16^3	16^2	16^1	16^0		16^{-1}	16^{-2}	16^{-3}
=4096	=256	=16	=1	.	=1/16	=1/256	=1/4096
Most Significant Digit				Hexadecimal point			Least Significant Digit

1.2 Code Conversion

Converting from one code form to another code form is called code conversion, like converting from binary to decimal or converting from hexadecimal to decimal.

- **Binary-To-Decimal Conversion:** Any binary number can be converted to its decimal equivalent simply by summing together the weights of the various positions in the binary number which contain a 1.

Binary	Decimal
11011_2	
$= (1*2^4)+(1*2^3)+0+(1*2^1)+(1*2^0)$	=16+8+0+2+1
Result	27_{10}

- **Decimal to binary Conversion:**

There are 2 methods:

- Reverse of Binary-To-Decimal Method
- Repeat Division

Reverse of Binary-To-Decimal Method

Decimal	Binary
45_{10}	=32 + 0 + 8 + 4 +0 + 1
	$=2^5+0+2^3+2^2+0+2^0$
Result	$=101101_2$

Repeat Division-Convert decimal to binary: This method uses repeated division by 2.

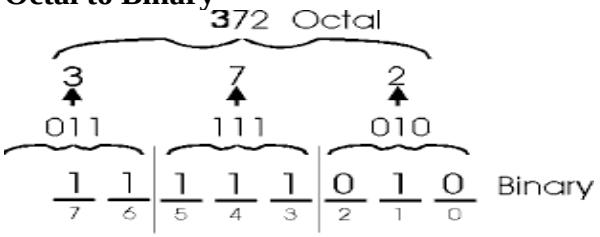
Division	Remainder	Binary
25/2	= 12+ remainder of 1	1 (Least Significant Bit)
12/2	= 6 + remainder of 0	0
6/2	= 3 + remainder of 0	0
3/2	= 1 + remainder of 1	1
1/2	= 0 + remainder of 1	1 (Most Significant Bit)

Result	2510	= 110012

- **Binary-To-Octal / Octal-To-Binary Conversion Binary to octal**

$100\ 111\ 010_2 = (100)\ (111)\ (010)2 = 4\ 7\ 2_8$

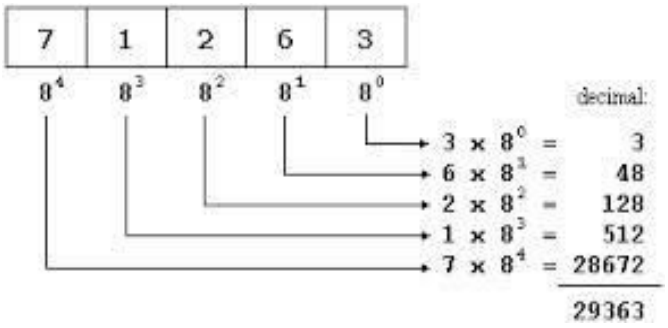
Octal to Binary



- **Decimal -To-Octal / Octal-To- Decimal Conversion Decimal to octal**

Division	Result	Binary
177/8	= 22+ remainder of 1	1 (Least Significant Bit)
22/ 8	= 2 + remainder of 6	6
2 / 8	= 0 + remainder of 2	2 (Most Significant Bit)
Result	17710	= 2618
Binary		= 0101100012

Octal to Decimal



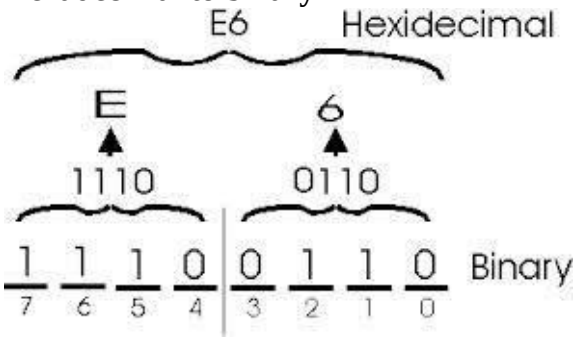
- **Hexadecimal to Decimal/Decimal to Hexadecimal Conversion Decimal to Hexadecimal**

Division	Result	Hexadecimal
378/16	= 23+ remainder of 10	A (Least Significant Bit)23
23/16	= 1 + remainder of 7	7
1/16	= 0 + remainder of 1	1 (Most Significant Bit)
Result	378_{10}	$= 17A_{16}$

- **Binary-To-Hexadecimal /Hexadecimal-To-Binary Conversion**

Binary-To-Hexadecimal: $1011\ 0010\ 1111_2 = (1011)\ (0010)\ (1111)_2 = B\ 2\ F_{16}$

Hexadecimal to binary



- **Octal-To-Hexadecimal Hexadecimal-To-Octal Conversion**
- Convert Octal (Hexadecimal) to Binary first.
- Regroup the binary number by three bits per group starting from LSB if Octal is required.
- Regroup the binary number by four bits per group starting from LSB if Hexadecimal is required.

Octal to Hexadecimal

Octal	Hexadecimal
= 2 6 5 0	
010 110 **101** 000	= **0101** 1010 **1000** (Binary)
Result	$=(5A8)_{16}$

Hexadecimal to octal

Hexadecimal	Octal
$(5A8)_{16}$	= **0101** 1010 **1000** (Binary)
	= **010** 110 **101** 000 (Binary)
Result	= 2 6 5 0 (Octal)

1's and 2's complement

Complements are used in digital computers to simplify the subtraction operation and for logical manipulation. There are TWO types of complements for each base-r system: the radix complement and the diminished radix complement. The first is referred to as the r's complement and the second as the (r - 1)'s complement, when the value of the base r is substituted in the name. The two types are referred to as

The 2's complement and 1's complement for binary numbers and the 10's complement and 9's complement for decimal numbers.

- The 1's complement of a binary number is the number that results when we change all 1's to zeros and the zeros to ones.

- The 2's complement is the binary number that results when we add 1 to the 1's complement. It is used to represent negative numbers.

  2's complement=1's complement+1

Example 1) : Find 1's complement of $(1101)_2$

1 1 0 1 number

0 0 1 0 1's complement

Example 2) : Find 1's complement of $(1001)_2$

1 0 0 1 number

0 1 1 0 1's complement

\+ 1

= 0 1 1 1

1.3 Arithmetic Operations

1.3.1Binary Addition

- **Rules of Binary Addition**
- 0 + 0 = 0
- 0 + 1 = 1
- 1 + 0 = 1
- 1 + 1 = 0, and carry 1 to the next more significant bit

Perform the binary addition

$11101.101_2 + 10010.001_2$

$$\begin{array}{r} 1 \\ 11101.101_2 + \\ 10010.001_2 \\ \hline 101111.110 \end{array}$$

Ans: 101111.110_2

1.3.2 Binary Subtraction

Rules of Binary Subtraction

- 0 - 0 = 0
- 0 - 1 = 1, borrow 1 from the next bit
- 1 - 0 = 1
- 1 - 1 = 0

Perform the binary subtraction

$11101.101_2 - 10010.001_2$

```
      1
   11101.101 -
   10010.001
  ----------
   01011.100
Ans: 01011.100₂
```

1.3.3 Binary Multiplication

Rules of Binary Multiplication

- 0 x 0 = 0
- 0 x 1 = 0
- 1 x 0 = 0
- 1 x 1 = 1, and no carry or borrow bits

Perform the binary multiplication

$111_2 * 11_2$

```
      1 1 1 *
        1 1
  -----------
    1 1        <-- carry
      1 1 1
    1 1 1
  -----------
  1 0 1 0 1₂ : Ans
  -----------
```

1.3.4Binary Division

Perform the binary division

$11110_2 \div 101_2$

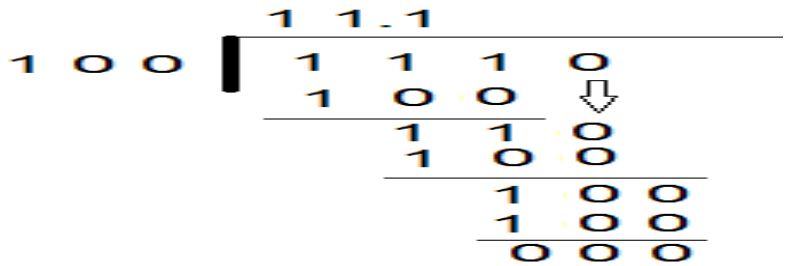
```
        1100   <-- Quotient
  101 | 11101
        101
        ----
         0101
          101
          ---
            001  <-- Remainder
```

$11110_2 \div 101_2 = 1100_2$ (Q), 001_2 (R)

Perform the binary division

$14 \div 4$

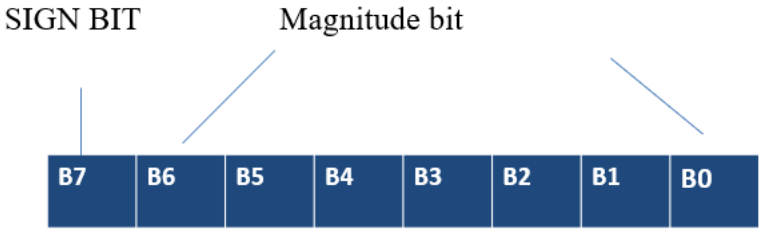
$= 1110_2 \div 100_2$



$1110_2 \div 100_2 = 1\ 1.1_2$

1.4 Signed Binary Numbers

- In computers both positive and negative numbers are represented only with binary digits.
- The left most bit (sign bit in the number represent sign of the number.
- The sign bit is 0 for positive numbers and 1 for negative numbers



- The most significant bit (MSB) represents sign of the number. If MSB is 1, number is negative . If MSB is 0, number is positive.
  Examples
  + 6 = 0000 0110
  - 14 = 1000 1110
  + 24 = 0001 1000
  -64 = 1100 0000

1.5 Binary Codes

Binary codes are codes which are represented in binary system with modification from the original ones. There are two types of binary codes: Weighted codes and Non-Weighted codes. BCD and the 2421 code are examples of weighted codes. In a weighted code, each bit position is assigned a weighting factor in such a way that each digit ca n be evaluated by adding the weight of all the 1's in the coded combination.

(i)Weighted Code

• 8421 code , Most common, Default

• The corresponding decimal digit is determined by adding the weights associated with the 1s in the code group. 62310 = 0110 0010 0011

2421, 5421,7536, etc… codes

- The weights associated with the bits in each code group are given by the name of the code

(ii)Nonweighted Codes

- **2421 code :** This is a weighted code; its weights are 2, 4, 2 and 1. A decimal number is represented in 4-bit form and the total four bits weight is 2 + 4 + 2 + 1 = 9. Hence the 2421 code represents the decimal numbers from 0 to 9.
- **5211 code:** This is a weighted code; its weights are 5, 2, 1 and 1. A decimal number is represented in 4-bit form and the total four bits weight is 5 + 2 + 1 + 1 = 9. Hence the 5211 code represents the decimal numbers from 0 to 9.

(iii) Reflective code : A code is said to be reflective when code for 9 is complement for the code for 0, and so is for 8 and 1 codes, 7 and 2, 6 and 3, 5 and 4. Codes 2421, 5211, and excess-3 are reflective, whereas the 8421 code is not.

(iv) Sequential code : A code is said to be sequential when two subsequent codes, seen as numbers in binary representation, differ by one. This greatly aids mathematical manipulation of data. The 8421 and Excess-3 codes are sequential, whereas the 2421 and 5211 codes are not.

- **Excess-3 code:** Excess-3 is a non weighted code used to express numbers. The code derives its corresponding 8421 code plus 0011(3).

Example: 1000 of 8421 = 1011 in Excess-3

- **Gray code :** The gray code belongs to a class of codes called minimum change codes, in which only one bit in the code changes when moving from one code to the next. The Gray code is non-weighted code, as the position of bit does not contain any weight. In digital Gray code has got a special place.

Decimal Number	Binary Code	Gray Code
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011

14	1110	1001
15	1111	1000

The gray code is a reflective digital code which has the special property that any two subsequent numbers codes differ by only one bit. This is also called a unit-distance code.

Important when an analog quantity must be converted to a digital representation. Only one bit changes between two successive integers which are being coded.

(v)Error Detecting and Correction Codes

- **Error detecting codes :** When data is transmitted from one point to another, like in wireless transmission, or it is just stored, like in hard disks and there are chances that data may get corrupted. To detect these data errors, we use special codes, which are error detection codes.

- **Error correcting code :** Error-correcting codes not only detect errors, but also correct them. This is used normally in Satellite communication, where turn-around delay is very high as is the probability of data getting corrupt.

(vi)Alphanumeric codes

The binary codes that can be used to represent the letters of the alphabet, numbers and mathematical symbols, punctuation marks are known as alphanumeric codes or character codes. These codes enable us to interface the input-output devices like the keyboard, printers, video displays with the computer.

- **ASCII codes :** Codes to handle alphabetic and numeric information, special symbols, punctuation marks, and control characters. ASCII (American Standard Code for Information Interchange) is the best known. Unicode – a 16-bit coding system provides for foreign languages, mathematical symbols, geometrical shapes, dingbats, etc. It has become a world standard alphanumeric code for microcomputers and computers. It is a 7-bit code representing 128 different characters. These characters represe upper case letters (A to Z), 26 lowercase letters (a to z), 10 numbers (0 to 9), 33 special characters and symbols and 33 control characters.

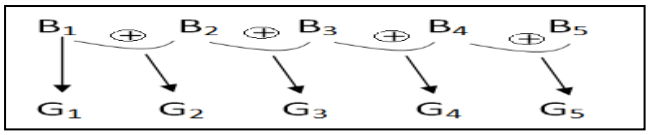
- **EBCDIC codes :** EBCDIC stands for Extended Binary Coded Decimal Interchange. It is mainly used with large computer systems like mainframes. EBCDIC is an 8-bit code and thus accommodates up to 256 characters. An EBCDIC code is divided into two portions: 4 zone bits (on the left) and 4 numeric bits (on the right).

Example 1: Give the binary, BCD, Excess-3, gray code representations of numbers: 5,8,14.

Decimal Number	Binary code	BCD code	Excess-3 code	Gray code
5	0101	0101	1000	0111

8	1000	1000	1011	1100
14	1110	0001 0100	0100 0111	1001

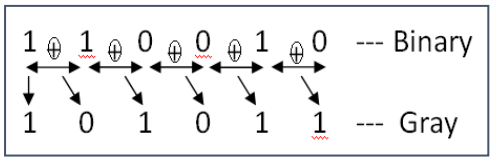
Binary to Gray code conversion



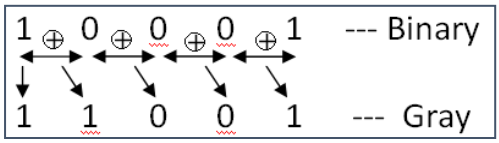
$G_1 = B_1$
$G_2 = B_1 \oplus B_2$
$G_3 = B_2 \oplus B_3$
$G_4 = B_3 \oplus B_4$
$G_n = B_{n-1} \oplus B_n$

Convert the following binary numbers into Gray

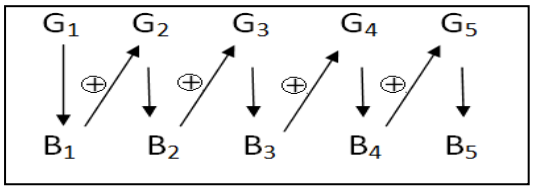
1) 1 1 0 0 1 0



2)1 0 0 0 1



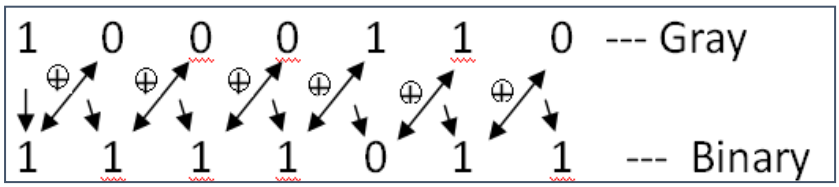
Gray to Binary code conversion



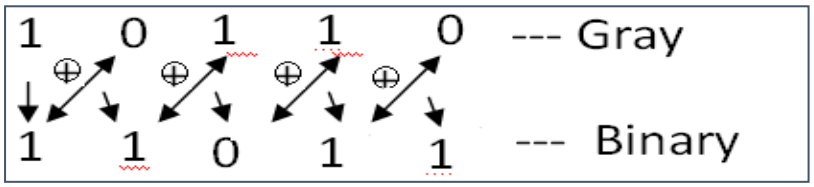
$B_1 = G_1$
$B_2 = B_1 \oplus G_2$
$B_3 = B_2 \oplus G_3$
$B_4 = B_3 \oplus G_4$
$B_n = B_{n-1} \oplus G_n$

Convert the following Gray numbers into binary

1)1 0 0 0 1 1 0



2)1 0 1 1 0



1.5 Boolean Algebra

In 1854, George Boole, an English mathematician, proposed algebra for symbolically representing problems in logic so that they may be analyzed mathematically. The mathematical systems founded upon the work of Boole are called **Boolean algebra** in his honor.

Fundamental postulates of Boolean algebra:

The postulates of a mathematical system forms the basic assumption from which it is possible to deduce the theorems, laws and properties of the system.

The most common postulates used to formulate various structures are

Closure:

A set S is closed w.r.t. a binary operator, if for every pair of elements of S, the binary operator specifies a rule for obtaining a unique element of S.
The result of each operation with operator (+) or (.) is either 1 or 0 and 1, 0 ЄB.

Identity element:
e* x = x * e = x

Eg: 0+ 0 = 0 0+ 1 = 1+ 0 = 1 a) **x+ 0= x**
1 . 1 = 1 1 . 0 = 0 . 1 = 1 b) **x. 1 = x**

Commutative law:

A binary operator * on a set S is said to be commutative if,
x * y = y * x

Eg: 0+ 1 = 1+ 0 = 1 a) **x+ y= y+ x**

0 . 1 = 1 . 0 = 0 b) **x. y= y. x**

Distributive law:

If * and • are two binary operation on a set S, • is said to be distributive over + whenever,

x . (y+ z) = (x. y) + (x. z)

Similarly, + is said to be distributive over • whenever,

x + (y. z) = (x+ y). (x+ z)

Inverse:

a) x+ x' = 1, since 0 + 0' = 0+ 1 and 1+ 1' = 1+ 0 = 1

b) x. x' = 1, since 0 . 0' = 0. 1 and 1. 1' = 1. 0 = 0

Summary:

Postulates of Boolean algebra:

POSTULATES	**(a)**	**(b)**
Postulate 2 (Identity)	**x + 0 = x**	**x . 1 = x**
Postulate 3 (Commutative)	**x+ y = y+ x**	**x . y = y. x**
Postulate 4 (Distributive)	**x (y+ z) = xy+ xz**	**x+ yz = (x+ y). (x+ z)**
Postulate 5 (Inverse)	**x+x' = 1**	**x. x' = 0**

1.6 Basic theorem and properties of Boolean algebra:

Basic Theorems:

The theorems, like the postulates are listed in pairs; each relation is the dual of the one paired with it. The postulates are basic axioms of the algebraic structure and need no proof. The theorems must be proven from the postulates. The proofs of the theorems with one variable are presented below. At the right is listed the number of the postulate that justifies each step of the proof.

1a) **x+ x = x**
1b) x. x = x
2) **x .0 = 0**
3) (x')' = x
Absorption Theorem:
x+ xy = x

x+ xy = x. 1 + xy ------- by postulate 2(b) [x. 1 = x]

= x (1+ y) ------- 4(a) [x (y+z) = (xy)+ (xz)]

= x (1) ------- by theorem 2(a [x+ 1 = x]

= x. ------- by postulate 2(a)[x. 1 = x]

x. (x+ y) = x

x. (x+ y) = x. x+ x. y --------- 4(a) [x (y+z) = (xy)+ (xz)]

= x + x.y ---------- by theorem 1(b) [x. x = x]

= x. ---------- by theorem 4(a) [x+ xy = x]

x+ x'y = x+ y

x+ x'y = x+ xy+ x'y ---------- by theorem 4(a) [x+ xy = x]

= x+ y (x+ x') ----------by postulate 4(a) [x (y+z) = (xy)+ (xz)]

= x+ y (1) ---------- 5(a)[x+ x' = 1]

= x+ y ---------- 2(b)[x. 1= x]

x. (x'+y) = xy

x. (x'+y) = x.x'+ xy --------- by postulate 4(a) [x (y+z) = (xy)+ (xz)]

= 0+ xy ----------5(b) [x. x' = 0]

= xy. ----------2(a) [x+ 0= x]

<u>Properties of Boolean algebra:</u>

<u>Commutative property</u>:

Boolean addition is commutative, given by

x+ y = y+ x

According to this property, the order of the OR operation conducted on the variables makes no difference.

Boolean algebra is also commutative over multiplication given by,

x. y = y. x

This means that the order of the AND operation conducted on the variables makes no difference.

<u>Associative property</u>:

The associative property of addition is given by,

A+ (B+ C) = (A+B) + C

The OR operation of several variables results in the same, regardless of the grouping of the variables.

The associative law of multiplication is given by,

A. (B. C) = (A.B) . C

It makes no difference in what order the variables are grouped during the AND operation of several variables.

Distributive property:

The Boolean addition is distributive over Boolean multiplication, given by

A+ BC = (A+B) (A+C)

The Boolean addition is distributive over Boolean addition, given by

A. (B+C) = (A.B)+ (A.C)

Duality:

It states that every algebraic expression deducible from the postulates of Boolean algebra remains valid if the operators and identity elements are interchanged.

If the dual of an algebraic expression is desired, we simply interchange OR and AND operators and replace 1's by 0's and 0's by 1's.

x+ x' = 1 is **x. x' = 0**

Duality is a very important property of Boolean algebra.

Summary:

Theorems of Boolean algebra:

	THEOREMS	(a)	(b)
1	Idempotent	**x + x = x**	**x . x = x**
		x + 1 = 1	**x . 0 = 0**
2	Involution	**(x')' = x**	
3	Absorption	**x+ xy = x**	**x (x+ y) = x**
		x+ x'y = x+ y	**x. (x'+ y)= xy**
4	Associative	**x+(y+ z)= (x+ y)+ z**	**x (yz) = (xy) z**
5	DeMorgan's Theorem	**(x+ y)'= x'. y'**	**(x. y)'= x'+ y'**

1.7 DeMorgan's Theorems:

Two theorems that are an important part of Boolean algebra were proposed by DeMorgan.

The first theorem states that the complement of a product is equal to the sum of the complements.

(AB)' = A'+ B'

The second theorem states that the complement of a sum is equal to the product of the complements.

(A+ B)' = A'. B'

Consensus Theorem:

In simplification of Boolean expression, an expression of the form **AB+ A'C+ BC**, the term BC is redundant and can be eliminated to form the equivalent expression AB+ A'C. The theorem used for this simplification is known as consensus theorem and is stated as,

AB+ A'C+ BC = AB+ A'C

The dual form of consensus theorem is stated as,

(A+B) (A'+C) (B+C) = (A+B) (A'+C)

1.8 Minimization of Boolean Expressions:

The Boolean expressions can be simplified by applying properties, laws and theorems of Boolean algebra.

Simplify the following Boolean functions to a minimum number of literals:

1. x (x'+y)
= xx'+ xy **[x. x'= 0]**
= 0 + xy **[x+ 0 = x]**
= xy.

2. x+ x'y
= x + xy + x'y **[x+ xy= x]**
= x+ y (x+x')
= x+ y (1) **[x+ x' = 1]**
= x+ y.

3. (x+ y) (x'+ z) (y+ z)
= (x+ y) (x'+ z) **[** dual form of consensus theorem**,**
(A+ B) (A'+ C) (B+ C) = (A+ B) (A'+ C)]

4. x'y+ xy+ x'y'
= y (x'+ x) + x'y' **[x (y+ z) = xy+ xz]**
= y (1) + x'y' **[x+ x' = 1]**
= y+ x'y' **[x+ x'y' = x+ y']**
= y+ x'.
5. x+ xy'+ x'y
= x (1+ y')+ x'y
= x (1) + x'y **[1+ x = 1]**
= x+ x'y **[x+ x'y = x+ y]**
= x+ y.

6. AB + (AC)' + AB'C (AB + C)
= AB + (AC)' + AAB'BC + AB'CC
= AB + (AC)' + 0+ AB'CC **[B.B' = 0]**
= AB + (AC)' + AB'C **[C.C = 1]**
= AB + A' + C' +AB'C **[(AC)' = A' + C']**
= AB + A' + C' + AB' **[C' + AB'C = C' + AB']**
= A' + B+ C'+ AB' **[A' + AB = A' + B]**
Re- arranging,
= A' + AB'+ B+ C' **[A' + AB = A' + B]**
= A' + B'+ B+ C' **[B'+ B= 1]**

= A' +1+ C' **[A+ 1= 1]**
= 1

7. (x'+ y) (x+ y)
= x'.x+ x'y+ yx+ y.y
= 0+ x'y+ xy+ y **[x.x'= 0]; [x. x= x]**
= y (x'+ x+ 1)
= y(1) **[1+ x = 1]**
= y.

8. x'yz+ xy'z'+ x'y'z'+ xy'z+ xyz
= yz (x'+x) + xy'z'+ x'y'z'+ xy'z
= yz (1) + y'z' (x+ x') **[x+ x'=1]**
= yz+ y'z' (1) + xy'z **[x+ x'=1]**
= yz+ y'z'+ xy'z
= yz+ y' (z'+ xz)
= yz+ y' (z'+ x) **[x'+ xy = x'+ y]**
= yz+ y'z'+ xy'

9. [(xy)'+ x'+ xy]'
= [x'+ y'+ x'+ xy]'
= [x'+ y'+ xy]' **[x+ x= x]**
= [x'+ y'+ x]' **[x'+ xy = x'+ y]**
= [y'+ 1]' **[x+ x'= 1]**
= [1]' **[1+ x = 1]**
= 0.

10. [xy+ xz]'+ x'y'z
= (xy)'. (xz)'+ x'y'z
= (x'+ y'). (x'+ z')+ x'y'z
= x'x'+ x'z'+ x'y'+ y'z'+ x'y'z
= x'+ x'z'+ x'y'+ y'z'+ x'y'z **[x+ x= x]**
= x'+ x'z'+ x'y'+ y' [z'+ x'z]
= x'+ x'z'+ x'y'+ y' [z'+ x'] **[x'+ xy = x'+ y]**
= x'+ x'y'+ y' [z'+ x'] **[x+ xy = x]**
= x'+ x'y'+ y'z'+ x'y'
= x'+ y'z'+ x'y' **[x+ xy = x]**
= x'+ y'z'. **[x+ xy = x]**

11. xy+ xy'(x'z')'
= xy+ xy' (x''+ z'')
= xy+ xy' (x+ z) **[x'' = x]**
= xy+ xy'x+ xy'z
= xy+ xy'+ xy'z **[x. x= x]**
= xy+ xy' [1+ z]

= xy+ xy' [1] **[1+ x = 1]**
= xy+ xy'
= x(y+ y')
= x [1] **[x+ x'= 1]**
= x.

12. [(xy'+ xyz)'+ x (y+ xy')]'
= [x(y'+yz)'+ x (y+ xy')]'
= [x(y'+z)'+ x (y+ x)]' **[x'+ xy = x'+ y]; [x+ x'y = x+ y]**
= [x(y'+z)'+ xy+ x.x)]'
= [(xy'+xz)'+ xy+ x)]' **[x. x= x]**
= [(xy'+xz)'+ x)]' **[x+ xy = x]**
= [(xy')'. (xz)'+ x]'
= [(x'+y''). (x'+z')+ x]'
= [(x'+y). (x'+z')+ x]' **[x'' = x]**
= [(x'+yz')+ x]' **[(x+ y) (x+ z)= x+ yz]**
= [x'+ yz'+ x]'
= [1+ yz']' **[x+ x'= 1]**
= [1]' **[1+ x = 1]**
= 0.

COMPLEMENT OF A FUNCTION:

The complement of a function F is F' and is obtained from an interchange of 0's for 1's and 1's for 0's in the value of F. The complement of a function may be derived algebraically through DeMorgan's theorem.

DeMorgan's theorems for any number of variables resemble in form the two- variable case and can be derived by successive substitutions similar to the method used in the preceding derivation. These theorems can be generalized as –

(A+ B+ C+ D+ … + F)' = A' B' C' D' … F'

(A B C D … F)' = A'+B'+ C'+ D'+ … +F'.

Find the complement of the following functions,

1. **F= x'yz'+ x'y'z**

F'= (x'yz'+ x'y'z)'
= (x''+ y'+ z'') . (x''+ y''+z')
= (x+ y'+ z). (x+ y+ z').

2. **F= (xy + y'z + xz) x.**

F' = [(xy + y'z + xz) x]'
= (xy + y'z + xz)' + x'
= [(xy)' . (y'z)'. (xz)'] + x'
= [(x'+y'). (y+z'). (x'+z')] + x'

= [(x'y+ x'z'+ 0+ y'z') (x'+z')] + x'

= x'x'y+ x'x'z'+ x'y'z'+ x'yz'+ x'z'z'+ y'z'z'+ x'

= x'y+ x'z'+ x'y'z'+ x'yz'+ x'z'+ y'z'+ x' [x+ x = x], [x. x =x]

= x'y+ x'z'+ x'z' (y'+ y) + y'z'+ x' [x+ x'= 1]

= x'y+ x'z'+ x'z' (1) + y'z'+ x'

= x'y+ x'z'+ y'z'+ x'

= x'y+ x'+ x'z'+ y'z'

= x'(y+1) + x'z+ y'z' [y+1= 1]

= x' (1+z) + y'z' [y+1= 1]

= x'+ y'z'

3. F= x (y'z'+ yz)

F'= [x (y'z'+yz)]'

= x'+ (y'z'+ yz)'

= x'+ (y'z')'. (yz)'

= x'+ (y"+ z") . (y'+ z')

= x'+ (y+ z) . (y'+ z').

4. F= xy'+ x'y

F'= (xy'+ x'y)'

= (xy')'. (x'y)'

= (x'+y) (x+y')

= x'x+ x'y'+ yx+ yy'

= x'y'+ xy.

5. f = wx'y + xy'+ wxz

f' = (wx'y + xy'+ wxz)'

= (wx'y)' (xy')' (wxz)'

= (w'+x+ y') (x'+ y) (w'+ x'+ z')

= (w'x'+ w'y+ xx'+ xy+ x'y'+ yy') (w'+ x'+ z')

= (w'x'+ w'y+ xy+ x'y') (w'+ x'+ z')

= w'x'. w'+ w'y. w'+ xy. w'+ x'y'. w'+ w'x'. x'+w'y. x'+ xy. x'+ x'y'. x'+ w'x'. z'+ w'y. z'+ xy. z'+ x'y'.z'

= w'x'+ w'y+ w'xy+ w'x'y'+ w'x'+ w'x'y+ 0 + x'y'+ w'x'z'+ w'yz'+ xyz'+ x'y'z'

= w'x'+ w'y+ w'xy+ w'x'y'+ w'x'y+ x'y'+ w'x'z'+ w'yz'+ xyz'+ x'y'z'

= w'x'(1+ y'+ y+ z')+ w'y(1+ x+ z')+ x'y'(1+ z')+ xyz'

= w'x'(1)+ w'y(1)+ x'y'(1)+ xyz'

= w'x'+ w'y+ x'y'+ xyz'

REFERENCES

1.Morris Mano, Digital Design, Prentice Hall of India, 2001.

2. S.Salivahanan and S.Arivazhagan―Digital Electronics‖, Ist Edition, Vikas Publishing House pvt Ltd, 2012.

3.Ronald J.Tocci, Digital System Principles and Applications, PHI, 6th Edition, 1997

Question bank

Part A

1. Solve A(A+B).
2. Analyze the octal and hexadecimal equivalent of (537)10.
3. Point out the gray code for the binary $(101101101)_2$ and $(1010111000)_2$.
4. Convert $(215)_{10}$ and $(235)_{10}$ to hexadecimal numbers.
5. State the Demorgans Theorem
6. Deduce: $(475.25)_8$ to its decimal equivalent and $(549.B4)_{16}$ to its binary equivalent
7. State and prove the Consensus theorem
8. Prove the following using DeMorgan's Theorem
   [(X + Y)'+(X + Y) '] '=X + Y.
9. Simplify Z = (AB +C) (B'D + C'E')+(AB+C)'
10. Find the octal equivalent of hexadecimal numbers of ABC.DE

Part B

1. Deduce $(FACE)_{16}$ in its binary, octal and decimal equivalent.
2. Briefly explain about the Binary Codes
3. Implement the various number system used in digital circuits with examples.
4. Evaluate: $(ABCD.1234)_{16} = (?)_8$
   $= (?)_{10}$
   $= (?)_2$

5.Solve by perfect induction
(a) A+AB = A
(b) A(A+B) = A
(c) A+A'B = A+B and
(d) A(A'+B) =AB

SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

(DEEMED TO BE UNIVERSITY)

Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE

www.sathyabama.ac.in

SCHOOL OF BIO AND CHEMICAL ENGINEERING

DEPARTMENT OF BIOMEDICAL ENGINEERING

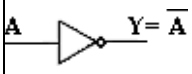
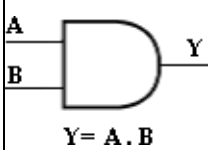
II. Logic Design and Minimisation Techniques

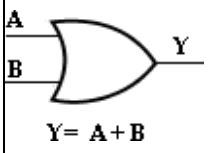
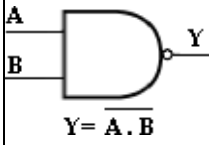
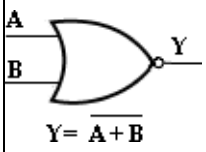
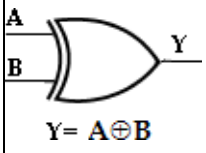
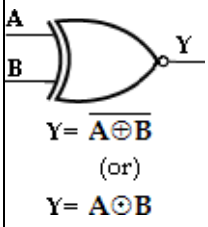
Introduction to logic gates- Design of two level gate network-Two level NAND-NAND and NOR-NOR networks, Universal property of NAND and NOR gates, Standard forms of Boolean equation –Minimization of SOP and POS- Karnaugh maps-Advantages and Limitations-Quine-Mclusky Methods.

2.1 Introduction to logic gates:

Logic gates are electronic circuits that can be used to implement the most elementary logic expressions, also known as Boolean expressions. The logic gate is the most basic building block of combinational logic.

There are three basic logic gates, namely the OR gate, the AND gate and the NOT gate. Other logic gates that are derived from these basic gates are the NAND gate, the NOR gate, the EXCLUSIVE- OR gate and the EXCLUSIVE-NOR gate.

<table>
<tr><th>GATE</th><th>SYMBOL</th><th>OPERATION</th><th>TRUTH TABLE</th></tr>
<tr><td>NOT
(7404)</td><td> 
</td><td>NOT gate (Invertion), produces an inverted output pulse for a given input pulse.</td><td>
<table>
<tr><th>A</th><th>Y= $\overline{A}$</th></tr>
<tr><td>0</td><td>1</td></tr>
<tr><td>1</td><td>0</td></tr>
</table>
</td></tr>
<tr><td>AND
(7408)</td><td> 
</td><td>AND gate performs logical multiplication. The output is HIGH only when all the inputs are HIGH. When any of the inputs are low, the output is LOW.</td><td>
<table>
<tr><th>A</th><th>B</th><th>Y= A . B</th></tr>
<tr><td>0</td><td>0</td><td>0</td></tr>
<tr><td>0</td><td>1</td><td>0</td></tr>
<tr><td>1</td><td>0</td><td>0</td></tr>
<tr><td>1</td><td>1</td><td>1</td></tr>
</table>
</td></tr>
</table>

<table>
<tr>
<td>OR
(7432)</td>
<td> 
</td>
<td>OR gate performs logical addition. It produces a HIGH on the output when any of the inputs are HIGH. The output is LOW only when all inputs are LOW.</td>
<td>
<table>
<tr><th>A</th><th>B</th><th>Y= A+B</th></tr>
<tr><td>0</td><td>0</td><td>0</td></tr>
<tr><td>0</td><td>1</td><td>1</td></tr>
<tr><td>1</td><td>0</td><td>1</td></tr>
<tr><td>1</td><td>1</td><td>1</td></tr>
</table>
</td>
</tr>
<tr>
<td>NAND
(7400)</td>
<td> 
</td>
<td>It is a universal gate. When any of the inputs are LOW, the output will be HIGH. LOW output occurs only when all inputs are HIGH.</td>
<td>
<table>
<tr><th>A</th><th>B</th><th>Y= $\overline{A.B}$</th></tr>
<tr><td>0</td><td>0</td><td>1</td></tr>
<tr><td>0</td><td>1</td><td>1</td></tr>
<tr><td>1</td><td>0</td><td>1</td></tr>
<tr><td>1</td><td>1</td><td>0</td></tr>
</table>
</td>
</tr>
<tr>
<td>NOR
(7402)</td>
<td> 
</td>
<td>It is a universal gate. LOW output occurs when any of its input is HIGH. When all its inputs are LOW, the output is HIGH.</td>
<td>
<table>
<tr><th>A</th><th>B</th><th>Y= $\overline{A+B}$</th></tr>
<tr><td>0</td><td>0</td><td>1</td></tr>
<tr><td>0</td><td>1</td><td>0</td></tr>
<tr><td>1</td><td>0</td><td>0</td></tr>
<tr><td>1</td><td>1</td><td>0</td></tr>
</table>
</td>
</tr>
<tr>
<td>EX- OR
(7486)</td>
<td> 
</td>
<td>The output is HIGH only when odd number of inputs is HIGH.</td>
<td>
<table>
<tr><th>A</th><th>B</th><th>Y= A⊕B</th></tr>
<tr><td>0</td><td>0</td><td>0</td></tr>
<tr><td>0</td><td>1</td><td>1</td></tr>
<tr><td>1</td><td>0</td><td>1</td></tr>
<tr><td>1</td><td>1</td><td>0</td></tr>
</table>
</td>
</tr>
<tr>
<td>EX- NOR</td>
<td> 
</td>
<td>The output is HIGH only when even number of inputs is HIGH. Or when all inputs are zeros.</td>
<td>
<table>
<tr><th>A</th><th>B</th><th>Y= A⊙B</th></tr>
<tr><td>0</td><td>0</td><td>1</td></tr>
<tr><td>0</td><td>1</td><td>0</td></tr>
<tr><td>1</td><td>0</td><td>0</td></tr>
<tr><td>1</td><td>1</td><td>1</td></tr>
</table>
</td>
</tr>
</table>

2.2 Universal property of NAND and NOR gates:

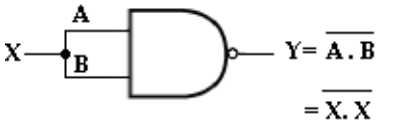
The NAND and NOR gates are known as universal gates, since any logic function can be implemented using NAND or NOR gates. This is illustrated in the following sections.

- **<u>NAND Gate</u>**:

The NAND gate can be used to generate the NOT function, the AND function,

the OR function and the NOR function.

- NOT function:



$= \overline{X.X}$

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

$= X$

	A	B	$Y = \overline{A.B}$	
X= 0	0	0	1	Y= 1
	0	1	1	
	1	0	1	
X= 1	1	1	0	Y= 0

By connecting all the inputs together and creating a single common input.

