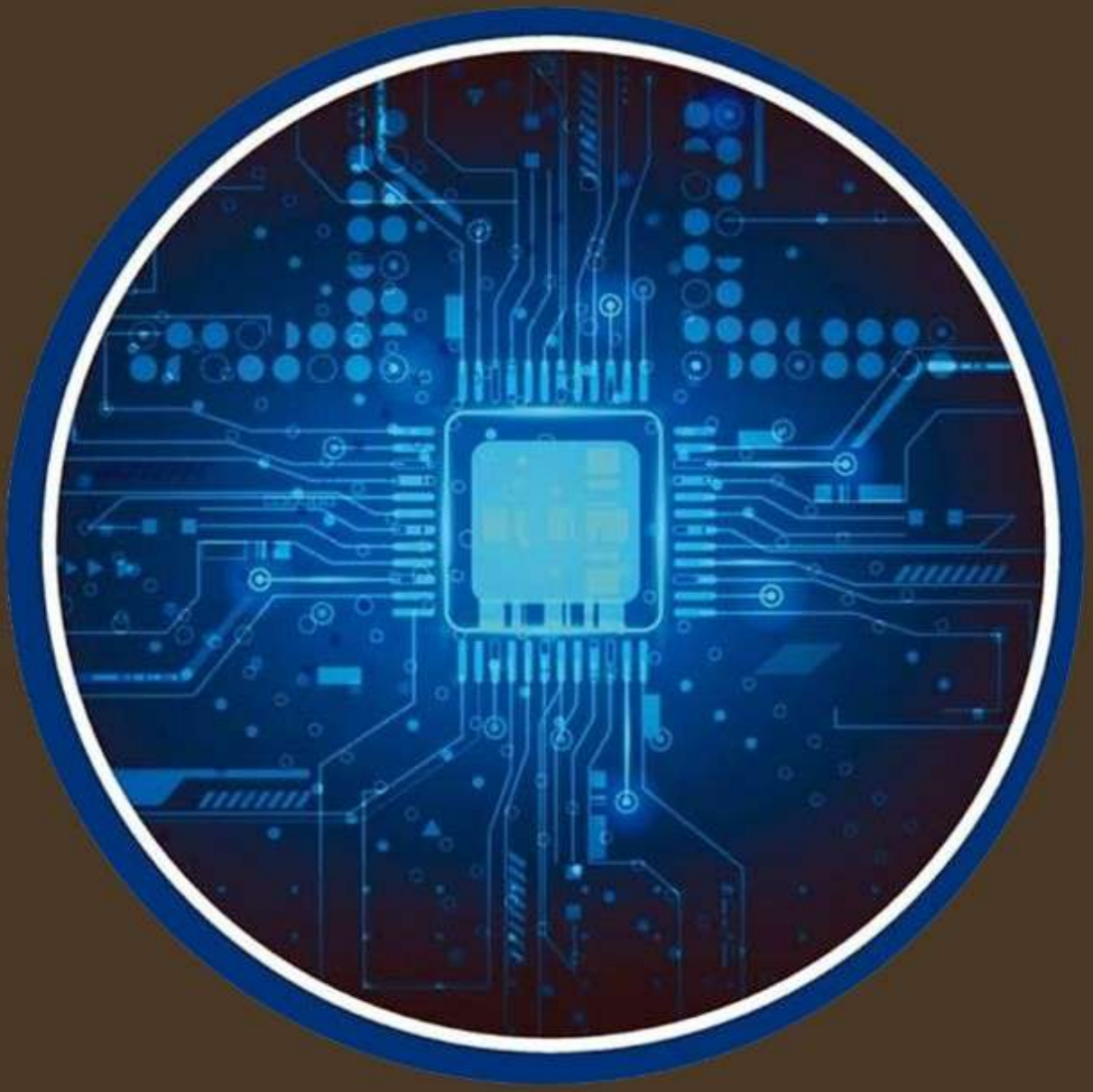
FUNDAMENTALS OF DIGITAL ELECTRONICS



Authors

Dr. Santhosh Kumar Allemki

Dr. Shatrughna Prasad Yadav

Dr. Vijayasaro Vijayaregunathan

Dr. Gambala Kiranmaye

EDUPRESS
PUBLISHERS

Fundamentals of Digital Electronics

By

Dr. Santhosh Kumar Allemki

Dr. Shatrughna Prasad Yadav

Dr. Vijayasaro Vijayaregunathan

Dr. G Kiranmaye

TABLE OF CONTENTS

CHAPTER 1
NUMBER SYSTEM

1.1Review of 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 1.1 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 1.2 Number system and their Base value

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
Hexa Decimal	16	0 1 2 3 4 5 6 7 8 9 A B C D E F

1.1.1 Decimal system:

Decimal system is composed of 10 numerals or symbols. These 10 symbolsare 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: 3.1410, 5210,102410

1.1.2 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.

1.1.3 Octal System:

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

8^3	8^2	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

1.1.4 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
110112	
$= (1*2^4)+(1*2^3)+0+(1*2^1)+(1*2^0)$	=16+8+0+2+1
Result	2710

- **Decimal to binary Conversion:**

There are 2 methods:

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

Reverse of Binary-To-Decimal Method

Decimal	Binary
4510	=32 + 0 + 8 + 4 +0 + 1
	$=2^5+0+2^3+2^2+0+2^0$
Result	=1011012

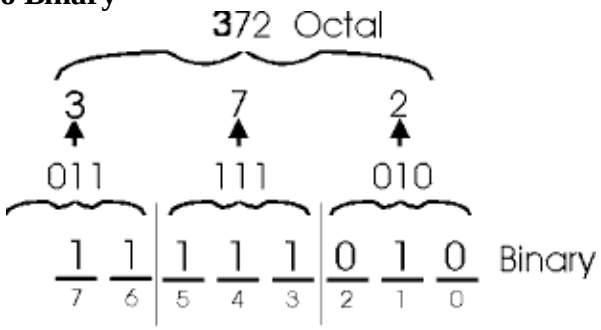
Repeat Division-Convert decimal to binary: This method uses repeated division by 2.

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 0102 = (100) (111) (010)2 = 4 7 28

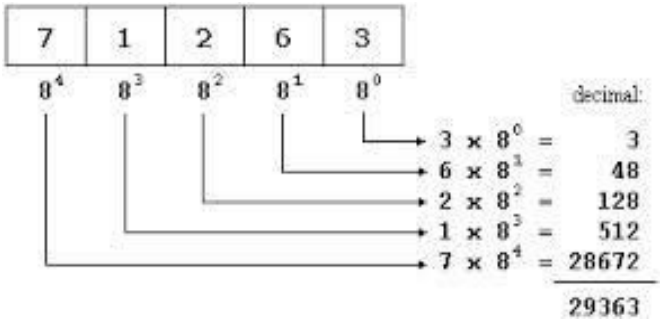
Octal to Binary



- **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

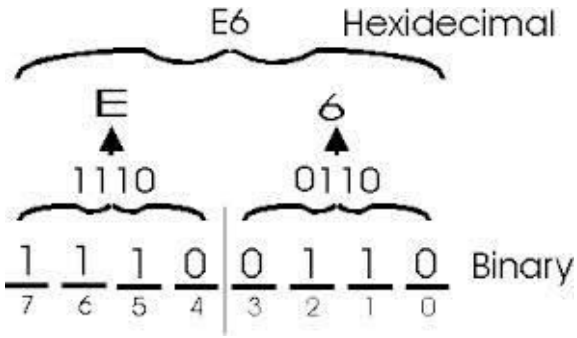
- **Decimal to Hexadecimal**

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 11112 = (1011) (0010) (1111)2 = B 2 F16

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)

1.3 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

1.4 Arithmetic Operations

Binary Equivalents
1 Nybble (or nibble) = 4 bits 1 Byte = 2 nibbles = 8 bits
1 Kilobyte (KB) = 1024 bytes
1 Megabyte (MB) = 1024 kilobytes = 1,048,576 bytes
1 Gigabyte (GB) = 1024 megabytes = 1,073,741,824 bytes

Binary 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

Example: **00011010 + 00001100 = 00100110**

Binary Subtraction
Rules of Binary Subtraction

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

Example:

00100101 - 00010001= 00010100	0	0	1	0	0	1	0	1
	+ 0	0	0	1	0	0	0	1
	0	0	0	1	0	1	0	0

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

Example

	0	0	1	0	1	0	0	1
×	0	0	0	0	0	1	1	0
	0	0	0	0	0	0	0	0
0	0	1	0	1	0	0	1	+
0	1	0	1	0	0	1		+
0	1	1	1	1	0	1	1	0

00101001 × 00000110 = 11110110

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 weightedcode, 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. Types of Binary Codes are

- Weighted Codes
- Non Weighted Codes
- Reflective Codes
- Sequential Codes
- Error Detecting and Correction Codes
- Alphanumeric Codes

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

Non weighted Codes

- 2421 code**:** This is a weighted code; its weights are 2, 4, 2 and 1. A decimal numberis 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 codes: 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.

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.

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. The gray code is a reflective digital code which has the special property that any twosubsequent 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.

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

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.

• **Hamming codes :** Hamming code adds a minimum number of bits to the data transmitted in a noisy channel, to be able to correct every possible one-bit error. It can detect (not correct) two-bit errors and cannot distinguish between 1-bit and 2-bits inconsistencies. It can't - in general - detect 3(or more)-bits errors.

• **Parity codes :** A parity bit is an extra bit included with a message to make the total number of 1's either even or odd. In parity codes, every data byte, or nibble (according to how user wants to use it) is checked if they have even number of ones or even number of zeros. Based on this information an additional bit is appended to the original data. Thus if we consider 8-bit data, adding the parity bit will make it 9 bit long.
At the receiver side, once again parity is calculated and matched with the received parity (bit 9), and if they match, data is ok, otherwise data is corrupt.

Two types of parity

1) Even parity: Checks if there is an even number of ones; if so, parity bit is zero. When the number of one's is odd then parity bit is set to 1.

2) Odd Parity: Checks if there is an odd number of ones; if so, parity bit is zero. When the number of one's is even then parity bit is set to 1.

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,

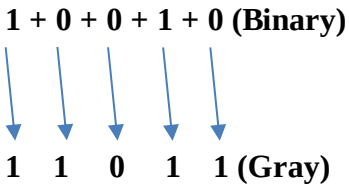
punctuation marks, and control characters. ASCII (American Standard Code for Information Interchange) is the best known. Unicode – a 16-bit coding system provides for foreign languages, mathematical symbols, geometrical shapes, dingbats, etc. It has become a world standard alphanumeric code for microcomputers 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).

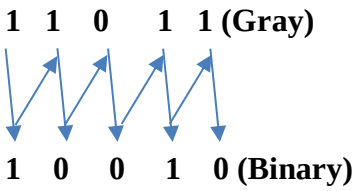
Example 1: Give the binary, BCD, Excess-3, gray code representations of numbers:5,8,14.

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

Example 2: Binary To Gray Code Conversion



Example 3: Gray Code To Binary Code Conversion



1.6 Code Converters:

Code to another code of binary code. The following are some of the most commonly used code converters:

- Binary-to-Gray code
- Gray-to-Binary code
- BCD-to-Excess-3
- Excess-3-to-BCD

- Binary-to-BCD
- BCD-to-binary
- Gray-to-BCD
- BCD-to-Gray
- 8 4 -2 -1 to BCD converter

Binary to Gray Converters:

The gray code is often used in digital systems because it has the advantage that only one bitin the numerical representation changes between successive numbers. The truth table for the binary-to-gray code converter is shown below,

Truth table

Decimal	Binary code				Gray code			
	B_3	B_2	B_1	B_0	G_3	G_2	G_1	G_0
0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1
2	0	0	1	0	0	0	1	1
3	0	0	1	1	0	0	1	0
4	0	1	0	0	0	1	1	0
5	0	1	0	1	0	1	1	1
6	0	1	1	0	0	1	0	1
7	0	1	1	1	0	1	0	0
8	1	0	0	0	1	1	0	0
9	1	0	0	1	1	1	0	1
10	1	0	1	0	1	1	1	1
11	1	0	1	1	1	1	1	0
12	1	1	0	0	1	0	1	0
13	1	1	0	1	1	0	1	1
14	1	1	1	0	1	0	0	1
15	1	1	1	1	1	0	0	0

K-map simplification:

For G_3

B_3B_2 \ B_1B_0	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	1	1	1	1
10	1	1	1	1

$G_3 = B_3$

For G_2

B_3B_2 \ B_1B_0	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	0	0	0	0
10	1	1	1	1

$G_2 = B_3'B_2 + B_3B_2'$
$= B_3 \oplus B_2$

For G_1

B_3B_2 \ B_1B_0	00	01	11	10
00	0	0	1	1
01	1	1	0	0
11	1	1	0	0
10	0	0	1	1

$G_1 = B_2'B_1 + B_2B_1'$
$= B_2 \oplus B_1$

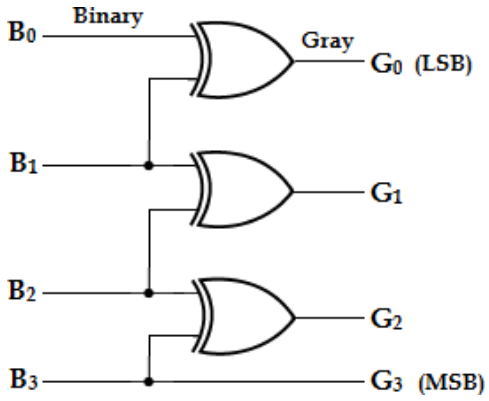
For G_0

B_3B_2 \ B_1B_0	00	01	11	10
00	0	1	0	1
01	0	1	0	1
11	0	1	0	1
10	0	1	0	1

$G_0 = B_1'B_0 + B_1B_0'$
$= B_1 \oplus B_0$

Now, the above expressions can be implemented using EX-OR gates as,

Logic Diagram:



Gray to Binary Converters:
The truth table for the gray-to-binary code converter is shown below,

Truth table:

Gray code				Binary code			
G_3	G_2	G_1	G_0	B_3	B_2	B_1	B_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	1
0	1	0	1	0	1	1	0
0	1	1	0	0	1	0	0
0	1	1	1	0	1	0	1
1	0	0	0	1	1	1	1
1	0	0	1	1	1	1	0
1	0	1	0	1	1	0	0
1	0	1	1	1	1	0	1
1	1	0	0	1	0	0	0
1	1	0	1	1	0	0	1
1	1	1	0	1	0	1	1
1	1	1	1	1	0	1	0

From the truth table, the logic expression for the binary code outputs can be written as,

$G_3 = \sum m\ (8, 9, 10, 11, 12, 13, 14, 15)$

$G_2 = \sum m\ (4, 5, 6, 7, 8, 9, 10, 11)$

$G_1 = \sum m\ (2, 3, 4, 5, 8, 9, 14, 15)$

$G_0 = \sum m\ (1, 2, 4, 7, 8, 11, 13, 14)$

K-map Simplification:
From the above K-map,

$\mathbf{B_3 = G_3}$

$B_2 = G_3'G_2 + G_3G_2'$

$\mathbf{B_2 = G_3 \oplus G_2}$

$B_1 = G_3'G_2'G_1 + G_3'G_2G_1' + G_3G_2G_1 + G_3G_2'G_1'$

$= G_3' (G_2'G_1 + G_2G_1') + G_3 (G_2G_1 + G_2'G_1')$

$= G_3' (G_2 \oplus G_1) + G_3 (G_2 \oplus G_1)'$ $\quad [x \oplus y = x'y + xy'], [(x \oplus y)' = xy + x'y']$

$\mathbf{B_1 = G_3 \oplus G_2 \oplus G_1}$

$B_0 = G_3'G_2' G_1'G_0 + G_3'G_2 \underline{\ } G_1G_0' + G_3G_2G_1 \underline{\ } G_0 + G_3G_2G_1 G_0' + G_3'G_2G_1'G_0' +$
$G_3G_2 \underline{\ } G_1'G_0' + G_3'G_2G_1G_0 + G_3G_2 \underline{\ } G_1 G_0.$

$= G_3'G_2' (G_1'G_0 + G_1G_0') + G_3G_2 (G_1'G_0 + G_1G_0') + G_1'G_0' (G_3'G_2 + G_3G_2') +$
$G_1G_0 (G_3'G_2 + G_3G_2').$

$= G_3'G_2' (G_0 \oplus G_1) + G_3G_2 (G_0 \oplus G_1) + G_1'G_0' (G_2 \oplus G_3) + G_1G_0 (G_2 \oplus G_3).$

$= G_0 \oplus G_1 (G_3'G_2' + G_3G_2) + G_2 \oplus G_3 (G_1'G_0' + G_1G_0)$

$= (G_0 \oplus G_1) (G_2 \oplus G_3)' + (G_2 \oplus G_3) (G_0 \oplus G_1)$ $\quad [x \oplus y = x'y + xy']$

$\mathbf{B_0 = (G_0 \oplus G_1) \oplus (G_2 \oplus G_3).}$

Now, the above expressions can be implemented using EX-OR gates as,

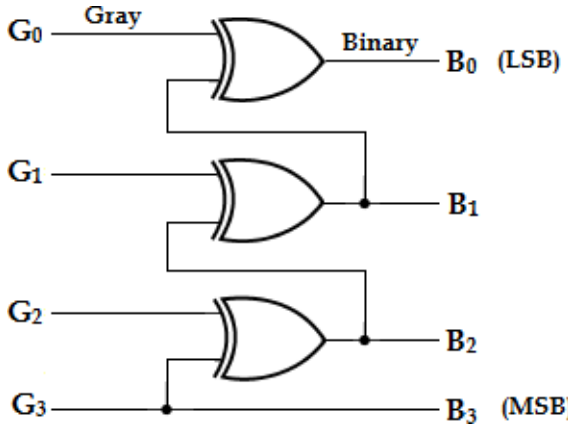


Fig. Logic diagram of 4-bit gray-to-binary converter

BCD –to-Excess-3 Converters:

Excess-3 is a modified form of a BCD number. The excess-3 code can be derived from the natural BCD code by adding 3 to each coded number.For example, decimal 12 can be represented in BCD as 0001 0010. Now adding 3 to each digitwe get excess-3 code as 0100 0101 (12 in decimal). With this information the truth table for BCD to Excess-3 code converter can be determined as,

Truth Table

Decimal	BCD code				Excess-3 code			
	B_3	B_2	B_1	B_0	E_3	E_2	E_1	E_0
0	0	0	0	0	0	0	1	1
1	0	0	0	1	0	1	0	0
2	0	0	1	0	0	1	0	1
3	0	0	1	1	0	1	1	0
4	0	1	0	0	0	1	1	1
5	0	1	0	1	1	0	0	0
6	0	1	1	0	1	0	0	1
7	0	1	1	1	1	0	1	0
8	1	0	0	0	1	0	1	1
9	1	0	0	1	1	1	0	0

From the truth table, the logic expression for the Excess-3 code outputs can be written as,

$E3 = \sum m\ (5, 6, 7, 8, 9) + \sum d\ (10, 11, 12, 13, 14, 15)$

$E_2 = \sum m\ (1, 2, 3, 4, 9) + \sum d\ (10, 11, 12, 13, 14, 15)$

$E_1 = \sum m\ (0, 3, 4, 7, 8) + \sum d\ (10, 11, 12, 13, 14, 15)$

$E_0 = \sum m\ (0, 2, 4, 6, 8) + \sum d\ (10, 11, 12, 13, 14, 15)$

K-map Simplification

For E_3

B_3B_2 \ B_1B_0	00	01	11	10
00	0	0	0	0
01	0	1	1	1
11	x	x	x	x
10	1	1	x	x

$E_3 = B_3 + B_2 (B_0 + B_1)$

For E_2

B_3B_2 \ B_1B_0	00	01	11	10
00	0	1	1	1
01	1	0	0	0
11	x	x	x	x
10	0	1	x	x

$E_2 = B_2B_1'B_0' + B_2' (B_0 + B_1)$

For E_1

B_3B_2 \ B_1B_0	00	01	11	10
00	1	0	1	0
01	1	0	1	0
11	x	x	x	x
10	1	0	x	x

$E_1 = B_1'B_0' + B_1B_0$

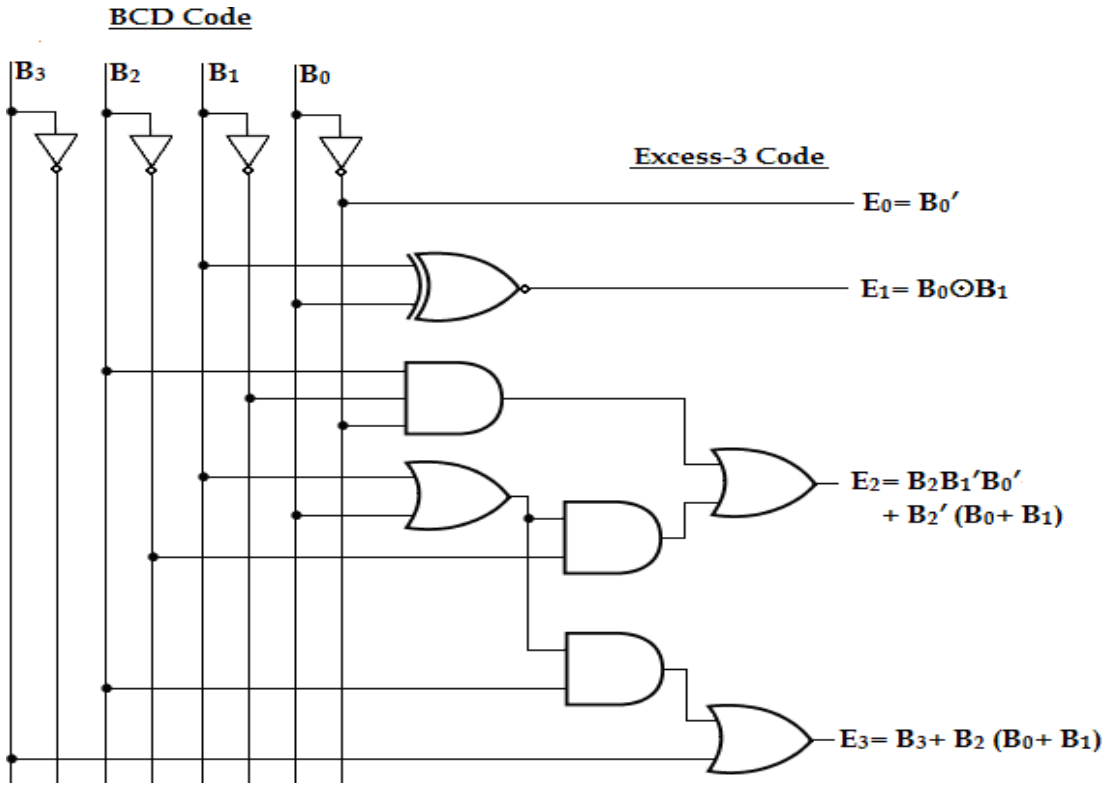
$= B_1 \odot B_0$

For E_0

B_3B_2 \ B_1B_0	00	01	11	10
00	1	0	0	1
01	1	0	0	1
11	x	x	x	x
10	1	0	x	x

$E_0 = B_0'$

Logic Diagram:



Excess-3 to BCD Converter: Truth table:

Decimal	Excess-3 code				BCD code			
	E_3	E_2	E_1	E_0	B_3	B_2	B_1	B_0
3	0	0	1	1	0	0	0	0
4	0	1	0	0	0	0	0	1
5	0	1	0	1	0	0	1	0
6	0	1	1	0	0	0	1	1
7	0	1	1	1	0	1	0	0
8	1	0	0	0	0	1	0	1
9	1	0	0	1	0	1	1	0
10	1	0	1	0	0	1	1	1
11	1	0	1	1	1	0	0	0
12	1	1	0	0	1	0	0	1

From the truth table, the logic expression for the Excess-3 code outputs can be written as,

$B_3 = \sum m\ (11, 12) + \sum d\ (0, 1, 2, 13, 14, 15)$

$B_2 = \sum m\ (7, 8, 9, 10) + \sum d\ (0, 1, 2, 13, 14, 15)$

$B_1 = \sum m\ (5, 6, 9, 10) + \sum d\ (0, 1, 2, 13, 14, 15)$

$B_0 = \sum m\ (4, 6, 8, 10, 12) + \sum d\ (0, 1, 2, 13, 14, 15)$

K-map Simplification

For B_3

E_3E_2 \ E_1E_0	00	01	11	10
00	X	X	0	X
01	0	0	0	0
11	1	X	X	X
10	0	0	1	0

$B_3 = E_3E_2 + E_3E_1E_0$

For B_2

E_3E_2 \ E_1E_0	00	01	11	10
00	X	X	0	X
01	0	0	1	0
11	0	X	X	X
10	1	1	0	1

$B_2 = E_2'E_1' + E_2E_1E_0 + E_3E_1E_0'$

Now, the above expressions the logic diagram can be implemented as,

For B_1

E_3E_2 \ E_1E_0	00	01	11	10
00	X	X	0	X
01	0	1	0	1
11	0	X	X	X
10	0	1	0	1

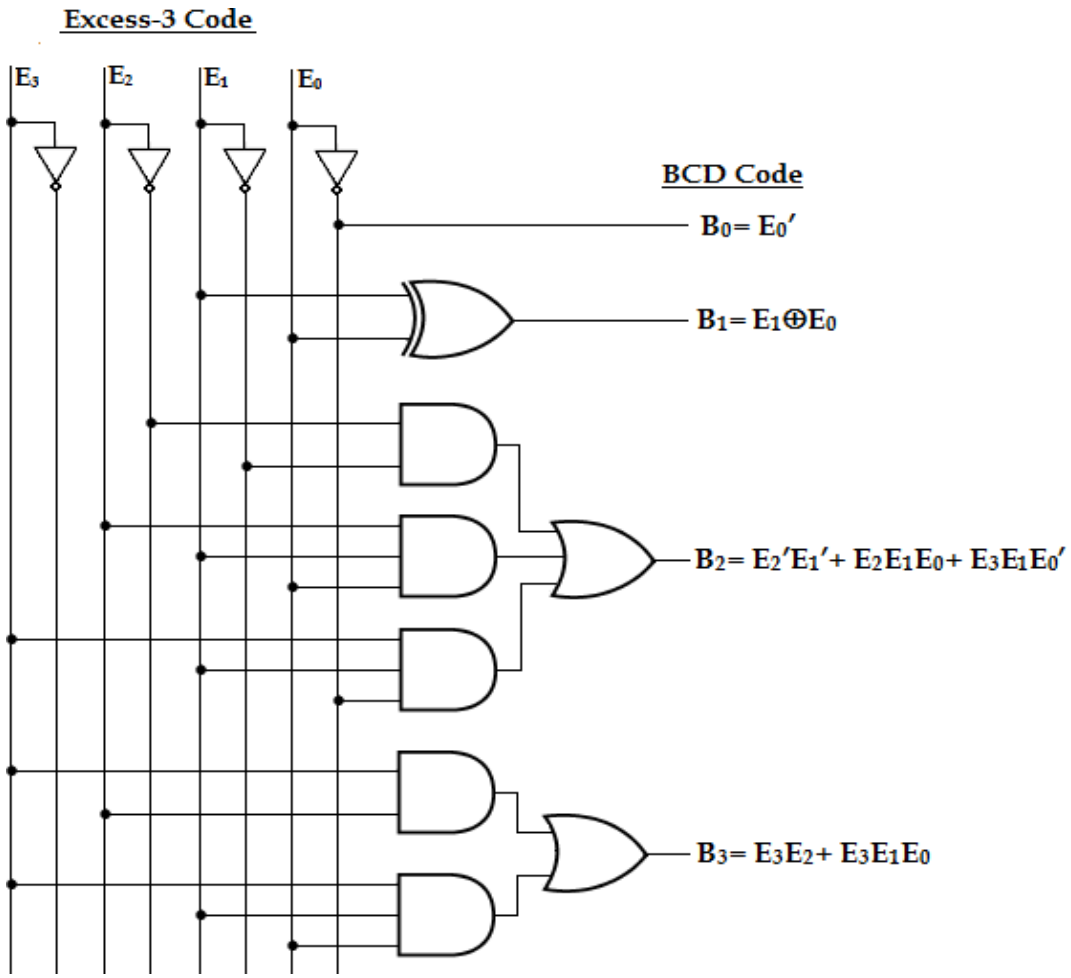
$B_1 = E_1'E_0 + E_1E_0'$
$= E_1 \oplus E_0$

For B_0

E_3E_2 \ E_1E_0	00	01	11	10
00	X	X	0	X
01	1	0	0	1
11	1	X	X	X
10	1	0	0	1

$B_0 = E_0'$

Logic Diagram:



BCD –to-Binary Converters:

The steps involved in the BCD-to-binary conversion process are as follows:

• The value of each bit in the BCD number is represented by a binary equivalent orweight.