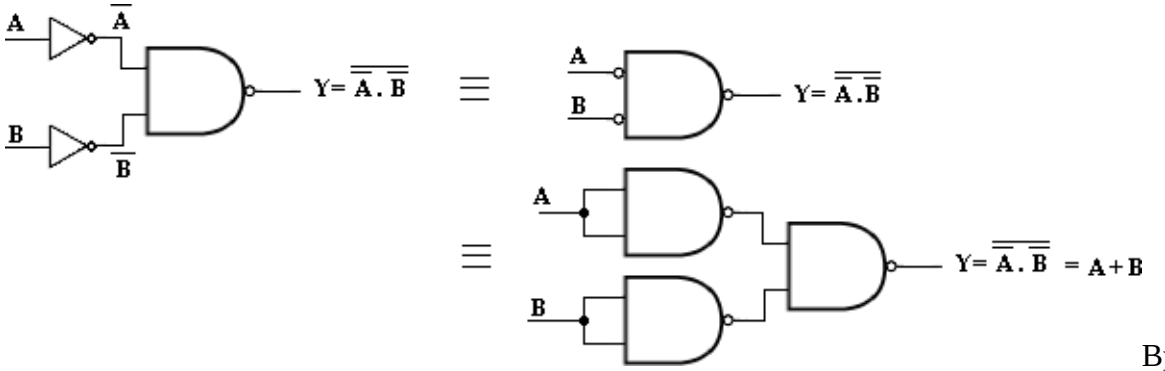
NOT function using NAND gate

- AND function:

By simply inverting output of the NAND gate. i.e.,

AND function using NAND gates

- OR function:



By simply inverting inputs of the NAND gate. i.e.,

OR function using NAND gates

Bubble at the input of NAND gate indicates inverted input.

A	B	Y= A+B
0	0	0
0	1	1
1	0	1
1	1	1

≡

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

- NOR function**:**

By inverting inputs and outputs of the NAND gate.

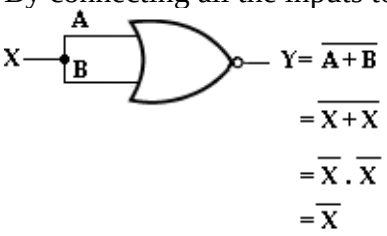
NOR function using NAND gates

- **NOR Gate**:

Similar to NAND gate, the NOR gate is also a universal gate, since it can be used to generate the NOT, AND, OR and NAND functions.

- NOT function:

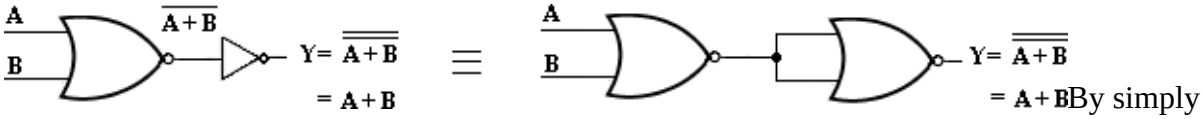
By connecting all the inputs together and creating a single common input.



	A	B	Y= $\overline{A+B}$	
X= 0	0	0	1	Y= 1
	0	1	0	
	1	0	0	
X= 1	1	1	0	Y= 0

NOT function using NOR gates

- OR function:



By simply inverting output of the NOR gate. i.e.,

OR function using NOR gates

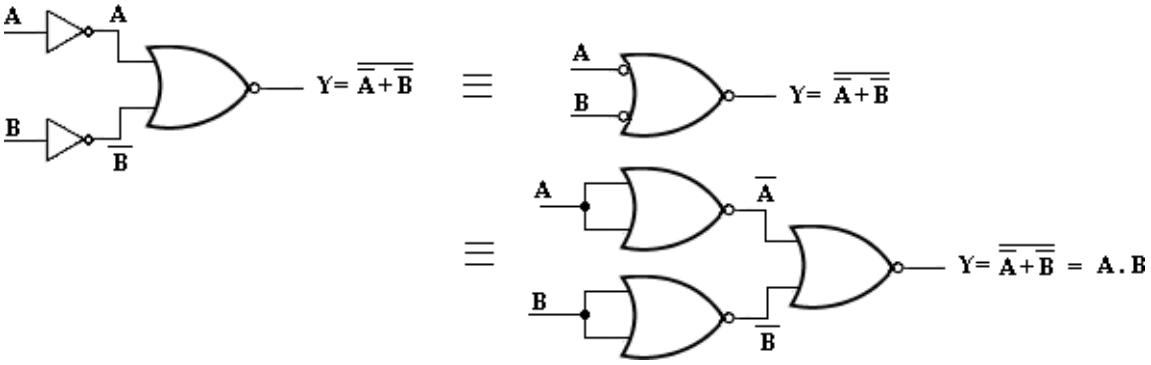
A	B	Y= A+B
0	0	0
0	1	1
1	0	1
1	1	1

≡

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

- AND function:

By simply inverting inputs of the NOR gate. i.e.,



AND function using NOR gates

Bubble at the input of NOR gate indicates inverted input.

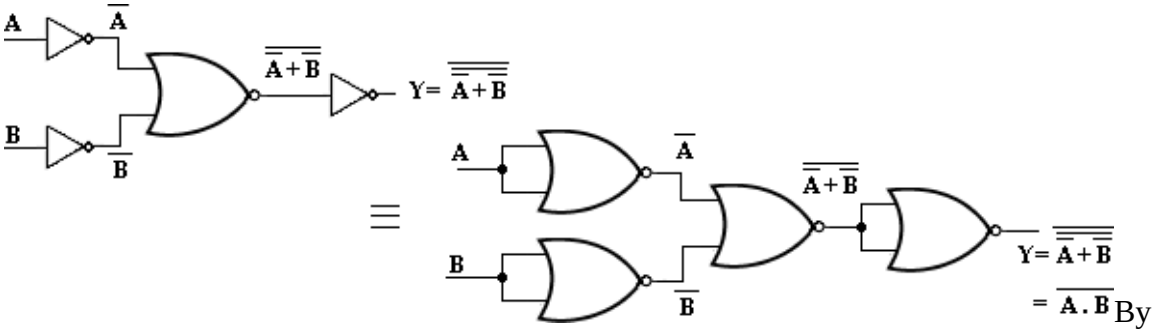
A	B	Y= A . B
0	0	0
0	1	0
1	0	0
1	1	1

≡

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

Truth table

- NAND Function:



By inverting inputs and outputs of the NOR gate.

NAND function using NOR gates

A	B	Y= $\overline{A.B}$
0	0	1
0	1	1
1	0	1
1	1	0

≡

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

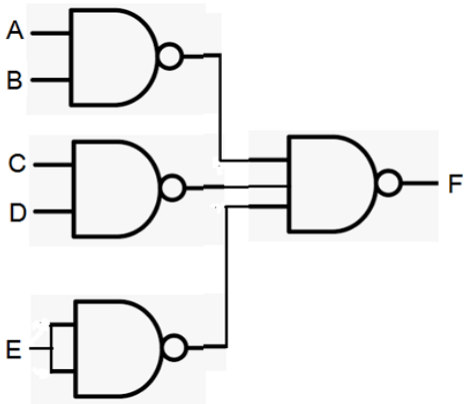
2.3 Design of two level gate network

- **The SOP can be implemented using NAND – NAND logic**
  1. Each product term is connected to NAND gates in level 1
  2. One NAND is connected in the second level 2

- **The POS can be implemented using NOR – NOR logic**
  1. Each sum term is connected to NOR gates in level 1
  2. One NOR is connected in the second level 2

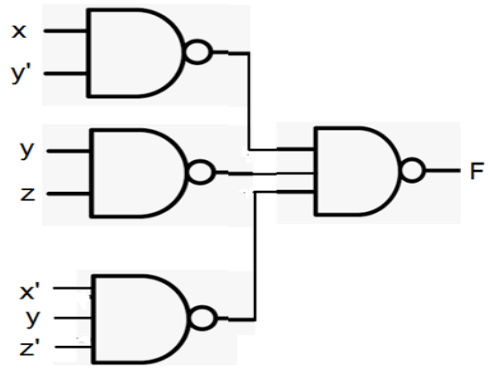
Implement Using NAND – NAND logic

$F = A.B + C.D + E$



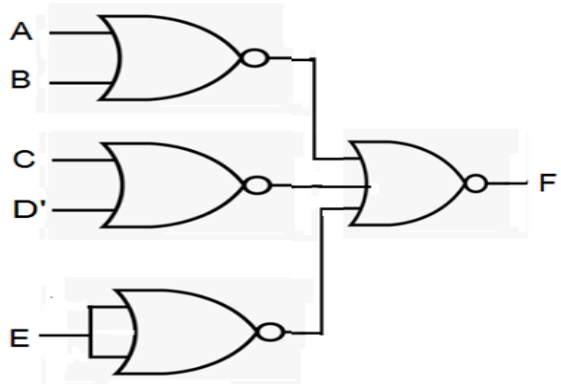
$F = (A+B)\,(C+D')\,E$

$F = x\,y' + y\,z + x'\,y\,z'$

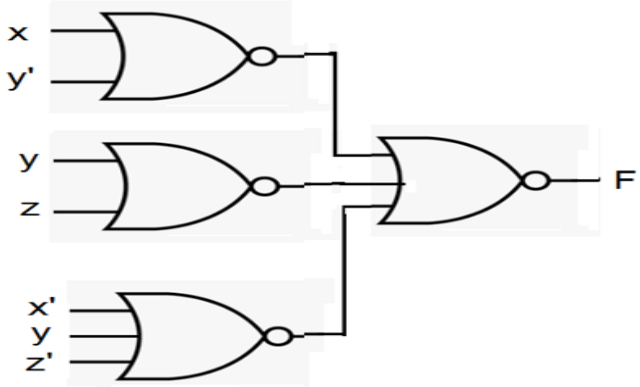


Implement Using NOR – NOR logic

$$F = (A+B)\,(C+D')\,E$$



$$F = (x + y')\,(y + z)\,(x' + y + z')$$



2.4 Standard forms of Boolean equation

CANONICAL AND STANDARD FORMS:

Minterms and Maxterms:

A binary variable may appear either in its normal form (x) or in its complement form (x'). Now either two binary variables x and y combined with an AND operation. Since each variable may appear in either form, there are four possible combinations:

x'y', x'y, xy' and xy Each of these four AND terms is called a '**minterm**'.

In a similar fashion, when two binary variables x and y combined with an OR operation, there are four possible combinations: x'+ y', x'+ y, x+ y' and x+ y

Each of these four OR terms is called a '**maxterm**'.

The minterms and maxterms of a 3- variable function can be represented as in table below.

Variables			**Minterms**	**Maxterms**
x	**y**	**Z**	**m_i**	**M_i**
0	0	0	x'y'z' = m0	x+ y+ z= M_0
0	0	1	x'y'z = m1	x+ y+ z'= M1
0	1	0	x'yz' = m2	x+ y'+ z= M2
0	1	1	x'yz = m3	x+ y'+ z'= M3
1	0	0	xy'z' = m4	x'+ y+ z= M_4
1	0	1	xy'z = m5	x'+ y+ z'= M5
1	1	0	xyz' = m6	x'+ y'+ z= M6
1	1	1	xyz = m_7	x'+ y'+ z'= M7

<u>Sum of Minterm</u>: (Sum of Products)

1. Y= AB+ BC+ AC

2. Y= AB+ $\overline{B}$ C+ A$\overline{C}$

Sum

Product terms

The logical sum of two or more logical product terms is called sum of products expression. It is logically an OR operation of AND operated variables such as:

<u>Sum of Maxterm</u>: (Product of Sums)

1. Y= (A+B). (B+C). (A+C)

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

Product

Sum terms

A product of sums expression is a logical product of two or more logical sum terms. It is basically an AND operation of OR operated variables such as,

<u>Canonical Sum of product expression</u>:

If each term in SOP form contains all the literals then the SOP is known as standard (or) canonical SOP form. Each individual term in standard SOP form is called minterm canonical form.

F (A, B, C) = AB'C+ ABC+ ABC'

<u>Steps to convert general SOP to standard SOP form</u>:

1) Find the missing literals in each product term if any.
2) AND each product term having missing literals by ORing the literal and its complement.
3) Expand the term by applying distributive law and reorder the literals in the product term.
4) Reduce the expression by omitting repeated product terms if any.

Obtain the canonical SOP form of the function:

1) Y(A, B) = A+ B

= A. (B+ B')+ B (A+ A')

= $\underline{AB}$+ AB'+ $\underline{AB}$+ A'B

= AB+ AB'+ A'B.

2) Y (A, B, C) = A+ ABC

= A. (B+ B'). (C+ C')+ ABC

= (AB+ AB'). (C+ C')+ ABC

= $\underline{ABC}$+ ABC'+ AB'C+ AB'C'+ $\underline{ABC}$

= ABC+ ABC'+ AB'C+ AB'C'

= m_7+ m_6+ m_5+ m_4

= ∑m (4, 5, 6, 7).

3) Y (A, B, C) = A+ BC

= A. (B+ B'). (C+ C')+(A+ A'). BC

= (AB+ AB'). (C+ C')+ ABC+ A'BC

= $\underline{ABC}$+ ABC'+ AB'C+ AB'C'+ $\underline{ABC}$+ A'BC

= ABC+ ABC'+ AB'C+ AB'C'+ A'BC

= m_7+ m_6+ m_5+ m_4+ m_3

= ∑m (3, 4, 5, 6, 7).

4) Y (A, B, C) = AC+ AB+ BC

= AC (B+ B')+ AB (C+ C')+ BC (A+ A')

= $\underline{ABC}$+ AB'C+ $\underline{ABC}$+ ABC'+ $\underline{ABC}$+ A'BC

= ABC+ AB'C+ ABC'+ A'BC

= ∑m (3, 5, 6, 7).

5) Y (A, B, C, D) = AB+ ACD

= AB (C+ C') (D+ D') + ACD (B+ B')

= (ABC+ ABC') (D+ D') + ABCD+ AB'CD

= $\underline{ABCD}$+ ABCD'+ ABC'D+ ABC'D'+ $\underline{ABCD}$+ AB'CD

= ABCD+ ABCD'+ ABC'D+ ABC'D'+ AB'CD.

Canonical Product of sum expression:

If each term in POS form contains all literals then the POS is known as standard (or) Canonical POS form. Each individual term in standard POS form is called Maxterm canonical form.

- F (A, B, C) = (A+ B+ C). (A+ B'+ C). (A+ B+ C')
- F (x, y, z) = (x+ y'+ z'). (x'+ y+ z). (x+ y+ z)

Steps to convert general POS to standard POS form:

1) Find the missing literals in each sum term if any.
2) OR each sum term having missing literals by ANDing the literal and its complement.
3) Expand the term by applying distributive law and reorder the literals in the sum term.
4) Reduce the expression by omitting repeated sum terms if any.

Obtain the canonical POS expression of the functions:

1. Y= A+ B'C

= (A+ B') (A+ C) [**A+ BC = (A+B) (A+C)**]
= (A+ B'+ C.C') (A+ C+ B.B')
= (A+ B'+C) (A+ B'+C') (A+ B+ C) (A+ B'+ C)
= (A+ B'+C). (A+ B'+C'). (A+ B+ C)
= M_2. M_3. M_0
= ∏M (0, 2, 3)

2. Y= (A+B) (B+C) (A+C)

= (A+B+ C.C') (B+ C+ A.A') (A+C+B.B')
= (A+B+C) (A+B+C') (A+B+C) (A'+B+C) (A+B+C) (A+B'+C)
= (A+B+C) (A+B+C') (A'+B+C) (A+B'+C)
= M_0. M_1. M_4. M_2
= ∏M (0, 1, 2, 4)

3. Y= A. (B+ C+ A)

= (A+ B.B'+ C.C'). (A+ B+ C)
= (A+B+C) (A+B+C') (A+B'+C) (A+ B'+C') (A+B+C)
= (A+B+C) (A+B+C') (A+B'+C) (A+ B'+C')
= M_0. M_1. M_2. M_3
= ∏M (0, 1, 2, 3)

4. Y= (A+B') (B+C) (A+C')

= (A+B'+C.C') (B+C+ A.A') (A+C'+ B.B')
= (A+B'+C) (A+B'+C') (A+B+C) (A'+B+C) (A+B+C') (A+B'+C')
= (A+B'+C) (A+B'+C') (A+B+C) (A'+B+C) (A+B+C')
= M_2. M_3. M_0. M_4. M_1
= ∏M (0, 1, 2, 3, 4)

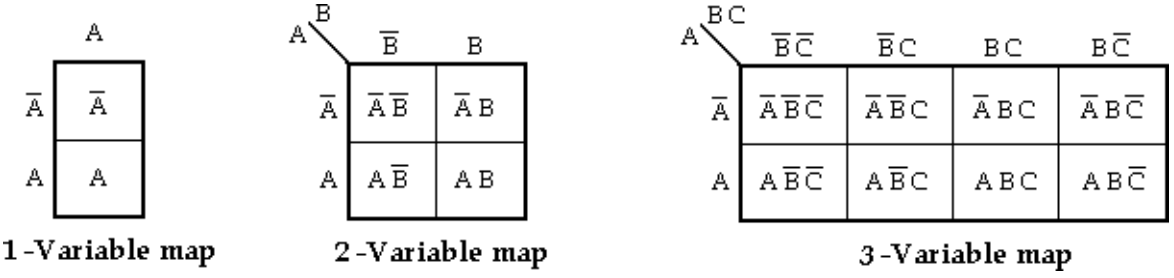
2.5 KARNAUGH MAP MINIMIZATION:

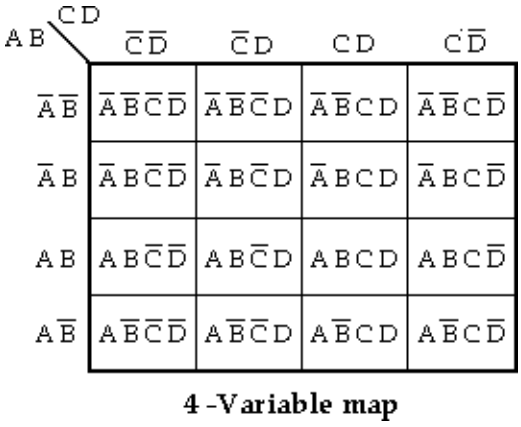
The simplification of the functions using Boolean laws and theorems becomes complex with the increase in the number of variables and terms. The map method, first proposed by Veitch and slightly improvised by Karnaugh, provides a simple, straightforward procedure for the simplification of Boolean functions. The method is called **Veitch diagram** or **Karnaugh map**, which may be regarded as a pictorial representation of a truth table.

The Karnaugh map technique provides a systematic method for simplifying and manipulation of Boolean expressions. **A** K-map is a diagram made up of squares, with each square representing one minterm of the function that is to be minimized. For n variables on a Karnaugh map there are 2^n numbers of squares. Each square or cell represents one of the minterms. It can be drawn directly from either minterm (sum-of- products) or maxterm (product-of-sums) Boolean expressions.

Two- Variable, Three Variable and Four Variable Maps

Karnaugh maps can be used for expressions with two, three, four and five variables. The number of cells in a Karnaugh map is equal to the total number of possible input variable combinations as is the number of rows in a truth table. For three variables, the number of cells is $2^3 = 8$. For four variables, the number of cells is 24 =16.



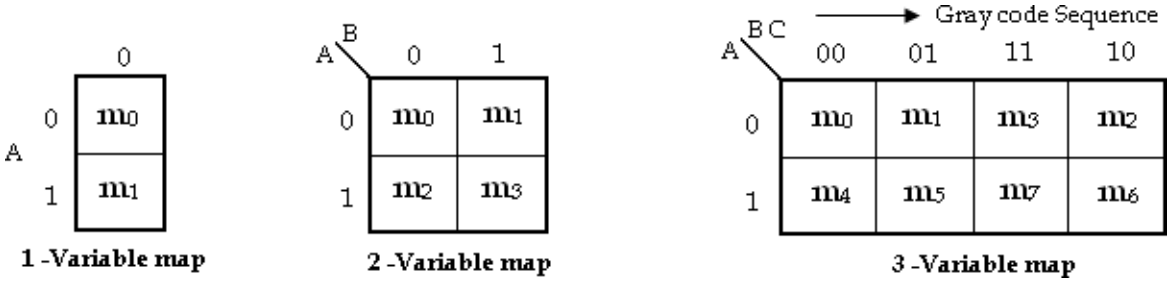


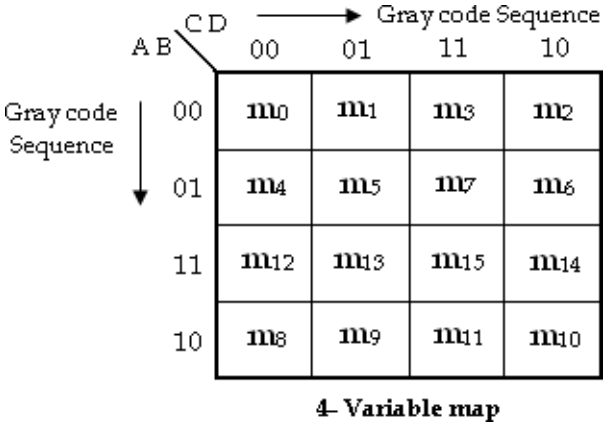
4 -Variable map

Product terms are assigned to the cells of a K-map by labeling each row and each column of a map with a variable, with its complement or with a combination of variables & complements. The below figure shows the way to label the rows & columns of a 1, 2, 3 and 4- variable maps and the product terms corresponding to each cell.

It is important to note that when we move from one cell to the next along any row or from one cell to the next along any column, one and only one variable in the product term changes (to a

complement or to an uncomplemented form). Irrespective of number of variables the labels along each row and column must conform to a single change. Hence gray code is used to label the rows and columns of K-map as shown ow.





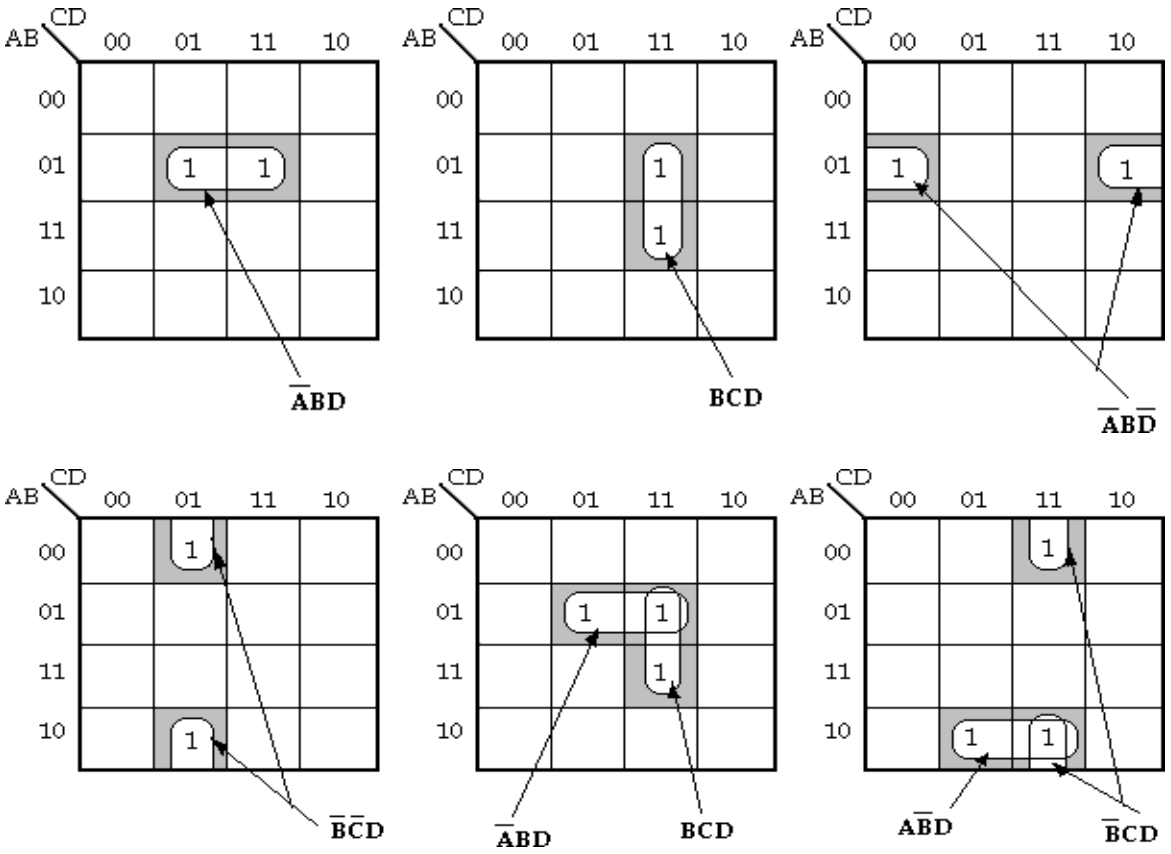
Grouping cells for Simplification:

The grouping is nothing but combining terms in adjacent cells. The simplification is achieved by grouping adjacent 1's or 0's in groups of 2i, where i = 1, 2, …, n and n is the number of variables. When adjacent 1's are grouped then we get result in the sum of product form; otherwise we get result in the product of sum form.

Department of Information Technology

Grouping Two Adjacent 1's: (Pair)

In a Karnaugh map we can group two adjacent 1's. The resultant group is called Pair.



Examples of Pairs

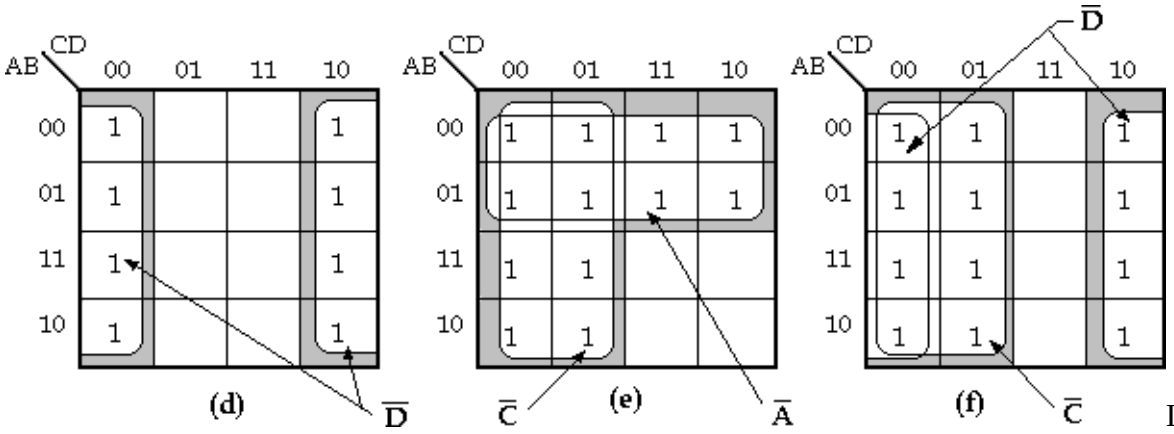
Grouping Four Adjacent 1's: (Quad)

In a Karnaugh map we can group four adjacent 1's. The resultant group is called Quad. Fig (a) shows the four 1's are horizontally adjacent and Fig (b) shows they are vertically adjacent. Fig (c) contains four 1's in a square, and they are considered adjacent to each other.

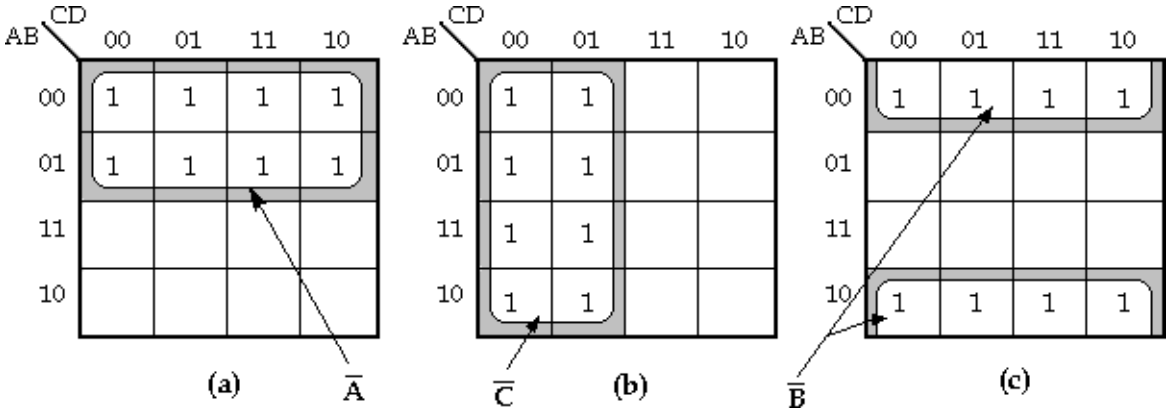
Examples of Quads

The four 1's in fig (d) and fig (e) are also adjacent, as are those in fig (f) because, the top and bottom rows are considered to be adjacent to each other and the leftmost and rightmost columns are also adjacent to each other.

Grouping Eight Adjacent 1's: (Octet)



In a Karnaugh map we can group eight adjacent 1's. The resultant group is called Octet.



Simplification of Sum of Products Expressions: (Minimal Sums)

The generalized procedure to simplify Boolean expressions as follows:

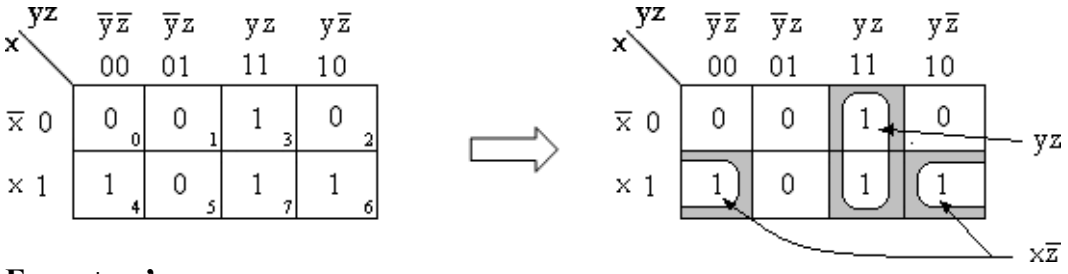
1) Plot the K-map and place 1's in those cells corresponding to the 1's in the sum of product expression. Place 0's in the other cells.
2) Check the K-map for adjacent 1's and encircle those 1's which are not adjacent to any other 1's. These are called **isolated 1's**.
3) Check for those 1's which are adjacent to only one other 1 and encircle such **pairs**.
4) Check for **quads** and **octets** of adjacent 1's even if it contains some 1's that have already been encircled. While doing this make sure that there are minimum number of groups.
5) Combine any pairs necessary to include any 1's that have not yet been grouped.
6) Form the simplified expression by summing product terms of all the groups.

Three- Variable Map:

1. Simplify the Boolean expression,

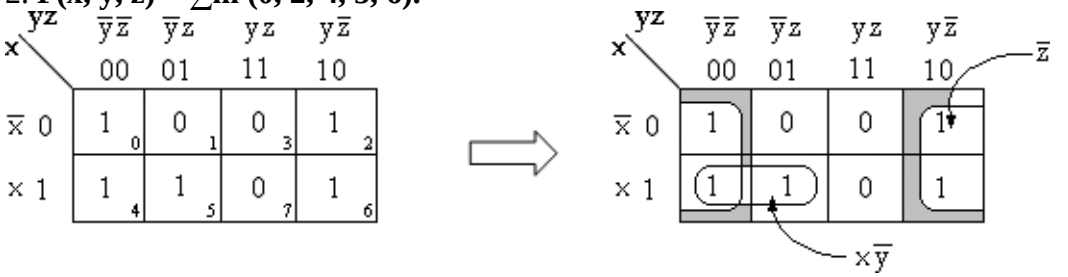
F(x, y, z) = ∑m (3, 4, 6, 7).

Soln:



F = yz+ xz'

2. **F(x, y, z) = ∑m (0, 2, 4, 5, 6).**



F = z'+ xy'

3. F = A'C + A'B + AB'C + BC

Soln:

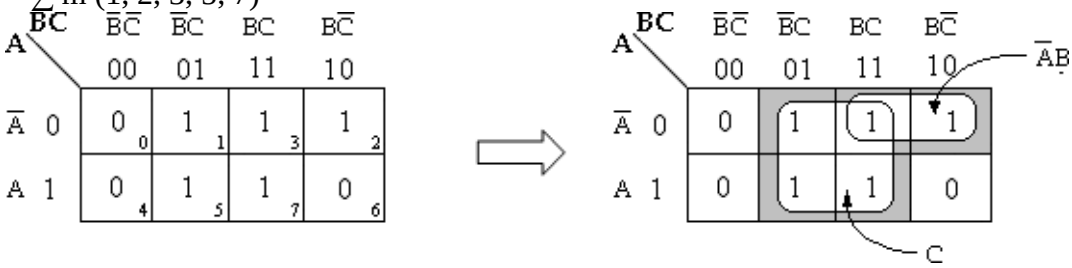
= A'C (B+ B') + A'B (C+ C') + AB'C + BC (A+ A')

= A'BC+ A'B'C + A'BC + A'BC' + AB'C + ABC + A'BC

= A'BC+ A'B'C + A'BC' + AB'C + ABC

= m_3+ m_1+ m_2+ m_5+ m_7

= ∑ m (1, 2, 3, 5, 7)



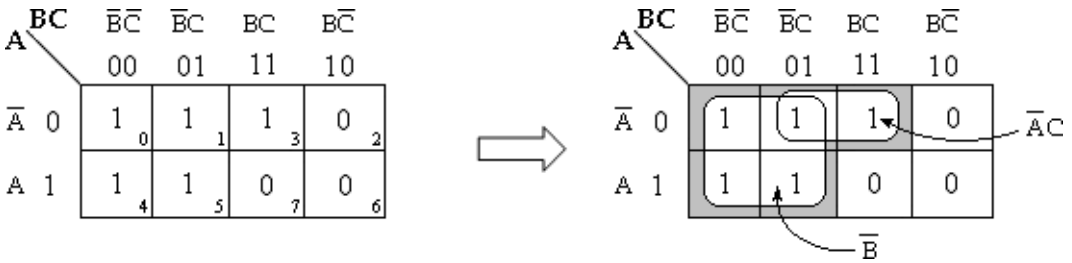
F = C + A'B

4. AB'C + A'B'C + A'BC + AB'C' + A'B'C'

Soln:

= m5 + m1 + m3 + m4 + m0

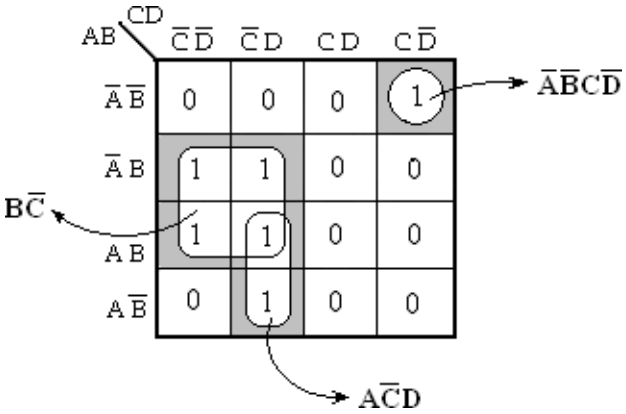
= ∑ m (0, 1, 3, 4, 5)



F = A'C + B'

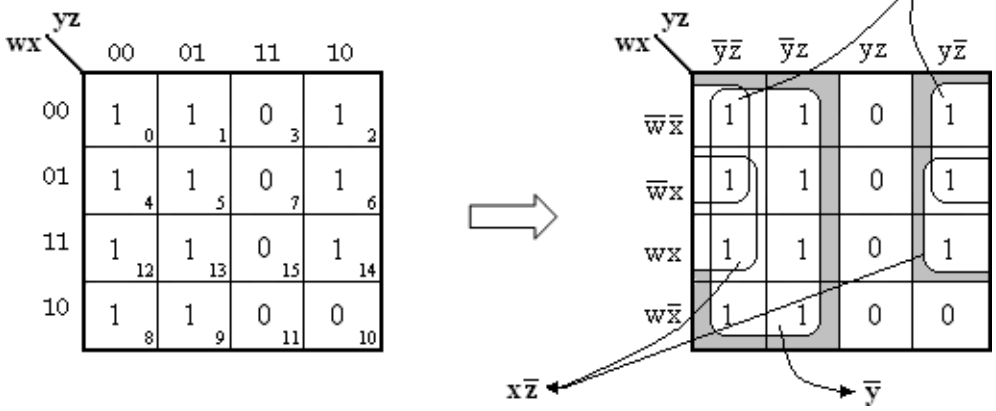
Four - Variable Map:

1. Simplify the Boolean expression,
Y = A'BC'D' + A'BC'D + ABC'D' + ABC'D + AB'C'D + A'B'CD'



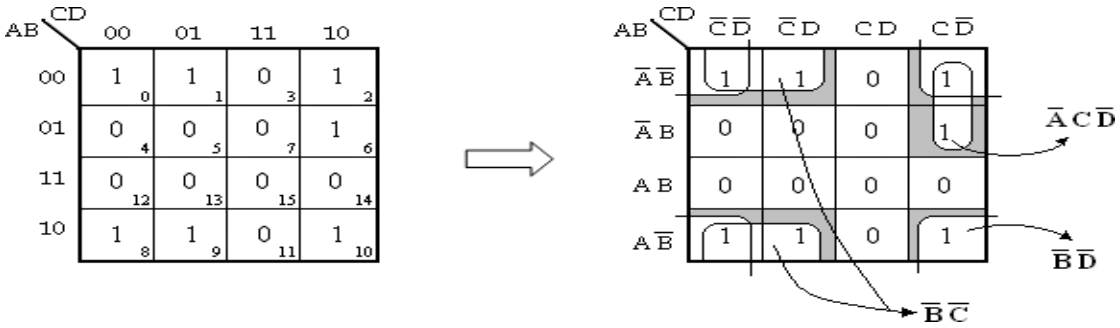
Soln:
Therefore, Y= A'B'CD'+ AC'D+ BC'

2. **F (w, x, y, z) = ∑ m(0, 1, 2, 4, 5, 6, 8, 9, 12, 13, 14)**
Soln:



Therefore,
F= y'+ w'z'+ xz'

3.F= A'B'C'+ B'CD'+ A'BCD'+ AB'C'
= A'B'C' (D+ D') + B'CD' (A+ A') + A'BCD'+ AB'C' (D+ D')
= A'B'C'D+ A'B'C'D'+ AB'CD'+ A'B'CD'+ A'BCD'+ AB'C'D+ AB'C'D'
= m_1+ m_0+ m_{10}+ m_2+ m_6+ m_9+ m_8

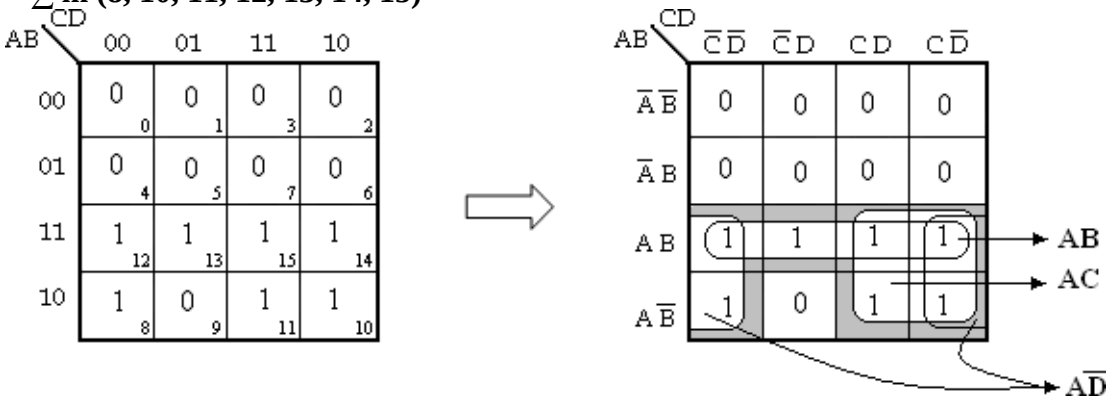


= ∑ **m (0, 1, 2, 6, 8, 9, 10)**
Therefore,
F= B'D'+ B'C'+ A'CD'.