• All the binary weights of the bits that are 1‘s in the BCD are added.

1 9

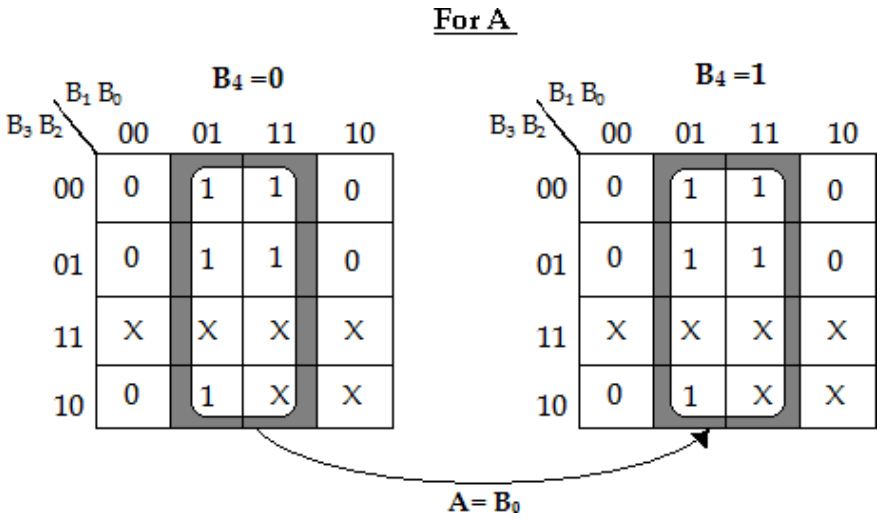
0001 1001

• The result of this addition is the binary equivalent of the BCD number. Two-digit decimal values ranging from 00 to 99 can be represented in BCDby two 4-bit code groups. For example, 19_{10} is represented as,

The left-most four-bit group represents 10 and right-most four-bit group represents 9. The binary representation for decimal 19 is 1910 = 110012.

BCD Code					Binary				
B_4	B_3	B_2	B_1	B_0	E	D	C	B	A
0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	1
0	0	0	1	0	0	0	0	1	0
0	0	0	1	1	0	0	0	1	1
0	0	1	0	0	0	0	1	0	0
0	0	1	0	1	0	0	1	0	1
0	0	1	1	0	0	0	1	1	0
0	0	1	1	1	0	0	1	1	1
0	1	0	0	0	0	1	0	0	0
0	1	0	0	1	0	1	0	0	1
1	0	0	0	0	0	1	0	1	0
1	0	0	0	1	0	1	0	1	1
1	0	0	1	0	0	1	1	0	0
1	0	0	1	1	0	1	1	0	1
1	0	1	0	0	0	1	1	1	0
1	0	1	0	1	0	1	1	1	1
1	0	1	1	0	1	0	0	0	0
1	0	1	1	1	1	0	0	0	1
1	1	0	0	0	1	0	0	1	0
1	1	0	0	1	1	0	0	1	1

K-map Simplification:



For B

B_4 =0

B_3B_2 \ B_1B_0	00	01	11	10
00	0	0	1	1
01	0	0	1	1
11	X	X	X	X
10	0	0	X	X

B_4 =1

B_3B_2 \ B_1B_0	00	01	11	10
00	1	1	0	0
01	1	1	0	0
11	X	X	X	X
10	1	1	X	X

$B = B_1B_4' + B_1'B_4$

$= B_1 \oplus B_4$

For C

B_4 =0

B_3B_2 \ B_1B_0	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	X	X	X	X
10	0	0	X	X

B_4 =1

B_3B_2 \ B_1B_0	00	01	11	10
00	0	0	1	1
01	1	1	0	0
11	X	X	X	X
10	0	0	X	X

$C = B_4'B_2 + B_2B_1' + B_4B_2'B_1$

For D

B_4 =0

B_3B_2 \ B_1B_0	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	X	X	X	X
10	1	1	X	X

B_4 =1

B_3B_2 \ B_1B_0	00	01	11	10
00	1	1	1	1
01	1	1	0	0
11	X	X	X	X
10	0	0	X	X

$D = B_4'B_3 + B_4B_3'B_2' + B_4B_3'B_1'$

For E

B_4 =0

B_3B_2 \ B_1B_0	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	X	X	X	X
10	0	0	X	X

B_4 =1

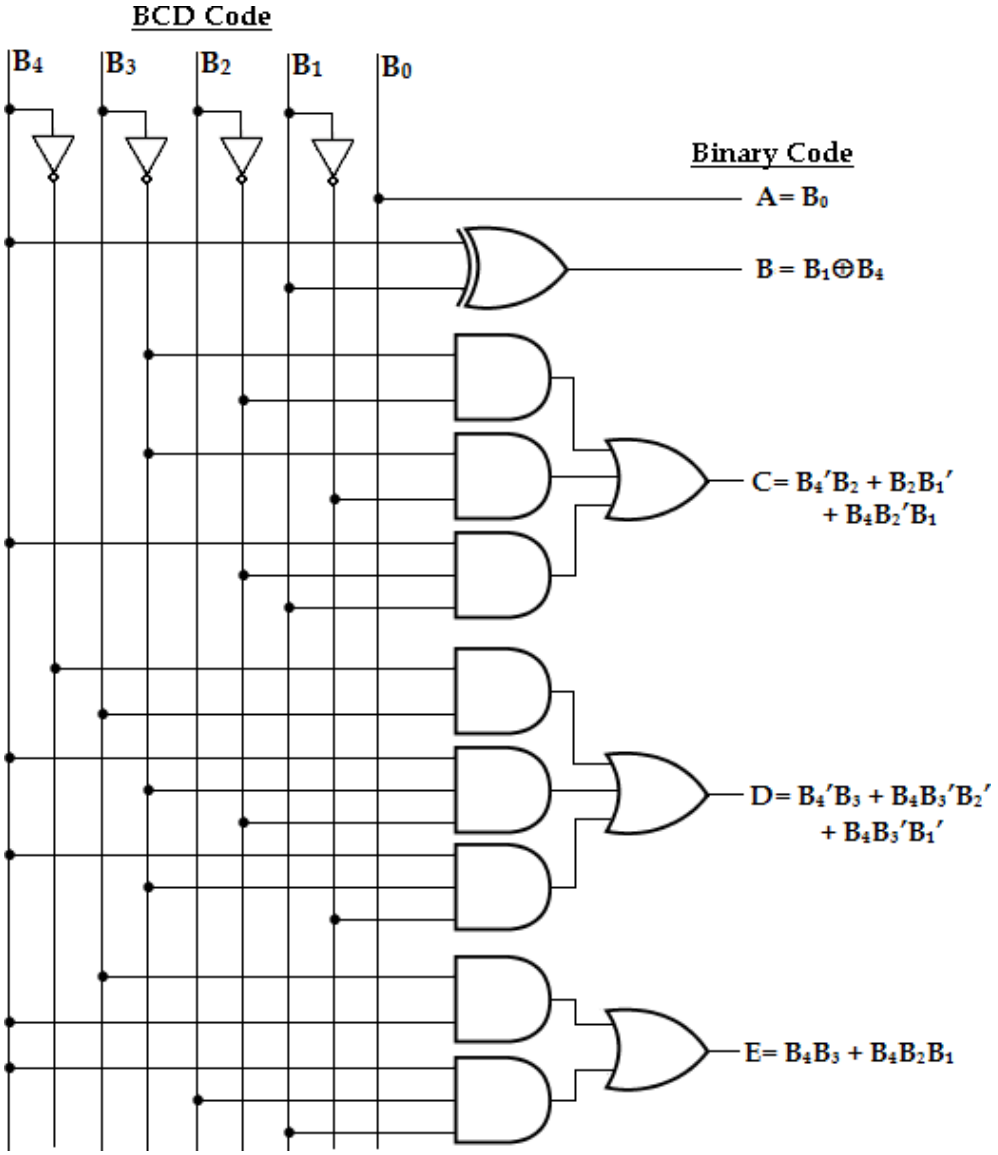
B_3B_2 \ B_1B_0	00	01	11	10
00	0	0	0	0
01	0	0	1	1
11	X	X	X	X
10	1	1	X	X

$E = B_4B_3 + B_4B_2B_1$

From the above K-map,
A= B_0
B= B_1B_4'+ B1'B4
= B_1 Ex-OR B_4
C= B_4'B2 + B_2B_1' + B4B_2'B1
D= B_4'B3 + B_4B_3'B2' + B4B_3'B1'E= B_4B_3 + $B_4B_2B_1$

Now, from the above expressions the logic diagram can be implemented as,

Logic Diagram:



Binary to BCD Converter:

The truth table for binary to BCD converter can be written as,

Truth Table

Decimal	Binary Code				BCD Code				
	D	C	B	A	B_4	B_3	B_2	B_1	B_0
0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	0	1
2	0	0	1	0	0	0	0	1	0
3	0	0	1	1	0	0	0	1	1
4	0	1	0	0	0	0	1	0	0
5	0	1	0	1	0	0	1	0	1
6	0	1	1	0	0	0	1	1	0
7	0	1	1	1	0	0	1	1	1
8	1	0	0	0	0	1	0	0	0
9	1	0	0	1	0	1	0	0	1
10	1	0	1	0	1	0	0	0	0
11	1	0	1	1	1	0	0	0	1
12	1	1	0	0	1	0	0	1	0
13	1	1	0	1	1	0	0	1	1
14	1	1	1	0	1	0	1	0	0
15	1	1	1	1	1	0	1	0	1

From the truth table, the logic expression for the BCD code outputs can be written as,

$B_0 = \sum m\ (1, 3, 5, 7, 9, 11, 13, 15)$

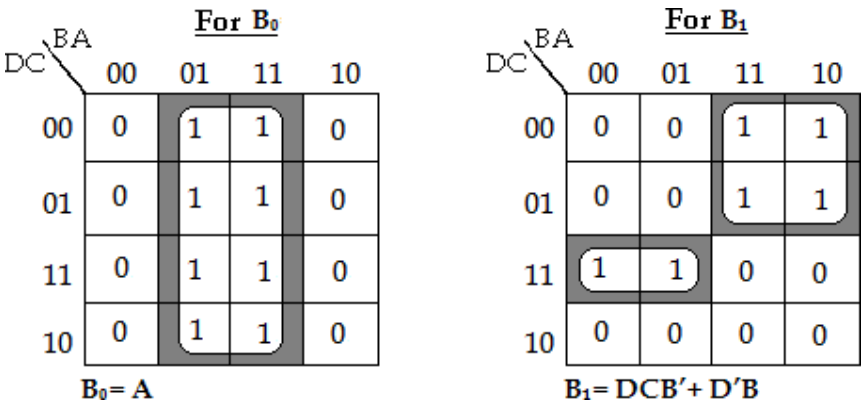
$B_1 = \sum m\ (2, 3, 6, 7, 12, 13)$

$B_2 = \sum m\ (4, 5, 6, 7, 14, 15)$

$B_3 = \sum m\ (8, 9)$

$B_4 = \sum m\ (10, 11, 12, 13, 14, 15)$

K-map Simplification:



For B_2

DC \ BA	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	0	0	1	1
10	0	0	0	0

$B_2 = D'C + CB$

For B_3

DC \ BA	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	0	0	0	0
10	1	1	0	0

$B_3 = DC'B'$

For B_4

DC \ BA	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	1	1	1	1
10	0	0	1	1

$B_4 = DC + DB$

From the above K-map, the logical expression can be obtained as,

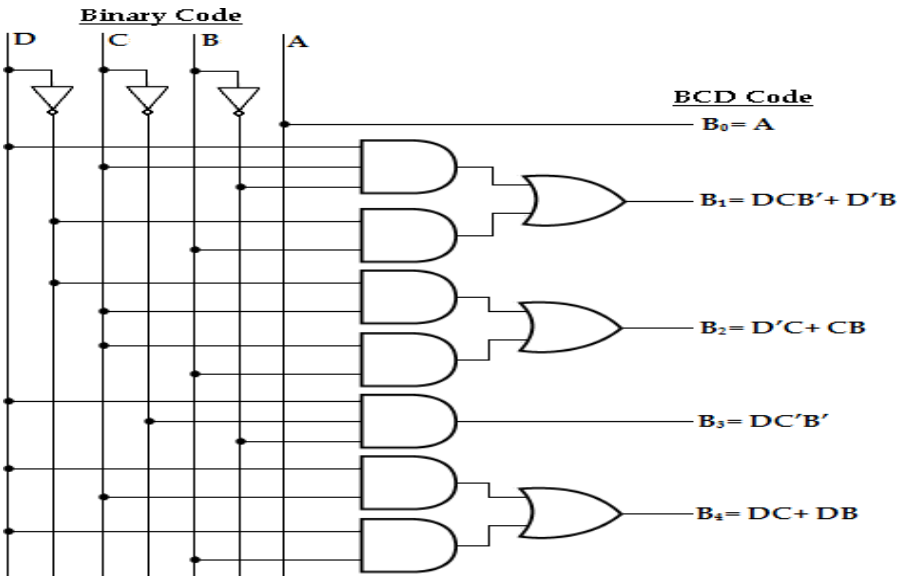
$B_0 = A$

$B_1 = DCB' + D'B$ $B_2 = D'C + CB$ $B_3 = DC'B'$

$B_4 = DC + DB$

Now, from the above expressions the logic diagram can be implemented as,

Logic Diagram:



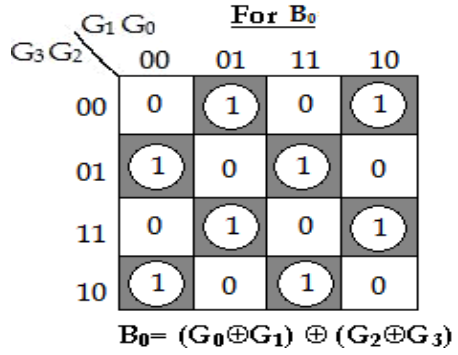
Gray to BCD Converter:

The truth table for gray to BCD converter can be written as,

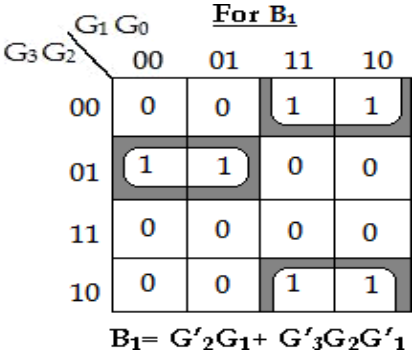
Truth Table:

Gray Code				BCD Code				
G_3	G_2	G_1	G_0	B_4	B_3	B_2	B_1	B_0
0	0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	0	1
0	0	1	1	0	0	0	1	0
0	0	1	0	0	0	0	1	1
0	1	1	0	0	0	1	0	0
0	1	1	1	0	0	1	0	1
0	1	0	1	0	0	1	1	0
0	1	0	0	0	0	1	1	1
1	1	0	0	0	1	0	0	0
1	1	0	1	0	1	0	0	1
1	1	1	1	1	0	0	0	0
1	1	1	0	1	0	0	0	1
1	0	1	0	1	0	0	1	0
1	0	1	1	1	0	0	1	1
1	0	0	1	1	0	1	0	0
1	0	0	0	1	0	1	0	1

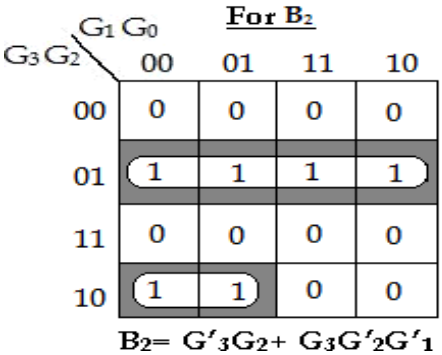
K-map Simplification:



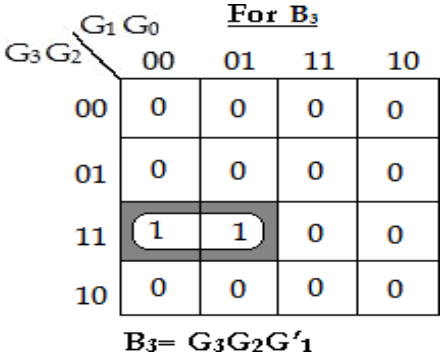
$B_0 = (G_0 \oplus G_1) \oplus (G_2 \oplus G_3)$



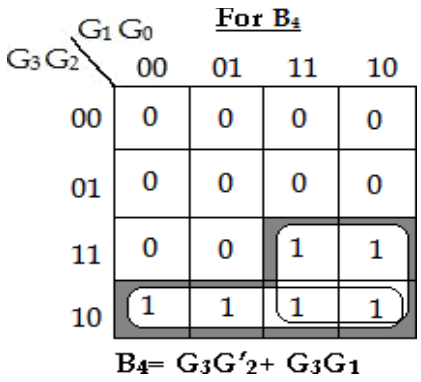
$B_1 = G'_2G_1 + G'_3G_2G'_1$



$B_2 = G'_3G_2 + G_3G'_2G'_1$



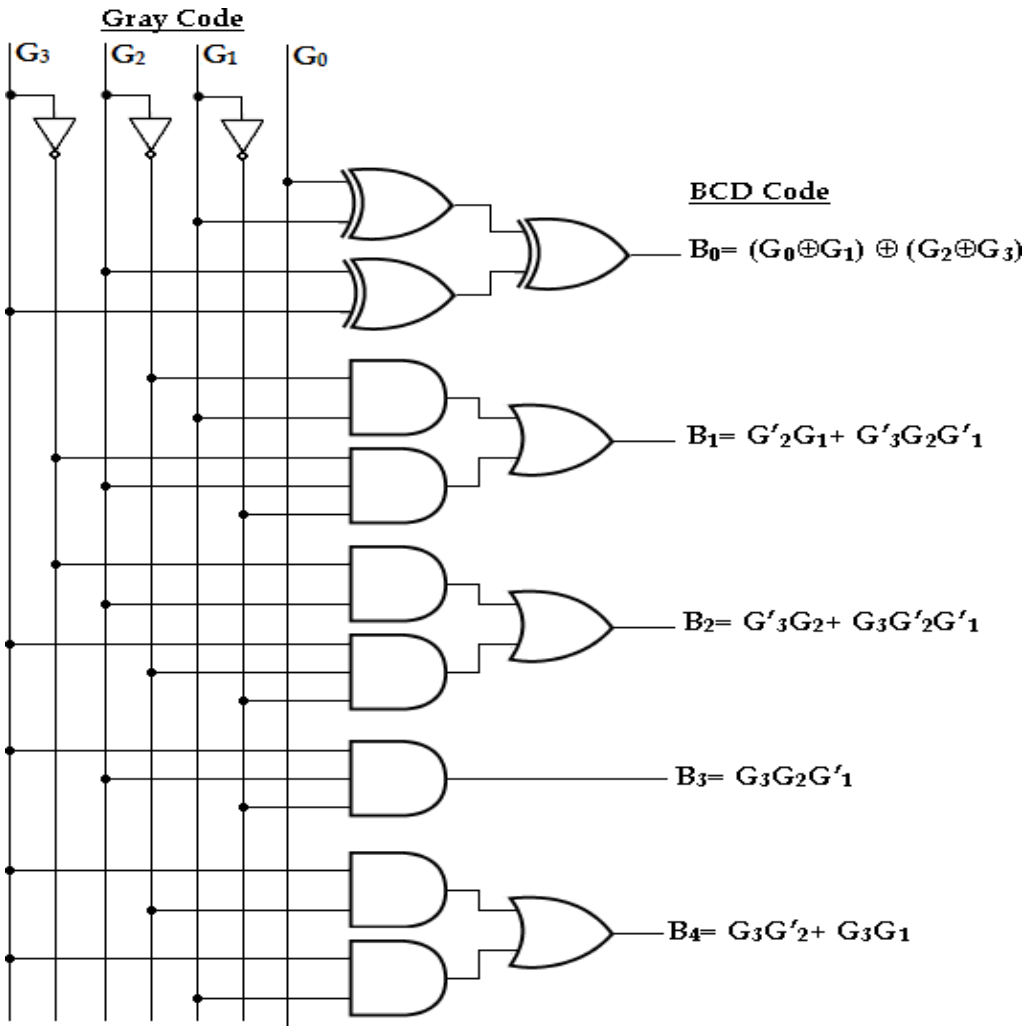
$B_3 = G_3G_2G'_1$



From the above K-map, the logical expression can be obtained as,
B_0= G'2G_1+ G'3G_2G'1 B_2= G'3G_2+ G_3G'2G'1 B_3= G_3G_2G'1
B_4= G_3G'2+ G_3G_1

Now, from the above expressions the logic diagram can be implemented as,

Logic Diagram:



BCD to Gray Converter:

The truth table for gray to BCD converter can be written as,

Truth table

BCD Code (8421)				Gray code			
B_3	B_2	B_1	B_0	G_3	G_2	G_1	G_0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	0	0	0	1	1
0	0	1	1	0	0	1	0
0	1	0	0	0	1	1	0
0	1	0	1	0	1	1	1
0	1	1	0	0	1	0	1
0	1	1	1	0	1	0	0
1	0	0	0	1	1	0	0
1	0	0	1	1	1	0	1

K-map Simplification:

For G_3

B_3B_2 \ B_1B_0	00	01	11	10
00	0	0	0	0
01	0	0	0	0
11	X	X	X	X
10	1	1	X	X

$G_3 = B_3$

For G_2

B_3B_2 \ B_1B_0	00	01	11	10
00	0	0	0	0
01	1	1	1	1
11	X	X	X	X
10	1	1	X	X

$G_2 = B_3 + B_2$

For G_1

B_3B_2 \ B_1B_0	00	01	11	10
00	0	0	1	1
01	1	1	0	0
11	X	X	X	X
10	0	0	X	X

$G_1 = B_2'B_1 + B_2B_1'$
$= B_2 \oplus B_1$

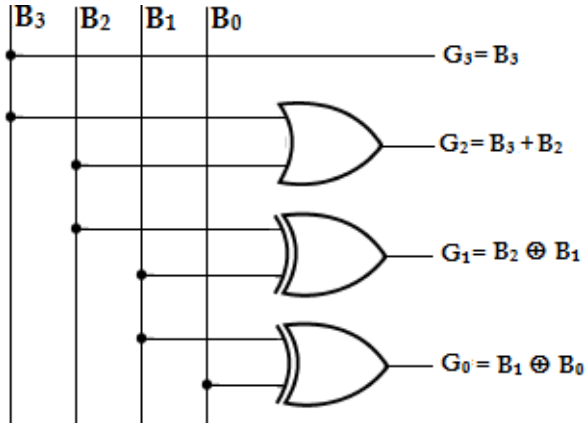
For G_0

B_3B_2 \ B_1B_0	00	01	11	10
00	0	1	0	1
01	0	1	0	1
11	X	X	X	X
10	0	1	X	X

$G_0 = B_1'B_0 + B_1B_0'$
$= B_1 \oplus B_0$

Now, from the above expressions the logic diagram can be implemented as,

Logic Diagram:



8 4 -2 -1 to BCD Converter:

The truth table for 8 4 -2 -1 to BCD converter can be written as,

Truth Table

Gray Code				BCD Code				
D	**C**	**B**	**A**	**B_4**	**B_3**	**B_2**	**B_1**	**B_0**
0	0	0	0	0	0	0	0	0
0	1	1	1	0	0	0	0	1
0	1	1	0	0	0	0	1	0
0	1	0	1	0	0	0	1	1
0	1	0	0	0	0	1	0	0
1	0	1	1	0	0	1	0	1
1	0	1	0	0	0	1	1	0
1	0	0	1	0	0	1	1	1
1	0	0	0	0	1	0	0	0
1	1	1	1	0	1	0	0	1
1	1	1	0	1	0	0	0	0
1	1	0	1	1	0	0	0	1
1	1	0	0	1	0	0	1	0

K-map Simplification:

For B_0

DC \ BA	00	01	11	10
00	0	X	X	X
01	0	1	1	0
11	0	1	1	0
10	0	1	1	0

$B_0 = A$

For B_1

DC \ BA	00	01	11	10
00	0	X	X	X
01	0	1	0	1
11	1	0	0	0
10	0	1	0	1

$B_1 = DCB'A' + D'B'A + D'BA' + C'B'A + C'BA'$
$= A'B'CD + D'(B'A + BA') + C'(B'A + BA')$
$= A'B'CD + D'(A \oplus B) + C'(A \oplus B)$
$= A'B'CD + (A \oplus B)(C' + D')$

For B_2

DC \ BA	00	01	11	10
00	0	X	X	X
01	1	0	0	0
11	0	0	0	0
10	0	1	1	1

$B_2 = D'CB'A' + C'A + C'B$
$= D'CB'A' + C'(A + B)$

For B_3

DC \ BA	00	01	11	10
00	0	X	X	X
01	0	0	0	0
11	0	0	1	0
10	1	0	0	0

$B_3 = ABCD + A'B'C'D$
$= D(ABC + A'B'C')$

For B_4

DC \ BA	00	01	11	10
00	0	X	X	X
01	0	0	0	0
11	1	1	0	1
10	0	0	0	0

$B_4 = B'CD + A'CD$
$= CD(A' + B')$

From the above K-map, the logical expression can be obtained as,

$B_0 = A$

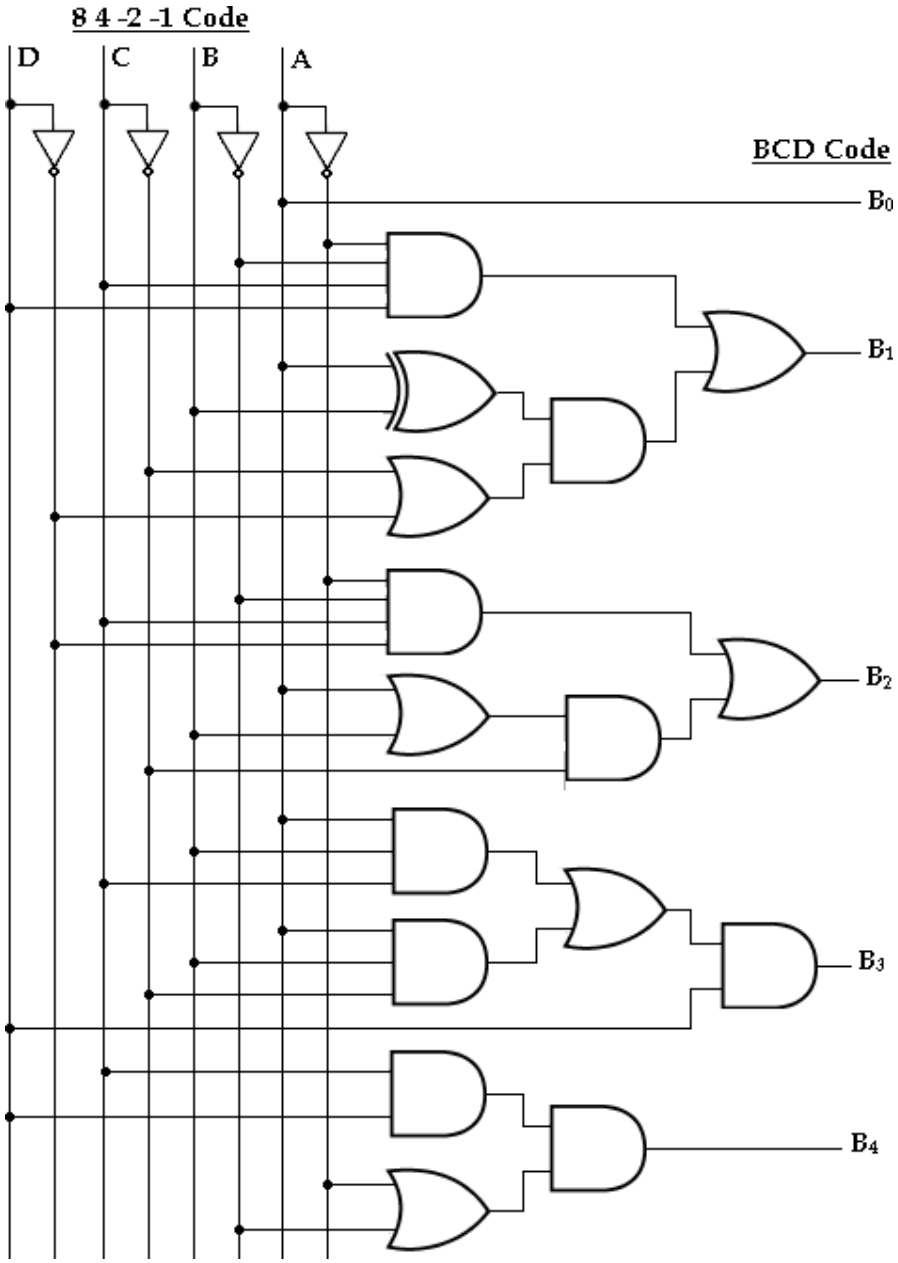
$B_1 = A'B'CD + (A \text{ Ex-OR } B)(C' + D')$

$B_2 = D'CB'A' + C'(A + B)$

$B_3 = D(ABC + A'B'C')$

$B_4 = CD(A' + B')$

Logic Diagram:



UNIT 2

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 calledBoolean algebra in his honor.

2.1 Fundamental postulates of Boolean algebra:

The postulates of a mathematical system forms the basic assumption from which it is possibleto 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

2.2 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]

2.3 Properties of Boolean algebra:

Commutative property:

Boolean addition is commutative, given by

$$x+ y = y+ x$$

According to this property, the order of the OR operation conducted on the variables makesno 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.

Associative property:

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 thevariables. 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 algebraremains 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.

$$x+ x' = 1 \text{ is } x.\ x' = 0$$

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'

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,

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

2.4 Boolean Functions:

Minimization of Boolean Expressions:

The Boolean expressions can be simplified by applying properties, laws and theorems ofBoolean 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') + xy'z [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]

2.5 Complement of A Function:

The complement of a function F is F' and is obtained from an interchange of 0's for 1'sand 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 –

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

$$(A B C D \dots F)' = A'+B'+ C'+ D'+ \dots +F'.$$

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]

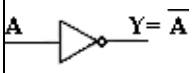
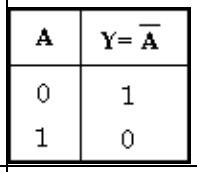
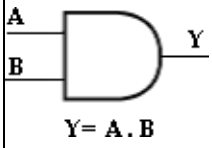
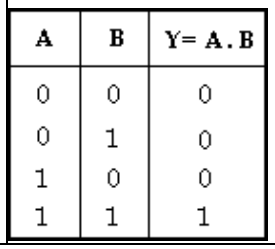
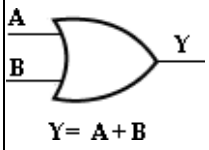
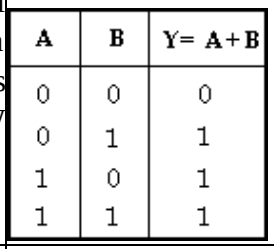
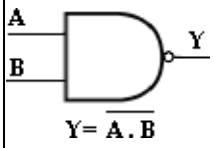
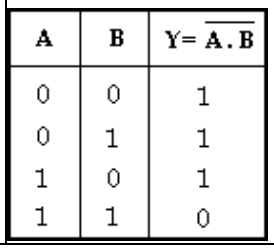
= x'+ y'z'

2.6 Logic Gates

2.6.1 Basic 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.

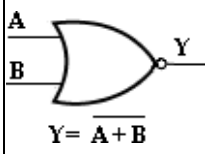
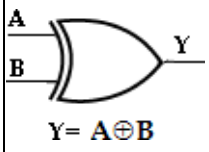
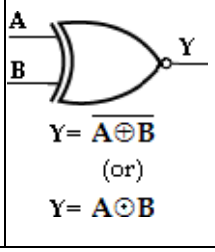
GATE	SYMBOL	OPERATION	TRUTH TABLE
NOT (7404)		NOT gate (Inversion), produces an inverted output pulse for a given input pulse.	
AND (7408)		AND gate performs logical **multiplication**. The output is HIGH only when all the inputs are HIGH. When any of the inputs are low, the output is LOW.	
OR (7432)		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.	
NAND (7400)		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.	

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

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

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

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

NOR		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.	A B $Y=\overline{A+B}$ 0 0 1 0 1 0 1 0 0 1 1 0
EX- OR		The output is HIGH only when odd number of inputs is HIGH.	A B Y= A⊕B 0 0 0 0 1 1 1 0 1 1 1 0
EX- NOR		The output is HIGH only when even number of inputs is HIGH. Or when all inputs are zeros.	A B Y= A⊙B 0 0 1 0 1 0 1 0 0 1 1 1

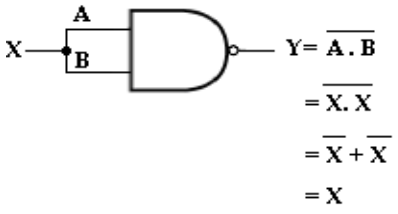
2.6.2 UNIVERSAL GATES:

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

- **NAND Gate**:

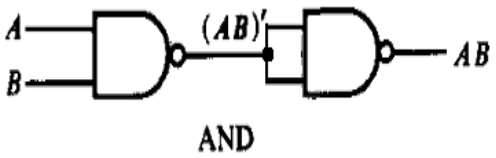
The NAND gate can be used to generate the NOT function, the AND function, the OR function and the NOR function.

NOT function: By connecting all the inputs together and creating a single common input.



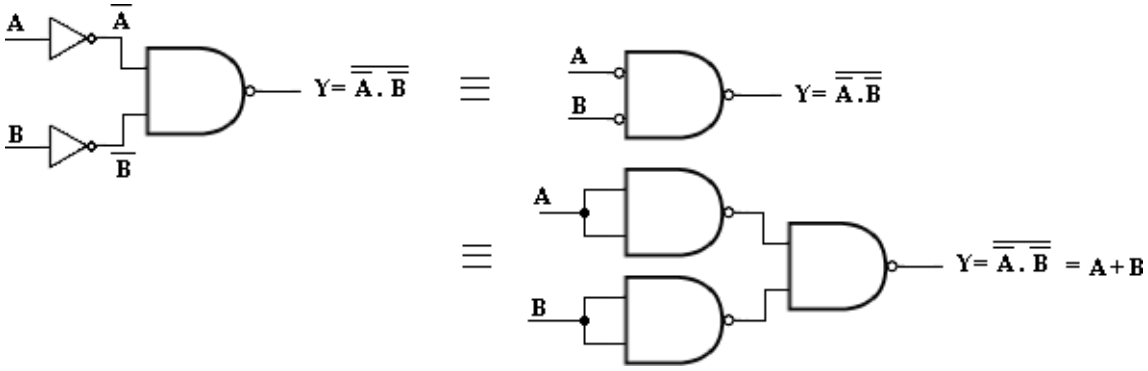
	A	B	$Y=\overline{A.B}$	
X= 0	0	0	1	Y= 1
	0	1	1	
	1	0	1	
X= 1	1	1	0	Y= 0

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



AND

OR function: Simply inverting inputs of the AND gate . i.e., By 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

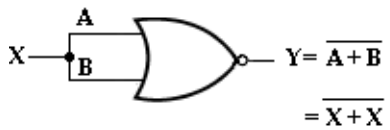
≡

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 Gate**:

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

• NOT function: By connecting all the inputs together and creating a single common input.



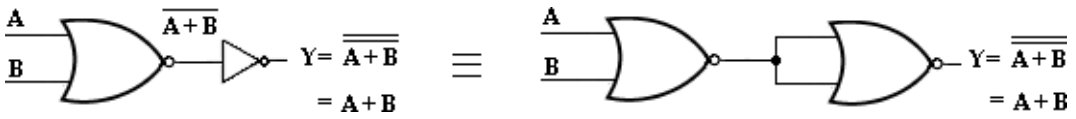
$= \overline{X+X}$

$= \overline{X} . \overline{X}$

$= \overline{X}$

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

- OR function: By Simply inverting output of the NOR gate.i.e,

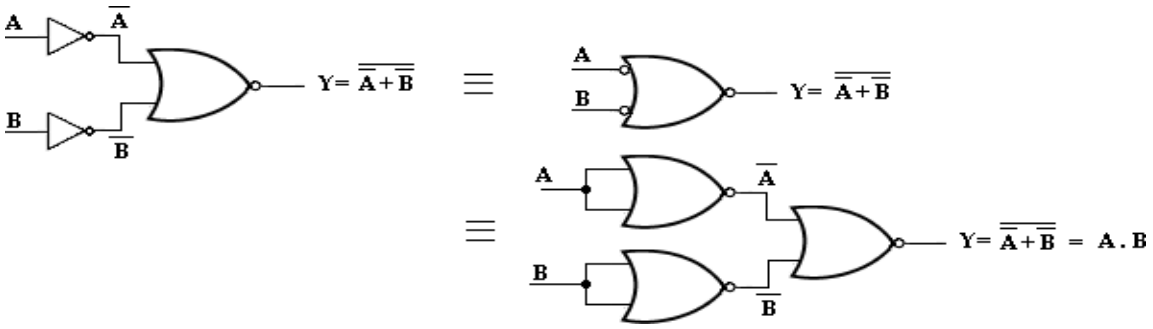


A	B	Y= A+B
0	0	0
0	1	1
1	0	1
1	1	1

≡

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., Bubble at the input of NOR gate indicates inverted input

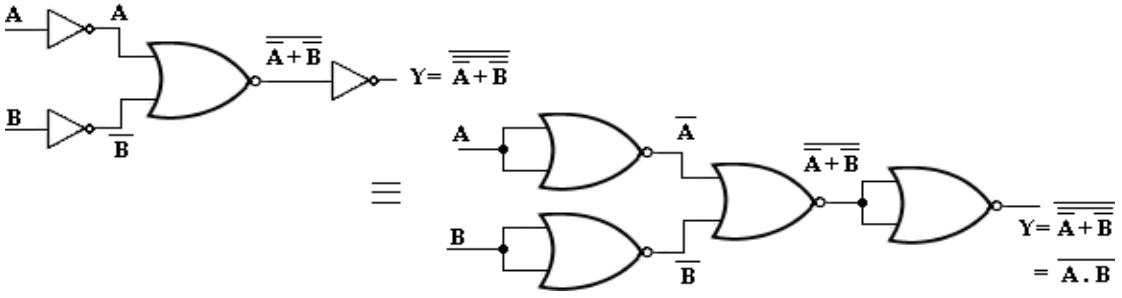


A	B	Y= A.B
0	0	0
0	1	0
1	0	0
1	1	1

≡

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

- NAND Function: By inverting inputs and outputs of the NOR gate



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

≡

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

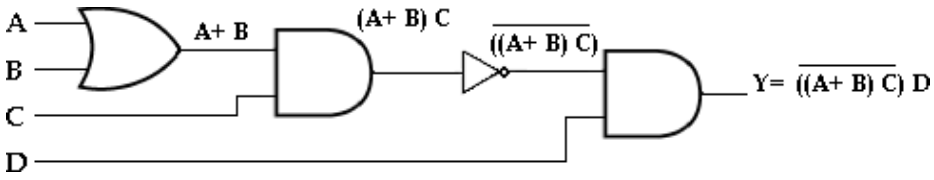
Conversion of AND/OR/NOT to NAND/NOR:

- Draw AND/OR logic.
- If NAND hardware has been chosen, add bubbles on the output of each AND gateand bubbles on input side to all OR gates.
- If NOR hardware has been chosen, add bubbles on the output of each OR gate and bubbles oninput side to all AND gates.
- Add or subtract an inverter on each line that received a bubble in step 2.
- Replace bubbled OR by NAND and bubbled AND by NOR.
- Eliminate double inversions.

Implement Boolean expression using NAND gates:

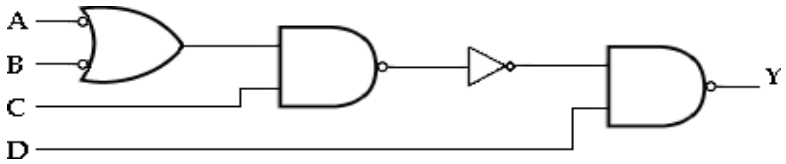
$\overline{((A+B)\,C)}\,D$

Original Circuit:



Soln:

NAND Circuit:



2.7 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, thereare 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= M0
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= M4
1	0	1	xy'z = m5	x'+ y+ z'= M5
1	1	0	xyz' = m6	x'+ y'+ z= M6
1	1	1	xyz = m7	x'+ y'+ z'= M7

Sum of Minterm: (Sum of Products)

1. Y= AB+ BC+ AC

2. Y= AB+ $\overline{B}$ C+ A$\overline{C}$ (Sum; Product terms)

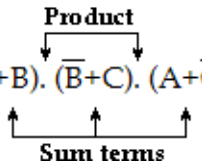
The logical sum of two or more logical product terms is called sum ofproducts expression.

It is logically an OR operation of AND operated variables such as:

Sum of Maxterm: (Product of Sums)

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

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



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

Canonical Sum of product expression:

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'

Steps to convert general SOP to standard SOP form:

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')

= AB+ AB'+ 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

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

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

= m7+ m6+ m5+ m4

= ∑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

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

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

= m7+ m6+ m5+ m4+ m3

= ∑m (3, 4, 5, 6, 7).

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

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

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

= ABCD+ ABCD'+ ABC'D+ ABC'D'+ 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 andits complement.
3) Expand the term by applying distributive law and reorder theliterals 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)

= M2. M3. M0

= ∏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)

= M0. M1. M4. M2

= ∏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')

= M0. M1. M2. M3

= ∏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')

= M2. M3. M0. M4. M1

= ∏M (0, 1, 2, 3, 4)

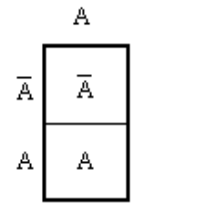
CHAPTER 3
KARNAUGH MAP MINIMIZATION

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.

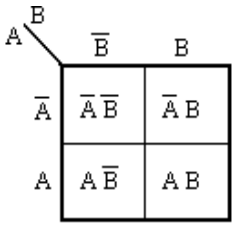
The Karnaugh map technique provides a systematic method for simplifying and manipulation of Boolean expressions. **A** K-map is a diagram made up of squares, with each square representing one minterm of the function that is to be minimized. For n variables on a Karnaugh map there are 2^n numbers of squares. 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.

3.1 Two- Variable, Three Variable and Four Variable Maps

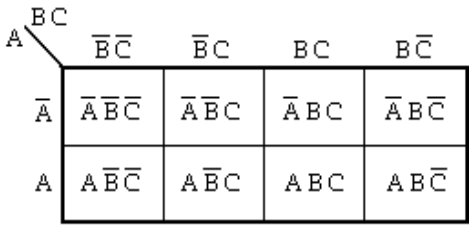
Karnaugh maps can be used for expressions with two, three, four and five variables. The number of cells in a Karnaugh map is equal to the total number of possible inputvariable combinations as is the number of rows in a truth table. For three variables, the number of cells is $2^3 = 8$. For four variables, the number of cells is $2^4 = 16$.



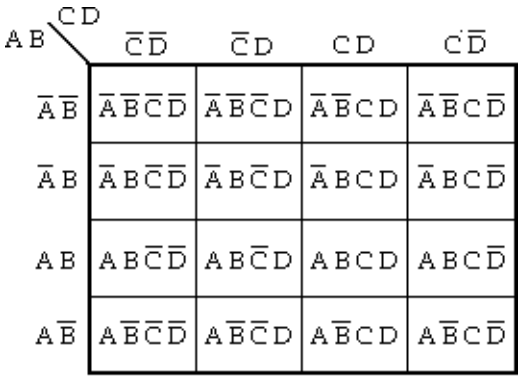
1-Variable map



2-Variable map



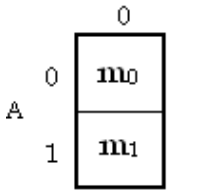
3-Variable map



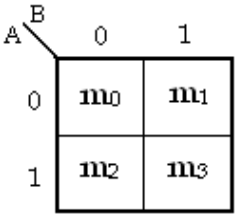
4-Variable map

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.

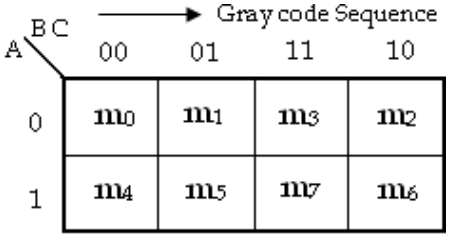
It is important to note that when we move from one cell to the next along any row or from one cell to the next along any column, one and only one variable in the product term changes (to a complement or to an uncomplemented form). Irrespective of number of variables the labels along each row and column must conform to a single change. Hence gray code is used to label the rows and columns of K-map as show now.



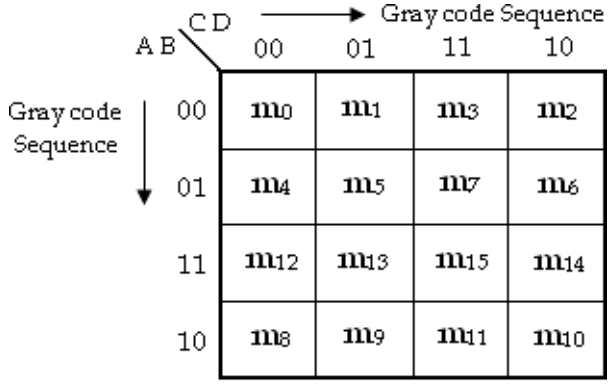
1 -Variable map



2 -Variable map



3 -Variable map



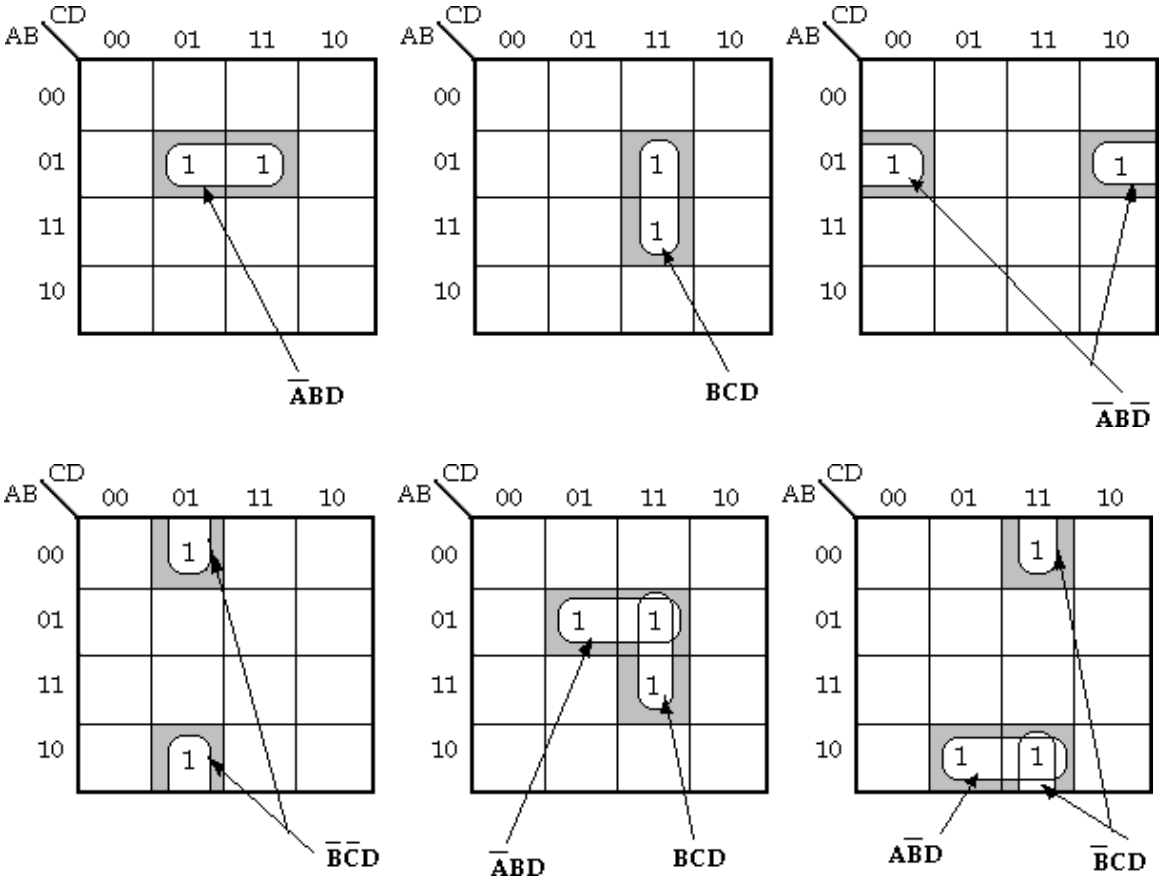
4- Variable map

3.2 Grouping cells for Simplification:

The grouping is nothing but combining terms in adjacent cells. The simplification is achievedby grouping adjacent 1's or 0's in groups of 2i, where i = 1, 2, …, n and n is the numberof 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.