4.Y= ABCD+ AB'C'D'+ AB'C+ AB

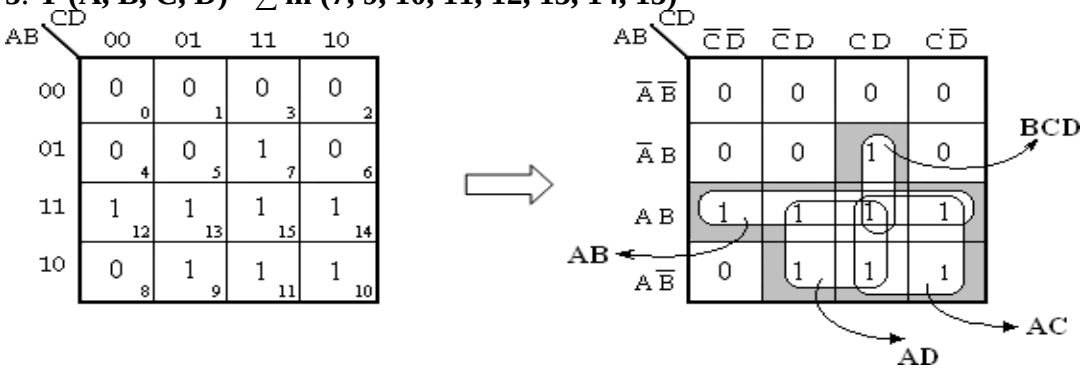
= ABCD+ AB'C'D'+ AB'C (D+D')+ AB (C+C') (D+D')
= ABCD+ AB'C'D'+ AB'CD+ AB'CD'+ (ABC+ ABC') (D+ D')
= ABCD+ AB'C'D'+ AB'CD+ AB'CD'+ ABCD+ ABCD'+ ABC'D+ ABC'D'
= ABCD+ AB'C'D'+ AB'CD+ AB'CD'+ ABCD'+ ABC'D+ ABC'D'
= m_{15}+ m_8+ m_{11}+ m_{10}+ m_{14}+ m_{13}+ m_{12}

= ∑ **m (8, 10, 11, 12, 13, 14, 15)**



Therefore,
Y= AB+ AC+ AD'.

5. Y (A, B, C, D)= ∑ m (7, 9, 10, 11, 12, 13, 14, 15)

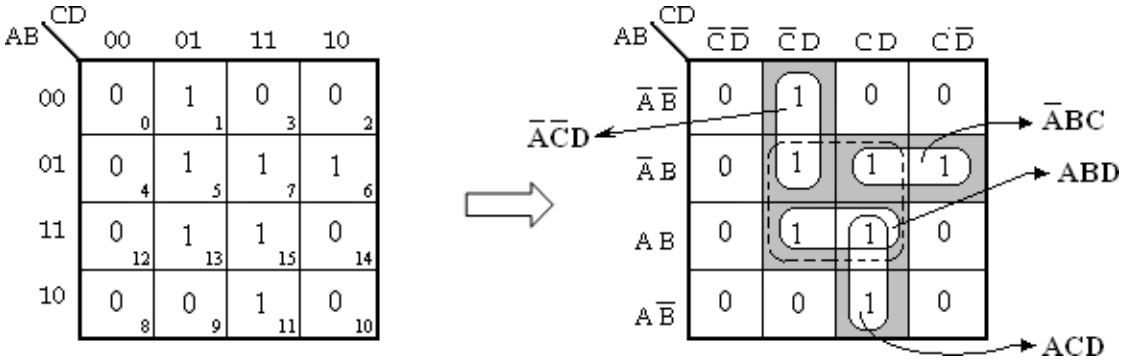


Therefore, **Y= AB+ AC+ AD+BCD.**

6. Y= A'B'C'D+ A'BC'D+ A'BCD+ A'BCD'+ ABC'D+ ABCD+ AB'CD

= m_1+ m_5+ m_7+ m_6+ m_{13}+ m_{15}+ m_{11}

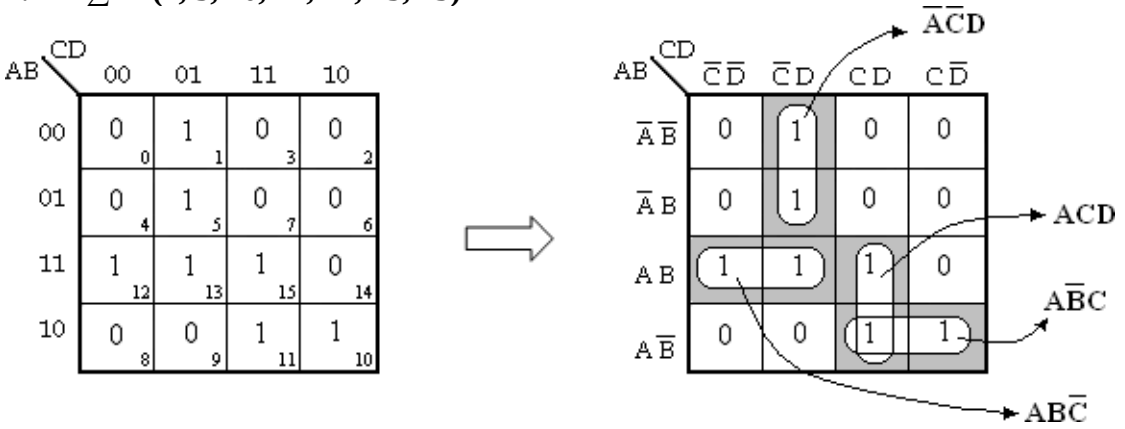
= ∑ **m (1, 5, 6, 7, 11, 13, 15)**



In the above K-map, the cells 5, 7, 13 and 15 can be grouped to form a quad as indicated by the dotted lines. In order to group the remaining 1's, four pairs have to be formed. However, all the four 1's covered by the quad are also covered by the pairs. So, the quad in the above k-map is redundant.
Therefore, the simplified expression will be,

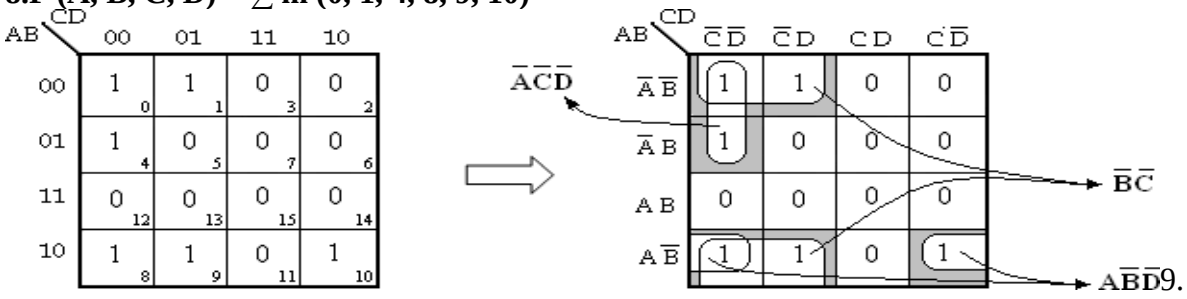
Y = A'C'D+ A'BC+ ABD+ ACD.

7. Y= ∑ m (1, 5, 10, 11, 12, 13, 15)



Therefore, **Y= A'C'D+ ABC'+ ACD+ AB'C.**

8.F (A, B, C, D) = ∑ m (0, 1, 4, 8, 9, 10)



Therefore, **F= A'C'D'+ AB'D'+ B'C'.**

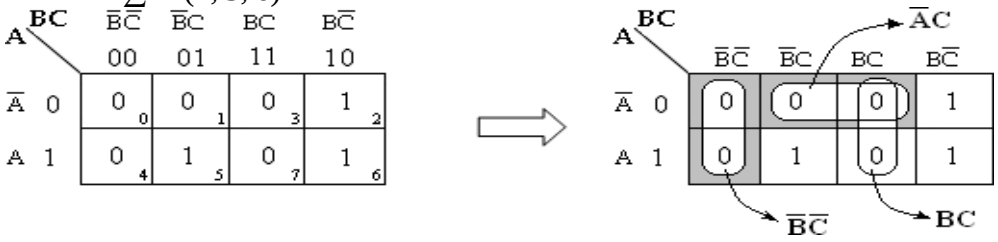
<u>Simplification of Sum of Products Expressions: (Minimal Sums)</u>

1. **Y= (A+ B+ C') (A+ B'+ C') (A'+ B'+ C') (A'+ B+ C) (A+ B+ C)**

= M_1. M_3. M_7. M_4. M_0

=∏ M (0, 1, 3, 4, 7)

= ∑ m (2, 5, 6)



Y' = B'C'+ A'C+ BC.

Y= Y" = (B'C'+ A'C+ BC)'

= (B'C')'. (A'C)'. (BC)'

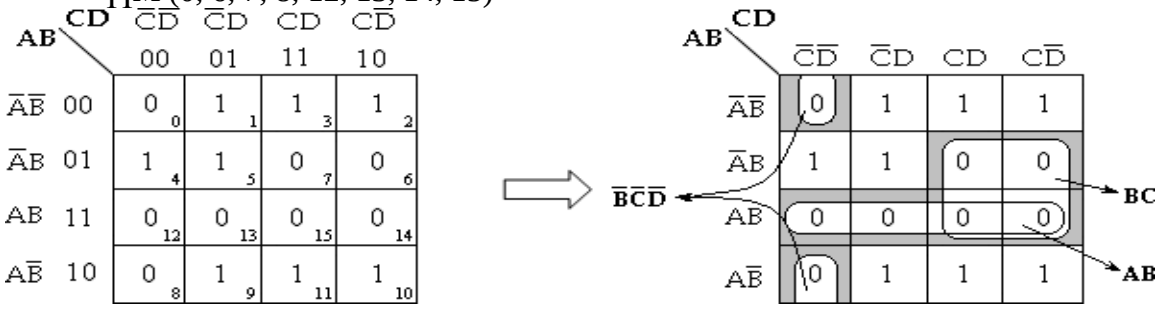
= (B"+ C"). (A"+C'). (B'+ C')

Y = (B+ C). (A+C'). (B'+ C')

2. **Y= (A'+ B'+ C+ D) (A'+ B'+ C'+ D) (A'+ B'+ C'+ D') (A'+ B+ C+ D) (A+ B'+ C'+ D) (A+ B'+ C'+ D') (A+ B+ C+ D) (A'+ B'+ C+ D')**

= M_{12}. M_{14}. M_{15}. M_8. M_6. M_7. M_0. M_{13}

= ∏M (0, 6, 7, 8, 12, 13, 14, 15)



Y' = B'C'D'+ AB+ BC

Y= Y" = (B'C'D'+ AB+ BC)'

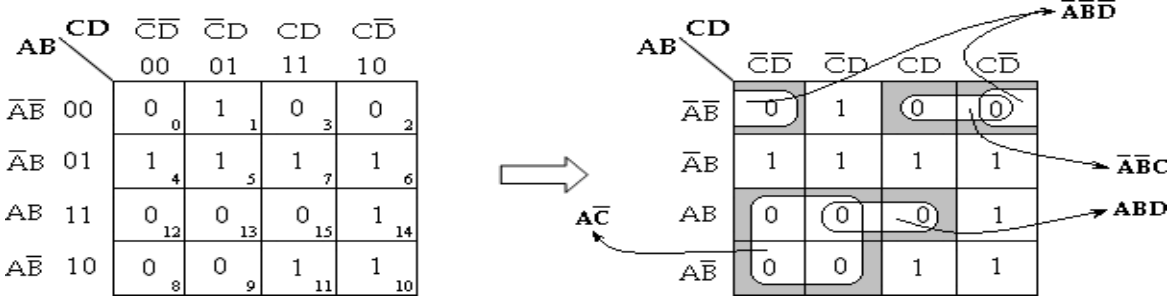
= (B'C'D')'. (AB)'. (BC)'

= (B"+ C"+D"). (A'+B'). (B'+ C')

= (B+ C+ D). (A'+ B'). (B'+ C')

Therefore, **Y= (B+ C+ D). (A'+ B'). (B'+ C')**

3. **F(A, B, C, D)=** ∏M (0, 2, 3, 8, 9, 12, 13, 14, 15)

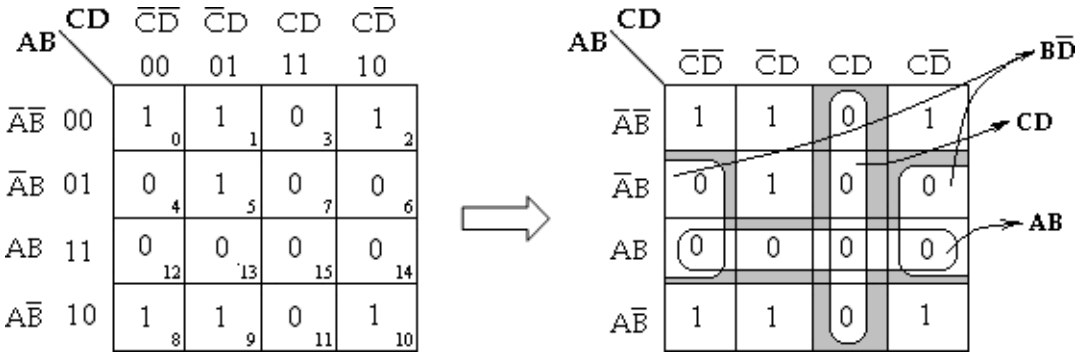


Y' = A'B'D'+ A'B'C+ ABD+ AC'

Y= Y" = (A'B'D'+ A'B'C+ ABD+ AC')'
= (A'B'D')'. (A'B'C)'. (ABD)'. (AC')'
= (A"+ B"+ D"). (A"+ B"+C'). (A'+ B'+ D'). (A'+ C")
= (A+ B+ D). (A+ B+ C'). (A'+ B'+ D'). (A'+ C)

Therefore, **Y= (A+ B+ D). (A+ B+ C'). (A'+ B'+ D'). (A'+ C)**

4. **F(A, B, C, D)=** ∑m (0, 1, 2, 5, 8, 9, 10)
= ∏M (3, 4, 6, 7, 11, 12, 13, 14, 15)

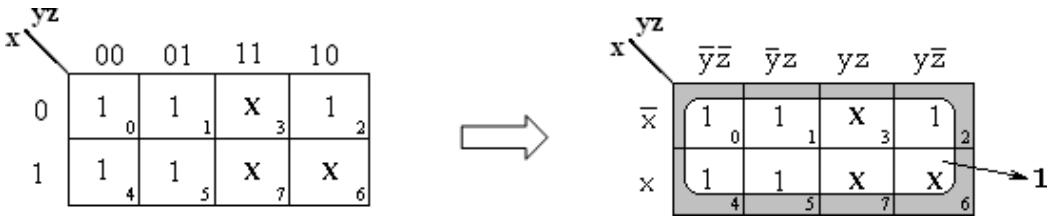


Y' = BD'+ CD+ AB

Y= Y" = (BD'+ CD+ AB)'
= (BD')'. (CD)'. (AB)' = (B'+ D"). (C'+ D'). (A'+ B')
= (B'+ D). (C'+ D'). (A'+ B')
Therefore, **Y= (B'+ D). (C'+ D'). (A'+ B')**

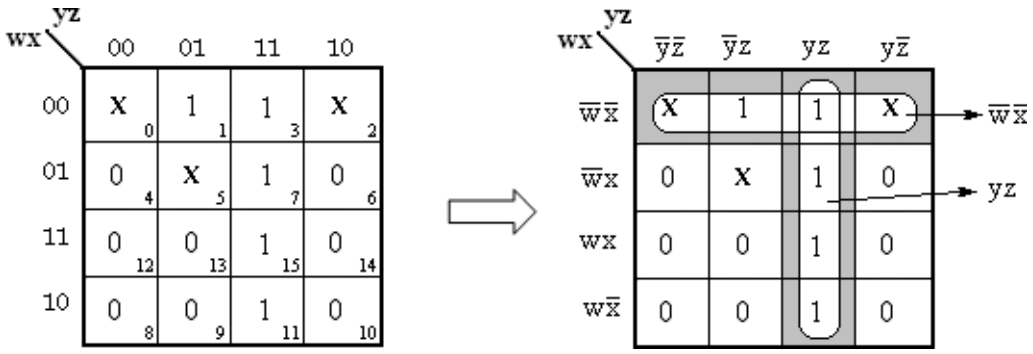
<u>Don't care Conditions</u>:
A don't care minterm is a combination of variables whose logical value is not specified. When choosing adjacent squares to simplify the function in a map, the don't care minterms may be assumed to be either 0 or 1. When simplifying the function, we can choose to include each don't care minterm with either the 1's or the 0's, depending on which combination gives the simplest expression.

1. F (x, y, z) = ∑m (0, 1, 2, 4, 5)+ ∑d (3, 6, 7)



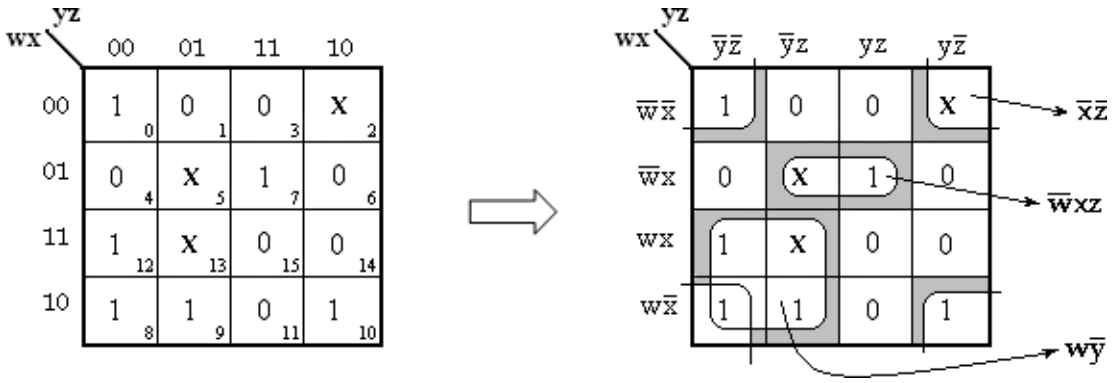
F (x, y, z) = 1

2. F (w, x, y, z) = ∑m (1, 3, 7, 11, 15)+ ∑d (0, 2, 5)



F (w, x, y, z) = w'x'+ yz

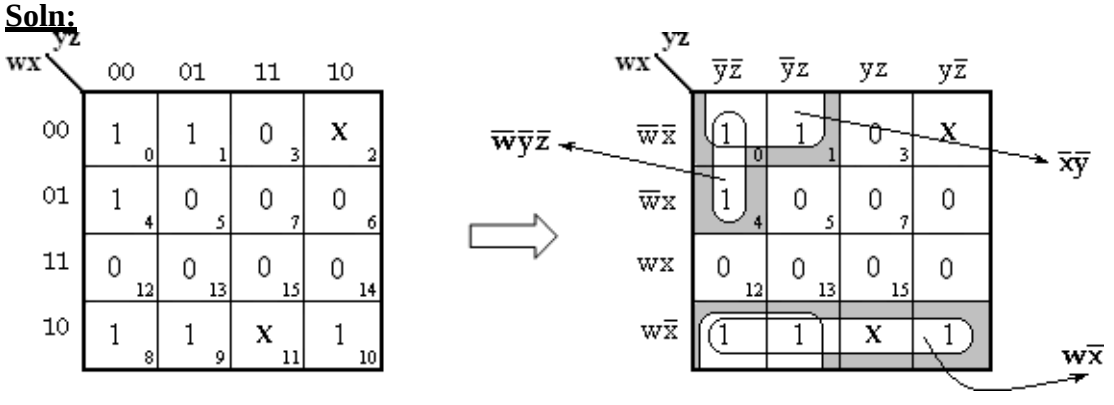
3. F (w, x, y, z) = ∑m (0, 7, 8, 9, 10, 12)+ ∑d (2, 5, 13)



F (w, x, y, z) = w'xz+ wy'+ x'z'.

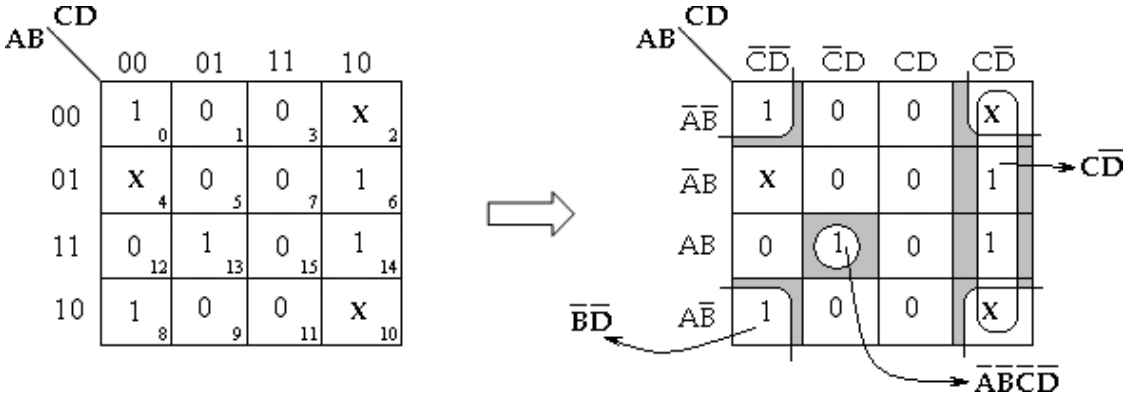
4. F (w, x, y, z) = ∑m (0, 1, 4, 8, 9, 10)+ ∑d (2, 11)

Soln:



F (w, x, y, z) = wx'+ x'y'+ w'y'z'.

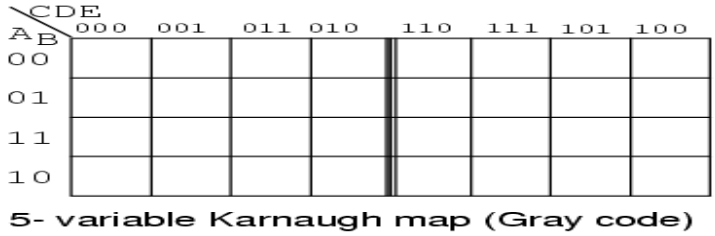
5. F(A, B, C, D) = ∑m (0, 6, 8, 13, 14)+ ∑d (2, 4, 10)
Soln:



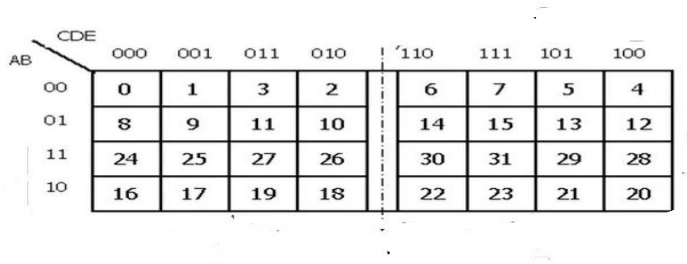
F(A, B, C, D) = CD'+ B'D'+ A'B'C'D'.

5 variable k map

A 5- variable K- map requires 25= 32 cells, but adjacent cells are difficult to identify on a single 32-cell map. Therefore, two 16 cell K-maps are used.

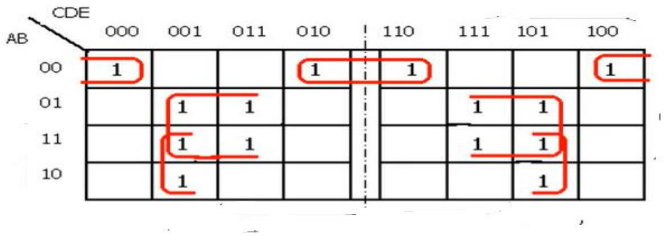


5- variable Karnaugh map (Gray code)



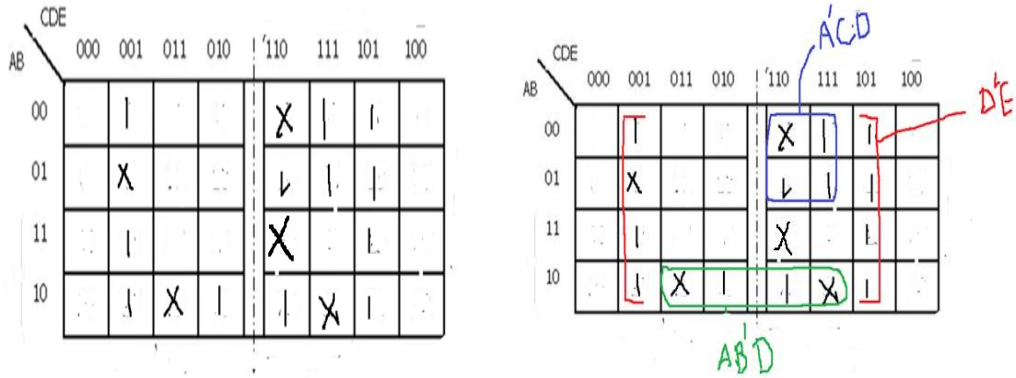
1.Simplify the Boolean expression

$F(A, B, C, D, E) = \sum(0, 2, 4, 6, 9, 11, 13, 15, 17, 21, 25, 27, 29, 31)$



$F_{(ABCDE)} = BE + AD'E + A'B'E'$

2.Simplify F(A,B,C,D,E)=(1,5,7,13,14,15,17,18,21,22,25,29)+d(6,9,19,23,30)



F= D'E+ A'CD+AB'D

Limitations of K map

- K map is a very effective tool for minimizations of logic functions with 4 or less variable.
- For logic expressions with more than 4 variables, the visualization of adjacent cells and the drawing of the k map become more difficult

2.6 QUINE- MCCLUSKEY METHOD or TABULATION METHOD

The tabular method which is also known as the Quine-McCluskey method is particularly useful when minimising functions having a large number of variables, e.g. The six-variable functions. Computer programs have been developed employing this algorithm. The method reduces a function in standard sum of products form to a set of prime implicants from which as many variables are eliminated as possible. These prime implicants are then examined to see if some are redundant.

The tabular method makes repeated use of the law $A + \bar{A} = 1$. Note that Binary notation is used for the function, although decimal notation is also used for the functions. As usual a variable in true form is denoted by 1, in inverted form by 0, and the abscence of a variable by a dash (-).

RULES OF TABULATION METHOD

- List all minterms in the binary form.
- Arrange the minterms according to number of 1's and separate by a horizontal line.

- Compare each binary number with every term in the adjacent next higher category and if they differ only by one position, put a check mark and copy the term in the next column with '-' in the position that they differed.
- Apply the same process described in step 3 for the resultant column and continue these until no further elimination of literals.
- List all the prime implicants.
- Select the minimum number of prime implicants which must cover all the minterms.

Simplify the given boolean expression using Tabulation method

1. $Y(A, B, C, D) = \sum m(2, 4, 5, 9, 12, 13)$

Table : 1

Min Term	Binary Representation				No.of 1s	
	A	B	C	D		
2	0	0	1	0	1	✓
4	0	1	0	0	1	✓
5	0	1	0	1	2	✓
9	1	0	0	1	2	✓
12	1	1	0	0	2	✓
13	1	1	0	1	3	✓

Table : 3

Min Term	Binary Representation				
	A	B	C	D	
4,5	0	1	0	_	✓
4,12	_	1	0	0	✓
5,13	_	1	0	1	✓
9,13	1	_	0	1	*
12,13	1	1	0	_	✓

Table : 2

Min Term	Binary Representation				
	A	B	C	D	
2	0	0	1	0	*
4	0	1	0	0	✓
5	0	1	0	1	✓
9	1	0	0	1	✓
12	1	1	0	0	✓
13	1	1	0	1	✓

Table : 4

Min Term	Binary Representation				
	A	B	C	D	
4,5,12,13	_	1	0	_	
4,12,5,13	_	1	0	_	

Table : 4 Reduced

4,5,12,13	_	1	0	_	*

Table: 5 Prime Implicants Table

Min Term	Binary Representation A	B	C	D	Product Term		2	4	5	9	12	13
2	0	0	1	0	A' B' C D'	✓	X					
9,13	1	_	0	1	A C' D	✓				X		X
4, 5, 12, 13	_	1	0	_	B C'	✓		X	X		X	X
							✓	✓	✓	✓	✓	

Select Single X column for Essential Prime Implecants

$Y = A' B' C D' + A C' D + B C'$

2. $f(A, B, C, D) = \sum m(0,1,2,3,5,7,8,10,12,13,15)$

Table : 1

Min Term	Binary Representation A	B	C	D	No.of 1s	
0	0	0	0	0	0	✓
1	0	0	0	1	1	✓
2	0	0	1	0	1	✓
3	0	0	1	1	2	✓
5	0	1	0	1	2	✓
7	0	1	1	1	3	✓
8	1	0	0	0	1	✓
10	1	0	1	0	2	✓
12	1	1	0	0	2	✓
13	1	1	0	1	3	✓
15	1	1	1	1	4	✓

Table : 2

Min Term	Binary Representation A	B	C	D	
0	0	0	0	0	✓
1	0	0	0	1	✓
2	0	0	1	0	✓
8	1	0	0	0	✓
3	0	0	1	1	✓
5	0	1	0	1	✓
10	1	0	1	0	✓
12	1	1	0	0	✓
7	0	1	1	1	✓
13	1	1	0	1	✓
15	1	1	1	1	✓

Table : 3

Min Term	Binary Representation A	B	C	D	
0,1	0	0	0	_	✓
0,2	0	0	_	0	✓
0,8	_	0	0	0	✓
1,3	0	0	_	1	✓
1,5	0	_	0	1	✓
2,3	0	0	1	_	✓
2,10	_	0	1	0	✓
8,10	1	0	_	0	✓
8,12	1	_	0	0	*
3,7	0	_	1	1	✓
5,7	0	1	_	1	✓
5,13	_	1	0	1	✓
12,13	1	1	0	_	*
7,15	_	1	1	1	✓
13,15	1	1	_	1	✓

Table : 4

Min Term	Binary Representation A	B	C	D	
0, 1, 2, 3	0	0	_	_	
0, 2, 1, 3	0	0	_	_	
0, 2, 8, 10	_	0	_	0	
0, 8, 2, 3	_	0	_	0	
1, 3, 5, 7	0	_	_	1	
1, 5, 3, 7	0	_	_	1	
5, 7, 13, 15	_	1	_	1	
5, 13, 7, 15	_	1	_	1	

Table : 4 Reduced

0, 1, 2, 3	0	0	_	_	*
0, 2, 8, 10	_	0	_	0	*
1, 3, 5, 7	0	_	_	1	*
5, 7, 13, 15	_	1	_	1	*

Table: 5 Prime Implicants Table

Min Term	Binary Representation A	B	C	D	Product Term	0	1	2	3	5	7	8	10	12	13	15
8,12	1	_	0	0	$A\bar{C}\bar{D}$							X		X		
12,13	1	1	0	_	$AB\bar{C}$									X	X	
0, 1, 2, 3	0	0	_	_	$\bar{A}\bar{B}$	X	X	X	X							
0, 2, 8, 10	_	0	_	0	$\bar{B}\bar{D}$ ✓	X		X				X	X			
1, 3, 5, 7	0	_	_	1	$\bar{A}D$		X		X	X	X					
5, 7, 13, 15	_	1	_	1	BD ✓					X	X				X	X
Select Single X column for Essential Prime Implecants													✓			✓

Missing min terms 1, 3, 12 so include 1, 3, 5, 7 & 12, 13 OR 1, 3, 5, 7 & 8, 12

$$f(A, B, C, D) = \bar{B}\bar{D} + BD + \bar{A}D + AB\bar{C} \quad \text{OR} \quad f(A, B, C, D) = \bar{B}\bar{D} + BD + \bar{A}D + A\bar{C}\bar{D}$$

3. $Y(A,B,C,D,E) = \sum m(0, 1, 9, 15, 24, 29, 30) + \sum d(8, 11, 31)$

Table : 1

Min Term	Binary Representation A	B	C	D	E	No.of 1s	
0	0	0	0	0	0	0	✓
1	0	0	0	0	1	1	✓
d8	0	1	0	0	0	1	✓
9	0	1	0	0	1	2	✓
d11	0	1	0	1	1	3	✓
15	0	1	1	1	1	4	✓
24	1	1	0	0	0	2	✓
29	1	1	1	0	1	4	✓
30	1	1	1	1	0	4	✓
d31	1	1	1	1	1	5	✓

Table : 2

Min Term	Binary Representation A	B	C	D	E	
0	0	0	0	0	0	✓
1	0	0	0	0	1	✓
8	0	1	0	0	0	✓
9	0	1	0	0	1	✓
24	1	1	0	0	0	✓
11	0	1	0	1	1	✓
15	0	1	1	1	1	✓
29	1	1	1	0	1	✓
30	1	1	1	1	0	✓
31	1	1	1	1	1	✓

Table : 3

Min Term	Binary Representation A	B	C	D	E	
0,1	0	0	0	0	_	✓
0,8	0	_	0	0	0	✓
1,9	0	_	0	0	1	✓
8,9	0	1	0	0	_	✓
8,24	_	1	0	0	0	*
9,11	0	1	0	_	1	*
11,15	0	1	_	1	1	*
15,31	_	1	1	1	1	*
29,31	1	1	1	_	1	*
30,31	1	1	1	1	_	*

Table : 4

Min Term	Binary Representation A	B	C	D	E
0,1,8,9	0	_	0	0	_
0,8,1,9	0	_	0	0	_

Table : 4 Reduced

0,1,8,9	0	_	0	0	_	*

Table: 5 Prime Implicants Table

Min Term	Binary Representation					Product Term		0	1	d8	9	d11	15	24	29	30	d31
	A	B	C	D	E												
8,24	_	1	0	0	0	BC'D'E'	✓			X				X			
9,11	0	1	0	_	1	A'BC'E					X	X					
11,15	0	1	_	1	1	A'BDE						X	X				
15,31	_	1	1	1	1	BCDE							X				X
29,31	1	1	1	_	1	ABCE	✓								X		X
30,31	1	1	1	1	_	ABCD	✓									X	X
0,1,8,9	0	_	0	0	_	A'C'D'	✓	X	X	X	X						
Select Single X column for Essential Prime Implecants								✓	✓					✓	✓	✓	

Missing min terms 15 so include 11, 15 OR 15,31

$Y = B\,C'\,D'\,E' + A\,B\,C\,E + A\,B\,C\,D + A'\,C'\,D' + A'\,B\,D\,E$ OR

- $Y = B\,C'\,D'\,E' + A\,B\,C\,E + A\,B\,C\,D + A'\,C'\,D' + B\,C\,D\,E$

References

1.Floyd, Digital Fundamentals, Universal Bookstall, New Delhi, 1986.

2. Jain, R.P., Modern Digital Electronics, Tata McGraw Hill, 3rd Edition,1997.

3. Malvino.A.P. and Donald.P.Leach, Digital Principal and Applications, 4th Edition, Tata McGraw Hill, 2007.

Question bank

Part A

1. Show how to connect NAND gates to get an AND gate and OR gate?
2. Implement the given function using NAND gates only.
   F(X, Y, Z) =Σ(0,2,7)
3. Find the canonical POS form of Y= A+B'C
4. Interpret the truth table of EX- OR gate.
5. Convert the given expression in canonical SOP form
   Y = AC + AB + BC
6. Write the POS representation of the following SOP function:f(x,y,z)= Σm(0,1,3,5,7)
7. Evaluate the function F(x,y,z)= Σ m(0,3,4,6,7).
8. Name the gates that are called universal gates. Give the reason.
9. Implement AND gate using only NOR gates

Part B

1. Examine how to minimize the function F(A,B,C,D)= Σm(0,4,6,8,9,10,12)

+Σd(2,13) and implement it using only NOR gates.

2. Interpret the logical expression using K-map in SOP and POS form
   F(A,B,C,D)=Σm(0, 2, 3, 6, 7) + d(8, 10, 11,15)
3. Identify the minimal SOP form for the following function
   F(A,B,C,D)=Σm(0,1,3,5,6,8,9,14,26,28,31) +Σd(4,13).
4. Minimize the function F(A, B, C,D)= Σm(0,4,6,8,9,10,12) with d=Σm(2,13) using tabulation method
5. Express the Boolean function using K-map and implement it using only NAND gates. F(A,B,C,D)=Σm(0,8,11,12,15)+Σd(1,2,4,7,10,14). Give the essential and non-essential prime implicants.
6. Simplify the function F in Sum of Products (SOP) and Product of Sum (POS). F=Σm(3,4,13,15) and d=Σm(1,2,5,6,8,10),where 'd' represent the don't cares. Also, implement using NAND gates.
7. Simplify the following function using Karnaugh Map.
   F(W,X,Y,Z)=Σm(0,1,3,9,10,12,13,14)+ Σd(2,5,6,11) and verify using k-map
8. Explain the minimization of the given Boolean function using Quine-Mc-Cluskey method F=Σ(15,13,10,9,8,7,5,2,1,0).Realize the simplified function using logic gates.
9. Implement the following function using Veitch method
   F=Σd(3,4,11) +Σm(31,27,25,24,21,17,15,9,8,2,1,0)

SCHOOL OF BIO AND CHEMICAL ENGINEERING

DEPARTMENT OF BIOMEDICAL ENGINEERING

UNIT – III – Fundamentals of Digital Systems – SBMA1401

III. Combinational Circuits

Binary Adder-Subtractor, Parallel Binary Adder, Parallel Binary Subtractor, Parallel Adder/Subtractor, Decoders, Encoders, Priority Encoders, Multiplexers and DeMultiplexer, Magnitude Comparators-one bit and two bit.

3.1Introduction:

The digital system consists of two types of circuits, namely

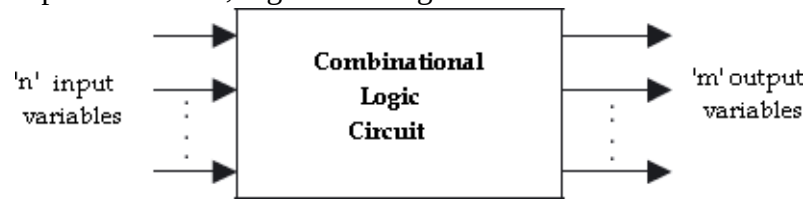
- Combinational circuits
- Sequential circuits

Combinational circuit consists of logic gates whose output at any time is determined from the present combination of inputs. The logic gate is the most basic building block of combinational logic. The logical function performed by a combinational circuit is fully defined by a set of Boolean expressions.

Sequential logic circuit comprises both logic gates and the state of storage elements such as flip-flops. As a consequence, the output of a sequential circuit depends not only on present value of inputs but also on the past state of inputs.

In the previous chapter, we have discussed binary numbers, codes, Boolean algebra and simplification of Boolean function and logic gates. In this chapter, formulation and analysis of various systematic designs of combinational circuits will be discussed.

A combinational circuit consists of input variables, logic gates, and output variables. The logic gates accept signals from inputs and output signals are generated according to the logic circuits employed in it. Binary information from the given data transforms to desired output data in this process. Both input and output are obviously the binary signals, *i.e.*, both the input and output signals are of two possible states, logic 1 and logic 0.



Block diagram of a combinational logic circuit

For n number of input variables to a combinational circuit, 2^n possible combinations of binary input states are possible. For each possible combination, there is one and only one possible output combination. A combinational logic circuit can be described by m Boolean functions and each output can be expressed in terms of n input variables.

DESIGN PROCEDURE:

- The problem is stated.
- Identify the input and output variables.
- The input and output variables are assigned letter symbols.
- Construction of a truth table to meet input -output requirements.
- Writing Boolean expressions for various output variables in terms of input variables.
- The simplified Boolean expression is obtained by any method of minimization—algebraic method, Karnaugh map method, or tabulation method.
- A logic diagram is realized from the simplified boolean expression using logic gates.

The following guidelines should be followed while choosing the preferred form for hardware implementation:

- The implementation should have the minimum number of gates, with the gates used having the minimum number of inputs.
- There should be a minimum number of interconnections.
- Limitation on the driving capability of the gates should not be ignored.

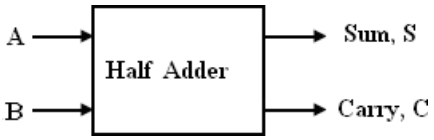
ARITHMETIC CIRCUITS – BASIC BUILDING BLOCKS:

In this section, we will discuss those combinational logic building blocks that can be used to perform addition and subtraction operations on binary numbers. Addition and subtraction are the two most commonly used arithmetic operations, as the other two, namely multiplication and division, are respectively the processes of repeated addition and repeated subtraction.

The basic building blocks that form the basis of all hardware used to perform the arithmetic operations on binary numbers are half-adder, full adder, half-subtractor, full- subtractor.

3.2 Half-Adder:

A half-adder is a combinational circuit that can be used to add two binary bits. It has two inputs that represent the two bits to be added and two outputs, with one producing the SUM output and the other producing the CARRY.



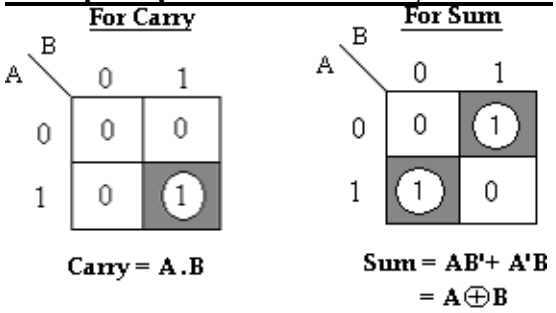
Block schematic of half-adder

The truth table of a half-adder, showing all possible input combinations and the corresponding outputs are shownbelow.

Truth table of half-adder

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

K-map simplification for carry and sum:



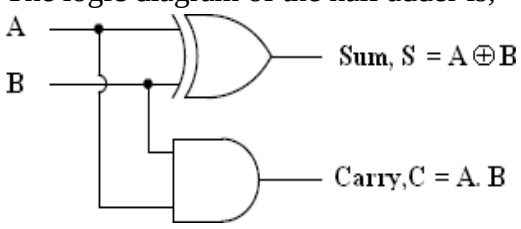
The Boolean expressions for the SUM and CARRY outputs are given by the equations,

Sum, S = A'B+ AB'

Carry, C = A . B

The first one representing the SUM output is that of an EX-OR gate, the second one representing the CARRY output is that of an AND gate.

The logic diagram of the half adder is,

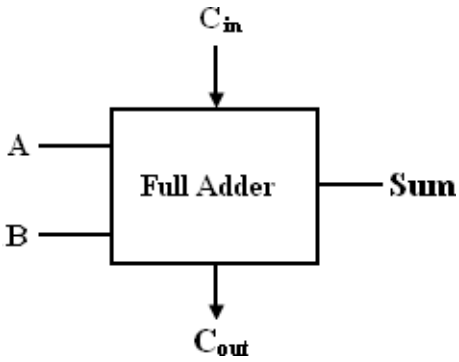


Logic Implementation of Half-adder

3.3 Full-Adder:

A full adder is a combinational circuit that forms the arithmetic sum of three input bits. It consists of 3 inputs and 2 outputs.

Two of the input variables, represent the significant bits to be added. The third input represents the carry from previous lower significant position. The block diagram of full adder is given by,



Block schematic of full-adder

The full adder circuit overcomes the limitation of the half-adder, which can be used to add two bits only. As there are three input variables, eight different input combinations are possible. The truth table is shown below,

Truth Table:

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

To derive the simplified Boolean expression from the truth table, the Karnaugh map method is adopted as,

For Carry

A \ BC_{in}	00	01	11	10
0	0	0	1	0
1	0	1	1	1

For Sum

A \ BC_{in}	00	01	11	10
0	0	1	0	1
1	1	0	1	0

Carry, C_{out} = AB+ AC_{in} + BC_{in}

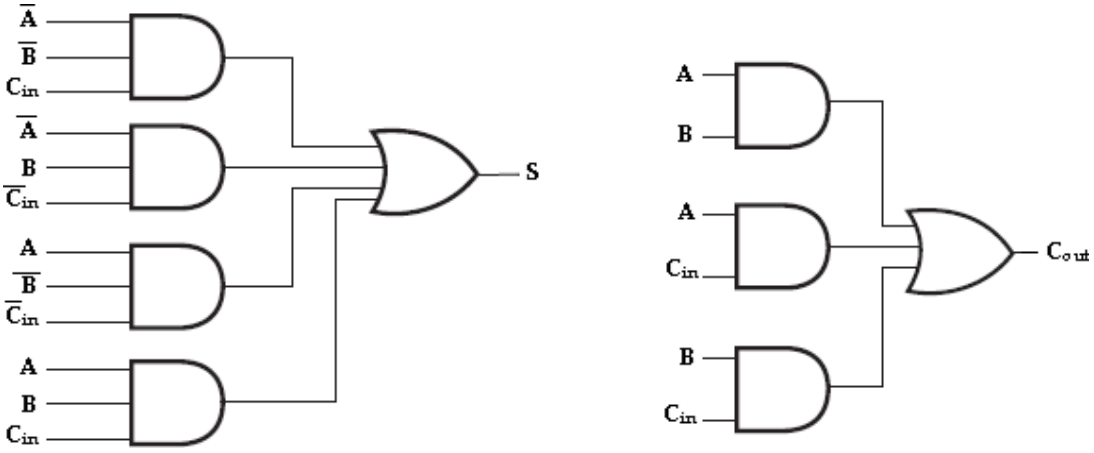
Sum, S = A'B'C_{in}+ A'BC'$_{in}$+ AB'C'$_{in}$+ ABC_{in}

The Boolean expressions for the SUM and CARRY outputs are given by the equations,

Sum, S = A'B'Cin+ A'BC'in + AB'C'in +ABC$_{in}$

Carry, C_{out} = AB+ AC_{in} + BC_{in}.

The logic diagram for the above functions is shown as,



Implementation of full-adder in Sum of Product

The logic diagram of the full adder can also be implemented with two half- adders and one OR gate. The S output from the second half adder is the exclusive-OR of C_{in} and the output of the first half-adder, giving

Sum = C_{in} ⊕ (A ⊕ B) [x⊕y = x'y+ xy']

= C_{in} ⊕ (A'B+AB')

= C'in (A'B+AB') + C_{in} (A'B+AB')' [(x'y+xy')'= (xy+x'y')]

= C'in (A'B+AB') + Cin (AB+A'B')

= A'BC'in + AB'C'in + ABC_{in} + A'B'Cin .

and the carry output is,

Carry, C_{out} = AB+ C_{in} (A'B+AB')

= AB+ A'BCin+ AB'Cin

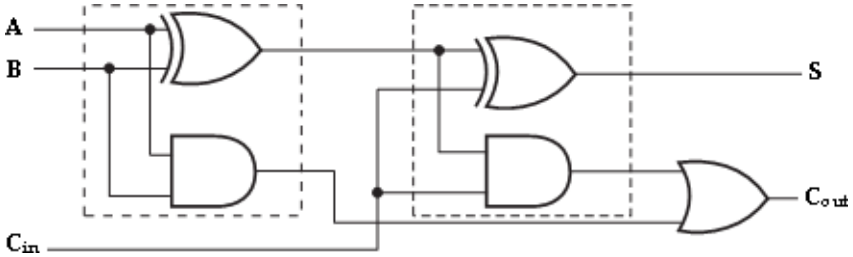
= AB (C_{in}+1) + A'BCin+ AB'Cin [C_{in}+1= 1]

= ABC_{in}+ AB+ A'BCin+ AB'Cin

= AB+ AC_{in} (B+B') + A'BCin

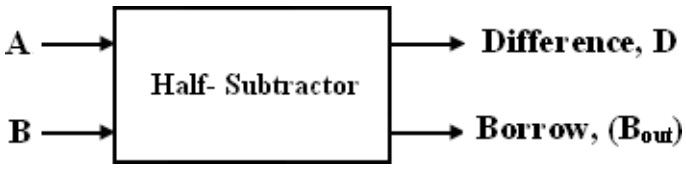
= AB+ AC_{in}+ A'BCin

= AB (C_{in}+1) + AC_{in}+ A‘BCin [C_{in}+1= 1]
= ABC_{in}+ AB+ AC_{in}+ A‘BCin
= AB+ AC_{in}+ BC_{in} (A +A‘)
= AB+ AC_{in}+ BC_{in}.



Implementation of full adder with two half-adders and an OR gate

3.4 Half -Subtractor:



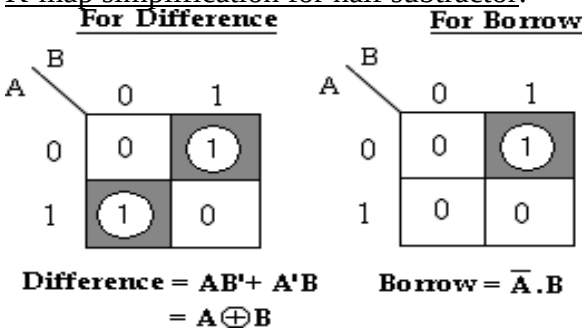
Block schematic of half-subtractor

A half-subtractor is a combinational circuit that can be used to subtract one binary digit from another to produce a DIFFERENCE output and a BORROW output. The BORROW output here specifies whether a _1‘ has been borrowed to perform the subtraction.

The truth table of half-subtractor, showing all possible input combinations and the corresponding outputs are shown below.

Input		Output	
A	**B**	**Difference (D)**	**Borrow (B_{out})**
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

K-map simplification for half subtractor:



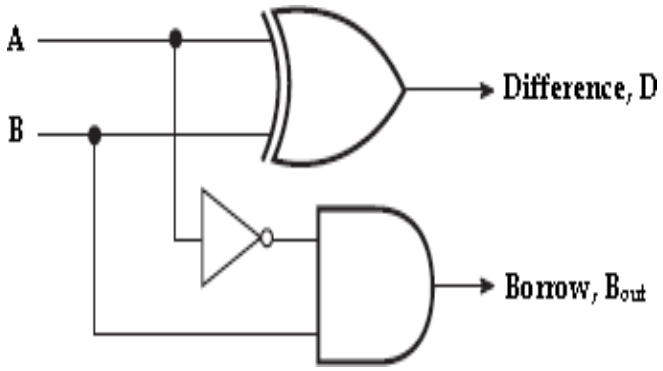
The Boolean expressions for the DIFFERENCE and BORROW outputs are given by the equations,

Difference, D = A'B+ AB'

Borrow, B_{out} = A' . B

The first one representing the DIFFERENCE (**D**)output is that of an exclusive-OR gate, the expression for the BORROW output (**B_{out}**) is that of an AND gate with input A complemented before it is fed to thegate.

The logic diagram of the half adder is,



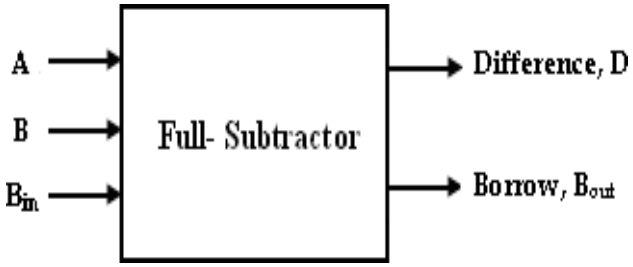
Logic Implementation of Half-Subtractor

Comparing a half-subtractor with a half-adder, we find that the expressions for the SUM and DIFFERENCE outputs are just the same. The expression for BORROW in the case of the half-subtractor is also similar to what we have for CARRY in the case of the half-adder. If the input A, ie., the minuend is complemented, an AND gate can be used to implement the BORROW output.

3.5 Full Subtractor:

A full subtractor performs subtraction operation on two bits, a minuend and a subtrahend, and also takes into consideration whether a _1' has already been borrowed by the previous adjacent lower minuend bit ornot.

As a result, there are three bits to be handled at the input of a full subtractor, namely the two bits to be subtracted and a borrow bit designated as B_{in}. There are two outputs, namely the DIFFERENCE output D and the BORROW output B_o. The BORROW output bit tells whether the minuend bit needs to borrow a _1' from the next possible higher minuend bit.



Block schematic of full-adder

The truth table for full-subtractor is,

Inputs			Outputs	
A	**B**	**B_{in}**	**Difference(D)**	**Borrow(B_{out})**
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

For Difference

A \ BB_{in}	00	01	11	10
0	0	(1)	0	(1)
1	(1)	0	(1)	0

For Borrow

A \ BB_{in}	00	01	11	10
0	0	(1)	(1)	(1)
1	0	0	(1)	0

Difference, $D = A'B'B_{in} + A'BB'_{in} + AB'B'_{in} + ABB_{in}$

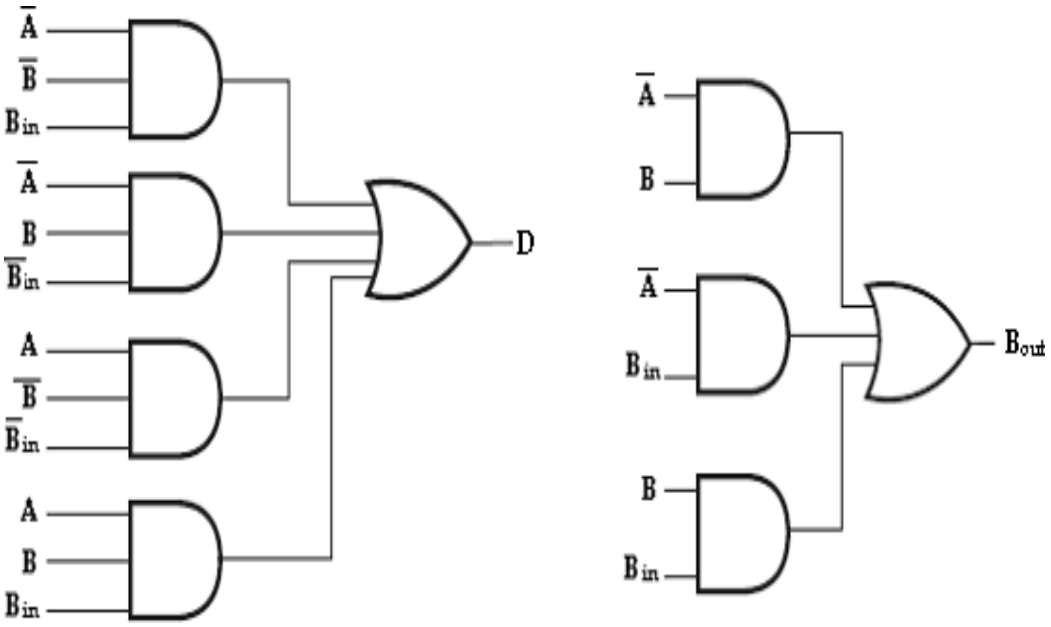
Borrow, $B_{out} = A'B + A'B_{in} + BB_{in}$ **K-map simplification for full-subtractor:**

The Boolean expressions for the DIFFERENCE and BORROW outputs are given by the equations,

Difference, D = A'B'Bin + A'BB'in + AB'B'in + ABB_{in}

Borrow, B_{out} = A'B + A'Cin + BB_{in}.

The logic diagram for the above functions is shown as,



Implementation of full-Subtractor using Half Subtractors

The logic diagram of the full-subtractor can also be implemented with two half- subtractors and one OR gate. The difference,D output from the second half subtractor is the Ex -OR of B_{in} and the output of the first half-subtractor, giving

Difference,D= $B_{in} \oplus (A \oplus B)$ $[x \oplus y = x'y + xy']$

$= B_{in} \oplus (A'B + AB')$

= B‘in (A‘B+AB‘) + B_{in} (A‘B+AB‘)‘ [(x‘y+xy‘)‘= (xy+x‘y‘)]

= B‘in (A‘B+AB‘) + Bin (AB+A‘B‘)

= A‘BB‘in + AB‘B‘in + ABB_{in} + A‘B‘Bin .

and the borrow output is,

Borrow, B_{out} = A’B+ B_{in} (A’B+AB’)’ [(x‘y+xy‘)‘= (xy+x‘y‘)]

= A‘B+ B_{in} (AB+A‘B‘)

= A‘B+ ABBin+ A‘B‘Bin

= A‘B (Bin+1) +ABB_{in}+ A‘B‘Bin [C_{in}+1= 1]

= A‘BBin+ A‘B+ ABBin+ A‘B‘Bin

= A‘B+ BBin (A+A‘) + A‘B‘Bin [A+A‘= 1]

= A‘B+ BBin+ A‘B‘Bin

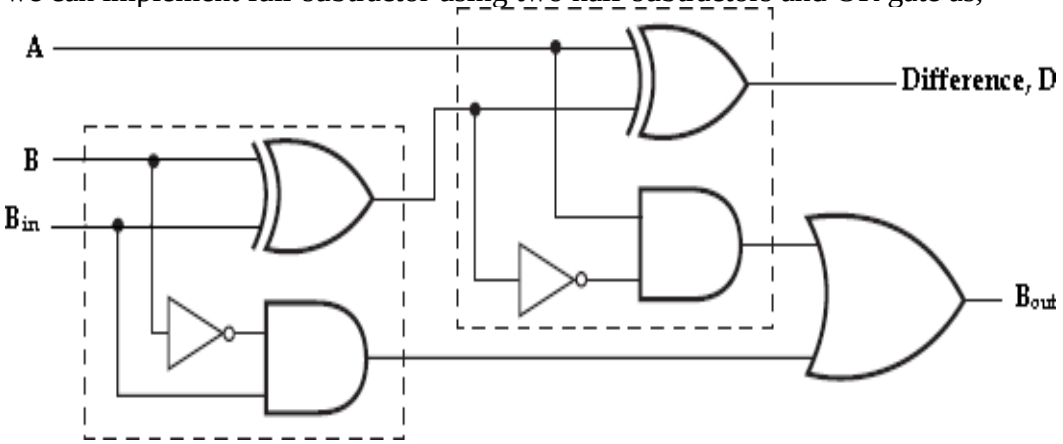
= A‘B (Bin+1) +BB_{in}+ A‘B‘Bin [C_{in}+1= 1]

= A‘BBin+ A‘B+ BBin+ A‘B‘Bin

= A‘B+ BBin+ A‘Bin (B +B‘)

= A‘B+ BBin+ A‘Bin.

Therefore,
we can implement full-subtractor using two half-subtractors and OR gate as,



Implementation of full-subtractor with two half-subtractors and an OR gate

3.6 Binary Adder (Parallel Adder):

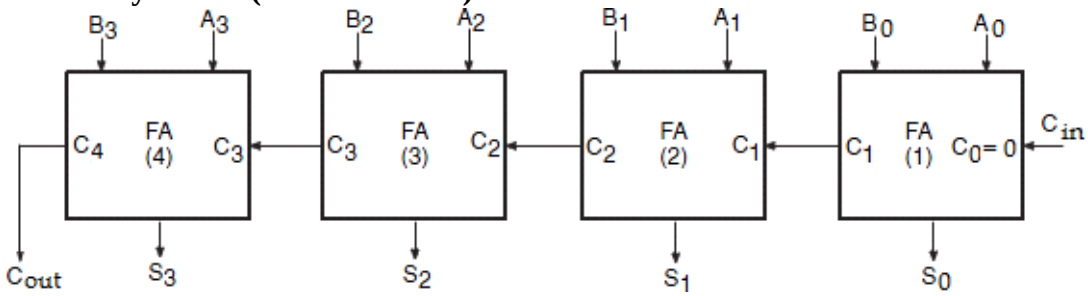


Fig. 4-bit binary parallel Adder

The 4-bit binary adder using full adder circuits is capable of adding two 4-bit numbers resulting in a 4-bit sum and a carry output as shown in figure below. Since all the bits of augend and addend are fed into the adder circuits simultaneously and the additions in each position are taking place at the same time, this circuit is known as parallel adder.

Significant place		4	3	2	1
Input carry		1	1	1	0
Augend word A :		1	1	1	1
Addend word B :		0	0	1	1
	1	0	0	1	0 ← Sum

↑ Output Carry

Let the 4-bit words to be added be represented by, $A_3A_2A_1A_0$= 1111 and $B_3B_2B_1B_0$= 0011.

The bits are added with full adders, starting from the least significant position, to form the sum it and carry bit. The input carry C_0 in the least significant position must be

- The carry output of the lower order stage is connected to the carry input of the next higher order stage. Hence this type of adder is called ripple-carry adder.

In the least significant stage, A_0, B_0 and C_0 (which is 0) are added resulting in sum S_0 and carry C_1. This carry C_1 becomes the carry input to the second stage. Similarly in the second stage,

A_1, B_1 and C_1 are added resulting in sum S_1 and carry C_2, in the third stage, A_2, B_2 and C_2 are added resulting in sum S_2 and carry C_3, in the third stage, A_3, B_3 and C_3 are added resulting in sum S_3 and C_4, which is the output carry. Thus the circuit results in a sum ($S_3S_2S_1S_0$) and a carry output (C_{out}).

3.7 Binary Subtractor (Parallel Subtractor):

The subtraction of unsigned binary numbers can be done most conveniently by means of complements. The subtraction A-B can be done by taking the 2‘s complement of B and adding it to A. The 2‘s complement can be obtained by taking the 1‘s complement and adding 1 to the least significant pair of bits. The 1‘s complement can be implemented with inverters and a 1 can be added to the sum through the input carry.

The circuit for subtracting A-B consists of an adder with inverters placed between each data input B and the corresponding input of the full adder. The input carry C_0 must be equal to 1 when performing subtraction. The operation thus performed becomes A, plus the 1‘s complement of B, plus1. This is equal to A plus the 2‘s complement of B.

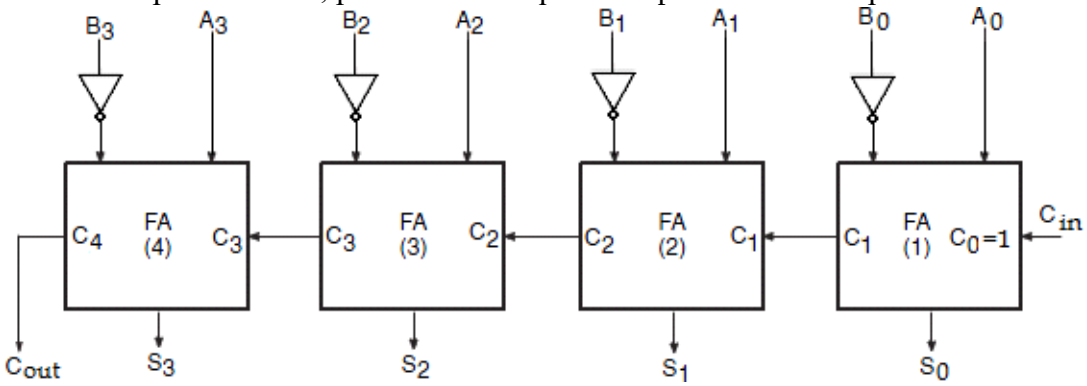


Fig. 4-bit Parallel Subtractor

3.8 Parallel Adder/ Subtractor:

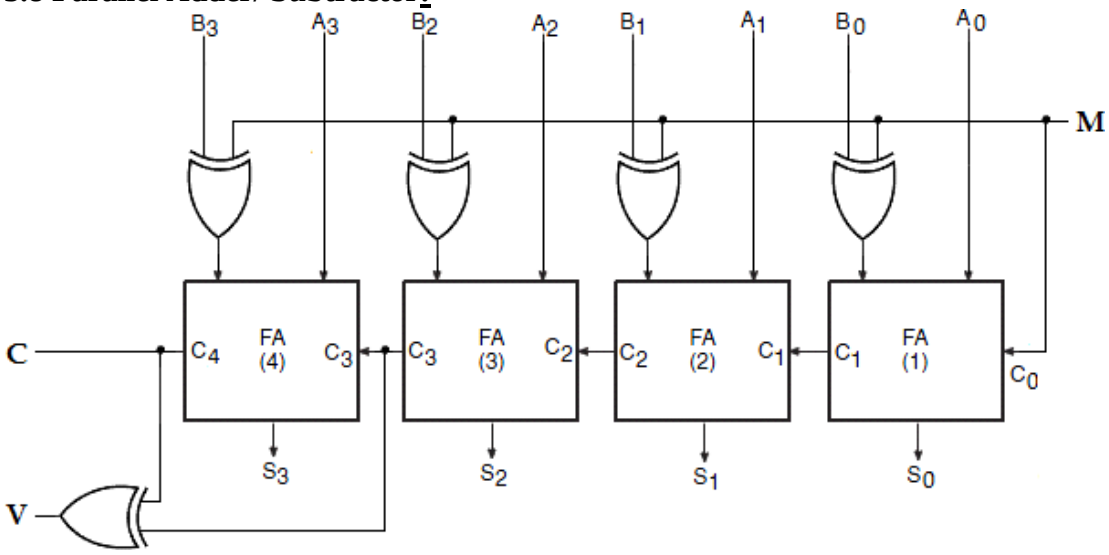
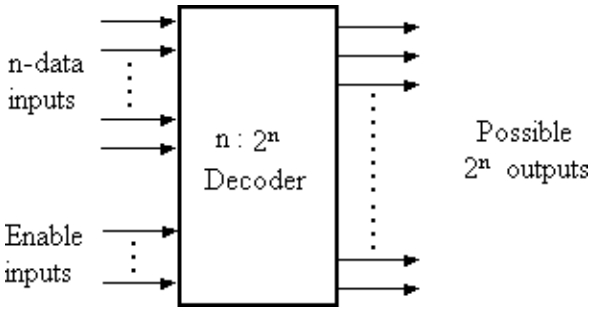


Fig. 4-Bit Adder Subtractor

The addition and subtraction operation can be combined into one circuit with one common binary adder. This is done by including an exclusive-OR gate with each full adder. A 4-bit adder Subtractor circuit is shown below.

The mode input M controls the operation. When M= 0, the circuit is an adder and when M=1, the circuit becomes a Subtractor. Each exclusive-OR gate receives input M and one of the inputs of B. When M=0, we have B Ex-OR 0 = B. The full adders receive the value of B, the input carry is 0, and the circuit performs A plus B. When M=1, we have B Ex –OR 1= B‘ and C0=1. The B inputs are all complemented and a 1 is added through the input carry. The circuit performs the operation A plus the 2‘s complement of B. The exclusive-OR with output V is for detecting an overflow.

3.9 DECODERS:

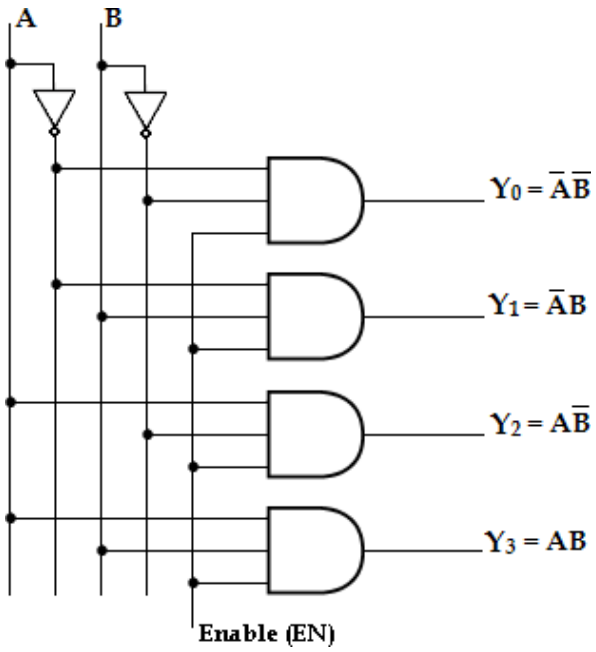


General structure of decoder

A decoder is a combinational circuit that converts binary information from _n‘ input lines to a maximum of _2n‘ unique output lines. The encoded information is presented as _n‘ inputs producing _2n‘ possible outputs. The 2n output values are from 0 through 2n-1. A decoder is provided with enable inputs to activate decoded output based on data inputs. When any one enable input is unasserted, all outputs of decoder are disabled.

Binary Decoder (2 to 4 decoder):

A binary decoder has _n‘ bit binary input and a one activated output out of 2^n outputs. A binary decoder is used when it is necessary to activate exactly one of 2n outputs based on an n-bit input value.



2-to-4 Line decoder

Here the 2 inputs are decoded into 4 outputs, each output representing one of the minterms of the two input variables.

Inputs			Outputs			
Enable	A	B	Y_3	Y_2	Y_1	Y_0
0	x	x	0	0	0	0
1	0	0	0	0	0	1
1	0	1	0	0	1	0
1	1	0	0	1	0	0
1	1	1	1	0	0	0

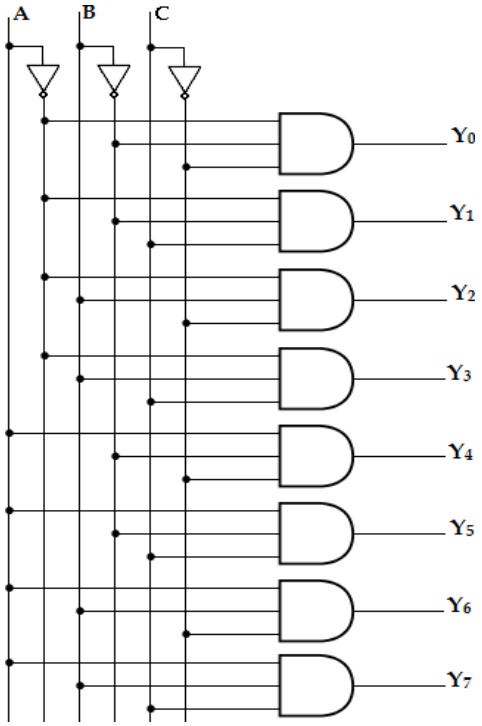
As shown in the truth table, if enable input is 1 (EN= 1) only one of the outputs ($Y_0 - Y_3$), is active for a giveninput. The output Y_0 is active, ie., Y_0= 1 when inputs A= B= 0, Y_1 is active when inputs, A= 0 and B= 1, Y_2 is active, when input A= 1 and B= 0, Y_3 is active, when inputs A= B= 1.

3 to-8 Line Decoder:

A 3-to-8 line decoder has three inputs (A, B, C) and eight outputs (Y_0- Y_7). Based on the 3 inputs one of the eight outputs is selected.

The three inputs are decoded into eight outputs, each output representing one of the minterms of the 3-input variables. This decoder is used for binary-to-octal conversion. The input variables may represent a binary number and the outputs will represent the eight digits in the octal number system. The output variables are mutually exclusive because only one output can be equal to 1 at any one time. The output line whose value is equal to 1 represents the minterm equivalent of the binary number presently available in the input lines.

Inputs			Outputs							
A	**B**	**C**	**Y_0**	**Y_1**	**Y_2**	**Y_3**	**Y_4**	**Y_5**	**Y_6**	**Y_7**
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1



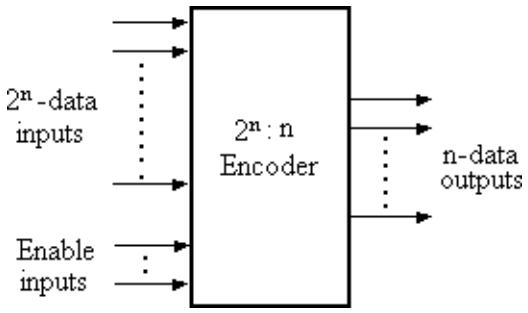
3-to-8 line decoder

Applications of decoders:

- Decoders are used in counter system.
- They are used in analog to digital converter.
- Decoder outputs can be used to drive a display system.

3.10 ENCODERS:

An encoder is a digital circuit that performs the inverse operation of a decoder. Hence, the opposite of the decoding process is called encoding. An encoder is a combinational circuit that converts binary information from 2^n input lines to a maximum of _n' unique output lines.



General structure of Encoder

It has 2^n input lines, only one which 1 is active at any time and _n' output lines. It encodes one of the active inputs to a coded binary output with _n' bits. In an encoder, the number of outputs is less than the number of inputs.

Octal-to-Binary Encoder:

It has eight inputs (one for each of the octal digits) and the three outputs that generate the corresponding binary number. It is assumed that only one input has a value of 1 at any given time.

Inputs								Outputs		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	A	B	C
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

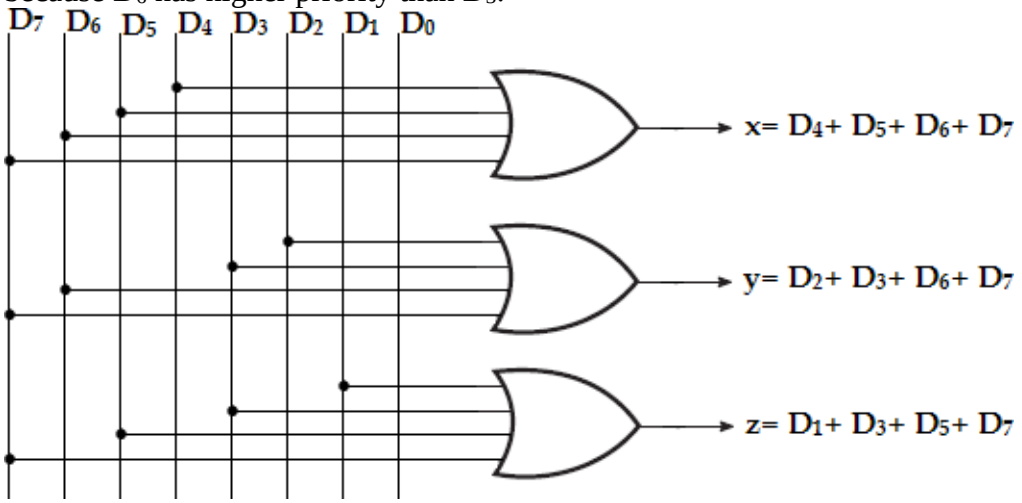
The encoder can be implemented with OR gates whose inputs are determined directly from the truth table. Output z is equal to 1, when the input octal digit is 1 or 3 or 5 or 7. Output y is 1 for octal digits 2, 3, 6, or 7 and the output is 1 for digits 4, 5, 6 or 7. These conditions can be expressed by the following output Boolean functions:

$z = D_1 + D_3 + D_5 + D_7$

$y = D_2 + D_3 + D_6 + D_7$ $x = D_4 + D_5 + D_6 + D_7$

The encoder can be implemented with three OR gates. The encoder defined in the below table, has the limitation that only one input can be active at any given time. If two inputs are active simultaneously, the output produces an undefined combination.

For eg., if D_3 and D_6 are 1 simultaneously, the output of the encoder may be 111. This does not represent either D_6 or D_3. To resolve this problem, encoder circuits must establish an input priority to ensure that only one input is encoded. If we establish a higher priority for inputs with higher subscript numbers and if D_3 and D_6 are 1 at the same time, the output will be 110 because D_6 has higher priority than D_3.



Octal-to-Binary Encoder

Another problem in the octal-to-binary encoder is that an output with all 0's is generated when all the inputs are 0; this output is same as when D_0 is equal to 1. The discrepancy can be resolved by providing one more output to indicate that atleast one input is equal to 1.

Priority Encoder:

A priority encoder is an encoder circuit that includes the priority function. In priority encoder, if two or more inputs are equal to 1 at the same time, the input having the highest priority will take precedence.

In addition to the two outputs x and y, the circuit has a third output, V (valid bit indicator). It is set to 1 when one or more inputs are equal to 1. If all inputs are 0, there is no valid input and V is equal to 0.

The higher the subscript number, higher the priority of the input. Input D_3, has the highest priority. So, regardless of the values of the other inputs, when D_3 is 1, the output for xy is 11.

D_2 has the next priority level. The output is 10, if D_2= 1 provided D_3= 0. The output for D_1 is generated only if higher priority inputs are 0, and so on down the priority levels.

Truth table:

Inputs				Outputs		
D_0	**D_1**	**D_2**	**D_3**	**x**	**y**	**V**
0	0	0	0	x	x	0

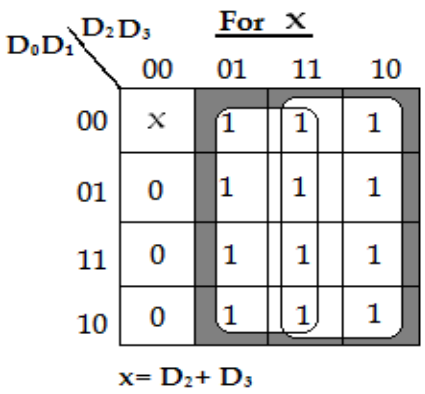
1	0	0	0	0	0	1
x	1	0	0	0	1	1
x	x	1	0	1	0	1
x	x	x	1	1	1	1

Although the above table has only five rows, when each don't care condition is replaced first by 0 and then by 1, we obtain all 16 possible input combinations. For example, the third row in the table with X100 represents minterms 0100 and 1100. The don't care condition is replaced by 0 and 1 as shown in the table below.

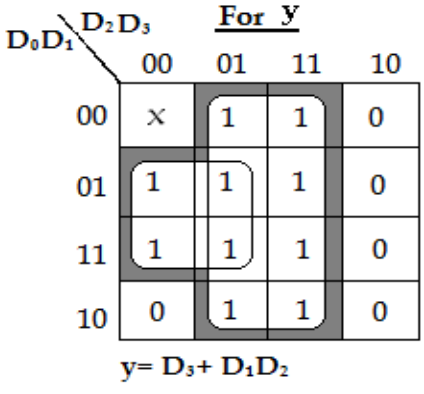
Modified Truth table:

Inputs				**Outputs**		
D_0	**D_1**	**D_2**	**D_3**	**X**	**y**	**V**
0	0	0	0	X	x	0
1	0	0	0	0	0	1
0	1	0	0	0	1	1
1	1	0	0			
0	0	1	0			
0 1	1 0	1 1	0 0	1	0	1
1	1	1	0			
0	0	0	1			
0	0	1	1			
0	1	0	1			
0	1	1	1	1	1	1
1	0	0	1			
1	0	1	1			
1	1	0	1			
1	1	1	1			

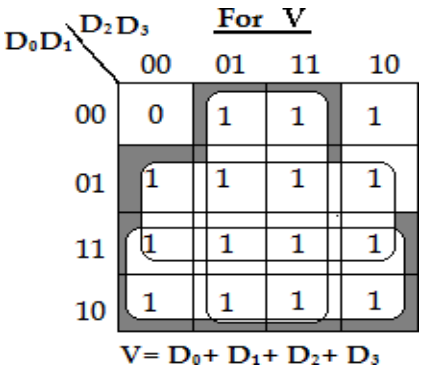
K-map Simplification:



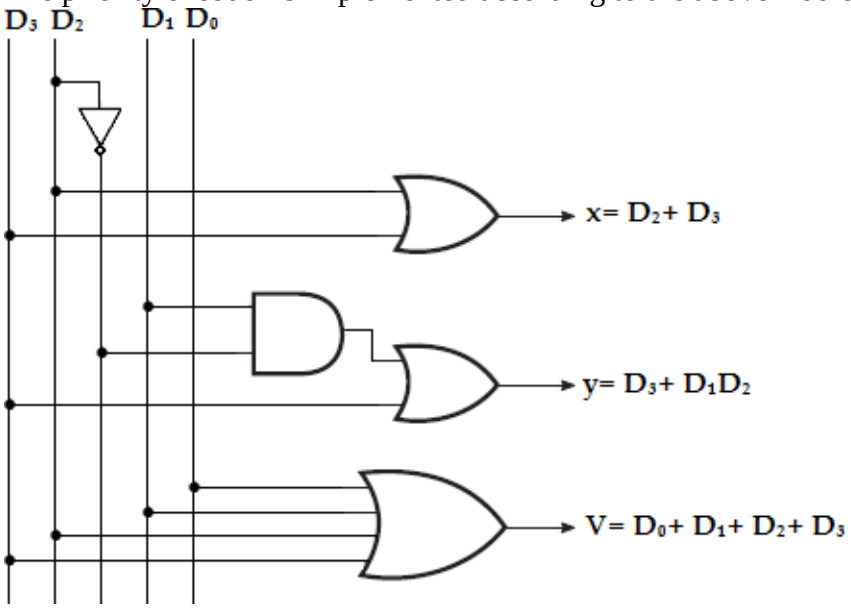
$x = D_2 + D_3$



$y = D_3 + D_1D_2$



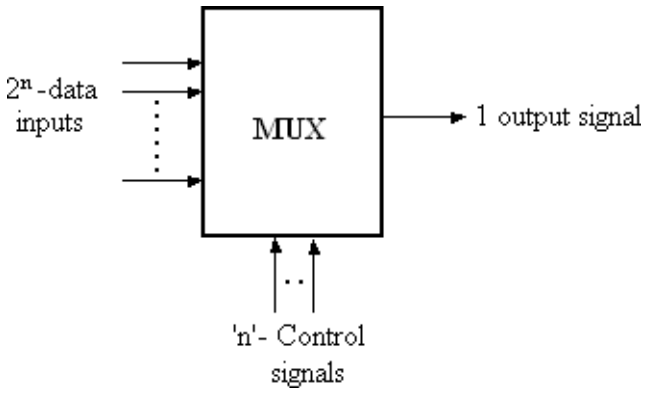
The priority encoder is implemented according to the above Boolean functions.