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

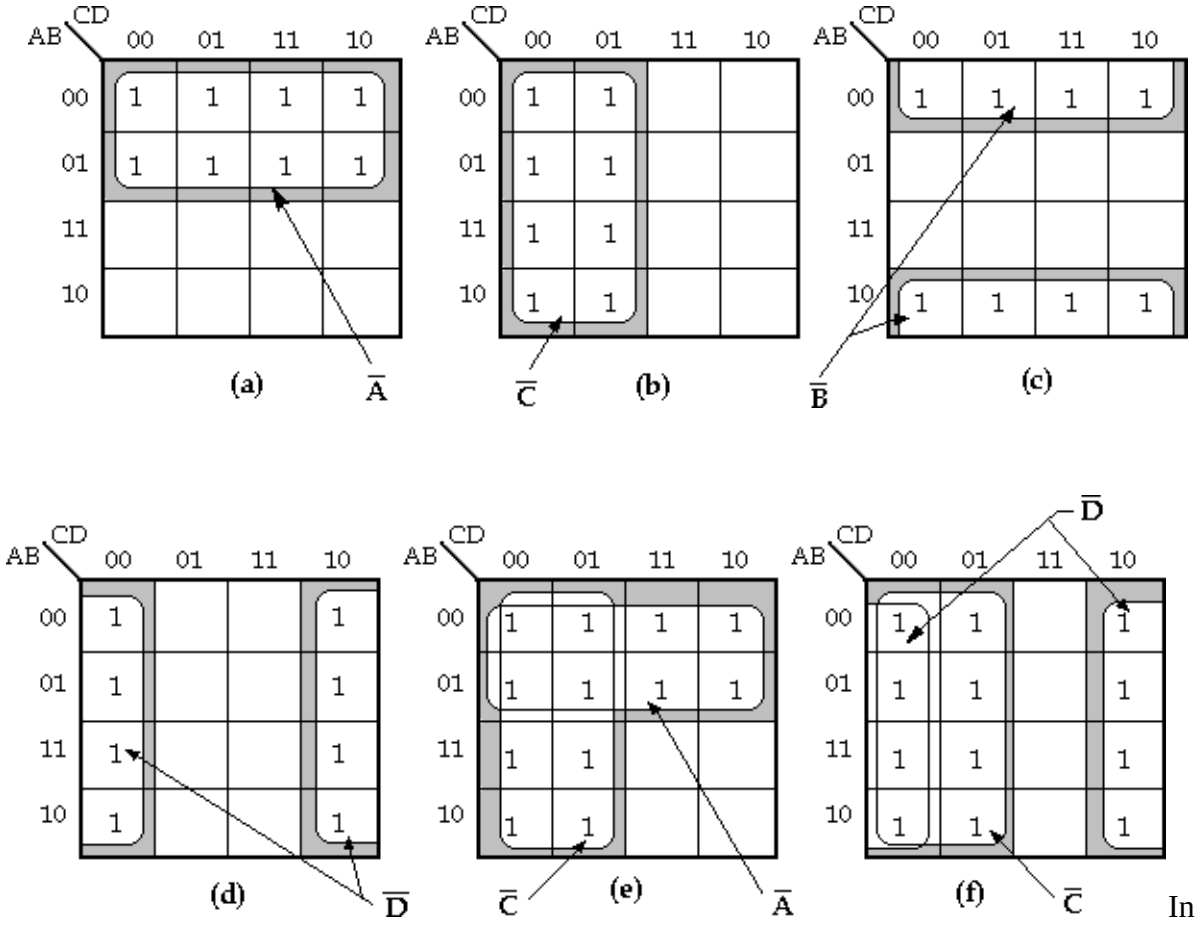
Grouping Four Adjacent 1's: (Quad)

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 verticallyadjacent. 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:

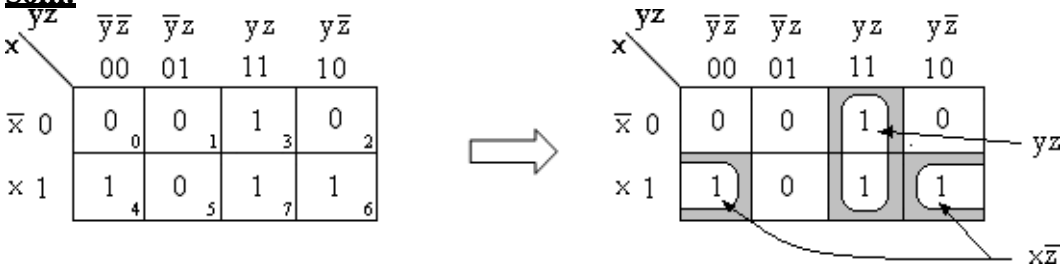
1) Plot the K-map and place 1's in those cells corresponding to the 1's in the sum of product expression. Place 0's in the other cells.
2) Check the K-map for adjacent 1's and encircle those 1's which are not adjacent to any other 1's. These are called **isolated 1's**.
3) Check for those 1's which are 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.
4) Combine any pairs necessary to include any 1's that have not yet been grouped.
5) Form the simplified expression by summing product terms of all the groups.

Three- Variable Map:

1. Simplify the Boolean expression,

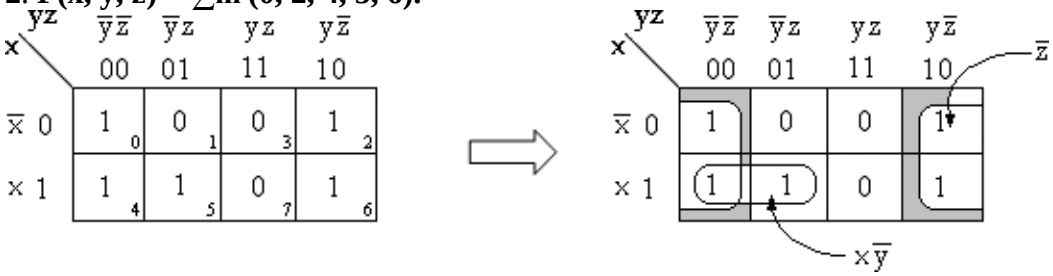
F(x, y, z) = ∑m (3, 4, 6, 7).

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:

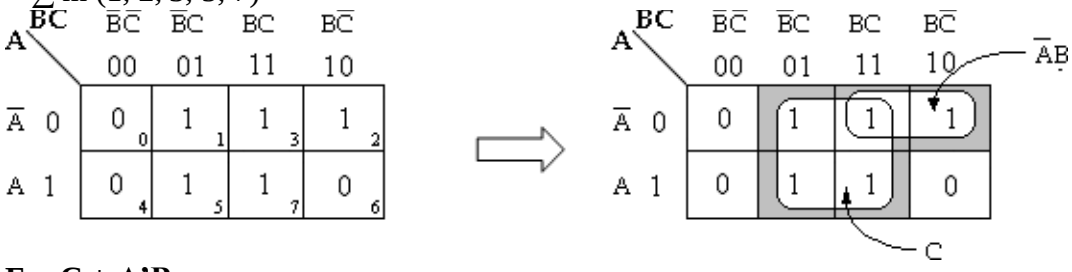
= A'C (B+ B') + A'B (C+ C') + AB'C + BC (A+ A')

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

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

= m3+ m1+ m2+ m5+ m7

= ∑ m (1, 2, 3, 5, 7)



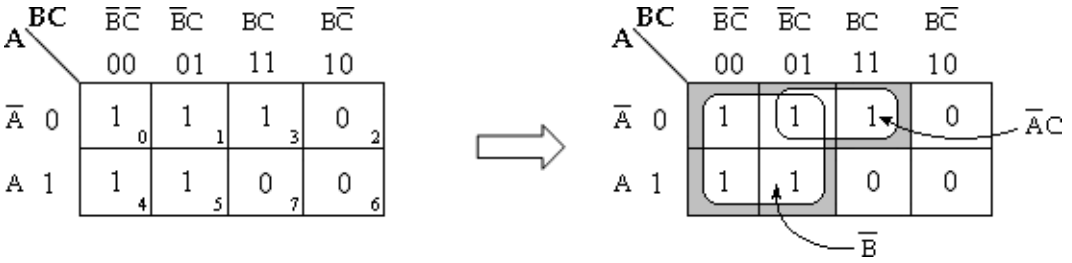
F = C + A'B

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)

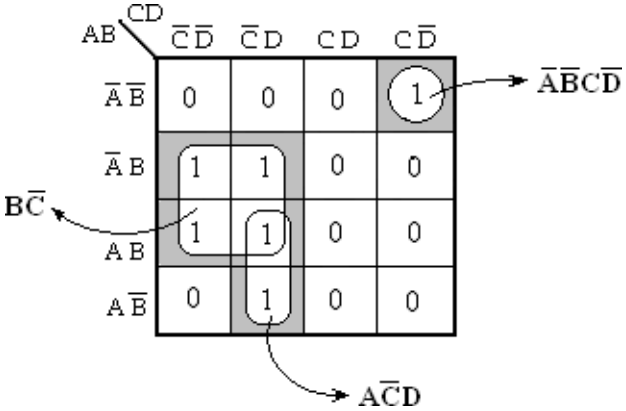


F = A'C + B'

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'

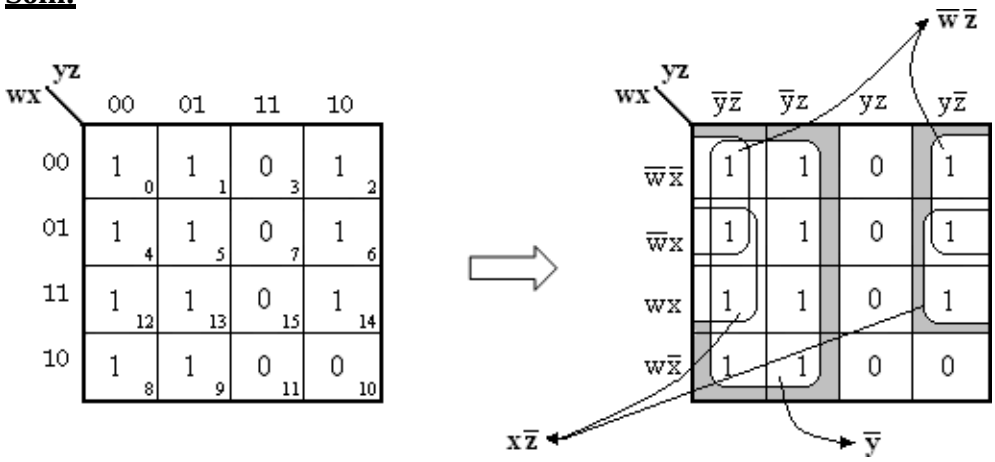


Soln:

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,

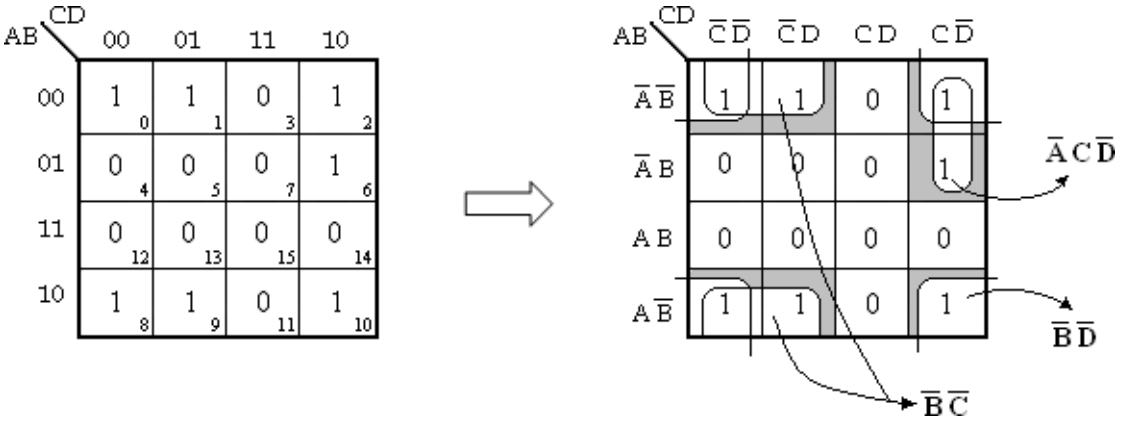
F= y'+ w'z'+ xz'

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'

= m1+ m0+ m10+ m2+ m6+ m9+ m8



= ∑ **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')

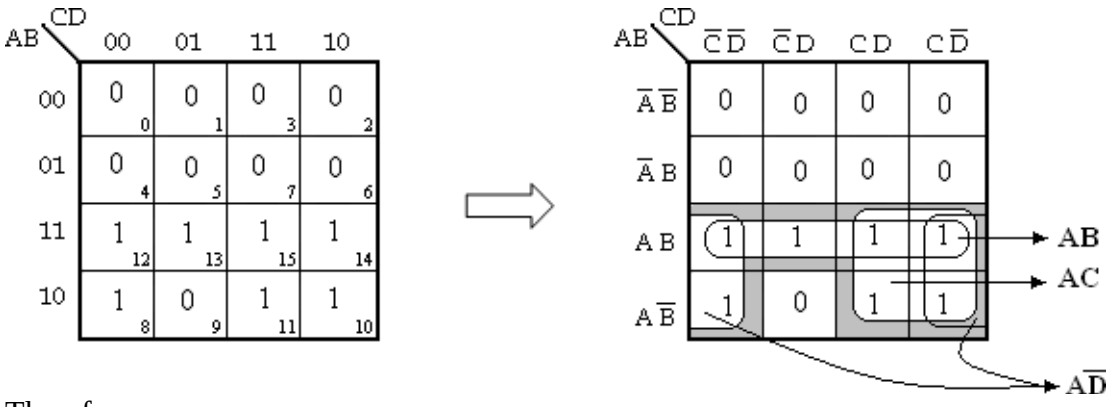
= ABCD+ AB'C'D'+ AB'CD+ AB'CD'+ (ABC+ ABC') (D+ D')

= 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'

= m15+ m8+ m11+ m10+ m14+ m13+ m12

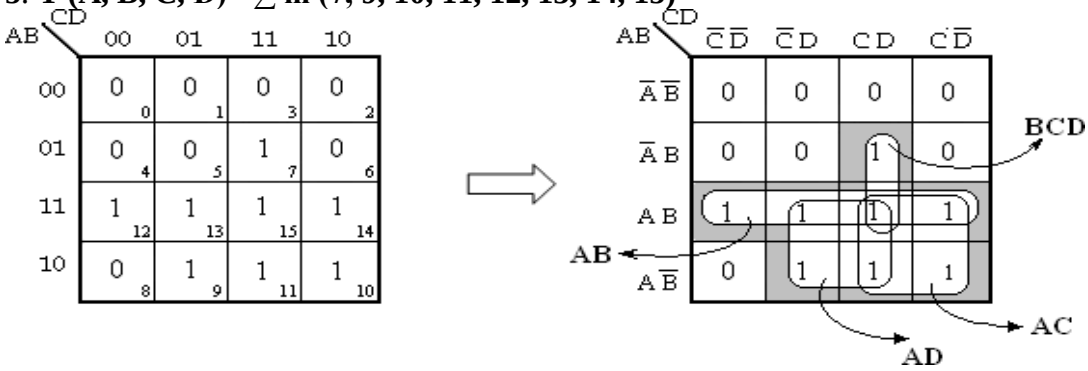
= ∑ **m (8, 10, 11, 12, 13, 14, 15)**



Therefore,

Y= AB+ AC+ AD'.

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

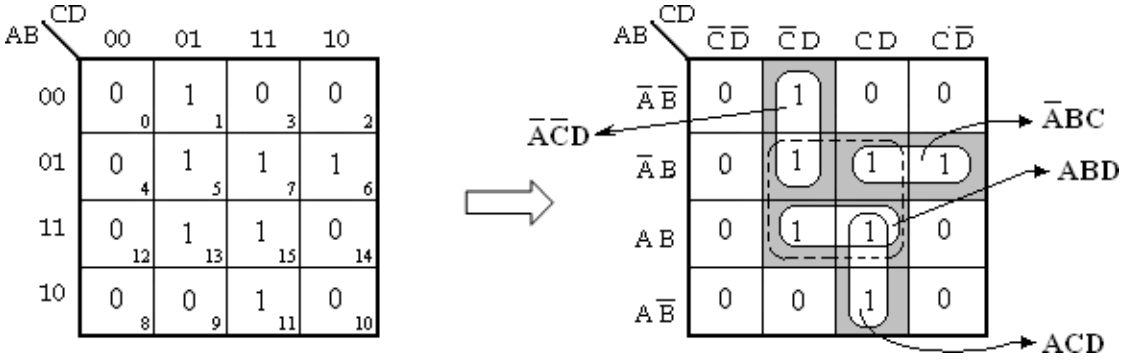


Therefore, **Y= AB+ AC+ AD+BCD.**

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

= m1+ m5+ m7+ m6+ m13+ m15+ m11

= ∑ m (1, 5, 6, 7, 11, 13, 15)

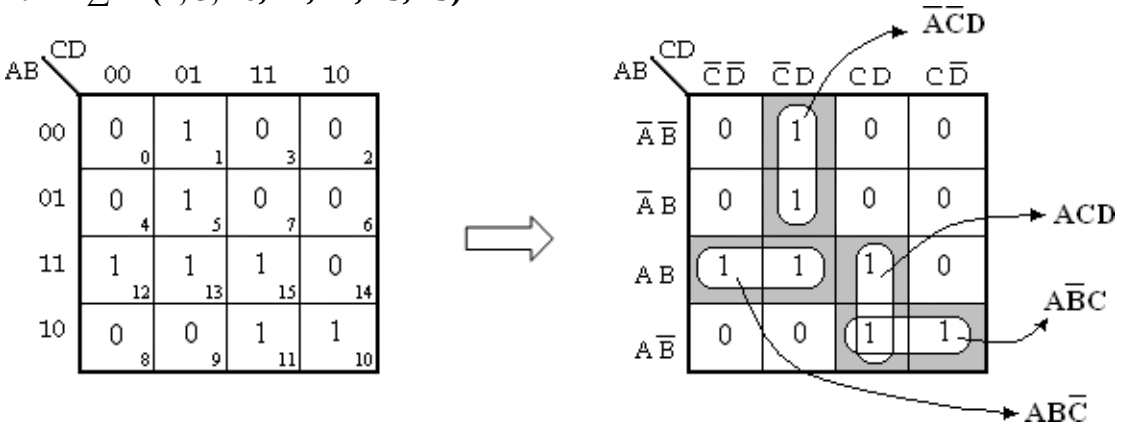


In the above K-map, the cells 5, 7, 13 and 15 can be grouped to form a quad as indicated by the dotted lines. In order to group the remaining 1's, four pairs have to be formed. However, all the four 1's covered by the quad are also covered by the pairs. So, the quad in the above k-map is redundant.

Therefore, the simplified expression will be,

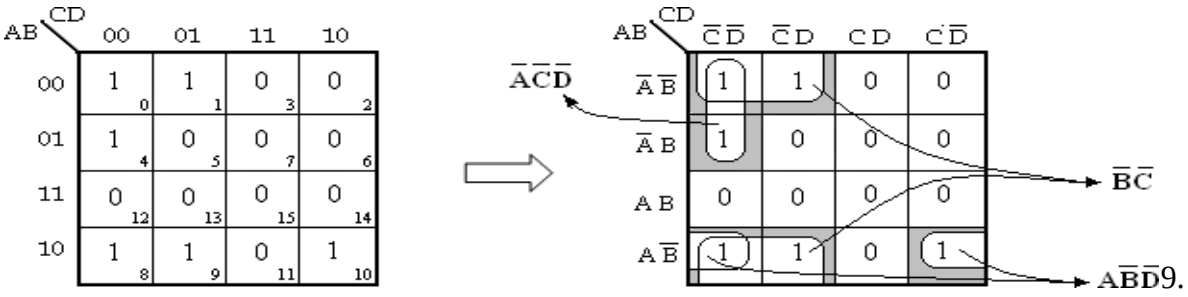
Y = A'C'D+ A'BC+ ABD+ ACD.

7. Y= ∑ m (1, 5, 10, 11, 12, 13, 15)



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

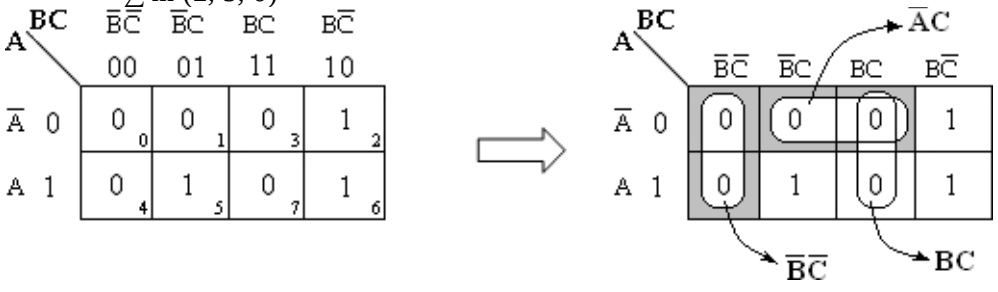
8.F (A, B, C, D) = ∑ m (0, 1, 4, 8, 9, 10)



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

Simplification of Sum of Products Expressions: (Minimal Sums)

1. **Y= (A+ B+ C') (A+ B'+ C') (A'+ B'+ C') (A'+ B+ C) (A+ B+ C)**
   = M1. M3. M7. M4. M0
   =∏ M (0, 1, 3, 4, 7)
   = ∑ m (2, 5, 6)



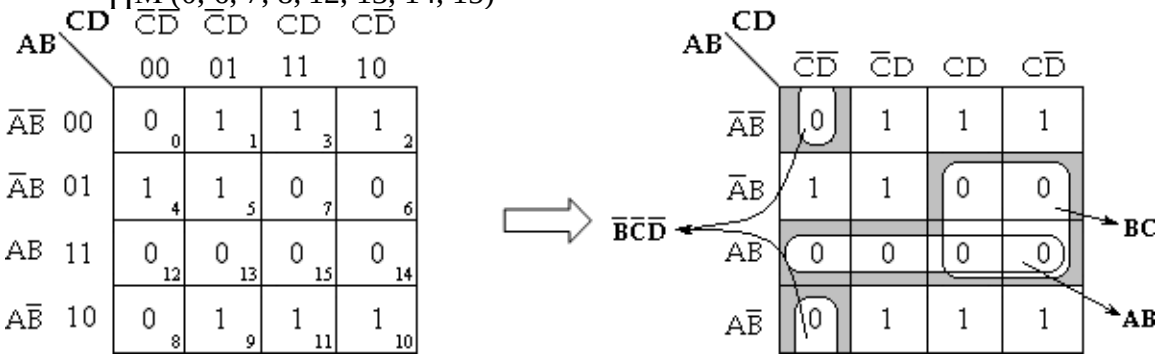
Y' = B'C'+ A'C+ BC.