- **Logic Diagram of Priority Encoder**

3.11 MULTIPLEXER: (Data Selector)

A multiplexer or MUX, is a combinational circuit with more than one input line, one output line and more than one selection line. A multiplexer selects binary information present from one of many input lines, depending upon the logic status of the selection inputs, and routes it to the output line. Normally, there are 2^n input lines and n selection lines whose bit combinations determine which input is selected. The multiplexer is often labeled as MUX in block diagrams.

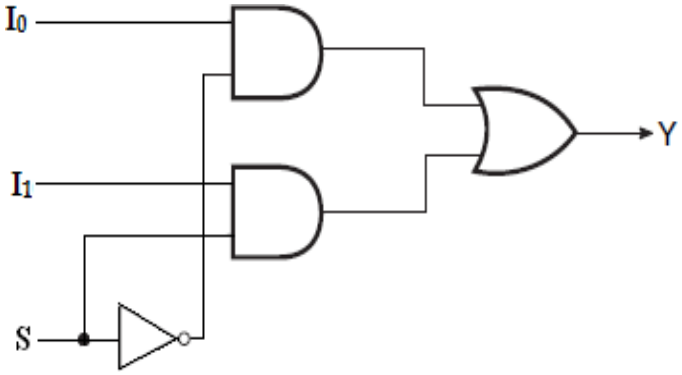


Block diagram of Multiplexer

A multiplexer is also called a **data selector**, since it selects one of many inputs and steers the binary information to the output line.

2-to-1- line Multiplexer:

The circuit has two data input lines, one output line and one selection line, S. When S= 0, the upper AND gate is enabled and I_0 has a path to the output.
When S=1, the lower AND gate is enabled and I_1 has a path to the output.



Logic diagram

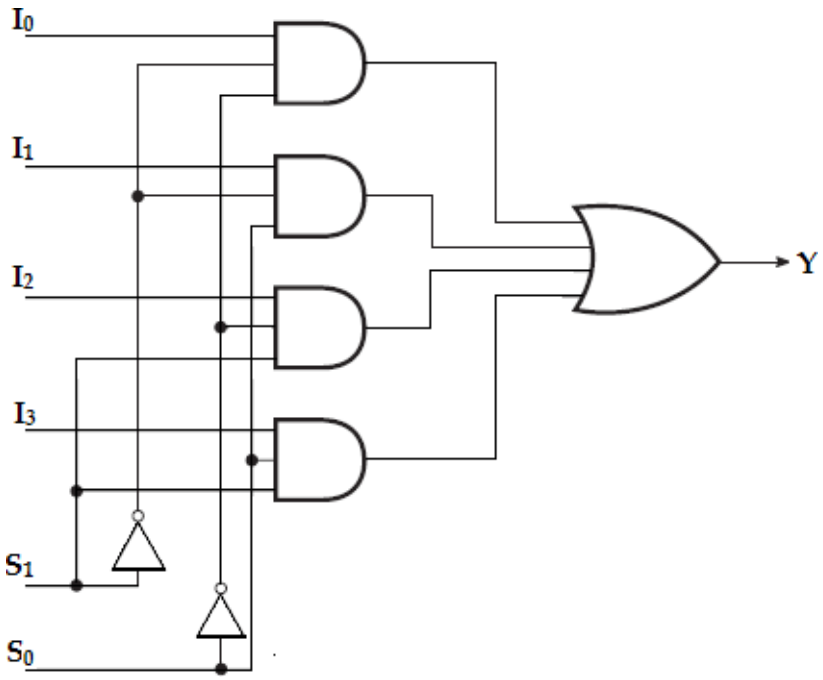
The multiplexer acts like an electronic switch that selects one of the two sources.

Truth table:

S	Y
0	I_0
1	I_1

4-to-1-line Multiplexer:

A 4-to-1-line multiplexer has four (2^n) input lines, two (n) select lines and one output line. It is the multiplexer consisting of four input channels and information of one of the channels can be selected and transmitted to an output line according to the select inputs combinations. Selection of one of the four input channel is possible by two selection inputs.



4-to-1-Line Multiplexer

Each of the four inputs I_0 through I_3, is applied to one input of AND gate. Selection lines S_1 and S_0 are decoded to select a particular AND gate. The outputs of the AND gate are applied to a single OR gate that provides the 1-line output.

Function table:

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

To demonstrate the circuit operation, consider the case when S_1S_0= 10. The AND gate associated with input I_2 has two of its inputs equal to 1 and the third input connected to I_2. The other three AND gates have atleast one input equal to 0, which makes their outputs equal to 0. The OR output is now equal to the value of I_2, providing a path from the selected input to the output.

The data output is equal to I_0 only if S_1= 0 and S_0= 0; Y= I_0S_1'S0'. The data output is equal to I_1 only if S_1= 0 and S_0= 1; Y= I_1S_1'S0. The data output is equal to I_2 only if S_1= 1 and S_0= 0; Y= $I_2S_1S_0$'. The data output is equal to I_3 only if S_1= 1 and S_0= 1; Y= $I_3S_1S_0$.
When these terms are ORed, the total expression for the data output is,

Y= I_0S_1'S0'+ I1S_1'S0 +$I_2S_1S_0$'+ I3S_1S_0.

As in decoder, multiplexers may have an enable input to control the operation of the unit. When the enable input is in the inactive state, the outputs are disabled, and when it is in the active state, the circuit functions as a normal multiplexer.

Application:

The multiplexer is a very useful MSI function and has various ranges of applications in data communication. Signal routing and data communication are the important applications of a multiplexer. It is used for connecting two or more sources to guide to a single destination among computer units and it is useful for constructing a common bus system. One of the general properties of a multiplexer is that Boolean functions can be implemented by this device.

Implementation of Boolean Function using MUX:

Any Boolean or logical expression can be easily implemented using a multiplexer. If a Boolean expression has (n+1) variables, then _n' of these variables can be connected to the select lines of the multiplexer. The remaining single variable along with constants 1 and 0 is used as the input of the multiplexer. For example, if C is the single variable, then the inputs of the multiplexers are C, C', 1 and 0. By this method any logical expression can be implemented. In general, a Boolean expression of (n+1) variables can be implemented using a multiplexer with 2^n inputs.

1.Implement the following boolean function using 4: 1 multiplexer,
F (A, B, C) = ∑m (1, 3, 5, 6).

Implementation table:

Apply variables A and B to the select lines. The procedures for implementing the function are:

- List the input of the multiplexer
- List under them all the minterms in two rows as shownbelow.

The first half of the minterms is associated with A' and the second half with A. The given function is implemented by circling the minterms of the function and applying the following rules to find the values for the inputs of the multiplexer.

- If both the minterms in the column are not circled, apply 0 to the corresponding input.
- If both the minterms in the column are circled, apply 1 to the corresponding input.
- If the bottom minterm is circled and the top is not circled, apply C to the input.
- If the top minterm is circled and the bottom is not circled, apply C' to the input.

Implementation Table:

	D_0	D_1	D_2	D_3
$\overline{C}$	0	(1)	2	(3)
C	4	(5)	(6)	7
	0	**1**	**C**	$\overline{C}$

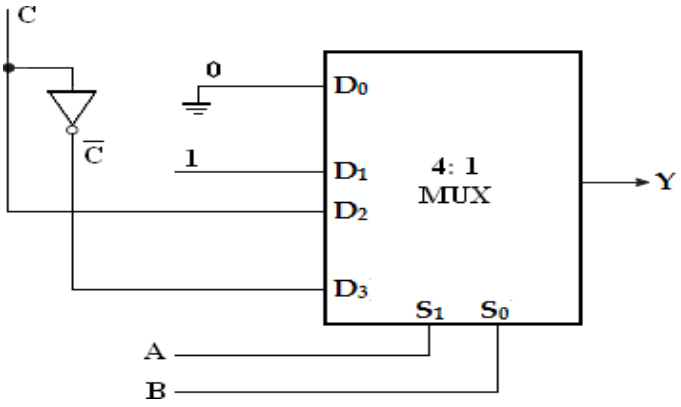


Fig. Multiplexer Implementation

2. F (x, y, z) = ∑m (1, 2, 6, 7)

Solution: Implementation table:

	D_0	D_1	D_2	D_3
$\overline{z}$	0	(1)	(2)	3
z	4	5	(6)	(7)
	0	$\overline{z}$	**1**	**z**

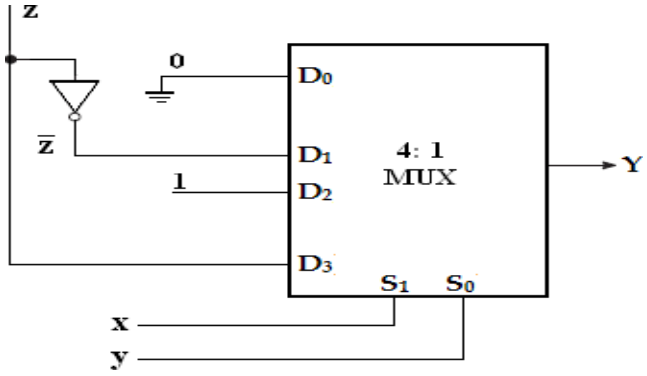


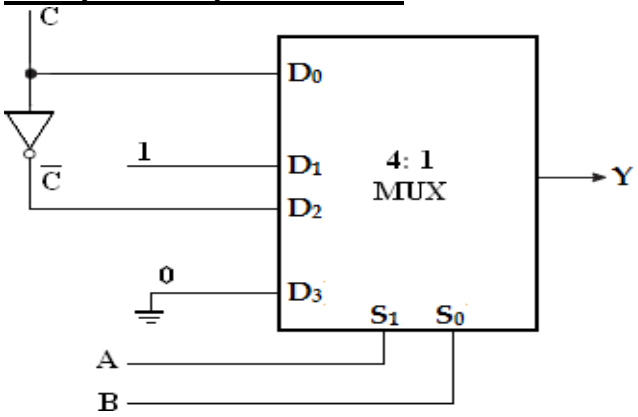
Fig. Multiplexer Implementation

3. **F (A, B, C) = ∑m (1, 2, 4, 5)**

	D_0	D_1	D_2	D_3
$\overline{C}$	0	(1)	(2)	3
C	(4)	(5)	6	7
	C	**1**	$\overline{C}$	**0**

Fig. Implementation table

Multiplexer Implementation:

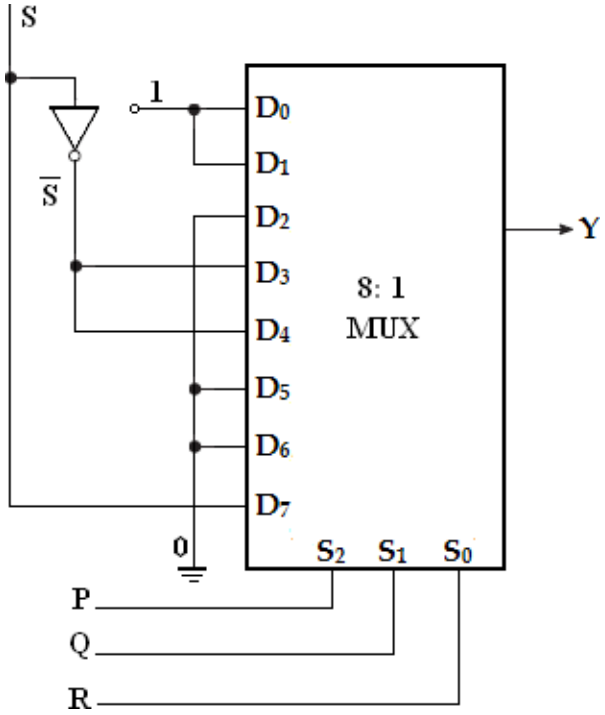


4. F(P, Q, R, S)= ∑m (0, 1, 3, 4, 8, 9, 15)

Implementation table:

	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
$\overline{S}$	(0)	(1)	2	(3)	(4)	5	6	7
S	(8)	(9)	10	11	12	13	14	(15)
	1	**1**	**0**	$\overline{S}$	$\overline{S}$	**0**	**0**	**S**

Multiplexer Implementation:

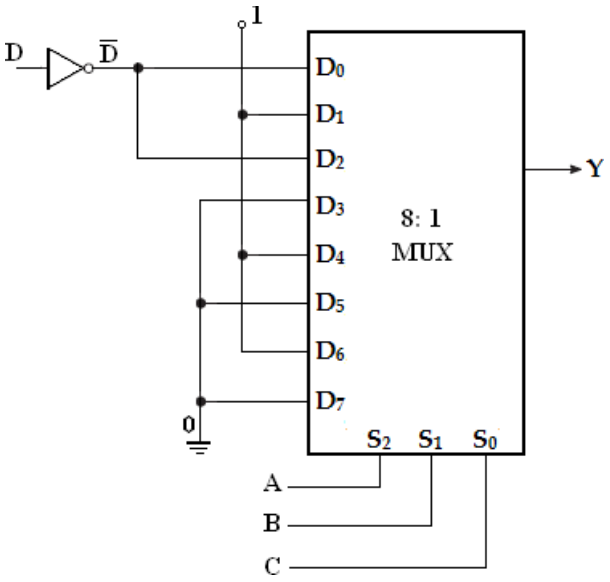


5)Implement the Boolean function using 8: 1 and also using 4:1 multiplexer
F (A, B, C, D) = ∑m (0, 1, 2, 4, 6, 9, 12, 14)

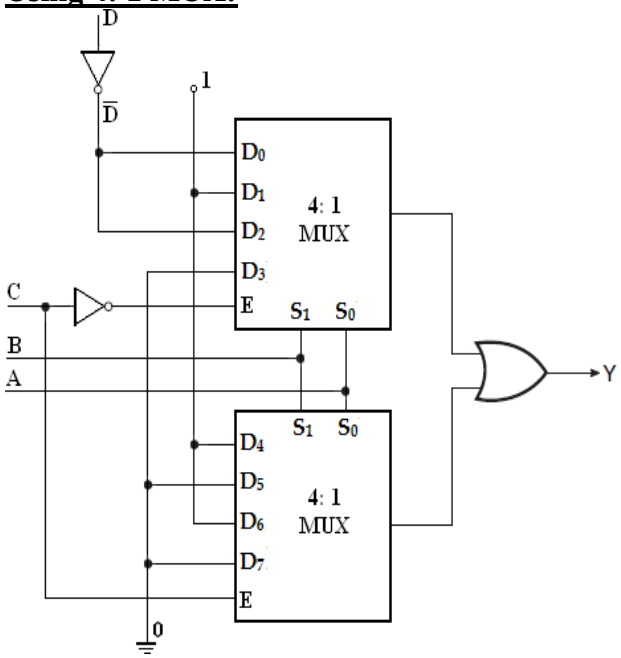
Implementation table:

	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
$\overline{D}$	(0)	(1)	(2)	3	(4)	5	(6)	7
D	8	(9)	10	11	(12)	13	(14)	15
	$\overline{D}$	**1**	$\overline{D}$	**0**	**1**	**0**	**1**	**0**

Multiplexer Implementation (Using 8: 1 MUX):



Using 4: 1 MUX:



6. F (A, B, C, D) = ∑m (1, 3, 4, 11, 12, 13, 14, 15)

Solution:

Variables, n= 4 (A, B, C, D) Select lines= n-1 = 3 (**S_2, S_1, S_0**)

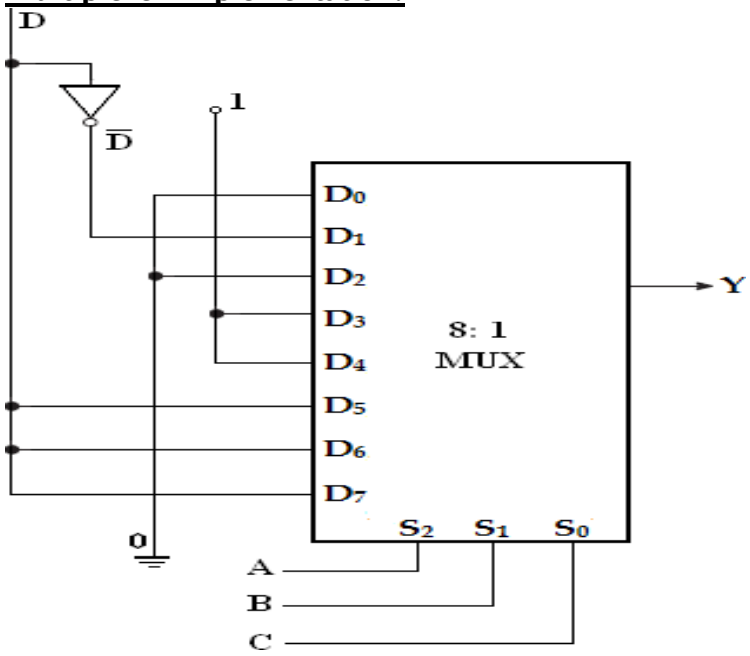
2^{n-1} to MUX i.e., 23 to 1 = 8 to 1 MUX

Input lines= $2^{n-1} = 2^3 = 8$ (**D_0, D_1, D_2, D_3, D_4, D_5, D_6, D_7**)

Implementation table:

	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
$\overline{D}$	0	(1)	2	(3)	(4)	5	6	7
D	8	9	10	(11)	(12)	(13)	(14)	(15)
	0	$\overline{D}$	**0**	**1**	**1**	D	D	D

Multiplexer Implementation:



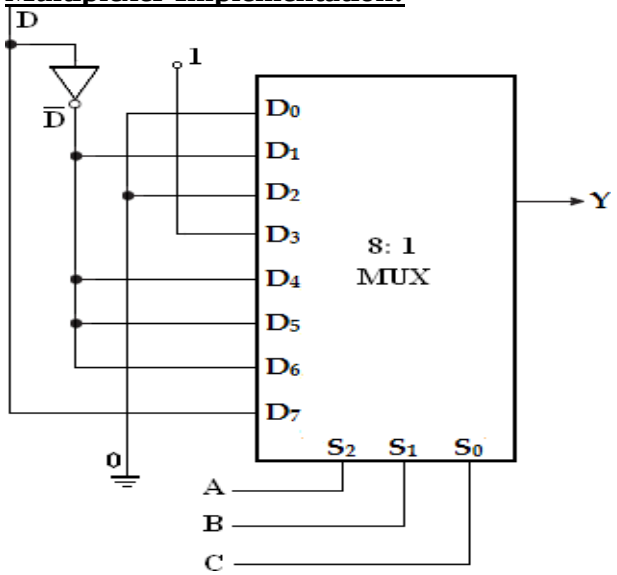
7)Implement the Boolean function using 8: 1multiplexer.
F (A, B, C, D) = A'BD' + ACD + B'CD + A'C'D.

Solution:
Convert into standard SOP form,
= A'BD' (C'+C) + ACD (B'+B) + B'CD (A'+A) + A'C'D (B'+B)
= A'BC'D' + A'BCD'+ AB'CD + ABCD +A'B'CD + AB'CD +A'B'C'D+ A'BC'D
= A'BC'D' + A'BCD'+ AB'CD + ABCD +A'B'CD +A'B'C'D+ A'BC'D
= m4+ m6+ m11+ m15+ m3+ m1+ m5
= ∑m (1, 3, 4, 5, 6, 11, 15)

Implementation table:

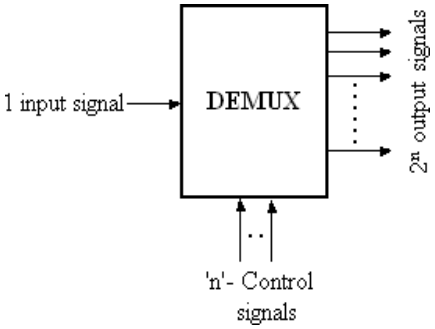
	D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7
$\overline{D}$	0	(1)	2	(3)	(4)	(5)	(6)	7
D	8	9	10	(11)	12	13	14	(15)
	0	$\overline{D}$	**0**	**1**	$\overline{D}$	$\overline{D}$	$\overline{D}$	D

Multiplexer Implementation:



3.12 DEMULTIPLEXER:

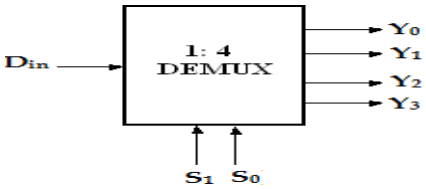
Demultiplex means one into many. Demultiplexing is the process of taking information from one input and transmitting the same over one of several outputs.
A demultiplexer is a combinational logic circuit that receives information on a single input and transmits the same information over one of several (2^n) output lines.



Block diagram of Demultiplexer

The block diagram of a demultiplexer which is opposite to a multiplexer in its operation is shown above. The circuit has one input signal, _n' select signals and 2^n output signals. The select inputs determine to which output the data input will be connected. As the serial data is changed to parallel data, i.e., the input caused to appear on one of the n output lines, the demultiplexer is also called a —data distributer‖ or a —serial-to-parallel converter‖ .

1-to-4 Demultiplexer:



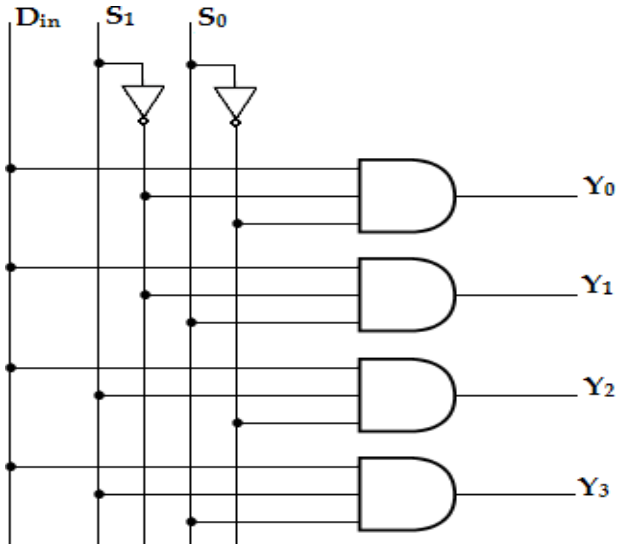
Logic Symbol

A 1-to-4 demultiplexer has a single input, **D_{in}**, four outputs (**Y_0 to Y_3**) and two select inputs (**S_1 and S_0**). The input variable D_{in} has a path to all four outputs, but the input information is directed to only one of the output lines. The truth table of the 1-to-4 demultiplexer is shown below.

Truth table of 1-to-4 demultiplexer

Enable	S_1	S_0	D_{in}	Y_0	Y_1	Y_2	Y_3
0	x	x	X	0	0	0	0
1	0	0	0	0	0	0	0
1	0	0	1	1	0	0	0
1	0	1	0	0	0	0	0
1	0	1	1	0	1	0	0
1	1	0	0	0	0	0	0
1	1	0	1	0	0	1	0
1	1	1	0	0	0	0	0
1	1	1	1	0	0	0	1

From the truth table, it is clear that the data input, D_{in} is connected to the output Y_0, when S_1= 0 and S_0= 0 and the data input is connected to output Y_1 when S_1= 0 and S_0= 1. Similarly, the data input is connected to output Y_2 and Y_3 when S_1= 1 and S_0= 0 and when S_1= 1 and S_0= 1, respectively. Also, from the truth table, the expression for outputs can be written as follows,



Logic diagram of 1-to-4 demultiplexer

Y_0= S_1'S0'Din Y_1= S_1'S0D_{in} Y_2= S_1S_0'Din Y3= S1S0Din

Now, using the above expressions, a 1-to-4 demultiplexer can be implemented using four 3-input AND gates and two NOT gates. Here, the input data line D_{in}, is connected to all the AND gates. The two select lines S_1, S_0 enable only one gate at a time and the data that appears on the input line passes through the selected gate to the associated output line.

<u>1-to-8 Demultiplexer</u>:

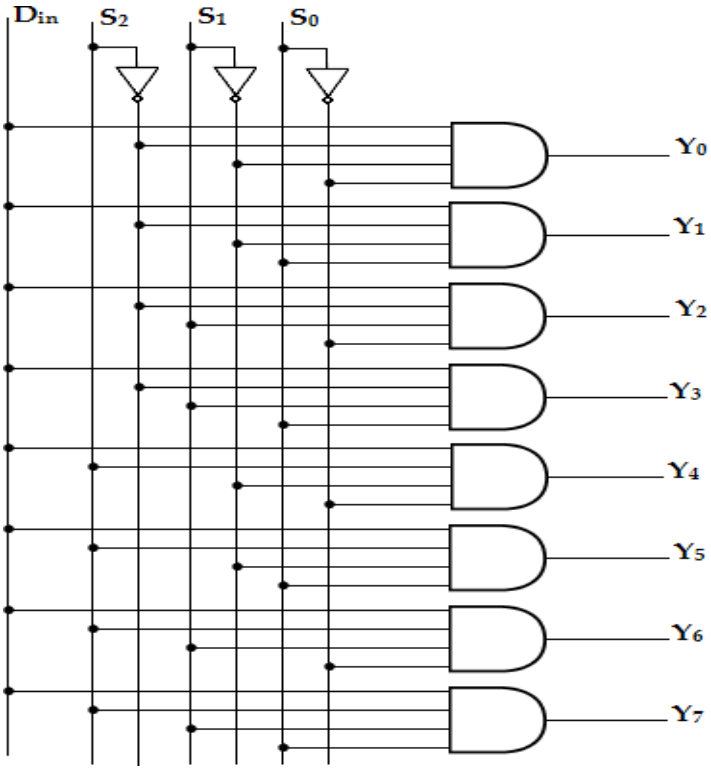
A 1-to-8 demultiplexer has a single input, **D_{in}**, eight outputs (**Y_0 to Y_7**) and three select inputs (**S_2, S_1 and S_0**). It distributes one input line to eight output lines based on the select inputs. The truth table of 1-to-8 demultiplexer is shown below.

Truth table of 1-to-8 demultiplexer

D_{in}	S_2	S_1	S_0	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
0	x	x	x	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0	0	0	1	0
1	0	1	0	0	0	0	0	0	1	0	0
1	0	1	1	0	0	0	0	1	0	0	0
1	1	0	0	0	0	0	1	0	0	0	0
1	1	0	1	0	0	1	0	0	0	0	0
1	1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0

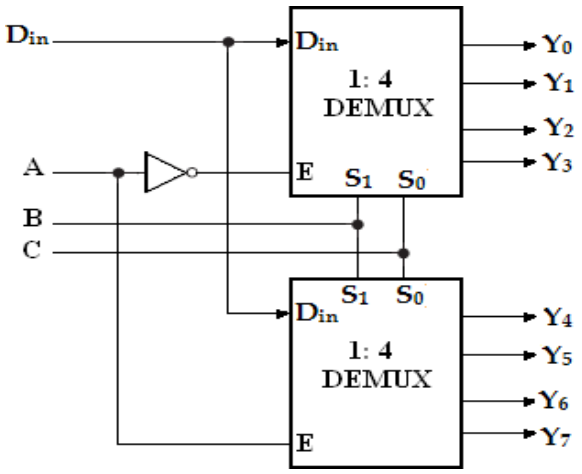
From the above truth table, it is clear that the data input is connected with one of the eight outputs based on the select inputs. Now from this truth table, the expression for eight outputs can be written as follows:

$Y_0 = S_2$‘$S1$‘$S0$‘Din $Y_4 = S_2\ S_1$‘$S0$‘Din $Y_1 = S_2$‘$S1$‘$S0D_{in}$ $Y_5 = S_2\ S_1$‘$S0D_{in}$
$Y_2 = S_2$‘$S1S_0$‘Din $Y_6 = S_2\ S_1S_0$‘Din $Y_3 = S_2$‘$S1S_0D_{in}$ $Y_7 = S_2S_1S_0D_{in}$



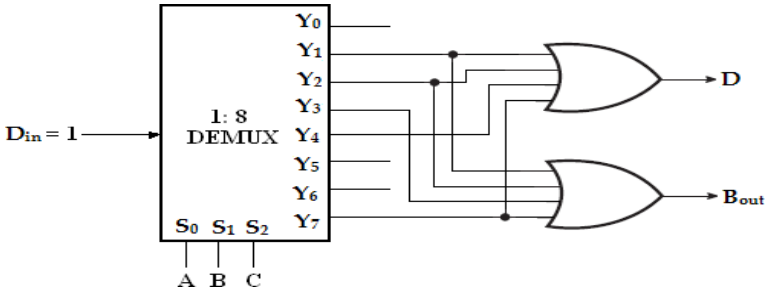
Logic diagram of 1-to-8 demultiplexer

1)Design 1:8 demultiplexer using two 1:4DEMUX.



2)Implement full subtractor using demultiplexer.

Inputs			Outputs	
A	**B**	**B_{in}**	**Difference(D)**	**Borrow(B_{out})**
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



3.13 MAGNITUDE COMPARATOR:

A magnitude comparator is a combinational circuit that compares two given numbers (A and B) and determines whether one is equal to, less than or greater than the other. The output is in the form of three binary variables representing the conditions A= B, A>B and A<B, if A and B are the two numbers being compared.

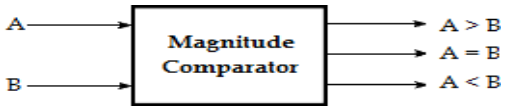


Fig. Block diagram of magnitude comparator

For comparison of two n-bit numbers, the classical method to achieve the Boolean expressions requires a truth table of 2^{2n} entries and becomes too lengthy and cumbersome.

2-bit Magnitude Comparator:

The truth table of 2-bit comparator is given in table below— Truth table:

Inputs				Outputs		
A_3	A_2	A_1	A_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

K-map Simplification:

For A>B

A_1A_0 \ B_1B_0	00	01	11	10
00	0	0	0	0
01	1	0	0	0
11	1	1	0	1
10	1	1	0	0

$A>B = A_0B_1'B_0' + A_1B_1' + A_1A_0B_0'$

For A=B

A_1A_0 \ B_1B_0	00	01	11	10
00	1	0	0	0
01	0	1	0	0
11	0	0	1	0
10	0	0	0	1

$A=B = A_1'A_0'B_1'B_0' + A_1'A_0B_1'B_0 + A_1A_0B_1B_0 + A_1A_0'B_1B_0'$

$= A_1'B_1' (A_0'B_0' + A_0B_0) + A_1B_1 (A_0B_0 + A_0'B_0')$

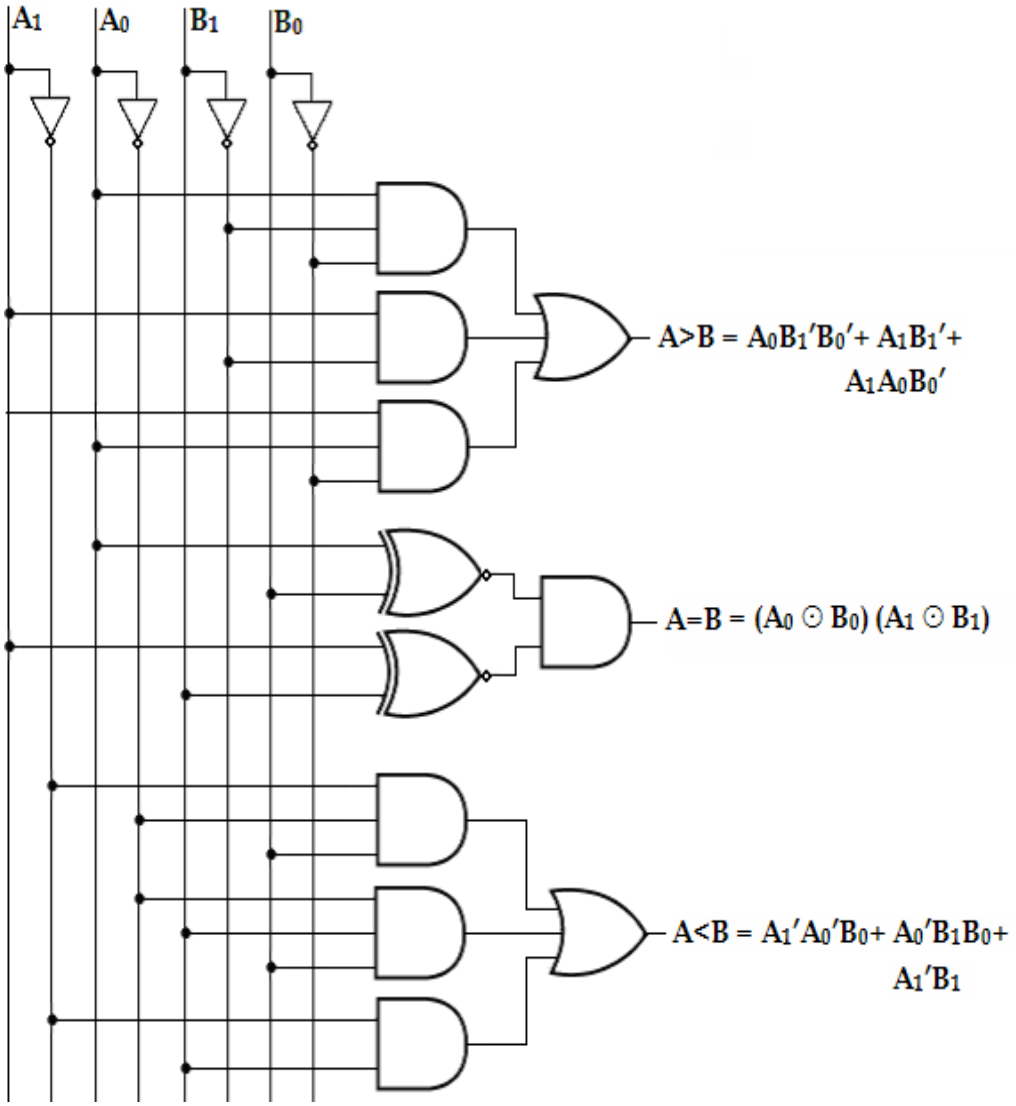
$= (A_0 \odot B_0)(A_1 \odot B_1)$

For A<B

A_1A_0 \ B_1B_0	00	01	11	10
00	0	1	1	1
01	0	0	1	1
11	0	0	0	0
10	0	0	1	0

$A<B = A_1'A_0'B_0 + A_0'B_1B_0 + A_1'B_1$

Logic Diagram:



References

1.Morris Mano, Digital Design, Prentice Hall of India, 2001.

2. S.Salivahanan and S.Arivazhagan―Digital Electronics‖, Ist Edition, Vikas Publishing House pvt Ltd, 2012.

3.Ronald J.Tocci, Digital System Principles and Applications, PHI, 6th Edition, 1997

Question Bank

Part A

1. Define combinational logic circuit
2. Define half adder and full adder
3. Define half subtractor and full subtractor
4. Draw a block diagram of half adder and show the truth table
5. Draw the schematic of a full subtractor circuit and show the truth table of full subtractor.
6. Draw the schematic of a full adder circuit and give its truth table.
7. Write the logic expressions for the difference and borrow of a full subtractor.
8. Design a 4 bit parallel adder
9. Define multiplexer
10. Write the truth table of 4:1 multiplexer.
11. Differentiate between multiplexer and demultiplexer.
12. Demonstrate priority encoder.
13. List the purpose of magnitude comparator?
14. Draw the circuit diagram and truth table of 4 to 2 encoder.

Part B

1. Develop a full adder using two half adders.
2. Design a half adder combinational circuit.
3. Explain the operation of full adder with the help of logic diagram and truth table
4. Draw and explain the working of 4 bit parallel adder and 4 bit parallel subtractor.
5. Construct 4-bit parallel adder/subtractor using Full adders and EXOR gates
6. Draw the circuit of 3 to 8 decoder and explain
7. Design and implement a full adder circuit using a 3:8 decoder.
8. Explain the octal to binary encoder.
9. Design a 4 bit priority encoder.
10. Describe the working of 8:1 multiplexer
11. Implement the following Boolean function using 4:1 multiplexer.F(A,B,C)= Σm (1,3,5,6)
12. Design a 1:8 demultiplexer.
13. Design a 2 bit comparator using gates.

SATHYABAMA
INSTITUTE OF SCIENCE AND TECHNOLOGY
(DEEMED TO BE UNIVERSITY)
Accredited "A" Grade by NAAC | 12B Status by UGC | Approved by AICTE
www.sathyabama.ac.in

SCHOOL OF BIO AND CHEMICAL ENGINEERING

DEPARTMENT OF BIOMEDICAL ENGINEERING

UNIT – IV – Fundamentals of Digital Systems – SBMA1401

IV. Sequential Circuits

Flipflops- SR, JK, T, D, Master slave FF, Characteristic and Excitation table, Shift Registers, Counters –two bit and three bit Asynchronous and Synchronous Counters -UP/DOWN Counter, State Diagram representation of filp flops, State Minimization Techniques, State Assignment.

4.1 Sequential Circuits

This sequential circuit contains a set of inputs and outputs. The outputs of sequential circuit depends not only on the combination of present inputs but also on the previous outputs. Previous output is nothing but the **present state**. Therefore, sequential circuits contain combinational circuits along with memory storage elements. A sequential circuit is a combinational circuit with some feedback from the outputs. The memory elements are connected to the combinational circuit as a feedback path.

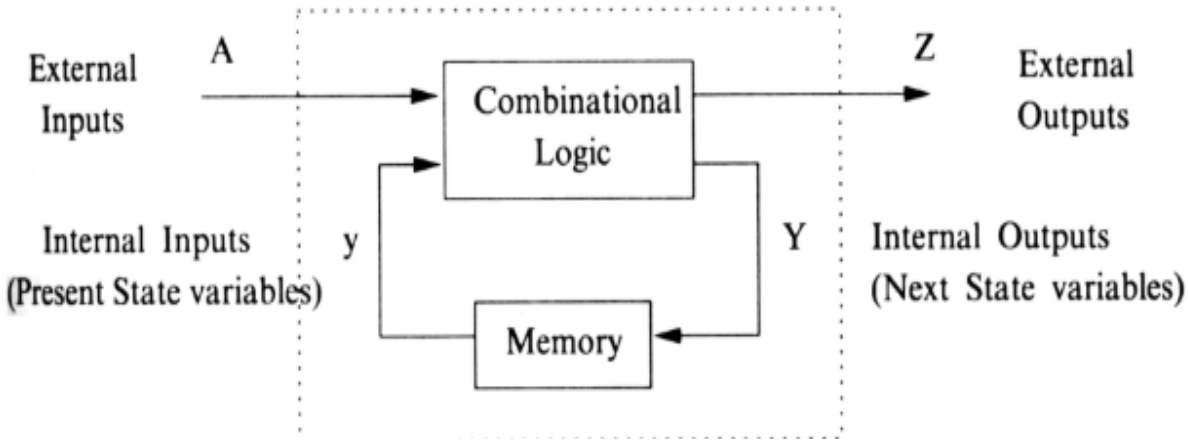


Figure : Sequential logic circuit

The information stored in the memory elements at any given time defines the present state of the sequential circuit. The present state and the external inputs determine the outputs and next sate of the sequential circuit. Thus we can specify the sequential circuit by a time sequence of external inputs,internal states (present state and next states) and outputs.