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')**

= M12. M14. M15. M8. M6. M7. M0. M13

= ∏M (0, 6, 7, 8, 12, 13, 14, 15)



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

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

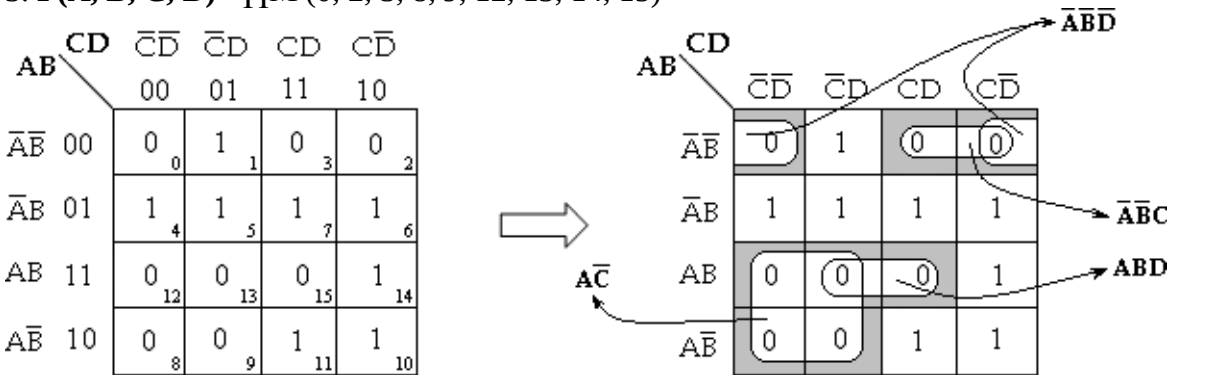
= (B'C'D')'. (AB)'. (BC)'

= (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')'

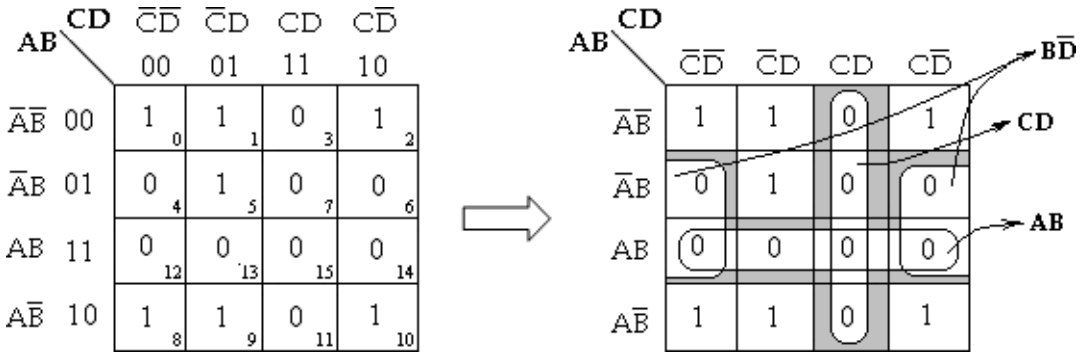
= (A'B'D')'. (A'B'C)'. (ABD)'. (AC')'

= (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)



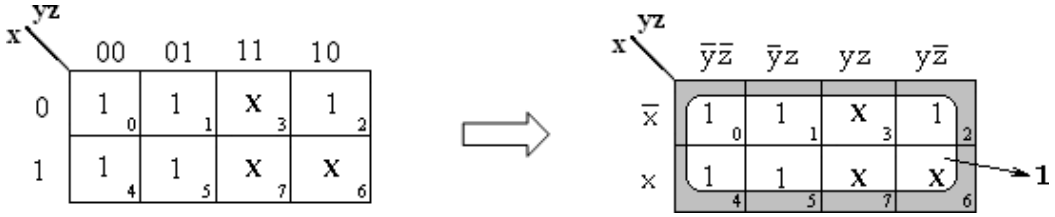
Y' = BD'+ CD+ AB

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')**

3.3 Don't care Conditions:

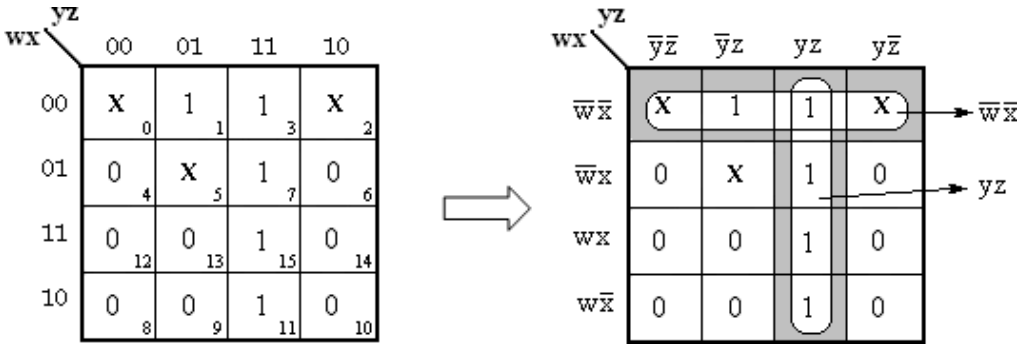
A don't care minterm is a combination of variables whose logical value is not specified. When choosing adjacent squares to simplify the function in a map, the don't care minterms may be assumed to be either 0 or 1. When simplifying the function, we can choose to include each don't care minterm with either 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)



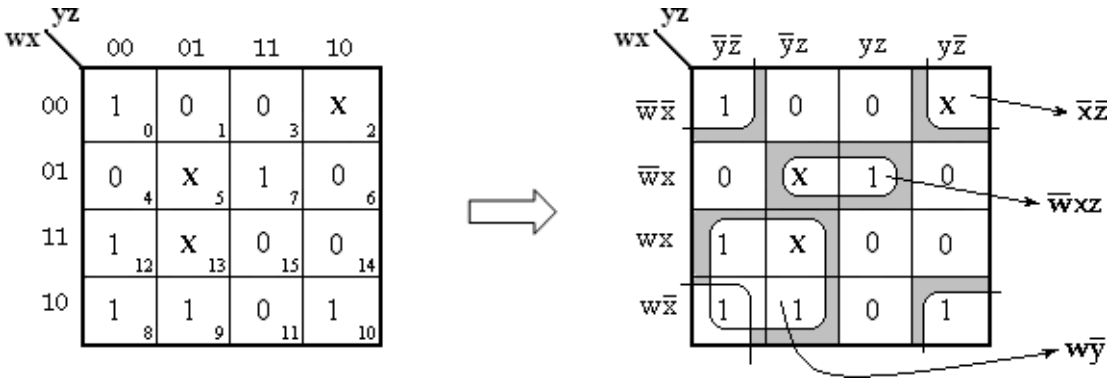
F (x, y, z) = 1

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



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

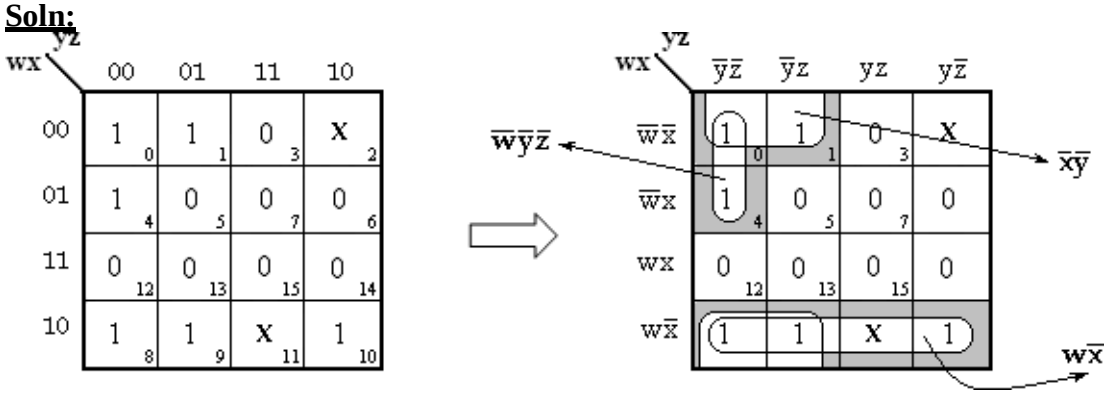
3. F (w, x, y, z) = ∑m (0, 7, 8, 9, 10, 12)+ ∑d (2, 5, 13)



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)

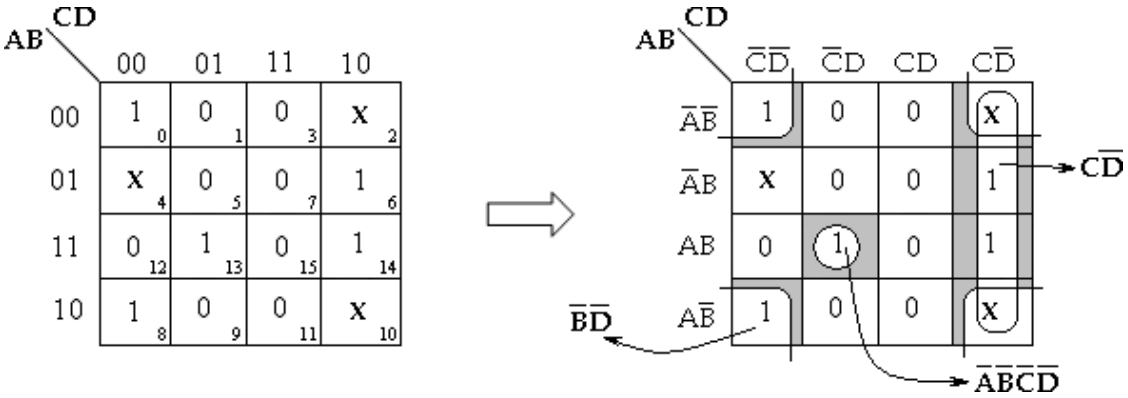
Soln:



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:

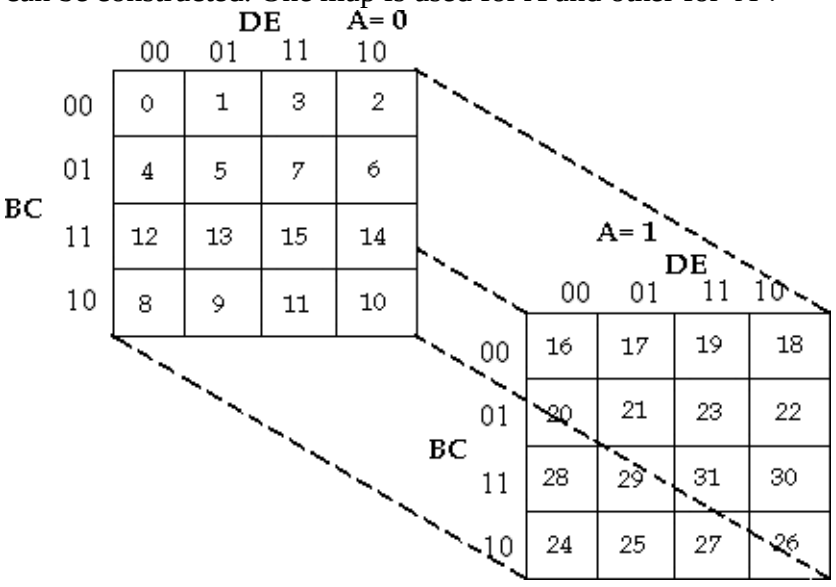


F(A, B, C, D) = CD'+ B'D'+ A'B'C'D'.

3.4 Five- Variable Maps:

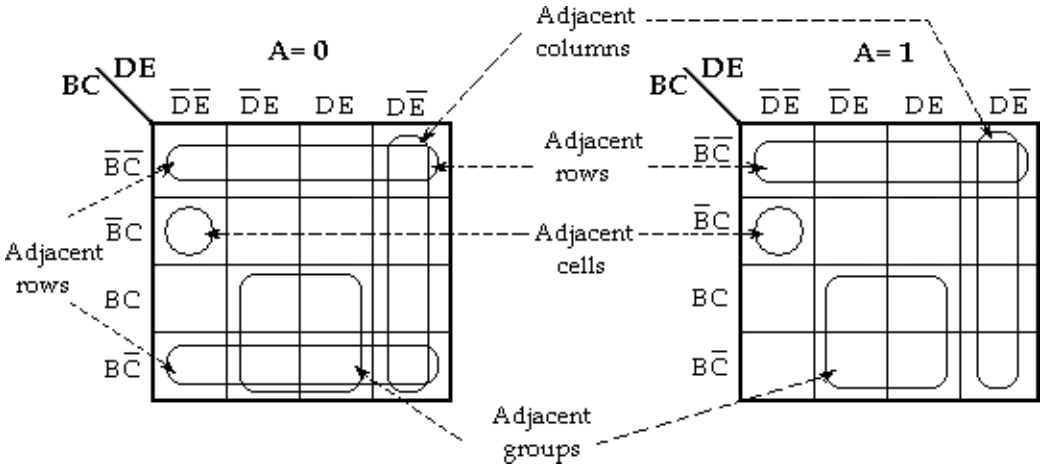
A 5- variable K- map requires 25= 32 cells, but adjacent cells are difficult to identifyon a single 32-cell map. Therefore, two 16 cell K-maps are used.

If the variables are A, B, C, D and E, two identical 16- cell maps containing B, C, D and E can be constructed. One map is used for A and other for A'.



Five- Variable Karnaugh map (Layer Structure)

In order to identify the adjacent grouping in the 5- variable map, we must imagine the two maps superimposed on one another ie., every cell in one map is adjacent to the corresponding cell in the other map, because only one variable changes between such corresponding cells.Thus, every row on one map is adjacent to the corresponding row (the one occupying thesame position) on the other map, as are corresponding columns. Also,



the rightmost and leftmost columns within each 16- cell map are adjacent, just as they are in any16- cell map, as are the top and bottom rows.

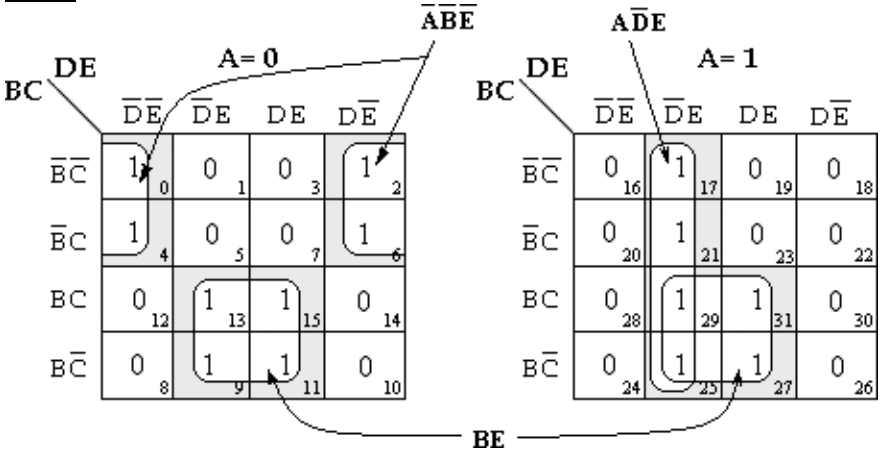
Typical sub cubes on a five-variable map

However, the rightmost column of the map is not adjacent to the leftmost column of the othermap.

1. Simplify the Boolean function

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

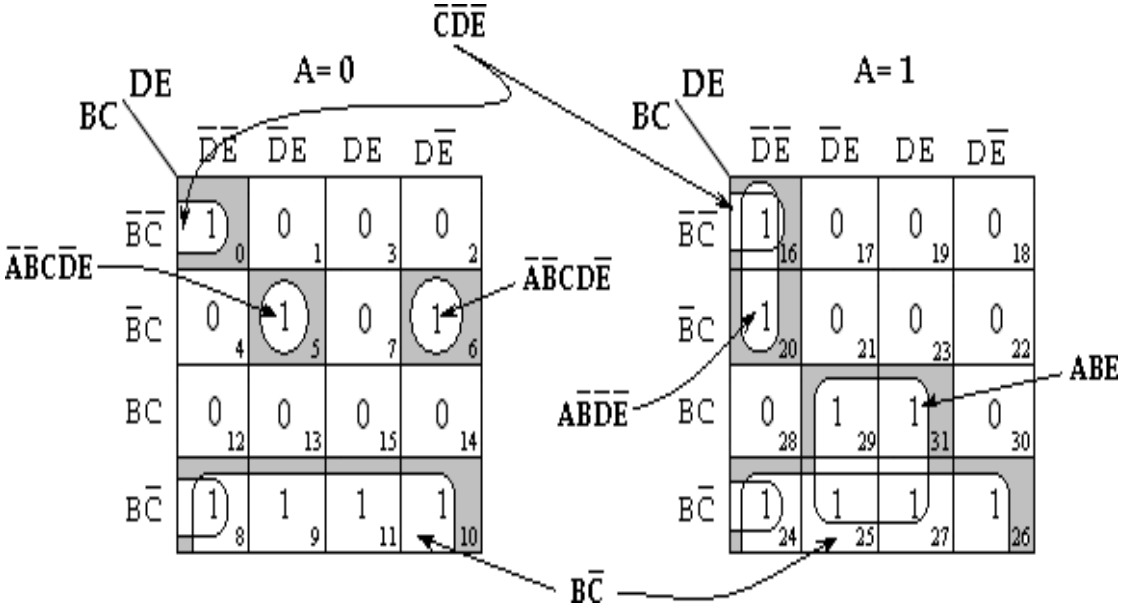
Soln:



F (A, B, C, D, E) = A'B'E'+ BE+ AD'E

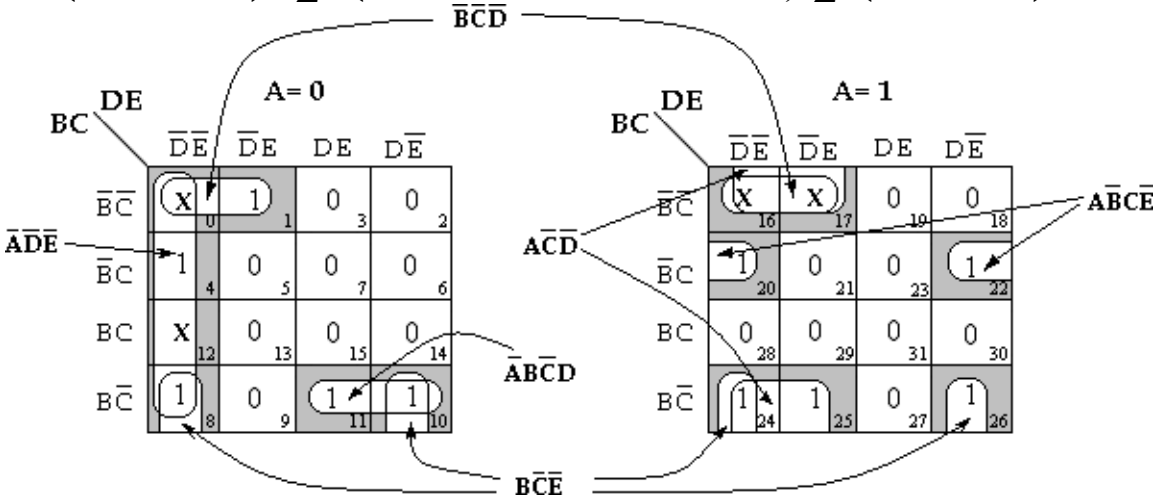
2. F (A, B, C, D, E) = ∑m (0, 5, 6, 8, 9, 10, 11, 16, 20, 24, 25, 26, 27, 29, 31)

Soln:



F (A, B, C, D, E) = C'D'E'+ A'B'CD'E+ A'B'CDE'+ AB'D'E'+ ABE+ BC'

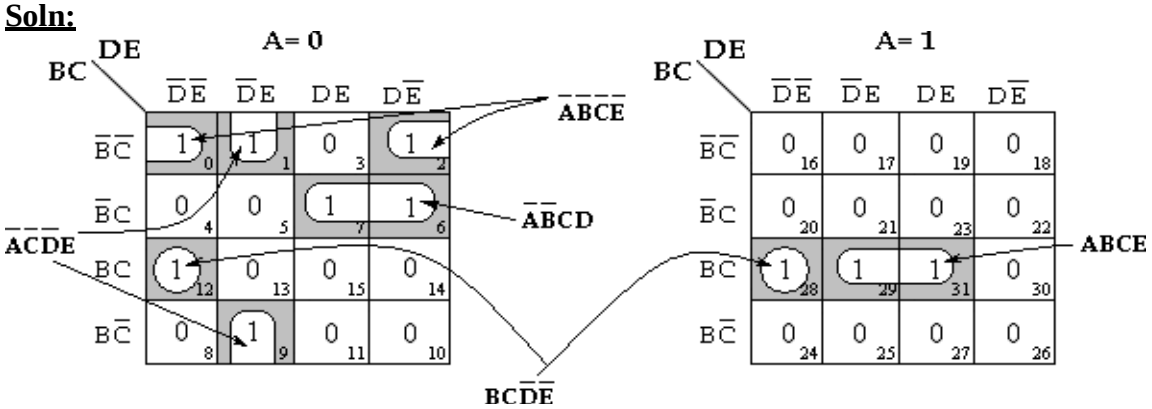
3. F (A, B, C, D, E) = ∑m (1, 4, 8, 10, 11, 20, 22, 24, 25, 26)+∑d (0, 12, 16, 17)



F (A, B, C, D, E) = B'C'D'+ A'D'E'+ BC'E'+ A'BC'D+ AC'D'+ AB'CE'

4. F (A, B, C, D, E) = ∑m (0, 1, 2, 6, 7, 9, 12, 28, 29, 31)

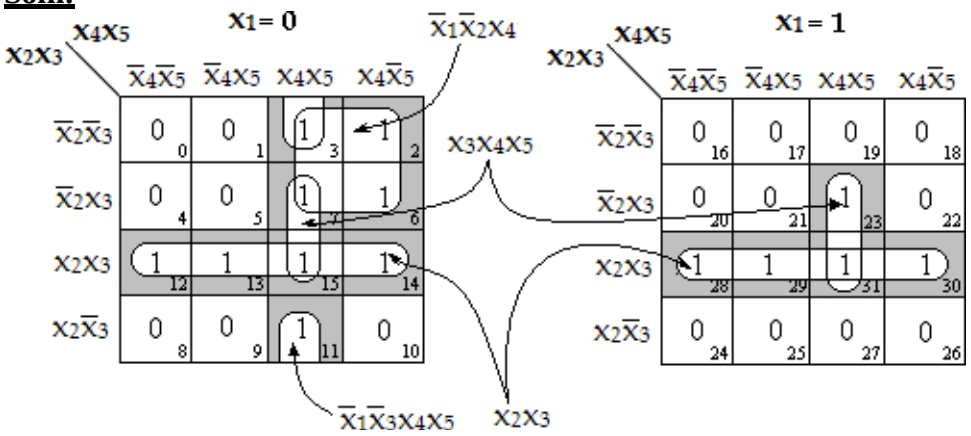
Soln:



F (A, B, C, D, E) = BCD'E'+ ABCE+ A'B'C'E'+ A'C'D'E+ A'B'CD

5. F (x_1, x_2, x_3, x_4, x_5) = ∑m (2, 3, 6, 7, 11, 12, 13, 14, 15, 23, 28, 29, 30, 31)

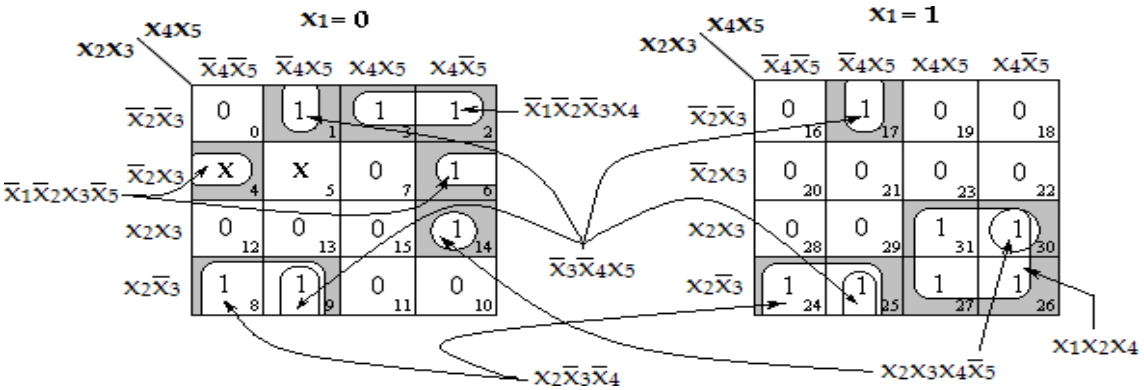
Soln:



F (x_1, x_2, x_3, x_4, x_5) = x_2x_3+ $x_3x_4x_5$+ $x_1'x_2'x_4$+ $x_1'x_3'x_4x_5$

6. F (x_1, x_2, x_3, x_4, x_5) = ∑m (1, 2, 3, 6, 8, 9, 14, 17, 24, 25, 26, 27, 30, 31)+ ∑d (4, 5)

Soln:



F (x_1, x_2, x_3, x_4, x_5) = $x_2x_3'x_4'$+ $x_2x_3x_4x_5'$+ $x_3'x_4'x5$+ $x_1x_2x_4$+ $x_1'x_2'x_3x_5'$+ $x_1'x_2'x_3'x_4$