The memory element used in sequential circuits is a flipflop which is capable of storing 1 bit binary information.

Differences between combinational circuits and sequential circuits

Combinational Circuits	Sequential Circuits
1. The circuit whose output at any instant depends only on the input present at that instant only is known as combinationational circuit.	1. The circuit whose output at any instant depends not only on the input present but also on the past output a is known as sequential circuit
2. This type of circuit has no memory unit.	2. This type of circuit has memory unit for store past output.
3. Examples of combinational circuits are half adder, full adder, magnitude comparator, multiplexer, demultiplexer e.t.c.	3. Examples of sequential circuits are Flip flop, register, counter e.t.c.
4. Faster in Speed	Slower compared to Combinational Circuit

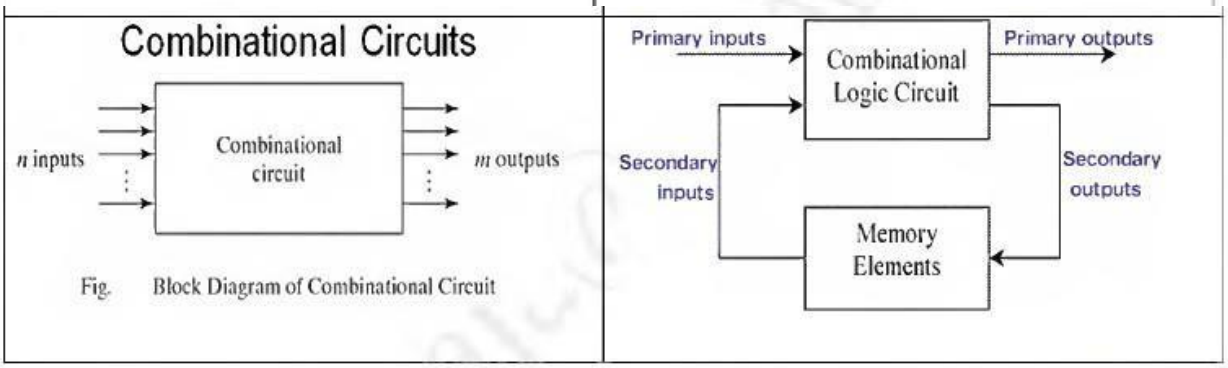


Fig. Block Diagram of Combinational Circuit

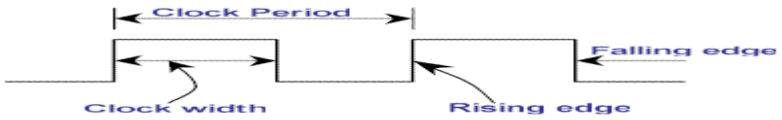
Types of Sequential Circuits

Following are the two types of sequential circuits −

- Asynchronous sequential circuits
- Synchronous sequential circuits

In asynchronous sequential circuits change in input signals can affect memory element at any instant of time. In Synchronous sequential circuits, signals can affect the memory elements only at discrete instants of time.

Clock Signal



A clock signal is a particular type of signal that oscillates between a high and a low state. It is produced by clock generator.

Types of Triggering

Level triggering

Edge triggering

Level triggering

- Positive level triggering
- Negative level triggering

If the sequential circuit is operated with the clock signal when it is in **Logic High**, then that type of triggering is known as **Positive level triggering**. It is highlighted in below figure.

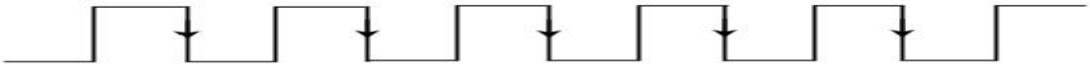
If the sequential circuit is operated with the clock signal when it is in **Logic Low**, then that type of triggering is known as **Negative level triggering**. It is highlighted in the following figure.

Edge triggering

- Positive edge triggering
- Negative edge triggering

If the sequential circuit is operated with the clock signal that is transitioning from Logic Low to Logic High, then this is known as **Positive edge triggering**. It is also called as rising edge triggering. It is shown in the following figure.

If the sequential circuit is operated with the clock signal that is transitioning from Logic High to Logic Low, then this is known as Negative edge triggering. It is also called as falling edge triggering. It is shown in the following figure.



4.2 Flip flop

A flip-flop or latch is a circuit that has two stable states and can be used to store state information. A flip-flop stores a single *bit* (binary digit) of data; one of its two states represents a "one" and the other represents a "zero". Latches and flip-flops are the basic elements for storing information. The main difference between latches and flip-flops is that, latches operate with enable signal, which is level sensitive whereas, flip-flops are edge sensitive. In latches, when they are enabled, their content changes immediately when their inputs change. Flip-flops, on the other hand, have their content change only either at the rising or falling edge of the enable signal. This enable signal is usually the controlling clock signal. After the rising or falling edge of the clock, the flip-flop content remains constant even if the input changes. There are mainly four types of flip flops that are used in electronic circuits. They are

1.SR Flipflop

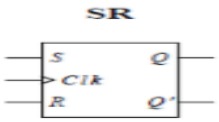
2.D Flipflop

3. JK Flipflop

4. T flipflop

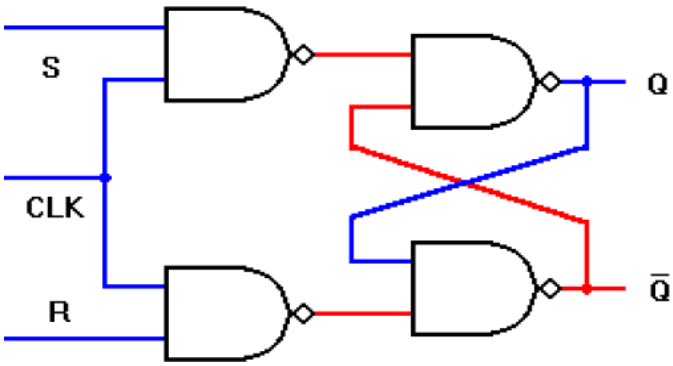
1.SR Flipflop

S - R flip - flop has 2 inputs, S (set) and R (reset) .



Logic symbol

Logic diagram of SR flipflop



Characteristic table of SR flipflop

CP	S	R	Q_n	Q_{n+1}	State
↑	0	0	0	0	No change(NC)
↑	0	0	1	1	
↑	0	1	0	0	Reset
↑	0	1	1	0	
↑	1	0	0	1	Set

↑	1	0	1	1	
↑	1	1	0	X	Indeterminate
↑	1	1	1	X	
0	x	x	0	0	No change(NC)
0	x	x	1	1	

Simplified truth table

S	R	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	X

Case 1:If S=R=0 and the clock pulse is applied, the output do not change, $Q_{n+1} = Q_n$

Case 2: If S= 0, R=1 and the clock pulse is applied, Q_{n+1}= 0

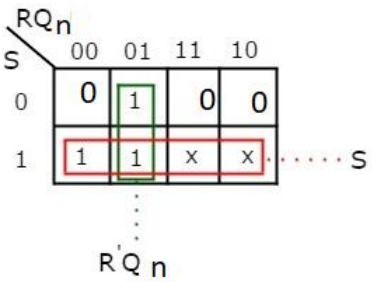
Case 3: If S= 1, R=0 and the clock pulse is applied, Q_{n+1}= 1

Case4: If S= 1, R=1 and the clock pulse is applied, the state of the flipflop is undefined.

Excitation table of SR flipflop

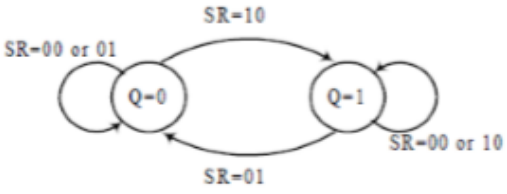
Q_n	Q_{n+1}	S	R
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

Characteristic equation of SR Flipflop



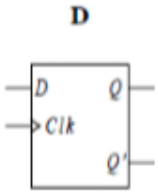
Q(n+1)=S+R′Q(t)

State diagram of SR Flipflop

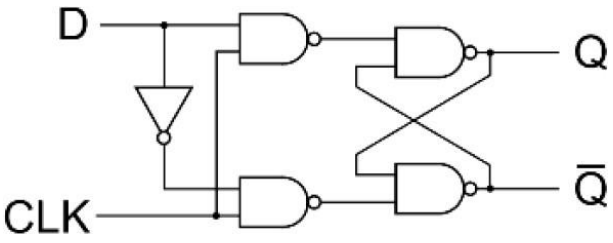


2. D Flipflop

The modified SR flipflop is known as delay flipflop (D Flipflop). D input is connected to S input and complement of D input is connected to R input of SR flipflop.



Logic symbol of D Flipflop



Logic diagram of D flipflop

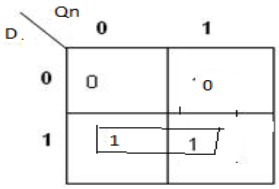
Two input conditions exist . (i)When D=1, S=1 and R=0 ; the ouput is set and (ii) When D=0, S=0 and R=1 ; the output is reset.

Characteristic table of D flipflop

CP	D	Q_n	Q_{n+1}
↑	0	0	0
↑	0	1	0
↑	1	0	1
↑	1	1	1
0	X	0	0
0	X	1	1

CP	D	Q_{n+1}
↑	0	0
↑	1	1
0	X	Q_n

Characteristic Equation

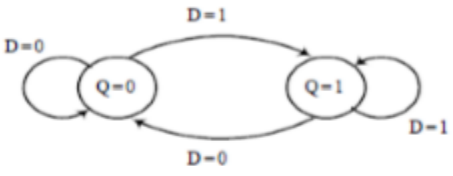


$Q_{n+1} = D$

Excitation table of D flipflop

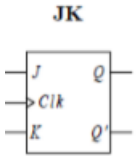
Q_n	Q_{n+1}	D
0	0	0
0	1	1
1	0	0
1	1	1

State diagram of D Flipflop

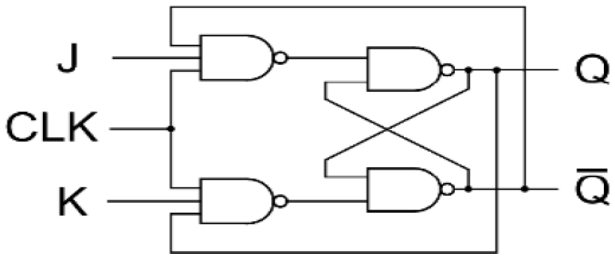


3. **JK Flipflop**

JK flipflop is also a modification of SR flipflop. The uncertainity in the state of an SR flipflop when S=R=1 is being eliminated in JK flipflop.



Logic symbol of JK Flipflop



Logic diagram of JK flipflop

Here, J input is ANDed with Q to obtain S input and K input is ANDed with Q' to obtain R input. Thus S= J.Q' and R= K.Q.

Case 1: J=K=0

When J=K=0, S=R=0 and according to truth table of SR flipflop, output does not change. Hence, when J=K=0; **Q_{n+1}= Q_n (no output change)**

Case 2: J=0 and K=1

Q=0, Q'=1 : When J=0 ,K=1 and Q=0; S=0 and R=0 then **Q_{n+1}=0.**

Q=1, Q'=0 : When J=0 ,K=1 and Q=1; S=0 and R=1 then **Q_{n+1}=0.**

Hence, when J=0 and K=1; **Q_{n+1}= 0(Reset)**

Case 3: J=1 and K=0

Q=0, Q'=1 : When J=1 ,K=0 and Q=0; S=1 and R=0 then Q_{n+1}=1.

Q=1, Q'=0 : When J=1 ,K=0 and Q=1; S=0 and R=0 then Q_{n+1}=1.

Hence, when J=1 and K=0 ; Q_{n+1}= 1 (Set)

Case 4: J=1 and K=1

Q=0, Q'=1 : When J=1 ,K=1 and Q=0; S=1 and R=0 then Q_{n+1}=1.

Q=1, Q'=0 : When J=1 ,K=1 and Q=1; S=0 and R=1 then Q_{n+1}=0

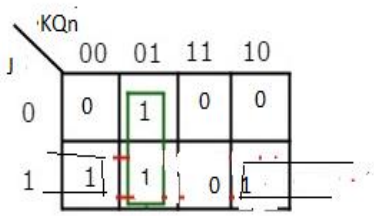
Hence, when J=1 and K=0 ; Q_{n+1}= **Q_n'(Toggle)**

Characteristic table of JK flipflop

CP	J	K	Q_n	Q_{n+1}	State
↑	0	0	0	0	No change(NC)
↑	0	0	1	1	
↑	0	1	0	0	Reset
↑	0	1	1	0	
↑	1	0	0	1	Set
↑	1	0	1	1	
↑	1	1	0	1	Toggle
↑	1	1	1	0	
0	x	x	0	0	No change(NC)
0	x	x	1	1	

J	K	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	Q_n'

Characteristic equation of JK flipflop

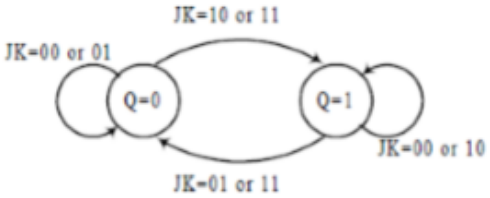


$$Q_{n+1} = JQ_n' + K'Q_n$$

Excitation table of JK flipflop

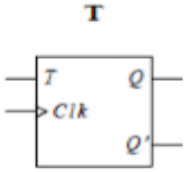
Q_n	Q_{n+1}	J	K
0	0	0	X
0	1	1	X
1	0	x	1
1	1	x	0

State diagram of JK Flipflop

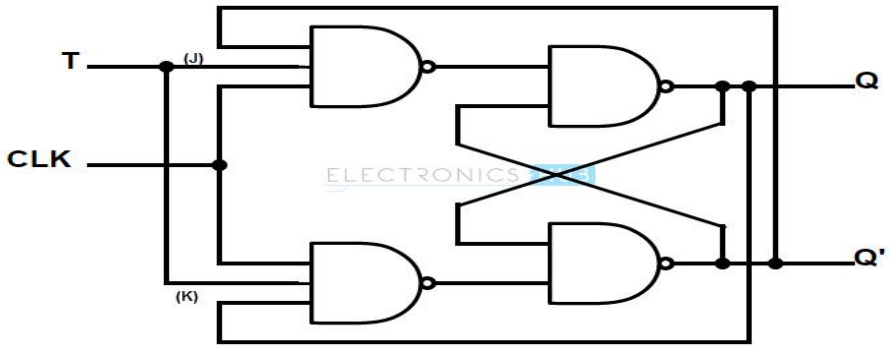


4. **T Flipflop**

T Flipflop is also known as toggle flipflop. It is a modification of JK flipflop. T Flipflop is obtained from a JK flipflop by connecting both inputs, J and K together**.**



Logic symbol of T Flipflop



Logic diagram of T flipflop

When T=0, J=K=0 and hence **Q_{n+1}= Q_n (no output change)**

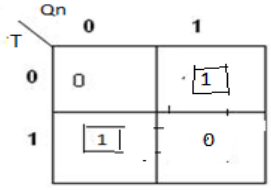
When T=1, J=K=1 and hence **Q_{n+1}= Q_n' (toggles)**

Characteristic table of T flipflop

CP	T	Q_n	Q_{n+1}
↑	0	0	0
↑	0	1	1
↑	1	0	1
↑	1	1	0
0	X	0	0
0	X	1	1

CP	T	Q_{n+1}
↑	0	Q_n
↑	1	Q_n'
0	X	Q_n

Characteristiic Equation

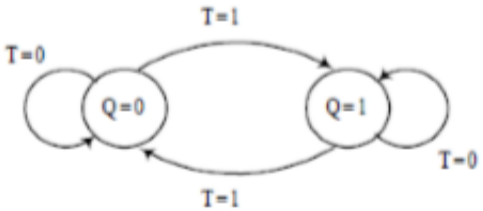


$Q_{n+1} = TQ_n' + T'Q_n$

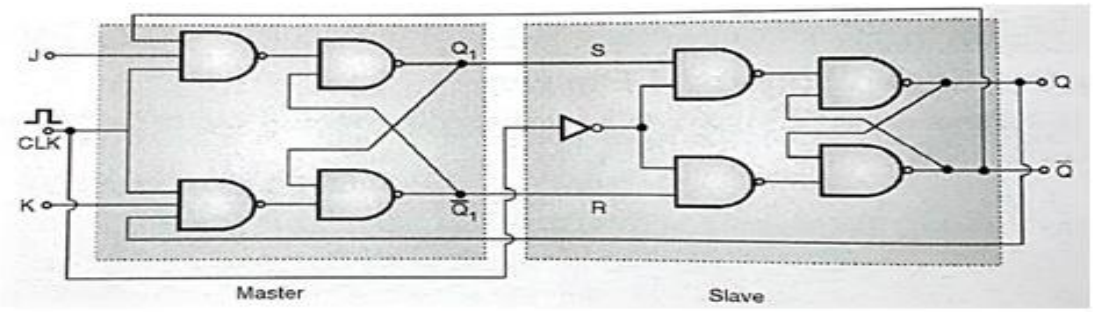
Excitation table of T flipflop

Q_n	Q_{n+1}	T
0	0	0
0	1	1
1	0	1
1	1	0

State diagram of D Flipflop



Master Slave Flipflop Circuit



Master slave JK flipflop

In master slave JK flip flop, it consists of clocked JK flipflop as a master and clocked SR flipflop as a slave. The output of master flipflop is fed as an input to the slave flipflop. Clock signal is directly connected to the master flipflop and it is connected through inverter to the slave flipflop. During the positive clock pulse, the information present at the J & K inputs is transmitted to the output of master flipflop and it is held there until the negative clock pulse occurs. After which it is allowed to pass through the output of slave flipflop.

When J=0, K=0,output of master remains same at the positive clock and the output of slave also remains same at the negative clock.When J=0, K=1,output of master resets at the positive clock and the output of slave also resets at the negative clock.When J=1, K=0,output of master sets at the positive clock and the output of slave also sets at the negative clock.When J=1, K=1,output of master toggles at the positive clock and the slave then copies the output of master at the negative clock. This prevents race around condition. [In JK flip flop as long as clock is high for the input conditions J&K equals to the output changes or complements its output from 1–>0 and 0–>1.This is known as race around condition.

Truthtable

CP	J	K	Q_{n+1}	State
⎍	0	0	Q_n	No change
⎍	0	1	0	Reset
⎍	1	0	1	Set
⎍	1	1	**Q_n'**	**Toggle**

Applications of flipflop

Data storage, data transfer, registers, counters, memory, Frequency Division

4.2 Shift Registers

Register is a group of flipflops that can be used to store a word. So n bit register has a group of n flipflops and is capable of storing any binary number containing n bits. Shift Registers are **sequential logic circuits**, capable of storage and transfer of data. They are made up of Flip

Flops which are connected in such a way that the output of one flip flop could serve as the input of the other flip-flop, depending on the type of shift registers being created.

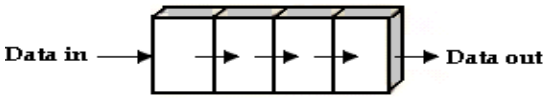
Shift registers are categorized into types majorly by their mode of operation, either serial or parallel.

1. Serial in – Serial out Shift Register (SISO)

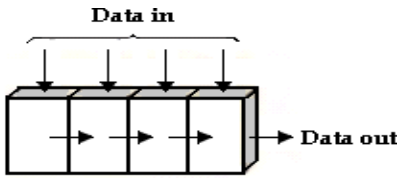
2. Serial In – Parallel out shift Register (SIPO)

3. Parallel in – Parallel out Shift Register (PIPO)

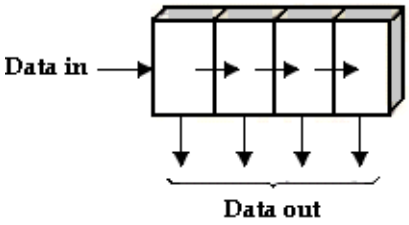
4. Parallel in – Serial out Shift Register (PISO)



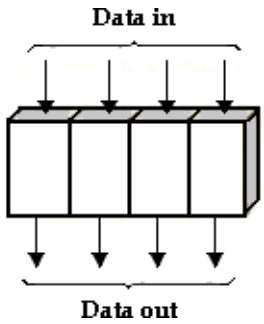
Serial in- serial out



Parallel in- serial out



Serial in- parallel out

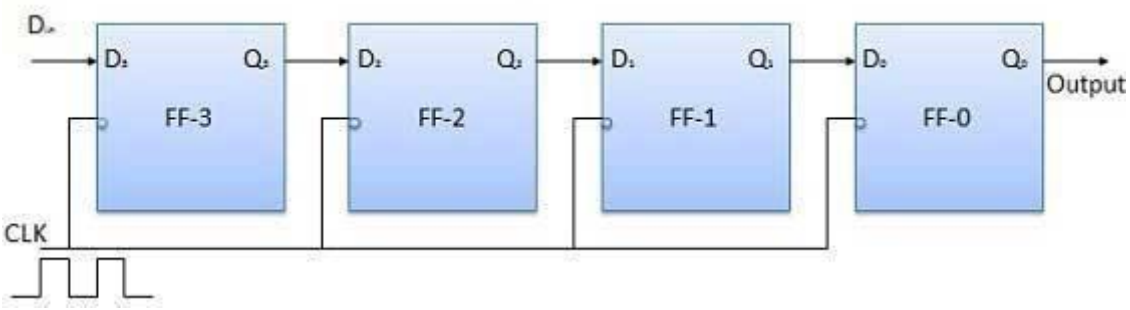


Parallel in- parallel out

1**. Serial in – Serial out Shift Register (SISO)**

The shift register, which allows serial input and produces serial output is known as Serial In – Serial Out SISO shift register.

Block Diagram



Operation

The four bit number 1111 is entered into the register.

Initially register is cleared. So $Q_3\ Q_2\ Q_1\ Q_0$= 0000

When data 1111 is applied serially , LSB bit applied as Din. So Din = D3 = 1.

Apply the clock.

(i)On the first falling edge of clock, FF-3 is set, and stored word in the register is

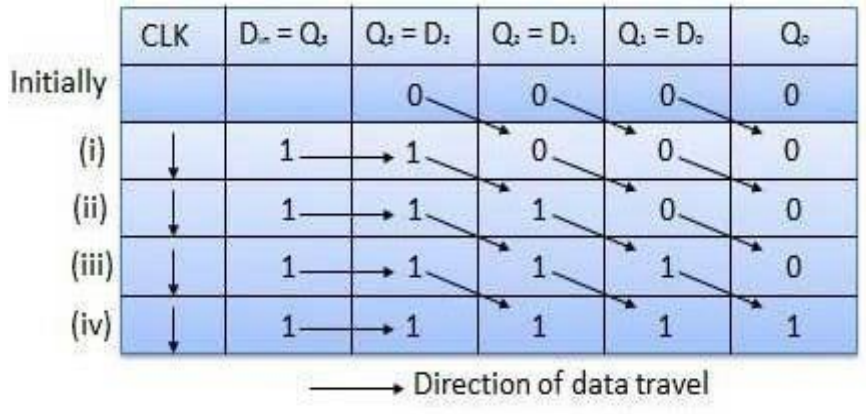
$Q_3\ Q_2\ Q_1\ Q_0$=1000.

(ii)Apply the next bit to Din. So Din = 1. As soon as the next negative edge of the clock hits, FF-2 will set and the stored word change to $Q_3\ Q_2\ Q_1\ Q_0$= 1100.

(iii)Apply the next bit to be stored i.e. 1 to Din. Apply the clock pulse. As soon as the third negative clock edge hits, FF-1 will be set and output will be modified to $Q_3\ Q_2\ Q_1\ Q_0$ = 1110.

(iv) Similarly with Din = 1 and with the fourth negative clock edge arriving, the stored word in the register is $Q_3\ Q_2\ Q_1\ Q_0$= 1111.

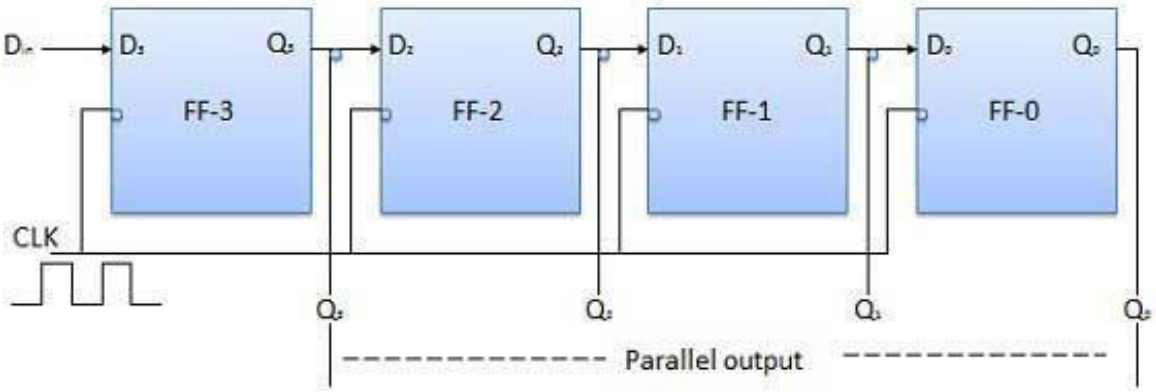
Truth Table



2. Serial In – Parallel out shift Register (SIPO)

The shift register, which allows serial input and produces parallel output is known as Serial In – Parallel Out SIPOSIPO shift register.Data is loaded bit by bit. As soon as the data loading gets completed, all the flip-flops contain their required data, the outputs are enabled so that all the loaded data is made available over all the output .lines at the sametime.4 clock cycles are required to load a four bit word. Hence the speed of operation of SIPO mode is same as that of SISO mode.

Block Diagram



FOUR BIT BINARY NUMBER 1111. Data is given serially. Initially $Q_3\ Q_2\ Q_1\ Q_0$= 0000

After CP 1 $Q_3\ Q_2\ Q_1\ Q_0$= 1000

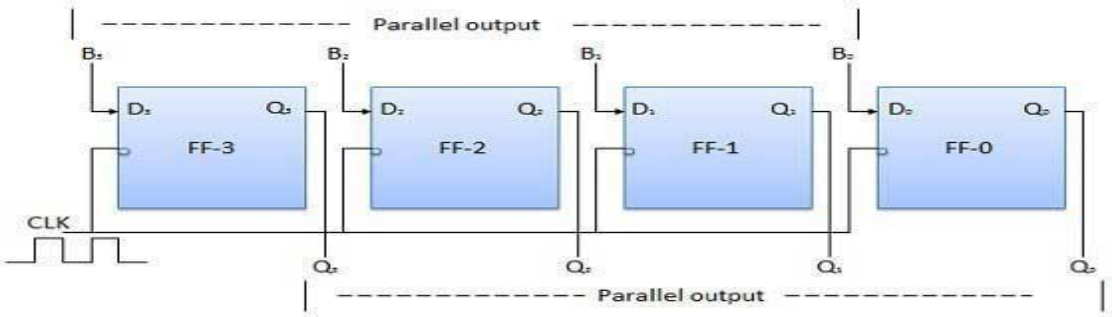
After CP 2 $Q_3\ Q_2\ Q_1\ Q_0$= 1100

After CP 3 $Q_3\ Q_2\ Q_1\ Q_0$= 1110

After CP 4 $Q_3\ Q_2\ Q_1\ Q_0$= 1110

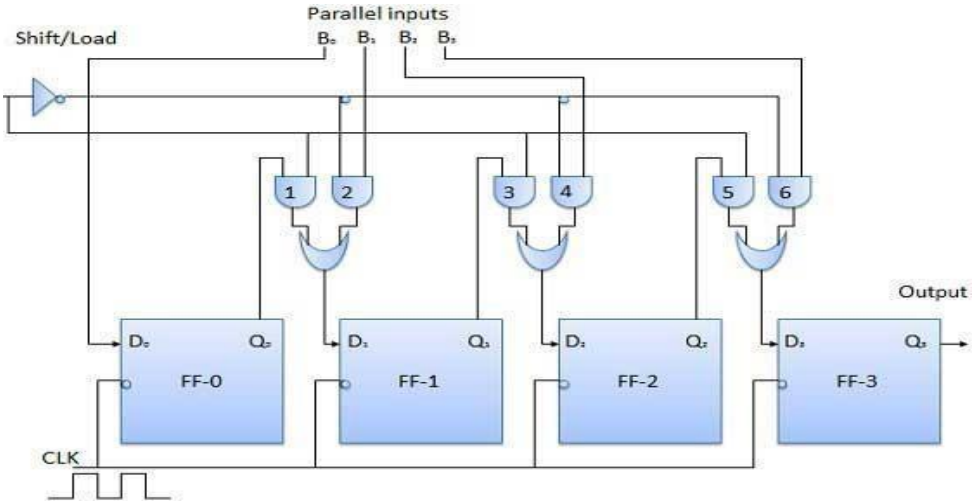
3. Parallel in – Parallel out Shift Register (PIPO)

Here the data is entered in parallel manner and data is also taken out in parallel manner. There is simultaneous entry of all data bits and the bits appear on parallel outputs simultaneously. The block diagram is as follows



4**. Parallel in – Serial out Shift Register (PISO)**

The shift register, which allows parallel input and produces serial output is known as Parallel In − Serial Out PISO shift register. In this type, the bits are entered parallel such a way simultaneously into their respective stages on parallel lines. There are four input lines B0, B1, B2,B3 for entering data parallel into the register. Shift/Load is the control input which allows shift or loading data operation of the register.



Load mode : When the shift/load bar line is low (0), the AND gate 2, 4 and 6 become active they will pass B1, B2, B3 bits to the corresponding flip-flops. On the low going edge of clock, the binary input B0, B1, B2, B3 will get loaded into the corresponding flip-flops. Thus parallel loading takes place.

Shift mode : When the shift /load bar line is low (1), the AND gate 2, 4 and 6 become inactive. Hence the parallel loading of the data becomes impossible. But the AND gate 1,3 and 5 become active. Therefore the shifting of data from left to right bit by bit on application of clock pulses. Thus the parallel in serial out operation takes place.

5. BI-DIRECTION SHIFT REGISTERS:

A bidirectional shift register is one in which the data can be shifted either left or right. It can be implemented by using gating logic that enables the transfer of a data bit from one stage to the next stage to the right or to the left depending on the level of a control line.

A 4-bit bidirectional shift register is shown below. A HIGH on the RIGHT/LEFT control input allows data bits inside the register to be shifted to the right, and a LOW enables data bits inside the register to be shifted to the left.

When the RIGHT/LEFT control input is **HIGH**, gates G_1, G_2, G_3 and G_4 are enabled, and the state of the Q output of each Flip-Flop is passed through to the D input of the following Flip-Flop. When a clock pulse occurs, the data bits are shifted one place to the right.

When the RIGHT/LEFT control input is **LOW**, gates G_5, G_6, G_7 and G_8 are enabled, and the Q output of each Flip-Flop is passed through to the D input of the preceding Flip-Flop. When a clock pulse occurs, the data bits are then shifted one place to the left.

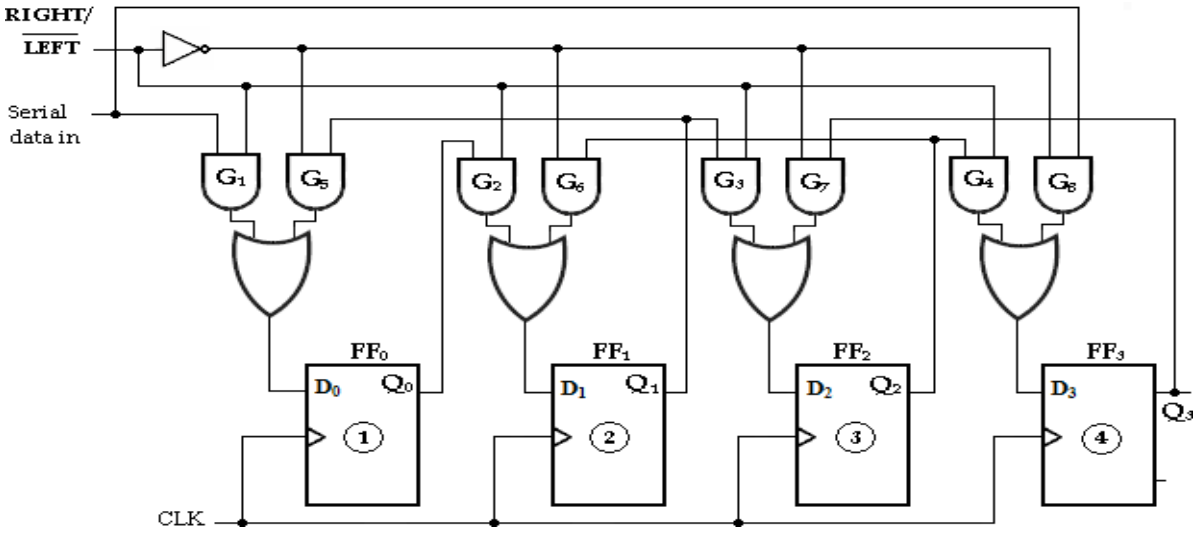


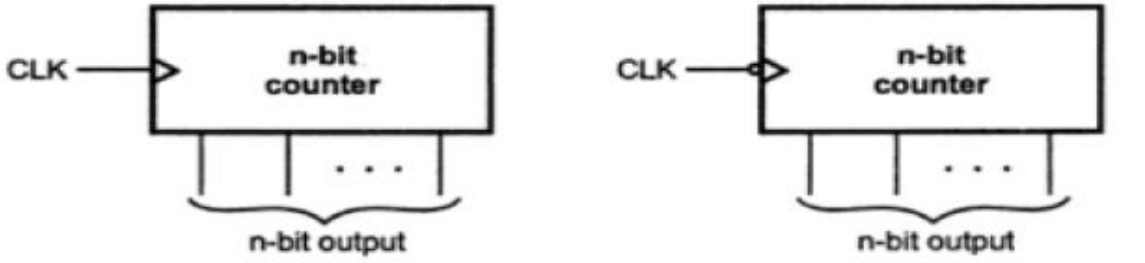
Fig. 4-bit bi-directional shift register

4.3 Counters

A counter is a register capable of counting the number of clock pulses arriving at its clock input. Count represents the number of clock pulses arrived. On arrival of each clock pulse,

- In case of Up counter, the counter is incremented by one
- In case of down counter, it is decremented by one

The Fig. shows the logic symbol of a binary counter. External clock is applied to the clock input of the counter. The counter can be positive edge triggered or negative edge triggered. The n-bit binary counter has n flip-flops, and it has 2^n distinct states of outputs. For example, 2-bit counter has 2 flip-flops and it has $4(2^2)$ distinct states : 00, 01, 10 and 11. Similarly, the 3-bit binary counter has 3 flip-flops and it has $8(2^3)$ distinct states : 000, 001, 010, 011, 100, 101 110 and 111.



(a) Positive edge triggered n-bit counter

(b) Negative edge triggered n-bit counter

Fig. Logic symbol of counter

Types of Counters

1) Asynchronous or ripplecounters 2) Synchronous counters

Asynchronous or ripplecounters

A binary ripple / asynchronous counter consists of a series connection of complementing flip –flops, with the output of each flip – flop connected to the clock input of

the next higher order flip- flop. The flip-flop holding the least significant bit receives the incoming clock pulses.

Synchronous counters

When counter is clocked such that each flip –flop in the counter is triggered at the same time, the counter is called as synchronous counter.

S.No	Asynchronous (ripple) counter	Synchronous counter
1	All the Flip-Flops are not clocked simultaneously.	All the Flip-Flops are clocked simultaneously.
2	The delay times of all Flip- Flops are added. Therefore there is considerable propagation delay.	There is minimum propagation delay.
3	Speed of operation is low	Speed of operation is high.
4	Logic circuit is very simple even for more number of states.	Design involves complex logic circuit as number of state increases.
5	Minimum numbers of logic devices are needed.	The number of logic devices is more than ripple counters.
6	Cheaper than synchronous counters.	Costlier than ripple counters.

SYNCHRONOUS COUNTERS

Flip-Flops can be connected together to perform counting operations. Such a group of Flip- Flops is a counter. The number of Flip-Flops used and the way in which they are connected determine the number of states (called the modulus) and also the specific sequence of states that the counter goes through during each complete cycle.

Counters are classified into two broad categories according to the way they are clocked: Asynchronous counters, Synchronous counters.

In asynchronous (ripple) counters, the first Flip-Flop is clocked by the external clock pulse and then each successive Flip-Flop is clocked by the output of the preceding Flip-Flop.

In synchronous counters, the clock input is connected to all of the Flip-Flops so that they are clocked simultaneously. Within each of these two categories, counters are classified primarily by the type of sequence, the number of states, or the number of Flip-Flops in the counter.

The term ‘synchronous’ refers to events that have a fixed time relationship with each other. In synchronous counter, the clock pulses are applied to all Flip- Flops simultaneously. Hence there is minimum propagation delay.

2-Bit Synchronous Binary Counter

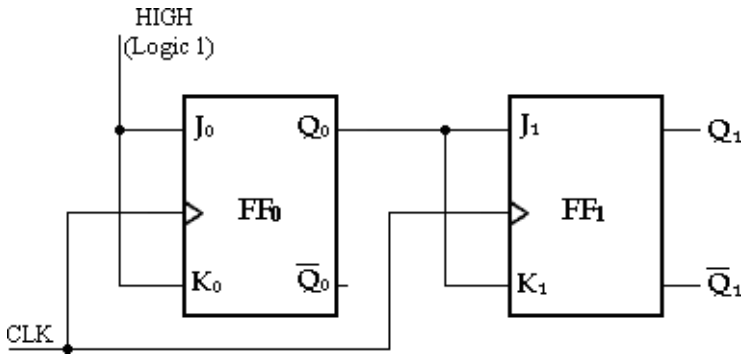


Fig. Logic diagram of 2-Bit Synchronous Binary Counter

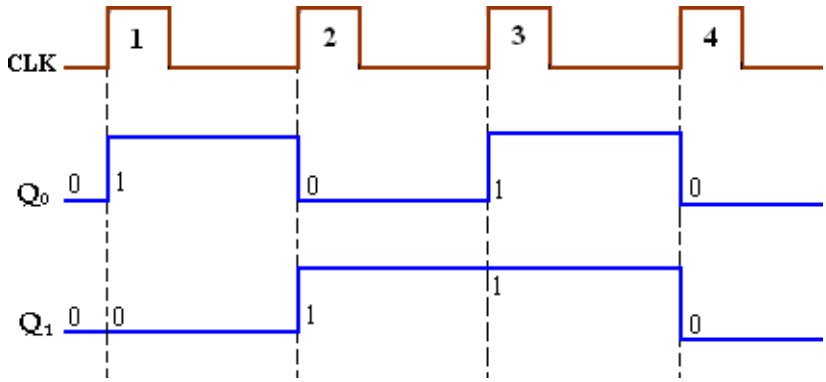
In this counter the clock signal is connected in parallel to clock inputs of both the Flip-Flops (FF_0 and FF_1). The output of FF_0 is connected to J_1 and K_1 inputs of the second Flip-Flop (FF_1).

Assume that the counter is initially in the binary 0 state: i.e., both Flip-Flops are RESET. When the positive edge of the first clock pulse is applied, FF_0 will toggle because J_0= k_0= 1, whereas FF_1 output will remain 0 because J_1= k_1= 0. After the first clock pulse Q_0=1 and Q_1=0.