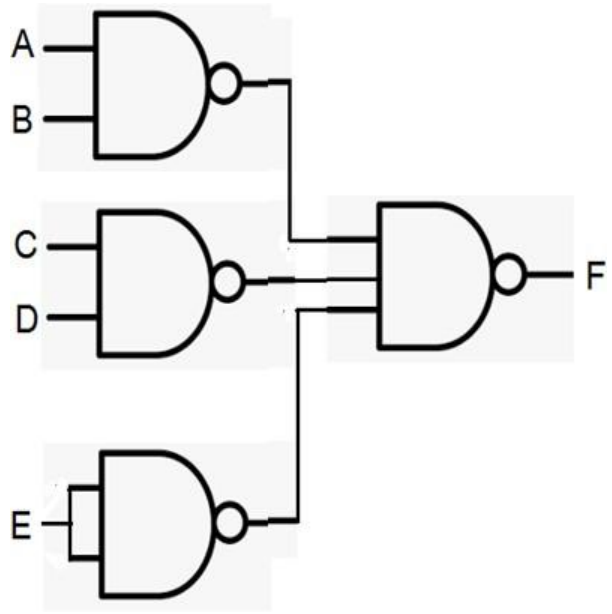
3.5 Two Level Gate Network

- 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

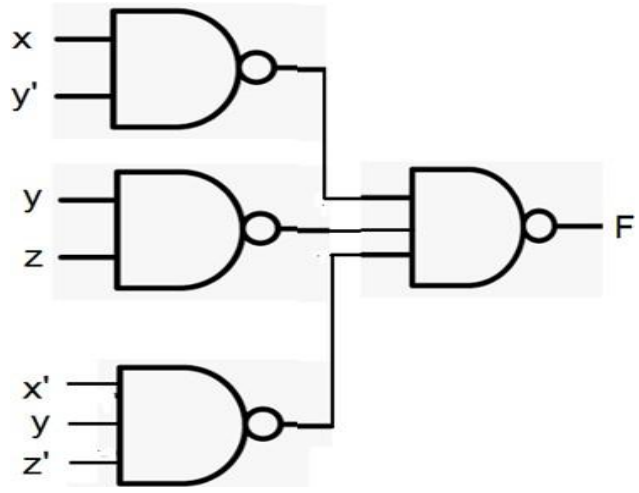
- The POS can be implemented using NOR – NOR logic
    1. Each sum term is connected to NOR gates in level 1
    2. One NOR is connected in the second level 2

Implement Using NAND – NAND logic

F=A.B+C.D+E

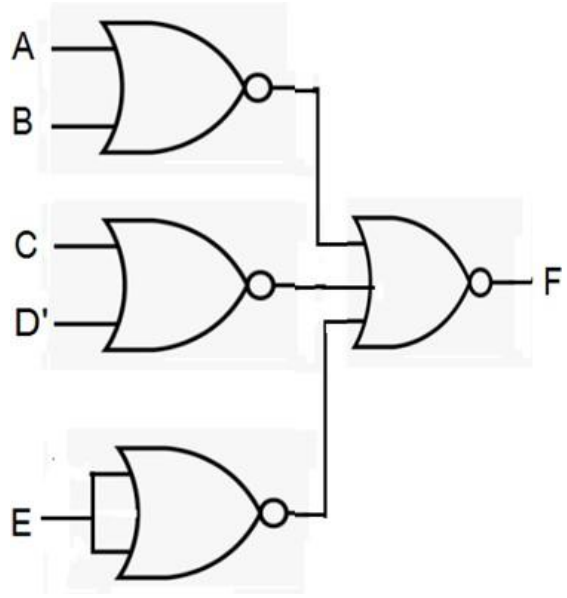


F=(A+B)(C+D')E

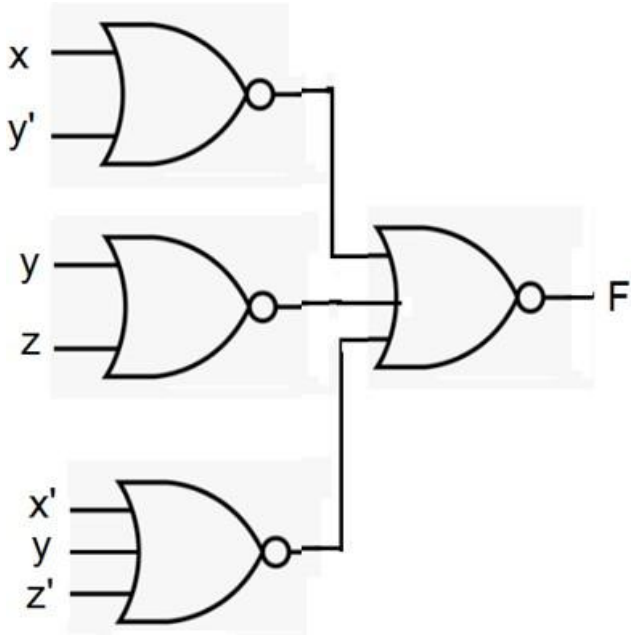


Implement Using NOR – NOR logic

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



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



CHAPTER 4
COMBINATIONAL CIRCUITS

4.1 Introduction

The digital system consists of two types of circuits, namely

- Combinational circuits
- Sequential circuits

Combinational circuit

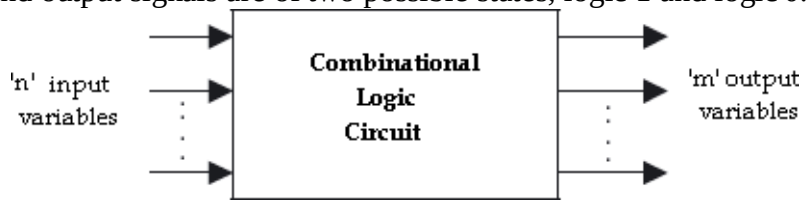
consists of logic gates whose output at any time is determined from the present combination of inputs. The logic gate is the most basic building block of combinational logic. The logical function performed by a combinational circuit is fullydefined by a set of Boolean expressions.

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. Thelogic 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 theinput 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.

4.2 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 usinglogic gates.

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

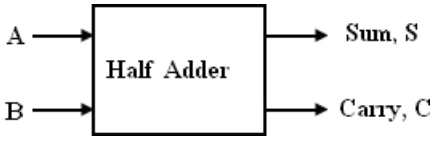
- The implementation should have the minimum number of gates, with the gates used having the minimum number of inputs.
- There should be a minimum number of interconnections.
- Limitation on the driving capability of the gates should not be ignored.

4.3 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 arethe 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.

4.3.1 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.



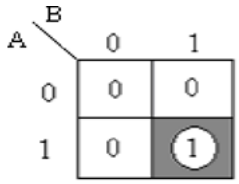
Block schematic of half-adder

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

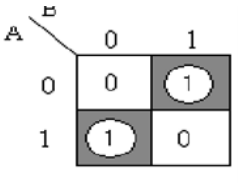
Truth table of half-adder

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**:**



Carry = A .B



Sum = AB'+ A'B
= A⊕B

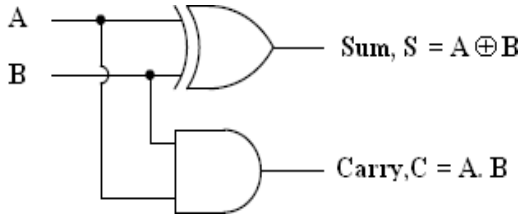
The Boolean expressions for the SUM and CARRY outputs are given by the equations,

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.

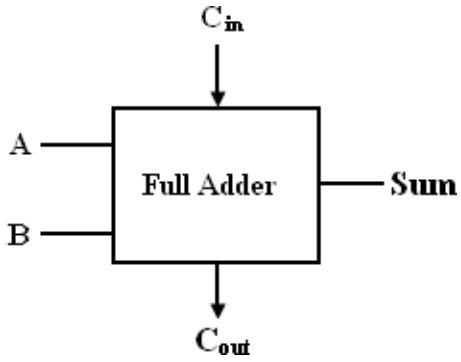
The logic diagram of the half adder is,



Logic Implementation of Half-adder

4.3.2 Full-Adder:

A full adder is a combinational circuit that forms the arithmetic sum of three input bits. Itconsists 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 adderis given by,



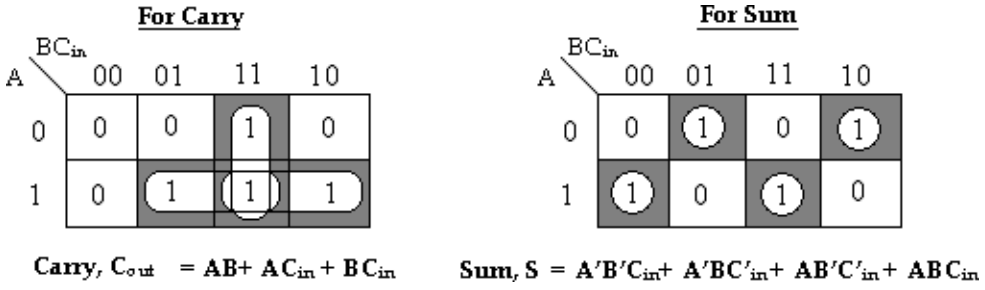
Block schematic of full-adder

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**	**Cin**	**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,



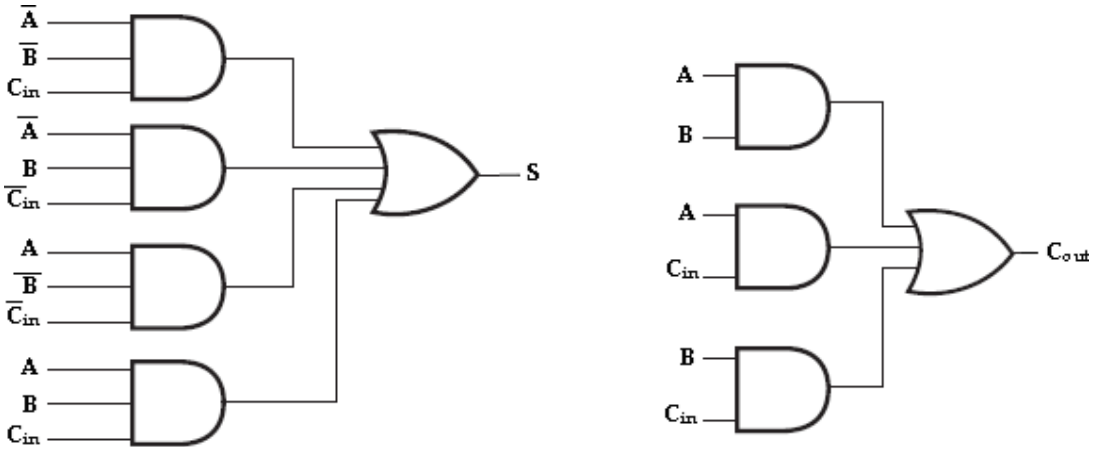
Carry, C_{out} = AB+ AC_{in} + BC_{in} **Sum, S = A'B'C_{in}+ A'BC'$_{in}$+ AB'C'$_{in}$+ ABC_{in}**

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 Cin and the output of the first half-adder, giving

$\mathbf{Sum = C_{in} \oplus (A \oplus B)}$ $[x \oplus y = x'y + xy']$

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

= C‘in (A‘B+AB‘) + Cin (A‘B+AB‘)‘ [(x‘y+xy‘)‘= (xy+x‘y‘)]

= C‘in (A‘B+AB‘) + Cin (AB+A‘B‘)

= A‘BC‘in + AB‘C‘in + ABCin + A‘B‘Cin .

and the carry output is,

Carry, C_{out} = AB+ C_{in} (A’B+AB’)

= AB+ A‘BCin+ AB‘Cin

= AB (Cin+1) + A‘BCin+ AB‘Cin [Cin+1= 1]

= ABCin+ AB+ A‘BCin+ AB‘Cin

= AB+ ACin (B+B‘) + A‘BCin

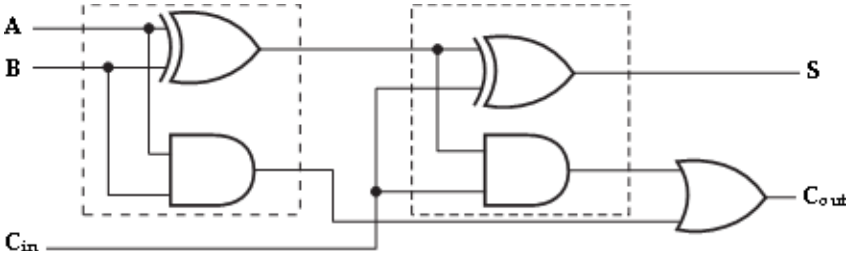
= AB+ ACin+ A‘BCin

= AB (Cin+1) + ACin+ A‘BCin [Cin+1= 1]

= ABCin+ AB+ ACin+ A‘BCin

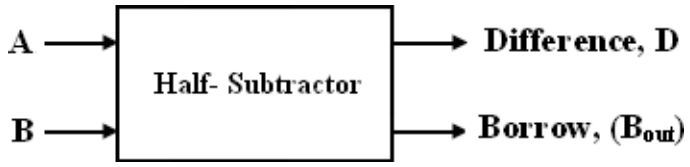
= AB+ ACin+ BCin (A +A‘)

= AB+ ACin+ BCin.



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

4.3.3 Half -Subtractor:



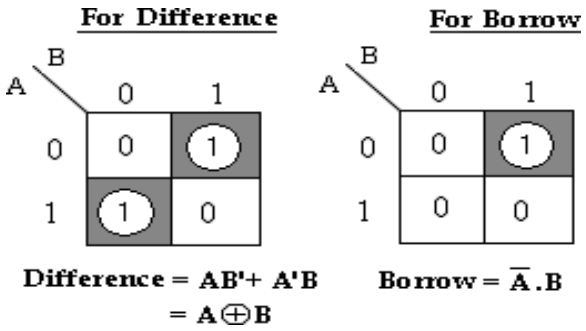
Block schematic of half-subtractor

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 thecorresponding 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:

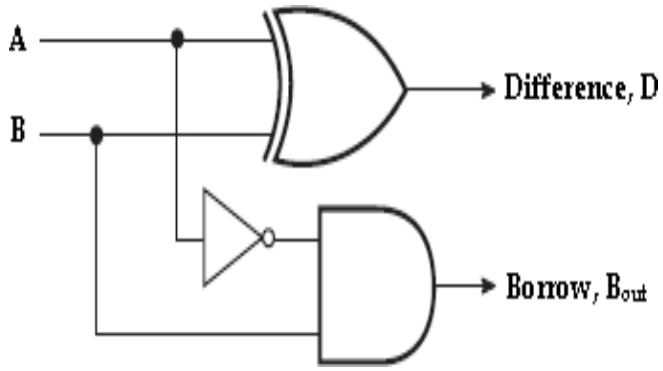


The Boolean expressions for the DIFFERENCE and BORROW outputs are given by the equations,

$$\text{Difference, } D = A'B + AB'$$

$$\text{Borrow, } B_{out} = A' \cdot B$$

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 the gate. The logic diagram of the half adder is,



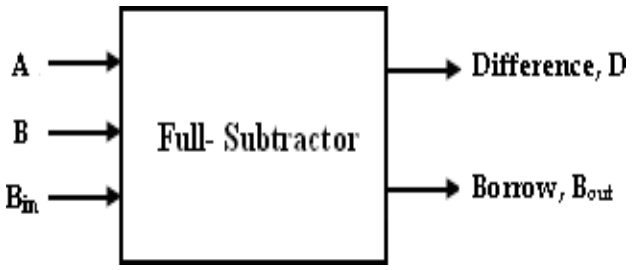
Logic Implementation of Half-Subtractor

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.

4.3.4 Full Subtractor:

A full subtractor performs subtraction operation on two bits, a minuend and a subtrahend, andalso takes into consideration whether a _1' has already been borrowed by the previous adjacentlower 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 Bin. 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.

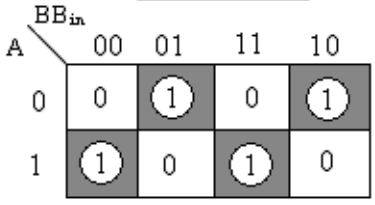


Block schematic of full-adder

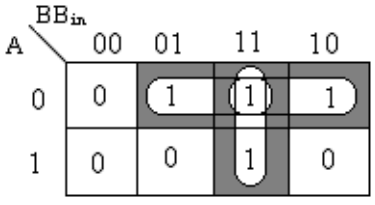
The truth table for full-subtractor is,

Inputs			Outputs	
A	**B**	**Bin**	**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



For Borrow



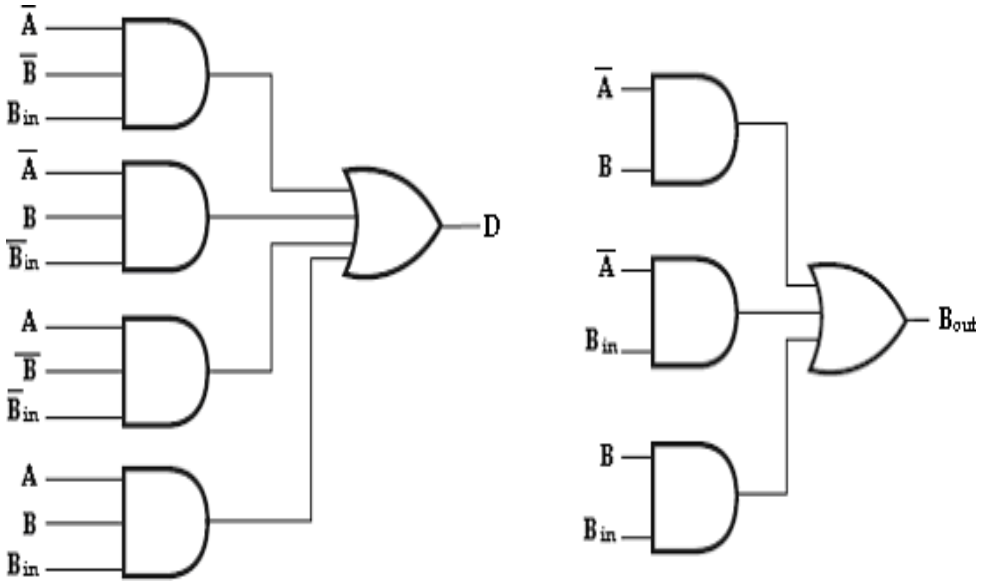
Difference, $D = A'B'B_{in} + A'BB'_{in} + AB'B'_{in} + ABB_{in}$

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-subtractorsand one OR gate. The difference, D output from the second half subtractor is the Ex -OR ofB_{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‘) + Bin (A‘B+AB‘)‘ [(x‘y+xy‘)‘= (xy+x‘y‘)]

= B‘in (A‘B+AB‘) + Bin (AB+A‘B‘)

= A‘BB‘in + AB‘B‘in + ABBin + 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+ Bin (AB+A‘B‘)

= A‘B+ ABBin+ A‘B‘Bin

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

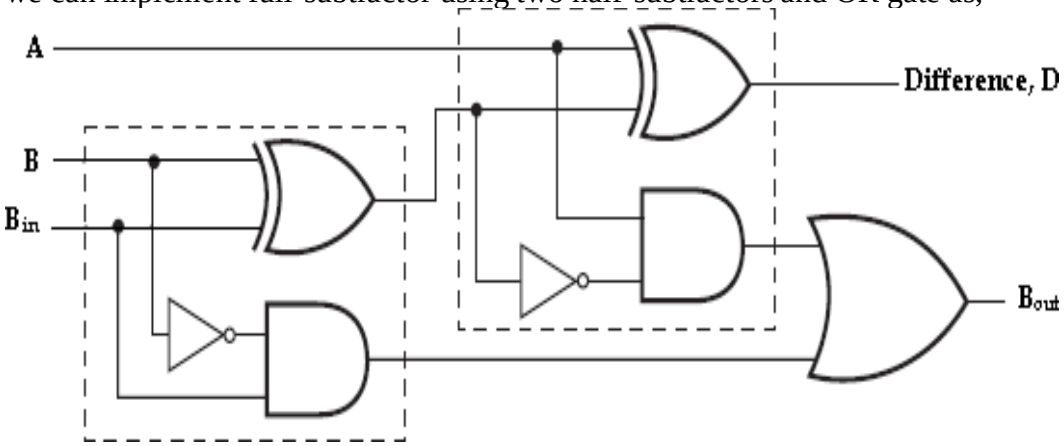
= A‘BBin+ A‘B+ BBin+ A‘B‘Bin

= 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

4.4 Binary Adder (Parallel Adder):

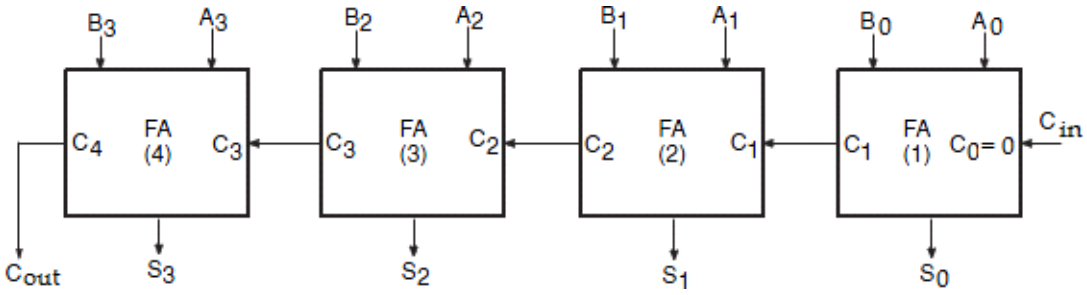


Fig. 4-bit binary parallel Adder

The 4-bit binary adder using full adder circuits is capable of adding two 4-bitnumbers 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, A3A2A1A0= 1111 and B3B2B1B0= 0011.

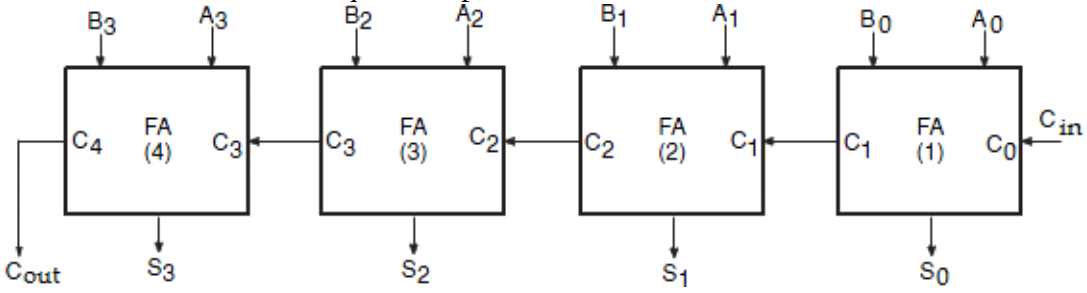
The bits are added with full adders, starting from the least significant position, to form thesum it and carry bit. The input carry C0 in the least significant position must be

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.

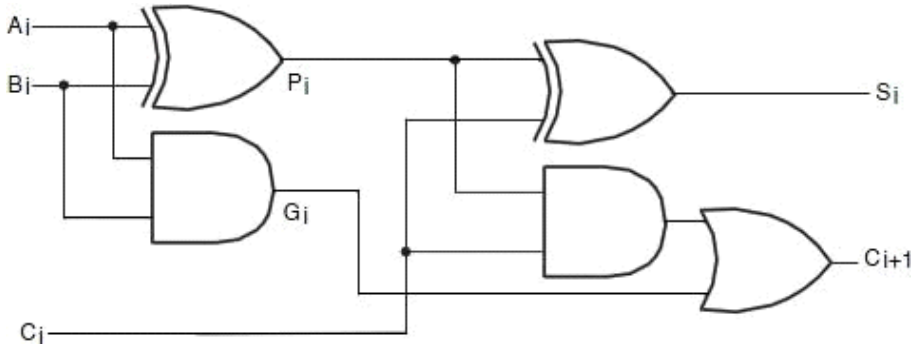
In the least significant stage, A0, B0 and C0 (which is 0) are added resulting in sum S0 and carry C1. This carry C1 becomes the carry input to the second stage. Similarly in the second stage, A1, B1 and C1 are added resulting in sum S1 and carry C2, in the third stage, A2, B2 and C2 are added resulting in sum S2 and carry C3, in the third stage, A3, B3 and C3 are added resulting in sum S3 and C4, which is the output carry. Thus the circuit results in a sum (S3S2S1S0) and a carry output (Cout).Though the parallel binary adder is said to generate its output immediately after the inputs areapplied, its speed of operation is limited by the carry propagation delay through all stages. However, there are several methods to reduce this delay. One of the methods of speeding up this process is look-ahead carry addition which eliminates the ripple-carry delay.

4.5 Carry Propagation–Look-Ahead Carry Generator:

In Parallel adder, all the bits of the augend and the addend are available for computation at thesame time. The carry output of each full-adder stage is connected to the carry input of the nexthigh-order stage. Since each bit of the sum output depends on the value of the input carry, time delay occurs in the addition process. This time delay is called as carry propagation delay.For example, addition of two numbers (0011+ 0101) gives the result as 1000. Addition of the LSB position produces a carry into the second position. This carry when added to the bits of thesecond position, produces a carry into the third position. This carry when added to bits of the third position, produces a carry into the last position. The sum bit generated in the lastposition (MSB) depends on the carry that was generated by the addition in the previous position. i.e., the adder will not produce correct result until LSB carry has propagated through the intermediate full-adders. This represents a time delay that depends on the propagation delay produced in an each full-adder. For example, if each full adder is considered to have a propagation delay of 30nsec, then S3 will not react its correct value until 90 nsec after LSB is generated. Therefore total time required to perform addition is 90+ 30 = 120nsec.



4-bit Parallel Adder



Full-Adder circuit

The method of speeding up this process by eliminating inter stage carry delay is called look ahead-carry addition. This method utilizes logic gates to look at the lower order bits of the augend and addend to see if a higher-order carry is to be generated. It uses two functions: carry generate and carry propagate.

Consider the circuit of the full-adder shown above. Here we define two functions:
carry generate (Gi) and carry propagate (Pi) as,

Carry generate, $\mathbf{G_i = A_i \oplus B_i}$

Carry propagate, $\mathbf{P_i = A_i \oplus B_i}$

the output sum and carry can be expressed as,

$$S_i = P_i \oplus C_i$$

$$C_{i+1} = G_i \oplus P_i C_i$$

Gi (carry generate), it produces a carry 1 when both Ai and Bi are 1, regardless of the input carry Ci. Pi (carry propagate) because it is the term associated with the propagation of the carry from C_i to C_{i+1}.

The Boolean functions for the carry outputs of each stage and substitute for each Ci its value from the previous equation:

C0= input

carry C1= G0 + P0C0

C2= G1 + P1C1 = G1 + P1 (G0 + P0C0) = G1 + P1G0 + P1P0C0

C3= G2 + P2C2 = G2 + P2 (G1 + P1G0 + P1P0C0) = G2 + P2G1 + P2P1G0 + P2P1P0C0

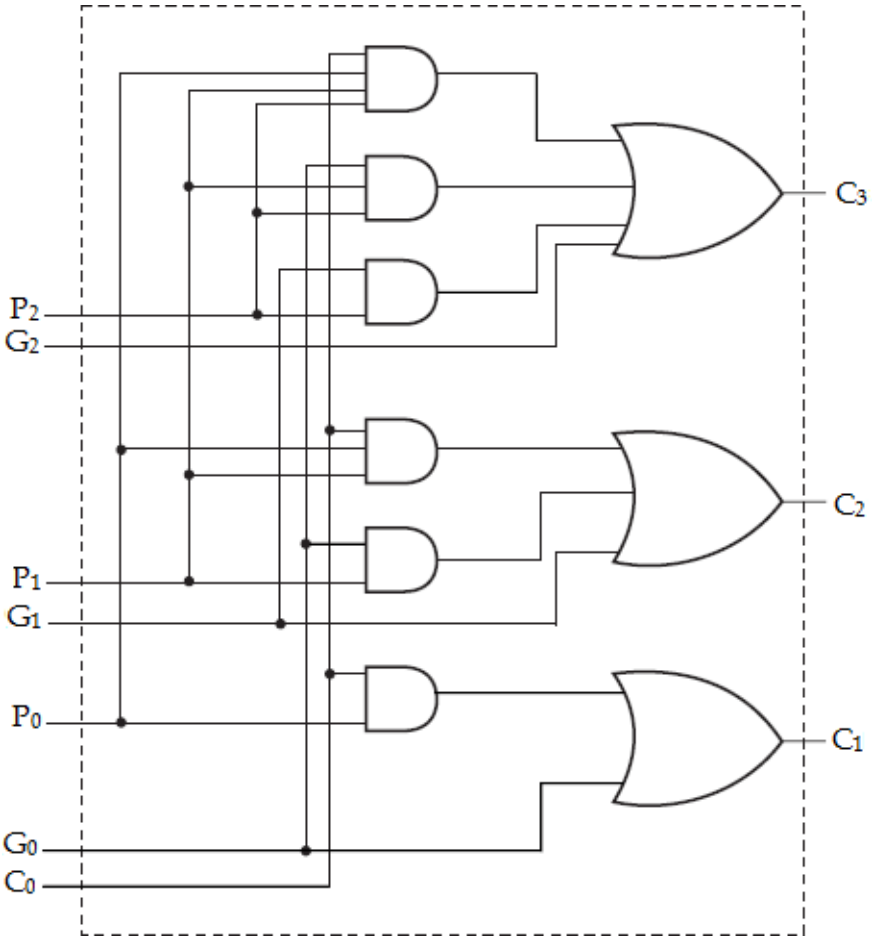


Fig. Logic diagram of Carry Look-ahead Generator

Since the Boolean function for each output carry is expressed in sum of products, each function can be implemented with one level of AND gates followed by an OR gate. The three Boolean functions for C1, C2 and C3 are implemented in the carry look-ahead generator as shown below.

Note that C3 does not have to wait for C2 and C1 to propagate; in fact C3 is propagated at the same time as C1 and C2.Using a Look-ahead Generator we can easily construct a 4-bit parallel adder with a Look- ahead carry scheme. Each sum output requires two exclusive-OR gates. The output of the first exclusive-OR gate generates the Pi variable, and the AND gate generates the Gi variable. The carries are propagated through the carry look-ahead generator andapplied as inputs to the second exclusive-OR gate. All output carries are generated after a delay through two levels of gates. Thus, outputs S1 through S3 have equal propagation delay times.

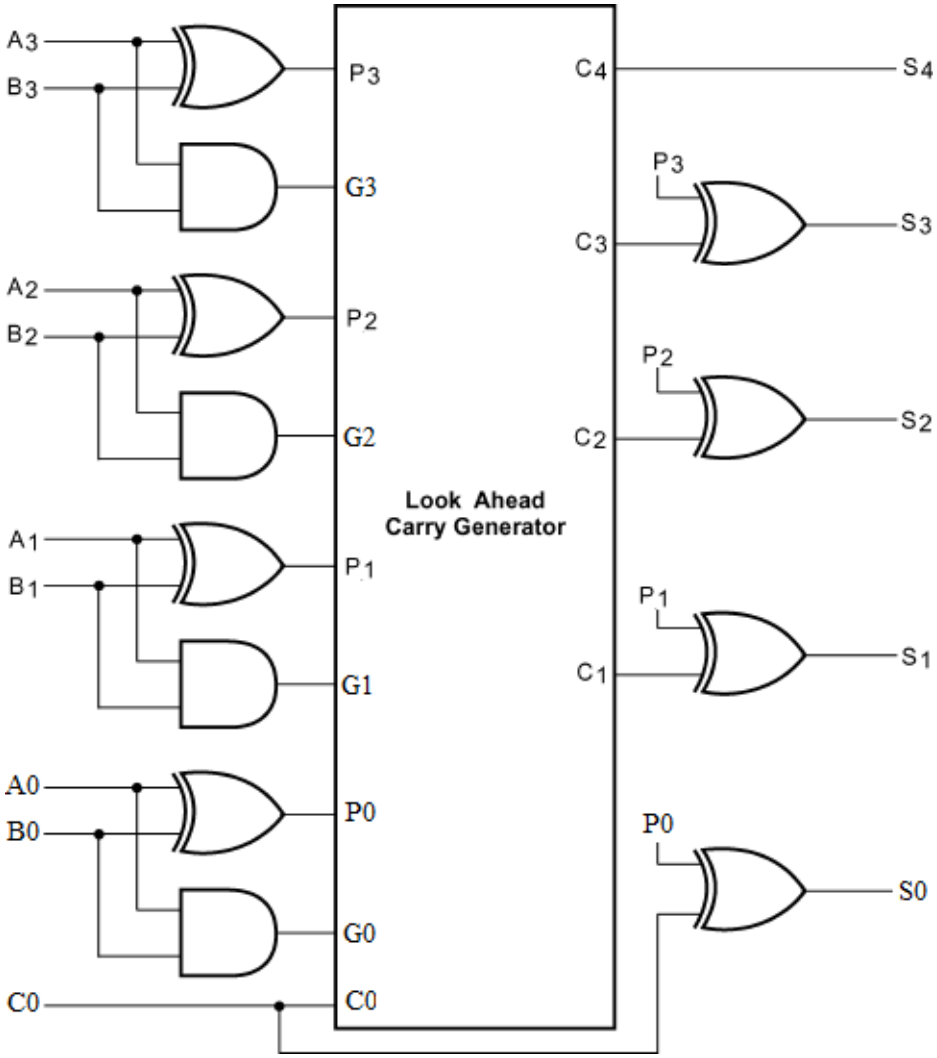


Fig. 4-Bit Adder with Carry Look-ahead

4.6 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 C0 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.

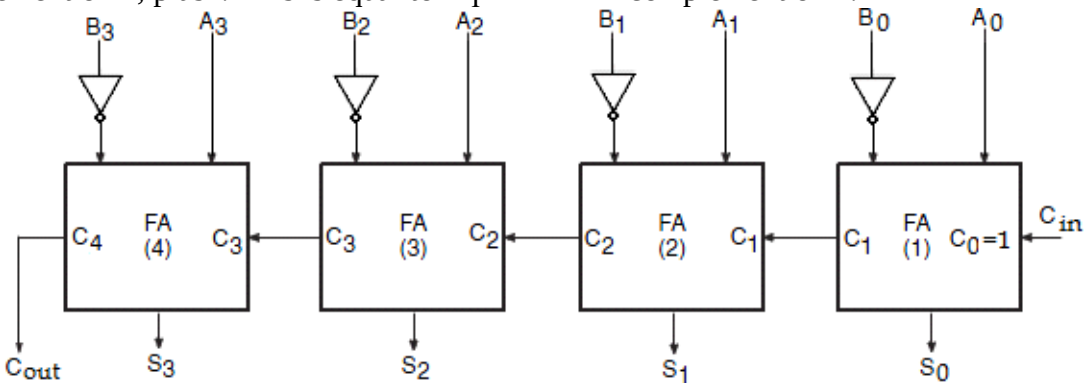


Fig. 4-bit Parallel Subtractor

4.7 Parallel Adder/ Subtractor:

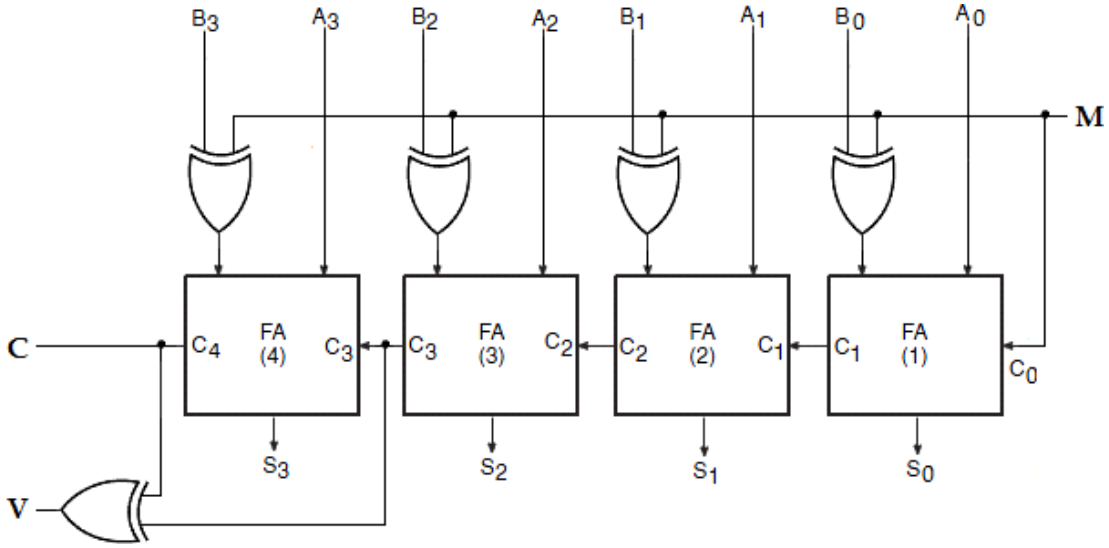


Fig. 4-Bit Adder Subtractor

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.

4.8 Decimal Adder (BCD Adder):

The digital system handles the decimal number in the form of binary coded decimal numbers (BCD). A BCD adder is a circuit that adds two BCD bits and produces a sum digit also in BCD.

Consider the arithmetic addition of two decimal digits in BCD, together with an input carry from a previous stage. Since each input digit does not exceed 9, the output sum cannot be greater than 9+ 9+1 = 19; the 1 is the sum being an input carry. The adder will form the sum in binary and produce a result that ranges from 0 through 19.

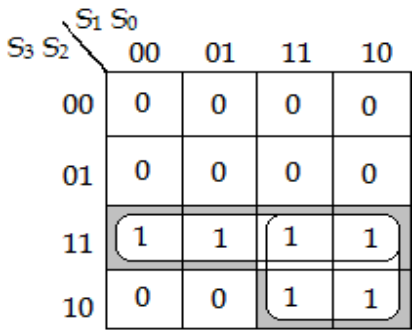
These binary numbers are labeled by symbols K, Z8, Z4, Z2, Z1, K is the carry. The columns underthe binary sum list the binary values that appear in the outputs of the 4- bit binary adder. The output sum of the two decimal digits must be represented in BCD.

Binary Sum					**BCD Sum**					**Decimal**
K	**Z_8**	**Z_4**	**Z_2**	**Z_1**	**C**	**S_8**	**S_4**	**S_2**	**S_1**	
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	2
0	0	0	1	1	0	0	0	1	1	3
0	0	1	0	0	0	0	1	0	0	4
0	0	1	0	1	0	0	1	0	1	5
0	0	1	1	0	0	0	1	1	0	6
0	0	1	1	1	0	0	1	1	1	7
0	1	0	0	0	0	1	0	0	0	8
0	1	0	0	1	0	1	0	0	1	9
0	1	0	1	0	1	0	0	0	0	10
0	1	0	1	1	1	0	0	0	1	11
0	1	1	0	0	1	0	0	1	0	12
0	1	1	0	1	1	0	0	1	1	13
0	1	1	1	0	1	0	1	0	0	14
0	1	1	1	1	1	0	1	0	1	15
1	0	0	0	0	1	0	1	1	0	16
1	0	0	0	1	1	0	1	1	1	17
1	0	0	1	0	1	1	0	0	0	18
1	0	0	1	1	1	1	0	0	1	19

In examining the contents of the table, it is apparent that when the binary sum is equal to or less than 1001, the corresponding BCD number is identical, and therefore no conversion is needed. When the binary sum is greater than 9 (1001), we obtain a non- valid BCD representation. The addition of binary 6 (0110) to the binary sum converts it to the correct BCD representation and also produces an output carry as required.

The logic circuit to detect sum greater than 9 can be determined by simplifying the boolean expression of the given truth table.

Inputs				Output
S_3	S_2	S_1	S_0	Y
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1



$Y = S_3S_2 + S_3S_1$

To implement BCD adder we require:

- 4-bit binary adder for initial addition
- Logic circuit to detect sum greater than 9 and
- One more 4-bit adder to add 01102 in the sum if the sum is greater than 9 or carry is

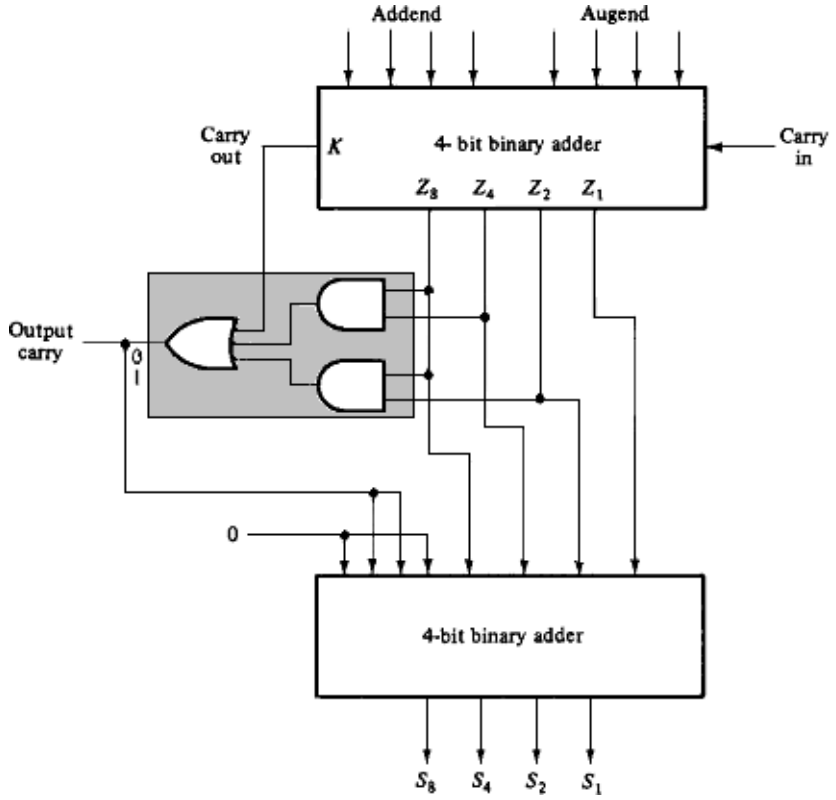


Fig. Logic diagram of BCD adder

The two decimal digits, together with the input carry, are first added in the top4- bit binary adder to provide the binary sum. When the output carry is equal to zero, nothing is added to the binary sum. When it is equal to one, binary 0110 is added to the binary sum through the bottom 4-bit adder. The output carry generated from the bottom adder can be ignored, since it supplies information already available at the output carry terminal. The output carry from one stage must be connected to the input carry of the next higher-order stage.

4.9 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 inthe form of three binary variables representing the conditions A= B, A>B and A<B, if A andB are the two numbers being compared.

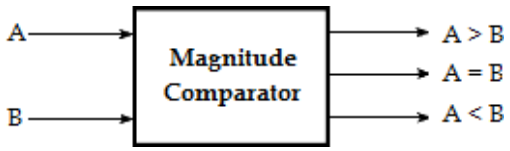


Fig. Block diagram of magnitude comparator

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

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

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

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

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

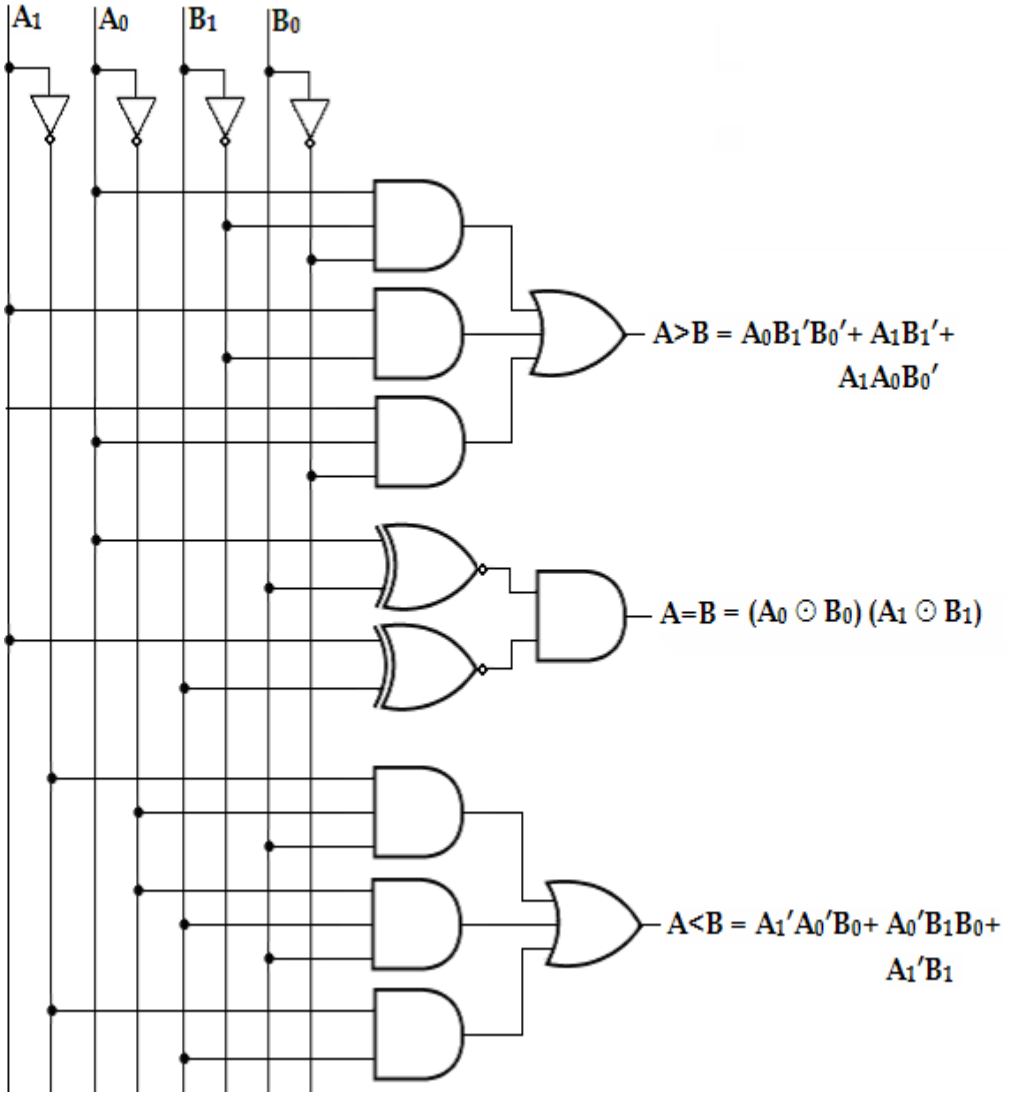
$= (A_0 \odot B_0) (A_1 \odot B_1)$

For A<B

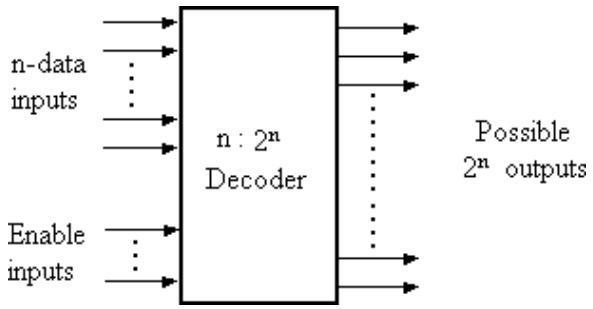
A_1A_0 \ B_1B_0	00	01	11	10
00	0	1	1	1
01	0	0	1	1
11	0	0	0	0
10	0	0	1	0

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

Logic Diagram:



4.10 Decoder:

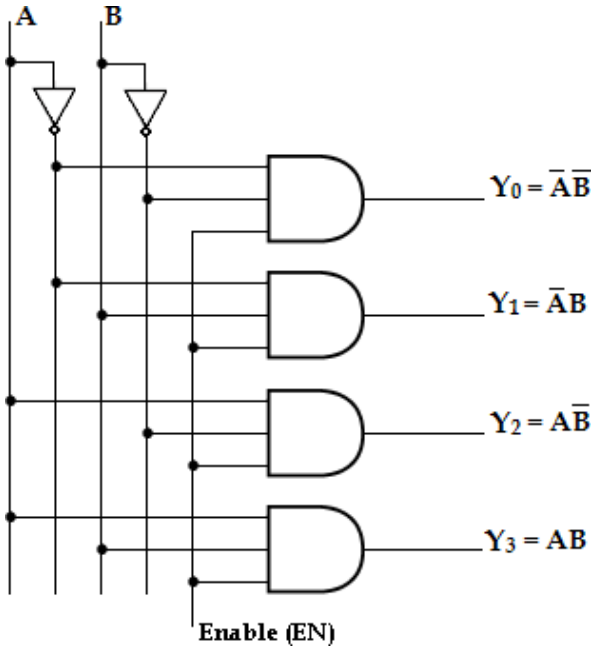


General structure of decoder

A decoder is a combinational circuit that converts binary information from _n' input lines to a maximum of _2^n' unique output lines. The encoded information is presented as _n' inputs producing _2^n' 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 thetwo 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

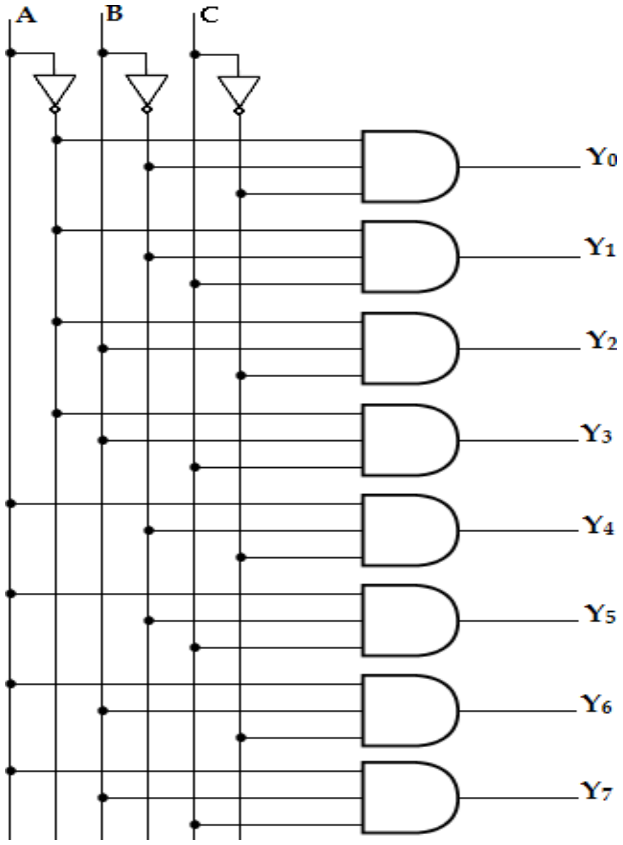
As shown in the truth table, if enable input is 1 (EN= 1) only one of the outputs (Y0 – Y3), is active for a given input. The output Y0 is active, ie., Y0= 1 when inputs A= B= 0, Y1 is active when inputs, A= 0 and B= 1, Y2 is active, when input A= 1 and B= 0, Y3 is active, wheninputs A= B= 1.