When the leading edge of CLK2 occurs, FF_0 will toggle and Q_0 will go LOW. Since FF_1 has a HIGH (Q_0 = 1) on its J_1 and K_1 inputs at the triggering edge of this clock pulse, the Flip-Flop toggles and Q_1 goes HIGH. Thus, after CLK2,
Q_0 = 0 and Q_1 = 1.

When the leading edge of CLK3 occurs, FF_0 again toggles to the SET state (Q_0 = 1), and FF_1 remains SET (Q_1 = 1) because its J_1 and K_1 inputs are both LOW (Q_0 = 0). After this triggering edge, Q_0 = 1 and Q_1= 1.

Finally, at the leading edge of CLK4, Q_0 and Q_1 go LOW because they both have a toggle condition on their J_1 and K_1 inputs. The counter has now recycled to its original state, Q_0 = Q_1 = 0.



Timing diagram

3-Bit Synchronous Binary Counter

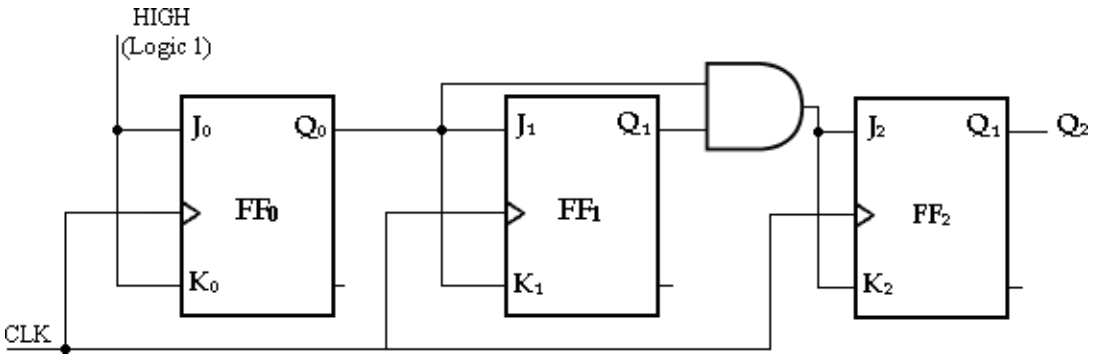


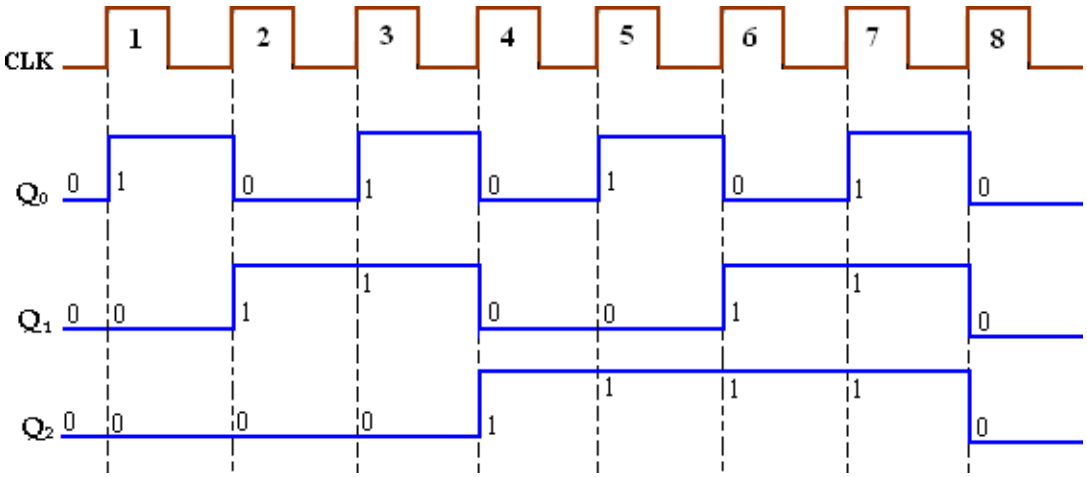
Fig. Logic diagram of 3-Bit Synchronous Binary Counter

A 3 bit synchronous binary counter is constructed with three JK Flip-Flops and an AND gate. The output of FF_0 (Q_0) changes on each clock pulse as the counter progresses from its original state to its final state and then back to its original state. To produce this operation, FF_0 must be held in the toggle mode by constant HIGH, on its J_0 and K_0inputs.

The output of FF_1 (Q_1) goes to the opposite state following each time Q_0= 1. This change occurs at CLK2, CLK4, CLK6, and CLK8. The CLK8 pulse causes the counter to recycle. To produce this operation, Q_0 is connected to the J_1 and K_1 inputs of FF_1. When Q_0= 1 and a clock pulse occurs, FF_1 is in the toggle mode and therefore changes state. When Q_0= 0, FF_1 is in the no-change mode and remains in its present state.

The output of FF_2 (Q_2) changes state both times; it is preceded by the unique condition in which both Q_0 and Q_1 are HIGH. This condition is detected by the AND gate and applied to the J_2 and K_2 inputs of FF_3. Whenever both outputs Q_0= Q_1= 1, the output of the AND gate makes the J_2= K_2= 1 and FF_2 toggles on the following clock pulse. Otherwise, the J_2 and K_2 inputs of FF2 are held LOW by the AND gate output, FF_2 does not change state.

CLOCK Pulse	**Q_2**	**Q_1**	**Q_0**
Initially	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1
8 (recycles)	0	0	0



Timing diagram

Synchronous UP/DOWN Counter

An up/down counter is a bidirectional counter, capable of progressing in either direction through a certain sequence. A 3-bit binary counter that advances upward through its sequence (0, 1, 2, 3, 4, 5, 6, 7) and then can be reversed so that it goes through the sequence in the opposite direction (7, 6, 5, 4, 3, 2, 1,0) is an illustration of up/down sequential operation.

The complete up/down sequence for a 3-bit binary counter is shown in table below. The arrows indicate the state-to-state movement of the counter for both its UP and its DOWN modes of operation. An examination of Q_0 for both the up and down sequences shows that FF_0 toggles on each clock pulse. Thus, the J_0 and K_0 inputs of FF_0 are, **$J_0 = K_0 = 1$**

CLOCK PULSE	UP	Q_2	Q_1	Q_0	DOWN
0		0	0	0	
1		0	0	1	
2		0	1	0	
3		0	1	1	
4		1	0	0	
5		1	0	1	
6		1	1	0	
7		1	1	1	

To form a synchronous UP/DOWN counter, the control input (UP/DOWN) is used to allow either the normal output or the inverted output of one Flip-Flop to the J and K inputs of the next Flip-Flop. When UP/DOWN= 1, the MOD 8 counter will count from 000 to 111 and UP/DOWN= 0, it will count from 111 to 000.

When UP/DOWN= 1, it will enable AND gates 1 and 3 and disable AND gates 2 and 4. This allows the Q0 and Q1 outputs through the AND gates to the J and K inputs of the following Flip-Flops, so the counter counts up as pulses are applied.
When UP/DOWN= 0, the reverse action takes place.

$J_1 = K_1 = (Q_0.UP) + (Q_0'.DOWN)$

$J_2 = K_2 = (Q_0.\ Q_1.UP) + (Q_0'.Q1'.DOWN)$

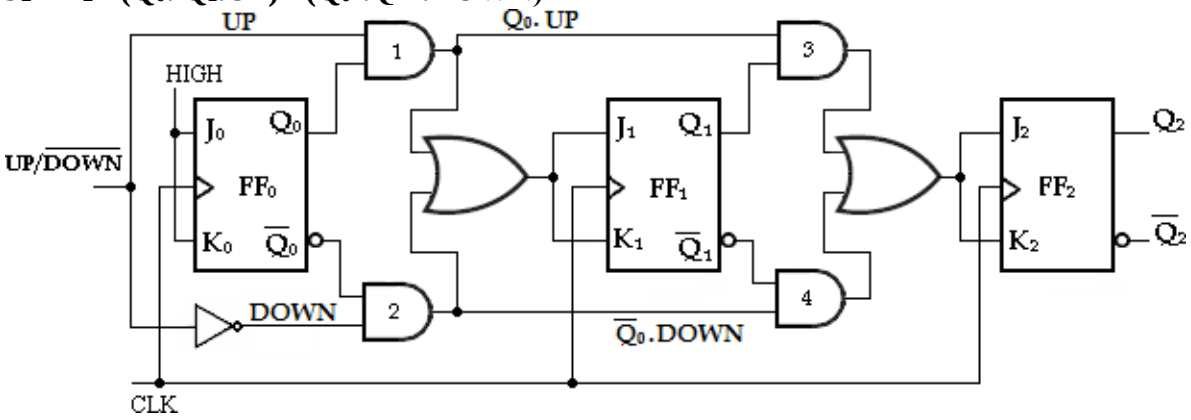


Fig. Circuit diagram of 3-bit UP/DOWN Synchronous Counter

Design of Synchronous MOD Counter

The counter with 'n' Flip-Flops has maximum MOD number 2^n. Find the number of Flip-Flops (n) required for the desired MOD number (N) using the equation,

$$2^n \geq N$$

For example, a 3 bit binary counter is a MOD 8 counter. The basic counter can be modified to produce MOD numbers less than 2^n by allowing the counter to skin those are normally part of counting sequence.

n= 3 , N= 8 , $2^n = 23 = 8 = N$

MOD 5 Counter:

$2^n = N$, $2^n = 5$, $2^2 = 4$ less than N., $2^3 = 8 > N(5)$

Therefore, 3 Flip-Flops are required.

MOD 10 Counter:

$2^n = N = 10$, $2^3 = 8$ less than N, $2^4 = 16 > N(10)$.

To construct any MOD-N counter, the following methods can be used.

- Find the number of Flip-Flops (n) required for the desired MOD number (N) using the equation, $\mathbf{2^n \geq N.}$
- Connect all the Flip-Flops as a required counter.
- Find the binary number for N.
  - Connect all Flip-Flop outputs for which Q= 1 when the count is N, as inputs to NAND gate.

- Connect the NAND gate output to the CLR input of each Flip-Flop.

When the counter reaches Nth state, the output of the NAND gate goes LOW, resetting all Flip-Flops to 0. Therefore the counter counts from 0 through N-1.

For example, MOD-10 counter reaches state 10 (1010). i.e., $Q_3Q_2Q_1Q_0$= 1 0 1 0. The outputs Q_3 and Q_1 are connected to the NAND gate and the output of the NAND gate goes LOW and resetting all Flip-Flops to zero. Therefore MOD-10 counter counts from 0000 to 1001. And then recycles to the zero value.

The MOD-10 counter circuit is shown below.

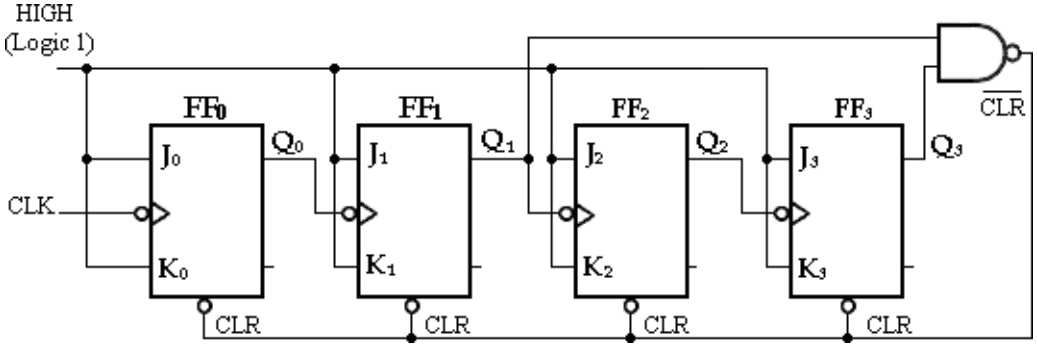


Fig. Logic diagram of MOD-10 (Decade) Counter

Design a MOD- 5 synchronous counter using JK flip flops and implement it, Also draw the timing diagram

Step 1 : Determine the number of flipflop needed. Her N=5, 2^n >= N, n=3 number of flipflops.
Step 2: type of flipflop is JK
Step 3: Determine the execution table
Step 4: Find the input of the Flipflop using K-Map
Step 5: Draw the Circuit diagram

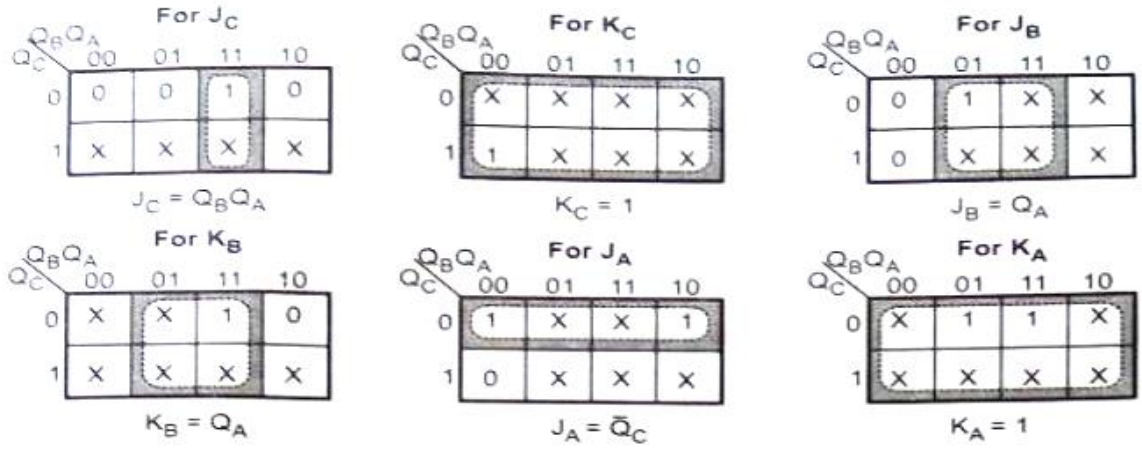
Excitation table of JK flip flop

Q_n	Q_{n+1}	J	K
0	0	0	X
0	1	1	X
1	0	X	1
1	1	X	0

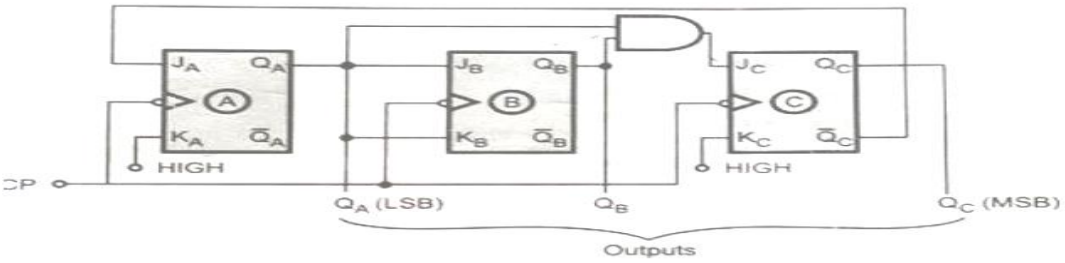
Excitation table for the counter

Present State			Next State			Flip- flop inputs					
Q_C	Q_B	Q_A	Q_{C+1}	Q_{B+1}	Q_{A+1}	J_C	K_C	J_B	K_B	J_A	K_A
0	0	0	0	0	1	0	X	0	X	1	X
0	0	1	0	1	0	0	X	1	X	X	1
0	1	0	0	1	1	0	X	X	0	1	X
0	1	1	1	0	0	1	X	X	1	X	1
1	0	0	0	0	0	X	1	0	X	0	X
1	0	1	X	X	X	X	X	X	X	X	X
1	1	0	X	X	X	X	X	X	X	X	X
1	1	1	X	X	X	X	X	X	X	X	X

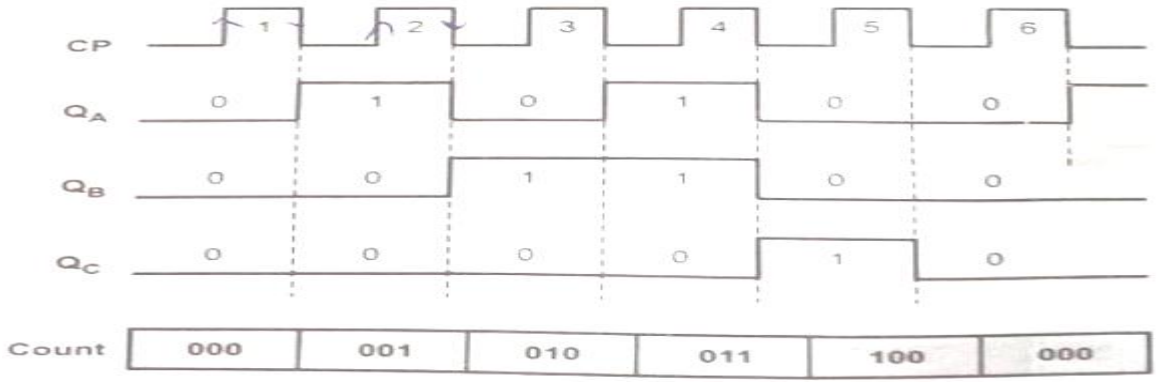
Step 4: Simplification using K - Map



Step 5: Draw the Logic diagram



Timing Diagram



Asynchronous or ripple counters

Asynchronous 2 bit Up counter:

A binary ripple/asynchronous counter consists of a series connection of complementing flip-flops, with the output of each flip-flop connected to the clock input of the next higher-order flip-flop. The flip-flop holding the least significant bit receives the incoming clock pulses. A complementing flip-flops can be obtained from a JK flip-flop with the J and K inputs tied together as shown in the Fig. or from a T flip-flop. A third alternative is to use a D flip-flop with the complement output connected to the D input.

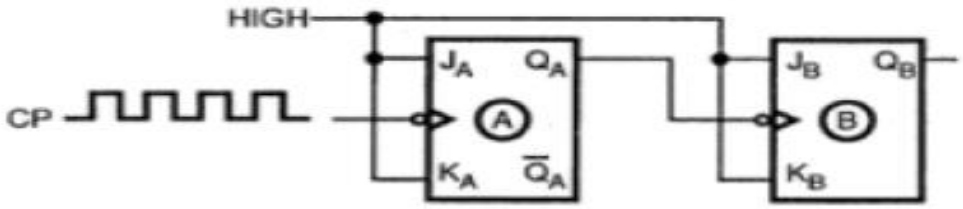


Fig. (a) A two-bit asynchronous binary counter

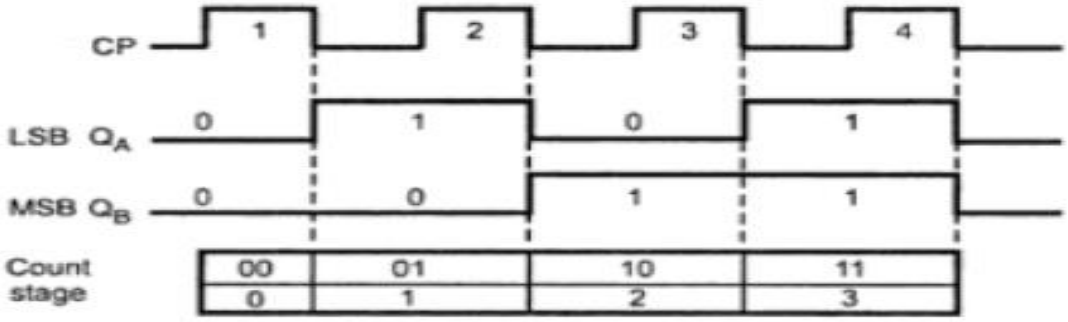


Fig. (b) Timing diagram for the counter

Fig. (a) shows 2-bit asynchronous counter using JK flip-flops. As shown in Fig. the clock signal is connected to the clock input of only first stage flip-flop. The clock input of the second stage flip-flop is triggered by the Q_A output of the first stage . Because of the inherent propagation delay time through a flip-flop, a transition of the input clock pulse and a transition of the Q_A output of first stage can never occur at exactly the same time. Therefore, the two flip-flops are never simultaneously triggered, which results in asynchronous counter operation.

Fig. (b) shows the timing diagram for two-bit asynchronous counter. It illustrates the changes in the state of the flip-flop outputs in response to the clock. J and K input of JK flip-flops are tied to logic HIGH hence output will toggle for each negative edge of the clock input.

Operation

Initially let both the FFs be in the reset state, $Q_AQ_B = 00$

As soon as the first negative clock edge is applied, FF-A will toggle and Q_A will change from 0 to 1. But at the instant of application of negative clock edge, Q_A , $J_B = K_B = 0$. Hence FF-B will not change its state. So Q_B will remain0. $Q_BQ_A = 01$ after the first clock pulse.

On the arrival of second negative clock edge, FF-A toggles again and Q_A changes from 1 to 0. But at this instant Q_A was 1. So $J_B = K_B = 1$ and FF-B will toggle. Hence Q_B changes from 0 to1. $Q_BQ_A = 10$ after the second clock pulse.

On application of the third falling clock edge, FF-A will toggle from 0 to 1 but there is no change of state for FF-B. $Q_BQ_A = 11$ after the third clock pulse. The output of counter is incremented by one for each clock transition. So,such counters are called as up counters.

Asynchronous 3 bit Up counter :

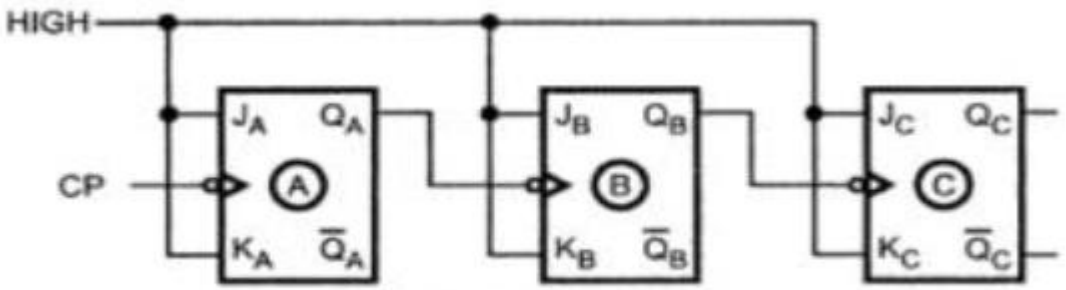


Fig. (a) Logic diagram

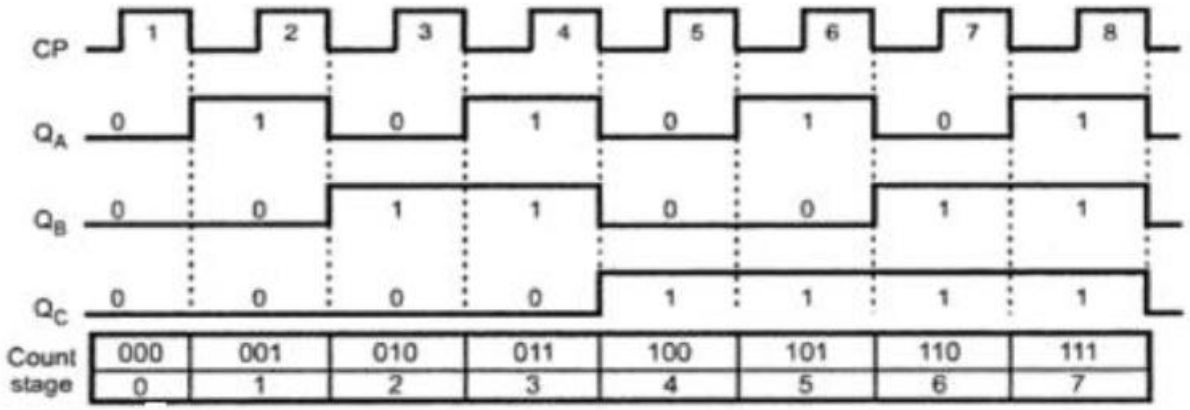
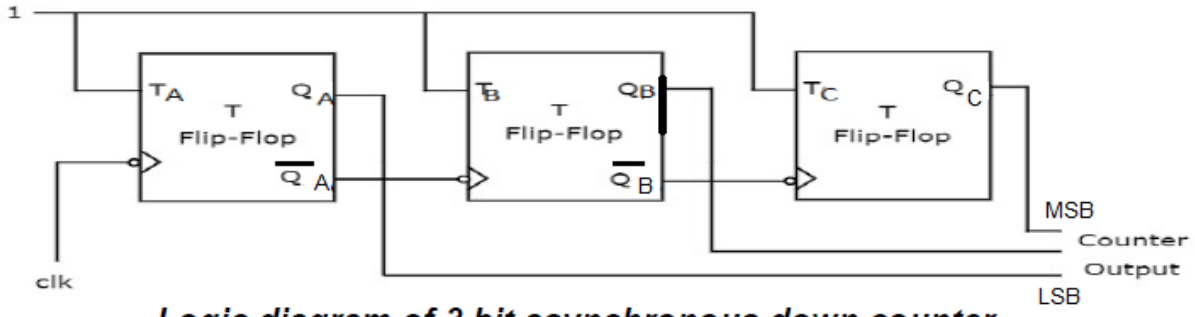


Fig. (b) Output waveforms for 3-bit asynchronous counter

Asynchronous 3 bit Down counter

An 'N' bit asynchronous binary down counter consists of 'N' T flip-flops. It counts from $2^N - 1$ to 0. The block diagram of 3-bit Asynchronous binary down counter is shown in figure.



Logic diagram of 3 bit asynchronous down counter

Down counters are not as widely used as up counters. They are used in situations where it must be known when a desired number of input pulses has occurred. In these situations the down counter is preset to the desired number and then allowed to count down as the pulses are applied. When the counter reaches the zero state it is detected by a logic gate whose output then indicates that the preset number of pulses have occurred.

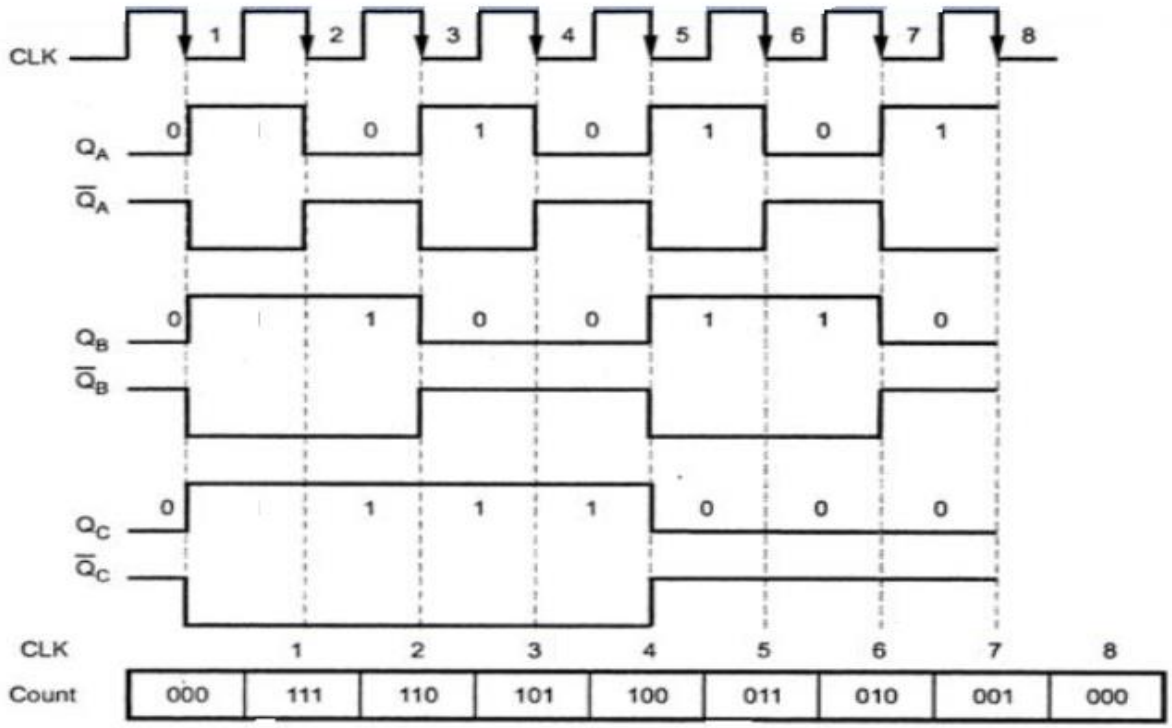
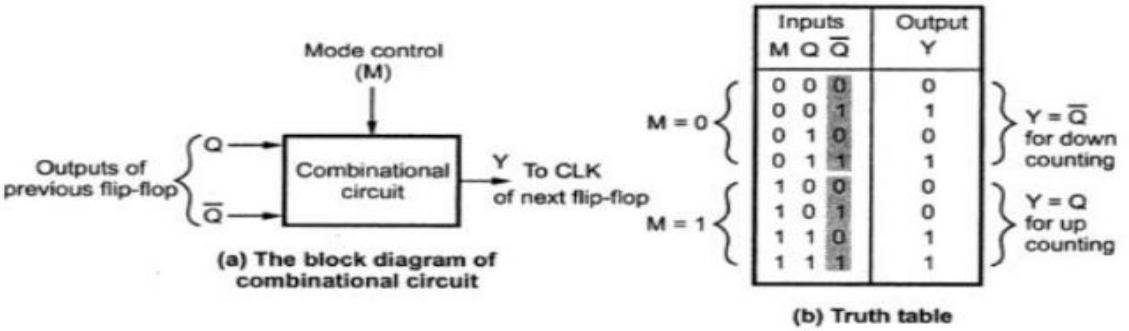


Fig. Timing diagram for 3-bit DOWN ripple counter

Asynchronous Up / Down counter

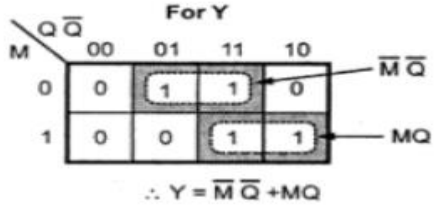
To form an asynchronous up/$\overline{\text{down}}$ counter one control input say M is necessary to control the operation of the up/$\overline{\text{down}}$ counter. When M = 1, the counter will count up and when M = 0, the counter will count down. To achieve this the M input should be used to control whether the normal flip-flop output (Q) or the inverted flip-flop output ($\overline{Q}$) is fed to drive the clock signal of the successive stage flip-flop, as shown in Fig. (a). The truth table for such combinational circuit is shown in Fig. (b).



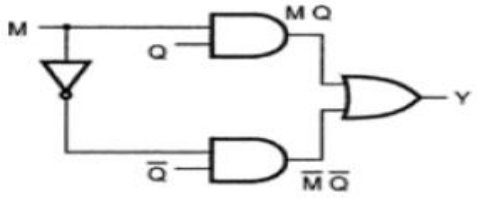
(a) The block diagram of combinational circuit

	Inputs M	Q	$\overline{Q}$	Output Y	
M = 0	0	0	0	0	$Y = \overline{Q}$ for down counting
	0	0	1	1	
	0	1	0	0	
	0	1	1	1	
M = 1	1	0	0	0	Y = Q for up counting
	1	0	1	0	
	1	1	0	1	
	1	1	1	1	

(b) Truth table



a) K-map simplification



b) Logic diagram

The Fig. shows the 3-bit up/down counter that will count from 000 up to 111 when the mode control input M is 1 and from 111 down to 000 when mode control input M is 0.

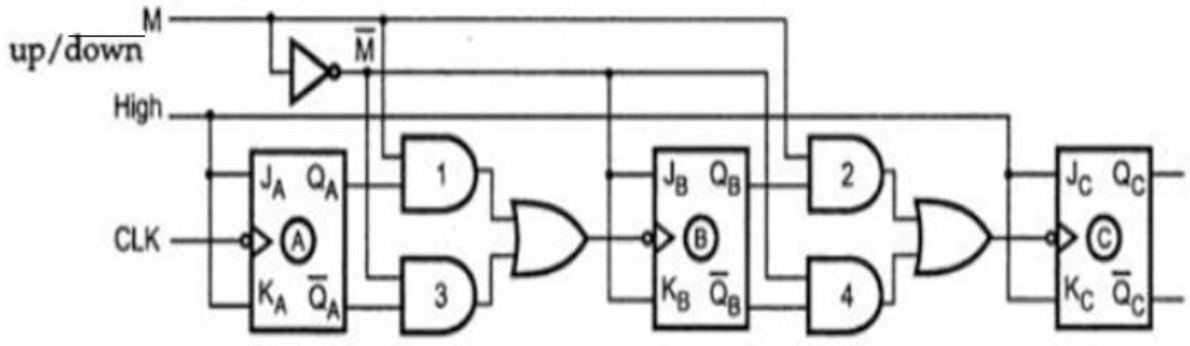


Fig. 3-bit asynchronous up/down counter

A logic 1 on M enables AND gates 1 and 2 and disables AND gates 3 and 4. This allows the Q_A and Q_B outputs to drive the clock inputs of their respective next stages. So that counter will count up. When M is logic 0, AND gates 1 and 2 are disabled and AND gete 3 and 4 are enabled. This allows the $\overline{Q}_A$ and $\overline{Q}_B$ outputs to drive the clock inputs of their respective next stages so that counter will count down.

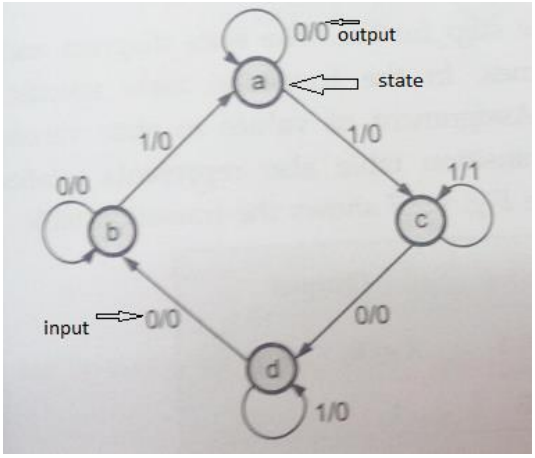
The synchronous or clocked sequential circuits are represented by two models.

Moore model : The output depends only on the present state of flipflops.

Mealy model : The output depends on both the present state of flipflops and on the inputs.

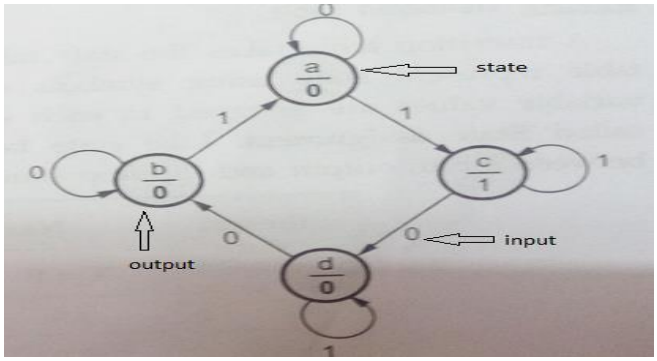
State diagram:

It is a pictorial representation of a behavior of a sequential circuit.The state is represented by the circle and the transition between states is indicated by directed lines connecting the circles.



state diagram for mealy circuit

The figure shows the state diagram for mealy circuit. The binary number inside each circle represents the state. The directed lines are labeled with two binary numbers separated by a symbol '/'. The input that causes the transition is labelled first and the output value during the present state is labeled after the symbol'/'.



state diagram for Moore circuit

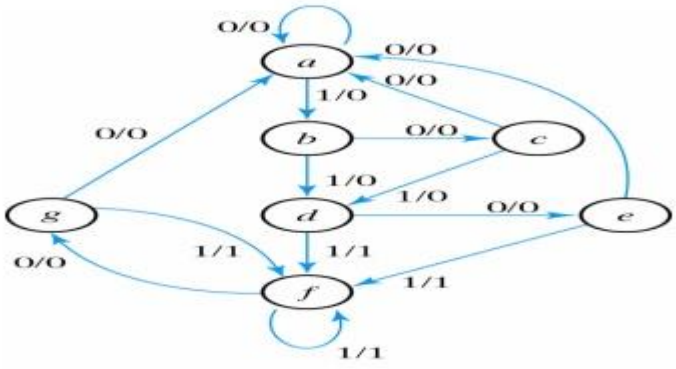
In case of moore circuit, the directed lines are labeled with only one binary number representing the input. The output state is indicated within the circle, below the present state because output state depends only on present state and not on the input.

State table: The information contained in the state diagram is translated into a tabular form.

State assignment: Assignment of values to state variables

State reduction: This technique avoids the introduction of redundant states. The reduction in redundant states reduce the number of required flipflops and logic gates, reducing the cost of the final circuit.

suppose a sequential circuit is specified by the following seven-state diagram:



There are an infinite number of input sequences that may be applied; each results in a unique output sequence. Consider the input sequence 01010110100 starting from the initial state a:

state	*a*	*a*	*b*	*c*	*d*	*e*	*f*	*f*	*g*	*f*	*g*	*a*
input	0	1	0	1	0	1	1	0	1	0	0	
output	0	0	0	0	0	1	1	0	1	0	0	

An algorithm for the state reduction quotes that:

"Two states are said to be equivalent if, for each member of the set of inputs, they give exactly the same output and send the circuit either to the same state or to an equivalent state."

Now apply this algorithm to the state table of the circuit:

Present State	Next State		Output	
	$x = 0$	$x = 1$	$x = 0$	$x = 1$
a	a	b	0	0
b	c	d	0	0
c	a	d	0	0
d	e	f	0	1
e	a	f	0	1
f	g	f	0	1
g	a	f	0	1

States g and e both go to states a and f and have outputs of 0 and 1 for x = 0 and x = 1, respectively.

The procedure for removing a state and replacing it by its equivalent is demonstrated in the following table:

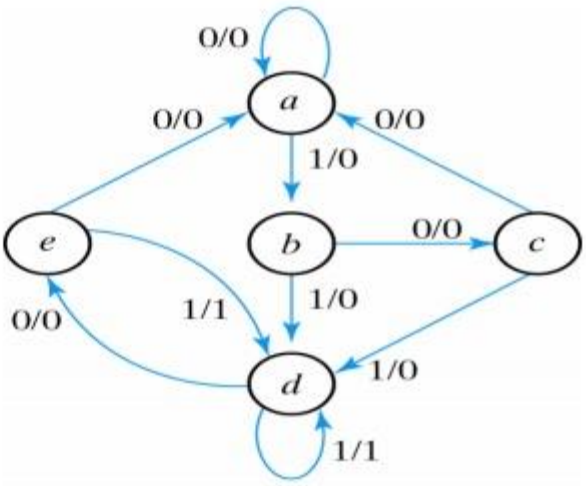
Present State	Next State		Output	
	$x = 0$	$x = 1$	$x = 0$	$x = 1$
a	a	b	0	0
b	c	d	0	0
c	a	d	0	0
d	e	f	0	1
e	a	f	0	1
f	e	f	0	1

Thus, the row with present state g is removed and stage g is replaced by state e each time it occurs in the next state columns. Present state f now has next states e and f and outputs 0 and 1 for x = 0 and x = 1. The same next states and outputs appear in the row with present state d. Therefore, states f and d are equivalent and can be removed and replaced with d.

The final reduced state table is:

Present State	Next State		Output	
	$x = 0$	$x = 1$	$x = 0$	$x = 1$
a	a	b	0	0
b	c	d	0	0
c	a	d	0	0
d	e	d	0	1
e	a	d	0	1

The state diagram for the above reduced table is:



This state diagram satisfies the original input output specifications.

Applying the input sequence previously used, the following list is obtained:

state	*a*	*a*	*b*	*c*	*d*	*e*	*d*	*d*	*e*	*d*	*e*	*a*
input	0	1	0	1	0	1	1	0	1	0	0	
output	0	0	0	0	0	1	1	0	1	0	0	

Note that the same output sequence results, although the state sequence is different.

References

1.Morris Mano M. and Michael D. Ciletti, “Digital Design”, IV Edition, Pearson Education, 2008.

2 A.P Godse & D.A Godse “Digital Electronics”, Technical publications, Pune, Revised third edition, 2008

Question bank

Part A

1. Define sequential logic circuit.
2. List the purpose of flip-flop and mention its types.
3. Draw the circuit for SR Flip-flop.
4. Explain briefly about JK Flip-flop
5. Show how a RS FF can be built using NAND gates.
6. Determine the characteristic equation of JK FF.
7. List the purpose of shift registers and mention its types.
8. Determine the characteristic equation of JK FF.
9. List the purpose of shift registers and mention its types.
10. Differentiate between asynchronous and synchronous counters.

11. Define state diagram.

Part B

1. Explain the operation of RS Flipflop and derive its characteristic equation and excitation table.
2. Discuss the working of JK Flipflop and derive its characteristic equation and excitation table.
3. Discuss the working of JK Flipflop and derive its characteristic equation and excitation table.
4. Describe the working of master Jk flipflop and the input and output action of JK master /slave flipflops.
5. With suitable example explain state reduction and state assignment.
6. Implement a mod 5 synchronous counter using JK flipflops.
7. Design a 3 bit UP/DOWN counter using T flipflops.
8. Design a 3 bit binary counter using T flipflops
9. Implement a synchronous counter which counts in the sequence 000,001,010,011,100,101,110,111,000 using D flip-flop.
10. Illustrate the operation of shift registers and its types.

SCHOOL OF BIO AND CHEMICAL ENGINEERING

DEPARTMENT OF BIOMEDICAL ENGINEERING

V. Logic Families and Memories

Classification and Characteristics of Logic Families - Operation of RTL, DTL, HTL, ECL, MOS and CMOS- Comparison of Logic Families Memories – Random Access Memory – Static RAM, Dynamic RAM, Read Only Memory, Programmable memory - EPROM,EEPROM.

5.1 Digital Logic Families

The switching characteristics of semiconductor devices have been discussed. Basically, there are two types of semiconductor devices: bipolar and unipolar. Based on these devices, digital integrated circuits have been made which are commercially available. Various digital functions are being fabricated in a variety of forms using bipolar and unipolar technologies. A group of compatible ICs with the same logic levels and supply voltages for performing various logic functions have been fabricated using a specific circuit configuration which is referred to as a logic family.

Classification of logic family

Bipolar Logic Families

The main elements of a bipolar IC are resistors, diodes (which are also capacitors) and transistors. Basically, there are two types of operations in bipolar ICs:

- Saturated, and Non-saturated.

In saturated logic, the transistors in the IC are driven to saturation, whereas in the case of non-saturated logic, the transistors are not driven into saturation.

The saturated bipolar logic families are:

- Resistor–transistor logic (RTL)

- Direct–coupled transistor logic (DCTL)
- Integrated–injection logic (I L)
- Diode–transistor logic (DTL)
- High–threshold logic (HTL)
- Transistor-transistor logic (TTL)

The non-saturated bipolar logic families are:

- Schottky TTL, and
- Emitter-coupled logic (ECL).

Unipolar Logic Families

MOS devices are unipolar devices and only MOSFETs are employed in MOS logic circuits. The MOS logic families are:

- PMOS
- NMOS
- CMOS

While in PMOS only *p*-channel MOSFETs are used and in NMOS only *n*-channel MOSFETs are used, in complementary MOS (CMOS), both *p*- and *n*-channel MOSFETs are employed and are fabricated on the same silicon chip.

CHARACTERISTICS:

With the widespread use of ICs in digital systems and with the development of various tech-nologies for the fabrication of ICs, it has become necessary to be familiar with the characteris-tics of IC logic families and their relative advantages and disadvantages. Digital ICs are classi-fied either according to the complexity of the circuit, as the relative number of individual basic gates (2-input NAND gates) it would require to build the circuit to accomplish the same logic function or the number of components fabricated on the chip.

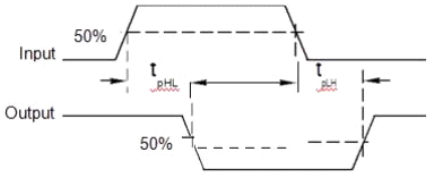
The various characteristics of digital ICs used to compare their performances are:

- Speed of operation
- Power dissipation
- Figure of merit
- Fan-out
- Current and voltage parameters
- Noise immunity
- Operating temperature range
- Power supply requirements
- Flexibilities available

1.Speed of Operation

The speed of a digital circuit is specified in terms of the propagation delay time. The input and output waveforms of a logic gate are shown in below Figure. The delay times are

measured between the 50 per cent voltage levels of input and out-put waveforms. There are two delay times: t_{pHL} - when the output goes from the HIGH state to the LOW state and t_{pLH} – corresponding to the output making a transition from the LOW state to the HIGH state. The propagation delay time of the logic gate



2.Power Dissipation

This is the amount of power dissipated in an IC. It is determined by the current, ICC , that it draws from the VCC supply, and is given by VCC * ICC . ICC is the average value of ICC (0) and ICC (1). This power is specified in milliwatts.

3.Figure of Merit

The figure of merit of a digital IC is defined as the product of speed and power. The speed is specified in terms of propagation delay time expressed in nanoseconds.
Figure of merit = propagation delay time (ns) *power (mW)

It is specified in pico joules (pJ) A low value of speed-power product is desirable. In a digital circuit, if it is desired to have high speed, i.e. low propagation delay, then there is a corresponding increase in the power dissipation and vice-versa.

4.**Fan-Out**

This is the number of similar gates which can be driven by a gate. High fan-out is advantageous because it reduces the need for additional drivers to drive more gates.

5. Current and Voltage Parameters

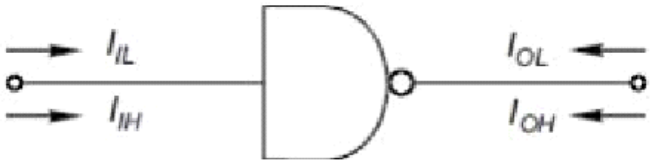
The following currents and voltages are specified which are very useful in the design of digital systems.

- High-level input voltage, VIH : This is the minimum input voltage which is recognized by the gate as logic 1.
- Low-level input voltage, VIL: This is the maximum input voltage which is recognized by the gate as logic 0.
- High-level output voltage, VOH : This is the minimum voltage available at the output corre-sponding to logic 1.
- Low-level output voltage, VOL: This is the maximum voltage available at the output correspond-ing to logic 0.
- High-level input current, IIH : This is the minimum current which must be supplied by a driving source corresponding to 1 level voltage.

- Low-level input current, IIL: This is the minimum current which must be supplied by a driving source corresponding to 0 level voltage.
- High-level output current, IOH : This is the maximum current which the gate can sink in 1 level.

- Low-level output current, IOL: This is the maximum cur- rent which the gate can sink in 0 level
- High-level supply current, ICC (1): This is the supply cur- rent when the output of the gate is at logic 1. Low-level supply current, ICC (0): This is the supply cur- rent when the output of the gate is at logic (0).

The current directions are illustrated in below Figure



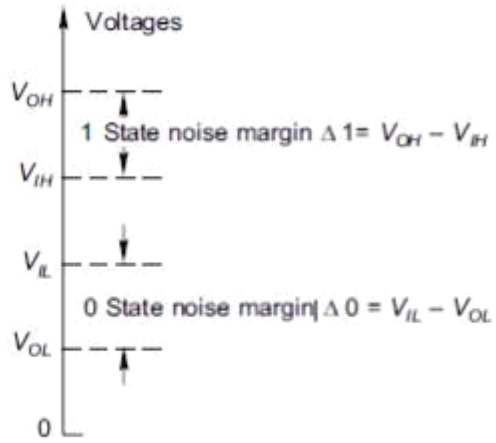
A gate with current directions marked.

6.Noise Immunity

The input and output voltage levels defined above are shown in Fig. Stray electric and magnetic fields may induce unwanted voltages, known as noise, on the connecting wires between logic circuits. This may cause the voltage at the input to a logic circuit to drop below VIH or rise above VIL and may produce undesired operation. The circuit's ability to tolerate noise signals is referred to as the noise immunity, a quantitative measure of which is called noise margin. Noise margins are illustrated in Fig. The noise margins defined above are referred to as dc noise margins. Strictly speaking, the noise is generally thought of as an a.c. signal with amplitude and pulse width. For high speed ICs, a pulse width of a few microseconds is extremely long in comparison to the propagation delay time of the circuit and therefore, may be treated as d.c. as far as the response of the logic circuit is concerned. As the noise pulse width decreases and approaches the propagation delay time of the circuit, the pulse duration is too short for the circuit to respond. Under this condition, a large pulse amplitude would be required to produce a change in the circuit output. This means that a logic circuit can effectively tolerate a large noise amplitude if the noise is of a very short duration. This is referred to as ac noise margin and is substantially greater than the dc noise margin. It is generally supplied by the manufacturers in the form of a curve between noise margin and noise pulse width.

7.Operating Temperature

The temperature range in which an IC functions properly must be known. The accepted temperature ranges are: 0 to + 70 °C for consumer and industrial applications and –55 °C to + 125 °C for military purposes.



Voltage levels and noise margins of ICs.

8. Power Supply Requirements

The supply voltage(s) and the amount of power required by an IC are important characteristics required to choose the proper power supply.

9. Flexibilities Available

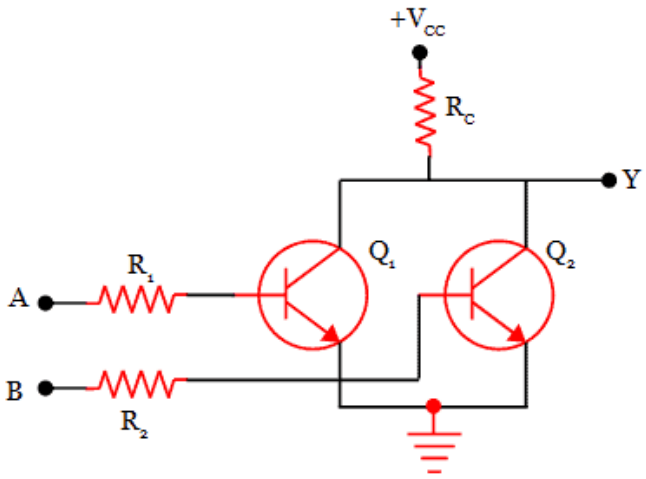
Various flexibilities are available in different IC logic families and these must be considered while selecting a logic family for a particular job. Some of the flexibilities available are:

- *The breadth of the series:* Type of different logic functions available in the series.
- *Popularity of the series:* The cost of manufacturing depends upon the number of ICs manufactured. When a large number of ICs of one type are manufactured, the cost per function will be very small and it will be easily available because of multiple sources.
- *Wired-logic capability:* The outputs can be connected together to perform additional logic without any extra hardware.
- *Availability of complement outputs:* This eliminates the need for additional inverters. 5. *Type of output:* Passive pull-up, active pull-up, open-collector/drain, and tristate.

SATURATED BIPOLAR LOGIC FAMILIES

5.2 RESISTOR-TRANSISTOR LOGIC (RTL)

RTL consists of resistors and transistors. In RTL, transistors operate in cut-off region or saturation region as per the input voltage applied. The circuit of a two-inputs resistor-transistor logic NOR gate is given below. Here A and B are the inputs of the gate and Y is the output.



2-input RTL NOR gate

Operation

- When the transistor operates in saturation region, maximum current flows through resistor *RC*. The output voltage *VY* = *V*CEsat (*V*CEsat = 0.2 V for silicon and 0.1 V for germanium); it is logic 0 level voltage. When the transistor operates in cut-off, no current flows through resistor *RC* and the output voltage *VY* = *V*CC = +5 V; it is logic 1 level voltage.
- When both the inputs are in logic 0, transistors *T*1 and *T*2 operate in cut-off, and the output is +*V*CC, i.e. +5 V (logic 1).
- When any one of the inputs is at logic 1 level, the corresponding transistor operates in saturation, and the output is *VY* = 0.2 V (logic 0).
- When both the inputs are at logic 1 level, both the transistors operate in saturation and the output is *VY* = 0.2 V (logic 0).

The operation of circuit is summarized in the below table

V_A	V_B	*Transistor T1*	*Transistor T2*	V_Y
Logic 0	Logic 0	Cut-off	Cut-off	Logic 1
Logic 0	Logic 1	Cut-off	Saturation	Logic 0
Logic 1	Logic 0	Saturation	Cut-off	Logic 0
Logic 1	Logic 1	Saturation	Saturation	Logic 0

In terms of 0 and 1, the above table can be written as in the below Table

Operation of RTL NOR gate

VA	*VB*	*VY*
0	0	1
0	1	0
1	0	0
1	1	0

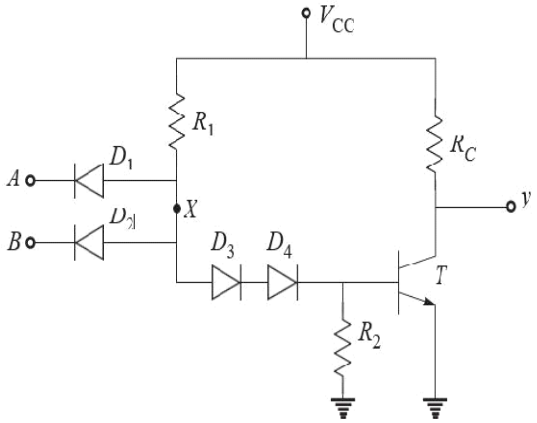
The circuit diagram acts as a two-inputs NOR gate and the above Table is the truth table of NOR gate.

The RTL suffers from a few drawbacks as listed below:

- Low noise margin (Typically 0.1 V)
- Fan-out is poor (Typically 5)
- Propagation delay is high and the speed of operation is low (Typically 12 ns)
- High power dissipation (Typically 12 mW)

5.2 DIODE-TRANSISTOR LOGIC (DTL)

The circuit of a DTL consists of diodes and transistors. The circuit of a two-inputs diode-transistor logic NAND gate is shown below.



Two-inputs DTL NAND gate Operation

- When the transistor operates in saturation, the output voltage $V(0) = V\text{CEsat} = 0.2$ V; and when it operates in cut-off, the output voltage $V(1) = V\text{CC} = +5$ V.
- When both the inputs are in logic 0, $V(0) = V\text{CEsat} = 0.2$ V, the input diodes are forward biased, voltage at point *x* is $Vx = V(0) + VD = 0.2 + 0.7 = 0.9$ V which is not sufficient to drive the transistor in saturation, because the voltage desired at point *x* to drive the transistor in saturation should be $V\text{BEsat} + V\text{D4} + V\text{D3} = 0.8 + 0.7 + 0.9 = 2.2$ V. The transistor operates in cut-off and the output voltage is in logic 1 state.
- When any one of the inputs is in logic 1, the corresponding diode is forward biased. Voltage at point *x* is $Vx = 0.2\text{ V} + 0.7\text{ V} = 0.9$ V; the transistor operates in cut-off and the output voltage is in logic 1 state.
- When all the inputs are in logic 1 state, the diodes *D*1 and *D*2 are reverse biased. The resistances *R*1 and *R*2 are selected such that the transistor operates in saturation and the output voltage is in logic 0 state.

The operation of the circuit is summarized in Table below

Inputs	Diodes	Transistor Output

A	B	D1	D2	T	Y
Logic 0	Logic 0	Forward biased	Forward biased	Cut-off	Logic 1
Logic 0	Logic 1	Forward biased	Reverse biased	Cut-off	Logic 1
Logic 1	Logic 0	Reverse biased	Forward biased	Cut-off	Logic 1
Logic 1	Logic 1	Reverse biased	Reverse biased	Saturation	Logic 0

In terms of 0 and 1, the above Table can be written as in Table below

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

Following are the advantages and disadvantages of DTL over RTL.

Advantages

- Fan-out is high
- Power dissipation is 8–12 mW
- Noise immunity is good

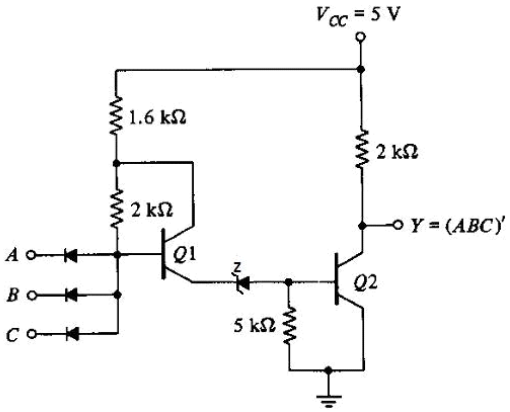
Disadvantages

- More elements are required
- Propagation delay is more (typically 30 ns) and hence the speed of operation is less

5.3 High Threshold Logic (**HTL**):

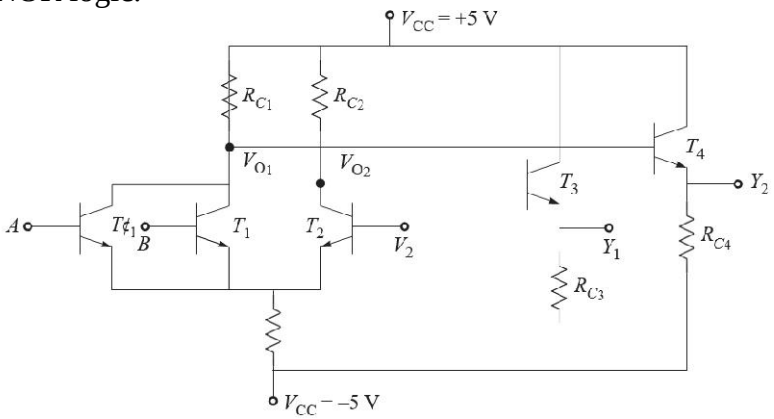
High Threshold Logic (**HTL**) is a variant of Diode–transistor logic which is used in such environments where noise is very high.

Operation : The threshold values at the input to a logic gate determine whether a particular input is interpreted as a logic 0 or a logic 1.(e.g. anything less than 1 V is a logic 0 and anything above 3 V is a logic 1. In this example, the threshold values are 1V and 3V). HTL incorporates Zener diodeto create a large offset between logic 1 and logic 0 voltage levels. These devices usually ran off a 15 V power supply and were found in industrial control, where the high differential was intended to minimize the effect of noise.



5.4 EMITTER COUPLED LOGIC

Emitter coupled logic (ECL) is faster than TTL family. The transistors of an emitter coupled logic are operated in cut-off or active region, it never goes in saturation and therefore the storage time is eliminated. Emitter coupled logic family is an example of unsaturated logic family. Figure below shows the circuit of an emitter- coupled logic OR/NOR gate. The circuit consists of difference amplifiers and emitter followers. Emitter terminals of the two transistors are connected together and hence it is called as emitter coupled logic. The emitter followers are used at the output of difference amplifier to shift the DC level. The circuit has two outputs Y1 and Y2, which are complementary. Y1 corresponds to OR logic and Y2 corresponds to NOR logic.



Emitter coupled Logic OR/NOR gate Operation

The input voltage of T_2 is $V_2 = V_{ref} = -1.15$ V.

- When both the inputs are in logic 0, $T_1$and T_1operate in cut-off and T_2operates in active region, voltage V_{O_1} is high, T_3 is ON, and the output at Y_2 is logic 1, voltage V_{O_2} is low, T_4 operates in cut-off and the output at Y_1 is logic 0.
- When any one of the inputs is in logic 1 level, the corresponding transistors T_1 or T_1 are operated in active region and T_2 operates in cut-off, voltage V_{O_1} is low, T_3 operates in cut-off and Y_2 is logic 0, voltage V_{O_2} is high, T_4 operates in active region and Y_1 is logic 1.

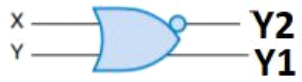
- When both the inputs are in logic 1 state, T_1 and T operate in active region and T_2 operates in cut-off, voltage V_{O_1} is low, T_3 operates in cut-off and Y_2 is logic 0, voltage V_{O_2} is high, T_4 operates in active region and Y_1 is logic 1.

The operation of the circuit is summarized in Table below.

Inputs		***Transistors***				***Output***	
A	***B***	***T1***	***T2***	***T3***	***T4***	***Y1***	***Y2***
Logic 0	Logic 0	Cut-off	Active	Active	Cut-off	Logic 0	Logic 1
Logic 0	Logic 1	Cut-off	Cut-off	Cut-off	Active	Logic 1	Logic 0
Logic 1	Logic 0	Activ	Cut-off	Cut-off	Active	Logic 1	Logic 0
Logic 1	Logic 1	Activ	Cut-off	Cut-off	Active	Logic 1	Logic 0

In terms of 0 and 1, Table above can be written as in Table below.

A	***B***	***Y1***	***Y2***
0	0	0	1
0	1	1	0
1	0	1	0
1	1	1	0



Symbol of OR / NOR gate

The circuit shown in above Fig acts as a two-input OR/NOR gate and its truth table is given in Table above. Also, the logic symbol of emitter coupled logic OR / NOR gate is shown here.

5.5 Classification of memories

Introduction: Memory is a collection of cells capable of storing a large quantity of binary information. In to which binary information is transferred for storage and from which information is available when needed for processing.

Memory Device: Device to which binary information is transferred for storage, and from which information is available for processing as needed.

Memory Unit: Is a collection of cells capable of storing a large quantity of binary Information Computer memory is broadly divided into two groups and they are:

- Primary /main memory
- Secondary memory/External Memory

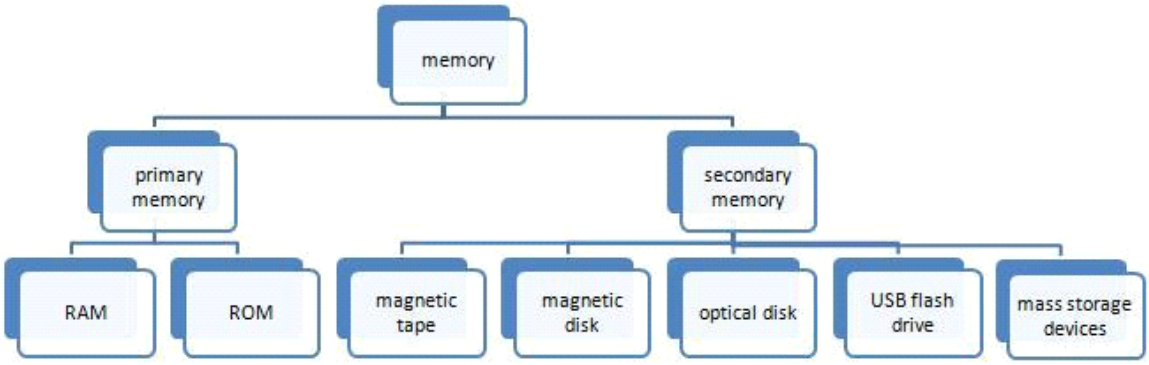


Fig Classification of Memory

Primary memory:

Primary memory is the only type of memory which is directly accessed by the CPU. The CPU continuously reads instructions stored in the primary memory and executes them. Any data that has to be operated by the CPU is also stored. The information is transferred to various locations through the BUS. Primary memories are of two types. They are:

- RAM
- ROM

RAM: It stands for Random Access Memory. Here data can be stored temporarily, so this type of memory is called as temporary memory or volatile memory because when power fails the data from RAM will be erased. The information stored in the RAM is basically loaded from the computer's disk and includes information related to the operating system and applications that are currently executed by the processor. RAM is considered random access because any memory cell can be directly accessed if its address is known. RAM is of distinct types like SRAM, DRAM, and VRAM.

ROM: It stands for Read Only Memory. In this, the data will be furnished by the manufacturers regarding the system, so this information can simply be read by the user but cannot add new data or it cannot be modified.

Types of ROM:

The required paths in a ROM may be programmed in four different ways.

- Mask programming
- Read-only memory (PROM)
- Erasable PROM (EPROM)
- Electrically-erasable PROM (EEPROM)

PROM : The PROM units contain all the fuses intact initially. Fuses are blown by application of a high voltage pulse to the device through a special pin by special instruments called PROM programmers. The program is once Written / programmed then it is irreversible. In A mask Programmable ROM, the data array is permanently stored during fabrication. This is done by selectively including switching element where a 1 is desired in the data array. The designer of the circuit should provide the ROM program, which is simply the content of the storage array to the IC manufacture. Once the ROM is fabricated, the data array cannot be charged. Mask

prorgammable ROMs are used when the ROM contents are not expected to change during the lifetime of the ROM.

EPROM: Floating gates served as programmed connections. When placed under ultraviolet light, short wave radiation discharges the gates and makes the EPROM returns to its initial state. It is reprogrammable. EPROMs use a special charge storage mechanism to enable or disable the switching element in the data array. A PROM programmer is used to store the charge at the selected switching elements while the EPROMs is programmed .The charge is retained by the EPROM. Thereby retaining the program until the EPROM is erased by using an ultraviolet light. Once erased, the EPROM can be reprogrammed. This type of ROM is useful in the early development phased of Digital circuit design, when it is often necessary to modify the data array

EEPROM: Erasable with an electrical signal instead of ultraviolet light. Longer time is needed to write flash ROM. It has limited times of write operations.

Random Access Memory (RAM)

The Random access memory, called "RAM" in short, is also known as the primary memory of the computer. RAM is considered as random access because of the fact that it can access any memory cell directly with the knowledge of the point of intersection of the row and column at that cell. It is also called as read write memory or the main memory or the primary memory. The programs and data that the CPU requires during execution of a program are stored in this memory. It is a volatile memory as the data loses when the power is turned off. RAM is further classified into two types- SRAM (Static Random Access Memory) and DRAM (Dynamic Random Access Memory).

Secondary memory: Secondary memory or auxiliary memory consists of slower and less expensive device that communicates indirectly with CPU via main memory. The secondary memory stores the data and keeps it even when the power fails. It is used to store or save large data or programs or other information. The secondary storage devices are explained below:

- Magnetic disks
- Magnetic tape
- Optical disk
- USB flash drive
- Mass storage devices

<u>Comparison between RAM and ROM</u>:

S.No	RAM	ROM
1	RAMs have both read and write capability.	ROMs have only read operation.
2	RAMs are volatile memories.	ROMs are non-volatile memories.

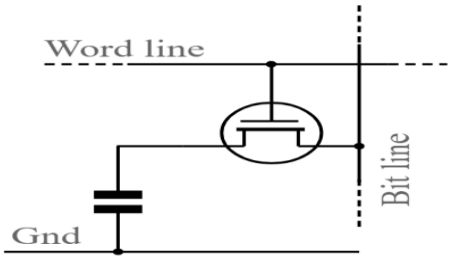
3	They lose stored data when the power is turned OFF.	They retain stored data even if power is turned off.
4	RAMs are available in both bipolar and MOS technologies.	RAMs are available in both bipolar and MOS technologies.
5	**Types**: SRAM, DRAM, EEPROM	**Types**: PROM, EPROM.

Comparison between SRAM and DRAM:

S.No	Static RAM	Dynamic RAM
1	It contains less memory cells per unit area.	It contains more memory cells per unit area.
2	Its access time is less, hence faster memories.	Its access time is greater than static RAM
3	It consists of number of flip-flops. Each flip-flop stores one bit.	It stores the data as a charge on the capacitor. It consists of MOSFET and capacitor for each cell.
4	Refreshing circuitry is not required.	Refreshing circuitry is required to maintain the charge on the capacitors every time after every few milliseconds. Extra hardware is required to control refreshing.
5	Cost is more	Cost is less.

Dynamic RAM

- DRAM memory technology has MOS technology at the heart of the design, fabrication and operation. Basic dynamic RAM or DRAM memory cell uses a capacitor to store each bit of data and a transfer device - a MOSFET - that acts as a switch.
- The level of charge on the memory cell capacitor determines whether that particular bit is a logical "1" or "0" - the presence of charge in the capacitor indicates a logic "1" and the absence of charge indicates a logical "0".
- The basic dynamic RAM memory cell has the format that is shown below. It is very simple and as a result it can be densely packed on a silicon chip and this makes it very cheap.
- Two lines are connected to each dynamic RAM cell - the Word Line (W/L) and the Bit Line (B/L) Connect as shown so that the required cell within a matrix can have data read or written to it.



The basic memory cell shown would be one of many thousands or millions of such cells in a complete memory chip. Memories may have capacities of 256 Mbit and more. To improve the write or read capabilities and speed, the overall dynamic RAM memory may be split into sub-arrays. The presence of multiple sub-arrays shortens the word and bit lines and this reduces the time to access the individual cells. For example a 256 Mbit dynamic RAM, DRAM may be split into 16 smaller 16Mbit arrays.

Static RAM (SRAM)

• Static Random Access Memory

• Static: Data value is retained as long as VDD is present.

• Random Access: Any location can read at a point in time.(Doesn't need sequential addresses)

SRAM can be built using either:

• D-type latch
• 6-transistor CMOS RAM cell

Edge Triggered D-type Register

• For use with a combinational circuit it is more important to have devices respond to clock edges.
  - D-type Latch „works" when **En**=1 or **En**=0
  - D-type Register „works" when **En** is rising or falling.

Edge triggered flip-flop for use in synchronous circuits.
• Uses 2 D-type transparent latches(Red Boxes) and 2 NOT gates

When the clock is low (Clk=0)
• The first D-type latch is ON,
• The value of D latched into first flipflop.

When clock goes high (Clk=1)
• The first D-type latch switches OFF and the second D-type latch is enabled.
• The output of latch 1 propagates through the second flipflop to the output.

Value of output is retained until next rising edge

Falling clock edges: Remove leftmost inverter from the circuit

6-Transistor Cell (Cross Coupled Inverter)

- For larger SRAM modules the above circuit is not very efficient o Transistor count per bit is too high

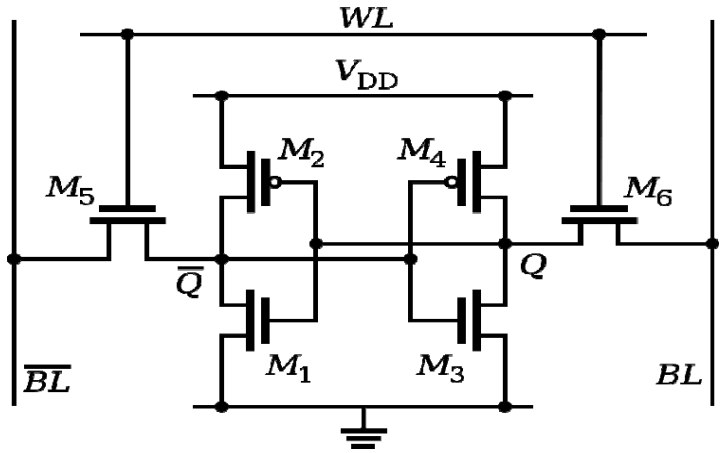


Figure transistor cell

TO READ:

- BIT lines are charged high
- Enable line WL is pulled high, switching access transistors M5 and M6 on`
- If value stored in /Q is 0, value is accessed through access transistor M5 on /BL.
- If value stored in Q is 1, charged value of Bit line BL is pulled up to VDD.
- Value is „sensed" on BL and /BL.

TO WRITE:

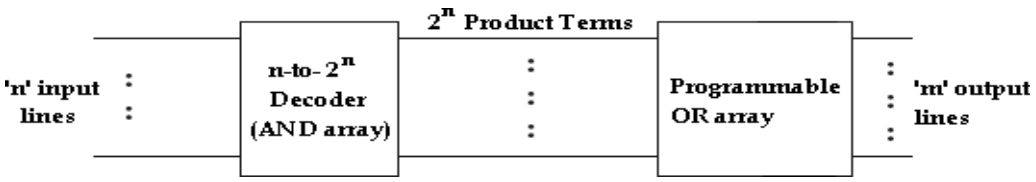
- Apply value to be stored to Bit lines BL and /BL
- Enable line WL is triggered and input value is latched into storage cell
- BIT line drivers must be stronger than SRAM transistor cell to override previous values

While Enable line is held low, the inverters retain the previous value Could use tri-state WE line on BIT to drive into specific state.

Transistor count per bit is only 6 + (line drivers & sense logic)

PROGRAMMABLE ROM:

PROMs are used for code conversions, generating bit patterns for characters and as look-up tables for arithmetic functions. As a PLD, PROM consists of a fixed AND-array and a programmable OR array. The AND array is an n-to-2^n decoder and the OR array is simply a collection of programmable OR gates. The OR array is also called the memory array. The decoder serves as a minterm generator. The n-variable minterms appear on the 2n lines at the decoder output. The 2n outputs are connected to each of the _m' gates in the OR array via programmable fusible links.



2^n x m PROM

Implementation of Combinational Logic Circuit using PROM

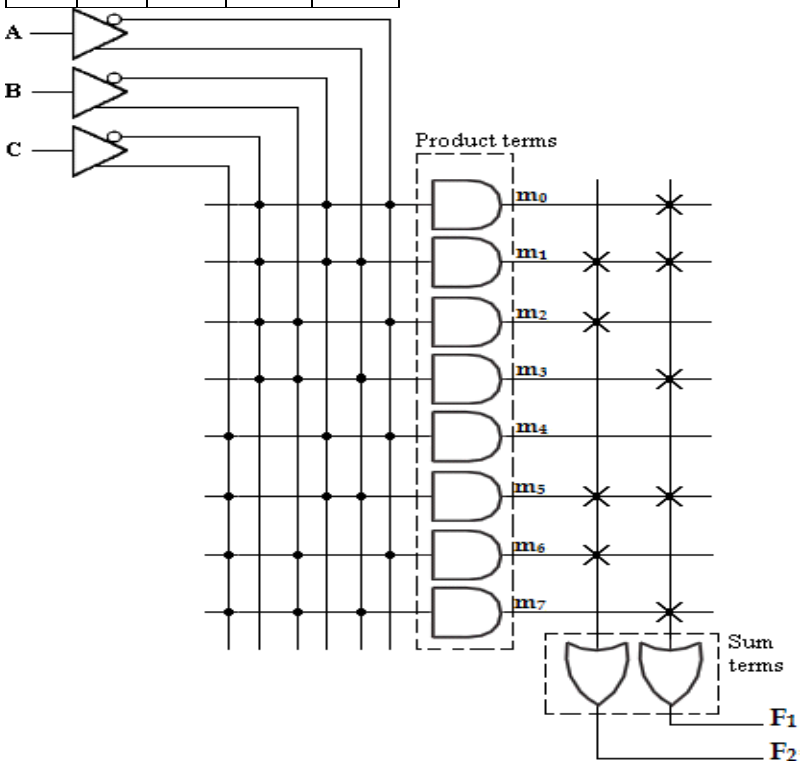
1.Using PROM realize the following expression

F_1 (A, B, C) = ∑m (0, 1, 3, 5, 7)

F_2 (A, B, C) = ∑m (1, 2, 5, 6)

Truth table for the given function

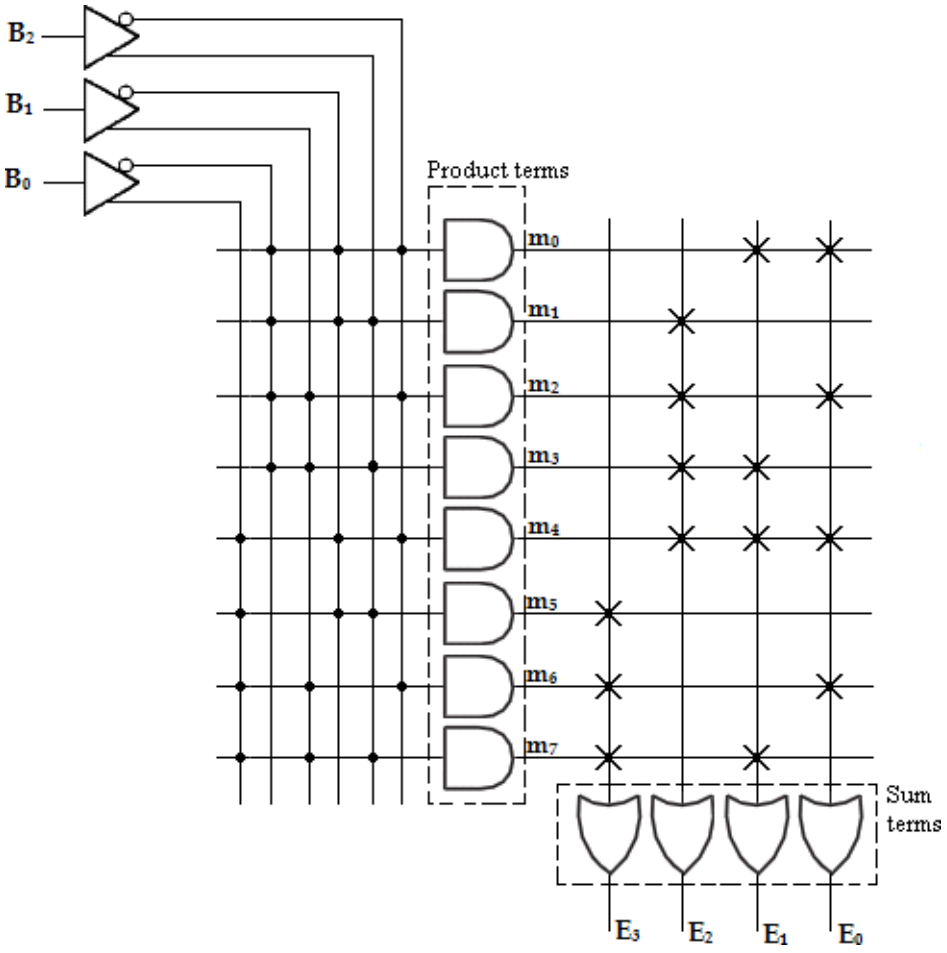
A	B	C	F_1	F_2
0	0	0	1	0
0	0	1	1	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	0
1	0	1	1	1
1	1	0	0	1
1	1	1	1	0



2. Design a combinational circuit using PROM. The circuit accepts 3-bit binary and generates its equivalent Excess-3 code.

Truth table for the given function

B_2	B_1	B_0	E_3	E_2	E_1	E_0
0	0	0	0	0	1	1
0	0	1	0	1	0	0
0	1	0	0	1	0	1
0	1	1	0	1	1	0
1	0	0	0	1	1	1
1	0	1	1	0	0	0
1	1	0	1	0	0	1
1	1	1	1	0	1	0



References

1.Morris Mano, Digital Design, Prentice Hall of India, 2001.
2. Jain, R.P., Modern Digital Electronics, Tata McGraw Hill, 3 rd Edition,1997.
3. Malvino.A.P. and Donald.P.Leach, Digital Principal and Applications, 4 th Edition, Tata McGraw Hill, 2007

Question Bank

Part A

1. Compare DTL and ECL gates.
2. Outline about CMOS logic circuits.
3. Point out the advantages of ECL.
4. How does ROM retain information?
5. Classify the logic gate families.
6. Define propagation delay in logic gate.
7. Summarize about EPROM
8. Enumerate the advantages of EEPROM over EPROM.
9. Compare SRAM and DRAM.

Part B

1. Discuss the basic concepts and the principle of operation of Bipolar SRAM cell.

2. Write short notes on the following:

i) RTL

ii) DTL

iii) ECL

3. Give the CMOS logic circuit for NOR gate and explain its

4. operation.

5. Explain NOR and OR gate construction using ECL. Also point out the characteristics of ECL family.

6. Draw the MOS logic circuit for NOT gate and explain its

7. Operation.

8. Design a CMOS inverter and explain its operation. Comment on its characteristics such as Fan-in, Fan-out, Power dissipation, Propagation delay and Noise margin. Compare its advantages over other logic families.

9. Describe with an aid of circuit diagram the operation of 2 input CMOS NAND gate and list out its advantages over other logic families.

10. Compare RAM, ROM , PROM,EPROM,EEPROM