3 to-8 Line Decoder:

A 3-to-8 line decoder has three inputs (A, B, C) and eight outputs (Y0- Y7). 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



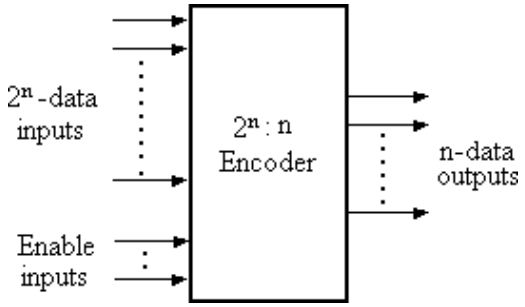
3-to-8 line decoder

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.

4.11 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 oneof 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

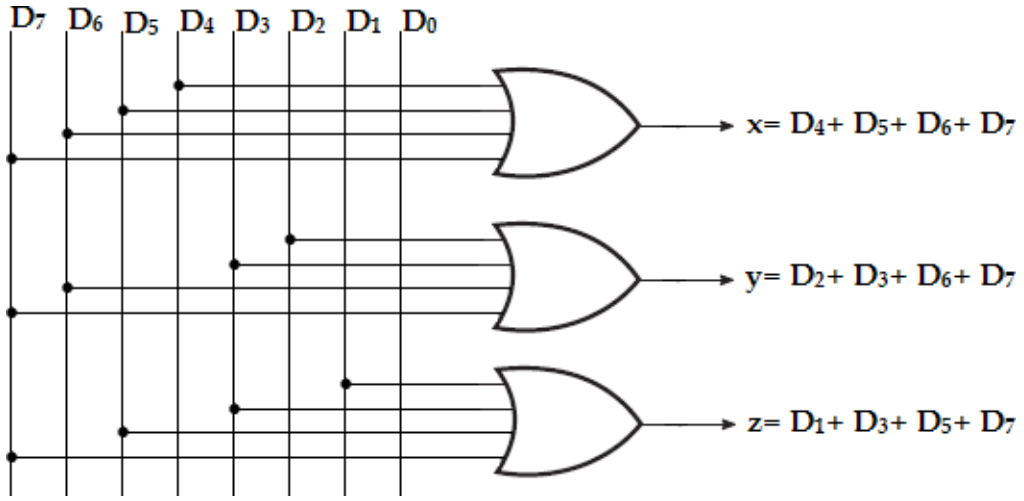
The encoder can be implemented with OR gates whose inputs are determined directly from the truth table. Output z is equal to 1, when the input octal digit is 1 or 3 or 5 or 7. Output y is 1 for octal digits 2, 3, 6, or 7 and the output is 1 for digits 4, 5, 6 or 7. These conditions can be expressed by 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 D3 and D6 are 1 simultaneously, the output of the encoder may be 111. This does not represent either D6 or D3. To resolve this problem, encoder circuits must establish aninput priority to ensure that only one input is encoded. If we establish a higher priority for inputs with higher subscript numbers and if D3 and D6 are 1 at the same time, the output will be 110 because D6 has higher priority than D3.



Octal-to-Binary Encoder

Another problem in the octal-to-binary encoder is that an output with all 0‘s is generatedwhen all the inputs are 0; this output is same as when D0 is equal to 1. The discrepancy canbe 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 willtake 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 D3, has the highest priority. So, regardless of the values of the other inputs, when D3is 1, the output for xy is11.D2 has the next priority level. The output is 10, if D2= 1 provided D3= 0. The output for D1 is generated only if higher priority inputs are 0, and so on down the priority levels.

Truth table

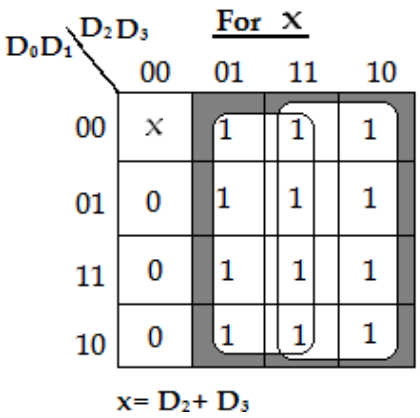
Inputs				**Outputs**		
D_0	**D_1**	**D_2**	**D_3**	**x**	**y**	**V**
0	0	0	0	x	x	0
1	0	0	0	0	0	1
x	1	0	0	0	1	1
x	x	1	0	1	0	1
x	x	x	1	1	1	1

Although the above table has only five rows, when each don‘t care condition is replaced first by 0and 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 by0 and 1 as shown in the table below.

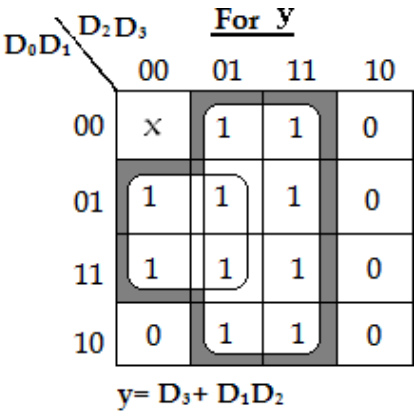
Modified Truth table

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			

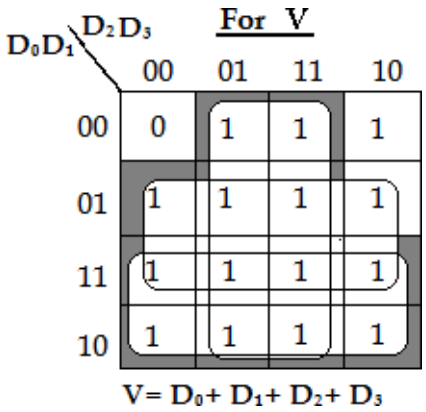
K-map Simplification:



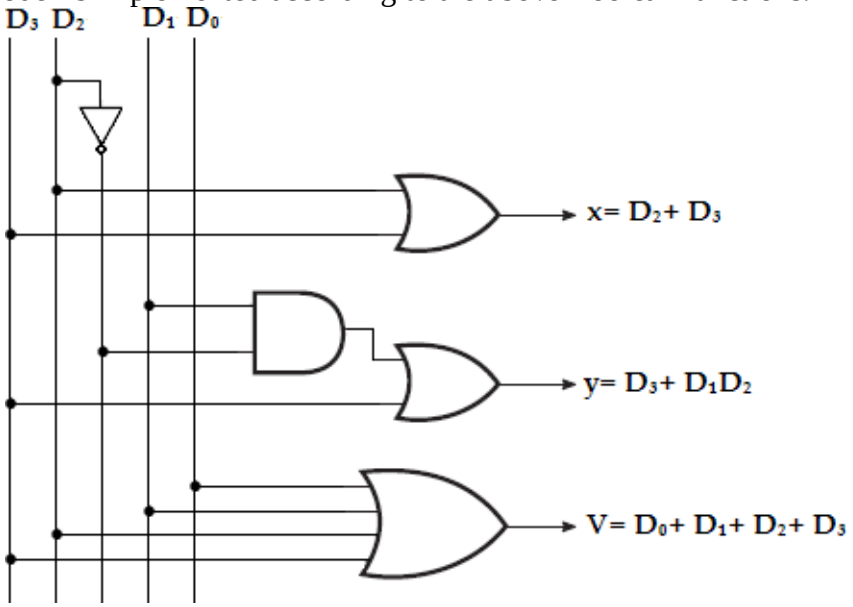
$x = D_2 + D_3$



$y = D_3 + D_1D_2$



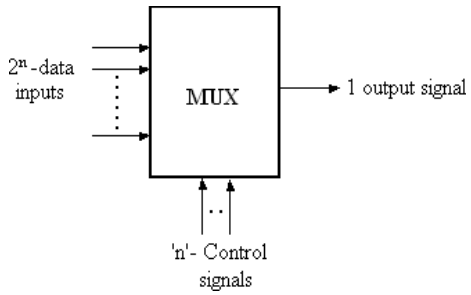
The priority encoder is implemented according to the above Boolean functions.



Logic Diagram of Priority Encoder

4.12 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.

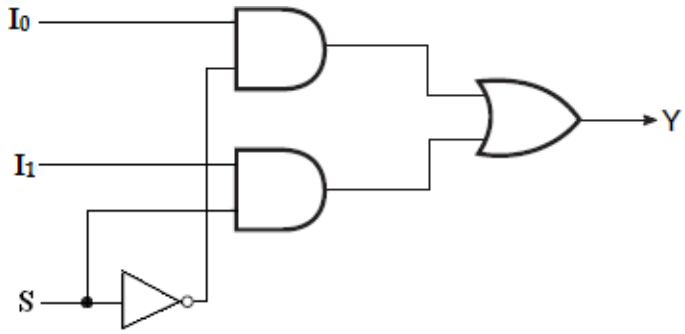


Block diagram of Multiplexer

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, theupper AND gate is enabled and I0 has a path to the output.When S=1, the lower AND gate is enabled and I1 has a path to the output.



Logic diagram

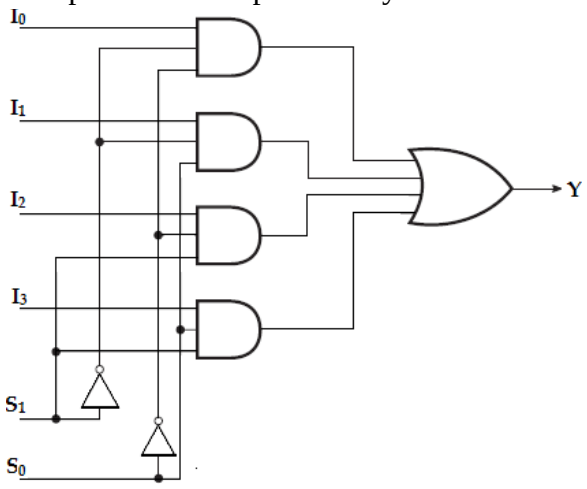
The multiplexer acts like an electronic switch that selects one of the two sources.

Truth table:

S	Y
0	I0
1	I1

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 I0 through I3, is applied to one input of AND gate. Selection lines S1 and S0 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	I0
0	1	I1
1	0	I2
1	1	I3

To demonstrate the circuit operation, consider the case when S1S0= 10. The AND gate associatedwith input I2 has two of its inputs equal to 1 and the third input connected to I2. 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 I2, providing a path from the selected input to the output.

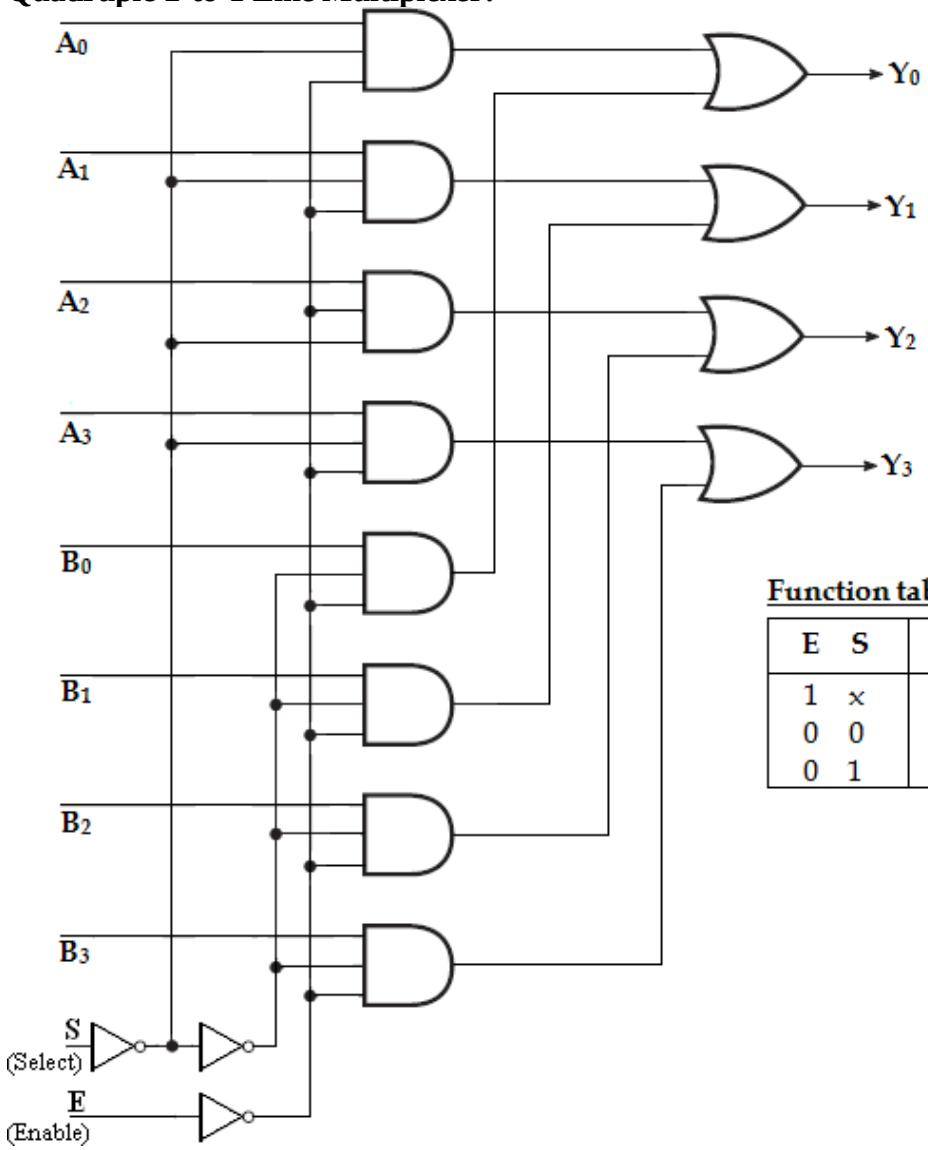
The data output is equal to I0 only if S1= 0 and S0= 0; Y= I0S1'S0'. The data output is equal toI1 only if S1= 0 and S0= 1; Y= I1S1'S0. The data output is equal to I2 only if S1= 1 and S0= 0; Y= I2S1S0'. The data output is equal to I3 only if S1= 1 and S0= 1; Y= I3S1S0. When these terms are ORed, the 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. This circuit has four multiplexers, each capable of selecting one of two input lines. Output Y0 can be selected to come from either A0 or B0. Similarly, output Y1 may have the value of A1or B1, and so on. Input selection line, S selects one of the lines in each of the four multiplexers. The enable input E must be active for normal operation. Although the circuit contains four 2-to-1-Line multiplexers, it is viewed as a circuit thatselects one of two 4-bit sets of data lines. The unit is enabled when E= 0. Then if S= 0, thefour A inputs have a path to the four outputs. On the other hand, if S=1, the four B inputs areapplied to the outputs.

The outputs have all 0's when E= 1, regardless of the value of S.

Quadruple 2-to-1 Line Multiplexer:



Function table:

E	S	Output Y
1	x	All 0's
0	0	Select A
0	1	Select B

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 beimplemented.

In general, a Boolean expression of (n+1) variables can be implemented usinga multiplexer with 2^n inputs.

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

Solution:

Variables, n= 3 (A, B, C) Select lines= n-1 = 2 (**S1, S0**)$_2$n-1 to MUX i.e., 22 to 1 = 4 to 1 MUX

Input lines= $2^{n-1} = 2^2 = 4$ (**D_0, D_1, D_2, D_3**)

Implementation table:

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

- List the input of themultiplexer
- 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	$\overline{C}$

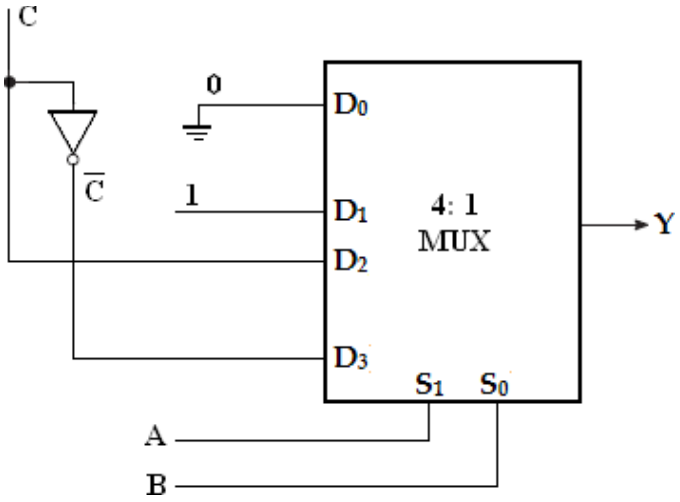
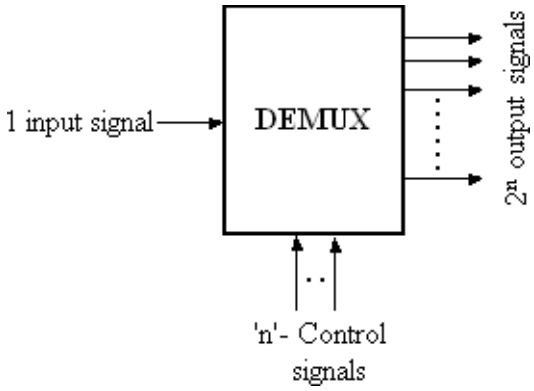


Fig. Multiplexer Implementation

4.13 Demultiplexer:

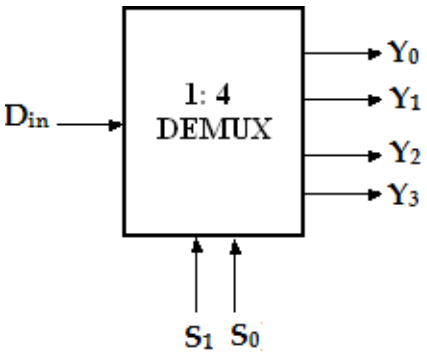
Demultiplex means one into many. Demultiplexing is the process of taking information fromone input and transmitting the same over one of several outputs.A demultiplexer is a combinational logic circuit that receives information on a single inputand transmits the same information over one of several (2^n)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



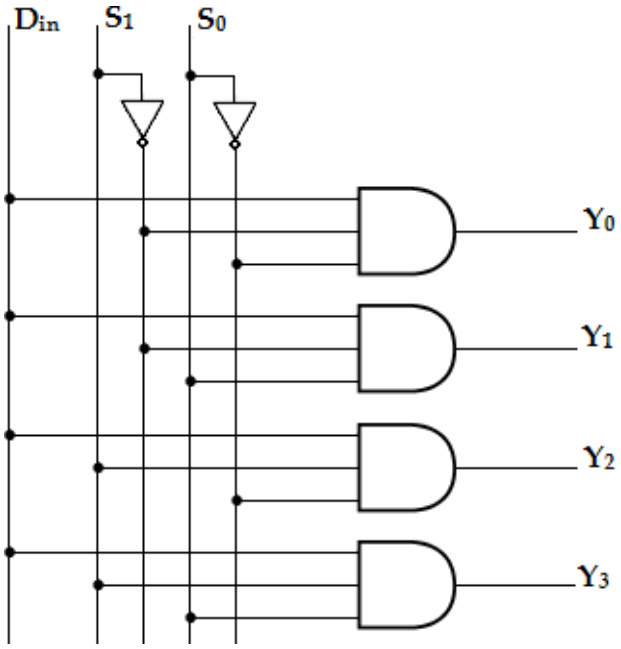
Logic Symbol

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 isdirected 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	Din	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, Din is connected to the output Y0, whenS1= 0 and S0= 0 and the data input is connected to output Y1 when S1= 0 and S0= 1. Similarly,the data input is connected to output Y2 and Y3 when S1= 1 and S0= 0 and when S1= 1 and S0= 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 Din, is connected to all the AND gates. The two select lines S1, S0 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.

1-to-8 Demultiplexer:

A 1-to-8 demultiplexer has a single input, $\mathbf{D_{in}}$, eight outputs ($\mathbf{Y_0}$ **to** $\mathbf{Y_7}$) and three select inputs($\mathbf{S_2}$, $\mathbf{S_1}$ **and** $\mathbf{S_0}$). 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

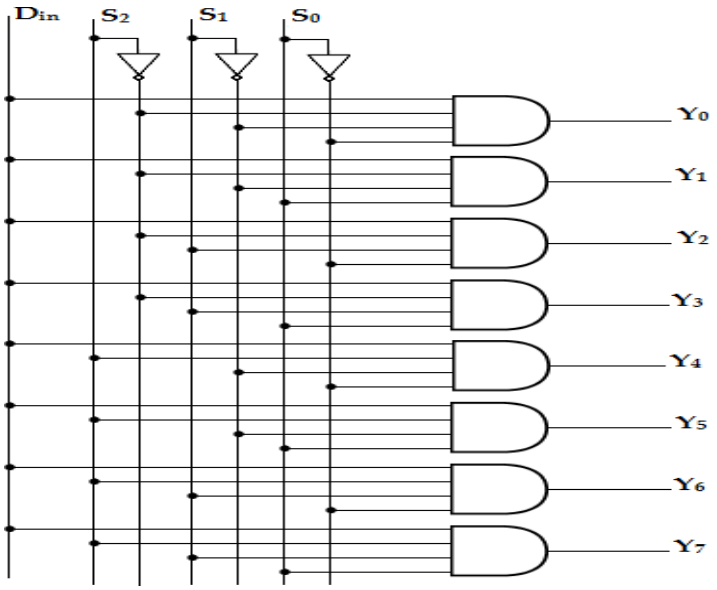
Din	S_2	S_1	S_0	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
0	x	X	x	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	1
1	0	0	1	0	0	0	0	0	0	1	0
1	0	1	0	0	0	0	0	0	1	0	0
1	0	1	1	0	0	0	0	1	0	0	0
1	1	0	0	0	0	0	1	0	0	0	0
1	1	0	1	0	0	1	0	0	0	0	0
1	1	1	0	0	1	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0	0	0

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 outputscan be written as follows:

Y0= S2‘S1‘S0‘Din Y4= S2 S1‘S0‘Din Y1= S2‘S1‘S0Din Y5= S2 S1‘S0Din

Y2= S2‘S1S0‘Din Y6= S2 S1S0‘Din Y3= S2‘S1S0Din Y7= S2S1S0Din

Now using the above expressions, the logic diagram of a 1-to-8 demultiplexer can be drawnas shown below. Here, the single data line, Din is connected to all the eight AND gates, but only one of the eight AND gates will be enabled by the select input lines. For example, if S2S1S0= 000, then only AND gate-0 will be enabled and thereby the data input, Din will appear at Y0. Similarly, the different combinations of the select inputs, the input Din will appear at the respective output.



Logic diagram of 1-to-8 demultiplexer

AUTHOR'S PROFILE

Dr. SANTHOSH KUMAR ALLEMKI is an Associate Professor in the Department of ECE at Guru Nanak Institute of Technology, Hyderabad. He holds a Ph.D., M.Tech, and B.Tech in ECE from JNTU Hyderabad, with 19 years of teaching experience. He has published 40 research papers, authored 6 textbooks, holds 6 patents, and successfully conducted an AICTE-funded ATAL FDP. His research interests include Electromagnetic Fields and Antennas. He is a reviewer for international journals and a member of MIAENG and MIJSR, with six completed MOOCs in emerging technologies.

Dr. SHATRUGHNA PRASAD YADAV is a Professor, Dean Academics, and Head of the Department of Electronics and Communication Engineering at Guru Nanak Institute of Technology. With over 37 years of combined experience—20 years in the Indian Air Force and 17 years in academia—he is a seasoned technocrat and academician. He holds a Ph.D. in ECE, an ME in Digital Systems, an MBA in Marketing Management, and a UGC-NET in Management. Dr. Yadav has published more than 57 research papers in reputed national and international journals and conferences. He is a Senior Member of IEEE (USA), and a Fellow of IEI, IETE, and BES. His research interests include Digital Communication and Digital Signal Processing. He is widely respected for his academic leadership and technical contributions

Dr. VIJAYASARO VIJAYAREGUNATHAN is an Associate Professor in the Department of ECE at Guru Nanak Institute of Technology, Hyderabad. She holds a Ph.D. in ECE from PRIST University, Tamil Nadu, and an M.E. in Embedded System Technologies from Anna University, Coimbatore. With 10 years of teaching and research experience, she has published 15 papers in national and international journals. She also serves as a reviewer and editorial board member for the *International Journal of Science and Engineering*. Her research interests include IoT and Embedded Systems.

Dr. GAMBALA KIRANMAYE is an Associate Professor in the Department of Electronics and Communication Engineering at Guru Nanak Institute of Technology, Hyderabad. She holds a Ph.D. in VLSI Systems & Architectures and Image Processing from JNTU Hyderabad, an M.Tech in VLSI System Design from JNTU Hyderabad, and a B.E. in ECE from Osmania University. With over 22 years of academic experience and strong industry exposure in VLSI, she has contributed to the development of advanced microcontroller bus architectures. She also serves as Academic Coordinator, Exam Branch In-Charge, and BOS Member at GNIT. Dr. Kiranmaye has published over 20 research papers and holds patents in drone technology, VLSI architectures, and spectrum detection techniques. Her research interests include VLSI System Design and Image Processing.She"Published author of technical books and also written a book called 'Growing Gracefully" for empowering girls through education and mentorship."

Publisher : Edupresspublishers, Hyderabad
www.edupresspublishers.in
Rs. 399/-

ISBN : 978-81-982681-7-4
Year of Publication : 12-03-2025